Density Matrices

Alberto Chimenti

Exercise n.9
Information Theory and Computation - 2019

Abstract

The following report is intended to explain a computational approach to study the eigenvalues behaviour of the Hamiltonian of N interacting particles with spin 1/2 in a one-dimensional lattice.

1 Theory

The Hamiltonian of this Ising model is:

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

We note that the operators written in the previous equation can be fully written as:

$$\mathcal{H} = \sigma_z^1 \otimes \mathbb{1}^2 \otimes \cdots \otimes \mathbb{1}^N + \cdots + \mathbb{1}^1 \otimes \cdots \otimes \mathbb{1}^{N-1} \otimes \sigma_z^N + \lambda \cdot \left(\sigma_x^1 \otimes \sigma_x^2 \otimes \cdots \otimes \mathbb{1}^N + \mathbb{1}^1 \otimes \cdots \otimes \sigma_x^{N-1} \otimes \sigma_x^N\right)$$

Indeed this fully expanded form respects the dimensionality of our Hamiltonian which is 2^N. As we can see the system qubits are subject to nearest neighbor interaction in a one-dimensional lattice.

We procede in computing such Hamiltonian diagonalize it to take a look at the behaviour of its eigenvalues.

2 Code Development

First thing one should do it to calculate the hamiltonian of the system. As stated before, the crucial point is that each term has to be calculated as a tensor product of N operators, yielding $2^N \times 2^N$ matrices.

2.1 Tensor product routine

Here we show the implementation of the generic tensor product between two operators.

```
function tensor_product(A, B) result(M)
    type(cmatrix) :: A, B, M
    integer :: ii, jj, kk, qq

    call allocate_cmatrix(M, A%dim(1)*B%dim(1), A%dim(1)*B%dim(1))
    do ii=1, B%dim(1)
        do jj=1, B%dim(2)
            do kk=1, A%dim(1)
                do qq=1, A%dim(2)
                    M%elem(kk+(ii-1)*A%dim(1), qq+(jj-1)*A%dim(2)) = B%elem(ii,jj)*A%elem(kk,qq)
                end do
            end do
        end do
    end do
```

```
end function
```

Then the generic tensor product was used to construct the two types of operators we need, the external field interaction and the pair interaction one.

2.2 External field operators

```
    function field_interaction_op(mat, index, N)result(self)
        type(cmatrix) :: mat, self, idn, temp
        integer :: index, ii, N

        ! Generate identity matrix
        call allocate_cmatrix(idn, mat%dim(1), mat%dim(2))
        idn%elem=0.
        do ii=1, mat%dim(1)
            idn%elem(ii,ii) = 1.
        end do

        ! Construct the actual operator
        call allocate_cmatrix(self, mat%dim(1), mat%dim(2))

        if (index==1) then
            self%elem = mat%elem
        else
            self%elem = idn%elem
        end if

        ! Perform tensor product
        do ii=2, N
            if (ii==index) then
                temp = tensor_product(self, mat)
            else
                temp = tensor_product(self, idn)
            end if

            call deallocate_cmatrix(self)
            call allocate_cmatrix(self, mat%dim(1)**ii, mat%dim(2)**ii)

            self%elem = temp%elem
            call deallocate_cmatrix(temp)
        end do
    end function
```

This function constructs the 2^N size external field operator by performing the tensor products among the identities and the particle Pauli matrix in position defined by the value of **index**.

2.3 Pair interaction

The following routine computes the pair interaction operator with a similar approach.

```
    function pair_interaction_op(mat, index1, index2, N)result(pair)
        type(cmatrix) :: mat, pair, idn, temp
        integer :: index1, index2, ii, N

        ! Generate identity matrix
        call allocate_cmatrix(idn, mat%dim(1), mat%dim(2))
        idn%elem=0.
        do ii=1, mat%dim(1)
            idn%elem(ii,ii) = 1.
        end do

        call allocate_cmatrix(pair, mat%dim(1), mat%dim(2))
```

```
13
14          if (index1==1 .or. index2==1) then
15              pair%elem = mat%elem
16          else
17              pair%elem = idn%elem
18          end if
19
20          ! Perform tensor product
21          do ii=2, N
22              if (ii==index1 .or. ii==index2) then
23                  temp = tensor_product(pair, mat)
24              else
25                  temp = tensor_product(pair, idn)
26              end if
27
28              call deallocate_cmatrix(pair)
29              call allocate_cmatrix(pair, mat%dim(1)**ii, mat%dim(2)**ii)
30
31              pair%elem = temp%elem
32              call deallocate_cmatrix(temp)
33          end do
34      end function
```

