Density Matrices

Davide Maniscalco

December 10, 2019

Abstract

In this paper, a general fortran code for handling density matrices is described. Further analysis are made for the case of two subsystems.

SUGGESTED COMPILATION:

- Code: `gfortran Maniscalco-MATRIX.f90 Ex8-Maniscalco-CODE.f90 -o Ex8-Maniscalco.out`
- Python script for graphics: `python3 Ex8-Maniscalco-SCRIPT.py`

1 Theory

Let's consider a quantum system Ψ formed by N subsystems each described by its wave function $\psi \in \mathcal{H} = \mathcal{C}^d$. The general wave function of the system in a pure state is the following:

$$|\Psi\rangle = \sum_{\alpha_1,\dots,\alpha_N} C_{\alpha_1,\dots,\alpha_N} |\psi_{\alpha_1}\rangle \otimes \dots \otimes |\psi_{\alpha_N}\rangle$$

with:

$$\alpha_i \in \{0 \dots d\}.$$

The total wave function is therefore a vector with d^N coefficients.
Instead, the wave function of the system for a N-body, non interacting, separable pure state is:

$$|\Psi\rangle = \sum_{\alpha_1} C_{\alpha_1} |\psi_{\alpha_1}\rangle \otimes \dots \otimes \sum_{\alpha_N} C_{\alpha_N} |\psi_{\alpha_N}\rangle$$

The separable case wave function is therefore a vector with $N \cdot d$ coefficients.

By definition, the *density matrix* of a generic pure state Ψ is $\rho = |\Psi\rangle\langle\Psi|$. The main proprieties of the density matrix, that we recall here, are:

1. $\text{Tr}\rho = 1$ (if Ψ was normalized)
2. $\rho = \rho^\dagger$

Let's now focus on the case of $N = 2$: here we have two systems A and B of a given dimension d. The general density matrix is a $d \times d$ matrix, where each line and each column can be identified with $N = 2$ numbers that assume all the possible values from 0 to d and make all the possible permutations.

For the particular case $N = 2$ and $d = 2$ the two *reduced density matrices* are defined as follows:

$$(\rho_B)_{m,n} = \sum_i \langle i| \langle m| \rho |n\rangle |i\rangle_A$$

$$(\rho_A)_{m,n} = \sum_i \langle i| \langle m| \rho |n\rangle |i\rangle_B$$

with $m, n = 0, 1$ and $i = 0, 1$. If we call ρ_{ij} the i-th row and j-th column element of ρ, in the case $N = 2, d = 2$ we can write

$$\rho_A = \begin{pmatrix} \rho_{11} + \rho_{22} & \rho_{13} + \rho_{24} \\ \rho_{31} + \rho_{42} & \rho_{33} + \rho_{44} \end{pmatrix}$$

$$\rho_B = \begin{pmatrix} \rho_{11} + \rho_{33} & \rho_{12} + \rho_{34} \\ \rho_{21} + \rho_{43} & \rho_{22} + \rho_{44} \end{pmatrix}$$

2 Code development

The `DMATRIX` and `DVECTORS` types defined in the second exercise are used. For the first task, a function called `RANDOM_INITV` is defined inside the matrices' module, in order to randomly ($\mathcal{U}(0,1)$) initialize the $N \cdot d$ elements of the separable case vector and the d^N elements of the general one. It directly performs the normalization. The program can read the input values d, N reading them from the file `inputdim.dat`.

```
FUNCTION RANDOM_INITV(length)
  INTEGER, INTENT(IN) :: length
  INTEGER :: ii
  REAL*8 xx,yy
  DOUBLE COMPLEX :: temp

  TYPE(DVECTOR) :: RANDOM_INITV
  RANDOM_INITV%N = length
  ALLOCATE(RANDOM_INITV%elem(length))
  temp = 0.0

  DO ii=1,length
  CALL RANDOM_NUMBER(xx)
  CALL RANDOM_NUMBER(yy)
     RANDOM_INITV%elem(ii) = complex(xx,yy)
     temp = temp + (RANDOM_INITV%elem(ii))*conjg(RANDOM_INITV%elem(ii))
  END DO

  RANDOM_INITV%elem = RANDOM_INITV%elem/cdsqrt(temp)

  RETURN
END FUNCTION RANDOM_INITV
```

The efficiency of the function in the two cases was tested: we expect a linear dependence $t = \beta N d$ for the separable case and an exponential dependence $t = \alpha d^N$ for the general case. The python script performs fits and graphics, taking as input the files `separable_times.txt` and `general_times.txt`, that contain four columns containing the variables d, N, $N \cdot d$ or d^N,time. In order to have good results in the fits, the range of N must be properly chosen in the two cases (up to ~ 25 for the general case, not too small in the separable one). Results are shown in Fig.1.

The predicted results are verified; in particular, the two fits gave the same coefficient in the two cases:

$$\alpha = \beta = 1.2236 \cdot 10^{-7} s \ :$$

this should be the time needed to generate one complex number and store it in a vector.

