Quantum Information and Computing 2020/21
Week 10 report

Rocco Ardino
(Dated: Tuesday 5th January, 2021)

In this work we deal with the monodimensional Ising model Hamiltonian and with the Renormalization Group techniques for the problem. In particular, starting from the code for the last week assignment, we implement the RSRG algorithm to find the ground state of the Hamiltonian with an improved accuracy. For the purpose, we exploit the Lapack library for the diagonalization and to find the eigenvalues. Lastly, the ground state energy is plotted with respect to the interaction strength parameter and the result is confronted with the mean field solution.

1 Theory

The monodimensional Ising model describes a linear chain of N spins under the action of a magnetic field. The Hamiltonian describing such a system reads:

$$H = \lambda \sum_{i=1}^{N} \sigma_i^z + \sum_{i=1}^{N-1} \sigma_i^x \sigma_{i+1}^x \quad , \tag{1}$$

where λ is the interaction strength and the σ's are the Pauli matrices, for which the following notation is employed:

$$\sigma_i^z = \mathbb{1}_1 \otimes \cdots \mathbb{1}_{i-1} \otimes \sigma_i^z \otimes \mathbb{1}_{i+1} \otimes \cdots \mathbb{1}_N \tag{2}$$

$$\sigma_i^x \sigma_{i+1}^x = \mathbb{1}_1 \otimes \cdots \mathbb{1}_{i-1} \otimes \sigma_i^x \otimes \sigma_{i+1}^x \otimes \mathbb{1}_{i+2} \otimes \cdots \mathbb{1}_N \quad . \tag{3}$$

In order to get the ground state of H with good accuracy, we can exploit the Real Space Renormalization Group (RSRG) algorithm. This is based on the hypothesis that the ground state of a composite system is a sum of the low energy states of the subsystem. Its application goes through the following steps:

- we consider the Hamiltonian H_N of a subsystem, whose eigenvalues and eigenvectors are known;
- we double this system into a system of $2N$ particles and form its Hamiltonian:

  $$H_{2N} = H_N \otimes \mathbb{1}_2^{\otimes N} + \mathbb{1}_2^{\otimes N} \otimes H_N + H_{\text{int}} \quad , \tag{4}$$

  where H_{int} can be separated in the two components acting on the two subsystems of N particles, namely H_L and H_R, and it reads:

  $$H_{\text{int}} = H_L \otimes H_R \quad ; \tag{5}$$

- we disagonalize H_{2N} and project the first N eigenvectors with lowest eigenvalue into the $N^2 \times N$ matrix P;
- we compute the new Hamiltonian matrix H, and the new left and right interaction matrices H_L and H_R:

  $$H_N = P^\dagger H_{2N} P \tag{6a}$$

  $$H_L = P^\dagger H_L P \quad ; \tag{6b}$$

  $$H_R = P^\dagger H_R P \tag{6c}$$

- we iterate the previous steps with the new H, H_L and H_R, until we reach a desired number of iterations n_{it}, so that the energy density will be given by:

$$\rho_{\text{gs}} = \frac{E_{\text{gs}}}{N \cdot 2^{n_{\text{it}}}} \quad . \tag{7}$$

After applying the RSRG algorithm, the results obtained should be compared with the mean field solution, which reads:

$$E_{\text{gs}}^{\text{mf}}(\lambda) = \begin{cases} -1 - \frac{\lambda^2}{4} & \lambda \in [-2, 2] \\ -|\lambda| & \lambda \notin [-2, 2] \end{cases} \tag{8}$$

In particular, we expect that the ground state energy in the non-interacting case, namely for $\lambda = 0$, should be equal to -1.

2 Code Development

For this work, the module **ising_utils** from the previous assignment has been updated and it contains several functions and subroutines for multiple purposes. In particular:

- **ising_tensor_prod(mat1,mat2)**, for executing the tensor product of the two input matrices `mat1` and `mat2`;
- **ising_identity(N)**, for instantiating the identity matrix for `N` spins, so of dimensions $2^{\mathtt{N}} \times 2^{\mathtt{N}}$;
- **ising_hmat_init(N,L)**, for initializing the Ising Hamiltonian for `N` spins and an interaction strength parameter `L`;
- **ising_hmat_diag(hmat,eigvc,eigvl)**, for diagonalizing the Ising Hamiltonian and returning both the eigenfunctions (in the `eigvc` input variable) and the eigenvalues (in `eigvl`);
- **ising_hmat_rsrg(hmat,hmat_L,hmat_R,N,L)**, for applying a single iteration of the RSRG algorithm, with `hmat` $= H_N$, `hmat_L` $= H_L$ and `hmat_R` $= H_R$;
- **ising_print_hmat_std(hmat,formatted)**, for printing the Ising Hamiltonian matrix entries on standard output and in a formatted way, if specified.

