Solution of linear systems and matrix inversion in the TT-format

(revised version: May 2011)

by

Sergey Dolgov, and Ivan V. Oseledets

Preprint no.: 19 2011

Solution of linear systems and matrix inversion in the TT-format

S. V. Dolgov and I. V. Oseledets

Institute of Numerical Mathematics, Russian Academy of Sciences, Gubkina Street, 8, Moscow, Russia

sergey.v.dolgov@gmail.com, ivan.oseledets@gmail.com, http://spring.inm.ras.ru/osel

Abstract. Tensors arise naturally in high-dimensional problems in chemistry, financial mathematics and many others. The numerical treatment of such kind of problems is difficult due to the curse of dimensionality: the number of unknowns and computational complexity grows exponentially with the dimension of the problem. To break the curse of dimensionality, low-parametric representations, or *formats* have to be used. In this paper we make use of the TT-format which is one of the most effective stable representations of high-dimensional tensors. Basic linear algebra operations in the TT-format are now well-developed. Our goal is to provide a "black-box"-type solver for linear systems where both the matrix and the right-hand side are in the TT-format. An efficient DMRG (Density Matrix Renormalization Group) method is proposed, and several tricks are employed to make it work. The numerical experiments confirm the effectiveness of our approach.

1. Introduction

Tensors arise naturally in high-dimensional problems, for example, in quantum chemistry [1, 2], financial mathematics [3, 4] and many others. The treatment of d-dimensional tensors is notoriosly difficult due to the *curse of dimensionality*: the number of elements of a tensor grows exponentially with the number of dimensions d, and so does the complexity to work with fully populated tensors. As in the matrix case, one can rely on the sparsity of tensors [5]. However, this helps only for small d. For d of order tens or hundrends other approaches are needed, and special low-parametric representations or *formats*, are requred. Several formats have been proposed to represent a tensor in a data-sparse way. They include canonical and Tucker formats, the two formats with well-established properties and application areas, see the review [6] for more details. They have known drawbacks. To avoid these drawbacks, the development of new tensor formats began. In 2009 independently Hackbusch and Kuhn and later Grasedyck [7, 8] and Oseledets and Tyrtyshnikov [9] proposed two (slightly different) hierarchical schemes for the tensor approximation, $\mathcal{H}$-Tucker and Tree Tucker formats. These formats depend on specially chosen *dimension trees* and require recursive procedures. To avoid the recursion, it was proposed to use a simple matrix product form of the decomposition [10, 11], that was called the *Tensor Train format*, or simply the TT-format.

[$]This work was supported in part by RFBR grants 09-01-12058, 10-01-00757, 11-01-00549, RFBR/DFG grant 09-01-91332, Russian Federation Gov. contracts No. П1178, П1112 and П940, 14.740.11.0345 and Promotion-stipendium of Max-Planck Gesellschaft. Part of this work was done during the stay of authors in Max-Plank Institute for Mathematics in Sciences, Leipzig, Germany.

A tensor $\mathbf{A}$ is said to be in the TT-format, if its elements are defined by formula

$$A(i_1, \dots, i_d) = G_1(i_1) \dots G_d(i_d), \tag{1.1}$$

where $G_k(i_k)$ is an $r_{k-1} \times r_k$ matrix for each fixed $i_k, 1 \leqslant i_k \leqslant n_k$. To make the matrix-by-matrix product in (1.1) a scalar, boundary conditions $r_0 = r_d = 1$ have to be imposed. The numbers r_k are called *TT-ranks* and $G_k(i_k)$ — cores of the TT-decomposition of a given tensor. If $r_k \leqslant r, n_k \leqslant n$, then the storage of the TT-representation requires $\leqslant dnr^2$ memory cells. If r is small, then this is much smaller than the storage of the full array, n^d.

The TT-format was introduced in [10, 11] as an alternative to two commonly used formats: the canonical format and the Tucker format. These two formats can be considered as different generalizations of the singular value decomposition (SVD) from matrices (i.e., $d = 2$) to higher order tensors. The tensor is said to be in the canonical format, if

$$A(i_1, \dots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) \dots U_d(i_d, \alpha).$$

This representation is often referred to as *CANDECOMP/PAFAFAC*, or simply the *CP model*. If r, the canonical rank, is small, then the number of parameters depends on d linearly. The Tucker model [12, 13] is the representation of form

$$A(i_1, \dots, i_d) = \sum_{\alpha_1, \dots, \alpha_d} G(\alpha_1, \dots, \alpha_d) U_1(i_1, \alpha_1) \dots U_d(i_d, \alpha_d),$$

where U_k are called *Tucker factors*, and the tensor $G(\alpha_1, \dots, \alpha_d)$ — *Tucker core*. The canonical format is a good candidate for low-parametric representation of tensors. It can be used to approximate high-dimensional operators and their inverses (see, for example, [14, 15, 16, 17, 18]) using certain analytical expansions. However, the canonical format has a serious drawback: there are no robust algorithms to compute the canonical approximation numerically. There are efficient algorithms for computing a low-rank approximation of a given tensor [19, 20, 21], but they are not robust. The best approximation problem in the canonical format can be ill-posed [22] and the best approximation may not even exist.

In two dimensions the CP decomposition reduces to the skeleton (dyadic) decomposition of a matrix, which can be computed via the SVD. The SVD is stable and efficient. These properties are lost for tensors of order $d \geqslant 3$. The Tucker decomposition is stable and an optimal approximation always exists, and a quasioptimal approximation can be computed via the Higher Order SVD (HOSVD) [13, 23]. However, it has exponential in d number of parameters, thus it can be applied only for small d (for the three-dimensional case $d = 3$ see Cross-3D method [24], and about application of the Tucker model to the solution of Hartree-Fock equation see [25]).

The TT-format is in the middle between CP and Tucker formats: it does not have an instinsic exponential dependence on d, but it is stable in the sense that the best approximation with bounded TT-ranks always exists and a quasioptimal approximation can be computed by a sequence of SVDs of auxiliary matrices [10, 11]. Thus it has a great potential for the stable and robust approximation of high-dimensional tensors.

The TT-format comes with all basic linear algebra operations. Addition, matrix-by-vector product, elementwise multiplication can be implemented in linear d and polynomial in r complexity with the result also in the TT-format [10, 11]. The problem is that after such operations TT-ranks are growing. For example, the TT-ranks of the sum of two tensors are equal (formally) to the sum of the TT-ranks of the addends. In the case of the matrix-by-vector product the result is also in the TT-format with the TT-ranks of matrix and vector multiplied. After several iterations, the TT-ranks will become too large, thus *rounding* is needed: a given tensor $\mathbf{A}$ is approximated by another tensor $\mathbf{B}$ with minimal possible TT-ranks with a prescribed accuracy ε:

$$||\mathbf{A} - \mathbf{B}||_F \leqslant \varepsilon ||\mathbf{A}||_F.$$

The rounding procedure in the TT-format can be implemented in $\mathcal{O}(dnr^3)$ operations [10, 11]. Using fast linear algebra with rounding in the TT-format, iterative solvers can be implemented. However, they can be rather slow due to the large number of iterations, thus a certain preconditioner is needed.

Our main goal in this paper is to present a "black-box" solver for the linear system with both the matrix and the right-hand side in the TT-format. Of course, the solution is almost always possible only approximately, thus the required accuracy of the result should be provided to the solver. In order to design such a solver, one has to step aside from the Krylov-type approaches and look at the special structure of the matrix and the solution vector involved.

Instead of utilizing common solvers (CG, GMRES) based on the Krylov subspaces with vectors and matrices stored in the structured form, the solution is sought in the TT-format *directly*. In order to do this, equations for defining parameters (cores) have to be derived. This can be achieved by reformulating the initial linear system

$$Ax = f,$$

as a minimization problem. The simplest functional is

$$||Ax - f|| \to \min \tag{1.2}$$

for $x \in \mathcal{S}$ where $\mathcal{S}$ is a certain class of structured solutions. In our case, x is associated with a tensor $X(i_1, \dots, i_d)$ with small TT-ranks. Then, (1.2) becomes a nonquadratic minimization problem, which has to be solved by a certain minimization method. For the symmetric positive definite case, instead of minimizing (1.2) one can solve

$$(Ax, x) - 2(f, x) \to \min, x \in \mathcal{S} \tag{1.3}$$

For the simplicity only the second case (1.3) will be considered, but it the obtained method is applicable also to the case of general A.

How to solve the minimization problem? It is a non-linear minimization problem in unknowns $G_k(i_k)$. It can be solved by any suitable general-purpose minimization method (Newton method, conjugate gradient). This approach may work, but it avoids the very specific structure of the solution and thus, of the functional.

The simplest method which makes use of this structure is an *alternating least squares method*. If all cores except one are fixed, the problem becomes a quadratic problem and can be reduced to a linear system with nr^2 unknowns. Then one can minimize over the next core and so on, up to d. This approach guarantees that the value of the functional will not increase at each iteration step. This method, however, requires the knowledge of all TT-ranks. There are $d-1$ TT-ranks, and if they are underestimated, the solution will not be close to the true solution. If they are overestimated, then the complexity may be too high. Usually, one can specify the accuracy ε, to which the solution is sought, and ALS is non-adaptive in the sence that it requires all TT-ranks to be known in advance. The second problem is that the convergence of the ALS approach may be too slow. How to modify the ALS method to avoid these problems?

The idea of modifying the ALS scheme was brought to the field of numerical analysis in the paper [26] by S. Holtz, T. Rohwedder, R. Schneider, and it was also used to construct efficient eigenvalue solvers for Hamiltonians in quantum molecular dynamics [27].

