Vector-Tensor-Approaches for calculation of Eigenvalues

April 21, 2010

Author:	Stephan M. Günther, B.Sc.
Supervisor:	Prof. Dr. Thomas Huckle
Advisor:	Dipl. Math. Konrad Waldherr

Department for Computer Science
Technische Universität München

CONTENTS

I Notation 3

II Introduction 4
II-A Quantum gates . . . 4
II-B Physical implementation of quantum gates . . . 5

III Representation of operands 6
III-A Hamiltonian . . . 6
III-B Vectors . . . 7

IV Derivation of the VTA algorithm 8
IV-A Start approximation . . . 8
IV-B Improving precision . . . 9
IV-C Outline of the VTA algorithm . . . 11

V Custom BLAS routines 12
V-A Inner product between Kronecker vectors with matching slices . . . 12
V-B Inner product between Kronecker vectors with matching slices and excludes . . . 12
V-C Inner product between Kronecker vectors with unequal slices . . . 12
V-D Inner product between Kronecker vectors with unequal slices and excludes . . . 13
V-E Product between a Hamiltonian and a normal vector . . . 14
V-F Product between a Hamiltonian and a Kronecker vector . . . 15

VI Numerical results 15
VI-A Non-overlapping slice sequences . . . 15
VI-B Overlapping slice sequences . . . 17
VI-C Optimized sequence of overlapping slices . . . 17
VI-D Large problem . . . 19

VII Performance results 19

VIII Conclusion 20

Appendix A: Implementational overview 21

Appendix B: Installation and usage instruction for the VTA test application 21

References 22

I. NOTATION

We denote matrices by bold capital letters, i.e. $\boldsymbol{I}_2$ stands for the unit matrix of dimension two. Vectors are normally written as bold lowercase letters. Scalars are in general written as lowercase greek letters. The conjugate of a complex number is denoted by over-lining. The hermitian of a matrix or vector is indicated by a superior H, i.e. $\boldsymbol{x}^H = \overline{\boldsymbol{x}}^T$. Numeric optimal values are indicated by a superior asterisk. The eigenvector resulting from the minimization of the Rayleigh quotient therefore reads as

$$\boldsymbol{y}^* = \arg\min_{\boldsymbol{y}\neq\boldsymbol{0}} \frac{\boldsymbol{y}^H \boldsymbol{H} \boldsymbol{y}}{\boldsymbol{y}^H \boldsymbol{y}}.$$

Deviant from the previous convention we denote the *Pauli matrices*

$$\mathcal{Q} := \{\sigma_x, \sigma_y, \sigma_z\} = \left\{ \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \right\} \tag{1}$$

by lowercase greek letters which is a common convention. Furthermore, we use the style of physicists for vectors in Section II. A column vector (ket vector) is then written as $|\psi\rangle$ and a row vector (bra vector) as $\langle\psi|$.

The *Kronecker product* between matrices $\boldsymbol{A} \in \mathbb{C}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{C}^{q\times r}$ is defined as

$$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix} \otimes \boldsymbol{B} = \begin{bmatrix} a_{1,1}\cdot\boldsymbol{B} & \dots & a_{1,n}\cdot\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m,1}\cdot\boldsymbol{B} & \dots & a_{m,n}\cdot\boldsymbol{B} \end{bmatrix}. \tag{2}$$

The resulting matrix $\boldsymbol{C}$ is of dimension $(m\cdot q)\times(n\cdot r)$. For vectors $\boldsymbol{a} \in \mathbb{C}^n$ and $\boldsymbol{b} \in \mathbb{C}^m$ it is analogous defined as

$$\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \otimes \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} a_1\cdot\boldsymbol{b} \\ \vdots \\ a_n\cdot\boldsymbol{b} \end{bmatrix} \tag{3}$$

where the resulting vector $\boldsymbol{c}$ is of dimension $n\cdot m$. For scalar values the Kronecker product reduces to ordinary multiplication. In the following we list some important properties of the Kronecker product which we will use in various equations:

1) For matrices $\boldsymbol{A} \in \mathbb{C}^{m\times n}$, $\boldsymbol{B} \in \mathbb{C}^{n\times k}$, $\boldsymbol{C} \in \mathbb{C}^{r\times s}$ and $\boldsymbol{D} \in \mathbb{C}^{s\times t}$ holds:

$$\boldsymbol{AB} \otimes \boldsymbol{CD} = (\boldsymbol{A}\otimes\boldsymbol{C})(\boldsymbol{B}\otimes\boldsymbol{D}) \tag{4}$$

2) For arbitrary matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ holds:

$$(\boldsymbol{A}\otimes\boldsymbol{B})^H = \boldsymbol{A}^H \otimes \boldsymbol{B}^H \tag{5}$$

3) For arbitrary matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and a scalar $\lambda \in \mathbb{C}$ holds:

$$\boldsymbol{A}\otimes(\boldsymbol{B}+\boldsymbol{C}) = \boldsymbol{A}\otimes\boldsymbol{B} + \boldsymbol{A}\otimes\boldsymbol{C} \tag{6}$$

$$\lambda(\boldsymbol{A}\otimes\boldsymbol{B}) = (\lambda\boldsymbol{A})\otimes\boldsymbol{B} = \boldsymbol{A}\otimes(\lambda\boldsymbol{B}) \tag{7}$$

4) The Kronecker product is associative. For arbitrary matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ holds:

$$(\boldsymbol{A}\otimes\boldsymbol{B})\otimes\boldsymbol{C} = \boldsymbol{A}\otimes(\boldsymbol{B}\otimes\boldsymbol{C}) \tag{8}$$

5) Commutativity does **not** hold.

The rest of this report is organized as follows: A brief introduction to quantum computing is given in Section II. Here, we motivate the origin of the system matrices called *Hamiltonians*. Section III introduces the representation of the Hamiltonian and the approximation of its eigenvectors. Afterwards we derive the VTA algorithm in Section IV. The arising BLAS operations are described in Section V. Numerical results of our approach are given in Section VI. The runtime performance is briefly summarized in Section VII. Finally we conclude in Section VIII and describe areas of future work. Appendix A contains a class diagram of the implementation while Appendix B describes the command line interface of the VTA test application.

II. Introduction

Conventional computers operate on Bits representing some kind of information. For processing, Bits are stored in *registers* which are arrays of 1 Bit memory cells. The physical representation of a Bit is a voltage level generated by a charged capacitor, i.e. high level for a logical one and low level for a logical zero. The bitwise logical NOT operation can be implemented by feeding the voltage of a memory cell to an inverter and saving the result back to the memory cell. The idea behind quantum computing is to use properties of particles to represent Bits. If a particle can reside in two different possible states at the time of observation, the quantum equivalent of a Bit is called *qubit*. For example, the spins $\pm 1/2$ of a nucleus may be interpreted as logical one and zero. In the same way, the spin of electrons or the polarization of photons may be used. The equivalent to logic gates used by conventional computers for data processing are *quantum gates* which modify the properties of individual particles, i.e. the spin of nuclei.

This Section gives a rough overview of the principles in quantum computing. It is destined to give an idea where the specific type of system matrices called *Hamiltonians* originate from, which make up the eigenvalue problems we seek to solve. At first we consider quantum gates operating on one and two qubits. Afterwards we turn to physical realizations which leads us to the Hamiltonians. General references for this Section are [5] and [4].

A. Quantum gates

Sticking to the example of nuclei, the two spin states $+1/2$ and $-1/2$ can be associated with the unit vectors

$$|0\rangle := \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{and} \quad |1\rangle := \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

living in the Hilbert space $\mathcal{H} = \mathbb{C}^2$. The components of these vectors specify the probabilities that a system remains in either of two states. The basis vectors given above deterministically specify the spin. While not being observed, a nucleus might also remain in a *superposition* state

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{with} \quad \alpha, \beta \in \mathbb{C} \quad \text{and} \quad |a|^2 + |b|^2 = 1.$$

The principles of quantum mechanics demand that a measurement on $|\psi\rangle$ turns out to be either $|0\rangle$ with probability $|\alpha|^2$ or $|1\rangle$ with probability $1 - |\alpha|^2 = |\beta|^2$. Consequently, the system is in one of two well-defined states immediately after a measurement.

A 2-qubit quantum gate is made up of two components each living in its own Hilbert space $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively. The complete system lives in the Hilbert space $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$. Let $\{|e_{ij}\rangle\}_{j=1}^2$ denote an orthonormal basis of the Hilbert space $\mathcal{H}_i$. The corresponding state vector is then given by

$$|\psi\rangle = \sum_{i,j \in \{1,2\}} \gamma_{ij} \cdot (|e_{1i}\rangle \otimes |e_{2j}\rangle) \quad \text{with} \quad \sum_{i,j \in \{1,2\}} |\gamma_{ij}|^2 = 1 \quad \text{and} \quad c_{ij} \in \mathbb{C}.$$

Note, that for the dimension of the resulting Hilbert space $\mathcal{H}$ holds $\dim \mathcal{H} = \dim \mathcal{H}_1 \dim \mathcal{H}_2$. A state which can be written as $|\psi\rangle = |\psi\rangle_1 \otimes |\psi\rangle_2$ is called *separable* since it can be described as a Kronecker product of two states living in their own Hilbert spaces which make up $\mathcal{H}$. This is not possible for all vectors as can easily be seen by considering the dimensions of the Hilbert spaces involved. Given $\mathcal{H}_1$ and $\mathcal{H}_2$ there are only $\dim \mathcal{H}_1 + \dim \mathcal{H}_2$ separable states. However, The dimension of the resulting Hilbert space is $\dim \mathcal{H}_1 \dim \mathcal{H}_2 \gg \dim \mathcal{H}_1 + \dim \mathcal{H}_2$ for higher dimensions. From that we can conclude that most states are not separable. A concrete example is given in Section III-B. Less mathematically speaking a state vector $|\psi\rangle \in \mathcal{H}$ is separable if it can be fully described by the states of its components $|\psi\rangle_1 \in \mathcal{H}_1$ and $|\psi\rangle_2 \in \mathcal{H}_2$. Otherwise it is called *entangled* and refuses a classical description.

Given a state $|\psi\rangle$ a quantum gate influences the components of the system in a way such that the resulting state $|\psi'\rangle$ delivers the intended result at observation. This transition can be described by the mapping

$$|\psi\rangle = \boldsymbol{U}|\psi'\rangle$$

where $\boldsymbol{U}$ is a unitary matrix. The following two examples describe basic quantum gates and derive their algebraic representation.

NOT gate:
The most basic non trivial operation on a single qubit is the NOT operation. It is defined as

$$\boldsymbol{U}_{\text{NOT}}: \ |0\rangle \to |1\rangle, \ |1\rangle \to |0\rangle$$

where the matrix $\boldsymbol{U}_{\text{NOT}}$ defines a unitary transformation. It can be obtained by

$$|0\rangle\langle 1| + |1\rangle\langle 0| = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} =: \boldsymbol{U}_{NOT}.$$

The effect of the quantum gate is verified by $|1\rangle = \boldsymbol{U}_{\text{NOT}}|0\rangle$ and $|0\rangle = \boldsymbol{U}_{\text{NOT}}|1\rangle$.

CNOT gate:
The most important 2-qubit quantum gate is the controlled NOT (CNOT) operation. It is of special relevance since it can be used to implement any of the classical logic gates. The CNOT gate flips the second qubit if the first qubit is $|1\rangle$ and does nothing otherwise. A state of the corresponding 2-qubit system is given by one of the unit vectors of the Hilbert space $\mathcal{H} = \mathbb{C}^4$ which are denoted by $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ respectively. The operation $\boldsymbol{U}_{CNOT}$ is then given by

$$\boldsymbol{U}_{CNOT}: |00\rangle \to |00\rangle, \ |01\rangle \to |01\rangle, \ |10\rangle \to |11\rangle, \ |11\rangle \to |10\rangle.$$

The matrix $\boldsymbol{U}_{CNOT}$ describes another unitary transformation and can be obtained by

$$|00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10| = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} := \boldsymbol{U}_{CNOT}.$$

B. Physical implementation of quantum gates

The algebraic descriptions of quantum gates through unitary transformations leaves the problem of physical implementation open. A quantum register may be physically represented by a molecule with multiple spins. The spins are fixed under a strong magnet field $\boldsymbol{B}_0$. Using additional time variant magnetic fields $\boldsymbol{B}_i(t)$ of different frequencies orthogonal to $\boldsymbol{B}_0$ the spins of individual nuclei can be targeted. The time evolution of such a system is governed by the Schrödinger equation

$$i\hbar \frac{\partial}{\partial t}|\psi(t)\rangle = \tilde{\boldsymbol{H}}|\psi(t)\rangle.$$

The Hamilton operator $\tilde{\boldsymbol{H}}$ describes both the influence of the magnetic fields and of the spins on each other. Assumed that $\tilde{\boldsymbol{H}}$ is time invariant (which is not true), the Schrödinger equation is solved to

$$|\psi(t)\rangle = \exp\left(-\frac{i}{\hbar}\tilde{\boldsymbol{H}}t\right)|\psi(t_0)\rangle = \boldsymbol{U}(t)|\psi(t_0)\rangle.$$

The matrix $\boldsymbol{U}(t)$ denotes a unitary transformation as introduced in Section II-A which is also called *time evolution operator*. Considering the time dependency of $\tilde{\boldsymbol{H}}$ makes things more difficult. Under certain assumptions an idealized model leads to a simplified Hamiltonian $\boldsymbol{H}$. In particular we assume that spins are linearly coupled and only direct neighbors influence each other. Furthermore, we can get rid of the time dependency of $\tilde{\boldsymbol{H}}$ by choosing a rotating frame of reference. The new Hamiltonian for a system with n spins is then given by

$$\boldsymbol{H} = \sum_{i=1}^{n-1} J_{i,i+1}\boldsymbol{I}_{z,i} \otimes \boldsymbol{I}_{z,i+1} + \sum_{i=1}^{n} \omega_i \left(\cos\phi_i \boldsymbol{I}_{x,i} + \sin\phi_i \boldsymbol{I}_{y,i}\right). \tag{9}$$

The coefficients $J_{i,i+1}$ represent the coupling strength between spins i and $i+1$. The symbol $\boldsymbol{I}_{k,i}$ denotes a sequence $\boldsymbol{I}_2 \otimes \ldots \otimes \boldsymbol{I}_k \otimes \ldots \boldsymbol{I}_2$ of Kronecker products with $\boldsymbol{I}_2$ as the two dimensional unit matrix and $\boldsymbol{I}_k$ as one of the three Pauli matrices σ_x, σ_y and σ_z scaled by some constant factor. The first sum in (9) describes the pairwise interaction between spins aligned along the z-axis according to the Ising model. The second sum models the influence of the orthogonal time variant magnetic fields where ω_i denotes the Lamour frequency and ϕ_i the phase shift of the i-th magnetic field.

