Wigner Functions in Modern Fortran

Oliver C. Gorton

This is a library of functions for computation of Wigner 3-j, 6-j and 9-j symbols using algebraic expressions in terms of factorials. Should be accurate to 10^{-10} relative error for values less than about j=20.

The code is available in this public [GitHub repository] (https://github.com/ogorton/wigner).

You can download this page as a pdf.

For an analysis of relative error compared to more modern methods, see arXiv:1504.08329 by H. T. Johansson and C. Forssen. A more accurate but slower method involves prime factorization of integers. In old Fortran, see work by Liqiang Wei: Computer Physics Communications 120 (1999) 222-230.

All integer arguments are 2j in order to accomadate half-integer arguments while taking advantage of faster integer-arithmetic. Invalid arguments return 0d0 and program continues.

Optionally, compile with OpenMP to accelerate table initialization.

List of real(kind=8) functions:

- `logfac(n)`
- `logdoublefac(n)`
- `triangle(two_j1, two_j2, two_j3)`
- `vector_couple(two_j1, two_m1, two_j2, two_m2, two_jc, two_mc)`
- `threej(two_j1, two_j2, two_j3, two_m1, two_m2, two_m3)`
- `threej_lookup(two_j1,two_j2,two_j3,two_l1,two_l2,two_l3)`
- `sixj(two_j1,two_j2,two_j3,two_l1,two_l2,two_l3)`
- `sixj_lookup(two_j1,two_j2,two_j3,two_l1,two_l2,two_l3)`
- `ninej(two_j1,two_j2,two_j3,two_j4,two_j5,two_j6,two_j7,two_j8,two_j9)`

List of subroutines:

- `threej_table_init(min2j, max2j)`
- `sixj_table_init(min2j, max2j)`

3-J and 6-J Symbols

Real function. Arguments of the function are twice those computed. For each of the following functions and routines, an equivalent one exists for the 'three'-J symbol.

```
function sixj(two_j1,two_j2,two_j3,two_l1,two_l2,two_l3) result(sj)
    ! Computes the wigner six-j symbol with arguments
    !    two_j1/2 two_j2/2 two_j3/2
    !    two_l1/2 two_l2/2 two_l3/2
    ! using explicit algebraic expressions from Edmonds (1955/7).
    implicit none
    integer :: j1,j2,j3,l1,l2,l3
    real(kind8) :: sj
```

Lookup table initialization. Optional arguments set the lower and upper limits of values stored in the table.

```
subroutine sixj_table_init(min2j, max2j)
    implicit none
    integer, optional :: min2j, max2j
```

Lookup table lookup-function. This function tries to lookup the requested symbols in the allocated table, otherwise it calls the `sixj` function.

```
function sixj_lookup(two_j1, two_j2, two_j3,&
                     two_l1, two_l2, two_l3) result(sj)

    implicit none
    integer :: two_j1,two_j2,two_j3,two_l1,two_l2,two_l3
    real(kind=8) :: sj
```

9-J Symbol

Real function. We don't include lookup table functions for the 9-J function.

```
function ninej(two_j1, two_j2, two_j3,&
               two_j4, two_j5, two_j6,&
               two_j7, two_j8, two_j9) result(nj)
    implicit none
    integer :: two_j1,two_j2,two_j3
    integer :: two_j4,two_j5,two_j6
    integer :: two_j7,two_j8,two_j9
    real(kind=8) :: nj
```

Compile and test

We include a test program which demonstrates how to implement the `wigner` functions and subroutines.

