Quantum Information and Computing 2020/21
Week 9 report

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

In this work we deal with the monodimensional Ising model Hamiltonian. The first problem faced is to write the code needed for the initialization of the Hamiltonian for N spins, so for its $2^N \times 2^N$ matrix representation. Then, we exploit the Lapack library for the diagonalization and to find the eigenvalues. Lastly, the first energy levels are computed for several values of N and plotted with respect to the interaction strength in order to study the system.

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:

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

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 find the energy levels of the system, we should diagonalize the Ising Hamiltonian $\hat{H}$. Then, by studying the eigenvalues, we expect for $\lambda = 0$ that the ground state is 2-fold degenerate and the first excited state is $2(N-1)$-fold degenerate. The physical reason is due to the fact that with no field the only term remaining is the interaction one, so spins align along the x-axis. If we increase the value of λ, we expect that the degeneracy disappears after a certain value of $\lambda = \lambda_c$, over which a phase transition happens. However, in practice we can only deal with a small number of spins N, so the degeneracy for $0 < \lambda < \lambda_c$ will be only "approximate" and it becomes more precise by increasing the number of spins N.

2 Code Development

For this work, the new module **`ising_utils`** is implemented, containing several functions and subroutines for multiple purposes:

- **`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 $2^{\texttt{N}} \times 2^{\texttt{N}}$;
- **`ising_hmat_init(N,L)`**, for initializing the Ising Hamiltonian for `N` spins and an interaction strength parameter `L`;
- **`ising_hmat_diag(hmat,eigs)`**, for diagonalizing the Ising Hamiltonian and returning both the eigenfunctions (in the `hmat` input variable) and the eigenvalues (in `eigs`);
- **`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.

2.1 Tensor product implementation

The code to perform the tensor product operation between two input matrices is showed in Listing 1. This part is crucial for the Ising Hamiltonian initialization, since this task relies on the tensor product between 2×2 identity and Pauli matrices.

```
function ising_tensor_prod(mat1, mat2) result(res)
    complex(8), dimension(:,:) :: mat1                ! input  arguments
    complex(8), dimension(:,:) :: mat2                ! input  arguments
    complex(8), dimension(:,:), allocatable :: res    ! output result

    integer(4) :: N1, N2, M1, M2
    integer(4) :: ii, jj, lli, llj, uui, uuj

    N1 = size(mat1,1)
    N2 = size(mat2,1)
    M1 = size(mat1,2)
    M2 = size(mat2,2)

    allocate(res(N1*N2,M1*M2))

    ! loop to execute tensor product between mat1 and mat2
    do ii=1,N1
        do jj=1,M1
            lli = (ii-1)*N2 + 1
            llj = (jj-1)*M2 + 1
            uui =  ii   *N2
            uuj =  jj   *M2
            res(lli:uui,llj:uuj) = mat1(ii,jj)*mat2
        end do
    end do
end function ising_tensor_prod
```

Listing 1. Implementation of the tensor product operation between two matrices.

2.2 Ising Hamiltonian initialization

The code to perform the initialization of the Ising Hamiltonian is showed in Listing 2. Given the number of spins in the linear chain and the interaction strength λ, this function returns the $2^N \times 2^N$ Hamiltonian matrix needed for the following studies. Note how the crucial operation, which constitutes the bottleneck, is the tensor product part.

```
function ising_hmat_init(N, L) result(hmat)
    integer(4) :: N                             ! input  arguments
    real(8)    :: L                             ! input  arguments
    complex(8), dimension(2**N,2**N) :: hmat    ! output result

    complex(8), dimension(2,2) :: s_x, s_z      ! Pauli matrices sx, sz
    integer(4) :: ii

    s_x      = COMPLEX( 0.0d0, 0.0d0)
    s_x(1,2) = COMPLEX( 1.0d0, 0.0d0)
    s_x(2,1) = COMPLEX( 1.0d0, 0.0d0)
    s_z      = COMPLEX( 0.0d0, 0.0d0)
    s_z(1,1) = COMPLEX( 1.0d0, 0.0d0)
    s_z(2,2) = COMPLEX(-1.0d0, 0.0d0)

    ! initialize Hamiltonian to zero
    hmat = COMPLEX(0.0d0, 0.0d0)
    ! fill with the first piece of Ising Hamiltonian
    do ii=1,N
        hmat = hmat + ising_tensor_prod(                &
            ising_tensor_prod(ising_identity(ii-1), s_z), &
            ising_identity(N-ii)                        &
        )
    end do
    ! multiply by lambda the first piece
```

```
    hmat = COMPLEX(L,0.0d0) * hmat
    ! fill with the second piece of Ising Hamiltonian
    do ii=1,N-1
        hmat = hmat + ising_tensor_prod(   &
            ising_tensor_prod(                                  &
                ising_tensor_prod(ising_identity(ii-1), s_x), &
                s_x                                             &
            ),                                                  &
            ising_identity(N-ii-1)                              &
        )
    end do
end function
```

Listing 2. Implementation of the Ising Hamiltonian initialization.

