Quantum Information and Computing

prof: *Simone Montangero*

Report of exercise 9

Transverse field Ising Model

Alberto Bassi

December 14, 2020

Abstract

The aim of this exercise is to solve the quantum transverse field Ising model.

Theory

Let us consider the N-bodies hamiltonian acting on $\mathcal{H}_N = \otimes_{k=1}^{N}\mathcal{H}$, where $\mathcal{H} \equiv \mathbb{C}^2$ is the one-body spin 1/2 Hilbert space,

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

where σ_x and σ_z are the Pauli matrices.
Generally, given an operator acting on a single body Hilbert space, its action on Hilbert spaces' tensor product can be obtatained by tensor multiplication with identities acting on the other Hilbert spaces. Namely, we have

$$\begin{aligned} \sigma_z^i &= \mathbb{1}_{i-1} \otimes \sigma_z \otimes \mathbb{1}_{N-i} \ , \\ \sigma_x^i \sigma_x^{i+1} &= \mathbb{1}_{i-1} \otimes \sigma_x \otimes \sigma_x \otimes \mathbb{1}_{N-i-1} \ , \end{aligned} \tag{2}$$

where $\mathbb{1}_k$ is the identity acting on $\mathcal{H}_k$.
The aim of this exercise is to solve exactly the model by implementing and diagonalizing the hamiltonian of Equation 1. We will repeat this procedure by varying the field strength λ in the interval $[0, 3]$.

Code Development

In the module named **manybodyQS** we have written more subroutines to deal with the problem given so far. We shall need to initialize the require matrices at first. For instance, by calling **init_sigmax** we will initialize σ_x acting on $\mathcal{H}$.

```fortran
subroutine init_sigmax(sigmax)
        implicit none
        double complex, dimension(:,:),allocatable :: sigmax
        allocate(sigmax(2,2))
        sigmax=cmplx(0d0,0d0)
        sigmax(1,2)=cmplx(1d0,0d0)
        sigmax(2,1)=cmplx(1d0,0d0)
      end subroutine init_sigmax
```

Then, the subroutine **init_identity** initializes a generic identity acting on a Hilbert space of dimension 2^k, which is indeed the tensor product of k identity operator acting on $\mathcal{H}$.

```fortran
subroutine init_identity(identity,DD,kk)
        implicit none
        integer DD,kk,ii
        double complex, dimension(:,:), allocatable::identity
        allocate(identity(DD**kk,DD**kk))
        identity=cmplx(0d0,0d0)
```

```
          do ii=1,DD**kk
             identity(ii,ii)=cmplx(1d0,0d0)
          enddo
        end subroutine init_identity
```

But these are specific matrices needed for our particular problem. In general, we would like to initialize generic Hamiltonian to deal with generic quantum problem. At first, we would like to compute the tensor product of two generic density matrices. This is achieved with the subroutine **matrix _tens _product**

```
 subroutine matrix_tens_product(mat1,dim1,mat2,dim2,out,dim,count)
          implicit none
          integer :: dim1,dim2,dim,count,jj,ii,ll,kk
          double complex, dimension(dim1,dim1) :: mat1
          double complex, dimension(dim2,dim2) :: mat2
          double complex, dimension(dim,dim):: out
          if(dim1*dim2.ne.dim) then
             print *, "Subsystems dimensions product different: ERROR"
             count=count+1
          endif
          do jj=0,dim1-1
             do  ii=0,dim1-1
                do kk=0,dim2-1
                   do ll=0,dim2-1
                      out(ii*dim2+ll+1,jj*dim2+kk+1)=mat1(ii+1,jj+1)*mat2(ll+1,kk+1)
                   enddo
                enddo
             enddo
          enddo
        end subroutine matrix_tens_product
```

