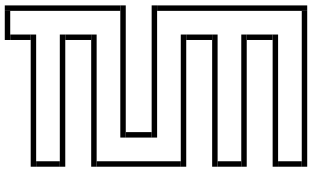


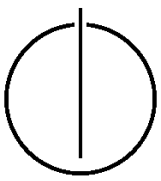
Computational Science and Engineering
(Int. Master's Program)

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis

# Numerical linear algebra problems in tensor formats


| | |
|---|---|
| Author: | Alfredo Parra Hinojosa |
| 1st examiner: | Univ.-Prof. Dr. Thomas Huckle |
| 2nd examiner: | Univ.-Prof. Dr. Michael Bader |
| Assistant advisor: | Dipl.-Math. Konrad Waldherr |
| Thesis handed in on: | August 29, 2013 |

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

August 29, 2013 Alfredo Parra Hinojosa

## Acknowledgments

I would like to express my gratitude to Univ.-Prof. Dr. Thomas Huckle and Univ.-Prof. Dr. Michael Bader for acting as examiners for this thesis. Prof. Huckle's constant support and valuable discussions made the realization of this work possible, adding always the enthusiasm and humor that make such a task easier to carry out.

A very special acknowledgment goes to Dipl.-Math. Konrad Waldherr, whose door was always open to me and whose acute understanding of the concepts treated in this thesis was invaluable to me.

I also appreciate the valuable suggestions made by Dipl.-Math. Benjamin Uekermann to improve the quality of this work.

For the financial support of my Master's studies I deeply thank the Consejo de Ciencia y Tecnología del Estado de Puebla (CONCYTEP).

Above all, I thank my parents and sisters for the kind of support that cannot be put into words. Studying low-rank tensor representations is not half as hard as being far away from them.

# Contents

# 1 Introduction

High dimensional problems arise naturally in the mathematical description of very diverse phenomena, since the dimensionality may depend simply on the number of parameters required to specify the solution of the problem. A common setting is a boundary value problem given in terms of a partial differential equation. As we will see throughout this work, looking for a *low-rank tensor* solution is the natural approach to attack such problems.

As a motivating example (found in [19]) consider the following two-dimensional Poisson equation with parametric boundary conditions:

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

This problem involves two physical dimensions $[x_1, x_2]$ plus four parametric dimensions $\alpha_i$, $i = 1, \ldots, 4$ (six dimensions in total). The authors used a discretization of $2^8 = 256$ points in each of the six dimensions, giving a total of $2^{48}$ discretization points ($\approx 2.8 \times 10^{14}$). After discretizing the problem with finite differences, the authors obtained a linear system of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, which they expressed using a tensor train (TT) representation. This problem was then solved in $\approx 3 \times 10^2$ seconds with the solution having a numerical error of $10^{-6}$, *sequentially and with a standard MATLAB distribution*. Another example described in the same paper involves a 64-dimensional reaction diffusion equation, solved in 1845 seconds with an accuracy of $10^{-5}$, for which a full matrix-vector representation would require $10^{154}$ data points. One final example considers a low-rank tensor variant of the preconditioned conjugate gradient (PCG) algorithm to obtain the solution of about $10^{18}$ linear systems of order 3644, with an accuracy of $10^{-3}$, in only 61 minutes [17]. This problem would take, according to the authors, approximately $3 \times 10^8$ CPU years if solved using full matrices and vectors.

This initially seems implausible, but we will see how to formulate and solve such problems using the tensor formalism described in Chapter 2. Clearly, the so called *curse of dimensionality* has been overcome in these examples, and we provide a set of theoretical tools that will aim clarify how this can be achieved. Given the good accuracy with which such problems can be solved serially with a standard personal computer, there should necessarily be a (very clever) way of storing such large vectors and matrices. This thesis will focus on one such strategy, namely, the *matrix product state* (MPS) formalism.

The application areas mentioned there cover a wide range of topics, from the solution of high-dimensional Schrödinger equations and stochastic PDEs, to computational finance and machine learning. [10] provides an updated and thorough survey of the existing literature on tensor representations, applications and algorithms. A detailed description of the MPS formalism and the closely related density matrix renormalization group (DMRG) algorithm (which we also present in this thesis) can be found in [29], and we refer to [12, 32] for comprehensive discussions on the efficient numerical calculations in MPS format.

Despite the many advantages of using tensor representations of vectors and matrices, several complications inherent to the tensor formalism require special techniques to ensure that the algorithms provide accurate numerical solutions. Such drawbacks may cause the algorithms to become too expensive or even fail, and we will study some cases where this happens. Nonetheless, the fact that some of the algorithms (such as DMRG for linear systems) can be used as black-box solvers makes tensor representations attractive and powerful, especially for problems that otherwise would not be possible to deal with due to memory or time restrictions in higher dimensions. This also means that no additional information about the problem geometry is required, which offers an advantage over other techniques where high-dimensional functions are involved, such as sparse grids [2].

The thesis is structured as follows: In Chapter 2 we present an overview of the MPS formalism used throughout this thesis. Chapter 3 describes an adaptation of the DMRG algorithm to the solution linear systems of equations in MPS format. Chapter 4 follows a very similar idea to Chapter 3 but with the goal of finding an approximate inverse matrix $\boldsymbol{M}$ of a given matrix $\boldsymbol{A}$, both written in the tensor format we have chosen. Although the problems discussed in Chapters 3 and 4 have been recently presented in other works (with slight variations - see the respective chapters), our treatment is necessary for the understanding of the rest of the chapters, and it will allow the reader to understand how to attack problems such as the example above. In Chapter 5 we discuss some new ideas concerning relaxation schemes and preconditioning in MPS format. The last two chapters deal with two well-known iterative schemes for solving linear systems numerically: a restarted Lanczos algorithm, and a full multigrid (FMG) scheme. We then propose to translate these two algorithms into MPS format to study their convergence properties and compare them to the existing algorithms. At the end of each chapter, we present the numerical results obtained for each of the algorithms. A MATLAB tensor toolbox was developed based on the existing implementations by K. Waldherr and can be obtained under request, including the algorithms described in this thesis.[1]

[1] All simulations were run on a MATLAB (R2012a) distribution installed on a 32-bit Linux system (under Ubuntu 12.04), with a 2.00 GHz Intel Pentium Dual Core processor and 2.9 GiB of RAM.

# 2 Preliminary theory

We start by describing the mathematical tools needed throughout this work, especially the notion of representing vectors as matrix product states (MPS) and its generalization to operators as matrix product operators (MPO).

## 2.1 A brief overview of Matrix Product States

Consider the following vector $\boldsymbol{x}$ with $2^7 = 128$ arbitrary complex entries (i.e., $\boldsymbol{x} \in \mathbb{C}^{2^7}$) and its expansion in the canonical base:

$$\begin{aligned}\boldsymbol{x} &= \begin{pmatrix} x_0 & x_1 & \dots & x_{126} & x_{127} \end{pmatrix}^\intercal \\ &= x_0\boldsymbol{e_0} + x_1\boldsymbol{e_1} + \ldots + x_{126}\boldsymbol{e_{126}} + x_{127}\boldsymbol{e_{127}} \\ &= \sum_{i=0}^{127} x_i \boldsymbol{e_i},\end{aligned}$$

where $\boldsymbol{e_i}$ is the $i$-th column of the $128\times128$ identity matrix. This is clearly a one-dimensional vector. Now, we will take the index $i = \{0, \ldots, 127\}$ and write it in a different base, say, binary, such that $i \equiv (i_1 i_2 i_3 i_4 i_5 i_6 i_7)_2$, where each $i_d = \{0, 1\}$. So, for example, $i = 35_{10} = 0100011_2$. Using this notation, the vector above can be written as

$$\boldsymbol{x} = \sum_{i_1,\ldots,i_7=0}^{1} x_{i_1 i_2 i_3 i_4 i_5 i_6 i_7} \boldsymbol{e}_{i_1 i_2 i_3 i_4 i_5 i_6 i_7}.$$

Additionally, one can see that the unit vectors $\boldsymbol{e_{i_1 i_2 i_3 i_4 i_5 i_6 i_7}}$ can be rewritten as follows:

$$\begin{aligned}\boldsymbol{e_{i_1 i_2 i_3 i_4 i_5 i_6 i_7}} &= \boldsymbol{e_{i_1}} \otimes \boldsymbol{e_{i_2}} \otimes \boldsymbol{e_{i_3}} \otimes \boldsymbol{e_{i_4}} \otimes \boldsymbol{e_{i_5}} \otimes \boldsymbol{e_{i_6}} \otimes \boldsymbol{e_{i_7}} \\ &\equiv \bigotimes_{d=1}^{7} \boldsymbol{e_{i_d}},\end{aligned}$$

where each of the $\boldsymbol{e_{i_d}}$ is a unit vector with two elements, $\boldsymbol{e_0} = (1, 0)^\intercal$, $\boldsymbol{e_1} = (0, 1)^\intercal$. In the above expression we have introduced the *Kronecker product* denoted by the symbol $\otimes$ (also called tensor product). The Kronecker product of two matrices $\boldsymbol{A} \otimes \boldsymbol{B}$, with $\boldsymbol{A} \in \mathbb{C}^{m\times n}$, $\boldsymbol{B} \in \mathbb{C}^{p\times q}$ is another matrix $\boldsymbol{C} \in \mathbb{C}^{mp\times nq}$ obtained by multiplying each entry of $\boldsymbol{A}$ by the matrix $\boldsymbol{B}$. For example, if both $\boldsymbol{A}$ and $\boldsymbol{B}$ are $2 \times 2$ matrices, then

$$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} a_{0,0}\boldsymbol{B} & a_{0,1}\boldsymbol{B} \\ a_{1,0}\boldsymbol{B} & a_{1,1}\boldsymbol{B} \end{pmatrix} = \begin{pmatrix} a_{0,0}b_{0,0} & a_{0,0}b_{0,1} & a_{0,1}b_{0,0} & a_{0,1}b_{0,1} \\ a_{0,0}b_{1,0} & a_{0,0}b_{1,1} & a_{0,1}b_{1,0} & a_{0,1}b_{1,1} \\ a_{1,0}b_{0,0} & a_{1,0}b_{0,1} & a_{1,1}b_{0,0} & a_{1,1}b_{0,1} \\ a_{1,0}b_{1,0} & a_{1,0}b_{1,1} & a_{1,1}b_{1,0} & a_{1,1}b_{1,1} \end{pmatrix}. \tag{2.1}$$

Before we continue our discussion, we briefly recall the definition of a *tensor* of order $N$ as a multidimensional array of size $i_1 \times i_2 \times \cdots \times i_N$ for integers $i_1, i_2, \ldots, i_N$, denoted by

$$\boldsymbol{X} \in \mathbb{C}^{i_1 \times i_2 \times \cdots \times i_N}. \tag{2.2}$$

The fact that the components of $\boldsymbol{x}$ are now characterized by the set of indices $\{i_1, \ldots, i_7\}$ (instead of only one large index $i$) allows us to interpret $\boldsymbol{x}$ as a 7-dimensional tensor of sizes $2 \times \cdots \times 2$, i.e., $\boldsymbol{x} \in \mathbb{C}^{i_1 \times \ldots \times i_7}$. Of course, this is nothing else than a reshaping operation and thus not much has changed. What we're interested in are vectors whose components $x_{i_1 \ldots i_7}$ can be written as follows:

$$x_i = x_{i_1 \ldots i_7} = \mathrm{Tr}\left(\boldsymbol{X}_{\boldsymbol{1}}^{(\boldsymbol{i_1})} \boldsymbol{X}_{\boldsymbol{2}}^{(\boldsymbol{i_2})} \boldsymbol{X}_{\boldsymbol{3}}^{(\boldsymbol{i_3})} \boldsymbol{X}_{\boldsymbol{4}}^{(\boldsymbol{i_4})} \boldsymbol{X}_{\boldsymbol{5}}^{(\boldsymbol{i_5})} \boldsymbol{X}_{\boldsymbol{6}}^{(\boldsymbol{i_6})} \boldsymbol{X}_{\boldsymbol{7}}^{(\boldsymbol{i_7})}\right), \tag{2.3}$$

where $\boldsymbol{X}_{\boldsymbol{d}}^{\boldsymbol{i_d}}$ denotes a pair of matrices $\boldsymbol{X}_{\boldsymbol{d}}^{\boldsymbol{i_d}} = \{\boldsymbol{X}_{\boldsymbol{d}}^{(\boldsymbol{0})}, \boldsymbol{X}_{\boldsymbol{d}}^{(\boldsymbol{1})}\}$, each with size $D_d \times D_{d+1}$. We will soon see why we look for such a decomposition.

Alternatively, if it happens that the first pair of matrices $\boldsymbol{X}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}$ are row vectors (matrices of size $1 \times D_2$) and the last matrices $\boldsymbol{X}_{\boldsymbol{7}}^{(\boldsymbol{i_7})}$ are column vectors (of size $D_7 \times 1$), the product of the seven matrix gives a scalar and trace becomes redundant, so the expression above reduces to

$$x_i = x_{i_1 \ldots i_7} = \boldsymbol{X}_{\boldsymbol{1}}^{(\boldsymbol{i_1})} \boldsymbol{X}_{\boldsymbol{2}}^{(\boldsymbol{i_2})} \boldsymbol{X}_{\boldsymbol{3}}^{(\boldsymbol{i_3})} \boldsymbol{X}_{\boldsymbol{4}}^{(\boldsymbol{i_4})} \boldsymbol{X}_{\boldsymbol{5}}^{(\boldsymbol{i_5})} \boldsymbol{X}_{\boldsymbol{6}}^{(\boldsymbol{i_6})} \boldsymbol{X}_{\boldsymbol{7}}^{(\boldsymbol{i_7})}. \tag{2.4}$$

So, for instance, supposing that such a decomposition is possible, the component corresponding to $i = 35$ would be obtained as

$$x_{35} = x_{0100011} = \boldsymbol{X}_{\boldsymbol{1}}^{(\boldsymbol{0})} \boldsymbol{X}_{\boldsymbol{2}}^{(\boldsymbol{1})} \boldsymbol{X}_{\boldsymbol{3}}^{(\boldsymbol{0})} \boldsymbol{X}_{\boldsymbol{4}}^{(\boldsymbol{0})} \boldsymbol{X}_{\boldsymbol{5}}^{(\boldsymbol{0})} \boldsymbol{X}_{\boldsymbol{6}}^{(\boldsymbol{1})} \boldsymbol{X}_{\boldsymbol{7}}^{(\boldsymbol{1})}. \tag{2.5}$$

Put into words, one carries out a matrix product in order to access an element of the vector $\boldsymbol{x}$, which means that all the information of the vector entries must be somehow encoded in the matrices $\boldsymbol{X}_{\boldsymbol{d}}^{(\boldsymbol{i_d})}$. This decomposition is somewhat analogous to that of the separation of variables of a multivariable function (e.g., $f(x_1, x_2) \equiv G_1(x_1)G_2(x_2)$. For a related discussion, see [21]), and several questions arise, the two immediate ones being: under which conditions is such a decomposition possible? and, why are we interested in such a decomposition? The answers will become clear by the end of this chapter.

Let us summarize our results by defining a *matrix product state* (MPS) as any vector $\boldsymbol{x} \in \mathbb{C}^{2^N}$ of the form

$$\begin{aligned} \boldsymbol{x} &= \sum_{i_1, \ldots, i_N = 0}^{p-1} \mathrm{Tr}\left(\boldsymbol{X}_{\boldsymbol{1}}^{(\boldsymbol{i_1})} \boldsymbol{X}_{\boldsymbol{2}}^{(\boldsymbol{i_2})} \cdots \boldsymbol{X}_{\boldsymbol{N-1}}^{(\boldsymbol{i_{N-1}})} \boldsymbol{X}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right) \bigotimes_{n=1}^{N} e_{i_n} \\ &= \sum_{i_1, \ldots, i_N = 0}^{p-1} \mathrm{Tr}\left(\prod_{d=1}^{N} \boldsymbol{X}_{\boldsymbol{d}}^{(\boldsymbol{i_d})}\right) \bigotimes_{n=1}^{N} e_{i_n}. \end{aligned} \tag{2.6}$$

The above definition is more general than our binary representation, since this definition allows the indices $i_d$ to go up to $p \geq 2$ (in our example, $p = 2$). Using the definition for the

trace of a product of matrices, we can obtain an alternative expression for 2.6 as follows:

$$\begin{aligned}\boldsymbol{x} &= \sum_{q_1=1}^{D_1} \cdots \sum_{q_N=1}^{D_N} \left( \sum_{i_1=0}^{p-1} x^{(i_1)}_{1;q_1,q_2} e_{i_1} \right) \otimes \cdots \otimes \left( \sum_{i_N=0}^{p-1} x^{(i_N)}_{N;q_N,q_1} e_{i_N} \right) \\ &= \sum_{q_1,\ldots,q_N} \boldsymbol{x_{1;q_1,q_2}} \otimes \boldsymbol{x_{2;q_2,q_3}} \otimes \cdots \otimes \boldsymbol{x_{N-1;q_{N-1},q_N}} \otimes \boldsymbol{x_{N;q_N,q_1}} \\ &= \sum_{q_1,\ldots,q_N} \bigotimes_{d=1}^{N} \boldsymbol{x_{d;q_d,q_{d+1}}}, \end{aligned} \tag{2.7}$$

where each $\boldsymbol{x_{d;q_d,q_{d+1}}}$ is a vector of size $p$, namely $\left(x^{(0)}_{d;q_d,q_{d+1}}, \ldots, x^{(p-1)}_{d;q_d,q_{d+1}}\right)^\intercal$. For clarity, we will often omit the upper limits of the sums and group them together in one sum, as done above. Eq. (2.7) tells us that we are decomposing our vector $\boldsymbol{x}$ as a sum of different vectors, each of them built as tensor products of size $p$ vectors. Also, we will call the largest size $D = \max_d D_d$ the *bond dimension*. We will then say $\boldsymbol{x} \in \mathrm{MPS}_D$ to denote that our vector $\boldsymbol{x}$ is written in MPS format with bond dimension $D$. Notice from Eq. (2.7) (second line) that the index $q_1$ appears in the first and last vectors $\boldsymbol{x_d}$, due to the action of the trace, and thus (2.6) is said to be an MPS with *periodic boundary conditions* (PBC-MPS). In the study of quantum spin systems, this fact has a straightforward physical interpretation [30]. Consequently, if $D_1 = 1$ (as in (2.4)), the MPS is said to have *open boundary conditions* (OBC-MPS), and takes the form

$$\begin{aligned}\boldsymbol{x} &= \sum_{i_1,\ldots,i_N=0}^{p-1} \prod_{d=1}^{N} \boldsymbol{X_d^{(i_d)}} \bigotimes_{n=1}^{N} e_{i_n} \\ &= \sum_{q_2,\ldots,q_N} \boldsymbol{x_{1;q_2}} \otimes \boldsymbol{x_{2;q_2,q_3}} \otimes \cdots \otimes \boldsymbol{x_{N-1;q_{N-1},q_N}} \otimes \boldsymbol{x_{N;q_N}}, \end{aligned} \tag{2.8}$$

The definition above is also called tensor train format (TT-format), introduced in [22, 20]. The case where $i_d$ is binary is also referred to as quantized tensor train (QTT) decomposition.

### 2.1.1 Matrix Product Operators

Extending the concept of MPS to represent a matrix $\boldsymbol{A} \in \mathbb{C}^{p^N \times p^N}$ is straightforward. We use the reshape $A_{i,j} \to A_{i_1 \ldots i_N, j_1 \ldots j_N}$ and through the same reasoning, we obtain

$$\begin{aligned}\boldsymbol{A} &= \sum_{\substack{i_1,\ldots,i_N=0 \\ j_1,\ldots,j_N=0}}^{p-1} \mathrm{Tr}\left( \boldsymbol{W_1^{(i_1,j_1)}} \cdots \boldsymbol{W_N^{(i_N,j_N)}} \right) \bigotimes_{n=1}^{N} e_{i_n} e_{j_n}^\intercal \\ &= \sum_{q_1,\ldots,q_N} \left( \sum_{i_1,j_1} w^{(i_1,j_1)}_{1;q_1,q_2} e_{i_1} e_{j_1}^\intercal \right) \otimes \cdots \otimes \left( \sum_{i_N,j_N} w^{(i_N,j_N)}_{N;q_N,q_1} e_{i_N} e_{j_N}^\intercal \right) \\ &= \sum_{q_1,\ldots,q_N} \boldsymbol{w_{1;q_1,q_2}} \otimes \cdots \otimes \boldsymbol{w_{N;q_N,q_1}}, \end{aligned} \tag{2.9}$$

where each $\boldsymbol{w_{d;q_d,q_{d+1}}}$ is a $p \times p$ matrix. For $p = 2$ they are given by

$$\boldsymbol{w_{d;q_d,q_{d+1}}} = \begin{pmatrix} w^{(0,0)}_{d;q_d,q_{d+1}} & w^{(0,1)}_{d;q_d,q_{d+1}} \\ w^{(1,0)}_{d;q_d,q_{d+1}} & w^{(1,1)}_{d;q_d,q_{d+1}} \end{pmatrix}. \tag{2.10}$$

Definition 2.9 is given in its PBC form (PBC-MPO), but the OBC representation is analogous to MPS (setting $q_1 = 1$ so that the trace disappears). As an example of a matrix written in MPO format, consider the 1D Laplace operator of size $2^N \times 2^N$ with Dirichlet boundary conditions

$$\Delta_1^{(N)} = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}. \tag{2.11}$$

The QTT representation of this operator, its generalizations to more dimensions, and their inverse, are presented in [14]. See also [15] for a more general treatment. To obtain this representation, we define

$$\boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \boldsymbol{J} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \tag{2.12}$$

and for $N \geq 2$, it holds that

$$\Delta_1^{(N)} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{J}^\intercal & \boldsymbol{J} \end{bmatrix} \bowtie \begin{bmatrix} \boldsymbol{I} & \boldsymbol{J}^\intercal & \boldsymbol{J} \\ & \boldsymbol{J} & \\ & & \boldsymbol{J}^\intercal \end{bmatrix}^{\bowtie(N-2)} \bowtie \begin{bmatrix} 2\boldsymbol{I} - \boldsymbol{J} - \boldsymbol{J}^\intercal \\ -\boldsymbol{J} \\ -\boldsymbol{J}^\intercal \end{bmatrix}, \tag{2.13}$$

where the symbol $\bowtie$ denotes an *inner core product*. Its action on block matrices is defined through

$$\begin{bmatrix} \boldsymbol{A_1} & \boldsymbol{A_2} \\ \boldsymbol{A_3} & \boldsymbol{A_4} \end{bmatrix} \bowtie \begin{bmatrix} \boldsymbol{B_1} & \boldsymbol{B_2} \\ \boldsymbol{B_3} & \boldsymbol{B_4} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A_1} \otimes \boldsymbol{B_1} + \boldsymbol{A_2} \otimes \boldsymbol{B_3} & \boldsymbol{A_1} \otimes \boldsymbol{B_2} + \boldsymbol{A_2} \otimes \boldsymbol{B_4} \\ \boldsymbol{A_3} \otimes \boldsymbol{B_1} + \boldsymbol{A_4} \otimes \boldsymbol{B_3} & \boldsymbol{A_3} \otimes \boldsymbol{B_2} + \boldsymbol{A_4} \otimes \boldsymbol{B_4} \end{bmatrix}, \tag{2.14}$$

which looks like a regular matrix product but replacing the multiplications with Kronecker products. How is (2.13) related to our MPO representation (2.9)? Well, OBC-MPO may be rewritten as

$$\boldsymbol{A} = \boldsymbol{Z_1} \bowtie \boldsymbol{Z_2} \bowtie \cdots \bowtie \boldsymbol{Z_{N-1}} \bowtie \boldsymbol{Z_N}. \tag{2.15}$$

The way to relate the core matrices $\boldsymbol{Z_d}$ with the MPO matrices $\boldsymbol{W_d^{(i_d,j_d)}}$ is as follows:

$$\boldsymbol{Z_d} = \left(\begin{array}{c|c} \boldsymbol{A_1} & \boldsymbol{A_2} \\ \hline \boldsymbol{A_3} & \boldsymbol{A_4} \end{array}\right) = \left(\begin{array}{cc|cc} a_{1;0,0} & a_{1;0,1} & a_{2;1,0} & a_{2;0,1} \\ a_{1;1,0} & a_{1;1,1} & a_{2;1,0} & a_{2;1,1} \\ \hline a_{3;0,0} & a_{3;0,1} & a_{4;0,0} & a_{4;0,1} \\ a_{3;1,0} & a_{3;1,1} & a_{4;1,0} & a_{4;1,1} \end{array}\right) \tag{2.16}$$

$$\implies \boldsymbol{W_d^{(0,0)}} = \begin{pmatrix} a_{1;0,0} & a_{2;0,0} \\ a_{3;0,0} & a_{4;0,0} \end{pmatrix}, \quad \boldsymbol{W_d^{(0,1)}} = \begin{pmatrix} a_{1;0,1} & a_{2;0,1} \\ a_{3;0,1} & a_{4;0,1} \end{pmatrix}, \tag{2.17}$$

$$\boldsymbol{W_d^{(1,0)}} = \begin{pmatrix} a_{1;1,0} & a_{2;1,0} \\ a_{3;1,0} & a_{4;1,0} \end{pmatrix}, \quad \boldsymbol{W_d^{(1,1)}} = \begin{pmatrix} a_{1;1,1} & a_{2;1,1} \\ a_{3;1,1} & a_{4;1,1} \end{pmatrix}. \tag{2.18}$$

As an example, take the 1D Laplacian (2.13) with $N = 2$ (size $4 \times 4$). The resulting MPO has the form

$$(\Delta^{(2)})_{i,j} = (\Delta^{(2)})_{i_1 i_2, j_1 j_2} = \boldsymbol{W_1^{(i_1,j_1)}} \boldsymbol{W_2^{(i_2,j_2)}},$$

with

$$\boldsymbol{W_1^{(0,0)}} = \begin{pmatrix} -1 & 0 & 0 \end{pmatrix} \quad \boldsymbol{W_1^{(0,1)}} = \begin{pmatrix} 0 & -1 & 0 \end{pmatrix} \quad \boldsymbol{W_2^{(0,0)}} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} \quad \boldsymbol{W_2^{(0,1)}} = \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}$$

$$\boldsymbol{W_1^{(1,0)}} = \begin{pmatrix} 0 & 0 & -1 \end{pmatrix} \quad \boldsymbol{W_1^{(1,1)}} = \begin{pmatrix} -1 & 0 & 0 \end{pmatrix} \quad \boldsymbol{W_2^{(1,0)}} = \begin{pmatrix} -1 \\ 0 \\ -1 \end{pmatrix} \quad \boldsymbol{W_2^{(1,1)}} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$$

For a thorough discussion of MPO representations, see [27].

Now we can start answering some questions. The first important point has to do with the amount of memory required to store an MPS or MPO. From Eq. (2.6), we observe that an MPS requires the storage $pN$ matrices, each of size $D_d \times D_{d+1}$, so the total storage is $\leq pND^2$. The amount of memory saved compared to the full vector of size $2^N$ depends of course on the bond dimension $D$. Therefore, one is interested in vectors and matrices that can be either expressed exactly or approximated well with small bond dimension $D$, usually of the order $10^1$ to $10^2$. The following theorem, presented in [26] and shown in [31], states the existence of an OBC-MPS representation for an arbitrary vector:

**Theorem 2.1 (Completeness and Canonical Form)** *Any state $\boldsymbol{x} \in \mathbb{C}^{p^N}$ has an OBC-MPS representation of the form Eq. (2.8) with bond dimension $D \leq p^{\lfloor N/2 \rfloor}$ and*

1. $\sum_{i_d} \boldsymbol{X_d^{(i_d)}} \boldsymbol{X_d^{(i_d)\mathrm{H}}} = \boldsymbol{I}_{D_d}$ *for all* $1 \leq d \leq N$,
2. $\sum_{i_d} \boldsymbol{X_d^{(i_d)\mathrm{H}}} \boldsymbol{\Lambda_{d-1}} \boldsymbol{X_d^{(i_d)}} = \boldsymbol{\Lambda_d}$, *for all* $1 \leq d \leq N$,
3. $\boldsymbol{\Lambda_0} = \boldsymbol{\Lambda_N} = 1$ *and each* $\boldsymbol{\Lambda_d}$ *is a* $D_{d+1} \times D_{d+1}$ *diagonal matrix which is positive, full rank and with* $\mathrm{Tr}\, \boldsymbol{\Lambda_d} = 1$.

