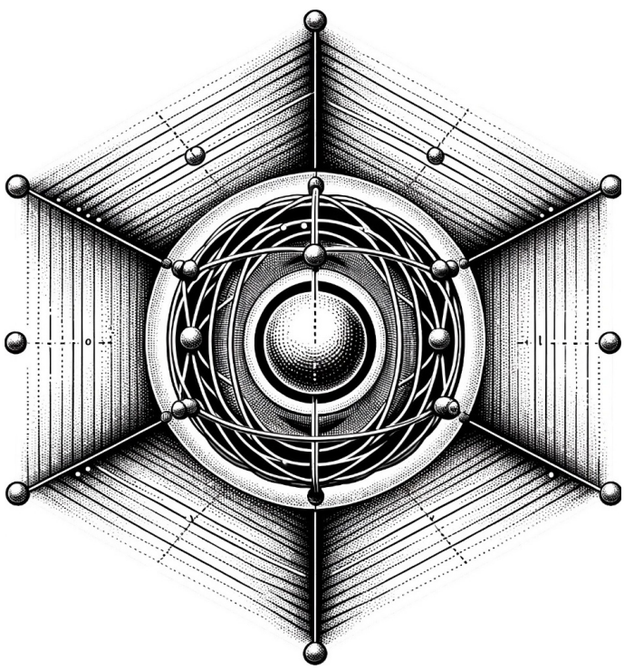

`qlanth`

version $|\alpha\rangle$

Juan David Lizarazo Ferro
& Christopher Dodson

Under the advisory of Dr. Rashid Zia

qlanth is a Mathematica package that can be used to estimate the level structure of lanthanide ions embedded in crystals. For this purpose it uses a single configuration description and a corresponding effective Hamiltonian. This Hamiltonian aims to describe the observed properties of ions embedded in solids in a picture that imagines them as free-ions but modified by the influence of the lattice in which they find themselves in.

This picture is one that developed and mostly matured in the second half of the last century by the efforts of Giulio Racah, Brian Judd, Hannah Crosswhite, Robert Cowan, Michael Reid, Bill Carnall, Clyde Morrison, Brian Wybourne, and Katherine Rajnak among others. The goal of this code is to provide a modern implementation of the calculations that resulted from their work, with the aim of fixing some small errors that might have been included at the time these calculations were made.

qlanth also includes data that might be of use to those interested in the single-configuration description of lanthanide ions, separate to their specific use in this code. These data include the coefficients of fractional parentage (as calculated by Velkov and parsed here), and reduced matrix elements for all the operators in the effective Hamiltonian. These are provided as standard Mathematica associations that should be simple to port elsewhere.

The included Mathematica notebook `qlanth.nb` has examples of the capabilities that this package offers, and the `/examples` folder includes a series of notebooks for most of the trivalent lanthanide ions in lanthanum fluoride. LaF3 is remarkable in that it was one of the systems in which a systematic study [Car+89] of all of the trivalent lanthanide ions were studied.

This code was originally authored by Christopher Dodson and Rashid Zia and has been modified and rewritten by David Lizarazo. It has benefited from conversations with Tharnier Puel at the University of Iowa.

Contents

1 The $|LSJM_J\rangle$ Basis

The basis used in **qlanth** is the $|LSJM_J\rangle$ basis. As such the basis vectors are common eigenvectors of the operators $\hat{L}^2$, $\hat{S}^2$, $\hat{J}^2$, and $\hat{J}_z$. The LS terms allowed in each configuration $\underline{\mathrm{f}}^n$ are obtained from tables that originate from the orginal work by Nielson and Koster [NK63]. In **qlanth** these are parsed from the file `B1F_ALL.TXT` which is part of the doctoral research of Dobromir Velkov [Vel00] in which he recomputed coefficients of fractional parentage under the advisory of Brian Judd.

One of the facts that have to be accounted for in a basis that uses L and S as quantum numbers, is that there might be several linearly independent path to couple the electron spin and orbital momenta to add up to given total L and total S. For this reason additional labels are necessary to distinguish between these different terms. The simplest way of doing this dates back to the tables of Nielson and Koster [NK63], and consists in assigning consecutive integers to degenerate LS terms, with no specific role given to them, except that of discriminating between different degenerate terms. It is also possible to add more useful labels that reflect additional symmetries that the f-electron wavefunctions find in the groups $\mathcal{SO}(7)$ and G_2.

The following are all the LS terms in the $\underline{\mathrm{f}}^n$ configurations. In the notation used the superscript index before the letter notes the spin multiplicity $2S+1$, the roman letter indicating the value of L in spectroscopic notation, and the final integer (if present) is the label that discriminates between several degenerate LS. This index we frequently label in the equations contained in this document with the greek letter α.

$\underline{\mathrm{f}}^0$
(1 LS term)

---

^1S

$\underline{\mathrm{f}}^1$
(1 LS term)

---

^2F

$\underline{\mathrm{f}}^2$
(7 LS terms)

---

^3P, ^3F, ^3H, ^1S, ^1D, ^1G, ^1I

$\underline{\mathrm{f}}^3$
(17 LS terms)

---

^4S, ^4D, ^4F, ^4G, ^4I, ^2P, $^2D1$, $^2D2$, $^2F1$, $^2F2$, $^2G1$, $^2G2$, $^2H1$, $^2H2$, ^2I, ^2K, ^2L

$\underline{\mathrm{f}}^4$
(47 LS terms)

---

^5S, ^5D, ^5F, ^5G, ^5I, $^3P1$, $^3P2$, $^3P3$, $^3D1$, $^3D2$, $^3F1$, $^3F2$, $^3F3$, $^3F4$, $^3G1$, $^3G2$, $^3G3$, $^3H1$, $^3H2$, $^3H3$, $^3H4$, $^3I1$, $^3I2$, $^3K1$, $^3K2$, ^3L, ^3M, $^1S1$, $^1S2$, $^1D1$, $^1D2$, $^1D3$, $^1D4$, ^1F, $^1G1$, $^1G2$, $^1G3$, $^1G4$, $^1H1$, $^1H2$, $^1I1$, $^1I2$, $^1I3$, ^1K, $^1L1$, $^1L2$, ^1N

$\underline{\mathrm{f}}^5$
(73 LS terms)

---

^6P, ^6F, ^6H, ^4S, $^4P1$, $^4P2$, $^4D1$, $^4D2$, $^4D3$, $^4F1$, $^4F2$, $^4F3$, $^4F4$, $^4G1$, $^4G2$, $^4G3$, $^4G4$, $^4H1$, $^4H2$, $^4H3$, $^4I1$, $^4I2$, $^4I3$, $^4K1$, $^4K2$, ^4L, ^4M, $^2P1$, $^2P2$, $^2P3$, $^2P4$, $^2D1$, $^2D2$, $^2D3$, $^2D4$, $^2D5$, $^2F1$, $^2F2$, $^2F3$, $^2F4$, $^2F5$, $^2F6$, $^2F7$, $^2G1$, $^2G2$, $^2G3$, $^2G4$, $^2G5$, $^2G6$, $^2H1$, $^2H2$, $^2H3$, $^2H4$, $^2H5$, $^2H6$, $^2H7$, $^2I1$, $^2I2$, $^2I3$, $^2I4$, $^2I5$, $^2K1$, $^2K2$, $^2K3$, $^2K4$, $^2K5$, $^2L1$, $^2L2$, $^2L3$, $^2M1$, $^2M2$, ^2N, ^2O

$\underline{\mathrm{f}}^6$
(119 LS terms)

---

^7F, ^5S, ^5P, $^5D1$, $^5D2$, $^5D3$, $^5F1$, $^5F2$, $^5G1$, $^5G2$, $^5G3$, $^5H1$, $^5H2$, $^5I1$, $^5I2$, ^5K, ^5L, $^3P1$, $^3P2$, $^3P3$, $^3P4$, $^3P5$, $^3P6$, $^3D1$, $^3D2$, $^3D3$, $^3D4$, $^3D5$, $^3F1$, $^3F2$, $^3F3$, $^3F4$, $^3F5$, $^3F6$, $^3F7$, $^3F8$, $^3F9$, $^3G1$, $^3G2$, $^3G3$, $^3G4$, $^3G5$, $^3G6$, $^3G7$, $^3H1$, $^3H2$, $^3H3$, $^3H4$, $^3H5$, $^3H6$, $^3H7$, $^3H8$, $^3H9$, $^3I1$, $^3I2$, $^3I3$, $^3I4$, $^3I5$, $^3I6$, $^3K1$, $^3K2$, $^3K3$, $^3K4$, $^3K5$, $^3K6$, $^3L1$, $^3L2$, $^3L3$, $^3M1$, $^3M2$, $^3M3$, ^3N, ^3O, $^1S1$, $^1S2$, $^1S3$, $^1S4$, ^1P, $^1D1$, $^1D2$, $^1D3$, $^1D4$, $^1D5$, $^1D6$, $^1F1$, $^1F2$, $^1F3$, $^1F4$, $^1G1$, $^1G2$, $^1G3$, $^1G4$, $^1G5$, $^1G6$, $^1G7$, $^1G8$, $^1H1$, $^1H2$, $^1H3$, $^1H4$, $^1I1$, $^1I2$, $^1I3$, $^1I4$, $^1I5$, $^1I6$, $^1I7$, $^1K1$, $^1K2$,

$^1K3$, $^1L1$, $^1L2$, $^1L3$, $^1L4$, $^1M1$, $^1M2$, $^1N1$, $^1N2$, ^1Q

$\underline{\mathrm{f}}^7$
(119 LS terms)

^8S, ^6P, ^6D, ^6F, ^6G, ^6H, ^6I, $^4S1$, $^4S2$, $^4P1$, $^4P2$, $^4D1$, $^4D2$, $^4D3$, $^4D4$, $^4D5$, $^4D6$, $^4F1$, $^4F2$, $^4F3$, $^4F4$, $^4F5$, $^4G1$, $^4G2$, $^4G3$, $^4G4$, $^4G5$, $^4G6$, $^4G7$, $^4H1$, $^4H2$, $^4H3$, $^4H4$, $^4H5$, $^4I1$, $^4I2$, $^4I3$, $^4I4$, $^4I5$, $^4K1$, $^4K2$, $^4K3$, $^4L1$, $^4L2$, $^4L3$, ^4M, ^4N, $^2S1$, $^2S2$, $^2P1$, $^2P2$, $^2P3$, $^2P4$, $^2P5$, $^2D1$, $^2D2$, $^2D3$, $^2D4$, $^2D5$, $^2D6$, $^2D7$, $^2F1$, $^2F2$, $^2F3$, $^2F4$, $^2F5$, $^2F6$, $^2F7$, $^2F8$, $^2F9$, $^2F10$, $^2G1$, $^2G2$, $^2G3$, $^2G4$, $^2G5$, $^2G6$, $^2G7$, $^2G8$, $^2G9$, $^2G10$, $^2H1$, $^2H2$, $^2H3$, $^2H4$, $^2H5$, $^2H6$, $^2H7$, $^2H8$, $^2H9$, $^2I1$, $^2I2$, $^2I3$, $^2I4$, $^2I5$, $^2I6$, $^2I7$, $^2I8$, $^2I9$, $^2K1$, $^2K2$, $^2K3$, $^2K4$, $^2K5$, $^2K6$, $^2K7$, $^2L1$, $^2L2$, $^2L3$, $^2L4$, $^2L5$, $^2M1$, $^2M2$, $^2M3$, $^2M4$, $^2N1$, $^2N2$, ^2O, ^2Q

$\underline{\mathrm{f}}^8$
(119 LS terms)

^7F, ^5S, ^5P, $^5D1$, $^5D2$, $^5D3$, $^5F1$, $^5F2$, $^5G1$, $^5G2$, $^5G3$, $^5H1$, $^5H2$, $^5I1$, $^5I2$, ^5K, ^5L, $^3P1$, $^3P2$, $^3P3$, $^3P4$, $^3P5$, $^3P6$, $^3D1$, $^3D2$, $^3D3$, $^3D4$, $^3D5$, $^3F1$, $^3F2$, $^3F3$, $^3F4$, $^3F5$, $^3F6$, $^3F7$, $^3F8$, $^3F9$, $^3G1$, $^3G2$, $^3G3$, $^3G4$, $^3G5$, $^3G6$, $^3G7$, $^3H1$, $^3H2$, $^3H3$, $^3H4$, $^3H5$, $^3H6$, $^3H7$, $^3H8$, $^3H9$, $^3I1$, $^3I2$, $^3I3$, $^3I4$, $^3I5$, $^3I6$, $^3K1$, $^3K2$, $^3K3$, $^3K4$, $^3K5$, $^3K6$, $^3L1$, $^3L2$, $^3L3$, $^3M1$, $^3M2$, $^3M3$, ^3N, ^3O, $^1S1$, $^1S2$, $^1S3$, $^1S4$, ^1P, $^1D1$, $^1D2$, $^1D3$, $^1D4$, $^1D5$, $^1D6$, $^1F1$, $^1F2$, $^1F3$, $^1F4$, $^1G1$, $^1G2$, $^1G3$, $^1G4$, $^1G5$, $^1G6$, $^1G7$, $^1G8$, $^1H1$, $^1H2$, $^1H3$, $^1H4$, $^1I1$, $^1I2$, $^1I3$, $^1I4$, $^1I5$, $^1I6$, $^1I7$, $^1K1$, $^1K2$, $^1K3$, $^1L1$, $^1L2$, $^1L3$, $^1L4$, $^1M1$, $^1M2$, $^1N1$, $^1N2$, ^1Q

$\underline{\mathrm{f}}^9$
(73 LS terms)

^6P, ^6F, ^6H, ^4S, $^4P1$, $^4P2$, $^4D1$, $^4D2$, $^4D3$, $^4F1$, $^4F2$, $^4F3$, $^4F4$, $^4G1$, $^4G2$, $^4G3$, $^4G4$, $^4H1$, $^4H2$, $^4H3$, $^4I1$, $^4I2$, $^4I3$, $^4K1$, $^4K2$, ^4L, ^4M, $^2P1$, $^2P2$, $^2P3$, $^2P4$, $^2D1$, $^2D2$, $^2D3$, $^2D4$, $^2D5$, $^2F1$, $^2F2$, $^2F3$, $^2F4$, $^2F5$, $^2F6$, $^2F7$, $^2G1$, $^2G2$, $^2G3$, $^2G4$, $^2G5$, $^2G6$, $^2H1$, $^2H2$, $^2H3$, $^2H4$, $^2H5$, $^2H6$, $^2H7$, $^2I1$, $^2I2$, $^2I3$, $^2I4$, $^2I5$, $^2K1$, $^2K2$, $^2K3$, $^2K4$, $^2K5$, $^2L1$, $^2L2$, $^2L3$, $^2M1$, $^2M2$, ^2N, ^2O

$\underline{\mathrm{f}}^{10}$
(47 LS terms)

^5S, ^5D, ^5F, ^5G, ^5I, $^3P1$, $^3P2$, $^3P3$, $^3D1$, $^3D2$, $^3F1$, $^3F2$, $^3F3$, $^3F4$, $^3G1$, $^3G2$, $^3G3$, $^3H1$, $^3H2$, $^3H3$, $^3H4$, $^3I1$, $^3I2$, $^3K1$, $^3K2$, ^3L, ^3M, $^1S1$, $^1S2$, $^1D1$, $^1D2$, $^1D3$, $^1D4$, ^1F, $^1G1$, $^1G2$, $^1G3$, $^1G4$, $^1H1$, $^1H2$, $^1I1$, $^1I2$, $^1I3$, ^1K, $^1L1$, $^1L2$, ^1N

$\underline{\mathrm{f}}^{11}$
(17 LS terms)

^4S, ^4D, ^4F, ^4G, ^4I, ^2P, $^2D1$, $^2D2$, $^2F1$, $^2F2$, $^2G1$, $^2G2$, $^2H1$, $^2H2$, ^2I, ^2K, ^2L

$\underline{\mathrm{f}}^{12}$
(7 LS terms)

^3P, ^3F, ^3H, ^1S, ^1D, ^1G, ^1I

$\underline{\mathrm{f}}^{13}$
(1 LS term)

^2F

$\underline{\mathrm{f}}^{14}$
(1 LS term)

^1S

In **qlanth** these terms may be queried through the function `AllowedNKSLTerms`.

```
AllowedNKSLTerms::usage = "AllowedNKSLTerms[numE] returns a list with
    the allowed terms in the f^numE configuration, the terms are given
    as strings in spectroscopic notation. The integers in the last
    positions are used to distinguish cases with degeneracy.";
AllowedNKSLTerms[numE_] := (Map[First, CFPTerms[Min[numE, 14-numE]]])
AllowedNKSLTerms[0] = {"1S"};
AllowedNKSLTerms[14] = {"1S"};
```

The LS quantum numbers, however, only get us half-way to the $|LSJM_J\rangle$ basis. For the J quantum number corresponding to total angular momentum, adding L and S can have values ranging from $|L-S|$ to $|L+S|$. Finally, for each of these values of J there are then $2J+1$ projections on the z-axis for the possible values of M_J. For instance, the ordered $|LSJM_J\rangle$ basis for $\underline{\mathrm{f}}^2$ is the one below.

