NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2002; **PRISM'01**:1–16 *Prepared using* nlaauth.cls *[Version: 2000/03/22 v1.0]*

Application of The Three-Way Decomposition for Matrix Compression

Ilghiz Ibraghimov

Universität des Saarlandes, FR 6.1 Mathematik,
D-66041, Saarbrücken, Germany,
ilgis@num.uni-sb.de

SUMMARY

We present a new method to compress and invert 3D integral operators on rectangular non-regular grids. This method requires a small amount of memory to store the compressed matrix and in most cases can provide a good preconditioner for the solution of linear systems with this matrix. We demonstrate efficiency of this method for the solution of some model discrete problems associated with

$$\int_{\mathbb{R}^3} A(\bar{x},\bar{y})f(\bar{x})d\bar{x} = u(\bar{y}), \qquad \bar{x},\bar{y} \in \mathbb{R}^3$$

where $A(\bar{x},\bar{y})$ such as $\dfrac{1}{|\bar{x}-\bar{y}|}$ is considered on a non-regular grid. The arithmetical complexity of matrix-vector and preconditioner-vector multiplications are about $N^{4/3}$ operations and there are only about $N^{2/3}$ words of memory to store.

KEY WORDS: Preconditioning, integral operators, three-way decomposition, Kronecker product

1. INTRODUCTION

We discuss the approximation by small-rank structures of matrices generated by integral operators on rectangular non-regular grids. The grid is a parallelepiped with different lengths of edges as shown in Fig. 1. This problem is important because the integral operator generates dense matrices — if we have a 3D problem on the $N = n \times n \times n$ mesh, the matrix elements need n^6 words of memory. If the integral operator depends only on the distance such as

$$\frac{1}{|\bar{x}-\bar{y}|},$$

and the regular grid is used, then it is possible to use a Toeplitz structure of this matrix and store this matrix in n^3 words of memory and multiply it for $n^3 \log_2 n$ arithmetical operations.

Nowadays there are several well known approaches working with different types of integral operators: the multipole method of Rokhlin [1], the mosaic-skeleton approximation method of

Received 5 Oct 2001
Revised 25 Feb 2002

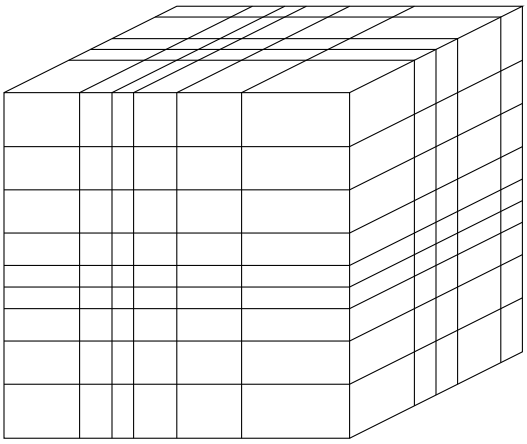

Figure 1. An example of a rectangular non-regular grid.

Tyrtyshnikov [2], the panel-clustering method [3, 4] and $\mathcal{H}$-matrix approach [5, 6] of Hackbusch and some other methods [7, 8]. All these methods are based on the idea of splitting discrete matrices generated by the integral operator to the set of blocks with a small rank, then each block is decomposed to factors by SVD-like approximation or by special type of functions (in the multipole method). If the rank of these blocks is small enough, SVD factors require a smaller amount of memory to store the matrix than the original block. In addition, it is possible to reduce the number of arithmetical operations required for one matrix-vector multiplication.

In this work we present a new idea for splitting the initial matrix. Suppose the initial matrix is presented as follows:

$$A(\bar{x},\bar{y}) = A(x_1,x_2,x_3,y_1,y_2,y_3) \in \mathbb{R}^{n_1n_2n_3\times n_1n_2n_3},$$
$$\bar{x} = (x_1,x_2,x_3), \qquad \bar{y} = (y_1,y_2,y_3) \in \mathbb{R}^{n_1n_2n_3}.$$

We decompose this matrix in the following form:

$$A(\bar{x},\bar{y}) \simeq \sum_{l=1}^{r} b_l(x_1,y_1)c_l(x_2,y_2)d_l(x_3,y_3), \tag{1}$$

then we need to store only $3rn^2$ words of memory instead of n^6. If r is small enough, it gives us an extremely high compression rate.

Here and later we use $n = n_1 = n_2 = n_3$ in some cases to demonstrate the general behavior of arithmetical complexity.

In this article we discuss the main properties of this decomposition:

- how to create this decomposition if we know all matrix elements or only parts of them (if we need all matrix elements stored at the same time, then this method requires n^6 words of memory which is not advisable);
- how to make matrix-vector and preconditioner-vector multiplications.

