Exercise 8: *Numerical computation of density matrices and partial traces in* FORTRAN

Iriarte Delfina (1231682)

Quantum Information

December 1, 2020

Abstract

In this report, a **generic** and a **separable** pure states are built using FORTRAN 95 and the available size memory for allocating such wave functions are studied. Moreover, the **density matrices** for both cases are computed. In particular, considering the the case of qubits, the **reduce density matrix** for left and right particles ar performed.

1 Introduction

The system wave function $|\Psi\rangle$ lives in the N -body Hilbert space $\mathcal{H}_N$ formed by the tensor product of all the local ones $\mathcal{H}_N = \mathcal{H}_1 \otimes \mathcal{H}_1 \otimes \cdots \otimes \mathcal{H}_1$. Considering a system composed of $N-$sub systems with each sub-system corresponding of a wave function $|\Psi_i\rangle$, then the wave function of a **general pure state** can be describe as $|\Psi\rangle \in \mathcal{H}^{D^N}$:

$$|\Psi\rangle = \sum_{\alpha} C_{\alpha_1 \alpha_2 \cdots \alpha_N} |\alpha_1\rangle \otimes |\alpha_2\rangle \otimes \cdots \otimes |\alpha_N\rangle \tag{1}$$

where each index α_i labels each single particle basis $\{|\alpha_i\rangle : \alpha_i = 0, \cdots, d-1\}$.

There are some special kinds of states that can also be solved called **separable states**, in which the wave function, given in Equation 1, can be describe as:

$$|\Psi\rangle = \sum_{\alpha_1,\cdots\alpha_N} C_{\alpha_1} \cdots C_{\alpha_N} |\alpha_i\rangle \otimes \ldots \otimes |\alpha_N\rangle = \sum_{\alpha_1} C_{\alpha_1}|\alpha_i\rangle \otimes \cdots \otimes \sum_{\alpha_N} C_{\alpha_N}|\alpha_N\rangle$$

Undoubtedly, in the separable case we have **simplified** from an exponential to a linear problem in the number of lattice sites N. In fact, the number of degrees of freedom is d^N in the original problem, $N \times d$ in this scenario.

Given a pure state the **density matrix** ρ is defined as a complex positive semi-definite ($\rho > 0$), **Hermitian** matrix with **unit trace** ($\mathrm{Tr}(\rho) = 1$). The density matrix of a pure state $|\Psi_j\rangle$ is given by the outer product of the state vector with itself:

$$\rho = |\Psi\rangle\langle\Psi| = \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} a^* & b^* \end{bmatrix} \tag{2}$$

where we have used the general two-dimensional state vector of $|\Psi_j\rangle$.

Given the density matrix, let's consider a bipartite system Ψ_{ab}. All the probabilities of measurements on the system a can be computed using the **(reduced) density matrix** obtained by taking the **partial trace** over system b:

$$\rho^a = \mathrm{Tr}_b(|\Psi_{ab}\rangle\langle\Psi_{ab}|) \tag{3}$$

In particular, for a system of two qubit the partial trace is calculated according to:

$$\rho_a = \mathrm{Tr}_b \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \rho_{14} \\ \rho_{21} & \rho_{22} & \rho_{23} & \rho_{24} \\ \rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} \\ \rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} \end{pmatrix} = \begin{pmatrix} \rho_{11} + \rho_{22} & \rho_{13} + \rho_{24} \\ \rho_{31} + \rho_{42} & \rho_{33} + \rho_{44} \end{pmatrix} \tag{4}$$

Finally, we mention that the partial trace Tr_a over the subsystem a is defined in the obvious way and can be calculated similarly to Tr_b:

$$\rho_b = \mathrm{Tr}_a \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \rho_{14} \\ \rho_{21} & \rho_{22} & \rho_{23} & \rho_{24} \\ \rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} \\ \rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} \end{pmatrix} = \begin{pmatrix} \rho_{11} + \rho_{33} & \rho_{12} + \rho_{34} \\ \rho_{21} + \rho_{43} & \rho_{22} + \rho_{44} \end{pmatrix} \tag{5}$$

In this project, a pure generic and separable state of N-body interacting are generated. The memory storage of such arrays is study in order to determine the possible number of particles and dimension that can be simulated in FORTRAN. Afterwards, the density matrix is computed by simply taking the external product between the states as in equation 2. Finally, fixing the dimension and the number of particles equal to two the reduce matrix of a generic state is computed.

