Institute for Computational Mathematics
Hong Kong Baptist University

ICM Research Report
08-03

LINEAR ALGEBRA FOR TENSOR PROBLEMS

I. V. OSELEDETS *, D. V. SAVOSTYANOV †, AND E. E. TYRTYSHNIKOV ‡

Abstract. By a *tensor problem* in general, we mean one where all the data on input and output are given (exactly or approximately) in tensor formats, the number of data representation parameters being much smaller than the total amount of data. For such problems, it is natural to seek for algorithms working with data only in tensor formats maintaining the same small number of representation parameters — by the price of all results of computation to be contaminated by approximation (recompression) to occur in each operation. Since approximation time is crucial and depends on tensor formats in use, in this paper we discuss which are best suitable to make recompression inexpensive and reliable. We present fast recompression procedures with sublinear complexity with respect to the size of data and propose methods for basic linear algebra operations with all matrix operands in the Tucker format, mostly through calls to highly optimized level-3 BLAS/LAPACK routines. We show that the canonical format can be avoided without any loss of efficiency. Numerical illustrations are given for approximate matrix inversion via proposed recompression techniques.

Key words. Multidimensional arrays, Tucker decomposition, tensor approximations, low rank approximations, skeleton decompositions, dimensionality reduction, data compression, large-scale matrices, data-sparse methods.

1. Introduction. By a *tensor problem* in general, we mean one where all the data on input and output are given (exactly or approximately) in tensor formats, the number of data representation parameters being much smaller than the total amount of data. For such problems, it is natural to seek for algorithms working with data exclusively in tensor formats enforcing the same small number of representation parameters. This suggests that all final and intermediate results of computation go through some approximation (recompression) inevitable at each step.

In this paper we consider the following basic problem: *given two matrices A and X, compute a tensor-structured approximation $\widetilde{Y}$ to their product*

$$\widetilde{Y} \approx Y = AX.$$

Of course, the number of columns in A must be equal to the number of rows in X. Denote this number by N. For ease of presentation, we assume that A is a square matrix of order N. In regard to X, important extreme cases are the following:

(a) X is a vector (a rectangular matrix of size $N \times 1$);

(b) X is a square matrix of the same order N.

The case (a) can be figured out as a basic operation in all iterative solvers for linear systems or eigenvalue problems with the coefficient matrix A. The case (b) is a workhorse operation in computation of matrix functions of A, in particular A^{-1}.

We intend to exploit a d-dimensional tensor structure in A and X. Above all, this means that N is a product of d positive integers referred to as the *mode sizes*:

$$N = n_1 n_2 ... n_d.$$

Throughout the paper we are interested in the memory and arithmetic cost asymptotics when the mode sizes may grow simultaneously. Thus, in order not to distract

*Institute of Numerical Mathematics, Russian Academy of Sciences, Gubkina Street, 8, Moscow(oseledets1983@yandex.ru, http://spring.inm.ras.ru/osel).

†Institute of Numerical Mathematics, Russian Academy of Sciences, Gubkina Street, 8, Moscow(draug@bach.inm.ras.ru).

‡Institute of Numerical Mathematics, Russian Academy of Sciences, Gubkina Street, 8, Moscow(tee@bach.inm.ras.ru).

the reader by minor details, we will expose the matters as if $n = n_1 = n_2 = ... = n_d$. Moreover, we consider the case $d = 3$, because generalizations to $d > 3$ are indeed straightforward.

We consider two basic options to represent data: the so-called *canonical format* and *Tucker format*. The former applied to A reads

$$A = \sum_{t=1}^{\rho} A_t \otimes B_t \otimes C_t,$$

where A_t, B_t, C_t are $n \times n$ matrices (when the mode sizes are different, these are matrices of order n_1, n_2, n_3). The number of summands ρ is called *canonical rank* of the corresponding canonical format. Here and throughout, $\otimes$ denotes the standard Kronecker product operation: if $C = [c_{ij}]$ then

$$C \otimes B = [c_{ij}B]$$

is naturally viewed as a block matrix with the row (column) size being the product of row (column) sizes of C and D. The canonical format applied to a vector X means a similar decomposition

$$X = \sum_{\tau=1}^{\rho} U_\tau \otimes V_\tau \otimes W_\tau$$

with the only difference that U_τ, V_τ, W_τ are now vectors of size n (when the mode sizes are different, these are vectors of size n_1, n_2, n_3).

By the Tucker format, we always mean a decomposition of the form

$$A = \sum_{\sigma=1}^{r_1} \sum_{\delta=1}^{r_2} \sum_{\tau=1}^{r_3} f_{\sigma\delta\tau} A_\sigma \otimes B_\delta \otimes C_\tau,$$

where A_σ, B_δ, C_τ are $n \times n$ matrices further referred to as *Tucker factors* or *mode factors*, and $f_{\sigma\delta\tau}$ are some numbers comprising the so-called *Tucker core* tensor. Note that here we use the same name Tucker for decompositions that may be free of certain orthogonality constraints considered when the Tucker decomposition comes up as a multidimensional extension of the SVD (singular value decomposition) [16, 15, 23]. Similarly, putting a vector X in the Tucker format means that we expand it as

$$X = \sum_{\alpha=1}^{r_1} \sum_{\beta=1}^{r_2} \sum_{\gamma=1}^{r_3} g_{\alpha\beta\gamma} U_\alpha \otimes V_\beta \otimes W_\gamma,$$

where U_α, V_β, W_γ are vectors of size n, and $g_{\alpha\beta\gamma}$ are components of the corresponding Tucker core tensor.

The quantities r_1, r_2, r_3 are referred to as *mode ranks* of the Tucker format. Again, in order not to obscure things by minor details we will proceed as if $r = r_1 = r_2 = r_3$ (all the same, one should keep in mind that mode ranks may and frequently turn out to be different in practical computations). Thus, any occurrence of α, β, γ and σ, δ, τ needs no specification of the range for these indices. These indices can be always thought of as running from 1 to r (or some other value depending on the context).

In total there are four combinations of tensor formats for A and X. However, it is obviously sufficient, and most viable for practice, to consider three of them as follows:

(CC) A and X are both in the canonical format (*Canonical-Canonical*);
(CT) A is in the canonical while X is in the Tucker format (*Canonical-Tucker*);
(TT) A and X are both in the Tucker format (*Tucker-Tucker*).

The result $\widetilde{Y} \approx Y = AX$ is assumed to keep the format of X.

When using a canonical format for X and $\widetilde{Y}$, we are faced a rather difficult problem of the canonical rank reduction. Several algorithms are available (cf. [4, 5, 6, 17, 14, 2, 19]) but none is claimed as reliable in producing a possible smaller rank with a prescribed accuracy in case it exists. In contrast to this, the Tucker rank reduction can be performed by well-understood matrix techniques with reliability of the SVD (cf. [16, 15]). Moreover, we can mention recent cross approximation techniques proposed in [20] (cf. [10, 11] for the 2D case) and developed in [7, 9] making a Tucker approximation feasible even for huge-scale tensors.