If conditions 1-3 hold, the OBC-MPS is said to be in the canonical form, since the MPS decomposition is not unique. Of course, in the worst case $D = p^{\lfloor N/2 \rfloor}$ the storage required is $p^N$ and there is no reduction in the storage (we fall back to the full vector case).

So now we can go back to our vector $\boldsymbol{x}$ with $2^7$ entries, taking its OBC-MPS representation (2.4). According to Theorem 2.1, this vector has an OBC-MPS representation, and the question is how large the matrices $\boldsymbol{X_d^{(i_d)}}$ have to be. Suppose the five inner matrix pairs $\boldsymbol{X_2^{(i_2)}}, \ldots, \boldsymbol{X_6^{(i_6)}}$ are of equal size $3 \times 3$ (5 pairs $\rightarrow$ 10 matrices, each with 9 entries $\rightarrow$ 90 entries), $\boldsymbol{X_1^{(i_1)}}$ is a pair of $1 \times 3$ matrices (6 entries), and $\boldsymbol{X_7^{(i_7)}}$ is a pair of $3 \times 1$ entries (6 entries). One would need to store a total of 102 entries, already less than the 128 of the full vector. Assume such representation approximates our vector $\boldsymbol{x}$ exactly[1]. Now, we

[1]This may or may not be true, depending on the particular structure of $\boldsymbol{x}$. For example, a constant vector requires only $1 \times 1$ matrices. Further symmetries allow further reduction of degrees of freedom, see [13]

may ask one further questions: is it possible to further compress the MPS by finding an approximation $\boldsymbol{y}$ with components given by

$$y_i = y_{i_1 \dots i_7} = \boldsymbol{Y_1^{(i_1)} Y_2^{(i_2)} Y_3^{(i_3)} Y_4^{(i_4)} Y_5^{(i_5)} Y_6^{(i_6)} Y_7^{(i_7)}}, \tag{2.19}$$

where the matrices $\boldsymbol{Y_d^{(i_d)}}$ are of size $D'_k \times D'_{k+1}$, and bounded by $D'$, i.e., $D'_k \leq D'$, and $D' \leq D$?. In other words, we are looking for matrices $\boldsymbol{Y_d^{(i_d)}}$ of sizes smaller or equal to those of $\boldsymbol{X_d^{(i_d)}}$ to approximate $\boldsymbol{x}$ with precision $\epsilon$, a problem that can be written as finding $\boldsymbol{y}$ such that

$$\|\boldsymbol{x} - \boldsymbol{y}\|_F \leq \epsilon \|\boldsymbol{x}\|_F \tag{2.20}$$

This operation is usually referred to as *compression*, *rounding* or *truncation*. One possible algorithm to solve such problem is described in [20], with a resulting complexity of $\mathcal{O}(pND^3)$. That is, it scales linearly with the number of matrices $N$ and the index dimension $p$ (in our example, $p = 2$), and cubically with the maximum size of such matrices (the bond dimension $D$). Such complexity results are, as we will see throughout this work, the main advantage of these tensor decompositions: the resulting algorithms usually scale polynomially with the size of the component matrices $\boldsymbol{X_d^{(i_d)}}$ and not with the size of the full vector (which is exponential in $N$, since we are interested in vectors of size $2^N$ with large $N$). This allows $N$ to grow as long as the bond dimension $D$ remains moderate, thus overcoming the curse of dimensionality. Such algorithms exploit the MPS and MPO structure by iterating through the component matrices (we will see in Chapters 3 and 4 how this occurs), thus translating one large problem of size $2^N$ to $N$ small problems whose size depend on $D$.

### 2.1.2 Algebraic operations

We briefly define two basic operations in MPS/MPO format: addition of two MPS vectors, and the product MPO times MPS. As one would hope, adding two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ written in MPS format results in another MPS vector $\boldsymbol{z}$, with the particularity that the matrices of $\boldsymbol{z}$ are of larger size. Indeed, we have

$$\begin{aligned}
\boldsymbol{x} + \boldsymbol{y} &= \sum_{i_1,\dots,i_N} \operatorname{Tr}\left(\prod_{d=1}^{N} \boldsymbol{A_d^{(i_d)}}\right) \boldsymbol{e_{i_1,\dots,i_N}} + \sum_{i_1,\dots,i_N} \operatorname{Tr}\left(\prod_{d=1}^{N} \boldsymbol{B_d^{(i_d)}}\right) \boldsymbol{e_{i_1,\dots,i_N}} \\
&= \sum_{i_1,\dots,i_N} \operatorname{Tr}\left[\begin{pmatrix} \prod_{d=1}^{N} \boldsymbol{A_d^{(i_d)}} & \\ & \prod_{d=1}^{N} \boldsymbol{B_d^{(i_d)}} \end{pmatrix}\right] \boldsymbol{e_{i_1,\dots,i_N}} \\
&= \sum_{i_1,\dots,i_N} \operatorname{Tr}\left[\prod_{d=1}^{N} \begin{pmatrix} \boldsymbol{A_d^{(i_d)}} & \\ & \boldsymbol{B_d^{(i_d)}} \end{pmatrix}\right] \boldsymbol{e_{i_1,\dots,i_N}} \\
&= \sum_{i_1,\dots,i_N} \operatorname{Tr}\left(\prod_{d=1}^{N} \boldsymbol{C_d^{(i_d)}}\right) \boldsymbol{e_{i_1,\dots,i_N}} \\
&= \boldsymbol{z},
\end{aligned} \tag{2.21}$$

which shows that the matrices $\boldsymbol{C_d^{(i_d)}}$ that make up the MPS vector $\boldsymbol{z}$ have a size equal to the sum of the sizes of $\boldsymbol{A_d}^{(i_d)}$ and $\boldsymbol{B_d^{(i_d)}}$. In the OBC case, the first and last sites have to be kept as vectors, so we set

$$\boldsymbol{C_1^{(i_1)}} = \begin{pmatrix} \boldsymbol{A_1^{(i_1)}} & \boldsymbol{B_1^{(i_1)}} \end{pmatrix}, \quad \boldsymbol{C_N^{(i_N)}} = \begin{pmatrix} \boldsymbol{A_N^{(i_N)}} \\ \boldsymbol{B_N^{(i_N)}} \end{pmatrix}.$$

The second important operation is the product of an MPO $\boldsymbol{A}$ with an MPS $\boldsymbol{x}$, which results in another MPS whose components are given by

$$\begin{aligned}
(\boldsymbol{Ax})_{i_1,\dots,i_N} &= \sum_{j_1,\dots,j_N} A_{\substack{i_1,\dots,i_N \\ j_1,\dots,j_N}} x_{j_1,\dots,j_N} \\
&= \sum_{j_1,\dots,j_N} \mathrm{Tr}\left(\boldsymbol{W_1^{(i_1,j_1)}} \cdots \boldsymbol{W_N^{(i_N,j_N)}}\right) \mathrm{Tr}\left(\boldsymbol{X_1^{(j_1)}} \cdots \boldsymbol{X_N^{(j_N)}}\right) \\
&= \mathrm{Tr}\left(\left(\sum_{j_1} \boldsymbol{W_1^{(i_1,j_1)}} \otimes \boldsymbol{X_1^{(j_1)}}\right) \cdots \left(\sum_{j_N} \boldsymbol{W_N^{(i_N,j_N)}} \otimes \boldsymbol{X_N^{(j_N)}}\right)\right) \\
&:= \mathrm{Tr}\left(\boldsymbol{B_1^{(i_1)}} \cdots \boldsymbol{B_N^{(i_N)}}\right).
\end{aligned} \tag{2.22}$$

Therefore, if the matrices $\boldsymbol{W_d^{(i_d,j_d)}}$ and $X_d^{(j_d)}$ have sizes $D_d \times D_{d+1}$ and $L_d \times L_{d+1}$ respectively, the resulting matrices $\boldsymbol{B_d^{(i_d)}}$ will have size $D_d L_d \times D_{d+1} L_{d+1}$.

In both operations, the sizes of the MPS matrices grow, which is why it is important in many cases to truncate the resulting MPS to a smaller bond dimension by means of Eq. (2.20). Following this reasoning, we propose in Chapter 3 to set $\boldsymbol{y} \equiv \boldsymbol{Az}$ in (2.20), so that the minimization problem translates to solving a linear system of equations $\boldsymbol{Az} = \boldsymbol{x}$. Chapter 4 extends this same principle to find an approximate inverse MPO $\boldsymbol{M}$ of a given MPO $\boldsymbol{A}$ by minimizing $\|\boldsymbol{AM} - \boldsymbol{I}\|_F$.

# 3 Solving linear systems in MPS format: an Alternating Least Squares procedure

## 3.1 Statement of the Problem

Representing vectors as MPS and matrices as MPO allows us to reduce the complexity of iterative algorithms for exponentially large systems, and we will now focus on a problem that is at the core of numerical linear algebra, namely, finding a solution to the linear system of equations given by

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \tag{3.1}$$

In what follows, we will use a binary decomposition of the indices for both MPS and MPO ($p = 2$ in their definitions (2.6),(2.9)). There are two approaches to this problem in the context of MPS and MPO:

1. The system matrix $\boldsymbol{A} \in \mathbb{C}^{2^N \times 2^N}$ and the right hand side $\boldsymbol{b} \in \mathbb{C}^{2^N}$ are given in MPO and MPS format respectively, and we wish to find an approximate solution $\boldsymbol{x}$ also in MPS format with a certain bond dimension $L$. Assuming that the system has a unique solution,[1] the question is whether such solution $\boldsymbol{x}$ can be approximated well by an MPS with a moderate bond dimension $L$. Therefore, we try to find $\boldsymbol{x}$ such that

$$\min_{x \in \mathrm{MPS}_L} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2. \tag{3.2}$$

2. The matrix $\boldsymbol{A}$ and the right hand $\boldsymbol{b}$ are not given in MPO/MPS format and therefore have to be converted to MPO and MPS format, which could result in only approximate representations depending on their bond dimensions. Then, $\boldsymbol{x}$ would be a solution of the approximated system.

We will focus on the first case, that is, $\boldsymbol{A}$ is a given MPO and $\boldsymbol{b}$ a given MPS, and we wish to find $\boldsymbol{x} \in \mathrm{MPS}_L$ that minimizes Eq. (3.2). Alternatively, one may specify the accuracy $\varepsilon$ with which the solution is to be calculated without restricting the bond dimension $L$. This can be achieved, as we will see, with the DMRG algorithm.

The most important question in the context of such a tensor representation is: *what is the bond dimension of the MPS solution $\boldsymbol{x}$*? We know from Theorem 2.1 that any vector can be represented in MPS format as long as the matrices that compose it are chosen large enough. Ideally, we want such matrices to be small. Can this be achievable? How well can the solution be approximated if we prescribe a maximum bond dimension? We will come back to these questions during the discussion of the results.

[1] Here, it is meant that the system of equations admits a unique solution, but the MPS representation of such solution (or of any vector) is not unique.

Using a tensor decomposition of vectors to solve linear systems is an idea used already in 2003 [9] for a certain class of matrices $\boldsymbol{A}$. Very recently, [19] provided a detailed description of the solution of (3.1) in TT format (OBC-MPS) using the DMRG algorithm as described in this chapter, thus outperforming our treatment. Nevertheless, we present our implementation (additionally including the PBC-MPS case) by providing a detailed description of the resulting algorithm and the steps to obtain it, also hoping to shed some light on the tensor contraction schemes involved.

## 3.2 Theory

Suppose we are given the linear system $\boldsymbol{Ax} = \boldsymbol{b}$ in MPO/MPS form, namely,

$$\boldsymbol{A} = \sum_{\substack{i_1,\dots,i_N \\ j_1,\dots,j_N}} \mathrm{Tr}\left(\boldsymbol{W_1^{(i_1,j_1)}} \cdots \boldsymbol{W_N^{(i_N,j_N)}}\right) \bigotimes_{n=1}^{N} e_{i_n} e_{j_n}^{\intercal}, \tag{3.3}$$

$$\boldsymbol{b} = \sum_{i_1,\dots,i_N} \mathrm{Tr}\left(\boldsymbol{M_1^{(i_1)}} \cdots \boldsymbol{M_N^{(i_N)}}\right) \bigotimes_{n=1}^{N} e_{i_n}. \tag{3.4}$$

Recall that each element $\boldsymbol{W_d^{(i_d,j_d)}}$ appearing in the definition of A is a set of four matrices (for $i_d = 0, 1$, $j_d = 0, 1$), each of size $D_d \times D_{d+1}$. The $\boldsymbol{M_d^{(i_d)}}$ matrices appearing in the definition of $\boldsymbol{b}$ are pairs of matrices of size $Q_d \times Q_{d+1}$. Now, we look for an MPS vector $\boldsymbol{x}$ of the form

$$\begin{aligned} \boldsymbol{x} &= \sum_{i_1,\dots,i_N} \mathrm{Tr}\left(\boldsymbol{X_1^{(i_1)}} \cdots \boldsymbol{X_N^{(i_N)}}\right) \bigotimes_{n=1}^{N} e_{i_n} \\ &= \sum_{q_1,\dots,q_N} \boldsymbol{x_{1;q_1,q_2}} \otimes \cdots \otimes \boldsymbol{x_{N;q_N,q_1}}, \end{aligned} \tag{3.5}$$

where each site $d$ has pairs of matrices $\boldsymbol{X_d^{(i_d)}}$, each of size $L_d \times L_{d+1}$, with bond dimension $L = \max_d L_d$, and we look for the MPS vector $\boldsymbol{x}$ such that

$$\min_{\boldsymbol{x} \in \mathrm{MPS}_L} \|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2. \tag{3.6}$$

Thus, we try to solve the linear system in a least squares sense, and we will describe an algorithm to find such a solution iteratively.

The product of $\boldsymbol{A}$ times $\boldsymbol{x}$ is given by

$$\begin{aligned} (\boldsymbol{Ax})_{i_1,\dots,i_N} &= \sum_{j_1,\dots,j_N} A_{\substack{i_1,\dots,i_N \\ j_1,\dots,j_N}} x_{j_1,\dots,j_N} \\ &:= \mathrm{Tr}\left(\boldsymbol{B_1^{(i_1)}} \cdots \boldsymbol{B_N^{(i_N)}}\right), \end{aligned} \tag{3.7}$$

with $\boldsymbol{B_d}$ matrices given by

$$\boldsymbol{B_d^{(i_d)}} := \sum_{j_d} \boldsymbol{W_d^{(i_d,j_d)}} \otimes \boldsymbol{X_d^{(j_d)}}. \tag{3.8}$$

We expand the norm as

$$\begin{aligned}\|\boldsymbol{Ax}-\boldsymbol{b}\|_2^2 = \quad & (\boldsymbol{Ax})^{\mathrm{H}}(\boldsymbol{Ax}) - (\boldsymbol{Ax})^{\mathrm{H}}\boldsymbol{b} - \boldsymbol{b}^{\mathrm{H}}(\boldsymbol{Ax}) + \boldsymbol{b}^{\mathrm{H}}\boldsymbol{b}, \\ = & \sum_{i_1,\ldots,i_n} \mathrm{Tr}\left(\bar{\boldsymbol{B}}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\cdots\bar{\boldsymbol{B}}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right)\mathrm{Tr}\left(\boldsymbol{B}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\cdots\boldsymbol{B}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right) \\ & - \sum_{i_1,\ldots,i_n} \mathrm{Tr}\left(\bar{\boldsymbol{B}}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\cdots\bar{\boldsymbol{B}}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right)\mathrm{Tr}\left(\boldsymbol{M}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\cdots\boldsymbol{M}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right) \\ & - \sum_{i_1,\ldots,i_n} \mathrm{Tr}\left(\bar{\boldsymbol{M}}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\cdots\bar{\boldsymbol{M}}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right)\mathrm{Tr}\left(\boldsymbol{B}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\cdots\boldsymbol{B}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right) \\ & + \sum_{i_1,\ldots,i_n} \mathrm{Tr}\left(\bar{\boldsymbol{M}}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\cdots\bar{\boldsymbol{M}}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right)\mathrm{Tr}\left(\boldsymbol{M}_{\boldsymbol{1}}^{(\boldsymbol{i_1})}\cdots\boldsymbol{M}_{\boldsymbol{N}}^{(\boldsymbol{i_N})}\right). \end{aligned} \tag{3.9}$$

The first term is given by

$$(\boldsymbol{Ax})^{\mathrm{H}}(\boldsymbol{Ax}) = \sum_{\substack{r_1,\ldots,r_N \\ s_1,\ldots,s_N}} \left(\sum_{i_1} \bar{b}_{1;r_1,r_2}^{(i_1)} b_{1;s_1,s_2}^{(i_1)}\right)\cdots\left(\sum_{i_N} \bar{b}_{N;r_N,r_1}^{(i_N)} b_{N;s_N,s_1}^{(i_N)}\right), \tag{3.10}$$

where

$$\bar{b}_{d;r_d,r_{d+1}}^{(i_d)} = \sum_{j_d'} \bar{w}_{d;m_d,m_{d+1}}^{(i_d,j_d')} \bar{x}_{d;n_d,n_{d+1}}^{(j_d')}, \tag{3.11}$$

$$b_{d;s_d,s_{d+1}}^{(i_d)} = \sum_{j_d} w_{d;p_d,p_{d+1}}^{(i_d,j_d)} x_{d;q_d,q_{d+1}}^{(j_d)}, \tag{3.12}$$

where the indices are related by

$$r_d := L_d(m_d - 1) + n_d, \tag{3.13}$$

$$s_d := L_d(p_d - 1) + q_d. \tag{3.14}$$

The second term in (3.9) gives

$$(\boldsymbol{Ax})^{\mathrm{H}}\boldsymbol{b} = \sum_{\substack{r_1,\ldots,r_N \\ s_1,\ldots,s_N}} \left(\sum_{i_1} \bar{b}_{1;r_1,r_2}^{(i_1)} m_{1;s_1,s_2}^{(i_1)}\right)\cdots\left(\sum_{i_N} \bar{b}_{N;r_N,r_1}^{(i_N)} m_{N;s_N,s_1}^{(i_N)}\right). \tag{3.15}$$

Alternatively, we can interpret $\boldsymbol{B}$ as a tensor with five indices instead of just three, by explicitly leaving the $p$ and $q$ indices:

$$b_{d;s_d,s_{d+1}}^{(i_d)} \equiv b_{\substack{d;p_d,p_{d+1} \\ q_d,q_{d+1}}}^{(i_d)} = \sum_{j_d} w_{d;p_d,p_{d+1}}^{(i_d,j_d)} x_{d;q_d,q_{d+1}}^{(j_d)} \tag{3.16}$$

$$\bar{b}_{d;r_d,r_{d+1}}^{(i_d)} \equiv \bar{b}_{\substack{d;m_d,m_{d+1} \\ n_d,n_{d+1}}}^{(i_d)} = \sum_{j_d'} \bar{w}_{d;m_d,m_{d+1}}^{(i_d,j_d')} \bar{x}_{d;n_d,n_{d+1}}^{(j_d')} \tag{3.17}$$

and write

$$
\begin{aligned}
(\boldsymbol{A}\boldsymbol{x})^{\mathrm{H}}(\boldsymbol{A}\boldsymbol{x}) &= \sum_{\substack{m_1,\dots,m_N\\ n_1,\dots,n_N\\ p_1,\dots,p_N\\ q_1,\dots,q_N}} \left(\sum_{i_1} \bar{b}^{(i_1)}_{1;\substack{m_1,m_2\\ n_1,n_2}} b^{(i_1)}_{1;\substack{p_1,p_2\\ q_1,q_2}}\right) \cdots \left(\sum_{i_N} \bar{b}^{(i_N)}_{N;\substack{m_N,m_1\\ n_N,n_1}} b^{(i_N)}_{N;\substack{p_N,p_1\\ q_N,q_1}}\right) \\
&= \sum_{m,n,p,q} c_{1;\substack{m_1,m_2\\ n_1,n_2\\ p_1,p_2\\ q_1,q_2}} \cdots c_{N;\substack{m_N,m_1\\ n_N,n_1\\ p_N,p_1\\ q_N,q_1}} \\
&= \sum_{m,n,p,q} \prod_{\ell=1}^{N} c_{\ell;\substack{m_\ell,m_{\ell+1}\\ n_\ell,n_{\ell+1}\\ p_\ell,p_{\ell+1}\\ q_\ell,q_{\ell+1}}}.
\end{aligned} \tag{3.18}
$$

The $\boldsymbol{C_d}$ tensors introduced above are defined componentwise as

$$
c_{d;\substack{m_d,m_{d+1}\\ n_d,n_{d+1}\\ p_d,p_{d+1}\\ q_d,q_{d+1}}} := \sum_{i_d} \bar{b}^{(i_d)}_{d;\substack{m_d,m_{d+1}\\ n_d,n_{d+1}}} b^{(i_d)}_{d;\substack{p_d,p_{d+1}\\ q_d,q_{d+1}}}, \tag{3.19}
$$

(index $N+1$ always corresponds to index 1) and

$$
\begin{aligned}
(\boldsymbol{A}\boldsymbol{x})^{\mathrm{H}}\boldsymbol{b} &= \sum_{\substack{m_1,\dots,m_N\\ n_1,\dots,n_N\\ s_1,\dots,s_N}} \left(\sum_{i_1} \bar{b}^{(i_1)}_{1;\substack{m_1,m_2\\ n_1,n_2}} m^{(i_1)}_{1;s_1,s_2}\right) \cdots \left(\sum_{i_N} \bar{b}^{(i_N)}_{N;\substack{m_N,m_1\\ n_N,n_1}} m^{(i_N)}_{N;s_N,s_1}\right) \\
&= \sum_{m,n,s} \tilde{c}_{1;\substack{m_1,m_2\\ n_1,n_2\\ s_1,s_2}} \cdots \tilde{c}_{N;\substack{m_N,m_1\\ n_N,n_1\\ s_N,s_1}} \\
&= \sum_{m,n,s} \prod_{\ell=1}^{N} \tilde{c}_{\ell;\substack{m_\ell,m_{\ell+1}\\ n_\ell,n_{\ell+1}\\ s_\ell,s_{\ell+1}}},
\end{aligned} \tag{3.20}
$$

with $\tilde{\boldsymbol{C}}_{\boldsymbol{d}}$ tensors defined componentwise as

$$
\tilde{c}_{d;\substack{m_d,m_{d+1}\\ n_d,n_{d+1}\\ s_d,s_{d+1}}} := \sum_{i_d} \bar{b}^{(i_d)}_{d;\substack{m_d,m_{d+1}\\ n_d,n_{d+1}}} m^{(i_d)}_{d;s_d,s_{d+1}}. \tag{3.21}
$$

This way, the contraction schemes and index relationships become more clear, taking care to handle the indices correctly.

Finding the solution $\boldsymbol{x}$ of (3.6) means finding the components of the matrices $\boldsymbol{X}_{\boldsymbol{d}}^{(\boldsymbol{i_d})}$ that minimize $\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2$, so we calculate the derivative of $(\boldsymbol{A}\boldsymbol{x})^{\mathrm{H}}(\boldsymbol{A}\boldsymbol{x}) - (\boldsymbol{A}\boldsymbol{x})^{H}\boldsymbol{b}$ with respect to the matrix elements $\bar{x}^{(j'_d)}_{d;n_d,n_{d+1}}$:

$$
\frac{\partial}{\partial \bar{x}^{(j'_d)}_{d;n_d,n_{d+1}}} \left((\boldsymbol{A}\boldsymbol{x})^{\mathrm{H}}(\boldsymbol{A}\boldsymbol{x}) - (\boldsymbol{A}\boldsymbol{x})^{H}\boldsymbol{b}\right) = 0. \tag{3.22}
$$

(The other two terms in (3.9) don't contribute to the derivative since they do not involve the matrices $\bar{\boldsymbol{X}}_{\boldsymbol{d}}^{(\boldsymbol{i_d})}$.) The first term gives

$$\begin{aligned}\frac{\partial\left((\boldsymbol{Ax})^{\mathrm{H}}(\boldsymbol{Ax})\right)}{\partial \bar{x}^{(j'_d)}_{d;n_d,n_{d+1}}} &= \sum_{\substack{m,p,q\\ n_1,\dots,n_{d-1}\\ n_{d+2},\dots,n_N}} c_{1;\substack{m_1,m_2\\n_1,n_2\\p_1,p_2\\q_1,q_2}} \cdots c_{d-1;\substack{m_{d-1},m_d\\n_{d-1},n_d\\p_{d-1},p_d\\q_{d-1},q_d}} \cdot \sum_{i_d,j_d} \bar{w}^{(i_d,j'_d)}_{d;m_d,m_{d+1}} w^{(i_d,j_d)}_{d;p_d,p_{d+1}} x^{(j_d)}_{d;q_d,q_{d+1}} \\ &\qquad c_{d+1;\substack{m_{d+1},m_{d+2}\\n_{d+1},n_{d+2}\\p_{d+1},p_{d+2}\\q_{d+1},q_{d+2}}} \cdots c_{N;\substack{m_N,m_1\\n_N,n_1\\p_N,p_1\\q_N,q_1}} \\ &:= \sum_{q_d,q_{d+1}} p^{(j'_d,j_d)}_{\substack{d;n_d,n_{d+1}\\q_d,q_{d+1}}} x^{(j_d)}_{d;q_d,q_{d+1}},\end{aligned} \tag{3.23}$$

and the tensor $\boldsymbol{P_d}$ is defined as

$$p^{(j'_d,j_d)}_{\substack{d;n_d,n_{d+1}\\q_d,q_{d+1}}} := \sum_{\substack{i_d,m,p\\ n_1,\dots,n_{d-1}\\ n_{d+2},\dots,n_N\\ q_1,\dots,q_{d-1}\\ q_{d+2},\dots,q_N}} \prod_{\substack{\ell=1\\ \ell\neq d}}^{N} c_{\ell;\substack{m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}\\p_\ell,p_{\ell+1}\\q_\ell,q_{\ell+1}}} \bar{w}^{(i_d,j'_d)}_{d;m_d,m_{d+1}} w^{(i_d,j_d)}_{d;p_d,p_{d+1}}. \tag{3.24}$$

(index $N+1$ always corresponding to index 1). The second term in the derivative gives

$$\begin{aligned}\frac{\partial\left((Ax)^{\mathrm{H}}b\right)}{\partial \bar{x}^{(j'_d)}_{d;n_d,n_{d+1}}} &= \sum_{\substack{m,s\\ n_1,\dots,n_{d-1}\\ n_{d+2},\dots,n_N}} \tilde{c}_{1;\substack{m_1,m_2\\n_1,n_2\\s_1,s_2}} \cdots \tilde{c}_{d-1;\substack{m_{d-1},m_d\\n_{d-1},n_d\\s_{d-1},s_d}} \sum_{i_d} \bar{w}^{(i_d,j'_d)}_{d;m_d,m_{d+1}} m^{(i_d)}_{d;s_d,s_{d+1}} \\ &\qquad \tilde{c}_{d+1;\substack{m_{d+1},m_{d+2}\\n_{d+1},n_{d+2}\\s_{d+1},s_{d+2}}} \cdots \tilde{c}_{N;\substack{m_N,m_1\\n_N,n_1\\s_N,s_1}} \\ &= \sum_{\substack{i_d,m,s\\ n_1,\dots,n_{d-1}\\ n_{d+2},\dots,n_N}} \prod_{\substack{\ell=1\\ \ell\neq d}}^{N} \tilde{c}_{\ell;\substack{m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}\\s_\ell,s_{\ell+1}}} \bar{w}^{(i_d,j'_d)}_{d;m_d,m_{d+1}} m^{(i_d)}_{d;s_d,s_{d+1}} \\ &:= \tilde{p}^{(j'_d)}_{d;n_d,n_{d+1}},\end{aligned} \tag{3.25}$$