The RAM capacity limits a lot the number N of bodies that the system can handle. In particular, recalling that a double complex number occupies 16 bits and that the minimum

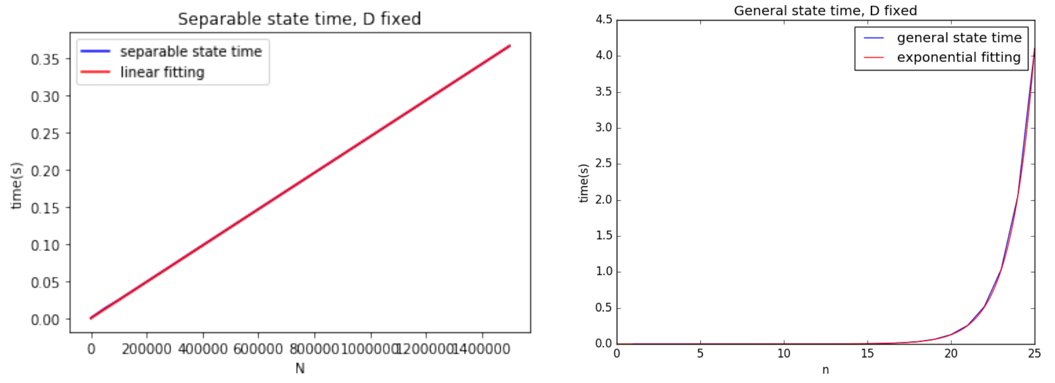

Figure 1: Time of `RANDOM_INITV` function as a function of N for $d = 2$, separable (left) and general (right) case.

value of d is 2, if x is the number of Gigabytes of the RAM, N can be at most:

$$N = \log_2\left(\frac{x \cdot 10^9}{16}\right) .$$

If $x = 8$, $N \simeq 28.89$, and in fact setting $N = 29$ in the fortran program makes a warning message about memory appear.

In the second part of the code, the density matrix of a general randomly initialized state $|\Psi\rangle$ with the desired value of d and $N = 2$ is computed:

```fortran
SUBROUTINE DENSITY_MATRIX(vector, DENS_MAT)
  INTEGER :: nrow, ii, jj
  TYPE(DMATRIX):: DENS_MAT
  TYPE(DVECTOR) :: vector

  nrow = vector%N
  DENS_MAT%N(1) = vector%n
  DENS_MAT%N(2) = vector%n
  ALLOCATE (DENS_MAT%elem(nrow,nrow))

  DO ii=1, nrow
    DO jj=1, nrow
      DENS_MAT%elem(ii,jj) = conjg(vector%elem(ii))*(vector%elem(jj))
    END DO
  END DO

  DENS_MAT%tr = TRACE(DENS_MAT)

END SUBROUTINE
```

The two reduced density matrices are calculated also. Recall from the definition that the indices of the rows and columns of the density matrix are combinations of N numbers that can assume d values. These can be seen as numbers in the d basis, and can be converted in decimal simply by doing:

$$n = \sum_{k=0}^{N-1} \alpha_k d^k$$

Now if $N = 2$ as in our case, and adding a +1, we have the formula that finds the row and column numbers corresponding to our coefficients:

$$n = 1 + \alpha_0 \cdot d^0 + \alpha_1 \cdot d^1$$

with $\alpha_{0,1} = \{0, \dots, d\}$.

For the two reduced matrices, two subroutines are written. Calling i, j, k the indices and recalling the definition of reduced matrix, it can be found that:

$$\rho^A_{i,j} = \sum_{k=0}^{d} \rho_{ik,jk}$$

$$\rho^B_{i,j} = \sum_{k=0}^{d} \rho_{ki,kj}$$

with $i, j \in \{0, \dots d\}$. The subroutines makes all the needed permutations of the indexes and performs the sums, accessing the elements thank to the conversion to decimal number just seen.

```fortran
SUBROUTINE REDUCED_A(DENS_MAT, RED_A, DD)
  TYPE(DMATRIX) :: DENS_MAT
  TYPE(DMATRIX) :: RED_A
  INTEGER :: ii, jj, kk ,index1, index2, NN
  INTEGER:: DD

  NN = 2

  RED_A%N(1) = DD
  RED_A%N(2) = DD
  ALLOCATE (RED_A%elem(DD,DD))
  RED_A%elem = complex(0d0,0d0)

  DO ii=0, DD-1
    DO jj=0, DD-1
      DO kk=0, DD-1
        index1 = 1 + ii*DD + kk
        index2 = 1 + jj*DD + kk
         RED_A%elem(ii+1,jj+1) = RED_A%elem(ii+1,jj+1) + DENS_MAT%elem(
   index1,index2)
      END DO
    END DO
  END DO

  RED_A%tr = TRACE(RED_A)

  END SUBROUTINE

SUBROUTINE REDUCED_B(DENS_MAT, RED_B, DD)
  TYPE(DMATRIX) :: DENS_MAT
  TYPE(DMATRIX) :: RED_B
  INTEGER :: ii, jj, kk ,index1, index2, NN
  INTEGER :: DD

  NN = 2

  RED_B%N(1) = DD
  RED_B%N(2) = DD
  ALLOCATE (RED_B%elem(DD,DD))
  RED_B%elem = complex(0d0,0d0)

  DO ii=0, DD-1
    DO jj=0, DD-1
      DO kk=0, DD-1
        index1 = 1 + kk*DD + ii
        index2 = 1 + kk*DD + jj
         RED_B%elem(ii+1,jj+1) = RED_B%elem(ii+1,jj+1) + DENS_MAT%elem(
   index1,index2)
```

```
      END DO
    END DO
  END DO

  RED_B%tr = TRACE(RED_B)
END SUBROUTINE
```

The matrix and the reduced matrices are printed in three different output files.

3 Results

The code has been tested for several simple systems with $N = 2, d = 2$. The correct normalization always lead to traces equal to one, the matrices are hermitian. In particular, the system seen in class:

$$\rho = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

has been tested, leading to the correct reduced matrices:

$$\rho^A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\rho^B = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

Results are printed in three output files.

4 Self evaluation

All the required tasks have been achieved. More could have been done, e.g. implementing a subroutine for the tensor product to obtain the general form of the separable wave function, or showing more results in the results section.