Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea Magistrale in Fisica

AN APPLICATION OF TROTTER'S ALGORITHM TO DMRG SIMULATION

Relatore:

Prof.essa Elisa Ercolessi

Presentata da:

Andrea Zocca

Correlatore:

Prof. Fabio Ortolani

Anno Accademico 2016/2017

Contents

Abstract **iii**

Introduction **v**

1 Density matrix renormalization group **1**

1.1 The density matrix 2

1.2 DMRG 5

1.2.1 Infinite size algorithm 9

1.2.2 Finite Size Algorithm 16

1.3 White's Prediction 19

1.4 Exact Diagonalization Algorithm 22

1.4.1 The Lanczos Algorithm 22

1.4.2 Thick restart method 27

2 Time Evolution DMRG **31**

2.1 Time Step Targetting 32

2.2 Time Evolving Block Decimation 34

2.3 Trotter vs R-K: a quick comparison 38

3 Coding the DMRG **41**

3.1 Implementation of Trotter's Algorithm 41

3.2 Trotter 43

3.3 Evolve 50

3.4 Trotterstep 53

4 Numerical Results **57**
4.1 An Exact solved system: Free spinless Fermions with open boundaries . . 57
4.2 Nearest neighbour approximation 60
4.3 Error Analysis 64

Conclusion **69**

Acknowledgments **71**

Bibliography **76**

Abstract

La seguente tesi presenta l'implementazione in codice C++ di un algoritmo per l'evoluzione temporale del DMRG, basato sull'approssimazione di Trotter per primi vicini. Il corretto funzionamento del codice è stato controllato calcolando un ristema risolubile esattamente, la catena di fermioni liberi senza spin (con condizioni al contorno aperte); in seguito il nuovo algoritmo è stato comparato con l'evoluzione temporale del DMRG basata sul metodo di Runge-Kutta. L'analisi degli errori ha mostrato come, per brevi periodi di tempo, il metodo di Runge-Kutta sia il più adatto fra i due, mentre per periodi di media durata il metodo di Trotter offra prestazioni migliori. Le evoluzioni temporali per tempi elevati sono attualmente al di là della portata di entrambi gli algoritmi.

Introduction

Decades of research in the field of condensed matter has conducted the physics to develop a variety of many-body models. These models are actually essential to describe the gigantic number of electrons and nucleons contained into the smallest sample of matter, because they gave the opportunity to describe different system with a relatively simple mathematical construction and deep physical meaning. The development of powerful experimental techniques enhanced the interest in this theoretical field, because by means of, for example, cold atoms techniques, it is possible now to recreate controlled low dimensional system which exhibit genuinely quantum properties. We are observing a quite amusing role reversing into the relationship between theoretical and experimental physics: while in the past simplyfied models were used to describe the low-energy physics of more complex systems, nowadays these models can be artificially recreated and their properties broadly investigated.

However, the simulation of quantum mechanics is still today a very challenging problem. Suppose we have a generic quantum system, with an Hamiltonian H. In order to performe a numerical analysis we have to discretize the problem and implement it on the computer. The amount of memory required for this purpose grows exponentially with the system size L, because the Hilbert space's dimension generally increases according to a relation of the form $dimH \propto a^L$. Then, we have to store in memory an $a^L \times a^L$ matrix for each system's observable, but these objects rapidly seize all the available memory! To have a concrete idea of the problem, think that to store a single state of a spin-1/2 chain of L sites, about 4 Terabyte are required. These limitations became more pronounced during the calculation of time evolution, which requires an exponentiation of these matrices.

To circumvent these difficulties in this thesis work we make use of the *Density matrix renormalization group* or DMRG, a numerical renormalization group (RG) method aimed

at obtaining a good variational approximate solution of a general many-body problem defined on a lattice. In a few words, this algorithm analyze the eigenvalue of the system's density matrix and discard all the states which have a low or near zero probability, reducing substantially the memory required for every matrix evaluation.

In particular, we implemented, in C++ language, a time evolution (t-DMRG) algorithm based on the Trotter expansion, a unitary time evolving operator particular suitable for systems in which the *nearest neighbour approximation* is valid.

Chapter 1

Density matrix renormalization group

The density matrix renormalization group (DMRG) was proposed by S.R. White [1][2][3] as a numerical renormalization group (RG) method aimed at obtaining a good variational approximate solution of a general many-body problem defined on a lattice. It can handle systems whose number of degree of freedom makes exact diagonalization impossible and isn't affected by the sign problem encountered in Quantum Monte Carlo calculations.

Originally, DMRG was employed for the study of spin chains[4], but it has subsequently been applied to systems containing phonons[5], to two dimensional classic system[6][7][8], to quantum chemistry[9], in momentum space[10], at finite temperature [11][12][13], to disordered systems[14] just to to quote some of the most diverse areas of application.

As far as the role of dimensionality is concerned, DMRG has been extended to two-dimensional quantum systems [15] and Bethe lattice[16][17], which can be considered as infinite-dimensional.

DMRG is based on Wilson's RG method, which was successfully used to treat the Kondo impurity problem[18], but was not as successful in handling Heisenberg and Hubbard-like hamiltonians. In conventional RG, the diagonalization of a system with an enormous number of degrees of freedom is performed in many steps. The hamiltonian is first diagonalized in a basis that describes a subset of the final system (only some sites on a lattice for real-space RG, or states within a shell of momentum values, in momentum-

space RG). Then a "decimation" of the states of the subset is performed and one keeps only the lowest-lying eigenstates of the hamiltonian. In the next step the hamiltonian is diagonalized in a truncated basis that contains these eigenstates plus some new degrees of freedom of the final system.

This method fail when applied to quantum system located on a lattice, like the Hubbard model, because of boundary conditions. The states chose as lowest energy eigenstates doesn't have the right features at lattice blocks extremities. The block's isolation impose the annulment of the wavefunction at the boundaries, and that doesn't allow a good description when we connect the block to another one to assemble a bigger block. We would need a larger amount of states to improve the description, losing efficiency.

The seemingly natural choice of the lowest-lying eigenstates is abandoned in DMRG. Since one is interested in describing correctly the final system rather than some fraction of it, one would like to choose states that have the maximum probability of representing a part of the system *interacting* with the remainder. The mathematical tool that gives us this information is the density matrix.

DMRG improves over conventional Rg at the price of diagonalizing a bigger physical system, conventionally called *superblock*, at each renormalization step. This superblock is composed by the *system* for which one wants to obtain an approximate basis, plus an *environment* or *universe* that provides the proper boundary conditions for the systems.

From the diagonalization of the superblock hamiltonian one obtains a wavefunction, represented in a basis of tensor product states, with one index referring to states of the system and the other referring to states of the environment.

The criterion for selecting the most relevant states of the system interacting with the environment is then provided by the magnitude of the eigenvalues of the system's density matrix, which is calculated by tracing over the states of the environment in the superblock ground state wavefunction.

1.1 The density matrix

The resolution of a quantum mechanical problem usually began with the partition, in two parts, of the universe: the *System* we want to observe, and everything else, a part we'll name *Environment*. Let's consider a system that's part of a closed system, and suppose

that the entire closed system is in a state described by a wavefunction $\psi(x, y)$ where x are the System's coordinate and y the remaining coordinates of the closed system. The wavefunction $\psi(x, y)$ isn't, except particular cases, factorable in a product of two function of, respectively, x and y, so the system doesn't have a proper wavefunction. The most general wafunction for the whole system can be written as:

$$\psi(x, y) = \sum_i E_i(y)\phi_i(x) \tag{1.1}$$

Introducing the Dirac's notation, we'll indicate with $|i\rangle$ a complet set of vector in the Hilbert space of the System ($\phi_i(x) = \langle x|i\rangle$) and with a complete set for the Environment ($\theta_j(y) = \langle y|j\rangle$). The general wavefunction will be:

$$|\psi\rangle = \sum_{ij} \psi_{ij} |i\rangle |j\rangle \tag{1.2}$$

If A is an operator acting only on the System, that is on $|i\rangle$ states, tha action of A on the state $|\psi\rangle = \sum_{ij} \psi_{ij} |i\rangle |j\rangle$ won't affect the $|j\rangle$, so:

$$\begin{aligned} \langle A\rangle = \langle\psi| A |\psi\rangle &= \sum_{iji'j'} \psi^*_{ij}\psi_{i'j'} \langle j| \langle i| A |i'\rangle |j'\rangle \\ &= \sum_{iji'} \psi^*_{ij}\psi_{i'j} \langle i| A |i'\rangle = \sum_{ii'} \rho^S_{ii'} \langle i| A |i'\rangle \end{aligned} \tag{1.3}$$

where we defined the *reduced density matrix* as

$$\rho^S_{ii'} = \sum_j \psi^*_{ij}\psi_{i'j} \tag{1.4}$$

The operator ρ is defined through its matrix elements $\rho_{ii'} = \langle i| A |i'\rangle$ and act only on the System, which is described by the cordinates x. We can now write operator A's mean value as:

$$\begin{aligned} \langle\psi| A |\psi\rangle &= \sum_i \langle i| A \sum_{i'} |i'\rangle \langle i'| \rho |i\rangle \\ &= \sum_i \langle i| A\rho |i'\rangle = \mathrm{Tr}(\rho A) \end{aligned} \tag{1.5}$$

Our dissertation until now considered only a special case, in which operator A acted only on the System. If the entire system, System+Environment, can be described by a *pure state* $|\psi\rangle = \sum_i C_i |i\rangle$, then we can define the *density matrix* as

$$\mathrm{P} = |\psi\rangle \langle\psi| \tag{1.6}$$

or equivalently

$$\mathrm{P}_{ii'} = C_i C_{i'}^* \tag{1.7}$$

The operator A mean value consequently became

$$\langle\psi| A |\psi\rangle = \mathrm{Tr}(|\psi\rangle \langle\psi| A) = \mathrm{Tr}(\mathrm{P}A) \tag{1.8}$$

With the density matrix it's possible to evaluate the mean value of any system's observable. Even if a system doesn't have a wavefunction, it nonetheless can be described by a density matrix, altough it dependes by the whole systems's coordinates x and y. The operator P is the most general instrument we can use to study a quantum-mechanical system; the wavefunction description, in effect, is just a particular case corresponding to a density matrix in the form $\mathrm{P}_{ii'} = C_i^* C_{i'}$

From equation (1.4) we note that ρ is hermitian, so the operator can be diagonalized with a complete orthonormal set of eigenvectors $|u_\gamma\rangle$ with real eigenvalues. Moreover, the system's Hilbert space can be described with the density matrix eigenfunctions.

Every quantum-mechanical system can therefore be described by a density matrix

$$\rho = \sum_\gamma w_\gamma |u_\gamma\rangle \langle u_\gamma| \tag{1.9}$$

that satisfy the following properties:

1. the set of eigenfunctions $|u_\gamma\rangle$ is orthonormal and complete

2. $w_\gamma \geq 0$

3. $\sum_\gamma w_\gamma = 1$

4. the expectation value of an operator A acting on the system is

$$\langle A \rangle = \mathrm{Tr}(\rho A) \tag{1.10}$$

If we explicit the equation of point 4, the mean value of operator A is expressed by:

$$\begin{aligned} \langle A \rangle = \mathrm{Tr}\langle \rho A \rangle &= \sum_{\gamma'} \langle u_{\gamma'} | \, \rho A \, | u_{\gamma'} \rangle = \sum_{\gamma\gamma'} w_{\gamma'} \, \langle u_{\gamma'} | u_{\gamma} \rangle \, \langle u_{\gamma} | \, A \, | u_{\gamma'} \rangle \\ &= \sum_{\gamma} w_{\gamma} \, \langle u_{\gamma} | \, A \, | u_{\gamma} \rangle \end{aligned} \tag{1.11}$$

because $\langle u_\gamma | \, A \, | u_\gamma \rangle$ is the expectation value of operator A on state $|u_\gamma\rangle$, it's possible to interpret the quantity w_γ as the probability to find the system in the state $|u_\gamma\rangle$. If all coefficients are null, except the i-th, the system will be in a *pure state*, otherwise the system is in a *mixed state*.

1.2 DMRG

Let us consider a quantum system in a well defined state. We denote with **Superblock** the whole system, with **System** the part of the superblock we're intrested in, and with **Environment** everything else.

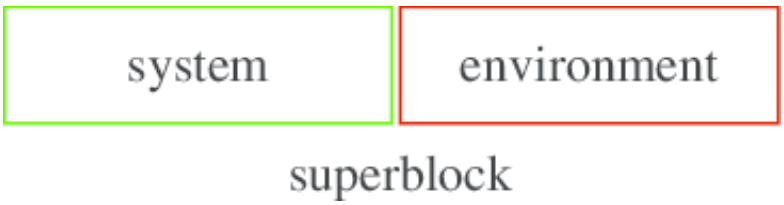


Figure 1.1: Schematic representation of a Superblock divided into a System block and an Environment block

The DMRG method consist in the construction of the density matrix for the System block, considering it as superblock's part.

Let $|\psi\rangle = \sum_{ij} |i\rangle \, |j\rangle$ be a vector of the Hilbert space, representing the Superblock's state ψ, in which the vectors $|i\rangle$ and $|j\rangle$ label, respectively, the System states and the

Environment states. The reduceed density matrix related to the System block will be

$$\rho_{ii'} = \sum_j \psi^*_{ij}\psi_{ij} \tag{1.12}$$

Once we diagonalize the density matrix, the mean value of any operator A acting on System block, will be evaluated by equation (1.11).

The renormalization routine can be directly applied to equation (1.11). Let's presume we want to discard some System states; if, for a given γ, the corresponding eigenvalue is $w_\gamma \approx 0$, to neglect the state $|u_\gamma\rangle$ won't entail a significative error in the mean value of $\langle A\rangle$. Since $Tr(\psi\psi^T) = \sum_\gamma w_\gamma = 1$, this approximation is good if the probabilities w_γ have a sufficiently rapid decrease to zero, so that $\sum_\gamma w_\gamma \approx 1$. At the best of our knowledge, all numerical experiments performed so far (see, e.g. ref[19]) confirm this rapid decrease of the probabilities w_γ. The problem then became how decide which states have to be kept and which ones have to be discarded.

More exactly, it's possible to demonstrate that the most probable state of ρ gives the most accurated representation for the Superblock state. Assume we can diagonalize the Superblock's Hamiltonian, and know consequently a particular state $|\psi\rangle$ (usually, the ground state). From now on, we have to find a states' set of the System block $|u_\gamma\rangle$ with $\gamma = 1, ..., m$ and $|u_\gamma\rangle = \sum_i u_i^\gamma |i\rangle$, that provides a good approximated representation of ψ.

There's a major limitation; $|u_\gamma\rangle$ is a finite set and, usually, the states' number m is smaller than l, the System's number of states, so the best we can do is to create a vector $|\bar{\psi}\rangle$ that is:

$$|\psi\rangle \approx |\bar{\psi}\rangle = \sum_{\gamma,j} a_{\gamma,j} |u_\gamma\rangle |j\rangle = \sum_\gamma a_\gamma |u_\gamma\rangle |v_\gamma\rangle \tag{1.13}$$

where in the last passage we sat $\sum_j a_{\gamma,j} |j\rangle = a_\gamma |v_\gamma\rangle$, so that $v_j^\gamma = \langle j|v_\gamma\rangle = N_\gamma a_{\gamma,j}$, with N_γ choose in order that $\sum_j |v_j^\alpha|^2 = 1$.

To improve the truncation we have to minimize the quantity:

$$S = | \, |\psi\rangle - |\bar{\psi}\rangle \, |^2 \tag{1.14}$$

and passing to matrix notation, for a given m,

$$S = \sum_{i,j} (\psi_{i,j} - \sum_{\gamma=1}^{m} a_\gamma u_i^\gamma v_j^\gamma)^2 \tag{1.15}$$

The solution to this minimization problem is known from linear algebra, and use the following singular value decomposition theorem[20]:

Theorem 1.2.1 (***Singular Value Decomposition***). *Let Ψ be an $l \times n$ matrix $\Rightarrow \exists$ Y (orthogonal and $l \times l$), W(column orthogonal and $l \times n$) and D (diagonal and $l \times l$) matrices, where $l \leq n$, such that*

$$\Psi = YDW^T \tag{1.16}$$

If we think about the density matrix coefficients $\psi_{i,j}$ as the elements of a rectangular matrix Ψ of $l \times n$ dimension, it's possible to apply theorem (1.2.1) and write the element $\psi_{i,j}$ as

$$\psi_{i,j} = \sum_{k=1}^{l} y_i^k d_k w_j^k \tag{1.17}$$

and equation (1.15) became

$$\begin{aligned} S &= \sum_{i,j} (\sum_{k=1}^{l} y_i^k d_k w_j^k - \sum_{\gamma=1}^{m} a_\gamma u_i^\gamma v_j^\gamma)^2 \\ &= || \sum_{k=1}^{l} d_k \left|y^k\right\rangle \left|w^k\right\rangle - \sum_{\gamma=1}^{m} a_\gamma \left|u_\gamma\right\rangle \left|v_\gamma\right\rangle ||^2 \end{aligned} \tag{1.18}$$

Writing explicitly the matrix elements $\Psi\Psi^\dagger$

$$(\Psi\Psi^\dagger)_{ii'} = \sum_j \psi_{i,j}\psi_{i',j} \tag{1.19}$$

we note that equation (1.19) correspond to the density matrix definition we gave in equation (1.12), when the elements $\psi_{i,j}$ are real. The Unitary matrix Y therefore diagonalize ρ^S and the l states $\left|y^k\right\rangle$ are eigenfunctions of ρ^S, just as the m states $\left|u_\gamma\right\rangle$, that is:

$$\{\left|u_1\right\rangle, ..., \left|u_m\right\rangle\} \subseteq \{\left|y_1\right\rangle, ..., \left|y_l\right\rangle\} \tag{1.20}$$

Now we just have to choose the m best functions between the $|y_l\rangle$. If we order the $|y_l\rangle$ set, in a way that $|y_1\rangle = |u_k\rangle, \forall k \leq m$ it's possible to rewrite S as:

$$S = || \sum_{k=1}^{l} |u_k\rangle \left(d_k \left|w^k\right\rangle - a_k |v_k\rangle\right)||^2 \tag{1.21}$$

where $a_k, |v_k\rangle = 0, \forall k \geq m$. Expanding upon the norm, we have:

$$\begin{aligned} S &= \sum_{k,l} [|u_k\rangle \left(d_k \left|w^k\right\rangle - a_k |v_k\rangle\right)]^\dagger [|u_l\rangle \left(d_l \left|w^l\right\rangle - a_l |v_l\rangle\right)] \\ &= \sum_{k,l} \delta_{k,l} [d_k d_l \langle w_k|w_l\rangle - a_k d_l \langle v_k|w_l\rangle - a_l d_k \langle v_k|w_l\rangle + a_k a_l \langle v_k|v_l\rangle] \\ &= \sum_{k} (d_k^2 + a_k^2 - 2 a_k d_k \langle u_k|w_l\rangle) \\ &= \sum_{k=1}^{l} ||d_k |w_k\rangle - a_k |u_k\rangle||^2 \\ &= \sum_{k=1}^{m} ||d_k |w_k\rangle - a_k |u_k\rangle||^2 + \sum_{k=m+1}^{l} d_k^2 \end{aligned} \tag{1.22}$$

The first sum is a defined positive quantity, and became minimum null when

$$a_k = d_k \qquad |w_k\rangle = |u_k\rangle \qquad \forall k \leq m, \tag{1.23}$$

the second term became minimum when values $d_k, k > m$ take on the smallest modulus.

