Density Matrices

Alberto Chimenti

Exercise n.8
Information Theory and Computation - 2019

Abstract

We proceed to explain the implementation of a Fortran module used to store pure states wavefunctions and compute the related density matrix together with a reduction algorithm for bipartite systems. In the last part results are shown for two qubits case.

1 Theory

Wavefunction of a pure state

Considering a system composed by N subsystems all with the same Hilbert space dimension, the general N-body pure state wavefunction $|\psi\rangle \in \mathcal{H}^{D^N}$ can be written as:

$$|\psi\rangle = \sum_{\alpha_1,\dots,\alpha_N=1}^{D} K_{\alpha_1,\dots,\alpha_N} |\psi_{\alpha_1}\rangle \otimes \cdots \otimes |\psi_{\alpha_1}\rangle$$

In simple words it is the sum of all the combination of the different possible states of each subsystem. We note that the coefficients $K_{\alpha_1,\dots,\alpha_N}$ are represented from a tensor which is defined by D^N numbers. The state is called separable if it can be rewritten in the following form:

$$|\psi\rangle = \sum_{\alpha_1} K_{\alpha_1} |\psi_{\alpha_1}\rangle \otimes \cdots \otimes \sum_{\alpha_N} K_{\alpha_N} |\psi_{\alpha_N}\rangle$$

Therefore in such a case one can store the full system wavefunction using only $N * D$ coefficients.

Density matrix

The density matrix is a hermitian unitary operator which is defined as:

$$\rho = |\psi\rangle \langle\psi|$$

It is a projector by definition and its trace is always equal to one since it corresponds to the sum of the probabilities of finding the full system in each possible configuration.

Reduced Density matrix

The reduced density matrix of a system relative to the *k-th* subsystem is computed as:

$$\rho_k = Tr_1 \dots Tr_{k-1} Tr_{k+1} \dots Tr_N \rho$$

It describes the state of one specific subsystem considering its interaction with the others.

2 Code Development

2.1 TYPE

The first thing that has been done was to define a specific type for the wavefunction

```
TYPE qstate
    integer :: n_subs ! number of subsystems (N)
    integer :: dim ! subsystem dimension (d)
    logical :: sep ! boolean for separable state
    double complex, dimension(:), allocatable :: coeff ! values stored N*d
    integer :: len ! lenght of the statevector
END TYPE qstate
```

This is useful to store the specifics of the full quantum system such as the number of subsystem it is composed by and the dimension of the singular subsystems. The integer **len** will be useful in differentiating a separable state from a general one.

2.2 Random initialization

Successively, a function, called **init_qstate**, for random initialization of such a wavefunction was implemented. It creates the type object starting from the number of subsystems, their dimension and a boolean quantity which tells whether to store a separable state or a general one. A separable state is created by generating $N * D$ random entries corresponding to the coefficients. This is done subsystem by subsystem and each of them is normalized to $\frac{1}{\sqrt{N}}$ so that the squared modulus of the full wavefunction is normalized to one.
An optional **debug** boolean input controls, if equal to **.TRUE.**, some optional printouts about normalization checking for the generated wavefunction.

2.3 Density matrix

The function **init_density** creates the density matrix, starting from a given state vector. Here the crucial point is that given the $N * D$ elements vector describing the separable state, one should first express it in general form before computing the density matrix. Such a procedure is achieved by computing the tensor products over all of the single wavefunctions. Another interesting way to transform the vector is to map each element of the $N * D$ vector in a D-basis written one.

Therefore the *i-th* element of the D^N vector can be written as:

$$K^{new}[i] = \prod_{j=1}^{N} K^{old}[\alpha_j + (j-1) * D]$$

where the indexes α_j have been computed by means of the change of basis. The code chunk related to the computation is reported in the following:

```
[...]
    sum = 0. ! needed to normalize
    psi_out%coeff=cmplx(1., 0.)
    do ii=1, psi_out%len !cycle over all the elements of the full vector
        kk = ii - 1 !due to fortran indexing
        do jj=1, psi_out%n_subs !cycle over the subsystems
            qq = psi_out%n_subs - jj
            coeff(qq+1) = int((kk-mod(kk,psi_out%dim**qq))/psi_out%dim**qq)+1
            !+1 to readjust the indexing
            kk = kk - (coeff(qq+1)-1)*psi_out%dim**qq
        end do
        do jj=1, psi_out%n_subs
            psi_out%coeff(ii) = psi_out%coeff(ii) * temp(coeff(jj)+(jj-1)*psi_out%dim)
        end do
        sum = sum + abs(psi_out%coeff(ii))**2
    end do
```

```
17
18      ! normalization
19      psi_out%coeff = psi_out%coeff/sqrt(sum)
20  [...]
```

Therefore, after the conversion the density matrix is computed as:

```
1  [...]
2      do ii=1, psi%len
3          do jj=1, psi%len
4              density%elem(ii,jj) = psi%coeff(ii)*dconjg(psi%coeff(jj))
5          end do
6      end do
7  [...]
```

Also here the **debug** logical input controls a check over: the normalization of the transformed state vector, hermitianity of the density matrix and that its trace equals 1.