2.3 Ising Hamiltonian diagonalization

For the calculation of the eigenvalues and eigenvectors of the discretized Hamiltonian, the Lapack function **zheev** is employed since the matrix we are working with is hermitian. Therefore, the subroutine **ising_hmat_diag** is built as a wrapper of the Lapack function, as it is showed in Listing 3. Note that the input arguments are changed after the execution of this subroutine, storing then the j^{th} eigenfunction in the j^{th} column of `hmat` and all the eigenvalues in `eigs`.

```
subroutine ising_hmat_diag(hmat, eigs)
    complex(8), dimension(:,:) :: hmat    ! input arguments
    real(8),    dimension(:)   :: eigs    ! input arguments

    complex(8), dimension(:), allocatable :: work(:)
    complex(8), dimension(:), allocatable :: rwork(:)
    integer(4) :: lwork, info
    integer(4) :: N

    N     = size(hmat, 1)
    lwork = max(1,2*N-1)

    allocate(work(lwork))
    allocate(rwork(max(1, 2*N)))
    call zheev('V', 'U', N, hmat, N, eigs, work, lwork, rwork, info)
    deallocate(work)
    deallocate(rwork)
end subroutine ising_hmat_diag
```

Listing 3. Implementation of the wrapper of `zheev` for Ising Hamiltonian diagonalization.

2.4 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 4. A complete list of the options is given in Table 1.

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

- initializes the Ising Hamiltonian $\hat{H}$ for N spins and for an interaction strength λ;
- diagonalizes $\hat{H}$ and computes the eigenvectors and, in particular, the eigenvalues;
- prints the first k eigenvalues of $\hat{H}$.

Arg option	Arg name	Arg meaning	Default
`-N`	`nspins`	N	2
`-L`	`lambda`	λ	1
`-K`	`levels`	First k levels to print	1
`-h`	`help`	print help	

Table 1. Command-line arguments of the program.

```
$ ./09.o -N 10 -K 4

Running with the following cmdline options:

N =        10    default [T/F]: F
L =       1.0    default [T/F]: T
K =         4    default [T/F]: F

-12.381489999654752
-12.082569625309047
-11.491406263829482
-11.192485889483818
```

Listing 4. Example of launch and user interface.

Last but not least, the program must be compiled with the flag `-ffree-line-length-0`.

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

3 Results

The main program is run requiring to print the first $k = 4$ eigenvalues of the Ising Hamiltonian. In particular, the cases analyzed are the ones for $N = 4, 6, 8, 10$ spins and for $\lambda \in [0, 3]$. The results for the ladders, plotted through a Gnuplot script, are showed respectively in Figures 1a, 1b, 1c and 1d. We can observe the following behaviour:

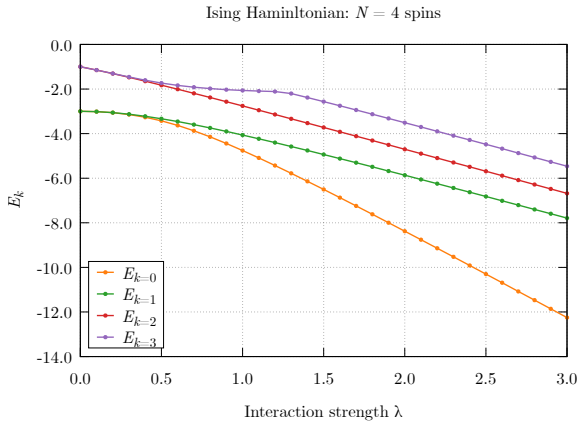
- $\lambda = 0$
  This is the non-interacting case. There is a 2-fold degeneracy of the first energy level.
- $\lambda > 0$
  This is the interacting case. The previously described 2-fold degeneracy of the non-interacting case is now broken, being however "approximately" true until a certain value of λ that depends on the number of spins.

So, we can observe how the value of λ_c where a phase transition occurs becomes higher when increasing N. We expect it to converge for large values of N.

