Low-rank Riemannian eigensolver for high-dimensional Hamiltonians

Maxim Rakhuba\textsuperscript{a,1,*}, Alexander Novikov\textsuperscript{b,c}, Ivan Oseledets\textsuperscript{d,b}

\textsuperscript{a}Seminar for Applied Mathematics, ETH Zurich, Rämistrasse 101, 8092 Zurich, Switzerland
\textsuperscript{b}Marchuk Institute of Numerical Mathematics of the Russian Academy of Sciences, 119333 Moscow, Russia
\textsuperscript{c}National Research University Higher School of Economics, 101000 Moscow, Russia
\textsuperscript{d}Skolkovo Institute of Science and Technology, Skolkovo Innovation Center, 143026 Moscow, Russia

Abstract

Such problems as computation of spectra of spin chains and vibrational spectra of molecules can be written as high-dimensional eigenvalue problems, i.e., when the eigenvector can be naturally represented as a multidimensional tensor. Tensor methods have proven to be an efficient tool for the approximation of solutions of high-dimensional eigenvalue problems, however, their performance deteriorates quickly when the number of eigenstates to be computed increases. We address this issue by designing a new algorithm motivated by the ideas of Riemannian optimization (optimization on smooth manifolds) for the approximation of multiple eigenstates in the tensor-train format, which is also known as matrix product state representation. The proposed algorithm is implemented in TensorFlow, which allows for both CPU and GPU parallelization.

1. Introduction

The paper aims at the approximate computation of $b$ lowest eigenvalues $\epsilon^{(i)}$ and corresponding eigenvectors $x^{(i)}$ of a high-dimensional Hamiltonian $H \in \mathbb{R}^{n^d \times n^d}$:

$$H x^{(i)} = \epsilon^{(i)} x^{(i)}, \quad i = 1, \ldots, b. \quad (1)$$

Such problems arise in different applications of solid state physics and quantum chemistry problems including, but not restricted to the computation of spectra of spin chains ($n = 2$) or vibrational spectra of molecules (usually $n < 20$).

High-dimensional problems are known to be computationally challenging due to the curse of dimensionality, which implies exponential growth of the number of parameters with respect to the dimensionality $d$ of the problem. For example, storage of a single eigenvector for a spin chain with $d = 50$ spins requires approximately 10 Pb of computer

*Corresponding author

Email addresses: maksim.rakhuba@sam.math.ethz.ch (Maxim Rakhuba), sasha.v.novikov@gmail.com (Alexander Novikov), i.oseledets@skoltech.ru (Ivan Oseledets)

\textsuperscript{1}Work by MR on this project was performed while he was a junior research scientist at Skolkovo Institute of Science and Technology.
memory, which is far beyond available RAM on modern supercomputers. One way to deal with the curse of dimensionality is to utilize tensor decompositions of vectors $x^{(i)}$, $i = 1, \ldots, b$ represented as multidimensional arrays $\lambda^{(i)}$ of size $n \times \cdots \times n$ also known as tensors. Tensor decompositions have appeared to be useful for a long time both in solid states physics and molecular dynamics communities where different kinds of tensor decompositions have been used. Recently, the tensor method has also attracted the attention in the numerical analysis community, where new ideas such as cross approximation method [1, 2, 3] have been developed.

In this work we introduce a new method for solving problem (1) using the tensor-train (TT) format [4], also known as matrix product state (MPS) representation, which has been used for a long time in quantum information theory and solid state physics to approximate certain wavefunctions [5, 6] (DMRG method), see the review [7] for more details. One of the peculiarities of the proposed method is that it utilizes ideas of optimization on smooth manifolds (Riemannian optimization). This is possible due to the fact that the set of tensors of a fixed TT-rank$^2$ forms a smooth nonlinear manifold [8]. For small TT-ranks the manifold is low-dimensional and hence all computations are inexpensive.

To find a single eigenpair $(\epsilon^{(1)}, x^{(1)})$ Riemannian optimization allows to naturally avoid the rank growth of the method (Sec. 2). This is essential for fast computations as the complexity of the tensor method usually has strong rank dependence. However, for finding several eigenvalues, the generalization is not straightforward and leads to a significant complexity increase. Alternatively, one can compute eigenpairs using Riemannian optimization one-by-one, i.e. using deflation techniques, but such an approach is known to have slow convergence if the spectra is clustered [9]. To avoid this we propose a modification of the method (Sec. 3) that on the one hand keeps the benefits of efficient single eigenstate computation using Riemannian approach, and on the other hand inherits faster convergence properties of block methods.

The idea of the proposed method is as follows. We suggest projecting all the eigenvectors at each iteration onto a single specifically chosen tangent space$^3$. However, there is no reason to believe that all eigenvectors can be approximated using tangent space of a single eigenvector. Thus, in our algorithm, we always treat the projection as a correction to the current iterate. Overall, this leads to a small, but non-standard optimization procedure for the coefficients of the iterative process (Sec. 5).

The main contributions of this paper are:

- We develop a low-rank Riemannian alternating projection (LRRAP) concept for block iterative methods and apply it to the locally optimal block preconditioned conjugate gradient (LOBPCG) method.

- We implement the proposed LRRAP LOBPCG solver using TensorFlow$^4$ library, which allows for parallelization on both CPUs and GPUs.

$^2$TT-rank controls the number of parameters in the decomposition (Sec. 2.1).

$^3$In a nutshell, tangent space is a linearization of the manifold at a given point (Sec. 2.2).

$^4$If you are unfamiliar with TensorFlow, see Appendix A for introduction.
We accurately calculate vibrational spectra of acetonitrile (CH\textsubscript{3}CN) and ethylene oxide (C\textsubscript{2}H\textsubscript{4}O) molecules as well as spectra of one-dimensional spin chains. The comparison with the state-of-the-art methods in these domains indicates that we obtain comparable accuracy with a significant acceleration (up to 20 times) for large $b$ thanks to the design of the method and the GPU support.

2. Riemannian optimization on TT manifolds

In this section we provide a brief description of TT-decomposition and Riemannian optimization essentials on the example of a single eigenvalue computation using LOBPCG method. In Section 3 the approach described here will be generalized to the computation of several eigenvalues.

2.1. TT representation

Recall that we consider problem (1) and represent each eigenvector of size $n^d$ as a $n \times \cdots \times n$ tensor. This allows for compression using tensor decompositions of multidimensional arrays, and particularly the TT decomposition. For tensor $\mathcal{X} = \{\mathcal{X}_{i_1,\ldots,i_d}\}_{i_1,\ldots,i_d=1}^n \in \mathbb{R}^{n\times\cdots\times n}$ its TT decomposition reads

$$\mathcal{X}_{i_1,\ldots,i_d} = G_1(i_1) G_2(i_2) \cdots G_d(i_d),$$

where $G_k(i_k)$ are $r_{k-1} \times r_k$ matrices, $k = 2, \ldots, d - 1$. For the product of matrices to be a number we require that $G_1(i_1)$ be row matrices $1 \times r_1$ and $G_d(i_d)$ be column matrices $r_{d-1} \times 1$, which means $r_0 = r_d = 1$. For simplicity we force $r_1 = \cdots = r_{d-1} = r$ and call $r$ the TT-rank. The same value of $r_k$ for all $k = 1, \ldots, d - 1$ implies that for some modes $r_k$ can be overestimated. Note that in order to store TT representation of array $\mathcal{X}$ one needs $O(d nr^2)$ elements compared with $n^d$ elements of the initial array. The TT representation of Hamiltonian\textsuperscript{5} is defined by analogy. We will denote the maximum rank of a Hamiltonian by $R$.

It rarely happens that some tensor can be represented with small TT-rank $r$ exactly. Therefore, to keep ranks small the tensor is approximated with some accuracy by another tensor with a small TT-rank. In the considered numerical experiments we expect exponential decay of the introduced error with respect to $r$.

From now on we say that vector $\mathbf{x}$ of length $n^d$ is of TT-rank $r$ implying that being reshaped into a $n \times \cdots \times n$ multidimensional array $\mathcal{X}$: $\mathbf{x} = \text{vec}(\mathcal{X})$ it can be represented with TT-rank equal to $r$.

2.2. Rayleigh quotient minimization using Riemannian optimization

Consider the problem of finding the smallest eigenvalue $\epsilon^{(1)}$ (assuming it is simple) and the corresponding eigenvector $\mathbf{x}^{(1)}$. Suppose we are given an a priori knowledge

\textsuperscript{5} Matrix $\mathbf{H} \in \mathbb{R}^{n^d \times n^d}$ can also be naturally considered as a multidimensional array $\mathcal{H}$ of dimension $2d$. The TT decomposition of $\mathcal{H}$ reads $\mathcal{H}_{i_1,\ldots,i_d,j_1,\ldots,j_d} = H_1(i_1,j_1) H_2(i_2,j_2) \cdots H_d(i_d,j_d)$, where $i_1,\ldots,i_d$ represent row indexing of $\mathbf{H}$, while $j_1,\ldots,j_d$ represent its column indexing, $H_k(i_k,j_k)$ are $R_{k-1} \times R_k$ matrices, $k = 2,\ldots,d - 1$ and $R_0 = R_d = 1$. 

3
that $x^{(1)}$ can be accurately approximated by a TT representation of a small TT-rank $r$, i.e. it lies on the manifold of tensors of fixed TT-rank $r$:

$$
\mathcal{M}_r \equiv \{ x \in \mathbb{R}^n \mid \text{TT-rank}(x) = r \}.
$$

To obtain the eigenpair $(\epsilon^{(1)}, x^{(1)})$ we pose a Rayleigh quotient $\mathcal{R}(x)$ minimization problem on the low-parametric manifold $\mathcal{M}_r$ instead of the full $\mathbb{R}^n$ space:

$$
\min_{x \in \mathcal{M}_r} \mathcal{R}(x), \quad \mathcal{R}(x) \equiv \frac{\langle x, Hx \rangle}{\langle x, x \rangle}, \quad (3)
$$

where we assume that $H$ is symmetric. Since $\mathcal{M}_r$ forms a smooth manifold [8] one can utilize the so-called methods of Riemannian optimization, i.e. optimization on smooth manifolds.

