Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter known, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Eric Yixuan Xu 9 April, 2025

Compression of Tensor Train Cores via Tucker Decomposition, and Applications to the Density Matrix Renormalization Group

By

Eric Yixuan Xu

Elizabeth Newman, Ph.D.
Advisor

Francesco Evangelista, Ph.D.
Co-Advisor

Department of Mathematics

Elizabeth Newman, Ph.D.
Advisor

Francesco Evangelista, Ph.D.
Co-Advisor

Cosmin Pohoata, Ph.D.
Committee Member

2025

Compression of Tensor Train Cores via Tucker Decomposition, and Applications to the Density Matrix Renormalization Group

By

Eric Yixuan Xu

Elizabeth Newman, Ph.D.
Advisor

Francesco Evangelista, Ph.D.
Co-Advisor

An abstract of
a thesis submitted to the Faculty of
Emory College of Arts and Sciences of Emory University
in partial fulfillment of the requirements for the degree of
Bachelor of Science with Honors

Department of Mathematics

2025

Abstract

Compression of Tensor Train Cores via Tucker Decomposition, and Applications to the Density Matrix Renormalization Group

By Eric Yixuan Xu

The tensor train (TT), or matrix product state (MPS), is a tensor decomposition which aims to avoid the curse of dimensionality in high-dimensional problems by compressing a tensor of arbitrary order into a chain of contractions of order-3 tensors, which we call the TT cores. In this work, we further compress the TT cores by means of a Tucker decomposition, or higher-order singular value decomposition (HOSVD). We perform error analysis for putting a TT into this form, and provide operations analogous to those for the uncompressed form, and discuss several properties. We showcase these theoretical results by compressing solutions of the density matrix renormalization group (DMRG) for the one-dimensional Hubbard model.

Compression of Tensor Train Cores via Tucker Decomposition, and Applications to the Density Matrix Renormalization Group

By

Eric Yixuan Xu

Elizabeth Newman, Ph.D.
Advisor

Francesco Evangelista, Ph.D.
Co-Advisor

A thesis submitted to the Faculty of
Emory College of Arts and Sciences of Emory University
in partial fulfillment of the requirements for the degree of
Bachelor of Science with Honors

Department of Mathematics

2025

Acknowledgments

This thesis and the extent of my development as a student would not have been possible without the immense generosity of Dr. Elizabeth Newman and Dr. Francesco Evangelista, who have supported me far beyond just advising the work contained here. Thanks to many, many others, including Dr. Cosmin Pohoata, Dr. Suresh Venapally, Dr. Yiran Wang, Dr. Difeng Cai, Dr. Jose Soria, Dr. Richard Himes, Dr. Peter Wakefield, and the Evangelista group for enlightening conversations, guidance, and enabling me to see new joys in math, science, and my other pursuits. Thanks to my family for their unwavering support in all aspects.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction and Background

Among some of the most expensive scientific computing methods are quantum chemical methods, particularly in molecular electronic structure and condensed matter physics. Often, these problems are large and high-dimensional such that naive approaches become quickly intractable. However, it is also often the case that these problems contain lots of structure, and recognizing and exploiting this structure opens the path to more efficient algorithms. A prime example of such an algorithm is the density matrix renormalization group (DMRG), which has become state-of-the-art in solving the aforementioned electronic problems. The key insight behind this algorithm is how it represents the wavefunction of a system: it uses what is known in the mathematics literature as the tensor train (TT), and in the physics/chemistry literature as the matrix product state.

This representation is compact (in the sense that it takes far less storage than is needed naively), and achieves this by exploiting physical features of certain systems, namely locality in low-energy lattice systems. Because of the empirical power of this representation, the DMRG and its extensions have garnered significant interest. More recently, this algorithm has been extended beyond lattices to molecular systems with extremely promising results, such as in [9].

Despite this success, the DMRG is still an expensive algorithm, and suffers especially when the system being studied leaves the regime of local interactions. The purpose of this work is to investigate the tensor train representation upon compressing it further, and observe how the DMRG is affected.

We first outline the main ideas from linear and multilinear algebra which are essential to our discussion—we begin with some concepts in matrix theory, then move to a discussion of tensors and the tensor train. We then proceed to a physical background in order to acquaint ourselves with some quantum mechanics, since the tensor train first appeared in that context, and conclude with a more detailed discussion of the DMRG.

1.1 Mathematical Background

Tensors arise naturally and ubiquitously in physics and chemistry, and have become essential tools in modern computation, with applications in statistics, machine learning, and data science. Taking the view of tensors as multi-way arrays or higher-dimensional matrices, it appears natural to study tensor decompositions in analogy to the more classical study of matrix decompositions. Keeping with this analogy, one might hope to reveal many of the same properties: existence and uniqueness, approximation error, and structure in general. Paired with these naturally also comes the computational component, i.e., algorithms with which one can compute these decompositions, and reveal desired properties of a given tensor.

It turns out, rather unsurprisingly, that these generalizations are no trivial task. While tensors are indeed powerful, they have several key drawbacks. In the theoretical realm, they fail to generalize many nice results in matrix theory. An archetypal example of this is tensor rank (in the sense of a linear combination of rank-1 tensors), which is not only NP-hard to compute, but also depends on the field over which

one works, i.e., the tensor rank over the reals is not necessarily the same as over the complex numbers, while in matrix theory these are always the same. In the practical realm, the size of a tensor grows fast with the number of dimensions, making it quickly intractable to store, much less perform operations on, large tensors.

In this section, we outline several ideas from linear and multilinear algebra which are essential to our discussion. We begin with some concepts in matrix theory, then move to a discussion of tensors and the tensor train.

1.1.1 Matrices and the Singular Value Decomposition

We first define the matrix rank:

Definition 1.1.1 (**Matrix Rank**). *Let $A \in \mathbb{C}^{m\times n}$, with*

$$A = \sum_{i=1}^{r} a_i b_i^*$$

where $a_i \in \mathbb{C}^m$ and $b_i \in \mathbb{C}^n$ for $i \in [r]$. If r is minimal, we say that A has rank r. In particular, r is the minimal number of rank-one matrices needed to sum to A.

Here, we have used the fact that rank-one matrices can be written as the outer product of vectors. Then, we define the matrix singular value decomposition:

Definition 1.1.2 (**Matrix SVD**). *Let $A \in \mathbb{C}^{m\times n}$. The matrix singular value decomposition is:*

$$A = U\Sigma V^*$$

where $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is of the form

$$\Sigma = \begin{bmatrix} \Sigma_r & \\ & 0 \end{bmatrix}, \qquad \Sigma_r = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$$

where σ_i, the singular values, are positive, real numbers s.t. $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$.

Notice that the SVD is rank-revealing, i.e., the rank of A is given by r, the number of nonzero entries is Σ. We now state an important result in matrix low-rank approximation which utilizes the SVD.

Theorem 1.1.1 (**Eckart–Young–Mirsky**). *Let $A, B \in \mathbb{C}^{m\times n}$ with $\operatorname{rank}(B) = k \leq \operatorname{rank}(A) = r$. Let A_k be the best rank-k approximation of A. Then,*

$$\|A - B\|_F^2 \geq \|A - A_k\|_F^2 = \sum_{i=k+1}^{r} \sigma_i^2$$

where $\|\cdot\|_F$ is the Frobenius norm and σ_i is the ith singular value of A.

The theorem stated with proof can be found in [7]. Note that there exists a similar result for the L^2-norm, but we primarily concern ourselves with the Frobenius norm here because it generalizes easily to tensors. Further, we'd like to work in an inner product space (and even further, in a Hilbert space), so we also introduce the following definition:

Definition 1.1.3 (**Frobenius Inner Product**). *Let $A, B \in \mathbb{C}^{m\times n}$. The Frobenius inner product $\langle \cdot, \cdot \rangle_F$ is defined as:*

$$\langle A, B \rangle_F = \sum_{i=1}^{m}\sum_{j=1}^{n} \bar{a}_{ij} b_{ij} = \operatorname{trace}(A^* B)$$

One can show that $\langle \cdot, \cdot \rangle_F$ does indeed satisfy the properties of an inner product and that it induces the Frobenius norm. One can view this as vectorizing each matrix and taking the Euclidean norm, but from this perspective the convenient trace property is not so obvious. Notice that if the field we work over is (the elements of our matrices are in) $\mathbb{R}$ or $\mathbb{C}$, which in this work is assumed, equipping our vector space with the Frobenius inner product makes our space a Hilbert space.

We now introduce an important matrix algorithm which appears as a subroutine in many of the tensor algorithms to be discussed: the Leading Left Singular Vectors (LLSV) algorithm. The procedure as defined below is borrowed from [2].

Algorithm 1: Leading Left Singular Vectors (LLSV)

Input: $Y \in \mathbb{R}^{m \times n}$, and one of: target rank $r \leq \min\{m, n\}$ or absolute error tolerance $\varepsilon \geq 0$

Output: $W \in \mathbb{R}^{m \times r}$ where W semiunitary, and $\text{ERR} \equiv \|(I - WW^\top)Y\|_F$ where $\|W^\top Y\|_F$ maximal (if ε provided, r maximal s.t. $\text{ERR} \leq \varepsilon$)

function $[W, \text{ERR}] = \text{LLSV}(Y, r \text{ or } \varepsilon)$
  $[U, \Sigma, V^*] \leftarrow \text{SVD}(Y)$
  if ε given:
    $r \leftarrow \min\left\{r \in [m] \mid \sum_{i=r+1}^{m} \sigma_i^2 \leq \varepsilon^2\right\}$
  end if
  $W \leftarrow U_{:,:r}$
  $\text{ERR} \leftarrow \sum_{i=r+1}^{m} \sigma_i^2$
  return $[W, \text{ERR}]$
end function

In practice, when implementing LLSV, one does not need to store V^*, and can use the reduced form of the SVD, i.e., enforce Σ to be square, and thus potentially compute fewer columns of U. Additionally, one can choose to keep the singular value matrix Σ and store it in a vector of length r. The reason one might make this choice becomes apparent in the discussion of the TTSVD. Notice also that incorporated into this algorithm is a way to perform low-rank approximation, taking advantage of the Theorem 1.1.1.

1.1.2 Tensor Preliminaries

The primary source for much of this section, including notational conventions, is [2] and [9]. One should consult these texts for more details if desired. Though there are several definitions for a tensor, for the purposes of this work we primarily use the following:

Definition 1.1.4 (**Tensor**). *A tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{F}^{n_1 \times n_2 \times \cdots \times n_d}$ *is a d-way array, where d is called the order of* $\boldsymbol{\mathcal{X}}$.

Notice that with this definition, we can define a matrix to be a tensor of order 2, and a vector to be a tensor of order 1. In this work, we will assume $\mathbb{F} = \mathbb{R}$, but in general we can have $\mathbb{F}$ be other fields, e.g., $\mathbb{C}$.

We now introduce a generalization of the familiar matrix-matrix product and matrix-vector product: the tensor contraction. As with the lower order cases, two tensors (of possibly differing order, as in the case of a matrix-vector product) admit contractions along indices of matching dimension. Since in higher orders we lack the convenient right- and left-multiplication notation that we have with matrices and vectors, notation quickly becomes cumbersome. To remedy this fact, it is often convenient to use a graphical notation, as in Figure 1.1.

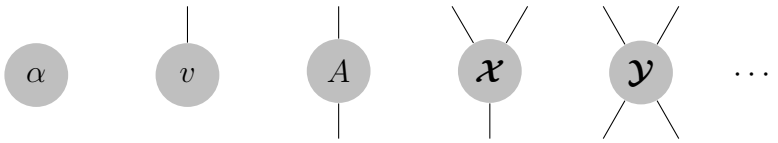


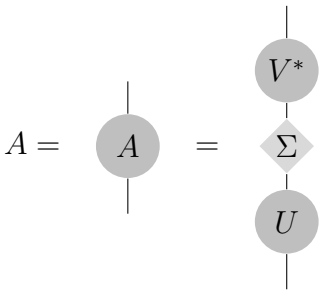
Figure 1.1: Tensors written in graphical notation. Here, the number of legs denotes the order of the tensor. That is, α is a scalar, v is a vector, A is a matrix, $\boldsymbol{\mathcal{X}}$ is an order-3 tensor, etc.