Since the computation $Y = AX$ is a basic operation repeated quite a few times in many algorithms, the canonical rank reduction does not look as very desirable. We advocate it be avoided any time it is not the chief purpose and focus only on the CT and TT combinations. The former is natural when A comes in the canonical format; if not, we still could admit some extra work to obtain a canonical approximation for A since this is done once in the beginning and not on many forthcoming steps. However, in computation of matrix functions we have to multiply new matrices arising on preceding steps, which makes the CT combination not very attractive. Anyway, in all cases the TT combination seems to be fine, although it has not got a proper attention as yet. In this paper we fill in this gap and propose fast recompression procedures in the TT case.

Moreover, we conclude that the canonical format can be avoided without any loss of efficiency. In the end we discuss how the proposed techniques work in approximate matrix inversion and present numerical illustrations for the 3D Laplacian and Newton potential matrices.

2. Pre-recompression stage. Consider the Tucker-Tucker (TT) combination of tensor formats for A and X:

$$A = \sum_{\sigma,\delta,\tau} f_{\sigma\delta\tau}\ A_\sigma \otimes\ B_\delta \otimes\ C_\tau, \tag{2.1}$$

$$X = \sum_{\alpha,\beta,\gamma} g_{\alpha\beta\gamma}\ U_\alpha \otimes\ V_\beta \otimes\ W_\gamma. \tag{2.2}$$

It implies that

$$Y = \sum_{\sigma,\delta,\tau} \sum_{\alpha,\beta,\gamma} f_{\sigma\delta\tau}\ g_{\alpha\beta\gamma}\ A_\sigma U_\alpha \otimes\ B_\delta V_\beta \otimes\ C_\tau W_\gamma. \tag{2.3}$$

Approximation of Y begins with a *pre-recompression stage* that consists in the computation of matrices

$$A'_{\sigma\alpha} = A_\sigma U_\alpha, \quad B'_{\delta\beta} = B_\delta V_\beta, \quad C'_{\tau\gamma} = C_\tau W_\gamma. \tag{2.4}$$

Consequently,

$$Y = \sum_{\sigma,\delta,\tau} \sum_{\alpha,\beta,\gamma} f_{\sigma\delta\tau}\ g_{\alpha\beta\gamma}\ A'_{\sigma\alpha} \otimes\ B'_{\delta\beta} \otimes\ C'_{\tau\gamma}. \tag{2.5}$$

Matrices Y, $A'_{\sigma\alpha}$, $B'_{\delta\beta}$, $C'_{\tau\gamma}$ can be transformed to vectors where the entries are successively picked up column by column. If we keep the same notation for the corresponding vectors, then it is easy to see that they *satisfy the same equation* (2.5). Hence, (2.5) unifies the above extreme cases (a) and (b) (see Introduction) and equally applies to the factor matrices U_α, V_β, W_γ consisting of any fixed number of columns.

Let us use indices i, j, k to point to the entries of vectors $A'_{\sigma\alpha}$, $B'_{\delta\beta}$, $C'_{\tau\gamma}$, respectively. Then the same i, j, k can be naturally used to mark the entries of the vector Y regarded as a three-dimensional array, [1] and (2.5) can be written in the entry-wise form as follows:

$$y_{ijk} = \sum_{\sigma,\delta,\tau} \sum_{\alpha,\beta,\gamma} f_{\sigma\delta\tau}\ g_{\alpha\beta\gamma}\ a'_{i\sigma\alpha}\ b'_{j\delta\beta}\ c'_{k\tau\gamma}, \tag{2.6}$$

where

$$A'_{\sigma\alpha} = [a'_{i\sigma\alpha}], \quad B'_{\delta\beta} = [b'_{\delta\beta}], \quad C'_{\tau\gamma} = [c'_{\tau\gamma}].$$

In the right-hand side of (2.6) one can recognize a Tucker format with the mode ranks r^2, already a way larger than the initial value r. Moreover, if we repeat operations of the same kind, as is typical in applications (cf. [3, 7, 12, 13, 21]), the resulting mode ranks can increase considerably compared to r. Even being independent of the mode sizes, they grow rapidly with each operation. To make the tensor algebra feasible, we need to find some fast ways of reducing the mode ranks by adding an admissible perturbation to the resulting tensor.

3. General rank reduction methods. Consider first a general tensor

$$Z = [z_{ijk}]$$

with the mode sizes n, which means that i, j, k run from 1 to n. Then the rank-r Tucker reduction is computed by the standard general method (known by the names of Tensor SVD or Higher Order SVD [16]) as follows.

Algorithm 3.1.

- *Construct the unfolding matrices*

$$Z_1 = [z_{i,(jk)}], \quad Z_2 = [z_{j,(ik)}], \quad Z_3 = [z_{k,(ij)}],$$

where the second index is a "long index" composed of two indices of the original tensor and, by definition,

$$z_{i,(jk)} = z_{j,(ik)} = z_{k,(ij)} = z_{ijk}.$$

- *Perform rank revealing decompositions (maybe via the SVD)*

$$Z_1 = Q_1 R_1 + E_1, \quad Z_2 = Q_2 R_2 + E_2, \quad Z_3 = Q_3 R_3 + E_3,$$

where

$$Q_1 = [q^1_{i\alpha}], \quad Q_2 = [q^2_{j\beta}], \quad Q_3 = [q^3_{k\gamma}]$$

are orthogonal $n \times r$ matrices [2] *and E_1, E_2, E_3 are supposedly of sufficiently small norm.*

[1] According to the definition of Kronecker product, this is an anti-FORTRAN style indexing, because the last index k runs first, then j, and in the last turn i.

[2] A rectangular matrix is said to be *orthogonal* if it has orthonormal columns.

- *Compute the Tucker core tensor*

$$h_{\alpha\beta\gamma} = \sum_{i,j,k} q^1_{i\alpha}\ q^2_{j\beta}\ q^3_{k\gamma}\ z_{ijk}.$$

- *Finish with the Tucker approximation in the form*

$$y_{ijk} \approx \widetilde{y}_{ijk} = \sum_{\alpha\beta\gamma} h_{\alpha\beta\gamma}\ q^1_{i\alpha}\ q^2_{j\beta}\ q^3_{k\gamma}.$$

We refer to the orthogonal Tucker factors Q_1, Q_2, Q_3 as the *mode frame matrices*. As soon as they are fixed, the above choice of the Tucker core $h_{\alpha\beta\gamma}$ guarantees the minimal possible value for the distance

$$||Z - \widetilde{Y}|| = \sqrt{\sum_{i,j,k} |z_{ijk} - \widetilde{y}_{ijk}|^2}.$$

Also, the Tucker core computation reduces to three matrix-matrix multiplications, so one can use perfectly optimized `dgemm` procedures from BLAS/LAPACK software.

Algorithm 3.1 requires $O(n^4)$ operations and, consequently, cannot be used for large n. Note that n^4 is due to the SVD-based computation of the mode frame matrices. We should add $O(n^3r + n^2r^2 + nr^3)$ operations for the Tucker core computation.

However, lower costs can be achieved by the ALS (alternating least squares) approach. In line with the above remark on optimality of the Tucker core choice, we can treat $\widetilde{Y}$ as a function of Q_1, Q_2, Q_3. Then, starting from some initial guess for Q_1, Q_2, Q_3 we can pick up each of them in turn and substitute it, with other two frozen, with a minimizer for $||Z - \widetilde{Y}(Q_1, Q_2, Q_3)||$. Then, we proceed with these three-step cycles until convergence. Usually very few (up to 3-4) cycles are needed, provided that a sufficiently accurate Tucker approximation of rank r exists.