One of the key concepts for the Riemannian optimization is tangent space, which consists of all tangent vectors to $\mathcal{M}_r$ at a given $x$ [10]. We will denote tangent space of $\mathcal{M}_r$ at $x$ by $T_x \mathcal{M}_r$. Tangent space can be viewed as the linearization of a manifold at the given point $x$. It has the same dimension as the manifold [10] and assuming that $r$ is small, it allows to locally replace the manifold with a low-dimensional linear space.

Provided a tangent space at hand we can discuss the simplest optimization method on $\mathcal{M}_r$ — the Riemannian gradient descent method, which consists of several steps and is illustrated in Fig. 1. Given the starting point $x_k$, the first step is to calculate the Riemannian gradient, which in this case is projection of the standard Euclidean gradient: $P_{x_k} \nabla \mathcal{R}(x_k)$, where $P_{x_k}$ denotes the orthogonal projection on $T_{x_k} \mathcal{M}_r$. This is simply the Rayleigh quotient steepest descent direction at $x_k$, restricted to the tangent space $T_{x_k} \mathcal{M}_r$. The second step is, given the search direction from $T_{x_k} \mathcal{M}_r$, map it back to the manifold to obtain $x_{k+1}$ of the iterative process. This is done by the smooth mapping $\tilde{T}_r(x, \cdot) : T_x \mathcal{M}_r \to \mathcal{M}_r$ called retraction. Note that in case of fixed-rank manifold $\mathcal{M}_r$ we use the retraction satisfying $\tilde{T}_r(x, \xi) \equiv T_r(x + \xi)$ [11] and call it truncation.

Similarly to the original gradient descent method, the convergence of such method can be slow and preconditioning has to be used. There are different ways how to account for a preconditioner in the Riemannian version of the iteration. We will use the idea considered in [12], where the preconditioner acts on the gradient and the result is projected afterwards

$$
x_{k+1} = \tilde{T}_r \left( x_k - P_{x_k} B^{-1} \nabla \mathcal{R}(x_k) \right),
$$

which for eigenvalue computations can be viewed as Riemannian generalization\(^6\) of the preconditioned inverse iteration (PINVIT). Indeed, one iteration of a classical version of PINVIT reads

$$
x_{k+1} = x_k - B^{-1} r_k,
$$

\(^6\)The search direction $P_{x_k} B^{-1} \nabla \mathcal{R}(x_k)$ is usually additionally multiplied by a constant $\tau_k$ to be found from the line search procedure $\mathcal{R}(x_k - \tau_k P_{x_k} B^{-1} \nabla \mathcal{R}(x_k)) \to \min_{\tau_k}$. This ensures the convergence in the presence of the nonlinear mapping $\tilde{T}_r$.\]
The gradient is projected on the tangent space and then $x_k$ is moved in the direction of the projected gradient and afterwards is retracted to the manifold: $x_{k+1} = T_r (x_k - P_{x_k} \nabla R(x_k))$.

where $r_k = Hx_k - \mathcal{R}(x_k)x_k$ denotes the residual, which is proportional to the gradient of the Rayleigh quotient:

$$\nabla \mathcal{R}(x) = \frac{2}{\langle x, x \rangle} (Hx - \mathcal{R}(x)x) .$$

Note that to avoid growth of $\|x_k\|$ additional normalization $x_k := x_k / \|x_k\|$ is done after each iteration to ensure $\langle x_k, x_k \rangle = 1$.

We could have restricted ourselves to the case of PINVIT (4), but to get faster convergence we utilize a superior method — locally optimal preconditioned conjugate gradients (LOPCG) and its block version (LOBPCG) to calculate several eigenvalues (see Sec. 3). To our knowledge the Riemannian version of LOPCG was not considered in the literature, so we provide it here. According to the classical LOBPCG the search direction $p_k$ is a linear combination of the preconditioned gradient and $p_{k-1}$. In the Riemannian setting instead of the preconditioned gradient $B^{-1}\nabla \mathcal{R}(x_k)$ we consider its projected analog $P_{x_k}B^{-1}\nabla \mathcal{R}(x_k) \in T_{x_k}M_r$. However, the problem is that $p_{k-1} \notin T_{x_k}M_r$, so similarly to [13] we use another important concept from the differential geometry – vector transport, which is a mapping from $T_{x_{k-1}}M_r$ to $T_{x_k}M_r$ satisfying certain properties [14]. Since $M_r$ is an embedded submanifold of $\mathbb{R}^{n_d}$, the orthogonal projection from $T_{x_{k-1}}M_r$ to $T_{x_k}M_r$ can be used as a vector transport [14], so that

$$p_{k+1} = c_1 P_{x_k}B^{-1}\nabla \mathcal{R}(x_k) + c_2 P_{x_k}p_k,$$

and

$$x_{k+1} = T_r (x_k + p_{k+1})$$
where $c_1, c_2$ are constants to be found from

$$
(c_1, c_2) = \arg \min_{(c_1, c_2)} \mathcal{R}(x_k + \zeta_1 P_{x_k} B^{-1} \nabla \mathcal{R}(x_k) + \zeta_2 P_{x_k} p_k)
$$

(7)

Note that we have omitted $\mathcal{T}_r$ in the optimization procedure. This allows to solve (7) exactly. Despite in numerical computations we have never faced the problem of functional increase with the omitted $\mathcal{T}_r$, additional line-search procedure could be introduced to ensure convergence.

2.3. Complexity reduction for computations on $\mathcal{M}_r$

One of the key benefits of the Riemannian optimization on $\mathcal{M}_r$ is that it allows to avoid the rank growth. This is particularly important in low-rank tensor computations due to the strong rank dependence of tensor methods, which sometimes is called the curse of the rank. The crucial property that allows for a significant speed-up of computations in the Riemannian approach is that TT-ranks of any vector from a tangent space of $\mathcal{M}_r$ has ranks at most $2r$. Since tangent space is a linear space, linear combination of any number of vectors from one tangent space also has rank at most $2r$. As a result, $p_{k+1}$ in (5) is of rank at most $2r$ and for the already computed $p_{k+1}$ the computation of (6) is inexpensive. By contrast, if in (6) we omit $P_{x_k}$, i.e. no Riemannian optimization approach is used, then this leads to complexity increase since matrix-vector multiplications considerably increase TT-ranks. Moreover, one can show that in (7) to find $c_1$ and $c_2$ scalar products of TT-tensors have to be computed. The calculation of scalar products of two vectors belonging to the same tangent space is of less complexity than of two general TT-tensors of the same rank. We provide details about the implementation of the Riemannian optimization for TT-manifolds in Sec. 4 after general description of the proposed method.

3. The proposed method

The main goal of the paper is to consider a problem of finding several eigenvalues and the corresponding eigenvectors. For convenience we rewrite (1) in the block form

$$
HX = X\Lambda, \quad X = [x^{(1)}, \ldots, x^{(b)}], \quad \Lambda = \text{diag}(\epsilon^{(1)}, \ldots, \epsilon^{(b)}),
$$

additionally assuming $\epsilon^{(1)} \leq \epsilon^{(2)} \leq \cdots \leq \epsilon^{(b)} \neq \epsilon^{(b+1)}$. Then the problem under the TT-rank constraint on $x^{(\alpha)}$, $\alpha = 1, \ldots, b$ can be reformulated as trace minimization:

$$
\begin{align*}
\text{minimize} & \quad \text{Tr}(X^\dagger HX) \\
\text{subject to} & \quad X^\dagger X = I_b \\
& \quad x^{(i)} \in \mathcal{M}_r, \quad i = 1, \ldots, b,
\end{align*}
$$

(8)

where $\text{Tr}(\cdot)$ denotes trace of a matrix.
3.1. Speeding-up computation of the smallest eigenvalue

As a starting point to generalize (6) to the block case assume that we are first aiming at finding the first eigenvalue. The iteration (6) can be improved by performing additional subspace acceleration by using more vectors in the tangent space of $\mathcal{M}_r$ at $x_k^{(1)}$. In particular, we suggest using the LOBPCG method and project all arising vectors to the tangent space $\mathcal{V}_k^{(1)}$:

$$R_k = HX_k - X_k \Lambda_k, \quad \Lambda_k = \text{diag} \left( \Re \left( x_k^{(1)} \right), \ldots, \Re \left( x_k^{(b)} \right) \right)$$

$$P_{k+1} = P_{x_k^{(1)}} B^{-1} R_k C_2 + P_{x_k^{(1)}} P_k C_3$$

$$X_{k+1} = T_r \left( P_{x_k^{(1)}} X_k C_1 + P_{k+1} \right),$$

where the truncation operator $T_r$ is applied independently to each column of the matrix and $C_i \in \mathbb{R}^{b \times b}, i = 1, 2, 3$ are matrices of coefficients to be found from trace minimization. In particular, introducing notation

$$V_k = P_{x_k^{(1)}} \begin{bmatrix} X_k & R_k & P_k \end{bmatrix} \in \mathbb{R}^{n^d \times 3b}, \quad C = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \in \mathbb{R}^{3b \times b},$$

we have

$$\begin{align*}
\text{minimize} & \quad \text{Tr} (C^T (V_k^T H V_k) C), \\
\text{subject to} & \quad C^T (V_k^T V_k) C = I_b,
\end{align*}$$

which reduces to a classical generalized eigenvalue problem of finding $b$ smallest eigenvalues $\theta_1, \ldots, \theta_b$ and corresponding eigenvectors of the matrix pencil $V_k^T H V_k - \theta V_k^T V_k$ of size $3b \times 3b$.

Since we use projection to the same tangent space, there is no rank growth even for large $b$ and hence the application of $T_r$ is inexpensive. Moreover, we expect that to achieve a given accuracy of $\epsilon^{(1)}$ the number of iterations for (9) is less than the number of iterations for (6) thanks to the additional subspace acceleration.

3.2. Riemannian alternating projection method

Similarly to (9) all column vectors of $[X_k, B^{-1} R_k, P_k]$ can be projected to the tangent space $T_{x_k^{(t_k)}} \mathcal{M}_r$ of $x_k^{(t_k)}$ for some integer $1 \leq t_k \leq b$, which not necessarily equals 1. However, there is no evidence that all eigenvectors except for the $t_k$-th one can be accurately approximated using $T_{x_k^{(t_k)}} \mathcal{M}_r$. Therefore, we propose to use $T_{x_k^{(t_k)}} \mathcal{M}_r$ to search for the correction to the already found approximations of $X_k$ as follows:

$$R_k = HX_k - X_k \Lambda_k,$n

$$P_{k+1} = P_{x_k^{(t_k)}} B^{-1} R_k C_2 + P_{x_k^{(t_k)}} P_k C_3$$

$$X_{k+1} = T_r \left( X_k \text{diag}(c) + P_{x_k^{(t_k)}} X_k C_1 + P_{k+1} \right),$$

7Note that $P_{x_k^{(1)}} x_k^{(1)} = x_k^{(1)}$. 

