Quantum Information and Computing 2020/21
Week 8 report

Rocco Ardino

(Dated: Monday 30th November, 2020)

In this work we deal with quantum N-body states and density matrices. We first face the problem of a minimal memory consuming description of those states in both separable and non-separable cases. Then, we implement the initialization of the density matrix from a certain wavefunction and the partial trace operation, performed on a certain subsystem. Lastly, we test everything on a system of 2 qubits.

1 Theory

Let us consider a quantum N-body system, where each subsystem wavefunction is denoted by $|\psi_i\rangle \in \mathcal{H}^d$. Without losing of generality, it is convenient to write each $|\psi_i\rangle$ as a linear combination of $|\alpha_j\rangle \in \{|0\rangle, \dots, |d-1\rangle\}$. Therefore, with this notation, the wavefunction $|\Psi\rangle \in \mathcal{H}^{d^N}$ of the whole pure state system reads:

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

with $C_{\alpha_1,\dots,\alpha_N}$ complex coefficients. Another way to express Eq. 1, which exploits the d-basis representation and which will be useful for the code implementation, reads:

$$|\Psi\rangle = \sum_{i=0}^{d^N-1} C_i |i\rangle \quad , \quad i = \sum_{j=1}^{N} \alpha_j d^{j-1} \quad . \tag{2}$$

A special case of the previous discussion is the one of a separable pure state. In this case the N-body wavefunction has a simpler form:

$$|\Psi\rangle = \left(\sum_{\alpha_1} C_{\alpha_1} |\alpha_1\rangle\right) \otimes \dots \otimes \left(\sum_{\alpha_N} C_{\alpha_N} |\alpha_N\rangle\right) \quad . \tag{3}$$

We can see the difference with the non-separable case. In fact, the ladder is defined by d^N complex coefficients, while the separable wavefunction requires $d \cdot N$ complex coefficients. This fact translates in a significantly smaller initialization time for a separable wave function.

After having determined the pure state N-body wavefunction $|\Psi\rangle$, its density matrix is defined as follows:

$$\rho = |\Psi\rangle\langle\Psi| = \sum_{i,j=0}^{d^N-1} C_j^* C_i |i\rangle\langle j| \quad . \tag{4}$$

Starting from this definition, a reduced density matrix on the kth subsystem is computed by tracing over all the remaining subsystems:

$$\rho_k = \mathrm{tr}_1 \cdots \mathrm{tr}_{k-1}\, \mathrm{tr}_{k+1} \cdots \mathrm{tr}_N\, \rho \quad , \tag{5}$$

where the trace with index i reads:

$$\mathrm{tr}_i\, \rho = \sum_{j=0}^{d-1} \langle j|\rho|j\rangle \quad . \tag{6}$$

2 Code Development

For this work, the new module **dmat** is implemented, containing several functions and subroutines for multiple purposes:

- **dmat_init_rand_state(N,D,isSep)**, for instantiating a N-body (non-)separable system, with each subsystem living inside $\mathcal{H}^d$;
- **dmat_init_rho(state)**, for initializing the density matrix, given the state;
- **dmat_rho_reduce(rho,nsub,N,D)**, for calculating the reduced density matrix, obtained by tracing the nsub subsystem;
- **dmat_print_state_std(state,formatted)**, for printing the state coefficients on standard output and in a formatted way, if specified;
- **dmat_print_rho_std(rho,formatted)**, for printing the density matrix entries on standard output and in a formatted way, if specified.

Moreover, another module, namely **cmdline**, is implemented to have a command-line argument parser and for default arguments handling. In particular, we explain the new core functionalities of both the modules in the following subsections.

2.1 Pure state initialization

The code to perform the initialization of the state coefficients is showed in Listing 1. Note how the operation is vectorized through the intrinsic function RANDOM_NUMBER and the difference in the number of coefficients depending on the separability of the state.

```fortran
function dmat_init_rand_state(N, D, isSep) result(state)
    ! input arguments
    integer(4) :: N
    integer(4) :: D
    logical    :: isSep

    integer(4) :: dim
    complex(8), dimension(:), allocatable :: state
    real(8),    dimension(:), allocatable :: re, im
    real(8)    :: norm

    if (isSep) then
        dim = D*N
    else
        dim = D**N
    end if

    allocate(state(dim))
    allocate(re(dim))
    allocate(im(dim))

    ! fill real and imaginary part vectors
    call RANDOM_NUMBER(re)
    call RANDOM_NUMBER(im)
    re = re*2.0d0 - 1.0d0
    im = im*2.0d0 - 1.0d0

    ! fill state vector with coefficients and normalize
    norm  = SUM(re**2 + im**2)
    state = (re*COMPLEX(1.0d0,0.0d0) + im*COMPLEX(0.0d0,1.0d0)) / COMPLEX(SQRT(norm),0.0d0)
end function dmat_init_rand_state
```

Listing 1. Implementation of the state coefficients initialization.