2. KRONECKER PRODUCT DECOMPOSITION

We need a well known Kronecker product technique to understand how to perform the decomposition (1). If $B \in \mathbb{R}^{m_1 \times n_1}$ and $C \in \mathbb{R}^{m_2 \times n_2}$, then their **Kronecker product** $B \otimes C$ is an $m_1 \times n_1$ block matrix whose (i, j) block is the $m_2 \times n_2$ matrix $b_{ij}C$. The basic properties of the Kronecker product are well known:

$$\begin{aligned} (B \otimes C)^* &= B^* \otimes C^*, \\ (B \otimes C)^{-1} &= B^{-1} \otimes C^{-1}, \\ (B \otimes C)(D \otimes F) &= BD \otimes CF, \\ B \otimes (C \otimes D) &= (B \otimes C) \otimes D \end{aligned}$$

In 1992 Van Loan and Pitsianis [9] showed how to solve

$$\min_{B,C} ||A - B \otimes C||_F,$$

where $A \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$, $B \in \mathbb{R}^{n_1 \times n_1}$, $C \in \mathbb{R}^{n_2 \times n_2}$. This problem then becomes the following problem:

$$\min_{B,C} ||P(A) - \mathrm{vec}(B)\mathrm{vec}(C)^*||_F,$$

where P is a matrix operator which permutates entries of A, and $\mathrm{vec}(B) \in \mathbb{R}^{n_1^2}$, $\mathrm{vec}(C) \in \mathbb{R}^{n_2^2}$ are vectors containing all entries of B and C respectively, so it is possible to approximate A by

$$\sum_{l=1}^{r} B_l \otimes C_l,$$

and if good accuracy is achieved for the sum of r Kronecker products of matrices, then the compression factor is equal to

$$\frac{n_1^2 n_2^2}{r(n_1^2 + n_2^2)}.$$

If we approximate our matrix by one Kronecker product matrix, we can easily invert it within n^3 arithmetical operations using the basic properties of the Kronecker product. For the class of pairs of Kronecker products we can use the following idea:

$$(B_1 \otimes C_1 + B_2 \otimes C_2)(X_1 \otimes X_2) = [B_2 X_1 (D_1 + I)] \otimes [C_2 X_2 (D_2 + I)],$$

$$B_1 X_1 = B_2 X_1 D_1, \qquad C_1 X_2 = C_2 X_2 D_2,$$

then we should first solve a generalized eigenvalue problem twice and invert X_1, X_2, $B_2 X_1$ and $C_2 X_2$. Again, this takes about n^3 arithmetical operations.

In this article we generalize this approach for the 3-dimensional case. We are searching for the decomposition of the initial array $A(\bar{x}, \bar{y}) \in \mathbb{R}^{n_1 n_2 n_3 \times n_1 n_2 n_3}$ as follows:

$$\min_{B,C,D} ||A - \sum_{l=1}^{r} B_l \otimes C_l \otimes D_l||_F,$$

Let us join all pairs of indeces of (x_1, y_1), (x_2, y_2), (x_3, y_3) to new 3 indeces i, j, k, then this problem assumes the following form:

$$\min_{B,C,D} \sum_{i,j,k} \left| a_{ijk} - \sum_{l=1}^{r} b_{il} c_{jl} d_{kl} \right|^2. \tag{2}$$

Actually, it is a three-way decomposition problem [10, 11] and it is well known in statistics. In the next chapter we will discuss how to find this decomposition for two cases:

- All entries of A are given; r is given or should be as small as possible. This problem we call the **dense** three-way decomposition.
- We have *a priori* information that r is small enough; we can compute any entry of A, but we do not want to compute and store all of them because of the degree of arithmetical complexity and memory requirements. This problem we call the **sparse** three-way decomposition.

Suppose we can compute factors B, C, D in (2). Can we multiply fast A to the vector $t(y_1, y_2, y_3) \in \mathbb{R}^{n_1 n_2 n_3}$? The answer is shown in the following formula:

$$\begin{aligned} At = \\ \sum_{y_1,y_2,y_3} A(x_1, x_2, x_3, y_1, y_2, y_3) t(y_1, y_2, y_3) = \\ \sum_{l=1}^{r} \sum_{y_1,y_2,y_3} b_l(x_1, y_1) c_l(x_2, y_2) d_l(x_3, y_3) t(y_1, y_2, y_3) = \\ \sum_{l=1}^{r} b_l(x_1, y_1) \sum_{y_1} p_l(y_1, x_2, x_3), \\ p_l(y_1, x_2, x_3) = \sum_{y_2} c_l(x_2, y_2) q_l(y_1, y_2, x_3), \\ q_l(y_1, y_2, x_3) = \sum_{y_3} d_l(x_3, y_3) t(y_1, y_2, y_3). \end{aligned} \tag{3}$$

Hence, the arithmetical complexity of this matrix-vector multiplication is $rn_1n_2n_3(n_1+n_2+n_3)$ operations.

Let us generalize an inverse 3D Kronecker product. We are searching for the following exact decomposition:

$$\forall l = 1, \dots R :$$

$$B_l \otimes C_l \otimes D_l = (X_1 \otimes X_2 \otimes X_3)(Z_{1,l} \otimes Z_{2,l} \otimes Z_{3,l})(Y_1 \otimes Y_2 \otimes Y_3), \tag{4}$$

If $Z_{1,l}$ are diagonal, we can easily invert this matrix with only n^3 arithmetical operations. This decomposition can be found the following way:

$$\begin{aligned} B_l &\simeq X_1 Z_{1,l} Y_1, \\ C_l &\simeq X_2 Z_{2,l} Y_2, \\ D_l &\simeq X_3 Z_{3,l} Y_3. \end{aligned}$$

Here the sizes of matrices B_l, C_l, D_l are small enough and there is no reason to discuss the sparse three-way decomposition.

