UNIVERSITY OF PADUA
PHYSICS AND ASTRONOMY DEPARTMENT

Homework #8
Density Matrices

Student name: *Beatrice Segalini* – 1234430

Course: *Quantum Information and Computing 2020* – Professor: *S. Montangero*
Due date: *December 1st, 2020*

Abstract

A general quantum system formed by N subsystems (spins, atoms, particles etc..) is considered. Each subsystem is described by its wave function $\psi_i \in \mathcal{H}^d$, where $\mathcal{H}^d$ is a $d-$dimensional Hilbert space. In this report, numerical techniques to write the total wave-function of such system with different properties (non-interacting systems, separable states...) are studied. Furthermore, the density matrix for a $N = 2$ quantum system and some partial tracing operations are analysed and tested.

Theory

Let $\mathcal{S}$ be a a quantum system, living in the Hilbert space $\mathcal{H}$, composed of N elementary subsystems, each of them living in the Hilbert spaces $\mathcal{H}_i = \mathbb{C}^d$ with $i = 1, 2 \dots N$.

Wave-function. Being $|\alpha_i\rangle$ with $i = 0, \dots, (d-1)$ an orthonormal basis of a generic $\mathcal{H}_i$, a generic wave-function in $\mathcal{H}$ can be written as:

$$|\Psi\rangle = \sum_{\alpha_1 \dots \alpha_N} C_{\alpha_1 \dots \alpha_N} |\alpha_1 \dots \alpha_N\rangle \tag{1}$$

where $C_{\alpha_1 \dots \alpha_N}$ are d^N complex coefficients.

However, this expression can be simplified with the *mean field approximation*: in this case, it is possible to neglect any kind of correlation between the subsystems, writing the system wave-function as the product of the independent wave-functions of the subsystems:

$$|\Psi\rangle = \bigotimes_{i=1}^{N} \psi_i \tag{2}$$

As a consequence, the complete wave-function has only $d \cdot N$ coefficients and can be represented by a $d \times N$ matrix which can be written by columns:

$$\Psi = (\psi_1 \dots \psi_N) \tag{3}$$

Density matrix. Given a generic pure state in $N = 2$, the density matrix ρ is defined as follows:

$$\rho = |\Psi\rangle\langle\Psi| = \sum C_{\alpha_1,\alpha_2} C^*_{\alpha'_1,\alpha'_2} |\alpha_1\alpha_2\rangle\langle\alpha'_1\alpha'_2| \tag{4}$$

The density matrix can be *reduced* considering only the k^{th} subsystem by computing the *partial trace* with the following expressions:

$$\begin{gathered} \rho_k = \mathrm{Tr}_1 \dots \mathrm{Tr}_{k-1} \, \mathrm{Tr}_{k+1} \dots \mathrm{Tr}_N \, \rho \\ \mathrm{Tr}_j \, \rho = \sum_{j=1}^{d} \langle\alpha_j|\rho|\alpha_j\rangle \end{gathered} \tag{5}$$

That, in the case of $N = 2$, is simply:

$$\rho_1 = \mathrm{Tr}_2(\rho) = \mathrm{Tr}_2\left(\sum \rho^{\alpha_1',\alpha_2'}_{\alpha_1,\alpha_2} |\alpha_1\alpha_2\rangle\langle\alpha_1'\alpha_2'|\right) \tag{6}$$

Code development

The type `qstate`. In order to properly describe the total wave-function of the quantum system considered, the `qstate` type is defined (Listing 1): a `qstate` variable will contain information about the size of d of the single Hilbert spaces, the number N of subsystems, a logical that if `.TRUE.` means that the system is separable, a long integer to contain information about the memory space occupied and finally a double complex vector for the wave-function coefficients.

Listing 1: User defined type `qstate`.

```
type qstate
        integer    :: d
        integer    :: n
        logical    :: sep
        integer*8  :: memo
        double complex, dimension(:), allocatable :: psi
end type
```

Allocating the systems, generating random states. The next step is to initialise the system accordingly to its properties. To do so, two cases are considered: the $N-$body non interacting *separable pure state* and the more general *pure* wave-function. The main difference between the two is the value of the `sep` attribute (`.TRUE.` for the first one, `.FALSE.` for the second one) and the number of coefficients stored in the attribute `psi`. In fact, as stated in the previous section, there are $N \cdot d$ coefficients for the separable case, while for the more general one they are d^N. The functions are reported in Listing 2.

Listing 2: Allocation functions for both cases considered.

```
function allocate_pure_sep(dimH, ns)result(state)
        integer, intent(in) :: dimH, ns
        type(qstate)        :: state

        state%d   = dimH
        state%n   = ns
        state%sep =.TRUE.
        allocate(state%psi(ns*dimH))
        state%memo = sizeof(state%psi)
end function

function allocate_pure(dimH, ns)result(state)
        integer, intent(in) :: dimH, ns
        type(qstate)        :: state

        state%d   = dimH
        state%n   = ns
        state%sep =.FALSE.
        allocate(state%psi(dimH**ns))
        state%memo = sizeof(state%psi)
end function
```

To test the program, N random normalized elements have been generated with the following code (Listing 3).

