UNIVERSITY OF PADUA
PHYSICS AND ASTRONOMY DEPARTMENT

Homework #10
Real Space Renormalization Group algorithm

Student name: *Beatrice Segalini* – 1234430

Course: *Quantum Information and Computing 2020* – Professor: *S. Montangero*
Due date: *January 11th, 2021*

Abstract

In this report, the Real Space Renormalization Group algorithm is developed and applied to the 1-dimensional Quantum Ising Hamiltonian in a transverse field with nearest neighbour interaction. The ground state energy as a function of the transverse field λ is studied and the performance of the code are evaluated for different parameters; the results obtained are compared with the mean field solution.

Theory

Hamiltonian of the system. The Hamiltonian $\mathcal{H}$ of the considered spin system is given by:

$$\mathcal{H} = \lambda \sum_{i=1}^{N} \sigma_z^i + \sum_{i=1}^{N-1} \sigma_x^i \sigma_x^{i+1} \tag{1}$$

where σ_x, σ_z are the Pauli matrices:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2}$$

and i is the index of the spin considered.

As a matter of facts, this notation simplifies the one of a *tensor product* that can be fully written as:

$$\sigma_k^i = \mathbb{1}^1 \otimes \cdots \otimes \mathbb{1}^{i-1} \otimes \sigma_k^i \otimes \mathbb{1}^{i+1} \otimes \cdots \otimes \mathbb{1}^N \tag{3}$$

with $\mathbb{1}$ the 2×2 identity, meaning that the complete Hamiltonian has size $2^N \times 2^N = N_0 \times N_0$.

As thoroughly described in Homework #9, the system considered shows a *quantum phase transition* for $\lambda \approx 1$: in fact one can distinguish an ordered phase for $\lambda \ll 1$ and a disordered one for $\lambda \gg 1$.

The Mean Field (MF) approximation predicts the quantum phase transition, but for $\lambda = 2$. The ground state energy, computed as the expected value of $\mathcal{H}$, has a dependence from λ given by the following relationship:

$$E_{gs} = \begin{cases} -1 - \dfrac{\lambda^2}{4} & \lambda \in [-2; 2] \\ -|\lambda| & \lambda \notin [-2; 2] \end{cases} \tag{4}$$

This theoretical result will be compared to the ones obtained from the Real Space Renormalization Group algorithm.

The Real Space Renormalization Group algorithm. The Real Space Renormalization Group (RSRG) method has been developed in order to efficiently approximate the diagonalization process for Hamiltonians of big dimensions.

In fact, the size of the Hamiltonian of the quantum Ising model becomes exponentially big when the number of particles N increases, hence diagonalizing exactly the $\mathcal{H}$ matrix is too computationally expensive.

The key idea is to build a system of size $2N$ from a truncated description of an original system of size N, under the assumption that the ground state of the bigger system is composed of *few* low energy eigenstates of each subsystem of size N.

The algorithm performs the following steps:

1. Starting from a system of size N with Hamiltonian $\mathcal{H}_N$, a compound system of size $2N$ is built by *doubling* the original one, and considering the interactions between the two replicas. The Hamiltonian will then be equal to:
$$\tilde{\mathcal{H}} = \mathcal{H}_N^A + \mathcal{H}_N^B + \mathcal{H}_{interaction}^{AB} \tag{5}$$
where $\mathcal{H}_N^A = \mathcal{H} \otimes \mathbb{1}_{N_0}$, $\mathcal{H}_N^B = \mathbb{1}_{N_0} \otimes \mathcal{H}$ (with $\mathbb{1}_{N_0}$ the $N_0 \times N_0$ identity) and the interaction term is given by the interaction between the leftmost spin of block A and the rightmost spin of block B, i.e.:
$$\mathcal{H}_{interaction}^{AB} = A \otimes B = \left(\mathbb{1}^1 \otimes \cdots \otimes \mathbb{1}^{N-1} \otimes \sigma_x\right) \otimes \left(\sigma_x \otimes \mathbb{1}^2 \otimes \cdots \otimes \mathbb{1}^N\right) \tag{6}$$
$\tilde{\mathcal{H}}$ will then be a matrix of size $2^{2N} \times 2^{2N} = N_0^2 \times N_0^2$.