The solution of the linear system with this preconditioner is based on the idea of multiplying the compressed matrix by the vector shown in (3). The multiplication to this preconditioner requires only $n_1 n_2 n_3 (1 + 2(n_1 + n_2 + n_3))$ arithmetical operations, but the computation of X_1, X_2, X_3, Y_1, Y_2, Y_3 requires 3 times the three-way decomposition and 6 times the computation of the inverse matrices. Here we should remark that the matrices X and Y should be well conditioned and we need the three-way decomposition to be restricted to the condition number of the matrix.

Thus we have demonstrated the techniques which we are going to use to create the compressed matrix form and the preconditioner. Now we need to discuss how to create the three-way decomposition.

3. DENSE THREE-WAY DECOMPOSITION

In this chapter we will make a short overview of the methods and properties of the dense three-way decomposition. Suppose the three-way array is

$$\begin{array}{lll} a_{ijk}, & i = 1, \ldots, m_1, & m_1 = n_1^2, \\ & j = 1, \ldots, m_2, & m_2 = n_2^2, \\ & k = 1, \ldots, m_3, & m_3 = n_3^2. \end{array}$$

Suppose a **triad** is $a_{ijk} = \alpha b_i c_j d_k$. Let

$$\begin{array}{lll} B = \{b_{il}\} & i = 1, \ldots, m_1 & \\ C = \{c_{jl}\} & j = 1, \ldots, m_2 & l = 1, \ldots, r \\ D = \{d_{kl}\} & k = 1, \ldots, m_3 & \end{array}$$

$$E = \operatorname{diag}(\alpha_1, \ldots, \alpha_r),$$

$$||b_l||_2 = ||c_l||_2 = ||d_l||_2 = 1,$$

then

$$A = [B, C, D, E],$$

i.e.

$$a_{ijk} = \sum_{l=1}^{r} \alpha_l b_{il} c_{jl} d_{kl}.$$

In 1977 Kruskal [12] proved the following theorem:

Theorem 1. *Suppose every I_0 columns of B are independent, every J_0 columns of C are independent and every K_0 columns of C are independent. Suppose*

$$I_0 + J_0 + K_0 \geq 2r + 2.$$

Then

$$[B, C, D, E] = [BP, CP, DP, E].$$

where P is only a matrix of transposition.

This result is contrary to 2D decomposition — SVD:

$$BC^* = (BT)(T^{-1}C^*),$$

where T is any nonsingular matrix.

At present there are several approaches for computation of the three-way decomposition.

3.1. Parallel Factor method

Harshman and Lundy [13] suggested a monotonically converging algorithm called Parallel Factors for minimizing (2). Let r be definite, freeze any two of B, C, D, and the functional is linear in the third. For example, if we temporarily fix the matrices C and D, then we seek $\tilde{B}$ so

$$\min_{\tilde{B}} ||\tilde{A} - \tilde{B}Q^*||_F^2$$

is minimized. Here $\tilde{A} \in \mathbb{R}^{m_1 \times m_2 m_3}$ is constructed from the original 3D array A joining the second and the third dimensions, $Q = (c_1 \otimes d_1, \ldots, c_r \otimes d_r) \in \mathbb{R}^{m_2 m_3 \times r}$. The new B and E are then defined by $\tilde{B} = BE$, where B's columns have unit l_2 norms. This is a standard least squares problem, which can be simplified as follows:

$$\tilde{B}^* = \left[(C^*C) \odot (D^*D)\right]^{-1} H, \qquad h_{li} = \sum_{jk} a_{ijk} c_{jl} d_{kl}, \tag{5}$$

where $\odot$ means the element-wise product of matrices.

3.2. Regularization

The Parallel Factor approach has poor convergence properties. Sometimes it runs in local minima [14] and/or it needs an extremely large amount of iterations. Usually it happens when two $|\alpha_l|$ become very large, and when corresponding triads are almost collinear to each others but have different signs.

In this work we suggest a new approach to improve the convergence. We perform a Tikhonov's regularization [15] and instead of problem (2) we solve the following problem:

$$\min_{B,C,D,\alpha} \sum_{i,j,k} \left| a_{ijk} - \sum_{l=1}^{r} \alpha_l b_{il} c_{jl} d_{kl} \right|^2 + \beta \sum_{l=1}^{r} \alpha_l^2.$$

where $\beta > 0$ is Tikhonov's parameter of the regularization. Hence, the computation of each step (see (5)) will follow the formula:

$$\tilde{B}^* = \left[(C^*C) \odot (D^*D) + \beta I\right]^{-1} H.$$

In this method $[(C^*C) \odot (D^*D) + \beta I]$ is far from the singular matrix, then it prevents the deadlocks with large $|\alpha_l|$ during convergence. This method can considerably reduce the total amount of iterations. Control of β is the subject of a different paper, but it should be large enough at the beginning and decrease during iterations.

