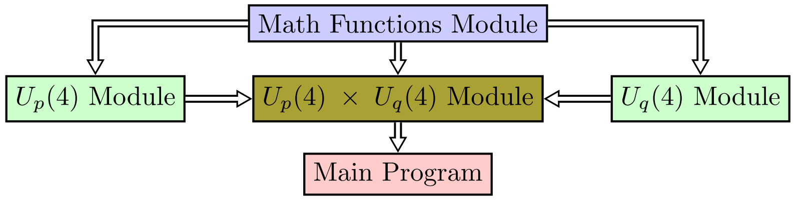
$U_p(4) \times U_q(4)$

Jamil KR

February 10, 2020



0.1 Math Functions Module → `MOD_matfun.f90`

- Functions:
  - `p_symbol`(a,b) = $(a)_s = a(a+1)...(a+s-1)$
  - `factorial`(n) = $n!$
  - `delta_function`(a,b,c) = Δ(abc)
  - `wigner_6j`$(j_1, j_2, j_3, l_1, l_2, l_3) = \begin{Bmatrix} j_1 & j_2 & j_3 \\ l_1 & l_2 & l_3 \end{Bmatrix}$

0.2 $U_p(4)$ Module → `MOD_Up4.f90`

Hamiltonian:

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

- Global definitions:
  - Npval: $U(4)$ Totally symmetric representation.
- Functions:
  - Function: `RME_Casimir_SOp4`
  - Function: `RME_Casimir_SOp3`
  - Function: `RME_Qp2`

0.3 $U_q(4)$ Module → `MOD_Uq4.f90`

Hamiltonian:

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

- Global definitions:
  - Nqval: $U(4)$ Totally symmetric representation.

- Functions:
  - Function: `RME_Casimir_Uq3`
  - Function: `RME_Casimir_SOq3`
  - Function: `RME_Casimir_SOq4`
  - Function: `RME_Qq2`

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

- Global definitions:
  -
- Functions:
  - Subroutine: `dimension_po`
  - Subroutine: `build_basis_po`
  - Subroutine: `initialize_position_index`

0.4.1 Building the basis

We have a block-diagonalizable system dividing the problem in para ($\mathrm{mod}(J,2) = 0$) and ortho ($\mathrm{mod}(J,2) = 1$) 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 `dim_[SYM]`$(\Lambda = 0)$, ... , `dim_[SYM]`$(\Lambda = \Lambda_{\max})$.

$$\begin{array}{c} \left.\left[\begin{array}{c} \left|\psi_1^{\Lambda=0}\right\rangle \\ \vdots \\ \left|\psi_{\texttt{dim\_[SYM]}(\Lambda=0)}^{\Lambda=0}\right\rangle \end{array}\right]\right\} \to 1{:}\texttt{dim\_[SYM]}(\Lambda=0) \\ \vdots \\ \left.\left[\begin{array}{c} \left|\psi_1^{\Lambda=\Lambda_{\max}}\right\rangle \\ \vdots \\ \left|\psi_{\texttt{dim\_[SYM]}(\Lambda=\Lambda_{\max})}^{\Lambda=\Lambda_{\max}}\right\rangle \end{array}\right]\right\} \to 1{:}\texttt{dim\_[SYM]}(\Lambda=\Lambda_{\max}) \end{array} \tag{3}$$

Therefore, the basis array is as follow:

$$\texttt{basis\_[SYM]} = \left[\left|\psi_1^{\Lambda=0}\right\rangle, ..., \psi_{\texttt{dim\_[SYM]}(\Lambda=0)}^{\Lambda=0}, ..., \left|\psi_1^{\Lambda=\Lambda_{\max}}\right\rangle, ..., \psi_{\texttt{dim\_[SYM]}(\Lambda=\Lambda_{\max})}^{\Lambda=\Lambda_{\max}}\right] \tag{4}$$

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

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

Dimension: `dimension_po`

Fortran90 subroutine.

Inputs:

- Npval. Integer
- Nqval. Integer
- $\Lambda_{\max}$. Integer

Outputs:

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

Basis: `build_basis_po`

Inputs:

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

Outputs:

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

Para-states

$$
\begin{array}{l}
\text{LOOP: } J = 0, 2, ..., N_p - \mathrm{mod}(N_p, 2) \\
\quad \text{LOOP: } L = 0, 1, ..., N_q \\
\quad\quad \text{CONDITIONAL: } |J - L| \leq \Lambda_{max} \text{ to continue, else go to next } L \\
\quad\quad\quad \text{LOOP: } \Lambda = |J - L|, |J - L| + 1, ..., \min(\Lambda_{max}, J + L) \\
\quad\quad\quad\quad \text{LOOP: } w = 0, 2, 4, ...J \\
\quad\quad\quad\quad\quad \text{LOOP: } n = L, L + 2, ..., Nq \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_para}\,(1, \texttt{dim\_para}\,(\Lambda)) = w \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_para}\,(2, \texttt{dim\_para}\,(\Lambda)) = J \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_para}\,(3, \texttt{dim\_para}\,(\Lambda)) = n \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_para}\,(4, \texttt{dim\_para}\,(\Lambda)) = L \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_para}\,(5, \texttt{dim\_para}\,(\Lambda)) = \Lambda
\end{array}
\tag{6}
$$

Ortho-states

$$
\begin{array}{l}
\text{LOOP: } J = 1, 3, ..., N_p - (1 - \mathrm{mod}(N_p, 2)) \\
\quad \text{LOOP: } L = 0, 1, ..., N_q \\
\quad\quad \text{CONDITIONAL: } |J - L| \leq \Lambda_{max} \text{ to continue, else go to next } L \\
\quad\quad\quad \text{LOOP: } \Lambda = |J - L|, |J - L| + 1, ..., \min(\Lambda_{max}, J + L) \\
\quad\quad\quad\quad \text{LOOP: } w = 1, 3, 5, ...J \\
\quad\quad\quad\quad\quad \text{LOOP: } n = L, L + 2, ..., Nq \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_ortho}\,(1, \texttt{dim\_ortho}\,(\Lambda)) = w \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_ortho}\,(2, \texttt{dim\_ortho}\,(\Lambda)) = J \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_ortho}\,(3, \texttt{dim\_ortho}\,(\Lambda)) = n \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_ortho}\,(4, \texttt{dim\_ortho}\,(\Lambda)) = L \\
\quad\quad\quad\quad\quad\quad \texttt{basis\_ortho}\,(5, \texttt{dim\_ortho}\,(\Lambda)) = \Lambda
\end{array}
\tag{7}
$$

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 stats
!
! This function is going to be of vital importance during the program's development
!
```