Schmidt Decomposition

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

Theoretical Physics Group
Visvesvaraya National Institute of Technology, Nagpur

1 Schmidt Decomposition

Schmidt Decomposition is a very Important tool in quantum information and quantum computing . The Schmidt decomposition (named after its originator, Erhard Schmidt) refers to a particular way of expressing a vector in the tensor product of two inner product spaces. It has numerous applications in quantum information theory, for example, in entanglement characterisation and state purification.
Let us consider a composite state $|\psi\rangle_{AB}$ with two subsystems A and B, with basis $|i\rangle$ and $|j\rangle$ in the Hilbert spaces $\mathcal{H}_A$ (dimension N) and $\mathcal{H}_B$ (dimension M), respectively. Then the state $|\psi\rangle_{AB}$ of the composite system AB can be written as:

$$|\psi_{AB}\rangle = \sum_{i=1}^{N}\sum_{\alpha=1}^{M} a_{ij}\,|i\rangle\,|j\rangle\,, \tag{1}$$

where $a_{i\alpha}$ is an $N \times M$ matrix, called the coefficient matrix.

$$\begin{aligned}\rho_{AB} = |\psi_{AB}\rangle\langle\psi_{AB}| &= (\sum_{i=1}^{N}\sum_{j=1}^{M} a_{ij}|i\rangle \otimes |j\rangle)(\sum_{k=1}^{N}\sum_{l=1}^{M} a^*_{kl}\langle k| \otimes \langle l|) \\ &= \sum_{i,j,k,l} a_{ij}a^*_{kl}|i\rangle\langle k| \otimes |j\rangle\langle l| \end{aligned} \tag{2}$$

Now, let us take the partial trace with respect to the B subsystem

$$\begin{aligned}\rho_A &= tr_B(|\psi_{AB}\rangle\langle\psi_{AB}|) \\ &= \sum_{p}\sum_{i,j,k,l} a_{ij}a^*_{kl}|i\rangle\langle k| \otimes \langle p|j\rangle\langle l|p\rangle \\ &= \sum_{p}\sum_{i,k} a_{ip}a^*_{kp}|i\rangle\langle k| \\ \rho_A &= \sum_{i,k}\sum_{p} a_{ip}a^*_{kp}|i\rangle\langle k| \end{aligned} \tag{3}$$

$$\rho_A = AA^\dagger \tag{4}$$

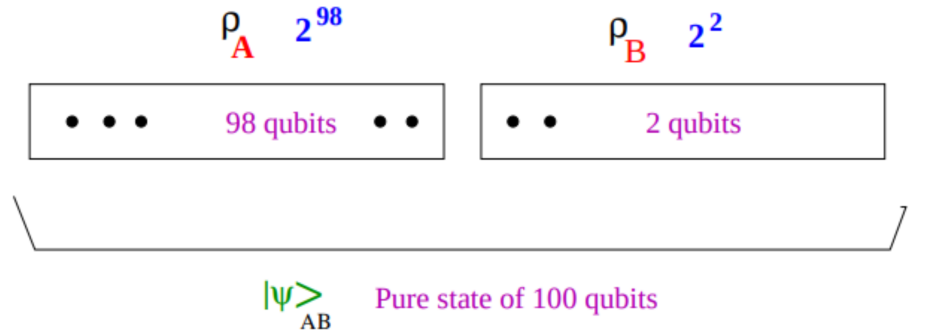
Similarly, one can easily calculate $\rho_B = A^\dagger A$.
Now, from Singular Value Decomposition, we can write for any matrix A, with U and V being unitary matrices of appropriate dimensions,

$$A = UDV^\dagger.$$

So,

$$\begin{aligned} AA^\dagger &= UDV^\dagger VDU^\dagger \\ AA^\dagger &= UD^2U^\dagger \end{aligned} \tag{5}$$

So, $\sqrt{\lambda_i}$ are the eigenvalues of $AA^\dagger$.
Similarly, $A^\dagger A = VD^2V^\dagger$.
So, $\sqrt{\lambda_i}$ are the eigenvalues of $A^\dagger A$.
The Schmidt decomposition can be understood using a pictorial representation as,



$$|\psi\rangle_{AB} = \underbrace{\sum_i^{2^{100}} c_{AB}|\alpha_i\rangle_A|\beta_i\rangle_B}_{2^{100}\ terms} = \underbrace{\sum_i^{2^2} \sqrt{\lambda_i}|\psi_{98}\rangle_i|\psi_2\rangle_i}_{Only\ 2^2\ terms}$$

Figure 1: Schmidt Decompostion

So if we have a 100 qubits state and we divide it into two subsystems one consisting 98 qubits and another consisting 2 qubits, then we do not need to calculate all the 2^{98} eigen values of the first subsystem because the eigenvalues are common to both the subsystems, so we can calculate them simply using 2 qubit state.