III. Representation of operands

The dimension of the Hamiltonian and therefore the size of the problem grows exponentially with the number of qubits under consideration. Therefore, we deal with very large matrices which can be constructed following a well-defined scheme. Given the dimension 2^P for a system of P qubits, explicit usage even of vectors is infeasible in regard to both memory requirements and computational complexity. This Section introduces at first the most general representation of the Hamiltonian. Afterwards a similar structure for vectors is introduced.

A. Hamiltonian

The Hamiltonian $\boldsymbol{H} \in \mathbb{C}^{2^P}$ as introduced in Section II is a hermitian and traceless matrix. For $P = 1$ the set $\mathcal{Q} := \{\sigma_x, \sigma_y, \sigma_z\}$ of Pauli matrices makes up a corresponding orthonormal basis under the field $(\mathbb{R}, +, \cdot)$. Scalar coefficients must not be complex since matrices with complex entries on their main diagonal are not hermitian. Note, that the Pauli matrices are themselves hermitian and traceless (see (1) for the definition of Pauli matrices). For $P \geq 1$, any hermitian matrix $\hat{\boldsymbol{H}} \in \mathbb{C}^{2^P}$ can then be written as a weighted sum of Kronecker products

$$\hat{\boldsymbol{H}} = \sum_{m=1}^{M} \alpha_m \cdot \boldsymbol{Q}_1^{(m)} \otimes \boldsymbol{Q}_2^{(m)} \otimes \ldots \otimes \boldsymbol{Q}_P^{(m)}, \quad \boldsymbol{Q}_k^{(m)} \in \mathcal{Q} \cup \{\boldsymbol{I}_2\}, \quad \alpha_m \in \mathbb{R} \tag{10}$$

where $\boldsymbol{I}_2$ denotes the unit matrix of dimension 2. Introducing the constraint

$$\forall m = \{1, \ldots, m\} \; \exists k \in \{1, \ldots, P\} \mid \boldsymbol{Q}_k^{(m)} \neq \boldsymbol{I}_2 \tag{11}$$

on (10) ensures that $\hat{\boldsymbol{H}}$ is traceless since the Pauli matrices are traceless and for the trace under Kronecker products holds that

$$\operatorname{tr}\left(\hat{\boldsymbol{H}}^{(m)}\right) = \prod_{k=1}^{P} \operatorname{tr}\left(\boldsymbol{Q}_k^{(m)}\right). \tag{12}$$

Equation (10) with constraint (11) therefore gives a decomposition of hermitian traceless matrices into a weighted sum of Kronecker products between Pauli matrices which is also a suitable representation of the Hamiltonian $\boldsymbol{H}$. In the following we will implicitly demand that constraint (11) is fulfilled and denote the Hamiltonian by $\boldsymbol{H}$.

In general, the Hamiltonian is a sparse matrix. Each summand in (10) has a sparsity of $1/2^P$ since the tensor product between two Pauli matrices (or the identity matrix) gives $8 + 4$ zero and 4 non-zero entries. Assuming $M < 2^{P/2}$ the Hamiltonian has still $\mathcal{O}(2^P)$ non-zero entries and is therefore also sparse. The actual sparsity pattern depends on the particle system under consideration. For a system with P qubits as defined in (9) without the influence of time-variant magnet fields, the Hamiltonian takes a sparse band pattern as shown in Figure 1. For physically interesting problems we assume $P > 50$ which points out the unfeasibility of storing the Hamiltonian in an explicit form. Thanks to its special construction scheme storage is not a problem. Unfortunately, efficient storage makes it impossible to use existing BLAS routines. Instead, a new set of routines for matrix-vector products is developed and described in more detail in Section V.

(a) $P = 4$, sparsity ≈ 0.2266 (b) $P = 5$, sparsity ≈ 0.1563 (c) $P = 6$, sparsity ≈ 0.0889

Fig. 1: Hamiltonian for particle systems of different sizes.

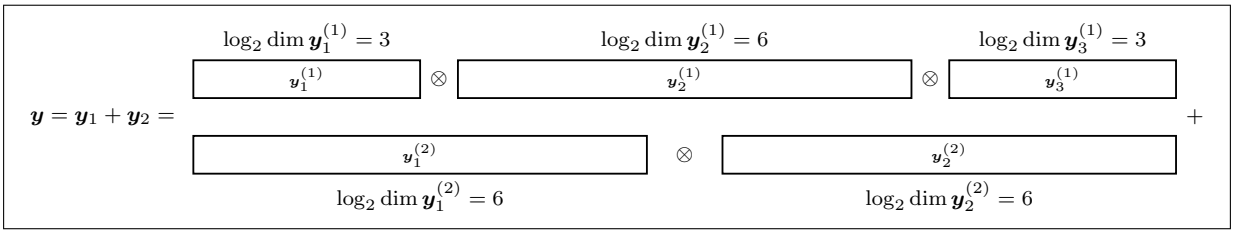


Fig. 2: Examplary slicing for $\boldsymbol{y} = \boldsymbol{y}_1^{(1)} \otimes \boldsymbol{y}_2^{(1)} \otimes \boldsymbol{y}_3^{(1)} + \boldsymbol{y}_1^{(2)} \otimes \boldsymbol{y}_2^{(2)}$.

B. Vectors

For the eigenvector $\boldsymbol{y}^*$ we can in general not exploit a special structure. Given $P = 50$ a complex vector $\boldsymbol{y}$ would require approximately 16 PiB of memory using double precision which is clearly infeasible. To circumvent this problem we decompose a vector $\boldsymbol{y}$ in a similar way as the matrix $\boldsymbol{H}$ and write it as a sum of Kronecker products of smaller vectors:

$$\boldsymbol{y} = \sum_{j=1}^{J} \boldsymbol{y}^{(j)} = \sum_{j=1}^{J} \boldsymbol{y}_1^{(j)} \otimes \boldsymbol{y}_2^{(j)} \otimes \ldots \otimes \boldsymbol{y}_{Q_j}^{(j)} \tag{13}$$

We refer to $\boldsymbol{y}^{(j)}$ as *Kronecker vector* and to $\boldsymbol{y}_i^{(j)}$ as *slice*. There are no restrictions on the dimension of a slice except that the resulting summand $\boldsymbol{y}^{(j)}$ must match in size with $\boldsymbol{y}$. Since the dimension of $\boldsymbol{H}$ is a power of two, $\dim \boldsymbol{y}^{(j)}$ and therefore also $\dim \boldsymbol{y}_i^{(j)}$ are powers of two. An example for $\boldsymbol{y} \in \mathbb{C}^{2^{12}}$ consisting of two summands is shown in Figure 2.
It is easy to verify that any vector of $\mathbb{C}^{2^P}$ can be represented using (13) with at most $J = 2^P$ summands by choosing $\boldsymbol{y}^{(j)}$ such that a single entry is non-zero and appropriately scaled. However, this is a rather simplistic upper bound for the number of summands. In most cases less summands are sufficient. Consider for example the vector $[1, 1, 0, 1]^T$ which should be represented by a sum of Kronecker products:

$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The preceding example gives an exact solution for $J = 2$. Note, that it is not possible to write $[1, 1, 0, 1]^T$ as a single Kronecker product of two smaller vectors. It is therefore not a separable vector but a superposition of states. Determining the number of summands necessary for an exact representation is in general difficult. However, we do not seek for an exact representation of the eigenvalue $\boldsymbol{y}^*$ but only for a meaningful close approximation. Using the representation as given in (13) allows to approximate the eigenvector within some error margin using a few summands. Afterwards, accuracy can be iteratively improved by adding more summands if needed.

Compared to an explicit representation the memory requirements remain affordable. For example, assume we seek to approximate $\boldsymbol{y}^* \in \mathbb{C}^{2^{50}}$ using J summands each represented by a sequence of five Kronecker products. Let each slice have 2^{10} entries making a total of $5 \cdot 2^{10}$ values per summand. This corresponds to 80 KiB of memory assuming complex double precision which allows for thousands of summands. Chances are, that a sufficient close approximation of $\boldsymbol{y}^*$ needs only $J \ll 2^P$. This unusual representation of the vectors requires custom BLAS routines especially for inner products as we will see in Section V.

IV. Derivation of the VTA algorithm

We consider the eigenvalue problem $\boldsymbol{H}\boldsymbol{y}^* = \lambda \boldsymbol{y}^*$ where $\boldsymbol{H}$ denotes a Hamiltonian of dimension $2^P \times 2^P$. We are interested in a numerical method to obtain the smallest eigenvalue $\lambda_{\min}$. Since the Hamiltonian is a hermitian matrix, the spectral theorem ensures the existence of a unitary matrix $\boldsymbol{U}$ such that $\boldsymbol{H} = \boldsymbol{U}^H \boldsymbol{\Lambda} \boldsymbol{U}$ where $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{2^P})$. We can now write the Rayleigh quotient $r(\boldsymbol{y})$ in respect of $\boldsymbol{H}$ as

$$r(\boldsymbol{y}) = \frac{\boldsymbol{y}^H \boldsymbol{H} \boldsymbol{y}}{\boldsymbol{y}^H \boldsymbol{y}} = \frac{\boldsymbol{y}^H \left(\boldsymbol{U}^H \boldsymbol{\Lambda} \boldsymbol{U}\right) \boldsymbol{y}}{\boldsymbol{y}^H \left(\boldsymbol{U}^H \boldsymbol{U}\right) \boldsymbol{y}}. \tag{14}$$

The substitution $\boldsymbol{\xi} := \boldsymbol{U}\boldsymbol{y}$ gives

$$r(\boldsymbol{y}) = \frac{\boldsymbol{\xi}^H \boldsymbol{\Lambda} \boldsymbol{\xi}}{\boldsymbol{\xi}^H \boldsymbol{\xi}} = \frac{\sum_{i=1}^{2^P} \lambda_i \xi_i^2}{\sum_{i=1}^{2^P} \xi_i^2} = \sum_{i=1}^{2^P} \frac{\xi_i^2}{\sum_{i=1}^{2^P} \xi_i^2} \lambda_i = \sum_{i=1}^{2^P} \tilde{\xi}_i \lambda_i. \tag{15}$$

The matrix $\boldsymbol{U}$ is unitary wherefore the non-negative $\tilde{\xi}_i$ must sum up to one. Minimization of (15) yields

$$\underset{\tilde{\xi}_i \geq 0}{\arg\min} \sum_{i=1}^{2^P} \tilde{\xi}_i \lambda_i = \begin{cases} 1, & \lambda_i = \lambda_{\min} \\ 0, & \lambda_i \neq \lambda_{\min} \end{cases}. \tag{16}$$

The Rayleigh quotient is only defined if $\boldsymbol{\xi} \neq 0$ wherefore at least one $\tilde{\xi}_i$ must be positive. Equation (16) therefore picks the smallest eigenvalue value $\lambda_{\min}$. Consequently, the smallest eigenvalue can be obtained by minimizing the Rayleigh quotient:

$$\lambda_{\min} = \min_{\boldsymbol{y} \neq \boldsymbol{0}} \frac{\boldsymbol{y}^H \boldsymbol{H} \boldsymbol{y}}{\boldsymbol{y}^H \boldsymbol{y}} \tag{17}$$

This problem could be solved directly using approved toolboxes. However, even by using the representation of $\boldsymbol{H}$ as described in Section III-A we had to deal with an eigenvalue problem of dimension 2^{50} for interesting instances which is computationally infeasible. For this reason we seek for a way to approximate the optimal solution $\boldsymbol{y}^*$ of problem (17) with much less computational effort.