Summarizing, the density matrix eigenvalue w_γ are the coefficients a_k^2 and the best describing states of the System $|u_k\rangle$ are the eigenstate of ρ corresponding to the higher eigenvalues. Every coefficient w_k represent the block's probability being in the state $|u_k\rangle$, with $\sum_k w_k = 1$. The variance between 1 and

$$p_m = \sum_{k=1}^{m} w_k \tag{1.24}$$

provides a measure of the truncation's (to m states) goodness.

Let's now proceed with the desciption of the DMRG method. It consists of two parts: in the infinite size algorithm one progressively enlarges the superblock, keeping $M = N$

up to reach the condition $M + N = L$, while in the finite size algorithm the size of the superblock is kept fixed and M, N are varied at each step.

1.2.1 Infinite size algorithm

Let the superblock be a chain of L sites and let α_M and β_N denote two subsets of respectively M, N sites, that is, the System and the Environment. A partition of the system will generally result in a mixed entangled states, so the ground state wavefunction of N electrons can be expressed in the following form:

$$|\psi\rangle = \sum_{\alpha_M}^{N_S} \sum_{\beta_N}^{N_E} \psi_{\alpha_M,\beta_N} |\alpha_M\rangle |\beta_N\rangle \tag{1.25}$$

or equivalently

$$|\psi\rangle = \sum_{\alpha_M}^{N_S} \sum_{\beta_N}^{N_E} \psi_{\alpha_M,\beta_N} \hat{A}_{\alpha_M} \hat{A}_{\beta_N} |g\rangle \tag{1.26}$$

where $|\alpha_M\rangle$ and $|\beta_N\rangle$ label, respectively, the states of a Sistem block of length M and the ones of an Environment block of lenght N, $\hat{A}_{\alpha_M}, \hat{A}_{\beta_N}$ are generic excitation operators for states in α, β, respectively and $|g\rangle$ denote a reference state.

These two notation are related by the convention $|\alpha_M\rangle = \hat{A}_{\alpha_M} |0\rangle$, $|\beta_N\rangle = \hat{A}_{\beta_N} |0\rangle$; we'll also make use of the expression $|\alpha_M\rangle |\beta_N\rangle$ to denote the compound state $\hat{A}_{\alpha_M} \hat{A}_{\beta_N} |0\rangle$ (this state is similar but not identical to the tensor product $|\alpha_M\rangle \otimes |\beta_N\rangle$, since the operators $\hat{A}_{\alpha_M}, \hat{A}_{\beta_N}$ don't necessarily commute). Clearly, varying the polinomials $\hat{A}_{\alpha_M}, \hat{A}_{\beta_N}$ in all possible independent ways, the states $|\alpha_M\rangle |\beta_N\rangle$ generate the whole Hilbert space. In principle the sums in α_M, β_N run over g^M, g^N states respectively. For example, an electron system may have 4^N states since the occupations numbers $n_\uparrow$, $n_\downarrow$ of a site can have four possible values: (0,0),(1,0),(0,1),(1,1). For a spinless fermion on a lattice site, the states are 2^N. However the number of spin up and spin down fermions are good quantum numbers and can be fixed; we can choose states $\hat{A}_{\alpha_M} |g\rangle$, $\hat{A}_{\beta_N} |g\rangle$, with fixed numbers of spin up and spin down fermions, and the coefficients ψ_{α_M,β_N} vanish unless this conservation law is fullfilled. Furthermore, during the iteration procedure, the number of states will be truncated; therefore in the wavefunction's expansion we'll keep in general only N_S states for the system block and N_E states for the environment block.

We stress out that states $|\alpha_M\rangle, |\beta_N\rangle$ are eigenstate of the density matrix. The reduced density matrix relative to System's block is defined as:

$$\rho_{\alpha_M \alpha'_M} = \sum_{\beta_N}^{N_E} \psi_{\alpha_M,\beta_N} \psi_{\alpha'_M,\beta_N} = (\psi\psi^T)_{\alpha_M \alpha'_M} \tag{1.27}$$

The dimension of the matrix ρ is $N_S \times N_E$; however, because of the number conservation laws described above, tha matrix is actually in block form, that is, the number of up and down fermions of the states α_M and α'_M must be the same. The trace of ρ equals unity due to wavefunction normalization.

The infinite size algorithm is generally used either to get information on the thermodynamic limit of a physical system or as a first step in finite size algorithm. We start from a basis of N_S states $|\alpha_m\rangle$ that describe the block (of length m) S and N_E states $|\beta_n\rangle$ that describes the block (of length n) E.

At the beginning these states are generally taken as single lattice sites, so that's easy to build the superblock Hamiltonian, diagonalize it with the Lanczos method and obtain the wavefunction ground state. After we create the density matrix, diagonalize it and find its eigenvalues and eigenvectors, maintaining only the largest N_S states. Then we have to find a matrix representation of operator related to the compound state $S + E$ starting from the operators defined for each block.

Next task consist of enlarging the blocks. In the infinite system method, since $M = N$ and the system we consider is translationally or reflection invariant, the states $|\beta_n\rangle$ can be simply obtained by translating or reflecting the states $|\alpha_m\rangle$. Hence, we can concentrate our attention on the block S.

The simplest way of enlarging the block S consist of adding a site σ to S, obtaining a new block $S' = S \cup \sigma$, denoted $S\bullet$ by White[1]:

$$\hat{A}_{\alpha_{m+1}} |g\rangle = |\alpha_{m+1}\rangle = \sum_{\alpha_m,\sigma} |\alpha_m\rangle |\sigma\rangle B^S_{m+1\ \alpha_m\sigma;\alpha_{m+1}} \tag{1.28}$$

where

$$\alpha_m = 1, ..., N_S \qquad \sigma = 1, ..., r \qquad \alpha_{m+1} = \alpha_m\sigma \tag{1.29}$$

in order to describe $S' = S\bullet = S \cup \sigma$. We define, inverting equation (1.28), the base

transformation matrix

$$B^{S}_{m+1\ \alpha_m\sigma;\alpha_{m+1}} = \langle \alpha_m, \sigma | \alpha_{m+1} \rangle \tag{1.30}$$

which express in the new base $|\alpha_{m+1}\rangle$, the enlarged system $|\alpha_m\rangle\,|\sigma\rangle$.

At the same time, we add an analogous site λ to the block E. If we want to use translational invariance we consider the vectors $|\beta_n\rangle\,|\lambda\rangle$ $(\beta_n = 1, ..., N_E, \lambda = 1, ..., g)$ in order to describe the block $E' = E\bullet = E \cup \lambda$. For reflection invariance we shall build instead $E' = \lambda \cup E$. With such a basis we can now proceed to compute the expansion (1.26) for the wavefunction relative to the new superblock $S' \cup E'$.

In order to build the Hamiltonian in this basis we need to separate it into parts that belong to each one of the four blocks. The Hamiltonian we'll consider contains a tight binding hopping term plus a long range density-density operator. The latter term can be reorganized most conveniently as follows:

$$\begin{aligned} \hat{V} = \sum_{\mu,\nu}^{L} \hat{O}^{\dagger}_{\mu} V_{\mu,\nu} \hat{O}_{\nu} &= \sum_{r=1}^{4} \sum_{s=1}^{4} \sum_{\mu_r} \sum_{\nu_s} \hat{O}^{\dagger}_{\mu_r} V_{\mu_r,\nu_s} \hat{O}_{\nu_s} \\ &= \sum_{r=1}^{4} \hat{M}_r + \sum_{r=1}^{4} \sum_{\mu_r} \hat{O}^{\dagger}_{\mu_r} \sum_{s\neq r} \hat{Q}_{r,\mu_r,s} \end{aligned} \tag{1.31}$$

where μ_r run over the sites belonging to block r and the operators $\hat{M}$ contain products of operators all internal to each of the four blocks:

$$\hat{M}_r = \sum_{\mu_r} \sum_{\nu_r} \hat{O}^{\dagger}_{\mu_r} V_{\mu_r,\nu_r} \hat{O}_{\nu_r} \tag{1.32}$$

while the operators $\hat{Q}$ are sums of single operators internal to each of the four blocks:

$$\hat{Q}_{r,\mu_r,s} = \sum_{\nu_s} V_{\mu_r,\nu_s} \hat{O}_{\nu_s} \qquad s \neq r \tag{1.33}$$

For a density-density interaction, the operator $\hat{O}$ is just the number operator. Remember that, at the beginning all of these operators are known exactly because the blocks are taken to be single site.

We now look for the ground state vector ψ of the truncated Hamiltonian H by using Lanczos' algorithm. The density matrix $\psi\psi^T$ and new state vectors $|\alpha_{m+1}\rangle$, that

represent S''s states now reads:

$$\begin{aligned} |\alpha_{m+1}\rangle &= \sum_{\alpha_m}^{N_S} \sum_{\sigma} B^S_{m+1\ \alpha_m\sigma;\alpha_{m+1}} |\alpha_m\rangle |\sigma\rangle \\ &= \sum_{\alpha_m}^{N_S} \sum_{\sigma} B^S_{m+1\ \alpha_m\sigma;\alpha_{m+1}} \hat{A}_{\alpha_{m+1}} |g\rangle \qquad \alpha_{m+1} = 1, ..., N_S^+ \end{aligned} \tag{1.34}$$

Again we do not keep all the vectors: N_S^+ is generally less than σN_S and often one puts $N_S^+ = N_S$, altough this choice is not necessary. The corresponding vector $|\beta_{n+1}\rangle$ that describe E' are obtained from the $|\alpha_{m+1}\rangle$ by translation or reflection.

In this new truncated basis we compute the matrix elements of all the operators needed to build the Hamiltonian at the next step, namely: all the operators $\hat{P}$, all the operators $\hat{O}_\mu$ and the operators $a_{\mu,\varsigma}$ at the boundaries of each block. If, for example, we have an operator $\hat{O}$ internal to block S, it is also internal to the new block S' and we have the following rule to update its matrix elements:

$$\begin{aligned} \langle\alpha_{m+1}| O |\alpha_{m+1}\rangle &= \sum_{\alpha_m,\sigma} \sum_{\alpha'_m,\sigma'} B^{S\dagger}_{m+1\ \alpha_m\sigma;\alpha_{m+1}} \langle g| \hat{A}^\dagger_{\alpha_{m+1}} O \hat{A}_{\alpha_{m+1}} |g\rangle B^S_{m+1\ \alpha'_m\sigma';\alpha'_{m+1}} \\ &= \sum_{\alpha_m,\sigma} \sum_{\alpha'_m,\sigma'} B^{S\dagger}_{m+1\ \alpha_m\sigma;\alpha_{m+1}} \langle\alpha_m| O |\alpha'_m\rangle \langle\sigma|\sigma'\rangle B^S_{m+1\ \alpha'_m\sigma';\alpha'_{m+1}} \\ &= \sum_{\alpha_m,\sigma} \sum_{\alpha'_m} B^{S\dagger}_{m+1\ \alpha_m\sigma;\alpha_{m+1}} \langle\alpha_m| O |\alpha'_m\rangle B^S_{m+1\ \alpha'_m\sigma;\alpha'_{m+1}} \end{aligned} \tag{1.35}$$

where

$$\alpha_{m+1}, \alpha'_{m+1} = 1, ..., N_S^+$$

.

If the operator isn't internal to the System in all of its terms, but has some on the new added site, the expression for $\langle\alpha_{m+1}| O |\alpha_{m+1}\rangle$ need an additional term. Assuming the operator $\hat{O}$ be a product of operator acting on the System and the new block

$$\hat{O} = \hat{O}_S \hat{O}_1 \tag{1.36}$$

we have

$$\langle \alpha_{m+1}| O |\alpha_{m+1}\rangle = \sum_{\alpha_m,\sigma} \sum_{\alpha'_m,\sigma'} B^{S\dagger}_{m+1\ \alpha_m\sigma;\alpha_{m+1}} \langle \alpha_m| O_S |\alpha'_m\rangle \langle \sigma| O_1 |\sigma'\rangle B^{S}_{m+1\ \alpha'_m\sigma';\alpha'_{m+1}} \tag{1.37}$$

Two more sites are added to the blocks S', E', giving rise to new blocks $S'' = S'\bullet$, $E'' = E'\bullet$ etc. By the systematic procedure of adding two more sites, truncating the basis and updating the hamiltonian matrix at each iteration, systems of large size can be handled.

A comment in in order about the choice of the two sites that are added and their position with respect to blocks S and E. We can form the superblock $S \bullet E\bullet$ or the superblock $S \bullet \bullet E$. White suggestes that the enlarged configuration $S \bullet E\bullet$ is to be preferred to $S \bullet \bullet E$ in case of periodic boundary conditions, the opposite holds in case of open boundary conditions.

In fact the blocks S and E are separated by the site λ in the case while they become adjacent by periodicity in $S \bullet \bullet E$. The kinetic part of the Hamiltonian "connects" two blocks only by its border sites, with operators whose matrix elements are known. These matrices are "big" for blocks S and E, ad "little" for the 1-site blocks σ and λ, so the matrix elements of the Hamiltonian H' are simpler when a "big" block is surrounded by 1-site blocks.

The infinite system algorithm is stopped when the number of sites of $S \cup E$ reaches the total number L of sites.

It was shown by Ostlund and Rommer[21] that the infinite size algorithm in the thermodynamic limit gives a state that is equivalent to a matrix product state.

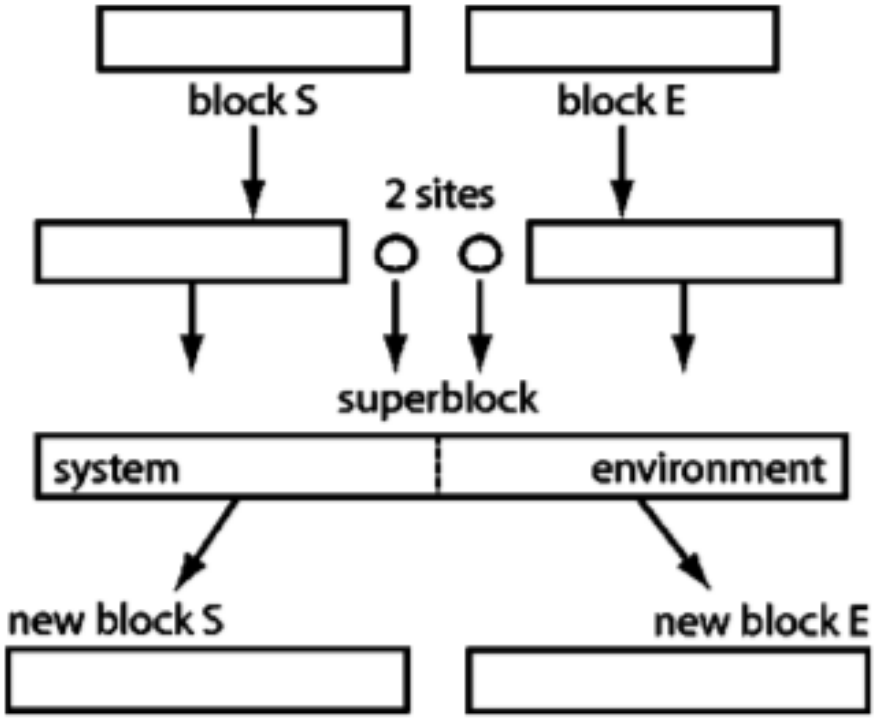


Figure 1.2: The Infinite System Algorithm

Table 1.1: Iterative scheme of the infinite algorithm

1. Build the first initial four blocks $S \bullet \bullet E$, each one describing a single site, and define the matrix representing the block Hamiltonian and the other operators.
2. Write the matrix of the Superblock's Hamiltonian
3. Diagonalize the Superblock's Hamiltonian with the Lanczos method and obtain the *target state* $\psi(\alpha_n, \sigma, \lambda, \beta_m)$
4. Build the density matrix for the 2-Block systems

$$\rho(\alpha_m, \sigma : \alpha'_m, \sigma') = \sum_{\lambda, \beta_n} \psi(\alpha_n, \sigma, \lambda, \beta_n)\psi(\alpha'_n, \sigma', \lambda', \beta'_n) \tag{1.38}$$

5. Diagonalize ρ and find its eigenvalues w_γ and eigenvectors $u^\gamma_{\alpha_n,\sigma}$. Store only the m highest eigenvalues and their eigenvectors.
6. Search for a matrix representation for the operators related to the system composed by α_n and σ, starting from the operators defined for each blocks.
7. Renormalize all relevant operators using the base of ρ's m eigenstates:

$$\hat{H}_{m+1} = B^S_{m+1} \hat{H}_m (B^S_{m+1})^\dagger \tag{1.39}$$

8. Substitute H_n with the reflected H_{m+1}
9. Start again from step 2

1.2.2 Finite Size Algorithm

In order to improve the accuracy of the method, White himself proposed a second algorithm, that we'll briefly describe. This second algorithm takes place after the infinite size algorithm reaches the end.

In the finite size algorithm, to an increase of S by one site corresponds a decrease of the environment E by one site. Denoting by S_m, E_n blocks S and E with m, n sites respectively, we start with the system $S_{L/2-1} \bullet E_{L/2-1}\bullet$ and we want to construct the systems $S_{L/2} \bullet E_{L/2-2}\bullet$, $S_{L/2+1} \bullet E_{L/2-3}\bullet$, etc. Therefore, in order to use the translational invariance, we need to keep in the computer memory all the relevant matrix elements of $S_{L/2-2}, S_{L/2-3}$, etc. in order to be able to use the simmetry and produce the matrix elements of $E_{L/2-2}, E_{L/2-3}$, etc. It should be notice that, when $N_E < N_S$, the rows of the new wavefunction ψ' cannot be linearly independent. As consequence, $\psi'\psi'^T$ has many eigenvalues equal to zero. If the system doesn't posses symmetry, the algorithm is the same, but we need to evaluate and store every block during the forward and backward zip.

