Assignment 8

Quantum Information and Computing Course 2022/2023

Massimo Colombo

Department of Physics and Astronomy "Galileo Galilei"
DEGREE IN PHYSICS

January 8, 2023

Renormalization Group: THEORY/algorithms

Considering the 1D-Ising model with **transversal magnetic** field of interaction strength **λ**:

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

It is requested to attack the thermodynamic limit (N $\rightarrow \infty$) problem with two algorithms:

Real space renormalization group:

1. Starting from H_N, the Ising Hamiltonian for N particles (dim $= D^N$, D = local dimension), construct the **Hamiltonian** H_{2N} of the system with $2N$ particles:
   $$H_{2N} = H_N \otimes \mathbb{I}_N + \mathbb{I}_N \otimes H_N + A \otimes B$$
   Where $A = \mathbb{I}_{N-1} \otimes \sigma_x$ $B = \sigma_x \otimes \mathbb{I}_{N-1}$. $A \otimes B$ corresponds to the interaction hamiltonian between two near blocks of N particles. $dim(H_{2N}) = D^{2N}$.
2. **Diagonalize** H_{2N} to find eigenvalues and eigenvectors. Sorting them in increasing order, consider the **first** D^N values. Construct the projector **P** as the matrix of dimension $D^{2N} \times D^N$, having as columns the eigenvectors associated with the considered eigenvalues.
3. **Project** H_{2N} into an effective Hamiltonian $H'_N = P^\dagger H_{2N} P$ and construct $A' = P^\dagger(\mathbb{I}_N \otimes A)P$ and $B' = P^\dagger(B \otimes \mathbb{I}_N)P$
4. **Repeat the first 3** steps substituting H_N, A and B with H'_N, A' and B'.

   At each iteration, we are considering a system with a fixed size but twice the number of particles concerning the previous one. The algorithm allows studying systems with an increasing amount of particles, truncating at each iteration the considered eigenvectors to a fixed number. The reason this should work is a physical assumption.

*The ITERATION stops when the density energy converges up to a value "**eps**"*

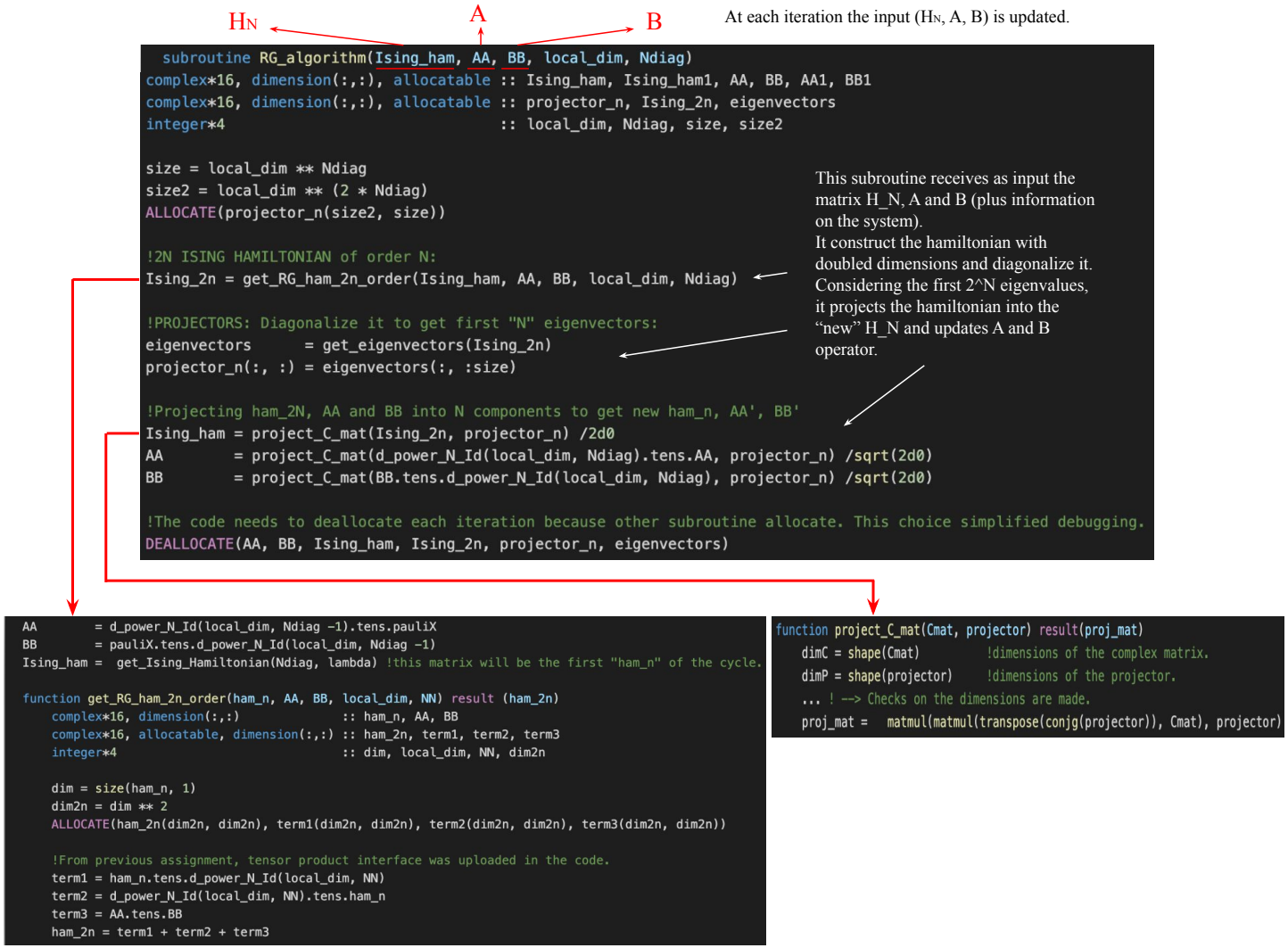
```
DO WHILE (.not.converged)

    CALL RG_algorithm(Ising_ham, AA, BB, local_dim, Ndiag)       !Update(Ising_ham)

    eigenvalues = diagonalize_herm_mat(Ising_ham) / Ndiag        !Get the Eigenvalues.

    IF (ii /= 0) THEN         !From second iteration on          !Check convergence
    converged = abs(groundstate - eigenvalues(1)) .LT. eps
    END IF
    groundstate = eigenvalues(1)
    DEALLOCATE(eigenvalues)
```