The VTA algorithm iteratively improves a starting solution $\boldsymbol{y}[1]$ which consists of a single Kronecker vector by adding a new summand per step. Using a Kronecker vector which is comprised of smaller slices the original problem is transformed into a sequence of smaller problems. Since any complex vector can be approximated arbitrarily close by a sum of Kronecker vectors, the VTA algorithm converges against $\boldsymbol{y}^*$ after a sufficient number of iterations. Section IV-A concentrates on finding the starting solution $\boldsymbol{y}[1]$ consisting of only a single Kronecker vector. Afterwards, Section IV-B extends the approach by allowing a sum of Kronecker vectors. General reference for this Section is [6].

A. Start approximation

For now we consider the case $\boldsymbol{y}[1] = \boldsymbol{x}$. In particular, $\boldsymbol{y}[1]$ is a single Kronecker vector of arbitrary slicing. Assuming Q slices $\boldsymbol{x}_1, \ldots \boldsymbol{x}_Q$ the resulting vector is of dimension $\prod_{i=1}^{Q} \dim \boldsymbol{x_i}$ and can be written as

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{x}_1 \otimes \boldsymbol{x}_2 \otimes \ldots \otimes \boldsymbol{x}_Q \\ &= \boldsymbol{x}_L \otimes \boldsymbol{x}_i \otimes \boldsymbol{x}_R. \end{aligned} \tag{18}$$

The vectors $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$ are shorthand representations of the Kronecker products on the left and right side of slice $\boldsymbol{x}_i$. The Hamiltonian $\boldsymbol{H} \in \mathbb{C}^{2^P \times 2^P}$ can be written in a similar form as

$$\begin{aligned} \boldsymbol{H} &= \sum_{m=1}^{M} \alpha_m \cdot \boldsymbol{Q}_1^{(m)} \otimes \boldsymbol{Q}_2^{(m)} \otimes \ldots \otimes \boldsymbol{Q}_P^{(m)} \\ &= \sum_{m=1}^{M} \alpha_m \cdot \boldsymbol{Q}_L^{(m)} \otimes \boldsymbol{Q}_i^{(m)} \otimes \boldsymbol{Q}_R^{(m)}. \end{aligned} \tag{19}$$

This allows us to work with slice $\boldsymbol{x}_i$ while considering other slices as constant vectors. Inserting (18) and (19) into the original optimization problem (17) yields:

$$\min_{\boldsymbol{x}\neq\boldsymbol{0}}\frac{\boldsymbol{x}^H\boldsymbol{H}\boldsymbol{x}}{\boldsymbol{x}^H\boldsymbol{x}} \doteq \min_{\boldsymbol{x}_i\neq\boldsymbol{0}}\frac{(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R)^H\left(\sum_{m=1}^{M}\alpha_m\cdot\boldsymbol{Q}_L^{(m)}\otimes\boldsymbol{Q}_i^{(m)}\otimes\boldsymbol{Q}_R^{(m)}\right)(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R)}{(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R)^H(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R)} \tag{20}$$

$$\begin{aligned}
&= \min_{\boldsymbol{x}_i\neq\boldsymbol{0}}\frac{\sum_{m=1}^{M}\alpha_m\cdot\left(\boldsymbol{x}_L^H\boldsymbol{Q}_L^{(m)}\boldsymbol{x}_L\right)\left(\boldsymbol{x}_i^H\boldsymbol{Q}_i^{(m)}\boldsymbol{x}_i\right)\left(\boldsymbol{x}_R^H\boldsymbol{Q}_R^{(m)}\boldsymbol{x}_R\right)}{(\boldsymbol{x}_L^H\boldsymbol{x}_L)(\boldsymbol{x}_i^H\boldsymbol{x}_i)(\boldsymbol{x}_R^H\boldsymbol{x}_R)}\\
&= \min_{\boldsymbol{x}_i\neq\boldsymbol{0}}\frac{\boldsymbol{x}_i^H\left(\sum_{m=1}^{M}\alpha_m\frac{\left(\boldsymbol{x}_L^H\boldsymbol{Q}_L^{(m)}\boldsymbol{x}_L\right)\left(\boldsymbol{x}_R^H\boldsymbol{Q}_R^{(m)}\boldsymbol{x}_R\right)}{(\boldsymbol{x}_L^H\boldsymbol{x}_L)(\boldsymbol{x}_R^H\boldsymbol{x}_R)}\boldsymbol{Q}_i^{(m)}\right)\boldsymbol{x}_i}{\boldsymbol{x}_i^H\boldsymbol{x}_i}\\
&= \min_{\boldsymbol{x}_i\neq\boldsymbol{0}}\frac{\boldsymbol{x}_i^H\left(\sum_{m=0}^{M}\alpha_m\frac{L_mR_m}{\gamma_L\gamma_R}\boldsymbol{Q}_i^{(m)}\right)\boldsymbol{x}_i}{\boldsymbol{x}_i^H\boldsymbol{x}_i}\\
&= \min_{\boldsymbol{x}_i\neq\boldsymbol{0}}\frac{\boldsymbol{x}_i^H\boldsymbol{H}_i\boldsymbol{x}_i}{\boldsymbol{x}_i^H\boldsymbol{x}_i}
\end{aligned} \tag{21}$$

We refer to $\boldsymbol{H}_i$ in (21) as the *effective Hamiltonian* which depends on the slice $\boldsymbol{x}_i$ being optimized, the scalar values γ_L, γ_R and the products $\boldsymbol{x}_L^H\boldsymbol{Q}_L^{(m)}\boldsymbol{x}_L$ and $\boldsymbol{x}_R^H\boldsymbol{Q}_R^{(m)}\boldsymbol{x}_R$. The latter two additionally depend on the sum index m. Equation 21 is another eigenvalue problem which is reduced to dimension $\dim\boldsymbol{x}_i$. For example, we reduce a single eigenvalue problem of dimension 2^{50} to a sequence of five eigenvalue problems of dimension 2^{10}. Since only a single slice is optimized in each iteration, we refer to (21) as *local optimization*. Due to the reduced size, this problem can be solved efficiently using existing solvers (Krylov based methods). In general, the exact eigenvalue $\boldsymbol{y}^*$ cannot be obtained by (21) since $\boldsymbol{x}$ as defined in (18) does not span the vector field $\mathbb{C}^{2^P}$. The optimal value $y[1]:=\boldsymbol{x}^*$ is at best a first approximation of $\boldsymbol{y}^*$. To obtain a tighter approximation we now extend the space of solutions by stepwise adding new summands.

B. Improving precision

To allow for a tighter approximation of the real eigenvalue $\boldsymbol{y}^*$ we now consider further approximations $\boldsymbol{y}[J]$ for $J>0$. These consist of $J-1$ preceding solutions which are considered as constants plus an additional new summand which is optimized in the J-th step. For ease of notation we demand equal slicing of all Kronecker vectors for now. The approximation $\boldsymbol{y}[J]$ can therefore be written as

$$\boldsymbol{y}[J]=\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R+\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}. \tag{22}$$

Basically, we only have to insert (22) into the original problem (17) as we did it before. Unfortunately, the calculation is a bit cumbersome wherefore we consider nominator and denominator of the Rayleigh quotient separately. We begin with the denominator $\boldsymbol{y}^H\boldsymbol{y}$:

$$\begin{aligned}
\boldsymbol{y}^H\boldsymbol{y} &= \left(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R+\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)^H\left(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R+\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)\\
&= (\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R)^H(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R)+(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R)^H\left(\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)\\
&\quad+\left(\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)^H(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R)+\left(\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)^H\left(\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)\\
&= \left(\boldsymbol{x}_L^H\boldsymbol{x}_L\right)\left(\boldsymbol{x}_R^H\boldsymbol{x}_R\right)\left(\boldsymbol{x}_i^H\boldsymbol{x}_i\right)+\left[\sum_{j=1}^{J-1}\boldsymbol{y}_i\left(\boldsymbol{x}_L^H\boldsymbol{y}_L^{(j)}\right)\left(\boldsymbol{x}_R^H\boldsymbol{y}_R^{(j)}\right)\right]^H\boldsymbol{x}_i+\boldsymbol{x}_i^H\left[\sum_{j=1}^{J-1}\boldsymbol{y}_i\left(\boldsymbol{x}_L^H\boldsymbol{y}_L^{(j)}\right)\left(\boldsymbol{x}_R^H\boldsymbol{y}_R^{(j)}\right)\right]+\sum_{j=1}^{J-1}\sum_{k=1}^{J-1}\boldsymbol{y}^{(j)H}\boldsymbol{y}^{(j)}\\
&= \gamma\left(\boldsymbol{x}_i^H\boldsymbol{x}_i\right)+\boldsymbol{v}^H\boldsymbol{x}_i+\boldsymbol{x}_i^H\boldsymbol{v}+\varrho
\end{aligned} \tag{23}$$

The same calculation is done for the nominator:

$$
\begin{aligned}
\boldsymbol{y}^H\boldsymbol{H}\boldsymbol{y} &= \left(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R+\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)^H\left(\sum_{m=1}^{M}\alpha_m\cdot\boldsymbol{Q}_L^{(m)}\otimes\boldsymbol{Q}_i^{(m)}\otimes\boldsymbol{Q}_R^{(m)}\right)\left(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R+\sum_{j=1}^{J-1}\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)\\
&= \sum_{m=1}^{M}\alpha_m\left(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R\right)^H\left(\boldsymbol{Q}_L^{(m)}\otimes\boldsymbol{Q}_i^{(m)}\otimes\boldsymbol{Q}_R^{(m)}\right)\left(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R\right)\\
&\quad+\sum_{j=1}^{J-1}\sum_{m=1}^{M}\alpha_m\left(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R\right)^H\left(\boldsymbol{Q}_L^{(m)}\otimes\boldsymbol{Q}_i^{(m)}\otimes\boldsymbol{Q}_R^{(m)}\right)\left(\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)\\
&\quad+\sum_{j=1}^{J-1}\sum_{m=1}^{M}\alpha_m\left(\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)^H\left(\boldsymbol{Q}_L^{(m)}\otimes\boldsymbol{Q}_i^{(m)}\otimes\boldsymbol{Q}_R^{(m)}\right)\left(\boldsymbol{x}_L\otimes\boldsymbol{x}_i\otimes\boldsymbol{x}_R\right)\\
&\quad+\sum_{j=1}^{J-1}\sum_{k=1}^{J-1}\sum_{m=1}^{M}\alpha_m\left(\boldsymbol{y}_L^{(j)}\otimes\boldsymbol{y}_i^{(j)}\otimes\boldsymbol{y}_R^{(j)}\right)^H\left(\boldsymbol{Q}_L^{(m)}\otimes\boldsymbol{Q}_i^{(m)}\otimes\boldsymbol{Q}_R^{(m)}\right)\left(\boldsymbol{y}_L^{(k)}\otimes\boldsymbol{y}_i^{(k)}\otimes\boldsymbol{y}_R^{(k)}\right)\\
&= \boldsymbol{x}_i^H\left(\sum_{m=1}^{M}\alpha_m\left(\boldsymbol{x}_L\boldsymbol{Q}_L^{(m)}\boldsymbol{x}_R\right)\left(\boldsymbol{x}_R\boldsymbol{Q}_R^{(m)}\boldsymbol{x}_R\right)\boldsymbol{Q}_i^{(m)}\right)\boldsymbol{x}_i+\left[\sum_{j=1}^{J-1}\sum_{m=1}^{M}\left(\boldsymbol{x}_L^H\boldsymbol{Q}_L^{(m)}\boldsymbol{y}_L^{(j)}\right)\left(\boldsymbol{x}_R^H\boldsymbol{Q}_R^{(m)}\boldsymbol{y}_R^{(j)}\right)\left(\boldsymbol{Q}_i^{(m)}\boldsymbol{y}_i^{(j)}\right)\right]^H\boldsymbol{x}_i\\
&\quad+\boldsymbol{x}_i^H\left[\sum_{j=1}^{J-1}\sum_{m=1}^{M}\left(\boldsymbol{x}_L^H\boldsymbol{Q}_L^{(m)}\boldsymbol{y}_L^{(j)}\right)\left(\boldsymbol{x}_R^H\boldsymbol{Q}_R^{(m)}\boldsymbol{y}_R^{(j)}\right)\left(\boldsymbol{Q}_i^{(m)}\boldsymbol{y}_i^{(j)}\right)\right]+\sum_{j=1}^{J-1}\sum_{k=1}^{J-1}\sum_{m=1}^{M}\alpha_m\cdot\boldsymbol{y}^{(j)H}\boldsymbol{Q}^{(m)}\boldsymbol{y}^{(k)}\\
&= \boldsymbol{x}_i^H\boldsymbol{H}_i'\boldsymbol{x}_i+\boldsymbol{u}^H\boldsymbol{x}_i+\boldsymbol{x}_i^H\boldsymbol{u}+\beta
\end{aligned}
\tag{24}
$$