The $N_S \times N_S$ matrix $\psi'\psi'^T$ and the smaller $N_E \times N_E$ one $\psi'^T\psi'$ have the same non vanishing eigenvalues. In practise, it's sufficient to diagonalize only the smallest of the two density matrices. The procedure stops when we reach the system $S_{L-3} \bullet E_1\bullet$, i.e. when the block E has reduced to a single site. We can now increase E and decrease S;

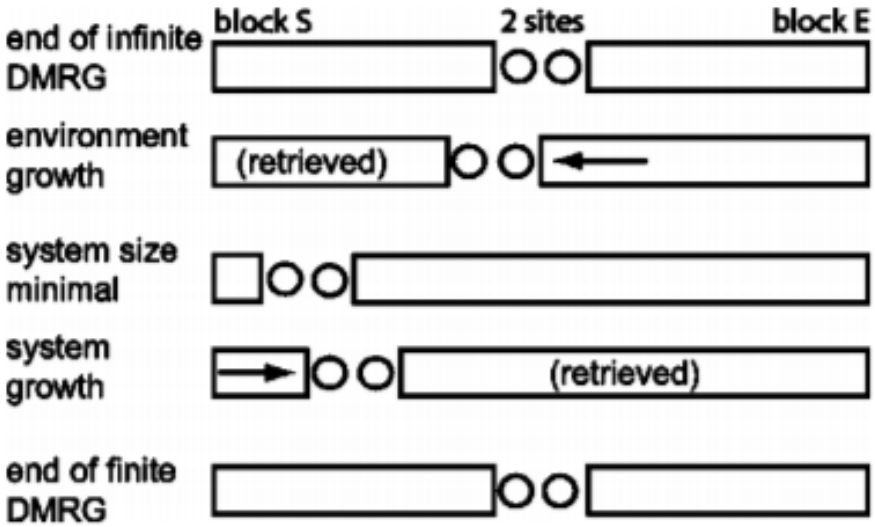


Figure 1.3: The Finite System algorithm

the subsystems S, E behave like if they were separated by a moving zipper. At every step we increase the accuracy of the states $|\alpha_m\rangle$ that describe the block S_m and after a few oscillations of the zipper all the blocks S_m with $2 \leq m \leq L-2$, accurately represent parts of a complete system of L sites, the remaining environment being the corresponding E_{L-m} block. Usually one stops when S and E have the same length.

The error[22] is an exponentially decreasing function of N. However, convergence is affected by many factors. The role of boundary conditions, for example, is very important. The density matrix eigenvalus decrease much faster in the case of open boundary conditions. So, if we have to impose periodicity, it will be necessary to keep more states to obtain the same accuracy that would be achieved with open boundaries.

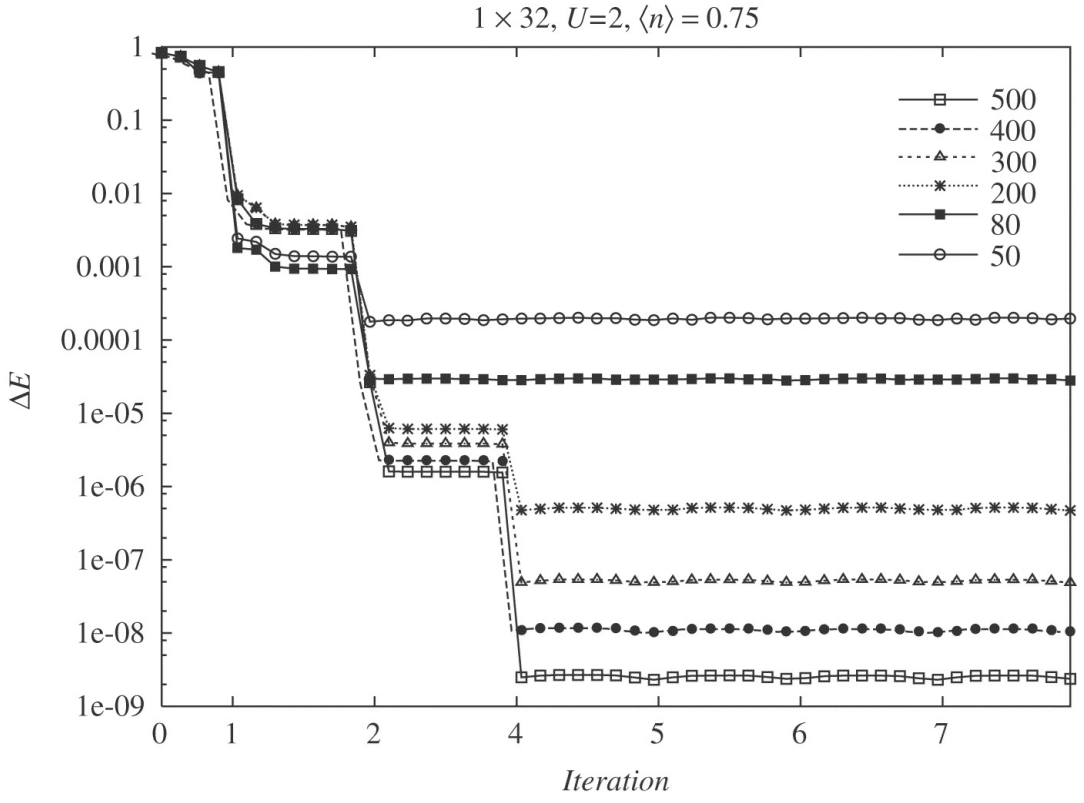


Figure 1.4: Difference between the ground-state energy obtained from the finite-system DMRG with different number of iterations and number of states kept m, and the exact energycalculated using Bethe Ansatz for a 32-site Hubbard model ($U = 2$ and filling $n = 3/4$). Reprinted from [23]

Table 1.2: Iterative scheme of the finite algorithm

1. Start from the last step of the Infinite system procedure, and take the Superblock of dimension L, composed by $S_{L/2-1} \bullet \bullet E_{L/2-1}$. From now on, we'll label the blocks as $S_l \bullet \bullet E_{L-l-2}$
2. Proceed with steps 2-8 of table (1.1)
3. Store the new block E_{L-l-1}, and substitute block S_l with S_{l-1}, calculated and stored during the infinite size algorithm. We obtain the configuration $S_{l-1} \bullet \bullet E_{L-l-1}$
4. If $l < L - 3 \Rightarrow l = l + 1$ and repeat again from step 2
5. When the configuration $S_1 \bullet \bullet E_{L-3}$ is reached, reverse the procedure, that is apply steps 2-8 of table (1.1), but store the growing System and overwrite the shrinking Environment
6. If $l < L - 3 \Rightarrow l = l + 1$ and repeat again from step 5
7. When the configuration $S_{L-3} \bullet \bullet E_1$ is reached, reverse the procedure, repeating steps 2-3 until $l < L/2 - 1$
8. We obtain the newly calculated configuration $S_{L/2-1} \bullet \bullet E_{L/2-1}$. This represent the end of a single iteration
9. Use the configuration from step 7 and repeat steps 2-7 until convergence (or the fixed number of iteration) is reached

1.3 White's Prediction

A major improvement of DMRG performance is provided by a wavefunction transformation proposed by White[24]. If one applies the finite system algorithm using the configuration $S \bullet \bullet E$, it's possible to obtain a good starting guess for the wavefunction from the wavefunction calculated at the previous step. In practice this makes it possible to iterate the finite-size algorithm many times, since at each step only a limited number of Lanczos iterations (less than 10) are necessary for convergence, while a random start may require more than a hundred Lanczos step.

This transformation is most easily implemented in the presence of reflection invariance. The general, non-reflection symmetric case is described by White[24]. Let $\left|\psi^L\right\rangle$ be the approximate ground state representing an L-sites system:

$$\begin{aligned} \left|\psi^L\right\rangle = & \sum_{\alpha_m,\sigma,\lambda,\beta_n} \psi^L_{\alpha_m,\sigma,\lambda,\beta_n} \left|\alpha_m\right\rangle \left|\sigma\right\rangle \left|\lambda\right\rangle \left|\beta_n\right\rangle \\ & L = m + n + 2 \end{aligned} \tag{1.40}$$

We want to obtain from this wavefunction a vector expressed in the following basis:

$$\left|\chi^L\right\rangle = \sum_{\alpha_{m+1},\sigma,\lambda,\beta_{n-1}} \chi^L_{\alpha_{m+1},\sigma,\lambda,\beta_{n-1}} \left|\alpha_{m+1}\right\rangle \left|\sigma\right\rangle \left|\lambda\right\rangle \left|\beta_{n-1}\right\rangle \tag{1.41}$$

If no truncation were involved, it would be possible to express ψ exactly in the new basis. Since during the finite-size algorithm the Hilbert space is changed, the transformation will only be approximate.

When the number of sites of the left part is increasing, we can project the odd state ψ onto the new basis obtained by diagonalization of the reduced density matrix:

$$\left|\alpha_{m+1}\right\rangle = \sum_{\alpha_m,\sigma} B^S_{m\ \alpha_m,\sigma;\alpha_{m+1}} \left|\alpha_m\right\rangle \left|\sigma\right\rangle \tag{1.42}$$

The projected state $\left|\phi\right\rangle = P\left|\psi\right\rangle$ is expressed as:

$$\left|\phi^L\right\rangle = \sum_{\lambda,\beta_n} \phi^L_{\alpha_{m+1},\lambda,\beta_n} \left|\alpha_{m+1}\right\rangle \left|\lambda\right\rangle \left|\beta_n\right\rangle \tag{1.43}$$

Its coefficients are obtained from those of ψ according to:

$$\phi^L_{\alpha_{m+1},\lambda,\beta_n} = B^S_{m+1\ \alpha'_{m+1},\lambda',\alpha_{m+1}} \psi^L_{\alpha'_{m+1},\lambda',\lambda,\beta_n} \tag{1.44}$$

based on the orthonormality of the basis transformation:

$$B_{L\ \alpha_{m+1},\lambda,\alpha_{m+2}} B_{L\ \alpha_{m+1},\lambda,\alpha'_{m+2}} = \delta_{\alpha_{m+2},\alpha'_{m+2}} \tag{1.45}$$

(note that this relation holds exactly, because truncation involves the indices $\alpha_{m+2}, \alpha'_{m+2}$, not α_{m+1}, λ).

The right part of the basis transformation can be related to the left part via reflection symmetry. Reflection symmetry ($\mathcal{R}$) is implemented by the requirement that:

$$|\beta_n\rangle = \mathcal{R}\,|\alpha_n\rangle \tag{1.46}$$

As far as operators acting on a site are concerned, the action of this symmetry operation is straightforward:

$$\langle\alpha_i|\,\mathcal{R}^\dagger O_\mu \mathcal{R}\,|\alpha_j\rangle = \langle\beta_i|\,O\,|\beta_j\rangle \tag{1.47}$$

Anyway, we must take care when relate two different basis, because of the sign that is due to the ordering of fermionic operators:

$$\begin{aligned} |\beta_n\rangle &= \mathcal{R}(B^E_{n\ \alpha_{n-1},\sigma,\alpha_n}\,|\alpha_{n-1}\rangle\,|\sigma\rangle) \\ &= B^E_{n\ \alpha_{n-1},\sigma,\alpha_n} \mathcal{R}(|\sigma\rangle)\mathcal{R}(|\alpha_{n-1}\rangle)(-1)^{T(\alpha_{n-1})T(\sigma)} \\ &= B^E_{n\ \beta_{n-1},\lambda,\beta_n}\,|\lambda\rangle\,|\beta_{n-1}\rangle\,(-1)^{T(\beta_{n-1})T(\lambda)} \end{aligned} \tag{1.48}$$

where the T's are the eigenvalues of the number operators acting on the vectors in parenthesis. This part of the trasnformation does not imply any further projection of the state obtained so far, so we obtain:

$$\left|\chi^L\right\rangle = \sum_{\alpha_{m+1},\sigma,\lambda,\beta_{n-1}} \chi^L_{\alpha_{m+1},\sigma,\lambda,\beta_{n-1}}\,|\alpha_{m+1}\rangle\,|\sigma\rangle\,|\lambda\rangle\,|\beta_{n-1}\rangle \tag{1.49}$$

with the coefficients given by:

$$\chi^L_{\alpha_{m+1},\sigma,\lambda,\beta_{n-1}} = \phi^L_{\alpha_{m+1},\sigma,\alpha'_{m+2}} B^E_{n\ \beta_{n-1},\lambda,\beta'_n} (-1)^{T(\lambda)T(\beta_{n-1})} \tag{1.50}$$

The only requirement to be able to perform this transformation is the storage of the basis transformation matrices B.

When $m = n = L/2$ one further transformation is necessary in order to reuse the old wavefunction corresponding to the decomposition $L/2-1, 1, 1, L/2-1$. In fact, we're now building a new basis $\alpha_{L/2}$ corresponding to finite size iteration $I+1$ for the system, and we also want to use it for the environment. On the other hand, the old wavefunction is expressed in terms of vectors $\tilde{\beta}_{L/2+1}$ which were obtained from the old basis $\tilde{\alpha}_{L/2}$ corresponding to iteration I.

Anyway, the two basis sets $\tilde{\alpha}_{L/2}$ and $\alpha_{L/2}$ corresponding to iteration I and $I+1$ can be related recursively, starting from the identity transformation on a single point, as follows:

$$\langle \alpha_1 | \tilde{\alpha}_1 \rangle = \delta_{\alpha_1, \tilde{\alpha}_1} \tag{1.51}$$

$$\left\langle \alpha_{L/2} \middle| \tilde{\alpha}_{L/2} \right\rangle = B^S_{L/2\, \alpha_{L/2-1}, \sigma; \alpha_{L/2}} \tilde{B}^S_{L/2\, \tilde{\alpha}_{L/2-1}, \tilde{\sigma}; \tilde{\alpha}_{L/2}} \left\langle \alpha_{L/2-1} \middle| \tilde{\alpha}_{L/2-1} \right\rangle \delta_{\sigma, \tilde{\sigma}} \tag{1.52}$$

where the matrix transformations $\tilde{U}$ and U correspond to iteration I and $I+1$. All in all, the transformation's second part in this case is given by:

$$\chi^L_{\alpha_{L/2}, \sigma, \lambda, \beta_{L/2}} = \phi^L_{\alpha_{L/2}, \sigma, \alpha_{L/2+1}} \left\langle \alpha_{L/2} \middle| \tilde{\alpha}_{L/2} \right\rangle B^E_{L/2+1\, \tilde{\alpha}_{L/2}, \lambda, \alpha_{L/2+1}} (-1)^{T(\lambda) T(\beta_{L/2-1})} \tag{1.53}$$

1.4 Exact Diagonalization Algorithm

One of the most common methods used to build the superblock's ground state is based on the Laczos algorithm. This method resolve the eigenvalues problem for a $H^{n\times n}$ matrix creating an orthonormal basis first, and then assembling an approximated solution via a *Rayleigh-Ritz projection*. This isn't the only exact diagonalization algorithm; in the case of symmetric problems, the Arnoldi and Davidson method are mathematically equivalent, but the Lanczos method employ less arithmetical operations.

In this paragraph we'll describe the algorithm mathematical details and the *thick-restart* method, on which the Lanczos algorithm is based.

1.4.1 The Lanczos Algorithm

Let's consider an hermitian matrix $H^{n\times n}$

$$H^\dagger = H \qquad H^\dagger = (H^T)^* \tag{1.54}$$

In the case of DMRG calculations, H will be the Hamiltonian. Through the repeated application of matrix H to a vector v_1 of unitary norm and random chooosen components, we build the vectorial spaces known as **Krylov subspaces**

$$\begin{aligned} &\mathcal{K}_m = \mathcal{L}\{v_1, Hv_1, ..., H^{m-1}v_1\} \qquad \|v_1\| = 1 \\ &\dim \mathcal{K}_m \leq m \end{aligned} \tag{1.55}$$

During the iterative process, $\mathcal{K}_j \subseteq \mathcal{K}_{j+1}$; these spaces maximum dimension will be equal to m, because if $\dim \mathcal{K}_m \geq m$ every new vector created would be linear dependent from the previous ones.

Using the Gram-Shmidt process on the vectors costructed at each iteration, it is possible to obtain an orthonormal basis for the space $\mathcal{K}_j$:

$$\begin{aligned} &\mathcal{K}_j = \mathcal{L}\{v_1, v_2, ..., v_j\} \\ &\langle v_k, v_l \rangle = \delta_{k,l} \end{aligned} \tag{1.56}$$

Every vector $v_j \in \mathcal{K}_{j-1}$ could be expressed by a linear combination of power of H applied

to v_1, until $j-1$ order:

$$v_j = \sum_{k=0}^{j-1} c_j^k H^k v_1 \tag{1.57}$$

Being v_j ortoghonal to every previously build vectors, it will be orthogonal also to the previous Krylov subspaces

$$\begin{aligned} &v_j \perp \mathcal{K}_{j-1} \\ &\langle H^k v_1, v_j \rangle = 0 \qquad k = 0, 1, ..., j-2 \end{aligned} \tag{1.58}$$

Furthermore, H is hermitian,

$$\langle H^k v_1, H, v_j \rangle = \langle H^{k+1} v_1, v_j \rangle = 0 \qquad k = 0, ...j-3 \tag{1.59}$$

so Hv_j is ortoghonal to the $\mathcal{K}_{j-2}$ subspace; moreover, it is possible to expand Hv_j by means of power of H until the order j, so it is trivial deduce that $Hv_j \in \mathcal{K}_j$.

We can now state the vector Hv_j in the following way:

$$Hv_j = \gamma_i v_{j-1} + \alpha_j v_j + \beta_{j+1} v_{j+1} \qquad \gamma_1 = 0 \tag{1.60}$$

where the coefficients are obtainted taking advantage to the orthogonality of the v_j

$$\begin{cases} \gamma_i = \langle v_{j-1}, Hv_j \rangle \qquad \gamma_1 = 0 \\ \alpha_j = \langle v_j, Hv_j \rangle \\ \beta_{j+1} = \langle v_{j+1}, Hv_j \rangle \end{cases} \tag{1.61}$$

Using the Krylov space basis, the matrix H can be represented by a tridiagonal matrix:

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \dots & 0 \\ \gamma_2 & \alpha_2 & \beta_3 & \dots & \vdots \\ 0 & \gamma_3 & \alpha_3 & \dots & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & & & \alpha_{m-1} & \beta_m \\ 0 & \dots & 0 & \gamma_m & \alpha_m \end{bmatrix} \tag{1.62}$$

T_m will be hermitian too, because its elements β_i and γ_i are complex conjugates, as consequence of H hermiticy:

$$\gamma_i = \langle v_{j-1}, Hv_j \rangle = \langle Hv_{j-1}, v_j \rangle = \langle v_{j-1}, Hv_j \rangle^* = \beta_j^* \qquad i = 2, ... \tag{1.63}$$

Vectors v_j can be assembled until $\mathcal{K}_{j-1} \subset \mathcal{K}_j$, otherwise the sequence stops and the Krylov subspace result invariant with respect to H; coefficients β_j and γ_j will be non-null for the considered subspaces. Them,we can cocnlude that the matrix T_m is irreducible (in other words $\beta_j, \gamma_j \neq 0, \forall j$).

