The Density Matrix Renormalization Group and its time-dependent variants

Adrian E. Feiguin

Department of Physics and Astronomy, University of Wyoming, Wyoming, USA 82071

BACKGROUND

Since its creation in 1992, the density matrix renormalization group (DMRG) method [1] has evolved and mutated. From its original formulation in a condensed matter context, it has been adapted to study problems in verious fields, such as nuclear physics and quantum chemistry, to become one of the dominant numerical methods to study strongly correlated systems.

The purpose of these lectures is to provide a clear and pedagogical introduction to the DMRG, and its time-dependent variants, using simple examples, and pieces of code. Reviews on the method abound [2, 3, 4, 5, 6, 7]. In terms of nice introductions, I refer the reader to the lectures notes by Noack and Manmanna, also originated after the Vietri School[6]. When planning this set of lectures, I decided that I would try to conceive them as a natural continuation of this work. Therefore, I strongly encourage the reader to look at Ref.[6] first as a warm-up.

In these lectures, we shall cover many technical aspects of the DMRG, from a "traditional", or "conventional" perspective, describing the theoretical fundamentation, as well as the details of the algorithm, briefly touching on some recent developments in connection to matrix product states (MPS). In the second part, I will describe the time-dependent extensions of the method, illustrating its application with several examples. I will accompany the discussion with pseudo code, and code snippets that can be freely downloaded from `http://physics.uwyo.edu/~adrian/programs/`. Documentation on how to install and run these applications is provided in the appendices. The programs are in C++, but written on top of libraries that implement all the basic objects, such as vectors and matrices. Therefore, the codes read pretty much like Matlab, and hopefully the reader will find little difficulty understanding them.

To conclude, I will provide a basic tutorial of the ALPS (Algorithms for Physics Simulations)[9] DMRG code. ALPS (`http://alps.comp-phys.org/`) provides a framework and collection of libraries and programs for simulating strongly correlated problems. The DMRG application is part of this collection that includes other powerful methods, such as exact diagonalization, quantum and classical Monte Carlo, and dynamical mean-field (DMFT), to mention a few. The ALPS DMRG application puts all the power of the DMRG method in a very friendly and accessible interface that enables anyone with no previous knowledge of the method or coding to run state-of-the-art simulations.

INTRODUCTION

In modern condensed matter theory, the study of strongly correlated systems occupies the foremost area of research. This is motivated by the rich and exotic physics that emerges when the interactions in a quantum system are strong. When interactions behave non-pertubatively, they give rise some

complex and intriguing phenomena, such as the Kondo effect [8], spin-charge separation [10, 11, 12], charge fractionalization, and fractional quantum statistics [13, 14, 15, 16].

In condensed matter we are accustomed to dealing with well defined particles, say electrons. In Landau's Fermi liquid theory the collective behavior of these particles can be described in terms of new excitations so-called "quasi-particles", that possess all the same quantum numbers as the original ones, with some "renormalized" properties, such as the mass. Therefore, embedding the particle in the new interacting environment does affect much our traditional understanding of the excitations of the systems. However, the combination of strong interactions and low dimensionality can have some dramatic effects. For instance, in fermionic systems in one spatial dimension, the pervasive nesting in the Fermi surfaces –which in this case consists of just two points at the Fermi level– prevents the application of perturbation theory, and as a consequence, Fermi liquid theory breaks down. In this case, the natural excitations of the systems are described in therms of bosonic collective modes, one carrying the charge quantum number, and the other carrying the spin. This is referred to as "spin-charge separation", and the corresponding low-energy theory as "Luttinger liquid theory" [10, 11, 12]. Another example in two-dimensions is the fractional quantum Hall effect [13, 14, 15, 16]. This is the quintessential interacting problem, where the system has no small parameter, and no "normal" state. The result is the emergence of exotic excitations called "anyons" that carry a fractional charge, and unlike bosons or fermions, obey fractional statistics. There are many more examples where Fermi liquid theory breaks down, or it just does not apply, for instance: High-temperature superconductivity, fractional and non-abelian statistics in quantum Hall systems, spin-liquids, colossal magnetorresistance, heavy-fermion systems...

Studying solid state systems can be extremely challenging, both theoretical and experimentally, with macroscopic number of degrees of freedom, disorder, unknown interactions... Moreover, all of these phenomena occur under very extreme conditions, such as ultra low temperatures, high pressure, and/or high magnetic fields.

From the theoretical perspective, in order to explain some particular physics, it is convenient to start by mapping the problem on to a simplified model, by getting rid of the unimportant high-energy degrees of freedom, in a way that we end up with a Hamiltonian that incorporates only the interactions that are believed to be relevant. However, even though the model may look simpler, its solution may be extremely complex and difficult to obtain, such as in the case of the paradigmatic Hubbard model.

Very recently, technology has enabled us to look into the quantum realm with a new light. On the one hand, thanks to advances in nanofabrication, experimentalist can routinely manufacture nanostructures that resemble artificial atoms –quantum dots–, that can be manipulated with an extreme degree of control[17, 18]. On the other hand, amazing developments in atomic physics have enabled experimentalist to recreate these model Hamiltonians in a lab, using lasers and neutral atoms[19, 20, 21]. These systems can be considered analog quantum simulators, [22]and promise to shed light onto some long standing open questions, such as what is the pairing mechanism in high-temperature superconductivity.

Besides the ability to simulate model Hamiltonians, a remarkable aspect about the two aforementioned developments is that they enable to study quantum systems very far from equilibrium, under very clean experimental conditions, and with an amazing degree of control. It is in this context, that the recently developed time-dependent density matrix renormalization group method has become a unique investigatory technique, and also a design tool for quantum computers and simulators.

Numerical methods

As mentioned before, in low-dimensional highly correlated problems, the strength of the interactions prevents analytical techniques based on perturbation theory of being used. The lack of small parameters for an expansion makes them difficult to control, and attemps of using them could lead to misleading results. For instance, in some cases they are unable to detect or describe exotic states such as spin liquids[23]. For this reason, the past decades have seen an explosion of activity in computational condensed matter, and numerical methods have experienced a remarkable evolution, shedding light on long-standing problems, and usually providing with the most reliable results. One paradigmatic example is Wilson's solution of the Kondo impurity problem using the numerical renormalization group (NRG)[24, 25]. Numerical methods have proven crucial also in Laughlin's solution to the fractional quantum Hall effect[26].

Generally speaking, we need numerical methods to:

- Validate predictions from a theory.
- Understand the physics when theory fails, to serve as a guide toward a new theory.
- Find new phases, or study phases that are analytically intractable.
- Compare competing orders.
- Make quantitative predictions for experiments.

The vast majority of these numerical methods can be grouped into three categories: (i) exact (ii) variational (ii) stochastic.

In the group (i) I will consider exact diagonalization techniques. The basic idea consists essentially in solving exactly the Scrödinger equation by brute force. However, the Hilbert space , or in other words, the size of the basis needed to represent the Hamiltonian, grows exponentially with the size of the systems, limiting their applicability to the study of ground-state properties and a few excited states at, or near zero temperature.

Variational methods rely on an ansatz: a proposed wave-function derived from some physical insight. This wave-function may, or may not describe the actual ground state of the system, but if it does it even in an approximate way, we may gain some enormous knowledge of the system. The Bethe ansatz for one dimensional systems is actually exact when applied to an integrable model, but it can also be an approximation such as in the case of $SU(N)$ models[27]. And Laughlin's wave-function for fractional quantum Hall state at filling fraction $\nu = 1/3$ has more than 95% accuracy[26]. The BCS theory of superconductivity also relies on a variational ansatz, [28] even though is originates from a mean-field treatment.

Quantum Monte Carlo techniques are based on the stochastic sampling of the Hilbert space using random walks. In its different flavors, these methods are extremely powerful, being able to study very large systems at finite temperatures. However, they suffer from the so called "minus-sign problem", that arises from the fermion statistics. This limits their applicability to "unfrustrated models".

The density matrix renormalization group

The Density Matrix Renormalization Group (DMRG) method [1] belongs to the category of variational methods, but relies heavily on exact diagonalization and numerical renormalization group (NRG) ideas. It was introduced by Steve White in 1992 as a development of Wilson's NRG[24]. It is variational because the proposed solution has the very peculiar form of a "matrix-product state" (we shall learn more about this later). However, no a priori assumptions are made about the form of

the coefficients, or the underlying physics. The power of the method is precisely that it is “smart” enough to be able to find for us the best possible wave-function of that form, without any “external bias”. Even though the accuracy is finite, it is totally under control, and we can obtain results that are essentially exact (also referred-to as quasi-exact). Another ingredient is a block decimation process, similar to the one introduced by Wilson. However, the NRG technique has very limited applicability, while the DMRG can be used for a wide range of problems.

The DMRG possesses features that make it extremely powerful: it is able to treat very large systems with hundreds of spins, and to provide the most accurate results for ground-state energies and gaps in low dimensional systems. Since its very early days the DMRG has shown spectacular success and, together with Quantum Monte Carlo, has dominated most of the numerical research in the field.

Time-dependent DMRG

Since its creation, the DMRG has been continuously enhanced with new improvements, and its range of applicability has grown with time, reaching other areas of research, such as quantum chemistry, and nuclear physics. Some notorious developments include the transfer matrix renormalization group,[29] which extends the original ideas to the temperature/imaginary-time domain. However, in all its applications, all the variations of the DMRG method do not differ that much from the original ground-state DMRG method, as conceived by Steve White. It was during the past five years that the DMRG method has experienced its most remarkable evolution, announcing the beginning of a new DMRG era.

This is due to a convergence with ideas coming from the field of Quantum Information. Recent papers have made this connection very transparent, re-defining DMRG in the context of quantum information theory[30, 32, 33]. As a consequence, we now have a much better understanding of the method, its efficiency, its range of applicability, and how we can better generalize if to study two-dimensional systems. Most of these developments rely on understading the behavior of the entanglement in quantum systems, and also the intrinsic properties of matrix-product states.

The time-dependent Density Matrix Renormalization Group [34, 35] extends the capabilities of the method even further into the time domain, by solving the time-dependent Schrödinger equation. The original ideas the lead to the development of the time-dependent DMRG were first outlined by G. Vidal, when he conceived his Time Evolving Block Decimation method (TEBD) [30]. Immeditely after, it was realized by several groups that this could also be formulated within a DMRG framework [34, 35]. However, we should mention that the idea of using the DMRG to solve the time-dependent Schrödinger equation was not new, and had beed previously attempted with some limited success[36].

Since the time-dependent DMRG simulates the time-evolution of strongly correlated system with unprecedented accuracy, one can employ the technique to study time-dependent correlation, and consequently, extract the entire excitation spectrum and density of state of the system from the spectral functions. It can also be used to study systems out of equilibrium, real-time transport, quantum quenches, response to external perturbations, quantum control and decoherence, among other problems. In a subsequent development, it was shown that the method can easily be extended to study time-evolution in imaginary time, and used to calculate thermodynamic properties [37, 38].

MODEL HAMILTONIANS

One might naively think that the properties of a collection of atoms can be guessed from the properties of the individual constituents. But in fact, when atoms form a solid, or even complex molecules,

the collective behavior od the system can easily defy intuition. We start studying the problem by putting together a collection of atoms, and we first may notice that electronic orbitals hybridize to form bands. The core electrons remain close to the nuclei, and their wave-functions are generally very localized. In general, valence electrons have more extended wave-functions and move more freely. When interactions are week, we can generally understand the behavior of the system in terms of nearly free electrons, of Femi-liquid theory, for instance. But when interactions are strong, and systems of low dimensionality, exotic behavior and new phases of matter may arise. The procedure to study these problems consists of discarding the high energy degrees of freedom, keeping only those that are more relevant in giving rise to the exotic physics. Therefore we have to leave aside the idea of a first principles or ab-initio calculation, and focus on studying the behavior of the valence electrons and their mutual interactions.

We start by assuming that only the valence electrons are responsible for the physics of interest. We know that we can only have a maximum of two electrons per orbital on each atom of the lattice, each of them with a different spin orientation. Hence, the local state for a single atom, or site of the lattice, will consist of one of the following possible configurations: an empty orbital $|0\rangle$, a single occupied orbital with spin up, $|\uparrow\rangle$, spin down, $|\downarrow\rangle$, or double occupied $|\uparrow\downarrow\rangle \equiv |2\rangle$.

We are going to further simplify the scenario by considering the case of a Mott insulator[39]. In this problem we have a valence electron per atom, with very localized wave-functions. The Coulomb repulsion between electrons on the same orbital is so strong, that electrons are bound to their host atom, and cannot move. For this reason, charge disappears from the equation, and the only remaining degree of freedom is the spin of the electrons. The corresponding local state can therefore be either $|\uparrow\rangle$ or $|\downarrow\rangle$. The only interaction taking place is a process that "flips" two anti-aligned neighboring spins $|\uparrow\rangle|\downarrow\rangle \rightarrow |\downarrow\rangle|\uparrow\rangle$.

Let us now consider a collection of spins residing on site of a –one-dimensional for simplicity– lattice. An arbitrary state of N-spins can be described by using the S^z projection $(\uparrow,\downarrow)$ of each spin as: $|s_1, s_2, ..., s_N\rangle$. As we can easily see, there are 2^N of such configurations.

We shall describe the interactions between neighboring spins using the so-called Heisenberg Hamiltonian:

$$\hat{H} = \sum_{i=1}^{N-1} \hat{\mathbf{S}}_i \cdot \hat{\mathbf{S}}_{i+1} \tag{1}$$

where $\hat{\mathbf{S}}_i = (\hat{S}^x, \hat{S}^y, \hat{S}^z)$ is the spin operator acting on the spin on site i.

Since we are concerned about spins one-half, $S = 1/2$, all these operators have a 2×2 matrix representation, related to the well-known Pauli matrices:

$$S^z = \begin{pmatrix} 1/2 & 0 \\ 0 & -1/2 \end{pmatrix}, S^x = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix}, S^y = \begin{pmatrix} 0 & -i/2 \\ i/2 & 0 \end{pmatrix}, \tag{2}$$

These matrices act on two-dimensional vectors defined by the basis states $|\uparrow\rangle$ and $|\downarrow\rangle$. It is useful to introduce the identities:

$$\hat{S}^{\pm} = \left(\hat{S}^x \pm i\hat{S}^y\right), \tag{3}$$

where S^+ and S^- are the spin raising and lowering operators. It is intuitively easy to see why by looking at how they act on the basis states: $\hat{S}^+|\downarrow\rangle = |\uparrow\rangle$ and $\hat{S}^-|\uparrow\rangle = |\downarrow\rangle$. Their corresponding 2×2 matrix representations are:

$$S^+ = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, S^- = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \tag{4}$$

We can now re-write the Hamiltonian (1) as:

$$\hat{H} = \sum_{i=1}^{N-1} \hat{S}_i^z \hat{S}_{i+1}^z + \frac{1}{2}\left[\hat{S}_i^+ \hat{S}_{i+1}^- + \hat{S}_i^- \hat{S}_{i+1}^+\right] \tag{5}$$

The first term in this expression is diagonal and does not flip spins. This is the so-called Ising term. The second term is off-diagonal, and involves lowering and raising operators on neighboring spins, and is responsible for flipping anti-aligned spins. This is the "XY" part of the Hamiltonian.

The Heisenberg spin chain is a paradigmatic model in condensed matter. Not only it is attractive due to its relative simplicity, but can also describe real materials that can be studied experimentally. The Heisenberg chain is also a prototypical integrable system, that can be solved exactly by the Bethe Ansatz, and can be studied using bosonization techniques and conformal field theory.

In these lectures, we will be interested in obtaining its ground state properties of this model by numerically solving the time-independent Schrödinger equation:

$$\hat{H}|\Psi\rangle = E|\Psi\rangle, \tag{6}$$

where H is the Hamiltonian of the problem, $|\Psi\rangle$ its eigenstates, with the corresponding eigenvalues, or energies E.

We shall use this simple model Hamiltonian throughout these lectures to demonstrate the application of the DMRG method. Generalizing the DMRG to models involving bosonic and fermionic degrees of freedom is a relatively straightforward exercise.

EXACT DIAGONALIZATION

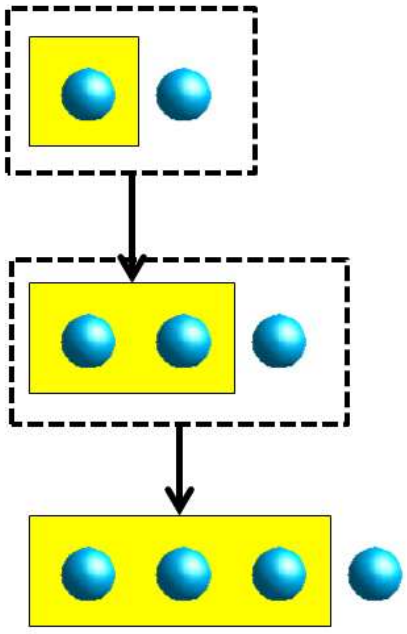


FIGURE 1. Pictorial representation of the Hamiltonian building recursion explained in the text. At each step, the block size is increased by adding a spin at a time.

In this section we introduce a technique that will allow us to calculate the ground state, and even excited states of small Heisenberg chains. Exact Diagonalization (ED) is a conceptually simple

technique which basically consists of diagonalizing the Hamiltonian matrix by brute force. Same as for the spin operators, the Hamiltonian also has a corresponding matrix representation. In principle, if we are able to compute all the matrix elements, we can use a linear algebra package to diagonalize it and obtain all the eigenvalues and eigenvectors [40].

In these lectures we are going to follow a quite unconventional procedure to describe how this technique works, in order to make the transition to DMRG more natural. It is important to point out that this is a quite inefficient and impractical way to diagonalize the Hamiltonian, and more sophisticated techniques are generally used in practice.

Two-spin problem

The Hilbert space for the two-spin problem consists of four possible configurations of two spins

$$\{|\uparrow\uparrow\rangle, |\uparrow\downarrow\rangle, |\downarrow\uparrow\rangle, |\downarrow\downarrow\rangle\} \tag{7}$$

The problem is described by the Hamiltonian:

$$\hat{H} = \hat{S}_1^z \hat{S}_2^z + \frac{1}{2}\left[\hat{S}_1^+ \hat{S}_2^- + \hat{S}_1^- \hat{S}_2^+\right] \tag{8}$$

The corresponding matrix will have dimensions 4×4. In order to compute this matrix we shall use some simple matrix algebra to first obtain the single-site operators in the expanded Hilbert space. This is done by following the following simple scheme: And operator O_1 acting on the left spin, will have the following 4×4 matrix form:

$$\tilde{O}_1 = O_1 \otimes \mathbb{1}_2 \tag{9}$$

Similarly, for an operator O_2 acting on the right spin:

$$\tilde{O}_2 = \mathbb{1}_2 \otimes O_2 \tag{10}$$

where we introduced the $n \times n$ identity matrix $\mathbb{1}_n$. The product of two operators acting on different sites can be obtained as:

$$\tilde{O}_{12} = O_1 \otimes O_2 \tag{11}$$

It is easy to see that the Hamiltonian matrix will be given by:

$$H_{12} = S^z \otimes S^z + \frac{1}{2}\left[S^+ \otimes S^- + S^- \otimes S^+\right] \tag{12}$$

where we used the single spin (2×2) matrices S^z and $S^\pm$. We leave as an exercise for the reader to show that the final form of the matrix is:

$$H_{12} = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 0 & -1/4 & 1/2 & 0 \\ 0 & 1/2 & -1/4 & 0 \\ 0 & 0 & 0 & 1/4 \end{pmatrix}, \tag{13}$$

Obtaining the eigenvalues and eigenvectors is also a straightforward exercise: two of them are already given, and the entire problem now reduces to diagonalizing a two by two matrix. We therefore obtain the well known result: The ground state $|s\rangle = 1/\sqrt{2}[|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle]$, has energy $E_s = -3/4$, and the other three eigenstates $\left\{|\uparrow\uparrow\rangle, |\downarrow\downarrow\rangle, 1/\sqrt{2}[|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle]\right\}$ form a multiplet with energy $E_t = 1/4$.

Many spins

Imagine now that we add a third spin to the right of our two spins. We can use the previous result to obtain the new 8×8 Hamiltonian matrix as:

$$H_3 = H_2 \otimes \mathbb{1}_2 + \tilde{S}_2^z \otimes S^z + \frac{1}{2}\left[\tilde{S}_2^+ \otimes S^- + \tilde{S}_2^- \otimes S^+\right] \tag{14}$$

Here we used the single spin S_1^z, $S_1^\pm$, and the 'tilde' matrices defined in Eqs.(9) and (10):

$$\tilde{S}_2^z = \mathbb{1}_2 \otimes S^z, \tag{15}$$

and

$$\tilde{S}_2^\pm = \mathbb{1}_2 \otimes S^\pm, \tag{16}$$

It is easy to see that this leads to a recursion scheme to construct the $2^i \times 2^i$ Hamiltonian matrix the i^{th} step as:

$$H_i = H_{i-1} \otimes \mathbb{1}_2 + \tilde{S}_{i-1}^z \otimes S^z + \frac{1}{2}\left[\tilde{S}_{i-1}^+ \otimes S^- + \tilde{S}_{i-1}^- \otimes S^+\right] \tag{17}$$

with

$$\tilde{S}_{i-1}^z = \mathbb{1}_{2^{i-2}} \otimes S^z, \tag{18}$$

and

$$\tilde{S}_{i-1}^\pm = \mathbb{1}_{2^{i-2}} \otimes S^\pm, \tag{19}$$

This recursion algorithm can be visualized as a left 'block', to which we add new 'sites' or spins to the right, one at a time, as shown in Fig.1.The block has a 'block Hamiltonian', H_L, that is iteratively built by connecting to the new spins through the corresponding interaction terms. A sample code is described in the Appendices, Listing 1.

TRUNCATED DIAGONALIZATION: THE NUMERICAL RENORMALIZATION GROUP IDEA

The process outlined above leads to a simple and elegant recursion scheme that allows one to construct the Hamiltonian matrix by using simple algebra. However, this idea is very impractical. The basis size, or the linear dimension of the matrix, grows with the number of spins N as 2^N. It is clear that this matrix sizes soon become unmanageable by our computer. One way to deal with this problem is by using the symmetries of the Hamiltonian and the lattice to reduce the Hamiltonian into a block form. This leads to powerful algorithms that can diagonalize dozens of spins. However, this strategy also runs out of steam very soon. Another way to deal with this problem can be traced back to Wilson's numerical renormalization group.

Suppose that we are able to diagonalize out Heisenberg spin chain, to obtain the ground state as:

$$|\Psi\rangle = \sum_{s_1, s_2, \ldots s_N} a_{s_1, s_2, \ldots, s_N} |s_1, s_2, \ldots, s_N\rangle \tag{20}$$

where the sum runs over all configurations of N spins. If we plot the weights $|a_{s_1,\ldots,s_N}|^2$ in decreasing order, we may find a structure like the one depicted in the left panel of Fig.2: Most of the weight is concentrated on a couple of configurations, in our case $|\uparrow\downarrow\uparrow\downarrow \ldots\rangle$ and $|\downarrow\uparrow\downarrow\uparrow \ldots\rangle$, and the rest of

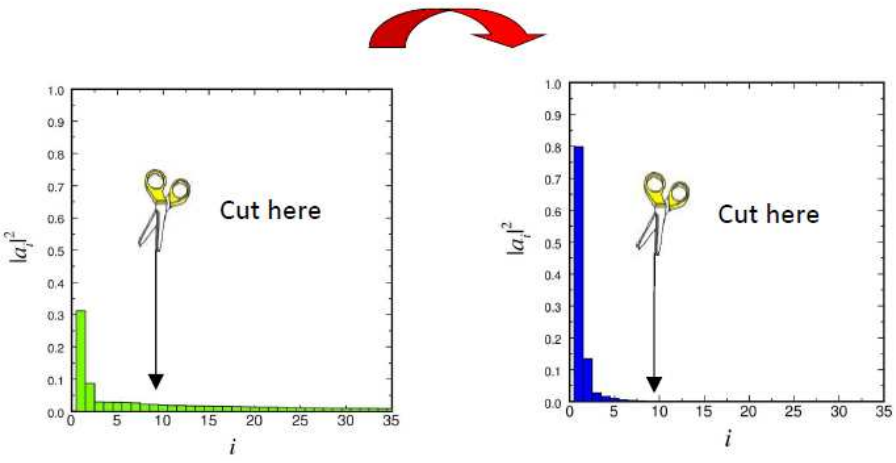


FIGURE 2. Schematic picture of the ideal basis truncation. Through a change of basis the ground-state weights are more concentrated in a few basis states, such that the loose of information after the truncation is minimal.

the weight is spread over a long tail. Usually only a few important states posess most of the weight, especially in ground states that resemble a classical state such as the antiferromanget. One then might feel inclined to take a pair of scissors, and truncate the basis to a few dozen states with largest weights, and get rid of the rest. However, this long tail of states with small weights are responsible for most of the interesting physics: the quantum fluctuations, and the difference in weight from one state to another in this tail cannot be necessarily ignored, since they are all of the same order of magnitude.

However, one may notice a simple fact: this is a basis dependent problem! What if, by some smart choice of basis, we find a representation in which the distribution of weights is such, that all the weight on the tail is 'shifted to the left' on the plot, as shown on the right panel of Fig.2. Then, if we truncate the basis, we would not need to worry about the loss of 'information'. Of course, this is a nice and simple concept that might work in practice, if we knew how to pick the optimal representation. And as it turns out, this is not in principle an easy task. As we shall learn, what we need is a way to quantify 'information'.

A simple geometrical analogy

Let us consider a vector in two dimensional space $\vec{v} = (x,y)$, as shown in Fig.3. We need two basis vectors $\hat{e}_1$ and $\hat{e}_2$ to expand it as $\vec{v} = x\hat{e}_1 + y\hat{e}_2$. A simple 2D rotation by an angle ϕ would be represented by an orthogonal matrix

$$U = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}, \tag{21}$$

After such a rotation, the new basis vectors will be $\hat{e}'_1 = \cos\phi\hat{e}_1 + \sin\phi\hat{e}_2$, and $\hat{e}'_2 = -\sin\phi\hat{e}_1 + \cos\phi\hat{e}_2$. If we pick the angle ϕ such that $\vec{v}$ is aligned along the new y-axis, parallel to $\hat{e}'_2$, we find that we need only one component to describe the vector in the new basis: $\vec{v} = (0,|v|)$, or $\vec{v} = |v|\hat{e}'_2$. Therefore, we would feel inclined to eliminating the vector $\hat{e}'_1$ from the basis. After truncating the basis, in order to rotate to the new one-dimensional space, we would use a rotation matrix:

$$U' = \begin{pmatrix} \cos\phi \\ -\sin\phi \end{pmatrix}, \tag{22}$$

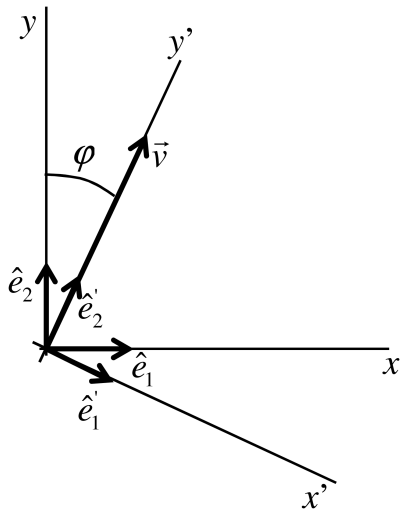


FIGURE 3. Rotation by an angle ϕ to a new reference system in the 2D plane

which still is orthogonal. Now, we clearly see that unless a vector is parallel to $\vec{v}$, we would loose all information regarding the component orthogonal to $\hat{e}'_2$. In other words, this transformation does not preserve the norm, and therefore, it no longer is unitary. For the case of operators represented by 2×2 matrices in the original basis, we find that they will be reduced to a 1×1 matrix, a scalar, or just a simple change of scale. If we apply an operation on the vector, we have to deal with the fact that there will be some loss of information as a result. We would like to think that this loss of information is minimal, meaning that the contributions with support in the orthogonal manifold are very small. This simplified analogy illustrates the consequences one has to deal with when the basis is truncated.

The case of spins

Let us revisit the case of two spins, and look again at the eigenvectors of the Hamiltonian (13). By direct inspection we find that the states $|+\rangle = |\uparrow\uparrow\rangle$ and $|-\rangle = |\downarrow\downarrow\rangle$ are already eigenstates with eigenvalues $E_\pm = 1/4$. The other eigenstates can be found by diagonalizing the remaining 2×2 matrix, yielding

$$|s\rangle = \frac{1}{\sqrt{2}}[|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle]$$
$$|t\rangle = \frac{1}{\sqrt{2}}[|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle]$$

with eigenvalues $E_s = -3/4$ and $E_t = 1/4$ respectively. The transformation matrix, to rotate to this new basis is simple given by the eigenvectors in columns as:

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{23}$$

If we focus on the $|s\rangle$ and $|t\rangle$ states, we see that the 2×2 rotation matrix is equivalent to the geometric rotation Eq.(21) with an angle $\phi = -\pi/4$:

$$\hat{e}'_1 = \frac{1}{\sqrt{2}}\hat{e}_1 - \frac{1}{\sqrt{2}}\hat{e}_2$$
$$\hat{e}'_2 = \frac{1}{\sqrt{2}}\hat{e}_1 + \frac{1}{\sqrt{2}}\hat{e}_2$$

The full transformation occurs in a four-dimensional space, but the two vectors $|+\rangle$ and $|-\rangle$ are untouched. Same as in the geometric case, the transformation preserves the norm (it is unitary!), and angles between basis states (it is an othogonal transformation!), and we can use the eigenvectors as a new basis in which the Hamiltonian is already diagonal. In the course of these lecture we shall use slightly more complicated rotations, in which the states are eigenstates of a different matrix, not necessarily the Hamiltonian.