Thus, the derivative (3.22) can be written as

$$\sum_{q_d,q_{d+1}} \sum_{j_d} p^{(j'_d,j_d)}_{\substack{d;n_d,n_{d+1}\\q_d,q_{d+1}}} x^{(j_d)}_{d;q_d,q_{d+1}} = \tilde{p}^{(j'_d)}_{d;n_d,n_{d+1}}, \tag{3.26}$$

which has to be solved for all $n_d, n_{d+1}$ and $j_d$. This can be rewritten in matrix form as

$$\begin{pmatrix} p^{(0,0)}_{d;(n_d,n_{d+1}),(q_d,q_{d+1})} & p^{(0,1)}_{d;(n_d,n_{d+1}),(q_d,q_{d+1})} \\ p^{(1,0)}_{d;(n_d,n_{d+1}),(q_d,q_{d+1})} & p^{(1,1)}_{d;(n_d,n_{d+1}),(q_d,q_{d+1})} \end{pmatrix} \begin{pmatrix} x^{(0)}_{d;q_d,q_{d+1}} \\ x^{(1)}_{d;q_d,q_{d+1}} \end{pmatrix} = \begin{pmatrix} \tilde{p}^{(0)}_{d;n_d,n_{d+1}} \\ \tilde{p}^{(1)}_{d;n_d,n_{d+1}} \end{pmatrix}. \tag{3.27}$$

The alternating least squares (ALS) procedure to solve for all matrices $\boldsymbol{X_d}$ is as follows. At a given site $d = 1, \dots, N-1$ (the algorithm is started at $d = 1$) we solve Eq. (3.27)

to obtain the two matrices $\boldsymbol{X_d^{(0)}}$, $\boldsymbol{X_d^{(1)}}$ and normalize this solution by taking its singular value decomposition (SVD), namely

$$\begin{pmatrix} \boldsymbol{X_d^{(0)}} \\ \boldsymbol{X_d^{(1)}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{U_d^{(0)}} \\ \boldsymbol{U_d^{(1)}} \end{pmatrix} \cdot \boldsymbol{\Sigma_d} \cdot \boldsymbol{V_d}, \tag{3.28}$$

where we have ordered the two matrices $\boldsymbol{X_d^{(i_d)}}$ rowwise into one rectangular matrix. The $\boldsymbol{U_d^{(i_d)}}$ matrices are unitary and satisfy the gauge condition

$$\sum_{i_d} \boldsymbol{U_d^{(i_d)H}} \boldsymbol{U_d^{(i_d)}} = \boldsymbol{I}. \tag{3.29}$$

Recall that $\boldsymbol{\Sigma_d}$ is diagonal with nonnegative real entries (called singular values) and $\boldsymbol{V_d}$ is also unitary. Now, we substitute our two solution matrices $\boldsymbol{X_d^{(i_d)}}$ by the unitary matrices $\boldsymbol{U_d^{(i_d)}}$ and move the remaining factor $\boldsymbol{\Sigma_d} \cdot \boldsymbol{V_d}$ to the right neighbor, that is,

$$\mathrm{Tr}\left( \boldsymbol{X_1^{(i_1)}} \cdot \boldsymbol{X_2^{(i_2)}} \cdots \boldsymbol{X_d^{(i_d)}} \cdot \boldsymbol{X_{d+1}^{(i_{d+1})}} \cdots \boldsymbol{X_{N-1}^{(i_{N-1})}} \cdot \boldsymbol{X_N^{(i_N)}} \right) \rightarrow \tag{3.30}$$

$$\mathrm{Tr}\left( \boldsymbol{X_1^{(i_1)}} \cdot \boldsymbol{X_2^{(i_2)}} \cdots \boldsymbol{U_d^{(i_d)}} \cdot \underbrace{(\boldsymbol{\Sigma_d} \cdot \boldsymbol{V_d}) \cdot \boldsymbol{X_{d+1}^{(i_{d+1})}}}_{\rightarrow \boldsymbol{X_{d+1}^{(i_{d+1})}}} \cdots \boldsymbol{X_{N-1}^{(i_{N-1})}} \cdot \boldsymbol{X_N^{(i_N)}} \right). \tag{3.31}$$

Once the matrices at site $d$ are substituted and those at site $d+1$ updated, we repeat the procedure at site $d+1$ until reaching site $N-1$. At this point, all sites except $d = N$ will be normalized, and a right-to-left sweep can be performed analogously, this time taking

$$\begin{pmatrix} \boldsymbol{X_d^{(0)}} & \boldsymbol{X_d^{(1)}} \end{pmatrix} = (\boldsymbol{U_d} \cdot \boldsymbol{\Sigma_d}) \cdot \begin{pmatrix} \boldsymbol{V_d^{(0)}} & \boldsymbol{V_d^{(1)}} \end{pmatrix}, \tag{3.32}$$

that is, substituting the matrices $\boldsymbol{X_d^{(i_d)}}$ with the unitary matrices $\boldsymbol{V_d^{(i_d)}}$ and finally moving the remaining factor $\boldsymbol{U_d} \cdot \boldsymbol{\Sigma_d}$ to the left neighbor, for $d = N, \ldots, 2$. One iteration of the ALS procedure consists in sweeping once from left to right, and once from right to left, at which point, all sites except $d = 1$ are normalized.

In the periodic boundary case, we carry out the same procedure but reaching site $N$ during the left-to-right sweep, at which point we shift the SVD factor to the right neighbor *site 1*, and analogously for the right-to-left sweep.

## 3.3 Numerical considerations

In order to perform the numerical computations efficiently, it is convenient to define the following left ($\boldsymbol{F_d}$) and right ($\boldsymbol{G_d}$) factors of the $\boldsymbol{P_d}$ tensor defined above, as

$$f_{\substack{d;m_1,m_d\\n_1,n_d\\p_1,p_d\\q_1,q_d}} := \sum_{\substack{m_2,\dots,m_{d-1}\\n_2,\dots,n_{d-1}\\p_2,\dots,p_{d-1}\\q_2,\dots,q_{d-1}}} \prod_{\ell=1}^{d-1} c_{\substack{\ell;m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}\\p_\ell,p_{\ell+1}\\q_\ell,q_{\ell+1}}} \tag{3.33}$$

$$g_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}\\p_1,p_{d+1}\\q_1,q_{d+1}}} := \sum_{\substack{m_{d+2},\dots,m_N\\n_{d+2},\dots,n_N\\p_{d+2},\dots,p_N\\q_{d+2},\dots,q_N}} \prod_{\ell=d+1}^{N} c_{\substack{\ell;m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}\\p_\ell,p_{\ell+1}\\q_\ell,q_{\ell+1}}} \tag{3.34}$$

so that the $\boldsymbol{P_d}$ tensor appearing in Eq.(3.23) can be expressed as

$$p^{(j'_d,j_d)}_{\substack{d;n_d,n_{d+1}\\q_d,q_{d+1}}} = \sum_{\substack{i_d\\m_1,n_1,p_1,q_1\\m_d,m_{d+1}\\p_d,p_{d+1}}} f_{\substack{d;m_1,m_d\\n_1,n_d\\p_1,p_d\\q_1,q_d}} \bar{w}^{(i_d,j'_d)}_{d;m_d,m_{d+1}} w^{(i_d,j_d)}_{d;p_d,p_{d+1}} g_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}\\p_1,p_{d+1}\\q_1,q_{d+1}}}. \tag{3.35}$$

This rewriting is useful to avoid performing unnecessary contractions when moving from site $d$ to site $d+1$ (or $d-1$ if sweeping from right to left). If we are minimizing site $d$, it means that the left factor (all tensors from 1 to $d-1$) has been calculated, and moving to the next site requires us to update the left factor $\boldsymbol{F_d}$ only with the newly calculated $\boldsymbol{X_d}$ matrices (hidden inside de $\boldsymbol{C_d}$ tensors) at site $d$, namely:

$$f_{\substack{d+1;m_1,m_{d+1}\\n_1,n_{d+1}\\p_1,p_{d+1}\\q_1,q_{d+1}}} = \sum_{\substack{m_d,n_d\\p_d,q_d}} f_{\substack{d;m_1,m_d\\n_1,n_d\\p_1,p_d\\q_1,q_d}} \underbrace{c_{\substack{d;m_d,m_{d+1}\\n_d,n_{d+1}\\p_d,p_{d+1}\\q_d,q_{d+1}}}}_{\text{Newly calculated}}. \tag{3.36}$$

For the right factor $\boldsymbol{G_d}$, we can use the following recurrence relation:

$$g_{\substack{d-1;m_1,m_d\\n_1,n_d\\p_1,p_d\\q_1,q_d}} = \sum_{\substack{m_{d+1},n_{d+1}\\p_{d+1},q_{d+1}}} c_{\substack{d;m_d,m_{d+1}\\n_d,n_{d+1}\\p_d,p_{d+1}\\q_d,q_{d+1}}} g_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}\\p_1,p_{d+1}\\q_1,q_{d+1}}} \tag{3.37}$$

For the $\tilde{\boldsymbol{P}}$ tensor appearing in Eq. (3.25), we define analogously

$$\tilde{f}_{\substack{d;m_1,m_d\\n_1,n_d\\s_1,s_d}} := \sum_{\substack{m_2,\dots,m_{d-1}\\n_2,\dots,n_{d-1}\\s_2,\dots,s_{d-1}}} \prod_{\ell=1}^{d-1} \tilde{c}_{\substack{\ell;m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}\\s_\ell,s_{\ell+1}}}, \tag{3.38}$$

$$\tilde{g}_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}\\s_1,s_{d+1}}} := \sum_{\substack{m_{d+2},\dots,m_N\\n_{d+2},\dots,n_N\\s_{d+2},\dots,s_N}} \prod_{\ell=d+1}^{N} \tilde{c}_{\substack{\ell;m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}\\s_\ell,s_{\ell+1}}}, \tag{3.39}$$

and express $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ as

$$\tilde{p}^{(j'_d)}_{d;n_d,n_{d+1}} = \sum_{\substack{i_d\\m_1,n_1,s_1\\m_d,m_{d+1}\\s_d,s_{d+1}}} \tilde{f}_{\substack{d;m_1,m_d\\n_1,n_d\\s_1,s_d}} \bar{w}^{(i_d,j'_d)}_{d;m_d,m_{d+1}} m^{(i_d)}_{d;s_d,s_{d+1}} \tilde{g}_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}\\s_1,s_{d+1}}}. \tag{3.40}$$

**Algorithm 1:** Approximate MPS solution of $\boldsymbol{Ax} = \boldsymbol{b}$.

**input** : $\boldsymbol{A} = \boldsymbol{A}[\boldsymbol{W}_1, \ldots, \boldsymbol{W_N}] \in \text{MPO}_D$, MPO system matrix.
$\boldsymbol{b} = \boldsymbol{b}[\boldsymbol{M_1}, \ldots, \boldsymbol{M_N}] \in \text{MPS}_N$, right hand side MPS.
**output**: $\boldsymbol{x} = \boldsymbol{x}[\boldsymbol{X_1}, \ldots, \boldsymbol{X_N}] \in \text{MPS}_L$, approximate solution of $\boldsymbol{Ax} = \boldsymbol{b}$.

generate initial guess for $\boldsymbol{x} \in \text{MPS}_L$;
pre-compute **all** right tensors $\boldsymbol{G_d}$, $\tilde{\boldsymbol{G}}_{\boldsymbol{d}}$ for $d = 1, \ldots, N-1$, with $\boldsymbol{G_N} = \tilde{\boldsymbol{G}}_{\boldsymbol{N}} = 1$ using Eqs. (3.34) and (3.39) ;
initialize $\boldsymbol{F_1} = \tilde{\boldsymbol{F}}_{\boldsymbol{1}} = 1$;
**while** *not converged* **do**
  **for** $d = 1$ **to** $N-1$ **do** `// left-to-right sweep`
    calculate $\boldsymbol{P_d}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ using Eqs. (3.35) and (3.40);
    solve for $\boldsymbol{X_d}$, the solution matrices at site $d$, Eq. (3.26);
    compute the SVD of $\boldsymbol{X}_{\boldsymbol{d}}^{(i_d)} = \boldsymbol{U}_{\boldsymbol{d}}^{(i_d)} \boldsymbol{\Sigma_d} \boldsymbol{V_d}$;
    substitute $\boldsymbol{X}_{\boldsymbol{d}}^{(i_d)} \leftarrow \boldsymbol{U}_{\boldsymbol{d}}^{(i_d)}$ and $\boldsymbol{X}_{\boldsymbol{d+1}}^{(i_{d+1})} \leftarrow \boldsymbol{\Sigma_d} \boldsymbol{V_d} \boldsymbol{X}_{\boldsymbol{d+1}}^{(i_{d+1})}$;
    calculate $\boldsymbol{F_{d+1}}$ and $\tilde{\boldsymbol{F}}_{\boldsymbol{d+1}}$ using Eqs. (3.36) and (3.41);
  **end**
  **for** $d = N$ **to** $2$ **do** `// right-to-left sweep`
    calculate $\boldsymbol{P_d}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ using Eqs. (3.35) and (3.40);
    solve for $\boldsymbol{X_d}$, the solution matrices at site $d$, Eq. (3.26);
    compute the SVD of $\boldsymbol{X}_{\boldsymbol{d}}^{(i_d)} = \boldsymbol{U_d} \boldsymbol{\Sigma_d} \boldsymbol{V}_{\boldsymbol{d}}^{(i_d)}$;
    substitute $\boldsymbol{X}_{\boldsymbol{d}}^{(i_d)} \leftarrow \boldsymbol{V}_{\boldsymbol{d}}^{(i_d)}$ and $\boldsymbol{X}_{\boldsymbol{d-1}}^{(i_{d-1})} \leftarrow \boldsymbol{X}_{\boldsymbol{d-1}}^{(i_{d-1})} \boldsymbol{V_d} \boldsymbol{\Sigma_d}$;
    calculate $\boldsymbol{G_{d-1}}$ and $\tilde{\boldsymbol{G}}_{\boldsymbol{d-1}}$ using Eqs. (3.37) and (3.42);
  **end**
**end**

The recurrence relations for $\tilde{\boldsymbol{F}}_{\boldsymbol{d}}$ and $\tilde{\boldsymbol{G}}_{\boldsymbol{d}}$ are given by

$$\tilde{f}_{d+1;\substack{m_1,m_{d+1}\\ n_1,n_{d+1}\\ s_1,s_{d+1}}} = \sum_{m_d,n_d,s_d} \tilde{f}_{d;\substack{m_1,m_d\\ n_1,n_d\\ s_1,s_d}} \tilde{c}_{d;\substack{m_d,m_{d+1}\\ n_d,n_{d+1}\\ s_d,s_{d+1}}}, \tag{3.41}$$

$$\tilde{g}_{d-1;\substack{m_1,m_d\\ n_1,n_d\\ s_1,s_d}} = \sum_{\substack{m_{d+1},n_{d+1}\\ s_{d+1}}} \tilde{c}_{d;\substack{m_d,m_{d+1}\\ n_d,n_{d+1}\\ s_d,s_{d+1}}} \tilde{g}_{d;\substack{m_1,m_{d+1}\\ n_1,n_{d+1}\\ s_1,s_{d+1}}}. \tag{3.42}$$

Algorithm 1 summarizes the steps required to solve our linear system (3.1) in MPO/MPS format.

### 3.3.1 Computational complexity

Let's look at Algorithm 1 from the point of view of its computational complexity. First, we suppose that both the MPO $\boldsymbol{A}$ and the right hand side MPS $\boldsymbol{b}$ are given. Recall that $\boldsymbol{A}$ is composed of $N$ matrices $\boldsymbol{W_d}$, $d = 1, \ldots, N$, each of size $D_d \times D_{d+1}$, with bond dimension $D$. The bond dimension of $\boldsymbol{x}$ is $L$ and for $\boldsymbol{b}$ it is $Q$. An efficient contraction scheme for MPS

contractions (which applies equally for MPO) is presented in [12], and it is this scheme that we shall use.

We start our discussion for the PBC case. Updating the left factors $\boldsymbol{F_{d+1}}$ and $\tilde{\boldsymbol{F}}_{\boldsymbol{d+1}}$ takes $\mathcal{O}(4L^5D^4)$ and $\mathcal{O}(L^3D^2Q^2)$ operations, respectively. The next step is to find the solution of the linear system $\boldsymbol{P_dX_d} = \tilde{\boldsymbol{P}}_{\boldsymbol{d}}$. The system matrix derived from the tensor $\boldsymbol{P_d}$ in Eq. (3.27) has size $2L_dL_{d+1} \times 2L_dL_{d+1}$. The complexity of solving this linear system depends on the method of choice. In [19] it is advised to use a direct solver if the local system matrix is of moderate size (for instance, the MATLAB backslash operator if the size is $\approx 10^3$). Such an approach has a complexity that is cubic in the matrix size. In our case, the size is $\mathcal{O}(2L^2)$, so the resulting complexity of solving the linear system would be $\mathcal{O}(8L^6)$. A whole sweep would then cost $\mathcal{O}(8NL^6)$. If the bond dimension $L$ becomes too large, an iterative solver is preferable.[2] Taking the SVD of the solution matrices $\boldsymbol{X_d}$ requires $\mathcal{O}(2L^3)$ operations, as well as multiplying $\boldsymbol{\Sigma_dV_d}$ times the right neighbor $\boldsymbol{X_{d+1}}$. Thus, the cost of one left-to-right ALS sweep costs approximately $\mathcal{O}(4NL^5D^4 + 8NL^6)$. Since the right-to-left sweep is equally expensive, this is also the total complexity of the algorithm.

The OBC case is less expensive, since the first and last indices are dummy. Updating the left tensors $\boldsymbol{F_{d+1}}$ and $\tilde{\boldsymbol{F}}_{\boldsymbol{d+1}}$ now takes $\mathcal{O}(4L^3D^2)$ and $\mathcal{O}(4L^2DQ)$, thus giving a complexity of approximately $\mathcal{O}(4NL^3D^2)$ in one ALS sweep.[3] The solution of the linear system has the same cost as in the PBC case, since the local matrices have the same size $\mathcal{O}(2L^2)$. Therefore, it still costs $\mathcal{O}(8NL^6)$ to solve the $N$ linear systems in one sweep using a direct solver. The total complexity is thus $\mathcal{O}(4NL^3D^2 + 8NL^6)$. Our implementation in this case seems to require more computational effort during the update of the left and right tensors, which indicates that the contractions might not be carried out in an optimal way.

## 3.4 Two-site optimization: DMRG

The main problem with our approach so far is that the dimensions of the solution matrices $\boldsymbol{X_d^{(j_d)}}$ are not know a priori. Here, we analyze how to overcome such problem. The idea, introduced in the context of quantum renormalization groups [29, 33], consists of optimizing two adjacent MPS sites simultaneously, i.e., $\boldsymbol{X_d^{(j_d)}X_{d+1}^{(j_{d+1})}}$, instead of only one. The linear systems to be solved are now larger, but once the two sites are optimized, we can perform the SVD of

$$\begin{pmatrix} \boldsymbol{X_d^{(0)}} \\ \boldsymbol{X_d^{(1)}} \end{pmatrix} \begin{pmatrix} \boldsymbol{X_{d+1}^{(0)}} & \boldsymbol{X_{d+1}^{(1)}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X_d^{(0)}X_{d+1}^{(0)}} & \boldsymbol{X_d^{(0)}X_{d+1}^{(1)}} \\ \boldsymbol{X_d^{(1)}X_{d+1}^{(0)}} & \boldsymbol{X_d^{(1)}X_{d+1}^{(1)}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{U_d^{(0)}} \\ \boldsymbol{U_d^{(1)}} \end{pmatrix} \boldsymbol{\Sigma_dV_d}, \tag{3.43}$$

and set $\boldsymbol{U_d^{(j_d)}}$ as the new $\boldsymbol{X_d^{(j_d)}}$ matrices, and the factor $\boldsymbol{\Sigma_dV_d}$ substitutes the $\boldsymbol{U_{d+1}^{(j_{d+1})}}$ matrices (it is not shifted and multiplied to the right, as before). This is to be performed during a left-to-right sweep for $d = 1, \ldots, N-2$. For the right-to-left sweep, we go from sites $N-1$ down to 1, but replacing the current site by the $\boldsymbol{V_d^{(i_d)}}$ matrices and the site to

[2] In our simulations, we kept the direct solver and worked with moderate bond dimensions, $L < 30$.

[3] The authors in [19] report a complexity that is only linear in $D$ instead of quadratic, but we were unable to obtain such result.

its left by the factor $\boldsymbol{U_d \Sigma_d}$. [4]

How is this related to the knowledge of the dimensions of the solution matrices? The important observation is that the $\boldsymbol{U_d}$ matrices in Eq. (3.43) are larger than the $\boldsymbol{X_d}$ matrices, since they have size $L_d \times 2L_{d+2}$ (the number of columns increases). They must therefore be truncated according to a given criterion, such that only the leading singular values of the matrix $\boldsymbol{\Sigma}$ are used and then truncate the columns of $\boldsymbol{U_d}$ accordingly. This results in the matrices $\boldsymbol{X_d}$ changing size depending on the number of singular values taken, and from this point of view the algorithm is adaptive. Several alternatives for the truncation are possible. If we denote $\hat{\boldsymbol{X}}_{\boldsymbol{d}}^{(\boldsymbol{i_d})}$ the solution matrices at site $d$ after the SVD truncation, [29] proposes to take $\|\boldsymbol{X}_{\boldsymbol{d}}^{(\boldsymbol{i_d})} - \hat{\boldsymbol{X}}_{\boldsymbol{d}}^{(\boldsymbol{i_d})}\| < \varepsilon$ with a fixed $\varepsilon$. It was recently proposed [19] to compare instead the residual of the local linear system after truncation, but the first alternative seems to show better convergence properties in our experiments.

## 3.5 Results

Algorithm 1 was implemented on a standard MATLAB distribution, as well as its DMRG variant. We start off with an example often found in literature, a $D$-dimensional Poisson equation on a unit $D$-cube

$$\begin{aligned} -\Delta_D u(x) &= f(x), \qquad x \in \Omega = [0,1]^D, \\ x &= [x_1, \ldots, x_D], \\ u(x) &= 0 \quad \text{in } \partial\Omega. \end{aligned} \tag{3.44}$$

The finite difference discretization of the 1D Laplacian using $2^N$ discretization points Eq. $\Delta_1^{(N)}$ (2.13) can be used to construct a $D$-dimensional Laplacian as follows [14]:

$$\Delta_D^{(N_1 \ldots N_D)} = \sum_{k=1}^{D} \left( \bigotimes_{i=1}^{k-1} \boldsymbol{I_{N_k}} \right) \otimes a_k \Delta_1^{(N_k)} \left( \bigotimes_{i=k+1}^{N} \boldsymbol{I_{N_k}} \right), \tag{3.45}$$

with $a_k = 1/h_k^2 = (2^{N_k}+1)^2$ and $\boldsymbol{I_{N_k}} = \boldsymbol{I}^{\otimes N_k}$ ($\boldsymbol{I}$ being the $2 \times 2$ identity matrix). Although Eq. (3.45) allows for a different number of discretization points in each dimension ($2^{N_k}$ on the $k$-th dimension), we will use the same number of points per dimension ($N_1 = \cdots = N_D \equiv N$). For small problem sizes, we compared the solutions with MATLAB's direct solver to confirm their correctness. As proposed in [6], we use

$$\frac{\|\boldsymbol{A}\boldsymbol{x}^{(j)} - \boldsymbol{b}\|}{\boldsymbol{b}} \leq \varepsilon$$

as convergence criterion for the $j$-th iteration. The MATLAB implementation of the $D$-dimensional Laplacian was adapted from the TT-Toolbox available at http://github.com/oseledets/TT-Toolbox to fit our data structures.

[4] In fact, the substitution by $\boldsymbol{U_d \Sigma_d}$ needs to be carried out only on the very last step of the iteration sweep (optimizing sites 1 and 2), since moving from sites $(j, j+1)$ to $(j-1, j)$ will overwrite the result in site $j$. In the left-to-right sweep, such substitution is never necessary.

### 3.5.1 ALS: a 1D Poisson example with known bond dimension

Fig. 3.5.1 exemplifies the main problem encountered by ALS, using a simple 1D Laplacian with $N = 10$. We can build an example for which we know the bond dimension a priori, by creating an arbitrary vector $\boldsymbol{x}$ (in our example, with random entries) of bond dimension 5, and then setting $\boldsymbol{b} \leftarrow \boldsymbol{A}\boldsymbol{x}$. The convergence of the ALS algorithm is then heavily dependent on the bond dimension of the initial guess, which can be clearly seen in the figure. As we will see in the next example, the DMRG algorithm overcomes this problem (to a good degree).

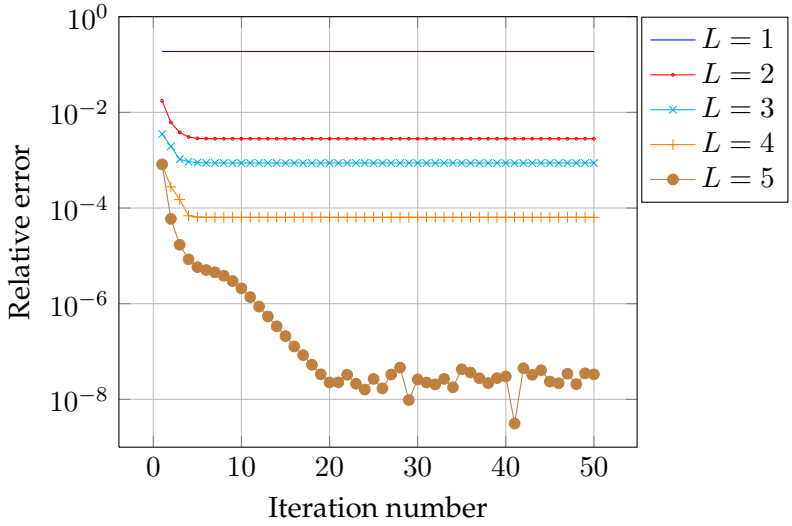



Figure 3.1: ALS convergence for Eq. (3.44) with $D = 1$ and $N = 10$, using a (randomly generated) solution of known bond dimension $L = 5$. If the bond dimension of the initial guess is underestimated (here, from $L = 1$ to $L = 4$), the algorithm quickly finds the optimal solution in that subspace (in the least squares sense) but then stalls.

### 3.5.2 DMRG for higher dimensions

Now we will fix the right hand side $f(x) = 1$, so that we don't know the bond dimension of the solution $\boldsymbol{x}$ a priori. The set of simulations ran are given in Table 3.1, where we increase the dimension of the Laplacian $D$ and the number of discretization points per dimension $2^N$. We consider the solution to have converged when the relative residual is smaller than $\varepsilon = 10^{-5}$. Instead of calculating the residual $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|$ directly, we approximate it[5] using the following convergence criterion [19] (which is also cheaper to calculate): we choose a random, normalized MPS vector $\boldsymbol{z}$ with bond dimension 1 and calculate the inner products $(\boldsymbol{A}\boldsymbol{x}^{(j)}, \boldsymbol{z})$ and $(\boldsymbol{b}, \boldsymbol{z})$, and then take the difference of both. Table 3.2 shows the number of DMRG iterations required for convergence, the total time, the final relative residual, and the bond dimension of the resulting solution.

[5]The result is slightly smaller than the actual residual due to the triangle inequality.

We observe that the algorithm performs better as the dimension of the Laplacian grows as compared to increasing the number of discretization points per dimension (even in 1D, choosing large $N$ shows stability problems, caused possibly by the large condition number of the Laplacian). From the algorithmic point of view, the main problem seems to be choosing the tolerance $\varepsilon$ for the SVD truncation. Choosing $\varepsilon$ too large may lead to a non-convergent solution, while choosing it too small causes the ranks to grow rapidly and even be overestimated. The alternative proposed in [20] (tracking the local residual for truncation) poses a similar problem (they introduce a parameter that depends from problem to problem). Notice that the largest problem solved (set 15) involves a 10-dimensional Laplacian with $2^8$ discretization points per dimension. That is, the full length solution vector would have $2^{80}$ entries.

