Ising Model

Umberto Maria Tomasini
Quantum Information Course

11/12/2018

Abstract

In this paper We consider N interacting particles with spin $1/2$ on a one-dimensional lattice. These particles can interact only with their next neighbours. The Hamiltonian of this Ising Model is: $H = \sum_i^N \sigma_z^i + \lambda \sum_i^{N-1} \sigma_x^{i+1} \sigma_x^i$, where σ's are the Pauli Matrices and λ the interaction strength, we compute the first $k = 4$ energy levels. Moreover, we determine which is the maximum N we can reach.

Theory

The Quantum Model

We consider N interacting particles with spin $1/2$ on a one-dimensional lattice. These particles can interact only with their next neighbours. Moreover, we consider an external transverse magnetic field. Hence the Hamiltonian of this Ising problem is:

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

The notation in (1) is a shortcut. The fully-expandend notation for the external field term, by means of tensor products, is:

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

And for the interaction term:

$$\lambda \cdot (\sigma_x \otimes \sigma_x \otimes \mathbb{1} ... \otimes \mathbb{1} + ... + \mathbb{1} ... \otimes \sigma_x \otimes \sigma_x) \tag{3}$$

It is noticeable that we dropped the now useless index i.

Mean-Field Solution

In order to have a qualitative (not quantitative idea) about what should happen, we mention the Mean-Field solution of this problem.
The Mean Field solution is obtained writing the generic wave function as separable: $|\psi_N\rangle = |\psi\rangle \otimes ... |\psi\rangle$, where $|\psi_N\rangle$ is N-body wave function and $|\psi\rangle$ is 1-body wave function. Then, the next step is to minimize the energy, i.e. the expectation value of the hamiltonian operator, in order to obtain the ground state. The result is the following:

$$\begin{cases} E_{gs} = N(-1 - \lambda^2/4), & \lambda \in [-2, 2] \\ E_{gs} = N(-|\lambda|), & \lambda \notin [-2, 2] \end{cases} \tag{4}$$

We can notice a quantum phase transiction (given by the discontinuity of the second derivative) in $\lambda = \pm 2$ (we have a parabola between 2 and -2 and a liner trend outside that interval). Given this fact, we should expect a similar phase transiction for the ground state energy for a certain λ. The Mean-Field solution would be getting closer to the real solution increasing N, because it is N-indipendent.

Code Development

Subroutines to write the Hamiltonian

The first step is to write the Hamiltonian, hence a $2^N \times 2^N$ matrix.
It turns extremely useful a subroutine which computes tensor product between two matrices.

Listing 1: Fortran code: Computation of the tensor product of two matrices.

```
        subroutine ten_prod (m1,m2, mr, debug)
!This subroutine performs a tensor product
!between two matrices m1 (at left) and m2
!(at right), giving as a result the matrix mr

type(dcm) :: m1
type(dcm) :: m2
type(dcm) :: mr
logical :: debug
integer :: ii, jj, kk, hh

call init_dcm_mat (mr, m1%nr*m2%nr, m1%nc*m2%nc,debug)

do ii=1, m2%nr

do jj=1, m2%nc

do kk=1, m1%nr

do hh=1, m1%nc

ind_r=kk+(ii-1)*m1%nr
ind_c=hh+(jj-1)*m1%nc

mr%elem(ind_r,ind_c)=
$                                   m2%elem(ii,jj)*m1%elem(kk,hh)


end do

end do


end do

end do



end subroutine ten_prod
```

The external field term is computed via the following subrotuine, which performs the tensor product of N-1 identies 2×2 and one matrix located in the position pos of the chain.

