Renormalization Group

Umberto Maria Tomasini
Quantum Information Course

14/01/2018

Abstract

By means of the real-space RG algorithm we compute the ground state energy for the transverse field Ising model (at different values of the strength of the mutual interaction between spins). We do the same by means of the Infinite DMRG algorithm

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:

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

Real-space RG algorithm

Starting with a system of little number of sites N, the goal of the real-space RG algorithm is to describe the ground state of system composed by as many as possible spins by means of an hamiltonian with size equal to the one of the initial system. The algorithm is the following:

- Start with an hamiltonian of N sites H_N.
- Form a compound system composed of two sub-systems with N spins, with hamiltonian:
$$H_{2N} = H_N \otimes \mathbb{1}_N + \mathbb{1}_N \otimes H_N + H_{int\,,2N}^N \tag{2}$$
where $H_{int\,,2N}^N = \lambda(\mathbb{1}_2 \otimes ...\sigma_x^N \otimes \sigma_x^{N+1} ... \otimes \mathbb{1}_2)$ is the term which describes the interaction between the N and $N+1$ sites in a system composed by $2N$ sites at the first cycle. From the second cycle onwards it must be updated. Later we show how.
- Diagonalize H_{2N}.
- Compose a matrix P whose columns are the 2^N lowest eigenvectors of H_{2N}.
- Project H_{2N} on an hamiltonian H_N^{tr} via:
$$H_N^{tr} = P^\dagger \, H_{2N} \, P$$
- Reiterate from the first point, a number n_{tr} of times.
- Update the interaction term as follows. Be $L = \mathbb{1}_2 \otimes ...\sigma_x$ and $R = \sigma_x ... \otimes \mathbb{1}_2$ the right and left parts of the first cycle interaction term. The new right and left parts are obtained as follows: $L' = P^\dagger(\mathbb{1}_2 \otimes ... \otimes L)P$ and $R' = P^\dagger(R \otimes ... \otimes \mathbb{1}_2)P$. The number of identities is N for both L' and R'. The interaction term updated is $H_{int\,,2N}^N = \lambda(L' \otimes R')$. The next cycle the update will be done on L' and R'.
- Take the lowest eigenvalue of the last H_N^{tr}, divide it for the number of sites it describes, i.e. $2^{n_{tr}}N$. This is the ground state energy given by the algorithm.

Infinite DMRG algorithm

Starting with a system of little number of sites $m + d$, also the goal of the real-space RG algorithm is to describe the ground state of system composed by as many as possible spins by means of an hamiltonian with a smaller size, in this case the one relative to a system of $2(m + d)$ sites. The algorithm is the following:

- Start with an hamiltonian of m sites H_m.
- Add d sites to the right, in a fashion similar to Infinite RG algorithm, obtaining H_{m+d}. It must be noticed that the right part (referred to the d sites) does not need to be updated.
- Form a compund system composed of two sub-systems with $m + d$ spins. This is done by reflection: we add another d sites at the right and then the system described by H_m. H_m has the dimension of an hamiltonian with m sites, but it will describe many more in the following cycles.The interaction terms need do be updated only in the parts related to the $m-$blocks. The final hamiltonian is $H_{2(m+d)}$.
- Diagonalize $H_{2(m+d)}$ and take the eigenvector relative to the lowest eigenvalue.
- Compose a density matrix ρ by means of that eigenvector.
- Trace out the right $m + d$ system, obtaining the reduced density matrix ρ_L of the left $m + d$ system.
- Diagonalize ρ_L.
- Compose a matrix P whose columns are the 2^m eigenvectors with higher eigenvalue of ρ_L, i.e. the most populated ones.
- Project H_{m+d} on an hamiltonian H_m^{tr} via:

$$H_m^{tr} = P^\dagger \, H_{m+d} \, P$$

- Reiterate from the first point, a number n_{tr} of times.
- Take the lowest eigenvalue of the last $H_{2(m+d)}$, divide it for the number of sites it describes, i.e. $2(m + n_{tr} \cdot d)$. This is the ground state energy given by the algorithm.

Code Development

Real-space RG algorithm

The following code implements the real-space RG algorithm, by means of the subroutines of the past weeks (not presented here). Each cycle λ and the new truncated hamiltonian are divided by 2, in order to keep the number low (it is the same of dividing the final ground state eigenvalue by the number of spins).