2.4 Main program

In the following we report the main program.

```
1  PROGRAM main
2
3      use ising
4
5      implicit none
6
7      type(cmatrix) :: pauli_x, pauli_z, H, temp
8      double precision :: lambda
9      integer :: N, ii, jj, istat, l_resolution
10
11     ! Inputs!!!!!!!!
12     print*, "Enter the number of qbits (INTEGER NUMBER): "
13     read(*,*) N
14     lambda = 0.
15     l_resolution = 20
16
17     call allocate_cmatrix(pauli_x, 2, 2)
18     call allocate_cmatrix(pauli_z, 2, 2)
19
20     ! Initialize Pauli matrices
21     pauli_x%elem(1,1)=(0.,0.)
22     pauli_x%elem(1,2)=(1.,0.)
23     pauli_x%elem(2,1)=(1.,0.)
24     pauli_x%elem(2,2)=(0.,0.)
25
26     pauli_z%elem(1,1)=(1.,0.)
27     pauli_z%elem(1,2)=(0.,0.)
28     pauli_z%elem(2,1)=(0.,0.)
29     pauli_z%elem(2,2)=(-1.,0.)
30
31     open(unit=50, file="l_eigenvalues.txt", status="REPLACE")
32
33     do jj=1, l_resolution+1
34
35         print*, "Iteration with N=", N, "lambda=", lambda
36
37         ! Hamiltonian construction
38         call allocate_cmatrix(H, 2**N, 2**N, .TRUE.)
```

```
        H%elem = 0.

        do ii=1, N
            temp = field_interaction_op(pauli_z, ii, N)

            H%elem = H%elem + temp%elem
            call deallocate_cmatrix(temp)
        end do

        do ii=1, N-1
            temp = pair_interaction_op(pauli_x, ii, ii+1, N)

            H%elem = H%elem + lambda*temp%elem
            call deallocate_cmatrix(temp)
        end do

        ! CHECK HERMITIANITY
        !H%adj = .adj.H
        !call check_hermitianity(H)

        call compute_eigenvals(H, H%eigenvals, H%eigenvects)

        ! write to text file
        write(50,*, iostat=istat) lambda, (H%eigenvals(ii), ii=1, 4)
        if (istat /= 0) then
            print*, "WARNING: ---> Error in wrinting, shutting down..."
            stop
        end if

        ! update lambda
        lambda = lambda + 3./(FLOAT(l_resolution))
        ! deallocate the old hamiltonian
        call deallocate_cmatrix(H, .TRUE.)

    end do

    call deallocate_cmatrix(pauli_x)
    call deallocate_cmatrix(pauli_z)

END PROGRAM
```

3 Results

The eigenvalues computation has been performed for several values of N taking 20 points for each of them spanning the interval $[0, 3]$ for the values of λ. The values of the first 4 energy levels is stored in a .txt file at each iteration and in the following we report their behaviour.

The maximum number of particles which can be stored in memory is $N = 15$ corresponding to a matrix of dimension 32768×32768.