Listing 3: Generation of random states for testing the code.

```
do ii = 1, N
        norm = 0
        do jj = 1, d
                call random_number(re_psi)
                call random_number(im_psi)
                psi_elem(ii, jj) = dcmplx(re_psi, im_psi)
                norm = norm + ABS(psi_elem(ii, jj))**2
        end do
        do jj = 1, d
                psi_elem(ii, jj) = psi_elem(ii, jj)/(sqrt(norm))
        end do
end do
```

Converting from array to tensor notation. The coefficient $C_{\alpha_1 \dots \alpha_N}$ is a tensor of rank N with d^N elements, in the general case of a non-separable state. As a consequence, one has to consider every possible combination of the vector $(\alpha_1 \dots \alpha_N)$.

A very practical way to handle numerically a generic number of indexes is to convert the *matricial* form to a more *tensor*-friendly notation. This is done through two functions found in Listing 4, that "linearise" the structure of the system.

The key idea is to pass from an array `psi_elem` with 2 dimensions to a vector representing the whole system that has only a single index, but in a way that does not provoke any loss of information or redundant calculations.

Listing 4: Function to convert from *matrix*-like to *tensor*-like indexes.

```
function to_mat(idx_t, d) result(mm)

integer, dimension(:), intent(IN) :: idx_t
integer, intent(IN)               :: d
integer*8                         :: mm

mm = 1

do aa = 1, size(idx_t)
        mm = mm + idx_t(aa)*( d**(aa - 1) )
end do
end function to_mat

function to_tensor(mm, N, d) result(idx_t)

integer*8              :: mm
integer, intent(IN)    :: N, d
integer, dimension(N) :: idx_t

do aa = 1, N
        idx_t(aa) = modulo( ( (mm - 1)/(d)**(aa-1) ), d) + 1
end do
end function to_tensor
```

Consequently, it is now possible to extract, given the correct indexation, the value of the coefficient $C_{\alpha_1 \dots \alpha_N}$ as required, this is performed by the following piece of code (Listing 5):

Listing 5: Function to compute the value of the coefficient C.

```
function getElement(psi_elem, idx, N)result(element)

integer, dimension(:), intent(IN)           :: idx
integer, intent(IN)                         :: N
double complex, dimension(:,:), intent(IN)  :: psi_elem
double complex                              :: element

element = dcmplx(1, 0)

do aa = 1, N
        element = element * psi_elem(aa, idx(aa))
end do
end function getElement
```

Combining everything, the tensor fo the whole system is computed with a simple loop (Listing 6).

Listing 6: Code to describe the total system tensor.

```
do ll = 1, d**N
        psi_sys%psi(ll) = getElement(psi_elem, to_tensor(ll, N, d), N)
end do
```

Density matrix. For handling with density matrices, the number of subsystems to is set equal to $N = 2$, and similarly the size of Hilbert spaces is considered to be fixed at $d = 2$.

Given the definition of density matrix displayed in 4, one can easily infer that the density matrix ρ associated to a couple of subsystems composed of $d-$dimensional spins/bits is a $d^2 \times d^2$ matrix.

Broadly speaking, ρ can be calculated from the total wave-function with the lines of code in Listing 7.

Listing 7: Computation of the density matrix.

```
do ll = 1, d**N
        do hh = 1, d**N
                rho(ll, hh)  = psi_sys%psi(ll)*CONJG(psi_sys%psi(hh))
        end do
end do
```

Partial trace and ρ_1. The following step in this analysis is to derive the the density matrix associated to only one of the two systems considered, i.e. ρ_1, defined in the Theory section in Equation 6.

To perform this task, the computation of the partial trace with respect to the other subsystem (see Equation 5) is required: the code in Listing 8 achieves this goal.

Listing 8: Computation of the partial trace and of density matrix ρ_1.

```
do ii = 1, d**(N-1)
  do jj = 1, d**(N-1)
        rho_1(ii, jj) = 0
        do kk = 1, d
          rho_1(ii, jj) = rho_1(ii, jj) + &
```

```
                rho(to_mat([ii-1, kk-1], d), to_mat([jj-1, kk-1], d))
        end do
  end do
end do
```

In the code the appropriate index conversions have been adjusted not to obtain misleading results.

Similarly to before, the code is tested with random normalised states generated with code in Listing 3.

Results

Efficiency. By using the built-in function `sizeof`, it is possible to verify that indeed the general pure state case is more computationally heavy: in fact, the number of coefficients to be stored in the memory is d^N while for the separable case there are only $N \cdot d$ coefficients.

Consequently, computation is significantly faster for the separable case and systems with much bigger dimensions can be studied. Considering the maximum computational power of the machine on which the program was tested and fixing $d = 2$, it can be calculated a maximum limit for N equal to approximately 29 in the general case, while for the separable case the limits are pushed over $3 \cdot 10^8$.

Testing on the density matrices. To check the correctness of the code, different quantities have been computed: $\mathrm{Tr}(\rho)$, $\mathrm{Tr}(\rho^2)$, $\mathrm{Tr}(\rho_1)$, $\mathrm{Tr}(\rho_1^2)$.

All these values should be, according to theory, equal to 1, and the program returns correct results with very little numerical fluctuations.

Self evaluation

All the implementation analysed produced results consistent with the theory, so one can conclude that the program worked properly. However, some improvements to this study can be made.

For example, a more deep analysis of the computational time as a function of the size af the system could be performed, to quantitatively express how the performances are impacted by the separability of the states.

Furthermore, testing on different types of matrices (particular cases, non-randomly generated, different sizes N and dimensions d) has not been included in this report, but could definitely be implemented with little effort on the provided code.