Table 3.1: Problem parameters

| Set No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 6 | 7 | 8 | 10 |
| N | 5 | 10 | 12 | 5 | 10 | 5 | 10 | 5 | 8 | 5 | 8 | 8 | 8 | 8 | 8 |

Table 3.2: Results

| Set No. | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $n$ iter. | 2 | 2 | 7 | 3 | 6 |
| t (s) | 8.31e-02 | 2.20e-01 | 9.58e-01 | 5.72e-01 | 5.74e+01 |
| res | 1.67e-12 | 3.21e-06 | 7.47e-06 | 4.54e-07 | 4.57e-07 |
| $L_{\text{final}}$ | 3 | 6 | 5 | 14 | 25 |

| Set No. | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|
| $n$ iter. | 4 | 7 | 4 | 7 | 4 |
| t (s) | 1.76e+1 | 1.07e+2 | 1.15e+01 | 6.33e+01 | 1.54e+01 |
| res | 2.17e-10 | 8.20e-06 | 9.26e-08 | 6.69e-06 | 4.39e-07 |
| $L_{\text{final}}$ | 25 | 26 | 19 | 19 | 19 |

| Set No. | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|
| $n$ iter. | 6 | 6 | 6 | 6 | 5 |
| t (s) | 6.49e+01 | 8.17e+01 | 9.83e+01 | 1.16e+02 | 1.03e+02 |
| res | 5.33e-06 | 9.70e-06 | 5.18e-06 | 4.98e-06 | 2.02e-06 |
| $L_{\text{final}}$ | 22 | 19 | 19 | 19 | 19 |

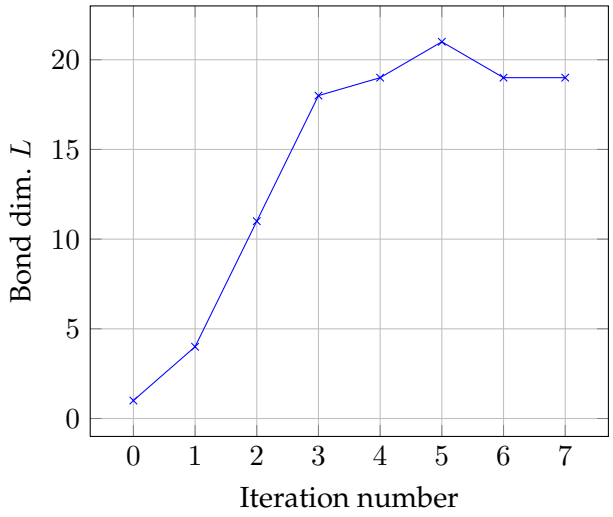



Figure 3.2: Evolution of the bond dimension $L$ of the solution for set 9 ($D = 4$, $N = 8$).

## 3.6 Conclusions

Both ALS and DMRG exploit the way the tensor is decomposed (as a product of matrices) by reducing the large, usually high-dimensional system of linear equations to $N$ smaller problems of moderate size, and then iterating over the local systems. If the bond dimension of the MPS and MPO are moderate, one can dramatically increase the dimensionality of the problem (or the refinement in each dimension) and still find a solution with good accuracy in little time. We will see in Chapter 7 how to include either ALS or DMRG in the context of Multigrid schemes (as smoothers). Nevertheless, DMRG has been the state of the art solver in the context of low-rank tensor representations, since it attacks the main problem of determining the bond dimension of the solution $\boldsymbol{x}$ directly. Very recently, however, new methods have been presented that are comparable or better than DMRG in TT-format [6, 7], so this is a highly active research field.

# 4 Approximate inverse of an MPO

## 4.1 Statement of the problem

Given a regular matrix $\boldsymbol{A}$, a good preconditioner $\boldsymbol{M}$ should provide an appropriate approximation of the inverse and thus the norm of the difference $\|\boldsymbol{AM}-\boldsymbol{I}\|$ should be small. The construcion of the sparse approximate inverse (SPAI) of a given matrix $\boldsymbol{A}$ is based on the minimization

$$\min_{\boldsymbol{M}\in\mathcal{M}} \|\boldsymbol{AM}-\boldsymbol{I}\|_F^2 = \min_{\boldsymbol{M}\in\mathcal{M}} \operatorname{Tr}\left((\boldsymbol{AM}-\boldsymbol{I})^{\mathrm{H}}(\boldsymbol{AM}-\boldsymbol{I})\right). \tag{4.1}$$

One typically specifies the set $\mathcal{M}\subseteq\mathbf{K}^{n\times n}$ of feasible preconditioners by prescribing the sparsity pattern of $\boldsymbol{M}$ in a static or dynamic way. We now want to study how the SPAI approach can be modified to the MPO concept, where both the original matrix $\boldsymbol{A}$ and the desired preconditioner $\boldsymbol{M}$ are represented as MPO. In other words, we are currently looking for a **data-sparse** approximate inverse of a given MPO, where the ranks of the desired preconditioner are bounded by some prescribed $L$:

$$\min_{\boldsymbol{M}\in\mathrm{MPO}_L} \|\boldsymbol{AM}-\boldsymbol{I}\|_F^2.$$

As in the previous chapter, a solution to this problem was also recently presented in [19] (as well as a variant in a related format [8]), but our discussion will be useful for Chapter 5, where we introduce new ideas on MPO preconditioners.

The discussion that follows resembles very much that of last chapter, and only the indices involved differ. We hope that this discussion will serve to add insight to the contractions of the tensor networks involved.

## 4.2 Theory

The idea of the approach is to evaluate Eq. (4.1) for the MPO representations of $\boldsymbol{A}$ and $\boldsymbol{M}$ and use an ALS method to solve iteratively for the components of $\boldsymbol{M}$. Once again, we write

$$\begin{aligned}
\boldsymbol{A} &= \sum_{\substack{i_1,\dots,i_N\\ j_1,\dots,j_N}} \operatorname{Tr}\left(\boldsymbol{W_1^{(i_1,j_1)}}\cdots\boldsymbol{W_N^{(i_N,j_N)}}\right)\bigotimes_{n=1}^{N} e_{i_n}e_{j_n}^{\mathsf{T}}\\
&= \sum_{p_1,\dots,p_N}\left(\sum_{i_1,j_1} w_{1;p_1,p_2}^{(i_1,j_1)} e_{i_1}e_{j_1}^{\mathsf{T}}\right)\otimes\cdots\otimes\left(\sum_{i_N,j_N} w_{N;p_N,p_1}^{(i_N,j_N)} e_{i_N}e_{j_N}^{\mathsf{T}}\right)\\
&= \sum_{p_1,\dots,p_N} \boldsymbol{w_{1;p_1,p_2}}\otimes\cdots\otimes\boldsymbol{w_{N;p_N,p_1}},
\end{aligned} \tag{4.2}$$

$$\begin{aligned}\boldsymbol{M} &= \sum_{\substack{i_1,\ldots,i_N\\ j_1,\ldots,j_N}} \mathrm{Tr}\left(\boldsymbol{X_1^{(i_1,j_1)}}\cdots\boldsymbol{X_N^{(i_N,j_N)}}\right)\bigotimes_{n=1}^{N} e_{i_n}e_{j_n}^{\intercal}\\ &= \sum_{q_1,\ldots,q_N}\left(\sum_{i_1,j_1} x_{1;q_1,q_2}^{(i_1,j_1)} e_{i_1}e_{j_1}^{\intercal}\right)\otimes\cdots\otimes\left(\sum_{i_N,j_N} x_{N;q_N,q_1}^{(i_N,j_N)} e_{i_N}e_{j_N}^{\intercal}\right)\\ &= \sum_{q_1,\ldots,q_N} \boldsymbol{x_{1;q_1,q_2}}\otimes\cdots\otimes\boldsymbol{x_{N;q_N,q_1}}. \end{aligned} \tag{4.3}$$

The product of $\boldsymbol{A}$ and $\boldsymbol{M}$ is also an MPO given by

$$\begin{aligned}\boldsymbol{AM} &= \sum_{\substack{i_1,\ldots,i_N\\ k_1,\ldots,k_N}} \mathrm{Tr}\left(\boldsymbol{B_1^{(i_1,k_1)}}\cdots\boldsymbol{B_N^{(i_N,k_N)}}\right)\bigotimes_{n=1}^{N} e_{i_n}e_{k_n}^{\intercal}\\ &= \sum_{s_1,\ldots,s_N}\left(\sum_{i_1,k_1} b_{1;s_1,s_2}^{(i_1,k_1)} e_{i_1}e_{k_1}^{\intercal}\right)\otimes\cdots\otimes\left(\sum_{i_N,k_N} b_{N;s_N,s_1}^{(i_N,k_N)} e_{i_N}e_{k_N}^{\intercal}\right)\\ &= \sum_{s_1,\ldots,s_N} \boldsymbol{b_{1;s_1,s_2}}\otimes\cdots\otimes\boldsymbol{b_{N;s_N,s_1}}. \end{aligned} \tag{4.4}$$

where the $\boldsymbol{B_d}$ matrices are given by

$$\boldsymbol{B_d^{(i_d,k_d)}} := \sum_{j_d} \boldsymbol{W_d^{(i_d,j_d)}}\otimes\boldsymbol{X_d^{(j_d,k_d)}}. \tag{4.5}$$

It is evident from the tensor product in (4.5) that the bond dimension of the resulting MPO is larger than that of $\boldsymbol{A}$ and $\boldsymbol{M}$. If matrix $\boldsymbol{W_d^{(i_d,j_d)}}$ has size $D_d\times D_{d+1}$ and matrix $\boldsymbol{X_d^{(j_d,k_d)}}$ has size $L_d\times L_{d+1}$, then $\boldsymbol{B_d^{(i_d,k_d)}}$ has size $(D_dL_d)\times(D_{d+1}L_{d+1})$. The components of the $\boldsymbol{B_d}$ matrices can be written as

$$b_{d;s_d,s_{d+1}}^{(k_d,i_d)} \equiv b_{\substack{d;p_d,p_{d+1}\\ q_d,q_{d+1}}}^{(k_d,i_d)} = \sum_{j_d} w_{d;p_d,p_{d+1}}^{(k_d,j_d)}\cdot x_{d;q_d,q_{d+1}}^{(j_d,i_d)}, \tag{4.6}$$

where the indices are related by

$$s_d := L_d(p_d-1)+q_d, \tag{4.7a}$$

$$s_{d+1} := L_{d+1}(p_{d+1}-1)+q_{d+1}. \tag{4.7b}$$

In the following, we will also need the term $(\boldsymbol{AM})^{\mathrm{H}}$, whose matrix components are given by

$$\bar{b}_{d;r_d,r_{d+1}}^{(k_d,i_d)} \equiv \bar{b}_{\substack{d;m_d,m_{d+1}\\ n_d,n_{d+1}}}^{(k_d,i_d)} = \sum_{j_d'} \bar{w}_{d;m_d,m_{d+1}}^{(k_d,j_d')}\cdot \bar{x}_{d;n_d,n_{d+1}}^{(j_d',i_d)}, \tag{4.8}$$

and the indices are related by

$$r_d := L_d(m_d-1)+n_d, \tag{4.9a}$$

$$r_{d+1} := L_{d+1}(m_{d+1} - 1) + n_{d+1}. \tag{4.9b}$$

From Eq. (4.1), the trace yields

$$\mathrm{Tr}\left((\boldsymbol{AM} - \boldsymbol{I})^{\mathrm{H}}(\boldsymbol{AM} - \boldsymbol{I})\right) = \mathrm{Tr}\left((\boldsymbol{AM})^{\mathrm{H}} \cdot \boldsymbol{AM} - (\boldsymbol{AM})^{\mathrm{H}} \cdot \boldsymbol{I} - \boldsymbol{I} \cdot \boldsymbol{AM} - \boldsymbol{I}\right). \tag{4.10}$$

The first term inside the trace gives

$$(\boldsymbol{AM})^{\mathrm{H}} \cdot (\boldsymbol{AM}) = \sum_{\substack{i,\ell \\ m,n,p,q}} \left(\sum_{k_1} \bar{b}^{(k_1,i_1)}_{1;\substack{m_1,m_2 \\ n_1,n_2}} b^{(k_1,\ell_1)}_{1;\substack{p_1,p_2 \\ q_1,q_2}}\right) \cdots \left(\sum_{k_N} \bar{b}^{(k_N,i_N)}_{N;\substack{m_N,m_1 \\ n_N,n_1}} b^{(k_N,\ell_N)}_{N;\substack{p_N,p_1 \\ q_N,q_1}}\right) \bigotimes_{n=1}^{N} e_{i_n} e_{\ell_n}^{\mathsf{T}}. \tag{4.11}$$

The other three terms inside the trace are trivial, and have the form

$$\begin{aligned}
\boldsymbol{I} &= \sum_{i_1,\ldots,i_N} \bigotimes_{n=1}^{N} e_{i_n} e_{i_n}^{\mathsf{T}}, \\
&= \sum_{\substack{i_1,\ldots,i_N \\ \ell_1,\ldots,\ell_N}} \delta_{i_1,\ell_1} \cdots \delta_{i_N,\ell_N} \bigotimes_{n=1}^{N} e_{i_n} e_{\ell_n}^{\mathsf{T}}. \qquad (4.12) \\
(\boldsymbol{AM})^{\mathrm{H}} \cdot \boldsymbol{I} &= \sum_{\substack{i_1,\ldots,i_N \\ \ell_1,\ldots,\ell_N}} \mathrm{Tr}\left(\bar{\boldsymbol{B}}_{\boldsymbol{1}}^{(\boldsymbol{\ell_1},\boldsymbol{i_1})} \cdots \bar{\boldsymbol{B}}_{\boldsymbol{N}}^{(\boldsymbol{\ell_N},\boldsymbol{i_N})}\right) \bigotimes_{n=1}^{N} e_{i_n} e_{\ell_n}^{\mathsf{T}}, \\
&= \sum_{i,l,m,n} \bar{b}^{(\ell_1,i_1)}_{1;\substack{m_1,m_2 \\ n_1,n_2}} \cdots \bar{b}^{(\ell_N,i_N)}_{N;\substack{m_N,m_1 \\ n_N,n_1}} \bigotimes_{n=1}^{N} e_{i_n} e_{\ell_n}^{\mathsf{T}}. \qquad (4.13) \\
\boldsymbol{I} \cdot (\boldsymbol{AM}) &= \sum_{\substack{i_1,\ldots,i_N \\ \ell_1,\ldots,\ell_N}} \mathrm{Tr}\left(\boldsymbol{B}_{\boldsymbol{1}}^{(\boldsymbol{i_1},\boldsymbol{\ell_1})} \cdots \boldsymbol{B}_{\boldsymbol{N}}^{(\boldsymbol{i_N},\boldsymbol{\ell_N})}\right) \bigotimes_{n=1}^{N} e_{i_n} e_{\ell_n}^{\mathsf{T}}, \\
&= \sum_{i,l,p,q} b^{(i_1,\ell_1)}_{1;\substack{p_1,p_2 \\ q_1,q_2}} \cdots b^{(i_N,\ell_N)}_{N;\substack{p_N,p_1 \\ q_N,q_1}} \bigotimes_{n=1}^{N} e_{i_n} e_{\ell_n}^{\mathsf{T}}. \qquad (4.14)
\end{aligned}$$

Therefore, the terms inside the trace in Eq. (4.10) can be written as

$$\begin{aligned}
(\boldsymbol{AM} - \boldsymbol{I})^{\mathrm{H}}(\boldsymbol{AM} - \boldsymbol{I}) = \Bigg[ & \sum_{i,l,m,n,p,q} \left(\sum_{k_1} \bar{b}^{(k_1,i_1)}_{1;\substack{m_1,m_2 \\ n_1,n_2}} b^{(k_1,\ell_1)}_{1;\substack{p_1,p_2 \\ q_1,q_2}}\right) \cdots \left(\sum_{k_N} \bar{b}^{(k_N,i_N)}_{N;\substack{m_N,m_1 \\ n_N,n_1}} b^{(k_N,\ell_N)}_{N;\substack{p_N,p_1 \\ q_N,q_1}}\right) - \\
& \sum_{i,l,m,n} \left(\bar{b}^{(\ell_1,i_1)}_{1;\substack{m_1,m_2 \\ n_1,n_2}} \cdots \bar{b}^{(\ell_N,i_N)}_{N;\substack{m_N,m_1 \\ n_N,n_1}}\right) + \sum_{i,l,p,q} \left(b^{(i_1,\ell_1)}_{1;\substack{p_1,p_2 \\ q_1,q_2}} \cdots b^{(i_N,\ell_N)}_{N;\substack{p_N,p_1 \\ q_N,q_1}}\right) + \\
& \sum_{i,l} \delta_{i_1,\ell_1} \cdots \delta_{i_N,\ell_N} \Bigg] \bigotimes_{n=1}^{N} e_{i_n} e_{\ell_n}^{\mathsf{T}}. \qquad (4.15)
\end{aligned}$$

Now we need to take the trace of this expression. To see it more clearly, the expression above can be written as

$$(\boldsymbol{AM} - \boldsymbol{I})^{\mathrm{H}}(\boldsymbol{AM} - \boldsymbol{I}) = \sum_{i,\ell} f(i,\ell) \bigotimes_{n=1}^{N} e_{i_n} e_{\ell_n}^{\mathsf{T}}.$$

which exhibits the MPO structure. The diagonal terms are those for which $(i_1, \ldots, i_N) = (\ell_1, \ldots, \ell_N)$. The trace is the sum over these terms, namely

$$\mathrm{Tr}\left( (\boldsymbol{AM} - \boldsymbol{I})^{\mathrm{H}} (\boldsymbol{AM} - \boldsymbol{I}) \right) = \sum_i f(i,i).$$

Having this in mind, the first terms of the trace yields

$$\begin{aligned}\mathrm{Tr}\left( (\boldsymbol{AM})^{\mathrm{H}} \cdot (\boldsymbol{AM}) \right) &= \sum_{m,n,p,q} \left( \sum_{k_1,i_1} \bar{b}^{(k_1,i_1)}_{1;\substack{m_1,m_2\\n_1,n_2}} b^{(k_1,i_1)}_{1;\substack{p_1,p_2\\q_1,q_2}} \right) \cdots \left( \sum_{k_N,i_N} \bar{b}^{(k_N,i_N)}_{N;\substack{m_N,m_1\\n_N,n_1}} b^{(k_N,i_N)}_{N;\substack{p_N,p_1\\q_N,q_1}} \right) - \\ &= \sum_{m,n,p,q} \prod_{\ell=1}^{N} c_{\ell;\substack{m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}\\p_\ell,p_{\ell+1}\\q_\ell,q_{\ell+1}}}, \end{aligned} \tag{4.16}$$

where we have introduced the $\boldsymbol{C_d}$ tensors defined as

$$c_{d;\substack{m_d,m_{d+1}\\n_d,n_{d+1}\\p_d,p_{d+1}\\q_d,q_{d+1}}} := \sum_{k_d,i_d} \bar{b}^{(k_d,i_d)}_{d;\substack{m_d,m_{d+1}\\n_d,n_{d+1}}} b^{(k_d,i_d)}_{d;\substack{p_d,p_{d+1}\\q_d,q_{d+1}}}. \tag{4.17}$$

The second term of the trace reads

$$\begin{aligned}\mathrm{Tr}\left( (\boldsymbol{AM})^{\mathrm{H}} \cdot \boldsymbol{I} \right) &= \sum_{m,n} \sum_{i_1} \bar{b}^{(i_1,i_1)}_{1;\substack{m_1,m_2\\n_1,n_2}} \cdots \sum_{i_N} \bar{b}^{(i_N,i_N)}_{N;\substack{m_N,m_1\\n_N,n_1}} \\ &= \sum_{m,n} \prod_{\ell=1}^{N} \tilde{c}_{\ell;\substack{m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}}}, \end{aligned} \tag{4.18}$$

with $\tilde{\boldsymbol{C}}_{\boldsymbol{d}}$ tensors defined as

$$\tilde{c}_{d;\substack{m_d,m_{d+1}\\n_d,n_{d+1}}} := \sum_{i_d} \bar{b}^{(i_d,i_d)}_{d;\substack{m_d,m_{d+1}\\n_d,n_{d+1}}}. \tag{4.19}$$

In fact, we need only these two terms to attack the minimization problem. Our goal is to find the MPO matrix elements $x^{(j_d,i_d)}_{d;q_d,q_{d+1}}$ (hidden inside the $\boldsymbol{B_d}$ matrices, see Eq. (4.6)) that minimize Eq. (4.10), for which we require

$$\frac{\partial}{\partial \bar{x}^{(j'_d,i_d)}_{d;n_d,n_{d+1}}} \mathrm{Tr}\left( (\boldsymbol{AM} - \boldsymbol{I})^{\mathrm{H}} (\boldsymbol{AM} - \boldsymbol{I}) \right) \overset{!}{=} 0. \tag{4.20}$$

This operation involves only the $\bar{\boldsymbol{B}}_{\boldsymbol{d}}$ tensors, so we need to perform the following derivative:

$$\frac{\partial}{\partial \bar{x}^{(j'_d,i_d)}_{d;n_d,n_{d+1}}} \left( \bar{b}^{(k_d,i_d)}_{d;\substack{m_d,m_{d+1}\\n_d,n_{d+1}}} \right) \overset{(4.8)}{=} \bar{w}^{(k_d,j'_d)}_{d;m_d,m_{d+1}} \tag{4.21}$$

Now, we can compute the derivative of the trace Eq. (4.20). The quadratic term (4.11) yields

$$\frac{\partial\left((\boldsymbol{AM})^{\mathrm{H}}\cdot(\boldsymbol{AM})\right)}{\partial\bar{x}^{(j_d',i_d)}_{d;n_d,n_{d+1}}} = \sum_{\substack{m,p,q\\ n_1,\dots,n_{d-1}\\ n_{d+2},\dots,n_N}} c_{1;\substack{m_1,m_2\\ n_1,n_2\\ p_1,p_2\\ q_1,q_2}}\cdots c_{d-1;\substack{m_{d-1},m_d\\ n_{d-1},n_d\\ p_{d-1},p_d\\ q_{d-1},q_d}}\cdot\left(\sum_{k_d,j_d}\bar{w}^{(k_d,j_d')}_{d;m_d,m_{d+1}}w^{(k_d,j_d)}_{d;p_d,p_{d+1}}x^{(j_d,i_d)}_{d;q_d,q_{d+1}}\right)$$

$$c_{d+1;\substack{m_{d+1},m_{d+2}\\ n_{d+1},n_{d+2}\\ p_{d+1},p_{d+2}\\ q_{d+1},q_{d+2}}}\cdots c_{N;\substack{m_N,m_1\\ n_N,n_1\\ p_N,p_1\\ q_N,q_1}}$$

$$:= \sum_{q_d,q_{d+1}}\sum_{j_d} p^{(j_d',j_d)}_{\substack{d;n_d,n_{d+1}\\ q_d,q_{d+1}}} x^{(j_d,i_d)}_{d;q_d,q_{d+1}}, \tag{4.22}$$

where the $\boldsymbol{P_d}$ tensor is given by

$$p^{(j_d',j_d)}_{\substack{d;n_d,n_{d+1}\\ q_d,q_{d+1}}} := \sum_{\substack{k_d,m,p\\ n_1,\dots,n_{d-1}\\ n_{d+2},\dots,n_N\\ q_1,\dots,q_{d-1}\\ q_{d+2},\dots,q_N}}\prod_{\substack{\ell=1\\ \ell\neq d}}^{N} c_{\ell;\substack{m_\ell,m_{\ell+1}\\ n_\ell,n_{\ell+1}\\ p_\ell,p_{\ell+1}\\ q_\ell,q_{\ell+1}}}\bar{w}^{(k_d,j_d')}_{d;m_d,m_{d+1}}w^{(k_d,j_d)}_{d;p_d,p_{d+1}}. \tag{4.23}$$

The linear term gives

$$\frac{\partial\left((\boldsymbol{AM})^{\mathrm{H}}\cdot\boldsymbol{I}\right)}{\partial\bar{x}^{(j_d',i_d)}_{d;n_d,n_{d+1}}} = \sum_{\substack{m\\ n_1,\dots,n_{d-1}\\ n_{d+2},\dots,n_N}}\left(\sum_{i_1}\bar{b}^{(i_1,i_1)}_{1;\substack{m_1,m_2\\ n_1,n_2}}\right)\cdots\bar{w}^{(i_d,j_d')}_{d;m_d,m_{d+1}}\cdots\left(\sum_{i_N}\bar{b}^{(i_N,i_N)}_{N;\substack{m_N,m_1\\ n_N,n_1}}\right)$$

$$= \sum_{\substack{m\\ n_1,\dots,n_{d-1}\\ n_{d+2},\dots,n_N}}\prod_{\substack{\ell=1\\ \ell\neq d}}^{N}\tilde{c}_{\ell;\substack{m_\ell,m_{\ell+1}\\ n_\ell,n_{\ell+1}}}\bar{w}^{(i_d,j_d')}_{d;m_d,m_{d+1}}$$

$$:= \tilde{p}^{(j_d',i_d)}_{d;n_d,n_{d+1}} \tag{4.24}$$

Thus, it is possible to write down Eq. (4.20) as the linear equation

$$\sum_{q_d,q_{d+1}}\sum_{j_d} p^{(j_d',j_d)}_{\substack{d;n_d,n_{d+1}\\ q_d,q_{d+1}}} x^{(j_d,i_d)}_{d;q_d,q_{d+1}} = \tilde{p}^{(j_d',i_d)}_{d;n_d,n_{d+1}}, \tag{4.25}$$

which has to be solved for all $n_d, n_{d+1}, i_d$ and $j_d'$. This can be rewritten in matrix form as

$$\begin{pmatrix} p^{(0,0)}_{d;(n_d,n_{d+1}),(q_d,q_{d+1})} & p^{(0,1)}_{d;(n_d,n_{d+1}),(q_d,q_{d+1})}\\ p^{(1,0)}_{d;(n_d,n_{d+1}),(q_d,q_{d+1})} & p^{(1,1)}_{d;(n_d,n_{d+1}),(q_d,q_{d+1})}\end{pmatrix}\begin{pmatrix} x^{(0,0)}_{d;q_d,q_{d+1}} & x^{(0,1)}_{d;q_d,q_{d+1}}\\ x^{(1,0)}_{d;q_d,q_{d+1}} & x^{(1,1)}_{d;q_d,q_{d+1}}\end{pmatrix} = \begin{pmatrix}\tilde{p}^{(0,0)}_{d;n_d,n_{d+1}} & \tilde{p}^{(0,1)}_{d;n_d,n_{d+1}}\\ \tilde{p}^{(1,0)}_{d;n_d,n_{d+1}} & \tilde{p}^{(1,1)}_{d;n_d,n_{d+1}}\end{pmatrix}, \tag{4.26}$$

where each matrix $\boldsymbol{P_d^{(j_d,j_d')}}$ has size $(L_dL_{d+1})\times(L_dL_{d+1})$.