2.2 Density matrix initialization

The code to perform the the initialization of the density matrix is showed in Listing 2. The technique employed is nothing more than a tensor product, performed using the intrinsic `MATMUL`. At the end of the execution, the apposite function returns a $d^N \times d^N$ complex matrix.

```fortran
function dmat_init_rho(state) result(rho)
    ! input arguments
    complex(8), dimension(:) :: state

    complex(8), dimension(:,:), allocatable :: rho
    complex(8), dimension(:,:), allocatable :: bra, ket
    integer(4) :: N

    N = size(state,1)

    allocate(rho(N,N))
    allocate(bra(1,N))
    allocate(ket(N,1))

    bra(1,:) = CONJG(state)
    ket(:,1) =       state

    rho = MATMUL(ket,bra)
end function dmat_init_rho
```

Listing 2. Implementation of the density matrix initialization.

2.3 Density matrix reduction

The code to perform the reduction of density matrix, by tracing on the kth subsystem, is showed in Listing 3. What is important to observe is that the technique of using a d-basis representation of Eq. 2 is employed. If we consider a number written in this representation, with the α_is representing its digits, we need two loops for every dimension of the density matrix:

- one for cycling over the less significant α_is, namely the ones with $i < k$;
- one for cycling over the more significant α_is, namely the ones with $i > k$.

Then, another loop over α_k is needed in order to perform the summations needed for the trace.

```fortran
function dmat_rho_reduce(rho, nsub, N, D) result(rho_red)
    ! input arguments
    complex(8), dimension(:,:) :: rho
    integer(4) :: nsub
    integer(4) :: N
    integer(4) :: D

    complex(8), dimension(D**(N-1),D**(N-1)) :: rho_red
    complex(8) :: sum
    integer(4) :: lli, llj, mmi, mmj, kk, ii, jj, iir, jjr

    ! loop over rho elements of all subsystems except nsub
    do lli=0,D**(nsub-1)-1
        do mmi=0,(D**(N-nsub))-1
            do llj=0,D**(nsub-1)-1
                do mmj=0,(d**(N-nsub))-1
                    ! trace the nsub subsystem
                    sum = COMPLEX(0.0d0,0.0d0)
                    do kk=0,D-1
                        ii = lli + 1 + (kk + mmi*D)*D**(nsub-1)
                        jj = llj + 1 + (kk + mmj*D)*D**(nsub-1)
                        sum = sum + rho(ii,jj)
                    end do
                    ! fill reduced density matrix
                    iir = lli + 1 + mmi*D**(nsub-1)
                    jjr = llj + 1 + mmj*D**(nsub-1)
                    rho_red(iir, jjr) = sum
                end do
            end do
        end do
    end do
end function dmat_rho_reduce
```

Listing 3. Implementation of the function for density matrix reduction over the k^{th} subsystem.

2.4 Main program

Lastly, all the functions described before are employed in a main program, in which a command-line argument parser is implemented in order to obtain an easier handling of the parameters of the simulation. Moreover, this functionality is also capable of giving default values to the arguments when they are not given. To make the program more user-friendly, a `-h` help option has been added and in the beginning of the execution all the arguments given are printed along with a flag: **F** if a value is given for it, **T** if default is taken. An example of this functionality and of the way to start the program is showed in Listing 4. A complete list of the options is given in Table 1.

Arg option	Arg name	Arg meaning	Default
-N	subsysN	N	2
-D	dimD	d	2
-S	separable	Separability	0
-M	mode	Execution routine	1
-K	reduceK	Subsystem to reduce	1
-h	help	print help	

Table 1. Command-line arguments of the program.

```
$ ./08.o -S 0 -M 1

Running with the following cmdline options:

N =         2     default [T/F]: T
D =         2     default [T/F]: T
S =         0     default [T/F]: F
M =         1     default [T/F]: F
K =         1     default [T/F]: T

CPU time:
3.1999999999999910E-005
```

Listing 4. Example of launch and user interface.