Now, we found that we do not need the basis states $|+\rangle$ and $-\rangle$ to obtain the ground-state. By discarding these two states we simplify the calculation by reducing a 4×4 eigenvalue problem to a 2×2 problem. If we knew in advance that the ground-state was in the subspace with $S^z_{total} = 0$ we could have formulated the problem directly in this subspace. This is a "trivial" truncation: if we are interested in a state with particular values of the quantum numbers, and if the Hamiltonian does not mix subspaces with different quantum numbers/symmetries, we can "block diagonalize" the Hamiltonian in each subspace separately.

Notice that the geometric analogy consists, in terms of spins, in truncating the Hilbert space to just one basis state $|s\rangle$ in this case, the eigenvector of the Hamiltonian with the lowest eigenvalue.

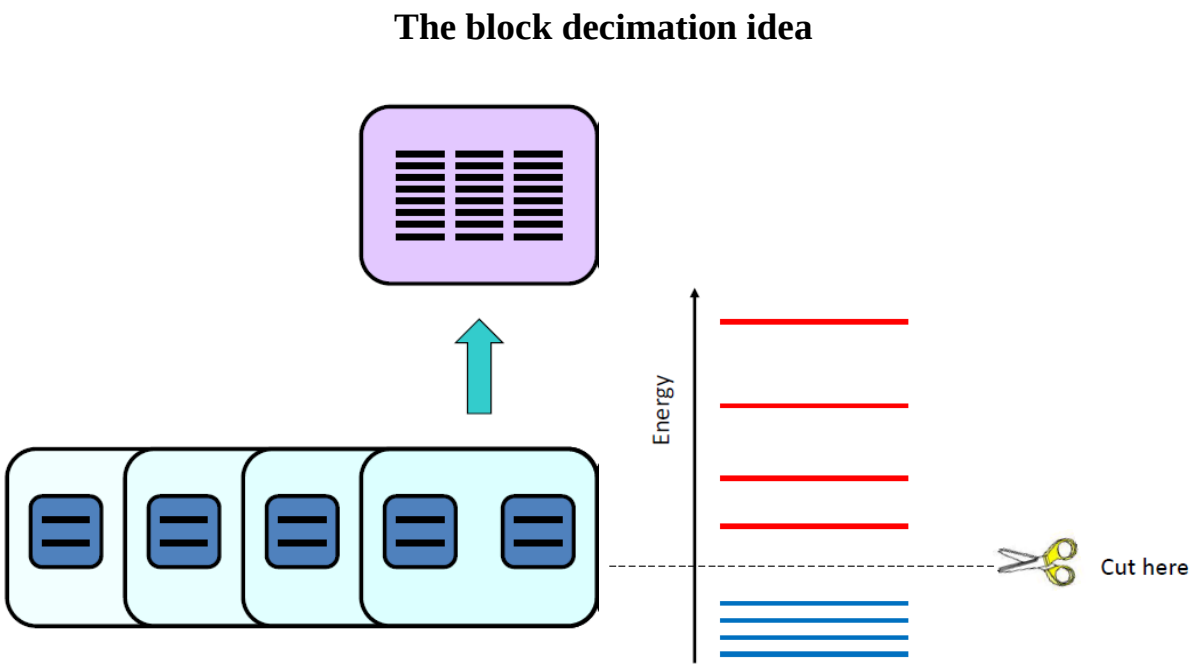
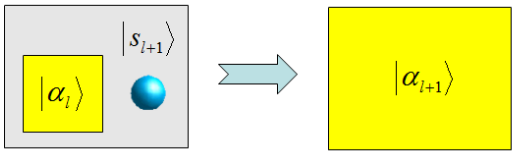


FIGURE 4. In the NRG scheme, we truncate the basis by keeping the *m* eigenstates of the Hamiltonian with the lowest eigenvalues

Let us try a simple idea, using the recursion scheme described above. At every step in the recursion, we add one spin on the right, and our basis dimension grows by a factor 2. At some point during this

recursion, the matrix will be too large to deal with. So let us fix a maximum number of states that we want to keep, m. At certain point during the process, the basis dimension will become larger than m. It is here that we start applying the truncation rule: diagonalize the Hamiltonian matrix exactly, and keep only the m states with *lowest* eigenvalues(see Fig. 4).

As the system grows, the basis of the left block changes as we rotate to the new basis of eigenstates of the Hamiltonian. This is done by using a unitary transformation U. This matrix U is nothing else but the matrix with the eigenstates ordered in columns. Therefore, adding a spin to the block now involves two steps: (i) we need to build the 'tilde' operators as before, and (ii) rotate the Hamiltonian matrix and the tilde operators to the new basis.



$$|\alpha_{l+1}\rangle = \sum_{s_{l+1},\alpha_l} \langle \alpha_l s_{l+1} | \alpha_{l+1} \rangle |\alpha_l\rangle \otimes |s_{l+1}\rangle = \sum_{s_{l+1},\alpha_l} \left(U_L^{l+1}\right)_{\alpha_l s_{l+1},\alpha_{l+1}} |\alpha_l\rangle \otimes |s_{l+1}\rangle$$

FIGURE 5. Adding a site to a block now involves a truncation and a change of basis

Let us assume that our old block before adding a site has a basis $\{|\alpha_{i-1}\rangle\}$, of dimension D_{i-1}, and the site has a basis $\{|s_i\rangle\}$ of dimension d. The new block basis $\{|\alpha_{i-1}, s_i\rangle\}$ has dimension $d \times D_{i-1}$, such that we can easily diagonalize it to obtain all the eigenvalues and corresponding eigenvectors $\{|\alpha_{i+1}\rangle\}$. We build the matrix U as the $D_{i-1} \times D_i$ unitary matrix with the $D_i = m$ eigenvectors with largest eigenvalues in the columns:

$$U_{\alpha_{i-1}s_i,\alpha_i} = \langle \alpha_{i-1} s_i | \alpha_i \rangle. \tag{24}$$

Before the rotation, the operators had matrix elements:

$$\tilde{O}_{\alpha_{i-1}s_i,\alpha'_{i-1}s'_i} = \langle \alpha_{i-1} s_i | \hat{O} | \alpha'_{i-1} s'_i \rangle. \tag{25}$$

We can now rotate all the tilde operators to the new basis as:

$$\begin{aligned} \tilde{O}_{\alpha_i,\alpha'_i} &= \langle \alpha_i | \hat{O} | \alpha'_i \rangle = \sum_{\alpha_{i-1},s_i} \sum_{\alpha'_{i-1}s'_i} \langle \alpha_i | \alpha_{i-1} s \rangle \langle \alpha_{i-1} s_i | \hat{O} | \alpha'_{i-1} s'_i \rangle \langle \alpha'_{i-1} s' | \alpha_i \rangle \\ &= \sum_{\alpha_{i-1},s_i} \sum_{\alpha'_{i-1}s'_i} (U^\dagger)_{\alpha_i,\alpha_{i-1}s_i} \tilde{O}_{\alpha_i\alpha'_i} U_{\alpha'_{i-1}s'_i,\alpha'_i} \end{aligned} \tag{26}$$

Where the new matrices will have dimensions $m \times m$. we can now use these matrices to continue to block-growing process by adding another site. This can be repeated until the energy per site converges, or until we reach a desired system size.

It may seem that this new basis would be a natural choice is we assume that the physics of the problem is described by different manifolds with different energy scales. If we keep the lowest energy states and we get rid of the high energy states we can expect to get the low energy physics right. This in fact is the case in problems such as the Kondo and Anderson impurity problems. However, in strongly correlated, many-body problems such as the Heisenberg chain, this scheme performs poorly.

THE DENSITY MATRIX TRUNCATION: THE KERNEL OF THE DMRG

The problem was solved by Steve White by using what he called the 'density matrix truncation'. He realized (without knowing at the time) that instead of getting rid of high energy states, one has to

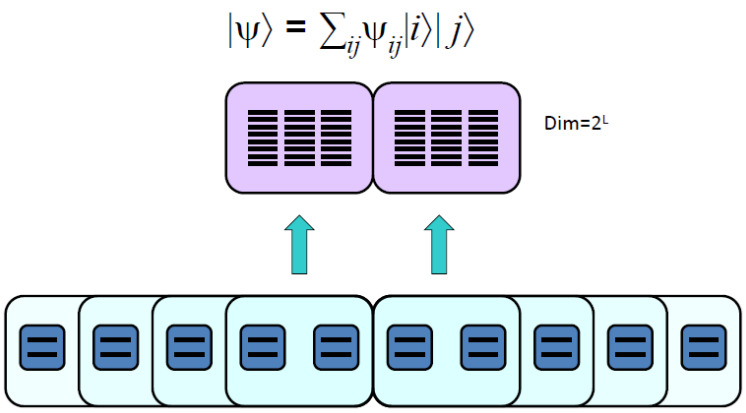


FIGURE 6. The DMRG modifies the NRG idea by adding a second block.

get rid of 'entanglement'. However, the way he formulated the problem did not incorporate the idea of entanglement, a concept that entered the picture much later after quantum information ideas were used to understand why and when the DMRG actually works. Before introducing these ideas, we shall describe the original formulation of the density matrix truncation[1].

In order to introduce this new concept, we are going to use a new scheme: We are going to use two blocks instead of one, a left block, and a right block, as shown in Fig.6. We are going to grow both blocks simultaneously using the same procedure outlined previously: at every step we add one site at the right of the left block, and one site to the left of the right block. The ground state can then be written as:

$$|\Psi\rangle = \sum_{i,j} \Psi_{ij} |i\rangle |j\rangle, \tag{27}$$

where the sum runs over all the states of the left block $|i\rangle$ and right block $|j\rangle$, with the corresponding coefficients Ψ_{ij}.

Now the idea is as follows: once we reach the desired basis dimension m, we shall rotate the left block to a new basis $|i\rangle \rightarrow |\alpha\rangle$. We want to pick these states $|\alpha\rangle$ in such a way that when we truncate the basis, the "distance" between the original ground state $|\Psi\rangle$, and the new, truncated, variational approximation $|\tilde{\Psi}\rangle$, is minimized:

$$S = \left| |\Psi\rangle - |\tilde{\Psi}\rangle \right|^2, \tag{28}$$

where

$$|\tilde{\Psi}\rangle = \sum_{\alpha=1}^{m} \sum_{j} \Psi_{\alpha j} |\alpha\rangle |j\rangle. \tag{29}$$

We are going to anticipate the solution: pick the basis $|\alpha\rangle$ given by the m eigenvectors of the reduced density matrix of the left block with the m largest eigenvalues. In order to justify this result, we first need to introduce some important concepts.

The reduced density matrix

Imagine that we have a bipartite system, composed by subsystem A and subsystem B, as shown in Fig.7. The Hilbert space of the system $A+B$ will be given by the tensor product of the Hilbert spaces of the two subsystems: $H_{A+B} = H_A \otimes H_B$, and will have dimension $D_{A+B} = D_A \times D_B$. Assume that

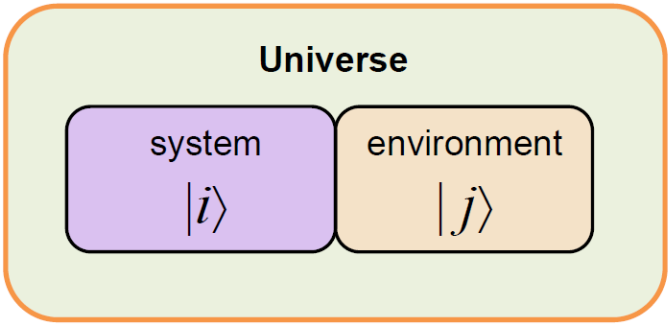


FIGURE 7. In the DMRG, one block acts as the environment for the second one

the state of our system is described by a normalized wave-function $|\Psi\rangle$ that has support on H_{A+B}. We define the reduced density matrix of subsystem A as

$$\hat{\rho}_A = \mathrm{Tr}_B |\Psi\rangle\langle\Psi|. \tag{30}$$

Its corresponding matrix form is

$$\rho_{Aii'} = \langle i|\hat{\rho}_A|i'\rangle = \sum_j \langle ij|\Psi\rangle\langle\Psi|i'j\rangle = \sum_j \Psi_{ij}\Psi^*_{i'j} \tag{31}$$

This operator describes the density matrix of a mixed state, in which the system A is in contact with a bath or environment B. This is the price we have to pay for our ignorance of subsystem B.

The reduced density matrix has some nice properties:

- It is Hermitian (or symmetric in case of real matrices). This means that its eigenvalues are real.
- Its eigenvalues are non-negative.
- The trace equals to unity: $\mathrm{Tr}\rho_A = 1$.
- Its eigenvectors $|\alpha\rangle$ and eigenvalues ω_α form an orthonormal basis.

This means that the reduced density matrix can be re-defined in the new eigenvector basis as:

$$\hat{\rho}_A = \sum_\alpha \omega_\alpha |\alpha\rangle\langle\alpha|; \tag{32}$$

with $\omega_\alpha \geq 0$ and $\sum_\alpha \omega_\alpha = 1$.

This same considerations are valid for the block B.

Exercise: Given a state $|\Psi\rangle$ defined in $A+B$, show that the mean value of an observable $\hat{O}_A$ acting on subsystem A, can be obtained as $\langle\Psi|\hat{O}_A|\Psi\rangle = \mathrm{Tr}\rho_A O_A$.

The Singular Value decomposition (SVD)

Consider an arbitrary matrix Ψ of dimensions $D_A \times D_B$. One can prove that Ψ can de factorized as

$$\Psi = UDV^\dagger, \tag{33}$$

where U is a $(D_A \times D_B)$ unitary matrix, V is a $(D_B \times D_B)$ unitary matrix, and D is a $(D_B \times D_B$ diagonal matrix with real non-negative numbers along the diagonal, and zeroes elsewhere. Since U and V are unitary, they satisfy:

$$UU^\dagger = \mathbb{1};$$
$$VV^\dagger = \mathbb{1}.$$

Their columns are orthonormal vectors, so U and V can be regarded as rotation matrices. The diagonal matrix elements λ_α of D are known as the "singular values" of Ψ.

The Schmidt decomposition

Let us apply the SVD to our quantum wave-function $|\Psi\rangle$ (27), and for illustration, let us assume that $D_B \leq D_A$. The coefficients Ψ_{ij} define a matrix Ψ. After a SVD, they can be re-written as:

$$\Psi_{ij} = \sum_\alpha^{D_B} U_{i\alpha}\lambda_\alpha (V^\dagger)_{\alpha j} = \sum_\alpha^{D_B} U_{i\alpha}\lambda_\alpha V^*_{\alpha j}. \tag{34}$$

The wave-function can now be expressed as:

$$\begin{aligned} |\Psi\rangle &= \sum_i^{D_A}\sum_j^{D_B}\sum_\alpha^{D_B} U_{i\alpha}\lambda_\alpha V^*_{\alpha j}|i\rangle|j\rangle \\ &= \sum_\alpha^{D_B}\left(\sum_i^{D_A} U_{i\alpha}|i\rangle\right)\lambda_\alpha\left(\sum_j^{D_B} V^*_{\alpha j}|j\rangle\right) \\ &= \sum_\alpha^{D_B}\lambda_\alpha|\alpha\rangle_A|\alpha\rangle_B, \end{aligned}$$

where we have defined the states $|\alpha\rangle_A = \sum_i U_{i\alpha}|i\rangle$ and $|\alpha\rangle_B = \sum_j V^*_{\alpha j}|j\rangle$. Due to the properties of U and V, these states define a new orthogonal basis. This final expression is known as the "Schmidt decomposition" of the state Ψ, and the bases $|\alpha\rangle$ as the "Schmidt bases"..

In general, we have that the state Ψ can be written in the new basis as:

$$|\Psi\rangle = \sum_\alpha^r \lambda_\alpha|\alpha\rangle_A|\alpha\rangle_B;\ r = \min(D_A, D_B). \tag{35}$$

In the Schmidt basis, the reduced density matrices for the subsystems A and B are

$$\rho_A = \mathrm{Tr}|\Psi\rangle\langle\Psi| = \sum_\alpha \lambda_\alpha^2|\alpha\rangle_{A\ A}\langle\alpha|, \tag{36}$$

and

$$\rho_B = \sum_\alpha \lambda_\alpha^2|\alpha\rangle_{B\ B}\langle\alpha| \tag{37}$$

At this point, we realize some interesting observations:

- The eigenvalues of the reduced density matrices are $\omega_\alpha = \lambda_\alpha^2$, the square of the singular values.
- The two reduced density matrices share the spectrum.
- The Schmidt bases are the eigenvectors of the reduced density matrices.

Optimizing the truncated wave-function

We here go back to the original problem of optimizing the wave-function in a reduced basis. In order to solve it, we are going to reformulate the question as: Given a matrix Ψ, what is the optimal matrix $\tilde{\Psi}$ with fixed rank m that minimizes the Frobenius distance between the two matrices? It turns out, this is a well known problem called the "low ranking approximation".

If we order the eigenvalues of the reduced density matrix in decreasing order $\omega_1, \omega_2, \ldots \omega_m, \ldots \omega_r$, it is straightforward to see that the Frobenius distance between the two matrices is given by

$$S = \left|\Psi - \tilde{\Psi}\right|^2 = \sum_{m+1}^{r} \omega_i \tag{38}$$

This proves that the optimal basis is given by the eigenvectors of the reduced density matrix with the m largest eigenvalues.

THE DMRG ALGORITHM

The above considerations allow us now to introduce the DMRG algorithm in a very natural way. We are going to present it in the traditional way, starting with the infinite-size algorithm, followed by the finite-size scheme.

Infinite-size DMRG

The main idea behind the infinite-size algorithm consists in growing the left and right blocks by adding one site at a time. As we add sites, the basis of the blocks will grow, until we reach the desired maximum number of states m. At this point we need to start applying the density matrix truncation on both blocks. This process is repeated until we reach a desired system-size, or the error in the energy is below a pre-defined tolerance.

The algorithm illustrated in Fig.8 could be outlined as below:

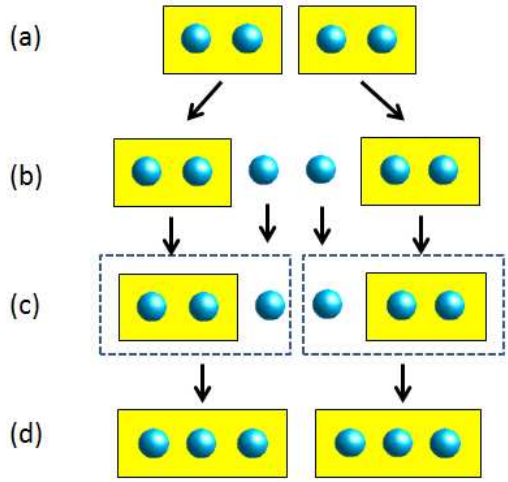


FIGURE 8. Step-by-step illustration of the block-growing scheme in the infinite-size DMRG algorithm: After obtaining the new blocks from the previous step (a), we add a new site to each block (b), we build the superblock and obtain the ground-state (c), and we calculate the reduced density-matrix, and rotate to the basis of the eigenvectors with m largest eigenvalues to build the new blocks for the next step (d).

- Build all the operator matrices for a single-site Hamiltonian, and the operators involved in the interactions between the site an the rest of the system.
- Start growing the blocks by adding single-sites, as outlined in the exact diagonalization section. We assume that the Hilbert space for the single site has dimension d.
- When the size of the bocks become larger than $d \times m$, we start applying the density matrix truncation as follows:
  1. Using a suitable library routine (Lanczos,Davidson), diagonalize the full Hamiltonian (sometimes called super-Hamiltonian) of the two blocks combined (sometimes refereed to as superblock), to obtain the ground state $|\Psi\rangle = \sum_{ij} \Psi_{ij} |i\rangle |j\rangle$.
  2. Calculate the reduced density matrix of the left block, and right blocks. When the system is symmetric under reflections, we only need one of them.
  3. For each of the blocks, diagonalize the density matrix to obtain the full spectrum and eigenvectors.
  4. Truncate the basis by keeping only the m eigenvectors with the largest eigenvalues.
  5. Rotate the Hamiltonian and the operators involved in the interactions between blocks to the new basis.
  6. Add a new site to the left and right blocks, to build new blocks of dimension $d \times m$, and reiterate the diagonalization and truncation steps. Stop when we reach the desired system-size, or the error in the energy is below a pre-defined tolerance.

In the early days of DMRG it was assumed that this scheme would lead to a good approximation of the system properties in the thermodynamic limit. Today we know that he best way to reach the thermodynamic limit is by using the finite-size algorithm on systems of fixed length, and doing a careful finite-size analysis of the results.

Let us now explain some of these steps in more detail.

Adding a single site to the block

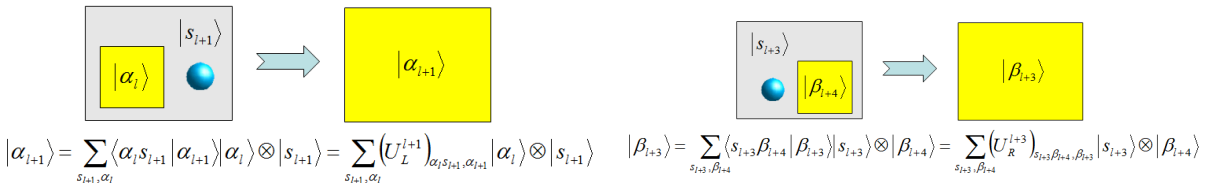


FIGURE 9. Adding sites to the blocks is done in the same way as in the NRG.

Same as we did in the exact diagonalization section, we can add sites to the blocks by performing tensor products of the "tilde" operators on the block, and single-site operators.

Assume that we are in the i^{th} iteration of the algorithm, with our left and right blocks having length i. Let us label our D_L basis states for the left block $\{|\alpha_i\rangle\}$, and our d basis states for the single site that comes to the right $\{|s_{i+1}\rangle\}$ (See Fig.9). When we add the site to the block, we obtain a new basis for the new combined block as $|\alpha_{i+1}\rangle = |\alpha_i\rangle \otimes |s_{i+1}\rangle$.

Let us assume for illustration purposes that we are dealing once more with the Heisenberg chain. All these ideas can be easily generalized to arbitrary models. Same as we did in the exact diagonalization section, we obtain the new Hamiltonian matrix for the combined block as:

$$H_{L,i+1} = H_{L,i} \otimes \mathbb{1}_2 + \tilde{S}^z_{L,i} \otimes S^z + \frac{1}{2}\left(\tilde{S}^+_{L,i} \otimes S^- + \tilde{S}^-_{L,i} \otimes S^+\right). \tag{39}$$

In this expression, the "tilde" operators are in the $|\alpha_i\rangle$ basis, while the others are defined in the single-site basis.

A similar expression applies to the right block, which is obtained from the single site at position $i+2$, with basis $\{|s_{i+2}\rangle\}$ and dimension d, and the right block with basis $\{|\beta_{i+3}\rangle\}$ and dimension D_R:

$$H_{R,i+2} = \mathbb{1}_2 \otimes H_{R,i+3} + S^z \otimes \tilde{S}^z_{R,i+3} + \frac{1}{2}\left(S^+ \otimes \tilde{S}^-_{R,i+3} + S^- \otimes \tilde{S}^+_{R,i+3}\right). \tag{40}$$

Building the super-Hamiltonian

We now need to combine the left and right blocks to form the super-Hamiltonian:

$$\hat{H} = \hat{H}_{L,i+1} + \hat{H}_{R,i+2} + S^z_{i+1} S^z_{i+2} + \frac{1}{2}\left(S^+_{i+1} S^-_{i+2} + S^-_{i+1} S^+_{i+2}\right) \tag{41}$$

where $\hat{H}_{L(R)}$ where obtained above, and only involve terms in the left (right) block. The single sites at positions $i+1$ and $i+2$ were absorbed by the left and right blocks, respectively, so in order to build the interactions, we have to rotate the corresponding operators to the new basis of the blocks. This is

again done in the same spirit of the "tilde" transformation:

$$\begin{aligned} H &= H_{L,i+1}\otimes \mathbb{1}_{D_R\times 2} + \mathbb{1}_{D_L\times 2}\otimes H_{R,i+2} \\ &+ \mathbb{1}_{D_L}\otimes S^z\otimes S^z\otimes \mathbb{1}_{D_R} \\ &+ \frac{1}{2}\mathbb{1}_{D_L}\otimes S^+\otimes S^-\otimes \mathbb{1}_{D_R} \\ &+ \frac{1}{2}\mathbb{1}_{D_L}\otimes S^-\otimes S^+\otimes \mathbb{1}_{D_R} \end{aligned}$$

or:

$$\begin{aligned} H &= H_{L,i+1}\otimes \mathbb{1}_{D_R\times 2} + \mathbb{1}_{D_L\times 2}\otimes H_{R,i+2} \\ &+ \tilde{S}^z_{L,i+1}\otimes \tilde{S}^z_{R,i+2} \\ &+ \frac{1}{2}\tilde{S}^+_{L,i+1}\otimes \tilde{S}^-_{R,i+2} \\ &+ \frac{1}{2}\tilde{S}^-_{L,i+1}\otimes \tilde{S}^+_{R,i+2} \end{aligned}$$

Obtaining the ground-state: Lanczos diagonalization

Once we have a superblock matrix, we can apply a library routine to obtain the ground state of the superblock $|\Psi\rangle$. The two algorithms widely used for this purpose are the Lanczos and Davidson diagonalization. Both are explained to great extent in Ref.[6], so we refer the reader to this material for further information. In these notes we will briefly explain the Lanczos procedure, since we will need some of these concepts later.

The basic idea of the Lanczos method [43, 44] is that a special basis can be constructed where the Hamiltonian has a tridiagonal representation. This is carried out iteratively as shown below. First, it is necessary to select an arbitrary seed vector $|\phi_0\rangle$ in the Hilbert space of the model being studied. If we are seeking the ground-state of the model, then it is necessary that the overlap between the actual ground-state $|\psi_0\rangle$, and the initial state $|\phi_0\rangle$ be nonzero. If no "a priori" information about the ground state is known, this requirement is usually easily satisfied by selecting an initial state with *randomly* chosen coefficients in the working basis that is being used. If some other information of the ground state is known, like its total momentum and spin, then it is convenient to initiate the iterations with a state already belonging to the subspace having those quantum numbers (and still with random coefficients within this subspace).

After $|\phi_0\rangle$ is selected, define a new vector by applying the Hamiltonian $\hat{H}$, over the initial state. Subtracting the projection over $|\phi_0\rangle$, we obtain

$$|\phi_1\rangle = \hat{H}|\phi_0\rangle - \frac{\langle\phi_0|\hat{H}|\phi_0\rangle}{\langle\phi_0|\phi_0\rangle}|\phi_0\rangle, \tag{42}$$

that satisfies $\langle\phi_0|\phi_1\rangle = 0$. Now, we can construct a new state that is orthogonal to the previous two as,

$$|\phi_2\rangle = \hat{H}|\phi_1\rangle - \frac{\langle\phi_1|\hat{H}|\phi_1\rangle}{\langle\phi_1|\phi_1\rangle}|\phi_1\rangle - \frac{\langle\phi_1|\phi_1\rangle}{\langle\phi_0|\phi_0\rangle}|\phi_0\rangle. \tag{43}$$

It can be easily checked that $\langle\phi_0|\phi_2\rangle = \langle\phi_1|\phi_2\rangle = 0$. The procedure can be generalized by defining an orthogonal basis recursively as,

$$|\phi_{n+1}\rangle = \hat{H}|\phi_n\rangle - a_n|\phi_n\rangle - b_n^2|\phi_{n-1}\rangle, \tag{44}$$

where $n = 0, 1, 2, ...$, and the coefficients are given by

$$a_n = \frac{\langle \phi_n | \hat{H} | \phi_n \rangle}{\langle \phi_n | \phi_n \rangle}, \qquad b_n^2 = \frac{\langle \phi_n | \phi_n \rangle}{\langle \phi_{n-1} | \phi_{n-1} \rangle}, \tag{45}$$

supplemented by $b_0 = 0$, $|\phi_{-1}\rangle = 0$. In this basis, it can be shown that the Hamiltonian matrix becomes,

$$H = \begin{pmatrix} a_0 & b_1 & 0 & 0 & ... \\ b_1 & a_1 & b_2 & 0 & ... \\ 0 & b_2 & a_2 & b_3 & ... \\ 0 & 0 & b_3 & a_3 & ... \\ \vdots & \vdots & \vdots & \vdots & \end{pmatrix} \tag{46}$$

i.e. it is tridiagonal as expected. Once in this form the matrix can be diagonalized easily using standard library subroutines. However, note that to diagonalize completely a Hamiltonian on a finite cluster, a number of iterations equal to the size of the Hilbert space (or the subspace under consideration) are needed. In practice this would demand a considerable amount of CPU time. However, one of the advantages of this technique is that accurate enough information about the ground state of the problem can be obtained after a small number of iterations (typically of the order of ~ 100 or less).

Another way to formulate the problem is by obtaining the tridiagonal form of the Hamiltonian starting from a Krylov basis, which is spanned by the vectors

$$\left\{ |\phi_0\rangle, \hat{H}|\phi_0\rangle, \hat{H}^2|\phi_0\rangle, ..., \hat{H}^n|\phi_0\rangle \right\} \tag{47}$$

and asking that each vector be orthogonal to the previous two. Notice that each new iteration of the process requires one application of the Hamiltonian. Most of the time this simple procedure works for practical purposes, but care must be payed to the possibility of losing orthogonality between the basis vectors. This may happen due to the finite machine precision. In that case, a re-orthogonalization procedure may be required.

