UDC 004.8, 004.9

Quantum Logical Bioinformatics: The Genetic Alphabet of 4 Unitary Hadamard Operators and Cyclic Groups

Petoukhov S.V. (Russian Engineering Academy, Moscow)

This paper presents data on an additional operator-alphabetic informatic system for biological inheritance in living organisms, which can provide inheritance of biological algorithms and functions and which exists alongside the well-known nucleotide-alphabetic system of information inheritance. This operator-alphabetic bioinformatics and its alphabet are linked to quantum mechanics, quantum logic, and quantum information science, since genetic molecules belong to the microworld of quantum mechanics. In the search for and justification of this operator bioinformatics and its alphabets, special attention is paid to unitary operators, which underlie all calculations in quantum computers and which, in quantum mechanics, describe the evolution of closed quantum systems. The quantum logic apparat being developed for this bioinformatics is based on a genetic alphabet of four unitary Hadamard matrices, families of unitary matrices based on this alphabet, and cyclic groups of unitary transformations. Some prospects of the proposed approach for the development of quantum logic biology, artificial intelligence, and biotechnology are discussed.

Keywords: *quantum logic, quantum bioinformatics, genetic automata, cycles, biorhythms, probability, unitary matrices, tensor product, systolic processors, artificial intelligence, qubits.*

1. Introduction

Information communication systems are built on the use of appropriate alphabets to form information messages. For example, all computer programs rely on corresponding alphabets. The amino acid sequences of proteins are genetically inherited using information messages in DNA and RNA molecules based on an alphabet of four DNA and RNA nucleotides: adenine A, guanine G, cytosine C, and thymine T (in RNA, uracil U replaces thymine T). Knowledge of this alphabet is useful for understanding the structure of proteins and nucleic acids. However, as Nobel laureate in chemistry T. Steitz emphasizes, all knowledge about the biochemical structure of proteins and nucleic acids encoded in the genome will not tell us, for example, how a butterfly flies [23]. Nor will it tell us how turtles, after hatching from an egg, immediately, without any training, begin to crawl toward the water with coordinated movements of their limbs, which requires the logically coordinated activity of millions of their nerve and muscle cells. Or how a newborn baby cries and begins to suckle at its mother's breast, which also requires the logically coordinated activity of billions of nerve and muscle cells. Knowledge of DNA/RNA nucleotide sequences also fails to

explain the inheritance of the mathematical beauty of bioforms (mollusk shells, etc.), which are repeated in bodies of very different biochemical compositions and are constructed through the spatiotemporal ordering of trillions of different molecules. Thus, the modern science of biological inheritance lacks knowledge of a bioinformation system capable of ensuring the inheritance of "cooperative biomechanics" phenomena, that is, the logically coordinated (coherent) behavior of body parts and the functions based on them.

The aforementioned inherited logical forms of collective behavior in biosystems require a search for a corresponding operator bioinformatics system based on a suitable alphabet for their modeling. One can believe, that, alongside nucleotide-alphabetic bioinformatics, an additional operator-alphabetic bioinformatics operates in living things. This hidden operator bioinformatics and its alphabet are apparently linked to quantum mechanics and quantum information, since genetic molecules belong to the microworld of quantum mechanics, and many authors have long suspected that living organisms are quantum-like entities. In search of an appropriate alphabet, special attention should be paid to unitary operators, which play a role of logical gates to provide all calculations in quantum computing [15] and which, in quantum mechanics, describe the evolution of closed quantum systems.

This article is devoted to the author's discovery of a genetic alphabet of four Hadamard unitary operators and the development of operator quantum-logical bioinformatics based on it for the mathematical modeling of logical features of the structure and behavior of inherited multicomponent body parts, including the properties of multiple, time-coordinated cyclic processes. The article provides specific examples of modeling known biological phenomena from the perspective of this emerging operator bioinformatics, which relies on the aforementioned alphabet of unitary operators, its algebraic extensions, cyclic groups of unitary operators, quantum logic formalisms, and Chargaff's second rule, well-known in genetics. These examples confirm the adequacy of the emerging quantum-logical bioinformatics system, which offers new approaches to modeling inherited, logically organized biological phenomena.

The founder of quantum information science, Yu. I. Manin (https://en.wikipedia.org/wiki/Yuri_Manin), introduced the concept of a quantum computer in his book [14, p. 15] precisely when analyzing the characteristics of high-speed processing of information in chromosomal DNA by "*genetic automata*," prophetically pointing out the important role of unitary operators and tensor products: "*A quantum automaton must be abstract: its mathematical model must use only the most general quantum principles, without prejudging*

physical implementations. Then the evolution model is a unitary rotation in a finite-dimensional Hilbert space, and the model of virtual separation into subsystems corresponds to the decomposition of space into a tensor product. Somewhere in this picture, there must be a place for interaction, traditionally described by Hermitian operators and probabilities." Thus, the very birth of quantum information science, so promising for the development of artificial intelligence and information science. The data in this article are consistent with this prophecy of Yu. I. Manin.

2. Genetic Matrices of DNA Nucleotide Alphabets and the Alphabet of 4 Unitary Hadamard Operators

Amino acid sequences of proteins are inherited through information messages on genetic DNA molecules, written in an alphabet of 4 nucleotides: adenine A, cytosine C, guanine G, and thymine T. This alphabet carries a system of binary-opposition indicators, which provides distinguishing three types of binary sub-alphabets within it:

1) two of these nucleotides are purines (A and G), having two rings in their molecule, and the other two (C and T) are pyrimidines, containing one ring. This yields a binary representation (binary sub-alphabet) **C=T=0**, **A=G=1**;

2) two of these nucleotides are keto-molecules (T and G), and the other two (C and A) are amino-molecules, yielding the binary representation **C=A=0**, **T=G=1**;

3) pairs of complementary nucleotides A-T and C-G are linked by 2 and 3 hydrogen bonds, respectively (called weak and strong hydrogen bonds in genetics), yielding the binary representation **C=G=0**, **A=T=1**.

These molecular binary-oppositional traits of the nucleotide alphabet of DNAs (and RNAs) of all living organisms are summarized in Table 1, which shows the distribution of traits within it.

Table 1. Binary distribution of molecular traits in the DNA nucleotide alphabet G, A, T, C. The fourth row of the table represents with the symbol +1 the fact of the presence of benzene rings in the molecules of all four nucleotides.

Molecular traits and their symbols	G	A	T	C
pyrimididines +1, purines -1	-1	-1	+1	+1
amino-nucleotides +1, кето-nucleotides - 1	-1	+1	-1	+1
complementarity with 3 or 2 hydrogen bonds: +1, -1	+1	-1	-1	+1
the presence of benzene rings +1	+1	+1	+1	+1

The right-hand side of this phenomenological Table 1 contains an appeared fourth-order Hadamard matrix, whose quadrants are occupied by four types of second-order Hadamard matrices (unitary when normalized). Hadamard matrices are fundamental building blocks of quantum

computers, enabling qubit superposition and also a key element of quantum parallelism in the quantum Fourier transform and other quantum algorithms.

This same set of four Hadamard matrices is revealed in the analysis of molecular genetic informatics from other approaches. For example, in the construction of genetic $(2^n \bullet 2^n)$-matrices, in which the columns are numbered with binary digits of one of the aforementioned DNA sub-alphabets, and the rows with digits of another. In this case, the same four matrices are derived from the 1:3 oppositional relationships between the four members of the DNA nucleotide alphabet (for example, only one nucleotide, thymine T, is phenomenologically replaced in the molecular genetic system by uracil U during the transition from DNA to RNA). In another approach, taking into account the well-known second Chargaff rule [5, 28] on the ratio of probabilities of 4 nucleotides in all long single-stranded DNAs (length greater than 100 kbps) leads to a real Hermitian probability matrix $\mathbf{W_D}$ = [0.5, 0.5; 0.5, 0.5]. This genetic Hermitian matrix $\mathbf{W_D}$ is doubly stochastic: the sum of the elements in each row and each column of such a matrix is equal to one. But with respect to doubly stochastic matrices, the following theorem is known [21]:

- If the matrix $\mathbf{V} = ||\mathbf{v_{ij}}||^n$ is unitary, then the matrix $\mathbf{W} = ||\mathbf{w_{ij}}||^n$, where $\mathbf{w_{ij}} = |\mathbf{v_{ij}}|^2$, is doubly stochastic.

According to this theorem, the doubly stochastic genetic matrix $\mathbf{W_D}$ of probabilities corresponds to four unitary Hadamard genetic matrices, denoted here as $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, and $\mathbf{H_G}$ (Fig. 1): squaring all components of each of these unitary matrices generates the doubly stochastic matrix WD. Note that these same four unitary matrices $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, and $\mathbf{H_G}$, but taken with a minus sign, are treated as their banal analogue and are not considered separately.

$$\mathbf{H_C} = 2^{-0.5} \begin{vmatrix} 1, -1 \\ 1, 1 \end{vmatrix}; \quad \mathbf{H_T} = 2^{-0.5} \begin{vmatrix} 1, 1 \\ 1, -1 \end{vmatrix}; \quad \mathbf{H_G} = 2^{-0.5} \begin{vmatrix} 1, 1 \\ -1, 1 \end{vmatrix}; \quad \mathbf{H_A} = 2^{-0.5} \begin{vmatrix} -1, 1 \\ 1, 1 \end{vmatrix}$$

Fig. 1 – Four unitary Hadamard matrices $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, and $\mathbf{H_G}$ obtained from the doubly stochastic Hermitian matrix $\mathbf{W_D}$ = [0.5, 0.5; 0.5, 0.5].

Identifying this connection between the statistical universals of genomic DNAs and these four unitary Hadamard matrices is important due to the significance of unitary transformations (unitary operators) for quantum mechanics, quantum computing, biosystems, signal processing engineering, and other fields. Unitary transformations preserve vector lengths and scalar products (preserve the metric), representing rotation and mirror reflection operators. Unitary matrices satisfy the criterion:

the product of a unitary matrix with its transpose is equal to one. Unitary transformations with real components are called orthogonal transformations, but in this article, we will use their more general name, "unitary transformations," by which they are better known in various fields of science. In quantum mechanics, unitary transformations describe the time evolution of isolated quantum systems, and in quantum mechanics (unlike classical mechanics), observable quantities are represented not by numbers, but by operators. In quantum computers, all calculations are performed on the basis of unitary operators, which act as logical gates, and any unitary operator can be used in quantum computing as a gate [15].

We will call the four unitary operators $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, and $\mathbf{H_G}$–(Fig. 1) the genetic Hadamard gates. These genetic gates are four versions of the Hadamard (2•2)-matrix with the weighting factor $2^{-0.5}$, which is traditionally used in quantum computing to transform the Hadamard matrix into a unitary operator. By definition, the Hadamard matrix H_n is a (n•n)-matrix, composed of the numbers 1 and -1, whose columns are orthogonal and the relation $H_n \bullet H^T = n \bullet E_n$ holds, where E_n is the identity matrix of order n. Among the properties of Hadamard matrices is the fact that permuting any rows and columns of the Hadamard matrix always yields a new Hadamard matrix. In passing, we note that in the general case, Hadamard matrices have many other remarkable properties and applications (see, for example, [1]).

One of these four Hadamard genetic unitary operators (gates) — $\mathbf{H_T}$ — has long been used in quantum computers for fundamental operations on qubits, serving as a key element in many quantum algorithms, including the Deutsch-Jozsa algorithm and Shor's algorithm. This Hadamard gate provides quantum algorithms with a superposition principle of quantum entanglement for quantum superiority of quantum algorithms in comparison to known classical algorithms [15].

Of the four genetic Hadamard gates, two gates $\mathbf{H_A}$ and $\mathbf{H_T}$ are mirror reflection operators. Raising them to integer powers generates the corresponding cyclic groups of unitary operators with period 2. The other two unitary matrices ($\mathbf{H_C}$, $\mathbf{H_G}$) are rotation operators ($\mathbf{H_C}$ counterclockwise, $\mathbf{H_G}$ clockwise) and matrix representations of the complex number $Z = (1+\mathbf{i}) \bullet 2^{-0.5}$, where **i** is the imaginary unit of the complex number ($i^2 = -1$). Repeated raising these unitary matrices $\mathbf{H_C}$ and $\mathbf{H_G}$ to integer powers (positive and negative) generates cyclic (with period 8) groups of unitary operators, which are matrix representations of the complex numbers (1).

$$\mathbf{H_C}^n = \mathbf{H_C}^{n+8}, \quad \mathbf{H_G}^n = \mathbf{H_G}^{n+8}, \quad \mathbf{H_A}^n = \mathbf{H_A}^{n+2}, \quad \mathbf{H_T}^n = \mathbf{H_T}^{n+2} \qquad (1)$$

Moreover, any of the unitary matrices $\mathbf{H_C}$ and $\mathbf{H_G}$ can be represented as the product of k unitary matrices, which are their kth roots. In other words, the action of a single unitary operator, for example, $\mathbf{H_C}$, can be represented as the action of a sequence of k more fractional unitary operators

$\mathbf{H_C}^{1/k}$. Thus, with the matrix-vector approach, any large transformation in the system from the action of such a large operator $\mathbf{H_C}$ can be represented as consisting of a sequence of arbitrarily small transformations from the action of the corresponding sequence of unitary operators $\mathbf{H_C}^{1/k}$ for modeling quasi-continuous transformations in the simulated cyclic processes. We also note that raising the genetic gate $\mathbf{H_C}$ or $\mathbf{H_G}$ to a power representing a cyclic function of time in its step-by-step states allows modeling quasi-continuous cyclic bioprocesses as vector sequences of their step-by-step functional states.

In quantum logic, projectors (projection operators) play an important role in quantum logic. In light of this, we note that the unitary Hadamard matrices in the H_C and H_G operators are sums of two sparse matrices representing projectors Ps satisfying the projector criterion $\mathbf{P_s}^2 = \mathbf{P_s}$ and shown in parentheses in expression (2):

$$\mathbf{H_C} = 2^{-0.5} \bullet [1\ -1;\ 1\ 1] = 2^{-0.5} \bullet ([1, 0; 1, 0] + [0, -1; 0, 1]),$$

$$\mathbf{H_G} = 2^{-0.5} \bullet [1\ 1;\ -1\ 1] = 2^{-0.5} \bullet ([1, 0; -1, 0] + [0, 1; 0, 1]) \quad (2)$$

Many genetically inherited biological structures in organisms are clearly linked to unitary transformations of rotations and mirror images. For example, the kinematic schema of the human body and its locomotion is based on unitary transformations of rotations at the joints (the human body has approximately 300 joints) and mirror symmetry of the left and right halves of the body. Human motor activity is reduced to the skillful control by the nervous system of ensembles of these unitary transformations in body kinematics, which is associated with the genetically inherited ability of the nervous system to operate unitary transformations. Moreover, a person's very concept of their body schema is innate: people with limbs missing from birth and no personal experience using them nevertheless perceive them as truly existing, with phantom pain in them [25, 26].

When studying human sensorimotor characteristics, it's important to consider that the genetically inherited nervous system is structurally related to genetic structures. Humans see the world through probabilities in statistical streams of signals from neurons in the retina (containing millions of receptor cells) and other sensory organs. Norbert Wiener, the father of cybernetics, asserted: "*Genetic memory—the memory of our genes—is essentially determined by nucleic acid complexes...there is reason to believe that the memory of the nervous system has a similar nature*" [22, 27].

Another example of the biological importance of unitary transformations is the construction of complex three-dimensional protein shapes in the body, known as protein folding. These shapes are

based on unitary transformations involving the rotation of protein molecule segments relative to each other around relatively strong carbon-carbon bonds.

The author proposes to consider and use the family of 4 genetic unitary Hadamard operators $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, $\mathbf{H_G}$ (Fig. 1) as a basic genetic quantum-logical alphabet for the development on its basis of the theory of a quantum-logical information system that allows modeling genetically inherited and logically organized biological structures and phenomena.

Here, the distinctive features of quantum logic should be clarified [3, 24]. Quantum logic is an algebraic system for describing, using quantum gates, how qubits operate and interact and how to extract information from them. In quantum logic, "logic" is not contained in reasoning, but in the mathematical description of states and operations. Quantum logic can be formulated as a modified version of propositional logic. For comparison, we recall that classical Boolean logic is a set of logical rules (AND, OR, NOT, etc.) describing how bits (0 or 1) can be combined and transformed according to the laws of Boolean algebra with its key principle of distributivity and the statements "true" or "false." Quantum logic considers not "true/false" statements, but questions to a quantum system. The answer to such a question is provided by the probability value obtained during measurement. Logical operations are replaced by quantum gates (unitary operations): NOT becomes an X gate, and completely new operations appear that have no analogues in classical logic, for example, the Hadamard gate, which creates superposition. Quantum logic operates on qubits, vectors, and matrices, not on sets. Its mathematical foundation is the theory of Hilbert spaces and projective and unitary operators. The state space of a quantum system is described by vectors, and rotations of these vectors serve as logical operations. Quantum logic lacks distributivity, which is considered its key difference from Boolean logic. Quantum logic is a branch of logic necessary for reasoning about propositions that take into account the principles of quantum theory. It was founded by the work of G. Birchow and J. von Neumann [2], who sought to reconcile the inconsistencies of classical logic with the facts about measurements in quantum mechanics and saw in quantum logic a possible foundation for physics.

The unitary Hadamard matrices of the basic operator genetic alphabet $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, $\mathbf{H_G}$ (Fig. 1) and many types of their combinations into unitary matrices of higher orders form - when they are repeatedly raised to powers - cyclic groups of operators with different periods and used to model cyclic sequences of states of quantum-like systems. The resulting algebraic-geometric apparatus is intended, first of all, for quantum-logical modeling of a set of genetically inherited cyclic and hypercyclic biostructures in genetic biomechanics. It should be noted that the quantum-logical approach to inherited cyclic and hypercyclic biostructures developed by the author, based on the alphabet of Hadamard genetic gates and the mathematical apparatus of quantum-logical biology,

differs fundamentally from the well-known biochemical concept of catalytic cycles and hypercycles [4].

Global research in genetic informatics relies heavily on the fundamental fact that the DNA of all organisms contains a molecular alphabet of 4 nucleotides: C, G, A, T, and its extensions into nucleotide alphabets of 16 doublets, 64 triplets, and so on. In parallel with this molecular alphabet of 4 nucleotides, bioinformatics and genetic biomechanics now allow and should work with an operator alphabet of 4 genetic Hadamard gates, that is, a fundamentally new type of alphabet: a quantum-logical operator alphabet of 4 unitary matrices - $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, $\mathbf{H_G}$ -, which gives rise to interconnected sets of higher-order unitary operators. These genetic unitary Hadamard matrices are conjugated with corresponding complete orthogonal systems of Walsh functions. The latter are the basis of a special spectral analysis of signals in digital informatics and are associated with cyclic Gray codes, logical holography, Walsh antennas, and the fractal Hilbert curve, which is known to correspond to the spatial packing of chromatin in the human genome [13]. The relationship of Hadamard matrices with the listed areas is described in our works [16, 17, 20].

The mathematical properties of the alphabet of 4 genetic gates and the sets of unitary operators constructed on their basis are subject to systematic study. For example, the question of the existence in this set of non-commuting unitary operators whose commutators are non-zero is subject to study (in quantum mechanics, the study of pairs of operators characterized by non-zero commutators led to the formulation of the Heisenberg uncertainty principle). Already in the alphabet of 4 genetic Hadamard gates, there are three pairs of non-commuting unitary operators characterized by non-zero commutators (3); the values of these commutators are equal in two cases to Hadamard matrices, and in the third case to the matrix representation of twice the imaginary unit i of a complex number:

$$\mathbf{H_A}\bullet\mathbf{H_C} - \mathbf{H_C}\bullet\mathbf{H_A} = [1, 1; 1, -1],$$
$$\mathbf{H_C}\bullet\mathbf{H_T} - \mathbf{H_T}\bullet\mathbf{H_C} = [-1, 1; 1, 1],$$
$$\mathbf{H_A}\bullet\mathbf{H_T} - \mathbf{H_T}\bullet\mathbf{H_A} = [0, -2; 2, 0] \qquad (3)$$

At this stage of research, the question of the genetic significance and possible interpretation of these and other facts of non-zero commutators among genetic unitary operators remains open for discussion (it is possible that when analyzed from the standpoint of quantum-logical bioinformatics, it will turn out to be associated with the phenomenon of chirality in biological structures, taking into account the known facts about the existence of chirality in the quantum physics of elementary particles).

The next section discusses the development of the mathematical apparatus of quantum-logical bioinformatics, which includes a set of genetic unitary operators and their cyclic groups for modeling genetically inherited cyclic biostructures and biorhythmic processes.

3. Expansion of the set of genetic unitary operators

The DNA alphabet of 4 nucleotides C, A, T, G is identical in number of elements to the quantum logic alphabet of 4 genetic Hadamard gates $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, and $\mathbf{H_G}$ proposed by the author (Fig. 1). In matrix genetics, it is known that, based on the genetic (2•2)-matrix of the 4-nucleotide alphabet, by raising it to tensor powers, genetic (2^n•2^n)-matrices are formed with a strictly regular arrangement of 16 duplets, 64 triplets, 256 tetraplets, etc. [16, 20]. Figure 2 shows examples of tensor-linked genetic matrices of 4 nucleotides, 16 duplets, and 64 triplets from these books.

	0	1
0	C	A
1	T	G

;

	00	01	10	11
00	CC	CA	AC	AA
01	CT	CG	AT	AG
10	TC	TA	GC	GA
11	TT	TG	GT	GG

	000	001	010	011	100	101	110	111
000	CCC	CCA	CAC	CAA	ACC	ACA	AAC	AAA
001	CCT	CCG	CAT	CAG	ACT	ACG	AAT	AAG
010	CTC	CTA	CGC	CGA	ATC	ATA	AGC	AGA
011	CTT	CTG	CGT	CGG	ATT	ATG	AGT	AGG
100	TCC	TCA	TAC	TAA	GCC	GCA	GAC	GAA
101	TCT	TCG	TAT	TAG	GCT	GCG	GAT	GAG
110	TTC	TTA	TGC	TGA	GTC	GTA	GGC	GGA
111	TTT	TTG	TGT	TGG	GTT	GTG	GGT	GGG

Fig. 2. Tensor family of genetic matrices of 4 nucleotides C, A, T, G, 16 nucleotide duplets, and 64 nucleotide triplets.

The tensor product is a critically important operation in the quantum mechanics of multicomponent systems and quantum information science [15]. By analogy with the tensor family of genetic matrices based on the alphabet of 4 nucleotides (Fig. 2), we construct a tensor family of genetic matrices based on the alphabet of 4 Hadamard genetic gates $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, and $\mathbf{H_G}$ (Fig. 1), for example, by simply replacing the nucleotide symbols C, A, T, and G with similarly indexed symbols of these gates and connecting adjacent gates in a bundle with the tensor multiplication sign ⊗ (Fig. 3).

H_C	H_A
H_T	H_G

;

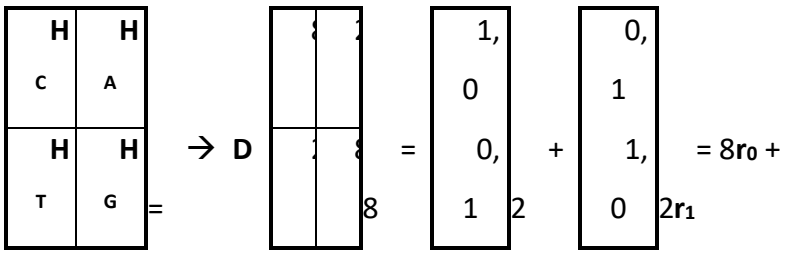
$H_C \otimes H_C$	$H_C \otimes H_A$	$H_A \otimes H_C$	$H_A \otimes H_A$
$H_C \otimes H_T$	$H_C \otimes H_G$	$H_A \otimes H_T$	$H_A \otimes H_G$
$H_T \otimes H_C$	$H_T \otimes H_A$	$H_G \otimes H_C$	$H_G \otimes H_A$
$H_T \otimes H_T$	$H_T \otimes H_G$	$H_G \otimes H_T$	$H_G \otimes H_G$

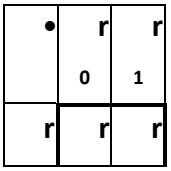
$H_C \otimes H_C \otimes H_C$	$H_C \otimes H_C \otimes H_A$	$H_C \otimes H_A \otimes H_C$	$H_C \otimes H_A \otimes H_A$	$H_A \otimes H_C \otimes H_C$	$H_A \otimes H_C \otimes H_A$	$H_A \otimes H_A \otimes H_C$	$H_A \otimes H_A \otimes H_A$
$H_C \otimes H_C \otimes H_T$	$H_C \otimes H_C \otimes H_G$	$H_C \otimes H_A \otimes H_T$	$H_C \otimes H_A \otimes H_G$	$H_A \otimes H_C \otimes H_T$	$H_A \otimes H_C \otimes H_G$	$H_A \otimes H_A \otimes H_T$	$H_A \otimes H_A \otimes H_G$
$H_C \otimes H_T \otimes H_C$	$H_C \otimes H_T \otimes H_A$	$H_C \otimes H_G \otimes H_C$	$H_C \otimes H_G \otimes H_A$	$H_A \otimes H_T \otimes H_C$	$H_A \otimes H_T \otimes H_A$	$H_A \otimes H_G \otimes H_C$	$H_A \otimes H_G \otimes H_A$
$H_C \otimes H_T \otimes H_T$	$H_C \otimes H_T \otimes H_G$	$H_C \otimes H_G \otimes H_T$	$H_C \otimes H_G \otimes H_G$	$H_A \otimes H_T \otimes H_T$	$H_A \otimes H_T \otimes H_G$	$H_A \otimes H_G \otimes H_T$	$H_A \otimes H_G \otimes H_G$
$H_T \otimes H_C \otimes H_C$	$H_T \otimes H_C \otimes H_A$	$H_T \otimes H_A \otimes H_C$	$H_T \otimes H_A \otimes H_A$	$H_G \otimes H_C \otimes H_C$	$H_G \otimes H_C \otimes H_A$	$H_G \otimes H_A \otimes H_C$	$H_G \otimes H_A \otimes H_A$
$H_T \otimes H_C \otimes H_T$	$H_T \otimes H_C \otimes H_G$	$H_T \otimes H_A \otimes H_T$	$H_T \otimes H_A \otimes H_G$	$H_G \otimes H_C \otimes H_T$	$H_G \otimes H_C \otimes H_G$	$H_G \otimes H_A \otimes H_T$	$H_G \otimes H_A \otimes H_G$
$H_T \otimes H_T \otimes H_C$	$H_T \otimes H_T \otimes H_A$	$H_T \otimes H_G \otimes H_C$	$H_T \otimes H_G \otimes H_A$	$H_G \otimes H_T \otimes H_C$	$H_G \otimes H_T \otimes H_A$	$H_G \otimes H_G \otimes H_C$	$H_G \otimes H_G \otimes H_A$
$H_T \otimes H_T \otimes H_T$	$H_T \otimes H_T \otimes H_G$	$H_T \otimes H_G \otimes H_T$	$H_T \otimes H_G \otimes H_G$	$H_G \otimes H_T \otimes H_T$	$H_G \otimes H_T \otimes H_G$	$H_G \otimes H_G \otimes H_T$	$H_G \otimes H_G \otimes H_G$

Fig. 3 – Tensor family of matrices of 4 genetic gates $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, $\mathbf{H_G}$, 16 duplets and 64 triplets of these gates.

We emphasize that the tensor product of unitary Hadamard operators always yields a unitary Hadamard operator of increased order, conjugate, as noted above, to the corresponding complete orthogonal system of Walsh functions, cyclic Gray codes, the fractal Hilbert curve, logical holography, Walsh antennas, etc. [15-17]. Accordingly, the content of each cell in the $(2^n \bullet 2^n)$-matrices of the tensor family $[\mathbf{H_C}, \mathbf{H_A}; \mathbf{H_T}, \mathbf{H_G}]^{(n)}$ represents a unitary Hadamard operator of the corresponding order. Raising each of these Hadamard gates to an integer power generates a cyclic group of unitary operators characterized by a certain period. It can be shown that the periods of all cyclic groups of gates within each of the matrices of this tensor family are interconnected based on multidimensional hypercomplex numbers. We demonstrate this using the example of the first two matrices of the tensor family under consideration, located at the top of Fig. 3.

The cyclic groups based on raising the alphabetic gates $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, $\mathbf{H_G}$ to integer powers have periods of 8 and 2, as indicated above in expression (1). Figure 4 shows a representation of the matrix $[\mathbf{H_C}, \mathbf{H_A}; \mathbf{H_T}, \mathbf{H_G}]$ from Figure 3 as a matrix **D**, which indicates the values of the periods of the cyclic groups of each of these genetic gates. It is also shown that this matrix **D** is the sum of two sparse matrices $\mathbf{r_0}$ and $\mathbf{r_1}$ with weight coefficients of 8 and 2. But the set of these matrices $\mathbf{r_0}$ and $\mathbf{r_1}$ is closed with respect to multiplication and determines a table of their multiplication (Fig. 4 below), which is known in mathematics as the multiplication table of basic elements of the algebra of 2-dimensional hyperbolic numbers [8, 16]. This means that the period matrix D is a matrix representation of the 2-dimensional hyperbolic number $8+2\mathbf{j}$, where **j** is the imaginary unit of the hyperbolic number.





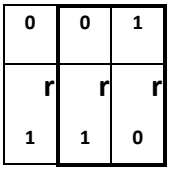


Fig. 4 – Representation of the matrix of four gates $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, $\mathbf{H_G}$ from Fig. 3 in the form of a matrix **D** of periods 8 and 2 corresponding cyclic groups based on raising these gates to integer powers n: $\mathbf{H_C^n}$, $\mathbf{H_A^n}$, $\mathbf{H_T^n}$, $\mathbf{H_G^n}$.-Also shown are the decomposition of this matrix **D** into the sum of two sparse matrices $8\mathbf{r_0}$ and $2\mathbf{r_1}$ and the multiplication table of the matrices $\mathbf{r_0}$ and $\mathbf{r_1}$, which coincides with the multiplication table of the algebra of 2-dimensional hyperbolic numbers.