7
where $c \in \mathbb{R}^b$ and $\text{diag}(c)$ denotes diagonal $b \times b$ matrix with the vector $c$ on the diagonal. By forcing $X_k$ to be multiplied by a diagonal matrix $\text{diag}(c)$ instead of a general square matrix, we reduce computational cost of the method. Indeed, in this case we avoid calculating linear combinations of the columns of $X_k$ which do not belong to the same tangent space.

Note that we may also write

$$X_{k+1} = T_c (X_k \text{diag}(c) + V C),$$

where

$$V_k = P_{x_k^{(\tau_k)}} [X_k, B^{-1} R_k, P_k],$$

and the matrices of coefficients

$$C = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix}, \quad C_j \in \mathbb{R}^{b \times b}, \quad j = 1, 2, 3$$

and $c \in \mathbb{R}^b$ are to be found from the trace minimization problem

\begin{align}
\text{minimize}_{c, C} & \quad \text{Tr} \left[ (X_k \text{diag}(c) + V_k C)^\top H (X_k \text{diag}(c) + V_k C) \right] \\
\text{subject to} & \quad (X_k \text{diag}(c) + V_k C)^\top (X_k \text{diag}(c) + V_k C) = I_b
\end{align}

or equivalently

\begin{align}
\text{minimize}_{c, C} & \quad \text{Tr} \left( \begin{bmatrix} \text{diag}(c) \\ C \end{bmatrix}^\top \begin{bmatrix} X_k^\top H X_k & X_k^\top H V_k \\ V_k^\top H X_k & V_k^\top H V_k \end{bmatrix} \begin{bmatrix} \text{diag}(c) \\ C \end{bmatrix} \right) \\
\text{subject to} & \quad \begin{bmatrix} \text{diag}(c) \\ C \end{bmatrix}^\top \begin{bmatrix} X_k^\top X_k & X_k^\top V_k \\ V_k^\top X_k & V_k^\top V_k \end{bmatrix} \begin{bmatrix} \text{diag}(c) \\ C \end{bmatrix} = I_b
\end{align}

which because of the diagonal constraint does not boil down to a standard generalized eigenvalue problem. Due to the description technicality of solution of (13), we postpone it to Section 5.

The convergence of the proposed method depends on the choice of the integer sequence $\{t_1, t_2, \ldots, t_n, \ldots\}$. We call the strategy that in a certain way chooses the tangent space on each iteration the tangent space schedule. Note that we could have found all eigenvalues one by one, i.e. $t_1 = \cdots = t_{k_1^1} = 1$, then $t_{k_1^1} = \cdots = t_{k_1^1 + k_2^2} = 2$ and so on, where $k_\alpha^\delta$ is the number of iterations for $\mathcal{R}(x^{(\alpha)})$ to achieve accuracy $\delta$. This strategy, however, for a large number of eigenvalues $b$ requires a lot of iterations to be done. Therefore, we utilize strategies that do not require all tangent spaces to be chosen at least ones. We found that although the random choice (discrete uniform
distribution) of \( t_i \): \( 1 \leq t_i \leq b \) already ensures convergence in most of the cases, the strategy

\[
t_k = \arg \max_i \left| \frac{\mathcal{R}(x^{(i)}_{k-1}) - \mathcal{R}(x^{(i)}_k)}{\mathcal{R}(x^{(i)}_k)} \right|
\]  

(14)
in which we choose eigenvalue with the current slowest convergence, yields more reliable results for larger number of eigenvalues. Before running the adaptive strategy for \( t_k \) we choose \( t_1 = \cdots = t_{k_0} = 1 \) while convergence criteria for \( \mathcal{R}(x^{(i)}) \) is not fulfilled. This corresponds to (9) instead of (11) and hence, speeds up computations. The implementation details of the iteration (11) will be given in the next section and the algorithm for the computation of coefficients will be described later in Section 5.

4. Implementation of tensor operations

In this section we provide a brief description of implementation details for Riemannian optimization on \( \mathcal{M}_r \) with complexity estimates and summarize the algorithm.

4.1. Representation of tangent space vectors

We start with the description of vectors from tangent spaces, as they are the main object we are working with. Let \( x \) be given by its TT decomposition (2). It is known [4] that by a sequence of QR decomposition of cores, (2) can be represented both as

\[
X_{i_1 \ldots i_d} = U_1(i_1)U_2(i_2)U_3(i_3) \ldots U_d(i_d),
\]

(15)
where \( U_k = \{U_k(i_k)\}_{i_k=1}^n \in \mathbb{R}^{r_{k-1} \times n \times r_k}, k = 1, \ldots, d-1 \) being reshaped into matrices \( \mathcal{M}_k^L(V_k) \) of size \( r_{k-1} \times r_k \) have orthogonal columns, \( r_1 = \cdots = r_{d-1} = r \) and \( r_0 = r_d = 1 \). Here \( \mathcal{M}_k^L : \mathbb{R}^{r_{k-1} \times n \times r_k} \to \mathbb{R}^{r_{k-1} \times n \times r_k} \) denotes the matricization operator, that maps first two indices of considered three-dimensional arrays into a single one. Similarly, we may have

\[
X_{i_1 \ldots i_d} = V_1(i_1)V_2(i_2)V_3(i_3) \ldots V_d(i_d),
\]

(16)
where \( V_k = \{V_k(i_k)\}_{i_k=1}^n \in \mathbb{R}^{r_{k-1} \times n \times r_k}, k = 2, \ldots, d \) being reshaped into matrices \( \mathcal{M}_k^R(V_k) \) of size \( r_{k-1} \times nr_k \) have orthogonal rows. Representations (15) and (16) are called correspondingly left- and right-orthogonalizations of TT-representation (2), which can be performed with \( \mathcal{O}(dnr^3) \) complexity. Using these notations, one possible way to parametrize tangent space \( T_x \mathcal{M}_r \) is as follows. Any \( \xi = \text{vec}(\Xi) \in T_x \mathcal{M}_r \) can be represented as

\[
\Xi_{i_1 \ldots i_d} = \delta G_1(i_1)V_2(i_2)V_3(i_3) \ldots V_d(i_d) + U_1(i_1)\delta G_2(i_2)V_3(i_3) \ldots V_d(i_d) + \cdots + U_1(i_1)U_2(i_2)U_3(i_3) \ldots \delta G_d(i_d),
\]

(17)
where cores \( \delta G_k = \{\delta G_k(i_k)\}_{i_k=1}^n \in \mathbb{R}^{r_{k-1} \times n \times r_k} \) additionally satisfy the following gauge conditions to ensure uniqueness\(^8\) of the representation:

\[
(\mathcal{M}_k^L(\delta G_k))^\top \mathcal{M}_k^L(U_k) = 0, \quad k = 1, \ldots, d - 1.
\]

\(^8\)To see that the tangent space is overparametrized by (17) note that the number of parameters in all the \( G_k(i_k) \) is \( (d - 2)nr^2 + 2nr \), while the dimension of the manifold \( \mathcal{M}_r \) and hence of the tangent space is smaller: \( \dim(T_x \mathcal{M}_r) = (d - 2)nr^2 + 2nr - (d - 1)r^2 \) [8].
To show that any vector from a tangent space has TT-rank not greater than 2r we note that for (17) the explicit formula holds:

$$\Xi_{i_1 \ldots i_d} = S_1(i_1)S_2(i_2) \ldots S_d(i_d),$$

where for $k = 2, \ldots, d - 1$

$$S_1(i_1) = \begin{bmatrix} \delta G_1(i_1) & U_1(i_1) \end{bmatrix}, \quad S_k(i_k) = \begin{bmatrix} V_k(i_k) \\ \delta G_k(i_k) & U_k(i_k) \end{bmatrix}, \quad S_d(i_d) = \begin{bmatrix} V_d(i_d) \\ \delta G_d(i_d) \end{bmatrix},$$

which can be verified by direct multiplication of all $S_k(i_k)$. From (17) it is also simply noticeable that if $\xi^{(1)}, \xi^{(2)} \in T_xM_r$ given by $\delta G_k^{(1)}(i_k)$ and $\delta G_k^{(2)}(i_k)$, $k = 1, \ldots, d$ correspondingly, then their linear combination $(\alpha \xi^{(1)} + \beta \xi^{(2)}) \in T_xM_r$ is given by $(\alpha \delta G_k^{(1)}(i_k) + \beta \delta G_k^{(2)}(i_k))$, $\alpha, \beta \in \mathbb{R}$.

### 4.2. Projection to a tangent space

Representation (17) allows to obtain explicit formulas for $\delta G_k(i_k)$ of $\xi = P_{T_xM_r}z$ without inversions of possibly ill-conditioned matrices [12]:

$$\text{vec}(\delta G_k) = (I_{rk-1} \otimes (I_{rk} - M^L(U_k) \cdot M^L(U_k)^T)) (X_{>k}^\top \otimes I_n \otimes X_{<k}^\top) z,$$

$$\text{vec}(\delta G_d) = (I_n \otimes X_{<d}^\top) z,$$

where $k = 1, \ldots, d - 1$ and

$$X_{<k} = [G_1(i_1) \ldots G_{k-1}(i_{k-1})] \in \mathbb{R}^{n^{k-1} \times r},$$

$$X_{>k} = [G_{k+1}(i_{k+1}) \ldots G_d(i_d)] \in \mathbb{R}^{n^{d-k} \times r}.$$  

Matrices $X_{<k}, X_{>k}$ are never formed explicitly and used here for the ease of notation. The complexity of projecting a vector $z$ given by its TT decomposition with TT-rank$(z) = r_z$ is $O(dnr_z^2 r^2)$.  