Algorithm 3.2.

- *Freeze Q_2, Q_3 and compute*

$$h^1_{i\beta\gamma} = \sum_{j,k} q^2_{j\beta}\ q^3_{k\gamma}\ z_{ijk}.$$

  Consider a matrix $H_1 = [h^1_{i\beta\gamma}]$ of size $n \times r^2$ (here β, γ form a "long index" for columns) and find Q_1 from a rank revealing decomposition

$$H_1 = Q_1 R_1 + F_1$$

  with minimal $||F_1||_F$. Note that Q_1 is a maximizer for $||Q^\top H_1||_F$ over all matrices Q with r orthonormal columns.

- *Freeze Q_1, Q_3 and compute*

$$h^2_{j\alpha\gamma} = \sum_{i,k} q^1_{i\alpha}\ q^3_{k\gamma}\ z_{ijk}.$$

  Consider a matrix $H_2 = [h^2_{j\alpha\gamma}]$ of size $n \times r^2$ (here α, γ form a "long index" for columns) and find Q_2 from a rank revealing decomposition

$$H_2 = Q_2 R_2 + F_2$$

  with minimal $||F_2||_F$. Note that Q_2 is a maximizer for $||Q^\top H_2||_F$ over all matrices Q with r orthonormal columns.

- *Freeze Q_1, Q_2 and compute*

$$h^3_{k\alpha\beta} = \sum_{i,j} q^1_{i\alpha}\ q^2_{j\beta}\ z_{ijk}.$$

Consider a matrix $H_3 = [h^3_{k\alpha\beta}]$ of size $n \times r^2$ (here α, β form a "long index" for columns) and find Q_3 from a rank revealing decomposition

$$H_3 = Q_3 R_3 + F_3$$

with minimal $||F_3||_F$. Note that Q_3 is a maximizer for $||Q^\top H_3||_F$ over all matrices Q with r orthonormal columns.
- *Repeat until convergence, then compute the Tucker core for the obtained mode frame matrices.*

A straightforward computation of matrices H_1, H_2, H_3 (by a cycle through all involved indices) formally results in the $O(n^3r^2)$ complexity. However, this cost can be easily dropped down to $O(n^3r + n^2r^2)$ by two-step computations as follows:

$$\widetilde{h}^1_{ij\gamma} = \sum_k q^3_{k\gamma} z_{ijk}, \quad h^1_{i\beta\gamma} = \sum_j q^2_{j\beta} \widetilde{h}^1_{ij\gamma}, \tag{3.1}$$

$$\widetilde{h}^2_{ij\gamma} = \sum_k q^3_{k\gamma} z_{ijk}, \quad h^2_{j\alpha\gamma} = \sum_i q^1_{i\alpha} \widetilde{h}^2_{ij\gamma}, \tag{3.2}$$

$$\widetilde{h}^3_{ik\beta} = \sum_j q^2_{j\beta} z_{ijk}, \quad h^3_{k\alpha\beta} = \sum_i q^1_{i\alpha} \widetilde{h}^3_{ik\gamma}. \tag{3.3}$$

Since the SVD cost for a matrix of size $n \times r^2$ is $O(\min(n^2r^2, nr^4))$, the overall complexity of Algorithm 3.2 becomes $O(n^3r + n^2r^2 + nr^3)$. We could think of a better rank revealing part, but it does nothing to the already existing term $O(n^2r^2)$.

For a fixed rank r, the ALS complexity is linear in the total data size (which is $N = n^3$). Nonetheless, both algorithms are of the cost we cannot afford for the rank reduction of data given already in a tensor format. In the recompression, only a sub-linear complexity could be feasible. Below we present such recompression algorithms with a linear in $n = N^{1/3}$ cost.

4. General recompression methods. Assume that a tensor $Z = [z_{ijk}]$ is given in the Tucker format

$$z_{ijk} = \sum_{\alpha,\beta,\gamma} h_{\alpha\beta\gamma}\ a_{i\alpha}\ b_{j\beta}\ c_{k\gamma}$$

with the mode sizes n and mode ranks r_0, and we are interested to approximate it by a tensor in the same format with the mode ranks $r < r_0$.

The storage for Z is $3nr_0 + r_0^3$. Obviously, we should not use a much larger storage during recompression, so computation of unfolding matrices of size $n \times n^2$ is prohibited. A direct way is as follows (we assume that $n \geq r_0$).

Algorithm 4.1.

- *Compute $n \times r_0$ matrices*

$$Q_1 = [q^1_{i\alpha'}], \quad Q_2 = [q^2_{j\beta'}], \quad Q_3 = [q^3_{k\gamma'}]$$

with orthonormal columns and $r_0 \times r_0$ matrices

$$R_1 = [r^1_{\alpha'\alpha}], \quad R_2 = [r^2_{\beta'\beta}], \quad R_3 = [r^3_{\gamma'\gamma}]$$

via the QR decompositions

$$[a_{i\alpha}] = Q_1 R_1, \quad [b_{j\beta}] = Q_2 R_2, \quad [c_{k\gamma}] = Q_3 R_3.$$

- *Acquire an auxiliary $r_0 \times r_0 \times r_0$ tensor* [3]

$$h'_{\alpha'\beta'\gamma'} = \sum_{\alpha,\beta,\gamma} r^1_{\alpha'\alpha}\ r^2_{\beta'\beta}\ r^3_{\gamma'\gamma}\ h_{\alpha\beta\gamma}.$$

in three steps as follows:

$$\begin{aligned} h^*_{\alpha'\beta\gamma} &= \sum_{\alpha} r^1_{\alpha'\alpha}\ h_{\alpha\beta\gamma}, \\ h^{**}_{\alpha'\beta'\gamma} &= \sum_{\beta} r^2_{\beta'\beta}\ h^*_{\alpha'\beta\gamma}, \\ h'_{\alpha'\beta'\gamma'} &= \sum_{\gamma} r^2_{\gamma'\gamma}\ h^{**}_{\alpha'\beta'\gamma}. \end{aligned}$$

- *Apply a mode rank reduction algorithm to the auxiliary tensor*

$$h'_{\alpha'\beta'\gamma'} \approx \sum_{\sigma=1}^{r}\sum_{\delta=1}^{r}\sum_{\tau=1}^{r} p^1_{\alpha'\sigma}\ p^2_{\beta'\delta}\ p^3_{\gamma'\tau} \widetilde{h}_{\sigma\delta\tau}.$$

- *Finally,*

$$z_{ijk} \approx \sum_{\sigma=1}^{r}\sum_{\delta=1}^{r}\sum_{\tau=1}^{r} \widetilde{q}^1_{i\sigma}\ \widetilde{q}^2_{j\delta}\ \widetilde{q}^3_{l\tau}\ \widetilde{h}_{\sigma\delta\tau}$$

with the Tucker factors

$$\widetilde{q}^1_{i\sigma} = \sum_{\alpha'=1}^{r_0} q^1_{i\alpha'} p^1_{\alpha'\sigma}, \quad \widetilde{q}^2_{j\delta} = \sum_{\beta'=1}^{r_0} q^2_{j\beta'} p^2_{\beta'\delta}, \quad \widetilde{q}^3_{j\tau} = \sum_{\gamma'=1}^{r_0} q^2_{k\gamma'} p^3_{\gamma'\tau}.$$

The complexity of QR decompositions is $O(nr_0^2)$ and dominates $O(nr_0 r)$ in the final computation of Tucker factors $\widetilde{q}^1_{i\sigma}$, $\widetilde{q}^2_{j\delta}$, $\widetilde{q}^3_{k\tau}$. The auxiliary core computation by the above prescriptions requires $O(r_0^4)$ operations, in contrast to a straightforward (naive) way with $O(r_0^6)$ operations. Moreover, all we need to do reduces to three matrix-matrix multiplications, where we can benefit a lot from using well-optimized BLAS/LAPACK routines. The rank reduction step for the auxiliary $r_0 \times r_0 \times r_0$ core can be handled by Algorithm 3.1 with the $O(r_0^4)$ complexity or, which is slightly better, by Algorithm 3.2 with the $O(r_0^3 r)$ complexity.