Putting the results from (23) and (24) together we finally obtain a new representation of the Rayleigh quotient:

$$
\frac{\boldsymbol{y}^H\boldsymbol{H}\boldsymbol{y}}{\boldsymbol{y}^H\boldsymbol{y}}=\frac{\boldsymbol{x}_i^H\boldsymbol{H}_i'\boldsymbol{x}_i+\boldsymbol{u}^H\boldsymbol{x}_i+\boldsymbol{x}_i^H\boldsymbol{u}+\beta}{\gamma\left(\boldsymbol{x}_i^H\boldsymbol{x}_i\right)+\boldsymbol{v}^H\boldsymbol{x}_i+\boldsymbol{x}_i^H\boldsymbol{v}+\varrho}=\frac{\begin{bmatrix}\boldsymbol{x}_i & \mathbf{1}\end{bmatrix}\begin{bmatrix}\boldsymbol{H}_i & \boldsymbol{u}\\ \boldsymbol{u}^H & \beta\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_i\\ \mathbf{1}\end{bmatrix}}{\begin{bmatrix}\boldsymbol{x}_i^H & \mathbf{1}\end{bmatrix}\begin{bmatrix}\gamma\boldsymbol{I} & \boldsymbol{v}\\ \boldsymbol{v}^H & \varrho\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_i\\ \mathbf{1}\end{bmatrix}}=\frac{\boldsymbol{\zeta}^H\boldsymbol{A}\boldsymbol{\zeta}}{\boldsymbol{\zeta}^H\boldsymbol{B}\boldsymbol{\zeta}}
\tag{25}
$$

Apart from matrix $\boldsymbol{B}$ in the denominator, Equation (25) is another eigenvalue problem. To eliminate the matrix we use the following decomposition:

$$
\boldsymbol{B}=\boldsymbol{L}\boldsymbol{L}^H=\begin{bmatrix}\tilde{\boldsymbol{L}} & 0\\ \boldsymbol{w}^H & \alpha\end{bmatrix}\begin{bmatrix}\tilde{\boldsymbol{L}}^H & \boldsymbol{w}\\ 0 & \overline{\alpha}\end{bmatrix}=\begin{bmatrix}\tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^H & \tilde{\boldsymbol{L}}\boldsymbol{w}\\ \left(\tilde{\boldsymbol{L}}\boldsymbol{w}\right)^H & \boldsymbol{w}^H\boldsymbol{w}+\alpha\overline{\alpha}\end{bmatrix}\overset{!}{=}\begin{bmatrix}\gamma\boldsymbol{I} & \boldsymbol{v}\\ \boldsymbol{v}^H & \varrho\end{bmatrix}
$$

Expressions for $\tilde{\boldsymbol{L}}$, $\boldsymbol{w}$ and α can be found easily. The matrix $\boldsymbol{L}$ is then given by

$$
\boldsymbol{L}=\begin{bmatrix}\sqrt{\gamma}\boldsymbol{I} & \mathbf{0}\\ {}^{1}\!/_{\sqrt{\gamma}}\boldsymbol{v}^H & \sqrt{\varrho-{}^{1}\!/_{\gamma}\boldsymbol{v}^H\boldsymbol{v}}\end{bmatrix}.
\tag{26}
$$

The inverse of $\boldsymbol{L}$ exists if $\sqrt{\varrho-\frac{1}{\gamma}\boldsymbol{v}^H\boldsymbol{v}}\neq 0$ and is analytically given by

$$
\boldsymbol{L}^{-1}=\begin{bmatrix}{}^{1}\!/_{\sqrt{\gamma}}\boldsymbol{I} & \mathbf{0}\\ -\frac{1}{\gamma\sqrt{\varrho-{}^{1}\!/_{\gamma}\boldsymbol{v}^H\boldsymbol{v}}}\boldsymbol{v}^H & \frac{1}{\sqrt{\varrho-{}^{1}\!/_{\gamma}\boldsymbol{v}^H\boldsymbol{v}}}\end{bmatrix}.
\tag{27}
$$

Using these results and the substitution $\boldsymbol{\eta}=\boldsymbol{L}^H\boldsymbol{\zeta}$ we can rewrite (25) as

$$
\frac{\boldsymbol{\zeta}^H\boldsymbol{A}\boldsymbol{\zeta}}{\boldsymbol{\zeta}^H\boldsymbol{B}\boldsymbol{\zeta}}=\frac{\boldsymbol{\zeta}^H\boldsymbol{A}\boldsymbol{\zeta}}{\left(\boldsymbol{L}^H\boldsymbol{\zeta}\right)^H\left(\boldsymbol{L}^H\boldsymbol{\zeta}\right)}=\frac{\boldsymbol{\eta}^H\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-H}\boldsymbol{\eta}}{\boldsymbol{\eta}^H\boldsymbol{\eta}}
\tag{28}
$$

where $\boldsymbol{L}^{-H}$ denotes the hermitian of $\boldsymbol{L}^{-1}$. Equation (28) is now a standard eigenvalue problem which can be solved using existing libraries.

So far we assumed that all summands of $\boldsymbol{y}$ are equally sliced. If we allow a sum of arbitrarily sliced vectors, partitioning of $\boldsymbol{y}$ as given by (22) is no longer possible. In its most general form, $\boldsymbol{y}[J]$ is given as

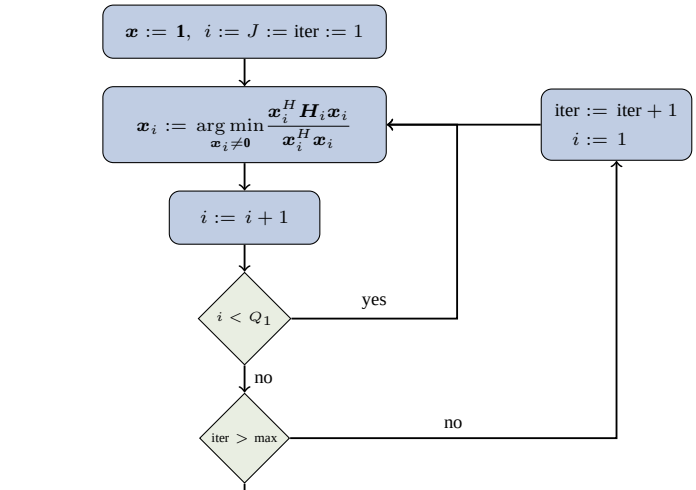
$$\boldsymbol{y}[J] = \boldsymbol{x}_L \otimes \boldsymbol{x}_i \otimes \boldsymbol{x}_R + \sum_{J=1}^{J-1} \boldsymbol{y}_1^{(j)} \otimes \ldots \otimes \boldsymbol{y}_{Q_j}^{(j)} \tag{29}$$

where Q_j denotes the number of slices of the j-th vector. Given this generalization some of the products in (23) and (24) refuse compact notation since we find no longer matching slices for $\boldsymbol{x}_i$. In particular, the expressions for $\boldsymbol{u}$ and $\boldsymbol{v}$ are no longer valid. In a similar way, the calculation of ϱ as a sum of scalar product of unequally sliced vectors becomes more difficult. A formal description of these products is skipped here but can be found in [1]. Instead, we focus on the insight of how to calculate these products in the next Section.

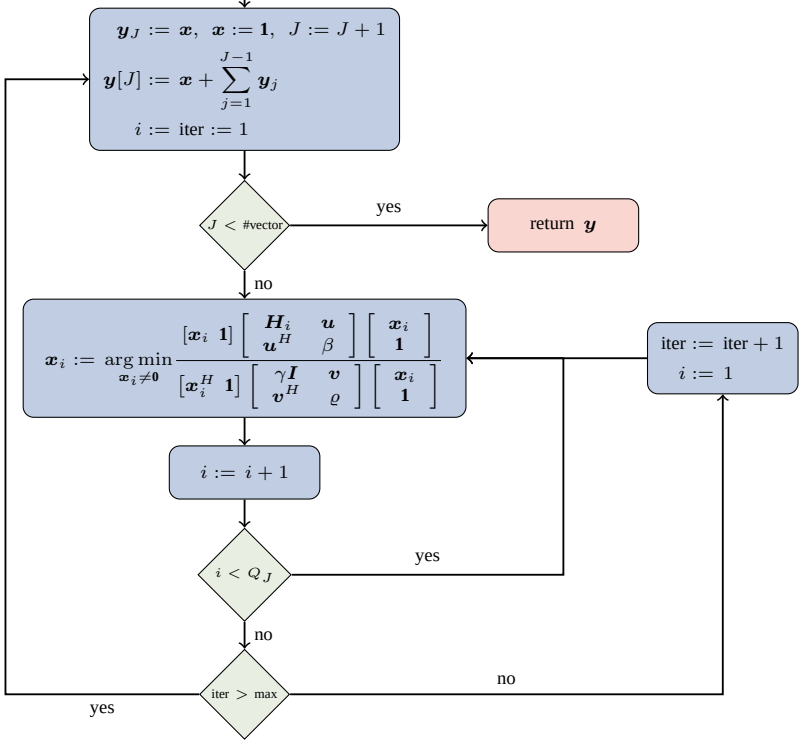
C. Outline of the VTA algorithm

Using the results of the previous Sections we can give an algorithm which calculates an arbitrary close approximation $\boldsymbol{y}$ of the eigenvector $\boldsymbol{y}^*$ and its corresponding eigenvalue $\lambda_{\min}^*$. The outline of the VTA algorithm is shown below:

Phase 1: Calculate a start solution $\boldsymbol{y}[1] = \boldsymbol{x}$ by solving (21) for all slices $i = 1, \ldots, Q_1$. Iterating multiple times over the whole vector $\boldsymbol{x}$ improves the solution since all but the current slice i are considered constant. After reaching the maximum number of iterations (or some numerical break criterion), the vector $\boldsymbol{x}$ is accepted as first approximation and the algorithm turns over in phase 2.



Phase 2: The starting solution is set to the current approximation $\boldsymbol{y}[1]$ and a new summand $\boldsymbol{x}$ is added. Afterwards, the new summand $\boldsymbol{x}$ is optimized while considering the other summand as constant. This results in problem (25) which can be transformed into another eigenvalue problem as described in Section IV-B. This in turn can be solved for each slice $\boldsymbol{x}_i$ individually. Multiple iterations over the new summand again improve the approximation. Finally, after reaching the maximum number of iterations (or some numerical break criterion), the summand $\boldsymbol{x}$ is accepted and considered as constant while another one is added. This continues until some final break condition is reached.



V. Custom BLAS routines

This section focuses on the new BLAS routines necessary due to the special layout of operands. At first we will briefly discuss the scalar product of Kronecker vectors with matching slices and how a slice can be excluded from the product. Afterwards we turn toward the case of unequal slicing. Finally, the products between the Hamiltonian and vectors are discussed in Section V-E and V-F.

A. Inner product between Kronecker vectors with matching slices

The inner product of two Kronecker vectors

$$\boldsymbol{x} = \boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_n \quad \text{and} \quad \boldsymbol{y} = \boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_n \tag{30}$$

of the same size with matching slices ($\dim \boldsymbol{x}_i = \dim \boldsymbol{y}_i \ \forall i \in \{1, \ldots, n\}$) is easy to calculate. According to the properties of the Kronecker product as given by (4) and (5) it reduces to a series of scalar products between slices:

$$\boldsymbol{x}^H \boldsymbol{y} = (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_n)^H (\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_n) = (\boldsymbol{x}_1 \boldsymbol{y}_1) \otimes \ldots \otimes \left(\boldsymbol{x}_n^H \boldsymbol{y}_n\right) = \prod_{j=1}^{n} \boldsymbol{x}_j^H \boldsymbol{y}_j \tag{31}$$

Note, that the Kronecker product between scalar values reduces to multiplication.

B. Inner product between Kronecker vectors with matching slices and excludes

Sometimes it is required to calculate the inner product between two Kronecker vectors

$$\boldsymbol{x} = \boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_i \otimes \ldots \otimes \boldsymbol{x}_n \quad \text{and} \quad \boldsymbol{y} = \boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_i \otimes \ldots \otimes \boldsymbol{y}_n \tag{32}$$

while excluding the slice $\boldsymbol{x}_i$ from multiplication. Provided that the slicing of both vectors match, we can easily calculate the scalar products on the right and left side of $\boldsymbol{x}_i$:

$$\begin{aligned} \boldsymbol{x}^H \boldsymbol{y} &= (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_i \otimes \ldots \otimes \boldsymbol{x}_n)^H (\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_i \otimes \ldots \otimes \boldsymbol{y}_n) \\ &= (\boldsymbol{x}_1 \boldsymbol{y}_1) \otimes \ldots \otimes \left(\boldsymbol{x}_i^H \boldsymbol{y}_i\right) \otimes \ldots \otimes \left(\boldsymbol{x}_n^H \boldsymbol{y}_n\right) = \boldsymbol{x}_i^H \left(\boldsymbol{y}_i \prod_{j=1, j\neq i}^{n} \boldsymbol{x}_j^H \boldsymbol{y}_j \right) = \boldsymbol{x}_i^H \tilde{\boldsymbol{y}}_i \end{aligned} \tag{33}$$

The dimension of the resulting vector $\tilde{\boldsymbol{y}}_i$ corresponds to the dimension of the excluded slice $\boldsymbol{x}_i$.