Equation (1.60) can be expressed in matricial form, we just need to define a rectangular matrix V_m constitute by the vectors recursively generated through the application of H; the element v_ij of V_m will be the i-th component of v_j:

$$(V_m)_{ij} = v_j^{(i)} = v_{ij} \tag{1.64}$$

With this notation, (1.60) became:

$$\begin{cases} \sum_{k=1}^{n} H_{ik} v_{kj} & = \gamma_j v_{ij-i} + \alpha_j v_{ij} \beta_{j+1} v_{ij+1} \\ & = \sum_{k=1}^{m} v_{ik} (T_m)_k j \qquad i = 1, ..., n \qquad j = 1, ..., m-1 \\ \sum_{k=1}^{n} H_{ik} v_{km} & = \gamma_m v_{im-i} + \alpha_j v_{im} \beta_{m+1} v_{im+1} \\ & = \sum_{k=1}^{m} v_{ik} (T_m)_k j + \beta_{m+1} v_{im+1} e_{jm} \qquad j = m \end{cases} \tag{1.65}$$

e_jm indicate the j-th components of the basis vector e_j; in a more compact way, we can write

$$\begin{aligned} HV_m &= V_m T_m + \beta_{m+1} v_{m+1} e_m^T \\ &= V_m T_m + \beta_{m+1} |v_{m+1}\rangle \langle e_m^T| \end{aligned} \tag{1.66}$$

The matrix V_m is composed by the Krylov space generator vectors, and establish a transformation between the Krylov space itself that coincide, with a proper choice of the initial vector v_1, with $\mathbb{C}$. We can deduce the properties of V_m starting from the

orthonomality properties of vectors v_j:

$$\begin{aligned} V_m^\dagger V_m &= 1_m \\ V_m V_m^\dagger &= P_m \\ T_m &= V_m^\dagger H V_m \end{aligned} \tag{1.67}$$

The last equation says that the matrix T_m is the projection of H on Kylov subspace $\mathcal{K}_m$.

It is now evident that Lanczos algorithm's goal is the construction of eigenvalues and eigenvectors of H through the eigenvalues and eigenvectors of T_m, which being tridiagonal, need less calculations and memory occupation. The matrix build from Krylov space vectors' establish a bound between the eigenvectors of H and T_m, as shown below.

*If $u \in \mathbb{C}$ is an eigenvector of T_m corresponding to the eigenvector μ, then the vector $y = V_m u \in \mathbb{C}^n$ (**Ritz's vector**) constitute an approximation for an eigenvector of H, and μ (**Ritz's value**) an approximation for the corresponding eigenvalue λ of H.*
The following theorem states how good is the approximation of μ with respect to λ:

Theorem 1.4.1 (***Wilkinson***)**.** *Let $H^{n\times n}$ be hermitian and let μ and y be respective a scalar and a vector not-null $\Rightarrow \exists\lambda$ eigenvector of H such that*

$$|\lambda - \mu| \leq \frac{\|Hy - \mu y\|}{\|y\|} \tag{1.68}$$

The valutation suggested by the theorem is translated in the disequality

$$|\lambda - \mu| \leq |\beta_{m+1} u^{(m)}| \tag{1.69}$$

Starting againg from equation (1.60) it is possible to express the vector $\beta_{j+1}v_{j+1}$ as result of the orthogonalization of Hv_j applied to vectors v_j, v_{j-1}

$$\beta_{j+1}v_{j+1} = Hv_j - \gamma_j v_{j-1} - \alpha_j v_j \tag{1.70}$$

The only restriction consist in considering only vectors with unitary norm, whereas we have a certain degree of freedom in the choice of the coefficients β_j, γ_j; if we choose to assign the whole complex phase to the vector v_{j+1} it is possible to consider β_j (and,

consequently γ_j that's his complex conjugate) real and positive:

$$w_{j+1} = Hv_j - \gamma_j v_{j-1} - \alpha_j v_j \tag{1.71}$$

with

$$\begin{cases} \alpha_j = \langle v_j, Hv_j \rangle \\ \gamma_j = \langle v_{j-1} Hv_j \rangle \qquad \gamma_1 = 0 \end{cases} \tag{1.72}$$

and

$$\begin{cases} \beta_{j+1} = \sqrt{\|w_{j+1}\|^2} \geq 0 \\ v_{j+1} = \frac{w_{j+1}}{\beta_{j+1}} \end{cases} \tag{1.73}$$

In table (1.3) is shown the iterative algorithm described above.

1.4.2 Thick restart method

The original Laczos method build the Krylov space starting from a single vector v_1, normalized to unity. Ideally, when Krylov spaces reach the dimension n of matrix H, T_m's eigenvalues give the complete spectrum of the Hamiltonian. In normal practice, however, the number of Ritz's vectors that can be stored in memory during the iteration is limited. Lanczos's algorithm is stopped after a finite number of steps, and the result is an approximation for the couple (λ, y). To improve eigenvalues and eigenvectors approximation, the Lanczos method is repeated many times, choosing conveniently the starting vector v_1.

The traditional restarting method consist in reducing the whole basis to a single vector, and start a new iteration with that vector. Usually, if we're looking for informations about only a couple eigenvalue-eigenvector (i.e. the one relative to the ground state), it's enough make use of Ritz's vector calculated by previous iteration as restarting vector.

In general, the best converging eigenvalues are the extreme ones, for whom the algorithm gave a good approximation only, for example, for the ground state. If we want a good description for the more internal eigenvalues and eigenvectors, which correspond, in a physical system, to the first excited states, we have to modify the method.

Classical restarting methods, named "blocks methods", like Davidson and Arnoldi, are difficoult and long to implement, plus they need a lot of time-machine; the *thick restart* method[25][26], on the other way, need only a relatively simple alteration of Lanczos' algorithm.

The basic idea consist in restart Lanczos' algorithm mantaining the informations related to a superior number of Ritz's vectors.

The thick restart methos start at first as a normal Lanczos' algorithm. If the maximum number of Lanczos' steps (this number will determine the dimension of Krylov's subspace) is fixed to m, then after m iterations the Lanczos' vector will fullfill the relation

$$HV_m = V_m T_m + \beta_{m+1} |v_{m+1}\rangle \langle e_m| \tag{1.74}$$

We're using the same notation of previous subsection, so T_m is a simmetric, tridiagonal matrix. Making us of the Rayleigh-Ritz's projection, we can obtain approximate solutions for the eigenvalues problem. If the couple (λ, y) is composed by an eigenvalue and an eigenvector of matrix T_m, then λ will give an approximation for the eigenvalue of H,

and $u = V_m y$ will approximate the corresponding eigenvector.

During the algorithm first iteration, a number k of Ritz's vectors will be stored. This number is related to the number of eigenvalue and eigenvector couples that we want to know with optimal precision. To obtain good results, it's enough that m is equal to the number of searched for couples. The k Ritz's vectors that we're looking for will be obtained applying the vectors' matrix of Krylov's space V_m to the corresponding $y_i \quad (i = 1, ..., k)$ eigenvectors of T_m:

$$\tilde{V}_k = V_m Y \tag{1.75}$$

$$\tilde{T}_k = Y^T T_m Y \tag{1.76}$$

$\tilde{T}_k$ represent a k-dimensional, diagonal matrix composed by the k eigenvalues corresponding to Y eigenvectors. From now on, the simbol ($\tilde{}$) will state the quantities after the restart.

Immediately after the restart, the new base's vectors will fullfill the relation:

$$H\tilde{V}_k = \tilde{V}_k\tilde{T}_k + \beta_m \left|\tilde{v}_{k+1}\right\rangle \left\langle s\right| \tag{1.77}$$

where $\left|\tilde{v}_{k+1}\right\rangle = \left|v_{m+1}\right\rangle$ and $\left\langle s\right| = Y \left\langle e_m\right|$.

The previous equalities include a crucial point of thick restart method: the vector $\left|v_{m+1}\right\rangle$ used to start again the iterations doesn't depend from the choice of vectors $\tilde{V}_k$, and possess the same direction of the last residual calculated during the previous iteration. Arrived at this step, we continue the standard Lanczos procedure, expanding base $\tilde{V}_k$ with the vector $\tilde{v}_{k+1}$, obtained from the previous iteration, and creating the Krylov space from $\tilde{v}_{k+1}$.

Lanczos' vectors calculated after the restart, have to be orthonormalized t all the conservated vectors. Using (1.77), we note that $\tilde{V}_k^T H \left|\tilde{v}_{k+1}\right\rangle = \beta_m \left|s\right\rangle$, so the Gram-Schmidt procedure produce the following result:

$$\begin{aligned}
\tilde{\beta}_{k+1} \left|\tilde{v}_{k+2}\right\rangle &= (\mathbb{I} - \tilde{V}_{k+1}\tilde{V}_{k+1}^T)H \left|\tilde{v}_{k+1}\right\rangle \\
&= (\mathbb{I} - \left|\tilde{v}_{k+1}\right\rangle \left\langle\tilde{v}_{k+1}\right| - \tilde{V}_k\tilde{V}_k^T)H \left|\tilde{v}_{k+1}\right\rangle \\
&= (\mathbb{I} - \left|\tilde{v}_{k+1}\right\rangle \left\langle\tilde{v}_{k+1}\right|)H \left|\tilde{v}_{k+1}\right\rangle - \tilde{V}_k\beta_m \left|s\right\rangle
\end{aligned} \tag{1.78}$$

Since vector $\beta_m |s\rangle$ is known, to orthonormalize vector $\tilde{v}_{k+2}$ we just need to evaluate $\tilde{\alpha}_{k+1}$, as in step 2 of table (3.3), and substitute the normalization step $w_j = Hv_j - \beta_j v_{j-1} - a_j v_j$ with

$$\tilde{w}_{k+1} = H\tilde{v}_{k+1} - \tilde{\alpha}_{k+1}\tilde{v}_{k+1} - \sum_{j=1}^{k} \beta_m s_j \tilde{v}_j \tag{1.79}$$

To calculate vector $\tilde{v}_{k+2}$, the matrix $\tilde{T}_k$ will be increased by one row and one column:

$$\tilde{T}_{k+1} = \begin{bmatrix} \tilde{T}_k & \beta_m s \\ \beta_m s^T & \tilde{\alpha}_{k+1} \end{bmatrix} \tag{1.80}$$

Obviously, Lanczos' recorsion after the restart will be unvaried, as in eq(1.66)
$H\tilde{V}_{k+1} = \tilde{V}_{k+1}\tilde{T}_{k+1} + \tilde{\beta}_{k+1} |\tilde{v}_{k+2}\rangle \langle e_{k+1}|$ with $\tilde{\beta}_{k+1} = \|\tilde{w}_{k+1}\|$.

After we calculate $|\tilde{v}_{k+1}\rangle$, $(i > 1)$, to obtain the remaining vectors $|\tilde{v}_{k+i+1}\rangle$ we use again the Gram-Schmidt procedure:

$$\tilde{\beta}_{k+i}\tilde{v}_{k+i+1} = H\tilde{v}_{k+i} - \tilde{\alpha}_{k+i}\tilde{v}_{k+i} - \tilde{\beta}_{k+i-1}\tilde{v}_{k+i-1} \tag{1.81}$$

This formula is valid for every $i > 2$, and at each iteration the matrix $\tilde{T}_{k+i}$ will enlarge as follow:

$$\tilde{T}_{k+i} = \begin{bmatrix} \tilde{T}_k & \beta_m s & 0 & \dots & 0 \\ \beta_m s^T & \tilde{\alpha}_{k+1} & \tilde{\beta}_{k+2} & & \\ 0 & \tilde{\beta}_{k+2} & \tilde{\alpha}_{k+2} & \tilde{\beta}_{k+2} & \\ \vdots & & & & 0 \\ \vdots & & & & \\ 0 & \dots & 0 & \tilde{\beta}_{k+i} & \tilde{\alpha}_{k+i} \end{bmatrix} \tag{1.82}$$

The new matrix $\tilde{T}_{k+i}$ will be composed by a diagonal part of order k (the matrix of conserved Ritz vectors), a part composed by row $\beta_m s^T$ and column $\beta_m s$, both of dimension k, and a tridiagonal part.

This way, repeating many times the algorithm, we obtain a good convergence for the k eigenvalue-eigenvector couples we were looking for.

The Thick Restart method's fundamental steps are shown in table (1.4).

Table 1.3: Iterative scheme of Lanczos algorithm

1. $u_1 = Hv_1, \quad \beta_1 = \gamma_1 = 0$
2. $u_j = Hv_j - \beta_j v_{j-1}$
3. $\alpha_j = \langle v_j, u_j \rangle$
4. $w_j = u_j - a_j v_j$
5. $\beta_{j+1} = \gamma_{j+1} = \|w_j\|$
6. $v_{j+1} = \frac{w_j}{\beta_{j+1}}$

Table 1.4: Iterative scheme of Lanczos thick-restart algorithm

- Inizialization

1. $v_{k+1} = \frac{w_k}{\beta_k} = \frac{w_k}{\|w_k\|}$
2. $u_{k+1} = Hv_{k+1}$
3. $\alpha_{k+1} = \langle v_{k+1}, u_{k+1} \rangle$
4. $w_{k+1} = u_{k+1} - a_{k+1} v_{k+1} - \sum_{j=1}^{k} \beta_m s_j \tilde{v}_j$
5. $\beta_{k+1} = \|w_k\|$

- Iteration (For$j = k+2, ...,$)

1. $u_1 = Hv_1, \quad \beta_1 = \gamma_1 = 0$
2. $u_j = Hv_j - \beta_j v_{j-1}$
3. $\alpha_j = \langle v_j, u_j \rangle$
4. $w_j = u_j - a_j v_j$
5. $\beta_{j+1} = \gamma_{j+1} = \|w_j\|$
6. $v_{j+1} = \frac{w_j}{\beta_{j+1}}$

Chapter 2

Time Evolution DMRG

In this section we describe an extension of the static DMRG, which incorporates real time evolution into the algorithm.

The aim of the time-dependent DMRG algorithm (t-DMRG) is to simulate the evolution of the ground state of a nearest-neighbor one dimensional system described by a Hamiltonian H_0, following the dynamics of a different Hamiltonian H. Remembering that time evolution in quantum mechanics is governed by the time-dependent Schröedinger equation

$$i\hbar\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = H\left|\psi(t)\right\rangle \tag{2.1}$$

whose formal solution is

$$\left|\psi(t)\right\rangle = e^{-iHt}\left|\psi(0)\right\rangle \tag{2.2}$$

we therefore want to evaluate the time evolution of a system after a perturbation $V(t)$ is switched on at $t = 0$, so that the effective Hamiltonian becomes

$$H(t) = H_0 + V(t) \tag{2.3}$$

where H_0 is the last superblock Hamiltonian approximating the system before the perturbation.

In order to calculate the time evolving eigenstates is necessary either to integrate equation (2.1) directly or find a good approximation for the unitary, time-evolution operator. Various different time-dependent simulation methods have been recently proposed[27] [28][29][30], but here we restrict our attention to two algorithm, the *Time-Step Targetting*,

based on the Rounge-Kutta time evolution, and the *Time Evolving Block Decimation* which relies on the Trotter time evolution.

The main difficulty in evaluating time evolution using DMRG is that the effective basis determined at the beginning of the time step cannot properly represent the evolved states. In fact, during the evolution the wave function changes and explores different parts of the Hilbert space, thus the truncated basis chosen to represent the initial state will be eventually no more accurate. To solve this problem the block basis should be updated at each temporal step, by adapting it to the instantaneous state. This can be done by repeating the DMRG renormalization procedure using the instantaneous state as the target state for the reduced density matrix.

2.1 Time Step Targetting

The main idea of the time-step targetted (TST) method, is to produce a basis which targets the states needed to represent one small but finite time step. Once this basis is complete enough, the time step is taken and the algorithm proceeds to the next time step.

This targetting is intermediate to two approaches: the Trotter methods, as we'll see in next section, target precisely one instant in time at any DMRG step, while Luo, Xiang, and Wangs approach[30] targetted the entire range of time to be studied. Targetting a wider range of time requires more density matrix eigenstates be kept, slowing the calculation. By targetting only a small interval of time, a smaller price is paid relative to the most efficient Trotter methods. In exchange for the modest loss of efficiency, we gain the ability to treat longer range interactions, ladder systems, and narrow two-dimensional strips. In addition, the error from a finite time step is greatly reduced relative to the second order Trotter method.

The procedure of Luo, et. al. for targetting an interval of time is nearly ideal: one divides the interval into n small steps of length ϵ , and targets $\psi(t=0), \psi(t=\epsilon), \psi(t=2\epsilon), ..., \psi(t=n\epsilon)$, simultaneously. By targetting these wavefunctions simultaneously, any linear combination of them is also included in the basis. This means than the basis is able describe an $n+1$-th order interpolation through these points, making it for reasonable ϵ and n essentially complete over the time interval. In the TST method the interval is short and n is fairly small: in the implementation of[31], $n = 3$ and the time step is

$\tau \approx J/10$ for a spin chain.

The Runge-Kutta (R-K) implementation of this approach is defined as follows: one takes a tentative time step at each DMRG step, the purpose of which is to generate a good basis. The standard fourth order R-K algorithm is used. This is defined in terms of a set of four vectors:

$$\begin{aligned} |k_1\rangle &= \tau \tilde{H}(t)\,|\psi(t)\rangle \\ |k_2\rangle &= \tau \tilde{H}(t+\tau/2)[|\psi(t)\rangle + 1/2\,|k_1\rangle] \\ |k_3\rangle &= \tau \tilde{H}(t+\tau/2)[|\psi(t)\rangle + 1/2\,|k_2\rangle] \\ |k_4\rangle &= \tau \tilde{H}(t+\tau)[|\psi(t)\rangle + |k_3\rangle] \end{aligned} \tag{2.4}$$

where $\tilde{H} = H(t) - E_0$. The state at time $t+\tau$ is given by

$$|\psi(t+\tau)\rangle \approx \frac{1}{6}[|k_1\rangle + 2\,|k_2\rangle + 2\,|k_3\rangle + |k_4\rangle] + O(\tau^5) \tag{2.5}$$

We target the state at times $t, t+\tau/3, t+2\tau/3$ and $t+\tau$. The R-K vectors have been chosen to minimize the error in $|\psi(t+\tau)\rangle$, but they can also be used to generate $|\psi\rangle$ at other times. The states at times $t+\tau/3$ and $t+2\tau/3$ can be approximated, with an error $O(\tau^4)$, as

$$\begin{aligned} |\psi(t+\tau/3)\rangle &\approx |\psi(t)\rangle + \frac{1}{162}[31\,|k_1\rangle + 14\,|k_2\rangle + 14\,|k_3\rangle - 5\,|k_4\rangle] \\ |\psi(t+2\tau/3)\rangle &\approx |\psi(t)\rangle + \frac{1}{81}[16\,|k_1\rangle + 20\,|k_2\rangle + 20\,|k_3\rangle - 2\,|k_4\rangle] \end{aligned} \tag{2.6}$$

Each half-sweep corresponds to one time step. At each step of the half-sweep, one calculates the R-K vectors (2.4), but without advancing in time. The density matrix is then obtained with the target states $|\psi(t)\rangle, |\psi(t+\tau/3)\rangle, |\psi(t+2\tau/3)\rangle$, and $|\psi(t+\tau)\rangle$. Advancing in time is done on the last step of a half-sweep. However, we may choose to advance in time only every other half-sweep, or only after several half-sweeps, in order to make sure the basis adequately represents the time-step. For the systems of Ref.[31],one half-sweep was adequate and the most efficient. The method used to advance in time in the last step need not be the R-K method used in the previous tentative steps. In fact,the computation time involved in the last step of a sweep is typically miniscule, so a more accurate procedure is warranted. A simple way to do this is to perform, say, 10 RK

iterations with step $\tau/10$. The relative weights of the states targetted can be optimized. An equal weighting is not optimal; the initial time and final time are more important. In Ref. [31], it was found that giving a weight of 1/3 for the first and final states, and 1/6 for the two intermediate states, gave excellent results.