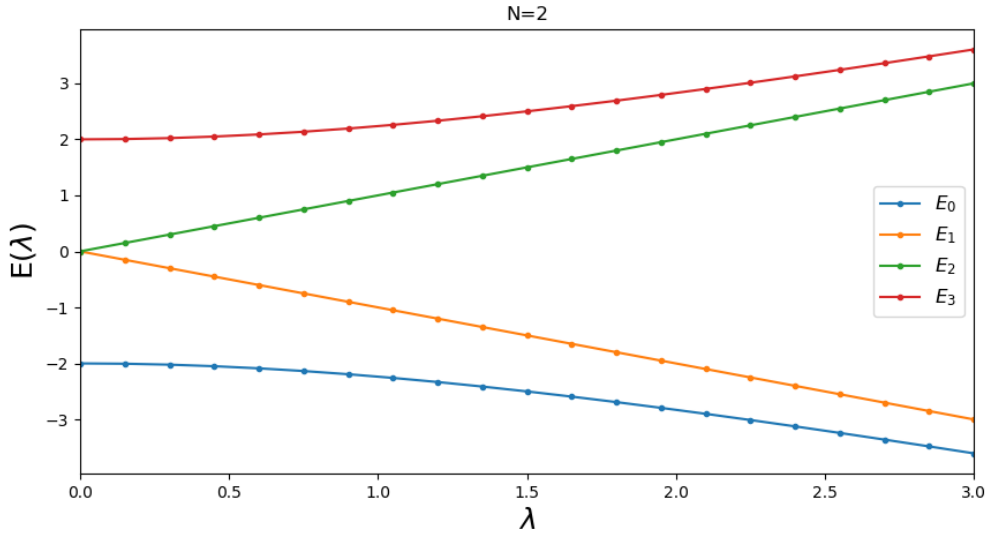

Figure 1: First 4 eigenvalues as a function of λ

Here we can see that for $\lambda = 0$ the system can assume three possible configuration. Indeed, the ground state has energy -2 corresponding to both spin down configuration while the highest one corresponding to both spins up has energy +2. The middle eigenvalues are degenerate which underlines the symmetry of the non-interacting system.

The insertion of the pair interaction splits the energy levels and removes the degeneracy.

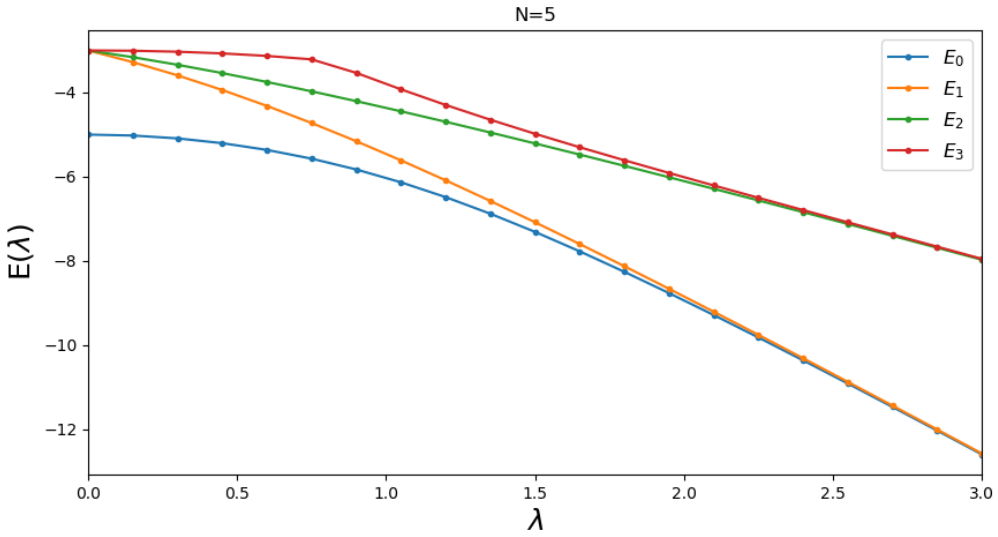

Figure 2: First 4 eigenvalues as a function of λ

Here we can see a different behaviour. The ground state has energy equal to -N as expected (for $\lambda = 0$), however the degeneracy here is restored and all energy levels assume a linear behaviour after a certain value of λ. A similar trend is shown for higher values of N.

