(Sort of efficient) computation of a partial trace of a matrix

Jaroslav Kysela

A concise (and kinda elegant) computer code is given for computation of a partial trace of a matrix over arbitrary number of subsystems with arbitrary dimensions. The mathematical explanation behind the algorithm is also given.

I. INTRODUCTION

We consider an input matrix $M \in \mathbb{C}^{d\times d}$, that is understood as a density matrix of n-partite system, where the k-th subsystem is associated with a d_k-dimensional subspace. That is, $d = \prod_{k=1}^{n} d_k$. The partial trace of matrix M over subsystems $(a_1, \ldots, a_K)$ with $K \leq n$ is given by

$$M' = \mathrm{Tr}_{a_1,\ldots,a_K} M = \sum_{i_{a_1}, i_{a_2}, \ldots, i_{a_K}} M_{i_1,\ldots,i_n;j_1,\ldots,j_n} \delta_{i_{a_1},j_{a_1}} \delta_{i_{a_2},j_{a_2}} \ldots \delta_{i_{a_K},j_{a_K}}, \tag{1}$$

where we used the multiindex notation for row and column indices. In this notation the row index i (for column indices the formula is analogous) is associated with a multiindex $(i_1, \ldots, i_n)$, such that

$$i = \sum_{j=1}^{n} \left(\prod_{k=j+1}^{n} d_k \right) i_j. \tag{2}$$

As follows from the properties of the partial trace, we can express the trace over K subsystems by K successive applications of a trace over a single subsystem. In the following we will therefore restrict ourselves only to the one-subsystem scenario.

There are multiple ways how to calculate a partial trace of a square matrix for a given subspace and the matrix dimension. One approach is to relabel matrix elements by multiindices and then sum only over those indices in the multiindex that correspond to a given subspace. Such a technique requires kinda low-level element-wise approach, which is, if nothing else, not elegant.

The same can be also done in a more high-level way, where entire blocks of matrix M are summed up instead of individual elements. The resulting computer code is also more elegant. In the following we analyse formulas for the partial trace and in the end we present a general explicit Mathematica code that performs a partial trace over multiple subsystems of arbitrary dimensions.

II. BACKGROUND

The method relies on identifying the roles of specific parts of the expression in Eq. (2), as explained below. The row index i of matrix M can be written as follows (for column indices an analogous discussion can be done, of course)

$$i = \sum_{j=1}^{p-1} \left(\prod_{k=j+1}^{n} d_k \right) i_j + \left(\prod_{k=p+1}^{n} d_k \right) i_p + \sum_{j=p+1}^{n} \left(\prod_{k=j+1}^{n} d_k \right) i_j \tag{3}$$

$$= \underbrace{\left(\prod_{k=p}^{n} d_k \right)}_{S} \underbrace{\sum_{j=1}^{p-1} \left(\prod_{k=j+1}^{p-1} d_k \right) i_j}_{r} + \underbrace{\left(\prod_{k=p+1}^{n} d_k \right)}_{Q} i_p + \underbrace{\sum_{j=p}^{n-1} \left(\prod_{k=j+2}^{n} d_k \right) i_{j+1}}_{q}, \tag{4}$$

where in the first term we factored out in front of the sum all addends that do not depend on the summation index and in the third term we shifted the summation index by one. The symbols in the last expression have the following properties:

- As i increases, q goes successively through $0, 1, \ldots, Q-1$. To see that the upper bound is $Q-1$ note that the

largest value of q is attained when $i_{j+1} = d_{j+1} - 1$, for which we get

$$q = \sum_{j=p}^{n-1} \left(\prod_{k=j+2}^{n} d_k \right) (d_{j+1} - 1) \tag{5}$$

$$= \sum_{j=p}^{n-1} \left(\prod_{k=j+1}^{n} d_k \right) - \sum_{j=p}^{n-1} \left(\prod_{k=j+2}^{n} d_k \right) \tag{6}$$

$$= \sum_{j=p}^{n-1} \left(\prod_{k=j+1}^{n} d_k \right) - \sum_{j=p+1}^{n} \left(\prod_{k=j+1}^{n} d_k \right) \tag{7}$$