C. Inner product between Kronecker vectors with unequal slices

We consider again the inner product $\boldsymbol{x}^H \boldsymbol{y}$. For Kronecker vectors of arbitrary slicing it is no longer possible to reduce it to a series of scalar products between slices. Consider the Kronecker vectors

$$\boldsymbol{x} = \boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_n \quad \text{and} \quad \boldsymbol{y} = \boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_m \tag{34}$$

where $\dim \boldsymbol{x} = \dim \boldsymbol{y}$ but in general $\dim \boldsymbol{x}_i \neq \dim \boldsymbol{y}_j$. The inner product $\boldsymbol{x}^H \boldsymbol{y}$ can be calculated by iteratively *contracting* the larger of the two right-most slices. Let $\dim \boldsymbol{x}_n = k$ and $\dim \boldsymbol{x}_m = l$. In each step we have to differentiate between the following two cases (exemplarily given for the first step only):

$$\begin{aligned} k \leq l: \quad \boldsymbol{x}^H \boldsymbol{y} &= (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_{n-1} \otimes \boldsymbol{x}_n)^H (\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_{m-1} \otimes \boldsymbol{y}_m) \\ &= (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_{n-1})^H \left(\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_{m-1} \otimes \begin{bmatrix} y_{m,1} & \ldots & y_{m,k} \\ \vdots & \ddots & \vdots \\ y_{m,l-k} & \ldots & y_{m,l} \end{bmatrix} \begin{bmatrix} \overline{x}_{n,1} \\ \vdots \\ \overline{x}_{n,k} \end{bmatrix} \right) \\ &= (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_{n-1})^H (\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_{m-1} \otimes \tilde{\boldsymbol{y}}_m) \end{aligned}$$

$$\begin{aligned} k > l: \quad \boldsymbol{x}^H \boldsymbol{y} &= (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_{n-1} \otimes \boldsymbol{x}_n)^H (\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_{m-1} \otimes \boldsymbol{y}_m) \\ &= \left(\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_{n-1} \otimes \begin{bmatrix} x_{n,1} & \ldots & x_{n,l} \\ \vdots & \ddots & \vdots \\ x_{n,k-l} & \ldots & x_{n,k} \end{bmatrix} \begin{bmatrix} \overline{y}_{m,1} \\ \vdots \\ \overline{y}_{m,l} \end{bmatrix} \right)^H (\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_{m-1}) \\ &= (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_{n-1} \otimes \tilde{\boldsymbol{x}}_n)^H (\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_{m-1}) \end{aligned}$$

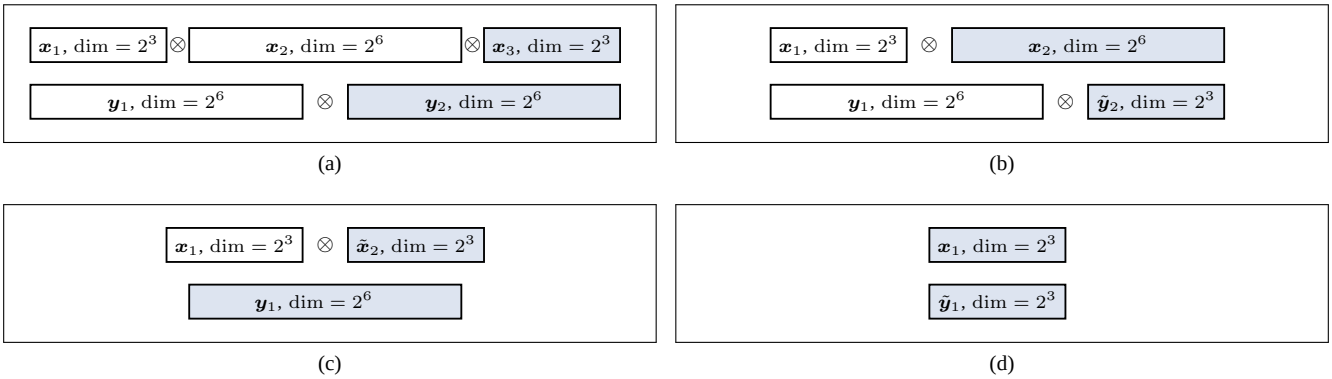


Fig. 3: Example: Inner product between $\boldsymbol{x}^H\boldsymbol{y} = (\boldsymbol{x}_1 \otimes \boldsymbol{x}_2 \otimes \boldsymbol{x}_3)(\boldsymbol{y}_1 \otimes \boldsymbol{y}_2)$. Subfigure (a) shows the first step where $\boldsymbol{x}_3$ is multiplied with $\boldsymbol{y}_2$ resulting in a contraction and a smaller slice $\tilde{\boldsymbol{y}}_2$. Afterwards, slice $\boldsymbol{x}_2$ is contracted in (b) by multiplication with slice $\tilde{\boldsymbol{y}}_2$. The process is continued until two slices of matching size remain (d).

The inner product can therefore be expressed by a sequence of matrix-vector products where the dimensions of the matrices are always determined by the size of the smaller slices. The size of the resulting vector is given by l/k in case of $k \leq l$ respectively k/l for $k > l$ which is why we call it *contraction*. Two slices of equal size are reduced to a scalar value which scales another slice in the subsequent step. A concrete example is shown in Figure 3.

D. Inner product between Kronecker vectors with unequal slices and excludes

The most complicated variant is the inner product of $\boldsymbol{x}^H\boldsymbol{y}$ where a specific slice $\boldsymbol{x}_i$ should be excluded from multiplication. We can start again by contracting the larger slice in each step beginning on the right side until we hit the slice $\boldsymbol{x}_i$:

$$\begin{aligned}\boldsymbol{x}^H\boldsymbol{y} &= (\boldsymbol{x}_1 \otimes \ldots \boldsymbol{x}_n)(\boldsymbol{y}_1 \otimes \ldots \otimes \boldsymbol{y}_m)\\ &= (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_i \otimes \tilde{\boldsymbol{x}}_{i+1})(\boldsymbol{y}_1 \otimes \ldots \otimes \tilde{\boldsymbol{y}}_q)\\ &= (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_i)\left(\boldsymbol{y}_1 \otimes \ldots \otimes \tilde{\boldsymbol{y}}'_q\right)\end{aligned}$$

At this point we turn over and start the contraction from the left-hand side. Again we decompose the larger slice into a matrix which matches in dimension with the smaller slice. Given $k = \dim \boldsymbol{x}_1$ and $l = \dim \boldsymbol{y}_1$ the first step of the contraction is given as follows:

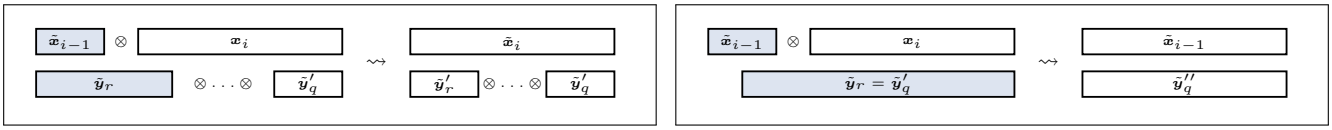
$$\begin{aligned}k \leq l: \quad (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_i)\left(\boldsymbol{y}_1 \otimes \ldots \otimes \tilde{\boldsymbol{y}}'_q\right) &= (\boldsymbol{x}_2 \otimes \ldots \otimes \boldsymbol{x}_i)^H \left(\begin{bmatrix} y_{1,1} & \ldots & y_{1,k} \\ \vdots & \ddots & \vdots \\ y_{1,l-k} & \ldots & y_{1,l} \end{bmatrix}^T \begin{bmatrix} \overline{x}_{1,1} \\ \vdots \\ \overline{x}_{1,k} \end{bmatrix} \otimes \boldsymbol{y}_2 \otimes \ldots \otimes \tilde{\boldsymbol{y}}'_q \right)\\ &= (\boldsymbol{x}_2 \otimes \ldots \otimes \boldsymbol{x}_i)^H \left(\tilde{\boldsymbol{y}}_1 \otimes \boldsymbol{y}_2 \otimes \ldots \otimes \tilde{\boldsymbol{y}}'_q\right)\end{aligned}$$

$$\begin{aligned}k > l: \quad (\boldsymbol{x}_1 \otimes \ldots \otimes \boldsymbol{x}_i)\left(\boldsymbol{y}_1 \otimes \ldots \otimes \tilde{\boldsymbol{y}}'_q\right) &= \left(\begin{bmatrix} x_{1,1} & \ldots & x_{1,l} \\ \vdots & \ddots & \vdots \\ x_{1,k-l} & \ldots & x_{1,k} \end{bmatrix}^T \begin{bmatrix} \overline{y}_{1,1} \\ \vdots \\ \overline{y}_{1,l} \end{bmatrix} \boldsymbol{x}_2 \otimes \ldots \otimes \boldsymbol{x}_i \right)^H \left(\boldsymbol{y}_2 \otimes \ldots \otimes \tilde{\boldsymbol{y}}'_q\right)\\ &= (\tilde{\boldsymbol{x}}_1 \otimes \boldsymbol{x}_2 \otimes \ldots \otimes \boldsymbol{x}_i)^H \left(\boldsymbol{y}_2 \otimes \ldots \otimes \tilde{\boldsymbol{y}}'_q\right)\end{aligned}$$

Contraction continues until we hit $\boldsymbol{x}_i$ from left again. Depending on the dimensions of the right-most slice of $\boldsymbol{y}$ and the excluded slice we have to differentiate between the following two cases:

1) $\dim \tilde{\boldsymbol{y}}'_q \leq \dim \boldsymbol{x}_i : \boldsymbol{x}^H\boldsymbol{y} = \boldsymbol{x}_i^H \left(\tilde{\boldsymbol{y}}'_r \otimes \ldots \otimes \tilde{\boldsymbol{y}}'_q\right)$
2) $\dim \tilde{\boldsymbol{y}}'_q > \dim \boldsymbol{x}_i : \boldsymbol{x}^H\boldsymbol{y} = (\boldsymbol{x}_{i-1} \otimes \boldsymbol{x}_i)^H \tilde{\boldsymbol{y}}'_q = \boldsymbol{x}_i^H \tilde{\boldsymbol{y}}''_q$

Case 1) is depicted in Figure 4a. The excluded slice $\boldsymbol{x}_i$ overlaps at least two slices of $\boldsymbol{y}$. Multiple special cases can arise here. In general, we have to build the Kronecker product between the remaining slices of $\boldsymbol{y}$. Note, that both $\tilde{\boldsymbol{y}}'_r$ and $\tilde{\boldsymbol{y}}'_q$ might degenerate to scalar values. The situation of case 2) is shown in Figure 4b. Slice $\boldsymbol{y}_q$ is contracted from both sides.



(a) $\dim \tilde{\boldsymbol{y}}'_q \leq \dim \boldsymbol{x}_i$ (b) $\dim \tilde{\boldsymbol{y}}'_q > \dim \boldsymbol{x}_i$

Fig. 4: Contraction from the left side after contraction from the right side has been completed.

E. Product between a Hamiltonian and a normal vector

The product between a Hamiltonian $\boldsymbol{H}$ and a normal vector $\boldsymbol{x}$ is given by

$$\boldsymbol{H}\boldsymbol{x} = \sum_{m=1}^{M} \alpha_m \cdot \left(\boldsymbol{Q}_1^{(m)} \otimes \boldsymbol{Q}_2^{(m)} \otimes \ldots \otimes \boldsymbol{Q}_P^{(m)} \right) \boldsymbol{x}, \quad \boldsymbol{Q}_i \in \mathcal{Q}. \tag{35}$$

First, we note that each summand $\boldsymbol{H}^{(m)}$ has exactly one non-zero entry $h_{i,j} \in \{\pm 1, \pm i\}$ per row and column which is additionally scaled by a single factor α_m. Therefore, the summands $\boldsymbol{H}^{(m)}$ are basically permutation matrices. We can further decompose a single summand $\boldsymbol{H}^{(m)}$ into

$$\alpha_m \boldsymbol{H}^{(m)} = \alpha_m \cdot \left(\boldsymbol{Q}_1^{(m)} \otimes \boldsymbol{Q}_2^{(m)} \otimes \ldots \otimes \boldsymbol{Q}_P^{(m)} \right) = \alpha_m \cdot \prod_{i=1}^{P} \underbrace{\boldsymbol{I}_2 \otimes \ldots \boldsymbol{I}_2}_{i-1 \text{ times}} \otimes Q_i^{(m)} \otimes \underbrace{\boldsymbol{I}_2 \otimes \ldots \otimes \boldsymbol{I}_2}_{P-i \text{ times}}. \tag{36}$$

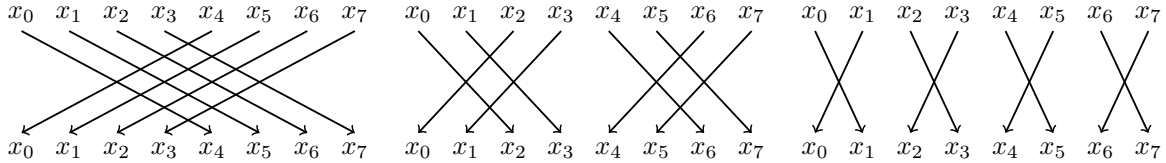
Note, that for $\boldsymbol{Q}_i = \boldsymbol{I}_2$ the corresponding i-th product has no effect. Let $\mathcal{X} \subseteq \{1, \ldots, P\}$ denote the set of indices for which $\boldsymbol{Q}_i \neq \boldsymbol{I}_2$. Then we can write the product $\boldsymbol{H}\boldsymbol{x}$ as

$$\sum_{m=1}^{M} \boldsymbol{H}^{(m)} \boldsymbol{x} = \sum_{m=1}^{M} \left[\left(\prod_{i \in \mathcal{X}} \hat{\boldsymbol{H}}_i^{(m)} \right) \cdot \boldsymbol{x} \right]. \tag{37}$$