The idea of the modified ALS was first prosed in the solid state physics for the computation of the minimal eigenvalue of quantum spin Hamiltonians. It is the so-called DMRG (Density Matrix Renormalization Group) method by S. White [28]. A quantum spin system consists of K *spins*, and such system is defined by a wavefunction with S^K parameters, where typically $S = 2$ or $S = 4$. This wavefunction can be considered as a K-dimensional tensor. The computation of the minimal eigenvalue can be reformulated as a problem of the minimization of the Rayleigh quotient

$$(H\psi, \psi), \quad ||\psi|| = 1,$$

(compare with (1.3), which is very similar), and the solution is sought as a matrix product state [29], which is equivalent to the TT-format. The DMRG is an alternating least squares method, but with one minor modification. Instead of minimizing over one core, the functional is minimized over a pair of cores $G_k(i_k), G_{k+1}(i_{k+1})$. This problem still can not be reduced to a linear eigenvalue problem, thus a *supercore* W is introduced by contracting over α_k:

$$W(i_k, i_{k+1}) = G_k(i_k)G_{k+1}(i_{k+1}). \tag{1.4}$$

The minimization problem in W is now quadratic, and can be reduced to a linear eigenvalue problem with S^2r^2 unknowns. After W is computed, the TT-structure can be recovered by computing the decomposition (1.4), which can be done by the SVD, and the rank r_k is determined *adaptively*. This step is called the *decimation step*. However, if A comes from the discretization of a high-dimensional equation, the one-dimensional mode size n is usually not small, and the size of the local problem is much larger. In this case, the QTT-format (*quantized TT-format* [1]) can be very helpful.

Idea behind the QTT-format is the following. Consider a vector of values of a univariate function on a grid with 2^d points, i.e. $n = 2^d$. By the binary coding this vector can be considered as a d-dimensional tensor with mode sizes 2. In combination with the TT-format, this yields the *QTT-format* for representing vectors [30, 31]. This transformation of vectors into tensors, "tensorization", can be generalized to an arbitrary dimension. If a function of K variables is discretized on a tensor grid with 2^d points in each direction, then the corresponding K-dimensional tensor can be reshaped into a dk-dimensional tensor with small mode sizes. This leads to an algebraically similar problem with small mode sizes as for the quantum spin Hamiltonian, and the size of the local DMRG problem is only twice larger, then for the traditional ALS step.

In this paper we make one step further (especially compared to the work [26]) to the efficient realistic black-box solver of linear systems in the TT-format. A direct implementation encounters serious difficulties, which have to be tackled. Several "tricks" will be proposed which can be considered as technical, but only them make the method work.

Local problems have $4r^2$ unknowns, thus for $r \geqslant 25$ the size of the local matrix exceeds 10000, which is too large. For the solution of linear systems the TT-ranks can be much larger, for example, $50 - 100$. This can be called *the curse of the rank*: the dependence from the TT-ranks is polynomial, and prohibits taking very large ranks. Thus, iterative methods should be used to solve each local problem. These problems can be also ill-conditioned, and certain preconditioning may be necessary.

[1]Originally the name was *Quantics TT-format*, but now it is clear that "Quantized" is a more appropriate term

The second crucial problem is in the decimation step. The truncation via the SVD guarantees an approximation in the Frobenius norm. However, if a matrix A is ill-conditioned (for example, it is a discretization of the differencial operator), then a small error in x may lead to a large error in the residue $||Ax - f||$ (which can be bounded only by $\mathrm{cond}(A)\varepsilon$) and a much smaller threshold ε should be used. Using a very small threshold leads to increased ranks and increased complexity, which should be avoided. A simple truncation scheme will be proposed, which solves this problem.

The third problem is that for certain problems the DMRG method converges to a *local minimum*, i.e. all local systems are reported to be solved up to the required accuracy, but the solution obtained is not an adequate solution of the initial linear system. This can be avoided by using an additional "random restart" in between of the DMRG-sweeps.

Another important problem considered in this paper is the approximate matrix inversion. It is interesting that the inversion can be performed using the same linear system solver applied to an auxiliary system, and the complexity is only slightly larger than the complexity to solve a single linear system. Moreover since the inversion with a very high accuracy is not always neccessary, computations with low accuracy may provide an excellent preconditioner!

The reduction of the inversion problem to a linear system is simple: Instead of solving

$$AX = I, \tag{1.5}$$

we solve

$$(A \otimes I)\mathrm{vec}(X) = \mathrm{vec}(I), \tag{1.6}$$

where $\mathrm{vec}(X)$ is in fact a $4 \times 4 \times \ldots \times 4$ d-dimensional tensor, and the right-hand side has rank 1, i.e. has a perfect structure. The matrix of the system (1.6), as it will be shown, has bounded TT-ranks, if A has bounded TT-ranks. Thus, a linear solver can be applied.

2. Introduction to notation used

In this section several basic facts and definitions are recollected. When dealing with the TT-representations it is convenient to work with parameter-dependent matrices, which are denoted by, for example, $G_k(i_k)$. The size of these matrices should be clear from the context (i.e, $r_{k-1} \times r_k$). The parameter-dependent matrices can be multiplied:

$$W(i_k, i_{k+1}) = G_k(i_k)G_{k+1}(i_{k+1}),$$

yielding a new matrix depending on a larger number of parameters. In this representation it is important to look at the order of the elements of the product. The row and column indices of a parameter-dependent matrix can be considered as parameters. For example, for an $r_{k-1} \times r_k$ parameter-dependent matrix $G_k(i_k)$ its elements (being scalars) are denoted by $G_k(\alpha_{k-1}, i_k, \alpha_k)$. Again, the actual size of the parameter-dependent matrix is defined by the context. For example, if $G_k(i_k)$ is an $r_{k-1} \times r_k$ matrix, then $G_k(\alpha_{k-1}, i_k)$ defines a row-vector of length r_k for each fixed α_{k-1} and i_k. In this form, the TT-format is represented as

$$A(i_1, \ldots, i_d) = G_1(i_1)G_2(i_2)\ldots G_d(i_d). \tag{2.1}$$

Vectors that are solutions to linear systems, considered in this paper, are in fact tensors. By a vector in the TT-format we mean, that it has length $N = n_1 \cdot \ldots \cdot n_d$ and it can be considered as an $n_1 \times \ldots \times n_d$ tensor which has low TT-ranks. Square matrices acting on such vectors have size $N \times N$ and their elements can be naturally indexed by a $2d$-tuple $(i_1, \dots, i_d, j_1, \dots, j_d)$, $1 \leqslant i_k, j_k \leqslant n_k$, i.e. $M(i_1, \dots, i_d, j_1, \dots, j_d)$. A matrix M is said to be in the TT-format, if

$$M(i_1, \dots, i_d, j_1, \dots, j_d) = M_1(i_1, j_1) M_2(i_2, j_2) \dots M_d(i_d, j_d), \tag{2.2}$$

where $M_k(i_k, j_k)$ is an $r_{k-1} \times r_k$ matrix, and $r_0 = r_d = 1$. If all TT-ranks are equal to 1, then (2.2) reduces to

$$M = M_1 \otimes \dots \otimes M_d,$$

i.e. it has Kronecker rank 1. Thus, (2.2) is a generalization of a standard low-rank approximation of high-dimensional operators [32, 14].

The TT-representation (2.1) is non-unique, since it is invariant under the transformation

$$G'_k(i_k) := G_k(i_k) S, \quad G'_{k+1}(i_{k+1}) := S^{-1} G_{k+1}(i_{k+1})$$

for any nonsingular matrix S. Using such transformations, certain cores of the TT-representation can be made *orthogonal*. There are two types of orthogonality of cores: left-orthogonality and right-orthogonality. A core $G_k(i_k)$ is said to be left-orthogonal, if

$$\sum_{i_k} G_k(i_k) G_k^{\top}(i_k) = I_{r_{k-1}},$$

and right-orthogonal, if

$$\sum_{i_k} G_k^{\top}(i_k) G_k(i_k) = I_{r_k}.$$

Another interpretation of the left-orthogonality is that the tensor $G_k(\alpha_{k-1}, i_k, \alpha_k)$, considered as a matrix of size $(r_{k-1} n_k) \times r_k$, has orthonormal columns. The right-orthogonality of the core means that G_k, considered as a matrix of size $r_{k-1} \times (n_k r_k)$ has orthonormal rows. The cores $G_k(i_k), \dots G_p(i_p)$, can be made left-orthogonal by equivalent transformations. Indeed $G_k(i_k)$ can be written as

$$G_k(i_k) = Q_k(i_k) R,$$

where $Q_k(i_k)$ is left-orthogonal by applying the QR-decomposition to a $(r_{k-1} n_k) \times r_k$ matrix, obtained from $G_k(i_k)$ by reshaping. The corresponding matrix R is constant and can be incorporated into the next core. Good news are that if the cores $G_1(i_1) \dots G_k(i_k)$ are left-orthogonal, their product

$$Q(i_1, \dots, i_k) = G_1(i_1) \dots G_k(i_k),$$

considered as an $N_k \times r_k$ matrix has orthonormal columns ($N_k = \prod_{s=1}^{k} n_s$). The proof can be found in [11].

3. DMRG scheme for linear systems

3.1. Basic idea

First let us focus on the DMRG scheme applied to the minimization of the functional

$$(Ax, x) - 2(f, x) \tag{3.1}$$

which is equivalent to the solution of a linear system

$$Ax = f,$$

for the symmetric and positive definite matrix A. If x is in the TT-format, then the DMRG algorithm is obtained by minimizing (3.1) over a supercore $W(i_k, i_{k+1}) = G_k(i_k)G_{k+1}(i_{k+1})$. In fact, this step is equivalent to an ALS step applied to a $(d-1)$-dimensional tensor of size $n_1 \times \ldots \times n_{k-1} \times (n_k n_{k+1}) \times \ldots \times n_d$, i.e. with modes $k, (k+1)$ treated as a one long mode of size $n_k n_{k+1}$. Thus, it is sufficient to derive a formula for one iteration step of the ALS method (and use it for the modified tensor).

There are several ways to derive an expression for the local subproblem, for example in [27] compact representations of matrix-by-vector and scalar products were used, or [26], where a diagrammatic representation was utilized.

Here slightly different derivation will be given, which will show that the DMRG (or ALS) is in fact a *projection method* on adaptively chosen subspaces. Moreover, an orthogonal basis can be selected in these subspaces.