After the parser, if no exception is found, a subroutine with the corresponding mode of execution is called. The possible modes are the following:

- **mode_1**: it executes state initialization and returns CPU time needed for the purpose;
- **mode_2**: it executes state and density matrix initialization (only for non-separable states), reduces by tracing on the k^{th} subsystem and prints both ρ and reduced ρ;
- **mode_3**: it does the same operations as `mode_2`, but with a system of two spin one-half qubits and with reduction of both subsytems.

Last but not least, the program must be compiled with the flag `-ffree-line-length-0`.

3 Results

3.1 State initialization efficiency

The first analysis performed concerns the CPU time needed to initialize a fully defined N-body pure state for both separable and non-separable cases. For this purpose, the code returning the CPU time is run for multiple combinations of N and d. The results are showed in Figure 1. What is important to remark is how, fixed d, the CPU time needed for the initialization of a separable state is polynomial in N, while it is exponential in N for the non-separable case. This is what we expect if we think in terms of the number of coefficients needed.

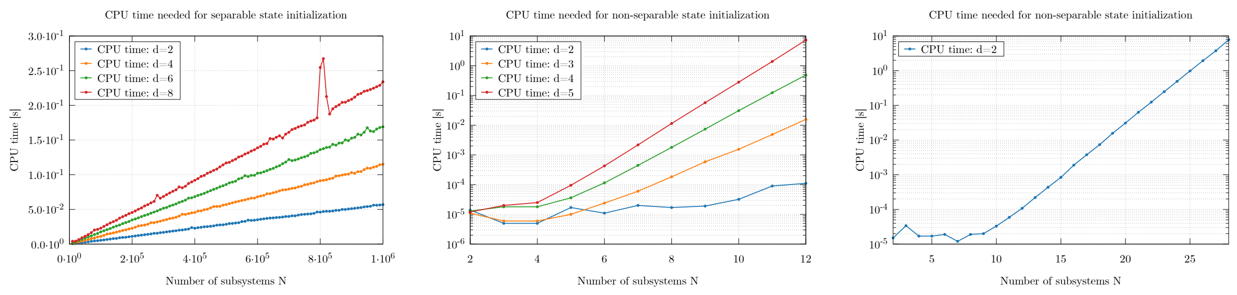

Figure 1. CPU time results for separable (left) and non-separable (center) state initialization, depending on N and d, and for a non-separable state initialization with fixed $d = 2$ and larger values of N (right).

3.2 Density matrix calculation and reduction for $N = 2$, $d = 2$

The code implementation for density matrix initialization and reduction is validated for $N = d = 2$, namely for the case of two spin one-half qubits. The execution mode 3 is run for several cases and an example result is showed in Listing 5. The ladder shows the correctness of the partial tracing operation implemented.

```
$ ./08.o -M 3

Running for two spin one-half qubits:

State coefficients:
(-0.3267 -0.4480i)
( 0.0629 +0.6569i)
( 0.4124 -0.1932i)
(-0.1970 +0.1043i)

Density matrix:
( 0.3074 +0.0000i) (-0.3148 +0.1864i) (-0.0482 -0.2479i) ( 0.0176 +0.1223i)
(-0.3148 -0.1864i) ( 0.4355 -0.0000i) (-0.1010 +0.2830i) ( 0.0561 -0.1360i)
(-0.0482 +0.2479i) (-0.1010 -0.2830i) ( 0.2074 +0.0000i) (-0.1014 -0.0049i)
( 0.0176 -0.1223i) ( 0.0561 +0.1360i) (-0.1014 +0.0049i) ( 0.0497 +0.0000i)

Reduced density matrix (on left system):
( 0.7429 +0.0000i) ( 0.0080 -0.3839i)
( 0.0080 +0.3839i) ( 0.2571 +0.0000i)

Reduced density matrix (on right system):
( 0.5148 +0.0000i) (-0.4162 +0.1815i)
(-0.4162 -0.1815i) ( 0.4852 -0.0000i)
```

Listing 5. Example of execution for two spin one-half qubits, with density matrix initialization and reduction.

4 Self-evaluation

In this work we have faced the problem of density matrices initialization and handling, as well as pure state definition efficiency for both separable and non-separable cases. The code implementation has led to successful results for the density matrix initialization and partial tracing, validated with the example of 2 qubits. Moreover, it has showed the efficiency of state coefficients initialization for different values of N and d.