When joining tensors to be contracted along some compatible indices, we simply connect the legs. For instance, if we wish to write the matrix-vector product $Ax = b$, we would write:

$$Ax = \quad = \quad = b \tag{1.1}$$

As another example, if we wish to write out a matrix in SVD form, $A = U\Sigma V^*$, we

can write:



(1.2)

In later sections, especially in the discussion of DMRG, most of the operations will be written entirely in this notation.

As a generalization of the matrix rank and the SVD as a rank-revealing factorization, we introduce the multilinear rank and the Tucker decomposition. We also introduce the sequentially truncated higher-order SVD (ST-HOSVD) algorithm, a direct algorithm which allows us to compute a Tucker decomposition with specified multilinear rank or error (though iterative methods to do this also exist).

Definition 1.1.5 (**Multilinear Rank**). *Let $\boldsymbol{\mathcal{X}}$ be a d-order tensor. The multilinear rank of $\boldsymbol{\mathcal{X}}$ is given by:*

$$\text{multirank}(\boldsymbol{\mathcal{X}}) = (r_1, r_2, \ldots, r_d), \qquad r_k = \text{rank}(X_{(k)})$$

where $k \in [d]$ and $X_{(k)}$ denotes the mode-k unfolding of $\boldsymbol{\mathcal{X}}$.

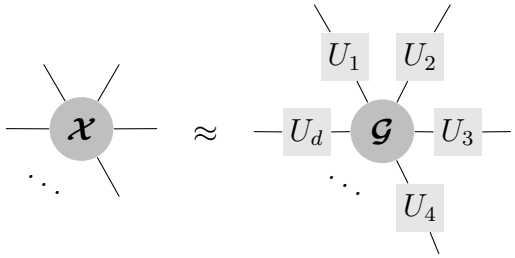
The mode-k unfolding $X_{(k)}$ of a tensor $\boldsymbol{\mathcal{X}}$, informally, is a particular matricization of $\boldsymbol{\mathcal{X}}$ where $X_{(k)} \in \mathbb{R}^{n_k \times N_k}$, where N_k is given by the product of all other dimensions. Note that there exists a natural ordering of the N_k column vectors, and for consistency we assume in this work that all unfoldings abide by this ordering.

Definition 1.1.6 (**Tucker Decomposition**). *Let $\boldsymbol{\mathcal{X}}$ be a d-order tensor with dimensions $n_1, n_2, \ldots, n_d$. Given target multilinear rank $r = (r_1, r_2, \ldots, r_d)$, the rank-$r$*

Tucker decomposition of $\boldsymbol{\mathcal{X}}$ *is:*

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_1 U_1 \times_2 U_2 \cdots \times_d U_d$$

where $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_d}$*, and* $U_i \in \mathbb{R}^{n_i \times r_i}$ *are semiunitary matrices. Graphically, we can write:*



More directly, we can write:

$$\boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, i_d) = \sum_{j_1=1}^{r_1} \sum_{j_2=1}^{r_2} \cdots \sum_{j_d=1}^{r_d} \boldsymbol{\mathcal{G}}(j_1, j_2, \ldots, j_d) \cdot \prod_{l}^{d} U_l(j_l, i_l)$$

The mode-i product $\times_i$ denotes what is effectively a matrix-matrix multiplication using the mode-i unfolding of $\boldsymbol{\mathcal{G}}$, followed by a reshaping back into a tensor which has the same dimensions, except r_i is replaced by n_i. Note that the Tucker decomposition is not in general unique, since we can insert the identity between the core and factor matrices. Algorithm 2 introduces this procedure.

Notice that by matricizing and using the LLSV subroutine, the way we control the rank/error is through the SVD and Theorem 1.1.1. Later on, we will use this algorithm directly in our compression of a TT. There is a more naive HOSVD algorithm which does not truncate $\boldsymbol{\mathcal{X}}$ as factor matrices are computed and instead compresses the full tensor directly to the core $\boldsymbol{\mathcal{G}}$ at the end, but it has worse computational complexity while giving the same result. Thus, we generally choose to use the ST-HOSVD. Now, as alluded to before, we can define a Hilbert space in which tensors are our vectors.

Algorithm 2: Sequentially Truncated Higher-Order SVD (ST-HOSVD)

Input: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, and one of: target multirank $r = (r_1, r_2, \ldots, r_d)$ with $r_i \leq n_i$ or absolute error tolerance $\varepsilon \geq 0$

Output: $\boldsymbol{\mathcal{G}} \times_1 U_1 \times_2 U_2 \cdots \times_d U_d$ where $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_d}$ and $U_i \in \mathbb{R}^{n_i \times r_i}$ (if ε provided, we compute r satisfying the error bound on the fly)

```
function [G, U_1, U_2, ..., U_d] = ST-HOSVD(X, r or ε)
    if ε is given:
        ε̄ ← ε√d‖X‖
    end if
    G ← X
    for i ∈ [d]:
        U_i ← LLSV(G_(i), r_i or ε̄)
        G ← G ×_i U_i^⊤
    end for
end function
```

Definition 1.1.7 (**Tensor Inner Product**). *Let $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$. We define the tensor inner product $\langle \cdot, \cdot \rangle$ to be:*

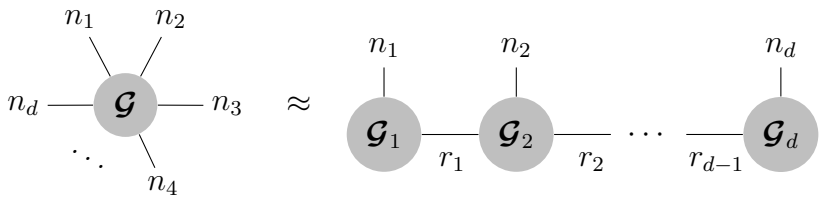
$$\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \rangle = \sum_{i_1}^{n_1} \sum_{i_2}^{n_2} \cdots \sum_{i_d}^{n_d} x_{i_1 i_2 \cdots i_d} \cdot y_{i_1 i_2 \cdots i_d}$$

One can see how this inner product generalizes Definition 1.1.3, and similarly, one can verify that this satisfies the definition of an inner product and that equipping $\mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ with it makes it a (finite-dimensional) Hilbert space. We induce the norm $\|\boldsymbol{\mathcal{X}}\| = \sqrt{\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}} \rangle}$ (and the metric $d(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}) = \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}\|$), which we will make use of later on. One can view this inner product as a special case of tensor contraction.

1.1.3 Tensor Train

We now begin our discussion of the tensor train (TT), or, in the chemistry and physics literature, what is known as the matrix product state (MPS). For a more detailed overview, one can consult [12] or [2]. For more historical background, i.e., its origins in many-body quantum physics, see [15] and [14].

Definition 1.1.8 (**Tensor Train Decomposition**). *Let $\boldsymbol{\mathcal{X}}$ be a d-order tensor with dimensions $n_1, n_2, \ldots, n_d$. Given target tensor train rank $r = (r_1, r_2, \ldots, r_{d-1})$, the rank-$r$ Tucker decomposition of $\boldsymbol{\mathcal{X}}$ is:*



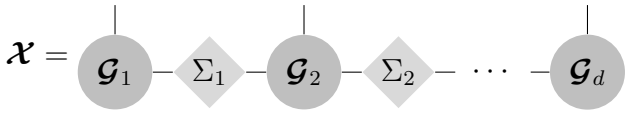
More directly, it can be written:

$$\boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, i_d) = \boldsymbol{\mathcal{G}}_1[i_1]\boldsymbol{\mathcal{G}}_2[i_2]\cdots\boldsymbol{\mathcal{G}}_d[i_d]$$

where $\boldsymbol{\mathcal{G}}_j[i_j]$ are matrices of size $r_{j-1} \times r_j$. For $j \in \{1, d\}$, these are vectors of size $1 \times r_1$ and $r_d \times 1$, respectively.

The TT can be viewed as a generalized SVD, though it may be not immediately apparent why. Observe Algorithm 3, which takes an unfactorized tensor and computes its TT form. We borrow again from [2]. From this, one can see this interpretation: for the computation of each core, we are matricizing, computing the LLSV, then reshaping.

Notice that in the way the algorithm is written, we do not factor out the singular value matrices at each iteration. However, we generally can choose to do so, outputting:



(1.3)

where each Σ_i is square, of size $r_i \times r_i$. This is also known as the canonical form, or Vidal gauge, of a TT. Having a TT in this form gives rise to a naive way of rounding, à la Eckart–Young–Mirsky, i.e., we simply truncate any of the Σ_i. As we will see later, this canonical form also allows us many computational conveniences.

Algorithm 3: Tensor Train SVD (TTSVD)

Input: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, and one of: target TT rank $r = (r_1, r_2, \ldots, r_d)$ or absolute error tolerance $\varepsilon \geq 0$

Output: TT cores $\boldsymbol{\mathcal{G}}_1, \boldsymbol{\mathcal{G}}_2, \ldots, \boldsymbol{\mathcal{G}}_d$ with TT rank r, and $\text{ERR} \leq \varepsilon \|\boldsymbol{\mathcal{X}}\|$ (if ε provided, we compute r satisfying the error bound on the fly)

function $[\boldsymbol{\mathcal{G}}_1, \boldsymbol{\mathcal{G}}_2, \ldots, \boldsymbol{\mathcal{G}}_d] = \text{TTSVD}(\boldsymbol{\mathcal{X}}, r \text{ or } \varepsilon)$
  if ε is given:
    $\bar{\varepsilon} \leftarrow \varepsilon \sqrt{d-1} \|\boldsymbol{\mathcal{X}}\|$
  end if
  $r_0 \leftarrow 1$
  $Y_1 \leftarrow \boldsymbol{\mathcal{X}}$
  for $k \in [d-1]$:
    $\bar{Y}_k \leftarrow \text{reshape}\left(Y_k, (r_{k-1} n_k\,,\; n_{k+1} n_{k+2} \cdots n_d)\right)$
    $[G_k, \varepsilon_k] \leftarrow \text{LLSV}(\bar{Y}_k, r_k \text{ or } \bar{\varepsilon})$
    $Y_{k+1} \leftarrow G_k^\top \bar{Y}_k$
    $\boldsymbol{\mathcal{G}}_k \leftarrow \text{reshape}(G_k, (r_{k-1}\,,\; n_k\,,\; r_k))$
  end for
  $\boldsymbol{\mathcal{G}}_d \leftarrow Y_d$
  $\text{ERR} \leftarrow \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}$
end function

Suppose we wish to put a TT into canonical form. Notice that given a matrix and its inverse $AA^{-1} = I$, we can insert it between two cores (assuming compatible dimensions) without affecting the overall TT, that is, we have a so-called gauge degree of freedom. We can factor out the singular value matrices as described in Figure 1.2.