Listing 1: Fortran code: Real-space RG algorithm

```
        program week11

        program week11

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
type2(dcm) :: m1,m2,mr
double precision :: lamb, truelamb
double precision, allocatable:: tempeval(:)
double complex, dimension (: , :), allocatable :: tempmat
double complex, dimension (: , :), allocatable :: tempmat1

complex*16, allocatable:: work(:)
integer lwork
double precision, allocatable:: rwork(:)
complex*16, allocatable:: work1(:)
integer lwork1
double precision, allocatable:: rwork1(:)
integer iinfo
integer :: aa,bb
type(dcm) :: dham
type(dcm) :: temp_dmat
type(dcm) :: temp_dmat1
type(dcm) :: temp_dmat2
type(dcm) :: temp_dmat22
type(dcm) :: idd
type(dcm) :: half1
type(dcm) :: half2
type(dcm) :: temp_int
double complex, dimension (: , :), allocatable :: pr
double complex, dimension (: , :), allocatable :: prt
double complex, dimension (: , :), allocatable :: temp_tr
integer :: ntr,gg
```

```
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

!number iterations RG algorithm

ntr=100


!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
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


!duplicating the hamiltonian: from nn to 2nn

!temporary double matrix
temp_dmat%nr=2**(2*nn)
temp_dmat%nc=2**(2*nn)

temp_dmat1%nr=2**(2*nn)
temp_dmat1%nc=2**(2*nn)

!allocation identity
allocate (idd%elem(2**(nn),2**(nn)),
$                        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**(nn)
idd%nc=2**(nn)


!initialization identity
do ii=1, 2**(nn)

do jj=1, 2**(nn)

if(ii==jj) then

idd%elem(ii,ii)=(1.0,0.0)

else

idd%elem(ii,jj)=(0.0,0.0)

end if

end do
end do


!two halves of the interaction term, first cycle
call ten_prod_vec1(pauli_x, temp_dmat2, nn, nn, debug)
call ten_prod_vec1(pauli_x, temp_dmat22, nn, 1, debug)

!allocation double hamiltonian
call init_dcm_mat(dham,2**(2*nn),2**(2*nn),debug)

!necessary to allocate work and rwork
!and initialize lwork
!for information read relative documentation
lwork= 2*(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

!second
lwork1= 2*(2**(nn))

allocate(work1(lwork1),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣work1␣with␣stat␣=␣'
$                , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if

allocate(rwork1(lwork1),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
```

```
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣rwork1␣with␣stat␣=␣'
$                 , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if







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

!projections

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

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

!auxialiary matrices for zheev


allocate(tempeval(2**(2*nn)))

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

allocate(tempmat1(2**(nn),2**(nn)))
```

```
!CYCLE OVER TRUNCATIONS

!in order to keep the numbers low, divide per 2 each cycle
do ii=1,2**(nn)
do jj=1,2**(nn)
ham%elem(ii,jj)=ham%elem(ii,jj)/2.
c                temp_dmat2%elem(ii,jj)=temp_dmat2%elem(ii,jj)
   /2.
end do
end do
truelamb=lamb
lamb=lamb/2.

!BEGINNING CYCLE

do gg=1, ntr

!connecting the two systems

!direct product

!first part
call ten_prod (ham,idd,temp_dmat, debug)

dham%elem = temp_dmat%elem

deallocate(temp_dmat%elem)
deallocate(temp_dmat%eval)


!second part

call ten_prod (idd,ham,temp_dmat1, debug)

dham%elem = dham%elem + temp_dmat1%elem

deallocate(temp_dmat1%elem)
deallocate(temp_dmat1%eval)

!nn interaction among the system

call ten_prod (temp_dmat2,temp_dmat22,temp_int, debug)
dham%elem = dham%elem + lamb*temp_int%elem
```

```
deallocate(temp_int%elem)
deallocate(temp_int%eval)

!calling the LAPACK subroutine for eigen values

tempmat=dham%elem

aa=2**(2*nn)
bb=2**(2*nn)



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

!"V"->eigenvalues and eigenvectors,
!stored inside tempeval and tempmat