Listing 2: Fortran code: Computation of the tensor product of N-1 identies 2×2 and one matrix located in the position pos of the chain.

```
subroutine ten_prod_vec1(mat_i, mat_f, nn, pos, debug)
!This subroutine performs the tensor product of
!N-1 identies 2x2
!and one matrix located in the position pos
!of the chain

type(dcm) :: mat_i
type(dcm) :: mat_f
type(dcm) :: idd
type(dcm) :: temp_mat_i
type(dcm) :: tempp_matf
logical :: debug
integer :: pos
integer :: ii, jj
integer :: my_stat
character (256) :: my_msg

!allocation iddentity
allocate (idd%elem(2,2),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate idd with stat = '
$                 , my_stat, ' and msg = '//trim(my_msg)
end if

idd%nr=2
idd%nc=2


!initialization iddentity 2x2
idd%elem(1,1)=(1.,0.)
idd%elem(1,2)=(0.,0.)
idd%elem(2,1)=(0.,0.)
idd%elem(2,2)=(1.,0.)



!allocate initial matrix of cycle
allocate(temp_mat_i%elem(2,2))
!initialization initial matrix of cycle
temp_mat_i%nr=2
```

```
temp_mat_i%nc=2

if(pos==1) then

temp_mat_i%elem= mat_i%elem

else

temp_mat_i%elem = idd%elem

end if


!cycle of tensor product
do ii=2, nn

!if pos==ii then the tensor product with the
!initial matrix is performed
!otherwise with the identity

if(pos==ii) then

call ten_prod(temp_mat_i,mat_i,tempp_matf, debug)

else

call ten_prod(temp_mat_i,idd,tempp_matf, debug)

end if

deallocate(temp_mat_i%elem)

allocate(temp_mat_i%elem(2**(ii),2**(ii)))

temp_mat_i%nr=2**(ii)
temp_mat_i%nc=2**(ii)

temp_mat_i%elem=tempp_matf%elem
deallocate(tempp_matf%elem)
deallocate(tempp_matf%eval)

end do

allocate(mat_f%elem(2**nn,2**nn), stat=my_stat, errmsg=my_msg
   )
```

```
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate mat_f with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

!final matrix 2**n x 2**n
mat_f%elem = temp_mat_i%elem

mat_f%nr = 2**nn
mat_f%nc = 2**nn


end subroutine ten_prod_vec1
```

The interaction term is computed via the following subroutine, which performs the tensor product of N-2 identies 2×2 and two identical matrices located in the position pos and $pos+1$ of the chain.

Listing 3: Fortran code: Computation of the tensor product of N-2 identies 2×2 and two identical matrices located in the position pos and $pos + 1$ of the chain.

```
        subroutine ten_prod (m1,m2, mr, debug)
subroutine ten_prod_vec2(mat_i, mat_f, nn, pos, debug)
!This subroutine performs the tensor product of
!N-2 identies 2x2
!and two identical matrices located in the position pos
! and pos+1 of the chain

type(dcm) :: mat_i
type(dcm) :: mat_f
type(dcm) :: idd
type(dcm) :: temp_mat_i
type(dcm) :: tempp_matf
logical :: debug
integer :: pos
integer :: ii, jj
integer :: my_stat
character (256) :: my_msg

!allocation iddentity
allocate (idd%elem(2,2),
$                       stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate idd with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
```

```
end if

idd%nr=2
idd%nc=2


!initialization iddentity 2x2
idd%elem(1,1)=(1.,0.)
idd%elem(1,2)=(0.,0.)
idd%elem(2,1)=(0.,0.)
idd%elem(2,2)=(1.,0.)



!allocate initial matrix of cycle
allocate(temp_mat_i%elem(2,2))
!initialization initial matrix of cycle
temp_mat_i%nr=2
temp_mat_i%nc=2

if(pos==1) then

temp_mat_i%elem = mat_i%elem

else

temp_mat_i%elem = idd%elem

end if


!cycle of tensor product
do ii=2, nn

!if pos==ii then the tensor product with the
!initial matrix is performed
!otherwise with the identity

if(pos==ii) then

call ten_prod(temp_mat_i,mat_i,tempp_matf, debug)

else if ((pos+1)==ii) then

call ten_prod(temp_mat_i,mat_i,tempp_matf, debug)
```

```
else

call ten_prod(temp_mat_i,idd,tempp_matf, debug)

end if

deallocate(temp_mat_i%elem)

allocate(temp_mat_i%elem(2**(ii),2**(ii)))

temp_mat_i%nr=2**(ii)
temp_mat_i%nc=2**(ii)

temp_mat_i%elem=tempp_matf%elem
deallocate(tempp_matf%elem)
deallocate(tempp_matf%eval)

end do

allocate(mat_f%elem(2**nn,2**nn), stat=my_stat, errmsg=my_msg
   )
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate mat_f with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

!final matrix 2**n x 2**n
mat_f%elem = temp_mat_i%elem

mat_f%nr = 2**nn
mat_f%nc = 2**nn


end subroutine ten_prod_vec2
```

Full Code

In this code, given a number nn of spin sites and the parameter of the interaction strength λ (red by a file), the corresponding Hamiltonian is written. Next, its eigenvalues are computed via the LAPACK subroutine $zheev$, and they are stored in ascending order. The first $k = 4$ eigenvalues are printed on a file, with a generic name: "ham_eval.txt".

