Real-Space Renormalization Group

Abstract

In this work, given the quantum Ising Hamiltonian in transverse field on a one dimensional lattice with nearest neighbor interaction, we compute the ground state energy as a function of the transverse field λ by means of the real-space Renormalization Group algorithm.

1 Theory

Real-Space Renormalization Group (RSRG) method is an approximation method based on the hypothesis that the ground state of a system is composed of low-energy states of the system (non-interacting) bipartitions. Based on this assumption, RSRG allows to describe the ground state properties of many-body quantum systems with large sizes N.
In this work we use RSRG considering N spin-1/2 particles on a one-dimensional lattice, described by the Hamiltonian

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

where

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

are the Pauli matrices and λ is the field strength. The algorithm is the following:

1. Consider a system composed of N sites that can be studied in an exact numerical way and build the Hamiltonian $H_N : \mathbb{C}^{d^N} \to \mathbb{C}^{d^N}$, where d is the dimension of each site, which is supposed to be the same for all of them;

2. Build the compound system of size $2N$ Hamiltonian as follows:

$$H_{2N} = H_N \otimes \mathbb{1}_N + \mathbb{1}_N \otimes H_N + H_{int}, \tag{2}$$

where the interaction term is given by

$$H_{int} = H_L \otimes H_R \tag{3}$$

with

$$H_L = \underbrace{\mathbb{1} \otimes \cdots \otimes \mathbb{1}}_{N-1 \ times} \otimes \sigma_x, \qquad H_R = \sigma_x \otimes \underbrace{\mathbb{1} \otimes \cdots \otimes \mathbb{1}}_{N-1 \ times}; \tag{4}$$

3. Diagonalize H_{2N}, finding its eigenvalues and eigenvectors $H_N = \sum_{i=1}^{d^N} E_i \left|E_i\right\rangle \left\langle E_i\right|$, where the eigenvalues E_i are in increasing order. Find the projector onto the lowest 2^N eigenstates

$$P = \sum_{i=1}^{2^N} \left|E_i\right\rangle \left\langle E_i\right|, \tag{5}$$

which is a $2^{2N} \times 2^N$ matrix that projects the Hilbert space on the subspace spanned by the first 2^N low-energy laying eigenstates. Compute the projected Hamiltonian

$$\tilde{H}_N = P^\dagger H_{2N} P; \tag{6}$$

4. Project H_L and H_R:

$$\tilde{H}_L = P^\dagger (H_L \otimes \mathbb{1}_N) P, \qquad \tilde{H}_R = P^\dagger (\mathbb{1}_N \otimes H_R) P; \tag{7}$$

5. Repeat steps 2-4 until the desired system size is reached or convergence to the renormalization group fixed point is achieved.

The advantage of this algorithm is that, at each step, the dimension of the described system is doubled, while the dimension of the Hamiltonian representation is kept constant to 2^N.