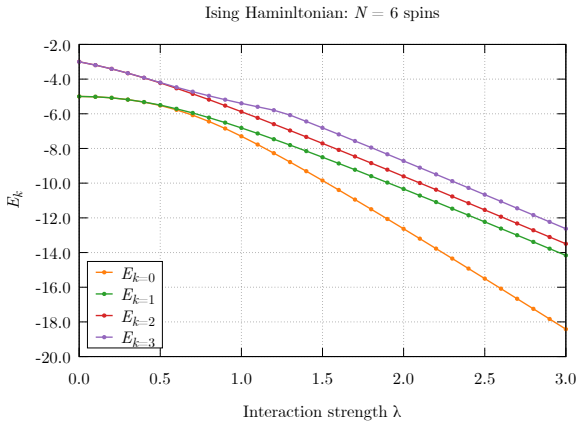
Another study has been performed by plotting the ground state eigenvalue depending on the field strength λ and on the number of spins N, after normalizing it for $N - 1$. This choice is due to the fact that the spins at the boundaries have only one neighbour and so, when considering their interaction, they weigh $\frac{1}{2}$ each and not 1. The result can be seen in Figure 2.

4 Self-evaluation

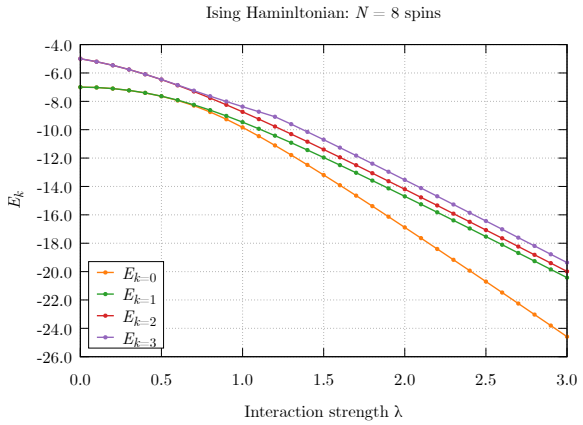
In this work we have successfully tackled the problem of Ising model simulation with a certain number of spins N and with an input interaction strength λ. The simulations showed in the results have been run for a maximum $N = 10$. However the machine on which the code is run is capable of reaching an $N_{\max} = 14$. In order to perform systematic studies for variable λ and for $N > 10$ a different and more efficient implementation should be written, in particular for the Hamiltonian initialization. The code written performs all the tensor products needed for this operation. This is the most complete and generic way to solve the task, however, it significantly slows the execution for large values of N. An hard-coded and efficient solution could be employed, but the most complete one has been chosen for a better understanding.



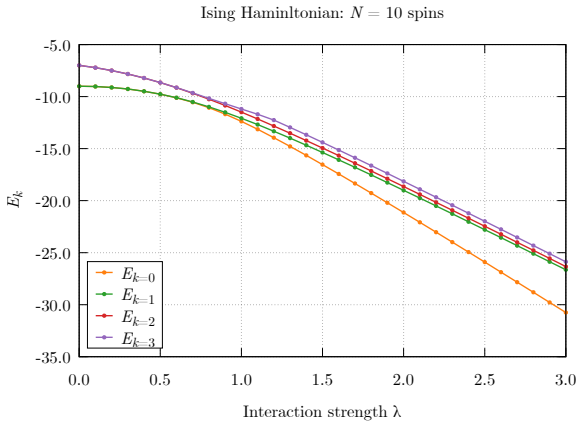
(a) $N = 4$



(b) $N = 6$



(c) $N = 8$



(d) $N = 10$

Figure 1. First $k = 4$ eigenvalues of Ising Hamiltonian depending on the interaction strength λ, for several number of spins N.

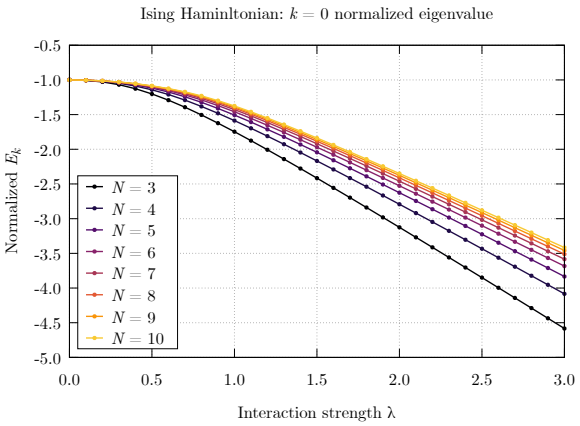


Figure 2. First $k = 0$ eigenvalue of Ising Hamiltonian depending on the interaction strength λ and for several number of spins N, normalized by $N - 1$.