Bearing this in mind,by means of the subroutine **init_ising _hamiltonian**, we initialize the hamiltonian given by Equation 1. At first, we initialize the one-particle matrices needed, the Hamiltonian completely filled by zeros and some temporary matrices which we will make use of during the initialization loop ahead.

```
subroutine init_ising_hamiltonian(hamiltonian,DD,NN,lambda,status,count)
          implicit none
          integer :: ii,jj,kk,status,DD,NN,count
          double precision ::lambda
          double complex,dimension (:,:),allocatable::hamiltonian,temp,temp_up,sigmax,sigmaz,
    identity,prod
          if(NN<2) then
             print *, "Invalid number of particles. Set NN=2"
             NN=2
          endif
          call init_sigmax(sigmax)
          call init_sigmaz(sigmaz)
          allocate(hamiltonian(DD**NN,DD**NN),stat=status) !allocate hamiltonian
          allocate(temp(DD**NN,DD**NN)) !allocate temp for swapping
          allocate(prod(DD**2,DD**2)) !compute sigmax tensprod(tp) sigmax
          call matrix_tens_product(sigmax,DD,sigmax,DD,prod,DD**2,count)
          hamiltonian=cmplx(0d0,0d0) !initialize to 0 hamiltonian
```

Then, we start by filling our Hamiltonian with the quadratic part, by looping between 0 and $N-2$ and computing each time the tensor product $\mathbb{1}_k \otimes \sigma_x \otimes \sigma_x \otimes \mathbb{1}_{N-k-2}$, for $k = 0, 1, \ldots, N-2$. The correctness at this stage is verified by printing (here left as comments) the temporary Hamiltonians for $N = 2, 3$.

```
!Quadratic part of hamiltonian
          do kk=0,NN-2
             !compute tensor product  of first k identities with sigmx tp sigmx
             allocate(temp_up(DD**(kk+2),DD**(kk+2)))
             call init_identity(identity,DD,kk)
             call matrix_tens_product(identity,DD**kk,prod,DD**2,temp_up,DD**(kk+2),count)
             deallocate(identity)
             !Compute the second tensor product
             call init_identity(identity,DD,NN-kk-2)
             call matrix_tens_product(temp_up,DD**(kk+2),identity,DD**(NN-kk-2),temp,DD**NN,
    count)
             deallocate(identity,temp_up)
             !Sum to get the hamiltonian
             hamiltonian=hamiltonian+temp
             !print *,"Temp_quad"
             !do ii=1,DD**NN
             !print *, (temp(ii,jj),jj=1,DD**NN)
             !enddo
          enddo
```

Then, in a similar manner we add to it the linear part by computing the tensor product $\mathbb{1}_k \otimes \sigma_z \otimes \mathbb{1}_{N-k-1}$, for $k = 0, 1, \dots, N-1$. The correctness is checked by looking at the temporary Hamiltonians for different N. Since now we expect them to diagonal, it is sufficient to print only the diagonal. Indeed, due to the properties of σ_z, which is diagonal, the entry corresponding to the binary index $(i_1, i_2, \dots, i_n)$, where $i_j = 0, 1, \forall j = 1, 2, \dots, N$, is $\sum_{j=1}^{N} (-1)^{i_j}$. By exploiting this property and also the properties of σ_x, we could have initialized the Hamiltonian in a more straightforward way. However, the initialization time is negligible with respect to diagonalization time, as we shall see, therefore we would have gained too little.

```
!Linear part of hamiltonian
         do kk=0,NN-1
            !compute tensor product of firts part. Now sigmaz
            allocate(temp_up(DD**(kk+1),DD**(kk+1)))
            call init_identity(identity,DD,kk)
            call matrix_tens_product(identity,DD**kk,sigmaz,DD,temp_up,DD**(kk+1),count)
            deallocate(identity)
            !Compute  second part
            call init_identity(identity,DD,NN-kk-1)
            call matrix_tens_product(temp_up,DD**(kk+1),identity,DD**(NN-kk-1),temp,DD**NN,
    count)
            deallocate(identity,temp_up)
            !Sum to get hailtonian
            hamiltonian=hamiltonian+temp*lambda
            !print *,"Temp_lin_diag"
            !do ii=1,DD**NN
            !print *, temp(ii,ii)
            !enddo
         enddo
          deallocate(temp,sigmax,sigmaz,prod)
       end subroutine init_ising_hamiltonian
```

In the end, in "main.f95" the program **ising** initialize the hamiltonian for a given N and it diagonalizes it for a number N_{lam} of λ's equispaced in $[0, 3]$.