2 Code development

The main program is structured as follows. We use two nested `for` loops: the first running on different initial values of N and the second on different values of λ. Inside the inner loop we initialize the Hamiltonian and we call the `RSRG` function to compute the ground state. For each value of N, we write the results in different output files.
The initialization of the Hamiltonian is carried out by means of the subroutine `IsingHamiltonian`, which constructs H by first computing the diagonal terms and then the interaction ones. Then, the two terms are summed. Inside the file `ising.f90`, two functions containing the RSRG algorithm are present. In the first one (`RSRG`) the number of iteration is fixed, whereas the second one (`RSRG1`) stops when a certain convergence threshold is reached, which is set to $|e_i - e_{i-1}| \leq 10^{-10}$, with e_i being the energy density of the ground state at iteration i. Here we show the second one.

```
function RSRG1(H_N, N, niter) result(gs)
    ! Real space Renormalization Group
    ! input Hamiltonian
    real*8, dimension(:,:) :: H_N
    ! H_2N: double size Hamiltonian
    ! H_2N_L: left part Hamiltonian
    ! H_2N_R: right part Hamiltonian
    ! Htmp: temporary variable to save H_2N
    real*8, dimension(:,:), allocatable :: H_2N, H_2N_L, H_2N_R, Htmp
    ! N: number of subsystems
    ! niter: number of iterations for the RSRG
    ! ii: variable to loop
    ! info: stat flag
    integer :: N, niter, ii, info
    ! projector
    real*8, dimension(:,:), allocatable :: P
    ! Pauli matrix
    real*8, dimension(2,2) :: sigma_x
    ! eigenvalues vector
    real*8, dimension(:), allocatable :: eig
    ! ground state final value
    real*8 :: gs
    ! ground state temporary value
    real*8, dimension(100) :: gs_tmp
    ! define sigma_x
    sigma_x(1, 1) = 0
    sigma_x(1, 2) = 1
    sigma_x(2, 1) = 1
    sigma_x(2, 2) = 0
    ! allocate memory
    allocate(H_2N(2**(2*N), 2**(2*N)), &
             H_2N_L(2**N, 2**N), &
             H_2N_R(2**N, 2**N), &
             Htmp(2**(2*N), 2**(2*N)), &
```

```
            eig(2**(2*N)), stat=info)
    call checkpoint(debug=(info.ne.0), message="Allocation failed!", &
                    end_program=.true.)
    ! build the first H_2N
    H_2N_L = KroneckerProd(idmatr(2**(N-1)), sigma_x)
    H_2N_R = KroneckerProd(sigma_x, idmatr(2**(N-1)))
    H_2N = KroneckerProd(H_N, idmatr(2**N)) + KroneckerProd(idmatr(2**N), H_N)
           + KroneckerProd(H_2N_L, H_2N_R)
    ! main loop
    do ii = 1, niter
        Htmp = H_2N
        ! diagonalize
        call DiagonalizeR(H_2N, eig, info)
        call checkpoint(debug=(info.ne.0), message="Diagonalization failed!", &
                        end_program=.true.)
        gs_tmp(ii) = eig(1)
        if (ii .ne. 1) then
            if (abs(gs_tmp(ii)/(N*2.0**(ii)) - gs_tmp(ii-1)/(N*2.0**(ii-1))) .le.
    1e-10) then
                gs = gs_tmp(ii)/(N*2.0**(ii))
                print *, "Convergence at iteration", ii
                exit
            end if
        end if
        ! build the projector P
        P = H_2N(:, 1:2**N)
        H_2N = Htmp
        ! project: Htilde_N = P^+ H_2N P
        H_N = matmul(transpose(P), matmul(H_2N, P))
        ! build the interaction term
        H_2N_L = matmul(transpose(P),matmul(KroneckerProd(idmatr(2**N),H_2N_L),P))
        H_2N_R = matmul(transpose(P),matmul(KroneckerProd(H_2N_R,idmatr(2**N)),P))
        H_2N = KroneckerProd(H_N, idmatr(2**N)) + KroneckerProd(idmatr(2**N), H_N)
               + KroneckerProd(H_2N_L, H_2N_R)
    end do
    ! deallocate memory
    deallocate(H_2N, H_2N_L, H_2N_R, Htmp, P, eig)
    return
end function RSRG1
```

Inside this function we use three other user-defined functions:

- `KroneckerProd`, which computes the tensor product between two real matrices;
- `idmatr`, which initializes an identity matrix of a specific dimension;
- `DiagonalizeR`, which is a wrapper for the LAPACK `dsyev` subroutine. It diagonalizes a real symmetric matrix.

3 Results

First, from running the code using the `RSRG1` function, we notice that the algorithm takes from 9 to 30 iterations to reach convergence. Given the fact that the number of iterations is set to 100, this check allows us to save time and avoid redundant computations. We also carry out another check, plotting the error with respect to the theoretical value versus the number of RSRG iterations in the case $\lambda = 0$, for which we know that $e_{th} = -1$. The result obtained starting with $N = 2$ is shown in Figure 1. The minimum of the error is reached for $\lambda \sim 45$, so, if we decide to keep the number of iterations fixed, this is a good value to set.