We have stated previously that we are working in a Hilbert space. Thus, we introduce the Hilbert space operations for TT. The inner product should appear intuitive: we simply connect two tensor trains by their matching indices, analogously to Definition 1.1.7. Scalar multiplication is also straightforward: we simply multiply any of the cores (or singular value matrices, if we are in canonical form) by the scalar. Vector addition is a slightly more complicated. Let $C = A + B$, where A and B are

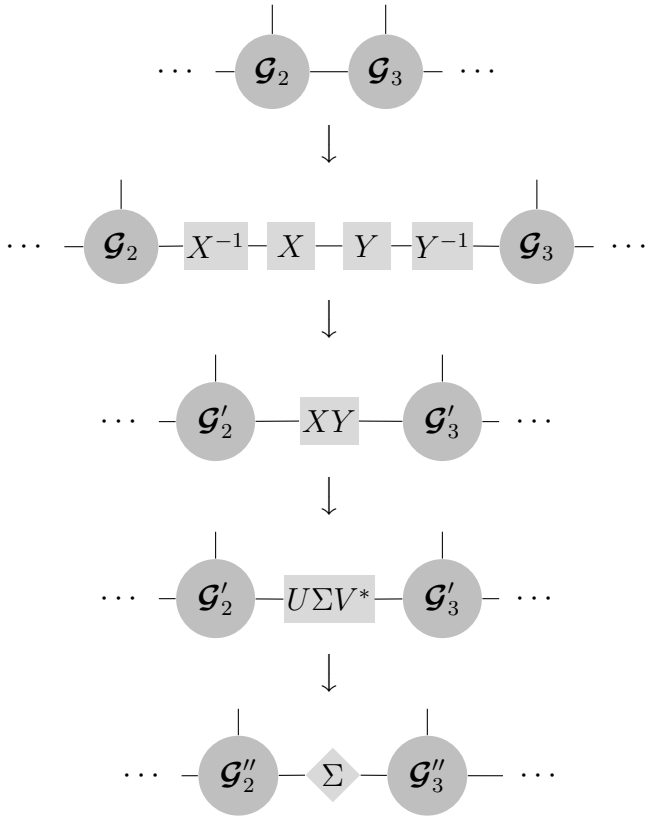
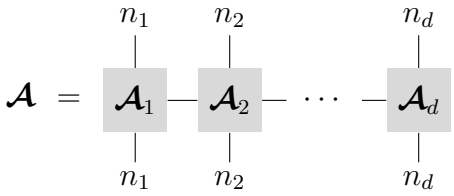


Figure 1.2: Graphical description of the TT canonicalization procedure. First, we insert the identity twice, in the form of $I = X^{-1}X = YY^{-1}$, where X and Y are the principal square root matrices of the self inner products of the left and right parts of the TT vector, respectively (noting that these self inner products are always Hermitian positive semidefinite). We then compute the SVD of $XY = U\Sigma V^*$. Multiplying matrices back into the cores, we are left with only the singular value matrix Σ linking the two cores. Performing this on each pair of cores yields the TT in canonical form.

tensor trains. We can write the cores of C in terms of the cores of A and B as follows:

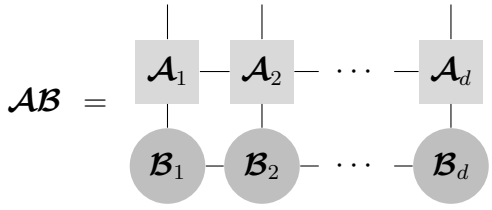
$$C_j[i_j] = \begin{cases} \begin{bmatrix} A_1[i_j] & B_1[i_1] \end{bmatrix} & \text{if } j = 1 \\ \begin{bmatrix} A_j[i_j] & 0 \\ 0 & B_j[i_j] \end{bmatrix} & \text{if } j \in \{2, 3, ..., d-1\} \\ \begin{bmatrix} A_d[i_d] \\ B_d[i_d] \end{bmatrix} & \text{if } j = d \end{cases} \tag{1.4}$$

From this, one might not be so surprised to learn that the sum has generally double the rank of the summands. Finally, we introduce the matrix analogy for tensor trains as vectors: the aptly named TT matrix (or matrix product operator). In general, we consider such a tensor $\boldsymbol{\mathcal{A}}$ to of be size $n_1 \times n_1 \times n_2 \times n_2 \times \cdots \times n_d \times n_d$. One might guess how this matrix appears in TT format:



(1.5)

Let $\boldsymbol{\mathcal{B}}$ be a TT vector of compatible dimension with $\boldsymbol{\mathcal{A}}$, i.e., of size $n_1 \times n_2 \times \cdots \times n_d$. To apply a TT matrix to a TT vector, we simply do the following, in analogy to Equation 1.1:



(1.6)

In general, TT vector addition and TT mat-vecs grow the ranks of the resulting TT vector. Before, we mentioned a way of rounding by truncating the singular value

matrices when a TT is in canonical form. Something nice about that method is that it is quite straightforward to implement. However, when we round in this way, it is not immediately obvious how overall error behaves. In particular, when we round locally, i.e., we truncate only one of our Σ_i, it is not in general clear how the entire tensor train changes. Later, in the DMRG section, we will introduce a different, more sophisticated rounding method which involves solving a nonlinear optimization problem.

1.2 Physical Background

In quantum chemistry and condensed matter, strongly correlated low-dimensional lattice systems constitute a family of rich but difficult problems. Only in a few special cases do these systems have analytical solutions, and, when treated numerically, many of these problems behave badly in the face of single-reference methods, e.g., Hartree–Fock, as a result of strong correlation. Hence, multireference algorithms, such as the density matrix renormalization group (DMRG), become important tools in the study of these systems.

1.2.1 Second Quantization

When one deals with quantum many-body systems, it can be convenient to use the language of second quantization, also known as the occupation-number representation. Here, we introduce some of the basic notation and concepts appearing in this work.

The key insight behind this formalism is the indistinguishability of particles. In particular, the many-body wavefunction is invariant, up to a phase factor, under particle exchange. We limit the discussion here to electrons, which are fermions, so our wavefunctions are antisymmetric, i.e., the phase factor appearing from particle exchange is -1.

What this indistinguishability implies is that given a multi-particle state, we cannot

ascribe a single-particle state to a given particle. Because of this, we can consider the set of single-particle states, and describe a multi-particle state by occupying the single-particle states. Let $\boldsymbol{\mathcal{F}}_1$ be the Hilbert space of fermionic single-particle states with basis $\{|\psi_i\rangle\}$. We can construct the Hilbert space of N-fermion states, which we denote $\boldsymbol{\mathcal{F}}_N$. $\boldsymbol{\mathcal{F}}_N$ has a basis $\{|\lambda_1, \lambda_2, \ldots, \lambda_N\rangle\}$ of N-particle wavefunctions (Slater determinants), i.e.,

$$|\lambda_1, \lambda_2, \ldots, \lambda_N\rangle \equiv \bigwedge_{i=1}^{N} |\psi_{\lambda_i}\rangle \tag{1.7}$$

where the $|\psi_{\lambda_i}\rangle$ are elements of the basis of $\boldsymbol{\mathcal{F}}_1$, and $\bigwedge$ denotes the exterior product. For more details on this, one can consult [1]. The λ_i denote occupied single-particle states, of which there are N by the Pauli exclusion principle. That is, since we are working with fermions, each single-particle state can have occupancy at most 1. To form the basis of $\boldsymbol{\mathcal{F}}_N$, the λ_i vary over all combinations of N elements in $\left[\dim \boldsymbol{\mathcal{F}}_1\right]$ (though $\boldsymbol{\mathcal{F}}_1$ may in general be infinite-dimensional). We will provide a concrete example later in the next section to make things clearer. Moving forward for now, we can use this definition to construct the so-called fermionic Fock space, containing all possible fermionic quantum states:

$$\boldsymbol{\mathcal{F}} \equiv \bigoplus_{i=0}^{\infty} \boldsymbol{\mathcal{F}}_i \tag{1.8}$$

It can be shown that this infinite sum converges (more specifically, that $\boldsymbol{\mathcal{F}}$ is the completion of the Hilbert direct sum), so this is still a Hilbert space, but we do not prove this in detail here. We can now introduce the creation and annihilation operators. Consider the so-called vacuum state, the vector $|\Omega\rangle$ s.t. $\text{span}\{|\Omega\rangle\} = \boldsymbol{\mathcal{F}}_0$, representing the state with no particles (it is important to note that this is distinct from the zero vector—$|\Omega\rangle$ is still a state with some nonzero amplitude). The annihilation operator $\hat{a}_i$ on state i and its adjoint, the creation operator $\hat{a}_i^\dagger$ on state i, are defined by the

following relations:

$$\hat{a}_i|\Omega\rangle = 0 \quad , \qquad \hat{a}^\dagger_{\lambda_N} \cdots \hat{a}^\dagger_{\lambda_2} \hat{a}^\dagger_{\lambda_1} |\Omega\rangle = |\lambda_1, \lambda_2, \ldots, \lambda_N\rangle \tag{1.9}$$

Then, in order for these operators not to violate the antisymmetry of the wavefunction, we impose also the following anticommutation relations:

$$\left\{\hat{a}_i, \hat{a}^\dagger_j\right\} = \delta_{ij} \ , \qquad \left\{\hat{a}_i, \hat{a}_j\right\} = 0 \ , \qquad \left\{\hat{a}^\dagger_i, \hat{a}^\dagger_j\right\} = 0 \tag{1.10}$$

where the anticommutator bracket is defined as $\{A, B\} = AB + BA$. It is often also convenient to define the so-called number operator $\hat{n}_i = \hat{a}^\dagger_i \hat{a}_i$ on site i, and one can see that the name is fitting, since its expectation value given a state is the occupation of site i. One might view these creation and annihilation operators as mappings from $\boldsymbol{\mathcal{F}}_N$ to $\boldsymbol{\mathcal{F}}_{N+1}$ and $\boldsymbol{\mathcal{F}}_{N-1}$, respectively, for some N. One can also view these as ladder operators for the total particle number operator $\hat{N} = \sum_i \hat{n}_i$. That is, given an operator A whose eigenvalues form a totally ordered set (e.g., the eigenvalues are all real, or all pure imaginary), a ladder operator B of A satisfies the commutation relation $[A, B] = \mu B$ where μ is a nonzero constant. Then, given an eigenpair (λ, x) of A, we can see:

$$ABx = (BA + \mu B)x = (\lambda + \mu)Bx \tag{1.11}$$

In particular, $(\lambda + \mu, Bx)$ is another eigenpair of A.

1.2.2 The Schrödinger Equation and the Hubbard Model

At the core of quantum mechanics is the Schrödinger equation, and in its time-independent form is:

$$\hat{H}|\Psi\rangle = E|\Psi\rangle \tag{1.12}$$

where $\hat{H}$ is the Hamiltonian operator, $|\Psi\rangle$ is the wavefunction, and E is the energy corresponding to $|\Psi\rangle$. That is, given a Hamiltonian $\hat{H}$, we seek its eigenpairs (E, Ψ). If one seeks the ground state wavefunction, i.e., the eigenfunction corresponding to the smallest eigenvalue, one can formulate and solve the following optimization problem:

$$\arg\min_{|\Psi\rangle} \langle\Psi|\hat{H}|\Psi\rangle \qquad \text{subject to} \quad \langle\Psi|\Psi\rangle = 1 \tag{1.13}$$

and one obtains the ground state energy by evaulating the Hamiltonian expectation value in the ground state (i.e., simply evaluate the objective function at the solution).

The Hubbard model is a lattice model for fermions, wherein each lattice site (spatial orbital) has a four-dimensional Fock space, with a basis of $B = \Big\{|\Omega\rangle, |\uparrow\rangle, |\downarrow\rangle, |\uparrow\downarrow\rangle\Big\}$. For the purpose of illustrating the concepts introduced in the previous section using a concrete example, consider the Hubbard model with two lattice sites. We have a 16-dimensional Fock space with a basis $|\psi_1\rangle \otimes |\psi_2\rangle$, where each $|\psi_i\rangle \in B$. We have 8 annihilation and creation operators (4 each), given by $\hat{a}_{1\uparrow}, \hat{a}_{1\downarrow}, \hat{a}_{2\uparrow}, \hat{a}_{2\downarrow}$ and their Hermitian conjugates $\hat{a}^\dagger_{1\uparrow}, \hat{a}^\dagger_{1\downarrow}, \hat{a}^\dagger_{2\uparrow}, \hat{a}^\dagger_{2\downarrow}$, respectively.

The relabeling of states to use lattice sites and spin as a generalization of the notation used before is hopefully clear. The Hubbard Hamiltonian in one dimension is given as:

$$\hat{H} = -t\sum_{i,\sigma}\left(\hat{a}^\dagger_{i,\sigma}\hat{a}_{i+1,\sigma} + \hat{a}^\dagger_{i+1,\sigma}\hat{a}_{i,\sigma}\right) + U\sum_i \hat{n}_{i\uparrow}\hat{n}_{i\downarrow} \tag{1.14}$$

where the first term describes the kinetic energy of the system (the so-called "hopping term"), parameterized by the hopping integral t, and the second describes the on-site interaction parameterized by repulsion/attraction strength U. The sum over i iterates over the lattice sites, and that over σ iterates over spin.