$(J=0)$
(2 kets)

$|^3P,0,0\rangle$, $|^1S,0,0\rangle$

$(J=1)$
(3 kets)

$|^3P,1,-1\rangle$, $|^3P,1,0\rangle$, $|^3P,1,1\rangle$

$(J=2)$
(15 kets)

$|^3P,2,-2\rangle$, $|^3P,2,-1\rangle$, $|^3P,2,0\rangle$, $|^3P,2,1\rangle$, $|^3P,2,2\rangle$, $|^3F,2,-2\rangle$, $|^3F,2,-1\rangle$, $|^3F,2,0\rangle$, $|^3F,2,1\rangle$, $|^3F,2,2\rangle$, $|^1D,2,-2\rangle$, $|^1D,2,-1\rangle$, $|^1D,2,0\rangle$, $|^1D,2,1\rangle$, $|^1D,2,2\rangle$

$(J=3)$
(7 kets)

$|^3F,3,-3\rangle$, $|^3F,3,-2\rangle$, $|^3F,3,-1\rangle$, $|^3F,3,0\rangle$, $|^3F,3,1\rangle$, $|^3F,3,2\rangle$, $|^3F,3,3\rangle$

$(J=4)$
(27 kets)

$|^3F,4,-4\rangle$, $|^3F,4,-3\rangle$, $|^3F,4,-2\rangle$, $|^3F,4,-1\rangle$, $|^3F,4,0\rangle$, $|^3F,4,1\rangle$, $|^3F,4,2\rangle$, $|^3F,4,3\rangle$, $|^3F,4,4\rangle$, $|^3H,4,-4\rangle$, $|^3H,4,-3\rangle$, $|^3H,4,-2\rangle$, $|^3H,4,-1\rangle$, $|^3H,4,0\rangle$, $|^3H,4,1\rangle$, $|^3H,4,2\rangle$, $|^3H,4,3\rangle$, $|^3H,4,4\rangle$, $|^1G,4,-4\rangle$, $|^1G,4,-3\rangle$, $|^1G,4,-2\rangle$, $|^1G,4,-1\rangle$, $|^1G,4,0\rangle$, $|^1G,4,1\rangle$, $|^1G,4,2\rangle$, $|^1G,4,3\rangle$, $|^1G,4,4\rangle$

$(J=5)$
(11 kets)

$|^3H,5,-5\rangle$, $|^3H,5,-4\rangle$, $|^3H,5,-3\rangle$, $|^3H,5,-2\rangle$, $|^3H,5,-1\rangle$, $|^3H,5,0\rangle$, $|^3H,5,1\rangle$, $|^3H,5,2\rangle$, $|^3H,5,3\rangle$, $|^3H,5,4\rangle$, $|^3H,5,5\rangle$

$(J=6)$
(26 kets)

$|^3H,6,-6\rangle$, $|^3H,6,-5\rangle$, $|^3H,6,-4\rangle$, $|^3H,6,-3\rangle$, $|^3H,6,-2\rangle$, $|^3H,6,-1\rangle$, $|^3H,6,0\rangle$, $|^3H,6,1\rangle$, $|^3H,6,2\rangle$, $|^3H,6,3\rangle$, $|^3H,6,4\rangle$, $|^3H,6,5\rangle$, $|^3H,6,6\rangle$, $|^1I,6,-6\rangle$, $|^1I,6,-5\rangle$, $|^1I,6,-4\rangle$, $|^1I,6,-3\rangle$, $|^1I,6,-2\rangle$, $|^1I,6,-1\rangle$, $|^1I,6,0\rangle$, $|^1I,6,1\rangle$, $|^1I,6,2\rangle$, $|^1I,6,3\rangle$, $|^1I,6,4\rangle$, $|^1I,6,5\rangle$, $|^1I,6,6\rangle$

The order above is exemplar of order in the bases, notice how the basis vectors are sorted in order of increasing J, so that for instance not all of the basis kets associated with the ^3P LS term are contiguous.

In **qlanth** the ordered basis used for a given $\underline{\mathrm{f}}^n$ is provided by BasisLSJMJ.

```
BasisLSJMJ::usage = "BasisLSJMJ[numE] returns the ordered basis in L-S-
    J-MJ with the total orbital angular momentum L and total spin
    angular momentum S coupled together to form J. The function returns
     a list with each element representing the quantum numbers for each
     basis vector. Each element is of the form {SL (string in
    spectroscopic notation),J,MJ}.";
```

```
BasisLSJMJ[numE_] := Module[{energyStatesTable, basis, idx1},
  (
    energyStatesTable = BasisTableGenerator[numE];
    basis = Table[
      energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
      {idx1, 1, Length[AllowedJ[numE]]}];
    basis = Flatten[basis, 1];
    Return[basis]
  )
  ];
```

2 The coefficients of fractional parentage

In the 1920s and 1930s when the spectroscopic evidence was being put together to ellucidate the principles of quantum mechanics, one of the conceptual tools that was put forward for the analysis of the complex spectra of ions [BG34] consisted in using the spectrum of an ion at a lower stage of ionization to understand a higher stage of ionization. For instance, using the third spectrum of oxygen (OIV) in order to understand the fourth spectrum (OIII) of the same element.

This idea matured a couple of decades until the work of Giulio Racah made it mathematically precise. Racah clarified the way in which the wavefunctions from configuration $\underline{\mathrm{f}}^{n-1}$ can be used to build up the wavefunction of configuration $\underline{\mathrm{f}}^{n}$, as a "sum over parents"

$$|\underline{\ell}^n \alpha LS\rangle = \sum_{\bar{\alpha}\bar{L}\bar{S}} \underbrace{\left(\underline{\ell}^{n-1}\bar{\alpha}\bar{L}\bar{S}\right\}\underline{\ell}^n \alpha LS\right)}_{\substack{\text{How much of parent } |\underline{\ell}^{n-1}\bar{\alpha}\bar{L}\bar{S}\rangle \\ \text{is in daughter} |\underline{\ell}^{n}\alpha LS\rangle}} \underbrace{|\left(\underline{\ell}^{n-1}\bar{\alpha}\bar{L}\bar{S}, \underline{\ell}\right) LS\rangle}_{\substack{\text{Couple an additional } \underline{\ell} \\ \text{to the parent } |\underline{\ell}^{n-1}\bar{\alpha}\bar{L}\bar{S}\rangle}} . \tag{1}$$

Most importantly for us, in **qlanth** the coefficients of fractional parentage can be used to evaluate matrix elements of operators, such as in Eqn-25, Eqn-27, Eqn-40, and Eqn-51. These formulas realize a convenient calculational advantage: if one knows matrix elements in one configuration, then one can immediately calculate them in any other configuration.

In principle all the data that is needed in order to evaluate the matrix elements that **qlanth** uses can all be derived from coefficients of fractional parentage, tables of 6j and 3j coefficients, and the LSUW labels for the terms in the $\underline{\mathrm{f}}^{n}$ configurations.

The data for the coefficients of fractional parentage we owe to [Vel00] from which the file `B1F_-all.txt` originates, and which we use here to extract this useful escalator up the $\underline{\mathrm{f}}^{n}$ configurations.

In **qlanth** the function `GenerateCFPTable` is used to parse the data contained in this file. From this data an association `CFP` is generated, whose keys are made to represent LS terms from a configuration $\underline{\mathrm{f}}^{n}$ and whose values are list which contain all the parents terms, together with the corresponding coefficients of fractional parentage.

3 The JJ' block structure

Now that we know how the bases are ordered, we can already understand the structure of how the final Hamiltonian matrix representation is put together.

For a given configuration $\underline{\mathrm{f}}^{n}$ and for each term $\hat{h}$ in the Hamiltonian, **qlanth** first calculates the matrix elements $\langle \alpha LSJM_J|\hat{h}|\alpha' L'S'J'M'_J\rangle$ so that for each interaction an association with keys of the form $\{J, J'\}$ are created. The values being rectangular rank-2 arrays.

Fig-1 shows roughly this block structure for $\underline{\mathrm{f}}^{2}$. In that figure the shape of the rectangular blocks is determined by the fact that for $J = 0, 1, 2, 3, 4, 5, 6$ there are (2, 3, 15, 7, 27, 11, 26) corresponding basis states. As such, for example, the first row of blocks consists of blocks of size (2×2), (2×3), (2×15), (2×7), (2×27), (2×11), and (2×26).

In **qlanth** these blocks are put together by the function `JJBlockMatrix` which adds together the contributions from the different terms in the Hamiltonian.

```
JJBlockMatrix::usage = "For given J, J' in the f^n configuration
    JJBlockMatrix[numE, J, J'] determines all the SL S'L' terms that
    may contribute to them and using those it provides the matrix
    elements <J, LS | H | J', LS'>. H having contributions from the
    following interactions: Coulomb, spin-orbit, spin-other-orbit,
    electrostatically-correlated-spin-orbit, spin-spin, three-body
    interactions, and crystal-field.";
Options[JJBlockMatrix] = {"Sparse"->True, "ChenDeltas"->False};
JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]]:= Module[
  {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
  SLterm, SpLpterm,
  MJ, MJp,
  subKron, matValue, eMatrix},
  (
    NKSLJMs  = AllowedNKSLJMforJTerms[numE, J];
    NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
    eMatrix =
```

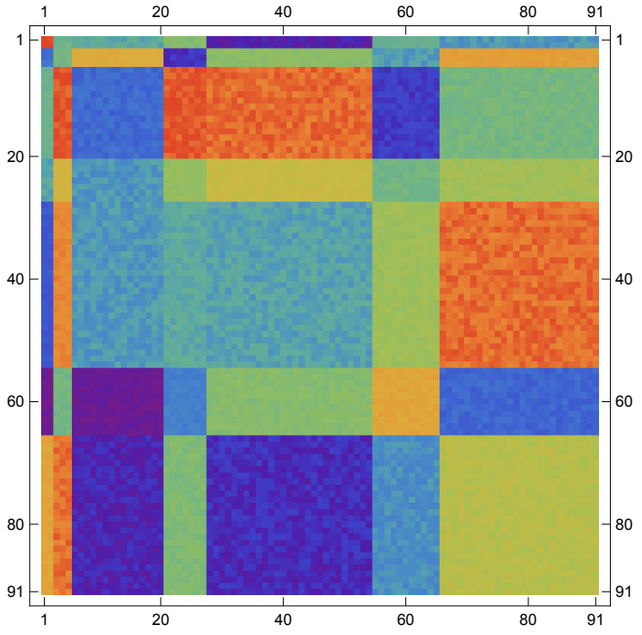

Figure 1: The JJ block structure for $\underline{f}^2$

```
      Table[
        (*Condition for a scalar matrix op*)
        SLterm   =  NKSLJM[[1]];
        SpLpterm = NKSLJMp[[1]];
        MJ       =  NKSLJM[[3]];
        MJp      = NKSLJMp[[3]];
        subKron  =
          (
            KroneckerDelta[J, Jp] *
            KroneckerDelta[MJ, MJp]
          );
        matValue =
          If[subKron==0,
            0,
              (
                ElectrostaticTable[{numE, SLterm, SpLpterm}] +
                ElectrostaticConfigInteraction[{SLterm, SpLpterm}] +
                SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
                MagneticInteractions[{numE, SLterm, SpLpterm, J}, "
    ChenDeltas" -> OptionValue["ChenDeltas"]] +
                ThreeBodyTable[{numE, SLterm, SpLpterm}]
              )
          ];
        matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp}];
        matValue,
      {NKSLJMp, NKSLJMps},
      {NKSLJM , NKSLJMs}
      ];
  If[OptionValue["Sparse"],
    eMatrix = SparseArray[eMatrix]
  ];
  Return[eMatrix]
)
];
```

Once these blocks have been calculate and saved to disk (in the folder `./hams/`) the function `HamMatrixAssembly` takes them, assembles the arrays in block form, and finally flattens it to provide a rank-2 array. This are the arrays that are finally diagonalized to find energies and eigenstates.

```
HamMatrixAssembly::usage="HamMatrixAssembly[numE] returns the
    Hamiltonian matrix for the f^n_i configuration. The matrix is
    returned as a SparseArray. The function admits an optional
    parameter \"FilenameAppendix\" which can be used to modify the
    filename to which the resulting array is exported to. It also
    admits an optional parameter \"IncludeZeeman\" which can be used to
     include the Zeeman interaction;";
Options[HamMatrixAssembly] = {"FilenameAppendix"->"","IncludeZeeman"->
```

```
    False};
HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
  {numE, ii, jj, howManyJs, Js, blockHam},
  (*######################################*)
  ImportFun = ImportMZip;
  (*######################################*)
  (*hole-particle equivalence enforcement*)
  numE = nf;
  allVars = {E0, E1, E2, E3, ζ, F0, F2, F4, F6, M0, M2, M4, T2, T2p,
    T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
    α, β, γ, B02, B04, B06, B12, B14, B16,
    B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16, S22,
    S24, S26, S34, S36, S44, S46, S56, S66, T11, T11p, T12, T14, T15,
    T16,
    T17, T18, T19, Bx, By, Bz};
  params0 = AssociationThread[allVars, allVars];
  If[nf > 7,
    (
      numE = 14 - nf;
      params = HoleElectronConjugation[params0];
    ),
    params = params0;
  ];
  (* Load symbolic expressions for LS,J,J' energy sub-matrices. *)
  emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
    OptionValue["FilenameAppendix"]];
  JJBlockMatrixTable = ImportFun[emFname];
  (*Patch together the entire matrix representation using J,J' blocks.
    *)
  PrintTemporary["Patching JJ blocks ..."];
  Js         = AllowedJ[numE];
  howManyJs  = Length[Js];
  blockHam   = ConstantArray[0, {howManyJs, howManyJs}];
  Do[
    blockHam[[jj, ii]] = JJBlockMatrixTable[{numE, Js[[ii]], Js[[jj
    ]]}];,
  {ii, 1, howManyJs},
  {jj, 1, howManyJs}
  ];
  (* Once the block form is created flatten it *)
  blockHam = ArrayFlatten[blockHam];
  blockHam = ReplaceInSparseArray[blockHam, params];
  If[OptionValue["IncludeZeeman"],
    (
      PrintTemporary["Including Zeeman terms ..."];
      {magx, magy, magz} = MagDipoleMatrixAssembly[numE];
      blockHam += - TeslaToKayser * (Bx * magx + By * magy + Bz * magz)
    ;
    )
  ];
  Return[blockHam];
  ]
```

4 Units

All of the matrix elements of the Hamiltonian are calculated using the Kayser ($\mathrm{K} \equiv \mathrm{cm}^{-1}$) as the (pseudo) energy unit. All the parameters (except the magnetic field which is in Tesla) in the effective Hamiltonian are assumed to be in this unit.

As is customary, the angular momentum operators assume atomic units in which $\hbar = 1$.

5 The effective Hamiltonian

Electrons in a multi-electron ion are subject to several interactions. Firstly, they are attracted to the nucleus around which they orbit. Additionally, they experience repulsion from other electrons. Electrons also possess spin, subjecting them to various magnetic interactions. The spin of each electron interacts with the magnetic field generated by either its own orbital angular momentum or that of another electron. Finally, among pairs of electrons, the spin of one can influence the other through the interaction of their respective magnetic dipoles.

This framework sufficiently describes the interactions within a free ion. However, to extend this model to ions within a crystal, we must incorporate the effects of the crystal field. This is often achieved by considering the electric field that an ion experiences from the surrounding charges in the crystal lattice, a concept referred to as the crystal field effect.