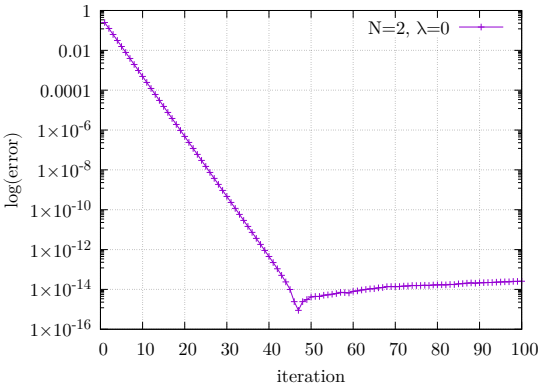
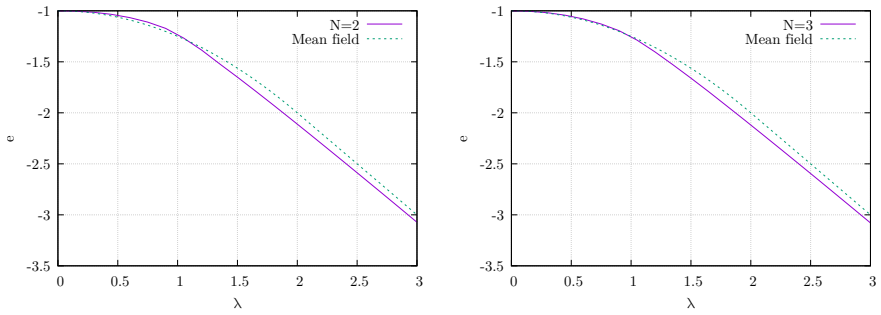


Figure 1: Log-error with respect to the theoretical $\lambda = 0$ ground state value $e_{th} = -1$ as a function of the number of RSRG iterations, starting with $N = 2$.

The results obtained for $\lambda \in [0, 3]$ and $N = 2, 3, 4, 5$ are shown in Figure 2. We know that the exact solution for the ground state energy for $\lambda = 0$ is $E_{\lambda=0} = -(N-1)$, so if we normalize by $N - 1$ we should get $e_{\lambda=0} = -1$. Indeed, this is what we get in all cases.
The results we get seem not to vary that much by increasing the value of the initial size N. In general, we can say that, as we expect, the mean field approximation is good for small values of λ, for which the two curves overlap. By increasing the value of λ, we see that the two curves split; in particular, for $\lambda \sim 2$ the curves reach their maximum distance. We can get a more detailed analysis of the difference with respect to the mean field solution by considering the difference between the two curves (in absolute value), which is shown in Figure 3. Notice that for $\lambda < 1$ the error is much smaller than the one obtained for $\lambda > 1$. Moreover, for $\lambda < 1$ we get smaller errors for high values of N, whereas for $\lambda > 1$ the converse holds. The maximum error is obtained for $\lambda \sim 2$, at which we known that the mean field (wrongly) predicts the phase transition.

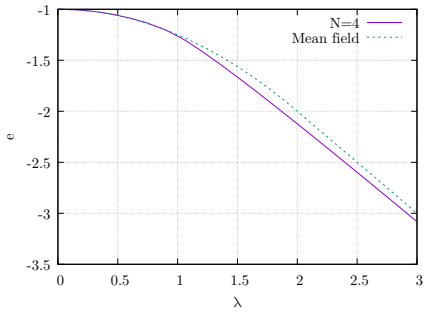
4 Self evaluation

This exercise was very instructive since it made me learn how to implement a vary powerful technique such as Real-Space Renormalization Group.

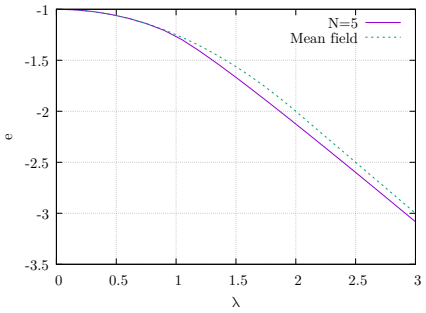


(a) $N = 2$.

(b) $N = 3$.



(c) $N = 4$.



(d) $N = 5$.

Figure 2: Ground state energy density as a function of λ for different values of initial dimension N.

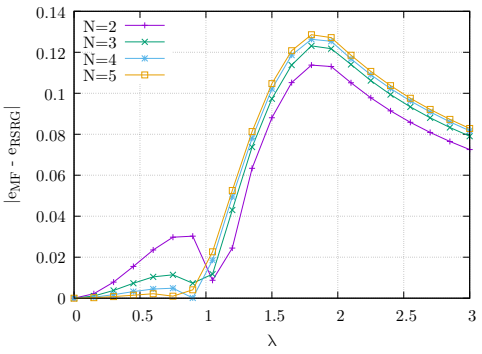


Figure 3: Difference between mean field and RSRG ground state energy density (in absolute value) for different values of initial dimension N.