Density matrix renormalization group:

1. Consider **4 blocks** of particles in 1-Dim, composed in order of N, 1, 1, and N particles. This system has a **central symmetry**. Construct the total Hamiltonian H_{tot} as follows:
   $$H_{tot} = H_1 \otimes \mathbb{I}_{N+2} + \mathbb{I}_N \otimes H_2 \otimes \mathbb{I}_{N+1} + \mathbb{I}_{N+1} \otimes H_3 \otimes \mathbb{I}_N + \mathbb{I}_{N+2} \otimes H_4 + \\ + H_{12} \otimes \sigma_x \otimes \mathbb{I}_{N+1} + \mathbb{I}_N \otimes H_{23} \otimes \mathbb{I}_N + \mathbb{I}_{N+1} \otimes \sigma_x \otimes H_{34}$$
   H_i $(i = 1, 2, 3, 4)$ are the Hamiltonian of the 4-blocks subsystems, dim $= D^N$, D, D, D^N. H_{ij} concern the boundary interactions between the blocks (i, j). H_{12} and H_{34} contain only the interaction part of blocks 1 and 4 respectively:
   $$H_{12} = \mathbb{I}_{N-1} \otimes \sigma_x, \quad H_{23} = \sigma_x \otimes \sigma_x \quad H_{34} = \sigma_x \otimes \mathbb{I}_{N-1}$$
2. **Diagonalize** H_{tot} to find the **ground state**. Define its **density matrix**. The density matrix eigenvalues (D^{2N} values) correspond to the ground state **population**. Tracing out the second half of the system, construct the **reduced density matrix** concerning the first 2 blocks. The system is symmetric so we can focus on the first half only.
3. After Diagonalizing the red. dens. mat., **sort** the population in **descending order**. The first N sorted eigenvectors will define the projector P we are using to truncate the system at each iteration.
4. **Increase the size** of H_1, H_4, H_{12}, and H_{34} of **1** particle. Due to the symmetry, consider only H_1 and H_{12}:
   $$H_{1p} = \mathbb{I}_1 \otimes H_1 \quad H_{12p} = \mathbb{I}_1 \otimes H_{12}$$
   Then, **project** them into the effective H'_1 and H'_{12}. For the system symmetry, we update H'_4 and H'_{34} in the same way.
   $$H'_1 = P^\dagger(H_{1p})P \quad H'_{12} = P^\dagger(H_{12p})P \quad H'_4 = H'_1 \quad H'_{34} = H'_{12}$$
5. **Repeat** previous steps substituting H_1, H_4, H_{12}, and H_{34}, with H'_1, H'_4, H'_{12}, and H'_{34}.