Listing 4: Fortran code: Full code.

```
        module hamiltonian

type dcm !defining the type: doublecomplex matrix
integer :: nr !number of rows
integer :: nc !number of columns
!the actual matrix
double complex, dimension (: , :), allocatable :: elem
!eigenvalues
double precision, dimension (:), allocatable :: eval
!trace
double complex :: m_trace
!determinant
double complex :: m_det
end type dcm


contains

subroutine ten_prod (m1,m2, mr, debug)
!This subroutine performs a tensor product
!between two matrices m1 (at left) and m2
!(at right), giving as a result the matrix mr

type(dcm) :: m1
type(dcm) :: m2
type(dcm) :: mr
logical :: debug
integer :: ii, jj, kk, hh

call init_dcm_mat (mr, m1%nr*m2%nr, m1%nc*m2%nc,debug)

do ii=1, m2%nr

do jj=1, m2%nc

do kk=1, m1%nr

do hh=1, m1%nc

ind_r=kk+(ii-1)*m1%nr
ind_c=hh+(jj-1)*m1%nc

mr%elem(ind_r,ind_c)=
$                                m2%elem(ii,jj)*m1%elem(kk,hh)
```

```
end do

end do


end do

end do



end subroutine ten_prod


subroutine ten_prod_vec1(mat_i, mat_f, nn, pos, debug)
!This subroutine performs the tensor product of
!N-1 identies 2x2
!and one matrix located in the position pos
!of the chain

type(dcm) :: mat_i
type(dcm) :: mat_f
type(dcm) :: idd
type(dcm) :: temp_mat_i
type(dcm) :: tempp_matf
logical :: debug
integer :: pos
integer :: ii, jj
integer :: my_stat
character (256) :: my_msg

!allocation iddentity
allocate (idd%elem(2,2),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate idd with stat = '
$                 , my_stat, ' and msg = '//trim(my_msg)
end if

idd%nr=2
idd%nc=2
```

```
!initialization iddentity 2x2
idd%elem(1,1)=(1.,0.)
idd%elem(1,2)=(0.,0.)
idd%elem(2,1)=(0.,0.)
idd%elem(2,2)=(1.,0.)



!allocate initial matrix of cycle
allocate(temp_mat_i%elem(2,2))
!initialization initial matrix of cycle
temp_mat_i%nr=2
temp_mat_i%nc=2

if(pos==1) then

temp_mat_i%elem= mat_i%elem

else

temp_mat_i%elem = idd%elem

end if


!cycle of tensor product
do ii=2, nn

!if pos==ii then the tensor product with the
!initial matrix is performed
!otherwise with the identity

if(pos==ii) then

call ten_prod(temp_mat_i,mat_i,tempp_matf, debug)

else

call ten_prod(temp_mat_i,idd,tempp_matf, debug)

end if

deallocate(temp_mat_i%elem)

allocate(temp_mat_i%elem(2**(ii),2**(ii)))
```

```
temp_mat_i%nr=2**(ii)
temp_mat_i%nc=2**(ii)

temp_mat_i%elem=tempp_matf%elem
deallocate(tempp_matf%elem)
deallocate(tempp_matf%eval)

end do

allocate(mat_f%elem(2**nn,2**nn), stat=my_stat, errmsg=my_msg
   )
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mat_f␣with␣stat␣=␣'
$                , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if

!final matrix 2**n x 2**n
mat_f%elem = temp_mat_i%elem

mat_f%nr = 2**nn
mat_f%nc = 2**nn


end subroutine ten_prod_vec1


subroutine ten_prod_vec2(mat_i, mat_f, nn, pos, debug)
!This subroutine performs the tensor product of
!N-2 identies 2x2
!and two identical matrices located in the position pos
! and pos+1 of the chain

type(dcm) :: mat_i
type(dcm) :: mat_f
type(dcm) :: idd
type(dcm) :: temp_mat_i
type(dcm) :: tempp_matf
logical :: debug
integer :: pos
integer :: ii, jj
integer :: my_stat
character (256) :: my_msg
```

```
!allocation iddentity
allocate (idd%elem(2,2),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate idd with stat = '
$                 , my_stat, ' and msg = '//trim(my_msg)
end if

idd%nr=2
idd%nc=2


!initialization iddentity 2x2
idd%elem(1,1)=(1.,0.)
idd%elem(1,2)=(0.,0.)
idd%elem(2,1)=(0.,0.)
idd%elem(2,2)=(1.,0.)



!allocate initial matrix of cycle
allocate(temp_mat_i%elem(2,2))
!initialization initial matrix of cycle
temp_mat_i%nr=2
temp_mat_i%nc=2

if(pos==1) then

temp_mat_i%elem = mat_i%elem

else

temp_mat_i%elem = idd%elem

end if


!cycle of tensor product
do ii=2, nn

!if pos==ii then the tensor product with the
!initial matrix is performed
!otherwise with the identity
```

```