1.2.3 The Curse of Dimensionality

In attempting to solve the fermionic many-body problem, one quickly runs into the so-called curse of dimensionality. In the case of the Hubbard model, the size of the Fock space for the system grows exponentially with the number of lattice sites, i.e., the dimension is 4^d, where d is the number of lattice sites. Exploring such a space can be computationally extremely expensive if approached naively. It turns out, however, that these problems are often highly structured, and we can take advantage of these physical symmetries to make solving these problems far more efficient. That is, out of the entire Hilbert space, physically meaningful states constitute only a small subset. Thus, the form of our solution (the ansatz) need not be so general as to express the entire Hilbert space, and a clever choice of ansatz often decreases the complexity of the problem significantly. One can utilize this, for instance, to approximate a solution with an inexpensive method, then use the approximate solution as a starting guess for a more expensive method. As an example, in molecular electronic structure, one often performs a mean-field treatment of the molecule via Hartree–Fock (HF) or multiconfigurational self-consistent field (MCSCF) before treating the system with more complex theories. In our case, with the Hubbard Hamiltonian, we have several advantages, namely the locality of the Hamiltonian and the low dimensionality of the lattice. In this next section, we will demonstrate how one might exploit these symmetries.

1.3 Density Matrix Renormalization Group

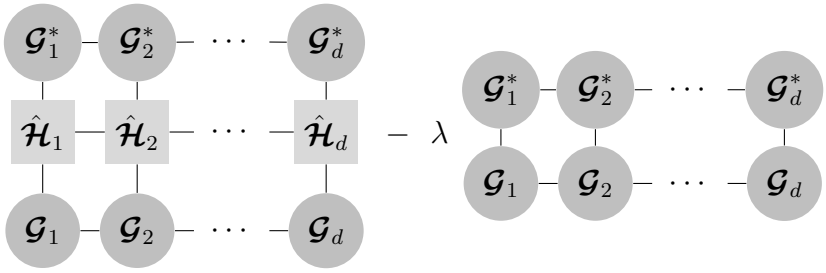
The density matrix renormalization group (DMRG) seeks to solve the following optimization problem:

$$\min_{\boldsymbol{\mathcal{X}}} \langle \boldsymbol{\mathcal{X}}, \hat{H}\boldsymbol{\mathcal{X}} \rangle \quad \text{subject to} \quad \langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}} \rangle = 1 \tag{1.15}$$

where $\hat{H}$ is the Hamiltonian expressed as a TT matrix, $\boldsymbol{\mathcal{X}}$ is a TT vector (the wavefunction), and each of these cores is associated with the Fock space of a given orbital. Notice that this just the variational principle of quantum mechanics rewritten such that instead of minimizing a functional, i.e., over a set of functions, we are minimizing over TT-vectors. It thus follows that the solution $\boldsymbol{\mathcal{X}}_*$ is the ground state wavefunction of the system. In other words, we wish to minimize the Hamiltonian expectation value with respect to a normalized wavefunction $\boldsymbol{\mathcal{X}}$. The Lagrangian for this problem is given by:

$$L(\boldsymbol{\mathcal{X}}, \lambda) = \langle \boldsymbol{\mathcal{X}}, \hat{H}\boldsymbol{\mathcal{X}} \rangle - \lambda \Big[\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}} \rangle - 1 \Big] \tag{1.16}$$

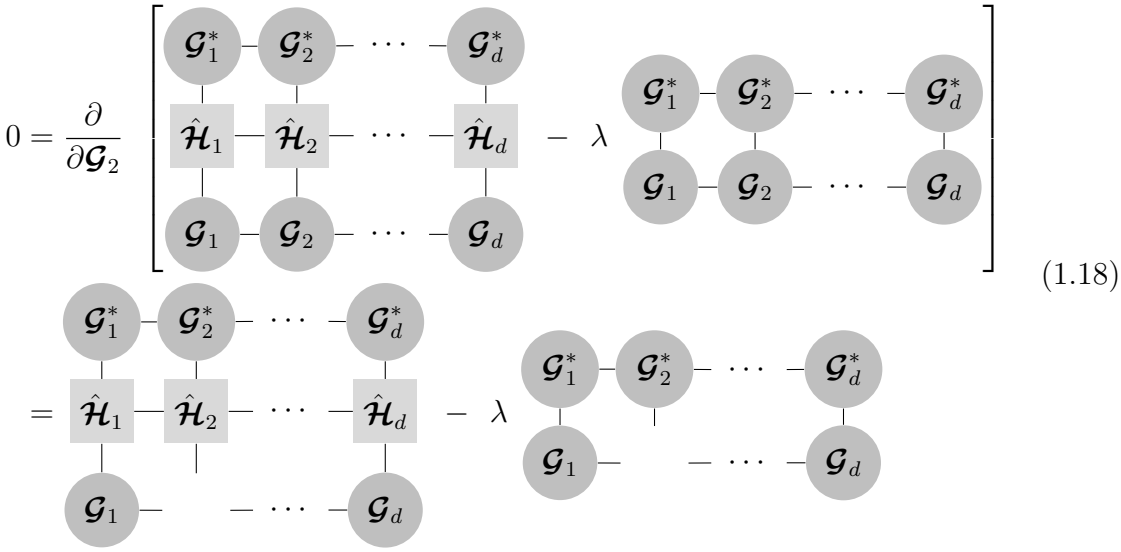
where λ is the Lagrange multiplier corresponding to the equality constraint. We can also write L diagrammatically as:

$L(\boldsymbol{\mathcal{X}}, \lambda) =$  (1.17)

Here, $\hat{\boldsymbol{\mathcal{H}}}_i$ are the TT matrix cores of $\hat{H}$.

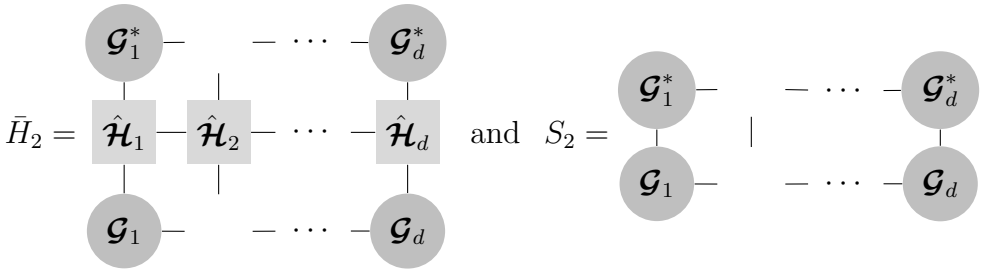
1.3.1 The DMRG Sweep Algorithm

A natural intuition when attempting to (numerically) solve an optimization problem like this might be to apply some sort of iterative minimization method. It turns out that we can compute analytic gradients quite easily, so this is a good intuition. Notice that our objective function is a bilinear (and further, a quadratic) form. Then, WLOG, consider the TT core $\boldsymbol{\mathcal{G}}_2$ of $\boldsymbol{\mathcal{X}}$. If we wish to compute the gradient with respect to $\boldsymbol{\mathcal{G}}_2$ and find a stationary point along that direction, we solve the following linear system:



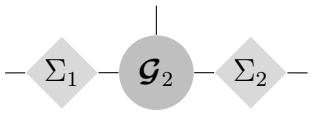
(1.18)

There are several convenient tricks to simplify these gradient equations. Notice that Equation 1.8 can be written as a generalized eigenvalue problem, $\bar{H}_2\boldsymbol{\mathcal{G}}_2 = \lambda S_2\boldsymbol{\mathcal{G}}_2$, where (by symmetry) we have



Recall the canonical form of a TT, as in Equation 1.3. It turns out that if we have $\boldsymbol{\mathcal{X}}$ in canonical form, and rewrite the generalized eigenvalue problem $\bar{H}_2\bar{\boldsymbol{\mathcal{G}}}_2 = \lambda\bar{S}_2\bar{\boldsymbol{\mathcal{G}}}_2$

such that

$\bar{\boldsymbol{\mathcal{G}}}_2 =$ 

then we have that $\bar{S}_2 = I$, the identity. One sees this by observing the orthogonality relations of the cores in the inner product, and "closing the zipper" from both sides of the diagram. Intuitively, one can imagine a "near-sighted" view from a given singular value matrix, where from each side everything appears orthogonal. This leaves us with the eigenvalue problem $\bar{H}_2\bar{\boldsymbol{\mathcal{G}}}_2 = \lambda\bar{\boldsymbol{\mathcal{G}}}_2$, (where historically, $\bar{H}_i$ is called the superblock Hamiltonian for site i) and we can solve for the lowest eigenvalue and its corresponding eigenvector. That is, we solve for the ground state of the superblock Hamiltonian. Since we have Hermiticity, the Lanczos iteration algorithm followed by direct diagonalization of the resulting tridiagonal system (e.g., with a bisection or divide-and-conquer eigenvalue algorithm) is a natural choice.

We'll describe the iteration briefly, though more details on the algorithm can be found in [7]. Let $\boldsymbol{\mathcal{G}}_0$ be an initial guess, and H be the superblock Hamiltonian. Consider the Krylov subspace

$$\boldsymbol{\mathcal{K}}_r\left(H, \boldsymbol{\mathcal{G}}_0\right) = \text{span}\left(\boldsymbol{\mathcal{G}}_0, H\boldsymbol{\mathcal{G}}_0, H^2\boldsymbol{\mathcal{G}}_0, \ldots, H^r\boldsymbol{\mathcal{G}}_0\right) \tag{1.19}$$

We use the Lanczos algorithm to build a tridiagonal approximation to the Hamiltonian within $\boldsymbol{\mathcal{K}}_r\left(H, \boldsymbol{\mathcal{G}}_0\right)$. To get started, the Lanczos iteration requires two orthonormal vectors in the Krylov subspace $\boldsymbol{\mathcal{K}}_2\left(H, \boldsymbol{\mathcal{G}}_0\right)$, i.e., an orthonormal basis, so we assume $\boldsymbol{\mathcal{G}}_0$ normalized and we construct

$$\boldsymbol{\mathcal{G}}_1' = b_1\boldsymbol{\mathcal{G}}_1 = H\boldsymbol{\mathcal{G}}_0 - a_0\boldsymbol{\mathcal{G}}_0 \tag{1.20}$$

where b_1 is the normalization constant $b_1 = \sqrt{\langle\boldsymbol{\mathcal{G}}_1', \boldsymbol{\mathcal{G}}_1'\rangle}$ and a_i are the Hamiltonian

expectation values $a_i = \langle \boldsymbol{\mathcal{G}}_i, H\boldsymbol{\mathcal{G}}_i \rangle$. We have our orthonormal basis $\{\boldsymbol{\mathcal{G}}_0, \boldsymbol{\mathcal{G}}_1\}$. Observe that $\boldsymbol{\mathcal{G}}_1'$ is obtained by an iteration of the method of steepest descent. We thus have the Lanczos iteration

$$\boldsymbol{\mathcal{G}}_{n+1}' = b_{n+1}\boldsymbol{\mathcal{G}}_{n+1} = H\boldsymbol{\mathcal{G}}_n - a_n\boldsymbol{\mathcal{G}}_n - b_n\boldsymbol{\mathcal{G}}_{n-1} \tag{1.21}$$

We can then build the Hamiltonian in the Krylov subspace as follows:

$$\boldsymbol{\mathcal{H}}_{\boldsymbol{\mathcal{K}}_r} = \begin{bmatrix} a_0 & b_1 & 0 & 0 & \cdots & 0 \\ b_1 & a_1 & b_2 & 0 & \ddots & \vdots \\ 0 & b_2 & a_3 & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & b_{r-1} & 0 \\ \vdots & \ddots & \ddots & b_{r-1} & a_{r-1} & b_r \\ 0 & \cdots & \cdots & 0 & b_r & a_r \end{bmatrix} \tag{1.22}$$

Relatively speaking, this matrix is extremely fast to diagonalize directly, so we can compute from this our next approximation to the ground-state. One can construct each next approximation to the solution using the solutions of the previous iterations. The reference given above gives more details into how, in practice, one may use other clever tricks to make this subroutine even more efficient.