!"U"->upper triangle of aa is stored, inside dd%elem
! tempeval 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
```

```
!take only first nn eigenvectors and form a projector
!i.e. the first nn columns of the temporary matrix tempmat

!projector

do ii=1, 2**(2*nn)

do jj=1, 2**(nn)

pr(ii,jj)=tempmat(ii,jj)

end do

end do



!transpose projector

do ii=1, 2**(nn)

do jj=1, 2**(2*nn)

prt(ii,jj)=conjg(pr(jj,ii))

end do

end do


!computation truncated matrix


call rowrowmatmul(2**(2*nn),2**(2*nn),2**(nn),dham%elem,
$                        pr,temp_tr)




call rowrowmatmul(2**(nn),2**(2*nn),2**(nn),prt,temp_tr,
$                        ham%elem)
```

```
aa=2**(nn)
bb=2**(nn)

!in order to keep the numbers low, divide per 2 each cycle
do ii=1,2**(nn)
do jj=1,2**(nn)
tempmat1(ii,jj)=ham%elem(ii,jj)/2.
ham%elem(ii,jj)=ham%elem(ii,jj)/2.
c                temp_dmat2%elem(ii,jj)=temp_dmat2%elem(ii,jj)
   /2.
end do
end do

lamb=lamb/2.

!COMPUTATION EIGENVALUE GS TRUNCATED SYSTEM

call zheev( "N", "U", aa , tempmat1 , bb , ham%eval , work1,
$                         lwork1, rwork1, iinfo )

print*, gg, (2**(gg+1)), ham%eval(1)

CCCCCCC !old halves tensor product with identities

call ten_prod_vec1m(temp_dmat2, half1 , 2**nn, 2**nn,
$                nn+1, nn+1, debug)
call ten_prod_vec1m(temp_dmat22, half2 , 2**nn, 2**nn,
$                nn+1, 1, debug)



!computation truncated interaction terms

!first half
call rowrowmatmul(2**(2*nn),2**(2*nn),2**(nn),half1%elem,
$                        pr,temp_tr)





call rowrowmatmul(2**(nn),2**(2*nn),2**(nn),prt,temp_tr,
$                        temp_dmat2%elem)

!second half
call rowrowmatmul(2**(2*nn),2**(2*nn),2**(nn),half2%elem,
$                        pr,temp_tr)




call rowrowmatmul(2**(nn),2**(2*nn),2**(nn),prt,temp_tr,
$                        temp_dmat22%elem)

deallocate(half1%elem)
deallocate(half2%elem)


end do

open(unit = 50, file = "RG_1gs.txt", 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(50,*) truelamb, -abs(ham%eval(1))


end program week11
        program week11

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, truelamb
double precision, allocatable:: tempeval(:)
double complex, dimension (: , :), allocatable :: tempmat
double complex, dimension (: , :), allocatable :: tempmat1

complex*16, allocatable:: work(:)
integer lwork
double precision, allocatable:: rwork(:)
complex*16, allocatable:: work1(:)
integer lwork1
double precision, allocatable:: rwork1(:)
integer iinfo
integer :: aa,bb
type(dcm) :: dham
type(dcm) :: temp_dmat
type(dcm) :: temp_dmat1
type(dcm) :: temp_dmat2
type(dcm) :: temp_dmat22
type(dcm) :: idd
type(dcm) :: half1
type(dcm) :: half2
type(dcm) :: temp_int
double complex, dimension (: , :), allocatable :: pr
double complex, dimension (: , :), allocatable :: prt
double complex, dimension (: , :), allocatable :: temp_tr
integer :: ntr,gg

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

!number iterations RG algorithm

ntr=100


!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
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


!duplicating the hamiltonian: from nn to 2nn

!temporary double matrix
temp_dmat%nr=2**(2*nn)
temp_dmat%nc=2**(2*nn)

temp_dmat1%nr=2**(2*nn)
temp_dmat1%nc=2**(2*nn)

!allocation identity
allocate (idd%elem(2**(nn),2**(nn)),
$                       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**(nn)
idd%nc=2**(nn)


!initialization identity
do ii=1, 2**(nn)

do jj=1, 2**(nn)

if(ii==jj) then

idd%elem(ii,ii)=(1.0,0.0)

else

idd%elem(ii,jj)=(0.0,0.0)

end if

end do

end do


!two halves of the interaction term, first cycle
call ten_prod_vec1(pauli_x, temp_dmat2, nn, nn, debug)
call ten_prod_vec1(pauli_x, temp_dmat22, nn, 1, debug)

!allocation double hamiltonian
call init_dcm_mat(dham,2**(2*nn),2**(2*nn),debug)

!necessary to allocate work and rwork
!and initialize lwork
!for information read relative documentation