2.2 Time Evolving Block Decimation

The time evolving block decimation is an algorithm based on matrix product states initially proposed by Vidal as an algorithm for simulating quantum time evolutions of one-dimensional systems efficiently on a classical computer[27]. As it turned out, it was so closely linked to DMRG concepts, that his ideas could being implemented easily into DMRG, leading to an adaptive time-dependent DMRG, where the DMRG state space adapts itself in time to the time-evolving quantum state.

The algorithm starts with a finite-system DMRG, in order to find an accurate approximation of the ground state ψ_0 of the static Hamiltonian H_0. Then the time evolution of ψ_0 is implemented, by using a Suzuki-Trotter[32][33] decomposition for the time evolution operator.

Let's consider an Hamiltonian with nearest-neighbor interactions; it can be divided in two addends

$$\hat{H} = \sum_{i\ odd} \hat{F}_{i,i+1} + \sum_{j\ even} \hat{G}_{j,j+1} \tag{2.7}$$

where $\hat{F}_{i,i+1}, \hat{G}_{j,j+1}$ are the local Hamiltonian on the odd bonds linking i and $i+1$, and the even bonds linking j and $j+1$. While all $\hat{F}$ and $\hat{G}$ terms commute among each other, $\hat{F}$ and $\hat{G}$ terms do in general not commute if they share one site, then the time evolution operator may be approximately represented by a (first order) Trotter expansion as:

$$e^{-i\hat{H}t} \approx \prod_{i\ odd} e^{-i\hat{F}\delta t} \prod_{j\ even} e^{-i\hat{G}\delta t} + O(\delta t^2) \tag{2.8}$$

and the time evolution of the state can be computed by repeated application of the two-site time evolution operators $e^{-i\hat{G}_{j,j+1}\delta t}$ and $e^{-i\hat{F}_{i,i+1}\delta t}$. This is a well known procedure in particular in Quantum Monte Carlo[32] where it serves to carry out imaginary time evolutions (e.g. checkerboard decomposition). Of course, one can enhance the precision

of the algorithm by using a fourth order expansion with error $O(\delta t^5)$.

The TEBD simulation algorithm now runs as follows:

1. Perform the following two steps for all even bonds (the order does not matter):
   (a) Apply $e^{-i\hat{G}_{j,j+1}\delta t}$ to $|\psi(t)\rangle$.For each local time update, a new wave function is obtained. The number of degrees of freedom on the active bond thereby increases, as will be detailed below.
   (b) Carry on the DMRG truncation, mantaining only the N_E states with the highest eigenvalues.
2. Repeat this two-step procedure for all odd bonds, applying $e^{-i\hat{F}_{i,i+1}\delta t}$.
3. This completes one Trotter time step. One may now evaluate expectation values at selected time steps, and continues the algorithm from step 1.

Let us now consider some computational details. Consider the decomposition of the system in

$$|\psi\rangle = \sum_{\alpha_M}^{N_S} \sum_{\beta_N}^{N_E} \psi_{\alpha_M,\beta_N} |\alpha_M\rangle |\beta_N\rangle \tag{2.9}$$

then move to the representation

$$|\alpha_M\rangle = \sum |\alpha_{M-1};\sigma\rangle \, B^S_{M\ \alpha_{M-1}\sigma;\alpha_M} \tag{2.10}$$

$$|\beta_N\rangle = \sum |\lambda;\beta_{N-1}\rangle \, B^U_{N\ \lambda\beta_{N-1};\beta_N} \tag{2.11}$$

and obtain the transformation matrix

$$B^S_{M\ \alpha_{M-1}\sigma;\alpha_M} = \langle \alpha_{M-1}\sigma|\alpha_M\rangle \tag{2.12}$$

$$B^U_{N\ \lambda\beta_{N-1};\beta_N} = \langle \lambda\beta_{N-1}|\beta_N\rangle \tag{2.13}$$

so that we reconduce to the form of a typical DMRG state for two blocks and two sites. To evolve the wavefunction we need a temporal evolution operator acting on two sites, which can be expanded as

$$\hat{O} = \sum |\sigma'\lambda'\rangle \, \langle \sigma''\lambda''| \, O_{\sigma'\lambda';\sigma''\lambda''} \tag{2.14}$$

We obtain

$$\begin{aligned}|\psi'\rangle = \hat{O}\,|\psi\rangle &= \sum (|\sigma'\lambda'\rangle\,\langle\sigma''\lambda''|)\,|\alpha_M;\beta_N\rangle\,O_{\sigma'\lambda';\sigma''\lambda''}\;\psi_{\alpha_M\beta_N} \\ &= \sum (|\sigma'\lambda'\rangle\,\langle\sigma''\lambda''|)\,|\alpha_{M-1}\sigma;\lambda\beta_{N-1}\rangle\,B^S_{M\;\alpha_{M-1}\sigma;\alpha_M} B^U_{N\;\lambda\beta_{N-1};\beta_N} O_{\sigma'\lambda';\sigma''\lambda''}\psi_{\alpha_M\beta_N} \\ &= \sum (|\sigma'\lambda'\rangle\,\langle\sigma''\lambda''|)\,|\alpha_{M-1}\sigma;\lambda\beta_{N-1}\rangle\,O_{\sigma'\lambda';\sigma''\lambda''}\tilde{\psi}_{\alpha_{M-1}\sigma;\lambda\beta_{N-1}}\end{aligned} \tag{2.15}$$

where we defined

$$\tilde{\psi}_{\alpha_{M-1}\sigma;\lambda\beta_{N-1}} \equiv B^S_{M\;\alpha_{M-1}\sigma;\alpha_M}\psi_{\alpha_M\beta_N}B^U_{N\;\lambda\beta_{N-1};\beta_N} \tag{2.16}$$

The expression of ψ' became

$$\begin{aligned}|\psi'\rangle &= \sum (-1)^{F(\sigma''\lambda'',\alpha_{M-1})}\,|\alpha_{M-1}\sigma';\lambda'\beta_{N-1}\rangle\,O_{\sigma'\lambda';\sigma\lambda}\tilde{\psi}_{\alpha_{M-1}\sigma;\lambda\beta_{N-1}} \\ &= \sum |\alpha'_M;\beta'_N\rangle\,\psi'_{\alpha'_M;\beta'_N}\end{aligned} \tag{2.17}$$

and

$$\begin{aligned}\psi'_{\alpha'_M;\beta'_N} &= \langle\alpha'_M;\beta'_N|\psi'\rangle \\ &= \sum \langle\alpha'_M|\alpha_{M-1}\sigma'\rangle\,\langle\beta'_N|\lambda'\beta_{N-1}\rangle\,O_{\sigma'\lambda';\sigma\lambda}\tilde{\psi}_{\alpha_{M-1}\sigma;\lambda\beta_{N-1}} \\ &= \sum O_{\sigma'\lambda';\sigma\lambda}\tilde{\psi}_{\alpha_{M-1}\sigma;\lambda\beta_{N-1}} \\ &= \tilde{\psi}'_{\alpha_{M-1}\sigma';\lambda'\beta_{N-1}}\end{aligned} \tag{2.18}$$

Finally we can reconduce to a wavefunction expressed on the system and environment basis

$$\tilde{\psi}'_{\alpha_{M-1}\sigma';\lambda'\beta_{N-1}} = \sum (B^S)^\dagger_{\alpha'_M;\alpha_{M-1}\sigma'}\psi'_{\alpha_{M-1}\sigma';\lambda'\beta_{N-1}}(B^U)^\dagger_{\beta'_N\lambda'\beta_{N-1}}(-1)^{F(\alpha'_M\beta'_N)} \tag{2.19}$$

A new renormalization can be carried out for $|\psi'\rangle$, to select only the best states in the new base $|\alpha'_M;\beta'_N\rangle$. In general the states and coefficients of the decomposition will have changed compared to the decomposition (2.9) previous to the time evolution, and hence they are adaptive. We indicate this by introducing a tilde for these states and coefficients.

The key point about the TEBD simulation algorithm, riassumed in table (2.1), is that a DMRG-style truncation to keep the most relevant density matrix eigenstates (or the

maximum amount of entanglement) is carried out at each time step. This is in contrast to previous time-dependentDMRG methods, where the basis states were chosen before the time evolution, and did not adapt to optimally represent the state at each instant of time.

Table 2.1: Iterative scheme of t-DMRG algorithm

1. Run the finite-system algorithm, in order to obtain the ground state $|\psi_0\rangle$ of $\hat{H}$.
2. Keep on the finite-system procedure by performing sweeps in which at each step the operator $e^{-i\hat{H}t}$ is applied to the system state.
3. Perform the renormalization, following the finite-system algorithm, and store the matrices B for the following steps.
4. At each step change the state representation to the new DMRG basis using Whites state prediction transformation
5. Repeat points 3 to 5, until a complete δt time evolution has been computed.

2.3 Trotter vs R-K: a quick comparison

The Trotter based methods for time evolution discussed above, while very fast, have two notable weaknesses: first, there is an error proportional to the time step τ squared. This error is usually tolerable and can be reduced to neglible levels by using higher order Trotter decompositions[31].

The second, and most serious, error in a t-DMRG program remains the truncation error.

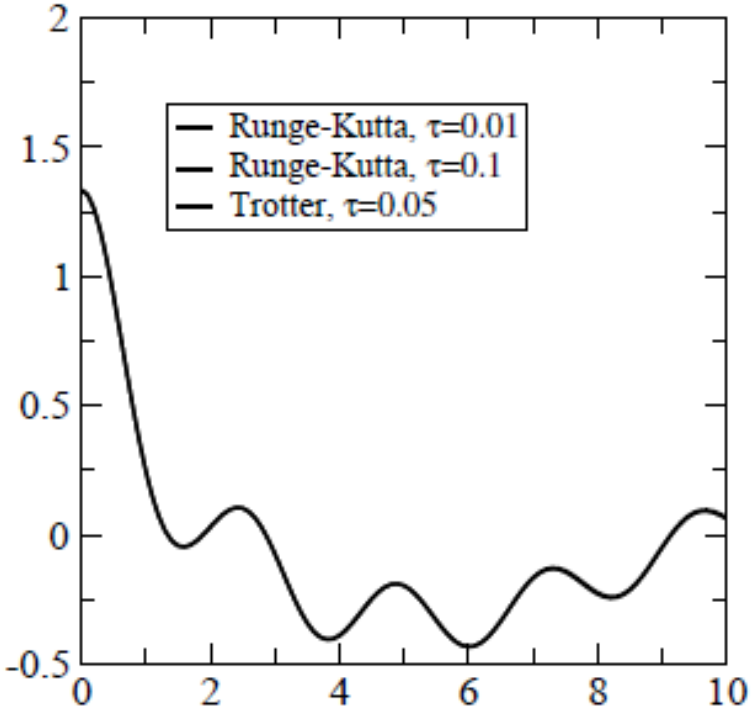
A nearly perfect time evolution with a negligible Trotter error is completely worthless if the wave function is affected by a relevant truncation error. It is worth to mention that t-DMRG precision becomes poorer and poorer as time grows larger and larger, due to the accumulated truncation error at each DMRG step. This depends on L, on the number of Trotter steps and, of course, on the number of maintained states N. At a certain instant of time, called the runaway time, the t-DMRG precision decreases by several order of magnitude. The runaway time increases with N, but decreases with the number of Trotter steps and with L[34].

Moreover, the Trotter's method is limited to systems with nearest neighbor interactions on a single chain. This limitation is more difficult to deal with. In the case of narrow ladders with nearest-neighbor interactions, one can avoid the problem by lumping all sites in a rung into a single supersite. Another approach would be to use a superblock configuration with, say, three center sites, which would allow one to treat two-leg ladders without using supersites. Unfortunately, these approaches become very inefficient for wider ladders, and are not applicable at all to general long-range interaction terms.

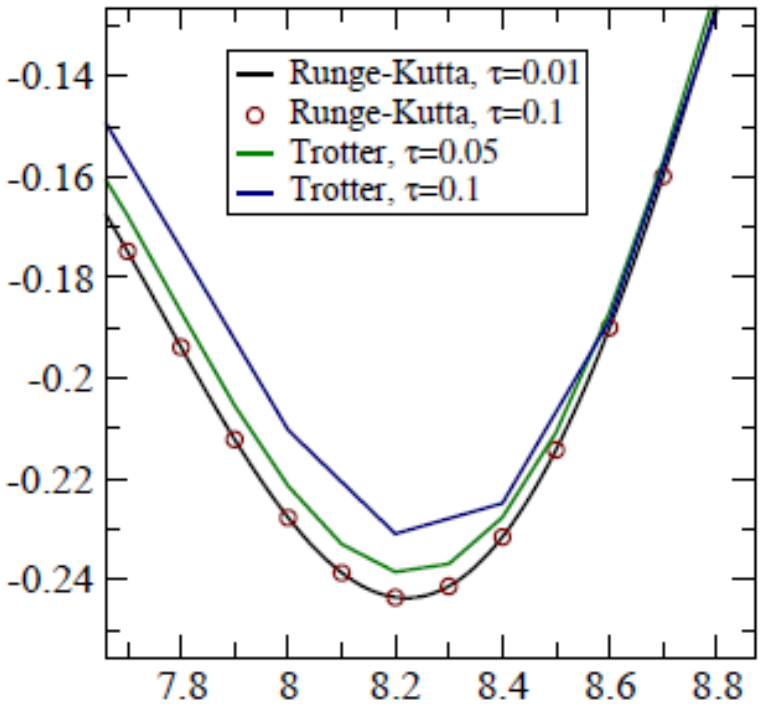
The time-step targetted (TST) method does not have these limitations, however, the Rounge-Kutta approximation does not preserve unitarity, so on long time the process become unstable and lose precision.

Both the Trotter method and the TST method give very accurate results. In figure (2.1)[35], we show a comparison of the methods. On a large scale, we cannot see any difference between the methods for times out to $t \approx 10$. If we zoom in on a particular region, we see the effects of the finite Trotter decomposition error, here falling as τ^2. We kept $N = 300$ states for the TST method, and $N = 200$ states for the Trotter methods. Typically, one finds that more states must be targetted for the TST method, because the targetting is over a finite interval of time rather than one instant. The

Trotter decomposition error can be eliminated almost completely by using a higher order decomposition. In this case, the smaller value of m still works as well as in the lower order methods. This combination gives the best combination of speed and accuracy.



(a) *Curves for time-evolution Heinseberg chains. The difference in results are not visible on this scale.*



(b) *Same curves, but showing only a small region so the differences become apparent.*

Figure 2.1: The value of $\langle S^-(16,t)|S^+(16,0)\rangle$ computed for a 31 site $S = 1$ Heisenberg chain, computed three different times. Here the curves labeled Runge-Kutta are the TST method, implemented using Runge Kutta. The time step is τ.

Chapter 3

Coding the DMRG

The DMRG used in this work has been primarily implemented by F. Ortolani, professor in the Department of Physics and Astronomy of the University of Bologna. It uses the Lanczos algorithm to provide iterative diagonalisation for the matrix representation of Hamiltonians. This implementation also admits time-dependent simulations, using the algorithm of Runge-Kutta to approximate the unitary time evolution of the system, as mentioned in the previous section. The codes provides the possibility to restrict the analysis to a particular symmetry present in the relative Hamiltonian. For example, it allows us to consider a smaller number of states, improve precision and obtains eigenstates with definite quantum numbers.

The purpose of this thesis work was the implementation of another method, the time evolving block decimation, based on the Trotter expansion, in order to improve flexibility and application range of the whole program. We stress out that Runge-Kutta and Trotter are complementary method, so it is the programmer who has the task to decide, for every problems he will manage, which one is the best suited choice.

3.1 Implementation of Trotter's Algorithm

The comprehension of the following code cannot be able to leave out of consideration some objects defined in the old version of the program, that we inherited and used.

Action class contains a matrix divided in blocks. Every operator such as Hamiltonian, potential, time evolution ecc. is represented by an Action class. The domain and range

spaces of the matrix are partitioned into subspaces (called *Ablock*) and every block of the matrix expresses matrix's action from a subspace of the domain to a subspace of the range. The true subspaces are labelled by a non null index while a null index labels the null subspace. Ideally every subspace of the domain and range is spanned by a set of states (with the same quantum numbers) and the block structure is suitable for operators which transform one subspace into only one subspace and most blocks are null (this is the case when operators carry well defined quantum properties). The single blocks can be considered as sparse matrices if the content of a sparsed matrix is not null. We do not implement a true sparsing of memory area, we build only the indexing part (both column-indexed and row-indexed part) and leave memory area unchanged. This scheme is derived from Numerical Recipes in C++.

Inside the class Action are defined a number of functions that give informations about the memory management of every block, and performe matrix algebra operations like transposing, conjugation, compression, sparsing and normalization, to name some.

Amono struct is a monomial of single Action factor; many Amono consitute an *Apoli* struct, a formal polinomial of Amono, by means we can modify and obtain the Action properties.

Block class describes the structure of a quantum lattice. Its members give the description of the corresponding Hilbert space (and subspaces) and of operators acting on this space (hamiltonian, global and local site operators). Every lattice block with more than one site is built composing two sublattices and the Hilbert space is the tensor product of the two Hilbert spaces (taking into account of the antisymmetry of fermionic states). Inside Block are written functions which label the number of sites and states, the partition in two sub-block (left and right) and the operations necessary to performe tensor products.

Now, to implement the time evolving block decimation, it's necessary to add 4 more functions, that take place after the standard DMRG routine. These functions are indicated as *Trotter, Evolve, Trotterstep* and the couple *trotterlft* and *trotterrgt*. The main function is Trotter, inside which is make use of the other three functions.

3.2 Trotter

The function Trotter provides the time evolution of the simulated system, making us of the nearest neighbour approximation and the Trotter decomposition.

First, Trotter acquire the System and the Environment from the arguments, and plot the sites and states ripartition of the Superblock (rows 1-8), then initialize the Hilbert subspaces of S and E, checking the initial state and storing it in *vket* (rows 11-29).

Next is applied the initial operator to the ground state. If it is null, *vket* is copied into *phi[0]*, otherwise we apply the function Superblock, which transform the formal polinomial $start_action$ (read from the input) into a sum of tensor products of operators acting on System and Universe Block's, storing both formal expressions and true Action. Then we apply the tensor product *hamaction* to *vket*, and save into *phi[0]* (30-35).

(36-44) evaluate, check and normalize the norm of *phi[0]*.