2 Code development

The generation of the pure states was done in the subroutines presented in Listing 1. The SUBROUTINE takes as input the number of particles `n_particles`, the dimension of the vector d and a complex allocatable array `pure_vector`. The implementation is done by allocating a complex array of size d^N for the generic case. Furthermore, they were build using uniform random numbers for the real and imaginary parts. The normalize wave function was produce by dividing itself with their correspondent norm. The following checks are implemented in this subroutines:

- dimension and number of particles must be positive and greater than zero
- successful allocation
- normalization of the wave function after implementing

```
allocate(pure_vector(d**n_particles), rand1(d**n_particles), rand2(d**n_particles))
!Generating random vectors
call random_number(rand1); call random_number(rand2)
!Computing norm of the vector
norm = sqrt(sum(abs(rand1**2+rand2**2)))
!Generating pure random vector
pure_vector = rand1/norm + rand2*complex(0,1/norm)
```

Listing 1. Main code of SUBROUTINE to generate generic states

For the separable case the dimension of the allocatable array is of $d \times N$ and the implementation is shown in Listing 3. Likewise the generic case, the SUBROUTINE takes as input the number of particles `n_particles`, the dimension of the vector d and a complex allocatable array; and the generation of the wave function is by producing random numbers. In this case, the first d coefficients of the vector correspond to the first particle, the following d coefficients to the second particle and so on. Hence each d coefficients represents their correspondent particle and each of them are normalize to one. Similarly as before, same checks as for the previous case are performed.

```fortran
allocate(pure_vector(d*n_particles))
!Generate separable states
do ii = 1, n_particles
do jj = 1, d
    call random_number(rand1)
    if (jj ==1) then
        pure_vector(jj+(ii-1)*d) = cmplx(rand1, 0._pr)
    else
        call random_number(rand2)
        pure_vector(jj+(ii-1)*d) = cmplx(rand1, rand2)
    end if
enddo
!Normalization
summ = 0._pr
do kk = 1, d
    tempp= abs(pure_vector(kk+(ii-1)*d))**2
    summ = summ + tempp
enddo
summ = sqrt(summ)
!Normalizing Vector
do kk = 1, d
    pure_vector(kk+(ii-1)*d) = pure_vector(kk+(ii-1)*d)/(summ)
enddo
```

Listing 2. Main code of SUBROUTINE to generate separable states

In order to study the available memory size of the allocatable array, a LOGICAL VARIABLE called `size_memory` is introduced. If the logical variable is `.True.`, then we generate pure states by implementing Listing 1 for different values of dimension and number of particles. When the program is not able to allocate more memory, it breaks, and the last value printed in the bash is the maximum size of an array.

Afterwards, the density matrix of the general case for $N = 2$ and $d = 2$ is computed. This is done by simply performing a matrix multiplication between the system wave function and the adjoint wave function as in Equation 2:

```fortran
  do i = 1, ham_dim ** num_part
      do j = 1, ham_dim ** num_part
           density_matrix(i, j) =  random_states(i) * conjg(random_states(j))
      enddo
  enddo
```

Listing 3. Density matrix generation for the pure state

However, for the separable case some additional computation are needed. First of all, since the vector has size of $d \times n$, a conversion of size d^n is done. This is shown in Listing 4. The SUBROUTINE takes as input the separable array and returns a vector of dimension d^n. This was carried out by using the fact that every "coefficient index" in the system wave function can be coded as a number written using N digits in base d. After performing the proper transformation of the vector, the density matrix is computed the same way as for the generic case. Both matrices were check to be Hermitian and to have trace equal to one. Furthermore, the square density matrix was computed using the function `matmul` provided by FORTRAN.

```fortran
 do ii= 1, dd**nn
    kk=ii-1
    do jj= 1, nn
        cc=nn-jj
        temp_vect(cc+1)=int((kk-mod(kk,dd**cc))/dd**cc +0.1) +1
        kk=kk-(temp_vect(cc+1)-1)*(dd**cc)
    end do
    temp=(1.,0.)
    do jj= 1, nn
        temp= temp*psi(temp_vect(jj)+(jj-1)*dd )
    end do
    psi_large(ii)=temp
end do
```