Notice that the new super-Hamiltonian matrix has dimensions $D_L D_R d^2 \times D_L D_R d^2$. This could be a large matrix. In state-of-the-art simulations with a large number of states, one does not build this matrix in memory explicitly, but applies the operators to the state directly in the diagonalization routine.

Density matrix truncation and the rotation to the new basis

The truncation process is similar to the one use in numerical renormalization group, but instead of using the matrix of eigenvectors of the Hamiltonian, we use the eigenvectors $\{|\alpha\rangle\}, \{|\beta\rangle\}$ of the left and right reduced density matrix. Therefore, the new basis states for the left and right block are related to the states in the previous step as:

$$\begin{aligned} |\alpha_{i+1}\rangle &= \sum_{s_{i+1}, \alpha_i} \langle \alpha_i s_{i+1} | \alpha_{i+1} \rangle | \alpha_i s_{i+1} \rangle = \sum_{s_{i+1}, \alpha_i} (U_L)_{\alpha_i s_{i+1}, \alpha_{i+1}} | \alpha_i s_{i+1} \rangle \\ |\beta_{i+2}\rangle &= \sum_{s_{i+2}, \beta_{i+3}} \langle s_{i+2} \beta_{i+3} | \beta_{i+2} \rangle | s_{i+2} \beta_{i+3} \rangle = \sum_{s_{i+2}, \beta_{i+3}} (U_R)_{s_{i+2} \beta_{i+3}, \beta_{i+2}} | s_{i+2} \beta_{i+3} \rangle \end{aligned} \tag{48}$$

where

$$(U_L)_{\alpha_i s_{i+1}, \alpha_{i+1}} = \langle \alpha_i s_{i+1} | \alpha_{i+1} \rangle \tag{49}$$

and

$$(U_R)_{s_{i+2}\beta_{i+3},\beta_{i+2}} = \langle s_{i+2}\beta_{i+3}|\beta_{i+2}\rangle. \tag{50}$$

If we keep only m states, the matrices $U_{L(R)}$ will have dimensions $D_{L(R)}d \times m$. If the basis had already been truncated in the previous step, then $D_L(R) = m$.

We can now use these transformations to obtain the matrix elements for all the operators in the new truncated basis. For instance, an operator acting on a site inside the left block will be transformed as:

$$\begin{aligned}
(\tilde{O}_{L,i+1})_{\alpha_{i+1},\alpha'_{i+1}} &= \langle\alpha_{i+1}|\hat{O}|\alpha'_{i+1}\rangle \\
&= \sum_{\alpha_i,s_{i+1}} \sum_{\alpha'_i,s'_{i+1}} \langle\alpha_{i+1}|\alpha_i s_{i+1}\rangle\langle\alpha_i s_{i+1}|\hat{O}|\alpha'_i s'_{i+1}\rangle\langle\alpha'_i s'_{i+1}|\alpha'_{i+1}\rangle \\
&= \sum_{\alpha_i s_{i+1}} \sum_{\alpha'_i s'_{i+1}} (U_L^\dagger)_{\alpha_{i+1},\alpha_i s_{i+1}} (\tilde{O_{L,i}})_{\alpha_i s_{i+1},\alpha'_i s'_{i+1}} (U_L)_{\alpha'_i s'_{i+1},\alpha'_{i+1}},
\end{aligned} \tag{51}$$

and a similar expression can be obtained for operators in the right block.

Storing matrices and states

In order to optimize memory usage and performance, we can use the symmetries of the model to store all the matrices in block form. We have already noticed in the two-spin example that we can store the Hamiltonian in block diagonal form, with each block corresponding to a well defined symmetry sector, or quantum number. We can do the same with all our operators, with the main difference being that they may not be diagonal. For instance, The $\hat{S}^z$ operator is diagonal, meaning that it does not mix subspaces with different quantum numbers. We can see in the trivial example of a single site (2) that we only need to know the diagonal elements. The $\hat{S}^+$ operator (4) mixes subspaces with the quntum number S^z differing by $+1$. So we can label the blocks by a pair of quantum numbers or, since we know how the operator changes the quantum numbers, we can use a single index. Again, in the single-site example we only need to store a single number, since the matrix elements are all zeroes, but one. In the code implementation, we can store the blocks as a list of matrices. The main drawback is that we need a lookup table to find a block with given quantum numbers, but this can be done very efficiently. Notice that this idea can be applied to Hamiltonians that conserve the quantum numbers: if the model mixes different subspaces, we may need to store the full matrix.

The same idea applies to the state vectors. We can store them in a list of arrays, or matrices, each corresponding to a subspace with well defined quantum numbers. In the examples shown here, we will not use this practice, since it may distract the reader from the main concepts.

Finite-size DMRG

As we mentioned before, the proper way to reach the thermodynamic limit with DMRG is by studying finite systems and performing a finite-size analysis. In order to study finite system, a generalization of the above ideas needs to be applied. The finite-size DMRG (illustrated in Fig.10) can be summarized as follows:

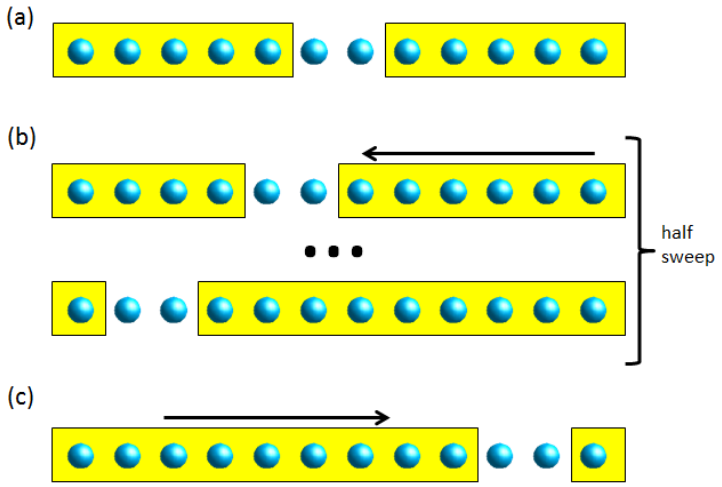


FIGURE 10. Schematic illustration of the finite-size DMRG algorithm: The infinite-size iteration stops when we reach the desired system size. Then, we start sweeping from left to right, and right to left. During the sweeping iterations, one block grows, and the other one "shrinks". The shrinking block is retrieved from the blocks obtained in the previous sweep in the opposite direction, which are stored in memory or disk.

- Run the infinite-size algorithm until the desired system size is reached. During this process, store all the left and right blocks, with their corresponding operators and basis transformations. This step is typically referred to as the "warmup".
- Once the desired system size is reached we start performing "DMRG sweeps", from right-to-left, and left-to-right to optimize the bases and improve accuracy. A left-to-right sweep is described as follows:
  1. Add a site to the left block using the same idea of the infinite-size DMRG. Since the total size of the system needs to be kept fixed, we need to "shrink" the right block. This is done by using the smaller right block from the infinite-size step, or from the previous right-to-left sweep.
  2. Using a suitable library routine (Lanczos,Davidson), diagonalize the super Hamiltonian (sometimes called super-Hamiltonian) of the two blocks combined, same as for the infinite-size DMRG.
  3. Calculate the reduced density matrix of the left block.
  4. Diagonalize the density matrix to obtain the full spectrum and eigenvectors.
  5. Truncate the basis by keeping only the m eigenvectors with the largest eigenvalues.
  6. Rotate the Hamiltonian and the operators of the left block involved in the interactions between blocks to the new basis.
  7. Iterate until reaching the far right end of the system, with a right block containing a single site. This completes the left-to-right sweep.
- Perform a right-to-left sweep, by growing the right block one site at a time, and using the left block from the previous left-to-right sweep.
- Re-iterate the sweeping. Stop when the change in the energy is below a pre-defined tolerance. One typically stops at a point when both blocks have the same size, the "symmetric configuration".

This sweeping process works in a similar fashion as a self-consistent loop, where we iteratively improve the solution. In fact, the DMRG can be formulated as a variational method, in which the variational parameters are continuously improved to minimize an energy functional. Intuitively a way to see it is by imagining a "demon" probing the environment around the block for the optimal states to improve the basis to represent the ground-state. These states are absorbed" inside the block by the density matrix and its eigenvectors.

As described above, the shrinking block is replaced by the block from the previous sweep in the opposite direction. This means that all the information about the block and its operators needs to be stored, either in memory, or dumped on disk or stored in memory.

Obtaining quasi-exact results with the finite-size DMRG

The DMRG accuracy is parametrized by a quantity called the "truncation error", which is nothing but the residue of the trace of the density matrix after the truncation, Eq.(38). This is equal to the sum of the eigenvalues of the discarded states. How well the DMRG performs for a particular problem, and how many states we need to keep in order to achieve the desired accuracy will depend on the behavior of the eigenvalues of the density matrix, something that we will discuss in another section below. We say that the DMRG results are quasi-exact when the accuracy is strictly controlled by the truncation error, and can be improved by increasing the number of states kept m. However, even though the DMRG guarantees that we can obtain quasi-exact results, there are many other factors that need to be taken into account to make sure that the simulation has properly converged. Failing to do so may produce biased results. In order to avoid "mistakes" when using the DMRG, we should pay attention to the following:

- Applying the infinite-size DMRG to finite systems is not quasi-exact, and this has been the source of many mistakes in the past. Sweeping is an essential step in order to get accurate results, and multiple sweeps are typically required. A way to make sure that convergence is achieved is by looking at observables as we sweep, and making sure that they respect the symmetries of the problem. For instance, in a uniform spin chain, correlations from the end sites or the central sites should be symmetric under reflections. Typically, observables do not converge as fast as the energy. Errors in the correlations tend to be higher than errors in the energy.
- A typical analysis of the results consists of looking at the behavior of the energy and observables as a function of the number of states kept m. Sometimes, in challenging situations that require large numbers of states to converge, an extrapolation with m or the truncation error can be attempted.
- In cases where our computer power allows, we can typically guarantee well converged results by fixing the truncation error beforehand. We can ask the algorithm to automatically increase the number of states m such that the truncation error always lies within a tolerance.
- A finite-size scaling requires well converged results for every system size. Poor results for large system sizes can seriously bias the extrapolations and analysis.
- In some peculiar cases it may happen that the system gets trapped in a local minimum. This may occur for several reasons. A common one is doing the warmup or infinite-size sweep with too few states. This problem usually arises in calculations in momentum space, or with large barriers between energy manifolds, or in proximity to a first order first transition. A way to avoid this "sticking" problem is by introducing a source of randomness in the density matrix that may induce the fluctuations necessary to escape the meta-stable state. Another possibility is by introducing fluctuations in some parameter of the model.

- Ultimately, the energy and ground-state are obtained by means of the Lanczos or Davidson diagonalization. It is essential that the diagonalization step is performed with an accuracy superior to the truncation error. Otherwise, the diagonalization errors will dominate over the truncation error, and the ground state will not have the expected accuracy, and may actually include a mixture of excited states (In fact, this may always happen to a certain extent, but a poor diagonalization may magnify the effects of the truncation).

Measuring observables

Let us assume that we have a one-dimensional chain of certain length, and we want to measure correlations between observables acting on different sites, O_i and O'_ji. Two cases can arise: (i) both sites are in separate blocks, or (ii) the two sites are inside the same block. Whether we find ourselves in situation (i) or (ii) will depend on the stage of the sweep. Sometimes we may find that the situation (i) will happen, and sometimes (ii). As we are going to see next, it is more convenient to measure the observables in case (i), which is illustrated in the sample code listed in the appendices, Listing 4.

Operators in separate blocks

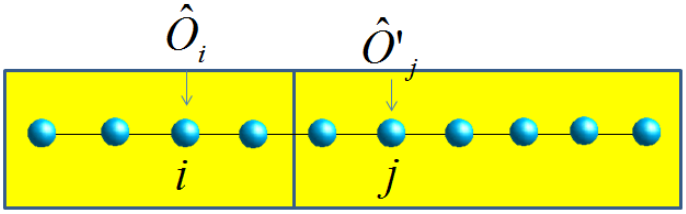


FIGURE 11. Setup for measuring observables acting on sites in separate blocks.

Let us assume a generic situation during the sweep where we find the two operators on separate blocks, as illustrated in Fig.11. Let us denote the basis states for the left block as $\{|\alpha\rangle\}$, and for the right block as $\{|\beta\rangle\}$. The ground state wave-function will be give as:

$$|\Psi\rangle = \sum_{\alpha\beta} \langle\alpha\beta|\Psi\rangle |\alpha\beta\rangle = \sum_{\alpha\beta} \Psi_{\alpha\beta} |\alpha\beta\rangle. \tag{52}$$

It is easy to see that the correlation can be obtained as:

$$\begin{aligned}
\langle \hat{O}_i \hat{O}'_j \rangle &= \langle\Psi|\hat{O}_i \hat{O}'_j|\Psi\rangle \\
&= \sum_{\alpha\beta,\alpha'\beta'} \Psi^*_{\alpha'\beta'} \Psi_{\alpha\beta} \langle\alpha'\beta'|\hat{O}_i \hat{O}'_j|\alpha\beta\rangle \\
&= \sum_{\alpha\beta,\alpha'\beta'} \Psi^*_{\alpha'\beta'} \Psi_{\alpha\beta} \langle\alpha'|\hat{O}_i|\alpha\rangle \langle\beta'|\hat{O}'_j|\beta\rangle \\
&= \sum_{\alpha\beta,\alpha'\beta'} \Psi^*_{\alpha'\beta'} \Psi_{\alpha\beta} (\tilde{O}_i)_{\alpha\alpha'} (\tilde{O}'_j)_{\beta\beta'}.
\end{aligned} \tag{53}$$

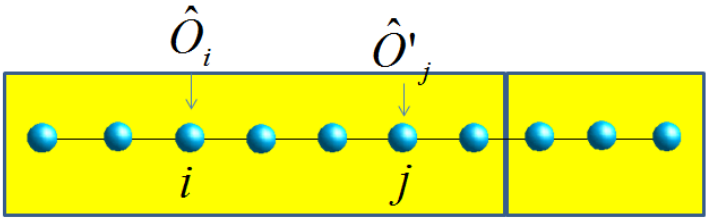


FIGURE 12. Setup for measuring observables acting on sites in the same block.

Operators in the same block

The situation with both operators in the same block is illustrated in Fig.12. The proper way to calculate the correlations it by defining the composite product operator $\hat{O}_{ij} = \hat{O}_i \hat{O}'_j$. The correlation is then expressed as:

$$\begin{aligned} \langle \hat{O}_i \hat{O}'_j \rangle &= \langle \Psi | \hat{O}_i \hat{O}'_j | \Psi \rangle = \langle \Psi | \hat{O}_{ij} | \Psi \rangle \\ &= \sum_{\alpha\beta,\alpha'\beta'} \Psi^*_{\alpha'\beta'} \Psi_{\alpha\beta} \langle \alpha' | \hat{O}_{ij} | \alpha \rangle \langle \beta' | \beta \rangle \\ &= \sum_{\alpha\beta,\alpha'} \Psi^*_{\alpha'\beta} \Psi_{\alpha\beta} (\tilde{O}_{ij})_{\alpha\alpha'}. \end{aligned} \tag{54}$$

We clearly see that the composite operator has to be stored in the block, together with the individual operators. We need to to represent it in the rotated basis the same as we do for the Hamiltonian when we do the truncation. Calculating this quantity as the product of two individual operators in the truncated basis is bad practice and should be avoided.

Clearly, storing and propagating the product operators for all pairs acting on sites i and j can be computationally very expensive. It is therefore convenient to store the individual operators, and calculate the correlations only when the operators are in separate blocks, as illustrated before.

Targeting states

It is important to point out that our basis has been optimized to accurately represent only the ground state. If we wanted to calculate other states, such as excited state, we need to build the density matrix using all these states as "target" states:

$$\rho = \sum_t w_t |\Psi_t\rangle\langle\Psi_t| \tag{55}$$

which is equivalent to the density matrix of a mixed state, with weights w_t, such that $sum_t w_t = 1$. Finding a good combination of weights is a matter of trial and error, and it may depend on the particular problem. Generally, one picks all the weights to be equal.

When one targets multiple states, the number of DMRG states that we need to keep in order to represent them with enough accuracy grows in the same proportion.

Calculating excited states

Sometimes we are interested in calculating excited states in sectors with different symmetry than the ground state. For instance, we may want to obtain the singlet-triplet gap by calculating the ground states in sectors with well defined total spin $S = 0$ and $S = 1$. If we could use the symmetries of the Hamiltonian to be able to restrict our calculation to a sector with well-defined quantum numbers, then this should not be a problem. In these lectures we are leaving the discussion on the use of symmetries aside for the moment. So the problem now is to obtain the excited states, regardless of their quantum numbers. In this case we apply a simple trick that is also used in exact diagonalization calculations: At every step during the simulation, we target the ground state $|\Psi\rangle$ of the Hamiltonian of interest $\hat{H}$, and the ground state $|\Psi_1\rangle$ of the modified Hamiltonian:

$$\hat{H}' = \hat{H} + \Lambda |\Psi\rangle\langle\Psi|. \tag{56}$$

We have introduced a projector on the ground state that shifts its energy by an amount proportional to a constant Λ, that we pick to be sufficiently large. As a result, the ground state will be shifted in energy to the top of the spectrum, leaving the first excited state as the new ground state. As we explained before, in order to accurately represent both states, we need to use the density matrix:

$$\rho = \frac{1}{2}|\Psi\rangle\langle\Psi| + \frac{1}{2}|\Psi_1\rangle\langle\Psi_1| \tag{57}$$

Wave-function prediction

At every step of the sweep we find that we have to obtain the ground-state of the Hamiltonian using some suitable diagonalization routine (lanczos, Davidson). These algorithms converge iteratively to the ground-state, typically starting from some random seed. Depending on the accuracy wanted, one would have to perform a number of iterations, say between 20 and 100, applying the Hamiltonian to a new state at every iteration, until convergence is achieved. S. White [31] realized that the process could be sped up if we could use the ground state from the previous step in the sweeping, as a starting seed for the diagonalization. All one would have to do is transform the old ground-state to the new basis. This process is usually referred to as "wave-function prediction" or "wave-function transformation". As we shall see later, this is a crucial step in the time-dependent DMRG algorithm.

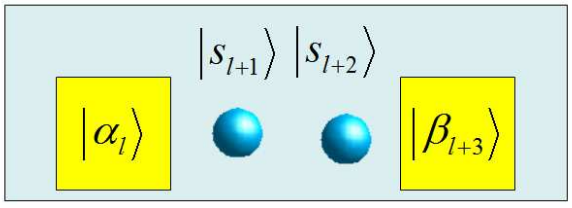


FIGURE 13. Four blocks used to represent the ground-state wave-function for the wave-function transformation.

We assume we obtained the ground-state of our Hamiltonian, which before the change of basis is written as (see Fig.13):

$$|\Psi\rangle = \sum_{\alpha_i, s_{i+1}, s_{i+2}, \beta_{i+3}} \langle \alpha_i s_{i+1} s_{i+2} \beta_{i+3} | \Psi \rangle | \alpha_i s_{i+1} s_{i+2} \beta_{i+3} \rangle \tag{58}$$

After the change of basis, we add a site to the left block, and we "spit out" one from the right block:

$$|\Psi\rangle = \sum_{\alpha_{i+1},s_{i+2},s_{i+3},\beta_{i+4}} \langle \alpha_{i+1} s_{i+2} s_{i+3} \beta_{i+4} | \Psi \rangle | \alpha_{i+1} s_{i+2} s_{i+3} \beta_{i+4} \rangle \tag{59}$$

After some algebra, and assuming that $\sum_{\alpha_i} |\alpha_i\rangle\langle\alpha_i| \approx 1$ and $\sum_{\beta_i} |\beta_i\rangle\langle\beta_i| \approx 1$ after the truncation, one can readily obtain:

$$\begin{aligned} \langle \alpha_{i+1} s_{i+2} s_{i+3} \beta_{i+4} | \Psi \rangle &\approx \sum_{\alpha_i, s_{i+1} \beta_{i+3}} \langle \alpha_{i+1} | \alpha_i s_{i+1} \rangle \langle s_{i+3} \beta_{i+4} | \beta_{i+3} \rangle \langle \alpha_i s_{i+1} s_{i+2} \beta_{i+3} | \Psi \rangle \\ &= \sum_{\alpha_i, s_{i+1} \beta_{i+3}} \left(U_L^\dagger \right)_{\alpha_{i+1}, \alpha_i s_{i+1}} (U_R)_{s_{i+3} \beta_{i+4}, \beta_{i+3}} \langle \alpha_i s_{i+1} s_{i+2} \beta_{i+3} | \Psi \rangle \end{aligned} \tag{60}$$

This operation has relatively little computational cost, especially after considering that it will reduce the ground-state calculation to just a few Lanczos or Davidson iterations.

Generalization to higher dimensions and complex geometries

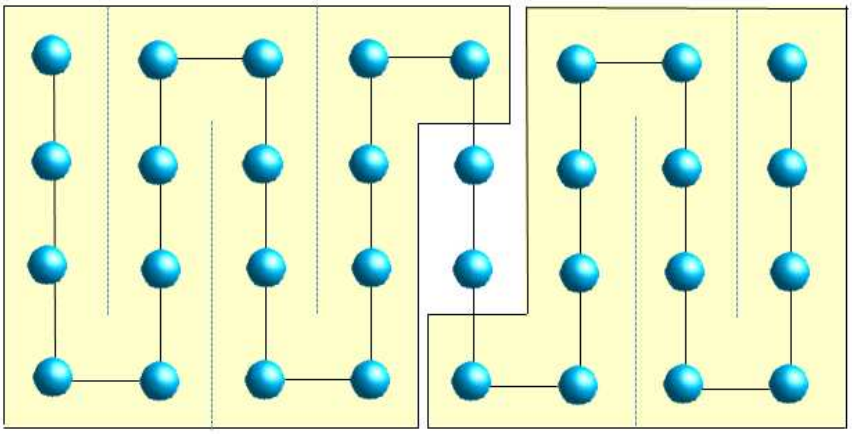


FIGURE 14. Generalizing the DMRG to 2d implies defining a one-dimensional path through the lattice.

The DMRG method was originally introduced to study one dimensional systems. However, it can be extended to higher dimensions in a very straightforward way[45, 46]. Consider for simplicity a system of orbitals in a rectangular arrangement, as shown in Fig.14, with first neighbor interactions, only. We can draw an imaginary path that scans through the lattice following a "snake"-like pattern. We can stretch the snake by putting all the orbitals aligned in a one-dimensional arrangement, and we obtain a system that we can study with the conventional DMRG. There is, however, a price tag to this simplification: the interactions have now long-range.

Other possible solutions have been explored in the literature. For instance, one could choose a vertical band composed by several sites along the *y* direction, and use the one-dimensional scheme in this super-site basis [47]. Another possibility is to use a different snake-like path that still adds one site at a time, but scans through the lattice more symmetrically [48]. Whatever the prefered solution is, in two-dimensional systems we always find the same barrier: entanglement grows with the size of the boundaries between left and right blocks. This affects the behavior of the density matrix spectrum, and as it turns out, we need to keep more dmrg states to achieve good results. The justification for

this behavior will be explained in the following sections. For now, we simply point out that using "cylindrical" boundary conditions, with open boundary conditions along the *x* direction, and periodic along *y*, is the preferred setup for two-dimensional problems.

WHEN AND WHY DOES THE DMRG WORK?

While experimenting with the newly discovered method, Steve White noticed systems with periodic boundary conditions were ill-behaved: in order to achieve a given accuracy for a simple one-dimensional model with periodic boundary conditions would require, for instance $O(10^3)$ states, while for the case of open boundary conditions, one could get away with just a fraction of those, say $O(10^2)$.He experimented with several approaches, but in the end, in seemed as though the only way to deal with this situation was by brute force, keeping lots of states. The reason for these different behaviors was somewhat a mistery, and it remained so until recently, when the quantum information community started exploring algorithms for simulating quantum many body problems (similar to the DMRG) [30], and this behavior was finally understood. These ideas rely on the concept of "quantum entanglement" [49, 50, 51] , and understanding how much information is needed to represent a quantum state faithfully can be quantified in terms of a "entanglement entropy" [53] Quantum information could now explain why certain systems behaved in a certain way, depending on the geometry and topology of the lattice, by understanding the behavior of the spectrum of the reduced density matrices.

Entanglement

Entanglement is a property of quantum mechanical states composed of two or more "objects": since in quantum mechanics the state of a system can be described as a linear superposition of basis states, we find that most of the time, we cannot describe the state of an object, or a part of the system, without knowledge of the rest. To illustrate this idea let us consider the simple case of two spins, and assume there state can be described as:

$$|\Psi\rangle = |\uparrow\uparrow\rangle + |\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle + |\downarrow\downarrow\rangle \tag{61}$$

We can readily see that this is equivalent to

$$\Psi\rangle = (|\uparrow\rangle + |\downarrow\rangle) \otimes (|\uparrow\rangle + |\downarrow\rangle) \tag{62}$$

Therefore, even though we started from a state that seemed to have a complicated structure, we found that in reality the two spins are not entangled: their wave-function can be written as the product of the states of the two individual spins. The two spins carry information individually, and knowing the state of one spin, does not tell us anything about the state of the second spin.

Instead, the following wave-function

$$|\Psi\rangle = |\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle \tag{63}$$

cannot be separated: if we look at one spin, and it is pointing in one direction, we know that the other spin will be pointing in the opposite direction. In fact, for this particular example, the state of one spin carries *all* the information about the state of the second spin. We are going to see later that this case is referred to as the "maximally entangled" state of the two spins.

Entanglement and the Schmidt decomposition

Let us assume that we define a partition in our system into parts A and B, same as we have been doing it all along during our discussion. The generic state of our system can be expressed, once more, as:

$$|\Psi\rangle = \sum_{ij} \Psi_{ij} |i\rangle |j\rangle, \tag{64}$$

where the states $\{|i\rangle\}$ and $\{|j\rangle\}$ live on parts A and B, and have dimensions D_A and D_B, respectively. This means that in order to describe the problem we need to know $D_A \times D_B$ complex coefficients.

Let us re-formulate the original DMRG premise: Can we simplify this state by changing to a new basis? And... what do we mean by "simplifying" the state, anyway?

We have seen, that through a SVD decomposition, we can re-write the state as:

$$|\Psi\rangle = \sum_{\alpha}^{r} \lambda_\alpha |\alpha\rangle |\alpha\rangle \tag{65}$$

where $r = \min(D_A, D_B)$, $\lambda_\alpha \geq 0$, and the states $\{|\alpha\rangle_A\}$ and $\{|\alpha\rangle_B\}$ form an orthogonal basis for the subsystems.

We notice right away, that if the Schmidt rank $r = 1$, then the wave-function reduces to a product state, "disentangling" the two subsystems.

How accurately or faithfully we will be able to represent the state by truncating the basis, will clearly depend on the behavior of the Schmidt coefficients λ_α. If we recall that these are related to the eigenvalues of the reduced density matrices as $\omega_\alpha = \lambda_\alpha^2$, we conclude that the efficiency of the DMRG will be completely determined by the spectrum of the reduced density matrices, the so-called "entanglement spectrum".:

- If the eigenvalues decay very fast (exponentially, for instance), then we introduce little error by discarding the smaller ones.
- Few coefficients mean less entanglement. In the extreme case of a single non-zero coefficient, the wave-function is a product state and completely disentangled
- The same way NRG minimizes the energy... DMRG minimizes the loss of information! The trick is to disentangle the quantum many body state! The closer the state resembles a product state, the more efficient our truncation will be. Let us be clear, the amount of entanglement is in reality always the same, but when we rotate to a new basis, we pick it in such a way that the entanglement is concentrated in as few states as possible, so we can discard the rest with a minimum loss of information.

Quantifying Entanglement

In order to quantify the entanglement, we define a quantity called the "entanglement entropy". There are many definition, we shall pick the so called "von Neumann entanglement entropy":

$$S = -\sum_{\alpha} \lambda_\alpha^2 \log \lambda_\alpha^2. \tag{66}$$

Or, in terms of the reduced density matrix:

$$S = -\mathrm{Tr}\left(\rho_A \log \rho_A\right) = -\mathrm{Tr}\left(\rho_B \log \rho_B\right). \tag{67}$$

To illustrate what this quantity represents, let us look again at the normalized state:

$$|\Psi\rangle = \frac{1}{\sqrt{2}}\left[|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle\right]. \tag{68}$$

We can obtain the reduced density matrix for the first spin, by tracing over the second spin

$$\rho = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{69}$$

We say that the state is “maximally entangled” when the reduced density matrix is proportional to the identity. The entanglement entropy in this case is:

$$S = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = \log 2. \tag{70}$$

In general, if the rank of the Schmidt decomposition is $r = \min(D_A, D_B)$, then the entanglement spectrum will be $\omega_\alpha = 1/r$, and the entanglement entropy of the maximally entangled state will be $S = \log r$.

If the state is a product state:

$$|\Psi\rangle = |\alpha\rangle|\beta\rangle, \tag{71}$$

All the eigenvalues of the reduced density matrices will $\omega_\alpha = 0$ except for one $\omega_1 = 1$, and the entanglement entropy will be $S = 0$.

The area law

We know understand how to quantify the accuracy of the DMRG in terms of entanglement, and how the DMRG works in terms of disentangling the left and right blocks. What we still do not understand is why the DMRG performs so well in one-dimensional cases, and not so-well in two dimensions, and in systems with periodic boundary conditions. We can see that these issues have to be somehow related to the behavior of the entanglement spectrum, and there must be something that makes it behave in a particular way for some problems.