The Hilbert space of a multi-electron ion is a vast stage. In principle the Hilbert space should have a countable infinity of discrete states and a uncountable infinity of states to describe the unbound states. This is clearly too much to handle, but thankfully, this large stage can be put in some order thanks to the exclusion principle. The exclusion principle (together with that graceful tendency of things to drift downwards the energetic wells) provides the shell structure. This shell structure, in turn, makes it possible that an atom with many electrons, can be described effectively as an aggregate of an inert core, and a fewer active valence electrons.

Take for instance a triply ionized neodymium atom. In principle, this gives us the daunting task of dealing with 57 electrons. However, 54 of them arrange themselves in a xenon core, so that we are only left to deal with only three. Three are still a challenging task, but much less so than fifty seven. Furthermore, the exclusion principle also guides us in what type of orbital we could possibly place these three electrons, in the case of the lanthanide ions, this being the 4f orbitals. But not really, there are many more unoccupied orbitals outside of the xenon core, two of these electrons, if they are willing to pay the energetic price, they could find themselves in a 5d or a 6s orbital.

Here we shall assume a single-configuration description. Meaning that all the valence electrons in the ions that we study here will all be considered to be located in f-orbitals, or what is the same, that they are described by $\underline{\mathrm{f}}^n$wavefunctions. This is, however, a harsh approximation, but thankfully one can make some amends to it. The effects that arise in the single configuration description because of omitting all the other possible orbitals where the electrons might find themselves, this is what we call *configuration interaction* or CI for short.

These effects can be brought within the simplified description only through the help of perturbation theory. The task not the usual one of correcting for the energies/eigenvectors given an added perturbation, but rather to consider the effects of using a truncated Hilbert space due to a known interaction. What results from this is are operators that now act solely within the single configuration but with a convoluted coefficient that depends on overlaps between different configurations. This coefficient one could try to evaluate, and there are some that have trodden this road. Others simply label that complex expression with an unassuming symbol, and leave it as a parameter that one can hope to fit against experimental data. It is from CI that the parameters $\alpha, \beta, \gamma, P^{(k)}, T^{(k)}$ enter into the description.

Something that is also borne out of the configuration interaction analysis is that their influence also modifies previously present intra-configuration operators. For instance, part of the configuration interaction influence that results from the Coulomb repulsion between electrons brings about new operators that need to be included, but they also contribute to the intra-configuration Slater integrals.

When finding the matrix elements of the Hamiltonian defined by these terms, one also requires the specification of the basis in which the matrix elements will be computed. What we shall use here are states determined by five quantum numbers: the total orbital angular momentum L, the total spin angular momentum S, the total angular momentum J, and the projection of the total angular momentum along the z-axis M_J. To account for the fact that there might be a few different ways to amount for a given LS, it becomes necessary to have a fifth quantum number that discriminates between these different cases. This other quantum number we shall simply call α, which in the notation of Nielson and Koster is simply an integer number that enumerates all the possible LS in a given $\underline{\mathrm{f}}^n$configuration.

Putting all of this together leads to the following Hamiltonian. In there, "v-electrons" is shorthand for valence electrons.

$$\hat{\mathcal{H}} = \underbrace{\hat{\mathcal{H}}_{\mathrm{k}}}_{\text{kinetic}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{e:sn}}}_{\text{e:shielded nuc}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{e:e}}}_{\text{e:e}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{s:o}}}_{\text{spin-orbit}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{s:s}}}_{\substack{\text{spin:spin} \\ \text{and spin:other-orbit}}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{s:oo\oplus ecs:o}}}_{\substack{\text{spin:other-orbit} \\ \text{ec-correlated-spin:orbit}}} + \tag{2}$$

$$\underbrace{\hat{\mathcal{H}}_{\mathcal{SO}(3)}}_{\text{Trees effective op}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{G_2}}}_{\mathrm{G_2}\text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{\mathcal{SO}(7)}}_{\mathcal{SO}(7)\text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{\text{}^{\bullet}_{\bullet\bullet}}}_{\substack{\text{effective} \\ \text{three-body}}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{cf}}}_{\text{crystal field}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{Z}}}_{\text{Zeeman}} \tag{3}$$

$$\hat{\mathcal{H}}_{\mathrm{k}} = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{n}\nabla_i^2 \text{ (kinetic energy of } n \text{ v-electrons)} \tag{4}$$

$$\hat{\mathcal{H}}_{\mathrm{e:sn}} = \sum_{i=1}^{n} V_{\mathrm{sn}}(\hat{r}_i) \text{ (interaction of v-electrons with shielded nuclear charge)} \tag{5}$$

$$\hat{\mathcal{H}}_{\mathrm{e:e}} = \sum_{i>j}^{n,n}\frac{e^2}{\|\hat{\vec{r}}_i - \hat{\vec{r}}_j\|} = \sum_{k=0,2,4,6} F^{(k)}\hat{f}_k \text{ (v-electron:v-electron repulsion)} \tag{6}$$

$$\hat{\mathcal{H}}_{\mathrm{s:o}} = \begin{cases} \sum_{i=1}^{n}\xi(r_i)\left(\hat{\underline{\jmath}}_i\cdot\hat{\underline{\ell}}_i\right) \text{ with } \xi(r_i) = \frac{\hbar^2}{2m^2c^2r_i}\frac{\mathrm{d}V_{\mathrm{sn}}(r_i)}{\mathrm{d}r_i} \\ \sum_{i=1}^{n}\zeta\left(\hat{\underline{\jmath}}_i\cdot\hat{\underline{\ell}}_i\right) \text{ with } \zeta \text{ the radial average of } \xi(r_i) \text{ or used as phenomenological parameter} \end{cases} \tag{7}$$

$$\hat{\mathcal{H}}_{\mathrm{s:s}} = \sum_{k=0,2,4} M^{(k)}\hat{m}_k^{ss} \tag{8}$$

$$\hat{\mathcal{H}}_{\mathrm{s:oo\oplus ecs:o}} = \sum_{k=2,4,6} P^{(k)}\hat{p}_k + \sum_{k=0,2,4} M^{(k)}\hat{m}_k \tag{9}$$

$\mathcal{C}(\mathcal{G}) \coloneqq$ The Casimir operator of group $\mathcal{G}$.

$$\hat{\mathcal{H}}_{\mathcal{SO}(3)} = \alpha\,\mathcal{C}(\mathbb{R}^3) = \alpha\hat{L}^2 \text{ (Trees effective operator)} \tag{10}$$

$$\hat{\mathcal{H}}_{\mathrm{G_2}} = \beta\,\mathcal{C}(\mathrm{G_2}) \tag{11}$$

$$\hat{\mathcal{H}}_{\mathcal{SO}(7)} = \gamma\,\mathcal{C}(\mathcal{SO}(7)) \tag{12}$$

$$\hat{\mathcal{H}}_{\text{}^{\bullet}_{\bullet\bullet}} = T'^{(2)}t'_2 + \sum_{\substack{k=2,3,4,6,7,8,\\ 11,12,14,15,\\ 16,17,18,19}} T^{(k)}\hat{t}_k \text{ (effective 3-body operators } \hat{t}_k) \tag{13}$$

$$\hat{\mathcal{H}}_{\mathrm{cf}} = \sum_{i=1}^{n} V_{\mathrm{CF}}(\hat{r}_i) = \sum_{i=1}^{n}\sum_{k=2,4,6}\sum_{q=-k}^{k} B_q^{(k)}C_q^{(k)}(i) \text{ (crystal field interaction of v-electrons with electrostatic field due to surroundings)} \tag{14}$$

$$\hat{\mathcal{H}}_{\mathrm{Z}} = -\vec{B}\cdot\hat{\mu} = -\mu_B\vec{B}\cdot\left(\hat{L} + g_s\hat{S}\right) \text{ (interaction with a magnetic field)} \tag{15}$$

It is of some importance to note that the eigenstates that we'll end up with have shoved under the rug all the radial dependence of the wavefunctions. This dependence has been already integrated in the parameters that the Hamiltonian has.

5.1 $\hat{\mathcal{H}}_{\mathrm{k}}$: kinetic energy

$$\hat{\mathcal{H}}_{\mathrm{k}} = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{N}\nabla_i^2 \text{ (kinetic energy of N v-electrons)} \tag{16}$$

Within the basis that we'll use, the kinetic energy simply contributes a constant energy shift, and since all we care about are energy transitions, then this term can be omitted from the analysis.

5.2 $\hat{\mathcal{H}}_{\mathrm{e:sn}}$: e:shielded nuc

$$\hat{\mathcal{H}}_{\mathrm{k}} = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{N}\nabla_i^2 \text{ (kinetic energy of N v-electrons)} \tag{17}$$

Instead of using the shielded nuclear charge this could have been instead the bare nuclear charge, but then we would have needed to take into account the repulsion from the electrons in closed shells. Here we are already bringing some simplification in that we approximate the compound effect on the valence electrons due to the charge of the filled shells and the charge of the nucleus is that of a central field.

Then again, this term also contributes a common energy shift to all the energies that we can obtain within the single-configuration description, so this one will also be omittted. It might be useful to use this term and the previous one to estimate the energy differences between the states in different configurations, but we will not do that here.

5.3 $\hat{\mathcal{H}}_{\text{e:e}}$: e:e repulsion

$$\hat{\mathcal{H}}_{\text{e:e}} = \sum_{i>j}^{n,n} \frac{e^2}{\|\hat{\vec{r}}_i - \hat{\vec{r}}_j\|} = \sum_{k=0,2,4,6} F^{(k)} \hat{f}_k = \sum_{k=0,1,2,3} E_k \hat{e}^k \tag{18}$$

This term is the first we will not discard. Calculating this term for the $\underline{\text{f}}^n$configurations was one of the contribution from Slater, as such the parameters we use to write it up are called *Slater integrals*. After the analysis from Slater, Giulio Racah contributed further to the analysis of this term. The insight that Racah had was that if in a given operator one identified the parts in it that transformed nicely according to the different symmetry groups present in the problem, then calculating the necessary matrix element in all $\underline{\text{f}}^n$configurations can be greatly simplified.

The functions used in `qlanth` to compute these LS-reduced matrix elements are `Electrostatic` and `fsubk`. In addition to these, the LS-reduced matrix elements of the tensor operators $\hat{C}^{(k)}$ and $\hat{U}^{(k)}$ are also needed. These functions are based in equations 12.16 and 12.17 from [Cow81] as specialized for the case of electrons belonging to a single $\underline{\text{f}}^n$configuration. By default this term is computed in terms of $F^{(k)}$ Slater integrals, but it can also be computed in of the E_k Racah parameters, the functions `EtoF` and `FtoE` instrumental for going from one representation to the other.

$$\langle \underline{\text{f}}^n \alpha^{2S+1} L \| \hat{\mathcal{H}}_{\text{e:e}} \| \underline{\text{f}}^n \alpha'^{2S'+1} L' \rangle = \sum_{k=0,2,4,6} F^{(k)} f_k(n, \alpha LS, \alpha' L' S') \tag{19}$$

where

$$f_k(n, \alpha LS, \alpha' L' S') = \frac{1}{2}\delta(S,S')\delta(L,L')\langle \underline{\text{f}} \| \hat{C}^{(k)} \| \underline{\text{f}} \rangle^2 \times$$
$$\left\{ \frac{1}{2L+1} \sum_{\alpha'' L''} \langle \underline{\text{f}}^n \alpha'' L'' S \| \hat{U}^{(k)} \| \underline{\text{f}}^n \alpha LS \rangle \langle \underline{\text{f}}^n \alpha'' L'' S \| \hat{U}^{(k)} \| \underline{\text{f}}^n \alpha' LS \rangle - \delta(\alpha, \alpha') \frac{n\,(4\underline{\text{f}} + 2 - n)}{(2\underline{\text{f}}+1)(4\underline{\text{f}}+1)} \right\} \tag{20}$$

```
Electrostatic::usage = "Electrostatic[{numE, NKSL, NKSLp}] returns the
    LS reduced matrix element for repulsion matrix element for
    equivalent electrons. See equation 2-79 in Wybourne (1965). The
    option \"Coefficients\" can be set to \"Slater\" or \"Racah\". If
    set to \"Racah\" then E_k parameters and e^k operators are assumed,
     otherwise the Slater integrals F^k and operators f_k. The default
    is \"Slater\".";
Options[Electrostatic] = {"Coefficients" -> "Slater"};
Electrostatic[{numE_, NKSL_, NKSLp_}, OptionsPattern[]]:= Module[
  {fsub0, fsub2, fsub4, fsub6,
   esub0, esub1, esub2, esub3,
   fsup0, fsup2, fsup4, fsup6,
   eMatrixVal, orbital},
  orbital = 3;
  Which[
    OptionValue["Coefficients"] == "Slater",
    (
      fsub0 = fsubk[numE, orbital, NKSL, NKSLp, 0];
      fsub2 = fsubk[numE, orbital, NKSL, NKSLp, 2];
      fsub4 = fsubk[numE, orbital, NKSL, NKSLp, 4];
      fsub6 = fsubk[numE, orbital, NKSL, NKSLp, 6];
      eMatrixVal = fsub0*F0 + fsub2*F2 + fsub4*F4 + fsub6*F6;
    ),
    OptionValue["Coefficients"] == "Racah",
    (
      fsup0 = fsupk[numE, orbital, NKSL, NKSLp, 0];
      fsup2 = fsupk[numE, orbital, NKSL, NKSLp, 2];
      fsup4 = fsupk[numE, orbital, NKSL, NKSLp, 4];
      fsup6 = fsupk[numE, orbital, NKSL, NKSLp, 6];
      esub0 = fsup0;
      esub1 = 9/7*fsup0 +   1/42*fsup2 +   1/77*fsup4 +  1/462*fsup6;
      esub2 =             143/42*fsup2 - 130/77*fsup4 + 35/462*fsup6;
      esub3 =              11/42*fsup2 + 4/77*fsup4   -  7/462*fsup6;
      eMatrixVal = esub0*E0 + esub1*E1 + esub2*E2 + esub3*E3;
    )
  ];
  Return[eMatrixVal];
]
```

```
fsubk::usage = "Slater integral f_k. See equation 12.17 in TASS.";
fsubk[numE_, orbital_, NKSL_, NKSLp_, k_] := Module[
  {terms, S, L, Sp, Lp, termsWithSameSpin, SL, fsubkVal,
    spinMultiplicity,
  prefactor, summand1, summand2},
  {S, L}   = FindSL[NKSL];
```

```
  {Sp, Lp} = FindSL[NKSLp];
  terms = AllowedNKSLTerms[numE];
  (* sum for summand1 is over terms with same spin *)
  spinMultiplicity  = 2*S + 1;
  termsWithSameSpin = StringCases[terms, ToString[spinMultiplicity] ~~
    __];
  termsWithSameSpin = Flatten[termsWithSameSpin];
  If[Not[{S, L} == {Sp, Lp}],
    Return[0]
  ];
  prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
  summand1 = Sum[(
      ReducedUkTable[{numE, orbital, SL, NKSL,  k}] *
      ReducedUkTable[{numE, orbital, SL, NKSLp, k}]
      ),
    {SL, termsWithSameSpin}
  ];
  summand1 = 1 / TPO[L] * summand1;
  summand2 = (
    KroneckerDelta[NKSL, NKSLp] *
      (numE *(4*orbital + 2 - numE)) /
      ((2*orbital + 1) * (4*orbital + 1))
    );
  fsubkVal = prefactor*(summand1 - summand2);
  Return[fsubkVal];
]
```

```
EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the corresponding {F0,
    F2, F4, F6} values. The inverse of FtoE.";
EtoF[E0_, E1_, E2_, E3_] := (Module[
  {F0, F2, F4, F6},
  F0 = 1/7       (7 E0 + 9 E1);
  F2 = 75/14     (E1 + 143 E2 + 11 E3);
  F4 = 99/7      (E1 - 130 E2 + 4 E3);
  F6 = 5577/350 (E1 + 35 E2 - 7 E3);
  Return[{F0, F2, F4, F6}];
]
);
```

```
FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the corresponding {E0,
    E1, E2, E3} values.
See eqn. 2-80 in Wybourne. Note that in that equation the subscripted
    Slater integrals are used but since this function assumes the the
    input values are superscripted Slater integrals, it is necessary to
     convert them using Dk.";
FtoE[F0_, F2_, F4_, F6_] := (Module[
  {E0, E1, E2, E3},
  E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
  E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
  E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;
  E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
  Return[{E0, E1, E2, E3}];
]
);
```

5.4 $\hat{\mathcal{H}}_{\text{s:o}}$: spin-orbit

Here one can be of two minds, one can either start from a relativistic description, only including the interaction of charged particles. Then when descending to the non-relativistic description one will notice a term that involves both the orbital angular momentum and the spin angular momentum. In this view the spin-orbit term arises as a relativistic correction to the non-relativistic Schrodinger equation.