Suppose that the k-th core is varied (and all others are fixed). The linear system

$$Ax = f$$

can be treated as an overdetermined linear system of form

$$\sum_{j=1}^{n_k} A(i_k, j_k)x(j_k) = f(i_k), \quad 1 \leqslant i_k, j_k \leqslant n_k,$$

where $A(i_k, j_k)$ is a "big" matrix of size $N_k \times N_k$, where $N_k = \prod_{s=1, s\neq k} n_s$ for each fixed (i_k, j_k), and $x(j_k), f(i_k)$ are vectors of length N_k. The requirement that all cores except G_k are fixed leads to the representation

$$x(j_k) = Qw(j_k), \tag{3.2}$$

where Q is an $N_k \times r_{k-1}r_k$ matrix. By equivalent transformations of the TT-representation of $\mathbf{X}$ the matrix Q can be made orthogonal. Indeed, it can be treated as a tensor with elements

$$Q(i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_d, \alpha_{k-1}, \alpha_k) = G_1(i_1)\ldots G_{k-1}(i_k, \alpha_{k-1})G_{k+1}(\alpha_{k+1}, i_{k+1})\ldots G_d(i_d).$$

If the cores $G_s(i_s), s = 1, \ldots, k-1$ are orthogonalized from the left, and the cores $G_s(i_s), s = k+1, \ldots, d$ are orthogonalized from the right, then the matrix Q will have orthonormal columns.

For a symmetric positive definite case, the minimization of the functional

$$(Ax, x) - 2(f, x) = \sum_{i_k, j_k} A(i_k, j_k)x(i_k)x(j_k) - \sum_{j_k} f(j_k)x(j_k),$$

under the restriction (3.2) leads to a smaller linear system of form

$$\sum_{j_k} (Q^\top A(i_k, j_k)Q)w(j_k) = Q^\top f(i_k). \tag{3.3}$$

The matrices $B(i_k, j_k) = Q^\top A(i_k, j_k)Q$ are of size $(r_{k-1}r_k) \times (r_{k-1}r_k)$ for each fixed pair i_k, j_k, and the full matrix B of linear system (3.3) has size $(r_{k-1}r_k n_k) \times (n_k r_{k-1} r_k)$.

An important observation is that even if A is indefinite or even is non-symmetric, the equations (3.3) can be used, but now it is not a minimization method, but a projection method on a subspace defined by the orthogonal matrix Q. In this setting it is unclear why this method should converge, but our numerical experiments show that in several cases it is a good choice.The matrices $B(i_k, j_k)$ can be computed from the TT-representations of A and X. At each step of the sweep one should solve a linear system with the matrix B. This matrix is small, compared to the full matrix, however its actual size can be quite large, thus special techniques are required to form this matrix and to solve the linear system with it. It appears that it might be more economical not to form the matrix B directly, but solve even the local linear system with the help of some iterative solver.

3.2. Fast computation in local problems

The matrix $B(i_k, j_k)$ of the system (3.3) can be treated as an array with elements $B(\beta_{k-1}, \beta_k, i_k, j_k, \gamma_{k-1}, \gamma_k)$, where $\beta_{k-1}, \gamma_{k-1}$ vary from 1 to r_{k-1}, and β_k, γ_k — from 1 to r_k, and these elements are defined via a "grand summation"

$$\begin{aligned} & B(\beta_{k-1}, \beta_k, i_k, j_k, \gamma_{k-1}, \gamma_k) = \\ = \ & \textstyle\sum_{i_s, s \neq k} \sum_{j_l, l \neq k} A(i_1, \dots, i_d, j_1, \dots, j_d) \quad Q(i_1, \dots, i_{k-1}, i_{k+1}, \dots, i_d, \beta_{k-1}, \beta_k) \times \\ & \qquad\qquad\qquad\qquad\qquad\qquad \times Q(j_1, \dots, j_{k-1}, j_{k+1}, \dots, j_d, \gamma_{k-1}\gamma_k). \end{aligned}$$

Using the TT-representations of A and Q, the following formula for B is obtained:

$$B(\beta_{k-1}, \beta_k, i_k, j_k, \gamma_{k-1}, \gamma_k) = \Psi_{k-1}(\beta_{k-1}, \gamma_{k-1}) A_k(i_k, j_k) \Phi_k(\beta_k, \gamma_k), \tag{3.4}$$

where $\Psi_{k-1}(\beta_{k-1}, \gamma_{k-1})$ is in fact a row vector of length R_{k-1}, and $\Phi_{k-1}(\beta_k, \gamma_k)$ is a column vector of length R_k, where R_k are the QTT-ranks of the matrix A. $\Psi_s, s = 1, \dots, k-1$ can be computed iteratively starting from $\Psi_0 = 1$ by a recursion

$$\Psi_s(\beta_s, \gamma_s) = \sum_{i_s, j_s, \beta_{s-1}, \gamma_{s-1}} \Psi_{s-1}(\beta_{s-1}, \gamma_{s-1}) A_s(i_s, j_s) X_s(\beta_{s-1}, i_s, \beta_s) X_s(\gamma_{s-1}, j_s, \gamma_s), \tag{3.5}$$

Analogous representation can be obtained for $\Phi_s, s = k, \dots, d$. Let us estimate the cost of computing Ψ_k using (3.5). If $\Psi_{s-1}(\beta_{s-1}, \gamma_{s-1})$ is reshaped into a "long" vector ψ_{s-1} of length $r_{s-1}^2 R_{s-1}$, and Ψ_s is reshaped into a "long" vector ψ_s of length $r_s^2 R_s$ the expression for Ψ_s can be rewritten in a compact form (see also [27]):

$$\psi_s = \psi_{s-1} \sum_{i_k, j_k} A_s(i_s, j_s) \otimes X_s(i_s) \otimes X_s(j_s).$$

The computation of ψ_s when ψ_{s-1} is known reduces to n_s^2 products of a Kronecker rank-1 matrix by a full vector. For each fixed i_s, j_s the cost is

$$\mathcal{O}(r_s r_{s-1}^2 R_{s-1} + r_s^2 r_{s-1} R_{s-1} + r_s^2 R_s).$$

If $r_s \approx r$ and $R_s \approx R$, then the total cost for computing Φ and Ψ is

$$\mathcal{O}(dn^2 r^2 R^2 + dn^2 r^3 R).$$

Often $r \gg R$ and in this case the complexity is linear in the matrix rank and cubic in the solution rank.

The matrix B in (3.4) has size $n^2 r^2 \times n^2 r^2$. Even for $n = 2$ its size becomes large for $r \geqslant 25$. However, it is completely defined by $\Psi_{k-1}(\beta_{k-1}, \gamma_{k-1})$, $A_k(i_k, j_k)$ and $\Phi_{k-1}(\beta_k, \gamma_k)$, which require $r_{k-1}^2 R_{k-1} + r_k^2 R_k + n^2 R_{k-1} R_k = \mathcal{O}(n^2 R^2 + r^2 R)$ parameters to store. Moreover, using this additional structure, B can be multiplied by an arbitrary vector of length $r_{k-1} n_k r_k$ fast. Indeed, it is a summation of form

$$W(\beta_{k-1}, i_k, \beta_k) = \sum_{\gamma_{k-1}, j_k, \gamma_k} B(\beta_{k-1}, \beta_k, i_k, j_k, \gamma_{k-1}, \gamma_k) Y(\gamma_{k-1}, j_k, \gamma_k).$$

Substituting the expression for B, we get

$$W(\beta_{k-1}, i_k, \beta_k) = \sum_{\gamma_{k-1}, j_k, \gamma_k} \Psi_{k-1}(\beta_{k-1}, \gamma_{k-1}) A_k(i_k, j_k) \Phi_k(\beta_k, \gamma_k) Y(\gamma_{k-1}, j_k, \gamma_k).$$

This sum is computed in three steps. First, the sum over γ_k yields:

$$\sum_{\gamma_k} \Phi_k(\beta_k, \gamma_k) Y(\gamma_{k-1}, j_k, \gamma_k) = Y'(\beta_k, \gamma_{k-1}, j_k),$$

and it can be realized as a product of a matrix of size $R_k r_k \times r_k$ by a matrix of size $r_k \times (n_k r_{k-1})$. The cost is $\mathcal{O}(nRr^3)$ operations. The contraction over j_k gives

$$Y''(\beta_k, i_k, \gamma_{k-1}) = \sum_{j_k} A_k(i_k, j_k) Y'(\beta_k, \gamma_{k-1}, j_k),$$

which is in fact a product of a matrix of size $(R_{k-1} n_k) \times (n_k R_k)$ by a matrix of size $(n_k R_k) \times (r_{k-1} r_k)$, and the cost is $\mathcal{O}(n^2 R^2 r^2)$. Finally, the summation over γ_{k-1} gives the result:

$$W(\beta_{k-1}, i_k, \beta_k) = \sum_{\gamma_{k-1}} \Psi_{k-1}(\beta_{k-1}, \gamma_{k-1}) Y''(\beta_k, i_k, \gamma_{k-1}),$$

which can be implemented via a product of a matrix of size $r_{k-1} \times (r_{k-1} R_{k-1})$ by a matrix of size $(r_{k-1} R_{k-1}) \times (n_k r_{k-1})$. The cost of computing such product is $\mathcal{O}(nRr^3)$. The total cost of computing W (i.e., the local matrix-by-vector product) is $\mathcal{O}(nRr^3 + n^2 R^2 r^2)$. If the matrix B is stored as a full matrix, the cost of its matrix-by-vector product is $\mathcal{O}(n^4 r^4)$.