The study of the entanglement properties of quantum systems is a relatively new subject. It has recently become very relevant in condensed matter due to its ability to characterize quantum many body systems. As it turns out, it can be shown that the ground states of certain Hamiltonians, obeying certain conditions, obey what is called "area laws" for the entanglement entropy [52]. That is why the entanglement entropy is sometimes also called “geometric entropy”. This is now a topic of intense research and in this lectures we shall only described the main ideas in a qualitative way.

Consider the ground-state of a local Hamiltonian (with interactions limited to close neighbors). n two spacial dimensions, that is pictorially shown in Fig.15. This state represents what is commonly known as a valence bond solid, in which some bonds form strong bonds, or singlets (63). These singlets are represented in the figure by thick lines. Let us now draw an imaginary partition, as shown with the gray shaded area. The boundary lines will cut a number of singlets, proportional to the length, or perimeter of the partition. Since we know that the entanglement entropy between two spins forming a singlet is $\log(2)$, we conclude that the entanglement entropy between the enclosed region and the outside will be

$$S = \log(2) \times (\#\text{ of bonds cut}) \approx L\log(2) \tag{72}$$

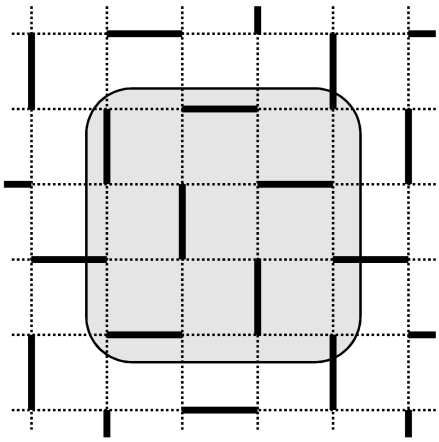


FIGURE 15. Schematic representation of a valence bond solid. Thick lines represent singlets.

Hence, the entanglement entropy is proportional to the area of the boundary separating both regions. This is the prototypical behavior of gaped systems, which we illustrate with this simple example[54]. The same ideas can be generalized to other contexts. Notice that this means that in *gaped* one-dimensional systems, the entanglement entropy between two pieces, left and right, is independent of the size of the partition (again, we are referring to models with short range interactions).

It can be shown using conformal field theoretical arguments [55], that the entanglement entropy of *critical* (gapless) one-dimensional system with periodic boundary conditions obeys the law:

$$S = \frac{c}{3}\log(L) \tag{73}$$

where c is the "central charge" of the system, a measure of the number of gapless modes. For the case of open boundary conditions, a similar expression is obtained:

$$S = \frac{c}{6}\log(L) \tag{74}$$

The factor 2 arises from the simple fact that a system with periodic boundary conditions has *two* boundaries, compared to one in the case of open boundary conditions.

In general, most systems obey the area law, with some exceptions such as free fermions, or fermionic systems with a 1D Fermi surface[56, 57, 58, 59].

Entanglement and the DMRG

In general, the number of DMRG state that we need to keep to represent a state is related to the entanglement entropy between the two blocks as[32, 53]:

$$m \approx \exp(S) \tag{75}$$

Therefore, we can predict the behavior or the algorithm for certain classes of systems:

- Gaped systems in 1D: $m = const$.
- Critical systems in 1D: $m \approx L^{\alpha}$.
- Gaped systems in 2D: $m = \exp(L)$.

- Periodic boundary conditions in 1D: we need to keep the square of the states needed for open boundary conditions, since the boundary area is now doubled.

These rules of thumb give an idea of the computational cost of a simulation. However, the entanglement entropy is not the only factor that determines the behavior of the DMRG, but also the internal structure of the wave function.

The DMRG wave-function: Matrix Product States (MPS)

Shortly after White proposed the DMRG, Ostlund and Rommer investigated the internal structure of the wave-function generated by the algorithm[60, 61], and found that it can be expressed as a "matrix product state" [62]. Let us go back to the change of basis transformation, and first consider the effect on the left block. We have seen that when we add a new site to a block, we obtain the new basis as:

$$|\alpha_i\rangle = \sum_{s_i,\alpha_{i-1}} (U_L)_{\alpha_{i-1},s_i,\alpha_i} |\alpha_{i-1}\rangle \otimes |s_i\rangle \tag{76}$$

We follow Ostlund and Rommer[60, 61], and we change the notation as

$$A[s_i]_{\alpha_{i-1},\alpha_i} = (U_L)_{\alpha_{i-1}s_i,\alpha_i} \tag{77}$$

This expression defines spin dependent matrices $A[s_i]$ as a function of the original unitary transformation U_L. Replacing by this new expression we obtain:

$$|\alpha_i\rangle = \sum_{s_i,\alpha_{i-1}} A[s_i]_{\alpha_{i-1},\alpha_i} |\alpha_{i-1}\rangle \otimes |s_i\rangle \tag{78}$$

We can repeat this transformation for each step, and recursively we find

$$|\alpha_i\rangle = \sum_{s_1,..s_i} A[s_1]_{\alpha_1} A[s_2]_{\alpha_1,\alpha_2} ... A[s_{i-1}]_{\alpha_{i-2},\alpha_{i-1}} A[s_i]_{\alpha_{i-1},\alpha_i} |s_1 ... s_i\rangle \tag{79}$$

Here we notice that the first matrix corresponding to the open end is actually a vector with a single index. As we shall see this will have important implications. The matrices A are derived from the rotation matrices U, and therefore, they also are unitary matrices $A^\dagger A = \mathbb{1}$.

We can repeat the previous recursion from left to right to generate the basis for the right block, yielding:

$$|\beta_i\rangle = \sum_{s_i,..s_L} B[s_i]_{\beta_i,\beta_{i+1}} B[s_{i+1}]_{\beta_{i+1},\beta_{i+2}} ... B[s_L - 1]_{\beta_{L-1}\beta_L} B[s_L]_{\beta_L} |s_i ... s_L\rangle \tag{80}$$

At a given point, our wave-function will be given by:

$$\begin{aligned} |\Psi\rangle &= \sum_{\alpha_i,\beta_{i+1}} \langle \alpha_i \beta_{i+1} | \Psi \rangle | \alpha_i \beta_{i+1} \rangle \\ &= \sum_{\{s\}} A[s_1]_{\alpha_1} A[s_2]_{\alpha_1,\alpha_2} ... A[s_{i-1}]_{\alpha_{i-2},\alpha_{i-1}} A[s_i]_{\alpha_{i-1},\alpha_i} \langle \alpha_i \beta_{i+1} | \Psi \rangle \\ &\times \; B[s_i]_{\beta_i,\beta_{i+1}} B[s_{i+1}]_{\beta_{i+1},\beta_{i+2}} ... B[s_L - 1]_{\beta_{L-1}\beta_L} B[s_L]_{\beta_L} |s_1 ... s_L\rangle \end{aligned} \tag{81}$$

Without loss of generality, we can re-write this expression as:

$$|\Psi\rangle = \sum_{\{s\}} M[s_1]_{\alpha_1} M[s_2]_{\alpha_1,\alpha_2} ... M[s_L - 1]_{\alpha_{L-1}\alpha_L} M[s_L]_{\alpha_L} |s_1 ... s_L\rangle \tag{82}$$

where we have absorbed the matrix $\langle \alpha_i \beta_{i+1} | \Psi \rangle$ into the adjacent matrices. This will clearly break the nice orthogonality properties of the M matrices, but we will not worry about this now. The expression (82) defines a particular class of wave functions called "matrix product states", or MPS, for systems with open boundary conditions. We have found that the DMRG internally assumes this wave-function structure, that can be consider a variational ansatz with lots of variational parameters in the matrix elements. The DMRG somehow manages to optimize the parameters to obtain the "best" wave-function to approximate our true ground-state.

Diagrammatic representation of MPS

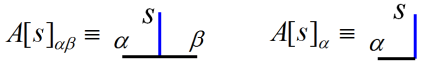


FIGURE 16. Diagrammatic representation of the matrices in the MPS framework.

The coefficient of an MPS is generated by contracting matrices that are identified by a label corresponding to the state of the physical degree of freedom (the spin in our examples). The row and column indices of the matrices correspond to the so-called "bond indices", with a "bond dimension" commonly labeled as D. It is convenient and practical to represent these matrices diagrammatically[63] as shown in Fig.16. The horizontal lines represent the bond indices, while the vertical one represent the physical variable.

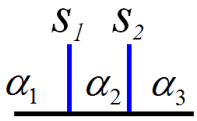


FIGURE 17. Diagrammatic representation of the contraction of two matrices.

It is easy to see that a contraction of indices (a matrix-matrix multiplication) corresponds to joining the corresponding legs of the diagrams, as shown in Fig..17. Using this constructions, we can now represent graphically the coefficients of the MPS (82)as a contraction of indices

$$\begin{aligned} |\Psi\rangle &= \sum_{\{s\}} M[s_1]_{\alpha_1} M[s_2]_{\alpha_1,\alpha_2} \ldots M[s_{L-1}]_{\alpha_{L-1}\alpha_L} M[s_L]_{\alpha_L} |s_1 \ldots s_L\rangle \\ &= \sum_{\{s\}} M[s_1] M[s_2] \ldots M[s_L] |s_1 \ldots s_L\rangle \\ &= \sum_{\{s\}} \prod_{i=1}^{L} M[s_i] |s_1 \ldots s_L\rangle \end{aligned} \tag{83}$$

The corresponding contraction is shown in Fig.17.

We can define the *inner product* of two matrix product states by contracting the physical indices, which is shown diagrammatically in Fig.18.

An operator acting on a physical degree of freedom (*e.g.* a spin), with matrix elements $\langle s|\hat{O}|s'\rangle$ can be diagrammatically represented as shown in Fig.19.

Matrix Product States enjoy some useful properties, which are beyond the scope of these lectures, and are extensively discussed in the review by Schollwöck [64] and in Refs.[65, 66, 67]. Here we shall briefly touch on the connection between DMRG and MPS to set the foundation for further studies.

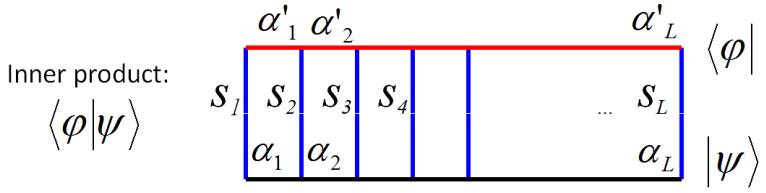


FIGURE 18. Diagram showing the inner product between two matrix product states.

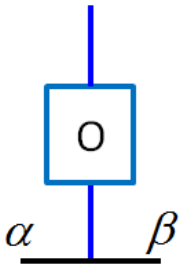


FIGURE 19. Diagram of an operator acting on the physical degree of freedom.

The DMRG wave-function in a Matrix Product Basis

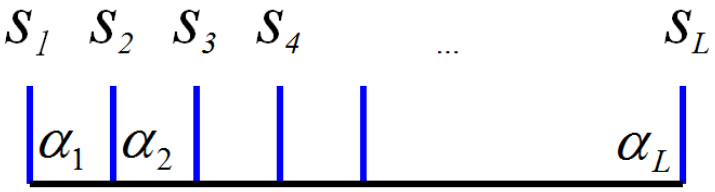


FIGURE 20. Diagram showing the MPS wave-function for a system with open boundary conditions

It is easy to see that the wave-function (82) can now be depicted as in Fig.20 However, we would like to dissect this wave function, and look at it in more detail.

It is useful to introduce a basis for left and right states, as generated in the recursion discussed above in Eqs.(79) and (80). The corresponding diagrams are displayed in Fig.21. Since the A and B matrices are unitary, we get that:

$$\langle\alpha|\alpha'\rangle = \sum_{\{s\}}(A[s_i]^\dagger ... A[s_1]^\dagger A[s_1] ... A[s_i]) = \delta_{\alpha,\alpha'} \text{ and } \langle\beta|\beta'\rangle = \sum_{\{s\}}(B[s_L]^\dagger ... B[s_i]^\dagger B[s_i] ... B[s_L])\delta_{\beta,\beta'} \tag{84}$$

Now we can also express the DMRG wave function in terms of diagrams as shown in Fig.22. Notice that the NRG wave-function can also be expressed in a similar way using only the left basis.

Periodic Boundary Conditions

In the previous sections we have been considering the case with open boundary conditions, since it is the form directly derived from the DMRG. We have also mentioned that the DMRG method does

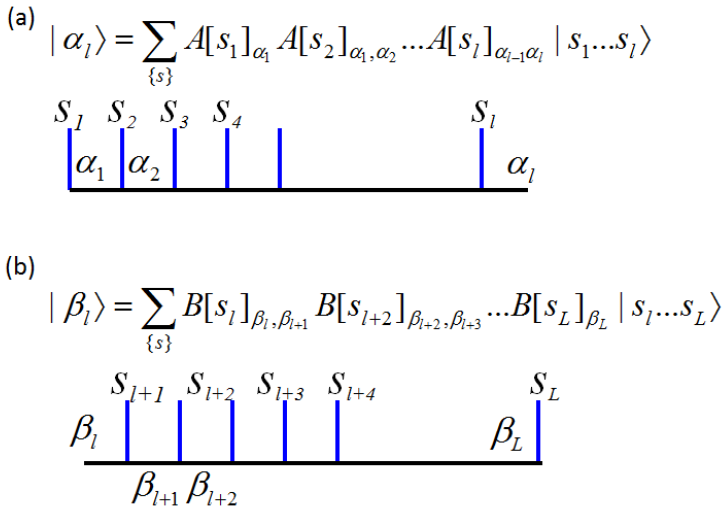


FIGURE 21. Left and right MPS basis for a system with open boundary conditions.

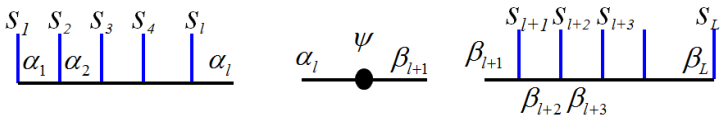


FIGURE 22. Diagramatic representation of the DMRG wave-function.

not perform well in systems with periodic boundary conditions, and that the reason could be traced back to the behavior of the entanglement entropy. In a brilliant paper by Verstraete, Porras and Cirac [72], it was shown that this problem could be circumvented by modifying the structure of the matrix product state to properly account for the entanglement between the first and last sites of the chain. The

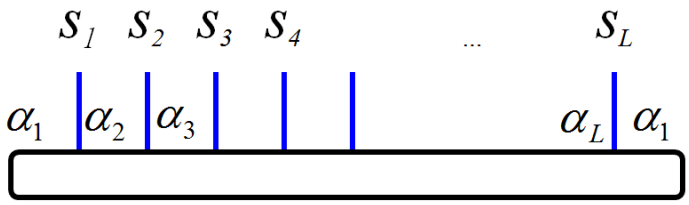


FIGURE 23. Diagram showing a matrix product state for a system with periodic boundary conditions.

elegant idea consists of replacing the first and last vectors in the contraction of the MPS, by matrices:

$$\begin{aligned}|\Psi\rangle &= \sum_{\{s\}} M[s_1]_{\alpha_1,\alpha_2} M[s_2]_{\alpha_2,\alpha_3} \ldots M[s_{L-1}]_{\alpha_{L-1}\alpha_L} M[s_L]_{\alpha_L,\alpha_1} |s_1 \ldots s_L\rangle \\ &= \sum_{\{s\}} \mathrm{Tr}\left(M[s_1]M[s_2]\ldots M[s_L]\right)|s_1 \ldots s_L\rangle \\ &= \sum_{\{s\}} \mathrm{Tr}\left(\prod_{i=1}^{L} M[s_i]\right)|s_1 \ldots s_L\rangle \end{aligned} \tag{85}$$

These extra bond variables account for the missing entanglement, as can be graphically seen in Fig.23.

This solution has also some collateral consequences: since the conventional DMRG is not suitable to optimize it, this led top the development of new algorithms and the generalization of the MPS idea to higher dimension in what is called Projected Entangled Paired States (PEPS), and Tensor Networks in general.

Variational optimization of MPS

As we have seen, the DMRG is basically an algorithm that optimizes a MPS wave-function to approximate the actual ground-state of a quantum many-body system. Since the proposal of MPS and PEPS as variational ansatze, a number of algorithms have been proposed to optimize these wave-functions [68, 69, 70, 71, 30, 32] . In principle, this is a complicated problem: each matrix M has $D \times D$ matrix elements, each of which can be considered an independent variational parameter. Thus, we have to minimize the energy functional with respect to each of these coefficients, leading to the following optimization problem:

$$\min_{\{M\}} \left[\langle\Psi|\hat{H}|\Psi\rangle - \lambda\langle\Psi|\Psi\rangle\right] \tag{86}$$

DMRG does something pretty close to this. In fact, the variational algorithms to solve MPS is one-dimension, with open boundary conditions, lead exactly to the same results, *i.e*, both algorithms are equivalent. Discussing these ideas is beyond the scope of these lectures. We shall mention that the DMRG method can be completely re-formulated in terms of MPS [42], and that the optimization can be carried out using variational Monte Carlo [73, 74]. For a more detailed account and description of the method, we refer the reader to the notes by Schollwöck [64], and Ref.[65, 66, 67].

THE TIME-DEPENDENT DMRG

In this section we describe how to generalize the previous ideas to solve the time-dependent Schrödinger equation using the DMRG concept (For reviews see Refs.[75, 77, 76, 80]) . The problem is simply solving the following well-known second-order differential equation:

$$i\hbar\frac{\partial}{\partial t}|\Psi t\rangle = \hat{H}|\Psi(t)\rangle, \tag{87}$$

which has a formal solution

$$|\Psi(t)\rangle = e^{-it\hat{H}}|\Psi(t=0)\rangle \tag{88}$$

The operator $\hat{U}(t) = e^{-it\hat{H}}$ is the quantum-mechanical time-evolution operator.

Let us assume we know the eigenstates E_n and eigenvalues $|n\rangle$ of the Hamiltonian $\hat{H}$. In this basis, the initial state can be expressed as:

$$|\Psi(t=0)\rangle = \sum_n c_n |n\rangle, \tag{89}$$

and the solution to our problem is simply given by:

$$|\Psi(t)\rangle = \sum_n c_n e^{-itE_n} |n\rangle. \tag{90}$$

In small systems, where full diagonalization of the Hamiltonian is viable, this solves our problem. However, in DMRG we work in an MPS basis of states, and in this basis the coefficients will be wildly oscillating functions of time with different frequencies. On top of that, we are working in a truncated basis that does not account for the full Hilbert space. Our wave-function will evolve in time and may "explore" regions of the Hilbert space that we have not properly accounted for. So the challenge is to adapt our basis as time evolves, such that at every time-step, our wave-function is optimized to represent the actual exact solution of the time-dependent Schrödinger equation, in the same spirit as the original DMRG.

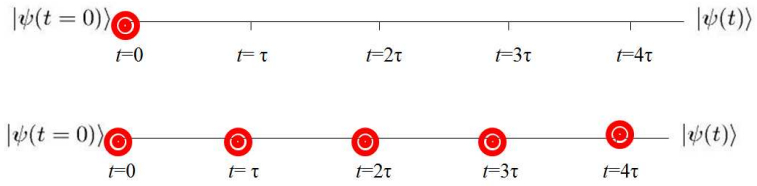


FIGURE 24. The top figure illustrates the original idea by Cazalilla and Marston: They would calculate the initial state, and evolve the wave-function in time without doing sweeping. The figure at the bottom shows Luo *et al*'s idea of targeting the states at every time-step, keeping track of the entire time-evolution of the wave-function.

The first attempts of using the DRMG method for solving the time-dependent problem can be traced back to a seminal paper by Cazalilla and Marston [36]. After solving the time-independent problem using the infinite-system DMRG, they evolved the wave-function in time using Runge-Kutta (R-K) methods. This approach lacked some fundamental ingredients, and therefore it was not quasiexact: they used the infinite-system DMRG, and used a fixed basis to evolve in time without sweeping. However, this work indicated the path toward the development of more sophisticated algorithms, and signified a remarkable first step. To illustrate its applicability, they simulated time-dependent transport in spinless chains, and they demonstrated that despite its limitations, the method worked well for moderate time ranges. Right after this work, Luo and collaborators [36] attempted to construct a quasiexact algorithm by targeting multiple states, keeping track of the entire story of the wave-function as it evolved in time (see Fig.24). The problem with this idea is that keeping track of the entire history of the state with enough accuracy requires a large number of DMRG basis states, and becomes highly inefficient. However, this scheme becomes quasiexact if one adds sweeping and can control the errors introduced by the Runge-Kutta method.

The adaptive time-dependent DMRG (tDMRG)

The crucial tipping point was the introduction by Vidal [30] of the time-evolving block decimation method (TEBD). In rough terms, this method uses the time-evolution operator in imaginary time to project out the ground-state of a Hamiltonian in a matrix-product basis. The main innovation consisted in the idea of using the Suzuki-Trotter (S-T) decomposition of the time-evolution operator.

The Suzuki-Trotter approach

Experts in quantum Monte Carlo are very familiar with the Suzuki-Trotter decomposition. When dealing with local Hamiltonians, with only nearest-neighbor interactions, it constitutes the most natural way to break-up the evolution operator. The idea is to evolve our ground stat by successive applications of the evolution operator:

$$|\Psi(t)\rangle = (e^{-i\hat{H}\delta t}...e^{-i\hat{H}\delta t})|\Psi(0)\rangle \tag{91}$$

where our Hamiltonian $\hat{H}$ can be decomposed as the sum of individual terms, each involving only nearest-neighbors:

$$\hat{H} = \hat{H}_1 + \hat{H}_2 + \hat{H}_3 ... + \hat{H}_{L-1} \tag{92}$$

One would feel tempted to decompose the evolution operator as:

$$e^{-i\hat{H}\delta t} \neq e^{-i\hat{H}_1\delta t}e^{-i\hat{H}_2\delta t}...e^{-i\hat{H}_{L-1}\delta t} \tag{93}$$

But in general the "bond Hamiltonians" do not commute with their neighboring terms, $[\hat{H}_i, \hat{H}_{i+1}] \neq 0$, and as a consequence:

$$e^{-i(\hat{H}_1+\hat{H}_2)\delta t} \neq e^{-i\hat{H}_1\delta t}e^{-i\hat{H}_2\delta t} \tag{94}$$

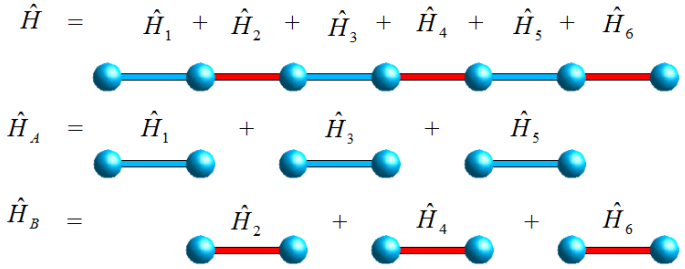


FIGURE 25. Suzuki-Trotter break-up scheme: separate the Hamiltonian into two pieces, one containing the odd links, and one containing the even ones.

In order to "cook up" a better break-up scheme, we start by grouping the terms into two pieces (see Fig.25) : a Hamiltonian $\hat{H}_A = \hat{H}_1 + \hat{H}_3 + ...$ containing only the odd terms, and a Hamiltonian $\hat{H}_B = \hat{H}_2 + \hat{H}_4 + ...$ containing the even terms. It is easy to see that all the terms inside $\hat{H}_A$ and $\hat{H}_B$ commute with each other, since they do not share any sites. Therefore, the evolution operator for the odd links can be *exactly* expanded as:

$$e^{-i\hat{H}_A\delta t} = e^{-i\hat{H}_1\delta t}e^{-i\hat{H}_3\delta t}... \tag{95}$$

and a similar result for $\hat{H}_B$.

One can show, that a first order S-T expansion can be expressed as:

$$e^{-i(\hat{H}_A+\hat{H}_B)\delta t} = e^{-i\hat{H}_A\delta t}e^{-i\hat{H}_B\delta t} + O(\delta t^2) \tag{96}$$

The second-order expansion is written as

$$e^{-i(\hat{H}_A+\hat{H}_B)\delta t} = e^{-i\hat{H}_A/2\delta t}e^{-i\hat{H}_B\delta t}e^{-i\hat{H}_A/2\delta t} + O(\delta t^3) \tag{97}$$

and fourth order expansion as:

$$e^{-i(\hat{H}_A+\hat{H}_B)\delta t} = e^{-i\hat{H}_A\delta t\theta/2}e^{-i\hat{H}_B\delta t\theta}e^{-i\hat{H}_A\delta t(1-\theta)}e^{-i\hat{H}_B\delta t(1-2\theta)e^{-i\hat{H}_A\delta t}(1-\theta)}e^{-i\hat{H}_B\delta t\theta}e^{-i\hat{H}_A\delta t\theta/2} + O(\delta t^3) \tag{98}$$

with $\theta = 1/(2-2^{1/3})$. This last expression is also known as the Forest-Ruth breakup [79]. We point out that the error in the expansion is typically referred-to as the "Trotter error".

The application of these operators reduces to taking two nearest neighbor sites, and evolving using a "bond evolution operator" $e^{-i\hat{H}_i\delta t}$ (with their corresponding phases). These bond evolution operators have a very simple matrix representation. For instance, in the case of the Heisenberg model that concerns us:

$$\hat{H}_i = \hat{\mathbf{S}}_i \cdot \hat{\mathbf{S}}_{i+1} \tag{99}$$

Using a two-site basis:

$$|ss'\rangle = \{|\uparrow\uparrow\rangle, |\uparrow\downarrow\rangle, |\downarrow\uparrow\rangle, |\downarrow\downarrow\rangle\} \tag{100}$$

we can easily obtain the 4×4 Hamiltonian matrix:

$$H_{i,i+1} = \begin{pmatrix} 1/4 & & & 0 \\ & -1/4 & 1/2 & \\ & 1/2 & -1/4 & \\ 0 & & & 1/4 \end{pmatrix}, \tag{101}$$

In order to obtain the exponential of the Hamiltonian matrix we follow these steps:

1. Diagonalize the matrix and calculate the eigenvalues and eigenvectors.
2. Calculate the exponential of $\hat{H}$ in the diagonal basis, with the exponential of the eigenvalues along the diagonal.
3. Rotate back to the original basis using the transformation matrix with the eigenvectors as columns.

We leave it as an exercise for the reader to calculate the exponential of the matrix (101).

Evolution using Suzuki-Trotter expansions

As we have seen, we can reduce the time-evolution to successive applications of a $d\times d$ matrix on pairs of sites, where d is the local dimension of the sites. This is a very fast en efficient operation. We only need to incorporate this idea in the context of the DMRG. We first realize that we always have two sites in between the DMRG blocks. It is therefore straightforward to apply the bond evolution operator to those two sites. If we express the operator matrix elements as:

$$\langle s_i', s_{i+1}'|e^{-i\hat{H}_{i,i+1}\delta t}|s_i s_{i+1}\rangle\langle = U_{s_i s_{i+1}}^{s_s', s_{i+1}'} \tag{102}$$

we obtain the wave function, after applying this operator as:

$$e^{-i\hat{H}_{i,i+1}\delta t}|\Psi\rangle = \sum_{\alpha_{i-1}\beta_{i+2}} \sum_{s_i, s_{i+1}} \sum_{s_i', s_{i+1}'} \langle \alpha_{i-1} s_i s_{i+1} \beta_{i+2}|\Psi\rangle U_{s_i s_{i+1}}^{s_s', s_{i+1}'} |\alpha_{i-1} s_i' s_{i+1}' \beta_{i+2}\rangle \tag{103}$$

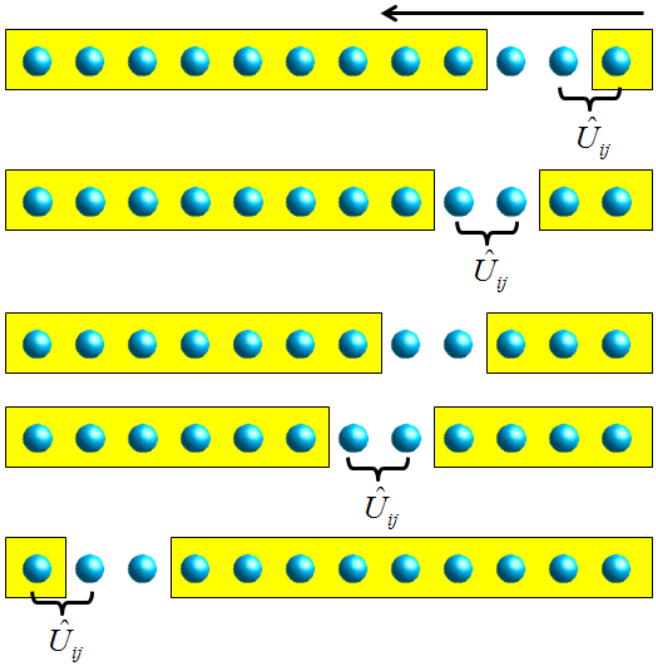


FIGURE 26. Illustration showing a right-to-left half-sweep during the time-evolution. The bond evolution operator is applied only to the odd links during this half-sweep. During the left-to-right half-sweep, only the even links will be evolved.