Problem (4) needs X and Y to be well conditioned. In our current notations there are C and D, so we will consider (2) with a penalty function being definite as the sum of $\mathrm{cond}_F(C)$ and $\mathrm{cond}_F(D)$:

$$\min_{B,C,D,\alpha} \sum_{i,j,k} \left| a_{ijk} - \sum_{l=1}^{r} \alpha_l b_{il} c_{jl} d_{kl} \right|^2 + \gamma \left(\mathrm{cond}_F(C) + \mathrm{cond}_F(D)\right). \tag{6}$$

The minimization procedure for this functional is nontrivial. We suggest a new approach for it: express D and α analytically and minimize them by only two matrices:

$$\min_{C,D} ||\tilde{A}Z||_F^2 + \gamma \left(\mathrm{cond}_F(C) + \mathrm{cond}_F(D)\right).$$

where $Z \in \mathbb{R}^{r \times m_2 m_3}$ contains an orthonormal subspace of the matrix Q. Now we can compute the minimum using quasi Newton methods, but we need to compute the gradient for these methods. It is possible to apply the Baur-Strassen algorithm for analytical computation of the gradient [23], but it will take r times more memory than during computation of the function.

Recently the computation of gradients for similar kinds of functionals was discussed in [25] and it was shown that it is possible to compute the gradient by all entries of B and C analytically and that the arithmetical complexity and memory requirements will be only 3 times bigger than the computation of the function.

3.3. Tucker approach

If r is much smaller than m_1, m_2 or m_3, then, for given r_1, r_2 and r_3:

$$\begin{array}{llll} B = U_B B', & U_B \in \mathbb{R}^{m_1 \times r_1}, & B' \in \mathbb{R}^{r_1 \times r}, & r \le r_1 \le m_1, \\ C = U_C C', & U_C \in \mathbb{R}^{m_2 \times r_2}, & C' \in \mathbb{R}^{r_2 \times r}, & r \le r_2 \le m_2, \\ D = U_D D', & U_D \in \mathbb{R}^{m_3 \times r_3}, & D' \in \mathbb{R}^{r_3 \times r}, & r \le r_3 \le m_3, \end{array}$$

where U_B, U_C, U_D are unitary matrices. It is possible to compute B', C', D' and U_B, U_C, U_D independently. This idea was originally introduced by Tucker [16] and developed by Kronenberg and de Leeuw [17]. Consider that (2) decomposes to the following two problems:

$$\min_{G,U_B,U_C,U_D} \sum_{i,j,k} \left| a_{ijk} - \sum_{i'}^{r_1} \sum_{j'}^{r_2} \sum_{k'}^{r_3} g_{i'j'k'} U_{B_{ii'}} U_{C_{jj'}} U_{D_{kk'}} \right|^2. \tag{7}$$

$$\min_{B',C',D'} \sum_{i',j',k'} \left| g_{i'j'k'} - \sum_{l=1}^{r} \alpha_l b_{i'l} c_{j'l} d_{k'l} \right|^2, \tag{8}$$

If (7) is exactly zero, then the solution of (8) and (7) gives the same result as (2), otherwise it could be a good approximation to the solution [18]. Here the array $g_{i'j'k'}$ is called a core array. The problem (8) is equal to (2) but contains smaller dimensions.

The algorithm that computes the core is similar to Parallel Factors: freeze any two of U_B, U_C, U_D, and the functional is linear in the third. For example, if we temporarily fix U_C and U_D matrices, then we seek $\tilde{U}_B$ and, as U_C and U_D are orthonormal, it leads to

$$\min_{G,U_B} \sum_{ij'k'} \left(\sum_{jk} a_{ijk} U_{C_{jj'}} U_{D_{kk'}} - \sum_{i'} U_{B_{ii'}} g_{i'j'k'} \right)^2 .$$

It is easy to see that this problem is again a least squares problem, where U_B contains left singular vectors of the matrix

$$W_{i,\{j'k'\}} = \sum_{jk} a_{ijk} U_{C_{jj'}} U_{D_{kk'}} ,$$

and the core can be obtained from its right singular vectors. In the Tucker's algorithm r_1, r_2 and r_3 should be given. We do not need the regularization because U_B, U_C, U_D are unitary matrices. The main advantage of this algorithm is better convergence: during computations of the core it is usually better than in Parallel Factors because B', C' and D' can be chosen as orthonormal.

Statement 1. *If the initial data have no noise at all, then U_B, U_C, U_D can be computed as the left singular vectors of matrices created from A by joining (2nd and 3rd), (3rd and 1st), (1st and 2nd) dimensions respectively.*

This statement is evident if we remark that in the case of no noise the problem (7) is exactly zero. In the general case, Tucker's algorithm still has only monotonical convergence. One iteration of the Parallel Factors algorithm depends linearly on the size of the initial problem. If $r_1 r_2 r_3$ is reasonably smaller than $m_1 m_2 m_3$, then the three-way decomposition with Tucker's approach works much faster [18].

A very important problem in nonlinear minimization is to compute an initial approximation. This subject was discussed by Leurgans, Ross and Abel [19]. The most popular initial approaches are based either on one iteration of Tucker's algorithm or on the generalized eigenvalue decomposition of A_1 and A_2 [24].