4. Tricks that make it work

4.1. Trick 1: Solve large systems iteratively, small – exactly

Let us investigate, how influences the residual in a local problem on the "global" accuracy. First, from the equation (3.2) we see that $[Qw(j_k)]_{j_k=1}^{n_k}$ is nothing else but the full vector x reshaped into an $N_k \times n_k$ matrix. So, the equation (3.3) can be written as:

$$Q^\top A x = Q^\top f,$$

and Q is an orthoprojector. Now, suppose the local residual $||Q^\top A x - Q^\top f|| = \varepsilon$. Then

$$\varepsilon \leqslant ||Q^\top|| \; ||Ax - f||,$$

and, as the norm of a orthoprojector is equal to 1, we have

$$||Ax - f|| \geqslant \varepsilon.$$

Thus, the global residual can not be less than the local one. So, local systems are to be solved with the accuracy at least as expected for the global problem. Despite the fast MatVec procedure described in the previous section, unpreconditioned iterative solvers may require a lot of iterations to converge to a desired residual. If the TT-ranks of the solution are small, it is better to assemble the full matrix B in (3.4) of size $n^2 r^2 \times n^2 r^2$, and solve the local system via the direct solver, thus providing the machine precision in the local residual. Thus, the following solution strategy for the local systems is proposed:

- At first steps, when the TT-ranks are small, assemble the full local matrix and solve the local system directly.
- When the TT-ranks become large (in practice a criteria of form
$$n^2 r^2 \gtrsim 1000 \tag{4.1}$$
is used), run some iterative solver with the fast MatVec. Usually when this second step starts, the residual is already small enough (e.g. 10^{-2}) to provide sufficiently fast convergence of the iterative solver.

This trick allows to maintain a good local residual and hence the rapid convergence of the whole DMRG scheme without getting to the time-consuming cases (the last steps with large ranks for the direct solution, the first steps with slow convergence for the iterative solver). Nevertheless, preconditioning techniques have to be considered in a future work.

4.2. Trick 2: Truncation based on the residual

The ALS scheme is good, but it is unapplicable as a general-purpose solver, since it requires the knowledge of the solution ranks and moreover, even if the TT-ranks are set correctly, the convergence may be quite poor. The DMRG scheme is different. Since the supercore is optimized, the k-th TT-rank can be determined adaptively from the SVD of the supercore. However, the approximation is accurate only in the discrete analogue of the L_2-norm. If A corresponds to the discretization of a differential operator, it inevitably has a large condition number, and if $\widehat{x}$ approximates x in the Frobenius norm well:
$$||x - \widehat{x}|| \leqslant \varepsilon ||x||,$$
the residue (which is the only available accuracy measure) $||A\widehat{x} - f||$ can be large. Thus, the approximation by the SVD in the decimation step of the DMRG,
$$W(i_k, i_{k+1}) \approx \widehat{W}(i_k, i_{k+1}) = G_k(i_k) G_{k+1}(i_{k+1}),$$
can make the local residue too large. To avoid this, a simple heuristics is proposed. It is based not on the Frobenius norm of the error $||W - \widehat{W}||$, but on the approximation with rank-r which provides the local residue $B\widehat{w} - f$ not worser than the true one up to a "magic factor". An optimal low-rank approximation for the solution is a difficult optimization problem, and it is replaced by a very simple procedure: the truncation to rank r is performed by setting singular values starting from $r + 1$ to zero, but the value of the rank is chosen by a try-and-trial method to take r such that the local residue is small. This requires several additional matrix-by-vector products, but this additional cost is fully compensated by much improved convergence. Now the accuracy parameter in the DMRG procedure influences only the accuracy of the local solves.

4.3. Trick 3: Random restart

There is no guarantee that the proposed algorithm will not converge to the local minimum of the functional. Suppose, that the situation is as follows: all local residues are reported to be small, but the obtained solution is not an adequate solution to the full system. This can be checked, in principle, by computing the matrix-by-vector product Ax in the TT-format and computing the residue directly. However, a cheaper way is the following randomized check. Take a random vector z with fixed TT-ranks (say, 2) and compare the scalar products (Ax, z) and (f, z). If the difference is large, then the convergence is not obtained. In the program implementation this check can be incorporated into the "restart scheme", which proceeds as follows:

Algorithm 1: Fast MatVec procedure for the block k

```
Input: Left 3D array $\{\Psi_{k-1}(\beta_{k-1},\gamma_{k-1})\}_{\beta_{k-1},\gamma_{k-1}=1}^{r_{k-1},r_{k-1}}$ of sizes $r_{k-1}\times r_{k-1}\times R_{k-1}$,
    right 3D array $\{\Phi_k(\beta_k,\gamma_k)\}_{\beta_k,\gamma_k=1}^{r_k,r_k}$ of sizes $r_k\times r_k\times R_k$,
    the core of matrix $\{A_k(i_k,j_k)\}_{i_k,j_k=1}^{n_k,n_k}$ of sizes $n_k\times n_k\times R_{k-1}\times R_k$,
    the vector $\{Y(\gamma_{k-1},j_k,\gamma_k)\}_{\gamma_{k-1},j_k,\gamma_k=1}^{r_{k-1},n_k,r_k}$ of sizes $r_{k-1}n_kr_k\times 1$.
Output: $W(\beta_{k-1},i_k,\beta_k)=\sum_{\gamma_{k-1},j_k,\gamma_k}B(\beta_{k-1},\beta_k,i_k,j_k,\gamma_{k-1},\gamma_k)Y(\gamma_{k-1},j_k,\gamma_k)$ the result of
    MatVec.
1: Reshape 3D array $\Phi_k$ as a $R_kr_k\times r_k$ matrix, and vector $Y$ as a $r_k\times n_kr_{k-1}$ matrix.
2: Multiply $Y'(1:R_kr_k,1:n_kr_{k-1})=\Phi_k(1:R_kr_k,1:r_k)Y(1:r_k,1:n_kr_{k-1})$. Reshape and
   permute $Y'$ to a matrix of sizes $n_kR_k\times r_{k-1}r_k$.
3: Reshape the matrix core $A_k$ to a $n_kR_{k-1}\times n_kR_k$ matrix.
4: Multiply $Y''(1:n_kR_{k-1},1:r_{k-1}r_k)=A_k(1:n_kR_{k-1},1:n_kR_k)Y'(1:n_kR_k,1:r_{k-1}r_k)$.
   Reshape and permute $Y''$ to a matrix of sizes $R_{k-1}r_{k-1}\times n_kr_k$.
5: Reshape $\Psi_k$ as a $r_{k-1}\times R_{k-1}r_{k-1}$ matrix.
6: Multiply $W(1:r_{k-1},1:n_kr_k)=\Psi_k(1:r_{k-1},1:R_{k-1}r_{k-1})Y''(1:R_{k-1}r_{k-1},1:n_kr_k)$.
7: Reshape $W$ to a vector of sizes $r_{k-1}n_kr_k\times 1$.
```

1. Generate a random tensor $\mathbf{Z}$ with cores Z_k and TT-ranks r_0 (say, $r_0=2$).
2. Add Z_k to the basis, i.e. set
$$X_k:=\begin{pmatrix} X_k & 0\\ 0 & Z_k\end{pmatrix}.$$
3. Orthogonalize X_k from right-to-left
4. Start left-to-right DMRG sweep

It is not difficult to see, that if $Ax\approx f$ then such modification will give the same local residues. If it is not true and the solution have non-zero components in the direction of the randomly generated tensor $\mathbf{Z}$, then the local system will reflect this and the initial local residue will be not small (and this will serve as an indicator for the restart of the method). Note, that the theoretical estimates on this random restart approach should be obtained, and we plan it for the future research. The previous considerations are summarized in the algorithms 1, 2, 3. In the main algorithm 3 we present only the DMRG scheme, as it contains more details than the ALS, and the latter can be understood by the similarity.

5. Inversion via DMRG

5.1. Reduction to a linear system

The proposed technique can be used not only to solve linear systems in the TT-format, but also to compute approximate inverses. Usually, the inversion of a matrix is not performed, since it is much more expensive than the solution of a linear system. For some classes of structured matrices it might not be the case, and the computation of the approximate inverse is “cheap” . Such approximate inverse can be used as a preconditioner. This of course requires that the inverse matrix can indeed be approximated by a structured matrix. The computation of approximate inverses via different approaches was performed for Toeplitz-like matrices [33], for computing sparse approximate inverses [34], for the inversion of low Kronecker-rank matrices

Algorithm 2: Solution of the local problem

```
Input: Left 3D array $\{\Psi_{k-1}(\beta_{k-1},\gamma_{k-1})\}_{\beta_{k-1},\gamma_{k-1}=1}^{r_{k-1},r_{k-1}}$ of sizes $r_{k-1}\times r_{k-1}\times R_{k-1}$,
    right 3D array $\{\Phi_k(\beta_k,\gamma_k)\}_{\beta_k,\gamma_k=1}^{r_k,r_k}$ of sizes $r_k\times r_k\times R_k$,
    the core of matrix $\{A_k(i_k,j_k)\}_{i_k,j_k=1}^{n_k,n_k}$ of sizes $n_k\times n_k\times R_{k-1}\times R_k$,
    the right-hand side vector $\{Y(\gamma_{k-1},j_k,\gamma_k)\}_{\gamma_{k-1},j_k,\gamma_k=1}^{r_{k-1},n_k,r_k}$ of sizes $r_{k-1}n_kr_k\times 1$,
    the critical size of the local problem $N_c$, number of iterations $I_t$ and residual tolerance $\varepsilon$ for
    the iterative solver.
Output: the approximate local solution $X\approx B^{-1}Y$ so that $X=\{X(\beta_{k-1},i_k,\beta_k)\}_{\beta_{k-1},i_k,\beta_k=1}^{r_{k-1},n_k,r_k}$.
 1: if $n_kr_{k-1}r_k<N_c$ then {Build and solve the local system directly}
 2:     Reshape $\Phi_k$ to a matrix of sizes $R_k\times r_kr_k$.
 3:     Reshape $A_k$ to a matrix of sizes $R_{k-1}n_kn_k\times R_k$.
 4:     Multiply $B'(1:R_{k-1}n_kn_k,1:r_kr_k)=A_k(1:R_{k-1}n_kn_k,1:R_k)\Phi_k(1:R_k,1:r_kr_k)$ and
        reshape $B'$ to a matrix $B'(1:R_{k-1},1:n_kn_kr_kr_k)$.
 5:     Reshape $\Psi_k$ to a matrix of sizes $r_{k-1}r_{k-1}\times R_{k-1}$.
 6:     Multiply $B(1:r_{k-1}r_{k-1},1:n_kn_kr_kr_k)=\Psi_k(1:r_{k-1}r_{k-1},1:R_{k-1})B'(1:R_{k-1},1:$
        $n_kn_kr_kr_k)$.
 7:     Reshape $B$ to a matrix $B(1:r_{k-1}n_kr_k,1:r_{k-1}n_kr_k)$.
 8:     Solve the local problem $X=B^{-1}Y$ using the direct elimination.
 9: else {Run iterative solver}
10:     Run an appropriate iterative solver (e.g. GMRES) using the Algorithm 1 for Matrix-by-
        Vector multiplications, right-hand side $Y$, with stopping criteria $I_t$ and $\varepsilon$.
11: end if
```

