General Partial Trace for n-parties and d-dimensional Qudits

Dr. M. S. Ramkarthik[1], Kaushal Joshi[2]

Theoretical Physics Group
Visvesvaraya National Institute of Technology, Nagpur

1 Partial Trace

The most important application of the density matrix is as a descriptive tool for the subsystems of the composite system, known as the reduced density matrix.

Suppose we have a physical system AB, whose state is given by ρ_{AB}, and if we want to study subsystem A, then we can get the reduced density matrix of A, using an operation known as partial trace taken over the subsystem B as,

$$\rho_A \equiv \mathrm{tr}_B(\rho_{AB}), \tag{1}$$

where tr_B is the partial trace operator. The partial trace is defined by,

$$\mathrm{tr}_B(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) \equiv |a_1\rangle\langle a_2|\mathrm{tr}(|b_1\rangle\langle b_2|), \tag{2}$$

where $|a_1\rangle$ and $|a_2\rangle$ are any two vectors in the state space of A and $|b_1\rangle$ and $|b_2\rangle$ are any two vectors in the state space of B.

Now, one may think, why is the partial trace used to describe part of a larger system? This is because the partial trace is a unique operation that gives the correct description of observable quantities for subsystems of a composite system. This can be generalised to multi-qudit systems as well. If we have an N-qudit state $\rho_{123\cdots N}$, if we are interested in the state of the even-qudit subsystem, then we can write:

$$\rho_{246\cdots} \equiv \mathrm{Tr}_{135\cdots}\left(\rho_{123\cdots N}\right). \tag{3}$$

This implies that the partial trace can be performed over any arbitrary subset of qudits. The partial trace operation serves as the quantum analogue of obtaining a marginal distribution from a multivariate probability distribution. We have made a Fortran program to compute the partial trace for n-parties and a d-dimensional system. The motivation behind this is that no subroutine in Fortran performs d-dimensional, n-parties partial trace; some subroutines perform partial trace for qubits, but when we are dealing with higher dimensions, there is no partial trace subroutine as such.

The importance of this subroutine is, it is the most general case of partial trace; it gives us a hand to trace out any number of parties, for any dimensional systems. Let us look into the details of this subroutine for real density matrices,

---

subroutine ptrace(rho, d, s, s1, site, rdm)

In/Out	Argument	Description
in	`rho`	rho is `real*8` array of dimension `(0:d**s-1, 0:d**s-1)` representing the input density matrix.
in	`d`	d (integer) is the dimension of each qudit.
in	`s`	s (integer) is the total number of qudits in the system.
in	`s1`	s1 (integer) is the number of qudits not to be traced out.

In/Out	Argument	Description
in	`site`	site is `real*8` array of dimension `(0:s-1)` containing site indices of qudits to keep.
out	`rdm`	rdm is `real*8` array of dimension `(0:d**s1-1, 0:d**s1-1)` containing reduced density matrix after partial trace.

Table 1: Arguments for the subroutine computing the reduced density matrix (RDM).

Implementation

```
subroutine ptrace(rho, d, s, s1, site, rdm)
    implicit none
    integer, intent(in) :: d, s, s1
    integer, intent(in) :: site(0:s1-1)
    real*8, intent(in) :: rho(0:d**s-1, 0:d**s-1)
    real*8, intent(out) :: rdm(d**s1, d**s1)
    integer, allocatable :: idits(:), jdits(:), ridits(:)
    integer, allocatable :: rjdits(:), to(:)
    integer :: i, j, k, p, flag, ri, rj
    real*8 :: a

    allocate(idits(0:s-1), jdits(0:s-1))
    allocate(ridits(0:s1-1), rjdits(0:s1-1))
    allocate(to(0:s-s1-1))

    ! Compute the indices to trace out
    call TOI(s, site, s1, to)

    rdm = 0.0d0
    do i = 0, d**s - 1
        do j = 0, d**s - 1
            if (rho(i,j) .ne. 0.0d0) then
                a = rho(i,j)
                call DECTOD(i, d, s, idits)
                call DECTOD(j, d, s, jdits)
                flag = 0
```

```
                do k = 0, s - s1 - 1
                    if (idits(to(k)) .ne. jdits(to(k))) then
                        flag = 1
                        exit
                    end if
                end do
                if (flag .eq. 0) then
                    do k = 0, s1 - 1
                        p = site(k)
                        ridits(k) = idits(p)
                        rjdits(k) = jdits(p)
                    end do
                    call DTODEC(d, s1, ridits, ri)
                    call DTODEC(d, s1, rjdits, rj)
                    rdm(ri+1, rj+1) = rdm(ri+1, rj+1) + a
                end if
            end if
        end do
    end do

    deallocate(idits, jdits, ridits, rjdits, to)
end subroutine ptrace
```

A few subroutines that are used in this are,

Subroutine	Function
`TOI`	This subroutine computes the indices to be traced out and returns them into the array `to`.
`DECTOD`	This subroutine computes the qudit equivalent of any decimal number and returns it into the array `dits`.
`DTODEC`	This subroutine computes the decimal equivalent of any qudit and returns it into the integer `dec`.

Implementation of these subroutines is,

```
subroutine TOI(s, site, s1, to)
    implicit none
    integer, intent(in) :: s, s1
    integer, intent(in) :: site(0:s1-1)
    integer, intent(out) :: to(0:s-s1-1)
    integer :: i, j, k, found

    k = 0
    do i = 0, s-1
        found = 0
        do j = 0, s1 - 1
            if (i == site(j)) then
```

```
                found = 1
                exit
            end if
        end do
        if (found == 0) then
            to(k) = i
            k = k + 1
        end if
    end do
end subroutine TOI

subroutine DECTOD(dec, d, s, dits)
    implicit none
    integer, intent(in) :: dec, d, s
    integer, allocatable, intent(out) :: dits(:)
    integer :: i, temp_dec

    allocate(dits(0:s-1))

    temp_dec = dec
    do i = s-1, 0, -1
        dits(i) = mod(temp_dec, d)
        temp_dec = temp_dec / d
    end do
end subroutine DECTOD

subroutine DTODEC(d, s, dits, dec)
    implicit none
    integer, intent(in) :: d, s
    integer, dimension(0:s-1), intent(in) :: dits
    integer, intent(out) :: dec
    integer :: i

    dec = 0
    do i = 0, s - 1
        dec = dec * d + dits(i)
    end do
end subroutine DTODEC
```

In the same way, one can get the subroutine for complex density matrices just by changing the data type of the arrays involved; the logic will be the same.