Because of this property, we can write any pure state as ,

$$|\psi\rangle_{AB} = \sum_i \lambda_i |i\rangle_A |i\rangle_B , \tag{6}$$

where λ_i are non-negative real numbers, satisfying $\sum_i \lambda_i^2 = 1$ known as Schmidt co-effecients. The bases $|i\rangle_A$ and $|i\rangle_B$ are called Schmidt bases for A and B, respectively, and the number of non-zero values of λ_i is called the Schmidt number of the state $|\psi\rangle_{AB}$. Now we aim to find the Schmidt coefficients and the orthonormal basis for A and B subsystems of any general state $|\psi\rangle_{AB}$. The Fortran program for this is as below,

```
program schmidtdec
    implicit none
    integer:: i, j , N, k, s, num, nt, s1, M, d
    integer:: r, q, bool, flag
    real*8, allocatable ::  b(:),res1(:,:), res2(:,:)
    character(len=:), allocatable :: state
    integer, allocatable :: state_matrix(:,:), site1(:), site2(:)
    double precision, allocatable ::psi(:,:), vin(:), rdm1(:,:), rdm2(:,:), W1
        (:), WORK1(:)
```

```
    double precision, allocatable ::U(:,:), V(:,:), E(:), F(:) , W2(:), WORK2
        (:)
    real*8, allocatable ::Ia(:,:), Ib(:,:), result
    double precision, allocatable :: phi(:,:), phi1(:,:)
    character*1 :: JOBZ, UPLO
    integer ::  LDA , LWORK , INFO
    ! Define the vectors for '0' and '1'
    integer, parameter :: zero_vector(2) = [1, 0]
    integer, parameter :: one_vector(2) = [0, 1]
    ! User Input total number of bits
    print*, "Enter total no of qubits: "
    read*, s
    ! Enter total number of bits:
    !s = 4
    num = 2**s
    ! User Input number of linearly independent terms
    print*, "Enter the total number of terms: "
    read*, nt
    ! Enter total number of terms:
    !nt = 2
    ! Allocate arrays after determining the value of num
    allocate(b(nt))
    allocate(double precision :: psi(num, 1))
    allocate(character(len=s) :: state)
    allocate(state_matrix(2**s, 1))
    ! Cacluating the state matrix from the lineraly independent terms
    psi = 0.0d0
    do i = 1, nt
        print *, "Enter the coefficient of", i ," term : "
        read *, b(i)
        print *, "Enter the ", i ," term : "
        read *, state

        ! Initialize the state matrix with the first vector
        if (state(1:1) == '1') then
            state_matrix(1, 1) = one_vector(1)
            state_matrix(2, 1) = one_vector(2)
        else
            state_matrix(1, 1) = zero_vector(1)
            state_matrix(2, 1) = zero_vector(2)
        end if

        ! Compute the tensor product iteratively
        do j = 2, s
            call genstate(state_matrix, state(j:j))
        end do
        psi = psi + b(i) * state_matrix
    end do
    print*, "Enter the number of qubits not to be traced out : "
    read*, s1
    !s1 = 2
```

```
59      ! Allocating matrices for PTRVR & DSYEV
60      N = 2**s1
61      LDA = N
62      LWORK = 3*N - 1
63      d = s - s1
64      allocate(site1(0:s1-1))
65      allocate(site2(0:d-1))
66      allocate(vin(0:2**s-1))
67      allocate(rdm1(0:2**s1-1,0:2**s1-1))
68      allocate(rdm2(0:2**d-1,0:2**d-1))
69      allocate(W1(0:2**s1-1))
70      allocate(WORK1(0:LWORK))
71      allocate(E(0:2**s1-1))
72      allocate(F(0:2**d-1))
73      allocate(V(0:2**d-1,0:2**d-1))
74      allocate(U(0:2**s1-1,0:2**s1-1))
75      allocate(res1(0:2**s1-1,1))
76      allocate(res2(0:2**d-1,1))
77      allocate(Ia(0:2**s1-1,1))
78      allocate(Ib(0:2**d-1,1))
79      do i = 0, s1-1
80          print*, "Enter the site indeces of qubits not to be traced out :"
81          read*, k
82          site1(i) = k - 1
83      end do
84      do i = 0, num - 1
85          vin(i) = psi(i+1,1)
86          write(90,*) i, vin(i)
87      end do
88      call PTRVR(s, s1, site1, vin, rdm1)
89      write(90,*) rdm1
90      U = rdm1