These observations gives rise to the so-called DMRG sweep algorithm, in which we optimize one TT core at a time, "sweeping" across the entire TT, until convergence. It also becomes clear why we expect the DMRG to converge to the ground state of the overall system, since at each step we have a monotonic decrease of the system energy, and the only minimum of the functional is the ground state (and every other stationary point is a saddle point and the set of saddle points has measure 0). One might see this as quite similar to a sort of alternating direction method of multipliers (ADMM) algorithm. If one wishes to fill in this analogy, one can consult [3].

Algorithm 4: DMRG Sweep Algorithm

Input: the Hamiltonian $\hat{H}$ in TT matrix form, and initial guess TT vector $\boldsymbol{\mathcal{X}}_0$ in canonical form,
Output: Converged solution $\boldsymbol{\mathcal{X}}$, and energy E

function $[E, \boldsymbol{\mathcal{X}}] = \text{DMRG}(\hat{H}, \boldsymbol{\mathcal{X}}_0)$
  $\boldsymbol{\mathcal{X}} \leftarrow \boldsymbol{\mathcal{X}}_0$
  while not converged:
    for $i \in [d]$
      $\bar{\boldsymbol{\mathcal{G}}}_i \leftarrow \text{DIAGONALIZE}(\bar{H}_i)$
    end for
  end while
  $E \leftarrow \langle \boldsymbol{\mathcal{X}}, \hat{H}\boldsymbol{\mathcal{X}} \rangle$
end function

The diagonalization step is typically the most expensive step by far, so we might wish to make it less expensive. When our TT is low-rank, the diagonalizations are cheaper, and we can still capture much of the physics of the system with a low-rank TT (since we keep the most important singular values). Thus, it often makes sense to have the first sweeps of the DMRG algorithm keep $\boldsymbol{\mathcal{X}}$ at relatively low rank, then increase it later on to capture the smaller singular values.

We note that there are other algorithms to solve the DMRG problem, for instance the time-evolving block decimation (TEBD) algorithm, but we limit our discussion in this work to the sweep algorithm.

1.3.2 Variational Rounding

Recall that performing TT mat-vecs and TT vector addition increases the ranks. As alluded to before, we introduce a more sophisticated rounding algorithm which takes advantage of the nice form of the gradients with respect to the cores. We solve the following minimization problem:

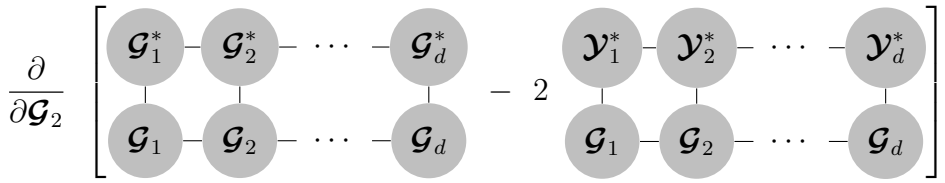
$$\arg\min_{\boldsymbol{\mathcal{X}}} \langle \boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}} \rangle \qquad \text{subject to} \qquad \text{rank}(\boldsymbol{\mathcal{X}}) = (r_1, r_2, \ldots, r_{d-1}) \tag{1.23}$$

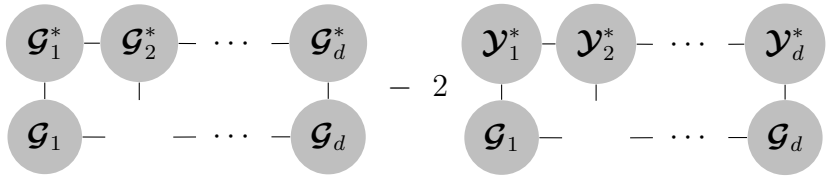
where $\boldsymbol{\mathcal{Y}}$ is a given TT vector and $\boldsymbol{\mathcal{X}}$ is a TT vector with same order and dimensions, but smaller TT rank (where smaller is in the sense of a partial order on $(d-1)$-tuples of natural numbers). Notice that this objective function has TT addition written in. We want to avoid this (since it defeats its own purpose), so we write the following:

$$\arg\min_{\boldsymbol{\mathcal{X}}} \langle \boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}} \rangle = \arg\min_{\boldsymbol{\mathcal{X}}} \langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}} \rangle - 2\langle \boldsymbol{\mathcal{Y}}, \boldsymbol{\mathcal{X}} \rangle \tag{1.24}$$

Then, we can apply the nice properties of gradients with respect to TT cores in TT inner products, which we outlined above, and apply an analogous sweep algorithm to find a local minimum. In particular, our gradients come down to solving a linear system. Again, WLOG, we consider $\boldsymbol{\mathcal{G}}_2$:

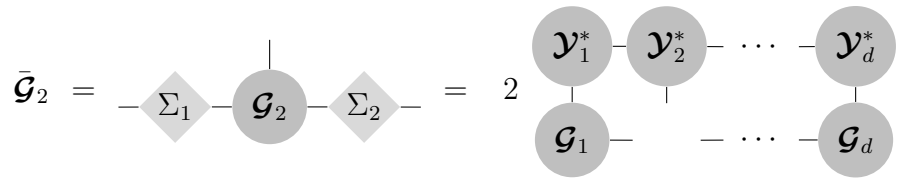
$$0 = \frac{\partial}{\partial \boldsymbol{\mathcal{G}}_2}$$





(1.25)

Then, performing the same trick as above using the canonical form, we obtain a satisfyingly simple expression for the iteration:



(1.26)

We then have the following algorithm:

A natural initial guess is $\boldsymbol{\mathcal{Y}}$ truncated to the target rank.

Algorithm 5: Variational TT Rounding

Input: TT vector $\boldsymbol{\mathcal{Y}}$ with TT rank $r_{\boldsymbol{\mathcal{Y}}}$, and initial guess TT vector $\boldsymbol{\mathcal{X}}_0$ in canonical form of same dimension with target TT rank $r \preceq r_{\boldsymbol{\mathcal{Y}}}$

Output: Converged solution $\boldsymbol{\mathcal{X}}$

```
function [X] = ROUND(X_0, r)
X ← X_0
    while not converged:
        for i ∈ [d]
            Ḡ_i ← 2Ȳ_i
        end for
    end while
end function
```

1.3.3 The Structure of the Hubbard Hamiltonian

There is an important aspect of this algorithm we haven't yet discussed: what does the Hamiltonian in TT matrix form look like? The most natural application of the TT vector ansatz is to problems on a one-dimensional lattice with open boundary conditions, especially with local interactions, like the aforementioned Hubbard model described by Equation 1.14. The way the TT vector reflects this physical structure is by associating each core to a lattice site, where the core indices maintain the corresponding physical adjacencies. The distance of interactions are reflected by the TT rank, where higher ranks generally indicate longer range interactions. Thus, when the system Hamiltonian has local interactions, we expect that the solution in TT form will be low-rank.

This locality is also reflected by sparsity in the full Hamiltonian, and we can take advantage of this in an exactly analogous fashion by expressing it as a TT matrix. That is, given the full Hamiltonian tensor, one can in principle decompose it using a (slightly more general) TTSVD. There is another way to construct this. In particular, we can express the local structure in terms of finite-state machines, and thus directly compute the form of each local MPO (i.e., each core of the Hamiltonian), as opposed to forming the full Hamiltonian and compressing it. A detailed explanation of this

method can be found in [13]. For systems like the Hubbard model, where each core of the TT matrix is the same (other than at the boundary), this method is particularly useful. In practical computations, one only needs to compute the Hamiltonian and store it once, so to use the more general, more expensive method is sometimes not an issue. Code to do this can be found in [6]. We are now equipped to run DMRG.

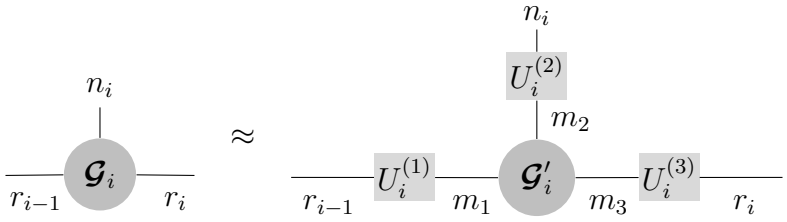
Chapter 2

Compression of TT Cores

Here we delve into the primary theoretical contributions of this work, where we consider compressing the TT cores. The motivation for doing so is that there are several issues of barebones TT that we may be able to alleviate. In particular, we may be able to improve the efficiency of operations, e.g., linear combinations, matrix-vector multiplication analogs, inner products, and rounding. We naturally also have the benefit of storing the TT with less memory. In this work, we have chosen to use the canonical Tucker decomposition, i.e., the Tucker decomposition with orthonormal factor matrices, for the compression.

2.1 The Basics

What we do in particular is, given a TT core $\boldsymbol{\mathcal{G}}_i$, perform a rank-$(m_1, m_2, m_3)$ Tucker decomposition of $\boldsymbol{\mathcal{G}}_i$:



(2.1)

where $U_i^{(j)}$ has orthonormal columns. The case for when the TT is in canonical form is exactly analogous, where we simply ignore the singular value matrix between cores. Notice also that if our TT is in canonical form, $\boldsymbol{\mathcal{G}}_1$ and $\boldsymbol{\mathcal{G}}_d$ are already orthogonal matrices, so we do not do anything. Modifying Algorithm 3, we can do this in the following way:

Algorithm 6: Tensor Train SVD with Core Compression

Input: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, and one of: target TT rank $r = (r_1, r_2, \ldots, r_d)$ and target multilinear ranks $m = \left\{(m_i^{(1)}, m_i^{(2)}, m_i^{(3)})\right\}_{i=2}^{d-1}$

Output: TT cores in Tucker format $\boldsymbol{\mathcal{G}}_1, \boldsymbol{\mathcal{G}}_2, \ldots, \boldsymbol{\mathcal{G}}_d$ with TT rank r and $\boldsymbol{\mathcal{G}}_i$ with multilinear rank $\left(m_i^{(1)}, m_i^{(2)}, m_i^{(3)}\right)$, and error ERR

function $[\boldsymbol{\mathcal{G}}_1, \boldsymbol{\mathcal{G}}_2, \ldots, \boldsymbol{\mathcal{G}}_d] = \text{TTSVD}(\boldsymbol{\mathcal{X}}, r \text{ and } m)$
$r_0 \leftarrow 1$
$Y_1 \leftarrow \boldsymbol{\mathcal{X}}$
for $k \in [d-1]$:
    $\bar{Y}_k \leftarrow \text{reshape}\left(Y_k, (r_{k-1}n_k\ ,\ n_{k+1}n_{k+2}\cdots n_d)\right)$
    $[\bar{G}_k, \varepsilon_k] \leftarrow \text{LLSV}(\bar{Y}_k, r_k \text{ or } \bar{\varepsilon})$
    $Y_{k+1} \leftarrow \bar{G}_k^\top \bar{Y}_k$
    $\bar{\boldsymbol{\mathcal{G}}}_k \leftarrow \text{reshape}(\bar{G}_k, (r_{k-1}\ ,\ n_k\ ,\ r_{\,k}))$
    $[\boldsymbol{\mathcal{G}}_k, \delta_k / \|\bar{Y}_k\|_F] \leftarrow \text{HOSVD}(\bar{\boldsymbol{\mathcal{G}}}_k, m_k)$
end for
$\boldsymbol{\mathcal{G}}_d \leftarrow Y_d$
$\text{ERR} \leftarrow \sqrt{\sum_{k=1}^{d-1}(\varepsilon_k + \delta_k)^2}$
end function

Notice that since we are performing two tensor approximations, there is now some ambiguity when it comes to choosing the source of approximation error, i.e., whether it comes from the TTSVD step or the HOSVD step. For that reason, and for the sake of simplicity, in the algorithm above we suppose that ranks, not an absolute error bound, are given, but in a later section we will describe ways in which one can control the absolute error. For now, we proceed.