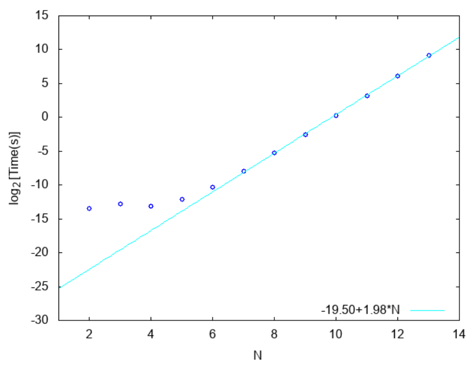
```
do zz=1,N_lam
        call cpu_time(start)
        lambda=3/dble(N_lam)*(zz-1)
        !Initialize hamiltonian
        call init_ising_hamiltonian(hamiltonian,DD,NN,lambda,status,count)
        call check_allocation(status,debug,printer)
        call cpu_time(finish1)
        !Diagonalize hamiltonian
        lda=max(1,DD**NN)
        lwork=max(1,2*DD**NN-1)
        allocate(w(DD**NN))
        allocate(work(lwork))
        allocate(rwork(max(1,3*DD**NN-2)))
        call zheev('N','U',DD**NN,hamiltonian,lda,w,work,lwork,rwork,info)
        do hh=1,kk
           levels(zz,hh)=w(hh)
        enddo
        deallocate(hamiltonian,w,work,rwork)
        call cpu_time(finish2)
        if (zz==1) then
           open(13,file="data/time.txt",status='unknown',access='append')
           write(13,*) NN, finish2-start,finish2-finish1
           close(13)
        endif
     enddo
```

Results

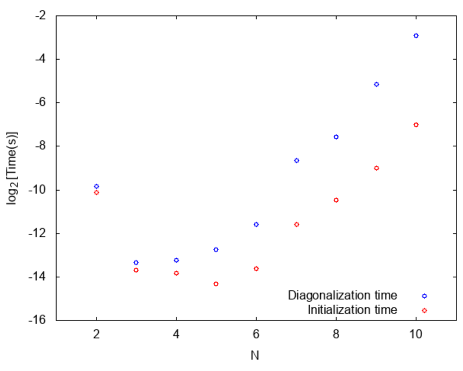
Computation time

We have calculated the first four eigenvalues for $N = 2, 3, 4, \dots, 12$, for 300 values of λ equispaced in $[0, 3]$. In order to compare the total computation time, we have completed a simulation also for $N = 13$, but since it took nearly 600 seconds, it was unthinkable to run it again for 300 times. In fact, the $N = 12$, $N_{lam} = 300$ had been completed in more than 4 hours[1].

[1]MSI Leopard Pro(2017).



(a)



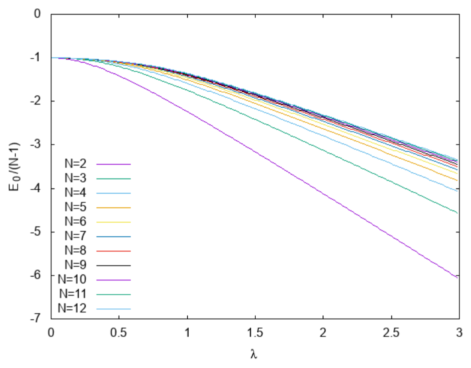
(b)

Figure 1: a) shows the total computation time vs N. b) shows the diagonalization time compared to initialization time.

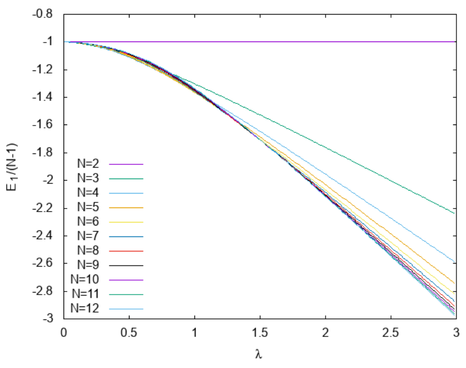
In Figure 1 graphs to compare computation time are reported. We can notice that in logarithmic scale ($\log_2(time)$ vs N) the plot is a straight line, which has been fitted from $N = 6$. We can appreciate that for lower N the total computation time is comparable with fixed-time operations, while this fixed-time becomes negligible as N grows, so that we can appreciate the linear growth, as expected. Moreover, from the right plot we can clearly see that the initialization time is negligible w.r.t. diagonalization time. Therefore, we do not loose too much time from this brute-force initialization instead of using the way which exploits Pauli's matrices symmetries.

