Renormalization Group

Alberto Chimenti

Exercise n.10
Information Theory and Computation - 2020

Abstract

The following report is intended to explain one possible solution to overcome memory problems in storing the Hamiltonian of a significant size Ising model. It has been tackled using the real-space RG algorithm and both the procedure and the results obtained are explained in the following.

1 Theory

We denote the Hamiltonian of an Ising model with N spins as:

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

Real-space RG algorithm

The aim of this algorithm is to be able to store information about a quantum system with a reduced size Hamiltonian.

1. Start with a small size quantum system with N particles.
2. Double the system size computing the new Hamiltonian as:

$$\mathcal{H}_{2N} = \mathcal{H}_N \otimes \mathbb{1}_N + \mathbb{1}_N \otimes \mathcal{H}_N + \mathcal{H}_{2N}^{int}$$

3. Diagonalize the doubled system matrix
4. Project the new Hamiltonian on a subsystem generated by the first 2^N eigenvectors.

$$\mathcal{H}_N^{tr} = P^\dagger \mathcal{H}_{2N} P$$

5. Iterate the process d times.

In such a way, one obtains at the end a truncated Hamiltonian of the same dimension of the starting one, describing N^{d+1} particles.

2 Code Development

The interesting part is to compute and update the interaction term $\mathcal{H}_{2N}^{int}$. Since we are solving a one-dimensional Ising model, such term reduces to:

$$\mathcal{H}_{2N}^{int} = \lambda \left(\mathbb{1}_2 \otimes \cdots \otimes \sigma_x^N \right) \otimes \left(\sigma_x^{N+1} \otimes \cdots \otimes \mathbb{1}_2 \right) = \lambda \mathcal{H}_{N,1}^{int} \otimes \mathcal{H}_{N,2}^{int}$$

Here we can project the subsystem interaction operators singularly and reuse the projected ones as the updated version for the next iteration. The computation of the updated operators can be expressed as:

$$\mathcal{H}_{N,1}^{int\,'} = P^{\dagger}\left(\mathbb{1}_N \otimes \mathcal{H}_{N,1}^{int}\right) P$$

$$\mathcal{H}_{N,2}^{int\,'} = P^{\dagger}\left(\mathcal{H}_{N,2}^{int} \otimes \mathbb{1}_N\right) P$$

A fortran subroutine was implemented for the computation of such reduced matrix.

```
    subroutine real_RG(H, N, subsystems, lambda, pauli_x, pauli_z)
        type(cmatrix), intent(inout) :: H
        type(cmatrix), intent(in) :: pauli_x, pauli_z
        double precision, intent(in) :: lambda
        integer, intent(in) :: N, subsystems
        type(cmatrix) :: subint1, subint2, tempdouble, idnn, temp
        type(cmatrix) :: P, tempP
        integer :: ii, dd

        ! REAL-RG ALGORITHM

        ! Pair interaction part between subsystems
        subint1 = field_interaction_op(pauli_z, N, N)
        subint2 = field_interaction_op(pauli_z, 1, N)

        ! Generate identity matrix (dim N)
        call allocate_cmatrix(idnn, 2**N, 2**N)
        idnn%elem=0.
        do ii=1, 2**N
            idnn%elem(ii,ii) = cmplx(1., 0.)
        end do

        call allocate_cmatrix(P, 2**(2*N), 2**N, .TRUE.)
        call allocate_cmatrix(tempP, 2**(2*N), 2**N)

        do dd=1, subsystems
            if (MOD(dd, 20)==0) then
                print*, "Simulated particles number N=", N, "**", dd
            end if
            call allocate_cmatrix(tempdouble, 2**(2*N), 2**(2*N), .TRUE.)


            ! Field interaction
            ! first part
            temp = tensor_product(H, idnn)
            tempdouble%elem = temp%elem
            call deallocate_cmatrix(temp)
            !second part
            temp = tensor_product(idnn, H)
            tempdouble%elem = tempdouble%elem + temp%elem
            call deallocate_cmatrix(temp)

            ! Pair interaction part
            temp = tensor_product(subint1, subint2)
            tempdouble%elem = tempdouble%elem + lambda*temp%elem
            call deallocate_cmatrix(temp)

            ! Compute eigenvectors
            call compute_eigenvals(tempdouble, tempdouble%eigenvals, tempdouble%eigenvects)

            ! Projector
            P%elem = tempdouble%eigenvects(:, 1:2**N)
            P%adj = .adj.P

            ! Project hamiltonian
            tempP%elem = matmul(tempdouble%elem, P%elem)
            H%elem = matmul(P%adj, tempP%elem)