(64-80) set time evolution (te), time (t) and increase time step τ expressed as increase time-zips number ratio. Follow a plot of time (t), evolved and initial norm of *phi[0]* and the matrix of states.

Now begin the cicle that explicitly evaluate the time evolution. First comes a check for the new sweep, which control that S and E have the same number of sites and increase te (84-93), second there is a parse of the lattice Hamiltonian from the input, at fixed time. *phi[0]* is plotted.

(101-130) apply the time evolving operator $\hat{U}$ to couples of sites. After the time evolution, like in the finite size algorithm, we have to increase-decrease the dimensions of S and E, making use of three different functions. *trotterstep* manage the general case, but the configuration $L-2 \bullet \bullet$ and the $\bullet \bullet L-2$ one are special cases, that need the dedicated functions trotterrgt and trotterlft.

In (132-140) the actions arrays (operators) of the actual system and universe blocks are transferred to oldsystem and olduniverse blocks (to save memory space); the system and universe action lists are now empty (the space structure is the same).

(140-164) select the DMRG states during the time evolution, choosing from a list of criterions as number of DMRG states, time, number of zips ecc.

(165-183) define the new S and E blocks, using the reflection symmetry if allowed by the lattice Hamiltonian, then increase the counter of *zipdir* (185-188) to guide the blocks' growth in the desired direction. Follow the projection of *phi[0]* into the evolved

states, the evaluation of the norm and its check.

Finally, the Superblock properties are plotted (210), the timesteps and the time evolution are increased, finishing the cicle iteration.

Listing 3.1: Trotter

```
1  void trotter (Block & system,
       Block & universe)
2  {
3  Aarray phi;
4  size_t ket,l;
5  size_t sites = system.sites
       () + universe.sites ();
6  cout << "==========> Trotter
       start " << system.sites ()
        << "+"
7  << universe.sites () << " "
      << system.states () << "x"
8  << universe.states () << endl
       ;
9  size_t oddsites = sites % 2;
10 //
11 if (lfttimerule.size () == 0)
        lfttimerule = lftrule;
12 if (rgttimerule.size () == 0)
        rgttimerule = rgtrule;
13 ket = 0;
14 for (l = 1; l < super_target.
       subspaces (); l++)
15 if (ttarget_id == super_target
        .id (l)) ket = l;
16 if (ket == 0) {
17 cout << "Unidentified ket as
       initial state "
18 << name_define(ttarget_id) <<
       endl;
19 exit(1);
20 }
21 if (ttarget_index <
       super_target.states (ket))
22 ket = super_target.offset (
       ket) + ttarget_index;
23 else {
24 cout << "Initial state target
       index " << ttarget_index
25 << " exceeds previously found
       targets!" << endl;
26 exit(1);
27 }
28 Action vket = super_state [ket
       ];
29 super_state.clear ();
30 if (start_action.size () ==
       0) phi [0] = vket;
31 else {
32 start_action.reorder (sites);
33 hamaction = Superaction (
       start_action, system,
       universe, true);
34 biapply (phi [0], hamaction,
       vket, true);
35 }
36 double norm2 = multiply (phi
       [0], phi [0]).real();
37 double norm = sqrt(norm2);
38 if (norm < 1.e-08) {
39 cout << "Initial evolution
       state null!" << endl;
40 exit(1);
41 }
42 phi [0] *= (1.0/norm);
43 phi [0].normalize ();
44 norm2 = norm = 1.0;
45 size_t  lft_sites = system.
       sites ();
```

```
46 size_t  rgt_sites = sites -
       lft_sites;
47 double  E0        = 0.0;
48 size_t  tsteps    =  0;
49 if (time_zips < 1) time_zips =
       1;
50 size_t  zips      = 0;
51 long      zipdirini = -1;
52 long    zipdir    = zipdirini;
53 double  t, te, tau;
54 double  norm0;
55 complex<double> survival;
56 truncation = 0.0;
57 show_states      = false;
58 show_selection = false;
59 show_density     = 0;
60
61 super_weight.clear();
62 super_weight.push_back(1.0);
63 tensorweight = 0.0;
64 t  = te = 0.0;
65 tau = timestep / time_zips;
66 cout << setw(8)  << "time"
67 << setw(15) << "<V(t)|V(t)>"
68 << setw(15) << "<V(0)|V(0)>"
69 << setw(15) << "Zip error"
70 << setw(15) << "sites (states
       )"
71 << endl;
72 cout << setw(8)  <<
       setprecision(4) << fixed
       << t
73 << setw(15) << setprecision
       (10) << norm2
74 << setw(15) << setprecision
       (10) << norm2
75 << " zip" << setw(3) << zips
76 << resetiosflags(ios_base::
       fixed) << right <<
       showpoint
77 << setw(8) << setprecision
       (3)
78 << truncation << right << " "
       << lft_sites << "+" <<
       rgt_sites
79 << " (" << system.states()
       << "x" << universe.states
       () << ")"
80 << endl;
81 phi[2] = phi[0];
82 properties(system, universe,
       phi[0], phi[0], norm2, t)
       ;
83 while(true) {
84 if((lft_sites == rgt_sites +
       oddsites) && (zipdir ==
       zipdirini)) {
85 zips++;
86 t = te;
87 te = t + tau;
88 }
89 if(zips <= time_zips) {
90 if(norm2 < 1.e-06) {
91 cout << "Null evolving state!"
       << endl;
92 return;
93 }
94 hamt_parse(sites, t);
95 hamiltonian += (-E0);
96 phi[1] = Action();
```

```
97  Action U;
98  size_t slft = system.sites() - 1;
99  if (tsteps <= 1) phi[0].show("phi");
100
101 if (rgt_sites == 2) {
102 U = evolve(hamiltonian, sites - 2, sites - 1, system, universe, tau);
103 trotterrgt(phi[1], U, phi[0], system, universe);
104 phi[0] = phi[1];
105 if (tsteps <= 1) {
106 cout <<sites-2 << " "<< sites-1<< "R "<< zipdir<< " ";
107 phi[0].show("Rphi");
108 }
109 }
110 if (((zipdir > 0) && (slft % 2 == 0)) ||
111 ((zipdir < 0) && (slft % 2 == 1)) ||
112 (lft_sites == 2) || (rgt_sites == 2)) {
113 U = evolve(hamiltonian, slft, slft + 1, system, universe, tau);
114 trotterstep(phi[1], U, phi[0], system, universe);
115 if (tsteps <= 1) {
116 cout << slft << " " << slft+1 << "  "<<zipdir << " ";
117 phi[1].show("Uphi");
118 }
119 }
120 else phi[1] = phi[0];
121 if (lft_sites == 2) {
122 phi[0] = phi[1];
123 U = evolve(hamiltonian, 0, 1, system, universe, tau);
124 trotterlft(phi[1], U, phi[0], system, universe);
125 if (tsteps <= 0) {
126 cout << 0 << " " << 1 << "L " << zipdir << " ";
127 phi[1].show("Lphi");
128 }
129 }
130
131 super_state[0] = phi[1];
132 Block oldsystem, olduniverse;
133 oldsystem    = system;
134 system.action(idop_base,0) = oldsystem.base();
135 olduniverse  = universe;
136 universe.action(idop_base,0) = olduniverse.base();
137 size_t actual_zip = zips;
138 if (zips == 0) actual_zip = 1;
139 min_lft = max_lft = min_rgt = max_rgt = 0;
140
141 for (size_t rule = 0; rule < lfttimerule.size(); rule++) {
142 double trule = lfttimerule[rule].br_time;
143 size_t z = lfttimerule[rule].br_zip;
```

```
144 size_t s = lfttimerule [rule]
        .br_sites;
145 if ((t >= trule) && (
        actual_zip >= z) && (system
         .sites () >= s)) {
146 if (min_lft < lfttimerule [
        rule] .br_min)
147 min_lft = lfttimerule [rule] .
        br_min;
148 if (max_lft < lfttimerule [
        rule] .br_max)
149 max_lft = lfttimerule [rule] .
        br_max;
150 n_cutlft = lfttimerule [rule]
        .br_cut;
151 }
152 }
153 for (size_t rule = 0; rule <
        rgttimerule .size (); rule
        ++) {
154 double trule = rgttimerule [
        rule] .br_time;
155 size_t z = rgttimerule [rule]
        .br_zip;
156 size_t s = rgttimerule [rule]
        .br_sites;
157 if ((t >= trule) && (
        actual_zip >= z) && (
        universe .sites () >= s)) {
158 if (min_rgt < rgttimerule [
        rule] .br_min)
159 min_rgt = rgttimerule [rule] .
        br_min;
160 if (max_rgt < rgttimerule [
        rule] .br_max)
161 max_rgt = rgttimerule [rule] .
        br_max;
162 n_cutrgt = rgttimerule [rule]
        .br_cut;
163 }
164 }
165 updatebase (system, universe,
        zipdir);
166 if (zipdir < 0) {
167 universe .reflectreset ();
168 blockrgt [rgt_sites] .
        actionclear (0, name_action
         ());
169 blockrgt [rgt_sites] =
        universe;
170 system   = Block (blocklft [
        lft_sites -2], blocklft [1])
        ;
171 if (reflect_universe)
172 universe = Block (blockrgt
        [1], blockrgt [rgt_sites]);
173 else
174 universe = Block (blockrgt [
        rgt_sites], blockrgt [1]);
175 } else {
176 blocklft [lft_sites] .
        actionclear (0, name_action
         ());
177 blocklft [lft_sites] = system;
178 system   = Block (blocklft [
        lft_sites], blocklft [1]);
179 if (reflect_universe)
180 universe = Block (blockrgt
        [1], blockrgt [rgt_sites
        -2]);
```

```
181 else
182 universe = Block (blockrgt [
        rgt_sites -2], blockrgt [1])
        ;
183 }
184
185 lft_sites += zipdir;
186 rgt_sites -= zipdir;
187 if ((lft_sites <= 2) || (
        rgt_sites <= 2)) zipdir = -
        zipdir;
188
189 phi [0] = Action(system .
        quantum (), universe .
        quantum ());
190 projection (phi [0], system,
        universe, phi [1],
        oldsystem, olduniverse);
191 norm2 = multiply (phi [0], phi
         [0]) .real ();
192 norm = sqrt(norm2);
193 phi [1] = Action(system .
        quantum (), universe .
        quantum ());
194 projection (phi [1], system,
        universe, phi [2],
        oldsystem, olduniverse);
195 phi [2] = phi [1];
196 norm0 = multiply (phi [2], phi
         [2]) .real ();
197
198 cout << setw (8) <<
        setprecision (4) << fixed
        << t
199 << setw (15) << setprecision
        (10) << norm2
200 << setw (15) << setprecision
        (10) << norm0
201 << " zip" << setw (3) << zips
202 << resetiosflags (ios_base::
        fixed) << right <<
        showpoint
203 << setw (8) << setprecision
        (3)
204 << truncation << right << " "
        << lft_sites << "+" <<
        rgt_sites
205 << " (" << system .states ()
        << "x" << universe .states
        () << ")"
206 << endl;
207
208 }
209 else {
210 properties (system, universe,
        phi [0], phi [0], norm2, t)
        ;
211 tsteps++;
212 if (tsteps >= steps_number)
        break;
213 zips   =  0;
214 zipdir =    zipdirini;
215 te = t;
216 } } }
```

3.3 Evolve

Evolve provides the time evolving operator $\hat{U}$. If the Hamiltonian is static, $\hat{U}$ is the same at every iteration, but to consider even the time-dependent Hamiltonian, we decided to calculate the operator anew for every iteration.

(7-28) select the Hamiltoninan's terms which act on the desidered sites, reading and confronting the monomials' index, and form a polinomial *hsub*. The new Hamiltonian (hsub) is then applied to a Block created from the System and Universe block, then stored into an Action hh.

(29-57) manage a hidden problem of the stored Hamiltonian. DMRG provides the block reordering of hsub into a block diagonal matrix, but to save memory space the null elements are not stored. However, when we apply an exponential to evolve the system, null elements give the Identity, not zero. To bypass this problem, we define a scalar identity with the same dimensions of hsub; only those blocks that in hsub are null get replaced with the corresponding identity blocks of the scalar identity function. The only remaining task is the initialization and evaluation of the eigenvalues, via the householder tridiagonalization(51).

(57-77) compute the time evolution, carrying out the matrix product

$$\hat{U} = e^{-ih_{sub}t} = Be^{-i\alpha_{eigen}t}B^{-1}$$

where α_{eigen} are the eigenvalues of hsub and B the Superblock base.

Listing 3.2: Evolve

```
1   Action evolve (const Apoli & hlattice, long splitlft, long splitrgt,
2   Block & system, Block & universe, double delta)
3   {
4   size_t m, n, sub, offset, dimension, sites;
5   sites = system.sites() + universe.sites();
6   Apoli hsub;
7   for (m = 0; m < hlattice.size(); m++) {
8   Amono mono = hlattice[m];
9   long inner = 0;
10  if (mono.order() == 1) {
11  long index = mono[n].af_st;
12  if ((index > 0) || (index < sites - 1)) mono *= 0.5;
13  }
14  for (n = 0; n < mono.order(); n++) {
15  Afactor af(mono[n]);
16  if (af.af_st < splitlft) continue;
17  if (af.af_st > splitrgt) continue;
18  inner++;
19  mono[n].af_st -= splitlft;
20  }
21  if (inner && (inner == mono.order())) {
22  hsub += mono;
23  }
24  }
25  Block pp(system.parent()[1], universe.parent()[1]);
26  Action hh, uu, bb;
27  bb = pp.base();
28  hh = pp.action(hsub, hh);
29  uu = Action(hh.range(), hh.domain());
30  uu.scalaridentity(1.0);
31  size_t states = pp.states();
32  double * eigen = new double[2 *states];
33  for (m = 0; m < 2 * states; m++) eigen[m] = 0.0;
34  double * offd = eigen + states;
35  Ablock * b = hh.block();
36  Ablock * ub = uu.block();
37  double * mm = hh.storage();
38  for (size_t nb = 0; nb < hh.blocks(); nb++) {
39  if (b[nb].ab_domain != b[nb].ab_range) {
40  cout << "errrrrrrore" << endl;
41  exit(0);
```

```
42      }
43      sub = b [nb] .ab_domain;
44      ub [sub - 1] .ab_roffset =
            0;
45      double * mr = 0;
46      double * mi = 0;
47      if (b [nb] .ab_roffset) mr
            = mm + b [nb] .
            ab_roffset;
48      if (b [nb] .ab_ioffset) mi
            = mm + b [nb] .
            ab_ioffset;
49      dimension = hh .width (sub
            );
50      offset    = hh .domain ()
            .offset (sub);
51      householder (mr, mi, eigen
            + offset, offd +
            offset, dimension,
            dimension);
52      if (tqli (eigen + offset,
            offd + offset, mr, mi,
            dimension, dimension))
            {
53      cout << "tqli errrrrror"
            << endl;
54      exit (0);
55      }
56      }
57      hh += uu;
58      uu = hh;
59      uu .dagger ();
60      complex<double> * mri =
            new complex<double> [
            states * states];
61      hh .expand (mri);
62      complex<double> zi (0.0, -
            delta);
63      for (n = 0; n < states; n
            ++) {
64      for (m = 0; m < states; m
            ++) {
65      mri [m + n*states] *= exp(
            zi * eigen [m]);
66      }
67      }
68      hh .compress (mri);
69      delete [] mri;
70      delete [] eigen;
71      uu *= hh;
72      hh = bb;
73      bb .dagger ();
74      uu *= bb;
75      hh *= uu;
76      return hh;
77      }
```

3.4 Trotterstep

Trotterstep provides the site decomposition of section 2.2. *Blft* and *Brgt* are respectively

$$B^S_{M\ \alpha_{M-1}\sigma;\alpha_M} = \langle \alpha_{M-1}\sigma|\alpha_M\rangle \qquad B^U_{N\ \lambda\beta_{N-1};\beta_N} = \langle \lambda\beta_{N-1}|\beta_N\rangle$$

and (rows 8-10) multiply these two blocks to obtain the superblock representation

$$Blft = \langle \alpha_{M-1}\sigma\lambda\beta_{N-1}|\psi\rangle$$

(rows 17-21) initialize the array of pointers which include the elements of *Blft* and the time evolution operator U. Note that for an $n \times m$ matrix, we initialize an array of pointer of dimension $n + m$, not a pointer of pointers, as recommended from C++ guidelines.

(rows 22-43) apply the time evolution operator $\hat{U}$ to the superblock in the representation $\alpha_{M-1}\sigma\lambda\beta_{N-1}$, to obtain the evolved ψ', as in equation (2.15).

(44-51) evaluate the matrix product of equation (2.19), that give back the evolved wavefunction expressed on the System and Environment basis.

As we mentioned before, the function Trotterstep isn't suited to described the border configuration. These special cases are handled by Trotterrgt and Trotterlft, simplyfied version of Trotterstep in which is performed only the final matrix product with the evolved wavefunction and the 2-sites basis matrix.

Listing 3.3: Trotterstep

```
1  void trotterstep (Action &
       result, Action & U,
       Action & state,
2  Block & system, Block &
       universe)
3  {
4  Action Blft, Brgt;
5  long sitestates = blocklft
       [1].states ();
6  Blft = system.base ();
7  Brgt = universe.base ();
8  Brgt.transpose ();
9  multiply (Blft, Blft,
       state);
10 multiply (Blft, Blft, Brgt
       );
11 size_t na, na1, nb, nb1,
       ia, ia1, sigma, sigma1,
        jb, jb1, lambda,
       lambda1,
12 sl, sl1, im, jm;
13 na = Blft.height ();
14 nb = Blft.width ();
15 na1 = na / sitestates;
16 nb1 = nb / sitestates;
17 size_t nd = Blft.height
       () * Blft.width ();
18 size_t nu = U.height () *
       U.width ();
19 complex<double> * mm = new
       complex<double> [2 *
       nd + nu];
20 complex<double> * mr = mm
       + nd;
21 complex<double> * mu = mr
       + nd;
22 Blft.expand (mm);
23 U.expand (mu);
24 for (jb = 0; jb < nb; jb
       ++) {
25 lambda1 = jb % sitestates;
26 jb1     = jb / sitestates;
27 for (ia = 0; ia < na; ia
       ++) {
28 sigma1 = ia / na1;
29 ia1    = ia % na1;
30 sl1    = sigma1 + lambda1
       * sitestates;
31 mr[ia + jb * na] = 0.0;
32 for (lambda = 0; lambda <
       sitestates; lambda++) {
33 jm = lambda + jb1 *
       sitestates;
34 for (sigma = 0; sigma <
       sitestates; sigma++) {
35 sl = sigma + lambda *
       sitestates;
36 im = ia1 + sigma * na1;
37 complex<double> uu = mu[
       sl1 + sl * sitestates *
        sitestates];
38 complex<double> psi = mm[
       im + jm * na];
39 mr[ia + jb * na] += uu *
        psi;
40 }
41 }
42 }
43 }
```

```
44      Blft .compress (mr);
45      Brgt = system . base ();
46      Brgt .dagger ();
47      multiply (Blft, Brgt, Blft
            );
48      Brgt = universe .base ();
49      Brgt .conjugate ();
50      multiply (result, Blft,
            Brgt);
51      }
```

Listing 3.4: trotterlft and trotterrgt

```
1       void trotterlft (Action &
            result, Action & U,
            Action & state,
2       Block & system, Block &
            universe)
3       {
4       Action Blft = system .base
            ();
5       Action bb   = Blft;
6       Blft .dagger ();
7       Blft *= U;
8       Blft *= bb;
9       Blft *= state;
10      result = Blft;
11      }
12      void trotterrgt (Action &
            result, Action & U,
            Action & state,
13      Block & system, Block &
            universe)
14      {
15      Action Ut = U;
16      Ut .transpose ();
17      Action Brgt = universe .
            base ();
18      Action bb = Brgt;
19      bb .dagger ();
20      Action aa = state;
21      aa *= bb;
22      aa *= Ut;
23      aa *= Brgt;
24      result = aa;
25      }
```

Chapter 4

Numerical Results

To test the newly implemented algorithm, we chose an exactly solvable system, the free spinless fermions with open boundaries, and compared the analytical solution to the numerical one. In particular, we're intrested in the time evolution of a monodimensional chain of L sites, in which $L/2$ fermions are initially confined in the left (or right) half of the chain, and then let free to move.