[3] Equivalently, $h'_{\alpha'\beta'\gamma'} = \sum_{\alpha,\beta,\gamma}\sum_{i,j,k} q^1_{i\alpha'} q^2_{j\beta'} q^3_{k\gamma}\ a_{i\alpha}\ b_{j\beta}\ c_{k\gamma}\ h_{\alpha\beta\gamma}$.

All in all, Algorithm 4.1 with an internal call to Algorithm 3.1 or Algorithm 3.2 requires $O(nr_0^2 + nr_0r + r_0^4)$ or $O(nr_0^2 + nr_0r + r_0^4 + r_0^3r)$ operations, respectively.

An alternative could be the ALS approach. It starts with some initial guess for the mode frame matrices $Q_1 = [q^1_{i\sigma}]$, $Q_2 = [q^2_{j\delta}]$, $Q_3 = [q^3_{k\tau}]$ with r orthonormal columns and then iterates with three-step cycles, where each of these three matrices, in turn, is replaced with the best fit while two others are kept frozen. In particular, if Q_3 is sought when Q_1 and Q_2 are frozen, we are to compute nr^2 values

$$h^3_{k\sigma\delta} = \sum_{\alpha,\beta,\gamma} \sum_{i,j} q^1_{i\sigma}\ q^2_{j\delta}\ a_{i\alpha}\ b_{j\beta}\ c_{k\gamma}\ h_{\alpha\beta\gamma}.$$

This can be done through a sequence of matrix-matrix computations as follows:

$$s^1_{\sigma\alpha} = \sum_i q^1_{i\sigma}\ a_{i\alpha}, \quad s^2_{\delta\beta} = \sum_j q^2_{i\delta}\ b_{j\beta},$$

$$d'_{\sigma\beta\gamma} = \sum_\alpha s^1_{\sigma\alpha}\ h_{\alpha\beta\gamma}, \quad d''_{\sigma\delta\gamma} = \sum_\beta s^2_{\delta\beta}\ d'_{\sigma\beta\gamma},$$

$$h^3_{k\sigma\delta} = \sum_\gamma d''_{\sigma\delta\gamma} c_{k\gamma}.$$

The cost amounts to $O(nr_0r^2 + r_0^3r)$ operations. Thus, the ALS approach might outperform Algorithm 4.1 only when $r^2 < r_0$. Since in our applications it is typical that $r_0 = r^2$, we may not pursue the ALS line in this place.

5. Recompression in the Tucker-Tucker case. From a general recompression case we now step back to a special one (most important for us) arising in the computation of a Tucker rank-r approximation $\widetilde{Y}$ for $Y = AX$ with the Tucker-Tucker combination in tensor formats for A and X. Upon completion of the pre-recompression stage we obtain a tensor of the form (2.6). Let us omit primes and get to a simpler notation as follows:

$$y_{ijk} = \sum_{\sigma,\delta,\tau} \sum_{\alpha,\beta,\gamma} f_{\sigma\delta\tau}\ g_{\alpha\beta\gamma}\ a_{i\sigma\alpha}\ b_{j\delta\beta}\ c_{k\tau\gamma}. \tag{5.1}$$

Assume that indices i, j, k run from 1 to n and $\sigma, \delta, \tau, \alpha, \beta, \gamma$ do from 1 to r. We take it for granted that Y admits a sufficiently accurate Tucker approximation with the mode ranks r.

The input (5.1) and a compressed tensor on output are stored in $O(nr^2 + r^3)$ and $O(nr + r^3)$ cells, respectively. An extra storage for intermediate steps is, of course, allowed and even necessary to reduce the computational costs, but it must not be too much larger than the input. Consider first a modification of Algorithm 4.1 assuming that $n \geq r^2$.

Algorithm 5.1.

- *Compute orthogonal $n \times r^2$ matrices*

$$Q_1 = [q^1_{i,(\sigma'\alpha')}], \quad Q_2 = [q^2_{j,(\delta'\beta')}], \quad Q_3 = [q^3_{k,(\tau'\gamma')}]$$

and $r^2 \times r^2$ matrices

$$R_1 = [r^1_{(\sigma'\alpha'),(\sigma\alpha)}], \quad R_2 = [r^2_{(\delta'\beta'),(\delta\beta)}], \quad R_3 = [r^3_{(\tau'\gamma'),(\tau\gamma)}]$$

via the QR decompositions

$$[a_{i,(\sigma\alpha)}] = Q_1 R_1, \quad [b_{j,(\delta\beta)}] = Q_2 R_2, \quad [c_{k,(\tau\gamma)}] = Q_3 R_3.$$

- *Define the auxiliary core tensor by*

$$h_{\sigma'\alpha'\delta'\beta'\tau'\gamma'} = \sum_{\sigma,\delta,\tau} \sum_{\alpha,\beta,\gamma} r^1_{\sigma'\alpha'\sigma\alpha}\ r^2_{\delta'\beta'\delta\beta}\ r^3_{\tau'\gamma'\tau\gamma}\ f_{\sigma\delta\tau}\ g_{\alpha\beta\gamma}$$

and compute it through the following prescriptions:

$$\begin{aligned} h'_{\sigma'\alpha'\delta\beta\tau\gamma} &= \sum_{\sigma,\alpha} r^2_{\sigma'\alpha'\sigma\alpha}\ f_{\sigma\delta\tau}\ g_{\alpha\beta\gamma}, \\ h''_{\sigma'\alpha'\delta'\beta'\tau\gamma} &= \sum_{\delta,\beta} r^2_{\delta'\beta'\delta\beta}\ h'_{\sigma'\alpha'\delta\beta\tau\gamma}, \\ h_{\sigma'\alpha'\delta'\beta'\tau'\gamma'} &= \sum_{\tau,\gamma} r^3_{\tau'\gamma'\tau\gamma}\ h''_{\sigma'\alpha'\delta'\beta'\tau\gamma}. \end{aligned}$$

- *Apply a mode rank reduction algorithm to the auxiliary tensor.*
- *Recompute the Tucker factors in the final Tucker approximation with mode ranks r.*

Algorithm 5.1 requires $O(nr^4 + r^8)$ operations and $O(nr^2 + r^6)$ memory cells. Note that $O(r^8)$ operations appear already when we get the auxiliary core. [4] We may add $O(r^8)$ operations in the call of Algorithm 3.1 at the rank reduction step and need not seek here for an improvement.