if(pos==ii) then

call ten_prod(temp_mat_i,mat_i,tempp_matf, debug)

else if ((pos+1)==ii) then

call ten_prod(temp_mat_i,mat_i,tempp_matf, debug)

else

call ten_prod(temp_mat_i,idd,tempp_matf, debug)

end if

deallocate(temp_mat_i%elem)

allocate(temp_mat_i%elem(2**(ii),2**(ii)))

temp_mat_i%nr=2**(ii)
temp_mat_i%nc=2**(ii)

temp_mat_i%elem=tempp_matf%elem
deallocate(tempp_matf%elem)
deallocate(tempp_matf%eval)

end do

allocate(mat_f%elem(2**nn,2**nn), stat=my_stat, errmsg=my_msg
   )
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate mat_f with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

!final matrix 2**n x 2**n
mat_f%elem = temp_mat_i%elem

mat_f%nr = 2**nn
mat_f%nc = 2**nn


end subroutine ten_prod_vec2
```

```
subroutine init_dcm_mat (aa, numrow, numcol,debug)
!this subroutine initializes a type dcm,
!given as imput. The matrix becomes Herminian.
!Also the numbers of rows and columns
!are given as imput. The subroutine creates random matrices
!with values in range [0,1].
!It gives a default value 0 to the determinant, the trace
!and to eigenvalues.
real*8 yy, xx
integer ii,jj
type(dcm) :: aa
integer numrow, numcol
integer :: my_stat
character (256) :: my_msg
logical debug


aa%nr= numrow
aa%nc= numcol

!ERROR HANDLING ALLOCATION VECTORS
allocate (aa%eval(aa%nr),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aa%eval with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

!ERROR HANDLING ALLOCATION MATRICES
allocate(aa%elem(aa%nr, aa%nc), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aa with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if



!WARNING IF NOT SQUARED
if(debug .eqv. .true.) then
if(numrow==numcol) then !checking squareness
print*, " "
print*, "INITIALIZATION"
```

```
print*, "Okay, squared"
else
print*, "WARNING: NOT SQUARED"
print*, "num rows= ", numrow
print*, "num columns= ", numcol
end if
end if

!INITIALIZATION
do ii= 1, aa%nr
do jj= 1, aa%nc
!call random_number(xx)
!call random_number(yy)
aa%elem(ii,jj)=cmplx(0.,0.)!complex number
end do
end do
aa%m_trace=(0d0,0d0)
aa%m_det=(0d0,0d0)

do ii= 1, numrow
aa%eval(ii)=0.
end do

end subroutine init_dcm_mat

subroutine print_matrices (AA, namefile, unitt)
! This subroutine prints a type doublecomplex_matrix:
!the numbers of row and columns, the elements (in the proper
   order)
!, the trace and the determinant. This is all printed on a
   file
type(dcm) :: AA
character(:), allocatable :: namefile
integer :: ii,jj, unitt
integer :: my_stat
character (256) :: my_msg

open(unit = unitt, file = namefile, status = "unknown",
$                        iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open failed with stat = ",
$                       my_stat, " msg = "//trim(my_msg)
end if
```

```
write (unitt,*) "NUMBER␣OF␣ROWS:", AA%nr
write (unitt,*) "NUMBER␣OF␣COLUMNS:", AA%nc
write (unitt,*) "ELEMENTS:"
do ii=1, AA%nr !do cycle in order to print in the proper
   order
write (unitt,*) (AA%elem(ii, jj), jj = 1, AA%nc)
end do
write (unitt,*) "TRACE:", AA%m_trace
write (unitt,*) "DET:", AA%m_det
write (unitt,*) "EIGENVALUES:"
do ii=1, AA%nr !do cycle in order to print in the proper
   order
write (unitt,*) AA%eval(ii)
end do
close(unitt)

end subroutine print_matrices

subroutine print_vector (vec, nn, namefile, unitt)
! This subroutine prints a vector on file,
! given its length nn
character(:), allocatable :: namefile
integer :: ii,nn, unitt
integer :: my_stat
character (256) :: my_msg
double precision, dimension(:), allocatable :: vec

open(unit = unitt, file = namefile, status = "unknown",
$                       iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open␣failed␣with␣stat␣=␣"
$                           , my_stat, "␣msg␣=␣"//trim(my_msg)
end if

do ii=1, nn !do cycle in order to print
write (unitt,*) vec(ii)
end do

close(unitt)
end subroutine print_vector


end module hamiltonian
```

```