[35, 36]. In all these cases, the Newton method for the matrix inversion was used. It has the form

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

and if $||AX_0-I||<1$, for some matrix norm, it converges to the inverse quadratically. The only problem is that it requires two matrix-by-matrix products at each step. If such multiplication can be implemented fast for a given class of structured matrices, then the Newton method can often provide good approximate inverses. The disadvantage in the TT-case is that the TT-ranks of the product X_kAX_k can be quite large, and the complexity is polynomial in the solution rank but with a quite large exponent. In this paper we propose an alternative approach, which is much simpler to implement, given an algorithm that solves linear systems. The inverse matrix X to a given matrix A satisfies

$$AX=I. \tag{5.1}$$

The equation (5.1) is in fact a linear system:

$$(A\otimes I)x=\mathrm{vec}(I),$$

where $x=\mathrm{vec}(X)$ is a vectorized form of X. Now suppose, that A comes with the TT-structure, i.e. it can be associated with a 2d-dimensional array with elements

$$A(i_1,\dots,i_d,j_1,\dots,j_d)=A_1(i_1,j_1)\dots A_d(i_d,j_d).$$

For the simplicity assume that all indices involved vary from 1 to 2, i.e. the QTT case. In the multiindex notation, the equation (5.1) is formulated as

$$A(j_1',\dots,j_d',j_1,\dots,j_d)X(j_1,\dots,j_d,k_1,\dots,k_d)=I(j_1',\dots,j_d',k_1,\dots,k_d).$$

For the solution the same "matrix ordering" of dimensions is used:

$$x = X(j_1, k_1, \dots, j_d, k_d) \approx X_1(j_1, k_1) \dots X_d(j_d, k_d).$$

The matrix of this big linear system (corresponding to the matrix $A \otimes I$) then depends on the $4d$ indices, $\widehat{A}(j'_1, \dots, j'_d, k'_1, \dots, k'_d, j_1, \dots, j_d, k_1, \dots, k_d)$. The modes of the solution are ordered as $(j_1, k_1), (j_2, k_2), \dots, (j_d, k_d)$, thus the indices for the matrix of the linear system are naturally ordered as $(j'_1, k'_1, j_1, k_1), \dots, (j'_d, k'_d, j_d, k_d)$, i.e. correspond to a $16 \times 16 \times \dots \times 16$ d-dimensional tensor. Its elements are defined as

$$\widehat{A}(j'_1, k'_1, j_1, k_1, \dots, j'_d, k'_d, j_d, k_d) = A(j'_1, \dots, j'_d, j_1, \dots, j_d) I(k'_1, \dots, k'_d, k_1, \dots, k_d), \qquad (5.2)$$

where $I(k'_1, \dots, k'_d, k_1, \dots, k_d) = \delta(k'_1, k_1)\delta(k'_2, k_2) \dots \delta(k'_d, k_d)$, corresponds to the identity matrix which has TT-ranks equal to 1. From (5.2) it is simple to find out that the cores of $\widehat{A}$ in the required permutation are defined as

$$\widehat{A}_p(j'_p, k'_p, j_p, k_p) = A_p(j_p, j'_p)\delta(k_p, k'_p),$$

i.e. the TT-ranks of the $\widehat{A}$ are not larger than the TT-ranks of A. The only difference is that the mode size of $\widehat{A}$ is 4 times larger. Thus, the TT-Solve algorithm can be applied to the solution of the linear system with the matrix $\widehat{A}$ directly. Moreover, the right-hand side here is of a very simple structure. It has TT-ranks equal to 1. The complexity is quadratic in the mode size, thus the constant is only 4 times larger. The dependence in the grid size is the same, the crucial parameter that influences the complexity is the solution rank, which is always larger for the inverse than for the particular solution. However, it is important that for computing an approximate inverse high accuracy is not often needed: even an approximation with small ranks may lead to an excellent preconditioner. Thus, the linear system approach for the inversion look promising.

The approach described above requires a small modification to make it more robust. Suppose that X is an approximate solution of $AX \approx I$, i.e. $||AX - I||$ is small. It does not mean that the right residue is $||XA - I||$ is small, even for the symmetric A, which means that the approximate inverse for a symmetric matrix, computed by such method, can be non-symmetric. It is very simple to avoid this. Instead of solving (5.1), let us solve

$$AX + XA = 2I,$$

and the TT-ranks for the corresponding matrix are only twice larger, then for the initial one. The solution obtained is now much better: the right and left residue are of the same order.

6. Numerical experiments

6.1. Data and benchmarks

All computations were performed using the TT-Toolbox (available at http://spring.inm.ras.ru/osel). The TT-Toolbox contains an object-oriented implementation of the TT-format and operations with tensors in such format. Our solver is included in the TT-Toolbox 2.1 (procedure `dmrg_solve`). A minimal input is the matrix A in the TT-format (i.e., `tt_matrix` object), the right-hand side which is also in the TT-format (i.e., `tt_tensor` object) and the required relative accuracy ε. There are also several tuning parameters. All results of the computations in all considered examples can be found at the webpage http://spring.inm.ras.ru/osel in the

Section "Benchmarks and Data". They are provided as .mat files with the matrix A in the TT-format and the right-hand size and the approximate solution in the TT-format. We hope that these data may be used as benchmarks in future research. As an illustrative example, consider the solution of the Laplace equation $\Delta_d u = 1$. The creation of a matrix and the solution can be done via the MATLAB code

```
%xlap.m --- simple code to run the solver
L=8; n=2^L; %One-dimensional grid size is 256
d=10; %10-dimensional problem
lap=tt_qlaplace_dd(L*ones(1,d));
lap=tt_matrix(lap); %The Laplace operator is created
rhs=tt_ones(d*L,2); %Right-hand side of all ones
rhs=tt_tensor(rhs);
%Solve, initial guess is the rhs
sol=dmrg_solve2(lap,rhs,rhs,1e-8);
```

On a particular machine, this results in

```
 >> tic; xlap; toc;
sweep=1/10 erank=1.0
sweep=5/10 erank=14.7
sweep=10/10 erank=33.1
Elapsed time is 119.772275 seconds.
```

and the accuracy of the solution is

```
>> norm(lap*sol-rhs)/norm(rhs)
ans =  2.5449e-05
```

More accurate solution can be also obtained from this one by using smaller ε:

```
  >> tic; sol=dmrg_solve2(lap,rhs,sol,1e-8,[],[],4,[]); toc;
sweep=1/4 erank=33.2
sweep=4/4 erank=46.8
Elapsed time is 85.726980 seconds.
>> norm(lap*sol-rhs)/norm(rhs)
ans = 2.9708e-07
```

The Laplace equation can be of course solved very fast by sinc-type quadratures. This example is given only to present the usage of the algorithm in the TT-Toolbox. However, the TT-Solve method does not require constant coefficients. In the next subsection more numerical results will be given.

6.2. Solution of linear systems, reaction-diffusion problems

For numerical experiments we consider several examples. First, consider the reaction-diffusion equation $-(\Delta + q)u = f, \quad q > 0$ in two dimensions with a variable coefficient $q = q(x, y)$. The domain is a rectangle, and the Dirichlet boundary conditions are imposed. For the discretization a uniform $2^{d_0} \times 2^{d_0}$ grid is used, and the central finite difference approximation to the Laplace operator is employed. The right-hand side is taken as a vector of all ones.

Example 1 (Table 6.1, Figures 6.1, 6.2)

For the first example, we have taken $q = 500 \cdot \exp(-x^2 - y^2)$. We are starting from a random rank 2 tensor as the initial guess. The grid size is 1024×1024 so $d_0 = 10$ and the whole tensor is $d = 2d_0$-dimensional, the residual tolerance is 10^{-6} and a local system was solved via direct solver if its size less than 2500 (see (4.1)).

Table 6.1

sweep	relative residual	maximal QTT rank	time of the current sweep, sec.
1	3.109e+02	2	0.027573
2	1.621e+01	4	0.043498
3	4.986e+01	8	0.156974
4	8.352e+01	16	1.02399
5	5.393e-01	32	8.93552
6	5.480e-04	46	17.9173
7	3.314e-05	65	7.03498
8	5.498e-06	80	6.83953
9	3.203e-06	81	6.87847
10	2.927e-06	81	6.84604

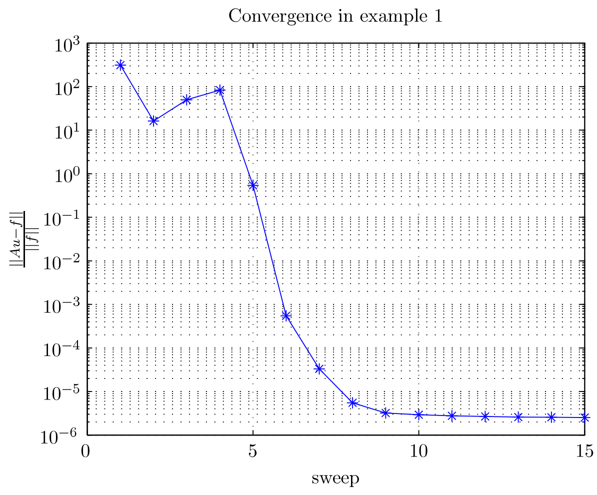

Figure 6.1. Relative residual versus the sweep number, example 1

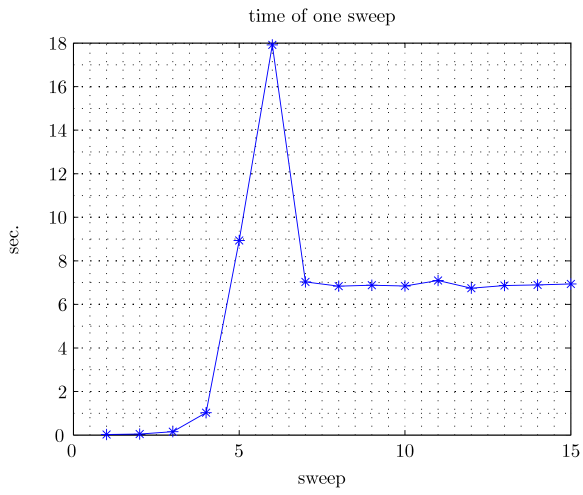

Figure 6.2. Time of one sweep versus the sweep number, example 1

We see, that the most time consuming are the middle sweeps, where the local system is already large and ill-conditioned, but the accuracy is still bad, hence a lot of local iterations are required. The singular values in the decimation steps were filtered so that the residual provided by a splitted superblock is less than 2 times the residual achieved in the local solver. So, the residual stabilizes below a value $2\sqrt{d-1}\varepsilon$ (see [11] for the reason why the factor $\sqrt{d-1}$ arise).