3.4. Parallel Decomposition method

Recently, the author suggested a new approach [20] to solve the three-way decomposition in the case where R is given and any pair out of B, C and D have full column rank. If A has an exact decomposition then this algorithm has a linear convergence. This approach can be written with the help of the following two statements [20]:

Statement 2. *If an exact solution of the three-way decomposition of $\{a_{ijk}\}$ with size $m_1 \times m_2 \times m_3$ contains at least two matrices with full column rank r, then it is possible to transform this problem to another three-way decomposition $\{x_{ijk}\}$ with size $r \times r \times m_3$ and the corresponding factors of the solution of new problem will be quadratic nonsingular matrices.*

Statement 3. *Suppose $\{a_{ijk}\}$ with the size $r \times r \times m_3$ has an exact three-way decomposition with two quadratic nonsingular factors B and C. Then there is an algorithm to compute independently at least one triad.*

The main idea is to transform the problem (2) to

$$\min_{B,T,D} \sum_{i,l,k} \left| \sum_j a_{ijk} t_{jl} \alpha_l^{-1} - b_{il} d_{kl} \right|^2, \quad \text{and} \quad \min_{S,C,D} \sum_{l,j,k} \left| \sum_i a_{ijk} s_{il} \alpha_l^{-1} - c_{jl} d_{kl} \right|^2,$$

where $S = B^{-*}$, $T = C^{-*}$. Then we have to solve r optimization problems with $2r$ unknowns instead of one minimization problem with r^2 unknowns and several local minima. These optimization problems are simple, because if we compute at least one local minima, then we find a corresponding triad and can subtract it from the initial data using **Statement 2** and finally repeating the decomposition. Another advantage of this method is the implicit computations of B^{-*} and C^{-*} which are needed for the preconditioner (see (4)).

The computational complexity is $r^4 \log_2 r$ arithmetical operations with a big constant and r^5 with a small constant. Since the Kronecker rank of the initial problem is small enough, the great computational complexity only affects the creation of the preconditioner.

4. SPARSE THREE-WAY DECOMPOSITION

Our main goal is to compress large $n^3 \times n^3$ matrices and avoid, if possible, the storage and computation of all n^6 matrix elements. Since the rank of the three-way decomposition is small enough, we should define only $3rn^2$ unknowns. Nowadays, there are some approaches described in the review [18] where it is possible to handle three-way arrays with some elements missing. However those methods are not suitable for our purposes because they deal with $10 - 20\%$ of missing data. In our case we want to compute only about n^2 matrix elements which can be considerably less than 1% of the total amount.

In this work we write down the Parallel Factor algorithm and its high performance implementation for the sparse case. In addition we suggest a new approach to deal with the sparse three-way decomposition when the total number of triads are considerably less than n^2.

4.1. Parallel Factor algorithm for sparse three-way decomposition

We assume that $\{a_{ijk}\}$ array has some elements missing, so the problem is to compute:

$$\min_{B,C,D,\alpha} \sum_{i,j,k} g_{ijk} \left(a_{ijk} - \sum_{l=1}^{r} \alpha_l b_{il} c_{jl} d_{kl} \right)^2,$$

where $g_{ijk} = 0, 1$, $\#\{g_{ijk} = 1\} = s$.

Suppose $A \in \mathbb{R}^s$, $I \in \mathbb{N}^s$, $J \in \mathbb{N}^s$, $K \in \mathbb{N}^s$ are arrays containing nonzero entries and their three-dimensional indices of $a_{ijk} g_{ijk}$. Then, to compute this approximation, we will use the Parallel Factor algorithm, so freeze any two of B, C, D, and the functional is linear in the third. For example, if we temporarily fix the matrices B and C, then we seek D and α so

Algorithm 1.

do $k = 1, m_3$
for $\mu, \nu = 1, \ldots, r$:

$$h_{\mu\nu} = \sum_{\lambda=1}^{s} \delta(K_\lambda, k) b_{I_\lambda,\mu} c_{J_\lambda,\mu} b_{I_\lambda,\nu} c_{J_\lambda,\nu}$$

for $\mu = 1, \ldots, r$:

$$t_\mu = \sum_{\lambda=1}^{s} \delta(K_\lambda, k) A_\lambda b_{I_\lambda,\mu} c_{J_\lambda,\mu}$$

Solve the linear system: $\sum_{\nu=1}^{r} h_{\mu\nu}(d_{k\nu}\alpha_\nu) = t_\mu,\ \mu = 1, \ldots, r$

enddo

where δ is a discrete δ-function.

Here we need $sr^2 + (m_1 + m_2 + m_3)r^3$ arithmetical operations. During most of the computational time we should fetch data randomly from the memory sr^2 times. Since this does not produce good computational performance we suggest transforming this algorithm to an equivalent algorithm with a better performance. Then we need additionally $m_1 + m_2 + m_3$ index arrays $\mathrm{Ind}_i(\xi)$, $\mathrm{Ind}_j(\xi)$, $\mathrm{Ind}_k(\xi)$ to point to λ when $\delta(I_\lambda, i)$, $\delta(J_\lambda, j)$ and $\delta(K_\lambda, k)$ are nonzero respectively. It is easy to see that the total memory requirement for them is $3s$ — the same as for I, J, K arrays. Finally, the algorithm looks like the following:

Algorithm 2.

do $k = 1, m_3$

$H = 0$

$t = 0$

do $\xi = 1$, Size-Of(Ind_k)

$\lambda = \mathrm{Ind}_k(\xi)$

do $\mu = 1, r$

$w_\mu = b_{I_\lambda,\mu} c_{J_\lambda,\mu}$

$t_\mu = t_\mu + A_\lambda b_{I_\lambda,\mu} c_{J_\lambda,\mu}$

enddo

$H = H + ww^*$

enddo

Solve the linear system: $\sum_{\nu=1}^{r} h_{\mu\nu}(d_{k\nu}\alpha_\nu) = t_\mu,\ \mu = 1, \ldots, r$

enddo

Hence, all three steps for updating B, C, and D need $4sr + sr^2 + (m_1 + m_2 + m_3)r^3$ arithmetical operations, but only $4sr$ arithmetical operations need random fetches from memory.