Let us now turn to the matrix of 16 gate tensor doublets from Fig. 3. The cyclic groups based on raising the terms of this matrix to integer powers have periods of 2, 4, and 8. Fig. 5 shows the (4•4)-matrix $\mathbf{D_2}$, representing this matrix of unitary operators as a matrix of periods of the corresponding cyclic groups of its 16 terms. This matrix $\mathbf{D_2}$ is a bisymmetric Hermitian real matrix. The sums of the components in each of its rows and columns are the same (with appropriate normalization, it becomes a doubly stochastic matrix). As shown in Fig. 5, the matrix $\mathbf{D_2}$ is the sum of 4 sparse matrices $\mathbf{s_0}$, $\mathbf{s_1}$, $\mathbf{s_2}$, $\mathbf{s_3}$ (the numbering corresponds to the order of their sequence in Fig. 5 from left to right) with weight coefficients 4, 8, 8, 2. The set of these 4 sparse matrices is closed under multiplication and determines their multiplication table, known in mathematics as the multiplication table of basis elements of the algebra of 4-dimensional hyperbolic numbers [8, 16]. This means that the period matrix of the considered cyclic groups of unitary operators $\mathbf{D_2}$ is a matrix representation of the 4-dimensional hyperbolic number $4\mathbf{s_0}+8\mathbf{s_1}+8\mathbf{s_2}+2\mathbf{s_3}$, where $\mathbf{s_0}$ is the identity matrix, and $\mathbf{s_1}$, $\mathbf{s_2}$, $\mathbf{s_3}$ are matrix representations of imaginary units of 4-dimensional hyperbolic numbers.

$$\mathbf{D_2} = \begin{pmatrix} 4 & 8 & 8 & 2 \\ 8 & 4 & 2 & 8 \\ 8 & 2 & 4 & 8 \\ 2 & 8 & 8 & 4 \end{pmatrix} = 4\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + 8\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} + 8\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} + 2\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

•	$\mathbf{s_0}$	$\mathbf{s_1}$	$\mathbf{s_2}$	$\mathbf{s_3}$
$\mathbf{s_0}$	$\mathbf{s_0}$	$\mathbf{s_1}$	$\mathbf{s_2}$	$\mathbf{s_3}$
$\mathbf{s_1}$	$\mathbf{s_1}$	$\mathbf{s_0}$	$\mathbf{s_3}$	$\mathbf{s_2}$
$\mathbf{s_2}$	$\mathbf{s_2}$	$\mathbf{s_3}$	$\mathbf{s_0}$	$\mathbf{s_1}$
$\mathbf{s_3}$	$\mathbf{s_3}$	$\mathbf{s_2}$	$\mathbf{s_1}$	$\mathbf{s_0}$

Fig. 5 – The matrix of periods of 16 cyclic groups, representing the matrix of 16 tensor duplets in Fig. 3 and being the sum of the four shown sparse matrices $\mathbf{s_0}$, $\mathbf{s_1}$, $\mathbf{s_2}$, $\mathbf{s_3}$ with their weight coefficients. The multiplication table of these sparse matrices is shown below.

In general, such a tensor family of $(2^n \bullet 2^n)$ matrices $[\mathbf{H_C}, \mathbf{H_A}; \mathbf{H_T}, \mathbf{H_G}]^{(n)}$, obtained by raising the original alphabetic matrix of 4 genetic Hadamard gates to the tensor power (n), contains 4^n unitary operators, each of which, when raised to an integer power, generates its own cyclic group of unitary operators with a certain period. A detailed analysis of the set of these tensor-generated Hadamard unitary operators remains to be conducted in the future, including the study of their transformations under cyclic permutations of their columns and rows, as well as the study of commutators between individual operators, etc. A living organism represents a huge genetically inherited ensemble of coordinated cyclic processes occurring at all levels of biological organization: molecular, subcellular, cellular, supracellular, and organismal [17, 18]. Therefore, cyclic groups of unitary operators associated with the structural features of genetic informatics are needed for quantum-logical modeling of these ensembles of cyclic bioprocesses.

In addition to the described tensor generation of new unitary operators based on the genetic alphabet of 4 Hadamard gates, there is also another approach to their formation. It consists of constructing multi-block matrices, the blocks of which are the alphabetic Hadamard gates $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, $\mathbf{H_G}$. In this way, unitary matrices are formed, which are, in particular, matrix representations of Hamiltonian quaternions and biquaternions, which are closely related to physics, robotics, artificial intelligence, etc. Thus, thousands of papers in the 20th century alone have been devoted to quaternions and biquaternions in physics [6]. This attention to them is due to the fact that quaternions are closely related to Pauli matrices, the theory of the electromagnetic field, the quantum-mechanical theory of chemical valence, the theory of spins, the rotation of bodies in three-dimensional space, etc. Fig. 6 shows one example of such a construction, in which a unitary Hadamard matrix **Q** arises, which is a matrix representation of the Hamiltonian quaternion.

$$\mathbf{Q} = 2^{-0.5} \begin{array}{|c|c|} \hline \mathbf{H_G} & \mathbf{H_A} \\ \hline \mathbf{-H_A} & \mathbf{H_G} \\ \hline \end{array} = 0.5 \begin{array}{|cc|cc|} \hline 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & 1 \\ \hline 1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \\ \hline \end{array}$$

Fig. 6 - Unitary matrix representation **Q** of the Hamiltonian quaternion.

Indeed, as shown in Fig. 7, this matrix **Q** is the sum of four sparse matrices $\mathbf{v_0}$, $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$: $\mathbf{Q} = 0.5(\mathbf{v_0} + \mathbf{v_1} + \mathbf{v_2} + \mathbf{v_3})$, where $\mathbf{v_0}$ is the identity matrix. The set of these sparse matrices is closed under multiplication and defines a multiplication table for them, which coincides with the well-known multiplication table of the basis elements of the Hamiltonian quaternion algebra [8]. This means that the matrix **Q** is a unitary matrix representation of a quaternion, where $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$ represent

the imaginary units of the quaternion. Quaternions that have unitary matrix representations will be called "unitary quaternions" for brevity.

$$0.5\begin{bmatrix} 1, 1,-1,1 \\ -1, 1, 1,1 \\ 1,-1, 1,1 \\ -1,-1,-1,1 \end{bmatrix} = 0.5\begin{bmatrix} 1,0,0,0 \\ 0,1,0,0 \\ 0,0,1,0 \\ 0,0,0,1 \end{bmatrix} + 0.5\begin{bmatrix} 0,1, 0,0 \\ -1,0, 0,0 \\ 0,0, 0,1 \\ 0,0,-1,0 \end{bmatrix} + 0.5\begin{bmatrix} 0, 0,-1,0 \\ 0, 0, 0,1 \\ 1, 0, 0,0 \\ 0,-1, 0,0 \end{bmatrix} + 0.5\begin{bmatrix} 0, 0,0,1 \\ 0, 0,1,0 \\ 0,-1,0,0 \\ -1, 0,0,0 \end{bmatrix}$$

$$= 0.5(\mathbf{v_0} + \mathbf{v_1} + \mathbf{v_2} + \mathbf{v_3}).$$

•	$\mathbf{v_0}$	$\mathbf{v_1}$	$\mathbf{v_2}$	$\mathbf{v_3}$
$\mathbf{v_0}$	$\mathbf{v_0}$	$\mathbf{v_1}$	$\mathbf{v_2}$	$\mathbf{v_3}$
$\mathbf{v_1}$	$\mathbf{v_1}$	$-\mathbf{v_0}$	$\mathbf{v_3}$	$-\mathbf{v_2}$
$\mathbf{v_2}$	$\mathbf{v_2}$	$-\mathbf{v_3}$	$-\mathbf{v_0}$	$\mathbf{v_1}$
$\mathbf{v_3}$	$\mathbf{v_3}$	$\mathbf{v_2}$	$-\mathbf{v_1}$	$-\mathbf{v_0}$

Fig. 7 – The unitary Hadamard matrix $\mathbf{Q} = 0.5(\mathbf{v_0} + \mathbf{v_1} + \mathbf{v_2} + \mathbf{v_3})$ as a sum of four sparse matrices. The multiplication table of this set of sparse matrices, closed under multiplication and corresponding to the multiplication table of the Hamiltonian quaternion algebra, is shown.

Raising this unitary quaternion to integer powers $\mathbf{Q^n}$ generates a cyclic group of unitary operators with a period of 6. This cyclic group models a number of genetically inherited biological structures. For example, the regular features of human color perception, represented in Newton's 6-sector color circle for the three primary and three complementary colors (Fig. 8), correspond to the cyclic group of the unitary quaternion $\mathbf{Q^n}$; its period contains 6 terms, the algebraic relationships of which correspond to the relationships of these colors in human color perception:

- 1) colors opposite on the circle cancel each other out when superimposed (just like unitary quaternion matrices opposite on the circle, whose addition yields the zero matrix);

- 2) each color on the circle is the sum of the two colors on its sides (the same is true for the corresponding unitary quaternions);

- 3) the three primary colors, like the three complementary colors at the vertices of the two triangles of the "Star of David," cancel each other out when superimposed (similarly, the sum of the unitary quaternions at the vertices of each of the two triangles of the "Star of David" is zero).

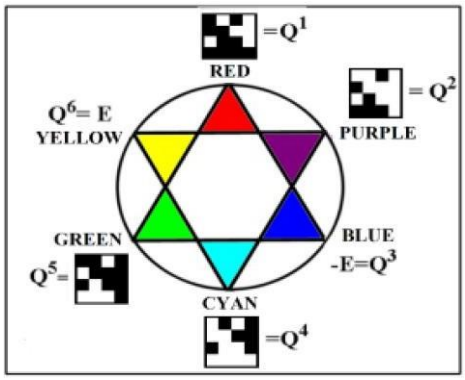


Fig. 8 - Newton's color circle from the psychophysics of color perception and the correspondence to it of the members of the matrix cyclic group of the unitary Hamilton quaternion $\mathbf{Q^n}$, the period of which is equal to 6. In each shown matrix, the black cells contain the numbers "+0.5", and the white ones - "-0.5" (the image is taken from the author's book [20]).

In psychophysics, it is well known that color is not a physical property of an object, but an inherited psychophysical response of a person to the light stimuli coming from the object. In modern literature, the characteristics of color perception, like other characteristics of sensory experience, are referred to as "qualia." The topic of qualia is one of the most pressing and widely discussed in modern philosophy, which sees it as the key to understanding the nature of consciousness. In light of this, it seems significant that the described quantum-logical bioinformation system, based on the genetic alphabet of Hadamard unitary operators, enables algebraic modeling of the properties of human color perception. The author believes that mastering quantum-logical knowledge of "qualia structures"—the inherited structural regularities of sensory experience—allows one to conceptualize and develop qualia technologies for medical, biotechnological, and agricultural purposes.

The limited space of this article does not allow for the presentation of numerous other results and examples of quantum-logical modeling of genetically inherited biostructures based on the alphabet of genetic gates $\mathbf{H_C}$, $\mathbf{H_A}$, $\mathbf{H_T}$, $\mathbf{H_G}$. These will be covered by the author later in a larger publication.

Some concluding remarks

In a quantum-logical approach to bioinformatics, taking into account the cyclical (or pulsating) properties of living bodies, the author relies on a model of biomechanical environments consisting of interconnected pulsating structures that change in a coordinated manner over time. The theory of such model software environments can be used in the development of artificial intelligence, including in connection with systolic processors and pulsating information lattice (pulser)

architectures known in computer technology [7, 10]. The name "pulsating" reflects the essence of this architecture, traditionally compared to a heartbeat or pulse. The pulsation appears as a wave of data, and the computational process appears as the propagation of waves of activity. Data received at the lattice inputs begins to "pulsate" through it, being transformed at each step. The lattice can be configured so that different data streams collide and interact in specific cells at strictly defined intervals, generating a new "pulse" of results.

This architecture is fundamentally different from the von Neumann architecture of conventional processors because it has no central control unit; all cells operate simultaneously and synchronously; data is not written in the classical sense, but continuously "pulses" through the processor structure, like the flow of blood through capillaries. The operation of such a pulsating information grid is compared to the work of the heart: the grid is compared to the muscle tissue of the myocardium; the processing elements are compared to individual muscle cells of the heart (cardiomyocytes); the heartbeat is compared to the electrical impulse from the sinoatrial node; computation is compared to the coordinated contraction of the heart pumping blood; information data is compared to the pumped blood. Moreover, "computation" (pumping blood) is an emergent property of the entire organ, pulsating in a coordinated rhythm; no individual cell is responsible for it, but all cells follow the general rhythm and local interactions. Significant challenges in programming and hardware have prevented pulsating information grids from becoming widespread. The most famous example of the pulsir concept is the Connection Machine, developed by Thinking Machines Corporation in the 1980s. The pulsir concept is also closely related to a number of modern architectural concepts, such as the concept of systolic arrays, named for their pulse-like behavior similar to cardiac systole [11, 12]. Bio-inspired systolic arrays are extremely effective in artificial intelligence, image processing, pattern recognition, computer vision, and other tasks. An example is Google's Tensor Processing Unit (TPU), which uses a large two-dimensional systolic array to perform the extensive matrix multiplications required by neural networks with high efficiency.

Quantum logic in the theory of Hilbert spaces deals not only with unitary operators but also with projection operators, which serve as measurement tools and which are also represented in the inherited functions of living things. An example is the structure of vision based on the projection of light rays onto the retina. One should note that a set of projection matrices **P**, which satisfy the projectors criterion $\mathbf{P} = \mathbf{P}^2$, is associated with the genetic alphabet of unitary Hadamard matrices for example: $\mathbf{P} = 2^{-0.5}[\mathbf{H}_C, -\mathbf{H}_A; -\mathbf{H}_A, \mathbf{H}_C] = 2^{-1}[1,-1,1,-1; 1, 1,-1,-1; 1,-1,1,-1; -1,-1,1,1]$.

The quantum-logical bioinformatics system presented in this article, based on alphabets of unitary operators, is useful for explaining why complex organisms evolved so rapidly: complex

organs and tissues are formed not so much by the emergence of new genes, but by changes in the ways existing genes are used under the influence of quantum-logical operators. The author calls it as "operator quantum-logical Darwinism," according to which natural selection and the inheritance of the most useful ensembles of quantum-logical operators play an important role in biological evolution.

This quantum-logical bioinformatics appears to be useful for the analysis of gene networks and related problems [9]. An important component of the formalisms of this quantum-logical informatics are tensor-unitary transformations, which provide an increase in the dimensionality of configurational vector Hilbert spaces in quantum-logical models of growing biosystems [19].

References

1. Balonin N. A., Sergeev A. M. Hadamard matrices with symmetries in illustrations. SPb.: GUAP, 2024.
2. Birkhoff G., von Neumann J. The Logic of Quantum Mechanics. 1936. Vol. 37, P. 823- 843.
3. Cohen D. An Introduction to Hilbert Space and Quantum Logic. - Springer-Verlag, 1989.
4. Eigen M., Schuster P. The Hypercycle. A Principle of Natural Self-Organization // Springer-Verlag Berlin Heidelberg New York, 1979.
5. Fimmel E., Gumbel M., Karpuzoglu A., Petoukhov S. On comparing composition principles of long DNA sequences with those of random ones // Biosystems. 2019. Vol. 180. P. 101-108.
6. Gsponer A., Hurni J.-P. Quaternions in mathematical physics (2): Analytical bibliography // arXiv:math-ph/0511092v3. 6 July 2008. 124 p. https://doi.org/10.48550/arXiv.math- ph/0511092.
7. Guzik V. F., Shmoilov V. I., Kirichenko G. A. Pulsating information grids with matrix switching // News of universities. North Caucasian region. Series: Technical sciences. 2014. No. 6 (181). URL: https://cyberleninka.ru/article/n/pulsiruyuschie-informatsionnye-reshyotki-s-matrichnoy-kommutatsiey.
8. Kantor I. L., Solodovnikov A. S. Hypercomplex numbers // Springer-Verlag. 1989.
9. Kolchanov N. A., Vishnevsky O. V., Furman D. P., editors. Introduction to Information Biology and Bioinformatics: a textbook // in 5 volumes. Novosibirsk State University. Novosibirsk: RIC NSU. 2015. Vol. 3. 298 p.
10. Kukharev G. A., Tropchenko A. Yu., Shmerko V. P. Systolic processors for signal processing // Minsk: Belarus. 1988. 127 p.
11. Kung H. T., Leiserson C. E. Algorithms for VLSI Processor Arrays // in: Introduction to VLSI Systems, C. Mead and L. Conway (eds.). Addison-Wesley. 1980. Sect. 8.3.

12. Li H., Gao B., Chen Z. et al. A learnable parallel processing architecture towards unity of memory and computing // Sci Rep. 5. 13330. 2015. https://doi.org/10.1038/srep13330.
13. Lieberman-Aiden E., van Berkum N.L., Williams L., Imakaev M., Ragoczy T., Telling A., Amit I., Lajoie B.R., Sabo P.J., Dorschner M.O., et al. Comprehensive mapping of longrange interactions reveals folding principles of the human genome // Science. 326. 289. 2009.
14. Manin Yu. I. Computable and non-computable // M.: Soviet Radio. 1980 (in Russian).
15. Nielsen M. A., Chuang I. L. Quantum Computation and Quantum Information // Cambridge Univ. Press. 2010. (https://doi.org/10.1017/CBO9780511976667).
16. Petoukhov S. V. Matrix genetics, algebras of the genetic code, noise-immunity. – Moscow: Regular and Chaotic Dynamics. 2008. 316 p. (in Russian), https://petoukhov.com/matrix-genetics-petoukhov-2008.pdf .
17. Petoukhov S. V. Genetic code, the problem of coding biological cycles, and cyclic Gray codes // Biosystems. 2024. Vol. 246. 105349. https://doi.org/10.1016/j.biosystems.2024.105349.
18. Petoukhov S. V. Cyclic physiology, cyclic codes and algebraic-logical features of the genetic coding system. Quantum bioinformatics and Ancient Indian philosophy // The International Conference on Science and Spirituality for Global Peace and Harmony IAPIC-2025. Hyderabad: Telangana State. India. April 9-12, 2025. Tutorial, P. 1-10. https://www.iapic.net/IAPIC2025-S01.pdf.
19. Petoukhov S. V. Genetic intelligence and tensor-unitary transformations // System Informatics. 2025. № 27. P. 1-14, DOI 10.31144/si.2307-6410.2025.n27.p1-14 (https://system-informatics.ru/article/372, https://system-informatics.ru/en/article/372 https://petoukhov.com/matrix-genetics-petoukhov-2008.pdf
20. Petoukhov S. V., He M. Algebraic biology, matrix genetics, and genetic intelligence // Singapore: World Scientific. 2023. 616 p., https://doi.org/10.1142/13468.
21. Prasolov V. V. Problems and theorems of linear algebra. 2nd ed. // M.: Regular and Chaotic Dynamics. 2008. 536 p.
22. Rebrova I. M., Rebrova O. Yu. Memory devices based on artificial DNA: birth of the idea and first publications. - Questions in the History of Natural Science and Technology. 2020. Vol. 41. No. 4. Pp. 666–676. DOI: 10.31857/S020596060013006-8.
23. Steitz T. A. Collecting Butterflies and the Protein Structure Initiative: The Right Questions? // Structure. 2007. Vol. 15, Issue 12. P. 1523–1524. DOI 10.1016/j.str.2007.11.005.
24. Vasyukov V. L. Quantum Logic. - M.: PER SE, 2005, 191 p. ISBN 5-9292-0142-0 (in Russian).
25. Vetter R. J., Weinstein S. The history of the phantom in congenitally absent limbs // Neuropsychologia. 1967. № 5. P. 335-338.
26. Weinstein S., Sersen E. A. Phantoms in cases of congenital absence of limbs // Neurology. 1961. #11. P. 905-911.

27. Wiener N. People and Machines. The Last Interview of the “Father of Cybernetics” Norbert Wiener // Izvestia (newspaper). Supplement “Week”. 1964. March 22–28. No. 13 (213). P. 12–13, 20–21 (in Russian).
28. Yamagishi M. E. B. Mathematical Grammar of Biology // Switzerland: Springer International Publishing AG.

UDC 534.11

Development of software that applies artificial intelligence methods to model and analyze vibrations and resonance effects in mechanical systems with changing boundaries

Litvinov V.L. (Samara State Technical University),
Tarakanov A.V. (Samara State Technical University)
Litvinova K.V. (Lomonosov Moscow State University)

This article presents the TB-ANALYSIS software package developed for intelligent analysis and control of resonance dynamics in one-dimensional systems with moving boundaries. Implemented in the MATLAB environment, the package combines classical methods for solving boundary value problems (analytical, asymptotic, and approximate) with artificial intelligence (AI) technologies to predict and prevent resonance phenomena. A key feature of the development is its hybrid architecture, which, along with numerical modeling and factor analysis modules, implements an intelligent module based on deep neural networks (DNNs). This module automatically predicts resonant frequencies based on model parameters and determines optimal values for damping, viscoelasticity, and stiffness coefficients to suppress resonance. Neural networks are trained using synthetic data generated by the package itself. The effectiveness and accuracy of the package have been confirmed by testing on model problems.

Key words: ***resonance, moving boundaries, boundary value problems, mathematical modeling, numerical methods, MATLAB, artificial intelligence, neural networks, parameter optimization***.

1. Introduction

Systems with moving boundaries find wide technical application in areas such as hoisting cables [1–5,7,8], flexible power transmission lines [6], and others. The mobility of the boundaries significantly complicates their mathematical description. Exact solution methods are generally limited to the wave equation and relatively simple boundary conditions [9]. Among approximate approaches, the most effective are the method based on constructing solutions of integro-differential equations [7,11,21–25], as well as the Kantorovich–Galerkin method [8–10,14]. This latter method has been extended to a broader class of model boundary value problems, which takes into account

the bending rigidity of the object, the resistance of the external environment, and the rigidity of the foundation. The solution to the problems is implemented in dimensionless variables using the TB-ANALYSIS software package, developed in the MATLAB environment. This implementation allows the obtained results to be applied to calculations of a wide range of technical objects.

This paper presents a specialized software package, TB-ANALYSIS for intelligent modeling and analysis of the resonant dynamics of objects with moving boundaries. Developed in MATLAB, the package addresses the pressing problem of predicting and preventing destructive resonant phenomena in variable-length systems by combining classical numerical methods with artificial intelligence (AI) approaches. The primary goal of this work is to create a universal tool for studying the dynamics of variable-length systems, analyzing their resonant properties, and determining conditions for preventing resonant phenomena that pose a danger to structures.

The package features a modular architecture and includes functionality for: numerical solution of boundary value problems using intelligent method selection (analytical variable substitution, asymptotic method, and approximate analytical method); factor analysis of the influence of model parameters (drag coefficients, viscoelasticity, stiffness) on resonant modes; and parameter optimization to suppress resonance. Particular attention is paid to the built-in procedure for estimating and monitoring computational errors. The effectiveness and correctness of the package are confirmed by testing results on model problems. A practical application example is a study of transverse vibrations of a viscoelastic rope of variable length on an elastic foundation, with the amplitude dependences on time and the boundary velocity visualized. It is established that the amplitude at zero damping coefficients serves as an upper bound for other cases.

The key advantages of the package include its versatility, automated selection of solution methods, a user-friendly interface, and accuracy assessment tools. Future development prospects lie in expanding the class of problems solved, optimizing algorithms, and, most importantly, integrating artificial intelligence and machine learning methods. In particular, the use of deep neural networks trained on model data enables automated prediction of resonant frequencies and determination of optimal system parameters to prevent resonance. The use of deep neural networks (DNNs), Monte Carlo methods, and adaptive control significantly improves the accuracy of predictions and the efficiency of system control. The neural network is trained using data on system behavior at various frequencies and parameters. The network predicts resonant frequencies and suggests optimal parameters. The AI's results were validated using a mathematical model. Calculations confirmed that the parameters suggested by the artificial intelligence not only prevent resonance but also significantly reduce computation time.

2. Working with the software package

The TB-ANALYSIS software suite features a user interface built around four logically interconnected windows. The initial point of interaction with the environment is the start window, which serves as the central navigation hub. From here, the user can directly navigate to other functional blocks and use a universal menu system accessible in any open interface window. This approach ensures consistency across the various sections of the program.

Control of the system's main features is organized through a graphical main menu, featuring three buttons with intuitive icons. These correspond to the basic modules: "Study of Solutions to Model Boundary Value Problems," "Analysis of Resonance Properties of Models," and "Management of Resonance Phenomena." For increased convenience, all options accessible via these buttons are duplicated in the traditional menu bar, which is present in all working windows except the start window.

The menu bar consists of two main sections. The "File" section contains data management operations: loading initial model settings, saving results in various formats, and terminating a session. When saving, the user can choose to export numerical data in TXT or Excel spreadsheet formats for further processing, or save the generated graphs and visualizations as EPS or PNG files. The "Select Research Direction" section provides quick access to the same three key modules as the graphics menu.

The "Study of Solutions to Model Boundary Value Problems" module focuses on constructing solutions using asymptotic and approximate analytical approaches. It also allows for a comparative evaluation of the effectiveness of these methods and an analysis of the computational error for each. The "Analysis of Resonance Properties of Models" module is designed for factor analysis, which examines the influence of various parameters—such as boundary conditions, vibrational mode number, resonant velocity, damping characteristics, viscoelastic properties, and system stiffness parameters–on the amplitude of the resulting vibrations. The practical module "Resonance Phenomena Management" addresses engineering challenges in identifying zones of resonant instability and enables the identification of combinations of system parameters that reliably prevent the occurrence of dangerous resonant modes.

3. Study of solutions of model boundary value problems using software tools

Clicking the top button in this window takes you to a specialized interface for exploring solutions to boundary value problems, shown in Figure 1.

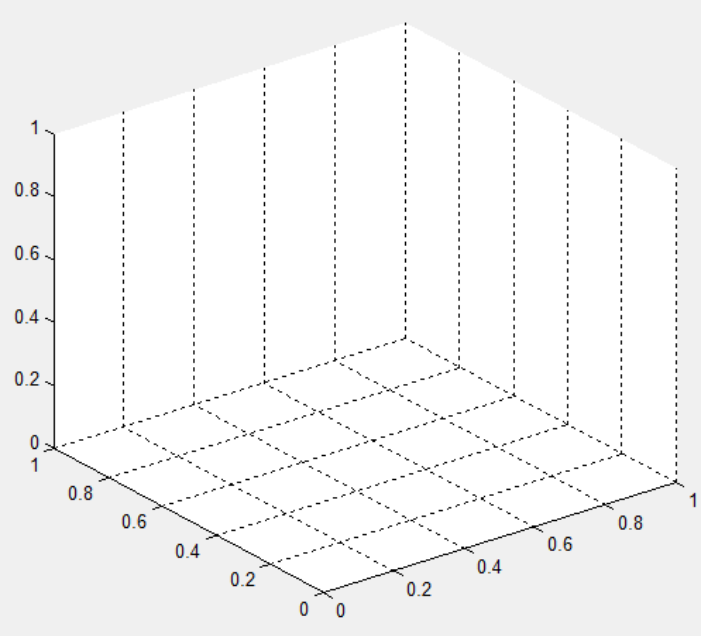


Fig. 1. Window for studying solutions of model boundary value problems

This interface retains the standard "File" and "Research Areas" menu items, making the program easy to use. Basic operations are available through the "File" section: "Import/Load" for batch loading source parameters from .xls/.xlsx and *.txt files, eliminating manual entry; "Export/Save" allows you to save numerical arrays in .xls/.xlsx and *.txt formats via the "Data" sub-item and export graphical results in *.eps and *.png formats via the "Graph" sub-item; and the "Exit" command allows you to completely terminate the application.

The "Research Areas" menu item contains a list of available program modules, including the subitems "Study of Solutions to Model Boundary Value Problems," "Analysis of Resonance Properties of Models," and "Comparison of Solution Methods." Each menu item corresponds to a separate functional block and is used to activate the corresponding software module.

Furthermore, the form contains two panels with radio buttons for "Study Object" and "Oscillation Type," an active "Calculate" button, and groups of windows for manual data entry: "Initial Model Parameters" (with the windows "Material Elastic Modulus (E)," "Initial Longitudinal Strain (epsilon0)," "Axial Moment of Inertia (I)," "Total Length of Undeformed Object (L0)," "Linear Density (r)," and "Cross-Sectional Area (S)"), "Initial Environmental Parameters" (with the "Environmental Drag Force (lambda)" window), and "Boundary Motion Parameters BsinW0(omega0t)." (with the windows “B=”, “W0=”, “omega0=”, “Linear velocity of movement (nu0)”), “Interval of change of coordinate (x)” (with the windows “start (x0)”, “step”, “end (xn)”), “Interval of change of time (t)” (with the windows “Start (t0)”, “Step” and “End (tn)”) and

“Calculation parameters” (with the windows “Accuracy of integral calculation” and “Number of terms of the solution function series”).

The software interface panel includes several key controls: the "Study Object" radio button allows you to select between two model objects—a rope and a beam—while the adjacent "Oscillation Type" panel allows you to specify the vibration type (longitudinal or transverse). The latter option is available exclusively for the "rope" object due to the physical properties of the model. The central control element is the "Calculate" button, which, when activated, initiates computational procedures and visualizes the results as graphical dependencies. A group of manual input fields is provided for setting the initial system parameters, where the names of each parameter clearly correspond to the text labels located to the left of the corresponding input elements.

The software implements three main methods for solving boundary value problems: an analytical method based on the substitution of variables in systems of differential-difference equations; an asymptotic approach for constructing solutions to homogeneous integro-differential equations and systems of ordinary differential equations modeling the dynamics of objects of variable length; As well as an approximate analytical method for solving integro-differential equations describing the motion of mechanical systems with moving boundaries.

The software package automatically selects a computational algorithm based on an analysis of the mathematical model type, the class of integro-differential equation, and the specified boundary conditions. For systems described by the wave equation, an analytical method with a change of variables in differential-difference equations is used; for models based on homogeneous integro-differential equations, an asymptotic method is applied; and for the analysis of complex non-homogeneous integro-differential equations, an approximate analytical method for constructing solutions is used. The algorithmic implementation of the computational methods is organized through a system of internal functions: the analytical method for solving boundary value problems is implemented in the "TBNumAnal" function, the asymptotic method is implemented in the "TBAsym" function, while the approximate analytical method for integro-differential equations is performed by the "TBNum" function. A visualization of the operation of this module is presented in Figure 2, which demonstrates typical calculation results.

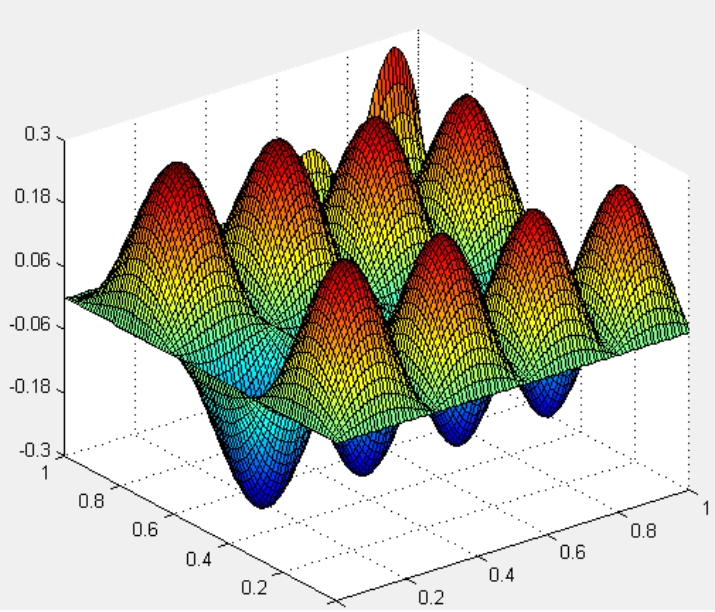


Fig. 2. Graph of the solution function of the boundary value problem for transverse vibrations of the rope

4. Analysis of resonance characteristics of models using a software package

Activating the "Analysis of Resonance Properties of Models" button in the start window takes the user to a specialized interface, the menu structure of which is identical to that of the module for studying boundary value problem solutions.

The resonance analysis module interface includes control panels for configuring study parameters. The "Study Object" radio button allows you to choose between rope and beam models, while the "Dependency Analysis" panel allows you to select the type of study: amplitude-time analysis or maximum amplitude-velocity analysis. Computational procedures are implemented through a system of built-in functions: the "met_ampl" function for calculating amplitude-time characteristics and the "met_ampl_max" function for analyzing the maximum amplitude-velocity dependence, which uses the first function as a subroutine.