Example 2 (Tables 6.2, 6.3) In this example we take $d_0 = 10$,

$$q = 400(x + y),$$

and check how the convergence and timings depend on the residual tolerance and grid size. First, set the Q-dimension $d_0 = 10$ and check the minimal achieved residual, the required number of sweeps, the full time of the solver and maximal QTT-ranks of the solution with respect to the residual tolerance ε. Now, check the performance with respect to the grid size. The residual

Table 6.2. Final residual, full time, conducted sweeps and maximal rank versus ε, example 2

ε	relative residual	maximal QTT rank	sweeps	full time, sec
10^{-4}	2.555e-04	23	12	14.1901
10^{-6}	2.362e-06	34	10	30.9652
10^{-7}	2.965e-07	38	12	52.0496

tolerance is fixed to $\varepsilon = 10^{-6}$. In the case $d_0 = 8$, the grid seems to be too coarse to resolve the

Table 6.3. Final residual, full time, conducted sweeps and maximal rank versus d_0, example 2

d_0	relative residual	maximal QTT rank	sweeps	full time, sec
8	1.260e-05	27	7	15.5158
9	2.165e-06	31	10	29.2388
10	2.362e-06	34	10	30.9652
11	3.823e-06	36	11	44.3646

solution with an accuracy 10^{-6}. The dependence of the full time versus the grid size is linear (the worst case) due to increasing ranks and number of sweeps. Nevertheless, in the cases where the number of sweeps is stable, the *logarithmic* complexity is achievable, so that all advantages of the QTT-format are exploited.

Example 2.1 (Table 6.4) We also compared the times of the TT-solve with the direct solution in the full representation and one Matrix-by-Vector multiplication. For the direct solver the MATLAB backslash operator (for the sparse matrix) was used. Here we consider the reaction diffusion-equation with $q = 100(x + y + z)$. The residual tolerance was fixed to a value $\varepsilon = 10^{-6}$. The residual was stabilized in 9 DMRG sweeps for all considered grid sizes. Here the

Table 6.4. Times of the TT-solve, the direct solution in the full representation and full MatVec

d_0	TT solver, sec.	Full direct solver, sec.	Full MatVec, sec	$\frac{\text{TT solver time}}{\text{MatVec time}}$
6	18.2063	155	0.00685	2658
7	43.0326	17134	0.0784	548
8	82.8516	*Out Of Memory*	0.6457	128

time of the TT-solve grows linearly with the grid size due to the increasing ranks, but remains significantly smaller than the time of the full solver. We could not solve the problem on grids finer than 128^3 due to memory limitations.

6.3. Solution of linear systems, multidimensional elliptic problems

Example 3 (Table 6.5, Figures 6.3, 6.4)

Now the algorithm is tested on a model high-dimensional problem. Consider a 8-dimensional reaction-diffusion equation

$$(-\Delta + 100\exp(-r^2))u = 1, \quad r^2 = \sum_{i=1}^{8}(x_i - 0.5)^2$$

with a positive definite operator. 256 grid nodes are taken in each direction, so the Q-dimension is $d_0 = 8$ and the full dimension is $d = 64$. The residual tolerance was set to $\varepsilon = 10^{-6}$.

Table 6.5

sweep	relative residual	maximal QTT rank	time of the current sweep, sec.
1	4.090e+00	4	0.173487
5	1.842e-02	42	32.8344
10	1.064e-05	63	28.3208
15	8.551e-06	62	27.0177

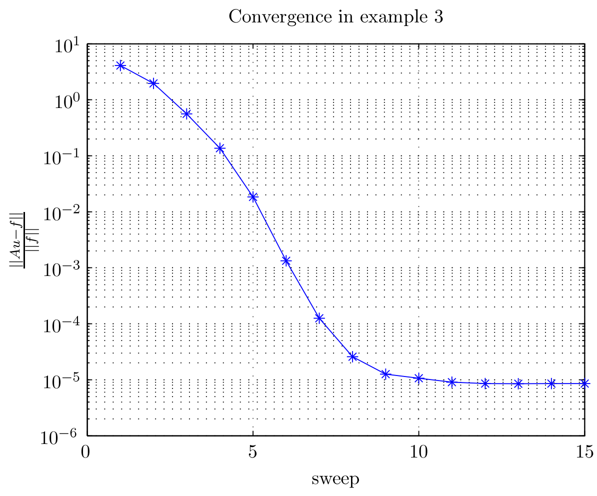

Figure 6.3. Relative residual versus the sweep number, example 3

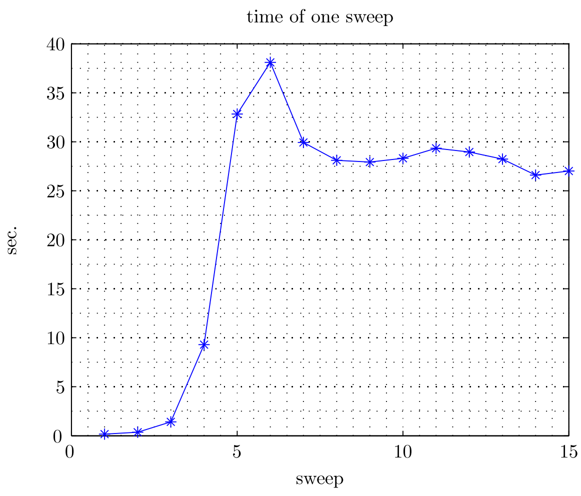

Figure 6.4. Time of one sweep versus the sweep number, example 3

The residual for a positive-definite problem decreases monotonously, but the minimal achievable value is greater then in the previous examples due to a larger $\sqrt{d-1}$ factor.

Example 4 (Table 6.6) Consider an essentially high-dimensional problem: 64-dimensional reaction-diffusion

$$(-\Delta + 100 \exp(-r^2))u = 1, \quad r^2 = \sum_{i=1}^{64} (x_i - 0.5)^2,$$

discretized on a grid with 256 points in each direction, thus providing a 512-dimensional tensor as a solution. Here, 15 sweeps and 1845 seconds are required to achieve a $\mathcal{O}(10^{-5})$ accuracy.

Table 6.6

sweep	relative residual	maximal QTT rank	time of the current sweep, sec.
1	1.399e+00	4	0.754949
5	1.410e-01	25	39.0297
10	3.973e-04	38	208.696
15	6.403e-05	39	166.427

Notice that the full representation of such tensor requires $2^{512} = 10^{154}$ data points.

Example 5 (Table 6.7, Figures 6.5, 6.6) In this example we solve a parameter-dependent

2D Poisson equation with parametric boundary conditions:

$$
\begin{aligned}
-\Delta u(\mathbf{x}, \alpha) = f = 1 \quad &\text{in } \Omega = [0,1]^2, \\
\mathbf{x} = [x_1, x_2], \quad &\alpha = [\alpha_1, \alpha_2, \alpha_3, \alpha_4], \\
\alpha_1 \in [0.5, 1], \quad &\alpha_2, ..., \alpha_4 \in [0,1], \\
\alpha_1 u + (1-\alpha_1)\partial u/\partial n &= 0 \text{ at } x_1 = 0, \\
\alpha_2 u + (1-\alpha_2)\partial u/\partial n &= 0 \text{ at } x_2 = 0, \\
\alpha_3 u + (1-\alpha_3)\partial u/\partial n &= 0 \text{ at } x_1 = 1, \\
\alpha_4 u + (1-\alpha_4)\partial u/\partial n &= 0 \text{ at } x_2 = 1.
\end{aligned}
$$

Hence we have a 6-dimensional problem. We discretize it using a uniform grid with 256 points in both physical and parametric spaces, so the solution is a 48-dimensional tensor. The residual tolerance is $\varepsilon = 10^{-6}$. For the comparison, the QTT-ranks of a solution of a "single" 2D Poisson equation with the Dirichlet boundary conditions are about 20. It is interesting that for such parametrized boundary condition the ranks are only 4 times larger. Another interesting feature of this example is that the approximate solution has maximal ranks in the middle sweeps, but then they stabilize with the convergence of the method.