We compared both algorithms on `AMD Athlon 1200` and obtained the following computational performance (Table `I`). We can see that the second algorithm is almost 5-10 times faster (when r is big enough) and reaches 400 MFlop/s which is good enough for the real applications (the peak is 1200 MFlop/s). The same type of Tikhonov's regularizator can be applied, so we need to substitute the matrix in the "solution of linear system" step as follows:

$$\sum_{\nu=1}^{r} (h_{\mu\nu} + \beta)(d_{k\nu}\alpha_\nu) = t_\mu, \qquad \mu = 1, \ldots, r.$$

Table I. Computational performance (in MFlop/s) for one iteration by different algorithms for different problem size on `AMD Athlon 1200`

r	n	s	Algorithm 1	Algorithm 2
1	30	2700	0.39	0.22
1	30	270	0.27	0.14
1	100	3×10^5	4.1	2.7
1	100	3×10^4	4.0	2.6
10	100	3×10^5	23	84
10	100	3×10^4	22	81
100	100	3×10^5	53	487
100	100	3×10^4	53	483

4.2. Tucker-like reduction for sparse three-way decomposition

In this article we suggest a new approach for dealing with sparse three-way decomposition. It contains the following steps:

1) computation of Tucker's factors for the sparse data;
2) computation of a dense core array for the sparse data;
3) application of all methods described in the previous section to compute three-way decomposition for this core array, and finally compute the three-way factors for the original sparse three-way data.

We will now discuss how to perform the first and the second steps. To compute the Tucker's factors we can use the idea from **Statement 1**. Suppose we are going to compute the factor U_B, then we should compute an orthonormal subspace of the left singular values of the matrix $\{\tilde{A}_{i,\{j,k\}}\} \in \mathbb{R}^{n_1^2 \times n_2^2 n_3^2}$, where $\{j,k\}$ means the 2nd and the 3rd indices are joined. To compute the orthonormal subspace of the left singular values we need no more than r linearly independent columns of $\tilde{A}$. The last problem is similar to problems discussed in [2, 7, 8] where it was suggested to pick the biggest $r+\epsilon$ columns, so that ϵ is smaller than an order of r. Hence, we can compute all factors of Tucker's decomposition.

Since we consider that (7) is exactly zero, we can take any linearly independent r_1, r_2, r_3 rows of U_B, U_C, U_D and restrict the initial array $\{a_{ijk}\}$ by the array $\{\tilde{a}_{i''j''k''}\}$ with the size $r_1 \times r_2 \times r_3$. Since r_1, r_2, r_3 are the order of r we need to compute no more than r^3 matrix elements.

As U is a unitary matrix, we can suggest the following idea to compute r_u linearly independent rows: first we take a row with the biggest l_2 norm; each new row should be taken so as to be maximally far in the l_2 norm from the subspace of all previously taken rows.

The total requirements for calculating the sparse three-way decomposition are summarized in the following items:

- we need to compute $\tilde{r}(\tilde{r}^2 + 3n^2)$ matrix elements;
- to make $3\tilde{r}^2(\tilde{r}^2 + n^2)$ arithmetical operations;
- to store no more than $\tilde{r}(\tilde{r}^2 + 3n^2)$ words of memory;

where $\tilde{r} = r + \epsilon$.

5. NUMERICAL EXPERIMENTS

To approximate the matrix by triads we

- approximate the sparse data by Tucker's algorithm for sparse data;
- compute three-way decomposition by the Parallel Decomposition algorithm;
- and tune the result by the Parallel Factor algorithm for sparse data with Tikhonov's regularization.

To create the preconditioner

- for each level we compute three-way decomposition by the Parallel Decomposition algorithm;
- tune the result by the Parallel Factor algorithm with $\texttt{cond}_F$ regularization;
- compute the final factors by formula (4) and invert it.

To demonstrate the efficiency of this method we take the following integral kernels:

A: $K(\bar{x},\bar{y}) = \dfrac{1}{|\bar{x}-\bar{y}|}$,
B: $K(\bar{x},\bar{y}) = \log(|\bar{x}-\bar{y}|)$,
C: $K(\bar{x},\bar{y}) = \dfrac{1}{(\bar{x},\bar{y})}$.

We take a non-regular grid with small steps in the middle, the difference between the smallest and biggest steps being 10 times. This type of grid for case A is desired for Hartree-Fock equations [26]; cases B and C were taken only as examples. For cases A and B we can take the regular grid (we refer to those experiments as A' and B' cases) and use Toeplitz matrices and multilevel circulant preconditioner [21], so we can compare the quality of our new preconditioner. We use a finite-element method with constant finite elements to remove any singularity in matrix elements. The results are presented in Tables `II-VII` (computational time was measured at AMD Athlon 1200).

6. DISCUSSION

We can summarize the main properties of this method in the following statements:

1) The memory requirements for the storage of compressed matrix or preconditioner is only $\mathcal{O}(N^{2/3})$ words.