The tDMRG algorithm

The implementation of the algorithm is very straightforward if we already count with a ground-state DMRG code that implements the ground-state prediction explained in previous sections[34, 35]. The idea is to initialize the state for the time-evolution, by calculating the ground-state of some Hamiltonian, for instance. We can choose to perturb the Hamiltonian by changing some parameter or adding a term to it, or we can also perturb the wave-function by applying some operator to it. We then evolve this initial state using the S-T breakup. There are two fundamental changes with respect to the ground-state scheme: (i) we turn off the Lanczos/Davidson diagonalization. This step will be replaces by the application of the evolution operator. (ii) we have to propagate the wave-function as we evolve in time, and sweep. This is done using the same transformation used for the wave-function prediction. The protocol (illustrated in Fig.26) can be summarized as follows:

- Run the ground-state DMRG to calculate the initial state. We could choose it to be the ground-state of some particular Hamiltonian. Finish your simulation with a full sweep (from left to right, for instance)
- Introduce a perturbation to the problem: we could change the Hamiltonian (*e.g.* by changing the value of some parameter or adding some extra term) or to the state (*e.g.* by applying an operator).
- Turn off the Lanczos/Davidson diagonalization.
- Start the sweeping process, applying the bond evolution operator to the single sites. This will depend on whether you are applying the operator to the even links or odd links. You can choose to use the odd links when sweeping from left to right, and the even links when sweeping form right to left. If the two single sites correspond to the wrong partition, you iterate without doing anything to the state. At each step you have to:
  1. Apply the bond evolution operator to the single sites, if the two sites correspond to the right partition. This step replaces the Lanczos/Davidson diagonalization.
  2. Increase the block size by adding a site, applying the density matrix truncation same as in the ground-state algorithm.
  3. Rotate all the operators to the new basis.
  4. Rotate the wave-function to the new basis, by using the wave-function prediction.
- Depending on the order of your S-T breakup, do as many half-sweeps as required to evolve one time-step.

Contrary to the ground-state DMRG, in which the wave-function transformation is merely a convenient trick to speed-up the calculation, the propagation of the wave-function is now the main purpose of the algorithm. Therefore, the wave-function will change, evolve in time at every-step of the sweep, and the DMRG states will accommodate accordingly to optimize the basis to represent it.

The number of half-sweeps required to evolve a single time-step will depend on the order of the S-T breakup. If we choose to use the 2nd order expansion, we need three half-sweeps: (i) a right-to-left sweep applying the evolution operator to the even links (ii) a left-to-right sweep applying the evolution operator to the odd links, and (iii) another right-to-left sweep acting on the even links. Typically one would perform another half-sweep to do measurements, without evolving in time (we cannot measure in an intermediate step during the time-evolution!).

The initial state and the perturbation applied will depend on the problem of interest. We will discuss some typical examples in the following sections.

Time-step targeting method

The application of the S-T break-up is limited to "nice" Hamiltonians with only nearest-neighbor iterations. If we want to study a problem with long range iterations, or with higher dimensionality, we need to find a different approach. One idea would be to modify the original proposal of Cazalilla and Marston, and Luo *et al.* [36] to adapt our basis in a similar fashion as the S-T approach, but without the computational cost of keeping track of the entire history of the state. In order to emulate the behavior of the S-T approach, we need to target *one time-step accurately*, before moving to the next step. To do so, we keep track of the states at some intermediate points between a time t and $t + \delta t$ (See Fig.27). Once we have performed a couple of half-sweeps to adapt the basis, we evolve one time-step.

In the original proposal [83], the time evolution was done by using Runge-Kutta integration.

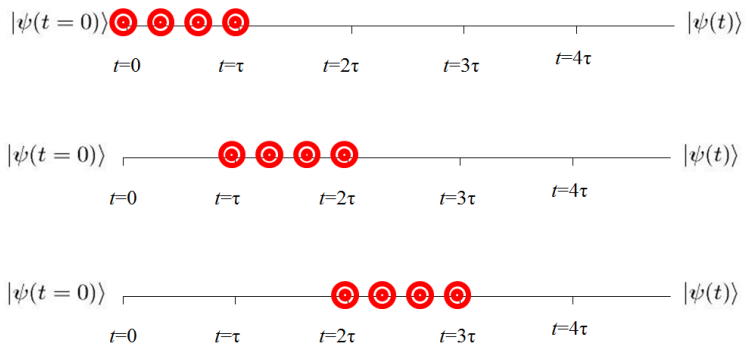


FIGURE 27. Time-targeting idea: at every time step, target four intermediate states to optimize the basis for the time evolution.

The intermediate states can be easily constructed as a spline, using the same auxiliary vectors. The standard fourth order Runge-Kutta method relies on a set of four auxiliary vectors:

$$\begin{aligned} |k_1\rangle &= \delta t \hat{H}(t)|\Psi(t)\rangle \\ |k_2\rangle &= \delta t \hat{H}(t+\delta t/2)\left[|\Psi(t)\rangle + 1/2|k_1\rangle\right] \\ |k_3\rangle &= \delta t \hat{H}(t+\delta t/2)\left[|\Psi(t)\rangle + 1/2|k_2\rangle\right] \\ |k_4\rangle &= \delta t \hat{H}(t+\delta t)\left[|\Psi(t)\rangle + |k_3\rangle\right] \end{aligned} \tag{104}$$

The final state at time $t+\delta t$ is given by:

$$|\Psi(t+\delta t)\rangle \approx |\Psi(t)\rangle + \frac{1}{6}\left[|k_1\rangle + 2|k_2\rangle + 2|k_3\rangle + |k_4\rangle\right] + O(\delta t^5) \tag{105}$$

We need to decide how to interpolate between these two states at times t and $t+\delta t$. We choose to target the states at times t,$t+\delta t/3$,$t+2\delta t/3$, and $t+\delta t$. A way to interpolate between two time step is by using a spline, for instance. We can actually approximate the states at intermediate time by using the same auxiliary vectors defined above. The states at times $t+\delta t/3$ and $t+2\delta t/3$ can be obtained, with an error of $O(\delta t^4)$, as

$$\begin{aligned} |\Psi(t+\delta t/3)\rangle &\approx |\Psi(t)\rangle + \frac{1}{162}\left[31|k_1\rangle + 14|k_2\rangle + 14|k_3\rangle - 5|k_4\rangle\right] \\ |\Psi(t+2\delta t/3)\rangle &\approx |\Psi(t)\rangle + \frac{1}{81}\left[16|k_1\rangle + 20|k_2\rangle + 20|k_3\rangle - 2|k_4\rangle\right]. \end{aligned} \tag{106}$$

In practice we proceed as follows: we can choose one half-sweep to correspond to a time step. At every step during the sweeping iteration we calculate the four Runge-Kutta vectors. We obtain the new density matrix by targeting the four states $|\Psi(t)\rangle$, $|\Psi(t+\delta t/3)\rangle$, $|\Psi(t+2\delta t/3)\rangle$, and $|\Psi(t+\delta t)\rangle$. We advance in time only on the last step of the half sweep. We could alternatively choose to perform more than one half-sweep to make sure that the basis is adequate to evolve one time-step. We typically find the a single half-sweep is sufficient. The method used to evolve in time in this last step is arbitrary, and need not be the Runge-Kutta method. Whatever the method of choice is, the calculation cost of this last time is very small, so more sophisticated approaches are viable. Typically one could perform

10 iterations of the Runge-Kutta algorithm using a step $\delta t/10$. Alternatively, one could evolve using the exponential of the Hamiltonian in a Krylov basis, which is obtained by the same procedure as the Lanczos diagonalization. The advantage of this last approach is that the evolution is unitary, while Runge-Kutta breaks unitarity. However, we point out that the truncation to a finite number of DMRG states introduced non-unitarity anyway, so the Lanczos procedure has in principle no special advantage, and in practice they are comparable in accuracy.

The target states can be weighted equally, or not. Tests using different combinations of weights indicate that the optimal choice is $w_1 = w_4 = 1/3$ and $w_2 = w_3 = 1/6$.

Sources of error

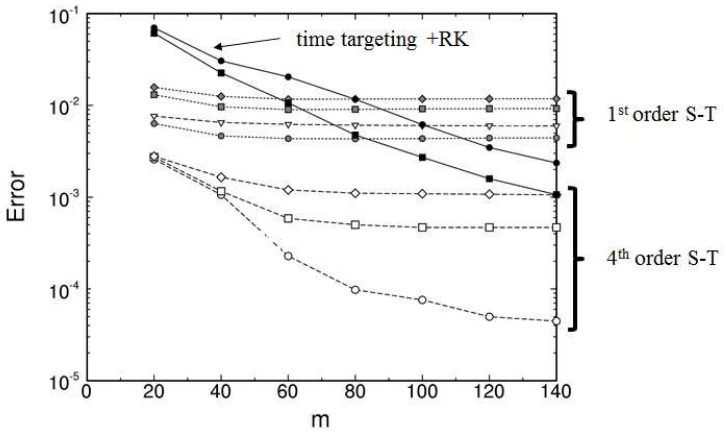


FIGURE 28. Error at time $t = 8$ for the Haldane chain ($L = 32$), as a function of the number of states kept m. We show results using a 1st order Suzuki-Trotter break-up (gray symbols), 4th order Suzuki-Trotter (empty symbols), and 4th order Runge-Kutta (filled symbols). See Ref.[83] for more details.

There are two fundamental sources of error: (i) The truncation error, and (ii) the error introduced in the time evolution by either S-T or the integration method of choice. The truncation error can in principle be controlled by keeping more DMRG state as the entanglement in the system grows. How the entanglement behaves during the time evolution is a topic of intense research, and will depend on the kind of perturbations introduced into the problem. We shall discuss this later. The Suzuki-Trotter error can also be controlled by using high order expansions or small time steps. In fourth-order schemes, such as Runge-Kutta or Forest-Ruth, the error is dominated by the truncation error.

For instance, in S-T simulations, the error typically decreases with increasing number of DMRG states, but it eventually plateaus and stays constant (see Fig.28). This occurs when the error is dominated by the S-T error. On the other hand, in the R-K simulations, the error steadily goes down, meaning that for the number of states kept in these simulations, the error is dominated by the truncation.

In the time-targeting schemes, we need to keep more DMRG state because we need the basis to optimize four different state at four different times. The general recipe to control the truncation error, both in ground-state, and time-dependent simulations, is to fix the truncation error below certain tolerance, and ask the algorithm to pick the optimal number of state to keep the error within that value. As mentioned before, the number of states needed to maintain a given tolerance may vary, and will typically grow. How fast it grows will depend on the particular problem. Sometimes it grows to a point when it becomes unmanageable by our computer. In that case we have two options: (i) stop the

simulation (ii) continue with a fixed number of states. It is important to point out that option (ii) will likely produce biased results, and the simulation will no longer be quasi-exact.

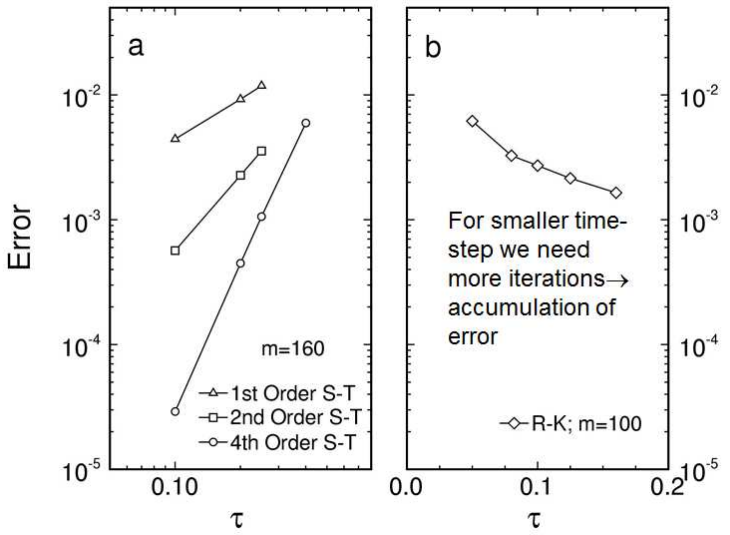


FIGURE 29. Error at time $t = 8$ for the Haldane chain for different time steps τ: a) 1st, 2nd, and 4th order Suzuki-Trotter break-ups and $m = 160$; b) Runge-Kutta and $m = 100$. Time is in units of the Heisenberg exchange J. See Ref.[83] for more details.

A noteworthy fact is that even though the error in the R-K schemes is expected to grow as δt^4, it actually *decreases* with increasing time-step (See Fig.29). This is due to the fact that the error accumulates with the number of steps: the larger the time-steps, the fewer iterations we need to reach a desired time, consequently the smaller is the accumulation of error.

The time-targeting scheme requires one half-sweep to evolve one time-step, while a fourth order S-T expansion will require seven. At the same time, R-K requires multiplying by the Hamiltonian several times per iteration during the sweep, while the S-T only requires a multiplication by a small matrix. On top of that, when evolving in time using S-T we only need to keep a fraction of the states, and we do not need to keep the Hamiltonian, nor the operators needed to build the interactions between blocks. In general, S-T is always the optimal choice for Hamiltonians with nearest-neighbor interactions.

Comparing Suzuki-Trotter and time-step targeting

Here we summarize the main facts to consider when picking a time-evolution method:

- S-T is fast and efficient for one-dimensional geometries with nearest-neighbor interactions.
- The S-T error is controlled by the Trotter error but can be reduced using higher order expansions of smaller time-steps.
- Time-step targeting methods can be applied to ladders, systems with complex geometries, and/or long-range interactions.
- Time-step targeting allows one to use larger time-steps, but it is computationally more expensive and requires keeping more DMRG states.
- In time-step targeting schemes it is convenient (and maybe a good practice), to perform an intermediate sweep to improve the basis before evolving in time.

- Time-evolution using R-K is non-unitary, which can be dangerous. A Krylov expansion is in principle the right choice. We have to remember, however, that unitarity is always broken by the DMRG truncation.

Evolution in imaginary time

The tDMRG method can easily be generalized to study time evolution in imaginary time[84, 85, 86]. The idea consists of enlarging the Hilbert space by adding auxiliary degrees of freedom called "ancillas" that are exact copies of the original, physical states. The use of auxiliary systems to study thermodynamics in quantum systems originated as a key idea in thermo field dynamics [38, 78, 81]. Let the energy eigenstates of the system in question be $\{n\}$. Introduce an auxiliary set of fictitious states $\{\tilde{n}\}$ in one-to-one correspondence with $\{n\}$. Define the unnormalized pure quantum state, in an enlarged Hilbert space,

$$|\psi(\beta)\rangle = e^{-\beta\hat{H}/2}|\psi(0)\rangle = \sum_n e^{-\beta E_n/2}|n\tilde{n}\rangle \tag{107}$$

where $\tilde{n}$ is the matching state to n, β is the inverse temperature, and $|\psi(0)\rangle = \sum_n |n\tilde{n}\rangle$ is our thermal vacuum. Then the partition function is

$$Z(\beta) = \langle\psi|\psi\rangle \tag{108}$$

and we can obtain the exact thermodynamic average of an operator A (acting only on the real states), as

$$\langle A\rangle = Z(\beta)^{-1}\langle\psi|A|\psi\rangle. \tag{109}$$

At $\beta = 0$, the state ψ is the maximally entangled state between the real system and the fictitious system. It is easy to show that if we change basis from the energy eigenstates n to some other arbitrary basis s, ψ is still maximally entangled[78], $|\psi(0)\rangle = \sum_s |s\tilde{s}\rangle$. A natural basis to use is the site basis, where the state of each site i takes on a definite value s_i. One finds

$$|\psi(0)\rangle = \prod_i \sum_{s_i} |s_i\tilde{s}_i\rangle = \prod_i |I_i\rangle \tag{110}$$

defining the maximally entangled state $|I_i\rangle$ of site i with its ancilla.

For practical purposes, it is useful to think of each ancilla as being the anti-site of its site. A state of the ancilla is given opposite quantum numbers to the corresponding state of the real site. In this way, the state of interest has both total charge and total z component of spin equal to zero. This is equivalent to applying a particle-hole transformation onf the maximally entangled state: After the particle-hole transformation, the vacuum state becomes $|I_i\rangle = \sum_{\{s_i\}} |\{s_i\},\{\tilde{s}'_i\}\rangle$, where the spin configurations $\{\tilde{s}'\}$ are copies of $\{s\}$ with all the spins reversed.

A useful example to see how these ideas work is to consider a single spin. We will square the Hilbert space by adding a second fictitious spin, our ancilla, and construct the following state:

$$|I_0\rangle = \frac{1}{\sqrt{2}}\left[|\uparrow\tilde{\downarrow}\rangle + |\downarrow\tilde{\uparrow}\rangle\right] \tag{111}$$

This is the maximally entangled state between the spin and its ancilla. We can easily obtain the reduced density matrix of the physical spin by tracing over the ancilla, yielding:

$$\rho = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{112}$$

This is telling us that the physical spin is effectively at infinite temperature. Of course, temperature is ill defined for a single spin, but the thermal state of a spin at $T = \infty$ is well defined. This also illustrates how the fictitious spin can act as an effective thermal bath for the physical one.

Next, we would want to build the thermal state for a chain of spins at infinite temperature. We simply multiply:

$$|\psi(0)\rangle = |I_0\rangle|I_0\rangle...|I_0\rangle. \tag{113}$$

We can build this state as the ground-state of an artificial "entangler" Hamiltonian acting on both, physical and fictitious spins:

$$\hat{H}_{aux} = -\sum_i (\hat{S}_i^+ \hat{\tilde{S}}_i^- + \hat{\tilde{S}}_i^+ \hat{S}_i^-) \tag{114}$$

where the tilde operators act on the ancillas.

This idea can be generalized to particle-number conservation in fermionic systems. After applying the particle-hole transformation on the ancilla, we obtain the corresponding "anti-state": $|\uparrow,\downarrow\rangle$, $|\downarrow,\uparrow\rangle$, $|0,2\rangle$, and $|2,0\rangle$.

The essence of the ancilla finite temperature method is to start in this local $\beta = 0$ state, and evolve in imaginary time through a succession of temperatures β using the physical Hamiltonian acting only on the physical spins(see Fig.30). The infinite temperature starting state has a correlation length of 0 and requires only one state per block. Since this state is a product state, the entanglement between left and right blocks is zero. As the system evolves in imaginary time, longer range entanglement is produced and the correlation length grows. The number of states needed for a given accuracy grows as the temperature decreases. It is most natural to slowly increase the size of the basis, in order to keep a roughly constant truncation error. One may wish to set a minimum basis set size to make the early evolution essentially exact with little computational cost.

The process outlined above is also called quantum purification of the state. Notice that we start from a product state of maximally entangled pairs, and we end at zero temperature with a state where the physical spins and the ancillas are completely decoupled: a product state between the ground-state of the physical spins, and the state of the ancillas.

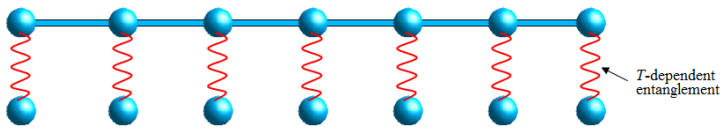


FIGURE 30. Illustration of the ancilla construction. Each wiggly line represent the quantum entanglement between a physical degree of freedom living on the chain, and its copy, or ancilla. The entanglement is temperature-dependent: it is maximum at infinite temperature, and zero at $T = 0$. The Hamiltonian –and the evolution operator– only acts on the physical degrees of freedom, and the ancillas only "evolve" through their entanglement with the physical spins.

Applications

Since the first seminal paper by Cazalilla and Marston [36] it became clear that a time-dependent DMRG method would enable on to study a whole zoo of new problems that were previously beyond reach. In their paper, they illustrated the first application to real-time transport through a resonant level. As soon as the more stable versions of the method were conceived, it was immediately applied to transport through quantum dots, and the calculation of spectral functions. When cold atom experiments started to provide a window into the atomic world, it was soon realized that it was an ideal

playground to study non-equilibrium physics, and the tDMRG was used to study quenches, decoherence, and other interesting phenomena[87, 88]. Some generic scenarios that can be of interest can be grouped into the following classes:

- Start from a system in equilibrium (an eigenstate of the Hamiltonian), add a perturbation term to the Hamiltonian, and evolve under the new Hamiltonian.
- Start from a system in a state that is not an eigenstate, and evolve under the original Hamiltonian.
- Start from a system in equilibrium, and "drive it" with a time-dependent perturbation.
- Start from a system in equilibrium, and apply a perturbation to the state, for instance by "kicking it" with an operator. Evolve under the original Hamiltonian.

The first class of problems are typically referred-to as "quenches". The last two are related to the calculation of response functions. Here I will illustrate examples of these applications in some detail.

Transport

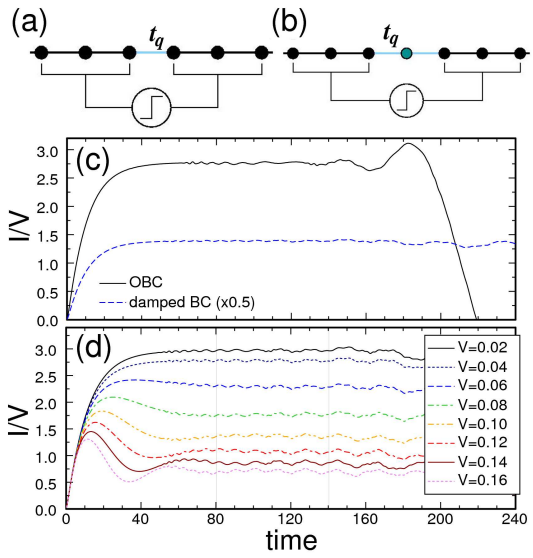


FIGURE 31. (a) Junction model consisting of a weak link connecting two interacting leads. (b) Quantum dot, or resonant level (c) Current through a junction after a step bias is applied. We show results for different boundary conditions. (d) Time-dependent current though the same system, with damped boundary conditions, and different values of the bias V. Time is measured in units of the hopping t_q.

Consider the case of a small interacting region connected to two reservoirs. We typically represent these reservoirs as non-interacting leads. The interacting region could be a quantum dot, for instance. We would like to learn the transport characteristics of the system by looking at its I-V characteristics. The generic setup was proposed by Cazalilla and Marston [36]: start from the system in the ground state, and apply a sudden bias difference to the two leads. This is done by shifting the chemical potential on the left and right leads by $\pm V/2$, where V is the external bias. One the bias is "turned on", a current will start flowing through the system. Typically, after a transient, the current stabilized at a constant value, or plateau, corresponding to the steady state (see Fig.31). We can then extract the full $I-V$ characteristic of the problem by repeating the simulation for different values of the bias. Since the systems we study are finite and have open boundary conditions, the charge will eventually be reflected at the boundaries, and the current will "bounce back" in the opposite direction,

changing sign. This gives us a window of time to reach the steady state where we can average and measure[89, 90, 91, 92, 93, 94, 95].

The transients have a duration that is related to the parameters of the problem, such as the coupling to the quantum-dot, the interactions in the dot, etc. Sometimes, we may find that for certain parameters the system does not reach a plateau before the current reverses. In such a case we have two options: (i) we can use longer leads, or (ii) we can use "damped boundary conditions" [96], or "Wilson leads" [90]. These solutions are very similar, and conceptually work in the same way: we tune the hopping matrix elements in the leads to decay with the distance from the dot, in such a way that the charge gets "trapped" at the boundaries, that act effectively as reservoirs, giving us more time to reach the steady state.

In general we want the algorithm to choose the number of states in such a way that the truncation error is always kept within a desired tolerance, say 10^-7. How fast the number of states grows with time depends on the behavior of the entanglement entropy. This an example of a sudden global quench, where the system is initialized in the ground state, and suddenly perturbed by the bias. We shall later see that the entanglement growth behaves for this class of problems.

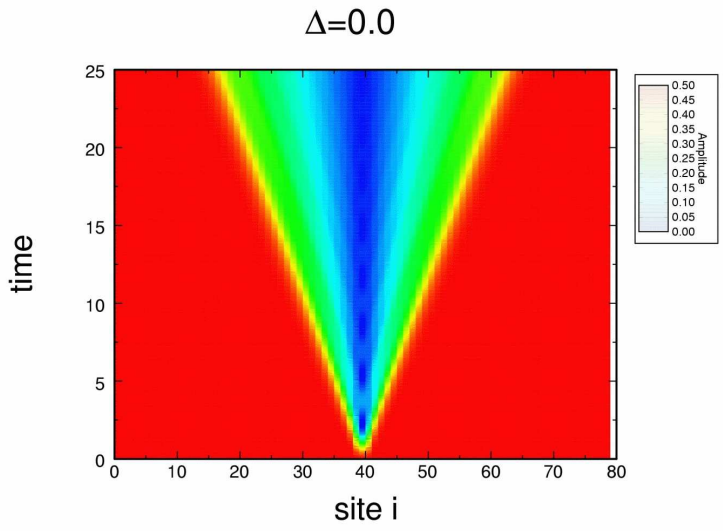


FIGURE 32. Color density plot showing the absolute value of the spin polarization in a space-time representation, for a system that was initially fully polarized in opposite directions on left and right halves of the chain.

An alternative approach to spin transport can be to initialize the system in a state that has all the spins on the left of the system polarized $\uparrow$, and the spins on the right, $\downarrow$. This can be achieved by using the ground state DMRG with an artificial Hamiltonian

$$\hat{H}' = \sum_{i=0}^{L/2-1} \hat{S}_i^z - \sum_{i=L/2}^{L-1} \hat{S}_i^z.$$

This creates a domain wall right at the center. We can the evolve the system under our original Heisenberg Hamiltonian, for instance, and look at the evolution of the spin. We will see wave fronts evolving in both directions, describing a "light-cone" in the $x-t$ plane, as shown in Fig.32. By measuring the slope of the boundaries, we can obtain the spin velocity. Another possibility is to create a Gaussian wave-packet with a given momentum and follow its evolution.[82, 94]

Time-dependent correlation functions

The knowledge of the excitation spectrum of a system allows for direct comparison with experiments, such as photoemission, or neutron scattering, for instance. The numerical evaluation of dynamical correlation functions is a very difficult task, since the methods are usually capable of calculating the ground-state, and maybe some low energy excitations. A number of techniques has been used in the past, however they all have very important limitations: exact diagonalization[97] is limited to small clusters, quantum Monte Carlo[98] suffers from the sign problem, and requires uncontrolled analytic continuations and the use of the max entropy approximation, and dynamical DMRG[99, 100, 101] is computationally very expensive.

For a decade or more, these limitations have in part prevented much progress in this area. However, our time-dependent DMRG method has the potential to tackle many open problems, due to its versatility and for being computationally less intensive. The time-dependent DMRG method can calculate time-dependent correlation functions with unprecedented accuracy,[102] providing a very good description of the entire spectrum of excitations with a single run, and with modest computational resources. This technique has been applied to a number of problems, such as the study of the spectral properties of a partially spin-polarized one-dimensional Hubbard chain [103], spin systems [105] and also in combination with imaginary time evolution to obtain the spectral properties of spin-incoherent Luttinger liquids [104], and spin systems [106]. In Fig.33 we show the momentum resolved spectrum of a $t-J$ chain to illustrate the remarkable capabilities of this algorithm, displaying all the distinctive features –spinon, and holon bands, shadow bands and continuum– with impressive resolution.

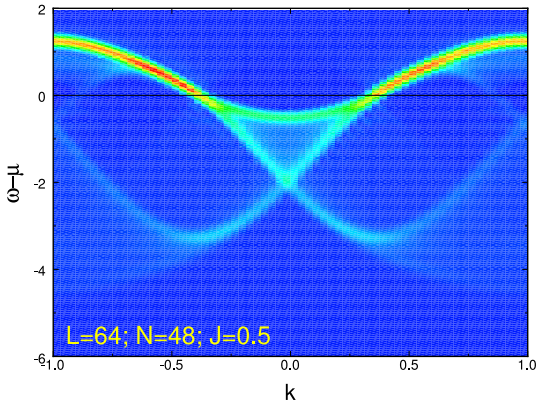


FIGURE 33. Momentum resolved spectrum of a $t-J$ chain of $L=64$ sites, with $J/t=0.5$ and density $n=0.75$, calculated with time-dependent DMRG, showing extremely well resolved spinon and holon bands, as well as shadow bands and continuum. Color density is proportional to the spectral weight, in linear scale.

Imagine that we want to calculate a time-dependent correlation function that looks like

$$G(t) = \langle \psi | \hat{B}(t) \hat{A}(0) | \psi \rangle$$

where $|\psi\rangle$ is the ground state of the Hamiltonian $\hat{H}$. To obtain $G(t)$ we follow the following steps:

- Using the conventional ground-state DMRG calculate $|\psi\rangle$. Turn off the Lanczos/Davidson diagonalization
- During a half sweep, apply $\hat{A}$ to $|\psi\rangle$ to obtain $|\phi\rangle = \hat{A}|\psi\rangle$, targeting both $|\psi\rangle$, and $|\phi\rangle$.
- Start the time-evolution sweeps, evolving and targeting both states.
- Measure $G(t)$ as $\langle \psi(t) | \hat{B} | \phi(t) \rangle$.

In the case presented here, since we have assumed that $|\psi\rangle$ is the ground state, we can replace the time-evolution on this state by a multiplication by a phase $exp(-itE_0)$.

If we are interested in obtaining the full momentum-resolved spectrum, we would want to calculate a correlator

$$G(x-x',t'-t) = i\langle \hat{O}(x',t')\hat{O}^{\dagger}(x,i)\rangle,$$

where O is an operator of interest. In practice we apply the operator $\hat{A} = \hat{O}^{\dagger}(x = L/2)$ at the center site at time $t = 0$ to obtain $|\phi\rangle$, and we apply $\hat{B} = \hat{O}(x')$ on all the sites x' at every time step. This will yield the correlations in real space and time. Fourier transforming of these functions to momentum and frequency, will then yields the corresponding spectral weights as functions of momentum and frequency [34, 107]:

$$I(k,\omega) = \sum_n |\langle \psi_n|\hat{O}_k|\psi_0\rangle|^2 \delta(\omega - E_n + E_0), \tag{115}$$

where E_0 is the ground state energy, and the sum runs over all the eigenstates of the system, with energy E_n.