Compile the `test` program:

```
gfortran wigner.f90 wigner_test.f90 -o test
```

Run the test program:

```
./test
```

Expected output:

```
Initializing three-j symbol table...
 Table min. 2J:           0
 Table max. 2J:          12
Memory required (MB):     38.61
 Table has been saved to memory.
 Seconds to initialize:   7.48580024E-02
Initializing six-j symbol table...
 Table min. 2J:           0
 Table max. 2J:          12
Memory required (MB):     38.61
Table has been saved to memory.
Seconds to initialize:    0.5009
 Jx2=          0
 Jx2=          1
 Jx2=          2
 Jx2=          3
 Jx2=          4
 Jx2=          5
 Jx2=          6
 Jx2=          7
 Jx2=          8
 Jx2=          9
 Jx2=         10
 Jx2=         11
 Jx2=         12
 Example sixj value, sixj(1,3,5,1,1,3):   4.3643578047198470E-002
 Time:  0.473100990
```

Theory

We implement a standard set of functions and subroutines for computing the vector-coupling 3-j, 6-j, and 9-j symbols using the Racah alebraic expressions found in Edmonds.

For the 3-j symbol, we use the relation to the Clebsh-Gordon vector-coupling coefficients:

$$\begin{pmatrix} j_1 & j_2 & J \\ m_1 & m_1 & M \end{pmatrix} = (-1)^{j_1-j_2-M}(2J+1)^{-1/2} \\ (j_1 j_2 m_1 m_2 | j_1 j_2; J, -M).$$

The vector coupling coefficients are computed as:

$$(j_1 j_2 m_1 m_2 | j_1 j_2; J, M) = \delta(m_1 + m_1, m)(2J+1)^{1/2}\Delta(j_1 j_2 J) \\ \times [(j_1+m_1)(j_1-m_1)(j_2+m_2)(j_2-m_2)(J+M)(J-M)]^{1/2} \sum_z (-1)^z \frac{1}{f(z)},$$

where

$$f(z) = z!(j_1+j_2-J-z)!(j_1-m_2-z)! \\ \times (j_2+m_2-z)!(J-j_2+m_1+z)!(J-m_1-m_2+z)!,$$

and

$$\Delta(abc) = \left[\frac{(a+b-c)!(a-b+c)!(-a+b+c)!}{(a+b+c+1)!}\right]^{1/2}.$$

The sum over z is over all integers such that the factorials are well-defined (non-negative-integer arguments).

Similarly, for the 6-j symbols:

$$\begin{Bmatrix} j_1 & j_2 & j_3 \\ m_1 & m_1 & m_3 \end{Bmatrix} = \Delta(j_1 j_2 j_3)\Delta(j_1 m_2 m_3)\Delta(m_1 j_2 m_3) \\ \times \Delta(m_1 m_2 j_3) \sum_z (-1)^z \frac{(z+1)!}{g(z)},$$

with

$$g(z) = (\alpha - z)!(\beta - z)!(\gamma - z)! \\ \times (z-\delta)!(z-\epsilon)!(z-\zeta)!(z-\eta)!$$

$$\begin{aligned} \alpha &= j_1 + j_1 + m_1 + m_2 & \beta &= j_2 + j_3 + m_2 + m_3 \\ \gamma &= j_3 + j_1 + m_3 + m_1 & & \\ \delta &= j_1 + j_2 + j_3 & \epsilon &= j_1 + m_2 + m_3 \\ \zeta &= m_1 + j_2 + m_3 & \eta &= m_1 + m_2 + j_3. \end{aligned}$$

For the 9-j symbol, we use the relation to the 6-j symbol:

$$\begin{Bmatrix} j_1 & j_2 & j_3 \\ j_4 & j_5 & j_6 \\ j_7 & j_8 & j_9 \end{Bmatrix} = \sum_k (-1)^{2k}(2k+1) \\ \times \begin{Bmatrix} j_1 & j_4 & j_7 \\ j_8 & j_9 & z \end{Bmatrix} \begin{Bmatrix} j_2 & j_5 & j_8 \\ j_4 & z & j_6 \end{Bmatrix} \begin{Bmatrix} j_3 & j_6 & j_9 \\ z & j_1 & j_2 \end{Bmatrix}.$$

The 6-j symbols used to calculate the 9-j symbol are first taken from any tabulated values. Otherwise, they are computed as previously described.