$$= \left(\prod_{k=p+1}^{n} d_k \right) - \left(\prod_{k=n+1}^{n} d_k \right) \tag{8}$$

$$= Q - 1. \tag{9}$$

$$\tag{10}$$

- As i increases, r goes successively through $0, 1, \ldots, R-1$, where

$$R = \prod_{k=1}^{p-1} d_k. \tag{11}$$

The derivation of this upper bound is completely analogous to the calculation in the previous item. Note that

$$R\,S = R\,d_p\,Q = \prod_{k=1}^{n} d_k \equiv d. \tag{12}$$

In the partial trace, we want to sum over i_p in Eq. (4). Values of S and Q are fixed, since we want to trace over the p-th subspace. The resulting matrix is equal to $M' \equiv \mathrm{Tr}_p M \in \mathbb{C}^{d' \times d'}$, where $d' = d/d_p$. Consider a row index i' of a matrix element $M'_{i'j'}$. In the multiindex notation, we get $i' = (i'_1, i'_2, \ldots, i'_{n-1})$. This multiindex is closely related to the multiindices of elements M_{ij} of the original matrix M that contributed to the value of element $M'_{i'j'}$. The partial trace effects the transformation

$$(i_1, i_2, \ldots, i_{p-1}, i_p, i_{p+1}, \ldots, i_n) \to (i_1, i_2, \ldots, i_{p-1}, i_{p+1}, \ldots, i_n), \tag{13}$$

from where we see that

$$i'_k = \begin{cases} i_k & k < p \\ i_{k+1} & k \geq p \end{cases}. \tag{14}$$

For the corresponding subspace dimensions we get

$$d'_k = \begin{cases} d_k & k < p \\ d_{k+1} & k \geq p \end{cases}. \tag{15}$$

We can do an analogous discussion for i' as we did for i in Eq. (4). We obtain

$$i' = \sum_{j=1}^{n-1} \left(\prod_{k=j+1}^{n-1} d'_k \right) i'_j \tag{16}$$

$$= \sum_{j=1}^{p-1} \left(\prod_{k=j+1}^{n} d_k \right) \frac{1}{d_p}\, i_j + \sum_{j=p}^{n-1} \left(\prod_{k=j+1}^{n} d_{k+1} \right) i_{j+1} \tag{17}$$

$$= \underbrace{\left(\prod_{k=p+1}^{n} d_k \right)}_{Q} \underbrace{\sum_{j=1}^{p-1} \left(\prod_{k=j+1}^{p-1} d_k \right) i_j}_{r} + \underbrace{\sum_{j=p}^{n-1} \left(\prod_{k=j+2}^{n} d_k \right) i_{j+1}}_{q}, \tag{18}$$

where now the symbols below the last expression are not definitions, as it was in Eq. (4), but are really expression that are identical to those labeled by respective symbols in Eq. (4). In summary, we have

$$i = S\,r + Q\,i_p + q = Q\,(d_p\,r + i_p) + q, \tag{19}$$

$$i' = Q\,r + q. \tag{20}$$

Using these relations, we can rewrite the formula for the partial trace as

$$M'_{Qr+q,Q\bar{r}+\bar{q}} = \sum_{i_p=0}^{d_p-1} M_{Q(d_p r+i_p)+q,Q(d_p \bar{r}+i_p)+\bar{q}}, \tag{21}$$

where $\bar{r}$ and $\bar{q}$ are column equivalents of r and q, respectively. From Eq. (20) we see that the reduced matrix can be understood as living in a tensor product space $\mathbb{C}^{R\times R} \times \mathbb{C}^{Q\times Q}$, where $RQ = d'$, see Eq. (12). The first space is indexed by r, the other by q. For each double (r, q) we have a set of row indices $\{i = S\,r + Q\,i_p + q\}_{i_p}$ (and similarly for column indices j), such that elements M_{ij} of the original matrix sum up to $M'_{i'j'}$ of the reduced matrix.

