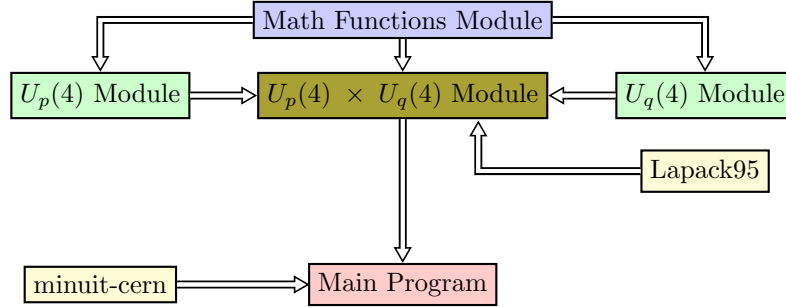


$$U_p(4) \times U_q(4)$$

Jamil KR

April 20, 2020



0.1 Math Functions Module \rightarrow MOD_matfun.f90

- Functions:

- `p_symbol(a,b)` = $(a)_s = a(a+1)\dots(a+s-1)$
- `factorial(n)` = $n!$
- `delta_function(a,b,c)` = $\Delta(abc)$
- `wigner_6j(j1,j2,j3,l1,l2,l3)` = $\left\{ \begin{matrix} j_1 & j_2 & j_3 \\ l_1 & l_2 & l_3 \end{matrix} \right\}_1$
- `wigner_9j(j1,j2,j3,l1,l2,l3,k1,k2,k3)` = $\left\{ \begin{matrix} j_1 & j_2 & j_3 \\ l_1 & l_2 & l_3 \\ k_1 & k_2 & k_3 \end{matrix} \right\}_2$

0.2 $U_p(4)$ Module \rightarrow MOD_Up4.f90

Hamiltonian:

$$\hat{H}_{U_p(4)} = \beta \hat{C}_2[so_p(4)] + \gamma \hat{C}_2[so_p(3)] + \gamma_2 \left[\hat{C}_2[so_p(3)] \right]^2 + \kappa \hat{C}_2[so_p(4)] \hat{C}_2[so_p(3)] \quad (1)$$

- Global definitions:

- Npval: $U(4)$ Totally symmetric representation.

- Functions:

- Function: `RME_Casimir_S0p4`
- Function: `RME_Casimir_S0p3`
- Function: `RME_Qp2`
- Function: `RME_np`

¹Definition of the book *Nuclear Shell Theory* of Amos de-Shalit and Igal Talmi

²Integers only

0.3 $U_q(4)$ Module \rightarrow MOD_Uq4.f90

Hamiltonian:

$$\hat{H}_{U_q(4)} = a \hat{C}_1 [u_q(3)] + b \hat{C}_2 [u_q(3)] + c \hat{C}_2 [so_q(3)] + d \hat{C}_2 [so_q(4)] \quad (2)$$

- Global definitions:
 - Nqval: $U(4)$ Totally symmetric representation.
- Functions:
 - Function: RME_Casimir_Uq3
 - Function: RME_Casimir_S0q3
 - Function: RME_Casimir_S0q4
 - Function: RME_Qq2
 - Function: RME_Dq-prima

0.4 $U_p(4) \times U_q(4)$ Module \rightarrow MOD_Up_x_Uq.f90

- Global definitions:
 - basis_para(1:5, no of para-states): Integers. Para-states
 - basis_ortho(1:5, no of ortho-states): Integers. Ortho-states
 - dim_para(1: Λ_{\max}): Integers. Para-dim blocks
 - dim_ortho(1: Λ_{\max}): Integers. Ortho-dim blocks
 - ijk_para(1: Λ_{\max}): Integers. Pseudo-pointer
 - ijk_ortho(1: Λ_{\max}): Integers. Pseudo-pointer
 - lambda_max: Integer. Maximum value of Λ
 - Type exp_point: exp_point%ist, exp_point%i_pos, exp_point%fst, exp_point%f_pos, exp_point%energy and exp_point%intensity
 - total_exp: Integer. Number of experimental data.
 - exp_data: Type(exp_point)
 - Type matrix: matrix(Λ)[SYM]
 - Ham(1: Λ_{\max}): Type(matrix). Hamiltonian para/ortho matrices
 - S0q4(1: Λ_{\max}): Type(matrix). $\hat{C}_2 [SO_q(4)]$ para/ortho matrices
 - QpQq(1: Λ_{\max}): Type(matrix). $\left[\hat{Q}_p^{(2)} \times \hat{Q}_q^{(2)} \right]^{(0)}$ para/ortho matrices
 - QpQqW(1: Λ_{\max}): Type(matrix). $\left[\hat{Q}_p^{(2)} \times \hat{Q}_q^{(2)} \right]^{(0)} \hat{C}_2 [SO_p(4)] + c.c.$ para/ortho matrices