From the non-relativistic viewpoint one may also take it as a given that the electron has an associated magnetic moment. From this one would then continue to consider the effect that magnetic fields have on it. One of those fields that one due to the motion of the electron around the nucleus, one would then conclude a term that involves both the spin and the orbital motion of the electron.

More generally one may picture an electron in a radial electrostatic potential $V(r)$, in which case the energy associated to the spin-orbit is

$$\hat{h}_{\text{s:o}} = \frac{\hbar^2}{2\mathrm{m}_e^2 c^2}\left(\frac{1}{r}\frac{\mathrm{d}V}{\mathrm{d}r}\right)\hat{l}\cdot\hat{s} \coloneqq \zeta(r)\hat{l}\cdot\hat{s}. \tag{21}$$

And adding up for all the n valence electrons

$$\hat{\mathcal{H}}_{\text{s:o}} = \sum_i^n \zeta(r_i)\hat{l}_i{\cdot}\hat{s}_i. \tag{22}$$

The matrix elements that we then require are

$$\begin{aligned}\langle \alpha LSJM_J|\hat{\mathcal{H}}_{\text{s:o}}|\alpha' L'S'J'M_{J'}\rangle &= \zeta\delta(J,J')\delta(M_J,M_{J'})\langle \alpha LSJM_J|\sum_i^n \hat{l}_i{\cdot}\hat{s}_i|\alpha' L'S'JM_J\rangle \\ &= \zeta(-1)^{J+L+S'}\begin{Bmatrix} L & L' & 1 \\ S' & S & J\end{Bmatrix}\langle \alpha LS|\sum_i^n \hat{l}_i{\cdot}\hat{s}_i|\alpha' L'S'\rangle \\ &= \zeta(-1)^{J+L+S'}\begin{Bmatrix} L & L' & 1 \\ S' & S & J\end{Bmatrix}\sqrt{\underline{\ell}(\underline{\ell}+1)(2\underline{\ell}+1)}\langle \alpha LS\|\hat{V}^{(11)}\|\alpha' L'S'\rangle \end{aligned} \tag{23}$$

Where $\hat{V}^{(11)}$ is a double tensor operator of rank one over spin and orbital parts defined as

$$\hat{V}^{(11)} = \sum_{i=1}^n \left(\hat{s}\hat{u}^{(1)}\right)_i, \tag{24}$$

where the rank on the spin operator $\hat{s}$ has been omitted, and the rank of the tensor operator shown explicitly as 1.

In **qlanth** the reduced matrix elements for this double tensor operator are calculated by ReducedV1k and aggregated in a static association called ReducedV1kTable. The reduced matrix elements of this operator are calculated using equation 2-101 from Wybourne's book on the spectroscopy of the rare earths [Wyb65]:

$$\begin{aligned}\langle \underline{\ell}^n\psi\|\hat{V}^{(1k)}\|\underline{\ell}^n\psi'\rangle = \langle \underline{\ell}^n\alpha LS\|\hat{V}^{(1k)}\|\underline{\ell}^n\alpha' L'S'\rangle = n\sqrt{\underline{\jmath}(\underline{\jmath}+1)(2\underline{\jmath}+1)}\sqrt{[\![S]\!]\,[\![L]\!]\,[\![S']\!]\,[\![L']\!]}\times \\ \sum_{\bar{\psi}}(-1)^{\bar{S}+\bar{L}+S+L+\underline{\ell}+\underline{\jmath}+k+1}\left(\psi\{\!|\bar{\psi}\right)\left(\bar{\psi}|\!\}\psi'\right)\begin{Bmatrix} S & S' & 1 \\ \underline{\jmath} & \underline{\jmath} & \bar{S}\end{Bmatrix}\begin{Bmatrix} L & L' & k \\ \underline{\ell} & \underline{\ell} & \bar{L}\end{Bmatrix}\end{aligned} \tag{25}$$

In this expression the sum over $\bar{\psi}$ depends on (ψ,ψ') and is over all the states in $\underline{\ell}^{n-1}$ which are common parents to both ψ and ψ'. Also note that in the equation above, since our concern are f-electron configurations, we have $\underline{\ell}=3$ and $\underline{\jmath}=\frac{1}{2}$ as is due to the electron.

```
ReducedV1k::usage = "ReducedV1k[n, l, SL, SpLp, k] gives the reduced
    matrix element of the spherical tensor operator V^(1k). See
    equation 2-101 in Wybourne 1965.";
ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
  {Vk1, S, L, Sp, Lp, Sb, Lb, spin, orbital, cfpSL, cfpSpLp,
  SLparents, SpLpparents, commonParents, prefactor},
  {spin, orbital} = {1/2, 3};
  {S, L}          = FindSL[SL];
  {Sp, Lp}        = FindSL[SpLp];
  cfpSL           = CFP[{numE, SL}];
  cfpSpLp         = CFP[{numE, SpLp}];
  SLparents       = First /@ Rest[cfpSL];
  SpLpparents     = First /@ Rest[cfpSpLp];
  commonParents = Intersection[SLparents, SpLpparents];
  Vk1 = Sum[(
      {Sb, Lb} = FindSL[\[Psi]b];
      Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
      CFPAssoc[{numE, SL, \[Psi]b}] *
      CFPAssoc[{numE, SpLp, \[Psi]b}] *
      SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
      SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
    ),
  {\[Psi]b, commonParents}
  ];
  prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp, Lp]
    ];
  Return[prefactor * Vk1];
  ]
```

5.5 $\hat{\mathcal{H}}_{\curlywedge}$: Three-body effective operators

Three body model interactions arise in the lanthanides due to configuration interaction between configurations $(4\underline{f})^n$ and $(4\underline{f})^{n\pm1}(n'\underline{\ell}')^{\mp1}$. As such they describe the effects of configuration interaction of single-electron excitations from the ground configuration to excited ones.

What results from the configuration interaction analysis (see [RW63],[Jud66], and [JS84], shows that the effective result from the model interaction arising from CI has the form of a three body operator.

These operators were initially studied by Wybourne and Rajnak in 1963 [RW63], their analysis was complemented soon after by [Jud66], and revisted again by Judd in 1984 [JS84].

This interaction is spanned by a set of 14 $\hat{t}_i$ of operators

$$\hat{\mathscr{H}}_{\curlywedge} = T'^{(2)}\hat{t}'_2 + \sum_{\substack{k=2,3,4,6,7,8,\\ 11,12,14,15,\\ 16,17,18,19}} T^{(k)}\hat{t}_k. \tag{26}$$

Where $\hat{t}'_2$ is a fifteenth legacy operator that is needed to reproduce old results from literature.

The calculation of a three body operator across the $\underline{f}^n$ configuration is analogous to how a two-body operator is calculated. Except that in this case what are needed are the reduced matrix elements in $\underline{f}^3$ and the equation that is needed to propagate these across the other configurations is modified accordingly to eqn. 4 (slightly modified to account for dependence on J and MJ) in [Jud66]:

$$\langle \underline{f}^n\psi|\hat{t}_i|\underline{f}^n\psi'\rangle = \delta(J,J')\delta(M_J,M'_J)\frac{n}{n-3}\sum_{\bar{\psi}\bar{\psi}'}\left(\psi\left\{\right\}\bar{\psi}\right)\left(\psi'\left\{\right\}\bar{\psi}'\right)\langle \underline{f}^{n-1}\bar{\psi}|\hat{t}_i|\underline{f}^{n-1}\bar{\psi}'\rangle \tag{27}$$

where the sum runs over the parents in $\underline{f}^{n-1}$ that are common to both the daughter terms ψ and ψ' in $\underline{f}^n$. The equation above yielding LSJMJ matrix elements, and being diagonal in J, MJ as is due to a scalar operator.

In **qlanth** this is all implemented in one burst in the function `GenerateThreeBodyTables`. Where the matrix elements in $\underline{f}^3$ are gotten from [JS84], where the data has been digitized in the files `Judd1984-1.csv` and `Judd1984-2.csv`, which are parsed through the function `ParseJudd1984`.

In `GenerateThreeBodyTables` a special case is made for $\hat{t}_2$ and $\hat{t}_{11}$ for which primed variants $\hat{t}'_2$ and $\hat{t}'_{11}$ are calculated differently beyond the half filled shell. In the case of the other operators, beyond $\underline{f}^7$ the matrix elements simply see a global sign flip, whereas in the case of $\hat{t}'_2$ and $\hat{t}'_{11}$ the coefficients of fractional parentage beyond $\underline{f}^7$ are used. This yields the unexpected result that in the $\underline{f}^{12}$ configuration, which corresponds to two holes, there is a non-zero three body operator $\hat{t}'_2$. This is an arcane result that was corrected by Judd in 1984 [JS84] but which lingered long enough that important work in the 1980s was calculated with it. When calculations are carried out if $\hat{t}'_2$ is used then $\hat{t}_2$ shouln't be used and vice versa.

```
GenerateThreeBodyTables::usage="This function generates the matrix
    elements for the three body operators using the coefficients of
    fractional parentage, including those beyond f^7.";
Options[GenerateThreeBodyTables] = {"Export" -> False};
GenerateThreeBodyTables[nmax_Integer : 14, OptionsPattern[]] := (
  tiKeys = {"t_{2}", "t_{2}^{'}", "t_{3}", "t_{4}", "t_{6}", "t_{7}",
    "t_{8}", "t_{11}", "t_{11}^{'}", "t_{12}", "t_{14}", "t_{15}",
    "t_{16}", "t_{17}", "t_{18}", "t_{19}"};
  TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
  juddOperators = ParseJudd1984[];
  (* op3MatrixElement[SL, SpLp, opSymbol] returns the value for the
    reduced matrix element of the operator opSymbol for the terms {SL,
    SpLp} in the f^3 configuration. *)
  op3MatrixElement[SL_, SpLp_, opSymbol_] := (
    jOP = juddOperators[{3, opSymbol}];
    key = {SL, SpLp};
    val = If[MemberQ[Keys[jOP], key],
      jOP[key],
      0];
    Return[val];
    );
  (*ti: This is the implementation of formula (2) in Judd & Suskin
    1984. It computes the matrix elements of ti in f^n by using the
    matrix elements in f3 and the coefficients of fractional parentage.
     If the option \"Fast\" is set to True then the values for n>7 are
    simply computed as the negatives of the values in the complementary
     configuration; this except for t2 and t11 which are treated as
    special cases. *)
  Options[ti] = {"Fast" -> True};
  ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
   Module[{nn, S, L, Sp, Lp,
      cfpSL, cfpSpLp,
      parentSL, parentSpLp, tnk, tnks},
    {S, L}   = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    fast     = OptionValue["Fast"];
    numH = 14 - nE;
    If[fast && Not[MemberQ[{"t_{2}","t_{11}"},tiKey]] && nE > 7,
      Return[-tktable[{numH, SL, SpLp, tiKey}]]
    ];
    If[(S == Sp && L == Lp),
     (
```

```
      cfpSL   = CFP[{nE, SL}];
      cfpSpLp = CFP[{nE, SpLp}];
      tnks = Table[(
          parentSL   = cfpSL[[nn, 1]];
          parentSpLp = cfpSpLp[[mm, 1]];
          cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
          tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
          ),
          {nn, 2, Length[cfpSL]},
          {mm, 2, Length[cfpSpLp]}
          ];
      tnk = Total[Flatten[tnks]];
      ),
     tnk = 0;
     ];
    Return[ nE / (nE - opOrder) * tnk];];
  (*Calculate the matrix elements of t^i for n up to nmax*)
  tktable = <||>;
  Do[(
    Do[(
      tkValue = Which[numE <= 2,
        (*Initialize n=1,2 with zeros*)
        0,
        numE == 3,
        (*Grab matrix elem in f^3 from Judd 1984*)
        SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
        True,
        SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2,
   3]]]
        ];
      tktable[{numE, SL, SpLp, opKey}] = tkValue;
      ),
    {SL, AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]},
    {opKey, Append[tiKeys, "e_{3}"]}
    ];
    PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], "
   configuration complete"]];
    ),
  {numE, 1, nmax}
  ];
```

```
ParseJudd1984::usage="This function parses the data from tables 1 and 2
     of Judd from Judd, BR, and MA Suskin. \"Complete Set of Orthogonal
     Scalar Operators for the Configuration f^3\". JOSA B 1, no. 2
    (1984): 261-65.\"";
Options[ParseJudd1984] = {"Export" -> False};
ParseJudd1984[OptionsPattern[]]:=(
  ParseJuddTab1[str_] := (
    strR = ToString[str];
    strR = StringReplace[strR, ".5" -> "^(1/2)"];
    num = ToExpression[strR];
    sign = Sign[num];
    num = sign*Simplify[Sqrt[num^2]];
    If[Round[num] == num, num = Round[num]];
    Return[num]);
```

5.6 $\hat{\mathcal{H}}_{\mathbf{cf}}$: crystal-field

The crystal-field aims to describe the influence of the surrounding lattice on the ion. The simplest way of doing this considers the lattice as a static aggregate of charges. For this aggregate of charges we could associate an electrostatic potential described as a multipolar sum of the form:

$$V(r_i, \theta_i, \phi_i) = \sum_{k=1}^{\infty} \mathcal{A}_q^{(k)} r^k C_q^{(k)}(\theta_i, \phi_i) \tag{28}$$

Where we have chosen a coordinate system with its origin at the position of the nucleus, and in which we only have positive powers of the distance r_i since here we have expanded the contributions from all the surrounding ions as a sum over spherical harmonics centered at the position of the nucleus, without r ever large enough to reach any of the positions of the lattice ions.