One of the most time-consuming operations arising in the algorithm is a projection to a tangent space of a matrix-vector product. Suppose that both $H$ and $y$ are given in the TT format with TT-ranks $R$ and $r_y$. Then $z = Hy$ can also be represented in the TT format with the TT-rank bounded from above as $r_y R$ [4] since

$$Z_{i_1 \ldots i_d} = Z_1(i_1) \ldots Z_d(i_d),$$

$$Z_k(i_k) = \sum_{j_k=1}^{n} H_k(i_k, j_k) \otimes G_k(i_k) \in \mathbb{R}^{R_{k-1} R_{k-1} R_k R_k}.$$  

Calculation of $Z_k(i_k)$ can be done in $O(n^2 R^2 r_y^2)$ complexity. Once $Z_k(i_k)$ $k = 1, \ldots, d$ are calculated, complexity of finding the TT representation of $P_{T_xM_r}z$ is $O(dnr_z^2 R^2) = O(dnr_y^2 R^2)$ as $r_z = r_y R$. This complexity can be additionally reduced down to $O(dn^2 r_y R^2)$ if we take care of the Kronecker-product structure of $Z_k(i_k)$. Moreover, since it does not require computation of $Z_k(i_k)$ explicitly, the storage is also less in this case.
4.3. Computation of inner products

Next thing that arises in the method, in particular in (13) is the computation of Gram matrices $V^\top V$, $V^\top HV$, $X^\top X$, $V^\top X$, $V^\top HV$, $V^\top HX$ and as we will see in Sec. 5 solution to (13) will involve only $\text{diag}(X^\top HX)$ instead of the full $X^\top HX$. Here $V$ consists of columns that belong to a single tangent space $T_x \mathcal{M}_r$ and $X$ are general tensors of TT-rank $r$.

Let us first discuss the computation of $V^\top V$. Let $v^{(\alpha)}$ be the $\alpha$-th column of $V$, $\alpha = 1, \ldots, 3b$ and be given by $\delta G_{x_k}(i_k) \equiv \delta G_{x_k}^{(\alpha)}(i_k)$ from (17). Then, thanks to gauge conditions (18), left orthogonality of $U$ and right orthogonality of $V$, we have

$$
\langle v^{(\alpha)}, v^{(\beta)} \rangle = \sum_{k=1}^{d} \delta G_{x_k}^{(\alpha)} \cdot \delta G_{x_k}^{(\beta)} \tag{19}
$$

where $\langle A, B \rangle_F \equiv \text{vec}(A)^\top \text{vec}(B)$ is the Frobenius inner product. Therefore, the complexity of computing $V^\top V$ is $O(b^2dnr^2)$. The computation of $V^\top HV$ is done using the following trick. Note that all columns of $V$ belong to a single tangent space $T_x \mathcal{M}_r$. Hence,

$$
V^\top HV = (P_{T_x \mathcal{M}_r}, V)^\top HV = V^\top (P_{T_x \mathcal{M}_r}, HV). \tag{20}
$$

As a result, we first compute $P_{T_x \mathcal{M}_r}, HV$ using the procedure described in Section 4.2. Then we compute the inner product of two vectors from the same tangent space as in (19). Thus, the complexity of finding $V^\top HV$ is $O(bdn^2r^2 + b^2dr^2)$. Similarly to (20) we have $V^\top HX = V^\top (P_{T_x \mathcal{M}_r}, HX)$ and $V^\top X = V^\top (P_{T_x \mathcal{M}_r}, X)$.

Computation of $X^\top X$ is a standard procedure, which consists of $b^2$ inner products of TT tensors. Since the complexity of inner product of two tensors of TT-rank $r$ is $O(dnr^3)$ [4], computing $b^2$ of inner products require $O(b^2dnr^3)$ floating point operations.

Finally, the computation of $x^{(\alpha)^\top} H x^{(\alpha)}$, $\alpha = 1, \ldots, b$ arising in $\text{diag}(X^\top HX)$ costs $O(ndRr^2(r+R))$. Eventually, computation of $\text{diag}(X^\top HX)$ has complexity $O(bndRr^2(r+nR))$.

4.4. Retraction computation

The final thing remains to compute to proceed to the next iteration after we have found the coefficients $c, C$ and $V_k$, is to retract the obtained vectors $X_k \text{diag}(c) + V_k C$ to the manifold $\mathcal{M}_r$. All columns $V_k$ are from the same tangent space at $x_k^{(t_k)}$, so the correction $(V_k C)[:, i]$ to each $x_k^{(i)}$, $i = 1, \ldots, b$ is of rank $2r$. Therefore, $x_k^{(i)} c[i] + (V_k C)[:, i]$ is of rank at most $2r$ for $i = t_k$ and $3r$ otherwise. The retraction is done by the TT-SVD algorithm [4] that consists of a sequence of QR and SVD decompositions applied to the unfolded tensor cores. It has complexity $O(dnr^3)$.

Note that the retraction is properly defined only for $i = t_k$, but we formally apply it to other vectors as well, as the coefficients $c, C$ were obtained to ensure the descent direction for all vectors. The descent direction is, however, chosen without regard to the retraction. This can be accounted for by introducing approximate line search, but numerical experiments showed this is in general redundant and $c, C$ already provide good enough approximation.
4.5. The algorithm description

Let us now discuss the iteration (11) step-by-step. For simplicity let us denote $P_k \equiv P_{x_k(t_k)}$. First, assuming we are given $P_k$, the calculation of $P_kP_k$ is done in $O(bdnr^3)$ complexity.

To calculate $P_k B^{-1} R_k$ term we split it into two parts:

$$P_k B^{-1} R_k = P_k B^{-1} H X_k - P_k B^{-1} X_k \Lambda_k.$$  

The $P_k B^{-1} X_k$ is a projected matrix vector product, which is calculated as described in Section 4.2 and costs $O(bdn r^2 R^2)$ operations. The term $P_k B^{-1} H X_k$ is more difficult to compute as it involves two sequential matrix-vector products. To deal with this term efficiently, we use the trick from [12] and assume that

$$B^{-1} = B_1 + \cdots + B_{\rho_B} \tag{21}$$

where $B_i, i = 1, \ldots, \rho_B$ are of TT-rank 1. This trick helps us thanks to the fact the multiplication of a TT-matrix of TT-rank 1 by a general TT-matrix does not change the rank of the latter. The assumption (21) holds for the preconditioner we use for the calculation of vibrational spectra of molecules, while for spin chains computation no preconditioner is used. Thus,

$$P_k B^{-1} H X_k = P_k B_1 H X_k + \cdots + P_k B_{\rho_B} H X_k \tag{22}$$

and hence the complexity of computing $P_k B^{-1} H X_k$ is $O(bdn^2 r^2 R^2 \rho_B)$. The truncation operation costs $O(bdn r^3)$, which is negligible compared to other operations. The overall algorithm is summarized in Algorithm 1.

5. Trace minimization problem for coefficients

Let us rewrite the problem (13) omitting index $k$ for simplicity

$$\min_{c,C, \mathcal{C}} \text{Tr} \left( \begin{bmatrix} \text{diag}(c) & X^\top H X & X^\top H V \\ C & V^\top H X & V^\top H V \end{bmatrix} \begin{bmatrix} \text{diag}(c) \\ C \end{bmatrix} \right) \quad \text{subject to} \begin{bmatrix} \text{diag}(c) & X^\top X & X^\top V \\ C & V^\top X & V^\top V \end{bmatrix} \begin{bmatrix} \text{diag}(c) \\ C \end{bmatrix} = \mathbf{I}_b \tag{23}$$

Unfortunately, this problem can not be reduced to a generalized eigenvalue problem, which can be solved by means of a reliable and optimized software packages. Therefore, to solve it we propose an iterative method. It is derived using the Lagrange multiplier method formally applied to the minimization problem. The Lagrange function of (23)
Algorithm 1 Low-Rank Riemannian Alternating Projection LOBPCG.

Require: TT-matrix $H$, initial guess $X_1 = [x_1^{(1)} \ldots x_1^{(b)}]$, where $x_i^{(1)}$ are TT-tensors, TT-rank $r$, convergence tolerance $\varepsilon$

Ensure: $X_k = [x_k^{(1)} \ldots x_k^{(b)}]$

1: Initialize $P_k = 0 \cdot X_1$
2: Set $t_1 = 1$
3: for $k = 1, 2, \ldots$ until converged do
4: \hspace{1em} if $\|P_{x_k^{(i)}}(Hx_k^{(i)} - \Re(x_k^{(i)}))\| > \varepsilon$ then
5: \hspace{2em} Set $t_k = 1$
6: \hspace{1em} else
7: \hspace{2em} Choose $t_k$ randomly or according to (14)
8: \hspace{2em} Calculate $P_k X_k$ and $P_k P_k$
9: \hspace{2em} Calculate $\Re(x_k^{(i)})$, $i = 1, \ldots, b$
10: \hspace{2em} Calculate $P_k B^{-1} R_k$ using (21) and (22)
11: \hspace{2em} Set $V_k = P_k [X_k, B^{-1} R_k, P_k]$ and calculate $V_k^\dagger V_k$, $V_k^\dagger HV_k$, $X_k^\dagger X_k$, $V_k^\dagger X_k$, $V_k^\dagger HV_k$, $V_k^\dagger HX_k$, diag($X_k^\dagger HX_k$) as described in Sec. 4.3
12: \hspace{1em} for $i = 1, \ldots, b$ do
13: \hspace{2em} $r_k^{(i)} = H x_k^{(i)} - \Re(x_k^{(i)}) x_k^{(i)}$
14: \hspace{2em} $c, C = \text{find\_coefficients}(V_k^\dagger V_k, V_k^\dagger HV_k, X_k^\dagger X_k, V_k^\dagger X_k, V_k^\dagger HV_k, V_k^\dagger HX_k, \text{diag}(X_k^\dagger HX_k))$, see Alg. 1.
15: \hspace{2em} $P_{k+1} = [P_k B^{-1} R_k, P_k P_k] C[:, b:3b]$
16: \hspace{2em} Calculate $X_{k+1} = X_k$ diag($c$) $+ P_k$ $X_k$ $C_1 + P_{k+1}$
17: Calculate $X_{k+1} := \mathcal{T}_r(X_{k+1})$

reads

$$
\mathcal{L}(c, C, \Lambda) = \frac{1}{2} \text{Tr} \left( \begin{bmatrix} \text{diag}(c) \\ \Lambda \end{bmatrix}^T \begin{bmatrix} X^\dagger HX & X^\dagger HV \\ V^\dagger HX & V^\dagger HV \end{bmatrix} \begin{bmatrix} \text{diag}(c) \\ \Lambda \end{bmatrix} - \begin{bmatrix} \text{diag}(c) \\ \Lambda \end{bmatrix}^T \begin{bmatrix} X^\dagger X & X^\dagger V \\ V^\dagger X & V^\dagger V \end{bmatrix} \begin{bmatrix} \text{diag}(c) \\ \Lambda \end{bmatrix} - I_b \right)
$$

Thanks to the symmetry of the constraint matrix we can consider $\Lambda^\dagger = \Lambda$ without the loss of generality. Then, the gradient of the Largangian reads

$$
\nabla_c \mathcal{L} = \text{diag}(X^\dagger HX)c + \text{diag}((X^\dagger HV)C - (X^\dagger V)CA)1 - (\Lambda \odot X^\dagger X)c,
$$

$$
\nabla_C \mathcal{L} = (V^\dagger HV)C + (V^\dagger HX)\text{diag}(c) - (V^\dagger V)\text{diag}(c)\Lambda - (V^\dagger V)CA,
$$

where $\text{diag}(A)$ denotes a diagonal matrix with the same diagonal as $A$, $1$ — vector of all ones of the corresponding size and $A \odot B$ — elementwise product of matrices $A, B$. 