The ALS procedure is carried out like analogously to the previous chapter, with the only differnece that, at a given site $d = 1,\dots,N-1$, we solve Eq. (4.25) to obtain now four

matrices (instead of two) that make up $\boldsymbol{X_d}$ and normalize this solution by taking its SVD, namely

$$\begin{pmatrix} \boldsymbol{X_d^{(0,0)}} \\ \boldsymbol{X_d^{(1,0)}} \\ \boldsymbol{X_d^{(0,1)}} \\ \boldsymbol{X_d^{(1,1)}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{U_d^{(0,0)}} \\ \boldsymbol{U_d^{(1,0)}} \\ \boldsymbol{U_d^{(0,1)}} \\ \boldsymbol{U_d^{(1,1)}} \end{pmatrix} \cdot \boldsymbol{\Sigma_d} \cdot \boldsymbol{V_d}, \tag{4.27}$$

where we have ordered the four matrices $\boldsymbol{X_d^{(i_d,j_d)}}$ rowwise into one rectangular matrix. The $\boldsymbol{U_d^{(i_d,j_d)}}$ matrices are unitary and now satisfy the following gauge condition

$$\sum_{i_d,j_d} \boldsymbol{U_d^{(i_d,j_d)H}} \boldsymbol{U_d^{(i_d,j_d)}} = \boldsymbol{I}. \tag{4.28}$$

Now, we substitute our four solution matrices $\boldsymbol{X_d^{(i_d,j_d)}}$ by the unitary matrices $\boldsymbol{U_d^{(i_d,j_d)}}$ and move the remaining factor $\boldsymbol{\Sigma_d} \cdot \boldsymbol{V_d}$ to the right neighbor, that is,

$$\mathrm{Tr}\left( \boldsymbol{X_1^{(i_1,j_1)}} \cdot \boldsymbol{X_2^{(i_2,j_2)}} \cdots \boldsymbol{X_d^{(i_d,j_d)}} \cdot \boldsymbol{X_{d+1}^{(i_{d+1},j_{d+1})}} \cdots \boldsymbol{X_{N-1}^{(i_{N-1},j_{N-1})}} \cdot \boldsymbol{X_N^{(i_N,j_N)}} \right) \rightarrow \tag{4.29}$$

$$\mathrm{Tr}\left( \boldsymbol{X_1^{(i_1,j_1)}} \cdot \boldsymbol{X_2^{(i_2,j_2)}} \cdots \boldsymbol{U_d^{(i_d,j_d)}} \cdot \underbrace{(\boldsymbol{\Sigma_d} \cdot \boldsymbol{V_d}) \cdot \boldsymbol{X_{d+1}^{(i_{d+1},j_{d+1})}}}_{\rightarrow \boldsymbol{X_{d+1}^{(i_{d+1},j_{d+1})}}} \cdots \boldsymbol{X_{N-1}^{(i_{N-1},j_{N-1})}} \cdot \boldsymbol{X_N^{(i_N,j_N)}} \right). \tag{4.30}$$

Once the matrices at site $d$ are substituted and those at site $d+1$ updated, we repeat the procedure at site $d+1$ until reaching site $N-1$. At this point, all sites except $d = N$ will be normalized, and a right-to-left sweep can be performed analogously. To this end, we take

$$\begin{pmatrix} \boldsymbol{X_d^{(0,0)}} & \boldsymbol{X_d^{(1,0)}} & \boldsymbol{X_d^{(0,1)}} & \boldsymbol{X_d^{(1,1)}} \end{pmatrix} = (\boldsymbol{U_d} \cdot \boldsymbol{\Sigma_d}) \cdot \begin{pmatrix} \boldsymbol{V_d^{(0,0)}} & \boldsymbol{V_d^{(1,0)}} & \boldsymbol{V_d^{(0,1)}} & \boldsymbol{V_d^{(1,1)}} \end{pmatrix}, \tag{4.31}$$

substitute the matrices $\boldsymbol{X_d^{(i_d,j_d)}}$ with the unitary matrices $\boldsymbol{V_d^{(i_d,j_d)}}$ and move the remaining factor $\boldsymbol{U_d} \cdot \boldsymbol{\Sigma_d}$ to the left neighbor, for $d = N, \ldots, 2$.

## 4.3 Numerical considerations

We now follow the same reasoning as in Chapter 3, section 3.3 and define the left ($\boldsymbol{F_d}$) and right ($\boldsymbol{G_d}$) factors of the $\boldsymbol{P_d}$ tensor as

$$f_{\substack{d;m_1,m_d \\ n_1,n_d \\ p_1,p_d \\ q_1,q_d}} := \sum_{\substack{m_2,\ldots,m_{d-1} \\ n_2,\ldots,n_{d-1} \\ p_2,\ldots,p_{d-1} \\ q_2,\ldots,q_{d-1}}} \prod_{\ell=1}^{d-1} c_{\ell;\substack{m_\ell,m_{\ell+1} \\ n_\ell,n_{\ell+1} \\ p_\ell,p_{\ell+1} \\ q_\ell,q_{\ell+1}}}, \tag{4.32}$$

$$g_{\substack{d;m_1,m_{d+1} \\ n_1,n_{d+1} \\ p_1,p_{d+1} \\ q_1,q_{d+1}}} := \sum_{\substack{m_{d+2},\ldots,m_N \\ n_{d+2},\ldots,n_N \\ p_{d+2},\ldots,p_N \\ q_{d+2},\ldots,q_N}} \prod_{\ell=d+1}^{N} c_{\ell;\substack{m_\ell,m_{\ell+1} \\ n_\ell,n_{\ell+1} \\ p_\ell,p_{\ell+1} \\ q_\ell,q_{\ell+1}}}, \tag{4.33}$$

such that the $\boldsymbol{P_d}$ tensor can be written as

$$p^{(j'_d,j_d)}_{\substack{d;n_d,n_{d+1}\\q_d,q_{d+1}}} = \sum_{\substack{k_d\\m_1,n_1,p_1,q_1\\m_d,m_{d+1}\\p_d,p_{d+1}}} f_{\substack{d;m_1,m_d\\n_1,n_d\\p_1,p_d\\q_1,q_d}} \bar{w}^{(k_d,j'_d)}_{d;m_d,m_{d+1}} w^{(k_d,j_d)}_{d;p_d,p_{d+1}} g_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}\\p_1,p_{d+1}\\q_1,q_{d+1}}}. \tag{4.34}$$

Likewise, for $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ we have,

$$\tilde{f}_{\substack{d;m_1,m_d\\n_1,n_d}} := \sum_{\substack{m_2,\ldots,m_{d-1}\\n_2,\ldots,n_{d-1}}} \prod_{\ell=1}^{d-1} \tilde{c}_{\substack{\ell;m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}}}, \tag{4.35}$$

$$\tilde{g}_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}}} := \sum_{\substack{m_{d+2},\ldots,m_N\\n_{d+2},\ldots,n_N}} \prod_{\ell=d+1}^{N} \tilde{c}_{\substack{\ell;m_\ell,m_{\ell+1}\\n_\ell,n_{\ell+1}}}, \tag{4.36}$$

and write the $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ tensor as

$$\tilde{p}^{(j'_d,i_d)}_{d;n_d,n_{d+1}} = \sum_{\substack{m_1,n_1\\m_d,m_{d+1}}} \tilde{f}_{\substack{d;m_1,m_d\\n_1,n_d}} \bar{w}^{(i_d,j'_d)}_{d;m_d,m_{d+1}} \tilde{g}_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}}}. \tag{4.37}$$

The key idea of writing the $\boldsymbol{P_d}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ tensors with left and right factors is that we can easily compute the new left and right factors when moving from site $d$ to site $d+1$ after having optimized site $d$ (or from site $d$ to site $d-1$ in a right-to-left sweep). Like in the previous chapter, the recurrence relations are the following:

$$f_{\substack{d+1;m_1,m_{d+1}\\n_1,n_{d+1}\\p_1,p_{d+1}\\q_1,q_{d+1}}} = \sum_{\substack{m_d,n_d\\p_d,q_d}} f_{\substack{d;m_1,m_d\\n_1,n_d\\p_1,p_d\\q_1,q_d}} c_{\substack{d;m_d,m_{d+1}\\n_d,n_{d+1}\\p_d,p_{d+1}\\q_d,q_{d+1}}}, \tag{4.38}$$

$$g_{\substack{d-1;m_1,m_d\\n_1,n_d\\p_1,p_d\\q_1,q_d}} = \sum_{\substack{m_{d+1},n_{d+1}\\p_{d+1},q_{d+1}}} c_{\substack{d;m_d,m_{d+1}\\n_d,n_{d+1}\\p_d,p_{d+1}\\q_d,q_{d+1}}} g_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}\\p_1,p_{d+1}\\q_1,q_{d+1}}}, \tag{4.39}$$

$$\tilde{f}_{\substack{d+1;m_1,m_{d+1}\\n_1,n_{d+1}}} = \sum_{m_d,n_d} \tilde{f}_{\substack{d;m_1,m_d\\n_1,n_d}} \tilde{c}_{\substack{d;m_d,m_{d+1}\\n_d,n_{d+1}}}, \tag{4.40}$$

$$\tilde{g}_{\substack{d-1;m_1,m_d\\n_1,n_d}} = \sum_{m_{d+1},n_{d+1}} \tilde{c}_{\substack{d;m_d,m_{d+1}\\n_d,n_{d+1}}} \tilde{g}_{\substack{d;m_1,m_{d+1}\\n_1,n_{d+1}}}. \tag{4.41}$$

Let us now summarize our discussion and present the final algorithm. The key idea is to find the small matrices $\boldsymbol{X_d}$ that make up the MPO $\boldsymbol{M}$, so that the algorithm's complexity depends on the dimensions $L_d \times L_{d+1}$ of these small matrices, rather than on the dimensions of $\boldsymbol{M}$ itself. Equation (4.25) (or, equivalently, (4.26)) tells us how to solve for each of the $\boldsymbol{X_d}$ matrices componentwise, for which the $\boldsymbol{P_d}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ tensors have to be calculated. These tensors give us the contractions relative to all other sites (that are not being minimized), and defining the recurrence relations for the left and right factors of these two tensors allows us to reuse previous contractions to perform the calculations more efficiently.

Algorithm 2 describes the steps to follow in order to obtain the approximate inverse MPO $\boldsymbol{M}$ of an MPO $\boldsymbol{A}$.

**Algorithm 2:** ALS for the Approximate Inverse of an MPO

**input** : $\boldsymbol{A} = \boldsymbol{A}[\boldsymbol{W_1}, \ldots, \boldsymbol{W_N}] \in \mathrm{MPO}_D$, MPO for which we want the inverse.
**output**: $\boldsymbol{M} = \boldsymbol{M}[\boldsymbol{X_1}, \ldots, \boldsymbol{X_N}] \in \mathrm{MPO}_L$, approximate inverse of $\boldsymbol{A}$.

1 generate initial guess for $\boldsymbol{M} \in \mathrm{MPO}_L$ ;
2 pre-compute **all** right tensors $\boldsymbol{G_d}$, $\tilde{\boldsymbol{G}}_{\boldsymbol{d}}$ for $d = 1, \ldots, N-1$, with $\boldsymbol{G_N} = \tilde{\boldsymbol{G}}_{\boldsymbol{N}} = 1$ using Eqs. (4.33) and (4.36);
3 initialize $\boldsymbol{F_1} = \tilde{\boldsymbol{F}}_{\boldsymbol{1}} = 1$;
4 **while** *not converged* **do**
5  **for** $d = 1$ **to** $N-1$ **do** `// left-to-right sweep`
6  calculate $\boldsymbol{P_d}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ using Eqs. (4.34) and (4.37);
7  solve for $\boldsymbol{X_d}$, the solution matrices at site $d$, Eq. (4.25);
8  compute the SVD of $\boldsymbol{X}_{\boldsymbol{d}}^{(j_d,i_d)} = \boldsymbol{U}_{\boldsymbol{d}}^{(j_d,i_d)} \boldsymbol{\Sigma_d} \boldsymbol{V_d}$;
9  substitute $\boldsymbol{X}_{\boldsymbol{d}}^{(j_d,i_d)} \leftarrow \boldsymbol{U}_{\boldsymbol{d}}^{(j_d,i_d)}$ and $\boldsymbol{X}_{\boldsymbol{d+1}}^{(j_{d+1},i_{d+1})} \leftarrow \boldsymbol{\Sigma_d} \boldsymbol{V_d} \boldsymbol{X}_{\boldsymbol{d+1}}^{(j_{d+1},i_{d+1})}$;
10  calculate $\boldsymbol{F_{d+1}}$ and $\tilde{\boldsymbol{F}}_{\boldsymbol{d+1}}$ using Eqs. (4.38) and (4.40) ;
11  **end**
12  **for** $d = N$ **to** $2$ **do** `// right-to-left sweep`
13  calculate $\boldsymbol{P_d}$ and $\tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ using Eqs. (4.34) and (4.37);
14  solve for $\boldsymbol{X_d}$, the solution matrices at site $d$, Eq. (4.25);
15  compute the SVD of $\boldsymbol{X}_{\boldsymbol{d}}^{(j_d,i_d)} = \boldsymbol{U_d} \boldsymbol{\Sigma_d} \boldsymbol{V}_{\boldsymbol{d}}^{(j_d,i_d)}$;
16  substitute $\boldsymbol{X}_{\boldsymbol{d}}^{(j_d,i_d)} \leftarrow \boldsymbol{V}_{\boldsymbol{d}}^{(j_d,i_d)}$ and $\boldsymbol{X}_{\boldsymbol{d-1}}^{(j_{d-1},i_{d-1})} \leftarrow \boldsymbol{X}_{\boldsymbol{d-1}}^{(j_{d-1},i_{d-1})} \boldsymbol{V_d} \boldsymbol{\Sigma_d}$;
17  calculate $\boldsymbol{G_{d-1}}$ and $\tilde{\boldsymbol{G}}_{\boldsymbol{d-1}}$ using Eqs. (4.39) and (4.41);
18  **end**
19 **end**

### 4.3.1 Computational complexity

Analogously to the discussion in Section 3.3.1, we can derive the computational costs of Algorithm 2, starting with the PBC case. Again, we follow the efficient contraction schemes described in [12].

The complexity of solving the linear system $\boldsymbol{P_d} \boldsymbol{X_d} = \tilde{\boldsymbol{P}}_{\boldsymbol{d}}$ depends on the solver used. The system matrix derived from the tensor $\boldsymbol{P_d}$ in Eq. (4.26) has size $2L'_d L'_{d+1} \times 2L_d L_{d+1}$, so the complexity of using a direct solver is the same as in the previous chapter, namely $\mathcal{O}(8NL^6)$ for the whole sweep. Taking the SVD of the solution matrices $\boldsymbol{X_d}$ requires $\mathcal{O}(4L^3)$ operations, as well as multiplying $\boldsymbol{\Sigma_d} \boldsymbol{V_d}$ times the right neighbor $\boldsymbol{X_{d+1}}$. Updating the left factors $\boldsymbol{F_{d+1}}$ and $\tilde{\boldsymbol{F}}_{\boldsymbol{d+1}}$ in line 10 requires $\mathcal{O}(8L^5D^4)$ and $\mathcal{O}(8L^3D^2)$ operations. Therefore, the cost of a left-to-right sweep is approximately $\mathcal{O}(8NL^5D^4)$, which is also the complexity of a whole ALS iteration, since the right-to-left sweep is equally expensive. In total, the complexity of our algorithm is $\mathcal{O}(8NL^5D^4 + 8NL^6)$.

The OBC case is less expensive since the first and last indices are dummy. This means that the calculation of $\boldsymbol{F_{d+1}}$ and $\tilde{\boldsymbol{F}}_{\boldsymbol{d+1}}$ reduces to $\mathcal{O}(8L^3D^2)$ and $\mathcal{O}(8L^2D)$ operations, respectively, thus reducing the overall complexity of an ALS sweep to approximately $\mathcal{O}(8NL^3D^2)$. Including the solution of the linear systems, we obtain a final complexity

of $\mathcal{O}(8NL^3D^2 + 8NL^6)$.

### 4.3.2 Initial guess for $M$

An appropriate choice of the initial guess for $\boldsymbol{M}$ might improve the convergence speed of the algorithm. Since we are looking for an MPO that is the inverse of $\boldsymbol{A}$, a first idea would be to take the inverse of the diagonal elements of $\boldsymbol{A}$, which is an MPO with components

$$(\boldsymbol{D_A})^{-1}_{\substack{i_1,\dots,i_N \\ i_1,\dots,i_N}} = \left( \sum_{p_1,\dots,p_N} w^{(i_1,i_1)}_{1;p_1,p_2} \cdots w^{(i_N,i_N)}_{N;p_N,p_1} \right)^{-1}, \tag{4.42}$$

where we denote $\boldsymbol{D_A}$ the diagonal part of $\boldsymbol{A}$. Therefore, we'd like our initial guess $\boldsymbol{M^{(0)}}$ to have the above components, namely

$$\sum_{q_1,\dots,q_N} x^{(i_1,i_1)}_{1;q_1,q_2} \cdots x^{(i_N,i_N)}_{N;q_N,q_1} \overset{!}{=} \frac{1}{\displaystyle\sum_{p_1,\dots,p_N} w^{(i_1,i_1)}_{1;p_1,p_2} \cdots w^{(i_N,i_N)}_{N;p_N,p_1}}. \tag{4.43}$$

One way to obtain such MPO is to use Algorithm 2 with $\boldsymbol{A} \equiv \boldsymbol{D_A}$, and solve

$$\min_{\boldsymbol{M} \in \mathrm{MPO}_L} \|\boldsymbol{D_A M^{(0)}} - \boldsymbol{I}\|_F \tag{4.44}$$

The algorithm turns out to be very efficient in this case, so only a couple iterations are necessary. This can then be used in line 1 of the algorithm as initial guess.

## 4.4 Two-site optimization: DMRG

One can follow the same reasoning of Section 3.4 and optimize two sites simultaneously in order for the algorithm to be able to determine the dimensions of $\boldsymbol{X_d^{(j_d,i_d)}}$ adaptively. This time, the solution should be reshaped as follows:

$$\begin{pmatrix} \boldsymbol{X_d^{(0,0)}} \\ \boldsymbol{X_d^{(1,0)}} \\ \boldsymbol{X_d^{(0,1)}} \\ \boldsymbol{X_d^{(1,1)}} \end{pmatrix} \begin{pmatrix} \boldsymbol{X_{d+1}^{(0,0)}} & \boldsymbol{X_{d+1}^{(1,0)}} & \boldsymbol{X_{d+1}^{(0,1)}} & \boldsymbol{X_{d+1}^{(1,1)}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X_d^{(0,0)} X_{d+1}^{(0,0)}} & \cdots & \boldsymbol{X_d^{(0,0)} X_{d+1}^{(1,1)}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{X_d^{(1,1)} X_{d+1}^{(0,0)}} & \cdots & \boldsymbol{X_d^{(1,1)} X_{d+1}^{(1,1)}} \end{pmatrix}$$

$$= \begin{pmatrix} \boldsymbol{U_d^{(0,0)}} \\ \boldsymbol{U_d^{(1,0)}} \\ \boldsymbol{U_d^{(0,1)}} \\ \boldsymbol{U_d^{(1,1)}} \end{pmatrix} \boldsymbol{\Sigma_d V_d}. \tag{4.45}$$

## 4.5 Results

The ALS algorithm 2 and its DMRG variant were implemented on MATLAB. A similar implementation [20] proposes to solve

$$\boldsymbol{A}\boldsymbol{M} - \boldsymbol{M}\boldsymbol{A} = 2\boldsymbol{I}, \tag{4.46}$$

instead of $\boldsymbol{A}\boldsymbol{M} = \boldsymbol{I}$, arguing that the solution of Eq. (4.46) is symmetric, whereas that of $\boldsymbol{A}\boldsymbol{M} = \boldsymbol{I}$ might not.

### 4.5.1 Heat Equation

As a first example, the operator $\boldsymbol{I} - \alpha\Delta_2$ (whith results from discretizing the 2D Heat Equation with an implicit Euler scheme) with $\alpha = 10^{-3}$, is inverted, and a grid with $2^{10}$ points per dimension is used.[1] Due to the quadratic complexity of our algorithm in the bond dimension of $\boldsymbol{A}$, we were forced to limit the maximal bond dimension of the inverse $\boldsymbol{M}$ to $L = 20$ (the permute operations in the calculation of the left and right factors $\boldsymbol{F_d}$ and $\boldsymbol{G_d}$ seem to be very expensive).

Fig. and Table 4.1 show the resulting convergence after 5 iterations (after which it stalls, having reached the upper limit for the bond dimension $L = 20$). Although there exist no results concerning the convergence of DMRG, we can observe a linear tendency (of order $\sim 0.85$). The inverse $\boldsymbol{M}$ was calculated with an accuracy of $\varepsilon = 2.37 \times 10^{-2}$ (relative residual), and the solution obtained seems to preserve the symmetry of $\boldsymbol{A^{-1}}$, since the left residual $\|\boldsymbol{M}\boldsymbol{A} - \boldsymbol{I}\|/\|\boldsymbol{I}\| = 2.37 \times 10^{-2}$ and the right residual $\|\boldsymbol{A}\boldsymbol{M} - \boldsymbol{I}\|/\|\boldsymbol{I}\| = 2.95 \times 10^{-2}$ are of the same order, and the error in symmetry $\|\boldsymbol{M} - \boldsymbol{M}^\intercal\|/\|\boldsymbol{M}\| = 4.23 \times 10^{-3}$ is smaller than $\varepsilon$.[2]

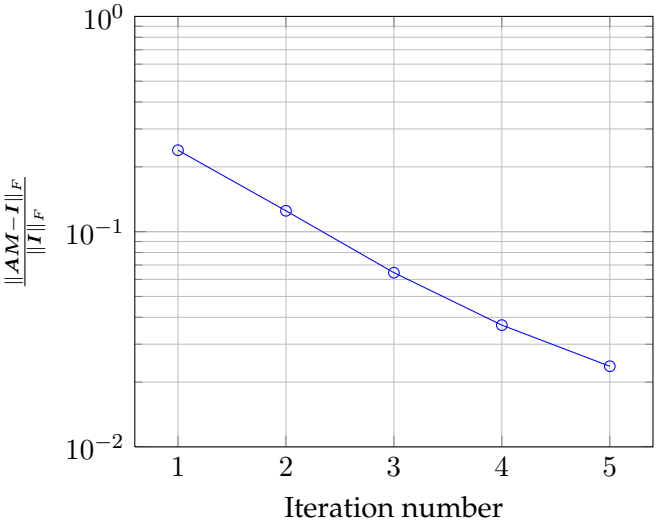



Figure 4.1: Convergence of the inverse of $\boldsymbol{I} - \alpha\Delta_2$ for $N = 10$.

| iter. | time current iter. (s) | $L$ |
|---|---|---|
| 1 | 0.365 | 5 |
| 2 | 2.26 | 12 |
| 3 | 14.3 | 20 |
| 4 | 36.4 | 20 |
| 5 | 50.3 | 20 |

Table 4.1: Results

[1] Storing such matrix would require $2^{40}$ cells, or $\mathcal{O}(2^{20})$ in the sparse case.
[2] These indicators are used in [20].

### 4.5.2 Random matrix

Now we generate a random OBC-MPO by defining the core matrices as

$$\boldsymbol{W}_d^{(i_d)} = \texttt{rand(D,D,2,2)}.$$

(taking care of the boundary condition $D_1 = D_{N+1} = 1$). We then build our system matrix $\boldsymbol{A}_s$ to be symmetric and diagonally dominant as $\boldsymbol{A}_s \leftarrow \frac{1}{2}(\boldsymbol{A} + \boldsymbol{A}^\intercal) + \boldsymbol{I}$. We choose the bond dimension of $A$ to be $D = 2$, so the system matrix $\boldsymbol{A}_s$ has bond dimension $D_s = 5$. The results below illustrate what happens after just one DMRG iteration while increasing $N = 2, \ldots, 20$. Since the resulting system matrix is dense, it would require in each case $2^{2N}$ cells just to store the matrix. In the MATLAB implementation used, the maximum size possible to store a matrix with zeros is $N = 14$.

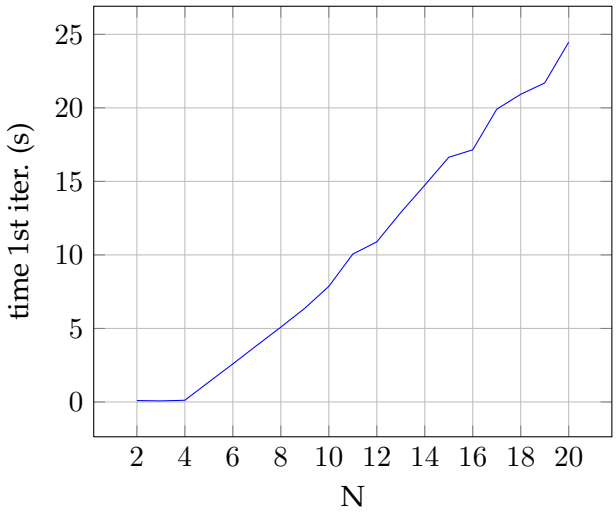



Figure 4.2: Time required to perform the first DMRG iteration of the random matrix $\boldsymbol{A}_s$

| $N$ | $\frac{\|\boldsymbol{AM}-\boldsymbol{I}\|_F}{\|\boldsymbol{I}\|_F}$ | $N$ | $\frac{\|\boldsymbol{AM}-\boldsymbol{I}\|_F}{\|\boldsymbol{I}\|_F}$ |
|---|---|---|---|
| 2 | 1.85e-08 | 12 | 9.18e-05 |
| 3 | 2.12e-08 | 13 | 6.32e-05 |
| 4 | 2.44e-08 | 14 | 1.11e-04 |
| 5 | 3.25e-08 | 15 | 4.08e-05 |
| 6 | 1.85e-04 | 16 | 5.73e-05 |
| 7 | 2.60e-04 | 17 | 3.81e-05 |
| 8 | 7.44e-04 | 18 | 2.82e-05 |
| 9 | 2.58e-04 | 19 | 1.68e-05 |
| 10 | 1.65e-04 | 20 | 1.67e-05 |
| 11 | 1.69e-04 | | |

Table 4.2: Relative residual after one iteration.

We observe from Table 4.2 that after only one iteration, the inverse is very well approximated. The resulting bond dimension of $\boldsymbol{M}$ is 4 when $N = 2, 3$, and 16 when $N = 4, \ldots, 20$. Under this observation it is clear why Fig. 4.2 is approximately linear. The resulting right and left residuals were monitored and resulted in each case of the same order of magnitude.

Additionally, several runs were performed on purely random MPO $\boldsymbol{A}$ (non-symmetric and non-diagonally dominant) and DMRG was observed to converge in many cases, a fact that might be worth investigating.

## 4.6 Conclusions

We have presented an algorithm to approximate the inverse of a matrix $\boldsymbol{A}$ given that it can be represented in MPO format. The complexity of the algorithm requires the bond dimensions of the resulting inverse $\boldsymbol{M}$ to remain moderate, otherwise one runs into memory problems. This is usually not the case, since even matrices with special structures (such as banded Toeplitz) have inverses that require high bond dimensions [23, 34]. This is a reason not to calculate the inverse $\boldsymbol{M}$ for the solution of linear equations, despite the fact that the algorithm has a complexity only slightly higher than DMRG from Chapter (3). Nonetheless, DMRG in this context can provide very good approximate inverses with small bond dimensions, a fact that we will exploit in the next chapter to generate preconditioners.

# 5 Relaxation schemes and preconditioning in MPS/MPO format

## 5.1 Iterative methods

We now turn our attention to a family of methods that seek a solution of the linear system of equations[1]

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

iteratively, where each iterate has the form

$$\boldsymbol{x}^{(x+1)} = \boldsymbol{B}\boldsymbol{x}^{(x)} + \boldsymbol{g}, \quad k \geq 0, \tag{5.1}$$

where $\boldsymbol{B}$ is a matrix depending on $\boldsymbol{A}$, and $\boldsymbol{g} = (\boldsymbol{I} - \boldsymbol{B})\boldsymbol{A}^{-1}\boldsymbol{b}$. The iterative scheme will then be convergent provided that the spectral radius of $\boldsymbol{B}$ is bounded by one, $\rho(\boldsymbol{B}) < 1$.

One can build an iterative scheme by choosing an adequate *splitting* of $\boldsymbol{A}$, namely, rewriting $\boldsymbol{A} = \boldsymbol{M} - (\boldsymbol{M} - \boldsymbol{A})$. Then, the linear system of equations reads

$$\boldsymbol{M}\boldsymbol{x} = (\boldsymbol{M} - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b},$$

which has the form (5.1) provided that $\boldsymbol{B} = \boldsymbol{M}^{-1}(\boldsymbol{M} - \boldsymbol{A}) = \boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}$ and $\boldsymbol{g} = \boldsymbol{M}^{-1}\boldsymbol{A}$. We want to investigate how such schemes look when $\boldsymbol{A}$ is given as an MPO,

$$\boldsymbol{A} = \sum_{\substack{i_1,\dots,i_N \\ j_1,\dots,j_N}} \mathrm{Tr}\left(\prod_{d=1}^{N} \boldsymbol{W}_{\boldsymbol{d}}^{(\boldsymbol{i_d},\boldsymbol{j_d})}\right) \bigotimes_{n=1}^{N} e_{i_n} e_{j_n}^{\intercal} = \sum_{p_1,\dots,p_N} \bigotimes_{d=1}^{N} \boldsymbol{w}_{\boldsymbol{d};\boldsymbol{p_d},\boldsymbol{p_{d+1}}}, \tag{5.2}$$

with

$$\boldsymbol{w}_{\boldsymbol{d};\boldsymbol{p_d},\boldsymbol{p_{d+1}}} = \begin{pmatrix} w_{1;p_1,p_2}^{(0,0)} & w_{1;p_1,p_2}^{(0,1)} \\ w_{1;p_1,p_2}^{(1,0)} & w_{1;p_1,p_2}^{(1,1)} \end{pmatrix}. \tag{5.3}$$

and $\boldsymbol{x}$ and $\boldsymbol{b}$ are given as MPS.

