Quantum Information and Computing - Assignment 6

In this assignment we were required to write and test various functions to deal with composite systems.

First of all, we were to write a function to describe the system in the case of a separable non-interacting pure state. This is best done using a list of states, each referring to the correspondingly indexed subsystem, as seen in the *random_separable* function.

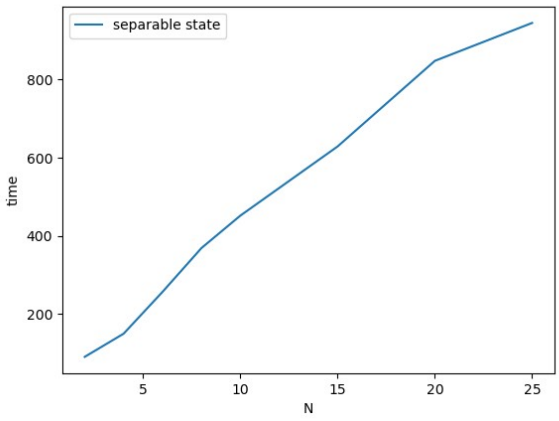
To build the vector describing the state in the full Hilbert space, the *separable* function can be used; this takes in as different arguments the states of the various subsystems and returns the full-space vector. The same function can also be applied to do the same with density matrices or operators, as needed.

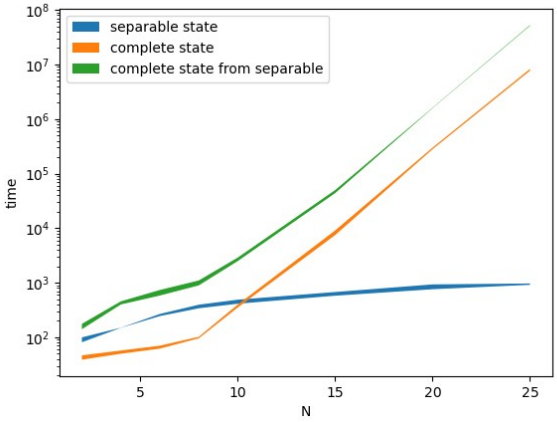
The function *random_composite* instead builds a random generic vector in the full space.

To find the density matrix of a state the *get_density_mat* can be used; this returns as a list the density matrices of all the states that are passed as different arguments, thus allowing to work with the subsystem-separate representation.

Finally, to trace out a subsystem from the full density matrix the *trace_out_subs* function can be used, specifying the index of the subsystem to be traced out and the dimension of all the subsystems.

I also added a function to pad an operator with another, which by default is the identity, to be able to construct the operator over the whole space given the local actions.





The execution time for the creation of a random separable state is linear (as seen in the left graph), while the creation of a generic random state is exponential. This is expected, since the number of entries scales as $N \cdot D$ for the separable case and as D^N in the generic one; we can see that for small N there is a slight overhead in the separable case due to the normalization of the state for each subsystem (so N normalizations instead of 1); the efficiency might also be improved by generating all the random numbers in a single call instead of creating N times the amount needed for a subsystem. The numbers used in the graphs were obtained for D=2.

The memory used to store either state is proportional to the number of entries, with a slight overhead due to the metadata.