4.1 An Exact solved system: Free spinless Fermions with open boundaries

This system is described by the Hamiltonian:

$$\hat{H} = -J \sum_{j=0}^{L-2} (c_j^\dagger c_{j+1} + c_{j+1}^\dagger c_j) - \mu \sum_{j=0}^{L-1} c_j^\dagger c_j \tag{4.1}$$

where J is the nearest neighbours hopping term, and μ the chemical potential. The eigenfunctions of $\hat{H}$ are the following set:

$$S_{jp} = \sqrt{\frac{2}{L+1}} \sin\left[\pi \frac{(p+1)}{(L+1)}(j+1)\right] \qquad p = 0, 1, ..., L-1 \tag{4.2}$$

Let's check out the orthonormality.

$$\begin{aligned}\sum_{j=0}^{L-1} S_{jp}S_{jq} &= \frac{2}{L+1}\sum_{j=0}^{L-1}\sin[k_p(j+1)]\sin[k_q(j+1)] \\ &= \frac{1}{L+1}\sum_{j=0}^{L-1}\{\cos[(k_p-k_q)(j+1)]-\cos[(k_p+k_q)(j+1)]\}\end{aligned} \tag{4.3}$$

where, end sites excluded,

$$\begin{aligned}-\pi <& k_p - k_q < \pi \\ 0 <& k_p + k_q < 2\pi\end{aligned}$$

If we consider the sums over sine and cosine as:

$$\sum_{j=0}^{L-1}\cos[\alpha(j+1)] = \mathrm{Re}\sum_{j=0}^{L-1} e^{i\alpha(j+1)} \qquad \sum_{j=0}^{L-1}\sin[\alpha(j+1)] = \mathrm{Im}\sum_{j=0}^{L-1} e^{i\alpha(j+1)} \tag{4.4}$$

we immediately obtain that, if $e^{i\alpha} = 1$

$$\sum_{j=0}^{L-1}\cos[\alpha(j+1)] = L \qquad \sum_{j=0}^{L-1}\sin[\alpha(j+1)] = 0 \tag{4.5}$$

and if $e^{i\alpha} \neq 1$

$$\sum_{j=0}^{L-1}\cos[\alpha(j+1)] = -\frac{1}{2}(1+\cos[\alpha(L+1)]) + \frac{1}{2}\cot(\alpha/2)\sin[\alpha(L+1)] \tag{4.6}$$

$$\sum_{j=0}^{L-1}\sin[\alpha(j+1)] = \frac{1}{2}\sin[\alpha(L+1)] + \frac{1}{2}\cot(\alpha/2)(1-\cos[\alpha(L+1)]) \tag{4.7}$$

Therefore, it's easy to check that

$$\sum_{j=0}^{L-1} S_{jp}S_{jq} = \delta_{pq} \qquad \sum_{p=0}^{L-1} S_{jp}S_{kp} = \delta_{jk} \tag{4.8}$$

Now we want to perform a canonical transformation

$$\begin{cases} b_p^\dagger = \sum_{j=0}^{L-1} c_j^\dagger S_{jp} \\ b_p = \sum_{j=0}^{L-1} c_j S_{jp} \end{cases} \qquad \begin{cases} c_j^\dagger = \sum_{p=0}^{L-1} S_{jp} b_p^\dagger \\ c_j = \sum_{p=0}^{L-1} S_{jp} b_p \end{cases} \tag{4.9}$$

that change the Hamiltonian in

$$\begin{aligned} \hat{H} &= -J \sum_{p,q=0}^{L-1} \left\{ \sum_{j=0}^{L-1} (S_{jp} S_{j+1,q} + S_{jp} S_{j-1,q}) \right\} b_p^\dagger b_p - \mu \sum_{p=0}^{L-1} b_p^\dagger b_p \\ &= \sum_{p=0}^{L-1} \varepsilon_p b_p^\dagger b_p - \mu \sum_{p=0}^{L-1} b_p^\dagger b_p \end{aligned} \tag{4.10}$$

where

$$\varepsilon_p = -2J \cos(k_p) = -2J \cos\left[\pi \frac{(p+1)}{(L+1)} \right] \tag{4.11}$$

The analytical diffusion of the spinless fermions is checked evaluating the number operator $\hat{n}_k$ from the ground state $|\psi_0\rangle = \prod_{j=0}^{L/2-1} c_j^\dagger |0\rangle$, imposing that at $t = 0$:

$$n_k = \langle \psi_0 | \, c_k^\dagger c_k \, | \psi_0 \rangle = \theta(L/2 - k) = \begin{cases} 1 & k \leq L/2 \\ 0 & k > L/2 \end{cases} \tag{4.12}$$

with the time evolving operator

$$\begin{cases} b_p^\dagger(t) = e^{-i\varepsilon_p t} b_p^\dagger(0) e^{i\mu t} \\ b_q(t) = e^{i\varepsilon_q t} b_q(0) e^{-i\mu t} \end{cases} \tag{4.13}$$

so

$$\begin{aligned} n_k(t) &= \langle \psi_0 | \, e^{i\hat{H}t} c_k^\dagger c_k e^{-i\hat{H}t} \, | \psi_0 \rangle \\ &= \sum_{p,q=0}^{L-1} S_{kp} S_{kq} e^{-i[\varepsilon_p - \varepsilon_q]t} \, \langle \psi_0 | \, b_p^\dagger(0) b_q(0) \, | \psi_0 \rangle \\ &= \sum_{p,q=0}^{L-1} \sum_{m,n=0}^{L-1} S_{kp} S_{kq} S_{mp} S_{nq} e^{-i[\varepsilon_p - \varepsilon_q]t} \, \langle \psi_0 | \, c_m^\dagger c_n \, | \psi_0 \rangle \end{aligned} \tag{4.14}$$

$$= \sum_{p,q=0}^{L-1} \sum_{m,n=0}^{L-1} S_{kp} S_{kq} S_{mp} S_{nq} e^{-i[\varepsilon_p - \varepsilon_q]t} \delta_{m,n} \theta(L/2 - m)$$
$$= \sum_{p,q=0}^{L-1} \sum_{m=0}^{L-1} S_{kp} S_{kq} S_{mp} S_{mq} e^{-i[\varepsilon_p - \varepsilon_q]t}$$

With some algebraic manipulation, the expression of $n_k(t)$ can be simplified, noting that

$$\sum_{m=0}^{L-1} S_{mp} S_{mq} = \begin{cases} \frac{1}{2} & for \quad q = p \\ \frac{1}{2(L+1)} \left\{ \frac{\sin[\frac{\pi}{2}(p-q)]}{\sin[\frac{k_p - k_q}{2}]} - \frac{\sin[\frac{\pi}{2}(p+q+2)]}{\sin[\frac{k_p + k_q}{2}]} \right\} & for \quad q \neq p \end{cases} \tag{4.15}$$

so we obtain

$$\begin{aligned} \hat{n}_k(t) =& \sum_{q=0}^{L-1} \frac{S_{kq}^2}{2} + \sum_{p \neq q}^{L-1} \frac{1}{2(L+1)} \left\{ \frac{\sin[\frac{\pi}{2}(p-q)]}{\sin[\frac{k_p - k_q}{2}]} - \frac{\sin[\frac{\pi}{2}(p+q+2)]}{\sin[\frac{k_p + k_q}{2}]} \right\} S_{kp} S_{kq} e^{-i[\varepsilon_p - \varepsilon_q]t} \\ =& \frac{1}{2} + \sum_{p \neq q}^{L-1} \frac{1}{2(L+1)} \left\{ \frac{\sin[\frac{\pi}{2}(p-q)]}{\sin[\frac{k_p - k_q}{2}]} - \frac{\sin[\frac{\pi}{2}(p+q+2)]}{\sin[\frac{k_p + k_q}{2}]} \right\} S_{kp} S_{kq} e^{-i[\varepsilon_p - \varepsilon_q]t} \end{aligned} \tag{4.16}$$

4.2 Nearest neighbour approximation

In the nearest neighbour approximation, we evolve the system repeatedly applying the 2-sites time evolution operator. We follow the finite size algorithm order, so starting from the middle of the chain, we have to apply $e^{i\hat{H}t}$ on the odd bonds moving to the left, the even bonds moving to the right and finally the remaining odd bonds moving to the left. Every bond must be considered only one time.

For $L = 2$ the Hamiltonian is

$$\hat{H} = -J(c_0^\dagger c_1 + c_1^\dagger c_0) - \mu(c_0^\dagger c_0 + c_1^\dagger c_1) \tag{4.17}$$

and the eigenfunctions became

$$S = \sqrt{\frac{2}{3}} \begin{bmatrix} \sin\left(\frac{\pi}{3}\right) & \sin\left(\frac{2}{3}\pi\right) \\ \sin\left(\frac{2}{3}\pi\right) & \sin\left(\frac{4}{3}\pi\right) \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.18}$$

so the canonical transformation is

$$\begin{cases} b_0^\dagger = \frac{1}{\sqrt{2}}(c_0^\dagger + c_1^\dagger) \\ b_1 = \frac{1}{\sqrt{2}}(c_0^\dagger - c_1^\dagger) \end{cases} \qquad \begin{cases} c_0^\dagger = \frac{1}{\sqrt{2}}(b_0^\dagger + b_1^\dagger) \\ c_1^\dagger = \frac{1}{\sqrt{2}}(b_0^\dagger - b_1^\dagger) \end{cases} \tag{4.19}$$

The Hamiltonian in the new base is

$$\hat{H} = -(\mu + J)b_0^\dagger b_0 - (\mu - J)b_1^\dagger b_1 \tag{4.20}$$

The generic exponential of excitations operators is

$$e^{\alpha b_k^\dagger b_k} = 1 + \sum_{n=1}^{\infty} \frac{\alpha^n}{n!}(b_k^\dagger b_k)^n = 1 + b_k^\dagger b_k \sum_{n=1}^{\infty} \frac{\alpha^n}{n!} = 1 + b_k^\dagger b_k(e^\alpha - 1) \tag{4.21}$$

so the Hamiltonian's time evolution is

$$\begin{aligned} \hat{U} = e^{i\hat{H}t} &= e^{i(\mu+J)tb_0^\dagger b_0} e^{i(\mu-J)tb_1^\dagger b_1} \\ &= [1 + (e^{i(\mu+J)t} - 1)b_0^\dagger b_0][1 + (e^{i(\mu-J)t} - 1)b_1^\dagger b_1] \\ &= 1 + \frac{1}{2}(e^{i(\mu+J)t} - e^{i(\mu-J)t} - 2)(c_0^\dagger c_0 + c_1^\dagger c_1) \\ &\quad + \frac{1}{2}(e^{i(\mu+J)t} - e^{i(\mu-J)t})(c_0^\dagger c_1 + c_1^\dagger c_0) \\ &\quad + (e^{i2\mu t} - e^{i(\mu+J)t} - e^{i(\mu-J)t} + 1)(c_0^\dagger c_1^\dagger c_1 c_0) \end{aligned} \tag{4.22}$$

The effect of $\hat{U} = e^{i\hat{H}t}$ on the 2-sites basis

$$|00\rangle \qquad |10\rangle \qquad |01\rangle \qquad |11\rangle \tag{4.23}$$

is

$$\begin{aligned} e^{i\hat{H}t}|00\rangle &= |00\rangle \\ e^{i\hat{H}t}|10\rangle &= e^{i\mu t}(|10\rangle\, cos(Jt) + i\,|01\rangle\, sin(Jt)) \\ e^{i\hat{H}t}|01\rangle &= e^{i\mu t}(i\,|10\rangle\, sin(Jt) + |01\rangle\, cos(Jt)) \\ e^{i\hat{H}t}|11\rangle &= e^{2i\mu t}|11\rangle \end{aligned} \tag{4.24}$$

Now that we know the explicit operation carried by implemented algorithm, let's check the numerical results for a chain of $L = 6$ sites. To simplify the calculations, we set $\mu = 0$. The ground state is

$$|\psi_0\rangle = |111000\rangle \tag{4.25}$$

which evolve as first iteration as

$$\begin{aligned} |\psi(t)\rangle_1 &= \hat{U}_{34}\hat{U}_{45}\hat{U}_{23}\hat{U}_{01}\hat{U}_{12}\,|111000\rangle \\ &= |111000\rangle\, cos(Jt) + i\,|110100\rangle\, sin(Jt)cos(Jt) - |110010\rangle\, sin^2(Jt) \end{aligned} \tag{4.26}$$

From now on, we will use the notation $sin(Jt) = S, cos(Jt) = C$ for brevity. The values of the number operator $\hat{n}_k(t)$ for the first iteration are

k	$\hat{n}_k(t)$	$\hat{n}_k(t=\pi/4)$	$\hat{n}_{k,exact}$	$\hat{n}_{k,DMRG}$
0	1	1	0.9947093572	1
1	1	1	0.9328354870	1
2	C^2	0.5	0.6113489711	0.5
3	S^2C^2	0.25	0.3886510289	0.25
4	S^4	0.25	0.0671645130	0.25
5	0	0	0.0052906428	0

(4.27)

while for the second iteration, $|\psi(t)\rangle_2 = \hat{U}_{34}\hat{U}_{45}\hat{U}_{23}\hat{U}_{01}\hat{U}_{12}\,|\psi(t)\rangle_1$:

k	$\hat{n}_k(t)$	$\hat{n}_k(t=\pi/2)$	$\hat{n}_{k,exact}$	$\hat{n}_{k,DMRG}$
0	$C^8 + 7C^4S^4 + 3C^2S^4$	0.8750000	0.8204117259	0.875
1	$C^8 + C^2(3C^2S^2 + S^2 + 2S^6 + S^8) + S^{10}$	0.8750000	0.6436187881	0.875
2	$C^2(S^4 + S^6 + C^6)$	0.2500000	0.5438765365	0.25
3	$4C^8S^2 + C^8S^4 + S^8$	0.2031250	0.4561234635	0.1875
4	$C^2S^4(1+2C^2)^2 + C^2S^6(1+C^2)^2 + C^2S^8 + S^{14}$	0.6796875	0.3563812119	0.6875
5	S^6	0.1250000	0.1795882741	0.125

(4.28)

where $\hat{n}_{k,exact}$ is the exact number operator of equation (4.16), $\hat{n}_{k,DMRG}$ is the numerical result of the program and $\hat{n}_k(t)$ is the hand-evaluated number operator within the Trotter approximation. Note that for $\hat{n}_k(t = \pi/2)$ we set the step number equal to 2, so that $jt = \pi/4$. The comparison between the hand-evaluated values of $\hat{n}_k$ and the computed ones are shown in table, and are a good match. The initial states are not exactly 1 because we add a little bit of energy to every states to ease the following diffusion.

As we can see, the difference between $\hat{n}_k(t = \pi/2)$ and $\hat{n}_{k,exact}$ is perceptible only after the second iteration, while $\hat{n}_{k,DMRG}$ suffer from the DMRG errors and set up even at the first iteration, although its trend is compatible with the exact solution.

Same behaviour is observed for a the larger system $L = 8$, with ground state

$$|\psi_0\rangle = |11110000\rangle \tag{4.29}$$

which evolve as first iteration as

$$\begin{aligned}
|\psi(t)\rangle &= \hat{U}_{56}\hat{U}_{67}\hat{U}_{45}\hat{U}_{23}\hat{U}_{01}\hat{U}_{12}\hat{U}_{34}\,|11110000\rangle \\
&= |11110000\rangle\, cos(jt) + i\,|11101000\rangle\, sin(jt)cos^2(jt) \\
&- |11100100\rangle\, sin^2(jt)cos^2(jt) - i\,|11100010\rangle\, cos(jt)sin^3(jt) \\
&- |11011000\rangle\, cos(jt)sin^2(jt) - i\,|11010100\rangle\, cos(jt)sin^3(jt) \\
&+ |11010010\rangle\, sin^4(jt)
\end{aligned} \tag{4.30}$$

The number operator values are

k	$\hat{n}_k(t)$	$\hat{n}_k(t = \pi/4)$	$\hat{n}_{k,exact}$	$\hat{n}_{k,DMRG}$
0	1	1	0.9997903835	1
1	1	1	0.9950377931	1
2	$C^2(S^2+1)$	0.75	0.9326804079	0.75
3	$C^2 + S^4$	0.75	0.6113929435	0.75
4	C^2S^2	0.25	0.3886070565	0.25
5	C^2S^4	0.125	0.0673195921	0.125
6	S^6	0.125	0.0049622069	0.125
7	0	0	0.0002096165	0

(4.31)

The second iteration bring the number of terms of $|\psi(t)\rangle$ to 47, and became quite hard to follow manually the time evolution. However, for $k = 7$, the second iteration gives

k	$\hat{n}_k(t)$	$\hat{n}_k(t = \pi/2)$	$\hat{n}_{k,exact}$	$\hat{n}_{k,DMRG}$
7	$C^8S^4 + 3C^2S^{10} + C^6(S^{10} + S^{12}) + S^{16}(1 + C^2 + C^4S^2)$	0.0668945	0.0308786915	0.0625

(4.32)

4.3 Error Analysis

Two main sources of error occur in the adaptive t-DMRG.

(i) The *Trotter error* due to the Trotter decomposition, which for a nth-order decomposition is of order Ldt^{n+1} in one time step dt. To reach a given time t one has to performe t/dt time-steps, such that in the worst case the error grows linearly in time t and the resulting error is of order $L(dt)^n t$.

(ii) The DMRG *truncation error* due to the representation of the time-evolving quantum state in reduced Hilbert spaces and to the repeated transformations between different truncated basis sets. While the truncation error ϵ that sets the scale of the error of the wave function and operators is typically very small, here it will strongly accumulate as $O(Lt/dt)$ truncations are carried out up to time t. This is because the truncated DMRG wave function has norm less than one and is renormalized at each truncation by a factor of $(1 - \epsilon)^{-1} > 1$. Truncations errors should therefore accumulate roughly exponentially with an exponent of $\epsilon Lt/dt$, such that eventually the adaptive t-DMRG will break down at too long times. The accumulated truncation error should decrease considerably with an increasing number of kept DMRG states m. For a fixed time t, it should decrease as the Trotter time step dt is increased, as the number of truncations decreses with the number of time steps t/dt.