lwork= 2*(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

!second
lwork1= 2*(2**(nn))

allocate(work1(lwork1),
$                               stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣work1␣with␣stat␣=␣'
$                     , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if

allocate(rwork1(lwork1),
$                               stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣rwork1␣with␣stat␣=␣'
$                     , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if







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

!projections

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

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

!auxialiary matrices for zheev


allocate(tempeval(2**(2*nn)))

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

allocate(tempmat1(2**(nn),2**(nn)))


!CYCLE OVER TRUNCATIONS

!in order to keep the numbers low, divide per 2 each cycle
do ii=1,2**(nn)
do jj=1,2**(nn)
ham%elem(ii,jj)=ham%elem(ii,jj)/2.
c                temp_dmat2%elem(ii,jj)=temp_dmat2%elem(ii,jj)
   /2.
end do
end do
truelamb=lamb
lamb=lamb/2.

!BEGINNING CYCLE

do gg=1, ntr

!connecting the two systems

!direct product
```

```
!first part
call ten_prod (ham,idd,temp_dmat, debug)

dham%elem = temp_dmat%elem

deallocate(temp_dmat%elem)
deallocate(temp_dmat%eval)


!second part

call ten_prod (idd,ham,temp_dmat1, debug)

dham%elem = dham%elem + temp_dmat1%elem

deallocate(temp_dmat1%elem)
deallocate(temp_dmat1%eval)

!nn interaction among the system

call ten_prod (temp_dmat2,temp_dmat22,temp_int, debug)
dham%elem = dham%elem + lamb*temp_int%elem

deallocate(temp_int%elem)
deallocate(temp_int%eval)

!calling the LAPACK subroutine for eigen values

tempmat=dham%elem

aa=2**(2*nn)
bb=2**(2*nn)



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

!"V"->eigenvalues and eigenvectors,
!stored inside tempeval and tempmat
!"U"->upper triangle of aa is stored, inside dd%elem
! tempeval 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

!take only first nn eigenvectors and form a projector
!i.e. the first nn columns of the temporary matrix tempmat

!projector

do ii=1, 2**(2*nn)

do jj=1, 2**(nn)

pr(ii,jj)=tempmat(ii,jj)

end do

end do



!transpose projector

do ii=1, 2**(nn)
do jj=1, 2**(2*nn)

prt(ii,jj)=conjg(pr(jj,ii))

end do

end do


!computation truncated matrix


call rowrowmatmul(2**(2*nn),2**(2*nn),2**(nn),dham%elem,
$                        pr,temp_tr)




call rowrowmatmul(2**(nn),2**(2*nn),2**(nn),prt,temp_tr,
$                        ham%elem)



aa=2**(nn)
bb=2**(nn)

!in order to keep the numbers low, divide per 2 each cycle
do ii=1,2**(nn)
do jj=1,2**(nn)
tempmat1(ii,jj)=ham%elem(ii,jj)/2.
ham%elem(ii,jj)=ham%elem(ii,jj)/2.
c                temp_dmat2%elem(ii,jj)=temp_dmat2%elem(ii,jj)
   /2.
end do
end do

lamb=lamb/2.

!COMPUTATION EIGENVALUE GS TRUNCATED SYSTEM

call zheev( "N", "U", aa , tempmat1 , bb , ham%eval , work1,
$                         lwork1, rwork1, iinfo )

print*, gg, (2**(gg+1)), ham%eval(1)
```

```
CCCCCCC !old halves tensor product with identities

call ten_prod_vec1m(temp_dmat2, half1 , 2**nn, 2**nn,
$                nn+1, nn+1, debug)
call ten_prod_vec1m(temp_dmat22, half2 , 2**nn, 2**nn,
$                nn+1, 1, debug)



!computation truncated interaction terms

!first half
call rowrowmatmul(2**(2*nn),2**(2*nn),2**(nn),half1%elem,
$                        pr,temp_tr)





call rowrowmatmul(2**(nn),2**(2*nn),2**(nn),prt,temp_tr,
$                        temp_dmat2%elem)

!second half
call rowrowmatmul(2**(2*nn),2**(2*nn),2**(nn),half2%elem,
$                        pr,temp_tr)





call rowrowmatmul(2**(nn),2**(2*nn),2**(nn),prt,temp_tr,
$                        temp_dmat22%elem)

deallocate(half1%elem)
deallocate(half2%elem)


end do

open(unit = 50, file = "RG_1gs.txt", 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(50,*) truelamb, -abs(ham%eval(1))


end program week11
```