```

```
            ! Project interaction subsystem terms
            temp = tensor_product(idnn, subint1)
            tempP%elem = matmul(temp%elem, P%elem)
            subint1%elem = matmul(P%adj, tempP%elem)
            call deallocate_cmatrix(temp)

            temp = tensor_product(subint2, idnn)
            tempP%elem = matmul(temp%elem, P%elem)
            subint1%elem = matmul(P%adj, tempP%elem)
            call deallocate_cmatrix(temp)

            call deallocate_cmatrix(tempdouble, .TRUE.)

        end do

        call deallocate_cmatrix(subint1)
        call deallocate_cmatrix(subint2)
        call deallocate_cmatrix(idnn)
        call deallocate_cmatrix(P)
        deallocate(P%adj)
        call deallocate_cmatrix(tempP)

    end subroutine
```

Main Program

```
PROGRAM main

    use ising

    implicit none

    type(cmatrix) :: pauli_x, pauli_z, H, tempH
    double precision :: lambda
    integer :: N, subsystems, ii, jj, istat, l_resolution

    ! Inputs!!!!!!!!
    print*, "Enter the number of qbits for each subsystem (INTEGER NUMBER): "
    read(*,*) N
    print*, "Enter the number of desired subsystems to add (INTEGER NUMBER):"
    read(*,*) subsystems

    lambda = 0
    l_resolution = 50

    call allocate_cmatrix(pauli_x, 2, 2)
    call allocate_cmatrix(pauli_z, 2, 2)

    ! Initialize Pauli matrices
    pauli_x%elem(1,1)=(0.,0.)
    pauli_x%elem(1,2)=(1.,0.)
    pauli_x%elem(2,1)=(1.,0.)
    pauli_x%elem(2,2)=(0.,0.)

    pauli_z%elem(1,1)=(1.,0.)
    pauli_z%elem(1,2)=(0.,0.)
    pauli_z%elem(2,1)=(0.,0.)
    pauli_z%elem(2,2)=(-1.,0.)

    open(unit=50, file="l_eigenvalues.txt", status="REPLACE")

    do jj=1, l_resolution+1

        print*, "Iteration with lambda=", lambda

```

```
40          ! Hamiltonian construction
41          call allocate_cmatrix(H, 2**N, 2**N, .TRUE.)
42          H%elem = 0.
43
44          do ii=1, N
45              tempH = field_interaction_op(pauli_z, ii, N)
46
47              H%elem = H%elem + tempH%elem
48              call deallocate_cmatrix(tempH)
49          end do
50
51          do ii=1, N-1
52              tempH = pair_interaction_op(pauli_x, ii, ii+1, N)
53
54              H%elem = H%elem + lambda*tempH%elem
55              call deallocate_cmatrix(tempH)
56          end do
57
58          ! REAL-RG ALGORITHM
59          call real_RG(H, N, subsystems, lambda, pauli_x, pauli_z)
60
61          call compute_eigenvals(H, H%eigenvals, H%eigenvects)
62
63          ! write to text file
64          write(50,*, iostat=istat) lambda, (H%eigenvals(ii), ii=1, 1)
65          if (istat /= 0) then
66              print*, "WARNING: ---> Error in wrinting, shutting down..."
67              stop
68          end if
69
70          ! update lambda
71          lambda = lambda + 5./(FLOAT(l_resolution))
72          ! deallocate the old hamiltonian
73          call deallocate_cmatrix(H, .TRUE.)
74
75      end do
76
77      call deallocate_cmatrix(pauli_x)
78      call deallocate_cmatrix(pauli_z)
79
80
81
82  END PROGRAM
```

3 Results

In the following we report the results obtained for the groundstate of the system. Several iterations of the same computation has been performed changing the values of the number of subsystems to add using real-space RG. To better understand the plots one should recall that the actual final number of simulated particles is N^{d+1}.