We typically study chains with an even number of sites. In order to recover the reflection symmetry, we may use $G'(x) = 1/2(G(L/2-x)+G(L/2+x))$ instead of $G(x)$. In order to study small frequencies, or long time scales, one can perform an extrapolation in time using linear prediction methods, asumming that the form of the correlations can be deduced from some physics insight [107].

We should point out that the Fourier transform, as well as the linear momentum, are ill-defined in a system with open-boundary conditions. The scheme described here will work if the system is large enough such that we can ignore the effects of the boundaries, and we can assume that the system is translationally invariant near the center. We can also do it if we measure in short enough time ranges . No matter what we do, it is likely that the edges may introduce some perturbation in the form of Friedel oscillations. If the problem has some instability toward some sort of commensurate order, dimerization, spin or charge ordering, then the boundaries may lock the order and our artificial picture of translational invariance will be destroyed, making the application of the method nonviable.

The enemy: Entanglement growth

We have seen that the truncation error, or the number of states that we need to keep to control it, depends fundamentally on the entanglement entropy

$$S = S(t)$$

We need to understand its behavior if we want to learn how to fight it. In broad terms, the possible scenarios that we can find are:

- Global quench
- Local quench
- Periodic quench
- Adiabatic quench

When we apply such a perturbation to our system it no longer is in the ground state, but some superposition of excited states. Then, important questions arise, such as the relationship between thermalization and the properties of the system. Without getting into that, we are going to try to characterize the basic behavior of the entanglement growth for each case. Although some specific

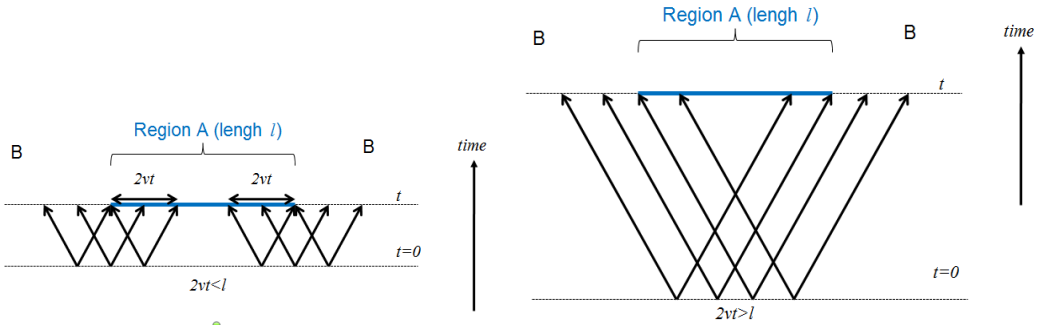


FIGURE 34. Illustration of the space-time evolution of the entanglement between two regions A and B, due to entangled quasi-particle pairs moving with velocity v. We assume that particles move along the light-cones, and we ignore scattering effects.

demonstrations exist for particular cases, this behavior is believed to be very general and apply to a broad family of problems. Here, we shall use an intuitive picture to illustrate the main concepts without attempting formal proofs.

Global quench: qualitative picture

We are going to follow the reasoning described in Ref.[108]. Let us assume that at $t = 0$ a perturbation is added to the Hamiltonian. This perturbation can be a change in the parameters (say J_z in the Heisenberg model), or an external magnetic field. This perturbation affects all the sites in the lattice, hence the "global" terminology. We are going to imagine that the perturbation acts as a source of quasi-particle excitations: we can picture quasi-particle pairs being emitted from all the sites on the lattice, which will be highly entangled (see Fig.34). This excitations will propagate with a velocity $v = dE/dk$. Ignoring scattering effects, a particle produced at position x, will be at $x + vt$ at time t. The wave-function at position x in the partition A in Fig.34, will be entangled with that at position x' in partition B, if there is an entangled pair connecting both sites. The region covered by the entangled pairs will be of size $2vt$, and if we assume that the entanglement is proportional to the number of entangled pairs inside and outside of region A, then we obtain that the entropy grows linearly in time

$$S = S_0 + ct$$

where c is some constant. When $2vt = l$, then the entanglement entropy saturates, and does no longer grow in time, rachign a constant value (see Fig.34)

Local quench: qualitative picture

To illustrate the case of a local quench, we are going to assume that the perturbation is introduced in the center of a one-dimensional chain. Following the previous considerations[109, 110], a quasi-particle paper will be created, entangling left and right halves of the chain, in a region of size vt on each side (see Fig.35. In a critical system, we know that the entanglement is proportional to the log of the size of the entangled region:

$$S = S_0 + c \log(vt)$$

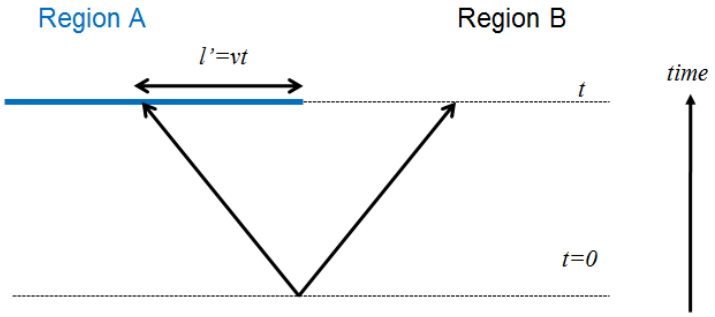


FIGURE 35. Illustration of the space-time evolution of the entanglement between left and right halves of a chain, due to entangled quasi-particle pairs created at the center of the system, moving with velocity *v* describing a light-cone.

This corresponds to the worse case scenario. In a gaped system, the entropy will grow until the pair reaches a size comparable to the correlation length, and saturate at a constant value.

Computational cost

After the above considerations, we find that the number of states needed for an efficient simulation grows as

- **Sudden global quench:** $m \sim exp(t)$.
- **Sudden local quench:** $m \sim t^{\alpha}$.
- **Adiabatic quench:** $m \sim const.$, since we always remain close to the ground state.

It is clear that a global quench corresponds to the worse possible scenario. Since most of the experiments involve global quenches, most of the research in the field has focused on this case, which is considerably more challenging. However, the calculation of correlations functions has the computational cost of a local quench, making the simulations extremelly efficient with moderate computational resources. Periodic quenches arise when the system is driven under a periodic perturbations, and the behavior of the entanglement growth depends on the problem, and the frequency and profile of the perturbation, since resonances may appear (see for instance Refs.[111, 112]).

ALPS DMRG

ALPS: Algorithms and Libraries for Physics Simulations

The last decade has seen tremendous progress in the development of computational algorithms. These advances come at the cost of substantially increased algorithmic complexity and challenge the current model of program development in condensed matter. In contrast to other research areas, in which large “community codes” are being used, the field of strongly correlated systems has so far been based mostly on single codes developed by individual researchers for particular projects. While the simple algorithms used a decade ago could be easily programmed by a beginning graduate student in a matter of weeks, it now takes substantially longer to master and implement the new algorithms. Thus, their use has increasingly become restricted to a small number of experts.

The ALPS project [9] aims to overcome the problems posed by the growing complexity of algorithms and the specialization of researchers onto single algorithms through an open-source software development initiative. Its goals are to provide:

• **standardized file formats** to simplify exchange, distribution and archiving of simulation results and to achieve interoperability between codes.
• **generic and optimized libraries** for common aspects of simulations of quantum and classical lattice models, to simplify code development.
• a set of **applications** covering the major algorithms.
• **license** conditions that encourage researchers to contribute to the ALPS project by gaining scientific credit for use of their work.
• **outreach** through papers [9], a web page (`http://alps.comp-phys.org/`) mailing lists and workshops to distribute the results and to educate researchers both about the algorithms and the use of the applications.
• **improved reproducibility** of numerical results by publishing source codes used to obtain published results.
• an **archive** to store simulation results.

The ready-to-use applications are useful for the non-experts, *theoreticians* who want to test theoretical ideas about quantum lattice models and to explore their properties, as well as for *experimentalists* trying to fit experimental data to theoretical models to obtain information about the microscopic properties of materials. The ALPS web page (`http://alps.comp-phys.org/`) is the central information repository of the ALPS collaboration. It makes source codes of the ALPS libraries and applications available, provides access to the documentation, and distributes information about the mailing lists, new releases, and workshops. If the reader decides to apply ALPS to a new project, it is highly recommendable that he/she subscribes to the mailing lists, where both users and developers can help answering any questions that may arise.

The ALPS webpage also provides with extensive and detailed tutorials, for different applications and problems, that can be of great help at the time of starting a new project. The tutorials also show how to run an application from the command line, or using python or VisTrails. In these lecture, we shall focus our attention on understanding how to define a problem with a simple example.

VISTRAILS

VisTrails (`http://www.vistrails.org/`) is a new scientific workflow and provenance management system developed at the University of Utah that provides support for data exploration and visualization. Workflows can capture complex analysis processes at various levels of detail and provide the provenance information necessary for reproducibility, result publication and result sharing among collaborators. VisTrails allows one to define a workflow, or script to analyze data or run an experiment trying different parameter values. The application maintains a detailed log or history of this process that allow to reproduce it step by step. This not only serves to enable and enhance reproducibility of results, but also to establish a clear provenance, or documentation that determines quality, and authorship of the product. At the same time, VisTrails could function as a graphic user interface for running ALPS scripts, and data analysis.

Defining a problem in ALPS

To define a problem we need to specify:

- The lattice, or geometry of the problem.
- The degrees of freedom, in terms of a local basis, and operators acting on the basis elements.
- The model, or Hamiltonian, in terms of the operators defined in the previous step.
- Measurements.
- The parameters for the simulations.

Once this has been defined using the ALPS interface, a standardized input is generated, and one can in principle solve the problem using any of the applications provided by ALPS, from exact diagonalization, to DMRG and Quantum Monte Carlo. Moreover, ALPS also provides analysis and plotting tools, and a graphic interface through VisTrails that makes this process very transparent and intuitive. Here we briefly address how to declare each of these three elements.

Many standard lattices and Hamiltonians are predefined in the `lattices.xml` and `models.xml` files provided by the ALPS package. In order to illustrate the DMRG application we shall consider the example of a frustrated spin chain with spin $S = 1/2$, and first and second neighbor interactions J_1 and J_2. The Hamiltonian reads:

$$\hat{H} = J_1 \sum_{i=1}^{N-1} \hat{\mathbf{S}}_i \cdot \hat{\mathbf{S}}_{i+1} + J_2 \sum_{i=1}^{N-2} \hat{\mathbf{S}}_i \cdot \hat{\mathbf{S}}_{i+2} \tag{116}$$

which is a trivial extension of Eq.(1).

The lattice

The lattice is a simple one dimensional chain. Internally, ALPS defined a lattice as a graph with vertices, and edges. In this particular example, the unit cell will contain a single site, but in order to define the interaction we need to specify two vertices connecting first neighbors and second neighbors:

```
<UNITCELL name="complex1d" dimension="1">
  <VERTEX/>
  <EDGE type="0"><SOURCE vertex="1" offset="0"/>
    <TARGET vertex="1" offset="1"/>
  </EDGE>
  <EDGE type="1"><SOURCE vertex="1" offset="0"/>
    <TARGET vertex="1" offset="2"/>
   </EDGE>
</UNITCELL>
```

We have named the unitcell as `''complex1d''`, of dimension 1, since it contains a single site. The edge of type “0” connects first neighbors separated by an offset of 1, while the edge of type “1” connect second neighbors, with offset 2.

Once defined the unit cell, we need to define a graph. This is done by superimposing the previous unit cell on top of a one dimensional finite lattice of length L:

```
<LATTICE name="chain lattice" dimension="1">
  <PARAMETER name="a" default="1"/>
```

```
  <BASIS>
    <VECTOR>a</VECTOR>
  </BASIS>
  <RECIPROCALBASIS>
    <VECTOR>2*pi/a</VECTOR>
  </RECIPROCALBASIS>
</LATTICE>

<LATTICEGRAPH name = "nnn open chain lattice">
  <FINITELATTICE>
    <LATTICE ref="chain lattice"/>
    <EXTENT dimension="1" size ="L"/>
    <BOUNDARY type="open"/>
  </FINITELATTICE>
  <UNITCELL ref="complex1d"/>
</LATTICEGRAPH>
```

We have chosen open boundary conditions for the DMRG simulation, since we have learned that this improves convergence. For more ellaborate examples, we suggest the reader to look at the ALPS documentation.

The basis

The basis for a single site consists in general of an enumeration of states with given quantum numbers. In our case, the basis will have only two states with $S^z = \pm 1/2$, and could be defined as:

```
<SITEBASIS name="spin-1/2">
  <QUANTUMNUMBER name="S" min="1/2" max="1/2"/>
  <QUANTUMNUMBER name="Sz" min="-1/2" max="1/2"/>
</SITEBASIS>
```

The <QUANTUMNUMBER> elements each take a name, and minimum and maximum values in the min and max attributes. Optionally, a type attribute can be set to bosonic (the default) or fermionic. It should be set to fermionic when the quantum number is a fermionic number operator. This information will be used when determining commutation relation between operators on different sites.

The operators $\hat{S}^z$,$\hat{S}^+$ and $\hat{S}^-$ acting on this basis will be represented by 2×2 matrices. These operators will connect subspaces with well defined quantum numbers: $\hat{S}^z$ is diagonal since it does not change the quantum numbers. But $\hat{S}^+$ and $\hat{S}^-$ will connect states differing in S^z by 1 and -1 respectively. The corresponding matrices will have a non-zero matrix element only if the row and column correspond to states connected by the quantum number rules. The ALPS implementation allows one to define these matrices symbolically for arbitrary values of the spin S:

```
<SITEBASIS name="spin">
  <PARAMETER name="local_spin" default="local_S"/>
  <PARAMETER name="local_S" default="1/2"/>
  <QUANTUMNUMBER name="S" min="local_spin" max="local_spin"/>
  <QUANTUMNUMBER name="Sz" min="-S" max="S"/>
  <OPERATOR name="Splus" matrixelement="sqrt(S*(S+1)-Sz*(Sz+1))">
    <CHANGE quantumnumber="Sz" change="1"/>
  </OPERATOR>
  <OPERATOR name="Sminus" matrixelement="sqrt(S*(S+1)-Sz*(Sz-1))">
```

```
    <CHANGE quantumnumber="Sz" change="-1"/>
  </OPERATOR>
  <OPERATOR name="Sz" matrixelement="Sz"/>
</SITEBASIS>
```

Notice that the value of the spin S is specified by a parameter ‘‘local_S’’ which assumes a defeault value of $1/2$.

The basis for the entire system is defined automatically by specifying the local basis for each site type. It can allow for constraints on the allowed values for the quantum numbers. For instance, if the total value of S^z can be conserved quantity, we can specify it with the following statements:

```
<BASIS name="spin">
   <SITEBASIS ref="spin"/>
   <CONSTRAINT quantumnumber="Sz" value="Sz_total"/>
 </BASIS>
```

Here, the ref attribute declares the site basis of type “spin”, as defined above.

As mentioned before, in order to define fermionic operators it is necessary to specify that they obey fermionic statistics, and the ALPS libraries will automatically account for the phases in their commutation rules. For the case of models with multiple sites per unit cell, such as the Kondo lattice, we may need to define a basis for each site type. Again, we refer the reader to the `models.xml` file and the ALPS documentation where these and more cases are discussed in more detail.

The model

It is time to define the interaction between spins sitting at different sites on the lattice graph. We first define the exchange bond operator acting on two spins on sites “x” and “y” as:

```
<BONDOPERATOR name="exchange_xy" source="x" target="y">
  1/2*(Splus(x)*Sminus(y)+Sminus(x)*Splus(y))
</BONDOPERATOR>
```

The Hamiltonian can now be expressed in terms of this bond operator as:

```
<HAMILTONIAN name="spin">
  <PARAMETER name="J0" default="0"/>
  <PARAMETER name="J" default="J0"/>
  <PARAMETER name="Jz" default="J"/>
  <PARAMETER name="Jxy" default="J"/>
  <PARAMETER name="Jz0" default="Jz"/>
  <PARAMETER name="Jxy0" default="Jxy"/>
  <PARAMETER name="J1" default="0"/>
  <PARAMETER name="Jz1" default="J1"/>
  <PARAMETER name="Jxy1" default="J1"/>
  <BASIS ref="spin"/>
  <BONDTERM source="i" target="j">
    <PARAMETER name="J#" default="0"/>
    <PARAMETER name="Jz#" default="J#"/>
    <PARAMETER name="Jxy#" default="J#"/>
    Jz#*Sz(i)*Sz(j)+Jxy#*exchange_xy(i,j)
  </BONDTERM>
</HAMILTONIAN>
```

The <BASIS> element specifes the basis used for the model, either fully specified inline or by a reference (using the ref attribute).

The <BONDTERM> element includes a <BONDOPERATOR> interaction involving two sites on an edge. It is also possible to define site term with local operators acting on a single site. This Hamiltonian structure also defines the parameters of the model followed by a “0” or a “1” for the corresponding edges that will replace the “#” wildcard character in the interactions, that can assume certain default values. With this, the definition of the problem is complete, and we only need to specify the quantities to measure, and the application-dependent parameters for the simulation.

Parameters

In order to define the quantities to measure and the parameters for the simulation, one needs to create a parameter file with some required attributes. These incllude the lattice type, the model, and conserved quantum numbers, and attributes of the model. For instance, for the $J_1 - J_2$ Heisenberg chain an example file would look:

```
LATTICE="nnn open chain lattice"
L=32
MODEL="spin"
CONSERVED_QUANTUMNUMBERS="Sz"
Sz_total=0
J=1
J1=0.2
```

Notice that J1 actually corresponds to J_2 in our model. One could specify multiple runs as:

```
LATTICE="nnn open chain lattice"
L=32
MODEL="spin"
CONSERVED_QUANTUMNUMBERS="Sz"
Sz_total=0
J=1
{ J1=0.1 }
{ J1=0.2 }
{ J1=0.3 }
{ J1=0.4 }
```

Measurements

Some ALPS applications provide a set of pre-defined measurements that can be performed. In case that the quantities you want to measure are not included, you need to define your custom measurements in the parameter file. The general syntax for local measurements is as follows:

```
MEASURE_LOCAL[Name]=Op
MEASURE_AVERAGE[Name]=Op
```

Here “Name” is the name under which your measurements appear in the xml output, and “Op” is the measurement operator, which must be defined in the `models.xml` file. MEASURE_AVERAGE

measures the quantum mechanical and (for finite temperature simulations) thermodynamic expectation value of the operator Op. MEASURE_LOCAL measures the expectation values of the operator Op for each site of the lattice. The operator must be local, *i.e.*, it can only have site terms.

In order to measure correlations one needs to specify:

```
MEASURE_CORRELATIONS[Name]="Op1:Op2"
MEASURE_CORRELATIONS[Name]=Op
```

MEASURE_CORRELATIONS measures the correlations of the operators Op1 and Op2 for all inequivalent pairs of sites of the lattice. The second form above, MEASURE_CORRELATIONS[Name]=Op is equivalent to MEASURE_CORRELATIONS[Name]="Op:Op". At present, only two-site correlation functions can be computed. That is, both Op1 and Op2 must be site operators.

In the provided DMRG examples the user will find, for instance:

```
MEASURE_AVERAGE[Magnetization]=Sz
MEASURE_AVERAGE[Exchange]=exchange
MEASURE_LOCAL[Local magnetization]=Sz
MEASURE_CORRELATIONS[Diagonal spin correlations]=Sz
MEASURE_CORRELATIONS[Offdiagonal spin correlations]="Splus:Sminus"
```

The ALPS DMRG module

Until now, most of the parameters described are common to all the ALPS applications. In order to run a DMRG simulation, we need to declare some DMRG-specific parameters:

NUMBER_EIGENVALUES
Number of eigenstates and energies to calculate. The default value is 1, and should be set to 2 to calculate gaps.

SWEEPS
Number of DMRG sweeps. Each sweep involves a left-to-right half-sweep, and a right-to-left half-sweep.

NUM_WARMUP_STATES
Number of initial states to grow the DMRG blocks. If not specified, the algorithm will use default value used of 20.

STATES
Number of DMRG states kept on each half sweep. The user should specify either 2×SWEEPS different values of STATES or one MAXSTATES or NUMSTATES value.

MAXSTATES
Maximum number of DMRG states kept. The user should choose to specify either STATES values for each half-sweep, or a MAXSTATES or NUMSTATES that the program will use to grow the basis. The program will automatically determine how many states to use for each sweep, growing in steps of STATES/(2× SWEEPS) until reaching MAXSTATES.

NUMSTATES
Constant number of DMRG states kept for all sweeps.

TRUNCATION_ERROR
The user can choose to set the tolerance for the simulation, instead of the number of states. The program will automagically determine how many states to keep in order to satisfy this tolerance. Care must be taken, since this could lead to an uncontrollable growth in the basis size, and a crash as a consequence. It is therefore advisable to also specify the maximum number of states as a constraint, using either `MAXSTATES` or `NUMSTATES`, as explained before.

LANCZOS_TOLERANCE
Tolerance for the exact diagonalization piece of the simulation. The default value is 10^-7.

CONSERVED_QUANTUMNUMBERS
Quantum numbers conserved by the model of interest. They will be used in the code in order to reduce matrices in block form. If no value is specified for a particular quantum number, the program will work in the grand canonical. For instance, in spin chains, if you do not specify `Sz_total`, the program will run using a Hilbert space with $dim = 2^N$ states. Running in the "canonical" ensemble (by setting `Sz_total=0`, for instance) will improve performance considerably by working in a subspace with a reduced dimension. For an example of how to do this, take a look at the parameter file included with the dmrg code.

VERBOSE
If set to an integer > 0, it will print extra output information, such as density-matrix eigenvalues. There are different verbose levels up to a maximum of 3, for debugging purposes, although the user shouldn't need a level larger than 1.

START_SWEEP
Starting sweep for resuming a simulation that was interrupted, or extending it with a new set of states.

START_DIR
Starting direction for the resumed simulation. Can assume the values 0 or 1, for "left-to-right" or "right-to-left", respectively. It has effect only in the presence of `START_SWEEP`. Its default value is 0.

START_ITER
Starting iteration for the resumed simulation. It has effect only in the presence of `START_SWEEP`. Its default value is 1.

TEMP_DIRECTORY
The DMRG program stores information in temporary files that reside in your local folder. There could be a large number of them, and this could overwhelm your file system, especially if you are using NFS. The address for storing these files could be changed by setting the variable `TEMP_DIRECTORY` in the parameter file. Another way to do this is by setting the system environment variable `TMPDIR`.

Running the simulation

We are now ready to write a parameter file for out simulation. Several examples are included in the distribution, acompanied by a tutorial that explains them in great detail. For ilustration purposes, we include a simple for the frustrated spin chain:

```
LATTICE="nnn open chain lattice"
L=32
MODEL="spin"
CONSERVED_QUANTUMNUMBERS="Sz"
Sz_total=0
NUMBER_EIGENVALUES=1
MEASURE_AVERAGE[Magnetization]=Sz
MEASURE_AVERAGE[Exchange]=exchange
MEASURE_LOCAL[Local magnetization]=Sz
MEASURE_CORRELATIONS[Diagonal spin correlations]=Sz
MEASURE_CORRELATIONS[Offdiagonal spin correlations]="Splus:Sminus"
MAXSTATES=40
SWEEPS=6
J=1
{ J1=0.1 }
{ J1=0.2 }
{ J1=0.3 }
{ J1=0.4 }
```

Asumming that the file has been called “parm”, we can use the following sequence of commands to first convert the input parameter file to XML format before running the DMRG application:

```
$ parameter2xml parm
$ dmrg parm.in.xml
```

(where “$” is the shell prompt)

DMRG will perform four sweeps, (four half-sweps from left to right and four half-sweeps from right to left) growing the basis in steps of `MAXSTATES/(2×SWEEPS)` until reaching the `MAXSTATES=40` value we have declared. This is a convenient default option, but the number of states can be customized. The results will correspond to the ground-state of the system in the sector with `Sz_total = 0`.

The outputfiles will contain all the computed quantities in XML format and can be viewed with a standard internet browser. ALPS also provides with a collection of tools that allow one to analize and plot the results, as well as to convert the files to different standard formats. It is also possible to define the runs from Python, which allow the user to write more ellaborate scripts.

OUTLOOK

With the advent of matrix product state algorithms, and quantum information ideas, we have seen a remarkable progress in the computational field. A repeated question that arises is wether the variational approaches using MPS are more efficient, or “better” that conventional DMRG. The short answer is “not necessarily”. If one is interested in studying one-dimensional problems, all methods are basically equivalent. One may argue that MPS algorithms work better for periodic boundary conditions, but the drawback is that the implementation of the MPS code suffers from normalization

problems that may lead to some instabilities. It is possible to reformulate the DMRG as an MPS optimization code[42], that may lead to a hybrid solution. What is certain is that most of the people that have been working with the DMRG for decades, have very polished, optmized, state-of-the-art codes that are hard to beat with existent MPS codes. However, progress is rapidly being made, and ultimately, it may become a matter of taste. DMRG may be easier to generalize to arbitrary Hamiltonians, such as the ALPS DMRG code [9]. On the other hand, MPS enjoy several advantages that in the end may make them the favorite choice: They are probably easier to understand intuitively; one only has to store matrices that are easy to deal with algebraically; they are easy to extend to the thermodynamic limit in translational invariant problems[113, 114, 115, 63]; overlaps between MPS's are easy to calculate; and most importantly, the MPS structure makes them more suitable for massive parallellization, expecially for time-evolution[116, 117, 118].

MPS's are extremely powerful objects, and we have been learning a big deal about entanglement[32], complexity[33, 119, 120], and structure of the quantum world through them [121]. The concept can readily be extended to higher dimensions, with matrices being replaced by tensors, leading to complex structures: tensor networks. Several ideas exploiting tensor networks have been proposed,such as PEPS (projected entangled pair states)[63] and MERA (multi-scale entanglement renormalization ansatz)[122, 123, 124], with very promissing, but not yet groundbreaking, results. However, until recent work[125], DMRG has always been more efficient at studying 2D problems[126]. The main reason for this is again, the lack of highly optimized code to deal with the tensor contractions in PEPS, which is a complex computational problem[127, 128, 129], and in the case of MERA, with the explicit breaking of the translational symmetry. But again, progress is rapidly being made, and we can anticipate a new wave of computational methods based on tensor network methods.

ACKNOWLEDGMENTS

I am grateful Prof A. Avella and Prof. F. Mancini for kindly inviting me to lecture at the Vietri Summer School and to write these notes that hopefully will find an audience in many young students interested in learning the DMRG method and its time-dependent extensions. I enormously enjoyed the experience. I am endebted to my collaborators in the ALPS project, especially Matthias Troyer and Ulrich Schollwöck for a continued and intense exchange of ideas, sharing some pedagogical material, and for comments and suggestions during the preparation of these notes. Contrinbuting my DMRG code to the ALPS project has been enriching and rewarding in many aspects, especially knowing that such a large community has benefitted from it. I would like to thank Steve White for encouragement and support in this and many other decisions. I would also like to thank Carlos Büsser, Mohammed Soltanieh-ha, Alberto Nocera, and Ryan Hurtt for a critical reading of the manuscript, and for asking questions that helped making the manuscript more pedagogical.

APPENDICES

In the appendices I will list several code snippets illustrating the main concepts introduced in these lectures. The codes make use of several C++ classes defined within the `dmtk` package, including `Matrix`, and `Vector`. Hopefully, the syntax will be transparent to the reader. We shall only point out, to avoid confusion, that the matrix elements are accessed as `(column,row)` as in Fortran. The class also allows to use C++ notation, `[row][column]`.

Installing the packages

The packages can be freely downloaded from `http://physics.uwyo.edu/~adrian/programs/`. The `dmtk` libraries that include all the object definitions are in the tarball `dmtk-basis.tar.gz`, while `tutorials.tar.gz` contains the complete codes described in this appendices. To install in a `Unix` environment one needs to downloads the files, and unpack them using the commands:

```
$ tar zxvf dmtk-basis.tar.gz
$ tar zxvf tutorials.tar.gz
```

It is recommendable that the *dmtk/* directory is moved to a standard place, such as `/usr/local/include` or `/home/myhome/include`, where `myhome` is typically your home directory. Once done this, the tutorials can be compiled from their corresponding directories by typing, for instance:

```
$ cd tutorials/00-heischain_ed
$ make
```

In order for this to work, you need modify the `Makefile` file, to replace the path to your `dmtk` libraries by the correct one.