## 5.2 Jacobi and Gauss-Seidel iterations

Choosing $\boldsymbol{M} := \boldsymbol{D}$, where $\boldsymbol{D}$ is the diagonal part of $\boldsymbol{A}$, gives a *Jacobi* iteration. Rewriting it in the form (5.1) gives

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \boldsymbol{D}^{-1}(\boldsymbol{D} - \boldsymbol{A})\boldsymbol{x}^{(k)} + \boldsymbol{D}^{-1}\boldsymbol{b}^{(k)} \\ &= \boldsymbol{x}^{(k)} + \boldsymbol{D}^{-1}\boldsymbol{r}^{(k)}, \end{aligned} \tag{5.4}$$

[1] We follow [28] for this discussion.

with residual $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}$. The diagonal part of an MPO can be written down easily, setting $i_d = j_d$ for all $d$,

$$(\boldsymbol{D})_{\substack{i_1,\ldots,i_N \\ i_1,\ldots,i_N}} = \sum_{p_1,\ldots,p_N} w_{1;p_1,p_2}^{(i_1,i_1)} \cdots w_{N;p_N,p_1}^{(i_N,i_N)}. \tag{5.5}$$

There are several ways to implement Eq. (5.4) numerically in the context of MPS/MPO. One can either

1. use the expression for $\boldsymbol{D}$ (Eq. (5.5)), and use Algorithm 2 to calculate its inverse (or an approximation). Then, carry out the product $\boldsymbol{g^{(k)}} := \boldsymbol{D}^{-1}\boldsymbol{r}^{(k)}$ and the sum $\boldsymbol{x}^{(k)} + \boldsymbol{g}^{(k)}$. After doing so, compress the result $\boldsymbol{x}^{(k+1)}$ to a certain bond dimension $L$; or

2. reformulate Eq. (5.4) as a minimization problem of the form

$$\min_{\boldsymbol{x}^{(k+1)} \in \mathrm{MPS}_L} \|\boldsymbol{D}\boldsymbol{x}^{(k+1)} - \boldsymbol{D}\boldsymbol{x}^{(k)} - \boldsymbol{r}^{(k)}\|_2^2, \tag{5.6}$$

   for which Algorithm 1 can be applied, using $\boldsymbol{A} \equiv \boldsymbol{D}$ and $\boldsymbol{b} \equiv \boldsymbol{D}\boldsymbol{x}^{(k)} + \boldsymbol{r}^{(k)}$.

Which approach is preferable? If $\boldsymbol{D}^{-1}$ can be calculated exactly in MPO format and has a small bond dimension, then the first approach is better. If calculating $\boldsymbol{D}^{-1}$ is expensive, the second approach might be advisable. In general, we have observed that the calculation of $\boldsymbol{D}^{-1}$ is fast and accurate. We will return to this point later.

A *Gauss-Seidel* iteration is obtained by setting $\boldsymbol{M} := \boldsymbol{D} - \boldsymbol{L}$, where $-\boldsymbol{L}$ is the strictly lower triangular part of $\boldsymbol{A}$. This gives

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + (\boldsymbol{D} - \boldsymbol{L})^{-1}\boldsymbol{r}^{(k)}. \tag{5.7}$$

In order to perform a Gauss-Seidel iteration, we first need an expression for $\boldsymbol{D} - \boldsymbol{L}$ in MPO format. To do this, we start with expression (5.2) and decompose each matrix $\boldsymbol{w_{d;p_d,p_{d+1}}}$ as a sum of the diagonal part, the strictly lower triangular part and the strictly upper triangular part, namely

$$\boldsymbol{w_{d;p_d,p_{d+1}}} = \boldsymbol{d_{d;p_d,p_{d+1}}} - (\boldsymbol{\ell_{d;p_d,p_{d+1}}} + \boldsymbol{u_{d;p_d,p_{d+1}}}), \tag{5.8}$$

where

$$\boldsymbol{d_{d;p_d,p_{d+1}}} := \begin{pmatrix} w_{1;p_1,p_2}^{(0,0)} & 0 \\ 0 & w_{1;p_1,p_2}^{(1,1)} \end{pmatrix}, \tag{5.9}$$

$$\boldsymbol{\ell_{d;p_d,p_{d+1}}} := -\begin{pmatrix} 0 & 0 \\ w_{1;p_1,p_2}^{(1,0)} & 0 \end{pmatrix}, \tag{5.10}$$

$$\boldsymbol{u_{d;p_d,p_{d+1}}} := -\begin{pmatrix} 0 & w_{1;p_1,p_2}^{(0,1)} \\ 0 & 0 \end{pmatrix}. \tag{5.11}$$

Inserting this into the definition of the MPO $\boldsymbol{A}$, we have

$$\boldsymbol{A} = \sum_{p_1,\ldots,p_N} \boldsymbol{w_{1;p_1,p_2}} \otimes \cdots \otimes \boldsymbol{w_{N;p_N,p_1}} \tag{5.12}$$

$$= \sum_{p_1,\ldots,p_N} (\boldsymbol{d_{1;p_1,p_2}} - (\boldsymbol{\ell_{1;p_1,p_2}} + \boldsymbol{u_{1;p_1,p_2}})) \otimes \cdots \otimes (\boldsymbol{d_{N;p_N,p_1}} - (\boldsymbol{\ell_{N;p_N,p_1}} + \boldsymbol{u_{N;p_N,p_1}})) \,. \tag{5.13}$$

From the above $3^N$ terms (when carrying out the tensor products of each trinomial), only some of them contribute to the lower part of $\boldsymbol{A}$. For example, the term

$$\boldsymbol{d_{1;p_1,p_2}} \otimes \boldsymbol{d_{2;p_2,p_3}} \otimes \ldots \otimes \boldsymbol{d_{N-1;p_{N-1},p_N}} \otimes \boldsymbol{d_{N;p_N,p_1}} \tag{5.14}$$

gives the diagonal part of $\boldsymbol{A}$. Inspecting the rest of the addends, one can observe that the terms corresponding to the lower part of $\boldsymbol{A}$ are

$$\begin{aligned}
&(-\boldsymbol{\ell_{1;p_1,p_2}}) \otimes \boldsymbol{w_{2;p_2,p_3}} \otimes \boldsymbol{w_{3;p_3,p_4}} \otimes \boldsymbol{w_{4;p_4,p_5}} \otimes \ldots \otimes \boldsymbol{w_{N-1;p_{N-1},p_N}} \otimes \boldsymbol{w_{N;p_N,p_1}} \\
&+ \boldsymbol{d_{1;p_1,p_2}} \otimes (-\boldsymbol{\ell_{2;p_2,p_3}}) \otimes \boldsymbol{w_{3;p_3,p_4}} \otimes \boldsymbol{w_{4;p_4,p_5}} \otimes \ldots \otimes \boldsymbol{w_{N-1;p_{N-1},p_N}} \otimes \boldsymbol{w_{N;p_N,p_1}} \\
&+ \boldsymbol{d_{1;p_1,p_2}} \otimes \boldsymbol{d_{2;p_2,p_3}} \otimes (-\boldsymbol{\ell_{3;p_3,p_4}}) \otimes \boldsymbol{w_{4;p_4,p_5}} \otimes \ldots \otimes \boldsymbol{w_{N-1;p_{N-1},p_N}} \otimes \boldsymbol{w_{N;p_N,p_1}} \\
&\vdots \\
&+ \boldsymbol{d_{1;p_1,p_2}} \otimes \boldsymbol{d_{2;p_2,p_3}} \otimes \boldsymbol{d_{3;p_3,p_4}} \otimes \boldsymbol{d_{4;p_4,p_5}} \otimes \ldots \otimes \boldsymbol{d_{N-1;p_{N-1},p_N}} \otimes (-\boldsymbol{\ell_{N;p_N,p_1}}),
\end{aligned} \tag{5.15}$$

or, written more compactly

$$\boldsymbol{L} = \sum_{i=1}^{N} \sum_{p_1,\ldots,p_N} \underbrace{\left[\bigotimes_{j=1}^{i-1} \boldsymbol{d_{j;p_j,p_{j+1}}}\right]}_{\text{diagonal}} \underbrace{\otimes \boldsymbol{\ell_{i;p_i,p_{i+1}}}}_{\text{lower triangular}} \underbrace{\left[\bigotimes_{j=i+1}^{N} \boldsymbol{w_{j;p_j,p_{j+1}}}\right]}_{\text{arbitrary}} . \tag{5.16}$$

The motivation behind this expression is as follows. Take, for example, the first addend, $i = 1$. The resulting MPO is a tensor product of $\boldsymbol{\ell_1}$ with the rest of the $\boldsymbol{w_j}$ matrices ($j = 2, \cdots, N$). No matter what these $\boldsymbol{w_j}$ matrices look like, the tensor product with $\boldsymbol{\ell_1}$ gives a lower triangular matrix. The rest of the addends can be reasoned analogously: there are diagonal matrices $\boldsymbol{d_j}$ multiplied to the left, followed by a single $\boldsymbol{\ell_i}$ matrix whose tensor product with the $\boldsymbol{w_j}$ matrices to its right gives a lower triangular matrix. Therefore, we can write

$$\boldsymbol{D} - \boldsymbol{L} = \sum_{i=1}^{N+1} \sum_{p_1,\ldots,p_N} \left[\bigotimes_{j=1}^{i-1} \boldsymbol{d_{j;p_j,p_{j+1}}}\right] \otimes (-\boldsymbol{\ell_{i;p_i,p_{i+1}}}) \left[\bigotimes_{j=i+1}^{N} \boldsymbol{w_{j;p_j,p_{j+1}}}\right] , \tag{5.17}$$

where the $N+1$st addend gives the diagonal of $\boldsymbol{A}$ (Eq. (5.14), discarding matrix $\boldsymbol{\ell_{N+1}}$). Notice that the above expression for $\boldsymbol{D} - \boldsymbol{L}$ is in fact a sum of $N+1$ MPOs, and is therefore an MPO, but with bond dimension $(N+1)D$, that is, it grows linearly with $N$.

Following the same reasoning, one can write the upper triangular part of $\boldsymbol{A}$ as

$$\boldsymbol{U} = \sum_{i=1}^{N} \sum_{p_1,\ldots,p_N} \underbrace{\left[\bigotimes_{j=1}^{i-1} \boldsymbol{d}_{j;p_j,p_{j+1}}\right]}_{\text{diagonal}} \underbrace{\otimes\, \boldsymbol{u}_{i;p_i,p_{i+1}}}_{\text{upper triangular}} \underbrace{\left[\bigotimes_{j=i+1}^{N} \boldsymbol{w}_{j;p_j,p_{j+1}}\right]}_{\text{arbitrary}}. \tag{5.18}$$

The main problem with any iterative scheme of the form (5.1) is the growth of the bond dimension $L$ of $\boldsymbol{x}^{(k)}$ with every iteration. The complexity of truncating the vector with a prescribed accuracy $\varepsilon$ is $\mathcal{O}(2NL^3)$. If we consider a Jacobi iteration, we need to compress $\boldsymbol{x}^{(k)} + \boldsymbol{D}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)})$. Due to its increased bond dimension, compressing this vector is more expensive than one ALS iteration. Such iterative methods are not usually used as solvers, but can be used as *smoothers* in the context of multigrid methods. If this is the case, careful attention must be paid to the compression of the solution at each iteration. If the MPS is not compressed with enought accuracy, the smoothing is affected. As a naïve example, consider five Jacobi iterations of a 1-dimensional Poisson equation with both right hand side and initial guess equal to 1. Fig. 5.2 shows the exact iterate and an iterate obtained using a low compression accuracy of $\varepsilon = 1$.

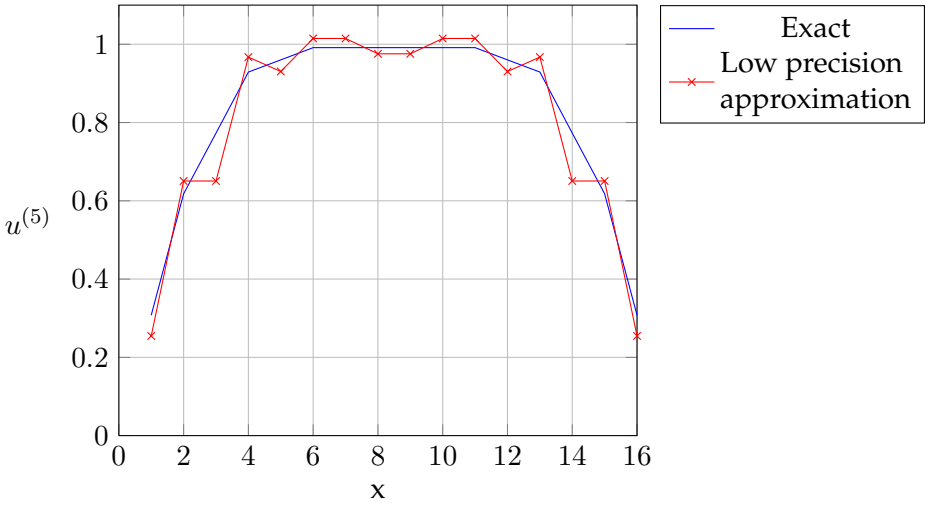



Figure 5.1: Two calculations of 5 Jacobi iterations for a 1D Poisson equation. If the compression is not carried out with enough accuracy, the smoothness of the solution is lost.

In the context of tensor representations, however, ALS or DMRG have been preferred as smoothers [4, 5], which is natural if we consider the complexity argument discussed above.

## 5.3 Preconditioning

In Chapter 4 we discussed a method to obtain an approximate inverse $\boldsymbol{M}$ of $\boldsymbol{A}$, which can be used as a preconditioner for linear systems given in MPS/MPO format. This means

that we try to solve

$$\boldsymbol{MAx} = \boldsymbol{Mb}, \tag{5.19}$$

instead of the original system $\boldsymbol{Ax} = \boldsymbol{b}$, hoping that the resulting system matrix $\boldsymbol{MA}$ will have a smaller condition number than $\boldsymbol{A}$. Note that the product $\boldsymbol{MA}$ results in an MPO with larger bond dimension (the product of both bond dimensions), and the same happens for the resulting MPS $\boldsymbol{Mb}$. A second alternative is to use

$$\boldsymbol{M}^{-1}\boldsymbol{Ax} = \boldsymbol{M}^{-1}\boldsymbol{b}, \tag{5.20}$$

which means looking for a matrix $\boldsymbol{M} \approx \boldsymbol{A}$, such that the spectrum of $\boldsymbol{M}^{-1}\boldsymbol{A}$ is better clustered then that of $\boldsymbol{A}$. Preconditioners have also been studied in the context of high-dimensional elliptics PDE eigenvalue problems in Tucker and canconical format [18, 16].

### 5.3.1 Stationary preconditioners

Choosing $\boldsymbol{D}$, the diagonal part of $\boldsymbol{A}$, to be the preconditioner $\boldsymbol{M}$ in Eq. (5.20), one can consider a Jacobi-like ALS minimization problem given by

$$\min_{x \in \mathrm{MPS}_L} \|\boldsymbol{D}^{-1}\boldsymbol{Ax} - \boldsymbol{D}^{-1}\boldsymbol{b}\|_2^2. \tag{5.21}$$

The inverse of $\boldsymbol{D}$ (or an approximation) can be obtained using Algorithm 2, and it offers the advantage of having, in general (as observed from numerical simulations), a small bond dimension. The performance of this preconditioner becomes better the more diagonally dominant the matrix $\boldsymbol{A}$ is.

A Gauss-Seidel preconditioner is obtained by defining $\boldsymbol{M} := \boldsymbol{D} - \boldsymbol{L}$, which leads to the following minimization problem:

$$\min_{x \in \mathrm{MPS}_L} \|(\boldsymbol{D} - \boldsymbol{L})^{-1}\boldsymbol{Ax} - (\boldsymbol{D} - \boldsymbol{L})^{-1}\boldsymbol{b}\|_2^2. \tag{5.22}$$

This Gauss-Seidel preconditioner, however, has a very large bond dimension (as we saw in Sec. 5.2, see Eq. (5.17)), so finding an approximate inverse can be very expensive.

An alternative idea would be to first compress $\boldsymbol{D} - \boldsymbol{L}$ (using an SVD truncation or an ALS compression scheme) so that it has a small bond dimension, and then finding an approximate inverse of this compressed MPO. In this case, if one starts with, for example, the following operator

$$\boldsymbol{A} = \begin{bmatrix} 1.0455 & 0.0440 & 0.0569 & 0.0578 & 0.0366 & 0.0359 & 0.0278 & 0.0283 \\ 0.0718 & 1.1680 & 0.0796 & 0.0808 & 0.0556 & 0.1082 & 0.0390 & 0.0402 \\ 0.0577 & 0.0575 & 1.1112 & 0.1124 & 0.0444 & 0.0447 & 0.0650 & 0.0657 \\ 0.0850 & 0.1357 & 0.1582 & 1.1891 & 0.0639 & 0.0846 & 0.0925 & 0.1112 \\ 0.1121 & 0.1109 & 0.0655 & 0.0665 & 1.0671 & 0.0658 & 0.0578 & 0.0587 \\ 0.1682 & 0.3021 & 0.0919 & 0.0959 & 0.1029 & 1.2087 & 0.0810 & 0.0831 \\ 0.1340 & 0.1353 & 0.1724 & 0.1742 & 0.0822 & 0.0826 & 1.1285 & 0.1299 \\ 0.1909 & 0.2325 & 0.2454 & 0.2957 & 0.1189 & 0.1645 & 0.1829 & 1.2194 \end{bmatrix},$$

where $\boldsymbol{A}$ was constructed with bond dimension 3, taking the lower triangular part (which has bond dimension 9) and compressing it to bond dimension 1 using purely SVD compression gives

$$\boldsymbol{M_1} = \begin{bmatrix} 1.0378 & 0.0242 & 0.0087 & 0.0002 & 0 & 0 & 0 & 0 \\ 0.1558 & 1.1617 & 0.0013 & 0.0097 & 0 & 0 & 0 & 0 \\ 0.1278 & 0.0030 & 1.0806 & 0.0252 & 0 & 0 & 0 & 0 \\ 0.0192 & 0.1430 & 0.1622 & 1.2096 & 0 & 0 & 0 & 0 \\ 0.2300 & 0.0054 & 0.0019 & 0.0000 & 1.0664 & 0.0249 & 0.0089 & 0.0002 \\ 0.0345 & 0.2574 & 0.0003 & 0.0022 & 0.1600 & 1.1937 & 0.0013 & 0.0100 \\ 0.0283 & 0.0007 & 0.2394 & 0.0056 & 0.1313 & 0.0031 & 1.1103 & 0.0259 \\ 0.0042 & 0.0317 & 0.0359 & 0.2680 & 0.0197 & 0.1469 & 0.1666 & 1.2429 \end{bmatrix},$$

which approximates the lower triangular part of $A$ with an error of $4.2293 \times 10^{-1}$. An approximation with bond dimension 2 gives an error of $6.7107 \times 10^{-2}$. Using a random right hand side $b = (0.6734, 0.1319, 0.7043, 0.3849, 0.3622, 0.9626, 0.0876, 0.2557)^\intercal$, ten regular Gauss-Seidel iterations give an error of $1.5793 \times 10^{-9}$, whereas the rank 1 approximation yields an error of $1.7821 \times 10^{-5}$, and the rank 2 approximation gives $1.6129 \times 10^{-9}$, while ten Jacobi iterations result in an error of $5.3785 \times 10^{-4}$.

Another idea for a preconditioner is what we will call an *incomplete* Gauss-Seidel preconditioner. It consists of the diagonal part of $\boldsymbol{A}$ plus a truncated number of lower terms in Eq. (5.16). For instance, the most dominant term corresponds to the addend $i = 1$, since it contributes the largest number of elements below the main diagonal. Adding the main diagonal plus this term results in an MPO of twice the bond dimension of $\boldsymbol{A}$ (instead of $N$ times, with the complete Gauss-Seidel), and it has the form

$$\boldsymbol{M} := \bigotimes_{i=1}^{N} \boldsymbol{d_{i;p_i,p_{i+1}}} - \boldsymbol{\ell_{1;p_1,p_2}} \bigotimes_{i=2}^{N} \boldsymbol{w_{i;p_i,p_{i+1}}}. \tag{5.23}$$

Applied to our previous example, Eq. (5.23) gives the following operator:

$$\boldsymbol{M} = \begin{bmatrix} 1.0455 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.1680 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.1112 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.1891 & 0 & 0 & 0 & 0 \\ 0.1121 & 0.1109 & 0.0655 & 0.0665 & 1.0671 & 0 & 0 & 0 \\ 0.1682 & 0.3021 & 0.0919 & 0.0959 & 0 & 1.2087 & 0 & 0 \\ 0.1340 & 0.1353 & 0.1724 & 0.1742 & 0 & 0 & 1.1285 & 0 \\ 0.1909 & 0.2325 & 0.2454 & 0.2957 & 0 & 0 & 0 & 1.2194 \end{bmatrix}.$$

The MPO obtained this way has bond dimension 4. Using the same right hand side as above we observe that ten iterations using the exact inverse of this incomplete Gauss-Seidel operator (which turns out to have also bond dimension 4) give an error of $1.5264 \times 10^{-7}$.

In the first approach, one finds a low bond dimension approximation of the inverse of lower part of $A$, whereas in this last approach one finds the approximate inverse of a low bond dimension representation of the lower part of $A$. The first approach seems thus more convenient since one can control the bond dimension of the resulting inverse.

Finally, one could consider finding an approximate inverse of the full matrix $\boldsymbol{A}$ directly. This can be done in two ways. 1) Compressing $\boldsymbol{A}$ to a certain bond dimension, and then finding an approximate inverse of this approximation, which might be useful if the bond dimension of $\boldsymbol{A}$ is large. Or 2) finding a low bond dimension approximate inverse of $\boldsymbol{A}$ directly.

## 5.4 Examples

Consider the following 3-dimensional PDE:

$$a_1 \frac{\partial f}{\partial x} + a_2 \left( \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \right) = 1, \quad \boldsymbol{x} = (x, y, z) \in \Omega = [0, 1]^3, \tag{5.24a}$$

$$f(x, y, z) = 0 \quad \text{in } \partial\Omega. \tag{5.24b}$$

where $a_1$ and $a_2$ are two real constants. We use the following finite difference scheme:

$$a_1 \frac{f_{i,j,k} - f_{i-1,j,k}}{h_x} + a_2 \left( \frac{f_{i,j+1,k} - 2f_{i,j,k} + f_{i,j-1,k}}{h_y^2} + \frac{f_{i,j,k+1} - 2f_{i,j,k} + f_{i,j,k-1}}{h_z^2} \right) = 1. \tag{5.25}$$

The MPO construction of the system matrix corresponding to this scheme can be obtained analogously to the $D$-dimensional Laplacian (3.45). Its resulting bond dimension is 6.

We consider four types of preconditioners for this problem:

1. A Jacobi preconditioner, taking the diagonal part of $\boldsymbol{A}$ and inverting it;
2. A Gauss-Seidel preconditioner, taking the lower triangular part of $\boldsymbol{A}$, then compressing it to a given bond dimension, and finding its approximate inverse;
3. An incomplete Gauss-Seidel preconditioner, taking the main diagonal plus the lower dominant term (Eq. (5.23));
4. A complete preconditioner, compressing $\boldsymbol{A}$ to a given bond dimensions and finding its approximate inverse.

In cases 2-4, we restricted the bond dimension of the preconditioner $\boldsymbol{M}$ to 2 (for Jacobi, it's equal to 1 since the matrix is Toeplitz), so that the bond dimension of the preconditioned MPO $\boldsymbol{MA}$ remains moderate (in this case, 12, except for Jacobi).

We start off by performing 20 iterations of the ALS algorithm (Algorithm 1). We seek a solution vector $\boldsymbol{f}$ with bond dimension 5, and we choose the following parameters: $a_1 = 10^{-3}$, $a_2 = 10^{-6}$, with discretization points per dimension $N_x = 2^{10}$, $N_y = N_z = 2^8$. The solution vector is thus an MPS of dimension 26 (with $2^{26}$ entries).

| Prec. | total time (s) | $\frac{\|\boldsymbol{AM}-\boldsymbol{I}\|_F}{\|\boldsymbol{I}\|_F}$ |
|---|---|---|
| None | 28.5 | - |
| Jac. | 30.3 | 1.36 |
| GS | 285 | 0.206 |
| Inc. GS | 288 | 1.35 |
| Comp. | 361 | 0.194 |

Table 5.1: Total time and preconditioner accuracy.

Figure 5.2 shows the effect of the four preconditioners. We observe that both Jacobi and the incomplete Gauss-Seidel have no effect on the convergence of the method, and are thus not recommendable. In the case of the Jacobi preconditioner, this might be due

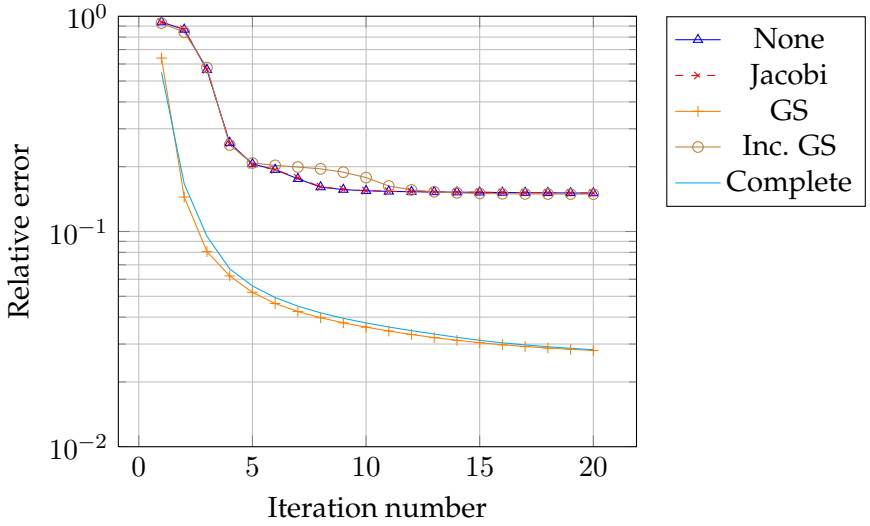



Figure 5.2: Preconditioned ALS for Eq (5.25).

to the fact that its MPO matrices are constant and site independent, causing the product $\boldsymbol{MA}$ to result only in a rescaling of the MPO matrices of $\boldsymbol{A}$. The Gauss-Seidel and the complete preconditioner behave similarly, and the advantage is that they can improve the convergence with a fixed bond dimension. Without such a preconditioner, the ALS solution stalls. The times reported in Table 5.1 include the times required to calculate each preconditioner, which are actually very small due to their reduced bond dimensions.

After several numerical experiments (using different MPOs as system matrices), it seems that the effect of preconditioning in DMRG is hardly noticeable. This might indicate that the local linear systems (when optimizing two cores) do not see the effect of preconditioning. Nevertheless, the preconditioners presented here might be useful in other families of methods, namely, those translated from numerical linear algebra to low-rank tensor representations. In the next section, we present such an example, introducing an MPS/MPO version of the preconditioned Lanczos algorithm.

# 6 A preconditioned Lanczos algorithm for linear systems

The ALS and DMRG methods (as described, for instance, in Chapters 3 and 4) have been the state of the art algorithms for solving numerical problems in MPS format. Recently, Dolgov [6] proposed a restarted GMRES algorithm in TT format (equivalent to OBC MPS), showing some advantages over DMRG in certain cases, especially in the estimation of the solution ranks. The Lanczos algorithm, being also a Krylov subspace iteration method, has proved especially powerful in the solution of linear systems with symmetric positive definite and semi-definite matrices. In its original form, it was used to find eigenpairs of large symmetric matrices, but it was soon adapted to solve linear systems with one or multiple right hand sides [25, 3]. Following this direction, a restarted Lanczos algorithm to calculate the eigenvalues of an MPO was proposed in [11] , showing good convergence properties (comparable to DMRG) and more flexibility in the choice of the bond dimension.

We start our discussion by describing the Lanczos algorithm for linear systems in its original form (as given in [25]). We then propose an adaptation of the algorithm to MPS/MPO format and discuss its convergence properties, comparing it to ALS and DMRG.