Infinite DMRG algorithm

The following code implements the infinite DMRGRG algorithm, by means of the subroutines of the past weeks (not presented here). Due to the lowest numbers of sites per number of cycles respect to the real space RG algorithm, we divide per the number of sites directly at the end of the algorithm.

Listing 2: Fortran code: Infinite DMRG algorithm

```
        program week11

        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
        type(dcm)  :: hamf
        integer :: my_stat
        character (256) :: my_msg
        integer :: nn, pos
        type(dcm) :: m1,m2,mr
        double precision :: lamb, truelamb
        double precision, allocatable:: tempeval(:)
        double precision, allocatable:: tempeval1(:)
        double complex, dimension (: , :), allocatable ::
           tempmat
        double complex, dimension (: , :), allocatable ::
           tempmat1
        type(dcm) :: tempdm
        complex*16, allocatable:: work(:)
        integer lwork
        double precision, allocatable:: rwork(:)
        complex*16, allocatable:: work1(:)
        integer lwork1
        double precision, allocatable:: rwork1(:)
        complex*16, allocatable:: work2(:)
        integer lwork2
        double precision, allocatable:: rwork2(:)
        integer iinfo
        integer :: aa,bb
        type(dcm) :: dham
        type(dcm) :: rham
        type(dcm) :: ddham
        type(dcm) :: temp_dmat
        type(dcm) :: temp_dmat1
        type(dcm) :: temp_dmat2
        type(dcm) :: temp_dmat3
        type(dcm) :: idd
        type(dcm) :: didd
        type(dcm) :: dnidd
        double complex, dimension (: , :), allocatable :: pr
        double complex, dimension (: , :), allocatable :: prt
        double complex, dimension (: , :), allocatable ::
           temp_tr
        integer :: ntr,gg
        integer :: dd
        type(state) :: psiex
        type(dcm):: denmat
        type(dcm):: denmat_red
        integer :: uu

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 "Parameters.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

!number iterations RG algorithm

ntr=1000
```

```
!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
c       do pos=1,nn-1
c
!tensor product of N-2 identies and
c               !two closepauli_x, the first is
   located in
c               !the position pos of the chain
c               call ten_prod_vec2(pauli_x,mat_f,nn,
   pos,debug)
c
c               ham%elem=ham%elem+lamb*mat_f%elem
c               deallocate(mat_f%elem)

c       end do


!adding dd sites to the hamiltonian
!ham-fixed (dd sites)
dd=1

!allocation hamf
call init_dcm_mat(hamf,2**dd,2**dd,debug)
```

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

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

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


        end do




        c       !computing the sites interactions part of the
            hamiltonian
        c       do pos=1,dd-1
        c
        c               !tensor product of dd-2 identies and
        c               !two close pauli_x, the first is
           located in
        c               !the position pos of the chain
        c               call ten_prod_vec2(pauli_x,mat_f,dd,
           pos,debug)
        c
        c               hamf%elem=hamf%elem+lamb*mat_f%elem
        c               deallocate(mat_f%elem)
        c
        c       end do


        !temporary double matrix
        temp_dmat%nr=2**(nn+dd)
        temp_dmat%nc=2**(nn+dd)

        temp_dmat1%nr=2**(nn+dd)
        temp_dmat1%nc=2**(nn+dd)

        !allocation identity nn
        allocate (idd%elem(2**(nn),2**(nn)),
        $                       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**(nn)
idd%nc=2**(nn)


!initialization identity nn
do ii=1, 2**(nn)

do jj=1, 2**(nn)

if(ii==jj) then

idd%elem(ii,ii)=(1.0,0.0)

else

idd%elem(ii,jj)=(0.0,0.0)

end if

end do

end do

!allocation identity dd
allocate (didd%elem(2**(dd),2**(dd)),
$                         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

didd%nr=2**(dd)
didd%nc=2**(dd)


!initialization identity dd
do ii=1, 2**(dd)
do jj=1, 2**(dd)

if(ii==jj) then

didd%elem(ii,ii)=(1.0,0.0)

else

didd%elem(ii,jj)=(0.0,0.0)

end if

end do

end do

!allocation identity dd+nn
allocate (dnidd%elem(2**(dd+nn),2**(dd+nn)),
$                        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

dnidd%nr=2**(dd+nn)
dnidd%nc=2**(dd+nn)


!initialization identity dd+nn
do ii=1, 2**(dd+nn)

do jj=1, 2**(dd+nn)

if(ii==jj) then

dnidd%elem(ii,ii)=(1.0,0.0)

else

dnidd%elem(ii,jj)=(0.0,0.0)

end if
```

```
        end do

        end do


        !nn interaction among the system (just two sites)
        !nn+dd system
        call ten_prod_vec2(pauli_x, temp_dmat2, nn+dd, nn,
           debug)
        !(nn+dd)+(nn+dd) system
        call ten_prod_vec2(pauli_x, temp_dmat3, 2*(nn+dd),(nn
           +dd), debug)

        !allocation dd+nn hamiltonian
        call init_dcm_mat(dham,2**(dd+nn),2**(dd+nn),debug)
        !allocation dd+nn hamiltonian
        call init_dcm_mat(rham,2**(dd+nn),2**(dd+nn),debug)
        !allocation 2(dd+nn) hamiltonian
        call init_dcm_mat(ddham,2**(2*(dd+nn)),2**(2*(dd+nn))
           ,debug)

        !necessary to allocate work and rwork
        !and initialize lwork
        !for information read relative documentation

        !2*(dd+nn) system
        lwork1= 2*(2**(2*(dd+nn)))

        allocate(work1(lwork1),
        $                        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(rwork1(lwork1),
        $                        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