2.1.1 Algebraic Operations

Because performing compression does not affect the noncontracted indices of the TT, operations like tensor inner products and tensor matvecs remain, strictly speaking, identical, in the same way that those operations are not affected by compressing an arbitrary tensor into TT vector format. Scalar multiplication is the same: take any component and scale it. In this form, as not to take break orthonormality of the columns of any factor matrix, we typically choose to scale the new core (though in principle one could scale the factor matrices). Vector addition is also analogous (see Equation 1.4), but since we deal with more components, we have nested expressions for the entries of the cores. As before, let $C = A + B$, where A and B are TT vectors, but now with the cores compressed. We denote the cores as $A_j = A'_j \times_1 U_j^{(1)} \times_2 U_j^{(2)} \times_3 U_j^{(3)}$ and $B_j = B'_j \times_1 V_j^{(1)} \times_2 V_j^{(2)} \times_3 V_j^{(3)}$. We then have:

$$
\begin{aligned}
C_1[i_1] &= \begin{bmatrix} A_1[i_1] & B_1[i_1] \end{bmatrix} \\
C_j[i_j] &= \begin{bmatrix} A'_j & 0 \\ 0 & B'_j \end{bmatrix} \times_1 \begin{bmatrix} U_j^{(1)} & 0 \\ 0 & V_j^{(1)} \end{bmatrix} \times_2 \begin{bmatrix} U_j^{(2)}[i_j] \\ V_j^{(2)}[i_j] \end{bmatrix} \times_3 \begin{bmatrix} U_j^{(3)} & 0 \\ 0 & V_j^{(3)} \end{bmatrix} \\
C_d[i_d] &= \begin{bmatrix} A_d[i_d] \\ B_d[i_d] \end{bmatrix}
\end{aligned}
\tag{2.2}
$$

where $j \in \{2, 3, \ldots, d-1\}$. Here, the notation for the order-3 cores is block diagonal in the sense that any order-2 slice gives back a matrix of block diagonal structure. The orientation of this new tensor is in the natural way, preserving the structure of the summands. Again, in a similar manner to the general case, the TT rank increases upon summation. Later, we will revisit TT rounding and see how one can perform the previously described algorithms on this ansatz.

2.1.2 Error Analysis

When we perform the compression, we'd like to know how much error we incur. We can make use of the already existing results for tensor train error. We first state a lemma.

Lemma 2.1.1 (**TTSVD error recursion**). *Let $\boldsymbol{\mathcal{X}}$ be a tensor of dimension d, and let $\boldsymbol{\mathcal{X}}_k$ be the partially formed tensor train at iterate k of the TTSVD algorithm (Algorithm 3) with cores $\boldsymbol{\mathcal{G}}_1, \boldsymbol{\mathcal{G}}_2, \ldots, \boldsymbol{\mathcal{G}}_k$ and residual tensor $\boldsymbol{\mathcal{Y}}_k$. Then, for $k \in [d-1]$,*

$$\|\boldsymbol{\mathcal{X}}_k - \boldsymbol{\mathcal{X}}_d\|_F^2 = \varepsilon_k^2 + \|\boldsymbol{\mathcal{X}}_{k+1} - \boldsymbol{\mathcal{X}}_d\|_F^2$$

where ε_i is the LLSV error at the i-th iteration.

The full proof of this lemma can be found in [2], but the primary tool we will use is the subspace error given by

$$\varepsilon_k = \|(I - G_k G_k^\top)\bar{Y}_k\|_F \tag{2.3}$$

Then, we have the total TTSVD error given by the following result:

Theorem 2.1.1 (**Error of the TTSVD**). *Let $\boldsymbol{\mathcal{X}}$ be a tensor of dimension d, and let $\bar{\boldsymbol{\mathcal{X}}}$ be a TT vector of dimension d with cores $\boldsymbol{\mathcal{G}}_1, \boldsymbol{\mathcal{G}}_2, \ldots, \boldsymbol{\mathcal{G}}_d$ constructed using the TTSVD. Then,*

$$\|\boldsymbol{\mathcal{X}} - \bar{\boldsymbol{\mathcal{X}}}\|_F^2 = \sum_{k=1}^{d-1} \varepsilon_k^2$$

This result follows from a direct application of the error recursion lemma. Notice that to use this lemma requires the error to be introduced by TTSVD, so for it to apply to our case, we compress the cores as we perform the TTSVD, not all afterwards. Nevertheless, we arrive at the following result:

Theorem 2.1.2 (**Error of TT core compression**). *Let $\boldsymbol{\mathcal{X}}$ be an order-d tensor with the following TTSVD error:*

$$\varepsilon = \sum_{k=1}^{d-1} \varepsilon_k^2$$

where ε_k is the LLSV error of core k. Upon compressing $\boldsymbol{\mathcal{X}}$ by Algorithm 6, the total error is bounded by

$$\bar{\varepsilon}^2 \leq \sum_{k=1}^{d-1} (\varepsilon_k + \delta_k)^2 \leq \sum_{k=1}^{d-1} \varepsilon_k^2 + \sum_{k=2}^{d-1} \delta_k^2 \tag{2.4}$$

where δ_k is the Tucker decomposition error of core k relative to the the residual tensor norm $\|\bar{Y}_{k+1}\|$.

Proof. Recall that the LLSV error ε_k is incurred upon the construction of the k-th core, and is not affected by the construction of the proceeding cores. Thus, we can consider the error incurred on a given core from performing the Tucker decomposition to be incurred simultaneously with the LLSV error if we compress at each iteration.

Let the recontracted Tucker decomposed cores be written $\boldsymbol{\mathcal{G}}_i = \bar{\boldsymbol{\mathcal{G}}}_i + \delta\boldsymbol{\mathcal{G}}_i$, where $\bar{\boldsymbol{\mathcal{G}}}_i$ is the core before Tucker decomposition and $\|\delta\boldsymbol{\mathcal{G}}_i\|_F = \delta_i / \|\bar{Y}_{i+1}\|_F$ for some $\delta_i \geq 0$. Then, we can write the error contribution from a given core by

$$\begin{aligned}
\bar{\varepsilon}_k &= \|(I - G_k \bar{G}_k^\top)\bar{Y}_k\|_F \\
&= \|(I - (\bar{G}_k + \delta G_k)\bar{G}_k^\top)\bar{Y}_k\|_F \\
&= \|(I - \bar{G}_k\bar{G}_k^\top - \delta G_k \bar{G}_k^\top)\bar{Y}_k\|_F \\
&\leq \|(I - \bar{G}_k\bar{G}_k^\top)\bar{Y}_k\|_F + \|\delta G_k \bar{G}_k^\top \bar{Y}_k\|_F \\
&\leq \varepsilon_k + \|\delta G_k\|_F \cdot \|\bar{G}_k^\top \bar{Y}_k\|_F \\
&= \varepsilon_k + \delta_k
\end{aligned} \tag{2.5}$$

We attain the intermediate bound for the full error by applying Theorem 2.1.1, and

the upper bound is then attained by triangle inequality. Note that $\delta_1 = 0$ since the first core is a matrix, so we do not perform a Tucker decomposition. □

We remark that by using an error which is relative to a quantity that needs to be computed on the fly can be a problem for practical purposes. In particular, in order to meet a prescribed overall error bound, the allowed HOSVD error at each iteration must be computed inside of the TTSVD loop by means of evaluating the norm of the residual tensor $\|\bar{Y}_k\|_F$.

To the end of an error analysis which does not depend on relative terms, there is an important aspect of the TT decomposition which was alluded to but not discussed in detail, namely the fact that the mode-3 unfoldings of the TT cores have orthonormal columns. (This is for the mode-3 foldings in particular because we use the left leading singular vectors, though in principle one could use the right leading singular vectors, in which case we would refer to the mode-1 unfolding.) We might like to retain this property with the compressed TT. However, we have no immediate guarantee that, upon Tucker decomposing the cores, we keep this orthonormality condition. The HOSVD subroutine in Algorithm 6 thus may not be identically Algorithm 2—the problem we solve has additional manifold constraints. What we are pointing out in particular is the fact that the $\delta\boldsymbol{\mathcal{G}}_i$ terms are generic perturbations, so we cannot say immediately, for instance, whether $\delta G_i \bar{G}_i$ is diagonal or has some other nontrivial structure. The main TTSVD error result relies on Eckart–Young–Mirsky (Theorem 1.1.1), and the HOSVD does not necessarily preserve the orthonality relations of the LLSV. If it were the case, it could allow a stronger error bound. To consider a Tucker decomposition problem wherein we constrain certain unfoldings to be orthonormal is thus a natural step to consider in future work.

2.2 Revisiting DMRG

We are interested in how compressing the cores affects the DMRG sweep algorithm. The problem is largely the same, except that we enforce the structure of the decomposed cores by adding constraints. In particular, we write

$$\min_{\boldsymbol{\mathcal{X}}} \langle \boldsymbol{\mathcal{X}}, \hat{H}\boldsymbol{\mathcal{X}} \rangle \quad \text{subject to} \quad \begin{cases} \langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}} \rangle = 1 \\ U_i^{(j)*} U_i^{(j)} = I \end{cases} \tag{2.6}$$

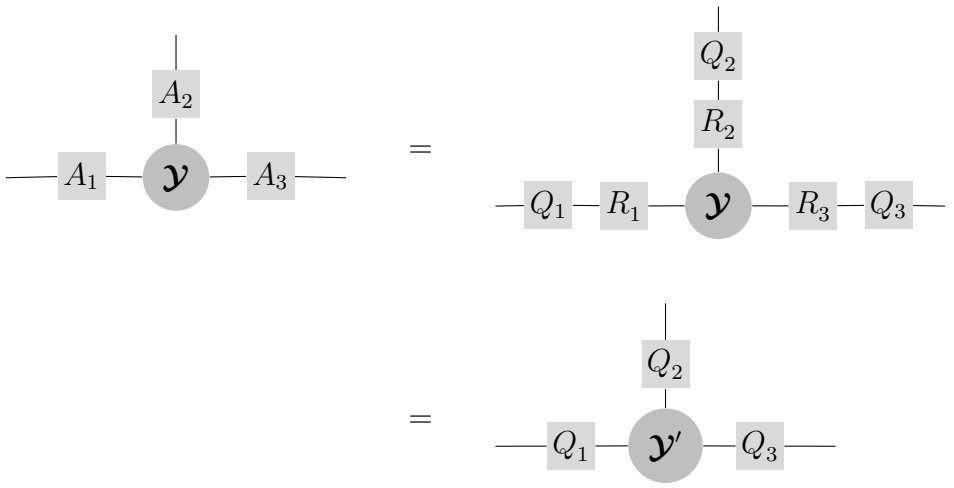
where $i \in \{2, 3, \ldots, d-1\}$ and $j \in [3]$. In particular, we are enforcing semiunitarity of each Tucker factor matrix, or in other words, the constraint set for our factor matrices are the corresponding Stiefel manifolds. Because the overall, big-picture algorithm has not changed, we deal only with the specifics of what changes at the level of the TT cores upon compressing them.

A quick remark before we proceed: we mentioned before that in practice, one may begin the DMRG algorithm with a TT vector of small rank, and grow the rank as the algorithm iterates. With the multilinear ranks as a new hyperparameter, one can now choose to begin in the compressed form with low multilinear ranks and grow to the exact TT as we iterate. Whether one might choose to do so simultaneously, sequentially, or alternatively with growing the TT rank is up to empirical results.

2.2.1 Utilizing Nonuniqueness of the Tucker Decomposition

It might appear that we have to reformulate the optimization problem with these manifold constraints, which would make the problem a bit more complicated. However, it turns out there is a way to avoid this completely at a relatively small computational cost, and comes as a result of the aforementioned nonuniqueness of the Tucker decomposition. We can perform QR factorizations of the factor matrices such that

the orthogonal part faces outward from the Tucker core and the triangular part faces inward, and multiply back into the core. That is, given $\mathcal{Y} \times_1 A_1 \times_2 A_2 \times_3 A_3$ with A_i not semiunitary,



The fact that we may not need to reform and refactor cores is important: the HOSVD is quasi-optimal in the sense that the approximation error is within a factor of $\sqrt{d}$ ($d = 3$ in our case) of the optimal Tucker error. We may be able to avoid this bit of suboptimality.