This block also provides the ability to modify key model parameters, such as the mode number, damping coefficients, object stiffness, viscoelastic properties, and substrate stiffness, through the corresponding input fields. The interactive "Calculate" button is used to launch calculation procedures and subsequently display the resulting dependencies as graphs.

The dependence of maximum amplitude on speed is determined using an original method developed as part of this study. The method is based on an analytical expression for the oscillation

amplitude, which is derived from the solution of integro-differential equations, taking into account the resonant characteristics of the modeled mechanical systems.

$$A_n^2(\tau) = E_n^2(\tau)\left\{\left[\int_0^\tau F_n(\zeta)\cos\Phi_n(\zeta)d\zeta\right]^2 + \left[\int_0^\tau F_n(\zeta)\sin\Phi_n(\zeta)d\zeta\right]^2\right\}. \quad (1)$$

The algorithm for numerically studying steady-state resonance and the phenomenon of passing through resonance is implemented in the "met_ampl_max" function. Figure 3 shows a graph of the dependence of the maximum amplitude of rope oscillations when passing through resonance on the boundary velocity for various values of the medium resistance coefficient (from top to bottom: $\lambda = 0; \lambda = 0,01; \lambda = 0,02$) with the following model parameters: mode number 1; object stiffness coefficient 0.01; viscoelasticity coefficient 0.01; substrate stiffness coefficient 0.02.

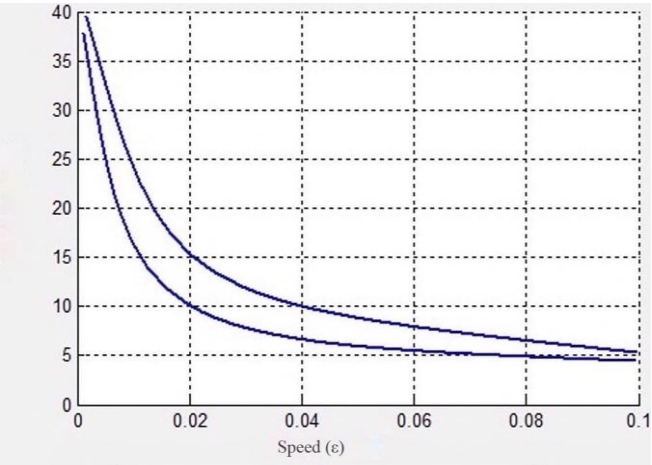


Fig. 3. Graph of the dependence of the maximum amplitude on the velocity of boundary movement for different values of the medium resistance coefficient

The calculation of the time dependence of the oscillation amplitude according to formula (1) is implemented in the internal function "met_ampl", which is used as a subroutine in the function "met_ampl_max". Figure 4 presents the results of test calculations demonstrating the change in the amplitude of transverse oscillations of a variable-length rope when passing through resonance in the first dynamic mode for a specific set of initial model parameters.

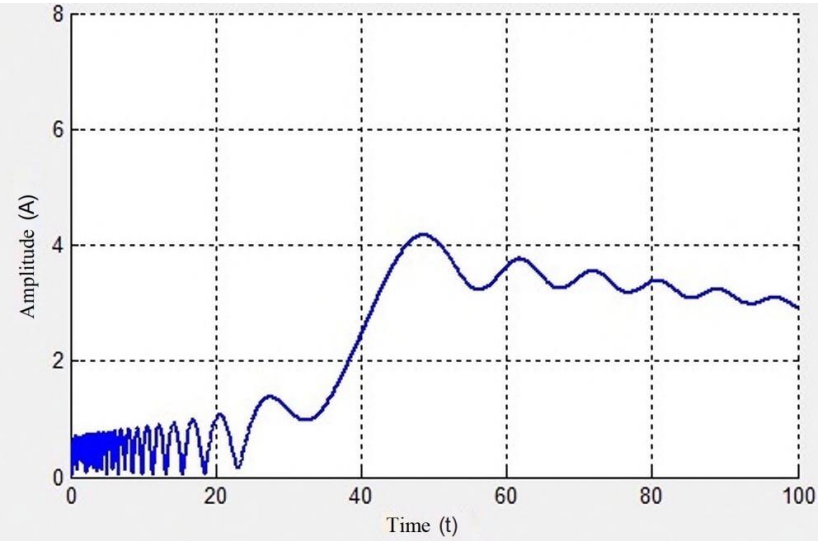


Fig. 4. Graph of amplitude versus time

5. Application of artificial intelligence to the example of vibrations of a variable-length rope

In addition to their direct functional role, the graphical dependencies shown in Figure 4 clearly demonstrate the characteristic features of the behavior of the oscillation amplitude, which form the basis of the method for determining the maximum amplitude.

Consider an example of using a neural network to predict resonant frequencies. Let the initial parameters of the system be given in dimensionless form:

-Rope stiffness: $k_0 = 100$,

-Damping: $c = 0.05$,

-Boundary velocity: $v_0 = 0.1$.

The neural network predicts a resonant frequency $\omega_n = 5$ and an amplitude $A_n = 0.2$. The allowable amplitude is $A_{allow} = 0.1$.

1. Calculate the loss function:

$$L_{res} = (0.2 - 0.1)^2 = 0.01.$$

2. Calculate the gradient:

$$\nabla_{\mathbf{p}} L_{res} = 2(0.2 - 0.1)\frac{\partial A_n}{\partial \mathbf{p}}.$$

3. Update the parameters:

$$\mathbf{p}_{new} = \mathbf{p}_{old} - \eta \nabla_{\mathbf{p}} L_{res}.$$

After several iterations, the system parameters are optimized, and the oscillation amplitude is reduced to the allowable level.

The developed software package TB-ANALYSIS is designed to solve a specific class of one-dimensional boundary value problems with moving boundaries, as well as for mathematical modeling and analysis of resonance properties of objects whose states are described by these boundary value problems. The package also enables the optimization of model parameters to prevent resonance phenomena using artificial intelligence (AI). The complex was developed in MATLAB as a standalone application. The results of this study have been incorporated into the package.

6. Conclusions

The TB-ANALYSIS software suite has proven itself as a versatile and reliable tool for mathematical modeling and analysis of the resonance characteristics of mechanical systems with moving boundaries. Testing has confirmed its high effectiveness in solving a wide range of boundary value problems.

Working with the suite is easy thanks to its intuitive interface, equipped with an intelligent system for selecting solution methods. The reliability of the obtained results is ensured by a built-in computational error assessment system. In this study, using the TB-ANALYSIS suite, enhanced with artificial intelligence technologies, we successfully determined the resonant frequencies of the system and developed conditions for resonance prevention. The initial system parameters were optimized to minimize the likelihood of resonance occurrence.

The study relied on collecting the system's amplitude-frequency characteristics, which allowed the identification of key parameters that have the greatest impact on resonance phenomena. A neural network capable of predicting resonant frequencies and suggesting optimal system settings was trained using this data. To address data limitations, the Monte Carlo method was extensively utilized. Machine learning methods were employed for an in-depth analysis of the parameters that contribute most to resonance.

All recommendations generated by artificial intelligence were thoroughly tested using a mathematical model. The calculations convincingly confirmed the high effectiveness of the proposed parameters in preventing resonance. Future development of TB-ANALYSIS lies in expanding its functionality and adapting it to solve more complex classes of problems, opening new horizons for research in the field of mechanical system dynamics. The approach presented in this paper, based on the integration of artificial intelligence and machine learning methods, not only

improves calculation accuracy but also significantly reduces the time required to determine the optimal parameters for complex dynamic systems.

References

1. Kolosov L.B., Zhigula T.I. Longitudinal–transverse vibrations of the rope of the lifting installation // Izv. Universities. Mining Journal. 1981. No. 3. P. 83–86.
2. Zhu W.D., Chen Y. Theoretical and experimental investigation of elevator cable dynamics and control // J. Vibr. Acoust. 2006. No. 1. P. 66–78.
3. Shi Y., Wu L., Wang Y. Nonlinear analysis of natural frequencies of a tether system // J. Vibr. Eng. 2006. No. 2. P.173–178.
4. Wang L., Zhao Y. Multiple internal resonances and non–planar dynamics of shallow suspended cables to the harmonic excitations // J. Sound Vib. 2009. No. 1–2. P. 1–14.
5. Zhao Y., Wang L. On the symmetric modal interaction of the suspended cable: three–to one internal resonance // J. Sound Vib. 2006. No. 4–5. P.1073–1093.
6. Vesnitsky A.I. Waves in systems with moving boundaries and loads. Moscow: Fizmatlit, 2001. 320 p.
7. Goroshko O.A., Savin G.N. Introduction to the mechanics of deformable one–dimensional bodies of variable length. Kiev: Nauk. dumka, 1971. 290 p.
8. Litvinov V.L., Anisimov V.N. Transverse vibrations of a rope moving in a longitudinal direction // Bulletin of the Samara Scientific Center of the Russian Academy of Sciences. 2017. T. 19.No. 4. – P.161–165.
9. Litvinov V.L., Anisimov V.N. Mathematical modeling and research of oscillations of one–dimensional mechanical systems with moving boundaries: monograph / V. L. Litvinov, V. N. Anisimov – Samara: Samar. state tech. un–t, 2017 .– 149 p.
10. Lezhneva A.A. Free bending vibrations of a beam of variable length // Uchenye zapiski. Perm: Perm. un–t, 1966. No. 156. P. 143–150.
11. Savin G.N., Goroshko O.A. Variable length thread dynamics. Kiev: Nauk. Dumka, 1962. 332 p.
12. Litvinov V.L. Investigation of free vibrations of mechanical objects with moving boundaries using the asymptotic method // Zh. Middle Volga Mathematical Society. 2014. T. 16.No. 1. P.83–88.
13. Liu Z., Chen G. Analysis of Plane Nonlinear Free Vibrations of a Carrying Rope Taking into Account the Influence of Flexural Rigidity // J. Vibr. Eng. 2007. No. 1. P. 57–60.
14. Litvinov V.L., Anisimov V.N. Application of the Kantorovich – Galerkin method for solving boundary value problems with conditions on moving boundaries // Bulletin of the Russian Academy of Sciences. Rigid Body Mechanics. 2018. No. 2. P. 70–77.
15. Litvinov V.L., Anisimov V.N. Transverse vibrations of a rope moving in a longitudinal direction // Bulletin of the Samara Scientific Center of the Russian Academy of Sciences. 2017. T. 19. No. 4. – P.161–165.

16. Litvinov V.L., Anisimov V.N. Mathematical modeling and research of oscillations of one – dimensional mechanical systems with moving boundaries: monograph / V. L. Litvinov, V. N. Anisimov – Samara: Samar. state tech. un–t, 2017 .– 149 p.
17. Elepov B. S., Kronberg A. A., Mikhailov G. A. and Sabelfeld K. K. Solution of boundary value problems by the Monte Carlo method. – Novosibirsk: Nauka, 1980. – 174 p.
18. Ermakov S. M. Monte Carlo Method in Computational Mathematics: An Introductory Course. – St. Petersburg: Nevsky Dialect; M.: BINOM. Knowledge Laboratory, 2009. – 192 pp.
19. Fishman G. S. Monte Carlo. Concepts, Algorithms, and Applications. — SpringerVerlag, 1995 (Corrected 3 rd printing, 1999). — 718 pp.
20. Litvinov V.L. Stochastic longitudinal oscillations of a viscoelastic rope with moving boundaries taking into account the action of damping forces // Probability Theory and Its Applications, RAS, 2022, Vol. 67, No. 4, pp. 835–836.
21. Litvinov V.L. Solution of boundary value problems with moving boundaries using an approximate method for constructing solutions of integro–differential equations // Tr. Institute of Mathematics and Mechanics, Ural Branch of the Russian Academy of Sciences. 2020.Vol. 26, №. 2. P. 188–199.
22. Litvinov V.L., Litvinova K.V. An approximate method for solving boundary value problems with moving boundaries by reduction to integro–differential equations// Computational Mathematics and Mathematical Physics, 2022, vol. 62, no. 6, pp. 945–954.
23. Litvinov V.L. Certificate of state registration of a computer program. Automated software package for studying oscillations and resonance phenomena in mechanical systems with moving boundaries "TB–Analysis–7" No. 2025613649, published 13.02.2025.
24. Litvinov V.L., Litvinova K.V. On an inverse method for solving problems of oscillations of mechanical systems with moving boundaries // Bulletin of Moscow University. Series 1: Mathematics. Mechanics. - 2024. - No. 3. - P. 53-59.
25. Litvinov V.L., Litvinova K.V. Application of the Kantorovich-Galerkin method for analyzing the resonance characteristics of damped systems // Journal of Theoretical and Mathematical Physics, 2025, Vol. 224, No. 1, pp. 129-138.
26. Selivanova N.Yu., Shamolin M.V. Local solvability of a one–phase problem with free boundary // Journal of Mathematical Sciences. – 2013. – Vol. 189, no. 2. – P. 274–283.
27. Selivanova N.Yu., Shamolin M.V. Studying the interphase zone in a certain singular–limit problem // Journal of Mathematical Sciences. – 2013. – Vol. 189, no. 2. – P. 284–293.
28. Erofeev V.I., Lisenkova E.E. Quasi-harmonic longitudinal wave propagating in a Mindlin–Hermann rod immersed in a nonlinear elastic medium // Theoretical and Mathematical Physics. – 2022. – Vol. 211, No. 2. – P. 216–235.
29. Erofeev V.I., Lisenkova E.E. General relations for waves propagating in one-dimensional elastic systems // Mathematical methods of mechanics: proceedings of the international conference. On the

90th Anniversary of Academician A.G. Kulikovsky. – Moscow: Steklov Mathematical Institute, 2023. – Pp. 26–27.

30. Litvinov V.L., Shamolin M.V. On One Asymptotic Method for Solving Homogeneous Integro–Differential Equations Describing Oscillations of Objects with Moving Boundaries // Siberian Journal of Industrial Mathematics, 2025, Vol. 28, No. 2, pp. 39–54.
31. Semenov A.L., Litvinov V.L., Shamolin M.V. Study of the Influence of Boundary Movement on the Oscillatory and Resonant Properties of Mechanical Systems of Variable Length // Computational Mathematics and Information Technologies, Vol. 9, No. 2, pp. 34–43.

УДК 004

Visual Graph Library: архитектура и возможности Java-библиотеки для визуализации графов

Золотухин Т.А. (Институт систем информатики им. А.П. Ершова СО РАН)

В статье описывается архитектура и возможности Visual Graph Library — библиотеки для хранения, укладки и визуализации иерархических атрибутированных графов с портами. Рассматриваются ключевые модули системы и интеграция с платформой Java. Приводится анализ существующих библиотек и приложений для работы с графами.

Ключевые слова: *иерархические атрибутированные графы с портами, графовые представления потоковых программ, визуализация графов, системы визуализации графов.*

1. Введение

Визуализация графов является фундаментальным инструментом анализа сложных программных и технических систем. Представление синтаксических деревьев, управляющих графов и структур вызовов в наглядной форме критически важно для верификации, оптимизации и проектирования современного ПО [1, 3, 7].

Однако инструменты общего назначения имеют архитектурные ограничения при работе со специфическими структурами — **иерархическими атрибутированными графами, содержащими порты**. Базовый функционал таких библиотек редко позволяет задать жесткую привязку дуг к конкретным точкам на границе вершин, а также корректно обработать их многоуровневую вложенность. Адаптация подобных решений под требования инженерного ПО неизбежно ведет к усложнению архитектуры приложения и реализации избыточных программных надстроек. Кроме того, большинство существующих инструментов не предоставляет гибких средств программного управления сложными графовыми структурами напрямую в рамках Java-экосистемы.

Библиотека **Visual Graph Library (VGL)** создана на основе архитектурного ядра системы «Visual Graph», разработанной в рамках исследования методов укладки и навигации в иерархических атрибутированных графах [4, 26]. В ходе развития проекта архитектурные решения были переосмыслены и унифицированы, а графическое ядро — изолировано от прикладной логики. Это позволило трансформировать специализированный инструментарий

в универсальную библиотеку для Java-экосистемы, предлагающую открытую модульную архитектуру для свободного использования.

Цель настоящей статьи — представить архитектуру VGL, описать модель хранения графовых структур, реализованные алгоритмы укладки и средства визуализации. Также в работе проводится обзор существующих аналогов и обосновывается необходимость разработки библиотеки VGL.

2. Архитектура библиотеки

Архитектура библиотеки построена на принципах модульности и разделения ответственности. В основу системы положен паттерн MVC (Model-View-Controller) [17], позволяя изолировать данные от механизмов их обработки и отображения. Структурно библиотеку можно разделить на следующие модули:

- Модуль хранения графов является обязательным ядром системы. Оно инкапсулирует всю логику работы с графовыми структурами, их топологией, иерархией, атрибутами и портами. Любое взаимодействие с библиотекой начинается с создания экземпляра хранилища, так как все остальные модули используют его в качестве единственного источника данных.
- Модуль импорта и экспорта графов обеспечивает преобразование внешних форматов во внутренние объекты модуля хранения и обратно.
- Модуль укладки графов предоставляет набор алгоритмов для автоматического вычисления пространственных характеристик графа. Укладчики модифицируют исключительно геометрические атрибуты элементов в хранилище, не затрагивая их семантику или топологию.
- Модуль визуализации отвечает за графическую интерпретацию графовых структур, находящихся в модули хранения графов, а также обработку действий пользователя (масштабирование, навигация, выделение элементов).

Подобная декомпозиция позволяет использовать библиотеку как «конструктор», адаптируя её под конкретные задачи без избыточности кода. Такой подход минимизирует связанность компонентов и упрощает внедрение библиотеки в существующие сторонние проекты. Детальный обзор каждого модуля и особенности их программной реализации будут приведены в последующих главах.

3. Хранение графов

Эффективная организация хранения графовых структур является фундаментом для их дальнейшей обработки и визуализации. Простых графовых структур, где вершины изображаются точками, а дуги прямыми, зачастую не хватает для прикладных инженерных задач, где элементы обладают сложной семантикой и иерархической структурой. Для решения таких задач в VGL было решено использовать **иерархические атрибутированные графы с портами**. Определение такой структуры строится на объединении трех понятий из теории графов:

- **Иерархический граф** определяется как кортеж (G, T), в котором G – это граф произвольного типа, а T – дерево вложенности, вершины которого соответствуют элементам некоторой иерархии в G, дуги же отражают отношение их непосредственной вложенности [3].
- **Граф с портами** определяется как кортеж (V, E, P, π), в котором функция $\pi: P \to V$, сопоставляет каждому порту единственную вершину, которой он принадлежит. Множество дуг E определяется как подмножество объединения трех типов декартовых произведений: $E \subseteq (V \times V) \cup (P \times V) \cup (V \times P)$.
- **Атрибутированный граф с портами** определяется как кортеж $(V, E, P, A, \pi, \alpha_V, \alpha_E, \alpha_P)$, в котором функции $\alpha_V: V \to A, \alpha_E: E \to A, \alpha_P: P \to A$ расширяют структурную модель графа с портами $G = (V, E, P, \pi)$. Эти функции сопоставляют вершинам, дугам и портам конечные наборы атрибутов из множества A. Элементами множества A являются пары вида $(name, value)$, где $name$ – имя атрибута, а $value$ – его значение.

Для реализации описанной выше теоретической концепции была спроектирована программная модель данных, основу которой составляют следующие сущности:

- Атрибут – типизированная запись вида «ключ – значение», расширяющая семантику любого элемента графа. В библиотеке атрибуты разделяются на пользовательские, описывающие данные исходной задачи, и системные, создаваемые библиотекой для управления визуализацией и состоянием (например, параметры шрифта, толщина границ или флаг выделения элемента). Для значений поддерживаются следующие типы данных: строковый, числовой (целые и вещественные числа), логический, уникальные идентификаторы, а также сложные структуры в формате JSON. Атрибуты позволяют гибко настраивать как визуальное представление, так и прикладную логику без внесения изменений в программную модель данных.

Несмотря на то что атрибут является элементом графа, он является исключением и не может быть расширен за счет других атрибутов.

- Графовая модель – логический контейнер и корневой идентификатор. В рамках физического хранения, идентификатор модели является обязательным ключом для доступа к любому элементу. Графовая модель является элементом графа и может быть расширена атрибутами, которые хранят некоторый набор метаданных для обмена состоянием между модулями системы.
- Вершина — элементарная единица топологии и основной объект для алгоритмов обработки, является элементом, а значит обладает набором атрибутов, которые могут хранить например его геометрические параметры, такие как координаты и размеры.
- Фрагмент — вершина, содержащая вложенный граф. Это программная реализация, позволяет рассматривать фрагмент как «черный ящик» для вышестоящего уровня иерархии.
- Порт — специализированная вершина, помеченная соответствующим атрибутом. Такая унификация позволяет использовать для портов стандартные механизмы поиска и навигации, принятые для обычных вершин.
- Фиктивный порт — вспомогательный порт, помеченный соответствующим атрибутом. Данный элемент создается библиотекой автоматически, когда дуга должна связать вершины, находящиеся в разных фрагментах.
- Вершина с портами — специфический вид фрагмента, который не содержит других элементов, кроме портов.
- Дуга — элемент, связывающий две вершины, находящиеся в одном фрагменте. Информация о том, к каким конкретно портам привязана дуга, хранится в ее атрибутах. Это позволяет библиотеке, делегировать расчеты конкретных точек входа/выхода алгоритмам укладки и отрисовки.
- Цепочка — логическая абстракция, представляющая собой последовательность сегментированных дуг и фиктивных портов. Она описывает сквозной путь между вершинами, которые находятся в разных фрагментах. Благодаря декомпозиции связи на локальные сегменты, алгоритмы обработки (например, укладка) могут работать с каждым подграфом изолированно, не нарушая границ инкапсуляции.

Центральным компонентом модуля хранения является хранилище графовых моделей. Данный компонент инкапсулирует логику работы с элементами и может рассматриваться как

модель в контексте классического паттерна MVC. Хранилище служит единой точкой доступа для всех потребителей — от алгоритмов укладки до модуля визуализации — и может передаваться между ними как самостоятельная единица. Техническая реализация позволяет использовать для хранения элементов как оперативную память, так и дисковое пространство, в зависимости от задач пользователя. Основные возможности данного компонента:

- Один экземпляр хранилища может управлять множеством независимых графовых моделей. Разделение данных происходит по уникальному идентификатору модели, что позволяет работать с ними параллельно, сохраняя полную изоляцию данных в рамках одного объекта.
- Базовый набор операций для создания, поиска и модификации элементов графа дополнен внутренней логикой контроля целостности. Это гарантирует, что все изменения структуры проходят с сохранением корректного состояния связей и атрибутов в любой момент времени.
- Встроенный механизм обхода графовой модели в глубину с поддержкой настраиваемых обработчиков позволяет внешним алгоритмам работать с иерархией элементов и концентрироваться на логике обработки данных, не дублируя код рекурсивного обхода в каждом модуле.
- Встроенные средства копирования данных обеспечивают миграцию графовых моделей или их фрагментов между различными экземплярами хранилищ (например, из оперативной памяти в дисковое пространство). Процесс гарантирует полное сохранение структурной целостности и атрибутивной семантики переносимых данных.

Таким образом, разработанный модуль определяет набор сущностей с расширяемой семантикой и механизм хранения, отделяющий данные от логики их обработки. Это позволяет алгоритмам укладки, навигации и отрисовки взаимодействовать с графовыми структурами через единый интерфейс, абстрагированный от способа их физического размещения.

4. Визуализация графов

Если модуль хранения определяет программную модель данных и способы управления для графовых структур, то модуль визуализации отвечает за их графическую интерпретацию. В рамках архитектуры VGL визуализация рассматривается не как жестко детерминированный процесс отрисовки элементов, а как динамическое преобразование набора атрибутов, закрепленных за элементами, в визуальные примитивы. Такой подход позволяет полностью отделить логику представления от структурных данных: модуль визуализации выступает внешним потребителем хранилища графовых моделей, восстанавливая визуальный облик всей

модели или её отдельных частей на основе системных и пользовательских атрибутов элементов, входящих в них.

Процесс отрисовки опирается на иерархический обход графовой модели, описанный в предыдущей главе. Модуль визуализации последовательно анализирует каждый полученный элемент и транслирует семантику его атрибутов в параметры графического контекста, формируя облик конкретных элементов:

- Визуализация вершины начинается с расчета её геометрических размеров, динамически определяемых на основе списка выбранных пользовательских атрибутов. Использование моноширинного шрифта позволяет точно вычислить ширину и высоту текстовой области по количеству знаков и размеру кегля. С учетом внутренних отступов и толщины границ полученный текстовый блок вписывается в выбранную геометрическую форму (прямоугольник, эллипс или ромб), что и определяет итоговые габариты вершины. Результаты расчета фиксируются в системных атрибутах, позволяя либо сразу выполнить отрисовку в координатах по умолчанию, либо использовать эти данные в модуле укладки для расчета финальной позиции с последующим сохранением координат в соответствующих атрибутах.
- Визуализация порта технически реализуется аналогично вершине, при этом алгоритмам укладки рекомендуется располагать их на границах элементов, к которым они относятся. Фиктивные порты отображаются в виде малых окружностей серого цвета. Реальный порт графически полностью идентичен вершине, что позволяет через его атрибуты задавать любую форму, цвет и состав отображаемых пользовательских атрибутов.
- Визуализация дуги представляет собой отрисовку линии, стиль которой (пунктирный или сплошной), цвет и толщина задаются атрибутами. Для направленных связей (дуг) на целевом конце формируется наконечник-стрелка, отсутствующий у ненаправленных. Дуга может включать текстовую метку, расчет размеров которой аналогичен логике вершины. Координаты точек маршрутизации (изломов) задаются модулем укладки; по умолчанию соединение строится по кратчайшему пути между геометрическими центрами фигур с автоматическим расчетом точек привязки на их границах. При отрисовке цепочек, состоящих из сегментов в разных фрагментах, обеспечивается их бесшовная стыковка для создания эффекта непрерывной линии связи сквозь все уровни иерархии.
- Визуализация фрагмента принципиально отличается от вершины тем, что его итоговые размеры напрямую зависят от размеров и расположения всех входящих в

него элементов, что делает их производными от содержимого нижних уровней иерархии. Расчет текстовой области заголовка фрагмента выполняется по тем же принципам, что и для вершины, однако позиционирование данной области делегируется модулю укладки. Наиболее простой формой фрагмента является прямоугольник, хотя допустимы и другие варианты в зависимости от возможностей конкретного алгоритма укладки. Важной особенностью является то, что внешние связи с внутренними элементами должны проходить через специализированные порты (реальные или фиктивные), располагаемые на границе фрагмента. Это обеспечивает визуальную упорядоченность иерархических переходов, позволяя рассматривать фрагмент как целостный функциональный блок.

После определения правил отрисовки элементов и общей философии их расположения необходимо рассмотреть программный инструментарий, реализующий эти принципы. В системе VGL разработан набор компонентов, позволяющий использовать библиотеку как для автономной генерации изображений на серверах, так и в составе интерактивных приложений. В этом наборе каждый последующий компонент расширяет возможности предыдущего и может быть рассмотрен как представление в контексте классического паттерна. Всего их три:

- **Статический компонент** — низкоуровневое ядро отрисовки, отвечающее исключительно за трансляцию элементов в графические примитивы, не содержит логики обработки событийной модели графического интерфейса, что позволяет использовать его для генерации отчетов на сервере. Результатом работы является графическое изображение, которое может быть сохранено в файл, передано по сети или выведено на печать. Важной особенностью компонента является поддержка выборочной отрисовки области: вместо формирования всего изображения целиком, компонент может отрисовать только заданную прямоугольную область. Это позволяет существенно экономить ресурсы памяти и процессора, а также служит фундаментом для работы механизмов оптимизации в последующих компонентах.
- **Кэширующий компонент** — слой оптимизации, выступающий декоратором для статического компонента. Его использование необходимо при работе с большими изображениями, содержащими большое количество элементов, так как без этого компонента невозможно добиться плавности при перемещении видимой области или масштабировании изображения. Согласно исследованиям [12], критические задержки при отрисовки значительно затрудняют когнитивный анализ структуры данных. Для решения этой проблемы реализован механизм, который запрашивает у статического компонента отрисовку области, размер которой превышает текущие

границы видимости, и сохраняет результат в памяти. При смещении видимой области в границах кэша изображение формируется мгновенно путем копирования из памяти. Перерисовка запрашивается только в тех случаях, когда требуемая область оказывается за границами кэшированного региона. Обновление кэша вынесено в фоновый поток, что сохраняет отзывчивость интерфейса и сводит к минимуму сценарии, в которых пользователю приходится ожидать завершения отрисовки.

- **Интерактивный компонент** — графический компонент, отвечающий за управление областью видимости изображения, а также обработку действий пользователя, таких как сигналы от мыши и клавиатуры. Он использует кэширующий компонент, и в целях оптимизации в нем реализована система слоев, которые накладываются друг на друга: изображение со структурой графа находится на самом нижнем уровне, тогда как динамические объекты, такие как рамка выбора или подсветка элементов, отрисовываются поверх на прозрачных плоскостях. Это позволяет ускорить отклик интерфейса, не затрагивая основное изображение и не задействуя вычислительные ресурсы нижестоящих компонентов.

Таким образом, разработанный модуль определяет правила отрисовки на основе семантики атрибутов, что позволяет динамически интерпретировать элементы графа в графические примитивы. При этом общая философия позиционирования элементов создает базу для работы алгоритмов укладки для достижения оптимального визуального результата. Разделение программного инструментария на статический, кэширующий и интерактивный компоненты позволяет полностью изолировать процесс отрисовки от логики управления интерфейсом. Применение механизмов выборочной отрисовки области, кэширования и системы наложенных слоев обеспечивает плавную навигацию по изображениям графов с большим количеством элементов, сохраняя возможность использования библиотеки для генерации отчетов на сервере.

5. Укладка графов

Развитие алгоритмов автоматической укладки прошло путь от упрощенных математических моделей до сложных систем визуализации инженерных данных. В этой ретроспективе можно выделить три ключевых стадии:

- Исторически первой стадией была визуализация графа как системы безразмерных точек, соединенных прямыми линиями. Основной задачей здесь являлось достижение эстетической симметрии и равномерного распределения вершин, что

решалось преимущественно силовыми алгоритмами. Эти подходы детально систематизированы в профильных фундаментальных работах [3, 7]. Однако отсутствие учета физических размеров вершин и текстовых пометок у дуг, делало их малопригодными для проектирования современных систем.

- С развитием CASE-средств и языков моделирования (UML) возникла необходимость отображать вершины как прямоугольные блоки с текстом. Сложность данной задачи заключалась в том, чтобы вершины не перекрывали друг друга, а дуги не проходили поверх вершин. Наиболее эффективными как с точки зрения производительности, так и с точки зрения визуального результата стали алгоритмы, основанные на поуровневой укладке графа [9, 20].
- Дальнейшее усложнение структур привело к тому, что «плоское» визуальное представление графа становилось перенасыщенным и малопригодным для анализа. Решением стала концепция иерархических графов, основанная на принципе декомпозиции: части графа группируются во вложенные подграфы (фрагменты), что существенно снижает когнитивную нагрузку на пользователя. Этот этап ознаменовал принципиальный сдвиг в теории укладки. Если классические алгоритмы оперировали фиксированными метриками элементов, то в иерархических моделях процесс становится рекурсивным: внутренняя геометрия элементов фрагмента определяет его габариты, что, в свою очередь, напрямую влияет на всю топологию уровня выше [1, 3, 7, 19].