2) The number of arithmetical operations required for one matrix-vector and preconditioner-vector multiplications is $\mathcal{O}(N^{4/3})$, for the generation of the compressed matrix — $\mathcal{O}(N^{4/3}+r^5)$ and the generation of the preconditioner $\mathcal{O}(N^{5/3})$.

This method is not asymptotically optimal with regards to the arithmetical complexity, but it is suitable for practical computations. If we compare our approach on the regular grid with multilevel Toeplitz and Circulant methods [21], we see that those methods have the best asymptotical estimations $\mathcal{O}(N\log_2 N)$ (i.e. $\mathcal{H}$-matrix approach has $\mathcal{O}(N\log_2^k N)$, $k>1$) and observe the following:

- The memory requirements for the storage of a compressed matrix and the preconditioner are very small, and this allows us to solve huge problems with non-regular grids with up to 200^3 on the PC (Tables `II-III`).
- On regular grids the computational time of matrix-vector and preconditioner-vector multiplications with compressed matrices are comparable with the Toeplitz matrices and circulant preconditioners respectively. Thus the computational time for one preconditioned iteration with three-way compressed matrix is smaller than one preconditioned iteration with a Toeplitz matrix (Table `V`).
  On non-regular grids the new method retains the same arithmetical complexity, but it is impossible to use the Toeplitz approach for this case. Other approaches such as $\mathcal{H}$-matrix approaches have worse estimations $\mathcal{O}(N \log_2^k N)$ than the Toeplitz approach $\mathcal{O}(N \log_2 N)$ on the regular grid, so our new method is reasonably faster than other approaches.
- When the preconditioner is computed, the quality of the new preconditioner is very high. It can solve most problems for few iterations (Tables `VI-VII`), and is considerably better than the circulant preconditioner on regular grids. Since we did not find in the literature any practical implementation of $\mathcal{H}$-matrix preconditioners [27] where the computational time and memory requirements are linearly dependent on the size of the matrix, we did not compare our approach for non-regular grids with any other method.
- The main disadvantage of this method is a high level of arithmetical complexity for creating the preconditioner (Table `IV`). It can be hundreds or thousands of times bigger than for the circulant preconditioner, but if we consider the problem where we should solve a linear system with several right-hand sides and non-regular grids where the circulant preconditioner is not applicable, then this method can help us to solve major problems.

The program that implements the method described in this work is freely available from the author on `http://www.ilghiz.com/`.

REFERENCES

1. Greengard L, Rokhlin V. A fast algorithm for particle simulation. *Journal of Computational Physics* 1987; **73**:325–348.
2. Goreinov SA, Tyrtyshnikov EE, Zamarashkin NL. A theory of pseudoskeleton approximations. *Linear Algebra and Its Applications* 1997; **261**:1–21.
3. Hackbusch W. The panel clustering algorithm. In *The Mathematics of Finite Elements and Applications, VII, (Uxbridge, 1990)*, Academic Press: London, 1991; 339–348.
4. Hackbusch W, Nowak ZP. On the fast matrix multiplication in the boundary element method by panel clustering. *Numerische Mathematik* 1989; **54**(4):463–491.
5. Hackbusch W. A Sparse Matrix Arithmetic Based in $\mathcal{H}$-Matrices. Part I: Introduction to $\mathcal{H}$-Matrices. *Computing* 1999; **62**:89–108.
6. Hackbusch W, Khoromskij BN. A Sparse $\mathcal{H}$-Matrix Arithmetic. Part II: Application to Multi-Dimensional Problems. *Computing* 2000; **64**:21–47.
7. Bebendorf M. Approximation of boundary element matrices. *Numerische Mathematik* 2000; **86**(4):565–589.
8. Bebendorf M, Rjasanow S. Matrix compression for the radiation heat transfer in exhaust pipes. *Preprint 9* 1999; Universitaet des Saarlandes.
9. Van Loan CF, Pitsianis N. Approximation with Kronecker products. *NATO Advantages in Scientific , Series E, Applied Science, 232*, Kluwer: Dordrecht, 1993; 293–314.
10. Harshman R. Foundations of PARAFAC procedure: Models and conditions for an "exploratory" multimode analysis. *UCLA Working Papers in Phonetics* 1970; **16**:1–84.

Table II. The total amount of triads to approximate the initial matrix with 10^{-4} and 10^{-6} residuals in the Frobenius norm.

Problem size	A	B	C	A'	B'
$10 \times 10 \times 10$	9/12	7 /9	7 /9	7 /10	6/8
$20 \times 20 \times 20$	11/14	8 /10	8 /10	8 /11	7/9
$50 \times 50 \times 50$	13/16	10/13	10/12	10/13	8/10
$100 \times 100 \times 100$	16/20	11/14	11/13	12/15	9/11
$200 \times 200 \times 200$	17/21	12/15	12/14	13/16	9/11