Table 6.7

sweep	relative residual	maximal QTT rank	time of the current sweep, sec.
1	1.006e+01	4	0.180521
5	2.181e+00	36	8.34796
9	1.082e-04	100	40.3669
10	2.236e-05	94	37.6331

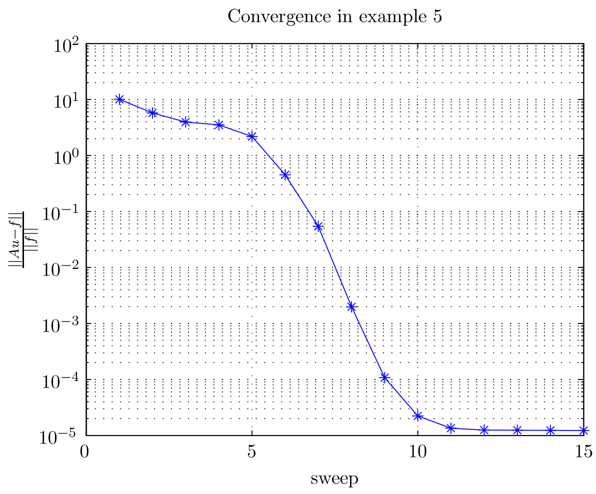

Figure 6.5. Relative residual versus the sweep number, example 5

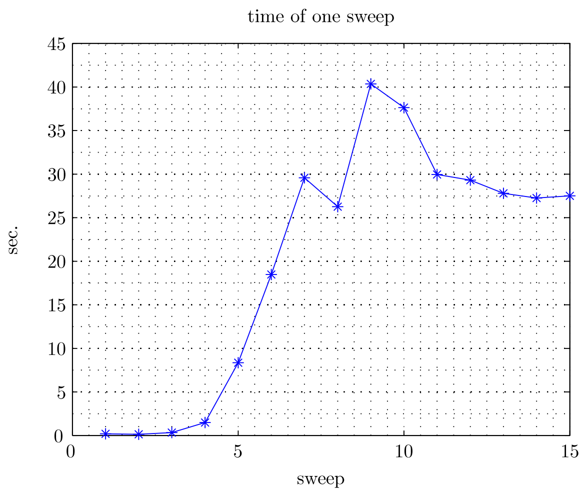

Figure 6.6. Time of one sweep versus the sweep number, example 5

6.4. Matrix inversion

Example 6 (Table 6.8, Figures 6.7, 6.8) In the last example the application of the TT-Solve to the approximate inversion is presented. One of the most promising applications for the approximate inversion is the solution of the parabolic problems via implicit schemes. For the simplest implicit Euler scheme, applied to the heat equation, one has to invert a matrix of form $A = I - \alpha\Delta, \quad \alpha = 10^{-3}$. The symmetrized equation $AX + XA = 2I$ is solved. A 2-dimensional problem is considered, discretized on a 1024×1024 grid so that the solution is a 20-dimensional tensor with mode sizes 4. The residual tolerance is set to $\varepsilon = 10^{-4}$. As an initial guess, the identity matrix was used to ensure the symmetry of the inverse matrix. After the solution we

Table 6.8

sweep	relative residual	maximal QTT rank	time of the current sweep, sec.
1	3.849e-01	4	0.027729
5	5.062e-01	5	0.25945
10	7.574e-02	41	4.86231
15	3.379e-04	49	9.63131

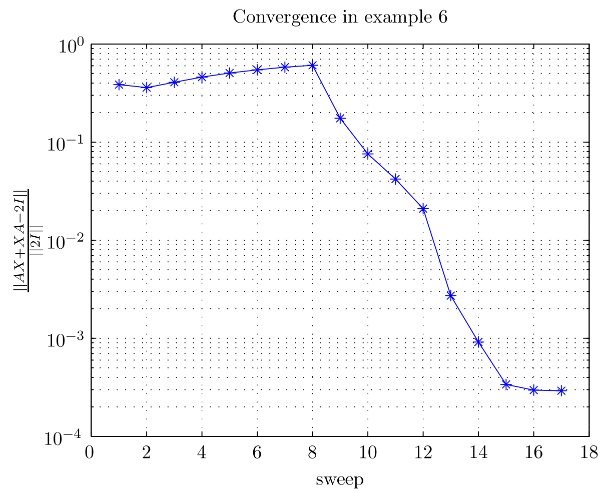

Figure 6.7. Relative residual versus the sweep number, example 6

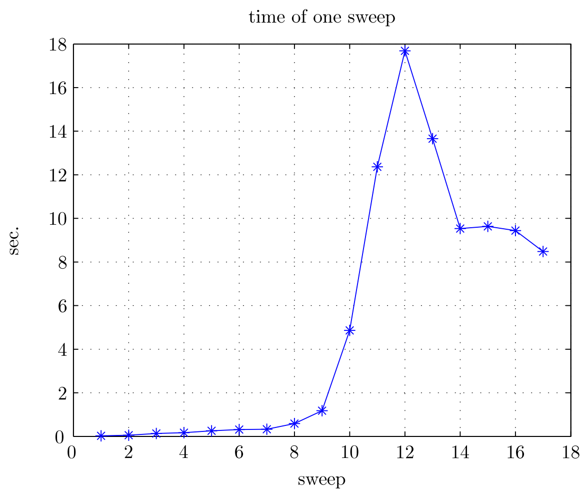

Figure 6.8. Time of one sweep versus the sweep number, example 6

checked the matrix related properties. The error in symmetry is $\frac{||X - X^\top||}{||X||} = 2.8872\text{e-}06$, even lower than ε. The right residual is $\frac{||AX - I||}{||I||} = 2.7156\text{e-}04$ and the left residual is $\frac{||XA - I||}{||I||} = 2.7365\text{e-}04$. Since we started from the identity matrix, the method required more "warming" iterations in comparison with a random initial guess. Fortunately these additional iterations are significantly cheaper than the "main" iterations, and we can admit them in order to obtain a symmetric matrix with a high accuracy.

7. Conclusion and future work

In this paper it is described, how to solve linear systems in the TT-format directly, using the DMRG approach. The idea with preliminary experiments was already described in the paper [26], however an efficient practical implementation requires several "tricks". The proposed solver, as confirmed by numerical experiments, is able to solve problems in a black-box fashion (i.e. only the matrix is the QTT format is provided) with 2^d unknows where d is of order several hundreds, and what is even more important, with TT-ranks up to hundred on even using a prototype MATLAB implementation.

The main trick is to truncate the local solution on the basis of the local residue, not on the relative Frobenius norm. Our method can handle non-constant coefficients, where approaches based on sinc-type quadratures can not be used. In principle,the TT-Solve can be used for solving any linear system, if the matrix is provided in the TT-format. It can be generalized effortlessly to compute approximate inverses to serve as preconditioners. To compute such preconditioners to higher accuracy, we propose to solve the Sylvester equation by the TT-Solve directly. With small mode sizes (for QTT they are usually equal to 2) the inversion is as cheap as the solution itself, making the TT-solve a very powerful approach for computing preconditioners for matrices corresponding to the discretizations of operators on structured tensor grids and even for computing the Green functions of certain operators.

There is no theory available yet for the convergence of the DMRG-type methods. In many cases the numerical experiments confirm that it converges in a number of sweeps independent of the grid size. This experimental fact deserves to be investigated and proved. From the computational point of view, the most time consuming part is the solution of local linear systems, which may require a lot of iterations (the single matrix-by-vector product is cheap). A certain preconditioner for the local problem is required, and we will continue to test different approaches which can be used for the preconditioning of the local systems.

We plan to apply this linear system solver for certain high-dimensional problems, for example for implicit schemes for parabolic high-dimensional problems, in particular, to the Fokker-Planck equation. This will be described in the future works.

References

[1] *Meyer H.-D., Gatti F., Worth G. A.* Multidimensional Quantum Dynamics: MCTDH Theory and Applications. — Weinheim: Wiley-VCH, 2009.

[2] *Lübich C.* From quantum to classical molecular dynamics: reduced models and numerical analysis. — Zurich: EMS, 2008.

[3] *Sloan I., Wozniakowski H.* When Are Quasi-Monte Carlo Algorithms Efficient for High Dimensional Integrals // *Journal of Complexity*. 1998. V. 14, № 1. P. 1–33.

[4] *Wang X., Sloan I.* Why are high-dimensional finance problems often of low effective dimension? // *SIAM J. Sci. Comp.* 2006. V. 27, № 1. P. 159–183.

[5] *Bader B., Kolda T.* Efficient MATLAB computations with sparse and factored tensors // *SIAM J. Sci. Comp.* 2007. V. 30, № 1. P. 205-231.

[6] *Bader B., Kolda T.* Tensor decompositions and applications // *SIAM Review*. 2009, Sep. V. 51, № 3. P. 455–500.

[7] *Hackbusch W., Kühn S.* A new scheme for the tensor representation // *J. Fourier Anal. Appl.* 2009. V. 15, № 5. P. 706–722.

[8] *Grasedyck L.* Hierarchical singular value decomposition of tensors // *SIAM J. Matrix Anal. Appl.* 2010. V. 31, № 4. P. 2029-2054.

[9] *Oseledets I. V., Tyrtyshnikov E. E.* Breaking the curse of dimensionality, or how to use SVD in many dimensions // *SIAM J. Sci. Comp.* 2009. V. 31, № 5. P. 3744-3759. doi: 10.1137/090748330.

[10] *Oseledets I. V.* Compact matrix form of the d-dimensional tensor decomposition: Preprint 2009-01. — Moscow: INM RAS, 2009. http://pub.inm.ras.ru.

[11] *Oseledets I. V.* Tensor train decomposition // *SIAM J. Sci. Comp.* 2011.

[12] *Tucker L. R.* Some mathematical notes on three-mode factor analysis // *Psychometrika*. 1966. V. 31. P. 279-311.

[13] *de Lathauwer L., de Moor B., Vandewalle J.* A multilinear singular value decomposition // *SIAM J. Matrix Anal. Appl.* 2000. V. 21. P. 1253–1278.

[14] *Beylkin G., Mohlenkamp M. J.* Numerical operator calculus in higher dimensions // *Proc. Nat. Acad. Sci. USA*. 2002. V. 99, № 16. P. 10246-10251.

[15] *Beylkin G., Mohlenkamp M. J.* Algorithms for numerical analysis in high dimensions // *SIAM J. Sci. Comp.* 2005. V. 26, № 6. P. 2133-2159.

[16] *Hackbusch W., Khoromskij B. N.* Low-rank Kronecker-product approximation to multi-dimensional nonlocal operators. I. Separable approximation of multi-variate functions // *Computing*. 2006. V. 76, №3-4. P. 177–202.

[17] *Hackbusch W., Khoromskij B. N.* Low-rank Kronecker-product approximation to multi-dimensional nonlocal operators. II. HKT representation of certain operators // *Computing*. 2006. V. 76, №3-4. P. 203–225.

[18] *Grasedyck L.* Existence and computation of low Kronecker-rank approximations for large systems in tensor product structure // *Computing*. 2004. V. 72. P. 247-265.

[19] *Carroll J. D., Chang J. J.* Analysis of individual differences in multidimensional scaling via n-way generalization of Eckart-Young decomposition // *Psychometrika*. 1970. V. 35. P. 283-319.