Подход, реализованный в библиотеке VGL, логически продолжает эту эволюцию, объединяя иерархическую декомпозицию с детальной маршрутизацией связей. Ключевым отличием является перенос точек соединения с границ вершин на специализированные интерфейсные элементы — порты. В рамках данной модели порт выступает не абстрактной точкой привязки, а самостоятельным геометрическим объектом с фиксированными размерами. Это накладывает дополнительные ограничения на алгоритмы укладки: траектории должны рассчитываться с учетом не только вложенности структур, но и габаритов самих портов для обеспечения корректных углов примыкания.

Модуль укладки графов реализован как совокупность автономных вычислительных компонентов, логика которых полностью изолирована от других модулей. Архитектурно эти компоненты выступают функциональными аналогами методов контроллера в парадигме MVC. Каждый компонент инкапсулирует конкретный алгоритм размещения и взаимодействует напрямую с хранилищем графовых моделей, принимая внешние параметры для настройки алгоритма и определения границ вычислений — от обработки всей модели до

укладки конкретного фрагмента. Несмотря на свободу реализации внутренних алгоритмов, все они объединены общей философией, которая базируется на следующих принципах:

- Алгоритм укладки должен строго следовать правилам интерпретации атрибутов в геометрические примитивы и их пространственное расположение (в частности — для фрагментов, портов и маршрутов дуг), изложенным в главе «Визуализация графов».
- Область ответственности ограничивается вычислением пространственного расположения элементов, маршрутизацией дуг и расчетом габаритов для фрагментов. Алгоритм не должен изменять параметры визуального стиля, такие как размеры текстовых полей, характеристики шрифтов, цветовые схемы, геометрические формы элементов и т.д.
- Алгоритм укладки должен корректно обрабатывать вложенные структуры. Для этого может быть использован иерархический обход графовой модели, описанный в главе «Хранение графов».
- По завершении работы алгоритм укладки фиксирует результаты вычислений в хранилище графовых моделей, обновляя координаты вершин, маршруты дуг, а также расположение и размеры фрагментов в соответствующих атрибутах.
- Алгоритм укладки должен обеспечивать корректную работу с портами. Порядок портов и их координаты должны учитываться как обязательные параметры при расчете расположения элементов и маршрутизации дуг.

В библиотеке VGL реализованы следующие алгоритмы укладки, которые позволяют эффективно визуализировать графовые структуры различных типов. Каждая укладка адаптирована для работы с иерархическими атрибутированными графами с портами:

- **Иерархическая укладка** на базе метода Сугиямы [8, 20] реализует рекурсивный обход структуры, начиная с наиболее вложенных фрагментов. Внедрение концепции цепочек на уровне хранения графов позволяет отойти от обработки связей между элементами из разных фрагментов. Весь граф рассматривается как набор независимых фрагментов, что существенно упрощает и ускоряет алгоритм. В рамках данной укладки все порты разделяются на входные (располагаются на верхней границе фрагмента или вершины) и выходные (на нижней). При этом реальные порты имеют определенный порядок, который выступает жестким ограничением и не может нарушаться алгоритмом, в то время как фиктивные порты лишены фиксированного порядка, что дает алгоритму необходимую степень свободы для

оптимизации пересечений и выравнивания линий «на лету». Такая геометрия в сочетании с выравниванием вершин по вертикальным осям минимизирует изломы дуг, что делает изображение более читаемым. Завершающая стадия определяет стиль линий: прямые дуги отрисовываются сплошными, а обратные — пунктирными. При этом для обратных дуг соблюдается общий принцип: они выходят из нижней границы элемента и входят в верхнюю, что полностью согласуется с принятой концепцией расположения портов.

- **Круговая укладка** [16] размещает вершины по периметру концентрических окружностей, что наиболее эффективно для анализа сетевых структур без выраженного направления потока данных. Реализация алгоритма опирается на последовательные проходы «вверх» и «вниз» по уровням иерархии: на восходящем этапе вычисляются габариты вложенных фрагментов, а на нисходящем — уточняются их финальные координаты и углы поворота. Использование двух проходов несколько снижает производительность по сравнению с однопроходными методами, однако это позволяет достичь более эстетичного результата за счет итерационного уточнения позиций элементов. В рамках данной укладки порты распределяются по границам окружностей, при этом реальные порты сохраняют свой жесткий порядок, а фиктивные служат свободными точками для оптимизации межуровневых связей. Минимизация пересечений внутри окружности достигается за счет эвристик группировки наиболее связанных элементов и циклической перестановки смежных элементов. Завершающая стадия маршрутизации использует «скругленные вставки», которые позволяют дугам обтекать неинцидентные вершины по радиусу. Это предотвращает визуальное наложение дуг на элементы и сохраняет чистоту иерархических переходов даже при высокой плотности графа. Пошаговая логика работы алгоритма, адаптированного под специфику иерархии и портов VGL, представлена ниже в виде трех основных этапов:

  1. Расчет размеров фрагментов. Используем восходящий обход дерева вложенности T – это гарантирует, что при обработке текущего фрагмента уже известны геометрические размеры всех его составляющих элементов. На их основе вычисляется радиус R для позиционирования внутренних вершин и фрагментов, а также определяется r как максимальный размер одного из портов. Итоговый размер фрагмента определяется формулой $2R + r + \mathrm{c}, \text{где } \mathrm{c} - \text{константа}$. Данное значение остается неизменным на протяжении всего алгоритма.

2. Расчет позиций элементов. Используем нисходящий обход дерева вложенности T. Процесс позиционирования внутри фрагмента выполняется в строгой последовательности. Сначала полученные от родительских структур координаты части портов конвертируются в базовый порядок, на который накладывается жестко заданный порядок реальных портов. В результате формируется набор портов с точными или аппроксимированными позициями, а также множество полностью свободных портов. Опираясь на зафиксированные позиции, выстраивается начальная последовательность внутренних элементов, которая затем оптимизируется эвристической сортировкой для получения их финального топологического порядка. Отсортированные элементы расставляются на несущей окружности. Далее выполняется вычисление оптимального угла поворота сформированного кольца элементов с целью минимизации длины связующих дуг до портов с известными позициями. На завершающем шаге производится финальное распределение аппроксимированных и свободных портов по периметру фрагмента.
3. Маршрутизация дуг. На завершающем этапе выполняется маршрутизация дуг. Благодаря применению концепции «цепочек», задача маршрутизации строго локализована: алгоритм обрабатывает исключительно сегменты, соединяющие элементы внутри границ одного фрагмента (все межуровневые связи предварительно декомпозированы). Для каждого такого локального соединения вычисляются точные координаты точек входа и выхода: позиция определяется как точка пересечения луча, исходящего из геометрического центра инцидентного элемента, с его фактической границей. Между вычисленными точками строится базовая прямолинейная траектория. В случае возникновения коллизии (если прямая пересекает тело неинцидентного элемента) применяется алгоритм обтекания препятствий: базовая траектория модифицируется так, чтобы дуга плавно огибала препятствие по контуру вспомогательной окружности.

В рамках представленных алгоритмов основное внимание было сосредоточено на задачах вычисления габаритов и точного позиционирования элементов. Маршрутизация связей реализовывалась по остаточному принципу с использованием достаточно стандартных подходов (прямолинейные соединения и базовое обтекание препятствий). Тем не менее, при анализе графов с высокой плотностью дуг такой подход может приводить к избыточному

визуальному шуму и засорению итоговой схемы. Для решения этой проблемы перспективным вектором развития выглядит применение методов жгутования дуг [5]. Данная техника позволила бы стягивать близлежащие дуги в единые пучки, существенно повышая читаемость.

6. Импорт и экспорт графов

Модуль импорта и экспорта графов представляет собой набор независимых компонентов, логика которых полностью изолирована от задач визуализации и хранения данных. Архитектурно эти компоненты выступают функциональными аналогами методов контроллера в парадигме MVC: их роль заключается в преобразовании внешних форматов во внутренние объекты библиотеки и обратно. Компоненты взаимодействуют напрямую с хранилищем графовых моделей, обеспечивая интерпретацию топологии и метаданных независимо от типа источника данных — будь то локальный файл, сетевой поток или область памяти. Несмотря на разнообразие поддерживаемых форматов, все компоненты данного модуля объединены общей философией, базирующейся на четырех принципах:

- Любой процесс обмена данными (импорт или экспорт) должен быть абстрагирован от физического носителя. Использование принципов потоковой обработки позволяет изолировать логику разбора формата от специфики работы с файловой системой или сетевыми протоколами.
- Процесс импорта обязан обеспечивать строгую валидацию входных данных. Любое нарушение синтаксиса или логической целостности графа должно сопровождаться детальным описанием ошибки с указанием контекста (номер строки, некорректный фрагмент), необходимого для оперативной отладки внешних файлов.
- Процесс экспорта должен обеспечивать сохранение не только топологии, но и полного визуального состояния графа. Это подразумевает обязательную фиксацию координат, габаритов элементов, маршрутов дуг и примененных стилей, что гарантирует идентичность модели при её повторной загрузке.
- Реализация должна быть универсальной по отношению к иерархическим структурам. Независимо от ограничений конкретного формата, процесс импорта или экспорта должен корректно восстанавливать или сохранять вложенность фрагментов, типизированные атрибуты и метаданные, описанные в главе «Хранение графов».

Для практической реализации описанных выше принципов в библиотеку VGL включен набор компонентов, адаптированных под специфику инженерных данных и иерархических атрибутированных графов с портами:

- **Компоненты для импорта и экспорта графов в формате GraphML** [22] на базе XML. Формат GraphML выступает в качестве основного способа хранения и обмена данными для библиотеки VGL, т.к. он обеспечивает полную поддержку объектной модели описанной в главе “Хранение графов”, а именно вложенные графы, типизированные атрибуты и порты. Стоит так же отметить что при экспорте, в итоговый файл, записываются не только топология, но и результаты работы модулей визуализации и укладки: координаты элементов, маршруты дуг и габариты фрагментов. Это позволяет восстановить результаты работы при последующем импорте.
- **Компонент для импорта графов в формате DOT** [21] обеспечивает совместимость с экосистемой инструментов Graphviz. Поддержка данного формата позволяет использовать библиотеку как средство визуализации и интерактивного анализа для графов, сгенерированных сторонними инженерными и диагностическими утилитами.
- **Компонент для импорта графов в формате GML** [13] является востребованным в задачах сетевого анализа и академических исследованиях благодаря своей высокой читаемости. Несмотря на то что официальная спецификация GML описывает исключительно «плоские» структуры и не поддерживает иерархию и порты, в VGL реализован механизм восстановления вложенности и портов через использование специальных атрибутов. Таким образом, плоская структура формата преобразуется в полноценную иерархическую модель с портами.

Текущий набор компонентов обеспечивает решение основных задач по обмену данными, потоковой загрузке и сохранению визуального состояния моделей. Список поддерживаемых форматов будет расширяться по мере развития библиотеки. Благодаря открытости архитектуры сторонние разработчики могут реализовывать собственные модули импорта и экспорта под специфические нужды, не дожидаясь выхода новых версий библиотеки и не внося изменений в её основной код.

7. Анализ существующих библиотек и приложений для работы с графами

Выбор инструмента для визуализации сложных инженерных систем требует баланса между алгоритмической мощностью, производительностью графической подсистемы и гибкостью программного управления. В данной главе проведен обзор графовых решений: от

низкоуровневых алгоритмических библиотек до развитых систем визуализации. Цель анализа — определить место разрабатываемой библиотеки среди существующих инструментов, выявить их сильные и слабые стороны, а также обозначить сценарии их применения.

Текущий рынок графового ПО представлен широким спектром инструментов, различающихся по архитектуре, языкам реализации и назначению: yFiles [25], Gephi Toolkit [10], Tulip Library [23], Cytoscape.js [6], Graphviz [11], Higres [18], JGraphT [15], igraph [14], OGDF [24], NetworkX, Boost Graph Library, JGraphX, GraphStream, Prefuse, Sigma.js, G6.js, D3.js, Vis.js, Graph-tool, JUNG, Piccolo2D, GoJS, KeyLines, NetBeans Visual Library, Zest. Подробный разбор каждого из них избыточен и приведет к потере фокуса исследования. Чтобы сохранить системность изложения, мы ограничим область анализа, последовательно просеивая инструменты по ключевым критериям.

Начнем процесс просеивания с исключения инструментов, не имеющих активного цикла поддержки: Prefuse, JUNG, Piccolo2D, JGraphX (mxGraph), GraphStream, Higres, Zest и NetBeans Visual Library. Отсутствие обновлений в течение длительного времени делает их интеграцию в современные системы нецелесообразной из-за рисков несовместимости с актуальными программными средами и неоправданного роста затрат на сопровождение такого кода.

Далее стоит исключить графовые СУБД (Neo4j, JanusGraph), предназначенные для персистентного хранения и обработки запросов к связанным данным. Подобные системы не содержат механизмов автоматизированной укладки и интерактивной визуализации. В рамках разработки графической подсистемы СУБД рассматриваются исключительно как возможные внешние источники данных, но не как инструменты построения интерактивных схем.

Также стоит исключить инструменты, базирующиеся на веб-окружении: D3.js, Sigma.js, Cytoscape.js, G6.js, Vis.js, GoJS и KeyLines. Несмотря на развитый функционал данных библиотек, их использование в нативном Java-ПО требует встраивания компонентов WebView. Подобный подход создает значительную архитектурную избыточность, усложняет прямой доступ к структурам данных в памяти JVM и ограничивает производительность интерфейса. При визуализации графов размерностью более 10 000 элементов наличие прослойки между графическим движком браузера и основным кодом приложения становится критическим узким местом.

Значимым критерием является технологическая совместимость с целевой средой. Это ограничение обуславливает отказ от NetworkX вследствие низкой производительности Python-интерпретатора на массивных графах и Graph-tool из-за его жесткой привязки к контейнеризированной инфраструктуре (Docker), неприемлемой для кроссплатформенного

десктопного ПО. Вне фокуса исследования остается и Boost Graph Library: при всей своей алгоритмической полноте она лишена графического ядра, а специфика её реализации на шаблонах C++ делает создание JNI-оберток избыточно трудоемким.

После первичного отсева инструментов, не соответствующих базовым требованиям по поддержке и архитектуре, оставшаяся группа требует детального разбора. Оценка проводится по вектору «от модели к реализации»: от фундаментальных возможностей до прикладных аспектов встраивания в инженерное ПО. В итоговый перечень для детального функционального сопоставления вошли нативные Java-библиотеки (yFiles, Gephi Toolkit, JGraphT) и группа низкоуровневых решений (igraph, OGDF, Tulip, Graphviz). Несмотря на сложность интеграции внешнего кода и ориентированность некоторых инструментов на генерацию статических схем, статус данных решений как индустриальных стандартов обосновывает их сохранение в выборке. Их анализ позволит определить технологический предел существующих систем и сформировать перечень требований к разрабатываемой библиотеке, подтвердив необходимость её создания в текущем технологическом ландшафте.

Эффективность применения рассматриваемых библиотек (yFiles, JGraphT, Gephi, Tulip, igraph, OGDF, Graphviz) в инженерных задачах зачастую ограничена архитектурой их внутренних моделей данных. В большинстве решений (JGraphT, igraph, Gephi) поддержка вложенных графов носит декларативный характер: алгоритмы автоматического размещения оперируют плоской топологией и не учитывают границы контейнеров. В инструментах с нативной иерархией (yFiles, Graphviz) работа со сложными структурами осложнена либо трудоемкостью программной настройки, либо статической природой кластеров. Аналогичное несоответствие наблюдается в реализации портов: если в yFiles и Graphviz они являются лишь логическими точками привязки, то в OGDF или JGraphT специализированный интерфейс для них отсутствует. Это приводит к трассировке ребер в геометрический центр узла и затрудняет построение схем с ортогональной укладкой без создания значительных надстроек над графическим ядром. Дополнительным барьером выступает изоляция атрибутов данных от их визуального воплощения. В yFiles или Tulip для отображения метаданных требуется разработка собственных компонентов отрисовки, а отсутствие механизмов автоматической синхронизации вынуждает разработчика вручную обновлять графический слой при смене внутреннего состояния объектов. Данные структурные ограничения предопределяют сложности не только при управлении графом в памяти, но и при попытках его корректного сохранения или передачи через внешние форматы.

Поддержка стандартов (GraphML, DOT, GML) в рассматриваемых библиотеках носит фрагментарный характер, что ограничивает их использование в единых инженерных

процессах. Решения общего назначения (JGraphT, igraph, Gephi) поддерживают DOT и GraphML, однако ограничиваются передачей топологии, игнорируя визуальные параметры и иерархию. Среда yFiles, напротив, сохраняет графические атрибуты в расширенном GraphML, но лишена нативной поддержки DOT, что исключает прямое взаимодействие с экосистемой Graphviz. Библиотеки, изначально разработанные на C++ (Tulip, OGDF), отдают приоритет собственным (TLP) или упрощенным (GML) форматам, что затрудняет трансляцию их специфических данных в Java-приложения без написания внешних адаптеров. Общей проблемой для всех инструментов остается несовместимость схем метаданных: при передаче графа инженерные атрибуты (порты, физические параметры, вложенность) либо игнорируются, либо сохраняются в нетипизированном виде. В результате стандартные форматы превращаются в средство передачи лишь «голой» топологии, а задача корректного восстановления полной инженерной модели перекладывается на разработчика, вынужденного создавать индивидуальные парсеры для каждой библиотеки.

Анализ алгоритмов укладки выявляет несоответствие между методами визуализации общих сетей и требованиями инженерного проектирования. Большинство открытых библиотек (JGraphT, igraph, Gephi, Tulip) ориентированы на силовые алгоритмы, которые эффективны для анализа связей, но не гарантируют сохранения структурной иерархии. Это приводит к потере целостности составных графов: алгоритмы зачастую игнорируют границы контейнеров, размещая дочерние узлы вне родительских областей. Качественные иерархические методы (алгоритм Сугиямы), минимизирующие пересечения, полноценно представлены лишь в yFiles и Graphviz, однако они функционируют как «черные ящики», затрудняя жесткую привязку ребер к конкретным портам. Для библиотеки OGDF характерно наличие сложной математической базы ортогональной трассировки, но её адаптация в Java-экосистеме требует значительных усилий по настройке параметров обхода тел компонентов. В отсутствие нативной поддержки портов большинство рассматриваемых решений используют маршрутизацию дуг от центра к центру, что критически снижает читаемость инженерных схем. Таким образом, рынок предлагает выбор между дорогостоящими закрытыми продуктами и трудоемкой разработкой собственных механизмов контроля геометрии на базе инструментов, изначально не рассчитанных на работу с портами и вложенностью.

Завершающим критерием анализа является простота интеграции библиотек в Java-экосистему и их способность обеспечивать интерактивное взаимодействие. Лидером сегмента остается yFiles, предлагающий нативные высокоуровневые компоненты для Swing и JavaFX. Библиотека поддерживает высокую производительность за счет механизмов виртуализации и

многоуровневой детализации, что позволяет сохранять плавность навигации при масштабировании схем. В противоположность этому, Graphviz ориентирован на статическую генерацию: его использование в Java-интерфейсах ограничено отображением готовых файлов, что исключает динамическое манипулирование узлами. Такие платформы как Gephi, обладают мощными средствами навигации и обработки данных, но несмотря на наличие программных интерфейсов для управления логикой, их графические движки тесно интегрированы с родительскими платформами (NetBeans, OSGi), что затрудняет встраивание интерактивного компонента в стороннее ПО в качестве изолированного виджета. Библиотеки общего назначения, такие как JGraphT, обладают развитым математическим аппаратом, но их визуализация часто вынесена в отдельные модули (например, JGraphX), что требует ручной синхронизации объектной модели и графического представления. Инструменты с C++ ядром (igraph, OGDF, Tulip) при встраивании через JNI механизмы создают риски нестабильности и усложняют кроссплатформенное развертывание. Отсутствие поддержки консольного режима во многих инструментах, описанных выше, ограничивает их применение на серверах: автоматическая генерация отчетов становится невозможной без настройки виртуального графического окружения.

8. Заключение

В статье представлена архитектура и возможности Visual Graph Library — специализированного библиотечного решения для работы с иерархическими атрибутированными графами с портами. Целью данной библиотеки является устранение технологического разрыва между низкоуровневыми алгоритмическими библиотеками и закрытыми графическими системами, чье использование затруднено в открытых инженерных проектах. Проведенный анализ показал, что существующие решения зачастую игнорируют специфику инженерных задач, связанных с вложенными графами и/или портами или накладывает критические ограничения на стабильность и развертывание в рамках Java-экосистемы.

Visual Graph Library реализует системный подход, объединяющий модель хранения сложных графовых структур, алгоритмы автоматической укладки и подсистему отрисовки в рамках единой модульной архитектуры. Ключевой особенностью данной системы является ее ориентированность на прикладные инженерные задачи, требующие работы с вложенностью и семантикой портов в среде Java. Разработанные подходы применимы для визуализации внутренних представлений программ в компиляторах (например, GCC), средах исполнения (например, Python), а также специализированных системах, таких как Cloud Sisal [2].

Визуальный анализ таких структур используется для отладки трансформаций кода и верификации алгоритмов оптимизации. Поддержка иерархии позволяет корректно группировать инструкции в базовые блоки, а строгая семантика портов — точно отслеживать зависимости по данным. Это обеспечивает построение читаемых схем, облегчающих разработчикам анализ внутреннего состояния транслятора.

Основные результаты:

- Реализован механизм гибридной схемы хранения на диске и в памяти, поддерживающий вложенные графы, порты и типизированные атрибуты, что позволяет корректно моделировать потоки данных и управляющие графы.
- Реализован набор специализированных компонентов, в модуле визуализации графов, обеспечивающий стабильную работу как в интерактивных интерфейсах, так и в серверных сценариях без жесткой зависимости от графического окружения ОС.
- Реализована поддержка портов и вложенности в алгоритмах Сугиямы и круговой укладки, что позволяет автоматизировать процесс построения читаемых схем для сложных инженерных задач.
- Разработан модуль импорта и экспорта, поддерживающий сериализацию иерархических атрибутированных графов с портами в формат GraphML, а также их десериализацию из форматов GraphML, DOT и GML.

Направления дальнейших исследований и разработки:

- Разработка графического компонента для иерархического отображения графа с поддержкой отложенной загрузки вложенных графов.
- Развитие библиотеки для решения проблем масштабируемости и обработки графов сверхбольшого размера. Оптимизация алгоритмов кэширования и индексации в рамках гибридной модели хранения для минимизации задержек при обращении к диску на графах с миллионами узлов.
- Оптимизация производительности алгоритмов укладки для графов сверхвысокой плотности путем внедрения параллельных вычислений.
- Внедрение алгоритма жгутования в качестве опционального режима маршрутизации, необходимого для предварительного визуального анализа графов с высокой плотностью дуг.

Библиотека распространяется с открытым исходным кодом, что позволяет использовать её как базу для создания собственных систем для визуализации и анализа сложно структурированной информации большого объема на основе графовых моделей.

Список литературы

1. **Касьянов В. Н.** Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. Новосибирск: ИСИ СО РАН, 1999. – С. 7–32.
2. **Касьянов В.Н., Гордеев Д.С., Золотухин Т.А., Касьянова Е.В., Кондратьев Д.А.** Система облачного параллельного программирования CPPS: визуализация и верификация Cloud Sisal программ : моногр. / Под ред. В.Н. Касьянова ; Ин-т систем информатики им. А.П. Ершова СО РАН. – Новосибирск: ИПЦ НГУ, 2020. – 256 с. – (Сер.: Конструирование и оптимизация программ; Вып. 22). ISBN 978-5-4437-1123-2. DOI: https://doi.org/10.31144/978-5-4437-1123-2.
3. **Касьянов В. Н., Евстигнеев В. А.** Графы в программировании: обработка, визуализация и применение. СПб.: БХВ-Петербург, 2003. – 1104 с.
4. **Касьянов В. Н., Золотухин Т. А.** Visual Graph — система для визуализации сложноструктурированной информации большого объема на основе графовых моделей // Научная визуализация. 2015. Т. 7, № 4. – С. 44–59.
5. **Apanovich Z.V., Kislicyna T.A., Vinokurov P.S.** Tools for Visual Analysis of Information Content of Portals Included in Linked Open Data Cloud // Proceedings of the 13th All-Russian Scientific Conference "Digital libraries: Advanced Methods and Technologies, Digital Collections". Voronezh, Russia, October 19-22, 2011. CEUR Workshop Proceedings. 2011. – Volume 803. – P. 113–120.
6. **Cytoscape**. [Электронный ресурс]. URL: https://cytoscape.org/ (дата обращения: 24.02.2026).
7. **Di Battista G., Eades P., Tamassia R., Tollis I. G.** Graph Drawing: Algorithms for Visualization of Graphs. Prentice Hall, 1999. – 397 p.
8. **Eiglsperger M., Siebenhaller M., Kaufmann M.** An Efficient Implementation of Sugiyama's Algorithm for Layered Graph Drawing // Journal of Graph Algorithms and Applications. – 2005. – Vol. 9, № 3. – P. 305–325.
9. **Gansner E. R., Koutsofios E., North S. C., Kiem-Phong Vo** A Technique for Drawing Directed Graphs // IEEE Transactions on Software Engineering. – 1993. – Vol. 19, № 3. – P. 214–230.
10. **Gephi**. [Электронный ресурс]. URL: https://gephi.org/ (дата обращения: 24.02.2026).
11. **Graphviz.** [Электронный ресурс]. URL: https://graphviz.org/ (дата обращения: 24.02.2026).
12. **Herman I., Melancon G., Marshall M. S.** Graph visualization and navigation in information visualization: a survey // IEEE Trans. on Visualization and Computer Graphics. – 2000. – Vol. 6, Issue 1. – P. 24–43.
13. **Himsolt M.** GML: A portable graph file format. – Technical Report, University of Passau, 1996. – 8 p.
14. **igraph.** [Электронный ресурс]. URL: https://igraph.org/ (дата обращения: 24.02.2026).
15. **JGraphT.** [Электронный ресурс]. – URL: https://jgrapht.org/ (дата обращения: 24.02.2026).

16. **Kasyanov V. N., Merculov A. M., Zolotuhin T. A.** A circular layout algorithm for attributed hierarchical graphs with ports // Journal of Physics: Conference Series. – 2021. – Vol. 2099, № 1. – Article ID 012051.
17. **Krasner G. E., Pope S. T.** A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80 // Journal of Object-Oriented Programming. – 1988. – Vol. 1, № 3. – P. 26–49.
18. **Lisitsyn I. A., Kasyanov V. N.** Higres – visualization system for clustered graphs and graph algorithms // Lecture Notes in Computer Science. – 1999. – Vol. 1731. – P. 82–89.
19. **Sugiyama K.** Graph drawing and applications. For software and knowledge engineers. – World Scientific, 2002. – 232 p.
20. **Sugiyama K., Tagawa S., Toda M.** Methods for Visual Understanding of Hierarchical System Structures // IEEE Transactions on Systems, Man, and Cybernetics. – 1981. – Vol. SMC-11, № 2. – P. 109–125.
21. **The DOT Specification.** Graphviz.org. [Электронный ресурс]. URL: https://graphviz.org/doc/info/lang.html (дата обращения: 24.02.2026).
22. **The GraphML Specification.** [Электронный ресурс]. URL: http://graphml.graphdrawing.org/ (дата обращения: 24.02.2026).
23. **Tulip.** [Электронный ресурс]. URL: https://tulip.labri.fr/ (дата обращения: 24.02.2026).
24. **OGDF**. [Электронный ресурс]. URL: https://ogdf.uos.de/ (дата обращения: 24.02.2026).
25. **yFiles.** [Электронный ресурс]. – URL: https://www.yworks.com/products/yfiles (дата обращения: 24.02.2026).
26. **Zolotuhin T.** Visual Graph: an interactive system for the visualization of hierarchical attributed graphs // Bulletin of the Novosibirsk Computing Center. Series: Computer Science. Issue 37. – 2014. – P. 163–180.

UDK 004.85 : 539.1.074 + 519.65

Machine learning augmented Tikhonov regularization with iterative approach for stable neutron spectrum unfolding

Chizhov K.A. (Joint Institute for Nuclear Research, Dubna State University)

Shirkov S.G. (Joint Institute for Nuclear Research)

Chizhov A.V. (Joint Institute for Nuclear Research)

A hybrid multi-stage algorithm is developed for solving the ill-posed inverse problem of unfolding the neutron energy spectrum from multi-sphere Bonner spectrometer measurements. Traditional approaches, such as Tikhonov regularization and iterative methods, have significant limitations due to the subjective choice of the regularization parameter or initial approximation, which compromises the solution's stability and accuracy. In the proposed method, the first stage automated machine learning (autoML) is used to find the optimal model to predict the global spectral shape. The second stage applies Tikhonov regularization, where regularization parameter is objectively optimized based on a similarity metric relative to the autoML prediction. The smoothing functional is minimized using convex optimization techniques. The third stage utilizes the obtained solution as the initial guess for an iterative refinement procedure. Physical prior knowledge is incorporated both through a parametrically generated training dataset (weighted sums of fission, evaporation, Gaussian, and high-energy spectral components). The hybrid approach demonstrates superior robustness to noisy input data compared to methods using solely Tikhonov regularization or machine learning. The developed methodology is applicable to neutron dosimetry at high-energy nuclear facilities and for solving a broad class of inverse problems described by Fredholm integral equations of the first kind.

Keywords: neutron spectrum unfolding, Tikhonov regularization, machine learning, autoML, iterative methods, Bonner sphere spectrometer, inverse problems.

1. Introduction

1.1. Problem Relevance

Accurate characterization of a wide range neutron energy spectra from eV to hundreds of MeV is necessary for radiation protection, shielding design and dosimetry for personnel of particle accelerators. Neutrons (as uncharged particles) require indirect detection methods. The Bonner Sphere Spectrometer (BSS) as a thermal neutron detector surrounded by polyethylene

moderators of varying diameters, is the most widely used instrument for neutron spectrometry [1]. However, calculating the neutron energy spectrum $\varphi(E)$ from the set of BSS readings constitutes an ill-posed inverse problem, described by a system of Fredholm integral equation of the first kind. This ill-posedness, characterized by non-uniqueness and high sensitivity to measurement noise, necessitates specialized unfolding techniques.

1.2. Problem Statement

The relationship between the neutron spectrum ϕ and the BSS readings for M spheres is given by:

$$Q_j = \int_{E_{\min}}^{E_{\max}} R_j(E)\,\phi(E)\,dE, \quad j = 1, \ldots, M, \tag{1}$$

where $R_j(E)$ is the response function of the j-th sphere [2]. Q_j is the Bonner spectrometer reading including measurement error ϵ, E – energy.