13
of the same size. Thus, the critical point of the Lagrangian can be found from
\[
\text{diag}(X^\top H X)c + \text{diag}((X^\top H V)C)1 = \text{diag}((X^\top V)\Lambda)1 + (\Lambda \odot X^\top X)c,
\]
\[
(X^\top H V)C + (V^\top H X)\text{diag}(c) = (V^\top X)\text{diag}(c)\Lambda + (V^\top V)\Lambda C,
\]
\[
(X \text{diag}(c) + VC)^\top (X \text{diag}(c) + VC) = I_b,
\]
(24)

We put emphasis on the fact that the latter equation does not depend on the whole \(X^\top H X\) which is present in (23), but only on \(\text{diag}(X^\top H X)\) instead. This significantly reduces complexity of computations.

Equations (24) can be rewritten in the following form
\[
\begin{bmatrix}
(x^{(\alpha)})^\top H x^{(\alpha)} & (x^{(\alpha)})^\top H V \\
V^\top H x^{(\alpha)} & V^\top H V
\end{bmatrix}
\begin{bmatrix}
\zeta_{\alpha} \\
c_{\alpha}
\end{bmatrix}
= \lambda_{\alpha\alpha}
\begin{bmatrix}
(x^{(\alpha)})^\top x^{(\alpha)} & (x^{(\alpha)})^\top V \\
V^\top x^{(\alpha)} & V^\top V
\end{bmatrix}
\begin{bmatrix}
\zeta_{\alpha} \\
c_{\alpha}
\end{bmatrix}
+ \sum_{\beta=1, \beta \neq \alpha}^b \lambda_{\alpha\beta}
\begin{bmatrix}
(x^{(\alpha)})^\top x^{(\beta)} & (x^{(\alpha)})^\top V \\
V^\top x^{(\beta)} & V^\top V
\end{bmatrix}
\begin{bmatrix}
\zeta_{\beta} \\
c_{\beta}
\end{bmatrix}
\]
(25)

where \(\delta_{\alpha\beta}\) is the Kronecker delta and \(c = [\zeta_1 \ldots \zeta_b]^\top\). For convenience we introduce the following notations:
\[
A_{\alpha} =
\begin{bmatrix}
(x^{(\alpha)})^\top H x^{(\alpha)} & (x^{(\alpha)})^\top H V \\
V^\top H x^{(\alpha)} & V^\top H V
\end{bmatrix},
\]
\[
G_{\alpha\beta} =
\begin{bmatrix}
(x^{(\alpha)})^\top x^{(\beta)} & (x^{(\alpha)})^\top V \\
V^\top x^{(\beta)} & V^\top V
\end{bmatrix},
\]
(26)
\[
s_{\alpha} = 
\begin{bmatrix}
\zeta_{\alpha} \\
c_{\alpha}
\end{bmatrix}.
\]

Using these notations (25) reads
\[
A_{\alpha}s_{\alpha} = \lambda_{\alpha\alpha}G_{\alpha\alpha}s_{\alpha} + \sum_{\beta=1, \beta \neq \alpha}^b \lambda_{\alpha\beta}G_{\alpha\beta}s_{\beta},
\]
(27)
\[
s_{\alpha}^\top G_{\alpha\beta}s_{\beta} = \delta_{\alpha\beta}, \quad \alpha, \beta = 1, \ldots, b.
\]
To solve (27) we propose the following iterative process:

\[ A_\alpha s^{(k+1)}_\alpha = \lambda^{(k+1)}_{\alpha\alpha} G_{\alpha\alpha} s^{(k+1)}_\alpha + \sum_{\beta < \alpha} \lambda^{(k+1)}_{\alpha\beta} G_{\alpha\beta} s^{(k+1)}_\beta + \sum_{\beta > \alpha} b^{(k)} G_{\alpha\beta} s^{(k)}_\beta, \]  

(28)

\[
\begin{align*}
(s^{(k+1)}_\alpha)^\top G_{\alpha\beta} s^{(k+1)}_\beta &= \delta_{\alpha\beta}, & \alpha \leq \beta, \\
(s^{(k+1)}_\alpha)^\top G_{\alpha\beta} s^{(k)}_\beta &= 0, & \alpha > \beta.
\end{align*}
\]

To reduce (28) to a sequence of generalized eigenvalue problems let us start with \( \alpha = 1 \) and proceed to \( \alpha = b \). For \( \alpha = 2, \ldots, b - 1 \) define

\[
S^{(k)}_{\alpha} = \begin{bmatrix}
G_{\alpha,1} s^{(k+1)}_1 & \cdots & G_{\alpha,\alpha-1} s^{(k+1)}_{\alpha-1} & G_{\alpha,\alpha+1} s^{(k)}_{\alpha+1} & \cdots & G_{\alpha,b} s^{(k)}_b
\end{bmatrix},
\]

\[
S^{(k)}_{1} = \begin{bmatrix}
G_{1,2} s^{(k)}_2 & \cdots & G_{1,b} s^{(k)}_b
\end{bmatrix},
\]

\[
S^{(k)}_{b} = \begin{bmatrix}
G_{b,1} s^{(k+1)}_1 & \cdots & G_{b,b-1} s^{(k+1)}_{b-1}
\end{bmatrix},
\]

with null spaces

\[
\mathcal{N}^{(k)}_\alpha = \text{Null} \left( S^{(k)}_\alpha \right).
\]

We also introduce \( Q_\alpha \) of size \((3b+1) \times \dim(\mathcal{N}^{(k)}_\alpha)\) whose columns form an orthonormal basis in \( \mathcal{N}^{(k)}_\alpha \). Accounting for the fact that \( s^{(k+1)}_\alpha \in \mathcal{N}^{(k)}_\alpha \), multiplying the first equation of (28) by \( Q_\alpha^\top \) and introducing \( z^{(k+1)}_\alpha = Q_\alpha z^{(k+1)}_\alpha \) we arrive at the sequence of generalized eigenvalue problems

\[
(Q_\alpha^\top A_\alpha Q_\alpha) z^{(k+1)}_\alpha = \lambda^{(k+1)}_{\alpha\alpha} (Q_\alpha^\top G_{\alpha\alpha} Q_\alpha) z^{(k+1)}_\alpha,
\]

(29)

\[
(z^{(k+1)}_\alpha)^\top (Q_\alpha^\top G_{\alpha\alpha} Q_\alpha) z^{(k+1)}_\alpha = 1,
\]

in which we are searching for the smallest eigenvalue \( \lambda^{(k+1)}_{\alpha\alpha} \) and the corresponding eigenvector \( z^{(k+1)}_\alpha \).

The matrix \( Q_\alpha \) is calculated using SVD of \( S^{(k)}_\alpha = U \Gamma V^\top \) by choosing \( b \) plus number of zero singular values of last columns of \( U \). The algorithm described above is summarized in Algorithm 2. Given matrices \( V, V^\top H V, X^\top X, V^\top X, V^\top H V, V^\top H X, \) diag(\( X^\top H X \)) the overall complexity to find the coefficients is \( O(b^3) \). This is due to the fact that we calculate full eigendecomposition that costs \( O(b^2) \) for each \( \alpha = 1, \ldots, b \). Note that we only need one eigenvalue and one eigenvector for each \( \alpha \). This does not significantly increase the complexity of the overall algorithm (including tensor operations) for moderate \( b \) up to 100. Arguably \( O(b^3) \) complexity may be reduced down to \( O(b^3) \) using low-rank update of matrix decompositions. We, however, do not consider it here as it does not significantly influence overall performance of the method on the considered range of \( b \) and requires a lot of technical details to be presented.
Algorithm 2 find_coefficients function from Algorithm 1

Require: Matrices \( V^TV, V^THV, X^TX, V^THX, V^HX, \text{diag}(X^HX) \), initial guesses \( s^{(1)}_b \in \mathbb{R}^{3b+1} \), \( \beta = 2, \ldots, b \), maximum number of iterations \( K \)

Ensure: \( c, C \) that approximate the solution to (23)

1: Assemble \( A_\alpha, G_\alpha, \alpha, \beta = 1, \ldots, b \) according to (26)

2: \( S^{(1)}_1 = [G_{1,2} s^{(1)}_2 \ldots G_{1,b} s^{(1)}_b] \)

3: for \( k = 1, 2, \ldots, K \) do

4: for \( \alpha = 1, 2, \ldots, b \) do

5: Compute SVD: \( S^{(k)}_\alpha = U \text{diag}(\sigma)V^t \)

6: Set \( Q_\alpha := U[:,2b + 1 - \#\text{zeros}(\sigma) : 3b + 1], \)

7: Compute \( Q^\dagger_\alpha A_\alpha Q_\alpha, Q^\dagger_\alpha G_\alpha Q_\alpha \)

8: Find \( z^{(k+1)}_\alpha \) from (29)

9: Compute \( s^{(k+1)}_\alpha = Q_\alpha z^{(k+1)}_\alpha \)

10: if \( \alpha \neq b \) then

11: Calculate \( S^{(k)}_{\alpha+1} = [G_{\alpha+1,\beta} z^{(k+1)}_\beta]_{\beta=1,\beta \neq \alpha+1} \)

12: \( S^{(k+1)}_1 = [G_{1,2} s^{(k+1)}_2 \ldots G_{1,b} s^{(k+1)}_b] \)

6. Numerical experiments

In this section, we numerically assess the proposed method. For all experiments we use NVIDIA DGX-1 station with 8 V100 GPUs (only one of which is used at a time) and Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz with 80 logical cores.

