Ising model

Davide Maniscalco

December 16, 2019

Abstract

One dimensional quantum Ising model with nearest-neighbour interaction and external field is computationally studied.

SUGGESTED COMPILATION:

- Code: `gfortran Maniscalco-MATRIX.f90 Ex9-Maniscalco-CODE.f90 -llapack -o Ex9-Maniscalco.out`
- Python script for graphics: `python3 Ex9-Maniscalco-SCRIPT.py`

1 Theory

Let's consider the model of N interacting particles with spin 1/2 in a one dimensional lattice, with nearest-neighbour interaction and with an external magnetic field:

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

where λ is a coefficient and σ_x and σ_z are Pauli's matrices. It must be said that this notation is a shortcut: in reality both first and second term are sums of tensor products, where each term of the sum is the tensor product between the Pauli matrix (for the first term, matrices for the second) placed in one of the $1, \dots, N$ positions and identity matrices both on the left and on the right for each other position of the lattice:

$$\sigma_z \otimes \mathbb{1} \otimes \ldots \otimes \mathbb{1} + \ldots + \mathbb{1} \otimes \ldots \otimes \sigma_z \otimes \ldots \otimes \mathbb{1} + \ldots + \mathbb{1} \otimes \ldots \otimes \sigma_z \tag{2}$$

$$\sigma_x \otimes \sigma_x \otimes \mathbb{1} \otimes \ldots \otimes \mathbb{1} + \ldots + \mathbb{1} \otimes \ldots \otimes \sigma_x \otimes \sigma_x \otimes \ldots \otimes \mathbb{1} + \ldots + \mathbb{1} \otimes \ldots \otimes \sigma_x \otimes \sigma_x \tag{3}$$

It is known from the mean field solution that this system shows a quantum phase transition at $\lambda = \pm 2$: the eigenvalues have a quadratic dependence by λ in $(-2, 2)$ and a linear one out of $(-2, 2)$.

2 Code development

The old module `Maniscalco-MATRIX.f90` was recovered and updated: it were added a new type for real matrices, a trace function for real matrices, the printing subroutine was modified for real numbers (and not for complex as in the old version). The calculation of the determinant wasn't implemented instead.

The code `Ex9-Maniscalco-CODE.f90` contains all the useful subroutines and functions for solving this Ising model. The first function performs the generic tensor product between two real tensors of rank 2 of any dimension.

```
FUNCTION TENSOR_PRODUCT(AA, BB)
  INTEGER :: dima, dimb
  INTEGER :: ib, jb
  TYPE(RMATRIX):: AA, BB
  TYPE(RMATRIX) :: TENSOR_PRODUCT

  dima = AA%N(1)
  dimb = BB%N(1)

  TENSOR_PRODUCT%N(1) = dima*dimb
  TENSOR_PRODUCT%N(2) = dima*dimb

  ALLOCATE(TENSOR_PRODUCT%elem(TENSOR_PRODUCT%N(1),TENSOR_PRODUCT%N(2)))

  DO ib=1, dima     !dima = #of blocks; dimb = dimension
    DO jb=1, dima
      TENSOR_PRODUCT%elem((ib-1)*dimb+1:(ib+1)*dimb,(jb-1)*dimb+1:(jb+1)*
   dimb) = AA%elem(ib,jb)*BB%elem
    END DO
  END DO

  TENSOR_PRODUCT%tr = .TR.(TENSOR_PRODUCT)
  RETURN
  DEALLOCATE(TENSOR_PRODUCT%elem)
END FUNCTION
```

We have two indices, one for the rows and one for the columns, that run on the number of rows and columns of the first matrix: these select the elements of the first matrix, multiply them for the whole second matrix and place the result in the right place of the output one. This was tested for small dimensions thank to a proper debugging subroutine.