Since the unfolding of the neutron spectrum is assumed to be a fairly wide energy range that exceeds 11 orders of magnitude, to reduce the error of numerical integration it is convenient to convert it to a new variable called lethargy $u(E) = \lg(E/E_{\min})$:

$$\ln 10 \times \int_0^{l_E} K_j(u)\,\phi(u)\,E(u)\,du = Q_j, \quad j = 1, \ldots, M, \tag{2}$$

where $l_E = \lg(E_{\max}/E_{\min})$.

The integrand form suggests finding $\varphi(u) = \phi(u) \cdot E(u)$, which we continue to call the neutron spectrum.

Discretizing the energy into N bins ($N \gg M$) transforms (2) into a system of linear equations:

$$\mathbf{A}\boldsymbol{\varphi} = \mathbf{q}, \tag{3}$$

where $\mathbf{A} \in \mathbb{R}^{M\times N}$ is the response matrix, $\boldsymbol{\varphi} \in \mathbb{R}^N$ is the discretized fluence spectrum, and $\mathbf{q} \in \mathbb{R}^M$ is the vector of measured counts. The system is underdetermined and ill-conditioned.

This inverse problem can be reformulated as a machine learning regression task. The input features are the M normalized BSS readings $\tilde{q}_j = q_j/k_j$, where $k_j = \sum_{i=1}^{M} q_i$. The target output is the discretized spectrum vector $\boldsymbol{\varphi}$. The ML model $\mathcal{F}$ learns the mapping:

$$\tilde{\boldsymbol{\varphi}}_{\text{pred}} = \boldsymbol{k}\mathcal{F}(\tilde{\mathbf{q}}). \tag{4}$$

1.3. Existing Approaches and Related Work

Neutron spectra unfolding methods include iterative algorithms (e.g., MLEM [3], Landweber [4]), maximum entropy deconvolution (MAXED) [5], Tikhonov regularization [6], TSVD [4], stochastic methods like genetic algorithms [8], parametric [9, 10] and other methods [11]. However, the results of application of a method depends on the chosen parameters. For iterative methods, the initial approximation and the choice of the number of iterations are important. For regularization methods, the choice of the regularization parameter can over smooth the solution or create nonphysical ripples in the spectrum if the parameter is too small. In unfolding process it is necessary to consider the physical correctness of the spectrum: smoothness and non-negativity. Conditional stability of the solution is needed, other war small errors in BSS readings may produce large uncertainty in spectra.

Machine learning [13] and neural networks (NN) [7, 14] of different types and architecture has been increasingly applied to this problem, including deep learning frameworks with input feature transformations [15], Kolmogorov-Arnold networks [16], radial basis function [17], convolutional [18], generalized regression [19] and Bayesian NN [20]. For interpretation of the neural network spectrum prediction, explainable artificial intelligence (XAI) methods such as SHAP [13], LIME [21], ANFIS-based [22] were applied.

ML models are typically trained on synthetic datasets generated via Monte Carlo codes [18] or parametric models [23]. A set of real spectra may also be incorporated into the dataset [13] with assumption that these spectra have been accurately unfolded via the methods employed by the authors. These datasets pair a neutron spectra with their corresponding BSS responses.

Finding the optimal model requires iterating over both the algorithms themselves and the model's hyperparameters, as well as consuming a significant amount of computational time. automatic ML AutoML frameworks [24], which include algorithms for efficiently finding hyperparameters within a given time, help with this.

However, data-driven ML approaches can struggle with generalization to spectra outside the training distribution and may lack physical consistency, for example producing negative fluence values or being overly sensitive to input noise. Therefore, the authors proposed in this work a synergistic approach: an ML model provides a rapid, data-driven initial guess, which is then refined and regularized by well-established iterative physical models, combining the speed of ML with the stability and physical grounding of traditional methods, as it was made in [25] for the dataset of 201 neutron energy spectra and the corresponding measured responses from [2] .

In this paper we propose a method with combination of algorithms, where on the first

stage autoML trained on a large synthetic dataset is used to find the optimal model to predict the global spectral shape. This shape was the init spectra for next method. For Tikhonov regularization unfolding, regularization parameter is objectively optimized based on a similarity metric relative to the autoML prediction. The smoothing functional is minimized using convex optimization techniques. For iterative methods, the unfolded by other method spectra was the initial guess. The uncertainty in spectrum unfolding was estimated using a Monte Carlo simulation for a set of random samples and a normally distributed error. Combining the methods yielded a robust solution for the error in the initial data.

2. Materials and Methods

2.1. Synthetic Dataset Generation

A dataset of 5×10^5 neutron spectra was generated. Each spectrum $\varphi(E)$ is modeled as a linear combination of four parametric functions [26]:

$$\varphi(E) = P_{\text{th}}\varphi_{\text{th}}(E) + P_{\text{e}}\varphi_{\text{e}}(E) + P_{\text{f}}\varphi_{\text{f}}(E) + P_{\text{hi}}\varphi_{\text{hi}}(E), \tag{5}$$

representing thermal, epithermal, fast, and high-energy components, respectively, with weights summing to unity. The energy range was 10^{-9} MeV to 6.31 MeV, discretized into $N = 60$ logarithmic bins.

For each spectrum, the readings $\mathbf{q}$ for a 10 sphere BSS GSF [2] (with diameters: 0, 2, 3, 5, 6, 8, 10, 12, 15, 18 inches) were calculated by convolving the spectrum with response functions $\mathbf{A}$. The dataset was split into training (70%) and testing (30%) sets.

2.2. Machine Learning Model and Feature Processing

In this work we used H2O – an ML framework for algorithm selection, feature generation, hyperparameter tuning, iterative modeling and model assessment. The H2O AutoML uses a combination of fast random search and stacked ensembles to achieve the best model metrics. H2O AutoML includes XGBoost Gradient Boosting Machines (GBM), Random Forests, Deep Neural Networks and Generalized Linear Models (GLM).

We trained $N = 60$ independent models for each energy bin. Each was trained to predict the normalized spectrum $\tilde{\boldsymbol{\varphi}}$ from the normalized BSS readings $\tilde{\mathbf{q}}$. Model performance was evaluated using the coefficient of determination R^2 and $RMSE$. A post-processing step sets all negative

values in $\boldsymbol{\varphi}_{\mathrm{ML}}$ to zero to ensure physical correctness before it is used as an initial guess for iterative methods.

2.3. Refinement Algorithms

2.3.1. Tikhonov Regularization

The solution is found by minimizing the regularized functional (6):

$$\boldsymbol{\varphi}^* = \arg\min_{\boldsymbol{\varphi}\geq 0} \left\{ \|\mathbf{A}\boldsymbol{\varphi} - \mathbf{q}\|_2^2 + \lambda^2 \|\mathbf{L}\boldsymbol{\varphi}\|_2^2 \right\}, \tag{6}$$

where $\mathbf{L}$ is the identity matrix (zeroth-order Tikhonov) and $\lambda \geq 0$ is the regularization parameter. The parameter λ is chosen to maximize the similarity metric between the Tikhonov solution and the ML estimate $\boldsymbol{\varphi}_{\mathrm{ML}}$. In this work a cosine similarity [27] was chosen as the metric (7). It is a measure of similarity between two non-zero vectors in n-dimensional space. For spectra the values of the vector cannot be negative, so for our task the value of cosine similarity is bounded in [0,1]. Two proportional vectors (spectra) have a cosine similarity of 1, two orthogonal vectors have a similarity of 0.

The algorithm was developed as a python code with *CVXPY* [28] package and ECOS [29] convex solver.

$$\lambda^* = \arg\max_{\lambda} \frac{\langle \boldsymbol{\varphi}(\lambda), \boldsymbol{\varphi}_{\mathrm{ML}} \rangle}{\|\boldsymbol{\varphi}(\lambda)\| \|\boldsymbol{\varphi}_{\mathrm{ML}}\|}. \tag{7}$$

2.3.2. Maximum Likelihood Expectation Maximization

Maximum Likelihood Expectation Maximization is an iterative process that maximizes the likelihood of obtaining the measured data when convergence is achieved, providing an accurate neutron spectrum [30]. The MLEM algorithm iteratively updates the spectrum estimate for k steps:

$$\varphi_i^{(k+1)} = \frac{\varphi_i^{(k)}}{\sum_{j=1}^{M} A_{ji}} \sum_{j=1}^{M} \frac{q_j A_{ji}}{\sum_{l=1}^{N} A_{jl} \varphi_l^{(k)}}, \quad i = 1, \ldots, N, \tag{8}$$

starting from the ML estimate $\boldsymbol{\varphi}^{(0)} = \boldsymbol{\varphi}_{\mathrm{ML}}$.

The algorithm was developed as a python code with *ODL* [3] package.

2.3.3. Landweber Iteration

The Landweber iteration is a gradient descent method for solving $\mathbf{A}\boldsymbol{\varphi} = \mathbf{q}$:

$$\boldsymbol{\varphi}^{(k+1)} = \boldsymbol{\varphi}^{(k)} + \alpha \mathbf{A}^T (\mathbf{q} - \mathbf{A}\boldsymbol{\varphi}^{(k)}), \tag{9}$$

with relaxation parameter α chosen as $\alpha = 1/\|\mathbf{A}^T\mathbf{A}\|_2$, and initialized with $\boldsymbol{\varphi}^{(0)} = \boldsymbol{\varphi}_{\mathrm{ML}}$, for chosen k steps.

These unfolding methods were implemented to the python package *bssunfold* [31].

2.4. Robustness Evaluation

To assess stability against measurement uncertainties, random Gaussian noise with a relative standard deviation of $\zeta_Q = 1\%$ was added to the simulated detector readings $\mathbf{q}$. For $N_{random} =$ 500 random samples of readings spectra were unfolded by methods and by mixture of methods.

2.5. Effective dose rate assessment

In accordance with radiation safety standards [32], personnel exposure to neutrons should be below certain levels. Therefore, the effective dose rate for the isotropic irradiation was used as one of metrics for comparison of the quality of unfolding, eq. (10).

$$\dot{H} = \int_{E_{min}}^{E_{max}} \phi(E) \cdot h(E) dE, \tag{10}$$

where $\dot{H}$ is the effective dose rate, $h(E)$ is the corresponding dose conversion coefficient [33], for monoenergetic particles in a certain irradiation geometry (ISO) [2]. For each unfolded spectrum the $\dot{H}$ was calculated.

2.6. Spectra similarity metrics

To assess the quality of spectrum unfolding, we used metrics such as MAE, MSE, Wasserstein distance, cosine similarity and the effective dose rate [24]. This set of metrics allows us to evaluate the unfolding quality for both individual energy bins and the spectrum shape as a whole.

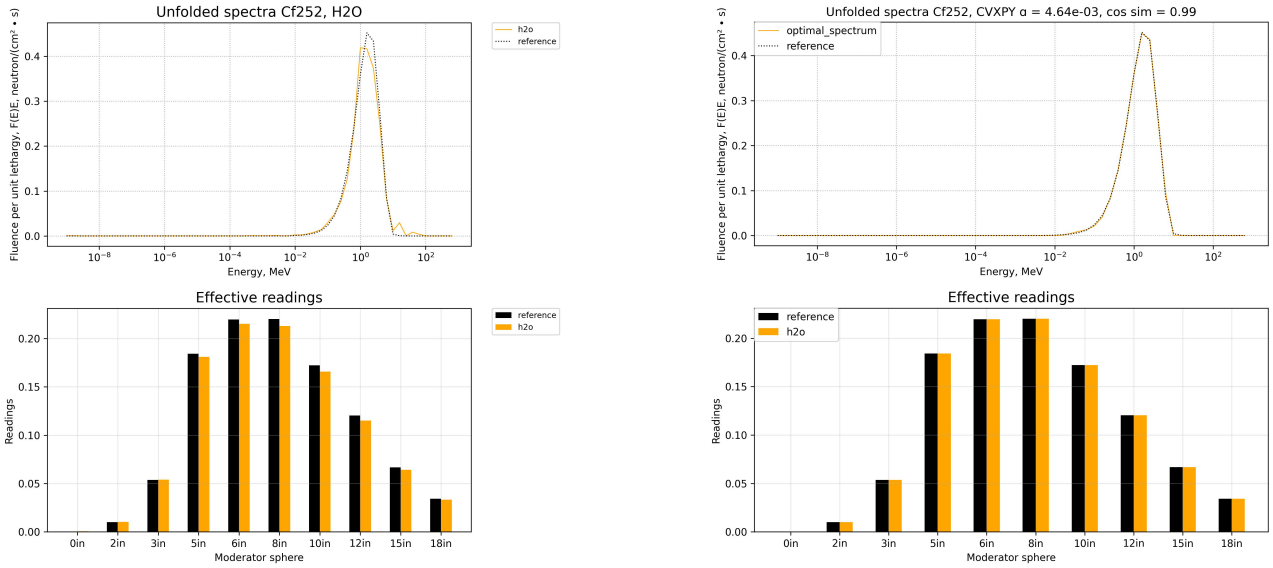
3. Results

3.1. ML Model Performance

Models were trained on the HybriLIT platform. It achieved an accuracy comparable to the values we had previously obtained using other ML algorithms [13, 15], mean R^2 on training set is 0.92 and mean R^2 on test set – 0.86. H2O autoML framework found XGBoost and GBM as the best models with individual hyperparameters for each model.

3.2. Results for ^{252}Cf spectra

The ML models were validated using reference spectra ^{252}Cf for GSF BSS [2]. This spectrum is good for validation; it can be measured experimentally using other methods or calculated using Monte Carlo programs. It has one peak, and the rest of the neutron energy range is zero. Accordingly, if the unfolding method produces oscillations near zero or negative values, this will be visible on the spectrum. The ML model reconstructed the spectrum quite well, correctly identifying the peak, although slightly undershooting it. There are some minor artifacts on the right side of the spectrum. Effective BSS readings were calculated by eq. (3) and are close to real measurements, Fig. 1.



(a) (b)

Figure 1. (a) – reference and unfolded by H2O autoML spectra for ^{252}Cf and effective BSS readings, (b) – Tikhonov regularization with parameter selected according to the best cosine similarity with H2O spectrum.

MLEM and Lanweber iterative algorithms with an uniform spectrum as the initial approximation yield better results than the ML, with slightly lower peak heights. With the correct selection of the regularization parameter, Tikhonov's method gives the best result, a near-perfect match, Fig. 2. But if we use use a combination of methods, then the initial approximation from ML allows both the iterative and regularization methods to obtain excellent agreement with reference spectra, Fig. 1,2.

3.3. Results for GRENF, position C spectra

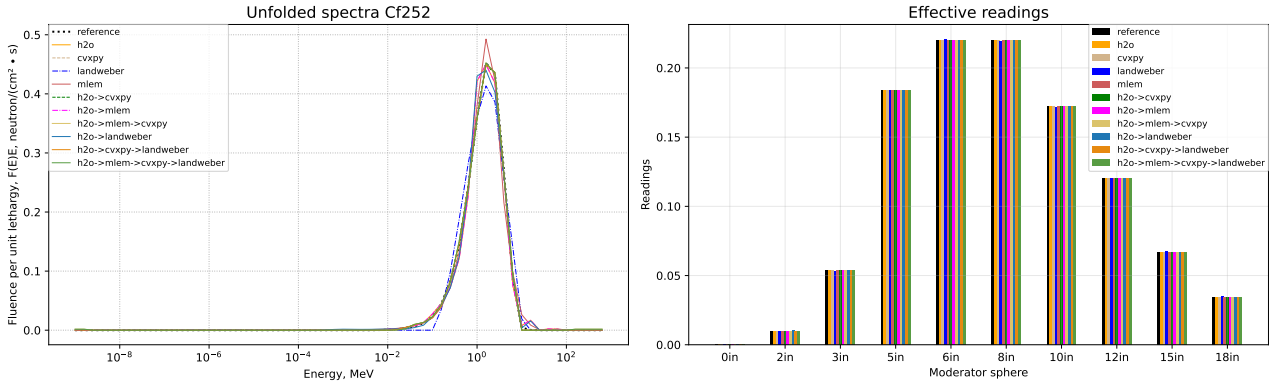


Figure 2. Comparison of reference and unfolded spectra and readings for ^{252}Cf.

The ML models was validated also on the Monte-Carlo calculated spectra from GSF realistic neutron field facility (GRENF), for position C that corresponds to a specific physical placement of the dosimeter relative to the plutonium source with significant scattering from surrounding materials [2]. This spectra has a wide neutron energy and it has a sharp peak, that is difficult to unfold.

Although the algorithms selected spectra with almost ideal values of effective readings, the shape of the spectra themselves is different, Fig. 3.

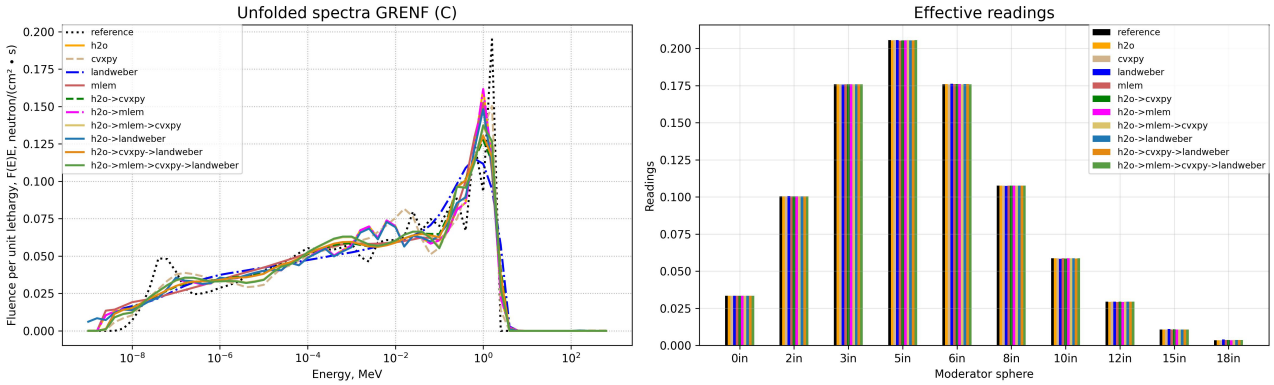


Figure 3. Reference and unfolded by separate and combinations of algorithms spectra for GRENF, position C.

For precise measurements, the CVXPY method shows the best result, table 1. However, the spectrum unfolding results for other methods we used are also quite good, with $\Delta\dot{H} < 1\%$. Additional information from H20 improves the spectrum unfolding quality of the iterative algorithms.

3.4. Stability Analysis

Table 1

GRENF(C). Metrics for assessing the similarity of a spectrum to a reference one for various unfolding methods.

method	R^2	CS	MSE
H2O	0.79	0.95	2.7×10^{-4}
cvxpy	0.89	0.97	1.4×10^{-4}
landweber	0.76	0.94	3.1×10^{-4}
mlem	0.79	0.95	2.8×10^{-4}
H2O-cvxpy	0.83	0.96	2.2×10^{-4}
H2O-mlem	0.79	0.95	2.7×10^{-4}
H2O-mlem-cvxpy	0.86	0.97	1.9×10^{-4}
H2O-landweber	0.80	0.95	2.6×10^{-4}
H2O-cvxpy-landweber	0.84	0.96	2.1×10^{-4}
H2O-mlem-cvxpy-landweber	0.86	0.97	1.9×10^{-4}
H2O-cvxpy-mlem	0.83	0.96	2.2×10^{-4}

method	WD	$\Delta\dot{H}, \%$	$\Delta\mathbf{q}$
H2O	3.8×10^{-1}	0.7%	3.0×10^{-4}
cvxpy	3.6×10^{-1}	0.4%	8.2×10^{-13}
landweber	3.8×10^{-1}	0.2%	7.6×10^{-4}
mlem	4.5×10^{-1}	0.8%	4.8×10^{-4}
H2O-cvxpy	3.1×10^{-1}	0.6%	3.1×10^{-4}
H2O-mlem	3.8×10^{-1}	0.7%	3.0×10^{-4}
H2O-mlem-cvxpy	2.8×10^{-1}	0.3%	1.6×10^{-4}
H2O-landweber	4.3×10^{-1}	0.5%	2.8×10^{-4}
H2O-cvxpy-landweber	2.8×10^{-1}	0.3%	3.0×10^{-4}
H2O-mlem-cvxpy-landweber	2.9×10^{-1}	0.4%	1.4×10^{-4}
H2O-cvxpy-mlem	3.1×10^{-1}	0.6%	3.1×10^{-4}

Measurement results always contain errors. In ill-posed problems, small errors in the input data can lead to large errors in the solution. Therefore, we proposed combination of methods to stabilize the solution. Spectra were unfolded for input noise level ζ_Q and N_{random} samples. For evaluation metrics from 2.6 we obtained results presented in table 2.

Table 2

GRENF(C). Average metrics with standard deviation for assessing the similarity of a spectrum to a reference one for various

method	R^2	CS	MSE
H2O	0.69 ± 0.09	0.93 ± 0.02	$(4 \pm 1.2) \times 10^{-4}$
cvxpy	-5.03 ± 4.15	0.55 ± 0.16	$(7.9 \pm 5.4) \times 10^{-3}$
landweber	0.75 ± 0.02	0.94 ± 0.01	$(3.3 \pm 0.3) \times 10^{-4}$
mlem	0.76 ± 0.03	0.94 ± 0.01	$(3.1 \pm 0.4) \times 10^{-4}$
H2O-cvxpy	0.76 ± 0.04	0.94 ± 0.01	$(3.1 \pm 0.5) \times 10^{-4}$
H2O-mlem	0.69 ± 0.09	0.93 ± 0.02	$(4 \pm 1.2) \times 10^{-4}$
H2O-mlem-cvxpy	0.73 ± 0.08	0.94 ± 0.02	$(3.5 \pm 1.1) \times 10^{-4}$
H2O-landweber	0.69 ± 0.09	0.93 ± 0.02	$(4 \pm 1.2) \times 10^{-4}$
H2O-cvxpy-landweber	0.73 ± 0.09	0.94 ± 0.02	$(3.5 \pm 1.2) \times 10^{-4}$
H2O-mlem-cvxpy-landweber	0.69 ± 0.13	0.93 ± 0.03	$(4 \pm 1.7) \times 10^{-4}$
method	WD	$\Delta\dot{H}, \%$	$\Delta\mathbf{q}$
H2O	0.63 ± 0.2	1.16	2.7×10^{-3}
cvxpy	2.33 ± 0.75	3.8	3.4×10^{-3}
landweber	0.48 ± 0.15	1.2	2.8×10^{-3}
mlem	0.52 ± 0.12	0.97	2.6×10^{-3}
H2O-cvxpy	0.52 ± 0.16	1.18	2.8×10^{-3}
H2O-mlem	0.63 ± 0.2	1.16	2.7×10^{-3}
H2O-mlem-cvxpy	0.68 ± 0.24	1.41	2.9×10^{-3}
H2O-landweber	0.74 ± 0.25	1.49	2.9×10^{-3}
H2O-cvxpy-landweber	0.7 ± 0.27	1.53	2.9×10^{-3}
H2O-mlem-cvxpy-landweber	0.81 ± 0.32	1.63	3×10^{-3}

For the GRENF(C) spectrum Figure 4 shows that the spectrum shape changes with errors in the source data: the range of possible spectra is shown in the semi-transparent area. Iterative

algorithms, which converge to a spectrum similar to the true one despite the error, prove to be the most stable. Tikhonov's method with convex optimization (CVXPY) produces excellent results with accurate data, but small errors significantly alter the spectrum shape. To achieve conditional stability of the solution, we propose using iterative methods. This yields a more accurate solution and reduces the uncertainty range. A combination of methods allows for the highest accuracy with an acceptable error. Selecting a regularization parameter based on the spectrum shape of the reconstructed ML yields significantly better results than using Tikhonov's method alone; the best result is $R^2 = 0.76$ for both the MSE $= 3.1 \times 10^{-4}$ and the MLEM method.

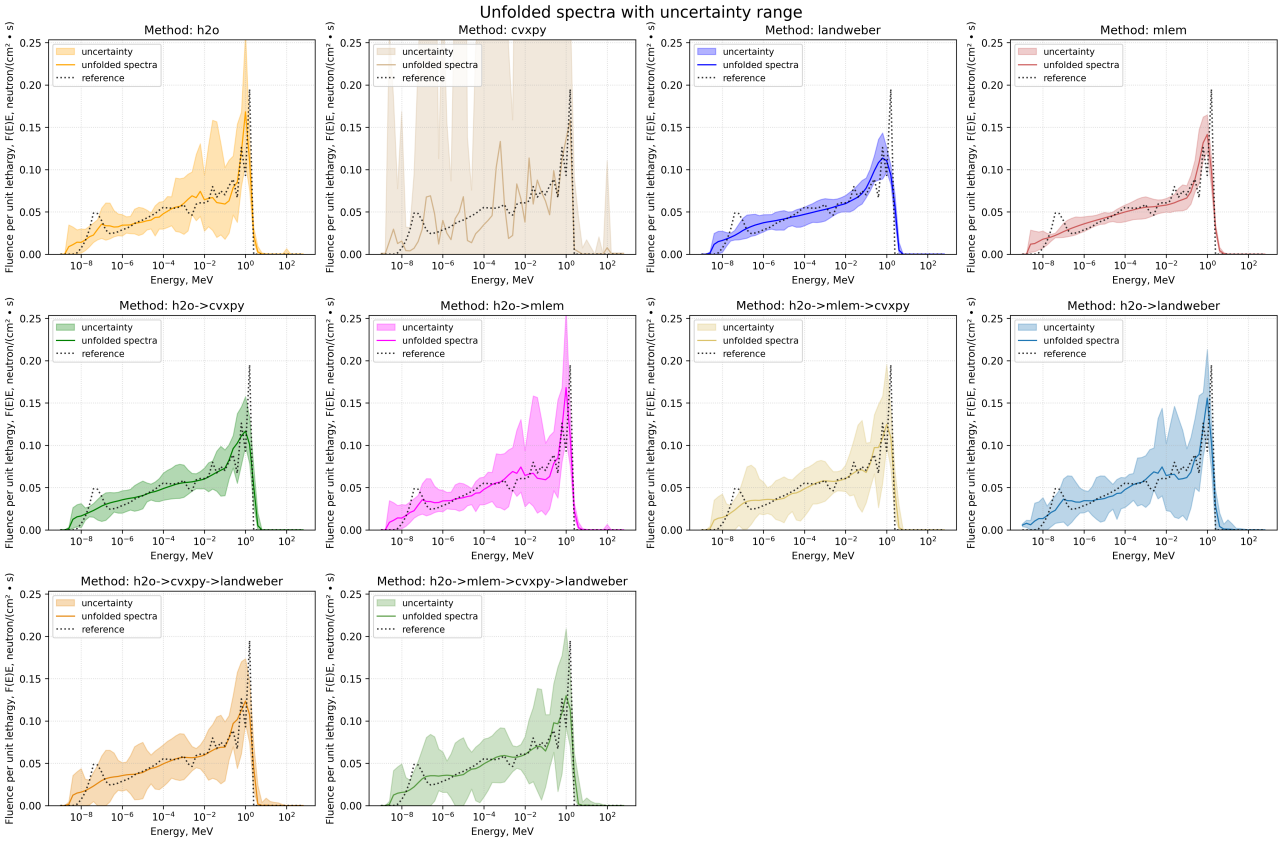


Figure 4. GRENF(C). Average unfolded spectra for developed unfolding algorithms, semi-transparent area shows the uncertainty of unfolding

4. Discussion

The results demonstrate that the proposed hybrid framework successfully leverages the complementary strengths of ML and analytical methods. The ML model provides a physically reasonable starting point that is already close to the true solution, effectively narrowing the solution space. This prior information is used in regularization and iterative algorithms and it

gives better results for MLEM and Landweber algorithms, if spectra has a complex form with sharp peaks.

The strategy of selecting the Tikhonov parameter λ by maximizing cosine similarity with the ML prior is a data-driven alternative to classical methods like the L-curve [34]. It automates parameter tuning and adapts it to the specific spectral shape suggested by the measurements, leading to more consistent performance. Of course, the result depends heavily on the trained model. It may fail to capture sharp peaks, which can be identified and unfolded more accurately with manual parameter selection (in accordance with the expert knowledge of spectra shape) and another solver such as Gurobi, Express [12, 28]. But if we don't know the shape, autoML selection can significantly improve the unfolding.

The primary limitation remains the dependence of the ML model on the quality and representativeness of the training data. Spectra very far from the FRUIT-generated distribution [9] may lead to poor initial guesses. The model metrics can be improved by incorporating a broader set of spectra from experimental and particle transport simulation software into the training dataset. Using combination of other types of ML algorithms and NN architectures with specific tabular data transformation [15, 35] can also yield better results.

Addition in the unfolding chain methods, such as variations of the conjugate gradient and quasi-Newton methods (e.g. L-BFGS-B [28, 36]) could also improve the unfolding results since they can give more accurate results than the methods used in this work [12].

The selection of the optimal set of moderator spheres is important for spectrum unfolding accuracy.It is necessary to ensure complete coverage of the neutron energy range. It is advisable to use spheres with the most diverse response functions. In practice, this means avoiding the use of multiple spheres of similar diameters. This reduces the correlation between the rows of the response matrix and lowers matrix's condition number [37, 38].

The most time is spent on training the model, since it needs to be trained for each set of moderator spheres and for each BSS. The method works quickly even for a trained model, allowing for the neutron spectra unfolding and assessment of doses to personnel. The method is implemented as a python code for cross-platform and convenient use by radiation safety specialists.

5. Conclusions

A hybrid multi-stage algorithm for neutron spectrum unfolding has been developed, combining a machine-learned prior with Tikhonov regularization and iterative algorithms (Landweber, MLEM). The algorithm uses an set of automatically fitted in H2O framework models (XGboost, GBM) trained on a large synthetic dataset to generate an initial spectrum estimate. This estimate is post-processed for physical consistency and then used to initialize the Tikhonov regularization process with the regularization parameter selected via optimization of cosine similarity.

The proposed approach was validated on a test set of spectra derived from Monte Carlo simulations: spectra for ^{252}Cf and realistic neutron field facility (GRENF). The stability against measurement noise compared to standalone ML or traditional unfolding algorithms was accessed.

It has been shown that the combination of a machine learning algorithm with a regularization method and the iteration algorithm provides high spectrum unfolding accuracy while reducing uncertainty caused by measurement errors. For example, with a measurement error of 1%, the error in estimation of the effective dose rate from the spectrum is 1.5%. In terms of the residual norm, the method yields effective reading values for BSS very close to the actual measurement data. The machine learning model has $R^2 = 0.69$, while the combined algorithm gives $R^2 = 0.76$.

This work establishes a practical and effective pipeline for spectra unfolding with machine learning and inverse problem solvers. The proposed method could be used for improving radiation protection in high-energy neutron fields.

5.0.1. Acknowledgments

Authors acknowledge the support of the JINR Multifunctional Information and Computing Complex for providing computational resources, http://hlit.jinr.ru.

5.0.2. Funding

This work was carried out within the framework of the state assignment of the Ministry of Education and Science of the Russian Federation (project No. 124112200072-2, "Application of Explainable Artificial Intelligence for the Interpretation of Machine Learning Algorithms").