91      ! Define parameters for DSYEV
92      JOBZ = 'V'  ! Compute eigenvalues and eigenvectors
93      UPLO = 'L'  ! Upper triangular part of A is stored
94      ! Call DSYEV to compute eigenvalues and eigenvectors
95      call DSYEV(JOBZ, UPLO, N, U, LDA, W1, WORK1, LWORK, INFO)
96      do i = 0,2**s1-1
97          E(i) = sqrt(W1(i))
98          write(90,*) i, E(i)
99          do j = 0,2**s1-1
100             write(90,*) j,i, U(j,i)
101         end do
102     end do
103     do i = 0, d -1
104         print*, "Enter the site indeces of qubits not to be traced out :"
105         read*, k
106         site2(i) = k - 1
107     end do
108     call PTRVR(s, d, site2, vin, rdm2)
109     write(90,*) rdm2
```

```
    V = rdm2
    M = 2**d
    allocate(W2(0:2**d-1))
    allocate(WORK2(0:3*M -1))
    ! Define parameters for DSYEV
    JOBZ = 'V'  ! Compute eigenvalues and eigenvectors
    UPLO = 'L'  ! Upper triangular part of A is stored

    ! Call DSYEV to compute eigenvalues and eigenvectors
    call DSYEV(JOBZ, UPLO, M, V, M, W2, WORK2, 3*M -1, INFO)
    do i = 0,2**d -1
        F(i) = sqrt(W2(i))
        write(90,*) i, F(i)
        do j = 0,2**d -1
            write(90,*) j,i,V(j,i)
        end do
    end do

    allocate(phi(0:2**s-1, 1))
    allocate(phi1(0:2**s-1,1))
    phi = 0.0d0
    do i = 0, 2**s1 - 1
        do j = 0, 2**d -1
                if (E(i) == F(j) .AND. E(i) /= 0 ) then
                    Ia(:,1) = U(:,i)
                    Ib(:,1) = V(:,j)
                    ! print*, Ia
                    ! print*, Ib
                    flag = 0.0d0
                    bool = 0.0d0
                    call MMULMR(rdm1, Ia, 2**s1, 2**s1, 1, res1)
                    call MMULMR(rdm2, Ib, 2**d, 2**d, 1, res2)
                    ! print*, res1
                    ! print*, res2
                    do r = 0, 2**s1 - 1
                        if (res1(r,1) - E(i)**2 *U(r,i) /= 0) then
                            flag = flag + 1
                        end if
                    end do
                    do q = 0, 2**d - 1
                        if (res2(q,1) - F(j)**2 *V(q,j) /= 0) then
                        bool = bool + 1
                        end if
                    end do
                    if (flag == 0 .AND. bool == 0 ) then
                        call tensor_product(U(:,i), V(:,j), phi1(:,1), s1, s)
                        !print*, phi1
                        phi = phi + E(i) * phi1
                    end if
                end if
        end do
```

```
161     end do
162
163     ! do i = 0, 2**s - 1
164     !     write(90,*) phi(i,1)
165     ! end do
166
167     do i = 0, 2**s -1
168         result = result + psi(i+1,1) * phi(i,1)
169     end do
170     print*, "<phi|psi> = ", result
171     deallocate(psi)
172     deallocate(state)
173     deallocate(state_matrix)
174 contains
175     ! Subroutine to compute the tensor product
176     subroutine genstate(matrix, bit)
177         integer, intent(inout) :: matrix(:,:)
178         character(len=1), intent(in) :: bit
179         integer :: current_rows, new_rows, l
180         integer, allocatable :: temp_matrix(:,:)
181         ! Determine current number of rows in the matrix
182         current_rows = size(matrix, 1)
183         new_rows = current_rows * 2
184         ! Allocate a new matrix for the tensor product
185         allocate(temp_matrix(new_rows, 1))
186
187         ! Fill the new tensor product matrix
188         if (bit == '1') then
189             do l = 1, current_rows
190                 temp_matrix(2*l-1, 1) = matrix(l, 1) * 0  ! Corresponds to [0]
191                 temp_matrix(2*l, 1) = matrix(l, 1) * 1      ! Corresponds to
                        [1]
192             end do
193         else
194             do l = 1, current_rows