Furthermore, since we have n valence electrons, then the total crystal field potential is

$$\hat{\mathcal{H}}_{\mathrm{cf}}(\vec{r}) = \sum_{i=1}^{n} \sum_{k=0}^{\infty} \sum_{q=-k}^{(k)} \mathcal{A}_q^{(k)} r_i^k C_q^{(k)}(\theta_i, \phi_i). \tag{29}$$

And if we average the radial coordinate,

$$\hat{\mathscr{H}}_{\rm cf} = \sum_{i=1}^{n} \sum_{k=1}^{\infty} \sum_{q=-k}^{k} B_q^{(k)} C_q^{(k)}(i) \tag{30}$$

where the radial average is included as

$$B_q^{(k)} := \mathscr{A}_q^{(k)} \langle r^k \rangle. \tag{31}$$

In principle the value for $B_q^{(k)}$ could have both real an imaginary parts, in **qlanth** this is taken into account by separating out the real and imaginary parts with the replacement in terms of two real-valued parameters

$$B_q^{(k)} \to B_q^{(k)} + i S_q^{(k)}. \tag{32}$$

A staple of the Wigner-Racah algebra is writing up operators on interest in terms of standard ones for which the matrix elements are straightforward. One such operator is the unit tensor operator $\hat{u}^{(k)}$ for a single electron. The Wigner-Eckart theorem –on which all of this algebra is an elaboration– effectively separates the dynamical and geometrical parts of a given interaction; the unit tensor operators isolate the geometric contributions. This irreducible tensor operator $\hat{u}^{(k)}$ is defined as the tensor operator having the following reduced matrix elements (written in terms of the triangular delta, see section on notation):

$$\langle \ell \| \hat{u}^{(k)} \| \ell' \rangle = \triangle(\ell, k, \ell'). \tag{33}$$

In terms of this tensor one may then define the symmetric (in the sense that the resulting operator is equitable among all electrons) unit tensor operator for n particles as

$$\hat{U}^{(k)} = \sum_{i}^{n} \hat{u}_i^{(k)}. \tag{34}$$

This tensor is relevant to the calculation of the above matrix elements since

$$C_q^{(k)} = \langle \underline{\ell} \| C^{(k)} \| \underline{\ell}' \rangle \hat{u}_q^{(k)} = (-1)^{\underline{\ell}} \sqrt{[\![\ell]\!]\,[\![\underline{\ell}']\!]} \begin{pmatrix} \underline{\ell} & k & \underline{\ell}' \\ 0 & 0 & 0 \end{pmatrix} \hat{u}_q^{(k)}. \tag{35}$$

With this the matrix elements of $\hat{\mathscr{H}}_{\rm cf}$ in the $|LSJM_J\rangle$ basis are:

$$\overbrace{\langle \underline{\ell}^n \alpha S L J M_J | \hat{\mathscr{H}}_{\rm cf} | \underline{\ell}^n \alpha' S L' J' M_{J'} \rangle}^{\text{Wybourne eqn. 6-3}} = \sum_{k=1}^{\infty} \sum_{q=-k}^{k} B_q^{(k)} \langle \underline{\ell}^n \alpha S L J M_J | \hat{U}_q^{(k)} | \underline{\ell}^n \alpha' S L' J' M_{J'} \rangle \langle \underline{\ell} \| \hat{C}^{(k)} \| \underline{\ell} \rangle \tag{36}$$

where the matrix elements of $\hat{U}_q^{(k)}$ can be resolved with a 3j symbol as

$$\overbrace{\langle \underline{\ell}^n \alpha S L J M_J | \hat{U}_q^{(k)} | \underline{\ell}^n \alpha' S' L' J' M_{J'} \rangle}^{\text{Wybourne eqn. 6-4}} = (-1)^{J - M_J} \begin{pmatrix} J & k & J' \\ -M_J & q & M_{J'} \end{pmatrix} \langle \underline{\ell}^n \alpha S L J \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S' L' \rangle \tag{37}$$

and reduced a second time with the inclusion of a 6j symbol resulting in

$$\begin{aligned} \overbrace{\langle \underline{\ell}^n \alpha S L J \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S' L' \rangle}^{\text{Wybourne eqn. 6-5}} = (-1)^{S+L+J'+k} \sqrt{[\![J]\!]\,[\![J']\!]} \times \\ \begin{Bmatrix} J & J' & k \\ L' & L & S \end{Bmatrix} \langle \underline{\ell}^n \alpha S L \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S' L' \rangle. \end{aligned} \tag{38}$$

This last reduced matrix element is finally computed with a sum over $\bar{\alpha}\bar{L}\bar{S}$ which are the parents in configuration $\underline{\mathrm{f}}^{n-1}$ which are common to $|\alpha L S\rangle$ and $|\alpha' L' S'\rangle$ from configuration $\underline{\mathrm{f}}^n$:

$$\begin{aligned} \overbrace{\langle \underline{\ell}^n \alpha S L \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S' L' \rangle}^{\text{Cowan eqn. 11.53}} = \delta(S,S') n (-1)^{\underline{\ell}+L+k} \sqrt{[\![L]\!]\,[\![L']\!]} \times \\ \sum_{\bar{\alpha}\bar{L}\bar{S}} (-1)^{\bar{L}} \begin{Bmatrix} \underline{\ell} & k & \underline{\ell} \\ L & \bar{L} & L' \end{Bmatrix} \Big(\underline{\ell}^n \alpha L S \Big\{ \underline{\ell}^{n-1} \bar{\alpha} \bar{L} \bar{S} \Big) \Big(\underline{\ell}^{n-1} \bar{\alpha} \bar{L} \bar{S} \Big\} \underline{\ell}^n \alpha' L' S' \Big). \end{aligned} \tag{39}$$

From the $\langle \underline{\ell} \| \hat{C}^{(k)} \| \underline{\ell} \rangle$, and given that we are using $\underline{\ell} = \underline{\mathrm{f}} = 3$ we can see that by the triangular condition $\triangle(3, k, 3)$ the non-zero contributions only come from $k = 0, 1, 2, 3, 4, 5, 6$. An additional selection rule on k comes from considerations of parity. Since both the bra and the ket in $\langle \underline{\ell}^n \alpha S L J M_J | \hat{\mathscr{H}}_{\rm cf} | \underline{\ell}^n \alpha' S L' J' M_{J'} \rangle$ have the same parity, then the overall parity of the braket is determined by the parity of $C_q^{(k)}$, and since the parity of $C_q^{(k)}$ is $(-1)^k$ then for the braket to be non-zero we require that k should also be even. In view of this, in all the above equations for the crystal field the values for k should be limited to $2, 4, 6$. The value of $k = 0$ having been omitted from the start since this only contributes a common energy shift.

The above equations are implemented in **qlanth** by the function `CrystalField`. This function puts together the symbolic sum in &qn-36 by using the function `Cqk`. `Cqk` then uses the diagonal reduced matrix elements of $C_q^{(k)}$ and the precomputed values for `Uk` (stored in `ReducedUkTable`).

The required reduced matrix elements of $\hat{U}^{(k)}$ are calculated by the function `ReducedUk`, which is used by `GenerateReducedUkTable` to precompute its values.

5.7 $\hat{\mathcal{H}}_{\mathcal{SO}(3)}, \hat{\mathcal{H}}_{\mathbf{G}_2}, \hat{\mathcal{H}}_{\mathcal{SO}(7)}$: electrostatic configuration interaction

This is a first term where we take into account the very important contributions from configuration interaction. When the interaction with configurations ?? and ?? it was realized that the way the omission of these configurations in the single configuration description was to relax the previous restriction that $F^{(k)}$ should only have even values for k. Parallel to this Trees noticed an interesting fact which is that a fair amount of correction to the calculated spectrum of would benefit if one added to all of the LS energies a term quadratic in L. Soon after this it was acknowledged that the inclusion of odd $F^{(k)}$ was equivalent to adding three terms related to the Casimir operators of the groups $SO(3)$, G_2, and $SO(7)$. In addition to this, the configuration interaction analysis, also showed that the contributions from other configuration would also overlap with the already allowed even $F^{(k)}$.

One of these Casimir operators is the familiar $\hat{L}^2$ from $\mathcal{SO}(3)$. In analogy to $\hat{L}^2$ in which the quantum number L can be used to determine the eigenvalues, in the cases of $\hat{\mathcal{H}}_{\mathrm{G}_2}$ the necessary state label is the U label of the LS term, and in the case of $\hat{\mathcal{H}}_{\mathcal{SO}(7)}$ the necessary label is W. If $\Lambda_{\mathrm{G}_2}(U)$ is used to note the eigenvalue of the Casimir operator of G_2 corresponding to label U, and $\Lambda_{\mathcal{SO}(7)}(W)$ the eigenvalue corresponding to state label W, then the matrix elements of $\hat{\mathcal{H}}_{\mathcal{SO}(3)}$, $\hat{\mathcal{H}}_{\mathrm{G}_2}$ and $\hat{\mathcal{H}}_{\mathcal{SO}(7)}$ are diagonal in all quantum numbers and are given by

$$\langle \underline{\ell}^n \alpha S L J M_J | \hat{\mathcal{H}}_{\mathcal{SO}(3)} | \underline{\ell}^n \alpha' S' L' J' M'_J \rangle = \alpha \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) L(L+1) \tag{40}$$

$$\langle \underline{\ell}^n U \alpha S L J M_J | \hat{\mathcal{H}}_{\mathrm{G}_2} | \underline{\ell}^n U \alpha' S' L' J' M'_J \rangle = \beta \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) \Lambda_{\mathrm{G}_2}(U) \tag{41}$$

$$\langle \underline{\ell}^n W \alpha S L J M_J | \hat{\mathcal{H}}_{\mathcal{SO}(7)} | \underline{\ell}^n W \alpha' S' L' J' M_J \rangle = \gamma \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) \Lambda_{\mathcal{SO}(7)}(W) \tag{42}$$

In **qlanth** the role of $\Lambda_{\mathcal{SO}(7)}(W)$ is played by the function `GSO7W`, the role of $\Lambda_{\mathrm{G}_2}(U)$ by `GG2U`, and the role of $\Lambda_{\mathcal{SO}(3)}(L)$ by `CasimirSO3`. These are used by `CasimirG2`, `CasimirSO3`, and `CasimirSO7` which find the corresponding U, W, L labels to the LS terms provided to them. Finally, the function `ElectrostaticConfigInteraction` puts them together.

5.8 $\hat{\mathcal{H}}_{\text{s:s-s:oo}}$: spin-spin and spin-other-orbit

The calculation of the $\hat{\mathcal{H}}_{\text{s:s-s:oo}}$ is qualitatively different from the previous ones. The previous ones were self-contained in the sense that the reduced matrix elements that we require we also computed on our own. In the case of the interactions that follow from here, we need to take precomputed values for reduced matrix elements either in $\underline{\mathrm{f}}^2$ or in $\underline{\mathrm{f}}^3$ and then we "pull" them up for all $\underline{\mathrm{f}}^n$ configuration with the help of formulas involving coefficients of fractional parentage.

The analysis of spin-other-orbit, and the spin-spin contributions we use in **qlanth** is that of Judd, Crosswhite, and Crosswhite [JCC68]. If the spin-orbit correction arrived from the influence that the orbital motion of an electron has on its own magnetic moment, the spin-other-orbit reflects the interaction that the motion of one electron has on the magnetic moment of another. Much as the spin-orbit effect can be extracted as a relativistic correction with the Dirac equation as the starting point. The multi-electron spin-orbit effects can be derived from the Breit operator [BS57] which is added to the relativistic description of a many-particle system in order to account for retardation of the electromagnetic field

$$\hat{\mathcal{H}}_B = -\frac{1}{2} e^2 \sum_{i>j} \left[(\alpha_i \cdot \alpha_j) \frac{1}{r_{ij}} + (\alpha_i \cdot \vec{r}_{ij}) (\alpha_j \cdot \vec{r}_{ij}) \frac{1}{r_{ij}^3} \right]. \tag{43}$$

When this operator is expanded in powers of v/c, a number of non-relativistic inter-electron interactions result. Two of them being the spin-other-orbit and spin-spin interactions.

As usual, the radial part of the Hamiltonian is averaged, which in this case gives appearance to the Marvin integrals

$$M^{(k)} := \frac{e^2 \hbar^2}{8 m^2 c^2} \langle (nl)^2 | \frac{r_<^k}{r_>^{k+3}} | (nl)^2 \rangle \tag{44}$$

With these, the expression for the spin-spin term is [JCC68]

$$\hat{\mathcal{H}}_{s:s} = -2 \sum_{i \neq j} \sum_k M^{(k)} \sqrt{(k+1)(k+2)(2k+3)} \langle \underline{\ell} \| C^{(k)} \| \underline{\ell} \rangle \langle \underline{\ell} \| C^{(k+2)} \| \underline{\ell} \rangle \left\{ \hat{w}_i^{(1,k)} \hat{w}_j^{(1,k+2)} \right\}^{(2,2)0} \tag{45}$$

and the one for spin-other-orbit

$$\begin{aligned} \hat{\mathcal{H}}_{s:oo} = \sum_{i \neq j} \sum_k & \sqrt{(k+1)(2\underline{\ell}+k+2)(2\underline{\ell}-k)} \times \\ & \left[\left\{ \hat{w}_i^{(0,k+1)} \hat{w}_j^{(1,k)} \right\}^{(11)0} \left\{ M^{(k-1)} \langle \underline{\ell} \| C^{(k+1)} \| \underline{\ell} \rangle^2 + 2 M^{(k)} \langle \underline{\ell} \| C^{(k)} \| \underline{\ell} \rangle^2 \right\} + \right. \\ & \left. \left\{ \hat{w}_i^{(0,k)} \hat{w}_j^{(1,k+1)} \right\}^{(11)0} \left\{ M^{(k)} \langle \underline{\ell} \| C^{(k)} \| \underline{\ell} \rangle^2 + 2 M^{(k-1)} \langle \underline{\ell} \| C^{(k+1)} \| \underline{\ell} \rangle^2 \right\} \right]. \end{aligned} \tag{46}$$

In the expressions above $\hat{w}_i^{(\kappa,k)}$ is a double tensor operator of rank κ over spin, of rank k over orbit, and acting on electron i. It is defined by its reduced matrix elements as

$$\langle \underline{\ell} \| \hat{w}^{(\kappa,k)} \| \underline{\ell} \rangle = \sqrt{[\![\kappa]\!]\,[\![k]\!]} \langle\!\rangle(l,\kappa,l) \langle\!\rangle(l,k,l) \tag{47}$$

The complexity of the above expressions for can be identified by identifying them with the scalar part of two new double tensors $\hat{\mathcal{T}}_0^{(11)}$ and $\hat{\mathcal{T}}_0^{(22)}$ such that

$$\sqrt{5}\hat{\mathcal{T}}_0^{(22)} \coloneqq \hat{\mathcal{H}}_{\text{s:s}} \tag{48}$$

$$-\sqrt{3}\hat{\mathcal{T}}_0^{(11)} \coloneqq \hat{\mathcal{H}}_{\text{s:oo}} \tag{49}$$

In terms of which the reduced matrix elements in the $|LSJ\rangle$ basis can be obtained by

$$\langle \gamma SLJ | \hat{\mathcal{H}} | \gamma' S' L' J' \rangle = \delta(J,J') \begin{Bmatrix} S' & L' & J \\ L & S & t \end{Bmatrix} \langle \gamma SL \| \hat{\mathcal{T}}^{(tt)} \| \gamma' S' L' \rangle. \tag{50}$$

This above relationship is used in **qlanth** in the functions SpinSpin and SOOandECSO.

```
SpinSpin::usage="SpinSpin[n, SL, SpLp, J] returns the matrix element <|
    SL,J|spin-spin|SpLp,J|> for the spin-spin operator within the
    configuration f^n. This matrix element is independent of MJ. This
    is obtained by querying the relevant reduced matrix element by
    querying the association T22Table and putting in the adequate phase
     and 6-j symbol.
This is calculated according to equation (3) in \"Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
    Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
    130.\"
\".
";
SpinSpin[numE_, SL_, SpLp_, J_] := Module[
  {S, L, Sp, Lp, α, val},
  α = 2;
  {S, L} = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  val = (
           Phaser[Sp + L + J] *
           SixJay[{Sp, Lp, J}, {L, S, α}] *
           T22Table[{numE, SL, SpLp}]
         );
  Return[val]
  ];
```

```
SOOandECSO::usage="SOOandECSO[n, SL, SpLp, J] returns the matrix
    element <|SL,J|spin-spin|SpLp,J|> for the combined effects of the
    spin-other-orbit interaction and the electrostatically-correlated-
    spin-orbit (which originates from configuration interaction effects
    ) within the configuration f^n. This matrix element is independent
    of MJ. This is obtained by querying the relevant reduced matrix
    element by querying the association SOOandECSOLSTable and putting
    in the adequate phase and 6-j symbol. The SOOandECSOLSTable puts
    together the reduced matrix elements from three operators.
This is calculated according to equation (3) in \"Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
    Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
    130.\".
";
SOOandECSO[numE_, SL_, SpLp_, J_]:= Module[
  {S, Sp, L, Lp, α, val},
  α = 1;
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  val = (
           Phaser[Sp + L + J] *
           SixJay[{Sp, Lp, J}, {L, S, α}] *
           SOOandECSOLSTable[{numE, SL, SpLp}]
         );
  Return[val];
]
```

For two-electron operators such as these, the matrix elements in $\underline{\mathfrak{f}}^n$ are related to those in $\underline{\mathfrak{f}}^{n-1}$ via:

$$\langle \underline{\mathfrak{f}}^n \psi \| \hat{\mathcal{T}}^{(tt)} \| \psi' \underline{\mathfrak{f}}^n \rangle = \frac{n}{n-2} \sum_{\bar{\psi},\bar{\psi}'} (-1)^{\bar{S}+\bar{L}+\underline{\mathfrak{s}}+\underline{\ell}+S'+L'} \sqrt{[\![S]\!]\,[\![S']\!]\,[\![L]\!]\,[\![L']\!]} \times$$

$$\left(\psi \left\{ \bar{\psi} \right) \left(\psi' \left\{ \bar{\psi}' \right) \begin{Bmatrix} S & t & S' \\ \bar{S}' & \underline{\mathfrak{s}} & \bar{S} \end{Bmatrix} \begin{Bmatrix} L & t & L' \\ \bar{L}' & \underline{\ell} & \bar{L} \end{Bmatrix} \right. \tag{51}$$

Where the sum runs over the terms $\bar{\psi}$ and $\bar{\psi}'$ in $\underline{\mathrm{f}}^{n-1}$ which are parents common to ψ and ψ'. Using these the matrix elements of $\hat{\mathcal{T}}^{(11)}$ and $\hat{\mathcal{T}}^{(22)}$ in $\underline{\mathrm{f}}^2$ can be used to compute all the reduced matrix elements in $\underline{\mathrm{f}}^n$. These could then be used, together with &eqn-50 to obtain the matrix elements of $\hat{\mathcal{H}}_{\mathrm{s:s}}$ and $\hat{\mathcal{H}}_{\mathrm{s:oo}}$. This is done for $\hat{\mathcal{H}}_{\mathrm{s:s}}$, but not for $\hat{\mathcal{H}}_{\mathrm{s:oo}}$, since this term is traditionally computed at the same as the electrostically-correlated-spin-orbit (see next section).