Energy levels

Since the chain is finite, it suffers from boundary conditions. In Figure 2 the plot of first two eigenvalues of the corrected energy is reported. Since we have two spins 1/2 at boundary, the corrected energy density is $E/(N-1)$, which converges to the true energy density in the thermodynamic limit. We can notice that for $\lambda = 0$, $E = N - 1$, as we would expected from an Ising chain with a null field with open boundary conditions.



(a) First eigenvalue.



(b) Second eigenvalue.

Figure 2: The energy density for the first eigenvalues.

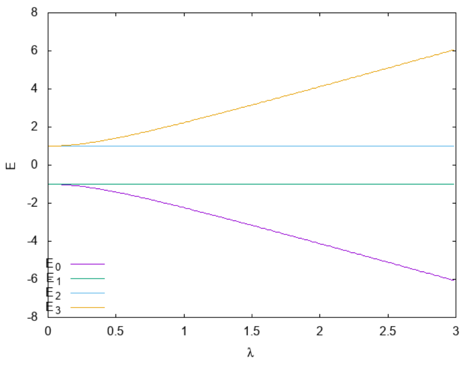
Degeneracy

We plot the the first 4 eigenvalues for different values of N. The results are reported in Figure 3.

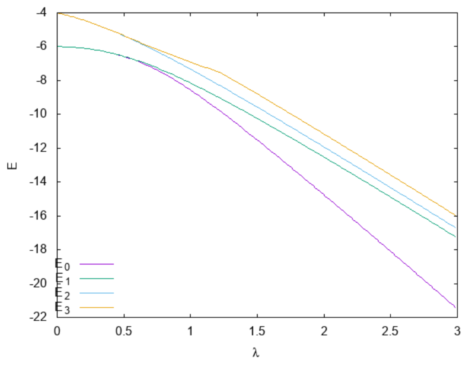
Since for $\lambda = 0$ the field is switched off, we would expect that the ground state is double degenerated, with the two states corresponding to either all spins up or all spins down. The results confirm this expectation.

Self Evaluation

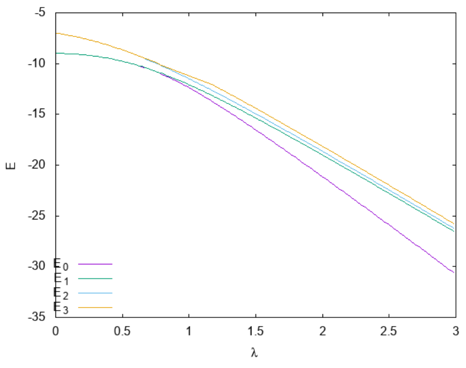
This week we have learned how to solve the transverse field Ising model. We solved it exactly, but with a limited number of spins, since the computation time grows exponentially. In order to handle with more spins,



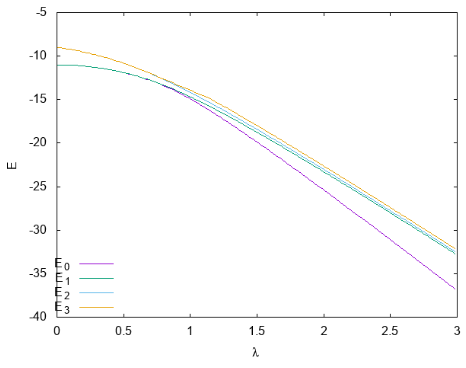
(a) N=2



(b) N=7



(c) N=10



(d) N=12

Figure 3: Degeneracy of first eigenvalues for different N.

approximated approaches must be performed.
To improve the code, we could initialize the Hamiltonian in a cleverer way, even if we have verified that the computation time needed is almost all due to diagonalization.
The code is very flexible, even if not very optimized, because it can be reconverted without too effort to other quantum problems.