Now it comes the specific function for this exercise, that performs the sums of tensor products explained in Eqs. 2,3. In the code below it will be used the function `IDENTITY` that returns a real squared identity matrix given the dimension.

```
 FUNCTION HAM_ISING(NN, lambda)
    TYPE(RMATRIX) :: prev_id, next_id, pauli_zz, pauli_xx, ham_z, ham_x,
   temp, temp2, temp3,ham_ising
    INTEGER :: NN, pp
    REAL*4 :: lambda

    ALLOCATE(pauli_xx%elem(2,2))
    ALLOCATE(pauli_zz%elem(2,2))
    pauli_zz%N(1) = 2
    pauli_zz%N(2) = 2
    pauli_xx%N(1) = 2
    pauli_zz%N(2) = 2

    pauli_xx%elem(1,1) = 0
    pauli_xx%elem(1,2) = 1
    pauli_xx%elem(2,1) = 1
    pauli_xx%elem(2,2) = 0

    pauli_zz%elem(1,1) = 1
    pauli_zz%elem(1,2) = 0
    pauli_zz%elem(2,1) = 0
    pauli_zz%elem(2,2) = -1

    ALLOCATE(ham_z%elem(2**NN,2**NN))
    ALLOCATE(ham_x%elem(2**NN,2**NN))
    ALLOCATE(ham_ising%elem(2**NN,2**NN))
    ham_x%N(1) = 2**NN
    ham_x%N(2) = 2**NN
    ham_z%N(1) = 2**NN
    ham_z%N(2) = 2**NN
    ham_ising%N(1) = 2**NN
    ham_ising%N(2) = 2**NN

    ham_z%elem = 0.0
    ham_x%elem = 0.0
    ham_ising%elem = 0.0

    DO pp=1, NN
      prev_id = IDENTITY(2**(pp-1))
      next_id = IDENTITY(2**(NN-pp))

      temp = TENSOR_PRODUCT(prev_id, pauli_zz)
      temp2 = TENSOR_PRODUCT(temp, next_id)

      ham_z%elem = ham_z%elem + temp2%elem

      DEALLOCATE(prev_id%elem, next_id%elem, temp%elem, temp2%elem)
    END DO

    DO pp=1, NN-1
      prev_id = IDENTITY(2**(pp-1))
      next_id = IDENTITY(2**(NN-pp-1))

      temp = TENSOR_PRODUCT(prev_id,pauli_xx)
      temp2 = TENSOR_PRODUCT(temp,pauli_xx)
      temp3 = TENSOR_PRODUCT(temp2,next_id)

      ham_x%elem = ham_x%elem + temp3%elem

      DEALLOCATE(prev_id%elem, next_id%elem, temp%elem, temp2%elem, temp3%
   elem)
    END DO

    ham_z%elem = lambda*ham_z%elem
    ham_ising%elem = ham_x%elem + ham_z%elem
    ham_ising%tr = .TR.(ham_ising)

    RETURN
    DEALLOCATE(ham_z%elem, ham_x%elem, pauli_xx%elem, pauli_zz%elem)
  END FUNCTION
```

The function uses the fact that the tensor product between a squared identity matrix of dimension n and a squared identity matrix of dimension m, is a squared identity matrix of dimension $m{\cdot}n$. The Pauli matrix σ_z in the position p of the lattice has therefore an identity matrix of dimension 2^{p-1} on its left and an identity matrix of dimension 2^{N-p} on its right, and therefore there are only two tensor products to be calculated. A do-loop performs this for each position of the lattice. This same procedure is used for the computation of Eq.3. In the end the two matrices obtained are summed, to be afterwards diagonalized by the DSYEV subroutine, in order to obtain the eigenvalues. The purpose of the program is of studying the eigenvalues as a function of λ: therefore the lattice dimension N and the number of eigenvalues to be printed k must be given as input, while a list of values of λ is read from the file lambdas.dat. The code writes in the output file for_graphics_eigens.dat one column containing the values of λ and k columns containing the eigenvalues.

The python script can automatize the whole process, creating the file containing the input values of λ, running the fortran program, performing the graphics using the content of the `for_graphics_eigens.dat` file.

3 Results

The program can handle a value of N not bigger than 12, otherwise it crashes. In the previous exercise bigger matrices could be stored, here the problem is probably due to the diagonalization.

It was decided to keep $\lambda \in [0:3]$ and to print the first 4 eigenvalues for each dimension of the lattice. 50 values of λ in the range $[0:3]$ were selected for each lattice dimension. In Fig.1 are shown the results for $N = 2$. As it can be seen, at $\lambda = 0$ occur degeneracy occurs:

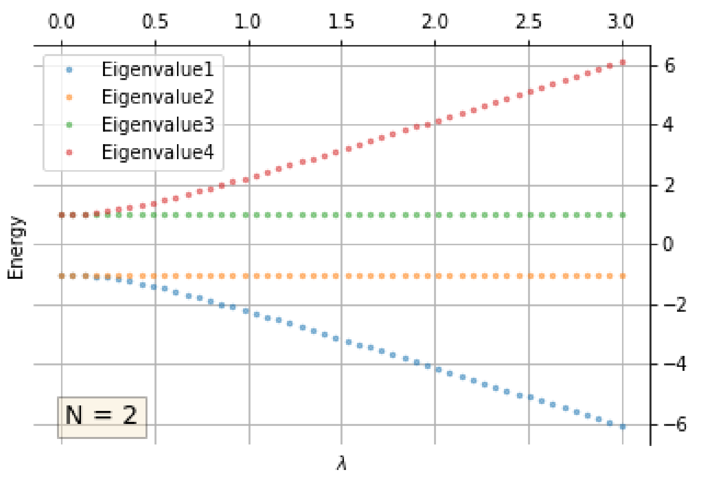

Figure 1: Energy spectrum for $N = 2$

this is reasonable, by the moment that state with both spins aligned and the ones with one spin up and one spin down are indistinguishable. When the external field is turned on, the degeneracy disappears.

In Fig.2 results for the bigger $N = 8$ are shown. The degeneracies at $\lambda = 0$ are still present;

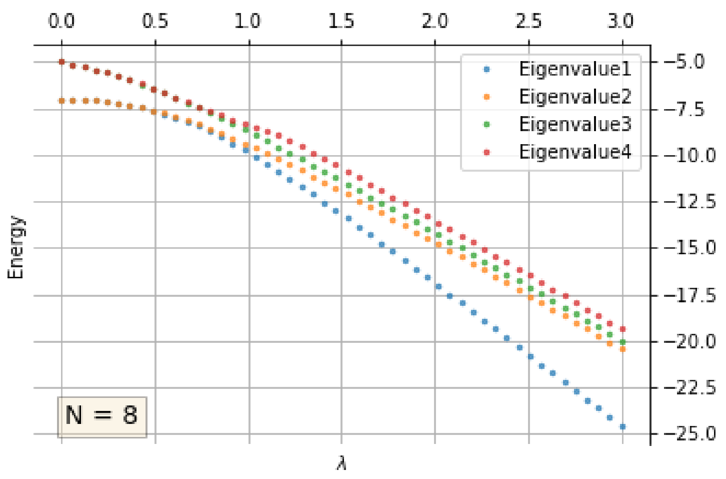

Figure 2: Energy spectrum of the first four eigenvalues for $N = 8$

at $\lambda = 0$ we have $E_{gs} \simeq -7$, that is reasonable: with 8 spins we have 7 interactions, that are all negative in the lower energy state. Finally we want to check that the quantum phase transition at $\lambda = 2$ predicted by the mean field theory occurs. For this purpose, the ground state was fitted with a line in the interval $[2 : 3]$ and with a quadratic polynomial in $[0 : 2]$. Results are shown in Fig.3. and in Tab.1.

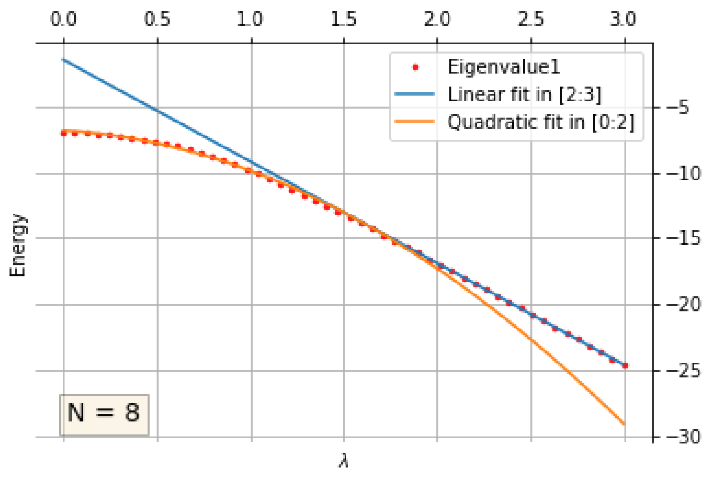

Figure 3: Fits for the ground state of $N = 8$.

	Linear	Quadratic
$\sum$Res	0.0016	0.62

Table 1: Sum of residuals

The linear fit is excellent, the quadratic one is reasonable. Surely the quantum phase transition happens, but more analysis should be done to verify if it exactly occurs at $\lambda = 2$

4 Self evaluation

All the tasks have been achieved. The program works, debugging was made, the python script can automatize the whole process till the graphics. More analysis could have been done about the quantum phase transition, and in studying the relation between the spectrum and the mean field solution.