- Functions:
 - Subroutine: `dimension_po`
 - Subroutine: `build_basis_po`
 - Subroutine: `initialize_position_index`
 - Function: `RME_Qp_x_Qq_0`
 - Function: `RME_Ip_x_S0q4`
 - Function: `RME_QpQqS0p4`
 - Subroutine: `build_Up_x_Uq_matrix`
 - Function: `pretty_braket`
 - Subroutine: `read_expdat`
 - Function: `exp_lines`
 - Subroutine: `build_ham`
 - Function: `find_pos`
 - Subroutine: `eigensystem`
 - Function: `assig_state`
 - Function: `chi2`
 - Function: `FCN`
 - Function: `RME_np_x_Dq_1`
 - Function: `RME_Qp2_x_Dq_1`

0.4.1 Building the basis

We have a block-diagonalizable system dividing the problem in para ($\text{mod}(J, 2) = 0$) and ortho ($\text{mod}(J, 2) = 1$) cases, and separating by different Λ . The states will be stored in the same array sorted by Λ . The dimension of each block will be saved in $\text{dim_}[SYM](\Lambda = 0), \dots, \text{dim_}[SYM](\Lambda = \Lambda_{\max})$.

$$\left. \left[\begin{array}{c} |\psi_1^{\Lambda=0}\rangle \\ \vdots \\ |\psi_{\text{dim_}[SYM](\Lambda=0)}^{\Lambda=0}\rangle \end{array} \right] \right\} \rightarrow 1:\text{dim_}[SYM](\Lambda = 0)$$

$$\vdots$$

$$\left. \left[\begin{array}{c} |\psi_1^{\Lambda=\Lambda_{\max}}\rangle \\ \vdots \\ |\psi_{\text{dim_}[SYM](\Lambda=\Lambda_{\max})}^{\Lambda=\Lambda_{\max}}\rangle \end{array} \right] \right\} \rightarrow 1:\text{dim_}[SYM](\Lambda = \Lambda_{\max})$$
(3)

Therefore, the basis array is as follow:

$$\mathbf{basis_}[SYM] = \left[\left| \psi_1^{\Lambda=0} \right\rangle, \dots, \psi_{\dim_ [SYM] (\Lambda=0)}^{\Lambda=0}, \dots, \left| \psi_1^{\Lambda=\Lambda_{\max}} \right\rangle, \dots, \psi_{\dim_ [SYM] (\Lambda=\Lambda_{\max})}^{\Lambda=\Lambda_{\max}} \right] \quad (4)$$

Each element of the the basis must cotain information about quantum numbers:

$$\left| \psi_i^{\Lambda} \right\rangle = (w \ J; \ n \ L; \ \Lambda \in [|J-L|, J+L]) \quad (5)$$

Dimension: `dimension_po`

Fortran90 subroutine.

Inputs:

- Npval. Integer
- Nqval. Integer
- Λ_{\max} . Integer

Outputs:

- dim_para. Integer, dimension(0: Λ_{\max})
- dim_ortho. Integer, dimension(0: Λ_{\max})

Basis: `build_basis_po`

Inputs:

- Npval. Integer
- Nqval. Integer
- Λ_{\max} . Integer
- total_para. Integer, total dimension of para basis
- total_ortho. Integer, total dimension of ortho basis
- dim_para. Integer, dimension(0: Λ_{\max})
- dim_ortho. Integer, dimension(0: Λ_{\max})

Outputs:

- basis_para. Integer, dimension $\left(1 : 5, 1 : \sum_{\Lambda=0}^{\Lambda_{\max}} \text{dim_para}(\Lambda)\right)$
- basis_ortho. Integer, dimension $\left(1 : 5, 1 : \sum_{\Lambda=0}^{\Lambda_{\max}} \text{dim_ortho}(\Lambda)\right)$
 - basis_[SYM](1,:) $\rightarrow w$
 - basis_[SYM](2,:) $\rightarrow J$
 - basis_[SYM](3,:) $\rightarrow n$
 - basis_[SYM](4,:) $\rightarrow L$
 - basis_[SYM](5,:) $\rightarrow \Lambda$