        !(nn+dd) system
        lwork= 2*(2**(nn+dd))

        allocate(work(lwork),
        $                        stat=my_stat, errmsg=my_msg)
        !error handling allocation
        if(my_stat /= 0) then
        print*, 'Failed␣to␣allocate␣work1␣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␣rwork1␣with␣stat␣=␣'
        $                 , my_stat, '␣and␣msg␣=␣'//trim(
           my_msg)
        end if




        !temporary matrix
        allocate(temp_tr(2**(dd+nn),2**(nn)),
        $                        stat=my_stat, errmsg=my_msg)
        !error handling allocation
        if(my_stat /= 0) then
        print*, 'Failed␣to␣allocate␣temp_tr␣with␣stat␣=␣'
        $                 , my_stat, '␣and␣msg␣=␣'//trim(
           my_msg)
        end if

        !projections

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

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

        !auxialiary matrices for zheev


        allocate(tempeval(2**(dd+nn)))

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

        allocate(tempeval1(2**(2*(dd+nn))))

        allocate(tempmat1(2**(2*(dd+nn)),2**(2*(dd+nn))))

        !allocation exact psi total system

        allocate(psiex%coeff(2*(nn+dd)))


        !CYCLE OVER TRUNCATIONS

        !in order to keep the numbers low, divide per 2 each
           cycle
        c       do ii=1,2**(nn)
        c       do jj=1,2**(nn)
        c               ham%elem(ii,jj)=ham%elem(ii,jj)/2.
        c               temp_dmat2%elem(ii,jj)=temp_dmat2%
           elem(ii,jj)/2.
        c       end do
        c       end do
        c       truelamb=lamb
        c       lamb=lamb/2.

        !BEGINNING CYCLE
```

```
        do gg=1, ntr

        !connecting the two systems: adding dd sites

        !direct product

        !first part
        call ten_prod (ham,didd,temp_dmat, debug)

        dham%elem = temp_dmat%elem

        deallocate(temp_dmat%elem)
        deallocate(temp_dmat%eval)


        !second part

        call ten_prod (idd,hamf,temp_dmat1, debug)

        dham%elem = dham%elem + temp_dmat1%elem

        deallocate(temp_dmat1%elem)
        deallocate(temp_dmat1%eval)

        !nn interaction among the system (just two sites)
        dham%elem = dham%elem + lamb*temp_dmat2%elem


        !DOUBLING THE SYSTEM: nn+dd->2(nn+dd)
        !done with a reflection

        !right part

        !direct product

        !first part
        call ten_prod (didd,ham,temp_dmat, debug)

        rham%elem = temp_dmat%elem

        deallocate(temp_dmat%elem)
        deallocate(temp_dmat%eval)
        !second part

        call ten_prod (hamf,idd,temp_dmat1, debug)

        rham%elem = rham%elem + temp_dmat1%elem

        deallocate(temp_dmat1%elem)
        deallocate(temp_dmat1%eval)

        !nn interaction among the system (just two sites)
        ccc     rham%elem = rham%elem + lamb*temp_dmat2%elem




        !direct product left part - right part

        !first part
        call ten_prod (dham,dnidd,temp_dmat, debug)

        ddham%elem = temp_dmat%elem

        deallocate(temp_dmat%elem)
        deallocate(temp_dmat%eval)


        !second part

        call ten_prod(dnidd,rham,temp_dmat1, debug)

        ddham%elem = ddham%elem + temp_dmat1%elem

        deallocate(temp_dmat1%elem)
        deallocate(temp_dmat1%eval)

        !nn interaction among the system (just two sites)
        ccc     ddham%elem = ddham%elem + lamb*temp_dmat3%
           elem


        !calling the LAPACK subroutine for eigen values
        !of 2*(dd+nn) system

        tempmat1=ddham%elem
        aa=2**(2*(nn+dd))
        bb=2**(2*(nn+dd))



        call zheev( "V", "U", aa , tempmat1 , bb , tempeval1,
            work1,
        $                       lwork1, rwork1, iinfo )
        !"V"->eigenvalues and eigenvectors,
        !stored inside tempeval and tempmat
        !"U"->upper triangle of aa is stored, inside dd%elem
        ! tempeval 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*,  "the␣algorithm␣failed␣to␣converge"
        end if
        end if


        !take only first eigenvector and form a
        !density matrix of 2*(nn+dd) sites


        do ii=1,2**(2*(nn+dd))
```

```
psiex%coeff(ii)=tempmat1(ii,1)

end do



psiex%len_state= 2**(2*(nn+dd))

psiex%sep=.false.
!already written in a large vector

psiex%pur=.true.