2. The Hamiltonian $\tilde{\mathcal{H}}$ is diagonalized and projected into a truncated space spanned by the N_0 smallest eigenvalues through the projector $\hat{P}$, which is simply the $N_0^2 \times N_0$ matrix of the first N_0 eigenvectors. The matrix $\mathcal{H}_N$ is so obtained:
$$\mathcal{H}_N = \hat{P}^T \tilde{\mathcal{H}} \hat{P} \tag{7}$$
The physical quantities desired (i.e., the ground state energy) are computed in this step.

3. The interaction terms A, B also need to evolve since the system has changed its dimension and then the change of basis has to be performed through operator $\hat{P}$. Hence, two new matrices are defined:
$$\tilde{A} = A \otimes \mathbb{1}_{N_0}; \quad \tilde{B} = \mathbb{1}_{N_0} \otimes B \tag{8}$$
and then projected via $\hat{P}$:
$$A = \hat{P}^T \tilde{A} \hat{P}; \quad B = \hat{P}^T \tilde{B} \hat{P} \tag{9}$$

4. $\mathcal{H}_N$, A and B can be used in point 1 to compute a new $\tilde{\mathcal{H}}$ and to iterate the process. The algorithm continues to run until a stopping condition is satisfied, which in this report is when the variation of the energy of the ground state becomes smaller that a certain ϵ.

Code development

Computing tensor products. In addition to the functions for computing tensor products defined in the previous assignment, a more general function is implemented to perform such calculation, which can be found in Listing 1 below.

Listing 1: Function to compute the tensor product $M_1 \otimes M_2$.

```
function tens_prod(M1, M2)result(tp)
! Kronecker product M1 (X) M2
double complex, dimension(:,:), intent(IN)  :: M1
double complex, dimension(:,:), intent(IN)  :: M2

integer                                     :: dim_1, dim_2
double complex, dimension(:,:), allocatable :: tp

dim_1 = size(M1, 1)
dim_2 = size(M2, 1)

allocate(tp(dim_1*dim_2, dim_1*dim_2))
```

```
! Kronecker product
do aa = 1, dim_1
  do bb = 1, dim_1
    tp(dim_2*(aa-1)+1:dim_2*aa, dim_2*(bb-1)+1:dim_2*bb ) = M1(aa, bb)*M2
  end do
end do

end function tens_prod
```

Projecting on subspace defined by $\hat{P}$. The projection operation key to the RSRG algorithm is performed through the `Projection` subroutine. The subroutine includes a check on input sizes which returns an error message if the operation is not possible, moreover it checks for previous allocation of output matrix $\tilde{M}$, and reallocates it with the correct dimensions.

Listing 2: Subroutine to project via $\hat{P}$ a matrix M, obtaining $\tilde{M}$ as in Eq. 7.

```
subroutine Projection(P, M, M_tilde)

double complex, dimension(:,:), intent(IN)                :: P, M
double complex, allocatable, dimension(:,:), intent(OUT) :: M_tilde

if (((size(P, 1) .eq. size(M, 2)) .and. (size(M, 2) .eq. size(M, 1))))then
  if ( allocated(M_tilde) ) then
    deallocate(M_tilde)
  end if
  allocate( M_tilde( size(P,2), size(P,2) ) )
  M_tilde = matmul( matmul( transpose(P), M), P )
else
  print*, shape(P)
  print*, shape(M)
  print*, "Incompatible dimensions"
  stop
end if

end subroutine Projection
```

RSRG algorithm. The matrix $\mathcal{H}$ is initialised with the same code described in Homework #9, following the Ising model Hamiltonian of Eq. 1. Then, after the initialization of the main parameters, the RSRG loop can start.

At each iteration, `H_tilde`, `A_tilde` and `B_tilde` are first computed with the `tens_prod` user defined function. Afterwards, eigenvalues and eigenvectors are calculated via the `findkEigenvalue` function, which uses the `LAPACK` subroutine `DSYEVR` to derive the first `mm=d**N` (N_0) eigenvalues and eigenvectors, stored in ascending order in `evls` e `evct`.

To take into account of the growth of system size, the support variable `fsize` is multiplied by 2 at each iteration and it is used to properly normalize the value of the ground state energy, i.e. the first eigenvalue, stored in the support variable `supp_gs`.