References

1. Chizhov K., Beskrovnaya L., Chizhov A. Neutron spectrum unfolding method based on shifted legendre polynomials, its application to the IREN facility // Physics of Particles and Nuclei Letters. 2025. Vol. 22, no. 2. P. 190. DOI: https://doi.org/10.1134/S154747712470239X.
2. Compendium of Neutron Spectra and Detector Responses for Radiation Protection Purposes. Vienna: INTERNATIONAL ATOMIC ENERGY AGENCY, 2001. (Technical Reports Series; no. 403). ISBN 92-0-102201-8.
3. Adler J. et al. odlgroup/odl: ODL 0.7.0. Zenodo, 2018. DOI: 10.5281/zenodo.1442734.
4. Chizhov A., Chizhov K. TSVD-based neutron spectra unfolding by Bonner multi-sphere spectrometer readings with iteration procedure // The International Conference "Distributed Computing and Grid-technologies in Science and Education". 2025.
5. Borshchev D.S., Akimochkina M.A., Chizhov K.A. A hybrid method for neutron spectrum unfolding based on Tikhonov regularization and the MAXED algorithm. // 33rd International Conference "Mathematics. Computer. Education". 2026.
6. Chizhov K., Chizhov A. Optimization of the Neutron Spectrum Unfolding Algorithm Based on Tikhonov Regularization and Shifted Legendre Polynomials // MMCP 2024. 2024. P. 74.
7. Akimochkina M.A., Borshchev D.S., Chizhov K.A. Neural network reconstruction of neutron spectrum with a physically informed loss function // V Scientific and Practical Conference "Physical and Technical Intelligent Systems" (FTIS-2026): Book of abstracts. - Tambov: Yulis Publishing House LLC, 2026. P. 14. (in Russian).
8. Freeman D.W., Edwards D.R., Bolon A.E. Genetic algorithms–a new technique for solving the neutron spectrum unfolding problem // Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 1999. Vol. 425, no. 3. P. 549–576.
9. Bedogni R., Domingo C., Esposito A., Fernández F. FRUIT: an operational tool for multisphere neutron spectrometry in workplaces // Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2007. Vol. 580, no. 3. P. 1301–1309.
10. Sannikov A.V., Peleshko V.N., Savitskaya E.N., et al. Multisphere neutron spectrometer based on the serial RSU-01 device // IHEP Preprint. 2007. Vol. 21.
11. Gómez-Ros J.M., Bedogni R., Domingo C., et al. Results of the EURADOS international comparison exercise on neutron spectra unfolding in Bonner spheres spectrometry // Radiation Measurements. 2022. Vol. 153. P. 106755.
12. Chizhov K.A., Chizhov A.V., Borshchev D.S., Akimochkina M.A. Methods for solving inverse problems for processing measurement results using the example of neutron spectrum unfolding // 33rd International Conference "Mathematics. Computer. Education". 2026.
13. Chizhov K. Random forest regression and Shapley additive explanation for effective dose rate estimation in high-energy neutron fields based on Bonner spectrometer measurements. Sirius, Sochi, 2025. URL: https://openreview.net/forum?id=S60GzItoqh (дата обращения : 21.05.2022).
14. Ortiz-Rodriguez J.M., Alfaro A.R., Haro A.R., et al. A neutron spectrum unfolding computer code

based on artificial neural networks // Radiation physics and chemistry. 2014. Vol. 95. P. 428–431.

15. Chizhov K., Bely A. Neutron spectrum unfolding using deep learning models for tabular data // Moscow University Physics Bulletin, 2025, Vol. 80, Suppl. 3, pp. S987–S995. DOI: 10.3103/S0027134925702832
16. Starikovskaya M.D., Chizhov K.A. Application of Kolmogorov-Arnold networks for solving the neutron spectrum unfolding problem // 33rd International Conference "Mathematics. Computer. Education". 2026.
17. Alvar A.A., Deevband M.R., Ashtiyani M. Neutron spectrum unfolding using radial basis function neural networks // Applied Radiation and Isotopes. 2017. Vol. 129. P. 35–41.
18. Bouhadida M., Mazzi A., Brovchenko M., et al. Neutron spectrum unfolding using two architectures of convolutional neural networks // Nuclear Engineering and Technology. 2023. Vol. 55, no. 6. P. 2276–2282.
19. Del Rosario Martinez-Blanco M., Ornelas-Vargas G., Castañeda-Miranda C.L., et al. A neutron spectrum unfolding code based on generalized regression artificial neural networks // Applied radiation and isotopes. 2016. Vol. 117. P. 8–14.
20. Zhou B., Hu Z., Zhong M., et al. Bayesian Neural Networks for the Neutron Spectrum Unfolding in the EAST Tokamak // IEEE Transactions on Instrumentation and Measurement. 2025.
21. Chizhov K. et al. Reconstruction of the energy spectrum of the neutron radiation flux using the random forest machine learning algorithm // Modern information technologies and IT education. 2024. Vol. 20, no. 4. (in Russian).
22. Chizhov K.A., Lebedev A.D., Trofimov Yu.V., et al. Interpretable neutron spectrum reconstruction based on two-stage ANFIS learning with SHAP regularization // 33rd International Conference "Mathematics. Computer. Education". 2026.
23. McGreivy J., Manfredi J.J., Siefman D. Data Augmentation for Neutron Spectrum Unfolding with Neural Networks // Journal of Nuclear Engineering. 2023. Vol. 4, no. 1. P. 77–95.
24. Chizhov K.A., Lebedev A.D., Trofimov Yu.V., et al. Automated machine learning spectrum unfolding for neutron spectrometry with Bonner spheres // The 11th International Conference "Distributed Computing and Grid-technologies in Science and Education July, 7-11, 2025 JINR, Dubna, Russia. 2025.
25. Shiwei L., Wenbao J., Wei C., et al. Neutron spectrum unfolding code based on iterative method combined with artificial neural networks for bonner sphere spectrometer // Journal of Radioanalytical and Nuclear Chemistry. 2024. Vol. 333, no. 1. P. 557–562.
26. Starikovskaya M.D., Chizhov K.A. Neutron spectrum unfolding based on random forest algorithm and generated training sample // Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems 202. Russian University of Peoples' Friendship named after Patrice Lumumba, 2025. P. 389–394. (in Russian).
27. Condon Z.T. Unfolding Neutron Energy Spectra with a Passive Neutron Spectrometer: PhD thesis The Ohio State University, 2024.
28. Diamond S., Boyd S. CVXPY: A Python-embedded modeling language for convex optimization //

Journal of Machine Learning Research. 2016. Vol. 17, no. 83. P. 1–5.

29. Domahidi A., Chu E., Boyd S. ECOS: An SOCP solver for embedded systems // 2013 European control conference (ECC). IEEE, 2013. P. 3071–3076.
30. Díaz-Comeche A., Oliver S., Juste B., et al. Monte Carlo study of neutron spectra unfolding for a proton beam using MLEM // Radiation Physics and Chemistry. 2025. Vol. 233. P. 112702.
31. Radiationsafety/bssunfold: Python package for neutron spectrum unfolding from measurements obtained with Bonner Sphere Spectrometer (BSS) URL: https://github.com/Radiationsafety/bssunfold (дата обращения : 04.05.2026).
32. Radiation Safety Standards (NRB-99/2009) Sanitary Rules and Regulations. SanPin 2.6.1.2523-09. Moscow Publ., 2009. (in Russian).
33. Petoussi-Henss N., et al. Conversion coefficients for radiological protection quantities for external radiation exposures // Annals of the ICRP. 2010. Vol. 40, no. 2-5. P. 1–257.
34. Koslowsky M. Spectral Unfolding: A Mathematical Perspective // IAEA TECDOC SERIES. 2020. P. 97.
35. Thielmann A.F., Kumar M., Weisser C., et al. Mambular: A Sequential Model for Tabular Deep Learning // arXiv preprint arXiv:2408.06291. 2024.
36. Zhu C., Byrd R.H., Lu P., Nocedal J. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization // ACM Transactions on mathematical software (TOMS). 1997. Vol. 23, no. 4. P. 550–560.
37. Chizhov A., Chizhov K. Dose assessment of personnel neutron irradiation on high-energy accelerators using a multi-sphere Bonner spectrometer // Mathematical Modeling. 2023. Vol. 7, no. 2. P. 63–64.
38. Chizhov A., Chizhov K. Optimization approach to neutron spectra unfolding with Bonner multi-sphere spectrometer // Mathematical Modeling. 2024. Vol. 8, no. 3. P. 89–90.
39. Tikhonov A.N., Goncharsky A.V., Stepanov V.V., Yagola A.G. Numerical methods for solving ill-posed problems. M.: Nauka (in Russian), 1990.
40. Aleinikov V.E., Bamblevskij V.P., Komochkov M.M., et al. Reference neutron fields for metrology of radiation monitoring // Radiation Protection Dosimetry. 1994. Vol. 54, no. 1. P. 57–59.
41. Awschalom M., Sanna R.S. Applications of Bonner sphere detectors in neutron field dosimetry. Fermi National Accelerator Lab., 1983.
42. Chizhov K., Chizhov A. Optimization of the Neutron Spectrum Unfolding Algorithm Using Shifted Legendre Polynomials Based on Weighted Tikhonov Regularization // Physics of Particles and Nuclei. 2025. Vol. 56, no. 6. P. 1395–1399. DOI: 10.1134/S106377962570056X.
43. Chizhov K., Bragin Yu., Sneve M.K., et al. The development and application of a method for assessing radionuclide surface contamination density based on measurements of ambient dose equivalent rate // Journal of Radiological Protection. 2019. Vol. 39, no. 2. P. 354.
44. Chizhov K., Bragin Yu., Sneve M.K., et al. Further development and application of a method for assessing radionuclide surface activity distribution and source location based on measurements of ambient dose equivalent rate. Examples for Andreeva Bay, Chernobyl NPP and Istiklol // Journal

of Radiological Protection. 2023. Vol. 43, no. 4. P. 041506. DOI: 10.1088/1361-6498/ad005c.

45. Chizhov K., Bragin Yu., Sneve M.K., et al. Further development and application of a method for assessing radionuclide surface activity distribution and source location based on measurements of ambient dose equivalent rate // Journal of Radiological Protection. 2023. Vol. 43, no. 4. P. 041505. DOI: 10.1088/1361-6498/ad005b.
46. Wang J., Zhou Y., Guo Z., Liu H. Neutron spectrum unfolding using three artificial intelligence optimization methods // Applied Radiation and Isotopes. 2019. Vol. 147. P. 136–143.
47. Jablonský J., et al. Benchmarks for current linear and mixed integer optimization solvers // Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis. 2015. Vol. 63, no. 6. P. 1923–1928.

УДК 004.052.42

На пути к дедуктивной верификации реализаций умножения матриц с направленными на повышение эффективности использования кэш-памяти оптимизациями*

Агибалов М.С. (Новосибирский государственный университет)

Кондратьев Д.А. (Институт систем информатики им. А.П. Ершова СО РАН)

Задача повышения производительности умножения матриц является актуальной задачей современного программирования. При решении данной задачи в программном коде умножения матриц реализуют оптимизации, что затрудняет анализ полученных реализаций и может приводить к появлению в них ошибок. Гарантировать корректность программного кода относительно спецификаций, описывающих результат работы программы в зависимости от ее входных данных, может только дедуктивная верификация. Итого, задача дедуктивной верификации реализаций умножения матриц с направленными на повышение производительности оптимизациями является актуальной. В данной статье представлен первоначальный этап исследования, нацеленный на дедуктивную верификацию реализаций классического умножения матриц с направленными на повышение эффективности использования кэш-памяти оптимизациями.

Ключевые слова: дедуктивная верификация, умножение матриц, Frama-C, WP, Coq, Rocq, кэш-память

1. Введение

Задача повышения производительности умножения матриц является актуальной задачей современного программирования. Решение данной задачи особенно важно для работы с широко распространенными в настоящее время сверточными нейронными сетями, основанными на умножении тензоров. При решении задачи быстрого умножения матриц полезно определить особую реализацию, с которой можно сравнивать другие подходы и которая при этом относительно эффективна. Такой применяемой при работе со сверточны-

*Проект поддержан Лабораторией YADRO НГУ

ми нейронными сетями и во многом ориентированной на сравнение реализацией является minigemm [11, 86], разработанная в компании YADRO.

При задачи повышения производительности умножения матриц в программном коде умножения матриц реализуют оптимизации, что затрудняет анализ полученных реализаций и может приводить к появлению в них ошибок Обычно корректность программ проверяется тестированием, и тестирование имеет цель – найти ошибки в программах, однако оно не может гарантировать отсутствие ошибок. Гарантировать корректность программ может только формальная верификация [3, 4, 6, 13, 31]. Для проверки корректности исходного кода программ относительно спецификаций, описывающих результат работы программы в зависимости от ее входных данных, применяется такой вид формальной верификации, как дедуктивная верификация [10, 14, 16, 19, 20, 43, 46, 49, 51].

В случае умножений матриц с оптимизациями важно верифицировать такое свойство, что результат такого умножения совпадает с результатом классического умножения матриц. В таком случае в формальных спецификациях верифицируемой реализации оптимизированного умножения матриц будет описано, что результат оптимизированного умножения будет равен результату классического умножения матриц. Таким образом дедуктивная верификация позволяет проверить функциональную эквивалентность неэффективной, но эталонной классической реализации умножения матриц и эффективной, но из-за сложности требующей тщательного анализа реализации оптимизированного умножения матриц.

Итого, задача проверки с помощью дедуктивной верификации функциональной эквивалентности реализации minigemm и классического умножения матриц является актуальной. Поэтому, в Лаборатории YADRO НГУ был запущен проект "Методы и средства дедуктивной верификации реализаций умножения матриц с направленными на повышение производительности оптимизациями", целью которого является дедуктивная верификация реализации minigemm. Данный проект выполняют авторы данной статьи, а именно студент НГУ Агибалов Матвей Сергеевич под руководством к.ф.-м.н., научного сотрудника ИСИ СО РАН и старшего преподавателя НГУ Кондратьева Дмитрия Александровича.

Однако дедуктивная верификация реализации minigemm осложнена из-за наличия в minigemm разнообразных модификаций относительно классического умножения матриц, вещественной арифметики, традиционно представляющей сложности для дедуктивной верификации, и ассемблерных вставок, не позволяющими верифицировать всю реализацию

только как код на языке C. Чтобы преодолеть данные сложности, было принято решение выполнять данный проект в итеративном порядке так, чтобы на каждой итерации проекта накапливался опыт и создавался задел для последующих итераций.

Первой итераций данного проекта стала работа на Зимнем Системном Буткемпе лаборатории YADRO НГУ. Первоочередной задачей данного проекта было обучение участника проекта, поэтому в ходе данного проекта была начата дедуктивная верификация слабосвязанной с minigemm, но зато относительно простой для понимания блочного умножения матриц реализации алгоритма Штрассена [83]. Умножения матриц с оптимизациями обычно в целях производительности реализуют на основанном на указателях языке программирования C, поэтому на данном этапе проекта применялась ориентированная на поддержу работы с указателями с помощью сепарационной логики [52, 72, 73, 77, 78] система дедуктивной верификации VST [17, 18, 32]. Система VST основана на системе доказательства Coq [25, 76], недавно переименованной в Rocq. Негативным опытом по итогам первой итерации проекта стало понимание, что проводить дедуктивную верификацию в системе VST сложно из-за того, что в данной системе нет возможности при работе с вещественной арифметикой использовать ее идеальное (математическое) представление, а есть только работа с машинной вещественной арифметикой по стандарту IEEE 754, что приводит к чрезмерному усложнению начального этапа проекта. Также негативным опытом стало понимание, что работать в системе доказательства VST относительно сложно из-за применяемого в системе VST основанного на прямом прослеживании программы исчисления сильнейшего постусловия [41, 48], приводящего к сложностям при прямом прослеживании программы задавать инвариант цикла путем модификации постусловия [47]. При этом по итогам первой итерации проекта были получен важный и пригодившийся потом опыт работы в системе Coq, а именно удалось задать в Coq-представлении спецификации реализации алгоритма Штрассена и доказать в системе Coq свойство, что если размер матрицы не подходящий для алгоритма Штрассена (не кратен степени двойки), то вместо него применяется классическое умножение матриц. Награждение по итогам Зимнего Системного Буткемпа лаборатории YADRO НГУ приведено на рисунке 1.

Итого, по итогам первого этапа проекта было принято решение перейти от использования системы VST к использованию системы Frama-C [21] с плагином WP [35, 36], где есть поддержка идеальной машинной арифметики и основанного на обратном прослеживании программы исчисления слабейшего предусловия [23, 40, 45, 62], упрощающего применение

Рис. 1. Награждение по итогам Зимнего Системного Буткемпа лаборатории YADRO НГУ

при задании инвариантов циклов метода модификации постусловия.

Также по итогам первого этапа проекта с целью упрощения итераций было принято решение сначала проводить дедуктивную верификацию в идеальной (математической) вещественной арифметике, и только потом переходить к верификации в машинной арифметике [50].

Кроме того, по итогам первого этапа проекта было принято решение отказаться от продолжения дедуктивной верификации алгоритма Штрассена и перейти к дедуктивной верификации последовательности реализаций умножений матриц из статьи [5], где описывается эффективная реализация умножения матриц шаг за шагом. Данная последовательность реализаций была выбрана потому, что она, стартуя с классического умножения матриц, приводит к реализации, приближающейся по структуре к minigemm. Началом такой последовательности служат реализации, где применяются направленные на повышение эффективности использования кэш-памяти оптимизации, а именно изменения поряд-

ка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы [12, 84]. Поэтому, целью работы на текущем этапе проекта является разработка комплексного подхода к дедуктивной верификации реализаций умножения матриц над математическими вещественными числами с направленными на повышение эффективности использования кэш-памяти оптимизациями. Для достижения данной цели поставлены следующие задачи:

1. Задание спецификаций для реализаций классического умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы.
2. Задание теории предметной области для верификации реализации классического умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы.
3. Разработка стратегий доказательства условий корректности при дедуктивной верификации реализации классического умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы.
4. Проведение с помощью разработанного комплексного подхода экспериментов по дедуктивной верификации реализации классического умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы.

В данной статье мы презентуем текущие результаты нашей работы, общедоступные в репозитории проекта [1].

Обзор родственных работ В качестве основной родственной работы отметим статью [85] про реализацию в компиляторе CompCert оптимизации в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы. Отличительной особенностью компилятора CompCert [28, 29, 63, 64] является формальная верификация его кода, и рассматриваемая оптимизация также была формально верифицирована. В ходе формальной верификации данной оптимизации

было проверено свойство, что ее применение не нарушает корректность оптимизируемой программы. Отметим, что такая формальная верификация довольна схожа с задачами текущего этапа нашего проекта. Также верификация осуществлялась в системе доказательства Coq, как в многом происходит и в нашем проекте. Однако, в случае верифицируемого компилятора при верификации осуществлялась работа с промежуточными представлениями C-программы, тогда как в нашем проекте верифицируется непосредственно исходный код. Кроме того, отличаются глобальные цели проектов: в проекте CompCert глобальной целью является расширение компилятора как можно большим числом верифицируемых оптимизаций, тогда как в нашем проекте глобальной целью является создание подходов к дедуктивной верификации реализаций умножения матриц с оптимизациями. Стремление расширить компилятор CompCert различными оптимизациями демонстрируется исследованием [55], где предлагаются формально верифицированные оптимизации для циклов. Однако, предложенные в данном исследовании оптимизации мало похожи на встречаемые в нашем проекте.

Отдельным классом родственных работ являются исследования [33, 39, 74] по дедуктивной верификации алгоритма Штрассена. Данные исследования привели к разработке полезных библиотек [39, 74] с теоремами о матрицах для систем доказательства Coq и ACL2 [69]. Однако, в данных исследованиях доказывались свойства программ на функциональных языках программирования WhyML (промежуточный язык верификации в системе Frama-C) [30, 44, 60], Gallina (входной язык системы доказательства Coq) [25, 76] и Applicative Common Lisp (входной язык системы доказательства ACL2) [54], что, в отличие от нашего проекта, упростило верификацию из-за отсутствия необходимости работать с моделью памяти языка C. Также алгоритм Штрассена удавалось синтезировать по спецификации [79], что близко к методам дедуктивной верификации, однако далеко от проблем дедуктивной верификации программ в модели памяти языка C.

Также в качестве родственного исследования рассмотрим дедуктивную верификации частей операционной системы в системе дедуктивной верификации AstraVer [7, 42, 65, 67, 87]. Но используемые в системе AstraVer для доказательства условий корректности такие автоматические системы, как SMT-решатели, могут не справляться с моделированием работы с памятью в C-программах [66].

Кроме того, в качестве родственного исследования рассмотрим исследование по разработке методов автоматизации дедуктивной верификации программ на подмножестве

языка программирования C [70, 71] и реализации таких методов в системе C-lightVer [58, 59, 68]. Данная система позволяет упростить доказательство с помощью генерации вспомогательных лемм по определенным шаблонам [56, 57]. Отметим, что некоторые стратегии системы C-lightVer, а именно стратегия для финитных итераций над изменяемыми массивами [58], стратегия для программ с финитными итерациями над массивами [58] и стратегия для программ, спецификации которых содержат функции со свойством конкатенации [58], близки к полученным в ходе нашего проекта стратегиям. Однако триггерами для срабатывания шаблонов в системе C-lightVer служат фрагменты программного кода, а не фрагменты дерева доказательства, как в нашем проекте.

Родственным нашему проекту мероприятием является серия российских соревнований по формальной верификации программ VeHa. В рамках данной серии на текущий момент состоялись три соревнования, VeHa-2023 [15], VeHa-2024 [9] и VeHa-2025 [80]. На данных соревнованиях команды исследователей также решают сложные для формальной верификации задачи. Но на нашем проекте, в отличие от соревнований серии VeHa, временные рамки намного более мягкие (проект продолжается уже семестр вместо трех дней). Другим родственным мероприятием является проект Большой Математической Мастерской 2025 года по формальной верификации хэш-функции «Стрибог» [8]. Однако, на нашем проекте, в отличие от проекта на Большой Математической Мастерской 2025 года, поставлены другие цели.

Структура статьи Данная статья имеет следующую структуру: в разделе 2 описаны основы дедуктивной верификации программ в системе Frama-C/WP, в разделе 3 описана проведенная нами дедуктивная верификация реализаций классического умножения матриц и умножения матриц с направленными на повышение эффективности использования кэш-памяти оптимизациями, в заключении приведен список наших результатов и описаны наши планы на будущее, в Приложении A приведен пример одного из доказанных условий корректности вместе с доказательством.

Благодарности Авторы статьи выражают благодарность компании YADRO и Лаборатории YADRO НГУ, а также лично Власову Александру Александровичу и Латкину Евгению Ивановичу за организацию и анализ проекта.

2. Основы дедуктивной верификации в системе Frama-C/WP

Процесс дедуктивной верификации программ в системе Frama-C/WP начинается с задания формальных спецификаций верифицируемой программы на языке ACSL, а именно предусловия (ограничение на входные данные), постусловия (связь входных и выходных данных) и инвариантов циклов (истинны на входе в цикл, истинны на итерациях цикла и обеспечивают выход из цикла). Спецификации задаются внутри C-комментариев, где можно формировать теорию предметной области. Для верификации функциональной эквивалентности полезно разместить в теории предметной области предикаты, описывающие на логическом уровне рекурсивные определения, с которыми будет соотноситься верифицируемая программа. Для задания таких предикатов удобно использовать механизм индуктивных предикатов и их свойств. Отметим, что формально индуктивные предикаты не задают вычислимое определение, однако в их свойствах можно и полезно фактически задать такое определение. Такие свойства могут зависеть от меток программы, которые позволяют с помощью конструкции `at` указывать, в каком состоянии программы брать значения тех или иных переменных. Также еще одной важной конструкцией для задания спецификаций является конструкция `separated`, которая позволяет указать на расположение в разных областях памяти. Кроме того, еще одной важной конструкцией для задания спецификаций является конструкция `assert`, накладывающая ограничение в определенной точке программы, чтобы дальше по ходу исполнения программы можно было полагаться на такое ограничение. Введение конструкций `assert` позволяет упростить дедуктивную верификацию программ за счет доказательства таких промежуточных утверждений [23, 45, 62].

Логической основой плагина WP для системы Frama-C является исчисление слабейшего предусловия, которое позволяет для программы и ее спецификаций генерировать условия корректности, истинность которых означает корректность программы относительно ее спецификаций. Так как верифицируются программы на языке C, то неизбежно встает вопрос о работе на данном этапе с указателями. Вместо широко применяемой для таких целей сепарационной логики, в системе Frama-C/WP используется модель памяти, поддерживающая отображения из адресов в значения. При работе с такой моделью памяти применяется конструкция `havoc`, который позволяет сравнивать, изменилась ли область памяти после операций над памятью. Поэтому, доказательства условий корректности се-

рьезно зависят от работы с предикатом `havoc`.

Доказывать условия корректности в системе Frama-C/WP можно и автоматически с помощью SMT-решателей и систем автоматического доказательства теорем, а также интерактивно с помощью системы Coq. Такие SMT-решатели и системы автоматического доказательства теорем, как Z3 [26, 38], CVC4 [24], CVC5 [22], Vampire [61], Alt-Ergo [34] и E Prover [81] позволяют доказывать многие относительно простые условия корректности в автоматическом режиме. Таким образом, пользователю для интерактивного доказательства в системе Coq в основном остаются самые сложные условия корректности.

Рассмотрим вопрос автоматизации дедуктивной верификации в системе Frama-C/WP. На разных стадиях дедуктивной верификации для такой автоматизации можно использовать следующие подходы:

1. Пытаться переиспользовать в разных проектах по верификации одни и те же индуктивные предикаты из теории предметной области.
2. Если постусловие задано как параметр индуктивного предиката, то можно использовать метод модификации постусловия, чтобы задавать инварианты цикла как индуктивные предикаты из постусловия, но с параметром от счетчика цикла.
3. Вводить промежуточные утверждения с помощью `assert`.
4. Применять SMT-решатели для автоматического доказательства условий корректности.
5. При доказательстве условий корректности в системе Coq использовать плагин QuickChick [75] для поиска возможных контрпримеров к условиям корректности.
6. При доказательстве условий корректности в системе Coq использовать плагин CoqHammer [37], который позволяет с помощью тактики `hammer` использовать SMT-решатели для доказательства условий корректности.
7. При доказательстве условий корректности в системе Coq использовать для генерации доказательств методы машинного обучения с помощью плагина Tactician [27].
8. При доказательстве условий корректности в системе Coq использовать для генерации доказательства Большие Языковые Модели (LLM).
9. При доказательстве условий корректности в системе Coq задавать и применять стратегии доказательства в виде шаблонов, описывающих ситуацию для применения стратегии и схему доказательства, которое нужно генерировать в данных ситуациях.

В нашем проекте мы активно использовали из данного списка подходы 1, 2, 3, 4, 6, 8 и

9.

3. Дедуктивная верификация реализаций классического умножения матриц и умножения матриц с направленными на повышение эффективности использования кэш-памяти оптимизациями

В данном разделе мы представляем как наши практические результаты в виде успешной верификации реализаций умножения матриц с оптимизациями, так и теоретические результаты в виде методов автоматизации доказательства условий корректности для таких матриц на основе введенных нами стратегий.

3.1. Задание спецификаций классического алгоритма умножения матриц

Схема классического алгоритма умножения матриц [2, 82] приведена приведено на рисунке 2.

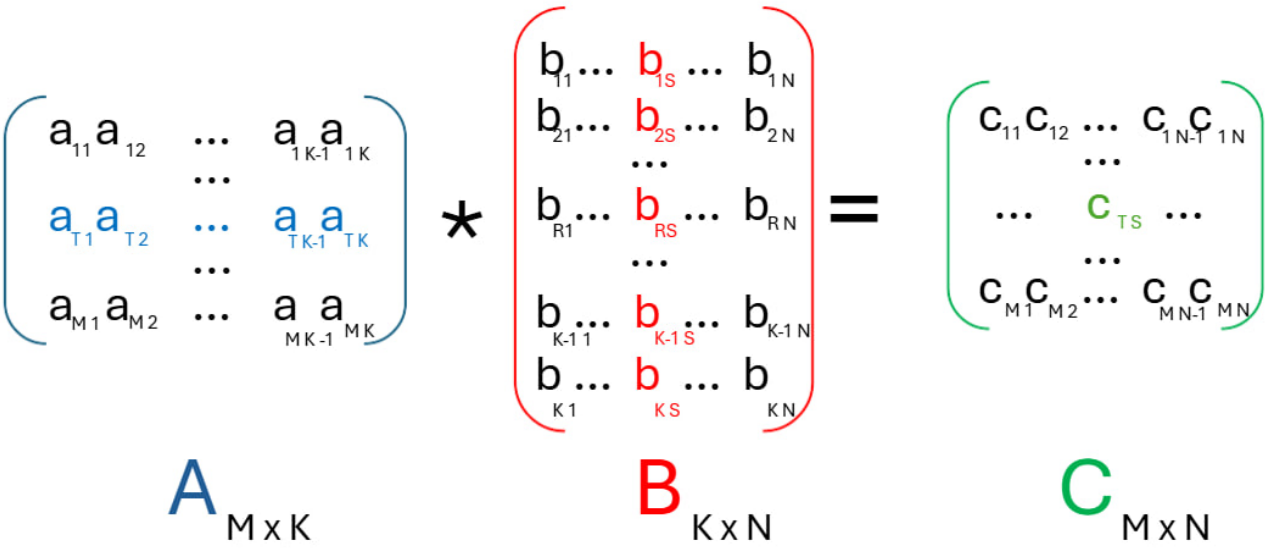


Рис. 2. Схема классического алгоритма умножения матриц

Данной схеме соответствует реализация в виде трех циклов, с которой начинается статья [5]:

```
void gemm_v0(int M, int N, int K, const float * A, const float * B, float *C)
{
    for (int i = 0; i < M; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            C[i*N + j] = 0.0;
            float sum = 0.0;

            for (int k = 0; k < K; ++k)
            {
                sum += A[i*K + k] * B[k*N + j];
            }
            C[i*N + j] = sum;
        }
    }
}
```

Для задания спецификаций такой реализации были заданы индуктивные предикаты. Индуктивный предикат `DotProduction` определяет, что его последний аргумент равен скалярному произведению строки на столбец:

```
    inductive DotProduction{L1}(float* A, float* B, integer to,
                                integer i, integer K, integer j, integer N,
                                real res)
 {
    case dotproduction_empty_range{L1}:
      \forall float* A, float* B, integer to, integer i, integer K, integer j,
      integer N;
      0 >= to ==> DotProduction{L1}(A, B, to, i, K, j, N, 0.0);

    case dotproduction_positive_range{L1}:
      \forall float* A, float* B, integer to, integer i,
      integer K, integer j, integer N, real res;
```

```
    (0 < to) &&
    DotProduction{L1}(A, B, to-1, i, K, j, N, res) ==>
    DotProduction{L1}(A, B, to, i, K, j, N,
    res + \at(A[i*K + (to-1)], L1) * \at(B[(to-1)*N + j], L1));
}
```

Данный предикат соответствует внутреннему циклу реализации.

