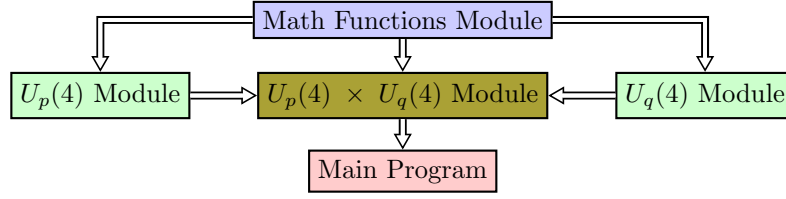


$$U_p(4) \times U_q(4)$$

Jamil KR

February 11, 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)` = $\left\{ \begin{matrix} j_1 & j_2 & j_3 \\ l_1 & l_2 & l_3 \end{matrix} \right\}_1$

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

Hamiltonian:

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

- Global definitions:

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

- Functions:

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

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

Hamiltonian:

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

- Global definitions:

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

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

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

- Global definitions:
 -
- Functions:
 - Subroutine: dimension_po
 - Subroutine: build_basis_po
 - Subroutine: initialize_position_index
 - Function: build_Qp_x_Qq_0_matrix

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 $\text{dim_}[\text{SYM}] (\Lambda = 0), \dots, \text{dim_}[\text{SYM}] (\Lambda = \Lambda_{\max})$.

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

$$\vdots$$

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

Therefore, the basis array is as follow:

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

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

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

Dimension: `dimension_po`

Fortran90 subroutine.
Inputs:

- `Npval`. Integer
- `Nqval`. Integer
- `Λmax`. Integer

Outputs:

- `dim_para`. Integer, `dimension(0:Λmax)`
- `dim_ortho`. Integer, `dimension(0:Λmax)`

Basis: `build_basis_po`

Inputs:

- `Npval`. Integer
- `Nqval`. Integer
- `Λmax`. Integer
- `total_para`. Integer, total dimension of para basis
- `total_ortho`. Integer, total dimension of ortho basis
- `dim_para`. Integer, `dimension(0:Λmax)`
- `dim_ortho`. Integer, `dimension(0:Λmax)`

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  
!
```