Operator `P` is hence built converting in double complex format the `evct` array, so that the projection step of the algorithm can be executed. `H`, `A`, `B` values are hence updated and the loop restart after incrementing the iteration number `n_it`.

The algorithm stops when the absolute value of the difference between `supp_gs` and `ground_state` is smaller than `eps`, a threshold set to 10^{-12}.

Listing 3: RSRG algorithm core implementation.

```
do while(abs(ground_state - supp_gs) > eps)
  ground_state = supp_gs
  ! build new Hamiltonian and interaction operators
  H_tilde = tens_prod(identity(d**N), H) + tens_prod(H, identity(d**N)) + &
            + tens_prod(A, B)
  A_tilde = tens_prod(A, identity(d**N))
  B_tilde = tens_prod(identity(d**N), B)
  ! double system size
  fsize = 2 * fsize
  ! diagonalization and computation of eigenvalues/eigenvectors
  call findkEigenvalue(H_tilde, evls, mm, evct)
  supp_gs = evls(1)/dble(fsize)
  ! build operator P
  P = dcmplx(evct)
  !projection on the subspace
  call Projection(P, H_tilde, H)
  call Projection(P, A_tilde, A)
  call Projection(P, B_tilde, B)
  ! increment iteration number
  n_it = n_it + 1
end do
```

The main physical quantity analysed is the ground state energy, the program furthermore stores the final system size `fsize`, the number of iterations to arrive to convergence `n_it` and the execution time. The RSRG loop is run via a `Python` script for different number of spins $N = 2, 3, 4, 5$ and for several values of λ so that the interval $[0, 3]$ is thoroughly studied.

Results

Ground state energy. The core point of the analysis consists of studying the behaviour of the ground state energy as a function of the external field strength λ. To do so, the graphs in Figure 1 are produced.

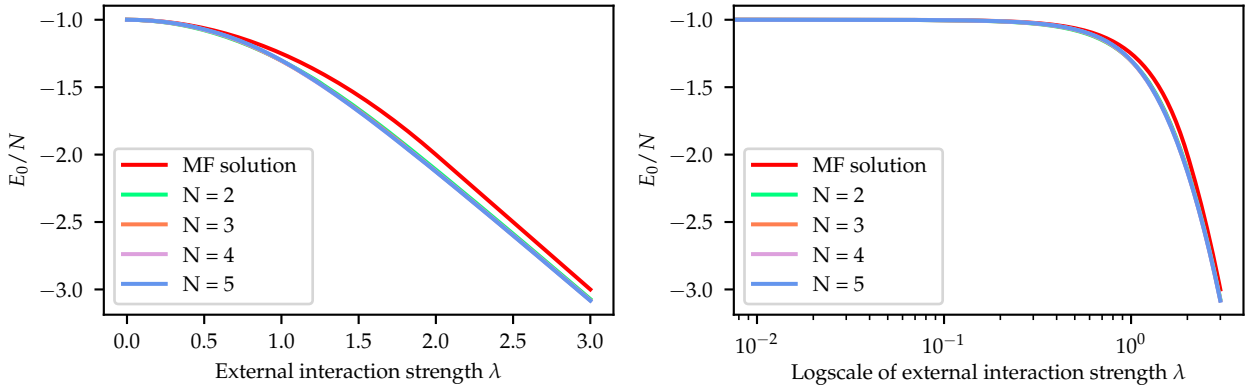

Figure 1: Ground state energy per particle as a function of λ, with $N \in [2:5]$, compared with the MF solution.

As one can easily notice, despite starting with a different number of particles, all the computed values tend to overlap and to produce consistent results: the diverging tendency that was observed in the previous assignment is not present. The quantum phase transition at $\lambda = 1$ is still noticeable as expected, and it is especially evident in the graph on the right, where the log-scale is used for the λ-axis. However, it is clear that there is a discrepancy with the MF approximation, as anticipated by the fact that indeed it is just an approximation for the thermodynamic limit.

Convergence of the algorithm. The number of iterations needed to arrive to convergence is then studied. As already mentioned, the RSRG loop implemented stops when the value of the ground state energy varies of less than $\epsilon = 10^{-12}$. In Figure 2, the number of iterations as a function of λ, for each starting value of N considered, is plotted on the left, while on the right, the average number of iterations with their error is represented.