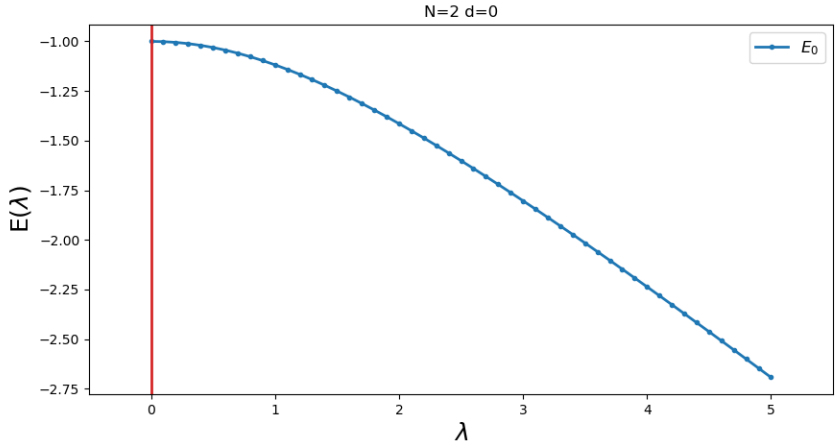

Figure 1: Ground state of the Hamiltonian, normalized over the number of simulated particles

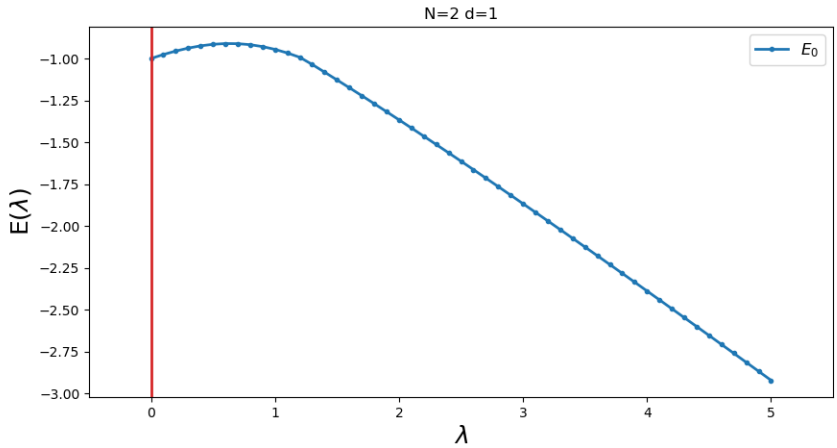

Figure 2: Ground state of the Hamiltonian, normalized over the number of simulated particles

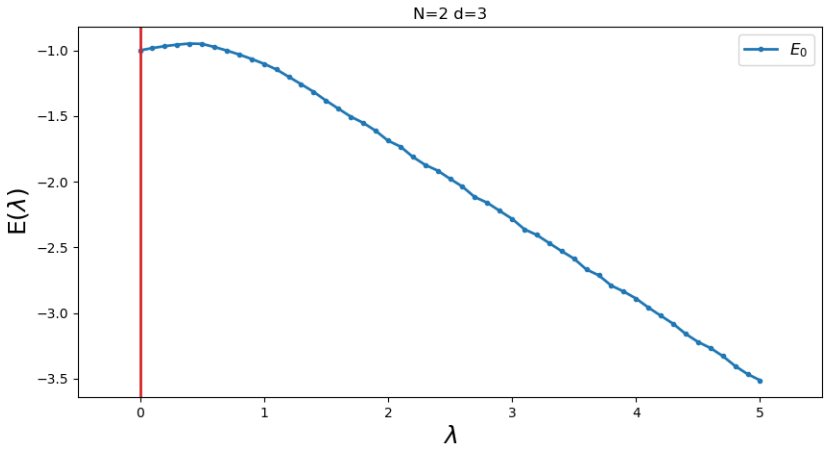

Figure 3: Ground state of the Hamiltonian, normalized over the number of simulated particles