program week9

use hamiltonian


logical :: debug
integer :: ii, jj, kk, hh
character(:), allocatable :: namefile
type(dcm) :: pauli_x
type(dcm)  :: pauli_z
type(dcm)  :: mat_f
type(dcm)  :: ham
integer :: my_stat
character (256) :: my_msg
integer :: nn, pos
type(dcm) :: m1,m2,mr
double precision :: lamb
double precision, allocatable:: tempeval(:)
double complex, dimension (: , :), allocatable :: tempmat

complex*16, allocatable:: work(:)
integer lwork
double precision, allocatable:: rwork(:)
integer iinfo
integer :: aa,bb

c       call init_dcm_mat(m1,3,3,debug)
c       call init_dcm_mat(m2,3,3,debug)


debug=.false.

!allocation Pauli matrices
allocate (pauli_x%elem(2,2),
$                       stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate pauli_x with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

pauli_x%nr=2
pauli_x%nc=2
```

```
!allocation Pauli matrices
allocate (pauli_z%elem(2,2),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate pauli_z with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

pauli_z%nr=2
pauli_z%nc=2

!initialization Pauli matrices
!along x
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.)
!along z
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.)

!HERE THE NUMBER SITES nn IS TAKEN AS IMPUT FROM
!FILE "MatDimension.txt"
!AND THE PARAMETER lambda

open(unit = 30, file = "Parameters.txt",
$       status = "unknown",
$  iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, 'Open -MatDimension- failed with stat = '
$                         , my_stat, ' msg = '//trim(my_msg)
end if

read(30,*) nn, lamb


!allocation ham
call init_dcm_mat(ham,2**nn,2**nn,debug)
```

```
!computing the external field part of the hamiltonian
do pos=1,nn

!tensor product of N-1 identies and one pauli_z
!located in the position pos of the chain
call ten_prod_vec1(pauli_z,mat_f,nn,pos,debug)

ham%elem=ham%elem+mat_f%elem
deallocate(mat_f%elem)

end do


!computing the sites interactions part of the hamiltonian
do pos=1,nn-1

!tensor product of N-2 identies and
!two closepauli_x, the first is located in
!the position pos of the chain
call ten_prod_vec2(pauli_x,mat_f,nn,pos,debug)

ham%elem=ham%elem+lamb*mat_f%elem
deallocate(mat_f%elem)

end do


c       do ii=1,ham%nr
c       do jj=1,ham%nc
c
c       print*, ham%elem(ii,jj)
c
c       end do
c       end do


c       print*, "vv",ham%elem(ham%nr,ham%nc)
c       print*, ham%nr, ham%nc

!calling the LAPACK subroutine for eigen values
!necessary to allocate work and rwork
!and initialize lwork
!for information read relative documentation
lwork= 2*(2**nn)
```

```
allocate(work(lwork),
$                         stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate work with stat = '
$                 , my_stat, ' and msg = '//trim(my_msg)
end if

allocate(rwork(lwork),
$                         stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate rwork with stat = '
$                 , my_stat, ' and msg = '//trim(my_msg)
end if

allocate(tempeval(2**nn))

allocate(tempmat(2**nn,2**nn))

tempmat=ham%elem

aa=2**nn
bb=2**nn

call zheev( "N", "U", aa , tempmat , bb , tempeval , work ,
   lwork,
$                         rwork, iinfo )

!"N"->only eigenvalues
!"U"->upper triangle of aa is stored, inside dd%elem
! aa%eval will contain the eigen values of aa
!in ascending order(if INFO==0)

!DEBUG
if(debug.eqv..TRUE.) then
print*, " "
print*, "DIAGONALIZATION"
print*, " "
end if


if(debug.eqv..TRUE.) then!check diag
if (iinfo==0) then
print*, " "
```

```
print*, "Successful␣DIAGONALIZATION"
else if (iinfo < 0) then
print*, "␣"
print*, "the", iinfo, "-th␣argument
$␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣of␣ipiv␣had␣an␣illegal␣value"

else
print*, "␣"
print*,  "the␣algorithm␣failed␣to␣converge"
end if
end if

namefile="ham_eval.txt"
call print_vector(tempeval, 2**nn, namefile, 10)




end program week9
```