These equations are implemented in **qlanth** through the following functions: GenerateT22Table, ReducedT22infn, ReducedT22inf2, ReducedT11inf2. Where ReducedT22inf2 and ReducedT11inf2 provide the reduced matrix elements for $\hat{\mathcal{T}}^{(11)}$ and $\hat{\mathcal{T}}^{(22)}$ in $\underline{\mathrm{f}}^2$ as provided in table II of [JCC68].

```
GenerateT22Table::usage="GenerateT22Table[nmax] generates the LS
    reduced matrix elements for the double tensor operator T22 in f^n
    up to n=nmax. If the option \"Export\" is set to true then the
    resulting association is saved to the data folder. The values for n
    =1 and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
     Physical Review 169, no. 1 (1968): 130.\", and the values for n>2
    are calculated recursively using equation (4) of that same paper.
This is an intermediate step to the calculation of the reduced matrix
    elements of the spin-spin operator.";
Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
GenerateT22Table[nmax_Integer, OptionsPattern[]]:= (
  If[And[OptionValue["Progress"], frontEndAvailable],
    (
      numItersai = Association[Table[numE->Length[AllowedNKSLTerms[numE
    ]]^2, {numE, 1, nmax}]];
      counters = Association[Table[numE->0, {numE, 1, nmax}]];
      totalIters = Total[Values[numItersai[[1;;nmax]]]];
      template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
      template2 = StringTemplate["`remtime` min remaining"];template3 =
     StringTemplate["Iteration speed = `speed` ms/it"];
      template4 = StringTemplate["Time elapsed = `runtime` min"];
      progBar = PrintTemporary[
        Dynamic[
          Pane[
            Grid[{{Superscript["f", numE]},
                  {template1[<|"numiter"->numiter, "totaliter"->
    totalIters|>]},
                  {template4[<|"runtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                  {template2[<|"remtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
    ]], 0.1]|>]},
                  {template3[<|"speed"->Round[QuantityMagnitude[Now-
    startTime, "ms"]/(numiter), 0.01]|>]},
                  {ProgressIndicator[Dynamic[numiter], {1, totalIters
    }]}},
                  Frame->All],
                  Full,
                  Alignment->Center]
              ]
            ];
    )
  ];
  T22Table = <||>;
  startTime = Now;
  numiter = 1;
  Do[
    (
      numiter+= 1;
      T22Table[{numE, SL, SpLp}] = Which[
        numE==1,
        0,
        numE==2,
        SimplifyFun[ReducedT22inf2[SL, SpLp]],
        True,
        SimplifyFun[ReducedT22infn[numE,  SL, SpLp]]
      ];
    ),
  {numE, 1, nmax},
  {SL,   AllowedNKSLTerms[numE]},
  {SpLp, AllowedNKSLTerms[numE]}
  ];
  If[And[OptionValue["Progress"],frontEndAvailable],
    NotebookDelete[progBar]
  ];
```

```
  If[OptionValue["Export"],
    (
      fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
      Export[fname, T22Table];
    )
  ];
  Return[T22Table];
);
```

```
ReducedT22infn::usage="ReducedT22infn[n, SL, SpLp] calculates the
    reduced matrix element of the T22 operator for the f^n
    configuration corresponding to the terms SL and SpLp. This is the
    operator corresponding to the inter-electron between spin.
It does this by using equation (4) of \"Judd, BR, HM Crosswhite, and
    Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
    Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
";
ReducedT22infn[numE_, SL_, SpLp_]:= Module[
  {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp, parentSL
    , parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
  {spin, orbital} = {1/2, 3};
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  t = 2;
  cfpSL    = CFP[{numE, SL}];
  cfpSpLp  = CFP[{numE, SpLp}];
  Tnkk =
    Sum[(
      parentSL = cfpSL[[idx2, 1]];
      parentSpLp = cfpSpLp[[idx1, 1]];
      {Sb, Lb} = FindSL[parentSL];
      {Sbp, Lbp} = FindSL[parentSpLp];
      phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
      (
        phase *
        cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
        SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
        SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
        T22Table[{numE - 1, parentSL, parentSpLp}]
      )
    ),
    {idx1, 2, Length[cfpSpLp]},
    {idx2, 2, Length[cfpSL]}
    ];
  Tnkk *= numE / (numE - 2) * Sqrt[TPO[S,Sp,L,Lp]];
  Return[Tnkk];
  ];
```

```
ReducedT22inf2::usage="ReducedT22inf2[SL, SpLp] returns the reduced
    matrix element of the scalar component of the double tensor T22 for
     the terms SL, SpLp in f^2.
Data used here for m0, m2, m4 is from Table I of Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
    Interactions for f Electrons. Physical Review 169, no. 1 (1968):
    130.
";
ReducedT22inf2[SL_, SpLp_] :=
  Module[{statePosition, PsiPsipStates, m0, m2, m4, Tkk2m},
  T22inf2 = <|
  {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
  {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
  {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
  {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
  {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
  |>;
  Which[
    MemberQ[Keys[T22inf2],{SL,SpLp}],
      Return[T22inf2[{SL,SpLp}]],
    MemberQ[Keys[T22inf2],{SpLp,SL}],
      Return[T22inf2[{SpLp,SL}]],
    True,
      Return[0]
  ]
  ];
```

```
Reducedt11inf2::usage="Reducedt11inf2[SL, SpLp] returns the reduced
    matrix element in f^2 of the double tensor operator t11 for the
    corresponding given terms {SL, SpLp}.
Values given here are those from Table VII of \"Judd, BR, HM Crosswhite
    , and Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
     Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
";
Reducedt11inf2[SL_, SpLp_]:= Module[
  {t11inf2},
  t11inf2 = <|
    {"1S", "3P"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
    {"3P", "3P"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
    {"3P", "1D"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
    {"1D", "3F"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
    {"3F", "3F"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
    {"3F", "1G"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
    {"1G", "3H"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
    {"3H", "3H"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
    {"3H", "1I"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
    |>;
  Which[
    MemberQ[Keys[t11inf2],{SL,SpLp}],
      Return[t11inf2[{SL,SpLp}]],
    MemberQ[Keys[t11inf2],{SpLp,SL}],
      Return[t11inf2[{SpLp,SL}]],
    True,
      Return[0]
  ]
]
```

5.9 $\hat{\mathcal{H}}_{\text{ecs:o}}$: electrostatically-correlated-spin-orbit and a note about CI

In the same paper [JCC68] that describes the spin-spin and spin-other-orbits consideration is also given to the emergence of additional corrections due to configuration interaction as described by the following operator (page. 169 of [JCC68])

$$\hat{\mathcal{H}}_{\text{ci}} = -\sum_{\chi}\sum_{i}\frac{1}{E_\chi}\xi(r_i)\left(\underline{\hat{s}}_i\cdot\underline{\hat{\ell}}_i\right)|\chi\rangle\langle\chi|\hat{\mathfrak{C}} - \frac{1}{E_\chi}\hat{\mathfrak{C}}|\chi\rangle\langle\chi|\xi(r_i)\left(\underline{\hat{s}}_i\cdot\underline{\hat{\ell}}_i\right) \tag{52}$$

where $\xi(r_h)(\underline{\hat{s}}_h\cdot\underline{\hat{\ell}}_h)$ is the customary spin-orbit interaction, E_χ is the energy of state $|\chi\rangle$, i is a label for the valence electrons, and $|\chi\rangle$ are states in the configurations to which one is "interacting" with.

Most importantly in the above, the term $\hat{\mathfrak{C}}$ stands for the non-central part of the Coulomb interaction. This serves as a reminder that the central field approximation of single-electron wavefunctions is, indeed, an approximation. The non-central part of the electrostatic field is defined as what remains after subtracting the radial component. This term is crucial to keep in mind because it facilitates parity-breaking transitions, such as forced electric dipole transitions. Moreover, the non-central nature of this term plays a significant role in configuration mixing (see [MR71]), which is why the operator $\hat{\mathfrak{C}}$ is prominently featured in the expression for configuration interaction, and why the modifier "electrostatically correlated" is prepended to spin-orbit to form "electrostatically correlated spin orbit". It's also worth keeping in mind that the derivation of such an expression is based on second-order perturbation theory.

This operator can be identified with a it being the scalar component of a double tensor operator of rank 1 both for the spin and orbital parts of the wavefunction.

$$\hat{\mathcal{H}}_{\text{ci}} = -\sqrt{3}\,\hat{t}_0^{(11)} \tag{53}$$

Judd *et al.* then go on to list the reduced matrix elements of this operator in the $\underline{f}^2$ configuration. When this is done the Marvin integrals $M^{(k)}$ appear again, but a second set of parameters is also necessary

$$P^{(k)} = 6\sum_{f'}\frac{\zeta_{ff'}}{E_{ff'}}R^{(k)}(ff,ff') \text{ for } k=0,2,4,6. \tag{54}$$

Where f notes the radial eigenfunction attached to an f-electron wavefunction, and f' similarly but for a configuration different from $\underline{f}^n$. And where

$$\zeta_{ff'} \coloneqq \langle f|\xi(r)|f'\rangle \tag{55}$$

$$R^{(k)}(ff,ff') \coloneqq e^2\langle f_1 f_2|\frac{r_<^k}{r_>^{k+1}}|f_1 f_2'\rangle. \tag{56}$$

In the semi-empirical approach embodied by **qlanth** , calculating these quantities *ab initio* is not the objective, rendering the precise definition of these parameters non-essential. Nonetheless,

these expressions frequently serve to justify the ratios between different orders of these quantities. Consequently, both the set of three $M^{(k)}$ and the set of $P^{(k)}$ ultimately rely on a single free parameter each. Such parsimony is desirable given the large number of parameters (about 20) that the Hamiltonian ends up having.

Judd *et al.* further note that $P^{(0)}$ is proportional to the spin orbit operator, and as such its effect is absorbed by the standard spin-orbit parameter ζ.

Judd *et al.* also develop an alternative approach based on group theory arguments. They put together the spin-other-orbit and the electrostatically-correlated-spin-orbit as a sum of operators $\hat{z}_i$ with useful transformation rules

$$\langle\psi\|\hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)}\|\psi'\rangle = \sum a_i \langle\psi\|\hat{z}_i\|\psi'\rangle \tag{57}$$

At this stage a subtle point needs to be raised. As Judd points out, in the sum above, the term $\hat{z}_{13}$ that contributes with a tensorial character equal to that of the regular spin-orbit operator. As such, if the goal is obtaining a parametric Hamiltonian that can be fit with uncorrelated parameters, it is then necessary to subtract this part from $\hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)}$. This point was clarified by Chen *et al.* [Che+08]. Because of this the final form of the operator contributing both to spin-other-orbit and the electrostatically-correlated-spin-orbit is:

$$\hat{\mathcal{H}}_{\text{s:oo}} + \hat{\mathcal{H}}_{\text{ecs:o}} = \hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)} - \frac{1}{6} a_{13} \hat{z}_{13} \tag{58}$$

where

$$a_{13} = -33M^{(0)} + 3M^{(2)} + 15/11M^{(4)} - 6P^{(0)} + 3/2(35P^{(2)} + 77P^{(4)} + 143P^{(6)}) \tag{59}$$

In **qlanth** the contributions from spin-spin, spin-other-orbit, and electrostatically-correlated-spin-orbit are put together by the function `MagneticInteractions`. That function queries precomputed values from two associations `SpinSpinTable` and `SOOandECSOTable`. In turn these two associations are generated by the functions `GenerateSpinOrbitTable` and `GenerateSOOandECSOTable`. Note that both spin-spin and spin-other-orbit end up contributing through $M^{(k)}$, however there doesn't seem to be consensus about adding them together, as such **qlanth** allows including or excluding the spin-spin contribution, this is done with a control parameter σ_{SS}.

```
MagneticInteractions::usage="MagneticInteractions[{numE, SLJ, SLJp, J}]
    returns the matrix element of the magnetic interaction between the
    terms SLJ and SLJp in the f^n configuration. The interaction is
   given by the sum of the spin-spin interaction and the SOO and ECSO
   interactions. The spin-spin interaction is given by the function
   SpinSpin[{numE, SLJ, SLJp, J}]. The SOO and ECSO interactions are
   given by the function SOOandECSO[{numE, SLJ, SLJp, J}]. The
   function requires chenDeltas to be loaded into the session. The
   option \"ChenDeltas\" can be use to include or exclude the Chen
   deltas from the calculation. The default is to exclude them.";
Options[MagneticInteractions] = {"ChenDeltas" -> False};
MagneticInteractions[{numE_, SLJ_, SLJp_, J_}, OptionsPattern[]] :=
  (
    key = {numE, SLJ, SLJp, J};
    ss = \[Sigma]SS * SpinSpinTable[key];
    sooandecso = SOOandECSOTable[key];
    total = ss + sooandecso;
    total = SimplifyFun[total];
    If[
      Not[OptionValue["ChenDeltas"]],
      Return[total]
    ];
    (* In the type A errors the wrong values are different *)
    If[MemberQ[Keys[chenDeltas["A"]], {numE, SLJ, SLJp}],
      (
        {S, L} = FindSL[SLJ];
        {Sp, Lp} = FindSL[SLJp];
        phase   = Phaser[Sp + L + J];
        Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
        Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
        {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ,
   SLJp}]["wrong"];
        total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
        total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
        total  = total /. Prescaling;
        total  = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
      )
    ];
    (* In the type B errors the wrong values are zeros all around *)
    If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
      (
        {S, L} = FindSL[SLJ];
```

```
        {Sp, Lp} = FindSL[SLJp];
        phase = Phaser[Sp + L + J];
        Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
        Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
        {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
        total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
        total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
        total  = total /. Prescaling;
        total  = wChErrB * total + (1 - wChErrB) * (ss + sooandecso)
      )
    ];
    Return[total];
  )
```

```
GenerateSpinOrbitTable::usage = "GenerateSpinOrbitTable[nmax] computes
    the matrix values for the spin-orbit interaction for f^n
    configurations up to n = nmax. The function returns an association
    whose keys are lists of the form {n, SL, SpLp, J}. If export is set
     to True, then the result is exported to the data subfolder for the
     folder in which this package is in. It requires ReducedV1kTable to
     be defined.";
Options[GenerateSpinOrbitTable] = {"Export" -> True};
GenerateSpinOrbitTable[nmax_Integer:7, OptionsPattern[]]:= Module[
  {numE, J, SL, SpLp, exportFname},
(
  SpinOrbitTable =
    Table[
      {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
    {numE, 1, nmax},
    {J, MinJ[numE], MaxJ[numE]},
    {SL,   Map[First, AllowedNKSLforJTerms[numE, J]]},
    {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
    ];
  SpinOrbitTable = Association[SpinOrbitTable];
```

```
GenerateSOOandECSOTable::usage="GenerateSOOandECSOTable[nmax] generates
     the matrix elements in the |LSJ> basis for the (spin-other-orbit +
     electrostatically-correlated-spin-orbit) operator. It returns an
    association where the keys are of the form {n, SL, SpLp, J}. If the
     option \"Export\" is set to True then the resulting object is
    saved to the data folder. Since this is a scalar operator, there is
     no MJ dependence. This dependence only comes into play when the
    crystal field contribution is taken into account.";
Options[GenerateSOOandECSOTable] = {"Export"->False}
GenerateSOOandECSOTable[nmax_, OptionsPattern[]]:= (
  SOOandECSOTable = <||>;
  Do[
    SOOandECSOTable[{numE, SL, SpLp, J}] = (SOOandECSO[numE, SL, SpLp,
    J] /. Prescaling);,
  {numE, 1, nmax},
  {J, MinJ[numE], MaxJ[numE]},
  {SL,   First /@ AllowedNKSLforJTerms[numE, J]},
  {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
  ];
  If[OptionValue["Export"],
  (
    fname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
    Export[fname, SOOandECSOTable];
  )
  ];
  Return[SOOandECSOTable];
);
```