195                 temp_matrix(2*l-1, 1) = matrix(l, 1) * 1  ! Corresponds to [1]
196                 temp_matrix(2*l, 1) = matrix(l, 1) * 0      ! Corresponds to
                        [0]
197             end do
198         end if
199         ! Update the original matrix with the new tensor product
200         matrix = temp_matrix
201         deallocate(temp_matrix)
202     end subroutine genstate
203
204     subroutine PTRVR(s,s1,site,vin,rdm)
205         implicit none
206         integer::s, s1,s2
207         real*8::trace
208         real*8,dimension(0:2**s-1)::vin
209         real*8,dimension(0:2**s1-1,0:2**s1-1)::rdm
```

```
210         integer::ii,i1,a,i0,ia,j1,oo,k1,x1,y1,j,t1,t2,z1,i
211         integer,dimension(0:s1-1)::site,site2
212         integer,dimension(0:s1-1)::bin
213         integer,dimension(0:(2**(s-s1))*(2**s1)-1)::ind
214         s2=s-s1
215         x1=0
216         y1=0
217         ind=0
218         do j1=2**s1-1,0,-1
219             do ii=0,2**s-1
220                 a=ii
221                 do i1=0,s1-1,1
222                     i0=site(i1)
223                     call DTOBONEBIT(a,ia,i0,s)
224                     site2(i1)=ia
225                 enddo
226                 call DTOB(j1,bin,s1)
227                 oo=1
228                 do k1=0,s1-1,1
229                     oo=oo*(bin(k1)-site2(k1))
230                 end do
231                 if(abs(oo)==1) then
232                     ind(x1)=ii
233                     x1=x1+1
234                 end if
235             end do
236         end do
237         rdm=0.0d0
238         do t1=0,2**s1-1,1
239             do t2=0,2**s1-1,1
240                 do z1=0,2**s2-1,1
241                     rdm(t1,t2)=rdm(t1,t2)+vin(ind((2**s2)*t1+z1)) &
242                     *vin(ind((2**s2)*t2+z1))
243                 enddo
244             end do
245         end do
246     end subroutine
247
248     subroutine tensor_product(Ia, Ib, phi1, s1,s)
249         implicit none
250         integer :: i, j, k, l, s1, s
251         double precision, dimension(0:2**s1-1,1) :: Ia
252         double precision, dimension(0:2**(s-s1)-1,1) :: Ib
253         double precision, dimension(0:2**s-1,1) :: phi1
254
255         ! Initialize the output matrix C to zero
256         phi1 = 0.0d0
257
258         !print*, Ia, Ib
259
260         ! Compute the tensor product
```

```
261         do i = 0, 2**s1-1
262             do k = 0, 2**(s-s1)-1
263                 phi1((i)*2**(s-s1) + k, 1) = Ia(i, 1) * Ib(k, 1)
264             end do
265         end do
266     end subroutine tensor_product
267
268     subroutine DTOB(m,tt,s)
269         implicit none
270         integer::s
271         integer,dimension(0:s-1)::tt
272         integer::m,k,a2
273         tt=0
274         a2=m
275         do k = 0,s-1,1
276             tt(s-k-1) = mod(a2,2)
277             a2 = a2/2
278             if (a2== 0) then
279                 exit
280             end if
281         end do
282     end subroutine
283
284     subroutine DTOBONEBIT(m,ia,i0,s)
285         implicit none
286         integer*4::s
287         integer,dimension(0:s-1)::tt
288         integer::m,ia,i0
289         call DTOB(m,tt,s)
290         ia=tt(i0)
291     end subroutine
292
293     subroutine MMULMR(A,B,m,n,p,C)
294         implicit none
295         integer::n,m,l,i,j,k,p
296         real*8::A(0:m-1,0:n-1),B(0:n-1,0:p-1),C(0:m-1,0:p-1),temp
297         ! print*, A
298         ! print*, B
299         do i=0,m-1,1
300             do j=0,p-1,1
301                 temp = 0.0d0
302                 do k=0,n-1,1
303                     temp=temp+A(i,k)*B(k,j)
304                 end do
305             C(i,j) = temp
306             end do
307         end do
308     end subroutine
309 end program schmidtdec
```