It is very important to note that another "naive" approach is largely not applicable. One may want to circumvent the computation of a full $r^2 \times r^2 \times r^2$ core and, to this end, to compress first the factor matrices via SVD (instead of QR). The following two-dimensional example shows that it may fail in some cases. Consider the following rank-2 matrix represented as

$$A = UV^\top = \begin{pmatrix} a & \varepsilon^2 b \end{pmatrix} \begin{pmatrix} c^\top \\ \varepsilon^{-1} d^\top \end{pmatrix} = ac^\top + \varepsilon b d^\top,$$

where a, b, c, d are unit-length vectors of size n and $(a, b) = (c, d) = 0$. If we choose ε to be very small, than A is close to a rank-1 matrix, but if we "compress" U and V matrices separately, we obtain the senior singular vector of U to be a and the same for V to be d, so we are to seek an approximation to A in the form

$$A \approx \gamma a d^\top,$$

which leaves us with no hope. It is easy to provide an example where the accuracy falls from ε to $\sqrt{\varepsilon}$ and the angles between the vectors come into play. In some cases, still, precompression of U and V matrices would work (in three-dimensional case the situation is the same but with three matrices instead of two) but surely this method is not satisfactory when we have many iterations in some iterative method. A simplest non-model example is the inverse matrix for a three-dimensional Laplacian: the matrix itself has mode ranks $(2, 2, 2)$, the inverse matrix has approximate mode ranks of order $10 \div 12$, their product has approximate inverse ranks $(1, 1, 1)$ (this is the identity

[4] It is instructive to remark that a naive computation of the auxiliary core by a cycle in all involved indices would lead to $O(r^{12})$ operations, which we can hardly afford even for $r = 10$.

matrix!) but the pre-recompression factors are not very close to rank-1 matrices. Thus, we have to compute the full core and treat all the factors *simultaneously*, not separately.

Consider also the ALS recompression for (5.1). As usual, it begins with some initial guess for the orthogonal Tucker factors $Q_1 = [q^1_{i\alpha'}]$, $Q_2 = [q^2_{j\beta'}]$, $Q_3 = [q^3_{k\gamma'}]$ (mode frame matrices) with α', β', γ' in the range from 1 to r.

Algorithm 5.2.

- *Freeze Q_2, Q_3 and find the best fit for Q_1. To this end, compute the values*

$$v_{\beta'\delta\beta} = \sum_j q^2_{j\beta'}\ b_{j\delta\beta},$$

$$w_{\gamma'\tau\gamma} = \sum_k q^3_{k\gamma'}\ c_{k\tau\gamma},$$

then acquire

$$h_{i\beta'\gamma'} = \sum_{\sigma,\delta,\tau}\sum_{\alpha,\beta,\gamma} f_{\sigma\delta\tau}\ g_{\alpha\beta\gamma}\ v_{\beta'\delta\beta}\ w_{\gamma'\tau\gamma}\ a_{i\sigma\alpha}$$

via the following prescriptions:

$$\begin{aligned} u'_{\alpha\beta\gamma'\tau} &= \sum_\gamma g_{\alpha\beta\gamma}\ w_{\gamma'\tau\gamma}, \\ u''_{\alpha\beta\gamma'\sigma\delta} &= \sum_\tau u'_{\alpha\beta\gamma'\tau}\ f_{\sigma\delta\tau}, \\ u_{\alpha\gamma'\sigma\beta'} &= \sum_{\delta,\beta} u''_{\alpha\beta\gamma'\sigma\delta}\ v_{\beta'\delta\beta}, \\ h_{i\beta'\gamma'} &= \sum_{\sigma,\alpha} u_{\alpha\gamma'\sigma\beta'}\ a_{i\sigma\alpha}. \end{aligned}$$

Obtain Q_1 from a rank revealing decomposition of the matrix $H_1 = [h_{i,(\beta'\gamma')}]$ of size $n \times r^2$.

- *Similarly, freeze Q_1, Q_3 and find the best fit for Q_2, then freeze Q_1, Q_2 and find the best fit for Q_3.*
- *Repeat until convergence.*

Algorithm 5.2 needs $O(nr^4 + r^6)$ operations and $O(nr^2 + r^5)$ memory cells.

6. Important implementation details. Essential parts of the above recompression algorithms consist in computation of multiple sums over several indices. As it was mentioned already, this must not be made straightforwardly but should evolve into a sequence of matrix-matrix multiplications. The involved rectangular matrices are just reshaped tensors with groups of indices considered as one "long index" and sometimes with a transposition of modes.

Assume that indices α, β, γ run from 1 to r_1, r_2, r_3, respectively, and similarly, α', β', γ' run from 1 to r'_1, r'_2, r'_3. Then set

$$r''_1 = r_1 r'_1, \quad r''_2 = r_2 r'_2, \quad r_3 r'_3$$

and let indices $\alpha'', \beta'', \gamma''$ run from 1 to r''_1, r''_2, r''_3. Consider in more detail the computation of a six-tuple sum

$$g''_{\alpha''\beta''\gamma''} = \sum_{\alpha,\beta,\gamma,\alpha',\beta',\gamma'} g_{\alpha\beta\gamma}\ g'_{\alpha'\beta'\sigma'}\ \xi_{\alpha''\alpha\alpha'}\ \eta_{\beta''\beta\beta'}\ \zeta_{\gamma''\gamma\gamma'}. \tag{6.1}$$

We propose to use efficient 3-level BLAS routines by the price of taking some additional memory.

First we summate over γ':

$$g^{(1)}_{\alpha'\beta'\gamma''\gamma} = \sum_{\gamma'} g'_{\alpha'\beta'\gamma'} \zeta_{\gamma''\gamma\gamma'}. \tag{6.2}$$

One can recognize in (6.2) the multiplication of a $(r'_1 r'_2) \times r'_3$ matrix G with elements $g'_{\alpha'\beta'\gamma'}$ (this is a so-called *unfolding* of a tensor) by a $r'_3 \times (r''_3 r_3)$ matrix with elements $\zeta_{\gamma''\gamma\gamma'}$. The result is a $(r'_1 r'_2) \times (r''_3 r_3)$ matrix which can be considered also as a four-dimensional tensor with elements $g^{(1)}_{\alpha'\beta'\gamma''\gamma}$. If the initial tensors were stored contiguously in memory in the Fortran style, then we do not have to perform any permutation of the elements and use just one call to `dgemm`. In Matlab we need to perform two reshape calls and one matrix multiplication (which takes most of the time).

Analogously, we perform the summation over γ and obtain a new array with elements

$$g^{(2)}_{\alpha\beta\alpha'\beta'\gamma''} = \sum_{\gamma} g_{\alpha\beta\gamma}\ g^{(1)}_{\alpha'\beta'\gamma''\gamma}.$$

Since the index γ is on the right for both multidimensional arrays, again no permutation of elements is needed to bring them to a matrix multiplication form and we just invoke a matrix multiplication procedure which multiplies a $(r_1 r_2 \times r_3)$ matrix by a $r_3 \times (r_3 r'_1 r'_2 r''_3)$ matrix yielding a $(r_1 r_2) \times (r'_1 r'_2 r''_3)$ matrix which can be considered as a five-dimensional tensor with elements $g^{(2)}_{\alpha\beta\alpha'\beta'\gamma''}$.