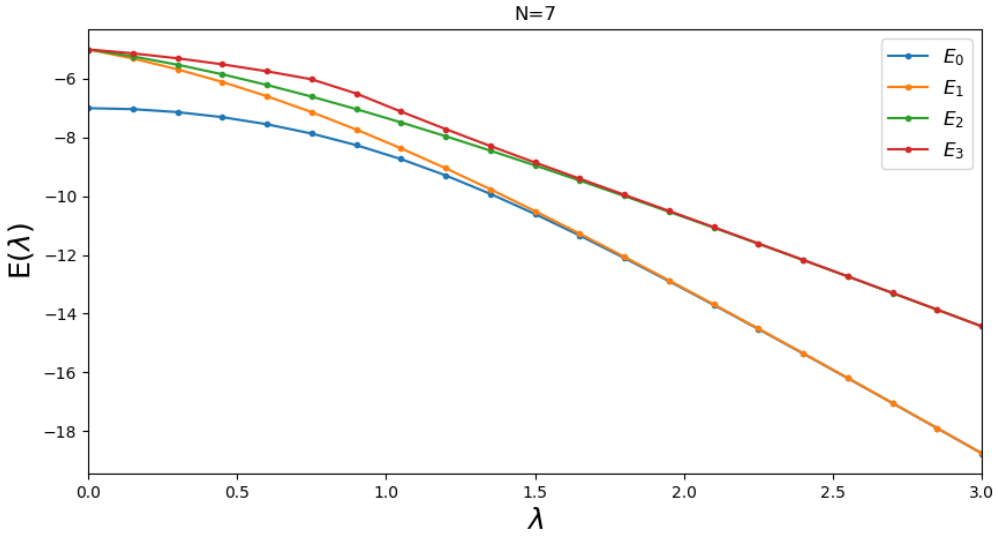

Figure 3: First 4 eigenvalues as a function of λ

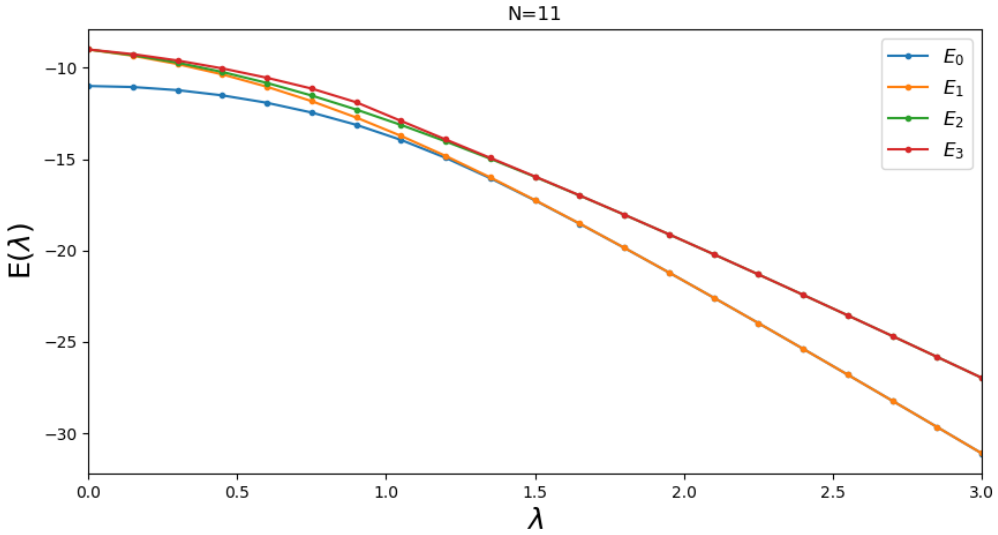

Figure 4: First 4 eigenvalues as a function of λ

We conclude that for the case in which $\lambda = 0$ the energy levels fit the possible configurations of the system. For growing values of λ the eigenvalues show a strange behaviour which might be due to a phase transition of the system when a certain threshold is reached, in fact the behaviour becomes linear. However, the change of the actual threshold depending on the value of N is probably due to the computational discretization since it should not depend on the number of particles considered.

4 Self-evaluation

The code implemented shows very big computational limitations of the method since the average number of particles which can be simulated is very low ($\simeq$ 12 depending on the available memory size). The shifting threshold of the lambda value needs some further theoretical analysis.