## 6.1 The Lanczos algorithm

Following the discussion in [25], we recall that the Lanczos algorithm to solve

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{3.1}$$

consists primarily in building an orthonormal basis $(\boldsymbol{q_1}, \boldsymbol{q_2}, \cdots, \boldsymbol{q_j})$ starting from the set $\{\boldsymbol{r_0}, \boldsymbol{A r_0}, \cdots, \boldsymbol{A}^{(j-1)}\boldsymbol{r_0}\}$ (know as the Krylov subspace), where $\boldsymbol{r_0} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^a$ is the initial residual and $\boldsymbol{x}^a$ is an approximation of the solution of (3.1). The orthonormal vectors $\boldsymbol{q_j}$ are obtained through a Gram-Schmidt procedure resulting in the following relation:

$$\boldsymbol{r_{j+1}} = \beta_{j+1}\boldsymbol{q_{j+1}} = \boldsymbol{A}\boldsymbol{q_j} - \alpha_j \boldsymbol{q_j} - \beta_j \boldsymbol{q_{j-1}}. \tag{6.1}$$

The projection of $\boldsymbol{A}$ onto this space transforms the system into a tridiagonal one, namely

$$\boldsymbol{T_j} = \boldsymbol{Q_j}^{\mathsf{T}} \boldsymbol{A} \boldsymbol{Q_j} = \text{tridiag}(\boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \tag{6.2}$$

where $\boldsymbol{Q_j} = [\boldsymbol{q_1 q_2} \cdots \boldsymbol{q_j}]$, $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_j)$ and $\boldsymbol{\beta} = (\beta_2, \beta_3, \cdots, \beta_j)$.

Now, we look for a correction $\boldsymbol{x}^c$ to be added to the initial approximation $\boldsymbol{x}^a$, which satisfies the relation

$$\boldsymbol{A}\boldsymbol{x}^c = \boldsymbol{r_0}, \tag{6.3}$$

and, from the properties of the resulting orthonormal space obtained before, one can show that $\boldsymbol{x}^c$ also satisfies

$$\boldsymbol{x_j}^c = \boldsymbol{Q_j}\boldsymbol{y_j}, \tag{6.4}$$

where $\boldsymbol{y_j}$ is the solution of the tridiagonal system

$$\boldsymbol{T_j y_j} = \boldsymbol{Q_j}^\mathsf{T} \boldsymbol{r_0}. \tag{6.5}$$

The last step is to perform an *LDL* decomposition of $\boldsymbol{T_j}$ to solve Eq. (6.5), namely

$$\boldsymbol{L_j D_j L_j}^\mathsf{T} \boldsymbol{y_j} = \boldsymbol{Q_j}^\mathsf{T} \boldsymbol{r_0}. \tag{6.6}$$

Note that in order to solve Eq. (6.6) one needs to build $\boldsymbol{Q_j}$ by saving all Lanczos vectors $\boldsymbol{q_i}$. Paige and Saunders [24] proposed the following alternative, which requires the storage of only two Lanczos vectors. Define $\boldsymbol{z_j}$ and $\boldsymbol{C_j}$ as

$$\boldsymbol{z_j} = \boldsymbol{L_j}^\mathsf{T} \boldsymbol{y_j} = \boldsymbol{L_j}^\mathsf{T} \boldsymbol{Q_j}^\mathsf{T} \boldsymbol{x_j}, \tag{6.7}$$

$$\boldsymbol{L_j C_j}^\mathsf{T} = \boldsymbol{Q_j}^\mathsf{T}, \tag{6.8}$$

where the superscript $c$ of $\boldsymbol{x_j}$ has been dropped for clarity. Equations (6.6) and (6.4) become

$$\boldsymbol{L_j D_j z_j} = \boldsymbol{Q_j}^\mathsf{T} \boldsymbol{r_0}, \tag{6.9}$$

$$\boldsymbol{x_j} = \boldsymbol{C_j z_j}, \tag{6.10}$$

or, rewriting Eq (6.9) and (6.8) in matrix form,

$$\begin{pmatrix} 1 & & & \\ \delta_2 & 1 & & \\ & \ddots & \ddots & \\ & & \delta_j & 1 \end{pmatrix} \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_j \end{pmatrix} \begin{pmatrix} \zeta_1 \\ \vdots \\ \zeta_j \end{pmatrix} = \begin{pmatrix} \boldsymbol{q_1}^\mathsf{T} \\ \vdots \\ \boldsymbol{q_j}^\mathsf{T} \end{pmatrix} \cdot \boldsymbol{r_0}, \tag{6.11}$$

and

$$\begin{pmatrix} 1 & & & \\ \delta_2 & 1 & & \\ & \ddots & \ddots & \\ & & \delta_j & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{c_1}^\mathsf{T} \\ \vdots \\ \boldsymbol{c_j}^\mathsf{T} \end{pmatrix} = \begin{pmatrix} \boldsymbol{q_1}^\mathsf{T} \\ \vdots \\ \boldsymbol{q_j}^\mathsf{T} \end{pmatrix}. \tag{6.12}$$

From these two equations one can derive the following:

$$\zeta_j = (\boldsymbol{q_j}^\mathsf{T} \boldsymbol{r_0} - \delta_j d_{j-1} \zeta_{j-1})/d_j, \tag{6.13}$$

$$\boldsymbol{c_j}^\mathsf{T} = \boldsymbol{q_j}^\mathsf{T} - \delta_j \boldsymbol{c_{j-1}}^\mathsf{T}. \tag{6.14}$$

Inserting these results into (6.10) gives the iterative correction

$$\boxed{\boldsymbol{x_j} = \boldsymbol{x_{j-1}} + \zeta_j \boldsymbol{c_j}.} \tag{6.15}$$

One can also allow the use of a preconditioner in order to solve the system

$$\boldsymbol{M}^{-1} \boldsymbol{A x} = \boldsymbol{M}^{-1} \boldsymbol{b}, , \tag{5.20}$$

resulting in Algorithm 3, where quantities with bars indicate preconditioned variables. We also set $\boldsymbol{R} = \boldsymbol{M M}^\mathsf{T}$.

**Algorithm 3:** Preconditioned Lanczos algorithm for linear systems

**input** : $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{b} \in \mathbb{R}^n$, $\boldsymbol{x_0} \in \mathbb{R}^n$, $\epsilon > 0$.
**output**: $\boldsymbol{x_j}$, an approximate solution to the system.

1 **Start:**
2 $\boldsymbol{r_1} := \boldsymbol{b} - \boldsymbol{A x_0}$, $\bar{\boldsymbol{r}}_{\boldsymbol{1}} := \boldsymbol{M}^{-1}\boldsymbol{r_1}$, $\bar{\beta}_1 := \|\bar{\boldsymbol{r}}_{\boldsymbol{1}}\| = \sqrt{\boldsymbol{r_1}^\intercal \boldsymbol{R}^{-1}\boldsymbol{r_1}}$, with $\boldsymbol{q_0} := 0$ and $\boldsymbol{q_1} := \boldsymbol{r_1}/\bar{\beta}_1$
3 $\bar{\boldsymbol{c}}_{\boldsymbol{0}} := 0$, $\bar{d}_0 = \bar{\delta}_1 := 0$
4 **for** $j = 1, 2, \ldots$ **do**
5  **Generate Lanczos vectors:**
6  $\bar{\boldsymbol{u}}_{\boldsymbol{j}} := \boldsymbol{R}^{-1}\boldsymbol{q_j}$
7  $\bar{\alpha}_j := \bar{\boldsymbol{u}}_{\boldsymbol{j}}{}^\intercal \boldsymbol{A}\bar{\boldsymbol{u}}_{\boldsymbol{j}}$
8  $\boldsymbol{r_{j+1}} := \boldsymbol{A u_j} - \bar{\alpha}_j \boldsymbol{q_j} - \bar{\beta}_j \boldsymbol{q_{j-1}}$
9  $\bar{\beta}_{j+1} := \sqrt{\boldsymbol{r_{j+1}}^\intercal \boldsymbol{R}^{-1}\boldsymbol{r_{j+1}}}$
10  $\boldsymbol{q_{j+1}} := \boldsymbol{r_{j+1}}/\bar{\beta}_{j+1}$
11  **Form the approximate solution:**
12  $\bar{d}_j := \bar{\alpha}_j - \bar{\delta}_j^2 \bar{d}_{j-1}$
13  $\bar{\delta}_{j+1} := \bar{\beta}_{j+1}/\bar{d}_j$
14  $\bar{\zeta}_j := -\bar{\delta}_j \bar{d}_{j-1}\bar{\zeta}_{j-1}/\bar{d}_j$ (with $\bar{\zeta}_1 := \bar{\beta}_1/\bar{d}_1$)
15  $\bar{\boldsymbol{c}}_{\boldsymbol{j}}{}^\intercal := \boldsymbol{q_j}{}^\intercal \boldsymbol{R}^{-1} - \bar{\delta}_j \bar{\boldsymbol{c}}_{\boldsymbol{j-1}}{}^\intercal$
16  $\boldsymbol{x_j} := \boldsymbol{x_{j-1}} + \bar{\zeta}_j \bar{\boldsymbol{c}}_{\boldsymbol{j}}$
17  **Convergence criterion:**
18  **if** $|\bar{\zeta}_j| \cdot \|\boldsymbol{r_j}\|/\|\boldsymbol{r_1}\| < \epsilon$ **then**
19  terminate algorithm
20  **end**
21 **end**

## 6.2 Translation to MPS/MPO: A restarted Lanczos algorithm

The main *for* loop in Algorithm 3 poses an evident problem for an MPS implementation, since the vector additions in lines 8, 15 and 16, as well as the matrix-vector product in line 6 would cause a considerable increase in the bond dimensions of the corresponding MPS vectors. A restarted Lanczos algorithm, along with an appropriate compression scheme for the MPS vectors, is a way possible to overcome this problem. The main difference is that the *for* loop will now generate a fixed number $m$ of Lanczos vectors and build the new approximate solution, after which the algorithm is restarted, using this new approximation as initial guess. Using a preconditioner $\boldsymbol{M}$ given in MPO format, the resulting scheme is summarized as Algorithm 4.

**Algorithm 4:** Preconditioned restarted Lanczos algorithm for linear systems in MPS/MPO format

**input** : $\boldsymbol{A} \in \mathrm{MPO}_D$, symmetric MPO system matrix; $\boldsymbol{b} \in \mathrm{MPS}_N$, right hand side MPS; $\boldsymbol{x_0} \in \mathrm{MPS}_L$, initial guess MPS; $\boldsymbol{M} \in \mathrm{MPO}_P$, a preconditioner MPO; $\varepsilon > 0$, convergence criterion; $m$, subspace dimension.

**output**: $\boldsymbol{x} \in \mathrm{MPS}_L$, approximate solution of $\boldsymbol{Ax} = \boldsymbol{b}$.

Set $\boldsymbol{r_0} := \boldsymbol{b} - \boldsymbol{Ax_0}$
**for** $i = 1, 2, ...$ **do** `// Restart until convergence`
  **Start:**
  $\boldsymbol{r_1} := \boldsymbol{b} - \boldsymbol{Ax_0}$, $\bar{\boldsymbol{r}}_\mathbf{1} := \boldsymbol{M}^{-1}\boldsymbol{r_1}$, $\bar{\beta}_1 := \|\bar{\boldsymbol{r}}_\mathbf{1}\| = \sqrt{\boldsymbol{r_1}^\intercal \boldsymbol{R}^{-1} \boldsymbol{r_1}}$, with $\boldsymbol{q_0} := 0$ and $\boldsymbol{q_1} := \boldsymbol{r_1}/\bar{\beta}_1$
  $\bar{\boldsymbol{c}}_\mathbf{0} := 0$, $\bar{d}_0 = \bar{\delta}_1 := 0$
  **for** $j = 1, 2, ..., m$ **do**
    **Generate Lanczos vectors:**
    $\bar{\boldsymbol{u}}_\boldsymbol{j} := \boldsymbol{R}^{-1}\boldsymbol{q_j}$
    $\bar{\alpha}_j := \bar{\boldsymbol{u}}_\boldsymbol{j}^\intercal \boldsymbol{A} \bar{\boldsymbol{u}}_\boldsymbol{j}$
    $\boldsymbol{r_{j+1}} := \boldsymbol{A}\bar{\boldsymbol{u}}_\boldsymbol{j} - \bar{\alpha}_j \boldsymbol{q_j} - \bar{\beta}_j \boldsymbol{q_{j-1}}$
    $\bar{\beta}_{j+1} := \sqrt{\boldsymbol{r_{j+1}}^\intercal \boldsymbol{R}^{-1} \boldsymbol{r_{j+1}}}$
    $\boldsymbol{q_{j+1}} := \boldsymbol{r_{j+1}}/\bar{\beta}_{j+1}$
    **Form the approximate solution:**
    $\bar{d}_j := \bar{\alpha}_j - \bar{\delta}_j^2 \bar{d}_{j-1}$
    $\bar{\delta}_{j+1} := \bar{\beta}_{j+1}/\bar{d}_j$
    $\bar{\zeta}_j := -\bar{\delta}_j \bar{d}_{j-1} \bar{\zeta}_{j-1}/\bar{d}_j$ (with $\bar{\zeta}_1 := \bar{\beta}_1/\bar{d}_1$)
    $\bar{\boldsymbol{c}}_\boldsymbol{j}^\intercal := \boldsymbol{q_j}^\intercal \boldsymbol{R}^{-1} - \bar{\delta}_j \bar{\boldsymbol{c}}_\boldsymbol{j-1}^\intercal$
    $\boldsymbol{x_j} := \boldsymbol{x_{j-1}} + \bar{\zeta}_j \bar{\boldsymbol{c}}_\boldsymbol{j}$
    **Convergence criterion:**
    **if** $|\bar{\zeta}_j| \cdot \|\boldsymbol{r_{j+1}}\|/\|\boldsymbol{r_0}\| < \epsilon$ **then**
      terminate algorithm
    **end**
  **end**
  $\boldsymbol{x_m} := \mathcal{P}_{\varepsilon,L}(\boldsymbol{x_m})$ `// Compress with accuracy` $\varepsilon$ `or to bond dim.` $L$
  Set $\boldsymbol{x_0} := \boldsymbol{x_m}$
**end**

## 6.3 Results

Algorithm 4 was implemented for different MPO/MPS linear systems. As an example, we go back to the $D$-dimensional Poisson equation

$$\begin{aligned} -\Delta_D u(x) = 1, \qquad & x \in \Omega = [0,1]^D, \\ & x = [x_1, \ldots, x_D], \\ & u(x) = 0 \quad \text{in } \partial\Omega. \end{aligned} \tag{6.16}$$

As mentioned, the main problem with the Lanczos algorithm is the increasing bond dimension of $\bar{\boldsymbol{u}}_j$ (and $\bar{\boldsymbol{r}}_j$), and these should not be compressed within the inner for loop, since doing it would mean leaving the Krylov subspace, thus adding errors to the following operations. As a consequence, the preconditioned MPO $\boldsymbol{R}$ should be chosen to have a small bond dimension.

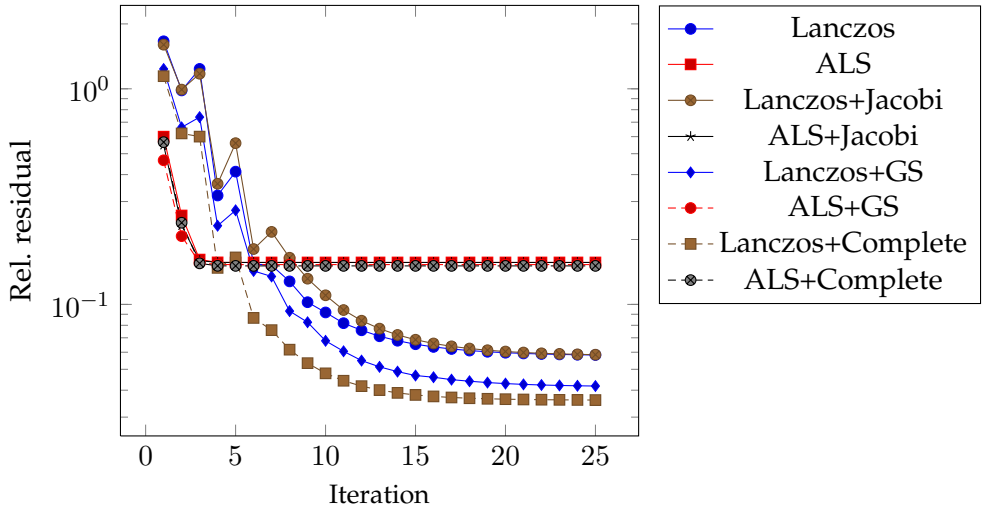



Figure 6.1: Comparison of ALS vs Lanczos for Eq. (6.16) using a solution with fixed bond dimension 3.

| ALS | |
|---|---|
| Preconditioner | Total time (s) |
| None | 6.95 |
| Jac. | 7.03 |
| GS | 6.98 |
| Comp. | 6.95 |

Table 6.1: Time for 25 iterations using ALS, solution with bond dimension 3.

| Lanczos | |
|---|---|
| Preconditioner | Total time (s) |
| None | 60.2 |
| Jac. | 59.5 |
| GS | 59.1 |
| Comp. | 59.2 |

Table 6.2: Time for 25 iterations using Lanczos, solution with bond dimension 3.

For this example, we take a 5-dimensional Laplacian with $2^4$ discretization points per dimension. The dimension of the Lanczos subspace $m$ is equal to 2, and we perform 25 iterations of the algorithm with four of the preconditioners discussed in the previous chapter. Fig. 6.1 shows the convergence properties of the preconditioned Lanczos algorithm compared to standard ALS iterations. We measured in both cases the preconditioned relative residual (instead of the criterion in line 19 of Algorithm 4). The times required for the 25 iterations are reported in Tables 6.1 and 6.2, where we have searched for a solution MPS with bond dimension 3. Although each iteration of the Lanczos algorithm takes approximately 10 times longer than an ALS iteration, the convergence can be improved by

up to a factor of approximately 5 (for a fixed bond dimension). We also observe that the effect of preconditioning in the Lanczos algorithm does make a small difference even if they have bond dimension 1. Additionally, increasing the bond dimension of the solution MPS shows similar convergence results (larger times for Lanczos but with improved convergence). The comparison with DMRG is difficult to carry out in this case, since the bond dimension changes in DMRG, which makes it in general considerably more efficient and faster (see Table 3.2, set 8, for an almost equivalent simulation).

A possible improvement to the algorithm would be to increase the bond dimension of the solution MPS when the convergence starts to stall, but this can only be done if the problem of increasing bond dimensions can be effectively overcome. Our simulations also show that the Lanczos algorithm starts to present difficulties as $N$ (the number of discretization points) increases, but performs well with increasing dimensionality.

We conclude that the Lanczos algorithm as presented has severe limitations and performs only slightly better than ALS in certain cases, namely, when the bond dimensions of both the system matrix MPO $\boldsymbol{A}$ and the solution MPS $\boldsymbol{x}$ are small, as well as the dimension of the Lanczos subspace and the number of discretization points per dimension. Still, there might exist scenarios where it might be preferabe to use such Krylov subspace approach instead of DMRG (for a related discussion, see [6]), in which case the Lanczos algorithm could be useful.

# 7 A Multigrid implementation in MPS format

Yet another algorithm that has proven highly efficient for the solution of discretized PDEs is the multigrid approach (in its different forms: two-grid, V-cycle, $\mu$-cycle, full multigrid, full approximation storage, etc.). A full multigrid (FMG) implementation using the canonical tensor decomposition (*CANDECOMP*) was recently presented [4], and another recent work presents a treatment of multigrid methods in MPS format applied to the solution of dilute lattice models [5]. In this chapter, a FMG method using the V-cycle is translated to the MPS/MPO formalism and we present different ideas on prolongation and restriction. We then discuss the performance of our implementation compared to the rest of the algorithms presented so far (ALS, DMRG, Lanczos). As in Chapter 6, we first start with a quick overview of the V-cycle, based on [1].

## 7.1 The V-cycle: full version

In the one dimensional case, one starts defining a grid $\Omega^h = [a, b]$ with $h = (b-a)/n$, $n = 2^N$, so there are $n-1$ interior points. The discretization of the PDE leads to a system of equations $\boldsymbol{A}^h\boldsymbol{x}^h = \boldsymbol{b}^h$ (the superscript $h$ denoting the grid spacing), each approximate solution $\boldsymbol{x}^h$ yielding a residual $\boldsymbol{r}^h := \boldsymbol{b}^h - \boldsymbol{A}^h\boldsymbol{x}^h$, and an error $\boldsymbol{e}^h := \boldsymbol{x}^h - \boldsymbol{x}_e$, where $\boldsymbol{x}_e$ is the exact solution to the system of equations. The main idea of the V-cycle is to recursively find better estimates for the error $\boldsymbol{e}^h$ on coarser grids, knowing that it satisfies the error equation

$$\boldsymbol{A}^h\boldsymbol{e}^h = \boldsymbol{r}^h = \boldsymbol{b}^h - \boldsymbol{A}^h\boldsymbol{x}^h. \tag{7.1}$$

At each mesh, one tries to eliminate high frequency components of the error through a number of pre- and postsmoothing operations (usually Jacobi or Gauss-Seidel iterations, Eqs. (5.4) and (5.7)), and the V-cycle is repeated until a certain convergence criterion is met.

In order to move from a coarse grid $\Omega^{2h}$ to a finer grid $\Omega^h$, one makes use of an *interpolation* operator, denoted $\boldsymbol{I}^h_{2h}$, which is a linear operator from $\mathbb{R}^{\frac{n}{2}-1}$ to $\mathbb{R}^{n-1}$. For $n = 8$, its action on a vector $\boldsymbol{x}^{2h}$ is as follows:

$$\boldsymbol{I}^h_{2h}\boldsymbol{x}^{2h} = \frac{1}{2}\begin{pmatrix} 1 & & \\ 2 & & \\ 1 & 1 & \\ & 2 & \\ & 1 & 1 \\ & & 2 \\ & & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}_{2h} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}_h = \boldsymbol{x}^h. \tag{7.2}$$

The corresponding operator to go from a fine grid $\Omega^h$ to a coarser grid $\Omega^{2h}$ is called a *restriction* operator, denoted $\boldsymbol{I}_h^{2h}$, which is a linear operator from $\mathbb{R}^{n-1}$ to $\mathbb{R}^{\frac{n}{2}-1}$. A usual choice for $\boldsymbol{I}_h^{2h}$ is the *full weighting* operator, whose action on a vector $\boldsymbol{x}^h$ for $n = 8$ is given by

$$\boldsymbol{I}_h^{2h}\boldsymbol{x}^h = \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 & & & & \\ & & 1 & 2 & 1 & & \\ & & & & 1 & 2 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}_h = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}_{2h} = \boldsymbol{x}^{2h}. \tag{7.3}$$

One can see that the interpolation and full weighting operators are related by

$$\boldsymbol{I}_h^{2h} = \frac{1}{2}(\boldsymbol{I}_{2h}^h)^\intercal. \tag{7.4}$$

This fact (called the *variational property*) offers advantages compared to the more obvious choice for restriction, namely, taking

$$x_j^{2h} = x_{2j}^h, \tag{7.5}$$

called *injection*. The V-cycle algorithm in its recursive form is summarized below.

**Algorithm 5:** vcycle - Full Matrix-Vector V-Cycle Scheme (Recursive Definition)

**input** : $\boldsymbol{A}^h \in \mathbb{R}^{(n-1)\times(n-1)}$, $\boldsymbol{b}^h \in \mathbb{R}^{n-1}$, $\boldsymbol{x}^h \in \mathbb{R}^{n-1}$.
**output**: $\boldsymbol{x}^h \in \mathbb{R}^{n-1}$.

1 **while** *not converged* **do**
2 relax $\nu_1$ times on $\boldsymbol{A}^h\boldsymbol{x}_e^h = \boldsymbol{b}^h$ with initial guess $\boldsymbol{x}^h$;
3 **if** $\Omega^h =$ *coarsest grid* **then**
4 go to line 11;
5 **else**
6 $\boldsymbol{b}^{2h} \leftarrow \boldsymbol{I}_h^{2h}(\boldsymbol{b}^h - \boldsymbol{A}^h\boldsymbol{x}^h)$;
7 $\boldsymbol{x}^{2h} \leftarrow 0$;
8 $\boldsymbol{x}^{2h} \leftarrow$ vcycle$(\boldsymbol{A}^{2h}, \boldsymbol{b}^{2h}, \boldsymbol{x}^{2h})$;
9 **end**
10 correct $\boldsymbol{x}^h \leftarrow \boldsymbol{x}^h + \boldsymbol{I}_{2h}^h\boldsymbol{x}^{2h}$;
11 relax $\nu_2$ times on $\boldsymbol{A}^h\boldsymbol{x}_e^h = \boldsymbol{b}^h$ with initial guess $\boldsymbol{x}^h$;
12 **end**

## 7.2 Elements of Multigrid in MPS/MPO format

If a PDE problem can be translated to the MPS/MPO formalism, a corresponding multigrid implementation requires several components to be well defined in this format. These

include smoothing, interpolation, restriction, and adequate compression schemes to handle an eventual growth of the ranks.

We saw in Chapter 5 how to translate Jacobi and Gauss-Seidel iterations to MPS format, but also discussed their main disadvantage: the bond dimensions of the vectors involved are too large, and compressing the solution vector after each iteration is more expensive than carrying out an ALS iteration on the main system. This is why we will prefer ALS or DMRG as smoothers, and the question is then if this is affordable, considering that an algorithm like DMRG is itself a powerful solver. We will try to answer this question in the next sections.

To indicate the level of coarseness, we now use a superindex $N$ instead of $h$, since it immediately reflects the number of MPO/MPS sites.

The authors in [14] describe how to obtain a TT representation of the interpolation and full-weighting operators (equivalent to OBC-MPO), with the small difference that the number of inner points considered in the grid is $2^N$ instead of $2^N - 1$, so that $h = (b-a)/(2^N+1)$. The one dimensional interpolation operator defined this way is a $2^N \times 2^{N-1}$ matrix of the form

$$\begin{aligned} \boldsymbol{P_1}^{(N)} &= \frac{1}{2}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{J}^\intercal\end{bmatrix} \bowtie \begin{bmatrix}\boldsymbol{I} & \boldsymbol{J}^\intercal \\ & \boldsymbol{J}\end{bmatrix}^{\bowtie(N-2)} \bowtie \begin{bmatrix}\begin{pmatrix}1\\2\end{pmatrix} \\ \begin{pmatrix}1\\0\end{pmatrix}\end{bmatrix} \\ &= \frac{1}{2}\begin{pmatrix} 1 & & & & \\ 2 & & & & \\ 1 & 1 & & & \\ & 2 & & & \\ & 1 & \ddots & 1 & \\ & & \ddots & 2 & \\ & & \ddots & 1 & 1 \\ & & & & 2 \end{pmatrix}, \end{aligned} \tag{7.6}$$

and the full-weighting restriction operator in one dimension is given by

$$\begin{aligned} \boldsymbol{R_1}^{(N)} &= \frac{1}{4}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{J}\end{bmatrix} \bowtie \begin{bmatrix}\boldsymbol{I} & \boldsymbol{J} \\ & \boldsymbol{J}^\intercal\end{bmatrix}^{\bowtie(N-2)} \bowtie \begin{bmatrix}1 & 2 \\ 1 & 0\end{bmatrix} \\ &= \frac{1}{2}(\boldsymbol{P}^{(N)})^\intercal. \end{aligned} \tag{7.7}$$

From their inner core product definitions, one can observe that the bond dimension of both interpolation and restriction operators is equal to 2.

It is also straightforward to determine how an MPS/MPO transforms in the case where restriction is carried out by injection (choosing only the even or the odd components of the vector), thanks to the binary construction of its components. For instance, if $N = 3$, we

have

$$(\boldsymbol{x})_{i_1,i_2,i_3} = \boldsymbol{X_1^{(i_1)}} \underbrace{\boldsymbol{X_2^{(i_2)}} \boldsymbol{X_3^{(i_3)}}}_{\rightarrow \boldsymbol{X_2^{(i_2 i_3)}}} \tag{7.8}$$

$$\rightarrow (\boldsymbol{x})_{i_1,i_2} = \boldsymbol{X_1^{(i_1)}} \boldsymbol{X_2^{\prime(i_2)}}, \tag{7.9}$$

where

$$\begin{aligned} \boldsymbol{X_2^{\prime(0)}} &:= \boldsymbol{X_2^{(00)}}, \\ \boldsymbol{X_2^{\prime(1)}} &:= \boldsymbol{X_2^{(10)}}. \end{aligned}$$

In words, one contracts the last two core matrices of the MPS (resp. MPO), which results in a larger physical index ($i_{N-1}i_N$), and takes the first two (resp. four) matrices of the resulting new core:

$$\begin{aligned} (\boldsymbol{x})_{i_1,i_2,i_3} &= (x_{000}, x_{001}, x_{010}, x_{011}, x_{100}, x_{101}, x_{110}, x_{111}), \\ \rightarrow (\boldsymbol{x})_{i_1,i_2} &= (x_{000}, x_{010}, x_{100}, x_{110}) \\ &= (x_{00}, x_{01}, x_{10}, x_{11}). \end{aligned}$$

The advantage of using injection instead of a restriction operator like (7.7) is that the bond dimension of the MPS does not change, so no compression is needed when going recursively to coarser grids. We would also like to find a similar approach for interpolation, but so far we have not found such an expression.