Consider a block $B'_{r,\bar{r}} \in \mathbb{C}^{Q\times Q}$ of matrix M', where r ($\bar{r}$) is the row (column) index of the block. Individual rows (columns) inside the block are indexed by q ($\bar{q}$). The equation Eq. (21) can be written in the block form as

$$B'_{r,\bar{r}} = \sum_{i_p=0}^{d_p-1} B_{d_p\,r+i_p,d_p\,\bar{r}+i_p}, \tag{22}$$

where B's are $Q \times Q$ dimensional blocks of the original matrix M. It is precisely this formula that is used in the computer code, presented in the next section.

III. CODE

At this point, we are ready to write down the explicit computer code. The partial trace of a matrix over an arbitrary number of subsystems for arbitrary dimensions is given below, encapsulated into function `ptraceSum`, where `mat` is matrix M, `idx` is a list of integers that index individual subsystems over which the trace is to be performed, and `dims` is a list of subsystem dimensions $(d_1, \ldots, d_n)$:

```
ptraceSum[mat_, idx_, dims_] := Module[{aux, lmat = mat, ldims = dims, pre, d, post},

    Do[
        pre = Times @@ ldims[[;; subidx - 1]];
        d = ldims[[subidx]];
        post = Times @@ ldims[[subidx + 1 ;;]];

        aux =
            Table[
                Sum[
                    lmat[[
                        post (d (is - 1) + k - 1) + 1 ;; post (d (is - 1) + k),
                        post (d (js - 1) + k - 1) + 1 ;; post (d (js - 1) + k)
                    ]]
                , {k, d}]
            , {is, pre}, {js, pre}
            ];

        lmat = ArrayFlatten[aux];
        ldims = Drop[ldims, {subidx}];

        , {subidx, Reverse[idx]}
    ];

    lmat
]
```

The outermost `Do` function applies the inner body iteratively to all subspaces over which the partial trace is to be done. The inner body implements a one-subspace partial trace, for which the mathematical explanation was given above. Variable `pre` is identical to R and variable `post` is identical to Q. The resulting matrix is calculated block-wise, where block B_{ab} is calculated as a sum over `k`, where `k` corresponds to i_p and goes from 1 to `d`, which is equal to d_p. Individual blocks are indexed by `is` and `js`. The row index `is` corresponds to r (and analogously for the column index `js`). The `ArrayFlatten` command only removes the formal block structure from M' to represent it as a truly 2-dimensional array. The last command `ldims=Drop[ldims,{subidx}]` is not a part of the tracing, it only prepares the variable `ldims` for the next iteration.

IV. NOTE ON ALTERNATIVE TECHNIQUES

In Mathematica, instead of the above-mentioned code an alternative code that uses built-in procedure `TensorContract` can be used. For the latter function we need to transform the input matrix into a tensor that can be accessed by a multiindex. We can do that at least in two ways. The first way reads as follows:

```
ptraceTensorContract[mat_, idx_, dims_] := Module[{aux, lidx},

  lidx = Transpose[{2 idx - 1, 2 idx}];
  aux = Fold[Partition[#1, {#2, #2}] &, mat, Most@Reverse@dims];

  aux = TensorContract[aux, lidx];

  aux = If[Length[dims] == Length[idx], {{aux}},
    Nest[ArrayFlatten, aux, Length[dims] - Length[idx] - 1]];
  aux
]
```

The second, which turns out to be faster, is given by:

```
ptraceTensorContract2[mat_, idx_, dims_] := Module[{aux, lidx, resdim},

  lidx = Transpose[{idx, idx + Length[dims]}];
  aux = ArrayReshape[mat, Flatten[{dims, dims}]];

  aux = TensorContract[aux, lidx];

  If[Length[idx] == Length[dims], aux = {{aux}}];
  resdim = (Times @@ dims)/(Times @@ dims[[idx]]);
  aux = ArrayReshape[aux, {resdim, resdim}];

  aux
]
```

Both functions use `TensorContract` to perform the actual partial trace, but the pre- and post-processing stages differ. In the first function, the input matrix is iteratively partitioned in the preprocessing stage, such that the resulting tensor corresponds to the tensor product of matrices of dimensions $d_k \times d_k$. Specifically, in step k the matrix is partitioned into square blocks of size $d_k \times d_k$. The partial trace over the p-th subspace then corresponds to `TensorContract` applied to indices $(2p-1, 2p)$, i.e. the two successive indices that correspond to the p-th partitioning. The resulting tensor is transformed back by successive block flattening implemented with `ArrayFlatten`.