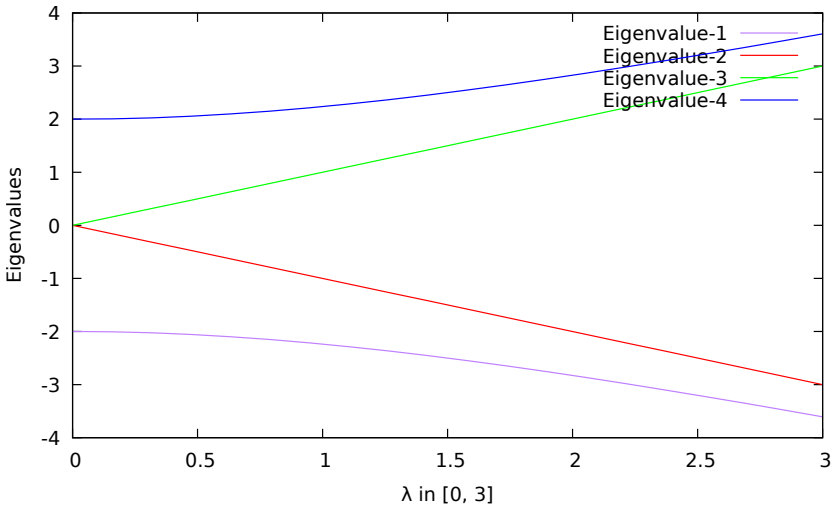
Results, without determinant

N_{max}

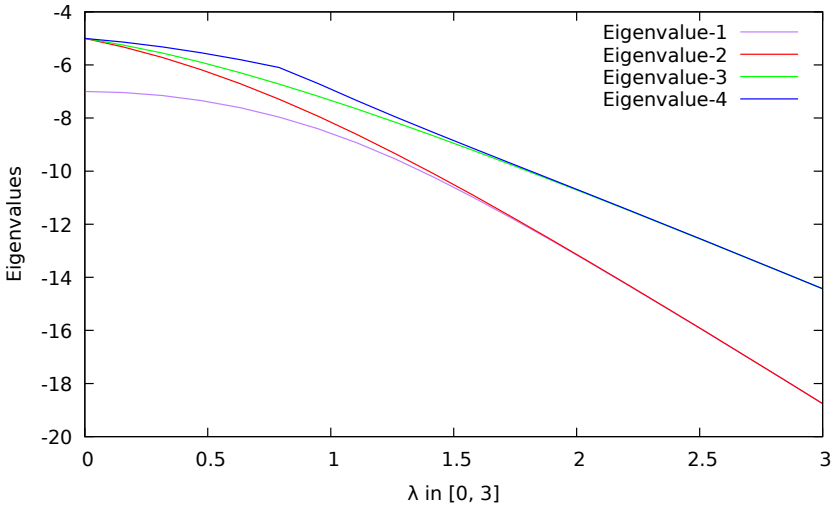
Via an opportune python script, we have tested the above code for different number of sites nn. We have increased that number until the program crashed. We have found that the upper limit of our Virtual Box was $N_{max} = 12$, which corresponds a hamiltonian matrix 4096×4096.