Moreover, another module, namely **cmdline**, is implemented to have a command-line argument parser and for default arguments handling. In particular, we explain the new core functionalities of both the modules in the following subsections. Note that the implementation of `ising_tensor_prod`, `ising_hmat_init` and `ising_hmat_diag` has already been treated in the previous week report, so we will not discuss about it for brevity.

2.1 RSRG algorithm implementation

The code to apply a single iteration of RSRG algorithm is showed in Listing 1. Note how the input matrices H_{2N}, H_L and H_R are updated and stored again in the same input variable.

```
1  subroutine ising_hmat_rsrg(hmat, hmat_L, hmat_R, N)
2      ! input arguments
3      complex(8), dimension(:,:) :: hmat, hmat_L, hmat_R    ! input arguments
4      integer(4) :: N                                       ! input arguments
5
6      complex(8), dimension(2**(2*N), 2**(2*N)) :: hmat_2N, hmat_2N_eigvc
7      complex(8), dimension(2**(2*N), 2**(2*N)) :: hmat_L_int, hmat_R_int
8      complex(8), dimension(2**(2*N), 2**(N)  ) :: hmat_P
9      complex(8), dimension(2**(N)  , 2**(2*N)) :: hmat_PA
10     real(8),    dimension(2**(2*N))           :: hmat_2N_eigvl
11
```

```
    ! initialize 1+2 Hamiltonian
    hmat_2N    = ising_tensor_prod(hmat, ising_identity(N)) + &
                 ising_tensor_prod(ising_identity(N), hmat) + &
                 ising_tensor_prod(hmat_L, hmat_R)
    ! diagonalize 1+2 Hamiltonian
    call ising_hmat_diag(hmat_2N, hmat_2N_eigvc, hmat_2N_eigvl)
    ! project Hamiltonian
    hmat_P     = hmat_2N_eigvc(:,:2**N)
    hmat_PA    = TRANSPOSE(CONJG(hmat_P))
    ! update Hamiltonian
    hmat       = MATMUL(MATMUL(hmat_PA, hmat_2N), hmat_P)
    ! update hmat_L and hmat_R
    hmat_L_int = ising_tensor_prod(hmat_L, ising_identity(N))
    hmat_R_int = ising_tensor_prod(ising_identity(N), hmat_R)
    hmat_L     = MATMUL(MATMUL(hmat_PA, hmat_L_int), hmat_P)
    hmat_R     = MATMUL(MATMUL(hmat_PA, hmat_R_int), hmat_P)
end subroutine ising_hmat_rsrg
```

Listing 1. Implementation of the single iteration of the RSRG algorithm.

2.2 Main program

Lastly, all the functions described before are employed in a main program, in which a command-line argument parser is implemented in order to obtain an easier handling of the parameters of the simulation. Moreover, this functionality is also capable of giving default values to the arguments when they are not given. To make the program more user-friendly, a `-h` help option has been added and in the beginning of the execution all the arguments given are printed along with a flag: `F` if a value is given for it, `T` if default is taken. An example of this functionality and of the way to start the program is showed in Listing 2. A complete list of the options is given in Table 1.

Arg option	Arg name	Arg meaning	Default
-N	nspins	N	2
-L	lambda	λ	1
-I	iterations	n_{it}	10
-h	help	print help	

Table 1. Command-line arguments of the program.

```
$ ./10.o -L 0 -I 100

Running with the following cmdline options:

N =         2    default [T/F]: T
L =       0.0    default [T/F]: F
I =       100    default [T/F]: F

-1.0000000000000324
```

Listing 2. Example of launch and user interface.

After the parser, if no exception is found, a subroutine with the main execution is run, which in order:

- initializes the Ising Hamiltonian H for N spins and for an interaction strength λ;
- initializes H_L and H_R;
- applies n_{it} iterations of the RSRG algorithm;
- prints the final result for the ground state energy on the standard output.