Listing 1: Exact diagonalization of a Heisenberg spin chain

This code constructs explicitly a $2^N \times 2^N$ Hamiltonian matrix by adding the interactions between nearest neighbors one by one. These interactions are constructed at the beginning through the tensor product of the corresponding operator matrices, and then shifted to the corresponding site along the chain by using apropriate identity matrices `Ileft` and `Iright`.

```
#include <iostream>
#include <dmtk/dmtk.h>

using namespace std;
using namespace dmtk;

#define POW2(i) (1 << i)


int main()
{
    // PARAMETERS
    int nsites;
    cout << "Number of sites: ";
    cin  >> nsites;

    // Single site operators
    Matrix<double> sz0(2,2); // single site Sz
    Matrix<double> splus0(2,2); // single site S+
    sz0(0,0)         = -0.5;
```

```
    sz0(1,1)         =  0.5;
    splus0(1,0)      =  1.0;

    Matrix<double> term_szsz(4,4); //auxiliary matrix to store Sz.
       Sz
    term_szsz = tensor(sz0,sz0);

    Matrix<double> term_spsm(4,4); //auxiliary matrix to store 1/2
       S+.S-
    term_spsm = tensor(splus0,splus0.t())*0.5;

    // Hamiltonian
    int maxdim = POW2(nsites); // 2^N states
    cout << "MAXDIM = " << maxdim << endl;
    Matrix<double> H(maxdim,maxdim); // Hamiltonian matrix

    for(int i = 1; i < nsites; i++){;
       // We add the term for the interaction S_i.S_{i+1}

       int diml = POW2(i-1);        // 2^(i-1)
       int dimr = POW2(nsites-i-1); // 2^(nsites-i-1)

       cout << "SITE " << i << " DIML= " << diml << " DIMR= " <<
          dimr << endl;

       Matrix<double> Ileft(diml,diml),Iright(dimr,dimr);
       Ileft = eye(diml);
       Iright = eye(dimr);

       Matrix<double> aux(2*diml,2*diml); // auxiliary matrix to
          store the term
       // Sz.Sz
       aux = tensor(Ileft,term_szsz);
       H = H + tensor(aux,Iright);

       // (1/2)(Sp.Sm + Sm.Sp)
       aux = tensor(Ileft,term_spsm);
       H = H + tensor(aux,Iright);
       aux = tensor(Ileft,term_spsm.t());
       H = H + tensor(aux,Iright);
    }

    GroundState(H); // Diagonalize the matrix
    return 0;
}
```

Listing 2: Adding sites to a block

This code constructs explicitly a $2^N \times 2^N$ Hamiltonian matrix starting from a single site block, and adding new sites to the right at every step. The terms are constructed exactly as in the previous example, but the iteration procedure to contruct the Hamiltonian resembles the idea behing the Numerical Renormalization Group.

```
int main()
{
    // PARAMETERS
    int nsites;
    cout << "Number of sites: ";
    cin  >> nsites;

    // Single site operators
    Matrix<double> sz0(2,2); // single site Sz
    Matrix<double> splus0(2,2); // single site S+
    sz0(0,0)         = -0.5;
    sz0(1,1)         =  0.5;
    splus0(1,0)      =  1.0;

    Matrix<double> term_szsz(4,4); //auxiliary matrix to store Sz.
       Sz
    term_szsz = tensor(sz0,sz0);

    Matrix<double> term_spsm(4,4); //auxiliary matrix to store 1/2
       S+.S-
    term_spsm = tensor(splus0,splus0.t())*0.5;

    // Hamiltonian
    int maxdim = POW2(nsites);
    cout << "MAXDIM = " << maxdim << endl;

    Matrix<double> H(2,2); // Hamiltonian matrix
    H = 0.;
    for(int i = 1; i < nsites; i++){;
       int diml = POW2(i);
       int dim = diml*2;
       cout << "ADDING SITE " << i << " DIML= " << diml << " DIM= "
           << diml*2 << endl;

       Matrix<double> Ileft(diml,diml);
       Ileft = eye(diml);

       Matrix<double> aux(dim,dim);
       aux = tensor(H,eye(2));
       H = aux;
      // Sz.Sz
       H = H + tensor(Ileft,term_szsz);
```

```
        // (1/2)(Sp.Sm + Sm.Sp)
        H = H + tensor(Ileft,term_spsm);
        H = H + tensor(Ileft,term_spsm.t());
        GroundState(H); //Diagonalize the matrix
    }

    return 0;
}
```

Listing 3: A DMRG code for the Heisenberg chain

In the following listing we introduce a C++ object `DMRGSystem` which contains container object `Vector` with a list of matrices corresponding to the dmrg blocks for the sweeps. The wave-function will also be represented by a matrix, which makes the implementation more efficient and convenient, since all operations can be reduced to matrix-matrix and matrix-vector multiplications:

```
class DMRGSystem
{
    public:
        Matrix<double> sz0; // single site Sz
        Matrix<double> splus0; // single site S+
        Vector<Matrix<double> > HL; // left block Hamiltonian
        Vector<Matrix<double> > HR; // right block Hamiltonian
        Vector<Matrix<double> > szL; // left block Sz
        Vector<Matrix<double> > szR; // right block Sz
        Vector<Matrix<double> > splusL; // left block S+
        Vector<Matrix<double> > splusR; // right block S+

        Matrix<double> psi;     // g.s. wave function
        Matrix<double> rho;     // density matrix
        double energy;
        double error;
        int nsites;
        int right_size;
        int left_size;

        DMRGSystem(int _nsites)
          {
            sz0.resize(2,2);
            splus0.resize(2,2);
            nsites = _nsites;
            HL.resize(nsites);
            HR.resize(nsites);
            szL.resize(nsites);
            splusL.resize(nsites);
```

```
            szR.resize(nsites);
            splusR.resize(nsites);

            // initialize Hamiltonian:
            sz0 = 0.0;
            splus0 = 0.0;
            HL[0] = sz0;
            HR[0] = sz0;
           // single-site operators
            sz0(0,0)         = -0.5;
            sz0(1,1)         =  0.5;
            splus0(1,0)      =  1.0;
            szR[0] = sz0;
            szL[0] = sz0;
            splusR[0] = splus0;
            splusL[0] = splus0;
          }

        // Methods
        void BuildBlockLeft(int _iter);
        void BuildBlockRight(int _iter);
        void GroundState();
        void DensityMatrix(int _position);
        void Truncate(int _position, int _m);
};
```

We shall proceed to describe each of the class methods one by one in some detail. `BuildBlockLeft` and `BuildBlockRight` precisely construct new blocks by adding a single site. The new block will be identified by the integers `left_size` and `right_size`, and will contain a Hamiltonian for the block containing all the terms iinvolving sites inside the block, as well as the matrices necessary to construct the interaction between the block and the external site:

```
void
DMRGSystem::BuildBlockLeft(int _iter)
{
    left_size = _iter;
    int dim_l = HL[left_size-1].cols();
    Matrix<double> I_left = eye(dim_l);
    Matrix<double> I2 = eye(2);
    //enlarge left block:
    HL[left_size]  = tensor(HL[left_size-1],I2) +
                     tensor(szL[left_size-1],sz0) +
                     0.5*tensor(splusL[left_size-1],splus0.t()) +
                     0.5*tensor(splusL[left_size-1].t(),splus0);
    splusL[left_size] = tensor(I_left,splus0);
    szL[left_size] = tensor(I_left,sz0);

}
```

```
void
DMRGSystem::BuildBlockRight(int _iter)
{
    right_size = _iter;
    int dim_r = HR[right_size-1].cols();
    Matrix<double> I_right= eye(dim_r);
    Matrix<double> I2 = eye(2);
    //enlarge right block:
    HR[right_size] = tensor(I2,HR[right_size-1]) +
                     tensor(sz0,szR[right_size-1]) +
                     0.5* tensor(splus0.t(),splusR[right_size-1]) +
                     0.5* tensor(splus0,splusR[right_size-1].t());
    splusR[right_size] = tensor(splus0,I_right);
    szR[right_size] = tensor(sz0,I_right) ;
}
```

The density matrix can be readily constructed from the ground state wave-function as:

```
void
DMRGSystem::DensityMatrix(int _position)
{
    int dim_l = HL[left_size].cols();
    int dim_r = HR[right_size].cols();
    // Calculate density matrix
    if(_position == LEFT){
      rho = product(psi,psi.t());
    } else {
      rho = product(psi.t(),psi);
    }
}
```

The truncation consists of diagonalizing the density matrix, and using the matrix of the *m* eigenvectors with the *m* largest eigenvalues to rotate to the new, truncated basis. The arguments are the position, which can assume the values `LEFT` or `RIGHT`, and the maximum number of states to keep *m*:

```
void
DMRGSystem::Truncate(int _position, int _m)
{
    // diagonalize rho
    Vector<double> rho_eig(rho.cols());
    Matrix<double> rho_evec(rho);
    rho_evec.diagonalize(rho_eig);

    // calculate the truncation error for a given number of states
       m
    for(int i = 0; i < rho.cols(); i++) cout << "RHO EIGENVALUE "
       << i << " = " << rho_eig(i) << endl;
```

```
    error = 0.;
    if (_m < rho_eig.size())
      for (int i = 0; i < rho_eig.size()-_m ; i++) error += rho_eig
         (i);
    cout << "Truncation error = " << error <<endl;

    // Truncate by keeping the first m columns
    if (rho.cols() > _m)
      rho_evec = rho_evec(Range(rho.cols()-_m,rho.cols()-1,1),Range
         (0,rho.rows()-1));

    // perform transformation:
    Matrix<double> U = rho_evec.t();
    Matrix<double> aux2;
    if(_position == LEFT){
      aux2 = product(HL[left_size],rho_evec);
      HL[left_size] = product(U,aux2);
      aux2 = product(splusL[left_size],rho_evec);
      splusL[left_size] = product(U,aux2);
      aux2 = product(szL[left_size],rho_evec);
      szL[left_size] = product(U,aux2);
    } else {
      aux2 = product(HR[right_size],rho_evec);
      HR[right_size] = product(U,aux2);
      aux2 = product(splusR[right_size],rho_evec);
      splusR[right_size] = product(U,aux2);
      aux2 = product(szR[right_size],rho_evec);
      szR[right_size] = product(U,aux2);
    }
}
```

We now need to introduce a method to perform the product of the super-Hamiltonian for the left and right halves put togetheri, with the wave-function. The wave-function is represented by a two-dimensional matrix, and the operators are applied in sequence:

```
Matrix<double>
product(DMRGSystem &S, const Matrix<double> & psi)
{
    int left_size = S.left_size;
    int right_size = S.right_size;
    int dim_l = S.HL[left_size].cols();
    int dim_r = S.HR[right_size].cols();
    Matrix<double> npsi(psi); //result

    npsi = product(S.HL[left_size],psi);

    npsi += product(psi,S.HR[right_size].t());
```

```
    Matrix<double> tmat(dim_l,dim_r);
    // Sz.Sz
    tmat= product(psi,S.szR[right_size].t());
    npsi += product(S.szL[left_size],tmat);
    // S+.S-
    tmat= product(psi,S.splusR[right_size])*0.5;
    npsi += product(S.splusL[left_size],tmat);
    // S-.S+
    tmat= product(psi,S.splusR[right_size].t())*0.5;
    npsi += product(S.splusL[left_size].t(),tmat);

    return npsi;
}
```

We finally put all the pieces together in the main block of the code. We will do this in a pretty non-elegant way, by explicily doing all the steps in sequence, which is better for illustration purposes:

```
int main()
{
    // PARAMETERS
    int nsites,n_states_to_keep,n_sweeps;
    cout << "Number of sites: ";
    cin  >> nsites;
    cout << "Number of states to keep: ";
    cin  >> n_states_to_keep;
    cout << "Number of sweeps in finite DMRG: ";
    cin  >> n_sweeps;
    // Operators:
    DMRGSystem S(nsites);

    //-----------------------------------
    // WARMUP: Infinite DMRG sweep
    //-----------------------------------

    for (int n = 1; n < nsites/2; n++){ // do infinite size dmrg
        cout << "WARMUP ITERATION " << n << endl;
        print_blocks(n,n);
        // Create HL and HR by adding the single sites to the two
           blocks
        S.BuildBlockLeft(n);
        S.BuildBlockRight(n);
        // find smallest eigenvalue and eigenvector
        S.GroundState();
        // Calculate density matrix
        S.DensityMatrix(LEFT);
        // Truncate
        S.Truncate(LEFT,n_states_to_keep);
```

```
        // Reflect
        S.DensityMatrix(RIGHT);
        S.Truncate(RIGHT,n_states_to_keep);
    }
    cout << "************************************" << endl;
    cout << "Start sweeps" << endl;
    cout << "************************************"<< endl;
    int first_iter = nsites/2;
    for (int sweep = 1; sweep <= n_sweeps; sweep++){
        for(int iter = first_iter; iter < nsites - 3; iter++){
            cout << "LEFT-TO-RIGHT ITERATION " << iter << endl;
            print_blocks(iter,nsites-iter-2);
            // Create HL and HR by adding the single sites to the
               two blocks
            S.BuildBlockLeft(iter);
            S.BuildBlockRight(nsites-iter-2);
            // find smallest eigenvalue and eigenvector
            S.GroundState();
            // Calculate density matrix
            S.DensityMatrix(LEFT);
            // Truncate
            S.Truncate(LEFT,n_states_to_keep);
            // We copy the left blocks onto the right blocks
        }
        first_iter = 1;
        for(int iter = first_iter; iter < nsites - 3; iter++){
            cout << "RIGHT-TO-LEFT ITERATION " << iter << endl;
            print_blocks(nsites-iter-2,iter);
            // Create HL and HR by adding the single sites to the
               two blocks
            S.BuildBlockRight(iter);
            S.BuildBlockLeft(nsites-iter-2);
            // find smallest eigenvalue and eigenvector
            S.GroundState();
            // Calculate density matrix
            S.DensityMatrix(RIGHT);
            // Truncate
            S.Truncate(RIGHT,n_states_to_keep);
            // We copy the left blocks onto the right blocks
        }
    }
    cout << "************************************"<<endl;
    return 0;
}
```

Listing 4: Introducing measurements

In order to introduce measurements, we need to add arrays to store all the corresponding operators for all the sites inside the block. In order to make this more transparent and elegant, we shall introduce a new class `Block`:

```
class DMRGBlock
{
    public:
        int size;
        Matrix<double> H; // block Hamiltonian
        Vector<Matrix<double> > sz; // Sz operators
        Vector<Matrix<double> > splus; // S+ operators

        DMRGBlock() { resize(1); }
        DMRGBlock(int _size) { resize(_size); }

        DMRGBlock &resize(int n)
          {
            size = n;
            sz.resize(n);
            splus.resize(n);
            return *this;
          }
};
```

The implementation of the `DMRGSystem` class has to reflect this new change:

```
class DMRGSystem
{
    public:
        Matrix<double> sz0; // single site Sz
        Matrix<double> splus0; // single site S+
        Vector<DMRGBlock> BlockL; // left blocks
        Vector<DMRGBlock> BlockR; // right blocks

        Matrix<double> psi;     // g.s. wave function
        Matrix<double> rho;     // density matrix
        double energy;
        double error;
        int nsites;
        int right_size;
        int left_size;

        DMRGSystem(int _nsites)
          {
            sz0.resize(2,2);
            splus0.resize(2,2);
            nsites = _nsites;
```

```
            BlockL.resize(nsites);
            BlockR.resize(nsites);

            // initialize Hamiltonian:
            sz0 = 0.0;
            splus0 = 0.0;
            BlockL[0].H = sz0;
            BlockR[0].H = sz0;
            // single-site operators
            sz0(0,0)         = -0.5;
            sz0(1,1)         =  0.5;
            splus0(1,0)      =  1.0;
            BlockL[0].sz[0] = sz0;
            BlockR[0].sz[0] = sz0;
            BlockL[0].splus[0] = splus0;
            BlockR[0].splus[0] = splus0;
          }

        void BuildBlockLeft(int _iter);
        void BuildBlockRight(int _iter);
        void GroundState();
        void DensityMatrix(int _position);
        void Truncate(int _position, int _m);
        void Measure();
};
```

Most of the implementation will look the same, except that we want account for all the operators inside the `Block` objects. For instance, `BuildBlockLeft` will look:

```
void
DMRGSystem::BuildBlockLeft(int _iter)
{
    left_size = _iter;
    BlockL[left_size].resize(left_size+1);
    Matrix<double> HL = BlockL[left_size-1].H;
    Matrix<double> szL = BlockL[left_size-1].sz[left_size-1];
    Matrix<double> splusL = BlockL[left_size-1].splus[left_size-1];
    int dim_l = HL.cols();
    Matrix<double> I_left = eye(dim_l);
    Matrix<double> I2 = eye(2);
    //enlarge left block:
    BlockL[left_size].H  = tensor(HL,I2) +
                    tensor(szL,sz0) +
                    0.5*tensor(splusL,splus0.t()) +
                    0.5*tensor(splusL.t(),splus0);


    for(int i = 0; i < left_size; i++){
```

```
      BlockL[left_size].splus[i] = tensor(BlockL[left_size-1].splus
         [i],I2);
      BlockL[left_size].sz[i] = tensor(BlockL[left_size-1].sz[i],I2
         );
    }

    BlockL[left_size].splus[left_size] = tensor(I_left,splus0);
    BlockL[left_size].sz[left_size] = tensor(I_left,sz0);
}
```

Now, in the `Truncate` method , we need to rotate all the operators as well:

```
   if(_position == LEFT){
      aux2 = product(BlockL[left_size].H,rho_evec);
      BlockL[left_size].H = product(U,aux2);

      for(int i = 0; i <= left_size; i++){
        aux2 = product(BlockL[left_size].splus[i],rho_evec);
        BlockL[left_size].splus[i] = product(U,aux2);
        aux2 = product(BlockL[left_size].sz[i],rho_evec);
        BlockL[left_size].sz[i] = product(U,aux2);
      }
    } else {
      aux2 = product(BlockR[right_size].H,rho_evec);
      BlockR[right_size].H = product(U,aux2);

      for(int i = 0; i <= right_size; i++){
        aux2 = product(BlockR[right_size].splus[i],rho_evec);
        BlockR[right_size].splus[i] = product(U,aux2);
        aux2 = product(BlockR[right_size].sz[i],rho_evec);
        BlockR[right_size].sz[i] = product(U,aux2);
      }
    }
```

The function `product` also has to be trivially modified:

```
Matrix<double>
product(DMRGSystem &S, const Matrix<double> & psi)
{
    int left_size = S.left_size;
    int right_size = S.right_size;
    Matrix<double> HL = S.BlockL[left_size].H;
    Matrix<double> szL = S.BlockL[left_size].sz[left_size];
    Matrix<double> splusL = S.BlockL[left_size].splus[left_size];
    Matrix<double> HR = S.BlockR[right_size].H;
    Matrix<double> szR = S.BlockR[right_size].sz[right_size];
    Matrix<double> splusR = S.BlockR[right_size].splus[right_size];
    int dim_l = HL.cols();
```

```
    int dim_r = HR.cols();
    Matrix<double> npsi(psi); //result

    npsi = product(HL,psi);
    npsi += product(psi,HR.t());

    Matrix<double> tmat(dim_l,dim_r);
    // Sz.Sz
    tmat= product(psi,szR.t());
    npsi += product(szL,tmat);
    // S+.S-
    tmat= product(psi,splusR)*0.5;
    npsi += product(splusL,tmat);
    // S-.S+
    tmat= product(psi,splusR.t())*0.5;
    npsi += product(splusL.t(),tmat);

    return npsi;
}
```

Since now we have the operators for all the sites in the lattice, we can readily introduce measurements. A local measurement can be implemented as:

```
double
measure(const Matrix<double> &op, const Matrix<double> &psi, int
   pos)
{
  double res = 0;
  Matrix<double> aux(psi); //result

  if(pos == LEFT)
    aux = product(op,psi);
  else
    aux = product(psi,op.t());

  Vector<double> v1 = aux.as_vector();
  Vector<double> v2 = psi.as_vector();
  res = product(v1,v2);
  return res;
}
```

And a two-site measurement, with each site belonging to a different block, as:

```
double
measure(const Matrix<double> &op_left, const Matrix<double> &
   op_right, const Matrix<double> &psi)
{
  double res = 0;
  Matrix<double> aux(psi); //result
```

```
  Matrix <double> kk = op_right.t();
  aux = product(op_left,psi);
  aux = product(aux,op_right.t());

  Vector<double> v1 = aux.as_vector();
  Vector<double> v2 = psi.as_vector();
  res = product(v1,v2);
  return res;
}
```

We will define a method that will make the interface with `DMRGSystem` more transparent:

```
void
DMRGSystem::Measure()
{
  const DMRGBlock &BL = BlockL[left_size];
  const DMRGBlock &BR = BlockR[right_size];

  for(int i = 0; i <= left_size; i++)
    cout << "Sz(" << i << ") = " << measure(BL.sz[i],psi,LEFT) <<
       endl;

  for(int i = 0; i <= right_size; i++)
    cout << "Sz(" << nsites-i-1 << ") = " << measure(BR.sz[i],psi,
       RIGHT) << endl;

  for(int i = 0; i <= left_size; i++)
    for(int j = 0; j <= right_size; j++)
      cout << "Sz(" << i << ")Sz(" << nsites-j-1 << ") = " <<
         measure(BL.sz[i],BR.sz[j],psi) << endl;
}
```

So now, all we have to do, is to introduce a call to `Measure` in main:

```
    for (int sweep = 1; sweep <= n_sweeps; sweep++){
        for(int iter = first_iter; iter < nsites - 3; iter++){
            cout << "LEFT-TO-RIGHT ITERATION " << iter << endl;
            print_blocks(iter,nsites-iter-2);
            // Create HL and HR by adding the single sites to the
               two blocks
            S.BuildBlockLeft(iter);
            S.BuildBlockRight(nsites-iter-2);
            // find smallest eigenvalue and eigenvector
            S.GroundState();
            // Measure correlations
            S.Measure();
            // Calculate density matrix
            S.DensityMatrix(LEFT);
```

```
            // Truncate
            S.Truncate(LEFT,n_states_to_keep);
        }
        first_iter = 1;
        for(int iter = first_iter; iter < nsites - 3; iter++){
            cout << "RIGHT-TO-LEFT ITERATION " << iter << endl;
            print_blocks(nsites-iter-2,iter);
            // Create HL and HR by adding the single sites to the
               two blocks
            S.BuildBlockRight(iter);
            S.BuildBlockLeft(nsites-iter-2);
            // find smallest eigenvalue and eigenvector
            S.GroundState();
            // Measure correlations
            S.Measure();
            // Calculate density matrix
            S.DensityMatrix(RIGHT);
            // Truncate
            S.Truncate(RIGHT,n_states_to_keep);
        }
    }
```

Listing 5: The wave-function transformation

Thsi is probably the most subtle and confusing element in the DMRG implementation. We shall follow the lecture notes step by step. Until now, we have been usiing a two-dimensional matrix to represent the ground state. This simplifies the implementation a great deal, since all the operations reduce to matrix-matrix and matrix-vector multiplications. But in order to do the wave-function transformation we need to be able to identify the degrees of freedom corresponding to a single-site in the center, and the left and right blocks individually. In order to do that, we use a `Tensor` class. We can seemlesly reshape our wave-function from a `Tensor` to a `Matrix`

The only part of the code where we need to account for these issues is in the method `BuildSeed` which transforms the wave-function from the previouse step in the sweep, to the new basis. Again, we remind the reader that the matrix elements are accessed as `(column,row)` which introduces some minor considerations:

```
// The direction of the sweep can be LEFT2RIGHT or RIGHT2LEFT
void
DMRGSystem::BuildSeed(int dir)
{
  int dim_l = BlockL[left_size-1].H.cols();
  int dim_r = BlockR[right_size-1].H.cols();
  Tensor<double> psi_new(dim_l,2,2,dim_r);

  if(dir == LEFT2RIGHT){
    if(left_size == 1) {
      seed = psi;
```

```
      return;
    } else {
      const Matrix<double> &UL = BlockL[left_size-1].U;
      const Matrix<double> &UR = BlockR[right_size].U;
      int old_dim_l = BlockL[left_size-2].U.cols();
      int old_dim_r = BlockR[right_size].U.cols();
// We copy the old g.s. (in matrix form) into the tensor
      Tensor<double> psi_old(old_dim_l*2,1,1,2*old_dim_r);
      for(int i = 0; i < 2*old_dim_l; i++)
        for(int j = 0; j < 2*old_dim_r; j++) psi_old(i,0,0,j) = psi
           (j,i);
// We first transform the left part
      psi_old.resize(old_dim_l*2,1,2,old_dim_r);
      Tensor<double> psi_aux(dim_l,1,2,old_dim_r);
      for(int i3 = 0; i3 < 2; i3++){
        for(int i4 = 0; i4 < old_dim_r; i4++){
          Vector<double> vaux = psi_old(Range(0,old_dim_l*2-1),0,i3
             ,i4);
          psi_aux(Range(0,dim_l-1),0,i3,i4) = product(UL.t(),vaux);
        }
      }
// We now transform the right part
      psi_aux.resize(dim_l,2,1,old_dim_r);
      psi_new.resize(dim_l,2,1,2*dim_r);
      for(int i1 = 0; i1 < dim_l; i1++){
        for(int i2 = 0; i2 < 2; i2++){
          Vector<double> vaux = psi_aux(i1,i2,0,Range(0,old_dim_r
             -1));
          psi_new(i1,i2,0,Range(0,2*dim_r-1)) = product(UR,vaux);
        }
      }
    }
  } else {
    if(right_size == 1) {
      seed = psi;
      return;
    } else {
      const Matrix<double> &UL = BlockL[left_size].U;
      const Matrix<double> &UR = BlockR[right_size-1].U;
      int old_dim_l = BlockL[left_size].U.cols();
      int old_dim_r = BlockR[right_size-2].U.cols();
      cout << old_dim_l << " " << old_dim_r << endl;
// We copy the old g.s. (in matrix form) into the tensor
      Tensor<double> psi_old(old_dim_l*2,1,1,2*old_dim_r);
      for(int i = 0; i < 2*old_dim_l; i++)
        for(int j = 0; j < 2*old_dim_r; j++) psi_old(i,0,0,j) = psi
           (j,i);
// We first transform the right part
```

```
      psi_old.resize(old_dim_l,2,1,2*old_dim_r);
      Tensor<double> psi_aux(old_dim_l,2,1,dim_r);
      for(int i1 = 0; i1 < old_dim_l; i1++){
        for(int i2 = 0; i2 < 2; i2++){
          Vector<double> vaux = psi_old(i1,i2,0,Range(0,old_dim_r
             *2-1));
          psi_aux(i1,i2,0,Range(0,dim_r-1)) = product(UR.t(),vaux);
        }
      }
// We now transform the left part
      psi_aux.resize(old_dim_l,1,2,dim_r);
      psi_new.resize(dim_l*2,1,2,dim_r);
      for(int i3 = 0; i3 < 2; i3++){
        for(int i4 = 0; i4 < dim_r; i4++){
          Vector<double> vaux = psi_aux(Range(0,old_dim_l-1),0,i3,
             i4);
          psi_new(Range(0,2*dim_l-1),0,i3,i4) = product(UL,vaux);
        }
      }
    }
  }

  // Let's transform back to a matrix - remeber that matrices obey
     Fortran conventions
  psi_new.resize(dim_l*2,1,1,2*dim_r);
  seed.resize(2*dim_r,2*dim_l);
  for(int i = 0; i < 2*dim_l; i++)
    for(int j = 0; j < 2*dim_r; j++) seed(j,i) = psi_new(i,0,0,j);
}
```

Now, we introduce a call to `BuildSeed` in the main block of the code:

```
    cout << "************************************" << endl;
    cout << "Start sweeps" << endl;
    cout << "************************************"<< endl;
    int first_iter = 1;
    for (int sweep = 1; sweep <= n_sweeps; sweep++){
        for(int iter = first_iter; iter < nsites - 2; iter++){
            cout << "RIGHT-TO-LEFT ITERATION " << iter << endl;
            print_blocks(nsites-iter-2,iter);
            // Create HL and HR by adding the single sites to the
               two blocks
            S.BuildBlockRight(iter);
            S.BuildBlockLeft(nsites-iter-2);
            // find smallest eigenvalue and eigenvector
            S.BuildSeed(RIGHT2LEFT);
            S.GroundState(true);  // Use the seed to start the
               lanczos iterations
            // Measure correlations
```

```
            S.Measure();
            // Calculate density matrix
            S.DensityMatrix(RIGHT);
            // Truncate
            S.Truncate(RIGHT,n_states_to_keep);
        }
        for(int iter = first_iter; iter < nsites - 2; iter++){
            cout << "LEFT-TO-RIGHT ITERATION " << iter << endl;
            print_blocks(iter,nsites-iter-2);
            // Create HL and HR by adding the single sites to the
               two blocks
            S.BuildBlockLeft(iter);
            S.BuildBlockRight(nsites-iter-2);
            // find smallest eigenvalue and eigenvector
            S.BuildSeed(LEFT2RIGHT);
            S.GroundState(true);  // Use the seed to start the
               lanczos iterations
            // Measure correlations
            S.Measure();
            // Calculate density matrix
            S.DensityMatrix(LEFT);
            // Truncate
            S.Truncate(LEFT,n_states_to_keep);
        }
    }
```

Listing 6: A time-dependent DMRG code

We have now introduced all the ingredient necessary to write a time-dependent implementation. We first need to calculate the bond evolution operator for two-sites. We have seen tthat for the case of spins we can do it analytically, but this piece of code can be generalized to more comlicated scenarios:

```
Matrix<MYTYPE >
DMRGSystem::BondExp(double tstep)
{
  Matrix<MYTYPE > Uij(4,4);
  Matrix<MYTYPE > Hij(4,4), U(4,4), Ut(4,4), aux(4,4);
  Vector<double> wk(4);
  Hij = tensor(sz0,sz0) +
        MYTYPE(0.5)*tensor(splus0,splus0.ct()) +
        MYTYPE(0.5)*tensor(splus0.ct(),splus0);
  U = Hij.diagonalize(wk);
  for(int i = 0; i < 4; i++) cout << i << " " << wk(i) << endl;
  Ut = ctranspose(U);

  Hij = MYTYPE(0);
```

```
  for(int i = 0; i < 4; i++) Hij(i,i) = std::exp(complex<double
     >(0,-wk[i]*tstep));

  aux = product(Hij,Ut);
  Uij = product(U,aux);
  for(int i = 0; i < 4; i++)
    for(int j = 0; j < 4; j++) cout << i << " " << j << " " << U(i,
       j) << " " << Uij(i,j) << endl;
  return Uij;
}
```