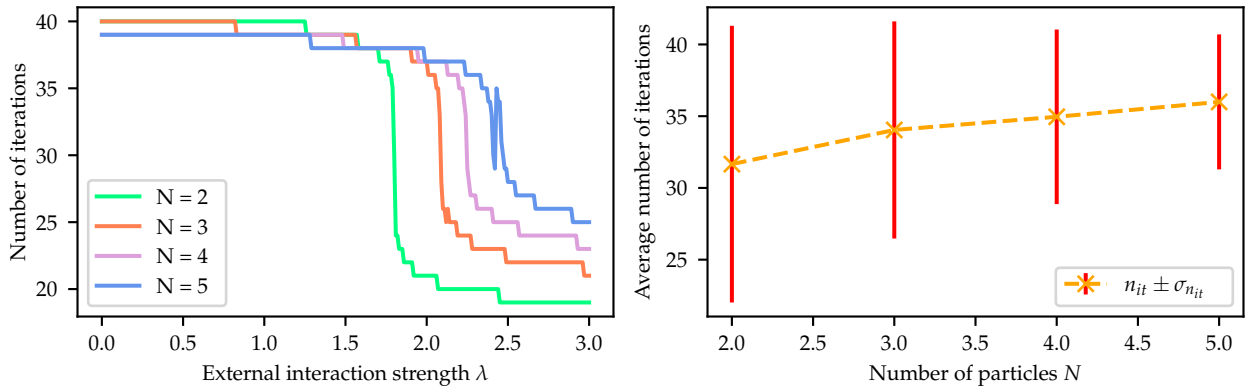

Figure 2: On the left: number of iteration to arrive to convergence as a function of λ, for different Ns. On the right: average number of iterations with standard deviation as a function of the starting value of N.

The most striking feature of the leftmost graph is the sudden decrease in the number of iterations with the increase of λ: for every value of N considered, for $\lambda \gtrsim 1.5$, the necessary number of iterations almost halves. In any case, 40 iterations are enough for the algorithm to converge for every N: this can be seen clearly on the graph on the right, in which we can observe that the average number of iterations is always around 35, with a maximum standard deviation of approximately 10 iterations. One can conclude that the algorithm has good convergence properties and that it needs about 40 iterations to give good results.

Execution time analysis. Finally, the execution time has been studied.
After computing with the `FORTRAN` intrinsic function `cpu_time` the time needed to arrive to convergence for each value of λ, the average execution time has been derived with its standard deviation for each starting size of the system N, obtaining the results displayed in Figure 3.
It is immediately noticeable that the graphs show an exponential growth with N increasing, making it impossible to run the program with bigger system sizes.

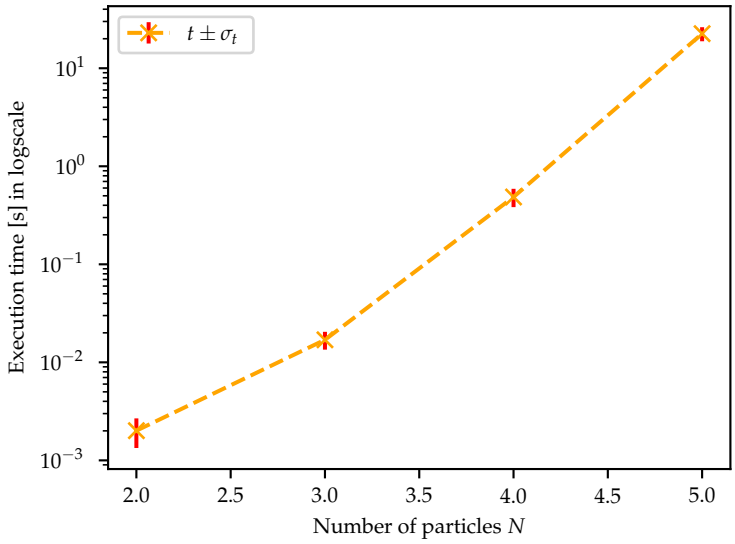

Figure 3: Average execution times in log-scale as a function of N.

Self evaluation

All the implementation analysed produced results consistent with the theory, so one can conclude that the program worked properly. The code written is sufficiently flexible and can handle some allocation or size issues automatically, but a more precise debugging implementation could be added.

To conclude, a more deep analysis might be carried out concerning the excited states and their degeneracy, also considering a wider range of λs.