   The concept of the DMRG algorithm, is to upgrade the truncation strategy (w.r.t. the RSRG), checking each time which vector states compose the ground state, and truncating to those which have the highest population. Moreover, the system size increases slower (2 particles in each iteration), to prevent mistakes that could happen using the RSRG algorithm.

Code: RSRG ALGORITHM




```
  subroutine RG_algorithm(Ising_ham, AA, BB, local_dim, Ndiag)
complex*16, dimension(:,:), allocatable :: Ising_ham, Ising_ham1, AA, BB, AA1, BB1
complex*16, dimension(:,:), allocatable :: projector_n, Ising_2n, eigenvectors
integer*4                               :: local_dim, Ndiag, size, size2

size = local_dim ** Ndiag
size2 = local_dim ** (2 * Ndiag)
ALLOCATE(projector_n(size2, size))

!2N ISING HAMILTONIAN of order N:
Ising_2n = get_RG_ham_2n_order(Ising_ham, AA, BB, local_dim, Ndiag)

!PROJECTORS: Diagonalize it to get first "N" eigenvectors:
eigenvectors      = get_eigenvectors(Ising_2n)
projector_n(:, :) = eigenvectors(:, :size)

!Projecting ham_2N, AA and BB into N components to get new ham_n, AA', BB'
Ising_ham = project_C_mat(Ising_2n, projector_n) /2d0
AA        = project_C_mat(d_power_N_Id(local_dim, Ndiag).tens.AA, projector_n) /sqrt(2d0)
BB        = project_C_mat(BB.tens.d_power_N_Id(local_dim, Ndiag), projector_n) /sqrt(2d0)

!The code needs to deallocate each iteration because other subroutine allocate. This choice simplified debugging.
DEALLOCATE(AA, BB, Ising_ham, Ising_2n, projector_n, eigenvectors)
```

```
AA        = d_power_N_Id(local_dim, Ndiag -1).tens.pauliX
BB        = pauliX.tens.d_power_N_Id(local_dim, Ndiag -1)
Ising_ham =  get_Ising_Hamiltonian(Ndiag, lambda) !this matrix will be the first "ham_n" of the cycle.

function get_RG_ham_2n_order(ham_n, AA, BB, local_dim, NN) result (ham_2n)
    complex*16, dimension(:,:)              :: ham_n, AA, BB
    complex*16, allocatable, dimension(:,:) :: ham_2n, term1, term2, term3
    integer*4                               :: dim, local_dim, NN, dim2n

    dim = size(ham_n, 1)
    dim2n = dim ** 2
    ALLOCATE(ham_2n(dim2n, dim2n), term1(dim2n, dim2n), term2(dim2n, dim2n), term3(dim2n, dim2n))

    !From previous assignment, tensor product interface was uploaded in the code.
    term1 = ham_n.tens.d_power_N_Id(local_dim, NN)
    term2 = d_power_N_Id(local_dim, NN).tens.ham_n
    term3 = AA.tens.BB
    ham_2n = term1 + term2 + term3
```

```
function project_C_mat(Cmat, projector) result(proj_mat)
    dimC = shape(Cmat)          !dimensions of the complex matrix.
    dimP = shape(projector)     !dimensions of the projector.
    ... ! --> Checks on the dimensions are made.
    proj_mat =   matmul(matmul(transpose(conjg(projector)), Cmat), projector)
```