As measure for the overall error we consider the *number deviation*, the maximum deviation of the number operator found by DMRG from the exact result,

$$err(t) = \max |\langle \hat{n}_{k,exact}(t) \rangle - \langle \hat{n}_{k,DMRG}(t) \rangle| \tag{4.33}$$

shown in figure (4.1).

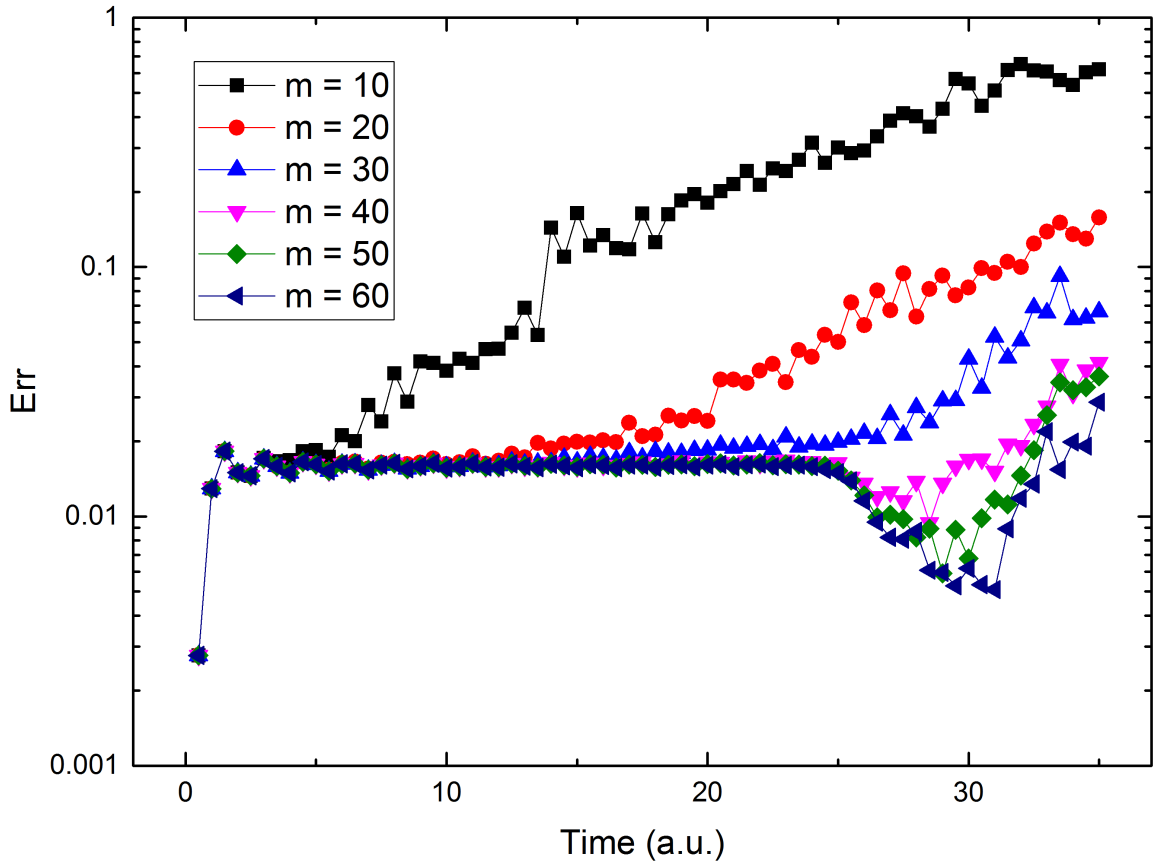


Figure 4.1: err(t) for a system of L=50 sites, dt=0.05 for various number of kept DMRG states m. The parameters were set as follow: time zips =10, timestep=0.5 and number of steps =70

In order to control Trotter and truncation error, two DMRG control parameters are available, the number of DMRG states m and the time step dt.

The dependence on dt is twofold: on one hand, decresing dt reduces the Trotter error by some power of dt^n; on the other hand, the number of truncations increases, such that the truncation error is enhanced. It is therefore not a good strategy to choose dt as small as possible. The truncation error can however be decreased by increasing m.

Consider the dependence of $err(t)$ on the number m of DMRG states. In figure (4.1), $err(t)$ is plotted for a fixed Trotter time step $dt = 0.05$ and different values of m. We can see that a m-dependent "runaway time" t_R separate two regimes: for $t < t_R$, the deviation grows essentially linearly in time and is independent of m, for $t > t_R$ it

suddenly starts to grow more rapidly than any power-law, as expected for the truncation error. As $m \to \infty$ corresponds to the complete absence of the truncation error, the m-independent bottom curve of figure (4.1) is a measure for the deviation due to the Trotter error alone, and the runaway time can be read off very precisely as the moment in time when the truncation error starts to dominate.

That the crossover from a dominating Trotter error at short times and a dominating truncation error at long times is so sharp may seem surprising at first, but can be explained easily by observing that the Trotter error grows only linearly in time, but the accumulated truncation error grows almost exponentially in time.

Figure (4.2) shows the time evolution of a $L = 50$ system obtained by the Runge-Kutta time step targetting method, the first algorithm implemented on the program.

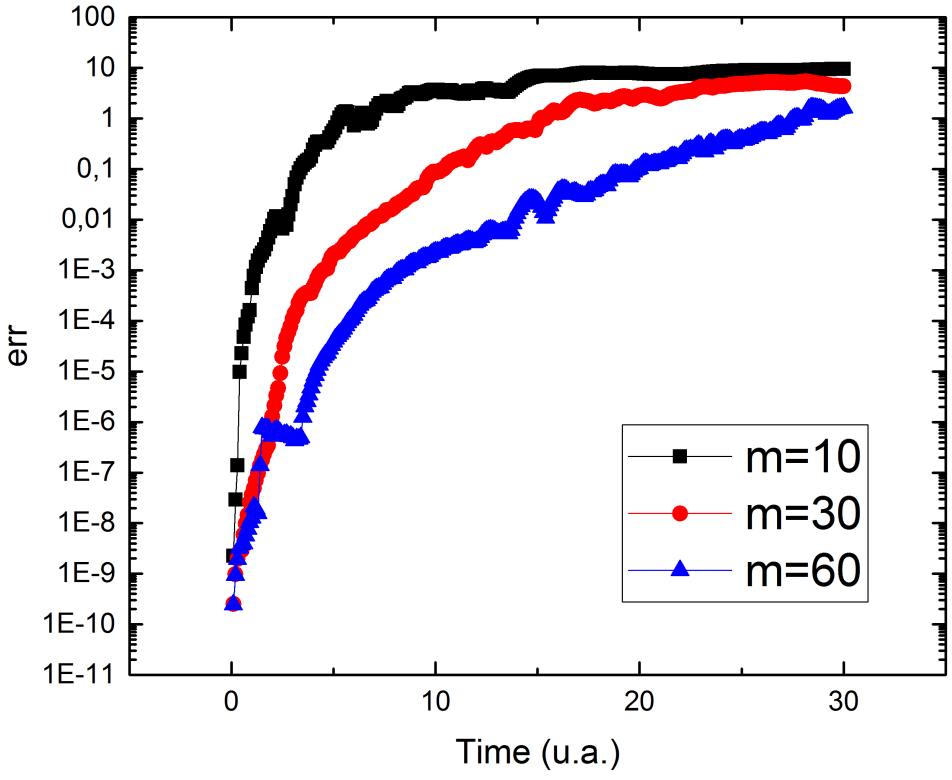


Figure 4.2: Runge-Kutta's time evolution err(t) for a system of L=50 sites, for various number of kept DMRG states m

A direct comparison of both algorithm is shown into figure (4.3).

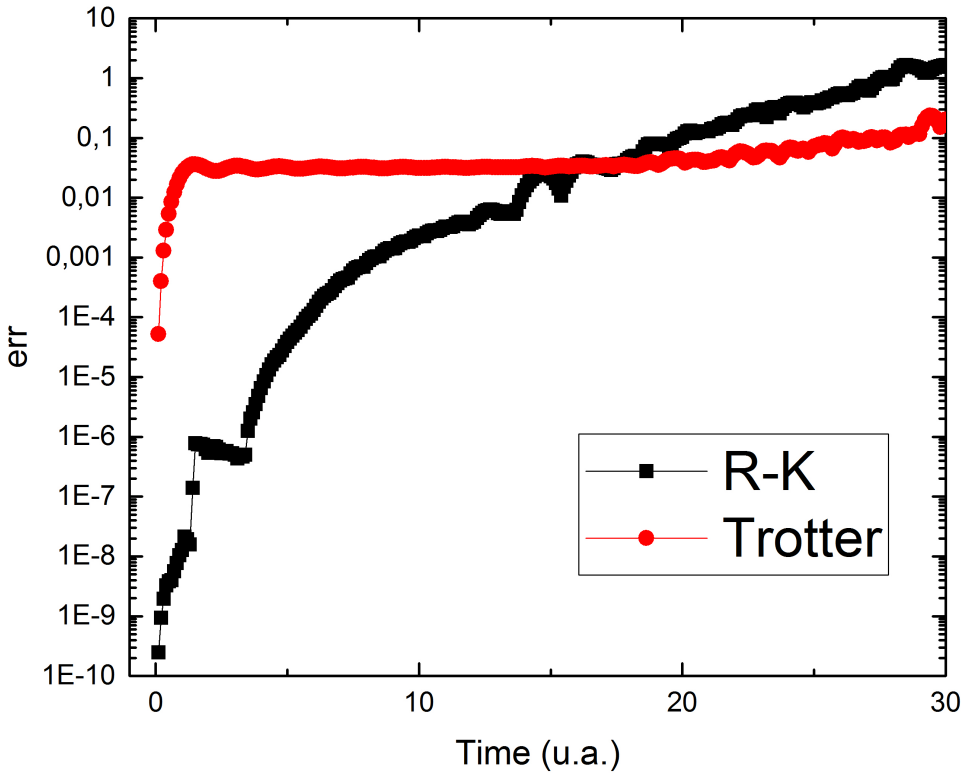


Figure 4.3: Runge-Kutta and Trotter's time evolution err(t) for a system of L=50 sites, for $m = 60$ kept DMRG states

The R-K method provides very accurate results for short amount of time, but the unitarity's loss implies an exponential error growth. The Trotter's method, on the other hand has a bigger initial error due to Trotter's truncation, but it gives a more stable evolution, altough after a longer time (up to 2 times with respect to R-K) the truncation error ruin the data's goodness. For short period of time, the Runge-Kutta method is more suitable, while in medium range time lenght the Trotter method has better performace. Long time evolution are actually out of range of both algorithm.

Conclusion

In this thesis we realized a functioning C++ code that performe the time evolution of quantum system in which can be applied the nearest neighbour approximation; the task has been achieved employing the time evolving block decimation method, jointly with the Trotter's decomposition.

To test and check the algorithm, an extensive simulation of an exactly solved system, the free spinless fermions chain with open boundaries, has been computed.

The results showed the tight correlations between the number of kept DMRG states m and the truncation error, even if actually it is not found an exact expression for this dependence.

As a future work, it would be interesting to focus on the code's optimization, with a deepened look to the bottle necks of the algorithm, like the matrix product (already optimized by Professor Ortolani). Moreover,considering the massive diffusion of multicore computers, a systematic algorithm's parallelization will be necessary to make use of the full computing capacity of future machines.

In conclusion, the t-DMRG code based on the Trotter decomposition has been successfully implemented on the pre-existing simulation program. The actual DMRG code can handle both long ranged and nearest neighbour interactions respectively with the Rounge-Kutta time step targetting and the Trotter's time evolving block decimation.

Acknowledgments

I would like to thank Prof E. Ercolessi and Prof. F. Ortolani for teaching and support. In particular, I'd like to thank Prof. Ortolani for the time he spent and the patience he showed in teaching me many aspects of computational and theoretical physics.

Thank you, it was fun.

Bibliography

[1] S. R. White, "Density matrix formulation for quantum renormalization groups," *Phys. Rev. Lett.*, vol. 69, pp. 2863–2866, Nov 1992.

[2] S. R. White, "Density-matrix algorithms for quantum renormalization groups," *Phys. Rev. B*, vol. 48, pp. 10345–10356, Oct 1993.

[3] S. White, "Physics report 301, 187 (1998). u. schollwoeck," *Rev. Mod. Phys*, vol. 77, p. 259, 2005.

[4] S. R. White and D. A. Huse, "Numerical renormalization-group study of low-lying eigenstates of the antiferromagnetic s=1 heisenberg chain," *Phys. Rev. B*, vol. 48, pp. 3844–3852, Aug 1993.

[5] C. Zhang, E. Jeckelmann, and S. R. White, "Density matrix approach to local hilbert space reduction," *Phys. Rev. Lett.*, vol. 80, pp. 2661–2664, Mar 1998.

[6] T. Nishino and K. Okunishi, "Corner transfer matrix renormalization group method," *Journal of the Physical Society of Japan*, vol. 65, no. 4, pp. 891–894, 1996.

[7] T. Nishino and K. Okunishi, "Corner transfer matrix algorithm for classical renormalization group," *Journal of the Physical Society of Japan*, vol. 66, no. 10, pp. 3040–3047, 1997.

[8] I. Peschel, M. Kaulke, and Ö. Legeza, "Density-matrix spectra for integrable models," *arXiv preprint cond-mat/9810174*, 1998.

[9] S. R. White and R. L. Martin, "Ab initio quantum chemistry using the density matrix renormalization group," *The Journal of chemical physics*, vol. 110, no. 9, pp. 4127–4130, 1999.

[10] T. Xiang, "Density-matrix renormalization-group method in momentum space," *Physical Review B*, vol. 53, no. 16, p. R10445, 1996.

[11] S. Moukouri and L. Caron, "Thermodynamic density matrix renormalization group study of the magnetic susceptibility of half-integer quantum spin chains," *Physical review letters*, vol. 77, no. 22, p. 4640, 1996.

[12] X. Wang and T. Xiang, "Transfer-matrix density-matrix renormalization-group theory for thermodynamics of one-dimensional quantum systems," *Physical Review B*, vol. 56, no. 9, p. 5061, 1997.

[13] R. Bursill, T. Xiang, and G. Gehring, "The density matrix renormalization group for a quantum spin chain at non-zero temperature," *Journal of Physics: Condensed Matter*, vol. 8, no. 40, p. L583, 1996.

[14] A. Juozapavičius, S. Caprara, and A. Rosengren, "Quantum ising model in a transverse random field: A density-matrix renormalization-group analysis," *Physical Review B*, vol. 56, no. 17, p. 11097, 1997.

[15] S. R. White and D. Scalapino, "Density matrix renormalization group study of the striped phase in the 2d t- j model," *Physical review letters*, vol. 80, no. 6, p. 1272, 1998.

[16] H. Otsuka, "Density-matrix renormalization-group study of the spin-1/2 xxz antiferromagnet on the bethe lattice," *Physical Review B*, vol. 53, no. 21, p. 14004, 1996.

[17] M.-B. Lepetit, M. Cousy, and G. Pastor, "Density-matrix renormalization study of the hubbard model [4] on a bethe lattice," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 13, no. 3, pp. 421–427, 2000.

[18] K. G. Wilson, "The renormalization group: Critical phenomena and the kondo problem," *Reviews of Modern Physics*, vol. 47, no. 4, p. 773, 1975.

[19] M.-B. Lepetit and G. Pastor, "Dimerization of polyacetylene using a distance-dependent hubbard model and the density-matrix renormalization-group method," *Physical Review B*, vol. 56, no. 8, p. 4447, 1997.

[20] G. Golub and C. Van Loan, "The singular value decomposition and unitary matrices," *Matrix Computations*, pp. 70–71, 1996.

[21] S. Östlund and S. Rommer, "Thermodynamic limit of density matrix renormalization," *Physical review letters*, vol. 75, no. 19, p. 3537, 1995.

[22] S. Liang and H. Pang, "Approximate diagonalization using the density matrix renormalization-group method: A two-dimensional-systems perspective," *Physical Review B*, vol. 49, no. 13, p. 9214, 1994.

[23] D.-M. Renormalization, "I. peschel, x. wang, m. kaulke and k. hallberg," *Lecture Notes in Physics*, vol. 528, 1999.

[24] S. R. White, "Spin gaps in a frustrated heisenberg model for cav 4 o 9," *Physical review letters*, vol. 77, no. 17, p. 3633, 1996.

[25] K. Wu, A. Canning, H. Simon, and L.-W. Wang, "Thick-restart lanczos method for electronic structure calculations," *Journal of Computational Physics*, vol. 154, no. 1, pp. 156–173, 1999.

[26] K. Wu and H. Simon, "Thick-restart lanczos method for large symmetric eigenvalue problems," *SIAM Journal on Matrix Analysis and Applications*, vol. 22, no. 2, pp. 602–616, 2000.

[27] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations," *Physical Review Letters*, vol. 91, no. 14, p. 147902, 2003.

[28] A. J. Daley, C. Kollath, U. Schollwöck, and G. Vidal, "Time-dependent density-matrix renormalization-group using adaptive effective hilbert spaces," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2004, no. 04, p. P04005, 2004.

[29] S. R. White and A. E. Feiguin, "Real-time evolution using the density matrix renormalization group," *Physical review letters*, vol. 93, no. 7, p. 076401, 2004.

[30] H. Luo, T. Xiang, and X. Wang, "Comment on time-dependent density-matrix renormalization group: a systematic method for the study of quantum many-body out-of-equilibrium systems," *Physical review letters*, vol. 91, no. 4, p. 049701, 2003.

[31] A. E. Feiguin and S. R. White, "Time-step targeting methods for real-time dynamics using the density matrix renormalization group," *Physical Review B*, vol. 72, no. 2, p. 020404, 2005.

[32] M. Suzuki, "Relationship between d-dimensional quantal spin systems and (d+ 1)-dimensional ising systems equivalence, critical exponents and systematic approximants of the partition function and spin correlations," *Progress of theoretical physics*, vol. 56, no. 5, pp. 1454–1469, 1976.

[33] H. F. Trotter, "On the product of semi-groups of operators," *Proceedings of the American Mathematical Society*, vol. 10, no. 4, pp. 545–551, 1959.

[34] G. De Chiara, M. Rizzi, D. Rossini, and S. Montangero, "Density matrix renormalization group for dummies," *Journal of Computational and Theoretical Nanoscience*, vol. 5, no. 7, pp. 1277–1288, 2008.

[35] U. Schollwöck, S. R. White, G. Batrouni, and D. Poilblanc, "Methods for time dependence in dmrg," in *AIP Conference Proceedings*, vol. 816, pp. 155–185, AIP, 2006.