2.4 Reduced Density matrix

The computation of the reduced density matrix was implemented for a bipartite system in the function **reduce_density_bipartite**. It asks for a mandatory integer input which has to be set to 1 or 2 in order to select the system to keep. The computation is reported in the following code:

```
1  [...]
2      if (sys_to_keep==1) then
3          do ii=1, dim
4              do jj=1, dim
5                  do kk=1, dim
6                      rdensity%elem(ii,jj) = rdensity%elem(ii,jj) &
7                          + density%elem((kk-1)*dim + ii,(kk-1)*dim + jj)
8                  end do
9              end do
10         end do
11
12     elseif (sys_to_keep==2) then
13         do ii=1, dim
14             do jj=1, dim
15                 do kk=1, dim
16                     rdensity%elem(ii,jj) = rdensity%elem(ii,jj) &
17                         + density%elem((ii-1)*dim + kk,(jj-1)*dim + kk)
18                 end do
19             end do
20         end do
21
22     else
23         print*, "Wrong sys_to_keep input.... exiting"
24         stop
25     end if
26 [...]
```

3 Results

In the following we report the results for the test of the algotithms for a system of 2 qubits

$$N = 2 \qquad D = 2$$

3.1 General

```
General State vector
            (0.65317173625180458,0.43088601075978716)
            (0.18294011001598698,0.33520769184288812)
            (0.42809748195746306,2.74354053966014109E-002)
            (2.43043811799029119E-002,0.23929435784800093)
```

```
 Density_matrix_General_Pure

 [dim],              4           4
 [trace],                (0.99999999999999978,0.0000000000000000)
[elem],           column[  1],          column[  2],          column[  3],          column[  4],
row[  1],          0.612+.000i,          0.264-.140i,          0.291+.167i,          0.119-.146i,
row[  2],          0.264+.140i,          0.146+.000i,          0.088+.138i,          0.085-.036i,
row[  3],          0.291-.167i,          0.088-.138i,          0.184+.000i,          0.017-.102i,
row[  4],          0.119+.146i,          0.085+.036i,          0.017+.102i,          0.058+.000i,

 Reduced_density_matrix_General_Pure

 [dim],              2           2
 [trace],                (0.99999999999999978,0.0000000000000000)
[elem],           column[  1],          column[  2],
row[  1],          0.758+.000i,          0.376+.131i,
row[  2],          0.376-.131i,          0.242+.000i,
```

3.2 Separable

```
Separable State vector
            (0.43592532255244248,0.47533726555826322)
            (0.26813116649484392,0.11013308703900777)
            (0.39519202981202761,0.11557597021440177)
            (0.42633379002247040,0.38562280049557346)
```

```
 Density_matrix_Separable_Pure

 [dim],              4           4
 [trace],                (1.0000000000000000,0.0000000000000000)
[elem],           column[  1],          column[  2],          column[  3],          column[  4],
row[  1],          0.282+.000i,          0.115+.054i,          0.354-.172i,          0.177-.002i,
row[  2],          0.115-.054i,          0.057+.000i,          0.111-.138i,          0.072-.035i,
row[  3],          0.354+.172i,          0.111+.138i,          0.550+.000i,          0.224+.105i,
row[  4],          0.177+.002i,          0.072+.035i,          0.224-.105i,          0.111+.000i,

 Reduced_density_matrix_Separable_Pure

 [dim],              2           2
 [trace],                (1.0000000000000000,0.0000000000000000)
[elem],           column[  1],          column[  2],
row[  1],          0.339+.000i,          0.426-.206i,
row[  2],          0.426+.206i,          0.661+.000i,
```

4 Self-evaluation

It is worth noting that even if the memory needed to store a pure separable state is much less than the one needed for a general pure state, the computational handling procedure is much more complex. In simple words, is seems to be easy to store but not to work with. All of the presented algorithms were implemented for a pure state and the generalization to mixed states would need much effort.