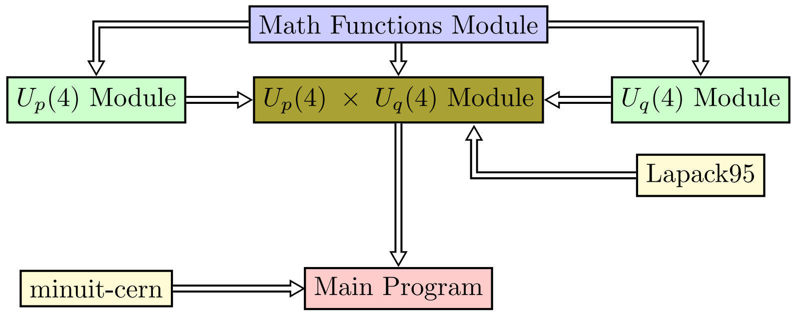
$U_p(4) \times U_q(4)$

Jamil KR

April 20, 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`$(j_1, j_2, j_3, l_1, l_2, l_3)$ $=$ $\begin{Bmatrix} j_1 & j_2 & j_3 \\ l_1 & l_2 & l_3 \end{Bmatrix}$ [1]
  - `wigner_9j`$(j_1, j_2, j_3, l_1, l_2, l_3, k_1, k_2, k_3)$ $=$ $\begin{Bmatrix} j_1 & j_2 & j_3 \\ l_1 & l_2 & l_3 \\ k_1 & k_2 & k_3 \end{Bmatrix}$ [2]

0.2 $U_p(4)$ Module → `MOD_Up4.f90`

Hamiltonian:

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

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

[1]Definition of the book *Nuclear Shell Theory* of Amos de-Shalit and Igal Talmi
[2]Integers only

0.3 $U_q(4)$ Module → `MOD_Uq4.f90`

Hamiltonian:

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

- 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:
  - `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:`$\Lambda_{\texttt{max}}$`)`: Integers. Para-dim blocks
  - `dim_ortho(1:`$\Lambda_{\texttt{max}}$`)`: Integers. Ortho-dim blocks
  - `ijk_para(1:`$\Lambda_{\texttt{max}}$`)`: Integers. Pseudo-pointer
  - `ijk_ortho(1:`$\Lambda_{\texttt{max}}$`)`: 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:`$\Lambda_{\texttt{max}}$`)`: Type(`matrix`). Hamiltonian para/ortho matrices
  - `SOq4(1:`$\Lambda_{\texttt{max}}$`)`: Type(`matrix`). $\hat{\mathcal{C}}_2\left[SO_q(4)\right]$ para/ortho matrices
  - `QpQq(1:`$\Lambda_{\texttt{max}}$`)`: Type(`matrix`). $\left[\hat{Q}_p^{(2)} \times \hat{Q}_q^{(2)}\right]^{(0)}$ para/ortho matrices
  - `QpQqW(1:`$\Lambda_{\texttt{max}}$`)`: Type(`matrix`). $\left[\hat{Q}_p^{(2)} \times \hat{Q}_q^{(2)}\right]^{(0)} \hat{\mathcal{C}}_2\left[SO_p(4)\right] + c.c.$ 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_SOq4`
- Function: `RME_QpQqSOp4`
- Subroutine: `build_Up_x_Uq_matrix`
- Function: `pretty_braket`
- Subroutine: `read_expdat`
- Function: `exp_lines`
- Function: `RME_sop4`
- Subroutine: `build_ham`
- Function: `find_pos`
- Subroutine: `eigensystem`
- Function: `assig_state`
- Function: `chi2`
- Function: `FCN`

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]`$(\Lambda = \Lambda_{\max})$.

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

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$\left(1:5,\ 1:\sum_{\Lambda=0}^{\Lambda_{\max}}\mathtt{dim_para}(\Lambda)\right)$
- basis_ortho. Integer, dimension$\left(1:5,\ 1:\sum_{\Lambda=0}^{\Lambda_{\max}}\mathtt{dim_ortho}(\Lambda)\right)$
  - 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

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

Ortho-states

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

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
!

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 matriz 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 $(\omega_1, J_1, n_1, L_1, \Lambda_1, \omega_2, J_2, n_2, L_2, \Lambda_2)$.

Braket 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
!
```