Now we have to summate over $\alpha, \alpha', \beta, \beta'$:

$$g''_{\alpha''\beta''\gamma''} = \sum_{\alpha\alpha'\beta\beta'} g^{(2)}_{\alpha\beta\alpha'\beta'\gamma''} \xi_{\alpha''\alpha\alpha'} \eta_{\beta''\beta\beta'}. \tag{6.3}$$

It is the only place where we have to permute elements of the tensor. It is evident that summation over α, α' and over β, β' can be done independently. To represent this kind of summation as a matrix-by-matrix multiplication we have to permute the tensor elements:

$$g^{(3)}_{\alpha\alpha'\beta\beta'\gamma''} = g^{(2)}_{\alpha\beta\alpha'\beta'\gamma''}.$$

In Matlab this can be done via a single call to the permute function. After such a transposition we obtain a $r_1 \times r'_1 \times r_2 \times r'_2 \times r''_3$ tensor which we consider as a $(r_1 r'_1) \times (r_2 r'_2) \times r''_3$ three-dimensional tensor and the sum in (6.3) is equivalent to two tensor-by-matrix multiplications (mode contractions) computed via two matrix-by-matrix multiplications. In Matlab, `ttm` function from the TensorToolbox [1] can be used. As the result, we obtain the tensor $g''_{\alpha''\beta''\gamma''}$.

Since all computations are based solely on matrix-by-matrix multiplications (plus a permutation of elements, which is negligible) we enjoy a high efficiency on any computer with optimized BLAS libraries. If $r_1 = r_2 = r_3 = r'_1 = r'_2 = r'_3 = r$ then the complexity if $O(r^8)$. The first step (summation over γ') requires $\mathcal{O}(r^7)$ operations, the summation over γ does $\mathcal{O}(r^8)$ operations. The computations in (6.3) are contractions of an $r^2 \times r^2 \times r^2$ tensor by $r^2 \times r^2$ matrices, each is realized via matrix-by-matrix multiplication with operands of size $r^4 \times r^2$ and $r^2 \times r^2$. Thus, these contractions cost $\mathcal{O}(r^8)$ each.

7. Matrix algebra in tensor formats. Basic operations with $N \times N$ matrices are multiplication, summation and inversion. Let $N = n^3$ and matrices are given in the Tucker format with mode ranks r. Concerning multiplication, here we have to remark that the pre-recompression costs amount to $O(n^3 r^2)$ operations and are linear in N in the 3D case. They become sublinear in N only when dimension is greater than 3. In the 3D case, sublinear costs can be still achieved by exploiting some further structure in the Tucker factors or some complementary compression techniques such as wavelets [8].

Also, in the 3D case the mode sizes on the recompression stage should be set to n^2. As a corollary, pre-recompression and then recompression by Algorithm 4.1 altogether take $O(n^3 r^2 + n^2 r^2 + r^8)$ operations.

The sum of two matrices A and A' can be computed much faster then their multiplication. Again, we constitute two "tall" matrices in each mode (with $r_l + r'_l$ columns for the lth mode), compute their QR-decompositions in $O * n^2 r^2)$ operations and then come up with a $(r_1 + r'_1) \times (r_2 + r'_2) \times (r_3 + r'_3)$ core tensor in $\mathcal{O}(r^4)$ operations. The compression of the core by Algorithm 3.1 takes $\mathcal{O}(r^4)$ operations (again, for simplicity of presentation we take all mode ranks equal). The cost estimate

$$\mathcal{O}(n^2 r^2) + \mathcal{O}(r^4),$$

shows that time for summations is negligible. It obviously applies to a `daxpy` operation.

The inversion can be made by the Newton-Schultz approximate iterations [21, 13]. We discuss it in more detail later.

8. Remarks on the Canonical-Tucker combination. As we conclude from the above, matrix algebra operations can be implemented efficiently and effectively within the Tucker format. But, the canonical one is still simpler. Sometimes one of the operands is naturally supplied in the canonical format as a result of trilinear (canonical) approximation (with analytical expressions for many operator matrices) while the other one is cast in the Tucker format. This happens in structured Krylov-type methods, where we have to compute Ax many times with one and the same matrix A and different vectors x. Thence we ought to use a Tucker format for x. For the Canonical-Tucker (CT) cases one may expect that a Tucker representation of AA' can be found in a faster way than for the TT combination. Below we show that in fact we have no gain.

Since

$$A = \sum_{t=1}^{\rho} U_t \otimes V_t \otimes W_t,$$

$$A' = \sum_{\alpha=1}^{r_1} \sum_{\beta=1}^{r_2} \sum_{\gamma=1}^{r_3} g_{\alpha\beta\gamma} \, U_\alpha \otimes V_\beta \otimes W_\gamma,$$

now the pre-recompression stage reads

$$AA' = \sum_{\alpha,\beta,\gamma} g_{\alpha\beta\gamma} \, (U_t U_\alpha) \otimes (V_t V_\beta) \otimes (W_t W_\gamma).$$

To make recompression, we proceed similar to the Tucker-Tucker case: compute QR decompositions and carry out mode contractions, now of a simpler form

$$g''_{\alpha''\beta''\gamma''} = \sum_{t,\alpha,\beta,\gamma} g_{\alpha\beta\gamma}\ \xi_{\alpha''t\alpha}\ \eta_{\beta''t\beta}\ \zeta_{\gamma''t\gamma}$$

with indices $\alpha'', \beta'', \gamma''$ running from 1 to

$$r''_1 = r_1\rho, \quad r''_2 = r_2\rho, \quad r''_3 = r_3\rho,$$

respectively. This auxiliary core computation reduces to three tensor-by-matrix multiplications (mode contractions) as follows:

$$\begin{aligned} g^1_{\alpha''\beta\gamma t} &= \sum_{\alpha} g_{\alpha\beta\gamma}\ \xi_{\alpha''t\alpha}, \\ g^2_{\alpha''\beta''\gamma t} &= \sum_{\beta} g^1_{\alpha''\beta\gamma t}\ \eta_{\beta''t\beta}, \\ g''_{\alpha''\beta''\gamma''} &= \sum_{\gamma,t} g^2_{\alpha''\beta''\gamma t}\ \zeta_{\gamma''t\gamma}. \end{aligned}$$

If $\rho = r_1 = r_2 = r_3 = r$, then the cost is still $O(r^8)$, and together with the complexity of QR decompositions it gets us up with the same $O(n^2r^4 + r^8)$ operations for recompression plus $O(n^3r^2)$ for matrix-by-matrix products on the pre-recompression stage.

One may still observe that the auxiliary $r''_1 \times r''_2 \times r''_3$ core tensor $G'' = [g''_{\alpha''\beta''\gamma''}]$ can be rewritten as

$$G = \sum_t G \times_1 X_t \times_2 Y_t \times_3 Z_t,$$

where matrices X_t, Y_t, Z_t are $r_1 \times r''_1$, $r_2 \times r''_2$, $r_3 \times r''_3$, respectively. Thus, we can compute ρ tensor-by-matrix products in parallel and sum up the results. However, the Tucker-Tucker combination has the same order complexity and certainly excels in having much wider applications.