Equation (37) only consists of permutations and some scaling operations. We now derive an efficient way to calculate this product by considering some concrete examples. Assume $m = 1$, $\alpha_1 = 1$ and $P = 3$. The vector $\boldsymbol{x}$ has thus eight components which we index as $x_0, \ldots, x_7$. Consider the three Hamiltonians

$$\hat{\boldsymbol{H}}_1 = \sigma_x \otimes \ \boldsymbol{I}_2 \otimes \boldsymbol{I}_2, \ \hat{\boldsymbol{H}}_2 = \boldsymbol{I}_2 \otimes \sigma_x \otimes \boldsymbol{I}_2, \ \hat{\boldsymbol{H}}_3 = \boldsymbol{I}_2 \otimes \boldsymbol{I}_2 \otimes \sigma_x.$$

Depending on the position i of the Pauli matrix $\boldsymbol{Q}_i \neq \boldsymbol{I}_2$ the permutation matrix takes a different shape. The permutations are illustrated in Figure 5. Chaining of these permutations gives the final result. This can be calculated easily by a sequence of



(a) Permutation defined by $\hat{\boldsymbol{H}}_1$ (b) Permutation defined by $\hat{\boldsymbol{H}}_2$ (c) Permutation defined by $\hat{\boldsymbol{H}}_3$

Fig. 5: Permutations defined by Hamiltonians consisting of 2×2 identiy matrices with a single σ_x at the i-th position.

logic operations. Assume we seek to calculate the n-th component x'_n of $\boldsymbol{x}' = \hat{\boldsymbol{H}}_1 \hat{\boldsymbol{H}}_2 \hat{\boldsymbol{H}}_3 \boldsymbol{x}$. Consider the binary representation of n which consists of three bits in our example. Let $n_{(2)}[i]$ denote the i-th bit in the binary representation of n beginning with the highest order bit, i.e. $n_{(2)}[1]$. The matrix $\hat{\boldsymbol{H}}_1$ obviously adds an offset of four if $n < 4$ or subtracts an offset of four if $n \geq 4$. This corresponds to the logical operation $n_{(2)}[1]$ XOR 1. In a similar way $\hat{\boldsymbol{H}}_2$ adds or subtracts an offset of two. In general, the effect of a Pauli matrix σ_x at position i is given by $n_{(2)}[i]$ XOR 1.

Similar rules can be derived for the other two Pauli matrices. However, scaling must be considered for σ_y and σ_z which can be achieved easily by using a temporary variable ν which accumulates all scaling factors. After the component x_k of the original vector $\boldsymbol{x}$ has been determined, scaling is done by $x'_n = x_k \cdot \nu$. The product $\boldsymbol{H}^{(m)}\boldsymbol{x}$ therefore requires $\dim(x) \cdot P$ operations. Thus, the product $\boldsymbol{H}\boldsymbol{x}$ can be calculated with $\mathcal{O}(\dim(x) \cdot mP)$ operations.

F. Product between a Hamiltonian and a Kronecker vector

Let $\boldsymbol{x}$ be a Kronecker vector of arbitrary slicing such that it matches in dimension with the Hamiltonian $\boldsymbol{H}$. Note, that for this reason the dimension of each slice has to be a power of two. Consequently, we can write the product as

$$\begin{aligned}\boldsymbol{H}\boldsymbol{x} &= \left(\sum_{m=1}^{M} \alpha_m \cdot \boldsymbol{Q}_1^{(m)} \otimes \ldots \otimes \boldsymbol{Q}_P^{(m)}\right) \cdot (\boldsymbol{x}_1 \otimes \cdots \otimes \boldsymbol{x}_Q) \\ &= \sum_{m=1}^{M} \alpha_m \cdot \tilde{\boldsymbol{Q}}_1^{(m)} \boldsymbol{x}_1 \otimes \ldots \otimes \tilde{\boldsymbol{Q}}_Q^{(m)} \boldsymbol{x}_Q. \end{aligned} \tag{38}$$

Thereby the original product is decomposed into a sequence of Q products between matrices $\boldsymbol{Q}_j^{(m)}$ made up exclusively of Pauli matrices and ordinary vectors $\boldsymbol{x}_j$ (slices). These we can calculated as outlined in Section V-E.

VI. Numerical results

To analyze the speed of convergence we rely on the relative error $e(J, i)$ which we define as

$$e(J, i) = \frac{\lambda[J, i] - \lambda^*}{\lambda^*}. \tag{39}$$

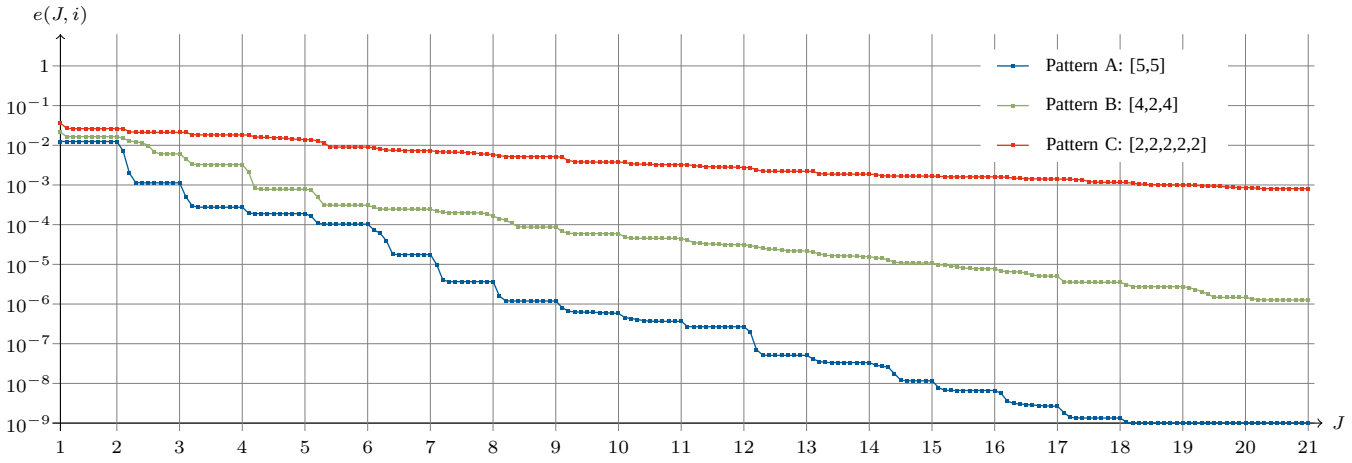
Here, the value $\lambda[J, i]$ denotes the approximation of the eigenvalue in the i-th iteration using J summands. We fix the number of iterations per summand to a constant value to point out eventual benefits of passing multiple times over a new summand. If the individual iterations for a given summand are not of interest, we use the relative error $e(J)$ which denotes the relative error using J summands after the active summand has been optimized for a constant number of iterations. The Hamiltonians in use are based on the 1D-Ising model and consist of $M = 2P - 1$ summands. The coefficients α_m are all set to 1. This way we can rely on known eigenvalue approximations obtained by MPS [3] even for problems which are too large to be solved exactly.

Section VI-A investigates the influence of slice sizes on the speed of convergence. Section VI-B considers sequences of overlapping slice patterns. In Section VI-C we derive an optimized sequence of overlapping slice patterns for some concrete examples. Finally, Section VI-D summarizes results for a large problem ($P = 100$) using different pattern sequences.

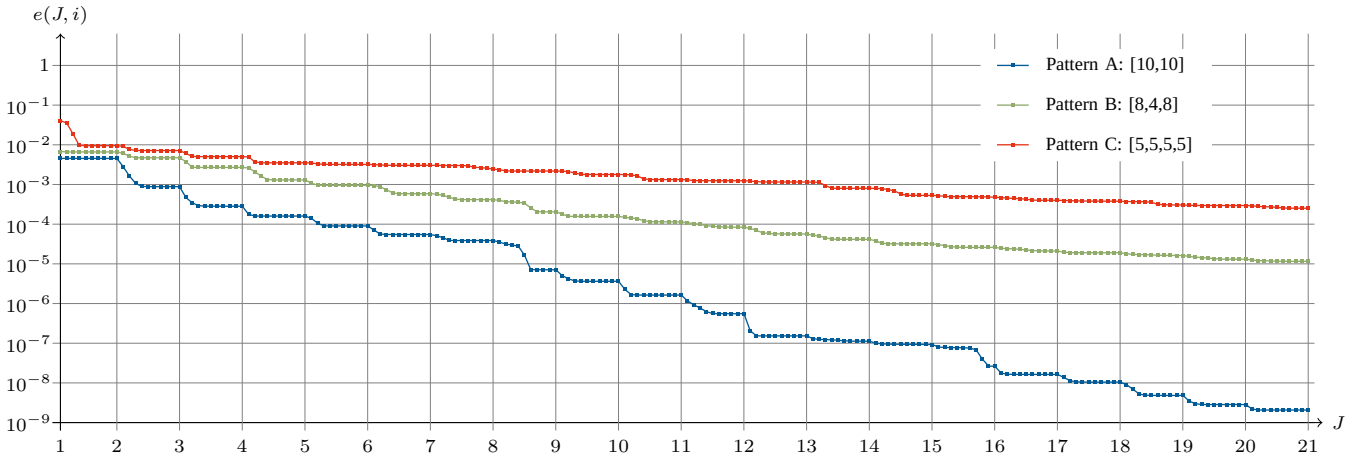
A. Non-overlapping slice sequences

To show the tremendous influence of the slice sizes we choose Hamiltonians for $P \in \{10, 20, 30\}$ and calculate the approximation of the eigenvalue using three different slice patterns for each size. The results for up to 20 summands with ten iterations per summand are shown in Figure 6. Patterns using large slices generally outperform patterns with smaller slices. This is no surprise since larger slices offer considerably more degrees of freedom for optimization. For example, pattern A for $P = 10$ is given as $[5, 5]$ and offers $2 \cdot 2^5 = 64$ degrees of freedom compared to pattern B given as $[4, 2, 4]$ which only offers $2 \cdot 2^4 + 2^2 = 36$ degrees of freedom. Consequently, large slices should be preferred as long as the underlying hardware is able to solve the subproblems in affordable time. Using smaller slices but more summands is not a good idea when it comes to computation time because each additional summand slows down vector operations. There is no general rule for the tradeoff between slice sizes and number of summands since it depends on the hardware in use.

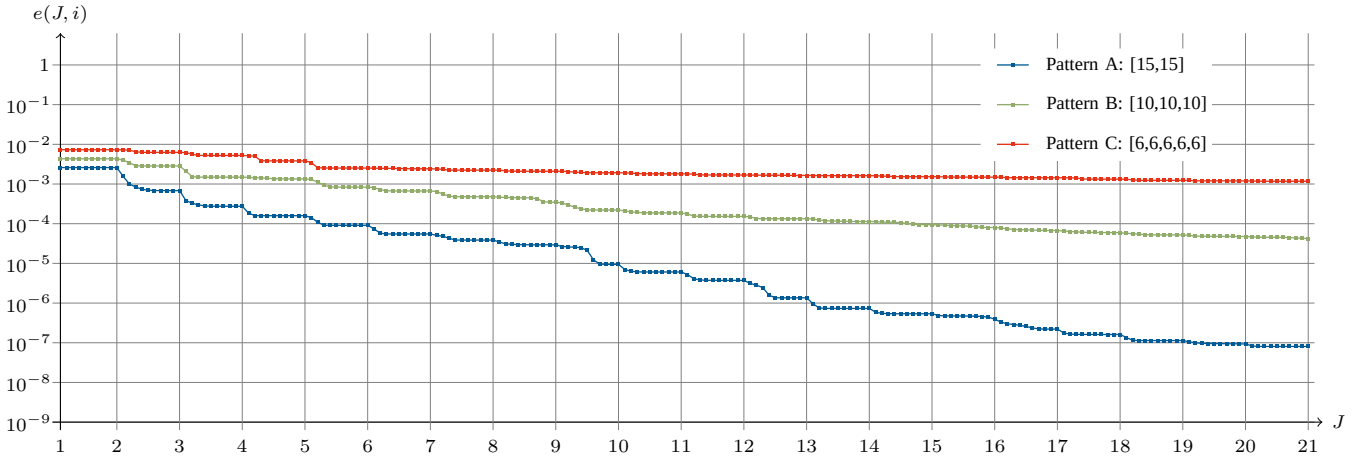
In this example we fixed the number of iterations to ten per summand. However, the results show some kind of step function. This suggests to use a numerical break condition, e.g. if the relative change between the i-th and $(i-1)$-th iteration is smaller than some threshold. Unfortunately, there is an anomaly which prevents such a break criterion. Consider for example pattern A at $J = 2$ in Figure 6b. As one would suspect, the approximation is immediately improved by the new summand. This is not the case for $J = 8$. Here, it takes multiple iterations before the new summand affects the approximation. A break condition as described before would eventually skip iterations which lead to a significant improvement. The reason of this behavior is unclear but we suspect that the optimization gets stuck in a local minimum which might got over after a sufficient number of iterations. This makes it difficult to use a numerical break condition since small changes over the last optimization steps do not necessarily indicate that the current slice is near an optimal value.