In the second function, the nesting partitioning is avoided by using `ArrayReshape`. This built-in function seems to work effectively as follows. Given the resulting dimensions $(b_1, \ldots, b_M)$, it first flattens the input array and then it partitions successively the one-dimensional list into blocks of length b_M, then b_{M-1} etc. Provided the input array is a matrix indexed by i and j, the action of `ArrayReshape` can be represented as

$$(i, j) \to d\, i + j \to (a_1, \ldots, a_M), \tag{23}$$

where $d \times d$ are the matrix dimensions and multiindex a satisfies

$$d\, i + j = \sum_{l=1}^{M} \left(\prod_{k=j+1}^{M} b_k \right) a_l, \tag{24}$$

with $0 \leq a_l < b_l$. In order for `TensorContract` to implement the partial trace, it needs as an input a tensor with $2n$ indices, where n is the number of subspaces. `ArrayReshape` can give us the $(2n)$-dimensional array, but the question is then, over which indices to perform the contraction. This obviously depends on the choice of dimensions $(b_1, \ldots, b_M)$, where $M = 2n$ and $b_j \in d_1, \ldots, d_n$. The choice corresponding to function `ptraceTensorContract` does not seem to give a nice result. On the other hand, when we choose $(b_1, \ldots, b_{2n}) = (d_1, \ldots, d_n, d_1, \ldots, d_n)$, the resulting expression is easy to interpret. Specifically, in such a case we obtain

$$d\,i + j = \sum_{l=1}^{n} \left(\prod_{k=l+1}^{2n} b_k \right) a_l + \sum_{l=n+1}^{2n} \left(\prod_{k=l+1}^{2n} b_k \right) a_l, \tag{25}$$

$$= \sum_{l=1}^{n} \left(\prod_{k=l+1}^{n} d_k \right) \left(\prod_{k=n+1}^{2n} d_{k-n} \right) a_l + \sum_{l=n+1}^{2n} \left(\prod_{k=l+1}^{2n} d_{k-n} \right) a_l, \tag{26}$$

$$= \left(\prod_{k=1}^{n} d_k \right) \sum_{l=1}^{n} \left(\prod_{k=l+1}^{n} d_k \right) a_l + \sum_{l=1}^{n} \left(\prod_{k=l+n+1}^{2n} d_{k-n} \right) a_{l+n}, \tag{27}$$

$$= d \sum_{l=1}^{n} \left(\prod_{k=l+1}^{n} d_k \right) a_l + \sum_{l=1}^{n} \left(\prod_{k=l+1}^{n} d_k \right) a_{l+n}, \tag{28}$$

$$= \sum_{l=1}^{n} \left(\prod_{k=l+1}^{n} d_k \right) (d\,a_l + a_{l+n}). \tag{29}$$

On the other hand, using the representation in Eq. (2) for i and similarly for j, we get

$$d\,i + j = d \sum_{l=1}^{n} \left(\prod_{k=l+1}^{n} d_k \right) i_l + \sum_{l=1}^{n} \left(\prod_{k=l+1}^{n} d_k \right) j_l, \tag{30}$$

$$= \sum_{l=1}^{n} \left(\prod_{k=l+1}^{n} d_k \right) (d\,i_l + j_l). \tag{31}$$

By comparison of Eqs. (29) and (31), we see that

$$a_l = i_l, \quad a_{l+n} = j_l, \quad 0 \leq l < n, \tag{32}$$

that is

$$(a_1, \ldots, a_n, a_{n+1}, \ldots, a_{2n}) = (i_1, \ldots, i_n, j_1, \ldots, j_n). \tag{33}$$

The partial trace over the p-th subspace therefore corresponds to the tensor contraction of indices (a_p, a_{p+n}). The redefinition of indices, over which to contract the tensor, is done in the preprocessing stage of function `ptraceTensorContract2`. The resulting array is than transformed back into a two-dimensional matrix by the second application of `ArrayReshape` in the postprocessing stage.