11. Carroll JD, Chang J. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decompositions. *Psychometrika* 1970, *35*:283–319.
12. Kruskal JB. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and Its Applications* 1977; **18**:95–138.
13. Harshman R, Lundy M. The PARAFAC model for three-way factor analysis and Multidimensional scaling. In *Research Methods for Multimode Data Analysis*, New York: Praeger, 1984; 122–215.
14. Henrion R. On global, local and stationary solutions in three-way data analysis. *Journal of Chemometrics* 2000; **14**(3):261–274.
15. Tikhonov AN, Samarskij AA. Equations of mathematical physics. *Translated from the Russian by A. R. M. Robson and P. Basu, Reprint of the 1963 translation,* Dover: New York 1990; xvi+765.
16. Tucker LR. Some mathematical notes on three-mode factor analysis. *Psychometrika* 1966; **31**:279–311.
17. Kronenberg PM, Leeuw J. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika* 1980; **45**:69–97.
18. Bro R. PARAFAC. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems* 1997; **38**:149–171.
19. Leurgans SE, Ross RT, Abel RB. A decomposition for three-way arrays. *SIAM Journal on Matrix Analysis and Applications* 1993; **14**(4):1064–1083.
20. Ibraghimov IV. The three-way decomposition. Submitted to *Computing* 2001.
21. Tyrtyshnikov EE. Optimal and superoptimal circulant preconditioners. *Matrix Analysis and Applications* 1992; **13**(2):459–473.
22. Orekhov V, Ibraghimov I, Billeter M. MUNIN: A new approach to multi-dimensional NMR spectra interpretation. *Journal of Biomolecular NMR* 2001; **20**: 49–60.
23. Baur W, Strassen V. The complexity of partial derivatives. *Theoretical Computational Science* 1983; **22**: 317–330.
24. Ibraghimov I, A new approach to solution of general singular vector decomposition problem. In *Matrix Methods and Algorithms* 1999; Moscow: INM RAS, 193–201.
25. Ibraghimov IV. SVD-Like Decomposition With Constraints. *Preprint 26* 2001; Universitaet des Saarlandes.
26. Ibraghimov I, An Application of Structured Matrices for solution of Hartree-Fock equations. In *Matrix Methods and Algorithms* 1999; Moscow: INM RAS, 144–174.
27. Bebendorf M, Hackbusch W. Existence of $\mathcal{H}$-Matrix Approximants to the Inverse FE-Method of Elliptic Operators with L^∞-Coefficients. *Preprint 21* 2002; Max-Planck Institut, Leipzig.

Table III. The average total amount of main memory required to store compressed matrix (**Triads**), to compute this compressed matrix and to create the preconditioner (**Work**) and the average ratio between the amount of matrix elements required for sparse three-way decomposition and the total amount of matrix elements (**Ratio**)

Problem size	**Triads** (Mb)	**Work** (Mb)	**Ratio**
$10 \times 10 \times 10$	0.03	0.9	3.1×10^{-2}
$20 \times 20 \times 20$	0.13	2.3	7.6×10^{-3}
$50 \times 50 \times 50$	0.92	23	4.8×10^{-4}
$100 \times 100 \times 100$	4.58	66	6.0×10^{-5}
$200 \times 200 \times 200$	19.2	201	7.5×10^{-6}

Table IV. The average computational time required to approximate the initial matrix by triads (**Triad**), the generation of the preconditioner (**Prec.**) and the generation of the circulant preconditioner (**Circulant**)

Problem size	**Triad**	**Prec.**	**Circulant**
$10 \times 10 \times 10$	15 ms	0.2 s	0.2 ms
$20 \times 20 \times 20$	0.1 s	5.8 s	2 ms
$50 \times 50 \times 50$	2.3 s	2.7 m	40 ms
$100 \times 100 \times 100$	18 s	19 m	0.4 s
$200 \times 200 \times 200$	71 s	2.1 h	3.5 s

Table V. The computational time required for matrix-vector multiplication with one triad (**Triad**), preconditioner-vector multiplication (**Prec.**), Toeplitz matrix-vector multiplication (**Toeplitz**) and circulant preconditioner-vector multiplication (**Circulant**)

Problem size	**Triad**	**Prec.**	**Toeplitz**	**Circulant**
$10 \times 10 \times 10$	0.03 ms	0.1 ms	3 ms	0.4 ms
$20 \times 20 \times 20$	0.5 ms	1.2 ms	32 ms	4 ms
$50 \times 50 \times 50$	21 ms	50 ms	0.6 s	84 ms
$100 \times 100 \times 100$	0.3 s	1 s	6 s	0.8 s
$200 \times 200 \times 200$	5 s	12 s	55 s	7 s

Table VI. The residual in (6) for approximation of the preconditioner

Problem size	A	B	C	A'	B'
$10 \times 10 \times 10$	0.0153	0.0101	0.0096	0.0115	0.0089
$20 \times 20 \times 20$	0.0081	0.0066	0.0059	0.0054	0.0051
$50 \times 50 \times 50$	0.0058	0.0042	0.0041	0.0029	0.0040
$100 \times 100 \times 100$	0.0047	0.0039	0.0038	0.0022	0.0036
$200 \times 200 \times 200$	0.0038	0.0038	0.0037	0.0017	0.0033

Table VII. The convergence (in the total number of iterations) of GMRES for the 10^{-6} residual on the l_2 norm for problem A' with three-way preconditioner (**Prec.**) and multilevel circulant preconditioner (**Circulant**) [21]

Problem size	**Prec.**	**Circulant**
$10 \times 10 \times 10$	6	26
$20 \times 20 \times 20$	8	32
$50 \times 50 \times 50$	10	43
$100 \times 100 \times 100$	14	51
$200 \times 200 \times 200$	16	—*

∗ the convergence was not achieved in 20 iterations, but the total memory requirements go above 1.5 Gb (available main memory)