[20] *Harshman R. A.* Foundations of the Parafac procedure: models and conditions for an explanatory multimodal factor analysis // *UCLA Working Papers in Phonetics*. 1970. V. 16. P. 1-84.

[21] *Espig M., Hackbusch W.* A regularized Newton method for the efficient approximation of tensors represented in the canonical format: Preprint 78. — Leipzig: MPI MIS, 2010. www.mis.mpg.de/preprints/2010/preprint2009_78.pdf.

[22] *de Silva V., Lim L.-H.* Tensor rank and the ill-posedness of the best low-rank approximation problem // *SIAM J. Matrix Anal. Appl.* 2008. V. 30, № 3. P. 1084–1127.

[23] *de Lathauwer L., de Moor B., Vandewalle J.* On best rank-1 and rank-$(R_1, R_2, ..., R_N)$ approximation of high-order tensors // *SIAM J. Matrix Anal. Appl.* 2000. V. 21. P. 1324-1342.

[24] *Oseledets I. V., Savostianov D. V., Tyrtyshnikov E. E.* Tucker dimensionality reduction of three-dimensional arrays in linear time // *SIAM J. Matrix Anal. Appl.* 2008. V. 30, № 3. P. 939-956. doi: 10.1137/060655894.

[25] *Khoromskij B. N., Khoromskaia V., Flad H.-J.* Numerical Solution of the Hartree–Fock Equation in Multilevel Tensor-Structured Format // *SIAM J. Sci. Comput.* 2011. V. 33, № 1. P. 45-65. doi: 10.1137/090777372. http://link.aip.org/link/?SCE/33/45/1.

[26] *Rohwedder T., Holtz S., Schneider R.* The alternation least squares scheme for tensor optimisation in the TT-format: Preprint DFG-Schwerpunktprogramm 1234 71: 2010.

[27] *Khoromskij B. N., Oseledets I. V.* DMRG + QTT approach to high-dimensional quantum molecular dynamics: Preprint 68. — Leipzig: MPI MIS, 2010. www.mis.mpg.de/preprints/2010/preprint2010_68.pdf.

[28] *White S. R.* Density-matrix algorithms for quantum renormalization groups // *Physical Review B*. 1993. V. 48, № 14. P. 10345–10356. doi: 10.1103/PhysRevB.48.10345. http://link.aps.org/doi/10.1103/PhysRevB.48.10345.

[29] *Östlund S., Rommer S.* Thermodynamic Limit of Density Matrix Renormalization // *Physical Review Letters*. 1995. V. 75, № 19. P. 3537–3540. doi: 10.1103/PhysRevLett.75.3537. http://link.aps.org/doi/10.1103/PhysRevLett.75.3537.

[30] *Oseledets I. V.* Approximation of $2^d \times 2^d$ matrices using tensor decomposition // *SIAM J. Matrix Anal. Appl.* 2010. V. 31, № 4. P. 2130-2145. doi: 10.1137/090757861.

[31] *Khoromskij B. N.* $\mathcal{O}(d \log N)$-Quantics approximation of N-d tensors in high-dimensional numerical modelling: Preprint 55. — Leipzig: MPI MIS, 2009. www.mis.mpg.de/preprints/2009/preprint2009_55.pdf.

[32] *Van Loan C. F., Pitsianis N.* Approximation with Kronecker products // Linear algebra for large scale and real-time applications (Leuven, 1992). — Dordrecht: Kluwer Acad. Publ., 1993. — V. 232 of *NATO Adv. Sci. Inst. Ser. E Appl. Sci.* — P. 293–314.

[33] *Pan V. Y., Rami Y., Wang X.* Structure matrices and Newton's iteration: unified approach // *Linear Algebra Appl.* 2002. V. 343-344. P. 232-265.

[34] *Chow E., Saad Y.* Approximate inverse preconditioners via sparse-sparse iterations // *SIAM J. Sci. Comp.* 1998. V. 19, № 3. P. 995–1023.

[35] *Oseledets I. V., Tyrtyshnikov E. E.* Approximate inversion of matrices in the process of solving a hypersingular integral equation // *Comp. Math. and Math. Phys.* 2005. V. 45, № 2. P. 302-313.

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

Algorithm 3: TT-solve

Input: Matrix A, right-hand side f and the initial guess x in the QTT format, the residual tolerance ε, the critical size of local problems N_c, the maximal number of local (I_t) and global N_{swp} iterations.

Output: The updated approximate solution x to a problem $Ax = f$ in the QTT format.

```
1:  for $i_{swp} = 1, ..., N_{swp}$ do
2:    {Orthogonalization from right to the left and $\Phi$ generation}
3:    Set $\Phi_d = 1$, $\Phi^f_d = 1$.
4:    for $k = d, d-1, ..., 2$ do
5:      Reshape the k-th core $\{X_k(i_k)\}_{i_k=1}^{n_k}$ to a matrix of size $n_k r_k \times r_{k-1}$.
6:      Compute QR decomposition $[Q, R] = \mathrm{qr}(X_k)$. Denote $r_{new} = \min(n_k r_k, r_{k-1})$.
7:      Reshape the matrix $Q$ to the parts of the new core $X_k(i_k)$ of sizes $r_{new} \times r_k$.
8:      Reshape the next core $\{X_{k-1}(i_{k-1})\}_{i_{k-1}=1}^{n_{k-1}}$ to a matrix $(r_{k-2}n_{k-1} \times r_{k-1}$.
9:      Multiply $\tilde{X}_k(1 : r_{k-2}n_{k-1}, 1 : r_{new}) = X_k(1 : r_{k-1}n_{k-1}, 1 : r_{k-1})R^\top(1 : r_{k-1}, 1 : r_{new})$
        and reshape it to the parts of the new core $X_{k-1}(i_{k-1})$.
        {Now, update $\Phi_{k-1}$ and $\Phi^f_{k-1}$}
10:     Reshape $\Phi_k$ to a matrix of sizes $r_k \times R_k r_k$.
11:     Reshape $X_k$ to a matrix of sizes $r_{k-1}n_k \times r_k$.
12:     Multiply $\Phi'(1 : r_{k-1}n_k, 1 : R_k r_k) = X_k(1 : r_{k-1}n_k, 1 : r_k)\Phi_k(1 : r_k, 1 : R_k r_k)$.
13:     Reshape $\Phi'$ as a $r_{k-1}r_k \times n_k R_k$ matrix, and the matrix core $A_k$ as a $n_k R_k \times n_k R_{k-1}$
        matrix.
14:     Multiply $\Phi''(1 : r_{k-1}r_k, 1 : n_k R_{k-1}) = \Phi'(1 : r_{k-1}r_k, 1 : n_k R_k)A_k(1 : n_k R_k, 1 : n_k R_{k-1})$.
15:     Reshape $\Phi''$ as a $r_{k-1}R_{k-1} \times n_k r_k$ matrix, and $X_k$ as a $n_k r_k \times r_{k-1}$ matrix.
16:     Multiply $\Phi_{k-1}(1 : r_{k-1}R_{k-1}, 1 : r_{k-1}) = \Phi''(1 : r_{k-1}R_{k-1}, 1 : n_k r_k)X_k(1 : n_k r_k, 1 : r_{k-1})$.

17:     Update $\Phi^f_{k-1}$: $\tilde{\Phi} = \Phi^f_k(1 : r_k, 1 : r^f_k)F_k(1 : r^f_k, 1 : n_k r^f_{k-1})$.
18:     $\Phi^f_{k-1} = X_k(1 : r_{k-1}, 1 : n_k r_k)\tilde{\Phi}(1 : n_k r_k, 1 : r^f_{k-1})$.
19:   end for
      {Main DMRG iteration}
20:   $\Psi_1 = 1$, $\Psi^f_1 = 1$.
21:   for $k = 1, 2, ..., d-1$ do
22:     Build right-hand side
        $Y(\gamma_{k-1}, i_k i_{k+1}, \gamma_{k+1}) = \Psi^f_k(\gamma_{k-1}, 1 : r^f_{k-1})Y_k(i_k)Y_{k+1}(i_{k+1})\Phi^f_{k+1}(1 : r^f_{k+1}, \gamma_{k+1})$.
23:     Solve the local system $BX = Y$ using the Algorithm 2 (using the parameters $N_c$, $I_t$ and
        $\varepsilon$) and obtain $X(\beta_{k-1}, j_k j_{k+1}, \beta_{k+1})$.
24:     Decimation step: compute $[X_1, S, X_2] = \mathrm{svd}(X(1 : r_{k-1}n_k, 1 : n_{k+1}r_{k+1}))$, $X_2 = SX_2$.
25:     Truncate singular values so that $||B(X_1X_2) - Y|| \leqslant 2\varepsilon||Y||$.
26:     Update the solution blocks $\{X_k(j_k)\}_{j_k=1}^{n_k} = X_1$, $\{X_{k+1}(j_{k+1})\}_{j_{k+1}=1}^{n_{k+1}} = X_2$
        {Update $\Psi_{k+1}$ and $\Psi^f_{k+1}$:}
27:     $\Psi'(1 : r_{k-1}R_{k-1}, 1 : n_k r_k) = \Psi_k(1 : r_{k-1}R_{k-1}, 1 : r_{k-1})X_k(1 : r_{k-1}, 1 : n_k r_k)$.
28:     $\Psi''(1 : n_k R_k, 1 : r_{k-1}r_k) = A_k(1 : n_k R_k, 1 : n_k R_{k-1})\Psi'(1 : n_k R_{k-1}, 1 : r_{k-1}r_k)$.
29:     $\Psi_{k+1}() = X_k(1 : r_k, 1 : n_k r_{k-1})\Psi''(1 : n_k r_{k-1}, 1 : R_k r_k)$.
30:     $\tilde{\Psi} = \Psi^f_k(r_{k-1}, r^f_{k-1})F_k(1 : r^f_{k-1}, 1 : n_k r^f_k)$.
31:     $\Psi^f_{k+1} = X_k(1 : r_k, 1 : n_k r_{k-1})\tilde{\Psi}(1 : n_k r_{k-1}, r^f_k)$.
32:   end for
33: end for
```