Listing 4. SUBROUTINE for converting a vector of size $d \times n$ to d^n

Finally, Listing 5 shows the the computation of the reduce matrix for the generic case for N=2. The FUNCTION takes as input the original density matrix, dimension, number of particles and a new variable called `particle` which allows to performed the partial trace of the left or right particle by choosing it values to 1 or 2.

```fortran
do ii=0, max_right_ind-1
    do jj=0, max_left_ind-1
        do kk=0, max_right_ind-1
            do ll =0, max_left_ind-1
                xxy = jj*hham_dim**(particle-1)+ii+1
                yyx = ll*hham_dim**(particle-1)+kk+1
                temp = complex(0.0,0.0)
                do ind=0,hham_dim-1
                    xx =(jj*hham_dim+ind)*hham_dim**(particle-1)+ii+1
                    yy =(ll*hham_dim+ind)*hham_dim**(particle-1)+kk+1
                    temp = temp + rho(xx,yy)
                end do
                rho_reduce(xxy,yyx)=temp
            end do
        end do
    end do
end do
```

Listing 5. Partial trace

3 Results

With respect to the size of the allocatable array,the maximum size is 2^{26} which correspond to a required memory allocation of $2^{26} \times 16 \sim 1GB$ for a `complex (16)`. Notice that for $2^{27} \times 16 \sim 2.1GB$ which, unfortunately, is higher than the available memory in the virtual machine. The density matrix obtained for a generic case are:

data.txt

```
#                              Matrix values

  0.35  +0.00 i   0.24  -0.11 i   0.35  -0.02 i   0.13  -0.13 i
  0.24  +0.11 i   0.20  +0.00 i   0.25  +0.10 i   0.13  -0.05 i
  0.35  +0.02 i   0.25  -0.10 i   0.36  +0.00 i   0.14  -0.12 i
  0.13  +0.13 i   0.13  +0.05 i   0.14  +0.12 i   0.09  +0.00 i
```

```
Number of rows:           4
Number of cols:           4
Trace:              1.0000000000000000,0.0000000000000000
```

As expected, the matrix is Hermitian with unit trace. In fact it it was found that the square of the density matrix is equal to the density matrix itself. For the separable case it was found:

```
data.txt
#                          Matrix values

  0.17  +0.00 i   0.19  +0.01 i   0.16  +0.14 i   0.17  +0.17 i
  0.19  -0.01 i   0.22  +0.00 i   0.19  +0.15 i   0.21  +0.18 i
  0.16  -0.14 i   0.19  -0.15 i   0.27  +0.00 i   0.30  +0.01 i
  0.17  -0.17 i   0.21  -0.18 i   0.30  -0.01 i   0.34  +0.00 i
 Number of rows:           4
 Number of cols:           4
 Trace:              1.0000000000000000,0.0000000000000000
```

which again the density matrix is Hermitian with unitary trace as expected. The reduced density matrix obtained for the generic matrix (left case) are:

```
data.txt
#                          Matrix values

  0.55  +0.00 i   0.48  -0.06 i
  0.48  +0.06 i   0.45  +0.00 i
 Number of rows:           2
 Number of cols:           2
 Trace:              1.0000000000000000,0.0000000000000000
```

where it can be seen that Equation 4 holds for this case. And for the right particle the results are:

```
data.txt
#                          Matrix values

  0.71  +0.00 i   0.37  -0.24 i
  0.37  +0.24 i   0.29  +0.00 i
 Number of rows:           2
 Number of cols:           2
 Trace:              1.0000000000000000,0.0000000000000000
```

which again behaves as expected following Equation 5.

4 Conclusion

In conclusion, the density matrices of a pure and separable state where successfully implemented in FORTRAN. One of the main drawbacks found is the computational cost required in storing a generic state which becomes highly demand. Moreover, this can be also extended for the mixed states but with some efforts. The reduced density matrices found for the case of qubits were behaving as expected.