6.1. Molecule vibrational spectra

One of the applications we consider is the computation of vibrational spectra of molecules. In particular, we consider Hamiltonians given as

\[
\mathcal{H} = -\frac{1}{2} \sum_{i=1}^{d} \omega_i \frac{\partial^2}{\partial q_i^2} + V(q_1, \ldots, q_d),
\]

(30)

where \( V \) denotes potential energy surface (PES). The proposed method is applicable if the Hamiltonian can be represented in the TT-format with small TT-ranks, which holds, e.g. for sum-of-product PES. Therefore, we consider PES given in the polynomial form as is used in [15]:

\[
V(q_1, \ldots, q_d) = \frac{1}{2} \sum_{i=1}^{d} \omega_i q_i^2 + \frac{1}{6} \sum_{i=1}^{d} \sum_{j=1}^{d} \sum_{k=1}^{d} \phi^{(3)}_{ijk} q_i q_j q_k
\]

\[
+ \frac{1}{24} \sum_{i=1}^{d} \sum_{j=1}^{d} \sum_{k=1}^{d} \sum_{l=1}^{d} \phi^{(4)}_{ijkl} q_i q_j q_k q_l.
\]
To discretize the problem we use the discrete variable representation (DVR) scheme on the tensor product of Hermite meshes [16]. The complexity of assembling the discretized PES $V$ is $O\left(\left(nnz\left(\phi^{(3)}_{ijk}\right) + nnz\left(\phi^{(4)}_{ijkl}\right)\right)dn^2R^3\right)$ and is negligibly small compared with the cost of one iteration of the iterative process. As a preconditioner we use approximate inversion of the harmonic part of (30) using [18] and formula (22) for efficient computations.

We calculate vibrational spectra of two molecules: acetonitrile (CH$_3$CN) and ethylene oxide (C$_2$H$_4$O) for which $d = 12$ and $d = 15$ correspondingly. The potential energy surfaces for these molecules were kindly provided by the group of Prof. Tucker Carrington. Table 1 contains TT-ranks of Hamiltonians of these two molecules with the mode order sorted in correspondence with the ascending order of $\omega_i$, $i = 1, \ldots, d$ with mode sizes chosen according to [19] for CH$_3$CN and $n = 15$ for all modes for C$_2$H$_4$O. Initial guess is chosen from the solution of the harmonic part of the Hamiltonian and can be derived analytically.

Table 1: TT-ranks of the Hamiltonian with $\varepsilon = 10^{-12}$ truncation tolerance.

|        | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ | $R_{11}$ |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|
| CH$_3$CN | 5     | 9     | 14    | 21    | 25    | 26    | 24    | 18    | 15    | 8        | 5        |
| C$_2$H$_4$O | 5    | 11    | 17    | 21    | 23    | 25    | 27    | 28    | 25    | 23       | 21       | 16       | 11       | 5       |
Figure 2: CH$_3$CN molecule, $r = 10$. Convergence plots of 40 eigenvalues for four different tangent space schedule scenarios: first tangent space is chosen for all the iterations, optimal choice of schedule (not practical), proposed scheduling by (14) and a random choice of tangent space.

Identifying convergence of larger eigenvalues. By contrast, the random choice of tangent spaces provides faster convergence of the smallest eigenvalues, while the error of larger eigenvalues fluctuates. In the following numerical experiments we choose the “argmax” strategy as a more reliable one and combine it with the usage of the first tangent space for the first 20 iterations.

In Figure 3a we provide computational time of one iteration with respect to the number of computed eigenvalues $b$ for acetonitrile molecule CH$_3$CN and $r = 25$. This figure illustrates that the tensor part of computations described in Section 4 scales effectively linearly in the given range of $b$. When $b$ approaches 100, the problem of finding coefficients (Alg. 2) starts dominating. We plan to improve the complexity of this part of the method in the future work.

Table 2 represents accuracies of energy levels for different methods for CH$_3$CN and C$_2$H$_4$O molecules. We measure the mean absolute error (MAE) of energy levels:

$$\text{MAE} = \frac{1}{b} \sum_{i=1}^{b} \left| \bar{\epsilon}^{(i)} - \epsilon^{(i)}_{\text{ref}} \right|,$$

where $\bar{\epsilon}^{(i)}$, $i = 1, \ldots, b$ are the calculated energy levels and $\epsilon^{(i)}_{\text{ref}}$ are accurate energy levels calculated in [17] for CH$_3$CN and in [20] for C$_2$H$_4$O. For comparison purposes we also provide timings of the methods. The timings of LRRAP LOBPCG and MP LOBPCG are measured on the same machine, whereas timings of hierarchical rank reduced block power method (H-RRBPM) are taken from [21]. We do not provide timings of H-RRBPM and HI-RRBPM for C$_2$H$_4$O since the calculations in [20] were done for different $b$ ($b = 200$) compared with $b = 35$ for LRRAP LOBPCG and MP
Figure 3: GPU energy spectra calculation for acetonitrile molecule CH$_3$CN (a) and for spin lattice with $d = 40$ spins (b). Time per one iteration vs. $b$ – number of computed eigenvectors, TT-rank for the both cases is 25. Plot illustrates linear scaling of tensor operations complexity with respect to $b$. For large $b$ problem of finding coefficients (Alg. 2) starts dominating.

LOBPCG. Table 2 illustrates that the method is capable of producing accurate results with speedups up to 20 times on GPU compared with CPU. We note that for larger molecules additional speedup can be obtained using several GPUs simultaneously. In contrast to the MP LOBPCG method [17] the proposed method is capable of producing comparably accurate results faster both on CPUs and GPUs. The fact is that for the considered examples the cost of each iteration of the proposed method is considerably less than the cost of an iteration of the MP LOBPCG method, although LRRAP LOBPCG requires more iterations. Moreover, MP LOBPCG introduces errors after such operations as calculation of linear combinations of vectors and matrix-vector products due to truncation errors, which eventually leads to lower accuracies of the result. At the same time in the LRRAP approach corrections on each iteration belong to a single tangent space, so no rank growth occurs. Moreover, before the retraction all tensor calculations are done with the machine precision. Table 2 also illustrates that LRRAP LOBPCG for $r = 25$ is more accurate than the most accurate basis (basis-3 or “b3” for short) considered in [21]. Acceleration on GPUs allows to get additional gain in time w.r.t. H-RRBPM method. We note, however, that the recently proposed HI-RRBPM [20] and its improved version [22] are superior to H-RRBPM. We also note that accuracy of eigenvalues can be additionally improved using the manifold-projected simultaneous inverse iteration (MP SII) proposed for the TT-format in [17]. This strategy was used to correct eigenvalues obtained using MP LOBPCG with small $r$.

6.2. Spin chains

As a second application, we consider Heisenberg model for one-dimensional lattices of spins. The goal it to compute minimal energy levels of Hamiltonian

$$ H = \sum_{i=1}^{d-1} \left( S^{(i)}_x S^{(i+1)}_x + S^{(i)}_y S^{(i+1)}_y + S^{(i)}_z S^{(i+1)}_z \right), $$

(32)

with

$$ S^{(i)}_\alpha = I \otimes \cdots \otimes I \otimes S_\alpha \otimes I \otimes \cdots \otimes I, \quad \alpha = x, y, z, $$
Table 2: Mean absolute error (MAE) (31) and computation times of different methods. Reference energies $\epsilon^{(i)}_{\text{ref}}$ are taken from [17] ($r = 40$) for CH$_3$CN and from [20] (setting D) for C$_2$H$_4$O. $b = 84$ energy levels are calculated for CH$_3$CN and $b = 35$ energy levels are calculated for molecule C$_2$H$_4$O. Note that timings of H-RRBPM method are taken from [21], so computations were performed on a different machine. MAE error for H-RRBPM and HI-RRBPM is measured for first $b$ eigenvalues.

|       | LRRAP LOBPCG (proposed) | MP LOBPCG [17] | H-RRBPM [21] | HI-RRBPM [20] |
|-------|-------------------------|----------------|--------------|---------------|
| CH$_3$CN | $r=15$ | $r=25$ | $r=15$ | $r=25$ | $b1$ | $b2$ | $b3$ |
| MAE, cm$^{-1}$ | 0.4 | 0.05 | 0.5 | 0.07 | 2.6 | 0.9 | 0.2 |
| GPU time | 51 s | 114 s | 12 min | 26 min | 27 min | 45 min | 44 s | 11 min | 3.2 h |
| C$_2$H$_4$O | $r=25$ | $r=35$ | $r=25$ | $r=35$ | E (H-RRBPM, [20]) | A | B | C |
| MAE, cm$^{-1}$ | 1.5 | 0.3 | 1.7 | 0.4 | 0.78 | 1.0 | 0.6 | 0.2 |
| GPU time | 92 s | 129 s | 25 min | 41 min | 32 min | 64 min |
| CPU time | 25 min | 41 min | 32 min | 64 min |

where $I$ is 2 $\times$ 2 identity matrix and $S_\alpha$, $\alpha = x, y, z$ are elementary Pauli matrices

$$S_x = \frac{1}{2} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad S_y = \frac{i}{2} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad S_z = \frac{1}{2} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. $$

It can be verified numerically that TT-rank of the Hamiltonian (32) is bounded from above by 5.

Similarly to Sec. 6.1 in Figure 4 we plot convergence of each of the eigenvalues for different choices of tangent space schedules: when only tangent space of the first eigenvector is used, strategy with the optimal choice of schedule, “argmax” strategy (14) and a random choice of tangent spaces. The convergence behaviour is similar to the convergence behaviour of vibrational spectra computation (see Sec. 6.1). The only difference we observe is that all eigenvectors can be well approximated in the tangent space of the first eigenvector. This allows to run most of the iterations in one tangent space. After that only a few iterations of the “argmax” strategy are needed to increase accuracy, which leads to a significant complexity reduction. Note that by contrast to the computation of vibrational spectra no preconditioner is used for (32).
Figure 4: Heisenberg model, $b = 30$, $r = 25$, $d = 40$. Convergence plots of 30 eigenvalues for four different tangent space schedule scenarios: first tangent space is chosen for all the iterations, optimal choice of schedule (not practical), proposed scheduling by (14) and a random choice of tangent space.