2.2.2 Lanczos Iteration on Tucker Decompositions

Since we wish to perform DMRG, we should be equipped to perform the Lanczos iteration on the compressed cores. Notice that the superblock Hamiltonian is Hermitian, so the matvecs do not change the size of the Tucker decomposition. In fact, since our uncontracted superblock Hamiltonian contains the factor matrices of adjacent cores, we can automatically have the matvec $H\mathcal{G}$ be in Tucker form, and with the same multilinear rank if all the cores have the same multilinear rank.

Since Lanczos requires that we take linear combinations of the cores, we need a way to add two Tucker decompositions. In general, doing this requires reforming the full tensor. For physical problems, however, the size n of each dimension of the overall

tensor is small, typically smaller than 10. For electrons, this size is $n = 4$. Thus, it is reasonable that in this dimenion, we do not factor out anything. That is, the core multilinear ranks look like (r_1, n, r_3). Then, notice that if we have no factor matrix in one of the three dimensions, the core looks itself like a tensor train, and we know how to take linear combinations of those. This has a caveat: it reintroduces the rank problem from TT addition. Luckily, these tensor trains are short and small, so we can expect the nonlinear optimization procedure for low-rank approximation to be fast and inexpensive. From an implementation standpoint, this can be nice. Rather than directly adding TT cores when taking linear combinations, we only need to store additional vectors since the addition of tensor trains is implicit.

Chapter 3

Numerical Results

The implementations are written in Julia, using ITensor. Much of the code to perform the computations can be found in the appendix. The complete Pluto notebook is available upon request.

3.1 More on the Error Bound

Because the only quantity which does not have tight bounds is the relative HOSVD error $\delta_k / \|\bar{Y}_{k+1}\|_F$, we want to characterize the behavior of

$$\frac{\left|\delta_k - \|\delta G_k \bar{G}_k^\top \bar{Y}_k\|_F\right|}{\|\delta G_k \bar{G}_k^\top \bar{Y}_k\|_F}$$

i.e., the looseness of this quantity's bound in a relative sense. It turns out that empirically, given some nontrivial iterate, this quantity looks roughly constant with respect to the multiranks for any nonexact Tucker decomposition (when exact, the value is 0). This suggests that there may exist a provable quasi-optimality result, improving on the bound given in Theorem 2.1.2 and giving better characterization.

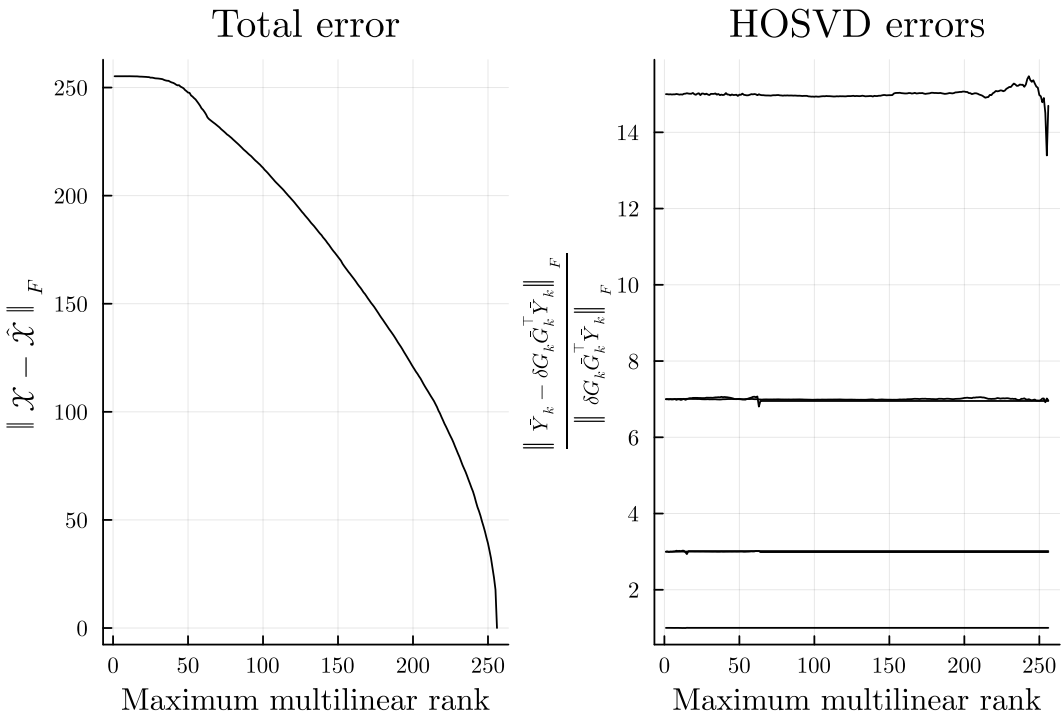
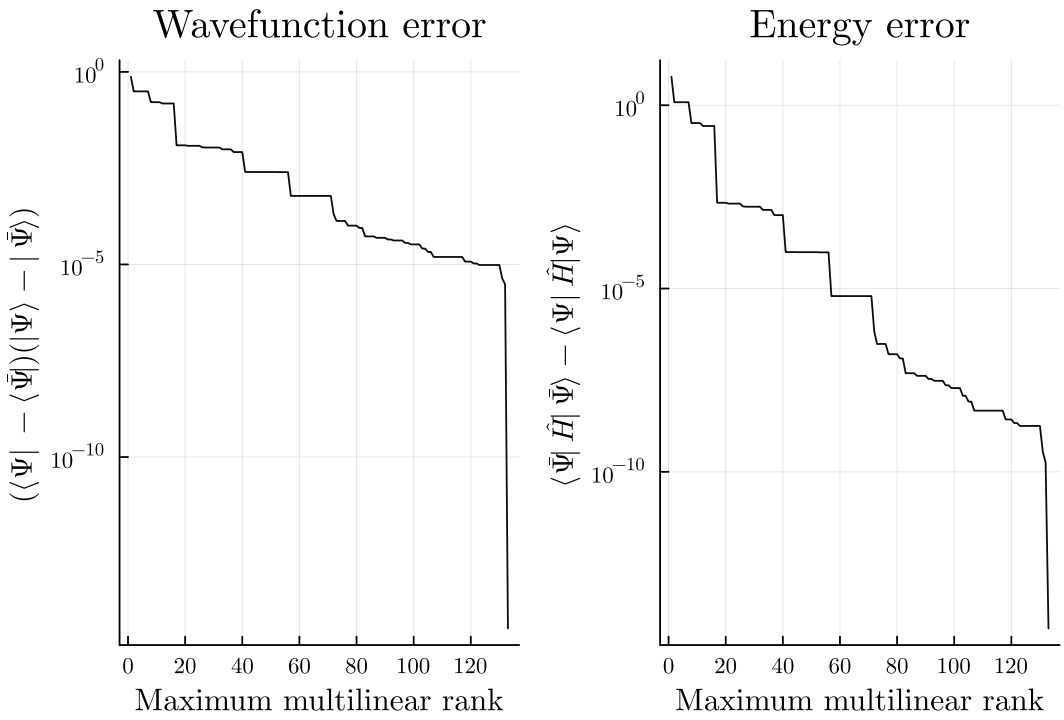


Figure 3.1: Error of TTSVD with core compression on a random tensor of order 8 and each index of size 4. As expected, the total error decreases as the maximum allowed multilinear rank increases, until it becomes exact. Observe that the relative HOSVD errors for each iteration look roughly constant. Each each line corresponds to an iteration k, though we have not labeled them explicitly here. The fact that this quantity appears constant suggests that there exists a tighter error bound.

3.2 Compression of DMRG Solutions

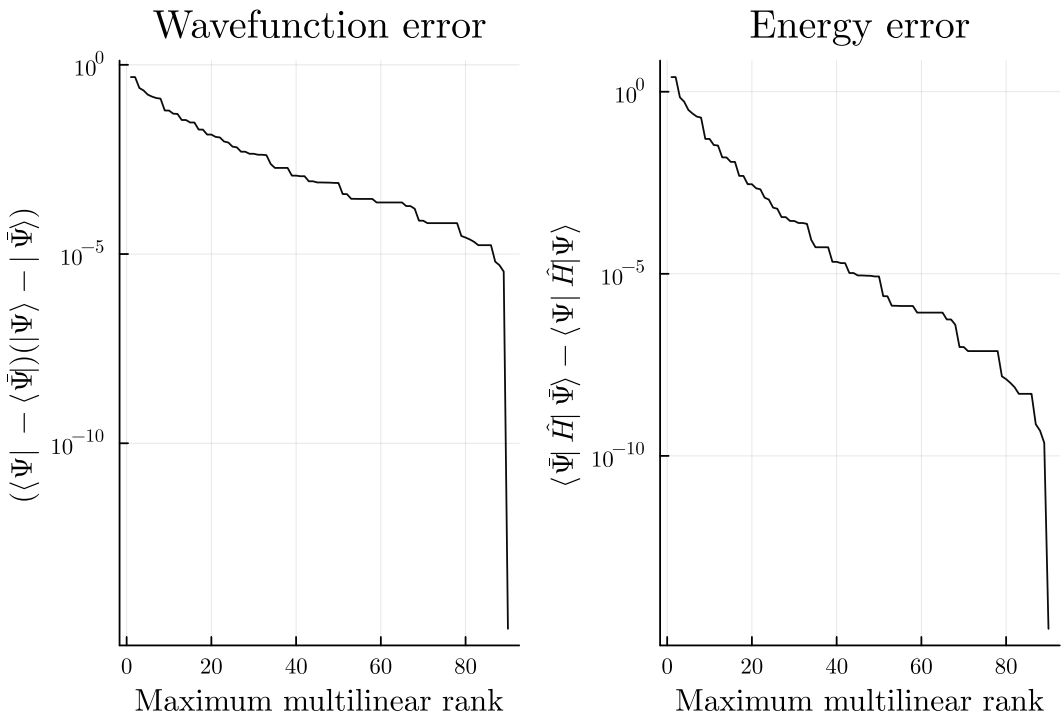
We demonstrate a naive application of this compression to DMRG. Given a DMRG solution, we compress the cores and observe how two quantities vary with the maximum multirank: the approximation error $\left(\langle\Psi| - \langle\bar{\Psi}|\right)\left(|\Psi\rangle - |\bar{\Psi}\rangle\right)$, and the Hamiltonian expectation value error $\langle\bar{\Psi}|\hat{H}|\bar{\Psi}\rangle - \langle\Psi|\hat{H}|\Psi\rangle$, where $|\Psi\rangle$ is the DMRG solution, $|\bar{\Psi}\rangle$ is the compressed solution, and $\hat{H}$ is the system Hamiltonian as an MPO (TT matrix).



(a) Wavefunction error and energy error for the solution to the half-filled ($N = 8$), 8-site Hubbard model with $U = 0$, $t = 1$

Figure 3.2: Compression of the DMRG solutions varying over multilinear rank, and the wavefunction error and energy error of the corresponding compressed tensor trains. The x-axis in both cases cuts off once the Tucker decomposition is exact.

One might wonder why we have not chosen to vary over U (or U/t). This is because for a fixed number of electrons (positive U), the on-site interaction term contribution is just a constant, so we can separate its contribution [5]. One could observe empirically as well that the wavefunction does not change for differing values



(b) Wavefunction error and energy error for the solution to the half-filled with a particle hole ($N = 7$), 8-site Hubbard model with $U = 0$, $t = 1$

Figure 3.2: Compression of the DMRG solutions varying over multilinear rank, and the wavefunction error and energy error of the corresponding compressed tensor trains. The x-axis in both cases cuts off once the Tucker decomposition is exact. (cont.)

of nonnegative U.

One might be curious as to whether this approximation conserves the quantum numbers of the system, in this case the particle number and S_z. Empirically, one can see that indeed this is the case. This is rather important, since the approximation might otherwise not be useful for DMRG applications. Formal proof of this conservation may be present in future works.