We also need to define the product of a rectangular MPO (such as 7.6) with an MPS. Take, for example, an interpolation operation $\boldsymbol{x}^{(N)} = \boldsymbol{P}^{(N)}\boldsymbol{x}^{(N-1)}$. In terms of the trace representation, using again $N = 3$ as an example, one has

$$\begin{aligned} (\boldsymbol{x}^{(N)})_{i_1,i_2,i_3} &= \sum_{j_1,j_2} (\boldsymbol{P_1}^{(N)})_{\substack{i_1,i_2,i_3 \\ j_1,j_2}} (\boldsymbol{x}^{(N-1)})_{j_1,j_2} \\ &= \sum_{j_1,j_2} \mathrm{Tr}\left(\boldsymbol{W_1^{(i_1,j_1)}} \boldsymbol{W_2^{(i_2,j_2)}} \boldsymbol{W_3^{(i_3)}}\right) \mathrm{Tr}\left(\boldsymbol{X_1^{(j_1)}} \boldsymbol{X_2^{(j_2)}}\right) \\ &= \mathrm{Tr}\left[\left(\sum_{j_1} \boldsymbol{W_1^{(i_1,j_1)}} \otimes \boldsymbol{X_1^{(j_1)}}\right) \left(\sum_{j_2} \boldsymbol{W_2^{(i_2,j_2)}} \otimes \boldsymbol{X_2^{(j_2)}}\right) \left(\boldsymbol{W_3^{(i_3)}} \otimes \boldsymbol{I}\right)\right] \\ &= \mathrm{Tr}\left(\boldsymbol{X_1^{\prime(i_1)}} \boldsymbol{X_2^{\prime(i_2)}} \boldsymbol{X_3^{\prime(i_3)}}\right). \end{aligned} \tag{7.10}$$

In the case of restriction, one has

$$
\begin{aligned}
(\boldsymbol{x}^{(N-1)})_{i_1,i_2} &= \sum_{j_1,j_2,j_3} (\boldsymbol{R_1}^{(N)})_{\substack{i_1,i_2\\ j_1,j_2,j_3}} (\boldsymbol{x}^{(N)})_{j_1,j_2,j_2} \\
&= \sum_{j_1,j_2,j_3} \mathrm{Tr}\left(\boldsymbol{W_1^{(i_1,j_1)}} \boldsymbol{W_2^{(i_2,j_2)}} \boldsymbol{W_3^{(j_3)}}\right) \mathrm{Tr}\left(\boldsymbol{X_1^{(j_1)}} \boldsymbol{X_2^{(j_2)}} \boldsymbol{X_3^{(j_3)}}\right) \\
&= \mathrm{Tr}\left[\left(\sum_{j_1} \boldsymbol{W_1^{(i_1,j_1)}} \otimes \boldsymbol{X_1^{(j_1)}}\right)\left(\sum_{j_2} \boldsymbol{W_2^{(i_2,j_2)}} \otimes \boldsymbol{X_2^{(j_2)}}\right)\left(\underbrace{\sum_{j_3} \boldsymbol{W_3^{(j_3)}} \otimes \boldsymbol{X_3^{(j_3)}}}_{:=\boldsymbol{Y}}\right)\right] \\
&= \mathrm{Tr}\left(\boldsymbol{X_1'^{(i_1)}} \underbrace{\boldsymbol{X_2'^{(i_2)}} \boldsymbol{Y}}_{\to \boldsymbol{X_2'^{(i_2)}}}\right) \\
&= \mathrm{Tr}\left(\boldsymbol{X_1'^{(i_1)}} \boldsymbol{X_2'^{(i_2)}}\right). \qquad (7.11)
\end{aligned}
$$

In the last step, we simply transfer the matrix $\boldsymbol{Y}$ to the left neighbor.

Both interpolation and restriction operators can also be applied to MPO, as can happen when translating the system matrix from a fine to a coarse grid using the Galerkin condition, i.e., $\boldsymbol{A}^{(N-1)} = \boldsymbol{R_1}^{(N)} \boldsymbol{A}^{(N)} \boldsymbol{P_1}^{(N)}$. However, this would increase the bond dimension of $\boldsymbol{A}$ (multiplying it by a factor of 4), so it is desirable to rediscretize the system MPO at each level of coarseness, if possible. In fact, this is usually possible, since the techniques to construct MPO from the finite difference discretization of many differential operators are available in the literature.

### 7.2.1 Operators in higher dimensions

The MPO formalism allows a natural extension of the operators we have discussed to higher dimensions . Using $\boldsymbol{P_1}^{(N)}$ (the interpolation operator in one dimension, Eq. (7.6)), we can obtain the interpolation operator in higher dimensions as the tensor product of one dimensional operators, namely

$$
\begin{aligned}
\boldsymbol{P_D}^{(N_1,\ldots,N_D)} &= \boldsymbol{P_1}^{(N_1)} \otimes \boldsymbol{P_1}^{(N_2)} \otimes \cdots \otimes \boldsymbol{P_1}^{(N_{D-1})} \otimes \boldsymbol{P_1}^{(N_D)} \qquad (7.12) \\
&= \bigotimes_{i=1}^{D} \boldsymbol{P_1}^{(N_i)}. \qquad (7.13)
\end{aligned}
$$

In the definition above, we have allowed a different number of discretization points in each dimension, namely, $2^{N_i}$ in the $i$-th dimension. From the inner core product definition of $\boldsymbol{P_1}^{(N_1,\ldots,N_D)}$, the two dimensional interpolation operator with $N = 3$ on both dimensions would then be given by

$$
\begin{aligned}
\boldsymbol{P_2}^{(3,3)} &= \boldsymbol{P_1}^{(3)} \otimes \boldsymbol{P_1}^{(3)} \\
&= \frac{1}{2}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{J}^\intercal\end{bmatrix} \bowtie \begin{bmatrix}\boldsymbol{I} & \boldsymbol{J}^\intercal \\ & \boldsymbol{J}\end{bmatrix} \bowtie \begin{bmatrix}\begin{pmatrix}1\\2\end{pmatrix} \\ \begin{pmatrix}1\\0\end{pmatrix}\end{bmatrix} \bowtie \frac{1}{2}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{J}^\intercal\end{bmatrix} \bowtie \begin{bmatrix}\boldsymbol{I} & \boldsymbol{J}^\intercal \\ & \boldsymbol{J}\end{bmatrix} \bowtie \begin{bmatrix}\begin{pmatrix}1\\2\end{pmatrix} \\ \begin{pmatrix}1\\0\end{pmatrix}\end{bmatrix}, \qquad (7.14)
\end{aligned}
$$

where its components in the MPO format would look as follows:

$$\left(\boldsymbol{P_2}^{(3,3)}\right)_{\substack{i_1,\dots,i_6\\ j_1,\dots,j_4}} = \boldsymbol{W_1}^{(i_1,j_1)}\boldsymbol{W_2}^{(i_2,j_2)}\boldsymbol{W_3}^{(i_3)}\boldsymbol{W_1}^{(i_4,j_3)}\boldsymbol{W_2}^{(i_5,j_4)}\boldsymbol{W_3}^{(i_6)}. \tag{7.15}$$

The same holds for the restriction operator,

$$\boldsymbol{R_D}^{(N_1\dots,N_D)} = \bigotimes_{i=1}^{D} \boldsymbol{R_1}^{(N_i)}. \tag{7.16}$$

Again, for $D = 2$ and $N = 3$ on both directions, the MPO components are given by

$$\left(\boldsymbol{R_2}^{(3,3)}\right)_{\substack{i_1,\dots,i_4\\ j_1,\dots,j_6}} = \boldsymbol{W_1}^{(i_1,j_1)}\boldsymbol{W_2}^{(i_2,j_2)}\boldsymbol{W_3}^{(j_3)}\boldsymbol{W_1}^{(i_3,j_4)}\boldsymbol{W_2}^{(i_4,j_5)}\boldsymbol{W_3}^{(j_6)}. \tag{7.17}$$

The case of restriction by injection in higher dimensions is straightforward. In the previous section we saw that in one dimension one must contract the last two matrices, so that only odd numbered elements remain. Take the two-dimensional case, i.e., the MPS solution represents a matrix. In this case, one must choose the odd numbered elements of each odd row, which translates into contracting the last **two** pairs of matrices into two larger matrices, instead of just the last pair of matrices. The extension to $D$ dimensions follows the same reasoning (contracting the last $D$ pairs of matrices).

With these elements available, we can proceed with an initial implementation of a V-cycle in MPS format.

## 7.3 Implementation: Multigrid in MPS format

Algorithm 6 translates the V-cycle Algorithm 5 from Section 7.1 with the use of the MPS/MPO concepts introduced in the previous section.

Whether or not one should compress the solution MPS in lines 3 and 14 depends on the smoother used. Since we will be using DMRG as smoother, the bond dimension is automatically adapted to achieve a certain accuracy, so no compression should be required.[1] The compression in line 12 is not marked as optional since the addition necessarily increases the bond dimension, and for now it also solves the issue of the increased bond dimension caused by the interpolation operator acting on $\boldsymbol{x}^{(N)}$.

The elevated cost of the algorithm using DMRG as smoother should be justified by an improved convergence compared to using DMRG directly on the large system. This is important to keep in mind, since one V-cycle with, for instance, ten different grid levels (say, from $N = 11$ to $N = 2$) performs in total $10\nu_1 + 10\nu_2$ DMRG iterations (with different problem sizes), so the original problem should exhibit a structure that necessarily requires the coarsening approach in order to be solved adequately. Such a scenario is presented in [5].

Coarsening and interpolating an MPS or MPO is very cheap (at most, products of MPO times MPS), and compression is also cheaper than DMRG smoothing. Furthermore, recall

[1] Nevertheless, it might happen that DMRG overestimates the bond dimension of the solution, in which case compression would be beneficial.

**Algorithm 6:** vcycle_mps - MPS/MPO V-cycle Scheme (Recursive Definition)

**input** : $\boldsymbol{A}^{(N)} \in \text{MPO}_D$, $\boldsymbol{b}^{(N)} \in \text{MPS}_Q$, $\boldsymbol{x}^{(N)} \in \text{MPS}_{L'}$.
**output**: $\boldsymbol{x}^{(N)} \in \text{MPS}_L$.

1 **while** *not converged* **do**
2 relax $\nu_1$ times on $\boldsymbol{A}^{(N)}\boldsymbol{x}_e^{(N)} = \boldsymbol{b}^{(N)}$ with initial guess $\boldsymbol{x}^{(N)}$;
3 $\boldsymbol{x}^{(N)} \leftarrow \mathcal{P}_\varepsilon(\boldsymbol{x}^{(N)})$; [optional] `// Compression with accuracy` $\varepsilon$
4 **if** $\Omega^{(N)} =$ *coarsest grid* **then**
5 go to line 13;
6 **else**
7 $\boldsymbol{b}^{(N-1)} \leftarrow \boldsymbol{R}^{(N)}(\boldsymbol{b}^{(N)} - \boldsymbol{A}^{(N)}\boldsymbol{x}^{(N)})$;
8 $\boldsymbol{x}^{(N-1)} \leftarrow 0$ (right-normalized);
9 $\boldsymbol{x}^{(N-1)} \leftarrow$ vcycle_mps$(\boldsymbol{A}^{(N-1)}, \boldsymbol{b}^{(N-1)}, \boldsymbol{x}^{(N-1)})$;
10 **end**
11 correct $\boldsymbol{x}^{(N)} \leftarrow \boldsymbol{x}^{(N)} + \boldsymbol{P}^{(N)}\boldsymbol{x}^{(N-1)}$;
12 $\boldsymbol{x}^{(N)} \leftarrow \mathcal{P}_\varepsilon(\boldsymbol{x}^{(N)})$;
13 relax $\nu_2$ times on $\boldsymbol{A}^{(N)}\boldsymbol{x}_e^{(N)} = \boldsymbol{b}^{(N)}$ with initial guess $\boldsymbol{x}^{(N)}$;
14 $\boldsymbol{x}^{(N)} \leftarrow \mathcal{P}_\varepsilon(\boldsymbol{x}^{(N)})$ [optional] ;
15 **end**

that the complexity of DMRG is dominated by the size of the bond dimensions of $\boldsymbol{x}$ and $\boldsymbol{A}$, and is only linear in the problem size $N$. However, the coarser the grid, the smaller the upper bound for the solution bond dimension is (for $N = 2$ the bond dimension of $\boldsymbol{x}$ is $L \leq 2$, see Theorem 2.1), which motivates an implementation of a FMG scheme in this format, which starts at the coarsest level, and moves up recursively. This can also potentially provide a good initial guess for the MPS solution at the finest level. The FMG algorithm in MPS format is presented as Algorithm 7.

The FMG algorithm tells us to restrict $\boldsymbol{b}$ until the coarsest level and from there call the V-cycle $\nu$ times, interpolate the solution to the next finer level and repeat the procedure recursively. Compressing $\boldsymbol{b}$ (line 6) is not needed if injection is used as method of restriction.

## 7.4 Discussion

Our tensor toolbox was extended to include the different multigrid operators described in Section 7.2.1, and both Algorithms 6 and 7 were implemented on MATLAB. In what follows, we choose DMRG as smoother with only one smoothing iteration at each level ($\nu_1 = \nu_2 = 1$), and we perform only one V-cycle MPS iteration in the FMG algorithm ($\nu = 1$). The correctness of the algorithm was verified by comparing our implementation with a full matrix-vector V-cycle implementation using standard iterative schemes as smoothers in both cases (Jacobi or Gauss-Seidel, then switching back to DMRG in the MPS implementation). With appropriate compression, the results were equal up to machine precision (for moderate problem size $N$).

The solution of Poisson's equation (6.16) illustrates well the different elements that make

**Algorithm 7:** fmg_mps - MPS/MPO Full Multigrid Scheme (Recursive Definition)

**input** : $\boldsymbol{A}^{(N)} \in \text{MPO}_D$, $\boldsymbol{b}^{(N)} \in \text{MPS}_Q$.
**output**: $\boldsymbol{x}^{(N)} \in \text{MPS}_L$.

1 **while** *not converged* **do**
2  **if** $\Omega^{(N)} =$ *coarsest grid* **then**
3  $\boldsymbol{x}^{(N)} \leftarrow 0$ and go to line 10;
4  **else**
5  $\boldsymbol{b}^{(N-1)} \leftarrow \boldsymbol{R}^{(N)}\boldsymbol{b}^{(N)}$;
6  $\boldsymbol{b}^{(N-1)} \leftarrow \mathcal{P}_\varepsilon(\boldsymbol{b}^{(N-1)})$; [optional] $\boldsymbol{x}^{(N-1)} \leftarrow$ fmg_mps$(\boldsymbol{A}^{(N-1)}, \boldsymbol{b}^{(N-1)})$;
7  **end**
8  correct $\boldsymbol{x}^{(N)} \leftarrow \boldsymbol{P}^{(N)}\boldsymbol{x}^{(N-1)}$;
9  $\boldsymbol{x}^{(N)} \leftarrow \mathcal{P}_\varepsilon(\boldsymbol{x}^{(N)})$;
10  $\boldsymbol{x}^{(N)} \leftarrow$ vcycle_mps$(\boldsymbol{A}^{(N)}, \boldsymbol{b}^{(N)}, \boldsymbol{x}^{(N)})$ $\nu$ times;
11 **end**

up the FMG-MPS algorithm. We take a 2D-Laplacian (which can be rediscretized at each level following its 1D-representation Eq. (2.13)), with a fine mesh of $2^6$ ($N = 6$) points per dimension. As a restriction operation for $\boldsymbol{b}$ and $\boldsymbol{x}$ we use simple injection, and we use the interpolation operator (7.6) in 2 dimensions to go to finer grids.

The first problem encountered with the algorithm has to do with the bond dimension of the solution MPS $\boldsymbol{x}$ at coarse levels. We argued that the smaller $N$ is, the smaller the upper bound for the bond dimension of $\boldsymbol{x}$. But we performed the following experiment: we construct a system $\boldsymbol{Ax} = \boldsymbol{b}$ with $\boldsymbol{x}$ chosen as an MPS with random entries and fixed bond dimension $L = 3$, and set $\boldsymbol{b} \leftarrow \boldsymbol{Ax}$. Fig. 7.4 illustrates the behavior of the bond dimension after each of the V-cycle is carried out at the different levels of coarseness. The solution $\boldsymbol{x}^N$ at a certain coarse level does not have a bond dimension equal to 3. Only when we go back to the finest level, the DMRG smoother is able to detect that the solution indeed has bond dimension 3, but in the meanwhile we have performed expensive operations on the coarser grids. The FMG algorithm took 13.7 s to converge with a relative residual norm of $3.83 \times 10^{-8}$, while only two iterations of DMRG on the large system resulted in an equivalent residual taking only 0.167 s.

We also mentioned that starting from coarser grids and then moving to finer grids can provide good initial guesses for the problem on the finest grid. Nevertheless, after carrying out several experiments we observed that the convergence properties of DMRG seem to be independent of the shape of the initial guess. Furthermore, since DMRG adaptively chooses the bond dimension of the solution, it is always preferable to start with an initial guess of bond dimension $L = 1$, which would not be the case in our FMG example (the bond dimension at the second finest level $N = 5$, which could be used as a good initial guess, is equal to 26).

For standard problems (such as Poisson's equation (6.16)), DMRG acts as a solver at each level, rather than as a smoother. Fig. 7.2 illustrates this point, where we have taken a 1D-Laplacian with right hand side $\boldsymbol{b} = 1$ and $2^{10}$ discretization points at the finest level,

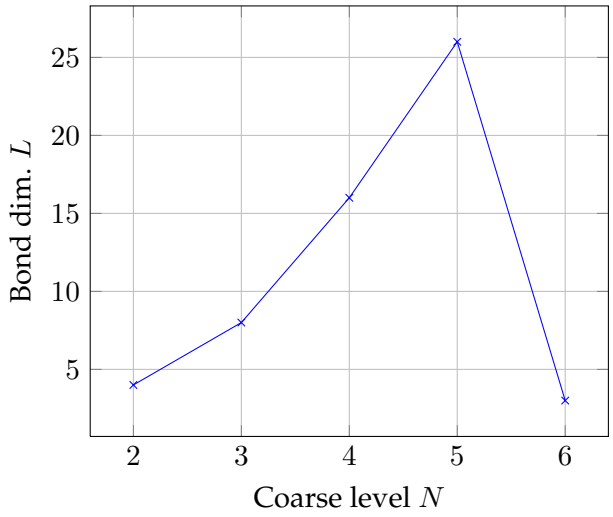



Figure 7.1: Evolution of the bond dimension $L$ of $\boldsymbol{x}$ after each V-cycle is performed inside the FMG.

and we perform one FMG iteration. As compression tolerance we take $\varepsilon = 10^{-8}$. We see from the small residuals that DMRG practically solves the problem at each level, and this requires a large computational effort. One FMG iteration took 8.42 s, while using DMRG directly on the finest grid gives a similar residual norm in 4 iterations requiring only 0.469 s.

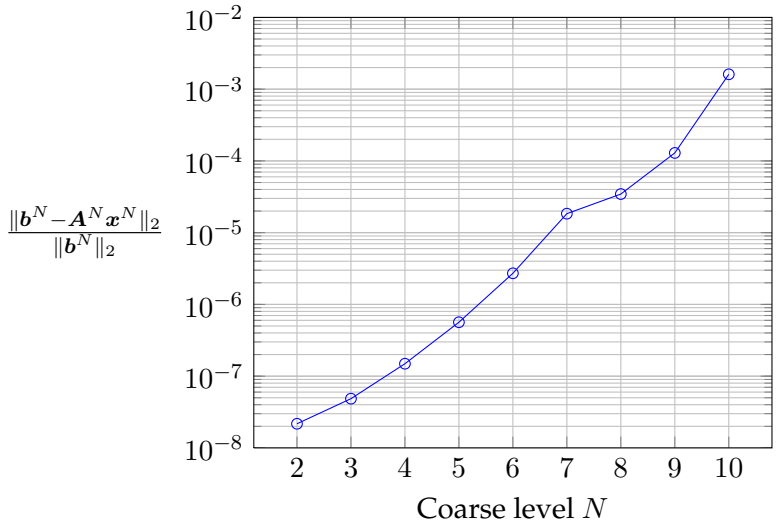



Figure 7.2: Used as a smoother, DMRG is actually solving the problem at each level, which involves too much computational effort.

This problem can be overcome by using a DMRG iteration with lower truncation precision at coarser levels (or, similarly, simple ALS iterations with small bond dimension for the solution subspace), but we are yet to find a scenario where this proves advantageous

over pure DMRG.

## 7.5 Conclusions

We presented the different multigrid elements required to implement a V-cycle and a FMG algorithm in $D$ dimensions using the MPS formalism, and we proposed and implemented these two algorithms. The convergence properties of the resulting FMG implementation are completely dominated by the effect of the DMRG smoother for our various tests, and we were unable to reproduce a scenario where the use of coarser grids provides advantages over using DMRG directly on the finest grid. We expect, however, that the discussions presented in this chapter offer an insight into the use of multigrid techniques in MPS format.

# 8 Conclusions

Various algorithms for the solution of linear systems of equations and the calculation of approximate inverses in MPS/MPO format were presented. ALS and DMRG are now well established methods to solve problems in high dimensions with relative ease, and we studied how to apply them to different examples arising from the discretization of PDEs. The scope of these algorithms is, however, much wider. The large scope of problems that can potentially be translated and solved in this formalism requires new mathematical tools and algorithms to handle them properly. This motivated our implementation of the Lanczos and FMG algorithms, as well as our study of preconditioners and iterative schemes.

For the examples treated, DMRG shows the best overall performance, and we presented several scenarios where it can be effectively used for high-dimensional problems, both for solving linear systems and findin approximate inverses. We discussed some possible improvements and illustrated some of its drawbacks. Several other algorithms available in the low-rank tensor community perform better than our implementation, but they were useful to us for the subsequent chapters. Both Lanczos and FMG algorithms as presented have several drawbacks, but our ideas might shed new light on the translation of Krylov subspace and multigrid algorithms to low-rank tensor formats. Likewise, we hope that our discussions of preconditioners and iterative schemes will be useful for future algorithms.

# Bibliography


[1] W.L. Briggs, V.E. Hemson, and S.F. McCormick. *A Multigrid Tutorial*. SIAM, 2 edition, 2000.

[2] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta numerica*, 13(1):147–269, 2004.

[3] C.S. Chien and S.L. Chang. Application of the Lanczos algorithm for solving the linear systems that occur in continuation problems. *Numer. Linear Algebra Appl.*, 10:335–355, 2003.

[4] H. de Sterck and K. Miller. An adaptive algebraic multigrid algorithm for low-rank canonical tensor decomposition. *SIAM J. Sci. Comp.*, 35(1):B1–B24, 2013.

[5] M. Dolfi, B. Bauer, M. Troyer, and Z. Ristivojevic. Multigrid algorithms for tensor network states. *Phys. Rev. Lett.*, 109(2):020604, 2012.

[6] S.V. Dolgov. TT-GMRES: on solution to a linear system in the structured tensor format. *Russ. J. Numer. Annal. Math. Modelling*, 28(2):149–172, 2013.

[7] S.V. Dolgov and D.V Savostyanov. Alternating minimal energy methods for linear systems in higher dimensions. part i: SPD systems,. *arXiv preprint arXiv:1301.6068*, 2013.

[8] L. Giraldi, A. Nouy, and G. Legrain. Low-rank approximate inverse for preconditioning tensor-structured linear systems. `arxiv.org/pdf/1304.6004`, 2013.

[9] L. Grasedyck. Existence and computation of low kronecker-rank approximations for large linear systems of tensor product structure. *Computing*, 72(3-4):247–265, 2004.

[10] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitt.*, 36(1):53–78, 2013.

[11] T. Huckle and K. Waldherr. Subspace iteration methods in terms of Matrix Product States. *Proc. Appl. Math. Mech.*, 12:641–642, 2012.

[12] T. Huckle, K. Waldherr, and T. Schulte-Herbrüggen. Computations in quantum tensor networks. *Linear Algebra and its Applications*, 438:750–781, 2013.

[13] T. Huckle, K. Waldherr, and T. Schulte-Herbrüggen. Exploiting matrix symmetries and physical symmetries in matrix product states and tensor trains. *Linear and Multilinear Algebra*, 61(1):91–122, 2013.

[14] V.A. Kazeev and B.N. Khoromskij. On explicit QTT representation of Laplace operator and its inverse. *SIAM Journal on Matrix Analysis and Applications*, 33:742–758, 2012.

[15] V.A. Kazeev, B.N. Khoromskij, and E.E. Tyrtyshnikov. Multilevel Toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity. *SIAM J. Sci. Comput.*, 35(3):A1511–A1536, 2013.

[16] B.N. Khoromskij. Tensor-structured preconditioners and approximate inverse of elliptic operators in $\mathbb{R}^d$. *J. Constructive Approx.*, 30(3):599–620, 2009.

[17] D. Kressner and C. Tobler. Low-rank tensor Krylov subspace methods for parametrized linear systems. *SIAM J. Matrix Anal. Appl.*, 32(4):1288–1316, 2011.

[18] D. Kressner and C. Tobler. Preconditioned low-rank methods for high-dimensional elliptic pde eigenvalue problems. *Comput. Meth. in Appl. Math.*, 11(3):363–381, 2011.

[19] I. Oseledets and S.V. Dolgov. Solution of linear systems and matrix inversion in the TT-format. *SIAM J. Sci. Comput.*, 34:A2718–A2739, 2012.

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

[21] I.V. Oseledets. Constructive representation of functions in low-rank tensor formats. *Constr. Appros.*, 37(1):1–18, 2013.

[22] I.V. Oseledets and E.E. Tyrtyshnikov. Breaking the curse of dimensionality, or how to use SVD in many dimensions. *SIAM J. Sci. Comput.*, 31(5):3744–3759, 2009.

[23] I.V. Oseledets, E.E. Tyrtyshnikov, and N.L. Zamarashkin. Tensor-Train ranks for matrices and their inverses. *Comput. Meth. Appl. Math.*, 11(3):394–403, 2011.

[24] C.C. Paige and M.A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[25] M. Papadrakis and S. Smerou. A new implementation of the Lanczos method in linear problems. *Int. J. Numer. Methods Eng.*, 29:141–159, 1990.

[26] D. Perez-Garcia, F. Verstraete, M.M. Wolf, and J.I. Cirac. Matrix product state representations. *Quantum Info. Comput.*, 7(5):401–430, 2007.

[27] B. Pirvu, V. Murg, J.I. Cirac, and F. Verstraete. Matrix product operator representations. *New J. Phys.*, 12(2):025012, 2010.

[28] A. Quarteroni and F. Saleri. *Scientific Computing with MATLAB and Octave*. Springer, 2nd edition, 2006.

[29] U. Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326:96–192, 2011.

[30] F. Verstraete, D. Porras, and J.I. Cirac. DMRG and periodic boundary conditions: a quantum information perspective. *Phys. Rev. Lett.*, 93(22):227205, 2004.

[31] G. Vidal. Efficient classical simulation of slightly entangled quantum computations. *Phys. Rev. Lett.*, 91(14):147902, 2003.

[32] K. Waldherr. *Numerical Linear And Multilinear Algebra in Quantum Control and Quantum Tensor Networks*. PhD thesis, Technische Universität München, 2013.

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

[34] N.L. Zamarashkin and I.V. Oseledets. The tensor structure of the inverse of a banded Toeplitz matrix. *Dokl. Math.*, 80(2):669–670, 2009.