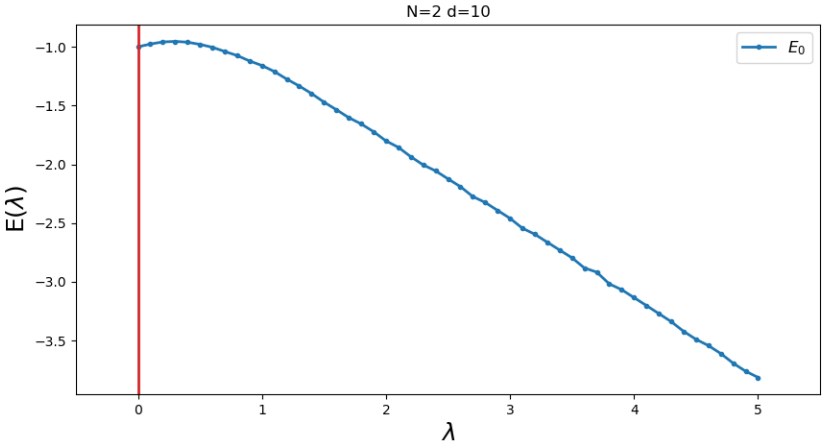

Figure 4: Ground state of the Hamiltonian, normalized over the number of simulated particles

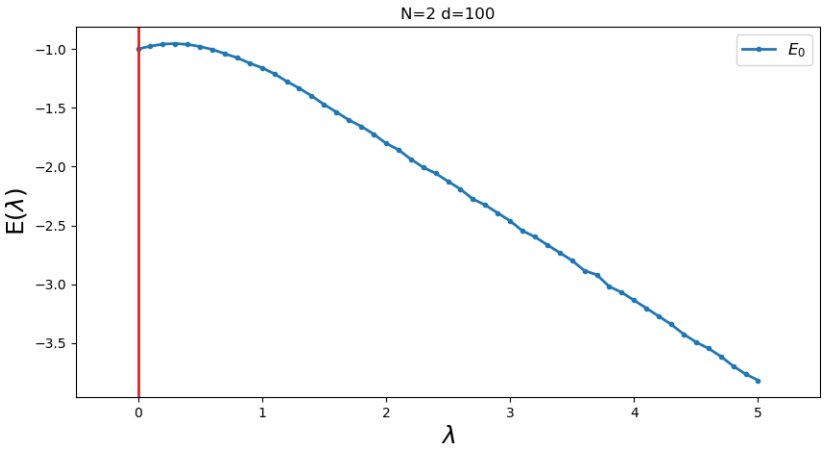

Figure 5: Ground state of the Hamiltonian, normalized over the number of simulated particles

As one can see, by looking at Figure (1) and (2) we can observe the approximation error of the algorithm. In fact, the first one describes a 2 particle system, while the second one describes an approximated 4 particle system. It is evident that there is an unphysical "kink" shortly after the $\lambda = 0$ value. This effect is reduced for higher number of RG iterations.

Finally one can observe by looking at Figure (4) and (5) that the behaviour of the ground state energy becomes independent from the number of particles contained in the system.

4 Self-evaluation

As shown, some of the computational limitations one can encounter while dealing with quantum systems could be overcome by means of some trade-offs. The real-space RG algorithm has been shown to be incorrect in some physical cases since it premises the assumption of being able to approximate the information of a bigger system ground state with only the low energy states of its two halves. However in this case it has shown to be useful to simulate systems over the thermodynamic limit and verify its theoretical behaviour.

It would be useful to formally quantify the error one is making, yet its "goodness" seems to be strictly dependent on the physical intuition of the user.