The implementation of these steps is showed in Listing 3. Last but not least, the program must be compiled with the flag `-ffree-line-length-0`.

```
hmat   = ising_hmat_init(N, L)                          ! initialization of H
hmat_L = ising_tensor_prod(ising_identity(N-1), s_x)    ! initialization of H_{L}
hmat_R = ising_tensor_prod(s_x, ising_identity(N-1))    ! initialization of H_{R}

do ii=1,I
    hmat   = hmat   * 0.5d0                             ! rescale to avoid overflow
    hmat_L = hmat_L * SQRT(0.5d0)                       ! rescale to avoid overflow
```

```
8       hmat_R = hmat_R * SQRT(0.5d0)                    ! rescale to avoid overflow
9
10      call ising_hmat_rsrg(hmat, hmat_L, hmat_R, N)    ! apply iterations of RSRG algorithm
11  end do
12
13  call ising_hmat_diag(hmat, eigvc, eigvl)             ! final diagonalization
14
15  write(*, "(g0)") eigvl(1) / N                        ! print results on stdout
```

Listing 3. Core operations performed in the main program.

In order to automatize every step for different parameters for N and λ, a Python script is employed to run multiple batch executions for different sets of input parameters. Then, the results are stored in appropriately labelled files.

3 Results

The main program is run requiring to print the ground state energy after $n_{\text{it}} = 100$. Therefore, we are simulating a system of $N \cdot 2^{n_{\text{it}}+1}$ spins. In particular, we choose $N = 2, 3, 4$ and $\lambda \in [-3, 3]$. The results obtained for this set of parameters are plotted through a Gnuplot script and they are showed in Figure 1 along with the mean field solution.

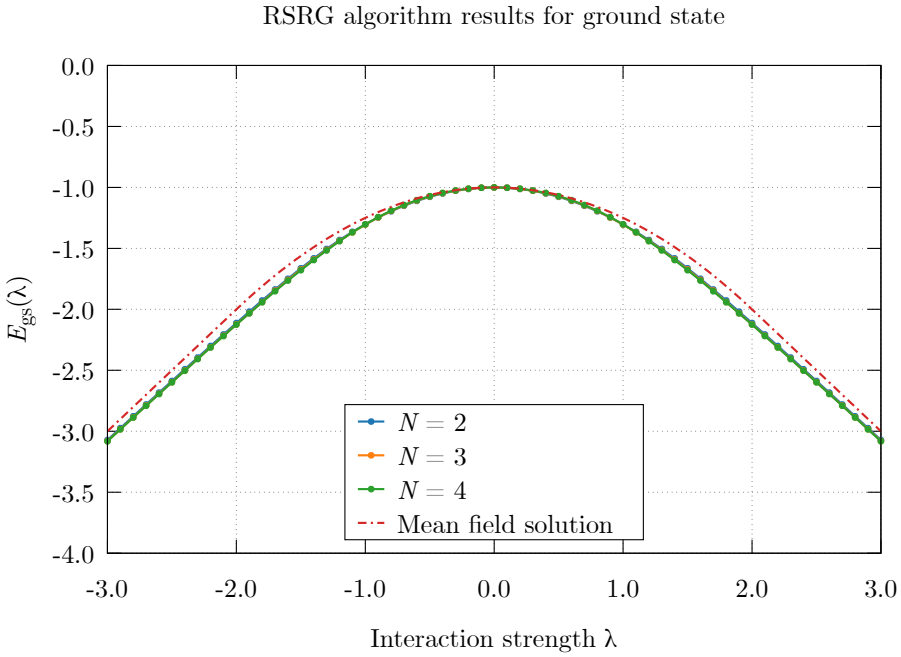

Figure 1. Ground state energy of Ising Hamiltonian depending on the interaction strength λ and for $N = 2, 3, 4$, and comparison with the mean field solution.

What we can observe is that the ground state energy for the non-interacting case, namely for $\lambda = 0$, is -1, as predicted by the mean field solution. We can also observe how the numerical solution obtained from the RSRG algorithm is practically the same for every value of N tested and that these are in agreement with the mean field solution only for small values of λ. When λ is increased, a discrepancy appears as a constant shift in the energy.

4 Self-evaluation

In this work we have successfully tackled the problem of one-dimensional Ising model simulation using the RSRG algorithm. The simulations showed in the results have been run for a number of iterations $n_{\text{it}} = 100$ and for several values of N, showing stable and accurate results. These are compared with the mean field solution, showing a discrepancy for higher values of λ. Further studies should be done in this direction, implementing the Infinite Density Matrix Renormalization Group (DMRG) algorithm and comparing again the results.