!calling the subroutine which computes the density
   matrix
call denmat_pure(psiex, denmat, debug)





!tracing out the right system:
!from dd+nn+1 to 2*(dd+nn) site
allocate(tempdm%elem(2**(2*(nn+dd)),2**(2*(nn+dd))))
tempdm%elem=denmat%elem

do uu= 1, (nn+dd)

call red_denmat(tempdm, denmat_red, 2,
$                       (2*(nn+dd)-(uu-1)), (2*(nn+
   dd)-(uu-1)),debug)

if(uu<(nn+dd)) then

deallocate(tempdm%elem)

allocate(tempdm%elem(2**(2*(nn+dd)-uu),2**(2*(nn+dd)-
   uu)))
tempdm%elem=denmat_red%elem

deallocate(denmat_red%elem)
deallocate(denmat_red%eval)

else
```

```
        deallocate(tempdm%elem)

        end if


        end do

c       do ii=1, 2**((nn+dd))
c       do jj=1, 2**((nn+dd))
c               print*, denmat_red%elem(jj,ii)
c       end do
c       print*, "denmat␣redcolumn", ii
c       end do



        !calling the LAPACK subroutine for eigen values
        !of (dd+nn) system

        tempmat=denmat_red%elem

        aa=2**((nn+dd))
        bb=2**((nn+dd))



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

        !"V"->eigenvalues and eigenvectors,
        !stored inside tempeval and tempmat
        !"U"->upper triangle of aa is stored, inside dd%elem
        ! tempeval 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


        !take only first 2**(nn) eigenvectors and form a
           projector
        !i.e. the first nn columns of the temporary matrix
           tempmat

        !projector

        do ii=1, 2**(nn+dd)

        do jj=1, 2**(nn)

        pr(ii,jj)=tempmat(ii,jj)

        end do

        end do



        !transpose projector

        do ii=1, 2**(nn)

        do jj=1, 2**(dd+nn)

        prt(ii,jj)=conjg(pr(jj,ii))

        end do
        end do


        !computation truncated matrix


        call rowrowmatmul(2**(dd+nn),2**(dd+nn),2**(nn),dham%
           elem,
        $                          pr,temp_tr)




        call rowrowmatmul(2**(nn),2**(dd+nn),2**(nn),prt,
           temp_tr,
        $                          ham%elem)


        !in order to obtain the gs of the previous cycle


        print*, gg, tempeval1(1)/(4+(gg-1)*2)

        !deallocations

        deallocate(denmat%elem)
        deallocate(denmat%eval)
        deallocate(denmat_red%elem)
        deallocate(denmat_red%eval)

        end do

        !writing on file gs

        open(unit = 80, file = "DMRG_1gs.txt", 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
```

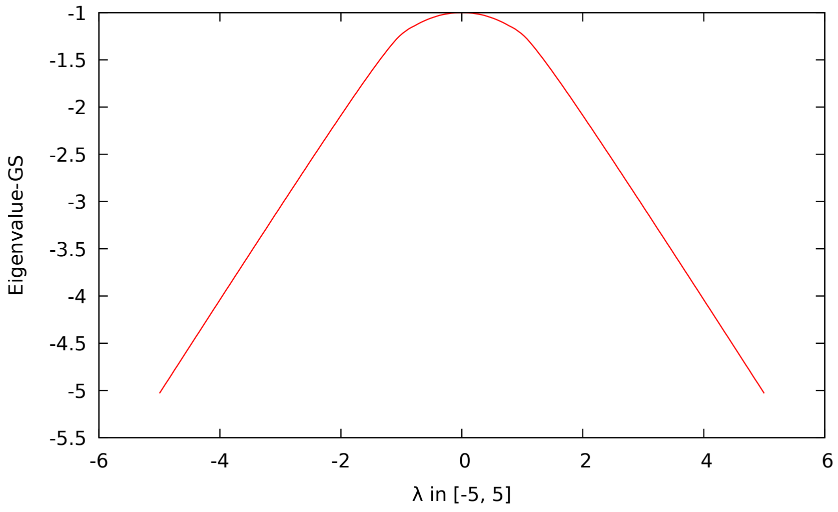
```
        write(80,*) lamb, tempeval1(1)/(4+(gg-1)*2)