(a) $P = 10$



(b) $P = 20$



(c) $P = 30$

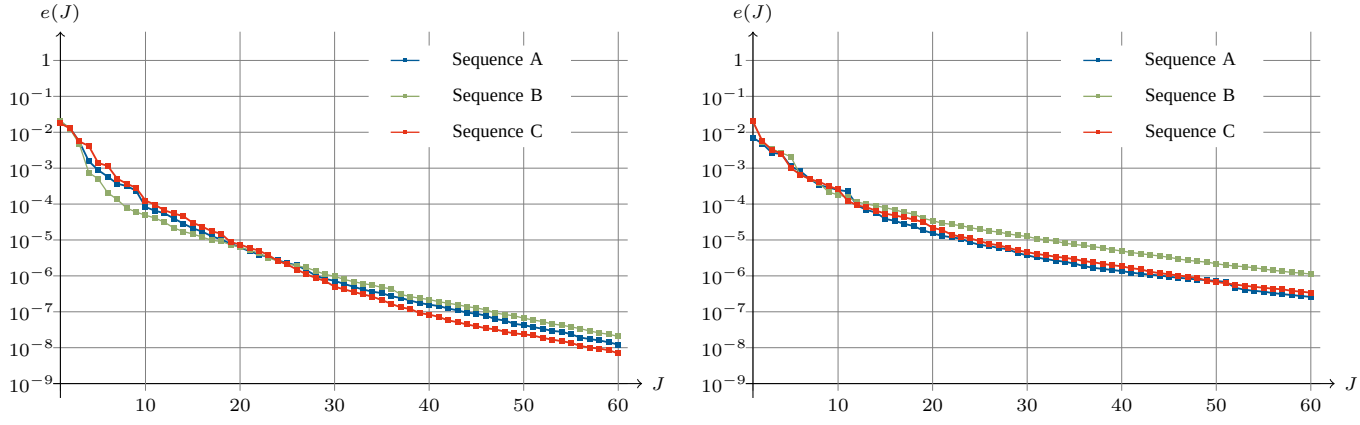
Fig. 6: Relative error for different problem sizes using non-overlapping slice patterns.

B. Overlapping slice sequences

So far we only considered sequences of a single slice pattern for all summands. For patterns using slices of different sizes like $[3,4,3]$ or $[7,6,7]$ hopes are that we can improve precision by choosing shifted patterns. This adds more degrees of freedom to optimization since the subspace spanned by Kronecker vectors using the slice pattern $[3,4,3]$ is different from the one spanned by other vectors using the pattern $[4,3,3]$ or $[3,3,4]$. To investigate the benefits of overlapping slice patterns we choose Hamiltonians of size $P \in \{10,20\}$. The tests were performed using the following sequences:

- $P = 10$:
  - Sequence A consits exclusively of the pattern $[3,4,3]$.
  - Sequence B uses periodically shifted patterns, i.e. $[4,3,3][3,4,3][3,3,4][4,3,3]\ldots$
  - Sequence C uses 20 times pattern $[4,3,3]$, afterwards 20 times $[3,4,3]$ and finally 20 times $[3,3,4]$.
- $P = 20$:
  - Sequence A consits exclusively of the pattern $[7,6,7]$.
  - Sequence B uses periodically shifted patterns, i.e. $[6,7,7][7,6,7][7,7,6][6,7,7]\ldots$
  - Sequence C uses 20 times pattern $[6,7,7]$, afterwards 20 times $[7,6,7]$ and finally 20 times $[7,7,6]$.

The sequences for $P = 20$ are defined analogous. The results for up to 60 summands are shown in Figure 7. In both cases, sequence B is the worst one. However, the differences between A and C are small and may be compensated by a small number of additional summands. Nevertheless, these examples show that the sequence of overlapping patterns has an influence on the speed of convergence. In the next Section we try to find an optimized sequence which outperforms non-overlapping sequences.



(a) $P = 10$ (b) $P = 20$

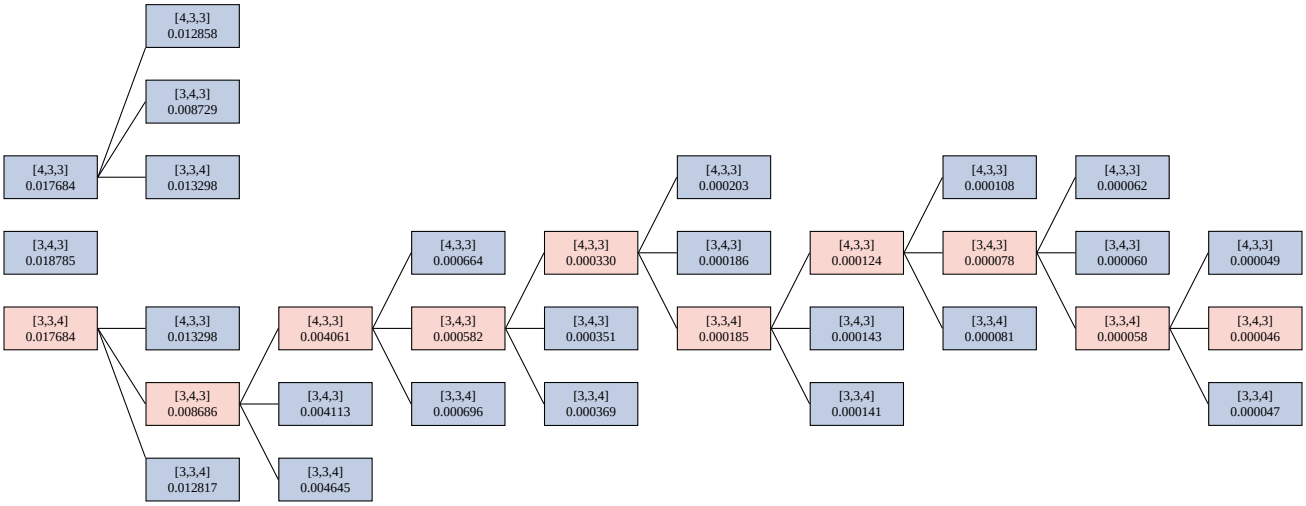
Fig. 7: Relative error for different problem sizes using overlapping slice patterns.

C. Optimized sequence of overlapping slices

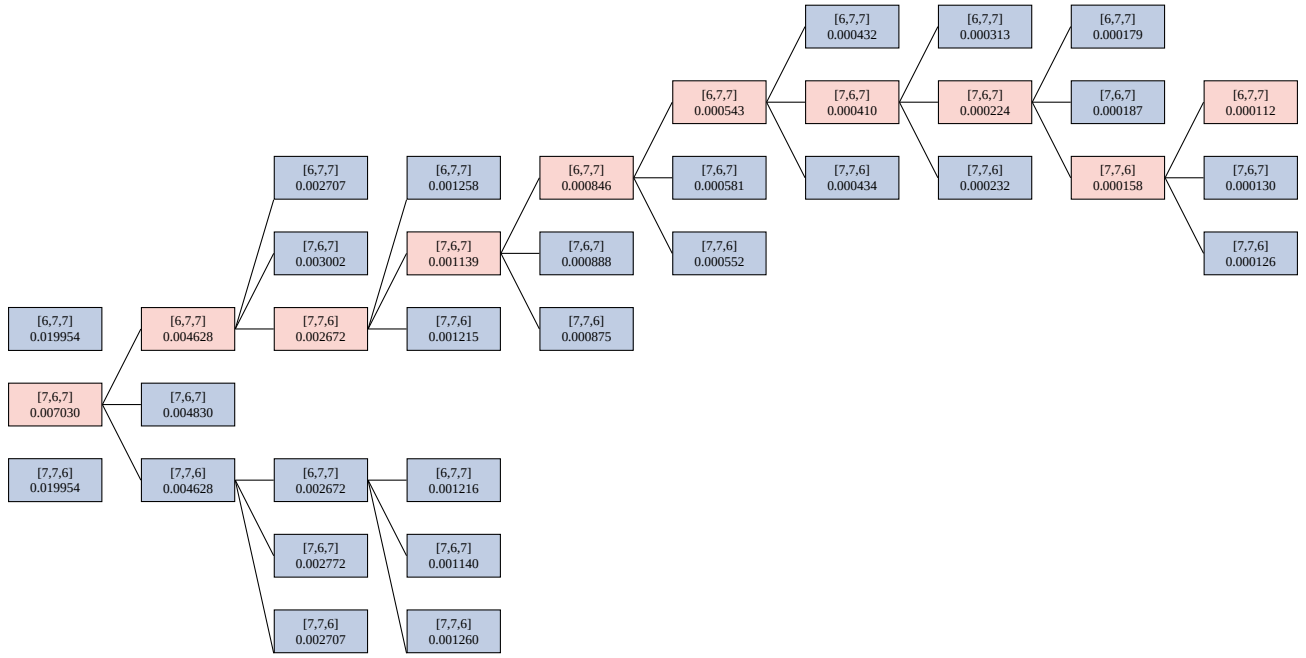
To investigate which improvement might be obtained using overlapping we now try to determine an optimized sequence of patterns for the examples introduced in Section VI-B. We use again Hamiltonians of size $P \in \{10,20\}$. The patterns we choose from are as follows:

- $[4,3,3]$, $[3,4,3]$ and $[3,3,4]$ for $P = 10$
- $[6,7,7]$, $[7,6,7]$ and $[7,7,6]$ for $P = 20$

For each new summand we test all three patterns using ten iterations and choose the one which minimizes the relative error $e(J)$. This results in some kind of greedy algorithm as depicted in Figure 8 (the optimal path is highlighted). Note, that this scheme is only a heuristic for an optimal sequence. Since we do not test all possibilities, there is no way to be sure that another path would not lead to a better result. According to the results there is no obvious rule to choose an optimized sequence without testing all possible patterns for a new summand. However, once such a sequence is found it leads to improved results compared to non-overlapping sequences consisting of only a single pattern as shown in Figure 10. Here, we compare the optimized sequence with the three sequences defined in Section VI-B. Unfortunately, such an optimization is useless without a predictor for the next pattern in a sequence. Probing patterns is computationally too expensive for large problems.

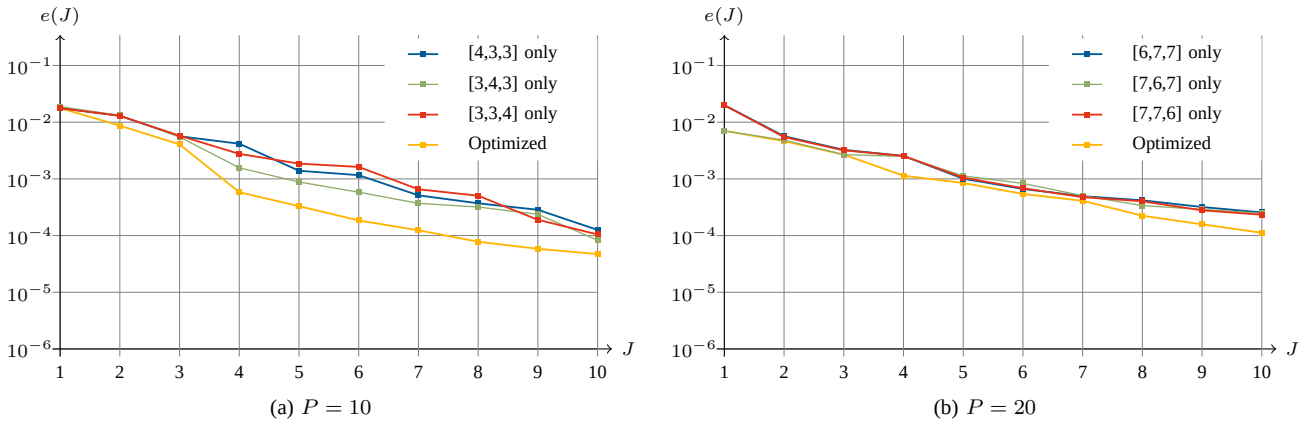


(a) $P = 10$



(b) $P = 20$

Fig. 8: Finding an optimized sequence of overlapping slice patterns using a greedy approach.



(a) $P = 10$ (b) $P = 20$

Fig. 9: Optimized slice pattern with overlaps (yellow) vs. non-overlapping slices.

D. Large problem

Finally, Figure 9 presents first results for a large problem ($P = 100$). We used the following three slice sequences:

- Sequence A: [20,20,20,20,20] only
- Sequence B: [20,20,20,20,20][20,20,20,20,20][10,10,10,10,10,10,10,10,10,10][10,10,10,10,10,10,10,10,10,10] repeated
- Sequence C: [10,10,10,10,10,10,10,10,10,10] only

It is important to note that for the sequences A and B a dynamic break criterion was active which skipped further iterations if the relative delta between the i-th and $(i-1)$-th iteration was smaller than 10^{-7}. As mentioned in Section VI-A it is likely that convergence for these two sequences could have been better using a fixed number of iterations. For sequence C we used ten iterations per new summand. Again, it is not surprising that larger slices lead to better results. However, the slope of the lines is interesting since the improvement per new summand seems to be independent of the slice size. This is probably not the case but a result of the break criterion. In fact, sequence C would not converge at all if the criterion was used here. This can be seen at the behavior of sequence B when looking at the $2 \leq J \leq 4$ and $6 \leq J \leq 8$ where the error remains almost constant. In both cases, the new summands use the tenner pattern. However, even the 20s pattern (sequence A) suffers from the break criterion as can be seen for $4 \leq J \leq 6$. Additionally one should keep in mind that calculating sequence C takes less than one hour while sequence A needs several days. This demonstrates that the current numerical break condition is useless.