9. Matrix inversion algorithms. As an application of the proposed recompression algorithms and developed BLAS-based routines we consider the Newton method for approximate inversion of a given matrix A. It reads

$$X_{k+1} = 2X_k - X_kAX_k, \quad k = 0, 1, \dots, \tag{9.1}$$

and converges quadratically (in exact arithmetics) provided that the initial residual satsfies

$$||I - AX_0|| < 1.$$

We suppose that A is given in the Tucker format and are going to maintain the same structure with limited mode ranks for all iterates X_k. An obvious way to do this is by a (nonlinear) projection onto the space of low tensor rank matrices:

$$Z_{k+1} = 2X_k - X_kAX_k, \quad X_{k+1} = \mathcal{P}(Z_{k+1}). \tag{9.2}$$

Let $\mathcal{P}$ be a projector onto the space of matrices in the Tucker format with mode ranks r_1, r_2, r_3 selected and fixed before the method starts. Another approach based

on fixing a bound ε for the approximation accuracy might be more robust, but as well it may lead to a higher growth of mode ranks during iterations and, sometimes, to a larger computational cost.

A natural condition for (9.2) to converge to an approximate inverse matrix is as follows:

$$\mathcal{P}(A^{-1}) = A^{-1} + E, \quad ||E|| \leqslant \varepsilon.$$

In this case we can prove that (9.2) converges (locally) to an approximate inverse matrix with a quadratic speed until $||X_k - A^{-1}|| > c\varepsilon$ for some $c > 1$. This is a special case of a more general result for approximate iterations [13].

Every iteration of the form (9.2) requires two matrix-by-matrix multiplications, AX_k and $X_k(AX_k)$. If we do not compress AX_k, than there will be r^3 multiplications of n-by-n in each iteration.

A modified version proposed in [21] reduces the number of required multiplications via rewriting (9.2) in a better "structure-friendly" format. Instead of a single sequence of matrices X_k we build up two sequences, X_k and Y_k, which are iterated in the following way:

$$\begin{aligned} Y_{k+1} &= Y_k(2I - Y_k), \\ X_{k+1} &= X_k(2I - Y_k), \end{aligned} \tag{9.3}$$

with X_0 being an initial guess for A^{-1} and $Y_0 = AX_0$. The equivalence of (9.3) and (9.2) in exact arithmetics stems from the fact that

$$\begin{aligned} Y_{k+1} &= Y_k H_k, \\ X_{k+1} &= X_k H_k, \\ H_k &= 2I - Y_k, \end{aligned} \tag{9.4}$$

implying that

$$X_{k+1}Y_{k+1}^{-1} = X_k H_k H_k^{-1} Y_k^{-1} = X_k Y_k^{-1} = \ldots = X_0 Y_0^{-1} = X_0 X_0^{-1} A^{-1} = A^{-1}.$$

Since the first equation in (9.3) is just the Newton method applied to the identity matrix, we conclude that $Y_{k+1} \to I$ and $X_{k+1} \to A^{-1}$. Note that $X_{k+1}Y_{k+1}^{-1}$ remains the same for any nonsingular H_k, and in pursuit of reduction of the number of multiplications we can substitute H_k with a *very crude* approximation $\mathcal{P}(H_k)$. Finally, the modified Newton method reads

$$\begin{aligned} H_k &= \mathcal{P}(2I - Y_k), \\ Y_{k+1} &= \mathcal{P}_1(Y_k H_k), \\ X_{k+1} &= \mathcal{P}_2(X_k H_k), \end{aligned} \tag{9.5}$$

where projectors $\mathcal{P}_1, \mathcal{P}_2$ ought to maintain the required accuracy and $\mathcal{P}$ can be a way less accurate. If X_0 is close enough to A^{-1} then Y_k is close to I and H_k is close to the identity matrix, the one with a "perfect structure". Experiments confirm that the modified Newton method is faster and more accurate than the standard Newton method.

For application of the inversion procedure (9.5) with a projection through recompression by Algorithm 5.1 we consider two typical examples:

(A) A discrete 3D Laplacian

$$A = \triangle \times I \times I + I \times \triangle \times I + I \times I \times \triangle,$$

where $\triangle = \text{tridiag}(-1, 2, -1)$.

(B) A Newton potentional matrix coming from discretization of the volume integral operator

$$Ku = \int_{[0,1]^3} \frac{u(x,y)}{||x-y||}\, dx\, dy$$

by the collocation using piecewise-constant functions on a uniform grid of size $n \times n \times n$. The matrix never appears as it is, and is replaced in the very beginning by its tensor approximation in the Tucker format computed by the Cross 3D algorithm proposed in [20], the accuracy being set to 10^{-5}.

Mode ranks for the Laplacian are equal to 2. Computed values of mode ranks (appeared to be equal for all modes: $r_1 = r_2 = r_3 = r$) for tensor approximations to the Newton potential matrix are collected in Table 9.1.

$N = n^3$	32^3	64^3	128^3	256^3
r	10	12	13	15

Table 9.1: Mode ranks for the Newton potential matrix, $\varepsilon = 10^{-5}$

$N = n^3$	32^3	64^3	128^3	256^3
Time	9 sec	11 sec	24 sec	227 sec
$\|\|AX - I\|\|_F / \|\|I\|\|_F$	$3 \cdot 10^{-6}$	$4 \cdot 10^{-6}$	$2 \cdot 10^{-5}$	10^{-4}

Table 9.2: Inversion timings for the 3D Laplacian

$N = n^3$	32^3	64^3	128^3	256^3
Time	60 sec	107 sec	360 sec	1574 sec
$\|\|AX - I\|\|_F / \|\|I\|\|_F$	10^{-2}	$9 \cdot 10^{-3}$	$5 \cdot 10^{-2}$	$4 \cdot 10^{-2}$

Table 9.3: Inversion timings for the 3D Newton potential matrix

For the Laplacian, projectors in (9.5) are chosen as follows:

$$\mathcal{P} = \mathcal{P}_{(2,2,2)}, \quad \mathcal{P}_1 = \mathcal{P}_2 = \mathcal{P}_{(12,12,12)},$$

where $\mathcal{P}_{(r_1 r_2 r_3)}$, designates the projector onto tensors with mode ranks (r_1, r_2, r_3). For the Newton potential matrix, to cause the method to converge we had to select

$$\mathcal{P} = \mathcal{P}_1 = \mathcal{P}_2 = \mathcal{P}_{(10,10,10)}.$$

Maybe in this case it is worthy to use projectors which preserve the accuracy, not the ranks — this is a subject of a future research. As initial guess we take αI for some small α, it is possible since both of these matrices are positive definite. Errors are measured through the ratio $||AX - I||_F/||I||_F$. The timins are presented in Tables 9.2 and 9.3.

All experiments were conducted in Matlab with the help of TensorToolbox [1]. Since all computations are BLAS-based, improvement in the actual speed when passing to a high-level programming language would not be very pronounced. We observe a somewhat irregular behavior of computational time because the asymptotically hardest pre-recompresison step of complexity ($\mathcal{O}(n^3r^2)$) is actually not like that for small values of n. The Laplacian inversion costs are $5 \div 10$ times smaller thanks to a lower rank projector $\mathcal{P}$. For the volume integral operator case, the same choice makes the method diverge, so we have to use a more accurate projector.