The function GenerateSpinSpinTable calls the function SpinSpin over all possible combinations of the arguments $\{n, SL, S'L', J\}$. In turn the function SpinSpin queries the precomputed values of the the double tensor $\hat{\mathcal{T}}^{(22)}$ which are stored in the association T22Table.

```
GenerateSpinSpinTable::usage="GenerateSpinSpinTable[nmax] generates the
     matrix elements in the |LSJ> basis for the (spin-other-orbit +
    electrostatically-correlated-spin-orbit) operator. It returns an
    association where the keys are of the form {numE, SL, SpLp, J}. If
    the option \"Export\" is set to True then the resulting object is
    saved to the data folder. Since this is a scalar operator, there is
     no MJ dependence. This dependence only comes into play when the
    crystal field contribution is taken into account.";
Options[GenerateSpinSpinTable] = {"Export"->False};
```

```
GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
(
  SpinSpinTable = <||>;
  PrintTemporary[Dynamic[numE]];
  Do[
    SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp, J])
    ;,
  {numE, 1, nmax},
  {J, MinJ[numE], MaxJ[numE]},
  {SL,   First /@ AllowedNKSLforJTerms[numE, J]},
  {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
  ];
  If[OptionValue["Export"],
  (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
    Export[fname, SpinSpinTable];
    )
  ];
  Return[SpinSpinTable];
  );
```

```
SpinSpin::usage="SpinSpin[n, SL, SpLp, J] returns the matrix element <|
    SL,J|spin-spin|SpLp,J|> for the spin-spin operator within the
    configuration f^n. This matrix element is independent of MJ. This
    is obtained by querying the relevant reduced matrix element by
    querying the association T22Table and putting in the adequate phase
     and 6-j symbol.
This is calculated according to equation (3) in \"Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
    Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
    130.\"
\".
";
SpinSpin[numE_, SL_, SpLp_, J_] := Module[
  {S, L, Sp, Lp, α, val},
  α = 2;
  {S, L} = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  val = (
          Phaser[Sp + L + J] *
          SixJay[{Sp, Lp, J}, {L, S, α}] *
          T22Table[{numE, SL, SpLp}]
        );
  Return[val]
  ];
```

The association `T22Table` is computed by the function `GenerateT22Table`. This function populates `T22Table` with keys of the form $\{n, SL, S'L'\}$. It does this by using the function `ReducedT22inf2` in the base case of $\underline{f}^2$, and `ReducedT22infn` for configurations above $\underline{f}^2$. When `ReducedT22infn` is called the sum in &qn-51 is carried out using $t = 2$. When `ReducedT22inf2` is called the reduced matrix elements from [JCC68] are used.

```
GenerateT22Table::usage="GenerateT22Table[nmax] generates the LS
    reduced matrix elements for the double tensor operator T22 in f^n
    up to n=nmax. If the option \"Export\" is set to true then the
    resulting association is saved to the data folder. The values for n
    =1 and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
     Physical Review 169, no. 1 (1968): 130.\", and the values for n>2
    are calculated recursively using equation (4) of that same paper.
This is an intermediate step to the calculation of the reduced matrix
    elements of the spin-spin operator.";
Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
GenerateT22Table[nmax_Integer, OptionsPattern[]]:= (
  If[And[OptionValue["Progress"], frontEndAvailable],
    (
      numItersai = Association[Table[numE->Length[AllowedNKSLTerms[numE
    ]]^2, {numE, 1, nmax}]];
      counters = Association[Table[numE->0, {numE, 1, nmax}]];
      totalIters = Total[Values[numItersai[[1;;nmax]]]];
      template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
      template2 = StringTemplate["`remtime` min remaining"];template3 =
     StringTemplate["Iteration speed = `speed` ms/it"];
      template4 = StringTemplate["Time elapsed = `runtime` min"];
      progBar = PrintTemporary[
        Dynamic[
          Pane[
```

```
            Grid[{{Superscript["f", numE]},
                  {template1[<|"numiter"->numiter, "totaliter"->
    totalIters|>]},
                  {template4[<|"runtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                  {template2[<|"remtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
    ]], 0.1]|>]},
                  {template3[<|"speed"->Round[QuantityMagnitude[Now-
    startTime, "ms"]/(numiter), 0.01]|>]},
                  {ProgressIndicator[Dynamic[numiter], {1, totalIters
    }]}},
                  Frame->All],
                  Full,
                  Alignment->Center]
               ]
            ];
    )
  ];
  T22Table = <||>;
  startTime = Now;
  numiter = 1;
  Do[
    (
      numiter+= 1;
      T22Table[{numE, SL, SpLp}] = Which[
        numE==1,
        0,
        numE==2,
        SimplifyFun[ReducedT22inf2[SL, SpLp]],
        True,
        SimplifyFun[ReducedT22infn[numE,  SL, SpLp]]
      ];
    ),
  {numE, 1, nmax},
  {SL,   AllowedNKSLTerms[numE]},
  {SpLp, AllowedNKSLTerms[numE]}
  ];
  If[And[OptionValue["Progress"],frontEndAvailable],
    NotebookDelete[progBar]
  ];
  If[OptionValue["Export"],
    (
      fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
      Export[fname, T22Table];
    )
  ];
  Return[T22Table];
);
```

```
ReducedT22infn::usage="ReducedT22infn[n, SL, SpLp] calculates the
    reduced matrix element of the T22 operator for the f^n
    configuration corresponding to the terms SL and SpLp. This is the
    operator corresponding to the inter-electron between spin.
It does this by using equation (4) of \"Judd, BR, HM Crosswhite, and
    Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
    Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
";
ReducedT22infn[numE_, SL_, SpLp_]:= Module[
  {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp, parentSL
    , parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
  {spin, orbital} = {1/2, 3};
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  t = 2;
  cfpSL    = CFP[{numE, SL}];
  cfpSpLp  = CFP[{numE, SpLp}];
  Tnkk =
    Sum[(
      parentSL = cfpSL[[idx2, 1]];
      parentSpLp = cfpSpLp[[idx1, 1]];
      {Sb, Lb} = FindSL[parentSL];
      {Sbp, Lbp} = FindSL[parentSpLp];
      phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
      (
        phase *
```

```
        cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
        SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
        SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
        T22Table[{numE - 1, parentSL, parentSpLp}]
      )
    ),
    {idx1, 2, Length[cfpSpLp]},
    {idx2, 2, Length[cfpSL]}
    ];
  Tnkk *= numE / (numE - 2) * Sqrt[TPO[S,Sp,L,Lp]];
  Return[Tnkk];
  ];
```

```
ReducedT22inf2::usage="ReducedT22inf2[SL, SpLp] returns the reduced
    matrix element of the scalar component of the double tensor T22 for
     the terms SL, SpLp in f^2.
Data used here for m0, m2, m4 is from Table I of Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
    Interactions for f Electrons. Physical Review 169, no. 1 (1968):
    130.
";
ReducedT22inf2[SL_, SpLp_] :=
  Module[{statePosition, PsiPsipStates, m0, m2, m4, Tkk2m},
  T22inf2 = <|
  {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
  {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
  {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
  {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
  {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
  |>;
  Which[
    MemberQ[Keys[T22inf2],{SL,SpLp}],
      Return[T22inf2[{SL,SpLp}]],
    MemberQ[Keys[T22inf2],{SpLp,SL}],
      Return[T22inf2[{SpLp,SL}]],
    True,
      Return[0]
  ]
  ];
```

The function GenerateSOOandECSOTable calls the function SOOandECSO over all possible combinations of the arguments $\{n, SL, S'L', J\}$ and uses their values to populate the association SOOandECSOTable. In turn the function SOOandECSO queries the precomputed values of &qn-58 as stored in the association SOOandECSOLSTable.

```
GenerateSOOandECSOTable::usage="GenerateSOOandECSOTable[nmax] generates
     the matrix elements in the |LSJ> basis for the (spin-other-orbit +
     electrostatically-correlated-spin-orbit) operator. It returns an
    association where the keys are of the form {n, SL, SpLp, J}. If the
     option \"Export\" is set to True then the resulting object is
    saved to the data folder. Since this is a scalar operator, there is
     no MJ dependence. This dependence only comes into play when the
    crystal field contribution is taken into account.";
Options[GenerateSOOandECSOTable] = {"Export"->False}
GenerateSOOandECSOTable[nmax_, OptionsPattern[]]:= (
  SOOandECSOTable = <||>;
  Do[
    SOOandECSOTable[{numE, SL, SpLp, J}] = (SOOandECSO[numE, SL, SpLp,
    J] /. Prescaling);,
  {numE, 1, nmax},
  {J, MinJ[numE], MaxJ[numE]},
  {SL,   First /@ AllowedNKSLforJTerms[numE, J]},
  {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
  ];
  If[OptionValue["Export"],
  (
    fname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
    Export[fname, SOOandECSOTable];
  )
  ];
  Return[SOOandECSOTable];
);
```

```
SOOandECSO::usage="SOOandECSO[n, SL, SpLp, J] returns the matrix
    element <|SL,J|spin-spin|SpLp,J|> for the combined effects of the
    spin-other-orbit interaction and the electrostatically-correlated-
```

```
    spin-orbit (which originates from configuration interaction effects
    ) within the configuration f^n. This matrix element is independent
    of MJ. This is obtained by querying the relevant reduced matrix
    element by querying the association SOOandECSOLSTable and putting
    in the adequate phase and 6-j symbol. The SOOandECSOLSTable puts
    together the reduced matrix elements from three operators.
This is calculated according to equation (3) in \"Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
    Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
    130.\".
";
SOOandECSO[numE_, SL_, SpLp_, J_]:= Module[
  {S, Sp, L, Lp, α, val},
  α = 1;
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  val = (
           Phaser[Sp + L + J] *
           SixJay[{Sp, Lp, J}, {L, S, α}] *
           SOOandECSOLSTable[{numE, SL, SpLp}]
         );
  Return[val];
]
```

```
SOOandECSO::usage="SOOandECSO[n, SL, SpLp, J] returns the matrix
    element <|SL,J|spin-spin|SpLp,J|> for the combined effects of the
    spin-other-orbit interaction and the electrostatically-correlated-
    spin-orbit (which originates from configuration interaction effects
    ) within the configuration f^n. This matrix element is independent
    of MJ. This is obtained by querying the relevant reduced matrix
    element by querying the association SOOandECSOLSTable and putting
    in the adequate phase and 6-j symbol. The SOOandECSOLSTable puts
    together the reduced matrix elements from three operators.
This is calculated according to equation (3) in \"Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
    Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
    130.\".
";
SOOandECSO[numE_, SL_, SpLp_, J_]:= Module[
  {S, Sp, L, Lp, α, val},
  α = 1;
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  val = (
           Phaser[Sp + L + J] *
           SixJay[{Sp, Lp, J}, {L, S, α}] *
           SOOandECSOLSTable[{numE, SL, SpLp}]
         );
  Return[val];
]
```

The association `SOOandECSOLSTable` is computed by the function `GenerateSOOandECSOLSTable`. This function populates `SOOandECSOLSTable` with keys of the form $\{n, SL, S'L'\}$. It does this by using the function `ReducedSOOandECSOinf2` in the base case of $\underline{f}^2$, and `ReducedSOOandECSOinfn` for configurations above $\underline{f}^2$. When `ReducedSOOandECSOinfn` is called the sum in &qn-51 is carried out using $t = 1$. When `ReducedSOOandECSOinf2` is called the reduced matrix elements from [JCC68] are used.

```
ReducedSOOandECSOinfn::usage="ReducedSOOandECSOinfn[numE, SL, SpLp]
    calculates the reduced matrix elements of the (spin-other-orbit +
    ECSO) operator for the f^n configuration corresponding to the terms
     SL and SpLp. This is done recursively, starting from tabulated
    values for f^2 from \"Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
     Physical Review 169, no. 1 (1968): 130.\", and by using equation
    (4) of that same paper.
";
ReducedSOOandECSOinfn[numE_, SL_, SpLp_]:= Module[
  {spin, orbital, t, S, L, Sp, Lp, idx1, idx2, cfpSL, cfpSpLp, parentSL
    , Sb, Lb, Sbp, Lbp, parentSpLp, funval},
  {spin, orbital} = {1/2, 3};
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  t = 1;
  cfpSL    = CFP[{numE, SL}];
  cfpSpLp  = CFP[{numE, SpLp}];
```

```
  funval =
    Sum[
      (
        parentSL = cfpSL[[idx2, 1]];
        parentSpLp = cfpSpLp[[idx1, 1]];
        {Sb, Lb}   = FindSL[parentSL];
        {Sbp, Lbp} = FindSL[parentSpLp];
        phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
        (
          phase *
          cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
          SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
          SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
          SOOandECSOLSTable[{numE - 1, parentSL, parentSpLp}]
        )
      ),
    {idx1, 2, Length[cfpSpLp]},
    {idx2, 2, Length[cfpSL]}
    ];
  funval *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
  Return[funval];
];
```

```
ReducedSOOandECSOinf2::usage="ReducedSOOandECSOinf2[SL, SpLp] returns
    the reduced matrix element corresponding to the operator (T11 + t11
     - a13 * z13 / 6) for the terms {SL, SpLp}. This combination of
    operators corresponds to the spin-other-orbit plus ECSO interaction
    .
The T11 operator corresponds to the spin-other-orbit interaction, and
    the t11 operator (associated with electrostatically-correlated spin
    -orbit) originates from configuration interaction analysis. To
    their sum the a facor proportional to operator z13 is subtracted
    since its effect is seen as redundant to the spin-orbit interaction
    . The factor of 1/6 is not on Judd's 1966 paper, but it is on \"
    Chen, Xueyuan, Guokui Liu, Jean Margerie, and Michael F Reid. \"A
    Few Mistakes in Widely Used Data Files for Fn Configurations
    Calculations.\" Journal of Luminescence 128, no. 3 (2008): 421-27\"
    .

The values for the reduced matrix elements of z13 are obtained from
    Table IX of the same paper. The value for a13 is from table VIII.";
ReducedSOOandECSOinf2[SL_, SpLp_] :=
Module[{a13, z13, z13inf2, matElement, redSOOandECSOinf2},
  a13 = (-33 M0 + 3 M2 + 15/11 M4 -
        6 P0 + 3/2 (35 P2 + 77 P4 + 143 P6));
  z13inf2 = <|
      {"1S","3P"} -> 2,
      {"3P","3P"} -> 1,
      {"3P","1D"} -> -Sqrt[(15/2)],
      {"1D","3F"} -> Sqrt[10],
      {"3F","3F"} -> Sqrt[14],
      {"3F","1G"} -> -Sqrt[11],
      {"1G","3H"} -> Sqrt[10],
      {"3H","3H"} -> Sqrt[55],
      {"3H","1I"} -> -Sqrt[(13/2)]
      |>;
  matElement = Which[
    MemberQ[Keys[z13inf2],{SL,SpLp}],
      z13inf2[{SL,SpLp}],
    MemberQ[Keys[z13inf2],{SpLp,SL}],
      z13inf2[{SpLp,SL}],
    True,
      0
  ];
  redSOOandECSOinf2 = (
      ReducedT11inf2[SL, SpLp] +
      Reducedt11inf2[SL, SpLp] -
      a13 / 6 * matElement
  );
  redSOOandECSOinf2 = SimplifyFun[redSOOandECSOinf2];
  Return[redSOOandECSOinf2];
  ];
```

5.10 $\hat{\mu}$ and $\hat{\mathscr{H}}_{\mathbf{Z}}$: the magnetic dipole operator and the Zeeman term

In atomic units, the operator associated with the magnetic dipole is

$$\hat{\mu} = -\mu_{\mathrm{B}} \left(\hat{L} + g_s \hat{S}\right)^{(1)}, \text{with } \mu_{\mathrm{B}} = 1/2. \tag{60}$$

Here we have emphasized the fact that the magnetic dipole operator corresponds to a rank-1 spherical tensor operator.