Notice that we have used a differnt argument `MYTYPE` in the templates. This is a C++ tool that allows us to choose the data type before compiling. For real-time evolution, `MYTYPE` should be set to`complex<double>`

We now introduce a method `TimeStep` that applies the bond evolution operator, or not, to the wave-function, according to the Suzuki-Trotter decomposition. Again, we transform the wave-function to a tensor, so we can operate with 2×2 matrices:

```
void
DMRGSystem::TimeStep()
{
  psi = seed;
  Matrix<MYTYPE > aux(psi);

  int dim_l = BlockL[left_size-1].H.cols();
  int dim_r = BlockR[right_size-1].H.cols();

  Tensor<MYTYPE > tstate(dim_l*2,1,1,2*dim_r);
  for(int i = 0; i < 2*dim_l; i++)
    for(int j = 0; j < 2*dim_r; j++) tstate(i,0,0,j) = aux(j,i);


  if(left_size == 1) {
    cout << "APPLYING exp(-itH_ij) " << 0 << endl;
    tstate.resize(4,1,2,dim_r);
    for(int i3 = 0; i3 < 2; i3++){
      for(int i4 = 0; i4 < dim_r; i4++){
        Vector<MYTYPE > vaux = tstate(Range(0,3),0,i3,i4);
        tstate(Range(0,3),0,i3,i4) = product(texp[0],vaux);
      }
    }
  }
  if(right_size == 1) {
    cout << "APPLYING exp(-itH_ij) " << nsites-2 << endl;
    tstate.resize(dim_l,2,1,4);
    for(int i1 = 0; i1 < dim_l; i1++){
      for(int i2 = 0; i2 < 2; i2++){
        Vector<MYTYPE > vaux = tstate(i1,i2,0,Range(0,3));
        tstate(i1,i2,0,Range(0,3)) = product(texp[nsites-2],vaux);
```

```
      }
    }
  }

  cout << "APPLYING exp(-itH_ij) " << left_size << endl;
  tstate.resize(dim_l,4,1,dim_r);
  for(int i1 = 0; i1 < dim_l; i1++){
    for(int i4 = 0; i4 < dim_r; i4++){
      Vector<MYTYPE > vaux = tstate(i1,Range(0,3),0,i4);
      tstate(i1,Range(0,3),0,i4) = product(texp[left_size],vaux);
    }
  }

  tstate.resize(dim_l*2,1,1,2*dim_r);
  for(int i = 0; i < 2*dim_l; i++)
    for(int j = 0; j < 2*dim_r; j++) psi(j,i) = tstate(i,0,0,j);

}
```

We have created a method `Sweep` that will calculate the ground state or evolve in time, accroding to the argument `time_evolution`:

```
void
DMRGSystem::Sweep(int direction, int n_states_to_keep, bool
   do_measure, bool time_evolution)
{
  if(direction == RIGHT2LEFT){
    for(int iter = 1; iter < nsites - 2; iter++){
      cout << "RIGHT-TO-LEFT ITERATION " << iter << endl;
      print_blocks(nsites-iter-2,iter);
      // Create HL and HR by adding the single sites to the two
         blocks
      BuildBlockRight(iter);
      BuildBlockLeft(nsites-iter-2);
      // find smallest eigenvalue and eigenvector
      BuildSeed(RIGHT2LEFT);
      if(time_evolution){
        TimeStep();
      } else {
        GroundState(true);
      }
      // Measure correlations
      if(do_measure && left_size == nsites/2-1) Measure();
//      if(do_measure) Measure();
      // Calculate density matrix
      DensityMatrix(RIGHT);
      // Truncate
      Truncate(RIGHT,n_states_to_keep);
    }
```

```
  } else { // LEFT2RIGHT
    for(int iter = 1; iter < nsites - 2; iter++){
      cout << "LEFT-TO-RIGHT ITERATION " << iter << endl;
      print_blocks(iter,nsites-iter-2);
      // Create HL and HR by adding the single sites to the two
         blocks
      BuildBlockLeft(iter);
      BuildBlockRight(nsites-iter-2);
      // find smallest eigenvalue and eigenvector
      BuildSeed(LEFT2RIGHT);
      if(time_evolution){
        TimeStep();
      } else {
        GroundState(true);
      }
      // Measure correlations
      if(do_measure && left_size == nsites/2-1) Measure();
//      if(do_measure) Measure();
      // Calculate density matrix
      DensityMatrix(LEFT);
      // Truncate
      Truncate(LEFT,n_states_to_keep);
    }
  }
}
```

Now the main block will reflect these changes and will look more transparent:

```
int main()
{
    // PARAMETERS
       ------------------------------------------------------------
    int nsites,n_states_to_keep,n_sweeps;
    double tstep;
    cout << "Number of sites: ";
    cin  >> nsites;
    cout << "Number of states to keep: ";
    cin  >> n_states_to_keep;
    cout << "Number of sweeps in finite DMRG: ";
    cin  >> n_sweeps;
    cout << "Time step: ";
    cin  >> tstep;
    // Operators:
    DMRGSystem S(nsites);

    S.Warmup(n_states_to_keep);
    cout << "**********************************" << endl;
    cout << "Start sweeps" << endl;
```

```
    cout << "************************************"<< endl;
    for (int sweep = 1; sweep <= n_sweeps; sweep++){
      S.Sweep(RIGHT2LEFT, n_states_to_keep, true, false);
      S.Sweep(LEFT2RIGHT, n_states_to_keep, true, false);
    }
    cout << "************************************" << endl;
    cout << "Start time evolution" << endl;
    cout << "************************************"<< endl;
    // Build evolution operators
    Vector<Matrix<MYTYPE > > Ueven(nsites), Uodd(nsites);
    Matrix<MYTYPE > Ueven0 = S.BondExp(tstep/2);
    Matrix<MYTYPE > Uodd0 = S.BondExp(tstep);
    for(int i = 0; i < nsites-1; i++){
      if(IS_EVEN(i)) {
        Ueven[i] = Ueven0;
        Uodd[i] = eye(4);
      } else {
        Uodd[i] = Uodd0;
        Ueven[i] = eye(4);
      }
    }

    double tmax = 10.;
    while(S.time < tmax){
      // time step
      for(int i = 0; i < nsites-1; i++) S.texp[i] = Ueven[i];
      S.Sweep(RIGHT2LEFT, n_states_to_keep, false, true);
      for(int i = 0; i < nsites-1; i++) S.texp[i] = Uodd[i];
      S.Sweep(LEFT2RIGHT, n_states_to_keep, false, true);
      for(int i = 0; i < nsites-1; i++) S.texp[i] = Ueven[i];
      S.Sweep(RIGHT2LEFT, n_states_to_keep, false, true);
      // Measurement sweep
      for(int i = 0; i < nsites-1; i++) S.texp[i] = eye(4);
      S.Sweep(LEFT2RIGHT, n_states_to_keep, true, true);
      S.time += tstep;
    }

    cout << "************************************"<<endl;
    return 0;
}
```

Notice that the first sweeps after the warmup are needed to calculate the initial state for the time-evolution. If we just calculate the ground-state, the time-evolution will not have any effect, since it is an eigenstate of the Hamiltonian. Therefore, we need to use a different Hamiltonian to initialize the system at time $t = 0$. Following the example in the notes, we polarize hall of the system in one direction, and the other half in the opposite direction, using magnetic fields. In order to do this, we need to modify `BuildBlockLeft`, `BuildBlockRight` and `product`, using the appropriate Hamiltonian:

```
void
DMRGSystem::BuildBlockLeft(int _iter)
{
    left_size = _iter;
    BlockL[left_size].resize(left_size+1);
    Matrix<MYTYPE > HL = BlockL[left_size-1].H;
    Matrix<MYTYPE > szL = BlockL[left_size-1].sz[left_size-1];
    Matrix<MYTYPE > splusL = BlockL[left_size-1].splus[left_size
       -1];
    int dim_l = HL.cols();
    Matrix<MYTYPE > I_left = eye(dim_l);
    Matrix<MYTYPE > I2 = eye(2);
    //enlarge left block:
    MYTYPE coef = (left_size <= nsites/2-1) ? 1. : -1;
    BlockL[left_size].H  = tensor(HL,I2) + coef*tensor(I_left,sz0);

    for(int i = 0; i < left_size; i++){
      BlockL[left_size].splus[i] = tensor(BlockL[left_size-1].splus
         [i],I2);
      BlockL[left_size].sz[i] = tensor(BlockL[left_size-1].sz[i],I2
         );
    }

    BlockL[left_size].splus[left_size] = tensor(I_left,splus0);
    BlockL[left_size].sz[left_size] = tensor(I_left,sz0);
}

void
DMRGSystem::BuildBlockRight(int _iter)
{
    right_size = _iter;
    BlockR[right_size].resize(right_size+1);
    Matrix<MYTYPE > HR = BlockR[right_size-1].H;
    Matrix<MYTYPE > szR = BlockR[right_size-1].sz[right_size-1];
    Matrix<MYTYPE > splusR = BlockR[right_size-1].splus[right_size
       -1];
    int dim_r = HR.cols();
    Matrix<MYTYPE > I_right= eye(dim_r);
    Matrix<MYTYPE > I2 = eye(2);

    //enlarge right block:
    MYTYPE coef = (right_size <= nsites/2-1) ? -1. : 1;
    BlockR[right_size].H = tensor(I2,HR) + coef*tensor(sz0,I_right)
       ;

    for(int i = 0; i < right_size; i++){
      BlockR[right_size].splus[i] = tensor(I2,BlockR[right_size-1].
         splus[i]);
```

```
      BlockR[right_size].sz[i] = tensor(I2,BlockR[right_size-1].sz[
         i]);
    }

    BlockR[right_size].splus[right_size] = tensor(splus0,I_right);
    BlockR[right_size].sz[right_size] = tensor(sz0,I_right) ;
}

Matrix<MYTYPE >
product(DMRGSystem &S, const Matrix<MYTYPE > & psi)
{
    int left_size = S.left_size;
    int right_size = S.right_size;
    Matrix<MYTYPE > szL = S.BlockL[left_size].sz[left_size];
    Matrix<MYTYPE > splusL = S.BlockL[left_size].splus[left_size];
    Matrix<MYTYPE > szR = S.BlockR[right_size].sz[right_size];
    Matrix<MYTYPE > splusR = S.BlockR[right_size].splus[right_size
       ];
    Matrix<MYTYPE > npsi(psi); //result
    Matrix<MYTYPE > HL = S.BlockL[left_size].H;
    Matrix<MYTYPE > HR = S.BlockR[right_size].H;

    npsi = product(HL,psi);
    npsi += product(psi,HR.t());

    return npsi;
}
```

REFERENCES

1. S.R. White, Phys. Rev. Lett. **69**, 2863 (1992); Phys. Rev. B **48**, 10345 (1993).
2. *Density-Matrix Renormalization - A New Numerical Method in Physics: Lectures of a Seminar and Workshop held at the Max-Planck-Institut für Physik*, Lecture Notes in Physics, Springer-Verlag (1999), edited by Ingo Peschel, Xiaoqun Wang, Matthias Kaulke and Karen Hallberg.
3. U. Schollwöck, Rev. Mod. Phys. **77**, 259 (2005).
4. K. Hallberg, *"Density Matrix Renormalization: A Review of the Method and its Applications"*, in the book *"Theoretical Methods for Strongly Correlated Electrons"*, CRM Series in Mathematical Physics, Springer, New York (2003). Edited by David Senechal, Andre-Marie Tremblay and Claude Bourbonnais.
5. K. Hallberg, Adv.Phys. **55**, 477 (2006).
6. R. M. Noack and S. R. Manmana, proceedings of the "IX. Training Course in the Physics of Correlated Electron Systems and High-Tc Superconductors", Vietri sul Mare (Salerno, Italy, October 2004), AIP Conf. Proc. **789**, 93 (2005).
7. G. De Chiara, M. Rizzi, D. Rossini, S. Montangero, J. Comput. Theor. Nanosci. **5**, 1277 (2008).
8. A.C. Hewson, *The Kondo Problem to Heavy Fermions*, Cambridge Univ. Press (1997)
9. B. Bauer et al. (ALPS collaboration), J. Stat. Mech. P05001 (2011); F. Albuquerque et al. (ALPS collaboration), Journal of Magnetism and Magnetic Materials **310**, 1187 (2007); F. Alet et al. (ALPS collaboration), J. Phys. Soc. Jpn. Suppl. **74**, 30 (2005).
10. T. Giamarchi, *Quantum Physics in One Dimension*, Clarendon Press, Oxford (2004).
11. A. O.Gogolin,A. A. Nersesyan, A. M.Tsvelik, *Bosonization of Strongly Correlated Systems*, Cambridge University Press, Cambridge, England (1998).
12. F.Haldane, J. of. Phys. C **14**, 2585 (1981).
13. S. M. Girvin, *"The Quantum Hall Effect: Novel Excitations and Broken Symmetries"* , in the book *Topological Aspects of Low Dimensional Systems*, edited by A. Comtet, T. Jolicoeur, S. Ouvry, F. David, Springer-Verlag, Berlin and Les Editions de Physique, Les Ulis, (2000).
14. Richard E. Prange, Steven M. Girvin, K.v. Klitzing and M.E. Cage, *The Quantum Hall Effect*, Graduate Texts in Contemporary Physics / Maryland Subseries: Based on Lectures at the University of Maryland, College Park.
15. D. Yoshioka, *The Quantum Hall Effect*, Springer (2002).
16. J. Jain, *Composite Fermions*, Cambridge University Press (2007).
17. R. Hanson, *et al.* , Rev. Mod. Phys. **79**, 1217 (2007). See also L. P. Kouwenhoven, D. G. Austing, and S. Tarucha, Rep. Prog. Phys **64**, 701 (2001).
18. M. Grobis, *et al.* , *Handbook of Magnetism and Advanced Magnetic Materials, Vol. 5.*, Wiley (2009).
19. Oliver Morsch and Markus Oberthaler, Rev. Mod. Phys. **78**, 179 (2006).
20. W. Ketterle and M. W. Zwierlein, *Making, Probing and Understanding Ultracold Fermi Gases*, Proceedings of the International School of Physics "Enrico Fermi", Course CLXIV, Varenna. Edited by M. Inguscio, W. Ketterle, and C. Salomon, IOS Press, Amsterdam (2008).
21. Immanuel Bloch, Jean Dalibard and Wilhelm Zwerger, Rev. Mod. Phys. **80**, 885 (2008).
22. A. Cho, "The Mad Dash to Make Light Crystals", Science **18**, 312 (2008).
23. Fabien Alet, Aleksandra M. Walczak, Matthew P.A. Fisher, "Exotic quantum phases and phase transitions in correlated matter" , Notes of the Lectures at the International Summer School on Fundamental Problems in Statistical Physics XI, September 2005, Leuven, Belgium; Physica A **369**, 122 (2006).
24. K. G. Wilson, Rev. Mod. Phys. **47**, 773 (1975)
25. R. Bulla, T. A. Costi, and T. Pruschke, Rev. Mod. Phys. **80**, 395 (2008).
26. R. B. Laughlin, Phys. Rev. Lett. **50**, 1395 (1983).
27. F. D. M. Haldane, Physics Letters **80A**, 281 (1980); T. C. Choy, Physics Letters **80A**, 49 (19080). See also S. R. Manmana *et al.*, to be published.
28. R. Schrieffer, *Theory of Superconductivity*, Advanced Books Classics, Perseus (1999).
29. T. Nishino, J. Phys. Soc. Jpn. **64**, 3598 (1995); R. J. Bursill, Phys. Rev. B **60**, 1643 (1999); X. Q. Wang and T. Xiang, Phys. Rev. B **56**, 5061 (1997); N. Shibata, J. Phys. Soc. Jpn. **66**, 2221 (1997).
30. G. Vidal, Phys. Rev. Lett. **91**, 147902 (2003). ; Phys. Rev. Lett. **93**, 040502 (2004).
31. S. R. White, Phys. Rev. Lett. **77**, 3633 (1996).
32. F. Verstraete, M. M. Wolf, D. Perez-Garcia, and J. I. Cirac, Phys. Rev. Lett. **96**, 220601 (2006)
33. N. Schuch, M. M. Wolf, F. Verstraete, and J. I. Cirac, Phys. Rev. Lett. **100**, 030504 (2008).
34. S.R. White and A. E. Feiguin, Phys. Rev. Lett. **93**, 076401 (2004).
35. A. J. Daley, C. Kollath, U. Schollwöck, and G. Vidal, J. Stat. Mech.: Theor. Exp. P04005 (2004).
36. M. A. Cazalilla and J. B. Marston, Phys. Rev. Lett. **88**, 256403 (2002); **91**, 049702 (2003). H. G. Luo, T. Xiang, and X. Q. Wang, Phys. Rev. Lett. **91**, 049701 (2003).
37. F. Verstraete, J. J. García-Ripoll, and J. I. Cirac, Phys. Rev. Lett. **93**, 207204 (2004).

38. Adrian E. Feiguin and Steven R. White, Phys. Rev. B **72**, 220401 (2005).
39. N. F. Mott, *Metal Insulator Transitions*, Taylor & Francis; 2nd Edition (1990). Masatoshi Imada, Atsushi Fujimori, and Yoshinori Tokura, Rev. Mod. Phys. **70**, 1039 (1998).
40. For instance, LAPACK (Linear Algebra Package), `http://www.netlib.org/lapack`
41. F. Verstraete, D. Porras, and J. I. Cirac, Phys. Rev. Lett. **93**, 227205 (2004). See also P. Pippan, S. R. White, and H. G. Evertz, Phys. Rev. B **81**, 081103 (2010).
42. I. McCulloch, J. Stat. Mech. P10014, (2007).
43. C. Lanczos, J. of Research of the National Bureau of Standards **45**, 255 (1950).
44. Pettifor, D. G.; Weaire, D. L. (eds.), *The Recursion Method and Its Applications*. Springer Series in Solid State Sciences 58, Springer-Verlag (1985).
45. R. M. Noack, S. R. White, D. J. Scalapino, "The Density-Matrix renormalizatioon Group for Fermion Systems", in *Computer Simulations in Condensed Matter Physics VII*, Eds. D.P. Landau, K.K. Mon, and H.B. Schüttler (Spinger Verlag, Heidelberg, Berlin, 1994), UCI-CMTHE-94-03.
46. S. Liang, and H. Pang, Europhys. Lett. **32**, 173 (1995).
47. M. S. L. du Croo de Jongh, and J. M. J. van Leeuwen, Phys. Rev. B **57**, 8494 (1998).
48. T. Xiang, J. Z. Lou, and Z. B. Zu, Phys. Rev. B **64** 104414 (2001).
49. M. B. Plenio, and S. Virmani, Quantum Inf. Comput. **7**,1 (2007).
50. Luigi Amico, Rosario Fazio, Andreas Osterloh, and Vlatko Vedral, Rev. Mod. Phys. **80**, 517 (2008).
51. R. Horodecki, P. Horodecki, M. Horodecki, and K. Horodecki, Rev. Med. Phys. **81**, 865 (2009).
52. J. Eisert, M. Cramer, and M. B. Plenio, Rev. Mod. Phys. **82**, 277 (2010).
53. F. Verstraete, and J. I. Cirac, Phys. Rev. B **73**, 094423 (2006).
54. M. B. Hastings, J. Stat. Mech.: Theory Exp. P08024 (2007).
55. P. Calabrese and J. Cardy, J. Stat. Mech.: Theory Exp. P06002 (2004); *ibid.* Int. J. Quantum Inf. 4, 429 (2006).
56. M. M. Wolf, Phys. Rev. Lett. **96**, 010404 (2006).
57. D. Gioev, I. Klich, Phys. Rev Lett. **96**, 100503 (2006).
58. W. Li, L. Ding, R. Yu, T. Roscilde, and S. Haas, Phys. Rev. B **74**, 073103 (2006).
59. T. Barthel, M.-C. Chung, U. Schollwöck, Phys. Rev. A **74**, 022329 (2006).
60. S. Ostlund and S. Rommer, Phys. Rev. Lett. **75**, 3537 (1995).
61. S. Rommer and S. Ostlund, Phys. Rev. B **55**, 2164 (1997).
62. H. A. Kramers and G. H. Wannier, Phys. Rev. **60**, 263 (1941); R. J. Baxter, J. Math. Phys. **9**, 650 (1968).
63. J. Jordan, R. Orus, G. Vidal, F. Verstraete, J. I. Cirac, Phys. Rev. Lett. **101**, 250602 (2008).
64. U. Schollwöck, Ann. of Phys. **326**, 96 (2011).
65. D. Perez-Garcia, F. Verstraete, MM. Wolf, and J. I. Cirac., Quantum Inf. Comput. **7**, 401 (2007)
66. F. Verstraete, J.I. Cirac, V. Murg, Adv. Phys. **57**,143 (2008).
67. J. I. Cirac and F. Verstraete, J. of Phys. A: Math. and Theor. **42** (2009) 504004.
68. J. Dukelsky, M. A. Martin-Delgado, T. Nishino, and G. Sierra, Erophys. Lett. **43**, 457 (1998).
69. Tomotoshi Nishino and Kouichi Okunishi, J. Phys. Soc. Jpn. **64** (1995).
70. H. Takasaki, T. Hikihara, and T. Nishino, J. Phys. Soc. Jpn. **68**, 1537 (1999).
71. Kouji Ueda, Tomotoshi Nishino, Kouichi Okunishi, Yasuhiro Hieida, Rene Derian, Andrej Gendiar, J. Phys. Soc. Jpn. **75** 014003 (2006).
72. F. Verstraete, D. Porras, and J. I. Cirac, Phys. Rev. Lett. **93**, 227205 (2004).
73. A. Sandvik and G. Vidal, Phys. Rev. Lett. **99**, 220602 (2007)
74. S. Sandvik, Phys. Rev. Lett. **101**, 140603 (2008).
75. Salvatore R. Manmana, Alejandro Muramatsu, Reinhard M. Noack,*"Time evolution of one-dimensional Quantum Many Body Systems"*. Contribution for the conference proceedings of the *"IX. Training Course in the Physics of Correlated Electron Systems and High-Tc Superconductors"* held in Vietri sul Mare (Salerno, Italy) in October 2004, AIP Conf. Proc. **789**, 269 (2005).
76. Ulrich Schollwöck, J. Phys. Soc. Jpn. 74 (Suppl.), **246** (2005).
77. U. Schollwöck, S. R. White, in *Effective models for low-dimensional strongly correlated systems*, p. 155, edited by G. G. Batrouni and D. Poilblanc. AIP, Melville, New York (2006)
78. M. Suzuki, J. Phys. Soc. Jpn. **12**, 4483 (1985), and references therein.
79. E. Forest and R.D. Ruth, Physica D **43**, 105 (1990). See also I.P. Omelyan, I.M. Mryglod, R. Folk, Comp. Phys. Comm **146**, 188 (2002).
80. J. J. García-Ripoll, New Journal of Physics **8**, 305 (2008).
81. S. M. Barnett and P. L. Knight, Phys. Rev. A **38**, 1657 (1988); J. Opt. Soc. Am. **B** , 467 (1985).
82. C. Kollath, U. Schollwöck, W. Zwerger, Phys. Rev. Lett. **95**, 176401 (2005).
83. Adrian E. Feiguin and Steven R. White, Phys. Rev. B **72**, 020404 (2005).
84. F. Verstraete, J. J. Garcia-Ripoll, J. I. Cirac, Phys. Rev. Lett. **93**, 207204 (2004).
85. Michael Zwolak, Guifre Vidal, Phys. Rev. Lett. **93**, 207205 (2004).

86. Adrian E. Feiguin and Steven R. White, Phys. Rev. B **72**, 220401 (2005).
87. F. Heidrich-Meisner, S. R. Manmana, M. Rigol, A. Muramatsu, A. E. Feiguin, and E. Dagotto, Phys. Rev. A **80**, R041603 (2009).
88. F. Heidrich-Meisner, M. Rigol, A. Muramatsu, A. E. Feiguin, and E. Dagotto, Phys. Rev. A **78**, 013620 (2008).
89. K.A. Al-Hassanieh, A.E. Feiguin, J.A. Riera, C.A. Busser, E. Dagotto, Phys. Rev. B **73**, 195304 (2006).
90. Luis G. G. V. Dias da Silva, F. Heidrich-Meisner, A. E. Feiguin, C. A. Busser, G. B. Martins, E. V. Anda, E. Dagotto, Phys. Rev. B **78**, 195317 (2008).
91. Adrian Feiguin, Paul Fendley, Matthew P.A. Fisher, Chetan Nayak, Phys. Rev. Lett **101**, 236801 (2008).
92. F. Heidrich-Meisner, A.E. Feiguin, E. Dagotto, Phys. Rev. B **79**, 235336 (2009).
93. F. Heidrich-Meisner, I. Gonzalez, K.A. Al-Hassanieh, A.E. Feiguin, M.J. Rozenberg, E. Dagotto, Phys. Rev. B **82**, 205110 (2010).
94. P. Schmitteckert, Phys. Rev. B **70**, 121302(R) (2004).
95. E. Boulat, H. Saleur, and P. Schmitteckert, Phys. Rev. Lett. **101**, 140601 (2008).
96. Dan Bohr, Peter Schmitteckert, Peter Woelfle , Europhys. Lett., **73**, 246 (2006).
97. Elbio Dagotto, Rev. Mod. Phys. **66**, 763 (1994).
98. J. E. Gubernatis, Mark Jarrell, R. N. Silver, and D. S. Sivia Phys. Rev. B 44, 6011 (1991).
99. K. Hallberg, Phys. Rev. B **52** R9827 (1995).
100. Till D. Kühner and Steven R. White, Phys. Rev. B **60**, 335 (1999).
101. E. Jeckelmann, Phys. Rev. B **66**, 045114 (2002).
102. A. E. Feiguin and S. R. White, Phys. Rev. B **72**, 20404 (2005).
103. A. E. Feiguin and D. A. Huse, Phys. Rev. B **79**, 100507 (2009).
104. A. E. Feiguin and G. Fiete, Phys. Rev. B **81**, 075108 (2010).
105. Pierre Bouillot, Corinna Kollath, Andreas M. LÃd'uchli, Mikhail Zvonarev, Benedikt Thielemann, Christian Rüegg, Edmond Orignac, Roberta Citro, Martin Klanjsek, Claude Berthier, Mladen Horvati/'c, and Thierry Giamarchi, Phys. Rev. B **83**, 054407 (2011).
106. Thomas Barthel, Ulrich Schollwöck, and Steven R. White, Phys. Rev. B **79**, 245101 (2009).
107. S. R. White and I. Affleck, Phys. Rev. B **77**, 134437 (2008)
108. Pasquale Calabrese, John Cardy, J.Stat.Mech. P04010 (2005); *ibid* Phys.Rev.Lett. **96** (2006) 136801.
109. Pasquale Calabrese, John Cardy, J. Stat. Mech. P10004 (2007).
110. V. Eisler, I. Peschel, J. Stat. Mech. P06005 (2007).
111. Viktor Eisler, Ingo Peschel, Ann. Phys. (Berlin) **17**, 410 (2008).
112. C. Kollath, A. Iucci, T. Giamarchi, W. Hofstetter, and U. Schollwöck, Phys. Rev. Lett. **97**, 050402 (2006).
113. Roman Orus, Guifre Vidal, Phys. Rev. B **78**, 155117 (2008).
114. I. P. McCulloch, arXiv:0804.2509.
115. B. Pirvu, F. Verstraete, G. Vidal, Phys. Rev. B **83**, 125104 (2011).
116. M. C. Bañuls, M. B. Hastings, F. Verstraete, J. I. Cirac, Phys. Rev. Lett. **102**, 240603 (2009).
117. I. Pizorn, L. Wang, and F. Verstraete, Phys. Rev. A **83**, 052321 (2011).
118. M. B. Hastings, J. Math. Phys. **50**, 095207 (2009).
119. Norbert Schuch, Michael M. Wolf, Frank Verstraete, J. Ignacio Cirac, Phys. Rev. Lett. **98**, 140506 (2007).
120. Norbert Schuch, Ignacio Cirac, Frank Verstraete, Phys. Rev. Lett. **100**, 250501 (2008).
121. F. Verstraete, J.I. Cirac,Phys.Rev.Lett.**104**, 190405 (2010);
122. Guifre Vidal, Phys. Rev. Lett.**99**, 220405 (2007); G. Evenbly, G. Vidal, Phys. Rev. B **79**, 144108 (2009).
123. Glen Evenbly, Guifre Vidal, Phys. Rev. Lett. **102**, 180406 (2009).
124. Guifre Vidal, chapter of the book *"Understanding Quantum Phase Transitions"*, edited by Lincoln D. Carr, Taylor & Francis, Boca Raton (2010).
125. P. Corboz, S. R. White, G. Vidal, M. Troyer, arXiv:1104.5463
126. Simeng Yan, David A. Huse, and Steven R. White, Science **332** 1173 (2011).
127. Zheng-Cheng Gu, Michael Levin, Xiao-Gang Wen, Phys. Rev. B **78**, 205116 (2008).
128. Z. Y. Xie, H. C. Jiang, Q. N. Chen, Z. Y. Weng, T. Xiang, Phys.Rev.Lett. **103** 160601 (2009).
129. H.H. Zhao, Z.Y. Xie, Q.N. Chen, Z.C. Wei, J.W. Cai, T. Xiang, Physical Review B **81**, 174411 (2010).