        end program week11
```

Results

Real-space RG algorithm

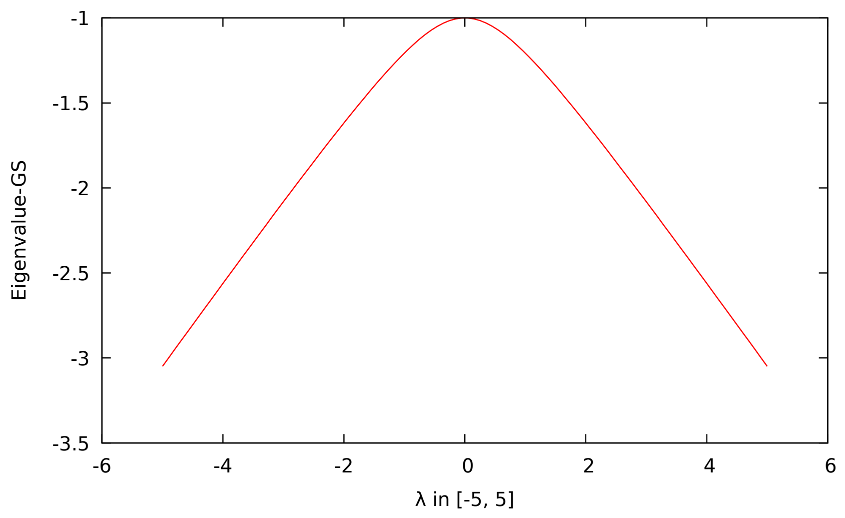
This the ground state energy for $\lambda \in [-5, 5]$ with an initial size 2 and number of cycles 100. Hence the number of spins described by H_N^{tr} is 2^{101}, well above the thermodynamic limit.



Ground state energy for $\lambda \in [-5, 5]$ with an initial size 2 and number of cycles 100. Real-space RG algorithm.

Infinite DMRG algorithm

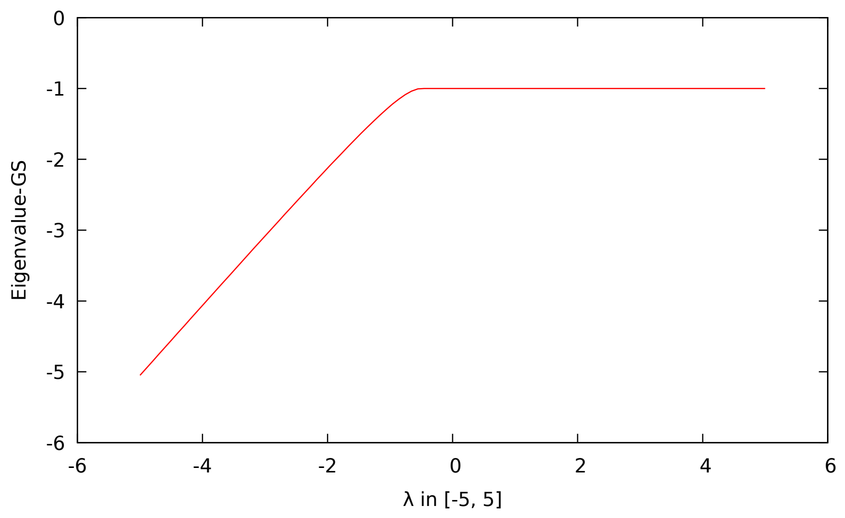
This the ground state energy for $\lambda \in [-5, 5]$ with a initial sizes $m = 1$, $d = 1$ and number of cycles 1000. Hence the number of spins described by $H_{2(m+d)}$ is 2002, under the thermodynamic limit. Since the code present some problems (for d greater than 1 bad memory allocations are given), the results are incorrect. We show them only to point the fact that at least the qualitative shape is correct.



Ground state energy for $\lambda \in [-5, 5]$ with a initial sizes $m = 1$, $d = 1$ and number of cycles 1000. Infinite DMRG algorithm. Incorrect results due to something wrong with the code.

Self-Evaluation

Comparison with Mean Field solution



Ground state energy for transverse field Ising model for $\lambda \in [-5, 5]$, translational invariance ansatz.

As discussed in ex.10, the Mean Field solution with translational invarince ansatz fails for $\lambda \geq -\frac{1}{2}$: it cannot capture the fact that the ground state is composed by alternating spins, not equal. For lower λ the behaviour is correct, although the actual values of energies are a little different, as we can notice looking at the above figures. This is reasonable: the RG algorithms take into account not only the ground state of the sub-systems of the total system (as done in MF solution), but also a certain number of excited states of those. Hence it is a more precise approximation. This is

Comparison between the two RG algorithms

We can notice that, since real space RG multiplys the number of spins per 2 at each cycle, the infinite DMRG adds 2 spins each cycle. As a consequence, it describes a lower number of spins for the same number of cycles.
We cannot make comparisons with the results shown above, since the DMRG results are incorrect due to some errors within the code (bad memory allocations for m greater than 1). We can say at least that the qualitative form is similar. So we can infer that in this case both RG and DMRG work (although in some cases real space RG fails, while Infinte DMRG not fails).

What can be done next:

- Study the behaviour of the algorithms for different values of the size of the initial system and for different number of cycles, taking into account the computational time.
- Understand the errors within the code for DMRG algorithm.

The main goals achieved:

- Apply real space RG algorithm (and a bit of infinite DMRG algorithm) on an actual physics problem.