In Figure 3b time of one iteration on GPU is provided for $r = 25$ and $d = 40$. By contrast to the computation of molecule vibrational energy levels (Fig. 3a), for $b = 84$ the time spent for tensor operations is less than the time to find coefficients. This is due to the fact that TT-ranks of molecule Hamiltonians (Tab. 1) are larger than those of the considered spin lattices, where TT-rank is bounded by 5.

We compare the proposed method with two open source packages. The first one is eigb method [23] available in the ttpy\textsuperscript{9} library. The second one is DMRG-based algorithm implemented in ALPS [24] (algorithms and libraries for physical simulations), which also allows to compute more than one of eigenstates. We present results of the comparison in Table 3. We observe that for small $b$ ($b = 5$) we are not able to be faster than both eigb and ALPS at comparable accuracies. For larger $b$ ($b = 35$) the proposed method is faster on CPU, and GPU provides additional acceleration up to 15 times. We note that with our method we are capable of calculating larger $b$ (up to 100) with no problems, while eigb struggles in this range. The point is that due to the usage of the block TT format, the rank in eigb rapidly grows with $b$. Therefore, much larger rank values (more than 1000 for $b > 40$ [23]) are required for eigb.

7. Related work

The computation of energy levels of multidimensional Hamiltonians using low-rank tensor approximations has been considered in several communities. In solid state

---

\textsuperscript{9}https://github.com/oseledets/ttpy
Table 3: Mean absolute error (MAE) (31) and computation times of different methods for open spin chains. Reference energies $\epsilon_{\text{ref}}^{(i)}$ are calculated using $\text{eigb}$ [23] with $\delta = 10^{-5}$, where $\delta$ denotes the truncation error for both $\text{eigb}$ and ALPS.

|          | LRRAP (proposed) | eigb [23] | ALPS [24]  |
|----------|------------------|-----------|------------|
|          | $r=20$           | $r=35$    | $r=45$     | $\delta=10^{-2}$ | $\delta=10^{-3}$ | $\delta=10^{-4}$ | $\delta=10^{-5}$ |
| MAE      | 1.0e-4           | 1.2e-5    | 2.2e-6     | 2.2e-4         | 2.4e-6         | 1.0e-2           | 1.2e-3           |
| GPU time | 9 s              | 25 s      | 44 s       | 6 s            | 26 s           | 35 s             | 44 s             |
| CPU time | 50 s             | 145 s     | 251 s      | 18 min         | 31 min         | 24 min           | 41 min           |
| $d = 40$, $b = 35$ | $r=20$           | $r=35$    | $r=45$     | $\delta=10^{-2}$ | $\delta=10^{-3}$ | $\delta=10^{-4}$ | $\delta=10^{-5}$ |
| MAE      | 7.7e-3           | 1.3e-4    | 5.1e-6     | 1.3e-4         | 3.0e-6         | 1.4e-2           | 1.6e-3           |
| GPU time | 28 s             | 51 s      | 85 s       | 18 min         | 31 min         | 24 min           | 41 min           |
| CPU time | 5.6 min          | 12 min    | 21 min     | 18 min         | 31 min         | 24 min           | 41 min           |

physics computation of energy levels of one-dimensional spin lattices using tensor decompositions has been known for a long time. For this kind of applications the matrix product state (MPS) representation [25, 26] is used to approximate eigenfunctions. MPS is known to be equivalent to the Tensor-Train format [4] used in the current paper. The classical algorithm to approximate ground state of a one-dimensional spin lattice using MPS representation is the density matrix renormalization group (DMRG) [5, 6], see review [7] for details. Although MPS and DMRG are widely used in solid state physics, they were unknown in numerical analysis.

Calculation of several eigenvalues using two-site DMRG was first considered in papers by S. White [5, 27]. Alternatively, one can use the numerical renormalization group (NRG) [28] and its improved version variational NRG [29]. In these methods index, corresponding to the number of an eigenvalue is present only in the last site (core) of the MPS representation. The $\text{eigb}$ algorithm [23], which is an extension of [30], alternately assigns eigenvalue index to different sites (cores) of the representation, which allows for rank adaptation. By contrast, in the proposed method we consider every eigenvector separately, which implies that eigenvectors do not share sites (cores) of the decomposition. This leads to smaller rank values of eigenvectors. Moreover, thanks to the Riemannian optimization approach no rank growth occurs at iterations.

Calculation of energy levels using tensor decompositions was also considered for molecule vibrational spectra computations. CP-decomposition [31] (canonical decom-
position) of eigenvectors of vibrational problems was considered in [19], where rank reduced block power method (RRBPM) was used. Its hierarchical version (H-RRBPM) was later proposed in [21]. The H-RRBPM was improved in [20] and later in [22] (HI-RRBPM, where “I” stands for “intertwined”). With HI-RRBPM vibrational energy levels of molecules with more than dozens of atoms such as uracil and naphthalene [22] were accurately calculated. We also note that tensor decompositions were used in quantum dynamics, and in particular in the multiconfiguration time dependent Hartree (MCTDH) method [32] as well as in its multilayer version [33] (ML-MCTDH), which is similar to the Hierarchical Tucker representation [34].

Finally, a number of methods was developed independently in the mathematical community. Tensor versions of power and inverse power methods (inverse iteration) were considered in [35, 36, 37]. For tensor versions of preconditioned eigensolvers such as preconditioned inverse iteration (PINVIT) and LOBPCG methods see [38, 39, 40, 41, 42]. The generalization of the Jacobi-Davidson method was considered in [43]. The solvers based on alternating optimization procedures such as ALS [44] or AMEn [45] are proposed in [46, 47].

8. Conclusion and future work

We propose an eigensolver for high-dimensional eigenvalue problems in the TT format (MPS). The ability of the solver to efficiently calculate up to 100 eigenstates is assured by the usage of Riemannian optimization, which allows to avoid the rank growth naturally. The solver is implemented in TensorFlow, thus allowing both CPU and GPU parallelization. In the considered numerical examples the GPU version of the solver produces 15-20 times acceleration compared with the CPU version.

At each iteration of the solver, there arises a small, but nonstandard optimization problem to find coefficients of the iterative process. The method proposed to address this problem allows to solve it with the complexity comparable to the complexity of tensor operations for up to $b = 100$ eigenvalues. For $b > 100$ there is still room for improvement, which we plan to address in the future work.

Acknowledgements

This work was supported by a grant of the Russian Science Foundation (Project No. 17-12-01587).
References

[1] I. V. Oseledets, D. V. Savostianov, and E. E. Tyrtyshnikov. Tucker dimensionality reduction of three-dimensional arrays in linear time. *SIAM J. Matrix Anal. Appl.*, 30(3):939–956, 2008.

[2] I. V. Oseledets and E. E. Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra Appl.*, 432(1):70–88, 2010.

[3] Jonas Ballani, Lars Grasedyck, and Melanie Kluge. Black box approximation of tensors in hierarchical Tucker format. *Linear Algebra Appl.*, 428:639–657, 2013.

[4] I. V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011.

[5] S. R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69(19):2863–2866, 1992.

[6] S. Östlund and S. Rommer. Thermodynamic limit of density matrix renormalization. *Phys. Rev. Lett.*, 75(19):3537–3540, 1995.

[7] U. Schollöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, 2011.

[8] S. Holtz, T. Rohwedder, and R. Schneider. On manifolds of tensors of fixed TT-rank. *Numer. Math.*, 120(4):701–731, 2012.

[9] Peter Arbenz, Daniel Kressner, and DME Zürich. Lecture notes on solving large scale eigenvalue problems. *D-MATH, EHT Zurich*, 2, 2012.

[10] J. W. Robbin and D. A. Salamon. Introduction to differential geometry. *Lecture Notes, ETH*, 2011.

[11] P. A. Absil and I. V. Oseledets. Low-rank retractions: a survey and new results. *Comput. Optim. Appl.*, 2014.

[12] Daniel Kressner, Michael Steinlechner, and Bart Vandereycken. Preconditioned low-rank Riemannian optimization for linear systems with tensor product structure. *SIAM J. Sci. Comput.*, 38(4):A2018–A2044, 2016.

[13] Bart Vandereycken. Low-rank matrix completion by Riemannian optimization. *SIAM J. Optim.*, 23(2):1214–1236, 2013.

[14] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2008.

[15] Gustavo Avila and Tucker Carrington Jr. Using nonproduct quadrature grids to solve the vibrational Schrödinger equation in 12D. *J. Chem. Phys.*, 134(5):054126, 2011.
[16] D. Baye and P.-H. Heenen. Generalised meshes for quantum mechanical problems. 
*J. Phys. A: Math. Gen.*, 19(11):2041–2059, 1986.

[17] Maxim Rakhuba and Ivan Oseledets. Calculating vibrational spectra of molecules using tensor train decomposition. *J. Chem. Phys.*, 145:124101, 2016.

[18] B. N. Khoromskij. Tensor-structured preconditioners and approximate inverse of elliptic operators in $\mathbb{R}^d$. *Constr. Approx.*, 30:599–620, 2009.

[19] Arnaud Leclerc and Tucker Carrington. Calculating vibrational spectra with sum of product basis functions without storing full-dimensional vectors or matrices. *J. Chem. Phys.*, 140(17):174111, 2014.

[20] Phillip S Thomas and Tucker Carrington Jr. An intertwined method for making low-rank, sum-of-product basis functions that makes it possible to compute vibrational spectra of molecules with more than 10 atoms. *J. Chem. Phys.*, 146(20):204110, 2017.

[21] Phillip S Thomas and Tucker Carrington Jr. Using nested contractions and a hierarchical tensor format to compute vibrational spectra of molecules with seven atoms. *J. Phys. Chem. A.*, page 1307413091, 2015.

[22] Phillip S Thomas, Tucker Carrington Jr, Jay Agarwal, and Henry F Schaefer III. Using an iterative eigensolver and intertwined rank reduction to compute vibrational spectra of molecules with more than a dozen atoms: Uracil and naphthalene. *The Journal of chemical physics*, 149(6):064108, 2018.