Of course, since we have not optimized the Tucker decompositions, these errors are only quasi-optimal, i.e., at most a factor of $\sqrt{3}$ from the error given by the optimal Tucker decomposition, not strictly optimal. We have also not realized any of the computational benefits of using such a form in the actual DMRG calculation. The clear next step is to implement, as described in the previous chapter, the new Lanczos

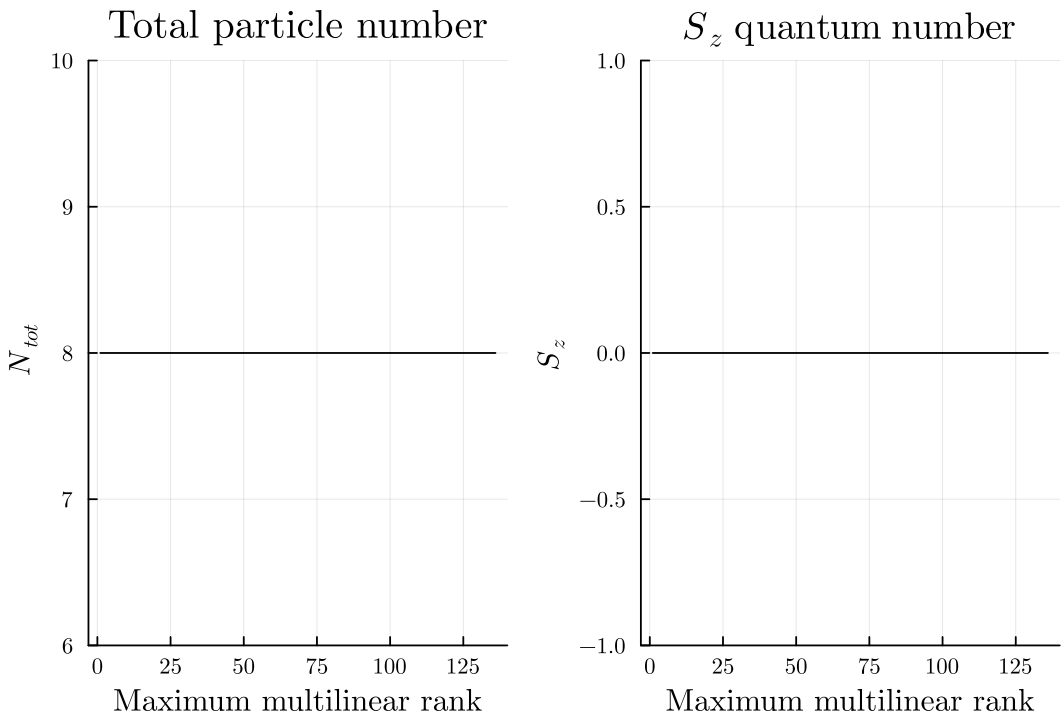


Figure 3.3: Evidence for the conservation of quantum numbers under core compression. The data correspond to the same numerical experiment as in Figure 3.2a, that is, we can see that the quantum numbers are exactly conserved even upon compressing.

algorithm for the Tucker cores. However, the numerical experiment above is evidence that doing so would indeed be useful, since even for generous truncations we still obtain chemical accuracy, and by optimizing in Tucker form, the accuracy can only improve.

Chapter 4

Concluding Remarks

Tensor-based methods, ranging from electronic structure and condensed matter physics to data science and machine learning, are among some of the most expensive yet widely used scientific computing methods. Because of this, it is natural that we study ways to make these algorithms faster and more efficient. Here, we have taken the key object of some of these algorithms, i.e., the tensor train, and studied its behavior upon further compression. We have observed how solutions from the DMRG, an algorithm which optimizes over a space of tensor trains, behaves upon this compression and seen that, at least naively, the behavior is desirable.

The compression of TT cores is, in some sense, a quite elementary extension while being relatively upstream. Because of this, there are many potential further extensions to this work aside from those already mentioned, many of which are just natural extensions of the theory of tensor trains but with the new ansatz. This might include, for instance, the computation of several eigenvectors simultaneously instead of only a single one (which has applications in quantum chemistry for the computation of excited states), the extension to the molecular Hamiltonian, or the decomposition of cores for higher dimensional tensor networks. Another extension might be to compress cores using different tensor decompositions, for instance, the

canonical polyadic decomposition (CPD). In the physical and chemical context, another interesting extension is to interpret the decompositions of the cores, and to determine whether we can say something further, in the physics context, about the solution set we constrain to by decomposing the cores.

Here, having developed some of the theory for this compression using the Tucker decomposition, and demonstrated the basic application of compressing the ground state for the one-dimensional Hubbard model with open boundary conditions, we show that this direction has potential.

Chapter 5

Appendix

Here, we provide the essential Julia code needed to reproduce the numerical results in Chapter 3. As stated, the complete Pluto notebook is available upon request. One first requires the ITensor packages ITensors.jl and ITensorMPS.jl.

We provide the functions and a short script used to generate the data. Below is the generic HOSVD:

```
function hosvd(T::ITensor , maxrank::Vector{Int64})
    G = copy(T)
    factors = Vector{ITensor}()
        for i in 1:3
                U,S,V = svd(G, inds(T)[i]; maxdim=maxrank[i])
                G = S*V
                push!(factors,U)
        end
        err = norm(T - G*factors[1]*factors[2]*factors[3])
    return [G;factors], err
end
```

Below is the function to perform the compression of a full tensor into TT form with

Tucker-compressed cores:

```
function tuckerttsvd(T::ITensor , ttrank::Vector{Int64},
multiranks::Vector{Vector{Int64}})

    TT = Vector{Vector{ITensor}}()
    Y = copy(T)
    indices = inds(Y)
    d = length(inds(Y))
    epsilons = zeros(d-1,1)
    deltas = zeros(d-1,1)

    rel_errs= []

    for k in 1:d-1
        if k==1
            U,S,V = svd(Y,indices[k]; maxdim=ttrank[k])
            epsilons[k] = norm(Y - U*S*V)
            push!(TT, [U])
            Y = S*V
        else
            idx = 0
            for i in inds(Y)
                if hastags(i,``Link``)
                    idx = i
                end
            end
            U,S,V = svd(Y,indices[k],idx; maxdim=ttrank[k])
```

```
            epsilons[k] = norm(Y - U*S*V)
            G, delta = hosvd(U, multiranks[k-1])
            err = norm((U - contracttucker(G))*S*V)
            push!(rel_errs, abs(err - delta*norm(S*V))/err)

            Y = S*V
            deltas[k] = delta*norm(S*V)
            push!(TT, G)
        end

    end
    push!(TT, [Y])
    err_bd = norm(epsilons+deltas)

    return TT, err_bd, rel_errs
end
```

Below is a short function to contract a Tucker-decomposed tensor back to the full tensor:

```
function contracttucker(T::Vector{ITensor})
    G = T[1]
    for U in T[2:length(T)]
        G = G*U
    end
    return G
end
```

Below is the subroutine to compress a core of the TT into Tucker form, where we only

compress in the direction of the two virtual indices, leaving the physical index alone:

```
function ttcorehosvd(T::ITensor , maxrank::Int64)
    G = copy(T)
    factors = Vector{ITensor}()
    for i in 1:3
        if !hastags(inds(T)[i],``Site'')
            U,S,V = svd(G, inds(T)[i]; maxdim=maxrank)
            G = S*V
            push!(factors,U)
        end
    end
    return G,factors[1],factors[2]
end
```

Finally, below is the script used to run the DMRG and generate the data for Figures 3.2 and 3.3:

```
begin
    # N is the number of sites
    # U is the repulsion/attraction parameter
    # t is the hopping parameter
    N = 8
    U=0
    t=1

    approx_err_data = []
    H_exp_err_data = []
    sol = []
```

```
fluxes = []

for k in U
    # here, we construct the Hubbard Hamiltonian
    os = OpSum()
    for j=1:N-1
        os += -t, ''Cdagup'',j,''Cup'',j+1
        os += -t,''Cdagup'',j+1,''Cup'',j
        os += -t, ''Cdagdn'',j,''Cdn'',j+1
        os += -t,''Cdagdn'',j+1,''Cdn'',j
        os += k, ''Ntot'', j
    end
    os += k, ''Ntot'', N

    sites = siteinds(''Electron'',N; conserve_qns=true)
    H = MPO(os,sites)

    # psi0 is the DMRG initial guess
    state = [isodd(n) ? ''Up'' : ''Dn'' for n in 1:N]
    # uncomment below line for particle hole
    # state[N] = ''Emp''
    psi0 = MPS(sites,state)

    # DMRG parameters
    nsweeps = 5
    maxdim = [10,20,50,100,200]
    cutoff = [1E-10]
```

```
        # running normal DMRG
        energy, psi = dmrg(H, psi0; nsweeps, maxdim, cutoff)

        # compressing the cores
        maxmultirank = 4:maxlinkdim(psi)
        approx_err = zeros(maxlinkdim(psi)-3)
        H_exp_err = zeros(maxlinkdim(psi)-3)

        for j in maxmultirank
            psi_comp = deepcopy(psi)
            for i in 2:N-1
                orthogonalize!(psi_comp, i)
                G,U1,U2 = ttcorehosvd(psi_comp[i], j)
                psi_comp[i] = G*U1*U2
            end

            approx_err[j-3] = norm(psi_comp-psi)
            H_exp_err[j-3] = abs(inner(psi_comp',H,psi_comp)
            - energy)
            # check whether the compression preserves QNs
            push!(fluxes, flux(psi_comp))
        end
        push!(approx_err_data, approx_err)
        push!(H_exp_err_data, H_exp_err)
    end
end
```

Bibliography

[1] Alexander Altland and Ben D. Simons. *Condensed Matter Field Theory*. Cambridge University Press, 2nd edition, 2010.

[2] Greg Ballard and Tamara G. Kolda. *Tensor Decompositions for Data Science*. preliminary draft copy edition, October 2024.

[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, July 2011. doi: 10.1561/2200000016.

[4] Garnet Kin-Lic Chan. Density matrix renormalisation group Lagrangians. *Phys. Chem. Chem. Phys.*, 10(23):3454–3459, June 2008. doi: 10.1039/B805292C.

[5] Fabian H. L. Essler, Holger Frahm, Frank Göhmann, Andreas Klümper, and Vladimir E. Korepin. *The One-Dimensional Hubbard Model*. Cambridge University Press, 2005.

[6] Matthew Fishman, Steven R. White, and E. Miles Stoudenmire. The ITensor Software Library for Tensor Network Calculations. *SciPost Phys. Codebases*, page 4, August 2022. doi: 10.21468/SciPostPhysCodeb.4.

[7] Gene Howard Golub and Charles F. Van Loan. *Matrix Computations*. JHU Press, February 2013. ISBN 978-1-4214-0794-4.

[8] Trygve Helgaker, Poul Jørgensen, and Jeppe Olsen. *Molecular Electronic-Structure Theory*. John Wiley & Sons, August 2014. ISBN 978-1-119-01955-8.

[9] Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIREV*, 51(3):455–500, 2009. doi: 10.1137/07070111X.

[10] Andor Menczer, Maarten van Damme, Alan Rask, Lee Huntington, Jeff Hammond, Sotiris S. Xantheas, Martin Ganahl, and Örs Legeza. Parallel Implementation of the Density Matrix Renormalization Group Method Achieving a Quarter petaFLOPS Performance on a Single DGX-H100 GPU Node. *J. Chem. Theory Comput.*, 20(19):8397–8404, 2024. doi: 10.1021/acs.jctc.4c00903.

[11] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. ISBN 978-0-387-30303-1. doi: 10.1007/978-0-387-40065-5.

[12] I. V. Oseledets. Tensor-Train Decomposition. *SIAM J. Sci. Comput.*, 33(5): 2295–2317, January 2011. doi: 10.1137/090752286.

[13] Sebastian Paeckel, Thomas Köhler, and Salvatore R. Manmana. Automated construction of U(1)-invariant matrix-product operators from graph representations. *SciPost Phys.*, 3(5):035, November 2017. doi: 10.21468/SciPostPhys.3.5.035.

[14] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Ann. Phys. (N. Y.)*, 326(1):96–192, January 2011. doi: 10.1016/j.aop.2010.09.012.

[15] Steven R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69(19):2863–2866, November 1992. doi: 10.1103/PhysRevLett.69.2863.