Индуктивный предикат `RowResult` определяет, что все элементы строки результирующей матрицы заполнены скалярными произведениями строк на столбцы:

```
inductive RowResult{L1, L2}(float* A, float* B, float* C, integer to,
                            integer i, integer K, integer N)
{
  case rowresult_empty{L1, L2}:
    \forall float* A, float* B, float* C, integer to, integer i,
    integer K, integer N;
    0 >= to ==> RowResult{L1, L2}(A, B, C, to, i, K, N);

  case rowresult_step{L1, L2}:
    \forall float* A, float* B, float* C, integer to, integer i,
    integer K, integer N;
    0 < to &&
    RowResult{L1, L2}(A, B, C, to-1, i, K, N) &&
    DotProduction{L1}(A, B, K, i, K, (to-1), N, \at(C[i*N + (to-1)], L2))
    ==> RowResult{L1, L2}(A, B, C, to, i, K, N);
}
```

Таким образом предикат `RowResult` определен с помощью `DotProduction`. Полученный предикат соответствует среднему по вложенности циклу реализации.

Индуктивный предикат `MatrixResult` определяет, что все строки результирующей матрицы заполнены скалярными произведениями строк на столбцы:

```
inductive MatrixResult{L1, L2}(float* A, float* B, float* C, integer to,
                               integer K, integer N, integer M)
{
  case matrixresult_empty{L1, L2}:
```

```
        \forall float* A, float* B, float* C, integer to, integer K, integer N
        integer M;
        0 >= to ==> MatrixResult{L1, L2}(A, B, C, to, K, N, M);

      case matrixresult_step{L1, L2}:
        \forall float* A, float* B, float* C, integer to, integer K,
        integer N, integer M;
        0 < to &&
        MatrixResult{L1, L2}(A, B, C, to-1, K, N, M) &&
        RowResult{L1, L2}(A, B, C, N, to-1, K, N)
        ==> MatrixResult{L1, L2}(A, B, C, to, K, N, M);
    }
  }
```

Таким образом предикат `MatrixResult` определен с помощью `RowResult`. Полученный предикат соответствует внешнему циклу реализации.

Предусловие задано следующим образом:

```
requires \valid_read(A + (0 .. M*K-1));
requires \valid_read(B + (0 .. K*N-1));
requires \valid(C + (0 .. M*N-1));
requires \separated(A + (0 .. M*K-1), B + (0 .. K*N-1), C + (0 .. M*N-1));
requires M > 0 && N > 0 && K > 0;
```

Предикат `valid_read` из предусловия определяет, что из такой области памяти можно читать. Предикат `valid` из предусловия определяет, что можно и писать в такую область памяти и читать из такой области памяти.

Постусловие задано следующим образом:

```
ensures MatrixResult{Pre, Post}(A, B, C, M, K, N, M);
```

Фактически такое постусловие означает, что результаты реализации описываются применением `MatrixResult`.

Инвариант внутреннего цикла задан следующим образом:

```
  loop invariant 0 <= i < M;
              loop invariant 0 <= k <= K;
              loop invariant 0 <= j < N;
```

```
            loop invariant DotProduction{Here}(A, B, k, i, K, j, N, sum);


  loop invariant \separated(A + (0 .. M*K-1), B + (0 .. K*N-1),
            C + (0 .. M*N-1));
            loop assigns sum, k;
            loop variant K-k;
```

Такой инвариант описывает, что результаты внутреннего цикла после `k` итераций описываются применением `DotProduction` от параметра `k`.

Инвариант среднего по вложенности цикла задан следующим образом:

```
 loop invariant 0 <= i < M;
         loop invariant 0 <= j <= N;
         loop invariant RowResult{Pre, Here}(A, B, C, j, i, K, N);


         loop invariant \separated(A + (0 .. M*K-1), B + (0 .. K*N-1),
         C + (0 .. M*N-1));
         loop assigns C[i*N .. i*N+N-1], j;
         loop variant N-j;
```

Такой инвариант описывает, что результаты среднего по вложенности цикла после `j` итераций описываются применением `RowResult` от параметра `j`.

Инвариант внешнего цикла задан следующим образом:

```
      loop invariant 0 <= i <= M;
      loop invariant MatrixResult{Pre, Here}(A, B, C, i, K, N, M);


      loop invariant \separated(A + (0 .. M*K-1), B + (0 .. K*N-1),
      C + (0 .. M*N-1));
      loop assigns C[0 .. M*N-1], i;
      loop variant M - i;
```

Данный инвариант получен методом модификации постусловия в виде замены параметра `MatrixResult` из постусловия на параметр цикла `i`. Получилось, что такой инвариант описывает, что результаты внешнего цикла после `i` итераций описываются применением `MatrixResult` от параметра `i`.

Для упрощения верификации во внутреннем цикле была задана следующая конструк-

ция assert:

```
assert step: DotProduction{Here}(A, B, k+1, i, K, j, N,
                                 sum + A[i*K+k] * B[k*N+j]);
```

Введение такого промежуточного утверждения позволило упростить доказательство сохранения на итерациях инварианта внутреннего цикла.

Для упрощения верификации в среднем по вложенности цикле была задана следующая конструкция assert:

```
assert row_step: RowResult{Pre, Here}(A, B, C, j+1, i, K, N);
```

Введение такого промежуточного утверждения позволило упростить доказательство сохранения на итерациях инварианта среднего по вложенности цикла.

Для упрощения верификации во внешнем цикле была задана следующая конструкция assert:

```
assert mat_step: MatrixResult{Pre, Here}(A, B, C, i+1, K, N, M);
```

Введение такого промежуточного утверждения позволило упростить доказательство сохранения на итерациях инварианта внешнего цикла.

3.2. Задание спецификаций умножения матриц с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы

Реализация умножения матриц с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы была задана на основе статьи [5]:

```
void gemm_v1(int M, int N, int K, const float * A, const float * B, float * C)
{
    for (int i = 0; i < M; ++i)
    {
        float * c = C + i * N;

        for (int j = 0; j < N; ++j)
        {
```

```
            c[j] = 0;
        }

        for (int k = 0; k < K; ++k)
        {
            const float * b = B + k * N;
            float a = A[i*K + k];

            for (int t = 0; t < N; ++t)
            {
                c[t] += a * b[t];
            }
        }
    }
}
```

Отметим, что в такой реализации появился внутренний цикл для заполнения строки результирующей матрицы нулями.

Для задания спецификаций данной реализации была переиспользована теория предметной области реализации классического умножения матриц. Это позволило задать для рассматриваемой реализации то же самое предусловие, то же самое постусловие и тот же самый инвариант внешнего цикла, а также использовать предикат `DotProduction` для задания инвариантов внутренних циклов. Таким образом, фактически с помощью дедуктивной верификации производится проверка эквивалентности заданной в теории предметной области классического умножения матриц и рассматриваемой реализации.

Для задания инварианта внутреннего цикла теория предметной области была расширена индуктивным предикатом `zeroed`:

```
inductive zeroed{L}(float* a, integer b, integer e){
case zeroed_empty{L}:
    \forall float* a, integer b, e; b >= e ==> zeroed{L}(a, b, e);
case zeroed_range{L}:
    \forall float* a, integer b, e; b < e ==>
    zeroed{L}(a, b, e-1) && a[e-1] == 0 ==> zeroed{L}(a,b,e);
```

Данный предикат описывает заполненность области массива нулями.

Рассмотрим новые относительно реализации классического умножения матриц спецификации. Инвариант цикла для заполнения строки результирующей матрицы нулями задан следующим образом:

```
loop invariant 0 <= j <= N;
loop invariant 0 <= i < M;
loop invariant \separated(A + (0 .. M*K-1), B + (0 .. K*N-1),
C + (0 .. M*N-1));
loop invariant zeroed{Here}(C, i*N, i*N + j);

loop assigns j, C[i*N .. i*N + N - 1];
loop variant N-j;
```

Такой инвариант описывает, что результаты цикла для заполнения строки результирующей матрицы нулями после `j` итераций описываются применением `zeroed` от параметра `j`.

Инвариант среднего по вложенности цикла с параметром `k` задан следующим образом:

```
loop invariant 0 <= k <= K;
loop invariant 0 <= i < M;
loop invariant \separated(A + (0 .. M*K-1), B + (0 .. K*N-1),
C + (0 .. M*N-1));
loop invariant \forall integer l; 0<=l<N ==>
DotProduction(A, B, k, i, K, l, N, c[l]);
loop assigns k, C[i*N .. i*N + N - 1];
loop variant K-k;
```

Отметим, что результаты среднего по вложенности цикла с параметром `k` можно описать с помощью `DotProduction`.

Инвариант внутреннего цикла с параметром `t` задан следующим образом:

```
loop invariant 0 <= t <= N;
loop invariant 0 <= k < K;
loop invariant 0 <= i < M;
loop invariant \separated(A + (0 .. M*K-1), B + (0 .. K*N-1),
C + (0 .. M*N-1));
```

```
loop invariant \forall integer q; 0<=q<t ==>
DotProduction(A, B, k+1, i, K, q, N, c[q]);
loop invariant \forall integer q; t <= q < N ==>
DotProduction(A, B, k, i, K, q, N, c[q]);
loop assigns t, C[i*N .. i*N + N - 1];
loop variant N-t;
```

Отметим, что результаты внутреннего цикла с параметром `t` можно описать с помощью `DotProduction`.

Для упрощения верификации во внутреннем цикле с параметром `t` была задана следующая конструкция `assert`:

```
assert step: DotProduction(A, B, k+1, i, K, t, N, c[t]+a*b[t]);
```

Введение такого промежуточного утверждения позволило упростить доказательство сохранения на итерациях инварианта внутреннего цикла.

Для упрощения верификации в среднем по вложенности цикле с параметром `k` была задана следующая конструкция `assert`:

```
assert dot_done:  \forall integer l; 0<=l<N ==>
            DotProduction(A, B, k+1, i, K, l, N, c[l]);
```

Введение такого промежуточного утверждения позволило упростить доказательство сохранения на итерациях инварианта среднего по вложенности цикла с параметром `k`.

Для упрощения верификации в цикле для заполнения строки результирующей матрицы нулями была задана следующая конструкция `assert`:

```
assert zeroed{Here}(C, i*N, i*N + j + 1);
```

Введение такого промежуточного утверждения позволило упростить доказательство сохранения на итерациях инварианта цикла для заполнения строки результирующей матрицы нулями.

Для упрощения верификации между циклом для заполнения строки результирующей матрицы нулями и средним по вложенности циклом с параметром `k` была задана следующая конструкция `assert`:

```
assert zeroed{Here}(C, i*N, i*N + N);
```

Введение такого промежуточного утверждения позволило упростить доказательство условия выполнения инварианта на входе в средний по вложенности цикл с параметром `k`.

Для упрощения верификации после среднего по вложенности цикла с параметром `k` была задана следующая конструкция `assert`:

```
assert row_done: RowResult{Here}(A, B, C, N, i, K, N);
```

Введение такого промежуточного утверждения позволило доказать, что результат среднего по вложенности цикла с параметром `k` описывается применением `RowResult`.

Для упрощения верификации во внешнем цикле была задана следующая конструкция `assert`:

```
assert mat_step: MatrixResult{Here}(A, B, C, i+1, K, N, M);
```

Введение такого промежуточного утверждения позволило упростить доказательство сохранения на итерациях инварианта внешнего цикла.

3.3. Доказательство условий корретности

И в случае верификации реализации обеих реализаций абсолютное большинство условий корректности были доказаны SMT-решателями. При этом особый интерес представляют случаи, когда условия корректности пришлось доказывать в интерактивной системе доказательства Coq. Для автоматизации такого доказательства были активно использованы возможности SMT-решателей для доказательства подцелей с помощью плагина CoqHammer, а также генерация доказательств с помощью большой языковой модели Gemini [53]. Отметим, что нами для автоматизации доказательства были выделены и применены стратегии доказательства в виде шаблонов, описывающих ситуацию для применения стратегии и схему доказательства, которое нужно генерировать в данных ситуациях. Рассмотрим такие стратегии подробнее. Пример условия корректности и его доказательства, где применяются все введенные нами стратегии доказательства, можно посмотреть в приложении А.

3.3.1. Стратегия доказательства того, что разделены области памяти, в которые осуществляется запись и из которых осуществляется чтение

Данная стратегия применяется в таких случаях, когда нужно доказать, что разделены области памяти в которые осуществляется запись и из которых осуществляется чтение. Отметим, что такие случаи встречаются в условиях корректности обеих реализаций, что указывает на востребованность введения такой стратегии. Особенно отметим, что данная стратегия применяется внутри определенной далее стратегии доказательства того, что

при записи в одну область памяти значения в другой области памяти не изменяются.

Так как разделение областей памяти в системе Frama-C/WP определяется с помощью предиката `separated`, то приведем определение такого предиката:

```
Definition separated (p:addr) (a:Numbers.BinNums.Z) (q:addr)
    (b:Numbers.BinNums.Z) : Prop :=
  (a <= 0%Z)%Z \/
  (b <= 0%Z)%Z \/
  ~ ((base p) = (base q)) \/
  (((offset q) + b)%Z <= (offset p))%Z \/
  (((offset p) + a)%Z <= (offset q))%Z.
```

Данная стратегия применяется тогда, когда есть гипотеза `Hxx` в которой говорится о том, что память разделена.

Отметим, что данная стратегия разбивается на два случая, отличающихся тем, когда именно возникает необходимость в применении данной стратегии.

Случай записи в одну область памяти и чтения из другой области памяти В данном случае стратегия зависит от следующих метапеременных:

- `R` – массив из которого читаем;
- `W` – массив в который пишем;
- `base_R`, `base_W` – адреса массивов;
- `offset_R`, `offset_W` – смещения внутри массивов;
- `size_R`, `size_W` – размеры массивов;
- `idx` – индекс ячейки которую читаем из R;
- `idx_write` – индекс ячейки в которую пишем в W.

Рассмотрим схему доказательства, порождаемую стратегией в данном случае:

```
(* Перед началом делаем:
   destruct R, W.
   unfold base, offset, shift, separated in *.
   injection Heq as HeqBase HeqOff.
*)
H_SEP: separated R size_R W size_W
```

```
destruct H_SEP as [Hb1 | [Hb2 | [Hb3 | [Hb4 | Hb5]]]].

(* ================================================================ *)
(* ВЕТКА 1: Размер массива ЧТЕНИЯ <= 0 *)
(* ================================================================ *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   H_dim1 : (0 < M)%Z  (Высота матрицы R > 0)
   H_dim2 : (0 < N)%Z  (Ширина матрицы R > 0)
   H_size1 : size_R = (M * N)%Z
   Hb1    : (size_R <= 0)%Z
*)
- nia.

(* ================================================================ *)
(* ВЕТКА 2: Размер массива ЗАПИСИ <= 0 *)
(* ================================================================ *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   H_dim1 : (0 < M2)%Z  (Высота матрицы W > 0)
   H_dim2 : (0 < N2)%Z  (Ширина матрицы W > 0)
   H_size1 : size_W = (M2 * N2)%Z
   Hb2    : (size_W <= 0)%Z
*)
- nia.

(* ================================================================ *)
(* ВЕТКА 3: адреса разные *)
(* ================================================================ *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   HeqBase : base_R = base_W
```

```
   Hb3     : base_R <> base_W
*)
- right. right. left. exact Hb3.
  (* ИЛИ просто ‘congruence.‘ *)

(* ================================================================ *)
(* ВЕТКА 4: Массив ЗАПИСИ (W) лежит целиком ДО массива ЧТЕНИЯ (R) *)
(* (Запись <= Конец_W <= Начало_R <= Чтение) *)
(* ================================================================ *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   Hb4          : (offset_W + size_W <= offset_R)%Z
   HeqOff       : (offset_W + idx_write = offset_R + idx)%Z
   H_read_min   : (0 <= idx)%Z
   H_write_outer: (i < M)%Z (Текущая строка записи < Высоты матрицы W)
   H_write_inner: (j < N)%Z (Текущий столбец записи < Ширины матрицы W)
   H_idx_w  : (idx_write < size_W)%Z
*)
- right. right. right. left.
  (* Подсказка 1: Адрес чтения не уходит левее начала массива R *)
  assert (HintR: (offset_R <= offset_R + idx)%Z) by nia.

  (*offset_W + size_w <= offset_W + idx_write)

  (* Подсказка 2: Адрес записи не превышает конец массива W. *)
  assert (HintW:
  (offset_W + idx_write + 1 <= offset_W + size_W)%Z) by nia.
  (* Противоречие *)
  nia.

(* ================================================================ *)
(* ВЕТКА 5: Массив ЧТЕНИЯ (R) лежит целиком ДО массива ЗАПИСИ (W) *)
```

```
(* (Чтение < Конец_R <= Начало_W <= Запись) *)
(* ============================================================== *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   Hb5          : (offset_R + size_R <= offset_W)%Z
   HeqOff       : (offset_R + idx = offset_W + idx_write)%Z
   H_read_max   : (idx < size_R)%Z
   H_write_outer_min : (0 <= i)%Z (Текущая строка записи >= 0)
   H_write_inner_min : (0 <= j)%Z (Текущий столбец записи >= 0)
   H_idx_wr : (idx_write >= 0)%Z
*)
- right. right. right. right.
  (* Подсказка 1: Адрес чтения строго меньше конца массива R *)
  assert (HintR: (offset_R + idx + 1 <= offset_R + size_R)%Z) by nia.

  (*offset_W + idx_write + 1<= offset_R + size_R <= offset W*)

  (* Подсказка 2: Адрес записи не уходит левее начала массива W. *)
  assert (HintW: (offset_W <= offset_W + idx_write)%Z) by nia.
  (* Противоречие *)
  nia.
```

Отметим, что такая схема доказательства разбивается на 5 ветвей, так как определение `separated` разбивается на 5 логических "или".

Случай применения `havoc` к блоку памяти В данном случае стратегия зависит от следующих метапеременных:

- `R` – массив из которого мы читаем;
- `W` – массив, в который мы писали и где применен `havoc`;
- `base_R`, `base_W` – адреса массивов;
- `off_R`, `off_W` – смещения внутри массивов;
- `size_R`, `size_W` – размеры массивов;
- `idx` – индекс ячейки, которую мы читаем из `R`;

- `idx_write_start` – индекс начала блока, который мы меняем в `W`.
- `size_W_havoc` – размер блока в `W`.

Рассмотрим схему доказательства, порождаемую стратегий в данном случае:

```
(* Перед началом делаем:
   destruct R as [base_R off_R].
   destruct W as [base_W off_W].
   unfold separated, shift, base, offset,
   <ПЕРЕМЕННЫЕ_АДРЕСОВ_ТИПА_x1_x2_a4_a5> in *.
   simpl in *.
*)

H_SEP: separated R size_R W size_W

destruct H_SEP as [Hb1 | [Hb2 | [Hb3 | [Hb4 | Hb5]]]].

(* ================================================================ *)
(* ВЕТКА 1: Размер массива ЧТЕНИЯ <= 0 *)
(* ================================================================ *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   H_dim1 : (0 < M)%Z  (Высота матрицы R > 0)
   H_dim2 : (0 < N)%Z  (Ширина матрицы R > 0)
   H_size1 : size_R = (M * N)%Z
   Hb1    : (size_R <= 0)%Z
*)
- nia.

(* ================================================================ *)
(* ВЕТКА 2: Размер массива ЗАПИСИ <= 0 *)
(* ================================================================ *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
```

```
   H_dim1 : (0 < M2)%Z  (Высота матрицы W > 0)
   H_dim2 : (0 < N2)%Z  (Ширина матрицы W > 0)
   H_size1 : size_W = (M2 * N2)%Z
   Hb2    : (size_W <= 0)%Z
*)
- nia.

(* ================================================================ *)
(* ВЕТКА 3: адреса разные *)
(* ================================================================ *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   Hb3    : base_R <> base_W
   Hxx    : base_R = base_W
*)
- (* Выбираем 3-е условие в нашей текущей цели separated *)
  right. right. left.
  exact Hb3.

(* ================================================================ *)
(* ВЕТКА 4: Массив записи (W) лежит целиком до массива чтения (R) *)
(* (Блок записи <= Конец_W <= Начало_R <= Индекс чтения) *)
(* ================================================================ *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   Hb4          : (off_W + size_W <= off_R)%Z
   Hidx_min     : (0 <= idx)%Z (Читаемый индекс >= 0)
   H_write_dims : Гипотезы о границах циклов,
                  чтобы доказать, что блок havoc не вылез за массив W.
*)
- (* Выбираем 4-е условие в нашей текущей цели separated *)
  right. right. right. left.
```

```
  (* Подсказка 1: Изменяемый блок не выходит за конец всего массива W *)
  assert (HintW: (idx_write_start + size_W_havoc <= size_W)%Z) by nia.

  (* Подсказка 2 (не всегда обязательна, но надежна): Адрес чтения
  не уходит в минус *)
  assert (HintR: (off_R <= off_R + idx)%Z) by nia.

  (* nia выстраивает цепочку: Конец_Блока_W <=
  Конец_W (HintW) <= Начало_R (Hb4) <= Индекс_Чтения_R (HintR) *)
  nia.

(* ================================================================= *)
(* ВЕТКА 5: Массив чтения (R) лежит целиком до массива записи (W) *)
(* (Индекс чтения < Конец_R <= Начало_W <= Блок записи) *)
(* ================================================================= *)
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   Hb5          : (off_R + size_R <= off_W)%Z
   Hidx_max     : (idx < size_R)%Z (Читаемый индекс строго внутри
   массива R)
   H_write_dims : Гипотезы о том, что блок записи не уходит в минус
   (idx_write_start >= 0)
*)
- (* Выбираем 5-е условие в нашей текущей цели separated *)
  right. right. right. right.

  (* Подсказка 1: Читаемая ячейка (idx) строго меньше конца всего
  массива R *)
  (* (Мы добавляем +1, так как размер ячейки равен 1) *)
  assert (HintR:
  (off_R + idx + 1 <= off_R + size_R)%Z) by nia.
```

```
(* Подсказка 2: Блок записи не уходит левее начала массива W *)
(* (Обычно это очевидно, так как индексы циклов i, j >= 0) *)
assert (HintW:
(off_W <= off_W + idx_write_start)%Z) by nia.

(* nia выстраивает цепочку: Конец_Ячейки_R (HintR) <= Конец_R
<= Начало_W (Hb5) <= Начало_Блока_W (HintW) *)
nia.
```

Отметим, что такая схема доказательства тоже разбивается на 5 ветвей, так как определение `separated` разбивается на 5 логических "или".

3.3.2. Стратегия доказательства того, что при записи в одну область памяти значения в другой области памяти не изменяются

Данная стратегия применяется в таких случаях, когда нужно доказать, что для разделенных с помощью `separated` областей памяти при записи в одну область памяти значения в другой области памяти не изменяются. Необходимость в данной стратегии возникает из-за следующей особенности Frama-C/WP: у Frama-C/WP единая память и после того, как происходит запись в массив `C`, гипотезы о том, что матрицы `A` и `B` не меняются, верны для старой памяти, но у нас уже появляется новая память, и в ней приходится это доказывать. Данная стратегия нужна чтобы доказать что матрицы `A` и `B` в новой и старой памяти совпадают. Отметим, что такие случаи встречаются в условиях корректности обеих реализаций, что указывает на востребованность введения такой стратегии.

Так как результат изменения памяти в системе Frama-C/WP определяется с помощью конструкции `havoc`, то приведем описывающую данную конструкцию аксиому:

```
Axiom havoc_access :
  forall {a:Type} {a_WT:WhyType a},
  forall (m0:addr -> a) (m1:addr -> a), forall (q:addr) (p:addr),
  forall (a1:Numbers.BinNums.Z),
  (separated q 1%Z p a1 -> ((havoc m0 m1 p a1 q) = (m1 q))) /\
  (~ separated q 1%Z p a1 -> ((havoc m0 m1 p a1 q) = (m0 q))).
```

Отметим, что данная стратегия разбивается на два случая, отличающихся тем, когда

именно возникает необходимость в применении данной стратегии.

Случай доказательства неизменности значений при применении `havoc` В данном случае стратегия применяется, когда гипотеза в доказательстве удовлетворяет следующему шаблону:

```
(*
   [ОБЩИЕ НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   <ГИПОТЕЗА_SEP> : separated ... (Глобальное разделение
   МАССИВА_ЧТЕНИЯ и МАССИВА_ЗАПИСИ)
*)
```

а цель доказательства удовлетворяет следующему шаблону:

```
assert (Eq_Memory: forall idx, (0 <= idx)%Z -> (idx <
<РАЗМЕР_МАССИВА_ЧТЕНИЯ>)%Z ->
    <НОВАЯ_ПАМЯТЬ> (shift <МАССИВ_ЧТЕНИЯ> idx) = <СТАРАЯ_ПАМЯТЬ>
    (shift <МАССИВ_ЧТЕНИЯ> idx)).
```

Рассмотрим схему доказательства, порождаемую стратегией для такой цели:

```
assert (Eq_Memory: forall idx, (0 <= idx)%Z -> (idx <
<РАЗМЕР_МАССИВА_ЧТЕНИЯ>)%Z ->
    <НОВАЯ_ПАМЯТЬ> (shift <МАССИВ_ЧТЕНИЯ> idx) = <СТАРАЯ_ПАМЯТЬ>
    (shift <МАССИВ_ЧТЕНИЯ> idx)).
{
  intros idx Hidx_min Hidx_max.

  (* Раскрываем переменные памяти(a6, a8, x2, x3 и т.д.) *)
  unfold <НОВАЯ_ПАМЯТЬ>, <shift_X1_X2>.

  (* Используем аксиому доступа для нашего конкретного читаемого адреса.
     АРГУМЕНТЫ ПО ПОРЯДКУ:
     1. tX (отображение addr->real)
     2. СТАРАЯ_ПАМЯТЬ (например Mf32, t1)
     3. АДРЕС_НАЧАЛА_ЗАПИСИ (shift <МАССИВ_ЗАПИСИ> <СМЕЩЕНИЕ_ЗАПИСИ>)
```

```
     4. РАЗМЕР_МАССИВА
     5. ЧИТАЕМЫЙ_АДРЕС (shift <МАССИВ_ЧТЕНИЯ> idx) *)

  destruct (havoc_access <tX> <СТАРАЯ_ПАМЯТЬ>
                         (shift <МАССИВ_ЗАПИСИ> <СМЕЩЕНИЕ_ЗАПИСИ>)
                         <РАЗМЕР_МАССИВА>
                         (shift <МАССИВ_ЧТЕНИЯ> idx)) as [Hacc _].

  (* Применяем левую часть аксиомы (условие, что адреса разделены) *)
  rewrite Hacc; [reflexivity |].

  (* Разбиваем абстракции Coq до чистых чисел *)
  (* Адрес в массиве - это пара начало массива и сдвиг *)
  destruct <МАССИВ_ЧТЕНИЯ> as [base_R off_R].
  destruct <МАССИВ_ЗАПИСИ> as [base_W off_W].

  (* Раскрываем определения для получения неравенств над
  числами, удобных для тактики nia. *)
  unfold separated, shift, base, offset,
  <ПЕРЕМЕННЫЕ_АДРЕСОВ_a3_a4_a5> in *.
  simpl in *. (* Избавляемся от конструкций match *)

  (* Применяем стратегию доказательства того, что разделены
 области памяти, в которые осуществляется запись и из которых
 осуществляется чтение *)
  destruct <ГИПОТЕЗА_SEP> as [Hb1 | [Hb2 | [Hb3 | [Hb4 | Hb5]]]].
  ...
}
```

Отметим, что рассматриваемая стратегия применяет рассмотренную ранее стратегию доказательства того, что разделены области памяти, в которые осуществляется запись и из которых осуществляется чтение.

Случай доказательства неизменности значений при применении `MAP.SET` В данном случае стратегия применяется, когда гипотеза в доказательстве удовлетворяет следующему шаблону:

```
(*
   [ОБЩИЕ НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   <ГИПОТЕЗА_SEP> : separated ... (Глобальное разделение
   МАССИВА_ЧТЕНИЯ и МАССИВА_ЗАПИСИ)
*)
```

а цель доказательства удовлетворяет следующему шаблону:

```
assert (Eq_Memory: forall idx, (0 <= idx)%Z -> (idx <
<РАЗМЕР_МАССИВА_ЧТЕНИЯ>)%Z ->
    <НОВАЯ_ПАМЯТЬ> (shift <МАССИВ_ЧТЕНИЯ> idx) = <СТАРАЯ_ПАМЯТЬ>
    (shift <МАССИВ_ЧТЕНИЯ> idx)).
```

Рассмотрим схему доказательства, порождаемую стратегией для такой цели:

```
assert (Eq_Memory: forall idx, (0 <= idx)%Z -> (idx <
<РАЗМЕР_МАССИВА_ЧТЕНИЯ>)%Z ->
    <НОВАЯ_ПАМЯТЬ> (shift <МАССИВ_ЧТЕНИЯ> idx) = <СТАРАЯ_ПАМЯТЬ>
    (shift <МАССИВ_ЧТЕНИЯ> idx)).
{
  intros idx Hidx_min Hidx_max.

  (* Раскрываем новую память и адрес записи *)
  unfold <НОВАЯ_ПАМЯТЬ>, <МАКРОС_АДРЕСА_ЗАПИСИ_A8>, Map.set.

  (* Разбиваем цель на 2 ветки: адреса совпали (Heq) или
  разные (Hneq) *)
  destruct (why_decidable_eq (shift <МАССИВ_ЗАПИСИ> <ИНДЕКС_ЗАПИСИ>)
  (shift <МАССИВ_ЧТЕНИЯ> idx)) as [Heq | Hneq].

  - (* АДРЕСА СОВПАЛИ. Доказываем, что это невозможно *)
    (* Адрес в массиве - это пара начало массива и сдвиг *)
```

```
    destruct <МАССИВ_ЧТЕНИЯ> as [base_R off_R].
    destruct <МАССИВ_ЗАПИСИ> as [base_W off_W].

    (* Раскрываем определения для получения неравенств над числами
       чтобы потом применить injection*)
    unfold separated, shift, base, offset,
    <ПЕРЕМЕННЫЕ_АДРЕСОВ_a3_a4_a5> in *.
    simpl in *.

    (* Извлекаем уравнение баз и смещений из факта совпадения адресов *)
    injection Heq as HeqBase HeqOff.

    (* Применяем стратегию доказательства того, что разделены
    области памяти, в которые
    осуществляется запись и из которых осуществляется чтение *)
    destruct <ГИПОТЕЗА_SEP> as [Hb1 | [Hb2 | [Hb3 | [Hb4 | Hb5]]]].
    ...
    - (* АДРЕСА РАЗНЫЕ *)
      reflexivity.
}
```

Отметим, что и в данном случае рассматриваемая стратегия применяет рассмотренную ранее стратегию доказательства того, что разделены области памяти, в которые осуществляется запись и из которых осуществляется чтение.

3.3.3. Стратегия доказательства того, что при записи в одну область памяти значения индуктивного предиката на другой области памяти остаются прежними

Данная стратегия применяется в таких случаях, когда нужно доказать, что при записи в одну область памяти значения индуктивного предиката на другой области памяти остаются прежними. Отметим, что такие случаи встречаются в условиях корректности обеих реализаций, что указывает на востребованность введения такой стратегии.