First $k = 4$ eigenvalues as a function of λ

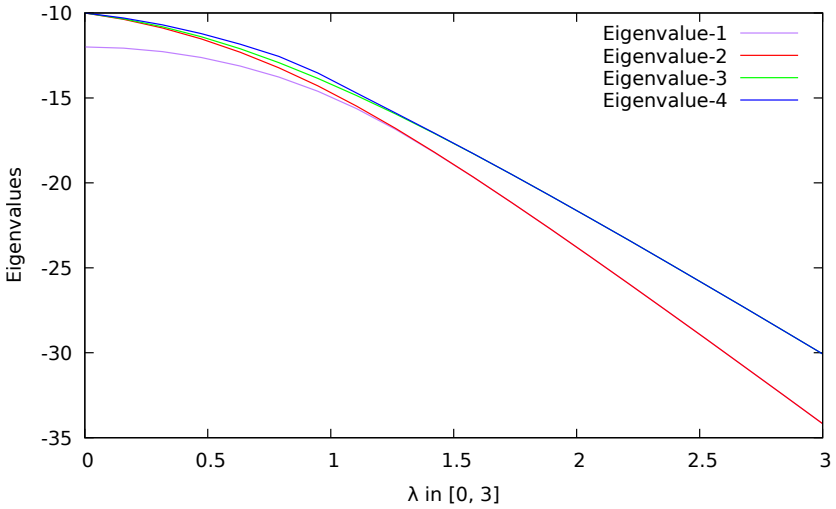
We have plotted the first $k = 4$ eigenvalues as a function of λ, with 20 values of λ taken inside the interval $[0, 3]$. Those lambda were chosen linearly: $\lambda_i = 0 + i * (3/(20 - 1))$.
We have done this for three different number of sites: 2, 7 and 12.



First $k = 4$ eigenvalues as a function of $\lambda \in [0, 3]$, for $nn = 2$.



First $k = 4$ eigenvalues as a function of $\lambda \in [0, 3]$, for $nn = 7$.



First $k = 4$ eigenvalues as a function of $\lambda \in [0, 3]$, for $nn = 12$.

Energy spectrum comments

First of all, we observe what happens without interactions, i.e. for $\lambda = 0$. There is only the external field. Focusing on the $nn = 2$ case, we notice there are only four eigenvalues, with two coincident and equal to zero. This is reasonable: we have only 4 configurations. The less energetic one is the one with both spins in the direction of the magnetic field, the most energetic one is the one with both spins in the direction opposite of the field, and then there are two configurations with correspond to the triplet and singlet state, with equal null energy. For $nn = 7$ and $nn = 12$ we have more configurations, but we can see ony the four less energetic, since we have plotted only the first four eigenvalues. Neverthless, we notice that for $nn = 7$ there is an eigenvalue with energy -7, which corresponds with the configuration with all spins in the external field's direction. The same happens for $nn = 12$, where the less energetic eigenvalues has energy -12.

Secondly, we notice that incresing the strenght of the interactions (increasing λ) the absolute value of the eigenvalues increase. This is reasonable: the interaction term in the hamiltonian is linear in λ. Moreover, the degeneration of the energy eigenvalues is removed until a certain value of λ, as we can infer from the $nn = 2$ state. In fact, the triplet and singlet state have different energies if the mutual interaction is switched on. The state which is a triplet for $\lambda = 0$ should have less energy than the other state. This is due to the fact that the spins are directed in the same verse, hence there is less mutual interaction. This is the "exchange splitting effect", referring to [1].

On the other hand, increasing further λ and overcoming a certain value λ_c, the eigenvalues seem to become degenerate again (this is visualized better in the higher nn cases). Moreover, the trend of the eigen values after λ_c tend to be linear. This is expected: this is the second-derivative discontinuity found in the mean-field solution, which is the signal of

a quantum phase transiction. In particular, for λ greater or equal to λ_c the average magnetization along the x axis is not null anymore. The fact that the eigenvalues return to be degenerate (in couples) is due to the fact a new Z_2 simmetry occurs. The mean-field approximation confirms to be qualitatively good: as nn increses, its predictive ability increase. Nevertheless, it is quantitatively inaccurate: in the mean-field solution the transition occurs for $\lambda_c = 2$, while for $nn = 7$ it occurs a little before $\lambda = 1.5$ and for $nn = 12$ a little more than $\lambda = 1$.