These two examples, being expository, still show very clear that the Tucker format is well-suited for structured iterations and allows one to compute approximate inverse matrices (hence, excellent preconditioners) in several minutes on an ordinary notebook. Actual times can be improved a lot by introducing an additional structure in the factors, for example, by using sparse representations in the wavelet basis or Toeplitz-like structures [3, 21, 18]. On this way we can eventually arrive at superfast algorithms capable to work with mode sizes of order $n \geq 1000$ and, hence, matrices of order $N \geq 1000^3$. That is not a subject of the current paper, and the related results will be reported elsewhere.

10. Conclusions and future work. In this paper we have presented methods for computing basic linear algebra operations in the Tucker format and showed that it keeps the same order complexity as the combination of the canonocal and Tucker formats but excels in exploiting well-established matrix tools based on the SVD (which are not available for the canonical format). Moreover, the number of defining parameters in the Tucker representation can be even smaller than in the canonical decomposition, because mode ranks are often smaller than the canonical rank of a tensor. Thus, for three-dimensional problems, when the size of the Tucker core array is next to negligible, the use of the Tucker format is highly recommended for both operands.

The price for computation of the basic operations is chiefly affected by computation of multi-fold sums which look very complicated. However, we have shown that these sums can be computed in a fast way and reduced to matrix-by-matrix multiplications. Thus, highly-optimized BLAS-3 software can be used that is available on almost every modern computer. In practice, the cost beyond BLAS-3 calls is under 1 percent of the computational time.

Still, there is a lot of future work to be done. Some issues to be investigated include:

- Additional structure for matrix factors (wavelet sparsification, Toeplitz-like structure, circulant-plus-low rank structure).

- Iterative methods for matrix functions (sign functions, square root, matrix exponential).

- Iterative methods for linear systems with structured vectors (PCG,BiCGStab) and preconditioners computed by the Newton method.

- Eigenvalue solvers with tensor-structured eigenvectors.

Last not the least, we encounter a challenge topic for the linear algebra and matrix analysis community:

Which properties of matrices do account for a Tucker approximation with low mode ranks for the inverses, matrix exponentials and other matrix functions?

As yet, all experiments witness in favour of the following hypothesis: for all practical operators of mathematical physics (i.e. differential operators, integral operators with smooth, singular and hypersingular kernels) on tensor grids, standard matrix functions can be approximated in the Tucker format with ranks of order

$$r \sim \log^{\alpha} n \log^{\beta} \varepsilon^{-1},$$

with some constants α, β.

The known theorems on tensor structure of the matrix functions rely on analytical considerations. In [22] an attempt was made to tackle this problem from the matrix point of view. However, some results were obtained only in a special two-dimensional case. The extension to three dimensions is underway.

11. Acknowledgements. This work was partially supported by Russian Fund of Basic Research (grant 08-01-00115), DFG and a Priority Research Grant of the Department of Mathematical Sciences of the Russian Academy of Sciences. The authors acknowledge a fruitful collaboration with colleagues at Max-Planck Institute in Leipzig and Technical University in Berlin. The paper was completed when the third author had a visiting professor position at the Baptist University of Hong Kong.

REFERENCES

[1] B.W. Bader and T.G. Kolda. Algorithm 862: MATLAB Tensor Classes for Fast Algorithm Prototyping. *ACM Trans. on Math. Soft.*, 32(4), 2006.

[2] B.W. Bader and T.G. Kolda. Tensor Decompositions and Applications. Technical Report Number SAND2007-6702, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2007.

[3] G. Beylkin and M. J. Mohlenkamp. Numerical operator calculus in higher dimensions. *Proc. Natl. Acad. Sci. USA*, 99(16):10246–10251, 2002.

[4] R. Bro. PARAFAC: Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 38(2):149–171, 1997.

[5] J.D. Caroll and J.J Chang. Analysis of individual differences in multidimensional scaling via n-way generalization of Eckart-Young decomposition. *Psychometrica*, 35:283–319, 1997.

[6] P. Comon. Tensor decomposition: State of the Art and Applications. IMA Conf. Math. in Signal Proc., Warwick, UK, Dec. 18-20, 2000.

[7] H.-J. Flad, B.N. Khoromskij, D.V. Savostyanov, and E.E. Tyrtyshnikov. Verification of the cross 3D algorithm on quantum chemistry data. *Rus. J. Numer. Anal. Math. Model.*, 23(4), 2008.

[8] J.M. Ford and E.E. Tyrtyshnikov. Combining Kronecker product approximation with discrete wavelet transforms to solve dense, function-related systems. *SIAM J. Sci. Comp.*, 25(3):961–981, 2003.

[9] S.A. Goreinov. On cross approximation of multi-index array. *Doklady Math.*, 420(4):1–3, 2008.

[10] S.A. Goreinov and E.E. Tyrtyshnikov. The maximal-volume concept in approximation by low-rank matrices. *Contemporary Mathematics*, 208:47–51, 2001.

[11] S.A. Goreinov, E.E. Tyrtyshnikov, and N.L. Zamarashkin. A theory of pseudo–skeleton approximations. *Linear Algebra Appl.*, 261:1–21, 1997.

[12] W. Hackbush, B.N. Khoromskij, and E.E. Tyrtyshnikov. Hierarchical Kronecker tensor-product approximations. *J. Numer. Math.*

[13] W. Hackbush, B.N. Khoromskij, and E.E. Tyrtyshnikov. Approximate iteration for structured matrices. *Numer. Math*, 2008.

[14] R.A. Harshman. Foundations of the Parafac procedure: models and conditions for an explanatory multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[15] L. De Lathauwer, B. De Moor, and J. Vandewalle. On best rank-1 and rank-$(R_1, R_2, ..., R_N)$ approximation of high-order tensors. *SIAM J. Matrix Anal. Appl.*, 21:1324–1342.

[16] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multlinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21:1253–1278, 2000.

[17] L. De Lathauwer, B. De Moor, and J. Vandewalle. Computing of Canonical decomposition by means of a simultaneous generalized Schur decomposition. *SIAM J. Matrix Anal. Appl.*, 26:295–327, 2004.

[18] V. Olshevsky, I.V. Oseledets, and E.E. Tyrtyshnikov. Superfast inversion of two-level Toeplitz matrices using Newton iteration and tensor-displacement structure. *Operator Theory: Advances and Applications*, 179:229–240, 2008.

[19] I.V. Oseledets, D.V. Savostyanov, and E.E. Tyrtyshnikov. Fast simultaneous orthogonal reduction to triangular matrices. *SIAM J. Matrix Anal. Appl.*, page to appear, 2008.

[20] I.V. Oseledets, D.V. Savostyanov, and E.E. Tyrtyshnikov. Tucker dimensionality reduction of three-dimensional arrays in linear time. *SIAM J. Matrix Anal. Appl.*, page to appear, 2008.

[21] I.V. Oseledets and E.E. Tyrtyshnikov. Approximate inversion of matrices in the process of solving a hypersingular integral equation. *Comp. Math. and Math. Phys.*, 45(2):302–313, 2005.

[22] I.V. Oseledets, E.E. Tyrtyshnikov, and N.L. Zamarashkin. Matrix inversion cases with size-independent tensor rank estimates. *Linear Algebra Appl.*, submitted, 2008.

[23] L.R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.