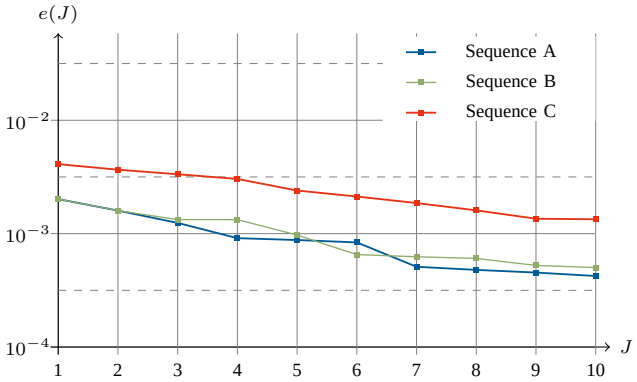


Fig. 10: Results for $P = 100$ using different slice sequences.

VII. Performance results

This Section briefly summarizes the performance of the VTA algorithm. We used two different systems for evaluation:

1) Dual Intel Xeon X5355 (2.66GHz, 8 cores, codename Clovertown)
   Intel compiler v10.1.011
2) Dual Intel Xeon L5520 (2.26GHz, 8 cores, codename Nehalem)
   Intel compiler v11.1.064

Both systems were running 64 Bit Linux. The VTA test application was compiled using the `-O3` and `-ipo` flags for optimization. The time was measured using a simple wall-clock approach. Table I shows results for different problem sizes, slice patterns and thread counts. The speedup S_n using n threads this defined as

$$S_n = \frac{T_1}{T_n} \tag{40}$$

where T_1 denotes the time needed for the sequential algorithm and T_n denotes the time needed by the parallel algorithm using n threads. Since we are using the sequential eigenvalue solver ARPACK++ for subproblems we cannot achieve linear speedup. In fact, parallelization at this time is limited to the custom matrix-vector products between Hamiltonians and Kronecker vectors using OpenMP. These are also used by ARPACK++. Consequently, there is a fraction p of the code which can be run in parallel while the rest of the program is sequential code. Amdahl's law gives an upper bound $S_{n,\max}$ for the speedup depending on p when using n threads:

$$S_{n,\max} = \frac{1}{(1-p) + \frac{p}{n}} \tag{41}$$

Finding a good estimate for p is difficult. One possibility is to use the measured speedup for 2 threads for an extrapolation of the possible speedup for n threads. However, the speedup for two threads already contains overhead induced by parallelization.

P	**Pattern**	**Threads**	**Time**	**Speedup**
30	[10,10,10]	1	0m52.387s	1.00
30	[10,10,10]	2	0m36.207s	1.44
30	[10,10,10]	4	0m25.905s	2.02
30	[10,10,10]	8	0m19.965s	2.62
60	[15,15,15,15]	1	101m53.073s	1.00
60	[15,15,15,15]	2	63m41.364s	1.60
60	[15,15,15,15]	4	45m31.483s	2.34
60	[15,15,15,15]	8	33m37.094s	3.03

(a) Dual Intel Xeon X5355 (2.66GHz)

P	**Pattern**	**Threads**	**Time**	**Speedup**
30	[10,10,10]	1	0m46.431s	1.00
30	[10,10,10]	2	0m29.022s	1.60
30	[10,10,10]	4	0m20.413s	2.27
30	[10,10,10]	8	0m16.006s	2.90
60	[15,15,15,15]	1	83m30.088s	1.00
60	[15,15,15,15]	2	50m45.998s	1.64
60	[15,15,15,15]	4	34m54.362s	2.39
60	[15,15,15,15]	8	26m15.854s	3.18

(b) Dual Intel Xeon L5520 (2.26GHz)

TABLE I: Computation time and speedup with multiple threads for different problems.

For this reason one cannot expect that extrapolation gives an upper bound. Instead, we used VTune [2] to determine p. Although p depends on the slice size we found that for $10 \leq P \leq 17$ a fraction $p \approx 0.85$ is spent in parallelized routines and their subroutines. Using this as an estimate we obtain $S_{8,\max} \approx 3.90$ as an upper bound for the speedup using Amdahl's law. For comparison, we currently obtain a maximum speedup of 3.03 respectively 3.18 on the test platforms.

VIII. Conclusion

In this work we presented a general approach to approximate the eigenvalue of arbitrary hermitian matrices of dimension $2^P \times 2^P$. The key of our approach is a decomposition of the large eigenvalue problem into a sequence of smaller ones which can be solved by approved algorithms. The most important criterion for the quality of the approximation is the size of the slices the eigenvector consists of. This directly corresponds to the dimension of the subproblems and conflicts with the effort to decompose the large problem. In general, larger slices should be preferred as long as the computational complexity remains affordable. For todays workstations, slices of size $10 \leq P \leq 15$ proved to be a good compromise between improvement in precision per step and processing time. The biggest problem of our approach is the lack of a reliable numerical break condition. As shown in Section VI-A, small changes between iterations are not a reliable indicator that the current approximation is near an optimal value.

For this reason, finding a more suitable break condition is part of future work. Another promising improvement are periodic boundaries of Kronecker vectors [7] which would allow overlaps between identical slice patterns. This way, overlapping at the beginning and end of vectors become possible which might increase the benefit of overlapping.

APPENDIX A
IMPLEMENTATIONAL OVERVIEW

A top level class diagram of the VTA test application is shown by Figure 11. The central class is the HamiltonianEVSolver which puts all parts together and implements the algorithm as outlined in Section IV. It relies on an instance of class Hamiltonian and a list of Kronecker vectors which represent the current eigenvector approximation. The layout of the Kronecker vectors is defined by an instance of class SlicePattern which is passed to the solver. It is also responsible to reduce Equation 20 and Equation 25 to eigenvalue problems by calculating the necessary values. These are used to create instances of the classes EffectiveHamiltonian and ExtendedHamiltonion repsectively which in turn offer compatible interfaces for matrix-vector products as needed by ARPACK++. Afterwards, the instance of HamiltonianEVSolver can create an instance of ARComStdEig which defines a complex eigenvalue problem for ARPACK++. Passing the appropriate instance of EffectiveHamiltonian or ExtendedHamiltonion to this problem makes it possible for ARPACK++ to use our custom BLAS routines. These do not belong to a specific class. Instead, all operands required are passed to the BLAS routines. The data elements are accessed through inlined routines to keep object orientation alive.

For more details regarding the implementation please refer to the accompanying Doxygen documentation and the detailed comments contained in the sources.

APPENDIX B
INSTALLATION AND USAGE INSTRUCTION FOR THE VTA TEST APPLICATION

Compilation

1) Obtain the tarball `vta.tar.gz` and decompress the archive:
   `tar -xzvf vta.tar.gz`
2) Change into the subfolder `vta/build`. The build process is based on CMake which must be available on your system. Out of the box, the default compiler of your system will be used (probably the GNU C++). Alternatively, you can specify the Intel optimization compiler. Create the build files:
   `cmake ..` or `CXX=icpc cmake ..`
3) Build the application:
   `make`

Afterwards, the binary `vta` should be located in the build directory.

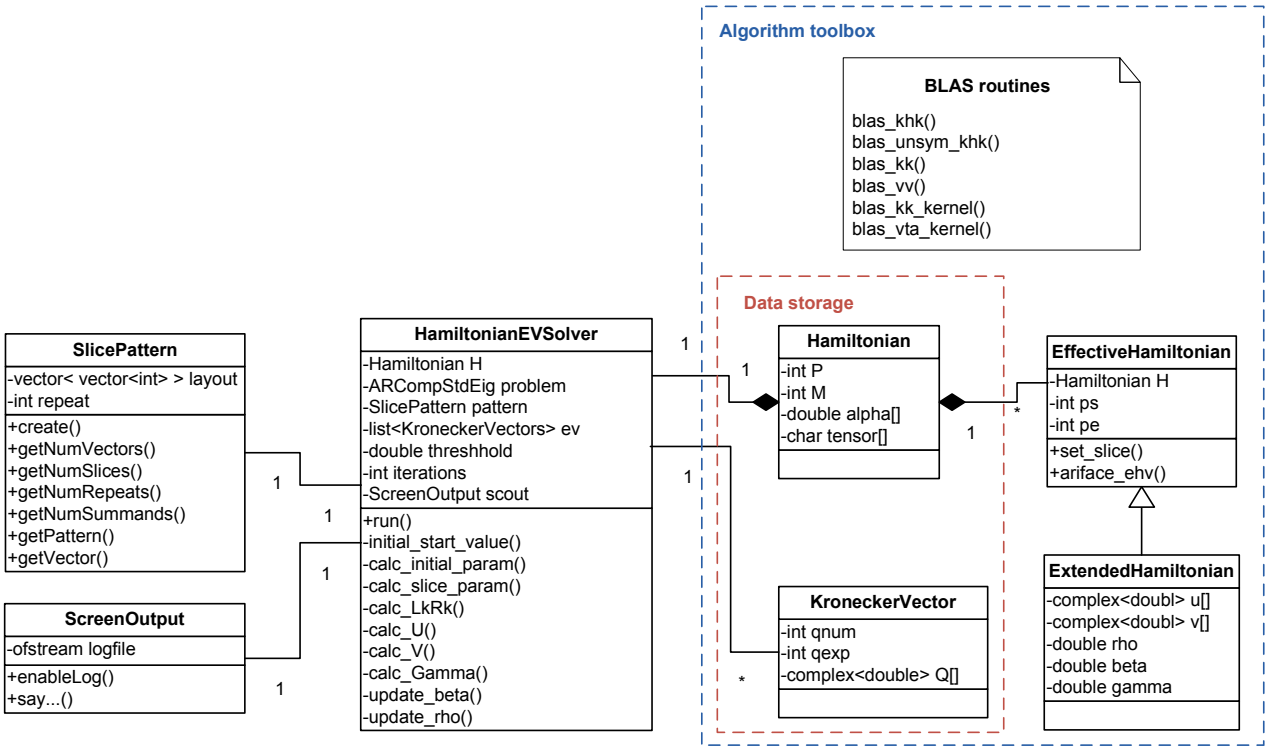


Fig. 11: Class diagram of the VTA test application

Command line reference

The current test application fixes the coefficients α_m to one. The Hamiltonian is based on the 1D-Ising model. The following mandatory command line arguments must be specified:

- `-p [[pattern 1][pattern 2]...]`, `--pattern [[pattern 1][pattern 2]...]`
  Specifies the slice pattern for the Kronecker vectors, e.g. $[[5, 5, 5, 5][10, 10]]$. The J-th sequence within the outer brackets specifies the layout of the J-th summand.
- `-r [value]`, `--repeat [value]`
  Determines how often the given pattern should be repeated. Setting this value to one means that the supplied pattern is used exactly once.
- `-i [value]`, `--iterations [value]`
  Controls the maximum number of iterations for each new summand. The test application uses a built-in numerical break condition which skips further iterations if the relative change between two subsequent iterations is less than 10^{-7}. See option `-t` to control or disable this break condition.

Additionally, there is a number of optional arguments:

- `-t [value]`, `--threshhold [value]`
  Allows to control the numerical break condition which enables the test application to use less iterations per summand than specified. The value supplied will be used as new threshold instead of 10^{-7} as set by default. Setting this value to -1 effectively disables the break condition.
- `-l`, `--log`
  If this switch is specified, the test application writes a log file named according to the current date and time. The log file contains a copy of the supplied pattern sequence. Afterwards, the following values are saved as comma separated values: `J,i,ev,abserr`
  Here, J denotes the number of summands currently in use and i the iteration of the J-th summand followed by the current approximation of the eigenvalue and the absolute improvement of the last iteration.

REFERENCES

[1] Himmelstoß, Thomas. Eigenwertprobleme bei Matrizen in Tensorprodukt-Form, 2010.
[2] Intel Corporation. Technologies for Measuring Software Performance, 2003.
[3] Nobuya Maeshima, Yasuhiro Hieida, Tomotoshi Nishino, and Kouichi Okunishi. Matrix product state approximation for the maximum-eigenvalue eigenstate of the quantum transfer matrix. *Progress of Theoretical Physics*, (145):204–207, 2002.
[4] Zdzislaw Meglicki. Quantum computing without magic: Devices (scientific and engineering computation). 2008.
[5] Mikio Nakahara and Tetsuo Ohmi. Quantum computing: From linear algebra to physical realizations. 2008.
[6] Konrad Waldherr. Computation of ground states - a mathematical point of view, October 2009.
[7] Konrad Waldherr. Numerical (Multi-)Linear Algebra: Tensors and Applications, 2010.