Code: DMRG ALGORITHM

```
subroutine infiniteDMRG_algorithm(H1, H2, H3, H4, H12, H23, H34, Htot, local_dim, Nblock, Nparticles)
      type(MB_wave) :: groundstate
      ...
  !Constructing Htot from H-interaction and H-non-interactiong (4 blocks structure)
      Htot = ... --> "Following theoretical-formula"
      eig_vec_4blocks = get_eigenvectors(Htot)    <-------- !Diagonalizes and returns Sorted Eigenvetors of Htot.
  !Reduced density matrix for 1st block +1 subsystem, starting from the Groundstate.
      groundstate%comp = eig_vec_4blocks(:, 1)    <-------- !1st Eigenvector is the groundstate of the system Htot.
      groundstate = .normalize.groundstate        <-------- !"groundstate is type(MBwave),
                                                            ! so it needs to be normalized to prop. use subroutines.
      dens_mat = get_density_matrix(groundstate)  <-------- !"Old" subroutine to get the density matrix
      red_dens_mat = get_red_density_matrix_first_kk_subsystems(dens_mat, local_dim, 2*(Nblock +1), Nblock +1)
  !Truncation
      eig_vec_rdm = get_eigenvectors(red_dens_mat)      ,     temp_eigvec = eig_vec_rdm
      red_eigenvalues = diagonalize_herm_mat(red_dens_mat)  <-------- !It returns red. dens. mat. eigenvalues.
      CALL sort_descending(red_eigenvalues, index_arr)      <-------- !Index array keeps track of the sorting order.
          DO ind = 1, block_dim * local_dim
              eig_vec_rdm (:, ind) = temp_eigvec(:, index_arr(ind))   <-- !Sorts in the desired way the eigenvectors.
          END DO
      projector = eig_vec_rdm(:, :block_dim)
!Consider only first "m" Eigenvectors to build the projector.
  !Projecting new H1', H2', ecc... adding 1 subsystem.
      H1proj  = (H1.tens.d_power_N_Id(local_dim, 1)) + (H12.tens.pauliX) + (d_power_N_Id(local_dim, Nblock).tens.H2)
      H12proj = (H12.tens.d_power_N_Id(local_dim, 1))
      H1 = project_C_mat(H1proj, projector)
      H12 = project_C_mat(H12proj, projector)
      H34 = H12
      H4 = H1
```

The updated matrices H1', H4', H12', H34' are defined.

WARNING: Here the order of the tensor product is inverted. However, the code returns acceptable results only in this way. (I really don't know why.)

2nd WARNING: The code returns physical results only for initial N > 1. Code's behaviour suggests that, for N = 1, the interaction part is not considered or it gets neglected while iterating the procedure.

The actual code contains more information and debugs statements. In the uploaded screenshot, the summary of all important parts of the subroutine DMRG is shown.

The subroutine exploits previous and new subroutines/functions to obtain the necessary results. Besides the "projecting" function, the new routines are presented in the following slide. Old ones have already been presented in previous assignments presentations.

Code: DMRG ALGORITHM 2

```
function get_red_density_matrix_first_kk_subsystems(density_matrix, DD, NN, kk) result(red_density_matrix)
    ...
    DO ii = 1, NN - kk
        dim_temp1 = (DD ** NN) / (DD ** (ii -1))
        dim_temp2 = (DD ** NN) / (DD ** ii)
        ALLOCATE(temp_red1(dim_temp1, dim_temp1))
            IF (ii == 1) THEN
                temp_red1 = density_matrix
            ELSE
                temp_red1 = temp_red2
                DEALLOCATE(temp_red2)
            END IF
        temp_red2 = get_red_density_matrix_tracing_out_Ith_system(temp_red1, DD, NN - (ii -1), NN - (ii -1))
        DEALLOCATE(temp_red1)
        IF (ii == NN - kk) THEN
            red_density_matrix = temp_red2
            DEALLOCATE(temp_red2)
        END IF
    END DO
```

This subroutine traces out all but the first "**kk**" subsystems from a density matrix. To do so, it exploits the previous assign. subroutine, which traced out the I-th subsystem from a dens. matrix.
With a **DO-loop** and proper **allocation,** it returns the desired reduced density matrix.

Using two temporary matrices, that get allocated and deallocated in different parts of the Do-loop, allows tracing out multiple subsystems from a density matrix.

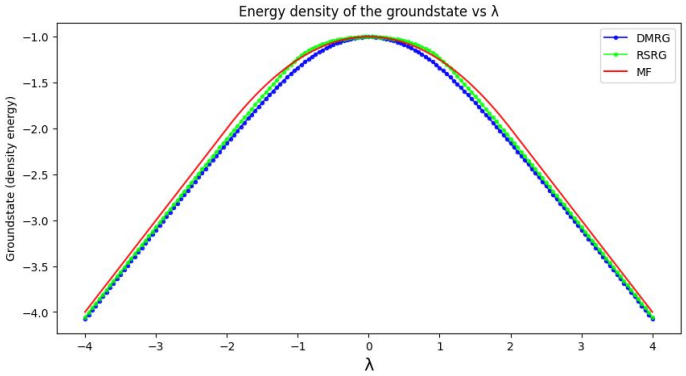
The algorithm requires sorting an array, depending on another sorting order. The following routine allows that:

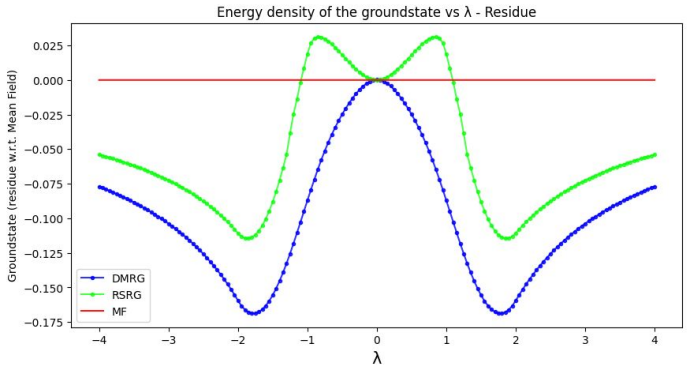
```
subroutine sort_descending(array, index_arr)
    real*8, dimension(:)                    :: array
    integer*4, allocatable, dimension(:)    :: index_arr
    dim = size(array)
    index_arr = (/(i, i=1,dim, 1)/)             !It creates an integer array : [1, 2, 3, ..., dim]

    DO ii=1, dim
        DO jj= 1, dim
            IF (array(ii) .GT. array(jj)) THEN  !It sorts in descending order an array.
                tt = array(ii)
                array(ii) = array(jj)
                array(jj) = tt

                ind = index_arr(ii)             !It sorts in the same way the "index_array",
                index_arr(ii) = index_arr(jj)   ! so that we save the order of sorting.
                index_arr(jj) = ind
```

This subroutine sorts an array in descending order, keeping track of the order in the "index_arr" array.

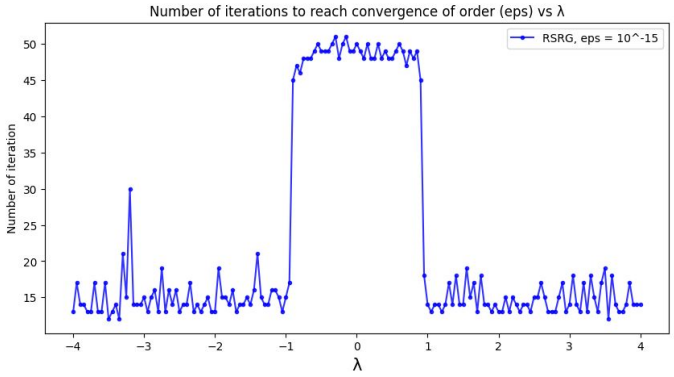
RESULTS

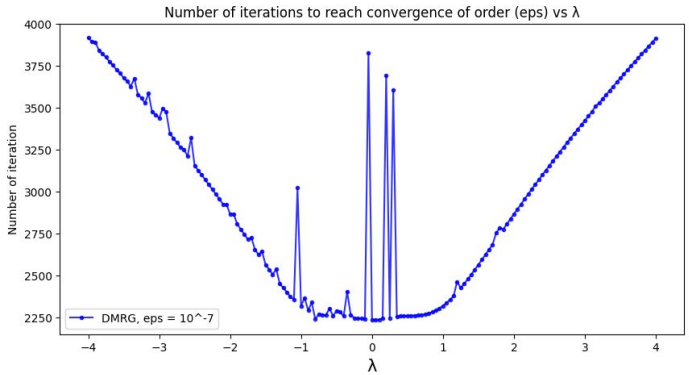
The plot of the energy density of the ground state as a function of the interaction strength **λ**, for the Mean Field approximation, the RSRG algorithm, and the DMRG algorithm, is presented. Both algorithm start with 2 particles as initial "block".





It is possible to observe similar shapes in the results of the three approaches. However, in the residue plot, different behaviors appear for -2 < λ < 2. Meanwhile, by increasing the absolute value of λ, the different approximations seem to converge.





The value of convergence "**eps**" has been chosen depending on the time requested by the algorithm. In the plots above, the number of iterations needed to reach converging eigenvalues (with fixed *eps*) is shown.
The two algorithms show opposite behaviors for $|\lambda| < 1$ and $|\lambda| > 1$. A guess is that, in the **RSRG**, the H_{tot} is "normalized" each iteration, while in the **DMRG** it's not. So, in the first case, the effective interaction part becomes smaller and smaller for an increasing number of particles, meanwhile, in the second it's kept constant. Depending on the case, this helps to reach the convergence for $|\lambda| > 1$ and $|\lambda| < 1$ respectively.