This is the general program for calculating the Schmidt coefficients and the orthonormal basis for any

pure state of any general qubit. One can understand this code using the flow chart given below,

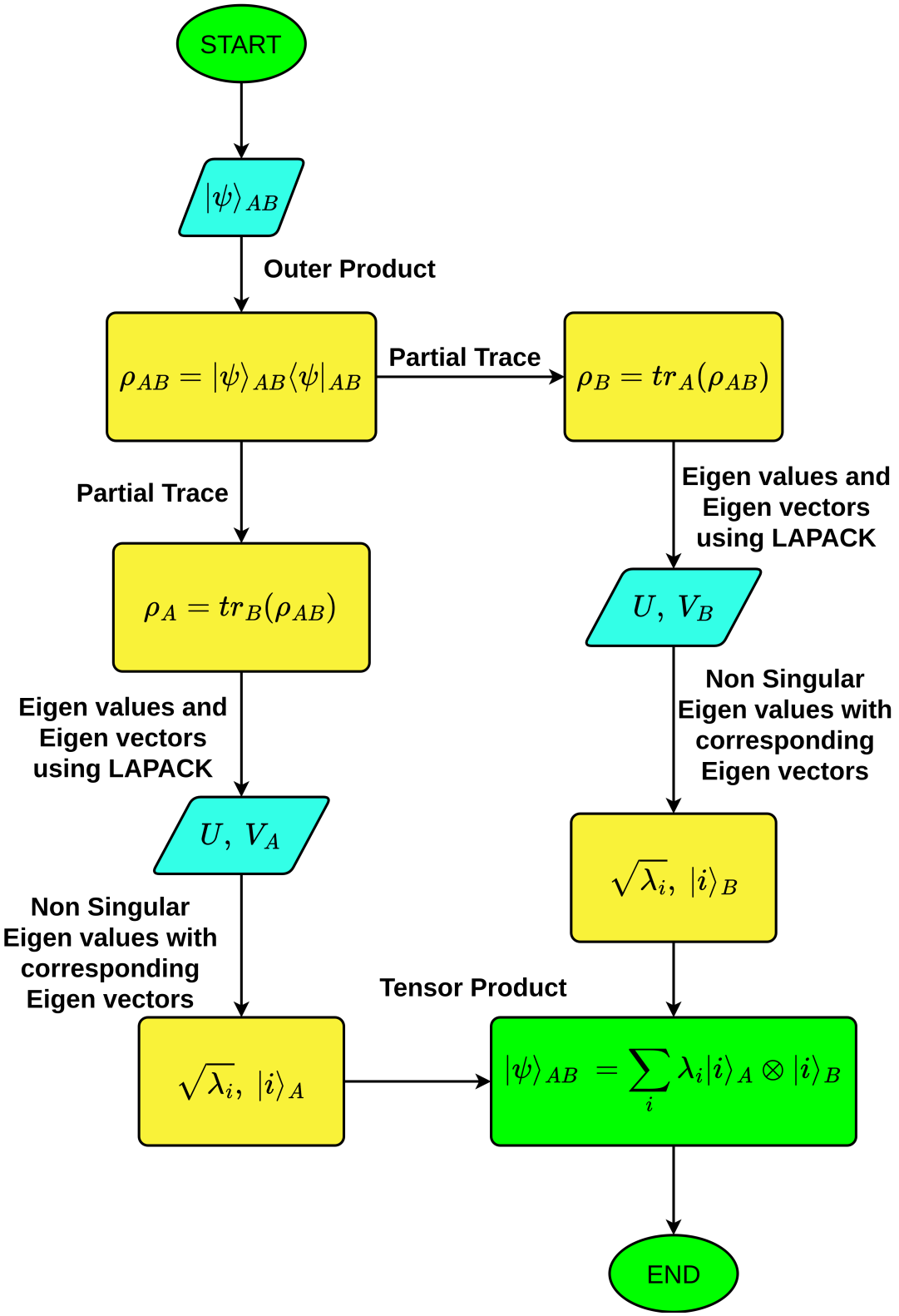


Figure 2: Flowchart for the Schmidt decomposition procedure