Self-Evaluation

The main goals achieved:

- Dealing with a one-dimensional Ising Model, with an nearest-neighbours interaction terms and a transverse magnetic field.
- Dealing with tensor products

What can be done next:

- Modify the interaction in a such a way that it is weaker as the spins are less close, in order to consider an intermediate case between the nearest-neighbours case and the mean-field case.
- Improve the efficiency: write a subroutine which is able to skip the tensor product with identities, giving the right result, not store the zeros, use a LAPACK subroutine which computes the eigenvalues for real matrices, instead of hermitian ones (as the used $zheev$ does).

(

Documentation)

Listing 5: Fortran code: Computation of the tensor product of N-1 identies 2×2 and one matrix located in the position pos of the chain.

```
      C  ================================================C
         ================================================
      C
      C       subroutine ten_prod (m1,m2, mr, debug)
      C
      C       .. Scalar Arguments ..
      C       logical           debug

      C       ..
      C       .. Array Arguments ..
      C       type(dcm)       m1
      C       type(dcm)       m2
```

```
C       type(dcm)       mr

C
C
C         Purpose
C         =======
C
C       \details \b Purpose:
C       \verbatim
C
C       !This subroutine performs a tensor product
!between two matrices m1 (at left) and m2
!(at right), giving as a result the matrix mr
C
C        Arguments:
C         ==========
C

C        \param[in] m1
C        \verbatim
C          m1 is type(dcm)
C        \endverbatimm
C
C        \param[in] m2
C        \verbatim
C          m2 is type(dcm)
C        \endverbatimm
C
C        \param[out] mr
C        \verbatim
C          mr is type(dcm)
C        \endverbatimm
C
C        \param[in] debug
C        \verbatim
C          debug is logical
C        \endverbatim
C
C  ================================================C
   ================================================
C
C       subroutine ten_prod_vec1(mat_i, mat_f, nn,
   pos, debug)
C
C       .. Scalar Arguments ..
```

```
C       logical            debug
C       integer            nn
C       integer            pos

C       ..
C       .. Array Arguments ..
C       type(dcm)        mat_i
C       type(dcm)        mat_f
C
C
C
C         Purpose
C         =======
C
C       \details \b Purpose:
C       \verbatim
C
!This subroutine performs the tensor product of
!N-1 identies 2x2
!and one mat_i located in the position pos
!of the chain

C
C        Arguments:
C         ==========
C

C         \param[in] mat_i
C         \verbatim
C           mat_i is type(dcm)
C         \endverbatimm
C
C         \param[out] mat_f
C         \verbatim
C           mat_f is type(dcm)
C         \endverbatimm
C
C         \param[in] nn
C         \verbatim
C           nn is integer
C         \endverbatimm
C
C         \param[in] pos
C         \verbatim
C           pos is integer
```

```
C         \endverbatimm

C         \param[in] debug
C         \verbatim
C           debug is logical
C         \endverbatim
C
C  ================================================C
   ================================================
C
C       subroutine ten_prod_vec2(mat_i, mat_f, nn,
   pos, debug)
C
C       .. Scalar Arguments ..
C       logical           debug
C       integer           nn
C       integer           pos

C       ..
C       .. Array Arguments ..
C       type(dcm)       mat_i
C       type(dcm)       mat_f
C
C
C
C         Purpose
C         =======
C
C       \details \b Purpose:
C       \verbatim
C
!This subroutine performs the tensor product of
!N-2 identies 2x2
!and two mat_i located in the position pos
!and pos+1 of the chain


C
C         Arguments:
C          ==========
C

C         \param[in] mat_i
C         \verbatim
C           mat_i is type(dcm)
```

```
C         \endverbatimm
C
C         \param[out] mat_f
C         \verbatim
C           mat_f is type(dcm)
C         \endverbatimm
C
C         \param[in] nn
C         \verbatim
C           nn is integer
C         \endverbatimm
C
C         \param[in] pos
C         \verbatim
C           pos is integer
C         \endverbatimm

C         \param[in] debug
C         \verbatim
C           debug is logical
C         \endverbatim
C
C
```

Bibliography

[1] Ashcroft, Mermin: *Solid State Physics*.