Para-states

```

LOOP:  $J = 0, 2, \dots, N_p - \text{mod}(N_p, 2)$ 
  LOOP:  $L = 0, 1, \dots, N_q$ 
    CONDITIONAL:  $|J - L| \leq \Lambda_{\max}$  to continue, else go to next  $L$ 
    LOOP:  $\Lambda = |J - L|, |J - L| + 1, \dots, \min(\Lambda_{\max}, J + L)$ 
      LOOP:  $w = N_p - \text{mod}(N_p, 2), N_p - \text{mod}(N_p, 2) - 2, \dots, J$ 
        LOOP:  $n = L, L + 2, \dots, N_q$ 
          basis_para(1, dim_para( $\Lambda$ )) =  $w$ 
          basis_para(2, dim_para( $\Lambda$ )) =  $J$ 
          basis_para(3, dim_para( $\Lambda$ )) =  $n$ 
          basis_para(4, dim_para( $\Lambda$ )) =  $L$ 
          basis_para(5, dim_para( $\Lambda$ )) =  $\Lambda$ 

```

(6)

Ortho-states

```

LOOP:  $J = 1, 3, \dots, N_p - (1 - \text{mod}(N_p, 2))$ 
  LOOP:  $L = 0, 1, \dots, N_q$ 
    CONDITIONAL:  $|J - L| \leq \Lambda_{\max}$  to continue, else go to next  $L$ 
    LOOP:  $\Lambda = |J - L|, |J - L| + 1, \dots, \min(\Lambda_{\max}, J + L)$ 
      LOOP:  $w = N_p - (1 - \text{mod}(N_p, 2)), N_p - (1 - \text{mod}(N_p, 2)) - 2, \dots, J$ 
        LOOP:  $n = L, L + 2, \dots, N_q$ 
          basis_ortho(1, dim_ortho( $\Lambda$ )) =  $w$ 
          basis_ortho(2, dim_ortho( $\Lambda$ )) =  $J$ 
          basis_ortho(3, dim_ortho( $\Lambda$ )) =  $n$ 
          basis_ortho(4, dim_ortho( $\Lambda$ )) =  $L$ 
          basis_ortho(5, dim_ortho( $\Lambda$ )) =  $\Lambda$ 

```

(7)

Pseudo-pointer: initialize_position_index

```

subroutine initialize_position_index(ijk,partial_dim,lambda_max)
!
! Inputs:
! o) partial_dim: Integer array dimension 0:lambda_max
! o) lambda_max
!
! Output:
! o) ijk: Integer array dimension 0:lambda_max
!
! This functions initializes the initial integer "pointer" ijk(0:lambda_max),
! so that
! ijk(0) = 1
! ijk(1) = position where lambda=1 block starts
! ...
! ijk(lambda_max) = position where lambda=lambda_max block starts
!
! This function is going to be of vital importance during the program's develop-
ment
!

```

Building all matrices: build_Up_x_Uq_matrix

This subroutine builds the matrices for the operators of $U_p(4) \times U_q(4)$ using para/ortho basis without mixing different Λ .

```

subroutine build_Up_x_Uq_matrix(basis,matrix,RME_fun,iprint)
!
! This function build the para / ortho matrices using the given basis.
! This procedure can be used to build Up4 x Uq4 operators' matrices.
!
! INPUTs:
! o) basis: para/ortho basis
! o) matrix: square matrix len(basis) x len(basis)
! o) RME_fun: function with w1,j1,n1,l1,lam1,w2,j2,n2,l2,lam2 dependences.
! o) iprint: printing control
!
! OUTPUT:
! o) matrix
!
! All position corresponding to lam1 /= lam2 will be ZERO!
!

```

The function **RME_fun** must deppend on $(\omega_1, J_1, n_1, L_1, \Lambda_1, \omega_2, J_2, n_2, L_2, \Lambda_2)$.

Bracket notation output: pretty_braket

Useless gadget ... but very nice.

```
function pretty_braket(w,j,n,l,lam,bk,Np,Nq)
!  
! INPUTs:  
! o) Np(opt), Nq(opt), w, j, n, l, lam: Quantum numbers  
! o) bk: one character = b (bra) or k (ket)  
!  
! OUTPUT:  
! o) pretty_braket: character type  
!
```