Так как данная стратегии зависит от индуктивного предиката, то приведем применяе-

мое в данной стратегии шаблонное определение индуктивного предиката:

```
(* Шаблон определения индуктивного приката *)
Inductive <ИМЯ_ПРЕДИКАТА>:
  (addr -> <ТИП_ЗНАЧЕНИЯ>) -> (* 1. Состояние памяти (Mem) *)
  addr -> ... ->              (* 2. Адреса читаемых
  массивов (Ptr1, Ptr2...) *)
  Numbers.BinNums.Z ->        (* 3. Параметры,ограничивающие
  обрабатываемую часть массива (from / to) *)
  Numbers.BinNums.Z -> ... -> (* 4. Константные параметры
  (Размеры, индексы строк/столбцов) *)
  <ТИП_ЗНАЧЕНИЯ> -> Prop :=   (* 5. НАКОПЛЕННЫЙ РЕЗУЛЬТАТ (res) *)

  | <КОНСТРУКТОР_БАЗОВОГО_СЛУЧАЯ> :
      forall (Mem: addr -> <ТИП_ЗНАЧЕНИЯ>)
             (Ptr1...: addr)
             (to: Numbers.BinNums.Z)
             (Ctx1...: Numbers.BinNums.Z),

      (* Условие: Длина <= 0 *)
      (to <= 0%Z)%Z ->

      <ИМЯ_ПРЕДИКАТА> Mem Ptr1... to Ctx1...
      <Начальное_значение_результата(res)>

  | <КОНСТРУКТОР_ШАГА_ИНДУКЦИИ> :
      forall (Mem: addr -> <ТИП_ЗНАЧЕНИЯ>)
             (Ptr1...: addr)
             (to: Numbers.BinNums.Z)
             (Ctx1...: Numbers.BinNums.Z)
             (OldRes: <ТИП_ЗНАЧЕНИЯ>),

      (* Создание переменной для предыдущего шага *)
```

```
      let prev_to := ((-1%Z)%Z + to)%Z in

      (* Условие продолжения и рекурсивный вызов для
      старого результата *)
      (0%Z < to)%Z ->
      <ИМЯ_ПРЕДИКАТА> Mem Ptr1... prev_to Ctx1... OldRes ->
      (<ФУНКЦИЯ_ОБРАБОТКИ_prev_to> Mem shift (Ptr1 ...) ...
      prev_to Ctx1) ->
      (* Заключение: Предикат для текущей длины равен комбинации
         старого результата и чтения из памяти *)
      <ИМЯ_ПРЕДИКАТА> Mem Ptr1... to Ctx1...
```

Отметим, что данному шаблону соответствуют заданные нами в теории предметной области индуктивные предикаты `DotProduction`, `RowResult` и `MatrixResult`, где также выделены отдельные свойства для базы индукции и шага индукции.

Стратегия применяется, когда гипотеза в доказательстве удовлетворяет следующему шаблону:

```
(*
   [НЕОБХОДИМЫЕ ГИПОТЕЗЫ В КОНТЕКСТЕ]:
   EqA : forall idx, ... -> <НОВАЯ_ПАМЯТЬ> (...) = <СТАРАЯ_ПАМЯТЬ> (...)
   EqB : forall idx, ... -> <НОВАЯ_ПАМЯТЬ> (...) = <СТАРАЯ_ПАМЯТЬ> (...)
*)
```

а цель доказательства удовлетворяет следующему шаблону:

```
assert (Hframe_pred: forall <ПАРАМЕТРЫ_ИНДУКЦИИ> <РЕЗУЛЬТАТ>,
    <ГРАНИЦЫ_ДЛЯ_ПАРАМЕТРОВ> ->
    <ПРЕДИКАТ> <СТАРАЯ_ПАМЯТЬ> <МАТРИЦЫ> <ИНДЕКСЫ> <РЕЗУЛЬТАТ> ->
    <ПРЕДИКАТ> <НОВАЯ_ПАМЯТЬ> <МАТРИЦЫ> <ИНДЕКСЫ> <РЕЗУЛЬТАТ>).
```

Рассмотрим схему доказательства, порождаемую стратегией для такой цели:

```
assert (Hframe_pred: forall <ПАРАМЕТРЫ_ИНДУКЦИИ> <РЕЗУЛЬТАТ>,
    <ГРАНИЦЫ_ДЛЯ_ПАРАМЕТРОВ> ->
    <ПРЕДИКАТ> <СТАРАЯ_ПАМЯТЬ> <МАТРИЦЫ> <ИНДЕКСЫ> <РЕЗУЛЬТАТ> ->
    <ПРЕДИКАТ> <НОВАЯ_ПАМЯТЬ> <МАТРИЦЫ> <ИНДЕКСЫ> <РЕЗУЛЬТАТ>).
{
```

```
(* ШАГ 1: Вводим переменные и гипотезу предиката *)
intros <ПАРАМЕТРЫ_ИНДУКЦИИ> <РЕЗУЛЬТАТ> H_bounds H_pred_old.

(* ШАГ 2: СОХРАНЕНИЕ ПРЕЖНИХ ПЕРЕМЕННЫХ *)
(* Если в <ИНДЕКСЫ> есть сложные математические формулы
(например, -1 + to), при индукции в Coq они могут быть потеряны.
Запишем их в не участвующие в индукции переменные: *)
remember <СЛОЖНАЯ_ФОРМУЛА_1> as idx_1 eqn:Heq_idx1.
remember <СЛОЖНАЯ_ФОРМУЛА_2> as idx_2 eqn:Heq_idx2.
(* Также полезно запомнить старую память, чтобы не возникла
путаница с новой *)
remember <СТАРАЯ_ПАМЯТЬ> as mem_old.

(* ШАГ 3: Запускаем индукцию по предикату в случае старой памяти*)
induction H_pred_old.

- (* ВЕТКА 1: Базовый случай (например, длина = 0) *)
  (* Возвращаем запомненные переменные из 'remember' обратно,
  если нужно *)
  subst.
  apply <КОНСТРУКТОР_EMPTY_RANGE>.
  lia.

- (* ВЕТКА 2: Шаг индукции (добавление нового элемента) *)
  (* Возвращаем запомненные переменные из 'remember' обратно,
  если нужно *)
  subst.

  (* ШАГ 4: Переписываем память в текущей цели (с новой на старую).
     Используем наши леммы EqA/EqB, доказанные ранее.
     Квадратные скобки автоматически доказывают индексы для EqA/EqB. *)
  rewrite <- EqA; [ | lia | nia ].
```

```
    rewrite <- EqB; [ | lia | nia ].
    (* rewrite EqC. -- (если есть равенство для матрицы C) *)

    (* Теперь память в формулах суммы/результата совпадает
    со СТАРОЙ памятью *)

    (* ШАГ 5: Применяем конструктор индуктивного шага *)
    apply <КОНСТРУКТОР_POSITIVE_RANGE>.

    + (* Подцель 5.1: Доказать, что длина > 0 *)
      lia.

    + (* Подцель 5.2: Доказать индукционный переход *)
      (* Применяем гипотезу индукции, которую сгенерировал Coq
      (IHH_pred_old) и доказываем её *)
      apply IHH_pred_old; try assumption; try nia; try lia; try reflexivity.
}
```

Отметим, что необходимость в данной стратегии возникает, например, при таких распространенных при работе с массивами случаях, когда нужно доказать, что при записи в ячейку массива значение индуктивного предиката на предыдущих ячейках массива не изменяется.

4. Заключение

В данной статье получены следующие результаты:

1. Заданы спецификации для реализаций классического умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы.
2. Задана теория предметной области для дедуктивной верификации реализации классического умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы.

3. Разработаны стратегии доказательства условий корректности при дедуктивной верификации реализации классического умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы. Разработаны следующие стратегии:
   (a) Стратегия доказательства того, что разделены области памяти, в которые осуществляется запись и из которых осуществляется чтение.
   (b) Стратегия доказательства того, что при записи в одну область памяти значения в другой области памяти не изменяются.
   (c) Стратегия доказательства того, что при записи в одну область памяти значения индуктивного предиката на другой области памяти остаются прежними.
4. Проведены эксперименты по дедуктивной верификации реализации классического умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы..

Отметим, что данные результаты могут служить прототипом комплексного подхода к дедуктивной верификации реализаций умножения матриц над математическими вещественными числами с направленными на повышение эффективности использования кэш-памяти оптимизациями.

Мы планируем дальше следовать планам нашего проекта по дедуктивной верификации все более и более оптимизированных реализаций умножения матриц из статьи [5], чтобы достить цели в виде дедуктивной верификации реализации minigemm.

Список литературы

1. Агибалов М.С. Верификация алгоритмов умножения матриц. URL: `https://github.com/ylab-nsu/wb26-trust-minigemm/tree/GemmV1-d` (дата обращения: 20.05.2026)
2. Алексеев Е.Р., Демин П.А., Болтачева Н.Ю. Новые технологии разработки высокоэффективных и параллельных приложений на современном Фортране // Прикладная информатика. 2018. Т. 13, № 1. С. 103–120.
3. Васенин В.А., Кривчиков М.А. Формальные модели программ и языков программирования. Часть 1. Библиографический обзор 1930–1989 гг. // Программная инженерия. 2015. № 5. С. 10–19.
4. Васенин В.А., Кривчиков М.А. Формальные модели программ и языков программирования.

Часть 2. Современное состояние исследований // Программная инженерия. 2015. № 6. С. 24–33.

5. Ермолаев И. Умножение матриц: эффективная реализация шаг за шагом. 2019. [Электронный ресурс]. URL: https://habr.com/ru/articles/359272/ (дата обращения: 20.05.2026)
6. Камкин А.С. Введение в формальные методы верификации программ. М.: ДМК Пресс, 2024. – 304 с.
7. Кокорин А.О., Тиевский С.Д., Девянин П.Н. Приемы дедуктивной верификации программного кода с использованием AstraVer Toolset // Прикладная дискретная математика. Приложение. 2022. № 15. С. 80–90.
8. Кондратьев Д.А., Бояндин Л.К., Гончар Г.Е., Марченко В.В., Обухова А.А., Разбитнова Ю.Ю., Хованская А.С., Янбулатов Д.Р. Формальная верификация реализации хэш-функции «Стрибог» с «Группой Астра» // Системная информатика. 2025. № 28. С. 25–52.
9. Кондратьев Д.А., Старолетов С.М., Шошмина И.В., Красненкова А.В., Зиборов К.В., Шилов Н.В., Гаранина Н.О., Черганов Т.Ю. Соревнования по формальной верификации VeHa-2024: накопленный в течение двух лет опыт и перспективы // Труды Института системного программирования РАН. 2025. Т. 37. № 1. С. 159–184.
10. Лейно К.Р.М. Доказательство корректности программ. М.: ДМК Пресс, 2024. – 522 с.
11. Лылова С.С., Власов А.А., Латкин Е.И., Знаменский И.И., Волокитин В.Д. Исследование производительности умножения плотных матриц в библиотеке OpenBLAS на архитектуре RISC - V с использованием векторных инструкций // Вычислительные технологии. 2026. Т. 31. № 2. С. 104–119.
12. Маркова В.П., Киреев С.Е., Остапкевич М.Б., Перепелкин В.А. Эффективное программирование современных микропроцессоров: учебное пособие. Новосибирск: Издательство НГТУ, 2014. – 148 с.
13. Миронов А.М. Методы верификации программ. М.: ДМК Пресс. 2023 – 332 с.
14. Непомнящий В.А., Рякин О.М. Прикладные методы верификации программ. М.: Радио и связь, 1988. – 256 с.
15. Старолетов С.М., Кондратьев Д.А., Гаранина Н.О., Шошмина И.В. Соревнования по формальной верификации VeHa-2023: опыт проведения // Труды Института системного программирования РАН. 2024. Т. 36. № 2. С. 141–168.
16. Шилов Н.В. Основы синтаксиса, семантики, трансляции и верификации программ: учебное пособие. Новосибирск: Издательство Новосибирского государственного университета, 2011. – 292 с.
17. Appel A.W. Verified Software Toolchain // Lecture Notes in Computer Science. 2011. Volume 6602. pp. 1–17.
18. Appel A.W., Beringer L., Cao Q., Dodds J. Verifiable C: Applying the Verified Software Toolchain to C programs. 2023. URL: https://github.com/PrincetonUniversity/VST/raw/master/doc/VC.pdf (Accessed 25 Jul 2025)

19. Apt K.R., Olderog E.-R. Assessing the Success and Impact of Hoare's Logic // Theories of Programming: The Life and Works of Tony Hoare. New York: ACM, 2021. pp. 41–76.
20. Apt K.R., Olderog E.-R. Fifty years of Hoare's logic // Formal Aspects of Computing. 2019. Volume 31. Issue 6. pp. 751–807.
21. Baudin P., Bobot F., Bühler D., Correnson L., Kirchner F., Kosmatov N., Maroneze A., Perrelle V., Prevosto V., Signoles J., Williams N. The dogged pursuit of bug-free C programs: the Frama-C software analysis platform // Communications of the ACM. 2021. Volume 64. Issue 8. pp. 56–68.
22. Barbosa H., Barrett C., Brain M., Kremer G., Lachnitt H., Mann M., Mohamed A., Mohamed M., Niemetz A., Nötzli A., Ozdemir A., Preiner M., Reynolds A., Sheng Y., Tinelli C., Zohar Y. cvc5: A Versatile and Industrial-Strength SMT Solver // Lecture Notes in Computer Science. Volume 13243. pp. 415–442.
23. Barnett M., Leino K.R.M. Weakest-precondition of unstructured programs // Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering. Lisbon, Portugal, September 5–6, 2005. New York: Association for Computing Machinery, 2005. pp. 82–87.
24. Barrett C., Conway C.L., Deters M., Hadarean L., Jovanović D., King T., Reynolds A., Tinelli C. CVC4 // Lecture Notes in Computer Science. 2011. Volume 6806. pp. 171–177.
25. Bertot Y. A Short Presentation of Coq // Lecture Notes in Computer Science. 2008. Volume 5170. pp. 12–16.
26. Bjørner N., Nachmanson L. Navigating the Universe of Z3 Theory Solvers // Lecture Notes in Computer Science. 2020. Volume 12475. pp. 8–24.
27. Blaauwbroek L., Urban J., Geuvers H. The Tactician // Lecture Notes in Computer Science. 2020. Volume 12236. pp. 271–277.
28. Blazy S., Dargaye Z., Leroy X. Formal Verification of a C Compiler Front-End // Lecture Notes in Computer Science. 2006. Volume 4085. pp. 460–475.
29. Blazy S., Leroy X. Mechanized Semantics for the Clight Subset of the C Language // Journal of Automated Reasoning. 2009. Volume 43. Issue 3. pp. 263–288.
30. Bobot F., Filliâtre J.-C., Marché C., Paskevich A. Let's verify this with Why3 // International Journal on Software Tools for Technology Transfer. 2015. Volume 17. Issue 6. pp. 709–727.
31. Brain M., Polgreen E. A Pyramid Of (Formal) Software Verification // Lecture Notes in Computer Science. 2025. Volume 14934. pp. 393–419.
32. Cao Q., Beringer L., Gruetter S., Dodds J., Appel A.W. VST-Floyd: A Separation Logic Tool to Verify Correctness of C Programs // Journal of Automated Reasoning. 2018. Volume 61. Issue 1. pp. 367–422.
33. Clochard M., Gondelman L., Pereira M. The Matrix Reproved (Verification Pearl) // Journal of Automated Reasoning. 2018. Volume 60. Issue 3. pp. 365–383.
34. Conchon S., Iguernelala M., Mebsout A. A collaborative framework for non-linear integer arithmetic reasoning in Alt-Ergo // Proceedings of the 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. Timișoara, Romania, September 23–26, 2013. IEEE,

2013. pp. 161–168.

35. Correnson L. Qed. Computing What Remains to Be Proved // Lecture Notes in Computer Science. 2014. Volume 8430. pp. 215–229.
36. Cuoq P., Monate B., Pacalet A., Prevosto V. Functional dependencies of C functions via weakest pre-conditions // International Journal on Software Tools for Technology Transfer. 2011. Volume 13. Issue 5. pp. 405–417.
37. Czajka Ł., Kaliszyk C. Hammer for Coq: Automation for Dependent Type Theory // Journal of Automated Reasoning. 2018. Volume 61. Issue 1. pp. 423–453.
38. de Moura L., Bjørner N. Z3: An Efficient SMT Solver // Lecture Notes in Computer Science. 2008. Volume 4963. pp. 337–340.
39. Dénès M., Mörtberg A., Siles V. A Refinement-Based Approach to Computational Algebra in Coq // Lecture Notes in Computer Science. 2012. Volume 7406. pp. 83–98.
40. Dijkstra E.W. Guarded Commands, Nondeterminacy and Formal Derivation of Programs // Communications of the ACM. 1975. Volume 18. Issue 8. pp. 453–457.
41. Dijkstra E.W., Schölten C.S. The strongest postcondition // Predicate Calculus and Program Semantics. New York: Springer, 1990. pp 209–215.
42. Efremov D., Mandrykin M., Khoroshilov A. Deductive verification of unmodified Linux kernel library functions // Lecture Notes in Computer Science. 2018. Volume 11245. pp. 216–234.
43. Filliâtre J.C. Deductive software verification // International Journal on Software Tools for Technology Transfer. 2011. Volume 13. Issue 5. Article ID: 397.
44. Filliâtre J.-C., Paskevich A. Why3 — Where Programs Meet Provers // Lecture Notes in Computer Science. Volume 7792. pp. 125–128.
45. Flanagan C., Saxe J.B. Avoiding exponential explosion: Generating compact verification conditions // Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '01). London, United Kingdom, January 17–19, 2001. New York: Association for Computing Machinery, 2001. pp. 193–205.
46. Floyd R.W. Assigning meanings to programs // Proc. Symposia in Applied Mathematics. Providence, 1967. Volume 19. pp. 19–32.
47. Furia C.A., Meyer B. Inferring Loop Invariants Using Postconditions // Lecture Notes in Computer Science. 2010. Volume 6300. pp. 277–300.
48. Grigore R., Charles J., Fairmichael F., Kiniry J. Strongest postcondition of unstructured programs // Proceedings of the 11th International Workshop on Formal Techniques for Java-like Programs (FTfJP '09). Genova, Italy, July 6, 2009. New York: Association for Computing Machinery, 2009. pp. 6:1–6:7.
49. Hähnle R., Huisman M. Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools // Lecture Notes in Computer Science. 2019. Volume 10000. pp. 345–373.
50. Harrison J. Theorem Proving with the Real Numbers. London: Springer, 1998. – 186 p.
51. Hoare C.A.R. An axiomatic basis for computer programming // Communications of the ACM. 1969. Volume 12. Issue 10. pp. 576–580.

52. Ishtiaq S.S., O'Hearn P.W. BI as an assertion language for mutable data structures // Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '01). London, United Kingdom, January 17–19, 2001. New York: Association for Computing Machinery, 2001. pp. 14–26.
53. Islam R., Ahmed I. Gemini-the most powerful LLM: Myth or Truth // Proceedings of the 2024 5th Information Communication Technologies Conference (ICTC). Nanjing, China, May 10–12, 2024. IEEE, 2024. pp. 303–308.
54. Kaufmann M., Moore J.S. An industrial strength theorem prover for a logic based on Common Lisp // IEEE Transactions on Software Engineering. 1997. Volume 23. Issue 4. pp. 203–213.
55. Knothe D., Bringmann O. Combining Small-Step and Big-Step Semantics to Verify Loop Optimizations // arXiv preprint arXiv:2602.19868. 2026. Access mode: `https://arxiv.org/abs/2602.19868`.
56. Kondratyev D.A., Maryasov I.V., Nepomniaschy V.A. The Automation of C Program Verification by the Symbolic Method of Loop Invariant Elimination // Automatic Control and Computer Sciences. 2019. Volume 53. Issue 7. pp. 653–662.
57. Kondratyev D., Maryasov I., Nepomniaschy V. Towards Automatic Deductive Verification of C Programs over Linear Arrays // Lecture Notes in Computer Science. 2019. Volume 11964. pp. 232–242.
58. Kondratyev D.A., Nepomniaschy V.A. Automation of C Program Deductive Verification without Using Loop Invariants // Programming and Computer Software. 2022. Volume 48. Issue 5. pp. 331–346.
59. Kondratyev D.A., Promsky A.V. Developing a self-applicable verification system. Theory and practice // Automatic Control and Computer Sciences. 2015. Volume 49. Issue 7. pp. 445–452.
60. Kosmatov N., Marché C., Moy Y., Signoles J. Static versus Dynamic Verification in Why3, Frama-C and SPARK 2014 // Lecture Notes in Computer Science. 2016. Volume 9952. pp. 461–478.
61. Kovács L., Voronkov A. // Lecture Notes in Computer Science. 2013. Volume 8044. pp. 1–35.
62. Leino K.R.M. Efficient weakest preconditions // Information Processing Letters. 2005. Volume 93. Issue 6. pp. 281–288.
63. Leroy X. A formally verified compiler back-end // Journal of Automated Reasoning. 2009. Volume 43. Issue 4. pp. 363–446.
64. Leroy X. Formal verification of a realistic compiler // Communications of the ACM. 2009. Volume 52. Issue 7. pp. 107–115.
65. Mandrykin M.U., Khoroshilov A.V. High-level memory model with low-level pointer cast support for Jessie intermediate language // Programming and Computer Software. 2015. Volume 41. Issue 4. pp. 197–207.
66. Mandrykin M.U., Khoroshilov A.V. Region analysis for deductive verification of C programs // Programming and Computer Software. 2016. Volume 42. Issue 5. pp. 257–278.
67. Mandrykin M.U., Khoroshilov A.V. Towards deductive verification of C programs with shared data // Programming and Computer Software. 2016. Volume 42. Issue 5. pp. 324–332.

68. Maryasov I.V., Nepomniaschy V.A., Promsky A.V., Kondratyev D.A. Automatic C Program Verification Based on Mixed Axiomatic Semantics // Automatic Control and Computer Sciences. 2014. Volume 48. Issue 7. pp. 407–414.
69. Moore J.S. Milestones from the Pure Lisp theorem prover to ACL2 // Formal Aspects of Computing. 2019. Volume 31. Issue 6. pp. 699–732.
70. Nepomniaschy V.A., Anureev I.S., Mikhailov I.N., Promsky A.V. Towards verification of C programs. C-light language and its formal semantics // Programming and Computer Software. 2002. Volume 28. Issue 6. pp. 314–323.
71. Nepomniaschy V.A., Anureev I.S., Promskii A.V. Towards Verification of C Programs: Axiomatic Semantics of the C-kernel Languages // Programming and Computer Software. 2003. Volume 29. Issue 6. pp. 338–350.
72. O'Hearn P. Separation logic // Communications of the ACM. 2019. Volume 62. Issue 2. pp. 86–95.
73. O'Hearn P. Separation Logic Tutorial // Lecture Notes in Computer Science. 2008. Volume 5366. pp. 15–21.
74. Palomo-Lozano F., Medina-Bulo I., Alonso-Jiménez J. Certification of matrix multiplication algorithms. Strassen's algorithm in ACL2 // Supplemental Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics. Edinburgh, United Kingdom, September 3–6, 2001. pp. 283–298.
75. Paraskevopoulou Z., HriŢcu C., Dénès M., Lampropoulos L., Pierce B.C. Foundational Property-Based Testing // Lecture Notes in Computer Science. 2015. Volume 9236. pp. 325–343.
76. Paulin-Mohring C. Introduction to the Coq Proof-Assistant for Practical Software Verification // Lecture Notes in Computer Science. 2012. Volume 7682. pp. 45–95.
77. Reynolds J.C. An Overview of Separation Logic // Lecture Notes in Computer Science. 2008. Volume 4171. pp. 460–469.
78. Reynolds J.C. Separation logic: a logic for shared mutable data structures // Proceedings 17th Annual IEEE Symposium on Logic in Computer Science. Copenhagen, Denmark, July 22–25, 2002. IEEE, 2002. pp. 55–74.
79. Srivastava S., Gulwani S., Foster J.S. From program verification to program synthesis // Proceedings of 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. Madrid, Spain, January 17–23, 2010. New York: Association for Computing Machinery, 2010. pp. 313–326.
80. Staroletov S., Kondratyev D., Shelekhov V., Kogtenkov A., Shilov N.V., Garanina N., Shoshmina I., Cherganov T., Dyadov V. VeHa: A Hybrid National Verification Hackathon for Better Formal Methods Education // Lecture Notes in Computer Science. 2026. Volume 16566. pp. 127–146.
81. Schulz S. E – a brainiac theorem prover // AI Communications. 2002. Volume 15. Issue 2-3. pp. 111–126.
82. Smith T.M. Theory and Practice of Classical Matrix-Matrix Multiplication for Hierarchical Memory Architectures: thes... doct. phylosophy (computer sci.). – Austin, 2017. – 136 p.
83. Strassen V. Gaussian elimination is not optimal // Numerische Mathematik. Volume 13. Issue 4.

pp. 354–356.

84. Sulatycke P.D., Ghose K. Caching-Efficient Multithreaded Fast Multiplication of Sparse Matrices // Proceedings of the 12th International Parallel Processing Symposium / 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP '98). Orlando, USA. March 30 – April 3, 1998. IEEE, 1998. pp.117–123.
85. Turner O., Chakraborty S. VLIM: Verified Loop Interchange for Optimised Matrix Multiplication // Proceedings of the 29th Design, Automation and Test in Europe Conference DATE 2026. Verona, Italy, April 20 – April 22, 2026.
86. Vlasov A.A., Lylova S.S., Latkin E.I., Veretennikov A.A. minigemm. URL: `https://gitflic.ru/project/nsu/minigemm` (Accessed 20 May 2025)
87. Volkov G., Mandrykin M. Efremov D. Lemma functions for Frama-C: C programs as proofs // Proceedings of the 2018 Ivannikov Ispras Open Conference (ISPRAS). Moscow, Russia, November 22–23, 2018. IEEE, 2018. pp. 31–38.

A. Условие корректности и его доказательство, где применяются все введенные нами стратегии доказательства

Рассмотрим одно из условий корректности реализации умножения матриц над математическими вещественными числами с оптимизациями в виде изменения порядка вложенности циклов для построчного обхода результирующей матрицы и умножаемой матрицы, а также доказательство такого условия корректности:

```
intros.
 assert (EqA: forall idx, (0 <= idx)%Z ->
 (idx < i * i1)%Z -> a13 (shift a idx) = a9 (shift a idx)).
  {
    intros idx Hidx_min Hidx_max. unfold a13, a11, Map.set.
    destruct (why_decidable_eq (shift a2 (i7 + x3)%Z)
    (shift a idx)) as [Heq | Hneq].
    - (* Раскрываем базу И СМЕЩЕНИЕ (HeqOff)! *)
      unfold shift, base, offset, a4, a5, x1, x2 in *.
      injection Heq as HeqBase HeqOff.
      (* Разбираем гипотезу H25 (A и C разделены) *)
      destruct H25 as[Hb1|[Hb2|[Hb3|[Hb4|Hb5]]]].
      * nia.
```

```
      * nia.
      * unfold base, shift in Hb3.
        unfold base in Hb3.
        congruence.
      * (* Подсказка: конец изменяемой строки C
      не выходит за пределы матрицы C *)
        assert (Step: (i7 + i2 * i3 < i1 * i2)%Z) by nia.
         destruct a2 as[b_a2 o_a2]. destruct a as [b_a o_a].
        unfold shift, offset, base in *. simpl in *.
      nia.

      *  destruct a2 as [b_a2 o_a2]. destruct a as [b_a o_a].
        unfold shift, offset, base in *. simpl in *.
        nia.
    - reflexivity.
 }
assert (EqB: forall idx, (0 <= idx)%Z ->
(idx < i * i2)%Z -> a13 (shift a1 idx) = a9 (shift a1 idx)).
 {
   intros idx Hidx_min Hidx_max.
   unfold a13, a11, Map.set.
   destruct (why_decidable_eq (shift a2 (i7 + x3)%Z)
   (shift a1 idx)) as[Heq | Hneq].
   -
     unfold shift, base, offset, a3, a5, x, x2, x3 in *.
     injection Heq as HeqBase HeqOff.

     (* Разбираем гипотезу H24 (B и C разделены) *)
     destruct H24 as [Hb1|[Hb2|[Hb3|[Hb4|Hb5]]]].
     * nia.
     * nia.
           * assert (Step: (i7 + i2 * i3 < i1 * i2)%Z) by nia.
```

```
        destruct a2 as[b_a2 o_a2]. destruct a1 as [b_a o_a].
        unfold shift, offset, base, x3 in *. simpl in *. nia.
    * (* Ветка 4: C ДО B *)
      assert (Step: (i7 + i2 * i3 < i1 * i2)%Z) by nia.
      destruct a2 as [b_a2 o_a2]. destruct a1 as [b_a1 o_a1].
      simpl in *.
      (* Теперь nia видит Hidx_min (0 <= idx), HeqOff и Step.
      Это тоже успешное доказательство! *)
      nia.
    * (* Ветка 5: B ДО C *)
      destruct a2 as [b_a2 o_a2]. destruct a1 as [b_a1 o_a1].
      simpl in *.
      (* Теперь nia видит Hidx_max (idx < i * i2) и HeqOff.
      Это тоже успешное доказательство! *)
      nia.
  - reflexivity.
}
assert (Hframe_dot: forall k_dot col res_,
    (k_dot <= i)%Z -> (col < i2)%Z -> (0 <= col)%Z ->
    P_DotProduction a9 a a1 k_dot i3 i col i2 res_ ->
    P_DotProduction a13 a a1 k_dot i3 i col i2 res_).
{
  intros k_dot col res_ Hk_max Hcol_min Hcol_max Hdot_k.

  (* Делаем индукцию. Переменная 'to' будет означать текущую длину *)
  induction Hdot_k as [to_empty Hempty | to_pos Hpos Hdot_prev IH].
  - apply Q_dotproduction_empty_range. lia.
  - (* Заменяем новую память на старую.
       nia видит Hcol_max (col < i2) и Hk_max (to_pos <= i) и
       легко докажет границы! *)
    rewrite <- EqA; [ | lia | nia ].
    rewrite <- EqB.
```

```
      apply Q_dotproduction_positive_range.
      + lia.
      + apply IHHdot_k; try assumption. nia.
      + nia. +nia.
  }

  assert (H_cases: (i4 < i7 \/ i4 = i7)%Z) by lia.
  destruct H_cases as [H_lt | H_eq].
-
  assert (EqC: a13 (shift a2 (i4 + x3)%Z) = a9 (shift a2 (i4 + x3)%Z)).
    {
      unfold a13, a11, Map.set.
      destruct (why_decidable_eq (shift a2 (i7 + x3)%Z)
      (shift a2 (i4 + x3)%Z)) as[Heq | Hneq].
      * unfold shift, offset in Heq. injection Heq.
        lia. (* nia понимает, что i7 не может быть равно i4 *)
      * reflexivity.
    }
rewrite EqC.
apply Hframe_dot.
lia. nia. nia.
apply H36.
nia. nia.
-rewrite H_eq.
 assert (EqC_new: a13 (shift a2 (i7 + x3)%Z) = a12).
    {
      unfold a13, a11, Map.set.
      destruct (why_decidable_eq (shift a2 (i7 + x3)%Z)
      (shift a2 (i7 + x3)%Z)) as[Heq | Hneq].
      * reflexivity.
      * congruence. (* Абсурд, адрес всегда равен самому себе *)
```

```
    }
rewrite EqC_new.
replace i7 with i4 by H_eq.
apply Hframe_dot.
lia.
rewrite H_eq. nia. nia.
replace i4 with i7 by H_eq.
exact H32.
```

В данном доказательстве применяются все заданные нами стратегии доказательства условий корректности.