[23] S. V. Dolgov, B. N. Khoromskij, I. V. Oseledets, and D. V. Savostyanov. Computation of extreme eigenvalues in higher dimensions using block tensor train format. *Computer Phys. Comm.*, 185(4):1207–1216, 2014.

[24] Bela Bauer, LD Carr, Hans Gerd Evertz, Adrian Feiguin, J Freire, S Fuchs, Lukas Gamper, Jan Gukelberger, E Gull, Siegfried Guertler, et al. The ALPS project release 2.0: open source software for strongly correlated systems. *J. Stat. Mech. Theory Exp.*, 2011(05):P05001, 2011.

[25] M. Fannes, B. Nachtergaele, and R.F. Werner. Finitely correlated states on quantum spin chains. *Comm. Math. Phys.*, 144(3):443–490, 1992.

[26] A. Klümper, A. Schadschneider, and J. Zittartz. Matrix product ground states for one-dimensional spin-1 quantum antiferromagnets. *Europhys. Lett.*, 24(4):293–297, 1993.

[27] S. R. White. Density-matrix algorithms for quantum renormalization groups. *Phys. Rev. B*, 48(14):10345–10356, 1993.

[28] K. G. Wilson. The renormalization group: Critical phenomena and the Kondo problem. *Rev. Mod. Phys.*, 47(4):773–840, 1975.
[29] Iztok Pižorn and Frank Verstraete. Variational Numerical Renormalization Group: Bridging the gap between NRG and Density Matrix Renormalization Group. Phys. Rev. Lett., 108(067202), 2012.

[30] B. N. Khoromskij and I. V. Oseledets. DMRG+QTT approach to computation of the ground state for the molecular Schrödinger operator. Preprint 69, MPI MIS, Leipzig, 2010.

[31] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. SIAM Rev., 51(3):455–500, 2009.

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

[33] Haobin Wang and Michael Thoss. Multilayer formulation of the multiconfiguration time-dependent hartree theory. J. Chem. Phys., 119(3):1289–1299, 2003.

[34] L. Grasedyck. Hierarchical singular value decomposition of tensors. SIAM J. Matrix Anal. Appl., 31(4):2024–2054, 2010.

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

[36] Gregory Beylkin, Jochen Garcke, and Martin Mohlenkamp. Multivariate regression and machine learning with sums of separable functions. SIAM J. Sci. Comput., 31(3):1840–1857, 2009.

[37] W. Hackbusch, B. N. Khoromskij, S. A. Sauter, and E. E. Tyrtyshnikov. Use of tensor formats in elliptic eigenvalue problems. Numer. Linear Algebra Appl., 19(1):133–151, 2012.

[38] T. Mach. Computing inner eigenvalues of matrices in tensor train matrix format. In Numerical Mathematics and Advanced Applications 2011, pages 781–788. Springer Berlin Heidelberg, 2013.

[39] M. V. Rakhuba and I. V. Oseledets. Grid-based electronic structure calculations: the tensor decomposition approach. J. Comp. Phys., 2016.

[40] M. V. Rakhuba and I. V. Oseledets. Fast multidimensional convolution in low-rank tensor formats via cross approximation. SIAM J. Sci. Comput., 37(2):A565–A582, 2015.

[41] O. S. Lebedeva. Tensor conjugate-gradient-type method for Rayleigh quotient minimization in block QTT-format. Russ. J. Numer. Anal. Math. Modelling, 26(5):465–489, 2011.

[42] D. Kressner and C. Tobler. Preconditioned low-rank methods for high-dimensional elliptic PDE eigenvalue problems. Computational Methods in Applied Mathematics, 11(3):363–381, 2011.
Appendix A. TensorFlow implementation overview

In this section, we provide brief introduction into TensorFlow (which we use to simplify GPU support) and details on how we implement tensor operations.

We implemented the proposed method in Python relying on two libraries: TensorFlow and T3F. TensorFlow [48] is a library written by Google to use for Machine Learning research (i.e., fast prototyping) and production alike. The focus is the ease of prototyping, GPU support, good parallelization abilities, and automatic differentiation\(^\text{10}\).

\(^{10}\)Given a computer program which defines a differentiable function, automatic differentiation generates another program which can compute the (exact) gradient of the function in the time at most 4 times slower than executing the original function. (In this paper we don’t use automatic differentiation.)
TensorFlow provides a library of linear-algebra (and some other) functions which abstract away the hardware details. For example, running the matrix multiplication function `tensorflow.matmul(A, B)` would multiply the two matrices on a CPU using all the available threads. Running the same code on a computer with a GPU will execute the matrix multiplication on the GPU. When executing a sequence of TensorFlow operations, TensorFlow makes sure that the data is not copied to and from GPU memory in between the operations. For example, running

```python
A = numpy.random.normal(100, 100)
matmul = tensorflow.matmul(A, A)
print(tensorflow.reduce_sum(matmul))
```

will generate a random matrix using Numpy on CPU, then copy it to the GPU memory, perform matrix multiplication on GPU, then compute the sum of all elements in the resulting matrix on GPU, and only then copy the result back to the CPU memory for printing. Moreover, TensorFlow allows to easily process pieces of data on multiple GPUs and then combine the result together on a single master GPU.

Another important TensorFlow feature is vectorization. Almost all functions support working with `batches` of objects, i.e. executing the same operations on a set of different arrays. For example, applying `tensorflow.matmul(A, B)` to two arrays of shapes $100 \times 3 \times 4$ and $100 \times 4 \times 5$ will return an array $C$ of shape $100 \times 3 \times 5$ such that $C^{(i)} = A^{(i)} B^{(i)}$, $i = 1, \ldots, 100$, and all the matrices are multiplied in parallel. This is especially important on GPUs: because of massive parallel resources of GPUs, running 100 small matrix multiplications sequentially (in a for loop) is almost 100 times slower than multiplying them all in parallel in a single vectorized operation.

The second library used in this paper is Tensor Train on Tensor Flow (T3F) [49], which is written on top of TensorFlow and provides many primitives for working with Tensor Train decomposition. To speed up the computations in the proposed method, we represent the current approximation to the eigenvectors $X = \{x^{(1)}, \ldots, x^{(b)}\}$ as a batch of $b$ TT-vectors, letting T3F and TensorFlow vectorize all the operations w.r.t. the number of TT-vectors. We also treat the basis $V$ on each iteration as a batch of TT-vectors. T3F library supports batch processing and for example executing `t3f.bilinear_form(A, X, X)` finds the value of $(x^{(i)})^T A x^{(i)}$ for each $i = 1, \ldots, b$ in parallel on a GPU. When dealing with large problems, we also use the multigpu feature of TensorFlow to use all the available GPUs on a single computer.

Let us consider a detailed example of adding two TT-tensors together. Given two tensors $A, B \in \mathbb{R}^{n \times \ldots \times n}$ represented in the TT-format

\[ A_{i_1, \ldots, i_d} = G^A_1(i_1) G^A_2(i_2) \ldots G^A_d(i_d) \]
\[ B_{i_1, \ldots, i_d} = G^B_1(i_1) G^B_2(i_2) \ldots G^B_d(i_d) \]

where for any $i_k = 1, \ldots, n$ the matrix $G^A_k(i_k)$ is of size $r \times r$ for $k = 2, \ldots, d-1$. $G^A_1(i_1)$ is of size $1 \times r$, and $G^A_d(i_d)$ is of size $r \times 1$. The goal is to find TT-cores $\{G^C_k(i_k)\}_{k=1}^d$ of
tensor $\mathcal{C} = \mathcal{A} + \mathcal{B}$. The expression for these TT-cores is given by [4]:

$$ G^C_k(i_k) = \begin{bmatrix} G^A_k(i_k) & 0 \\ 0 & G^B_k(i_k) \end{bmatrix}, \quad k = 2, \ldots, d - 1 $$

Note that we also want to support batch processing, e.g., adding 100 TT-tensors $\mathcal{C}^{(i)}_i = \mathcal{A}^{(i)}_i + \mathcal{B}^{(i)}_i$, $i = 1, \ldots, 100$. The resulting program is similar to any implementation of summation of TT-tensors with two exceptions: support of batch processing and using TensorFlow for all elementary operations to allow GPU support.

T3F represents a batch of $d$-dimensional TT-tensors as a list of $d$ arrays, $k$-th array representing $k$-th TT-core of all the tensors in the batch. The shape of the $k$-th array ($k = 2, \ldots, d - 1$) is $b \times r \times n \times r$, where $b$ is the batch-size, the shape of the first array ($k = 1$) is $b \times 1 \times n \times r$, and the shape of the last array ($k = d$) is $b \times r \times n \times 1$.

Algorithm 3 Implementation of adding two batches of TT-tensors in T3F ($t3f.add(A, B)$).

| Require: | Arrays representing TT-cores of batches of tensors $\{\mathcal{A}^{(i)}_i\}_{i=1}^b$ and $\{\mathcal{B}^{(i)}_i\}_{i=1}^b$ |
| Ensure: | Array representing TT-cores of batch $\{\mathcal{C}^{(i)}_i\}_{i=1}^b = \{\mathcal{A}^{(i)}_i + \mathcal{B}^{(i)}_i\}_{i=1}^b$ |

1. Concatenate $G^A_1$ and $G^B_1$ (of shape $b \times 1 \times n \times r$) along the 4-th axis to form array $G^C_1$ of shape $b \times 1 \times n \times 2r$
2. for $k = 2, \ldots, d - 1$ do
3. create an array of zeros of size $b \times r \times n \times r$
4. Concatenate $G^A_k$ with array of zeros along 4-th axis into array $U$ of shape $b \times r \times n \times 2r$
5. Concatenate array of zeros with $G^B_k$ along 4-th axis into array $D$ of shape $b \times r \times n \times 2r$
6. Concatenate $U$ and $D$ along 2-nd axis into $G^C_k$ of shape $b \times 2r \times n \times 2r$.
7. Concatenate $G^A_d$ and $G^B_d$ (of shape $b \times r \times n \times 1$) along the 2-nd axis to form array $G^C_d$ of shape $b \times 2r \times n \times 1$

To add two TT-tensors, T3F calls TensorFlow functions to create arrays of appropriate sizes filled with zeros and to concatenate the TT-cores of tensors $\mathcal{A}$ and $\mathcal{B}$ with each other and with zeros (see Alg. 3). Note that a user of T3F can ignore this implementation details and just call $t3f.add(A, B)$. 

29