In the $|LSJM_J\rangle$ basis that we use in **qlanth** the LSJ reduced-matrix elements are computed using eqn. 15.7 in [Cow81]

$$\langle \alpha LSJ \| \left(\hat{L} + g_s \hat{S}\right)^{(1)} \| \alpha' L'S'J' \rangle = \delta(\alpha LSJ, \alpha' L'S'J') \sqrt{J(J+1)(2J+1)} + \\ \delta(\alpha LS, \alpha' L'S') (-1)^{L+S+J+1} \sqrt{[\![J]\!]\,[\![J]\!]} \begin{Bmatrix} L & S & J \\ 1 & J' & S \end{Bmatrix} \tag{61}$$

And then those reduced matrix elements are used to resolve the M_J components for $q = -1, 0, 1$ through Wigner-Eckart

$$\langle \alpha LSJM_J | \left(\hat{L} + g_s \hat{S}\right)^{(1)}_q | \alpha' L'S'J'M_{J'} \rangle = \\ (-1)^{J-M_J} \begin{pmatrix} J & 1 & J' \\ -M_J & q & M'_J \end{pmatrix} \langle \alpha LSJ \| \left(\hat{L} + g_s \hat{S}\right)^{(1)} \| \alpha' L'S'J' \rangle \tag{62}$$

These two above are put together in `JJBlockMagDip` for given $\{n, J, J'\}$ returning a rank-3 array representing the quantities $\{M_J, M'_J, q\}$.

```
JJBlockMagDip::usage="JJBlockMagDip[numE, J, Jp] returns the LSJ-
    reduced matrix element of the magnetic dipole operator between the
    states with given J and Jp. The option \"Sparse\" can be used to
    return a sparse matrix. The default is to return a sparse matrix.
See eqn 15.7 in TASS.
Here it is provided in atomic units in which the Bohr magneton is 1/2.
\[Mu] = -(1/2) (L + gs S)
We are using the Racah convention for the reduced matrix elements in
    the Wigner-Eckart theorem. See TASS eqn 11.15.
";
Options[JJBlockMagDip]={"Sparse"->True};
JJBlockMagDip[numE_, braJ_, ketJ_, OptionsPattern[]]:=Module[
  {braSLJs,ketSLJs,
  braSLJ,ketSLJ,
  braSL,ketSL,
  braS, braL,
  braMJ, ketMJ,
  matValue,magMatrix},(
  braSLJs  = AllowedNKSLJMforJTerms[numE,braJ];
  ketSLJs  = AllowedNKSLJMforJTerms[numE,ketJ];
  magMatrix  = Table[
    braSL     = braSLJ[[1]];
    ketSL     = ketSLJ[[1]];
    {braS, braL}   = FindSL[braSL];
    {ketS, ketL}   = FindSL[ketSL];
    braMJ       = braSLJ[[3]];
    ketMJ       = ketSLJ[[3]];
    summand1    = If[Or[braJ  != ketJ,
                        braSL != ketSL],
      0,
      Sqrt[braJ(braJ+1)TPO[braJ]]
    ];
    (* looking at the string includes checking L=L' S=S' \alpha=\alpha'
    *)
    summand2 = If[braSL!= ketSL,
      0,
      (gs-1) *
        Phaser[braS+braL+ketJ+1] *
          Sqrt[TPO[braJ]*TPO[ketJ]] *
            SixJay[{braJ,1,ketJ},{braS,braL,braS}] *
              Sqrt[braS(braS+1)TPO[braS]]
    ];
    matValue = summand1 + summand2;
    (* We are using the Racah convention for red matrix elements in
    Wigner-Eckart *)
    threejays = (ThreeJay[{braJ, -braMJ}, {1, #}, {ketJ, ketMJ}] &) /@
    {-1,0,1};
    threejays *= Phaser[braJ-braMJ];
```

```
    matValue  = - 1/2 * threejays * matValue;
    matValue,
  {braSLJ, braSLJs},
  {ketSLJ, ketSLJs}
  ];
  If[OptionValue["Sparse"],
    magMatrix= SparseArray[magMatrix]
  ];
  Return[magMatrix])
];
```

The JJ' blocks that are generated with this function are then put together by MagDipoleMatrixAssembly into the final matrix form and the cartesian components calculated according to

$$\hat{\mu}_x = \frac{\hat{\mu}^{(1)}_{-1} - \hat{\mu}^{(1)}_{+1}}{\sqrt{2}} \tag{63}$$

$$\hat{\mu}_y = i\frac{\hat{\mu}^{(1)}_{-1} + \hat{\mu}^{(1)}_{+1}}{\sqrt{2}} \tag{64}$$

$$\hat{\mu}_z = \hat{\mu}^{(1)}_0 \tag{65}$$

```
MagDipoleMatrixAssembly::usage="MagDipoleMatrixAssembly[numE] returns
    the matrix representation of the operator - 1/2 (L + gs S) in the f
    ^numE configuration. The function returns a list with three
    elements corresponding to the x,y,z components of this operator.
    The option \"FilenameAppendix\" can be used to append a string to
    the filename from which the function imports from in order to patch
     together the array. For numE beyond 7 the function returns the
    same as for the complementary configuration.";
Options[MagDipoleMatrixAssembly]={"FilenameAppendix"->""};
MagDipoleMatrixAssembly[nf_,OptionsPattern[]]:=Module[
  {ImportFun, numE, appendTo, emFname, JJBlockMagDipTable, Js,
    howManyJs, blockOp, rowIdx, colIdx},
  (
  ImportFun = ImportMZip;
  numE      = nf;
  numH      = 14 - numE;
  numE      = Min[numE,numH];
```

Using the cartesian components of the magnetic dipole operator, the matrix elements of the Zeeman term can then be evaluated. This term can be included in the Hamiltonian through an option in HamMatrixAssembly. Since the magnetic dipole operator is calculated in atomic units, and it seeming desirable that the input units of the magnetic field be Tesla, a conversion factor is included so that the final terms be congruent with the energy units assumed in the other terms in the Hamiltonian, namely the pseudo-energy unit Kayser (cm^{-1}). The conversion factor is called TestaToKayser in the file qonstants.m.

5.11 Going beyond f7

In most cases all matrix elements in **qlanth** are only calculated up to and including $\underline{f}^7$. Beyond $\underline{f}^7$ adequate changes of sign are enforced to take into account the equivalence that can be made between $\underline{f}^n$ and $\underline{f}^{14-n}$. This is enforced when the function HamMatrixAssembly is called. In there HoleElectronConjugation is the function responsible for enforcing a global sign flip for the following operators (or equivalently to their accompanying coefficients):

$$\zeta, T^{(2)}, T^{(3)}, T^{(4)}, T^{(6)}, T^{(7)}, T^{(8)}, B^{(k)}_q$$

```
HoleElectronConjugation::usage = "HoleElectronConjugation[params] takes
     the parameters (as an association) that define a configuration and
     converts them so that they may be interpreted as corresponding to
    a complentary hole configuration. Some of this can be simply done
    by changing the sign of the model parameters. In the case of the
    effective three body interaction the relationship is more complex
    and is controlled by the value of the isE variable.";
HoleElectronConjugation[params_] :=
  Module[{newparams = params},
    (
      flipSignsOf = {ζ, T2, T3, T4, T6, T7, T8};
      flipSignsOf = Join[flipSignsOf, cfSymbols];
      flipped =
        Table[(flipper -> - newparams[flipper]),
        {flipper, flipSignsOf}
        ];
      nonflipped =
```

```
        Table[(flipper -> newparams[flipper]),
        {flipper, Complement[Keys[newparams], flipSignsOf]}
        ];
      flippedParams = Association[Join[nonflipped, flipped]];
      flippedParams = Select[flippedParams, FreeQ[#, Missing]&];
      Return[flippedParams];
    )
  ]
```

6 Magnetic Dipole Transitions

qlanth can also calculate magnetic dipole transitions. To this end are the functions MagDipLineStrength, MagDipoleRates, and GroundStateOscillatorStrength.

```
MagDipLineStrength::usage="MagDipLineStrength[theEigensys, numE] takes the eigensystem of an ion and the number numE of f-electrons that correspond to it and it calculates the line strength array Stot.
The option \"Units\" can be set to either \"SI\" (so that the units of the returned array are A/m^2) or to \"Hartree\".
The option \"States\" can be used to limit the states for which the line strength is calculated. The default, All, calculates the line strength for all states. A second option for this is to provide an index labelling a specific state, in which case only the line strengths between that state and all the others are computed.
The returned array should be interpreted in the eigenbasis of the Hamiltonian. As such the element Stot[[i,i]] corresponds to the line strength states |i> and |j>.";
Options[MagDipLineStrength]={"Reload MagOp" -> False, "Units"->"SI", "States" -> All};
MagDipLineStrength[theEigensys_List, numE0_Integer, OptionsPattern[]]:=Module[
  {allEigenvecs, Sx, Sy, Sz, Stot ,factor},
(
  numE = Min[14-numE0, numE0];
  (*If not loaded then load it, *)
  If[Or[
      Not[MemberQ[Keys[magOp], numE]],
      OptionValue["Reload MagOp"]],
  (
    magOp[numE] = ReplaceInSparseArray[#,{gs->2}]& /@ MagDipoleMatrixAssembly[numE];
  )
  ];
  allEigenvecs = Transpose[Last/@theEigensys];
  Which[OptionValue["States"] === All,
    (
      {Sx,Sy,Sz}   = (ConjugateTranspose[allEigenvecs].#.allEigenvecs) & /@ magOp[numE];
      Stot         = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
    ),
    IntegerQ[OptionValue["States"]],
    (
      singleState  = theEigensys[[OptionValue["States"],2]];
      {Sx,Sy,Sz}   = (ConjugateTranspose[allEigenvecs].#.singleState) & /@ magOp[numE];
      Stot         = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
    )
  ];
  Which[
    OptionValue["Units"] == "SI",
      Return[4 \[Mu]B^2 * Stot],
    OptionValue["Units"] == "Hartree",
      Return[Stot],
    True,
    (
      Print["Invalid option for \"Units\". Options are \"SI\" and \"Hartree\"."];
      Abort[];
    )
  ];
)
]
```

```
MagDipoleRates::usage="MagDipoleRates[eigenSys, numE] calculates the magnetic dipole transition rate array for the provided eigensystem.
```

```
    The option \"Units\" can be set to \"SI\" or to \"Hartree\". If
    the option \"Natural Radiative Lifetimes\" is set to true then the
    reciprocal of the rate is returned instead. The energy unit assumed
    in eigenSys is kayser. The returned array should be interpreted in
    the eigenbasis of the Hamiltonian. As such the element AMD[[i,i]]
    corresponds to the transition rate (or the radiative lifetime,
    depending on options) between eigenstates |i> and |j>.";
Options[MagDipoleRates]={"Units"->"SI", "Lifetime"->False};
MagDipoleRates[eigenSys_List, numE0_Integer,OptionsPattern[]]:=Module[
  {AMD,gKramers,Stot,eigenEnergies,transitionWaveLengthsInMeters},(
  numE           = Min[14-numE0, numE0];
  gKramers       = If[OddQ[numE],2,1];
  Stot           = MagDipLineStrength[eigenSys, numE, "Units"->
    OptionValue["Units"]];
  eigenEnergies  = Chop[First/@eigenSys];
  energyDiffs    = Outer[Subtract,eigenEnergies,eigenEnergies];
  energyDiffs    = ReplaceDiagonal[energyDiffs, Indeterminate];
  (* Energies assumed in pseudo-energy unit kayser.*)
  transitionWaveLengthsInMeters = 0.01/energyDiffs;
```

```
GroundStateOscillatorStrength::usage="GroundStateOscillatorStrength[
    eigenSys, numE] calculates the oscillator strength between the
    ground state and the excited states as given by eigenSys. The
    energy unit assumed in eigenSys is kayser. The returned array
    should be interpreted in the eigenbasis of the Hamiltonian. As such
    the element fMDGS[[i]] corresponds to the oscillator strength
    between ground state and eigenstate |i>.";
GroundStateOscillatorStrength[eigenSys_, numE_]:=Module[
{eigenEnergies, SMDGS, GSEnergy, gKramers, energyDiffs,
    transitionWaveLengthsInMeters, factor},
(
  eigenEnergies     = First/@eigenSys;
  SMDGS             = MagDipLineStrength[eigenSys,numE, "Units"->"SI", "
    States"->1];
  GSEnergy          = eigenSys[[1,1]];
  gKramers          = If[OddQ[numE],2,1];
  energyDiffs       = eigenEnergies-GSEnergy;
  energyDiffs[[1]] = Indeterminate;
  transitionWaveLengthsInMeters = 0.01/energyDiffs;
  factor = (8\[Pi]^2 me)       /(3 hPlanck eCharge^2 cLight);
  fMDGS=unitFactor/gKramers/transitionWaveLengthsInMeters*SMDGS;
  Return[fMDGS];
)
]
```

7 Accompanying notebooks

qlanth is accompanied by the following auxiliary Mathematica notebooks:

- qlanth.nb: gives an overview of the different included functions.
- qlanth - Table Generator.nb: generates the basic tables on which every calculation is based.
- qlanth - JJBlock Calculator.nb: can be used to generate the JJ blocks for the different interactions. The data files produced here are necessary for HamMatrixAssembly to work.
- The Lanthanides in LaF3.nb: runs **qlanth** over the lanthanide ions in LaF3 and compares the results agains the published values from Carnall. It also calculates magnetic dipole transition rates and oscillator strengths.

8 Notation

orbital angular momentum operator of a single electron

$$\overbrace{\hat{l}} \tag{66}$$

total orbital angular momentum operator

$$\overbrace{\hat{L}} \tag{67}$$

spin angular momentum operator of a single electron

$$\overbrace{\hat{s}} \tag{68}$$

total spin angular momentum operator

$$\overbrace{\hat{S}} \tag{69}$$

Shorthand for all other quantum numbers

$$\overbrace{\Lambda} \tag{70}$$

orbital angular momentum number

$$\overbrace{\underline{\ell}} \tag{71}$$

spinning angular momentum number

$$\overbrace{\underline{s}} \tag{72}$$

Coulomb non-central potential

$$\overbrace{\hat{\mathfrak{C}}} \tag{73}$$

LS-reduced matrix element of operator $\hat{O}$ between ΛLS and $\Lambda' L' S'$

$$\overbrace{\langle \Lambda LS \| \hat{O} \| \Lambda' L' S' \rangle} \tag{74}$$

LSJ-reduced matrix element of operator $\hat{O}$ between ΛLSJ and $\Lambda' L' S' J'$

$$\overbrace{\langle \Lambda LSJ \| \hat{O} \| \Lambda' L' S' J' \rangle} \tag{75}$$

Spectroscopic term αLS in Russel-Saunders notation

$$\overbrace{{}^{2S+1}\alpha L \equiv |\alpha LS\rangle} \tag{76}$$

spherical tensor operator of rank k

$$\overbrace{\hat{X}^{(k)}} \tag{77}$$

q-component of the spherical tensor operator $\hat{X}^{(k)}$

$$\overbrace{\hat{X}^{(k)}_q} \tag{78}$$

The coefficient of fractional parentage from the parent term $|\underline{\ell}^{n-1} \alpha' L' S'\rangle$ for the daughter term $|\underline{\ell}^n \alpha LS\rangle$

$$\overbrace{\left(\underline{\ell}^{n-1} \alpha' L' S' \middle\} \underline{\ell}^n \alpha LS\right)} \tag{79}$$

9 Definitions

two plus one

$$\overbrace{[\![x]\!] \coloneqq 2x + 1} \tag{80}$$

irreducible unit tensor operator of rank k

$$\overbrace{\hat{u}^{(k)}} \tag{81}$$

symmetric unit tensor operator for n equivalent electrons

$$\overbrace{\hat{U}^{(k)} \coloneqq \sum_{i=1}^{n} \hat{u}^{(k)}} \tag{82}$$

Renormalized spherical harmonics

$$\overbrace{\mathcal{C}^{(k)}_q \coloneqq \sqrt{\frac{4\pi}{2k+1}} \mathrm{Y}^{(k)}_q} \tag{83}$$

Triangle "delta" between j_1, j_2, j_3

$$\overbrace{\triangle(j_1, j_2, j_3) \coloneqq \begin{cases} 1 & \text{if } j_1 = (j_2 + j_3), (j_2 + j_3 - 1), \ldots, |j_2 - j_3| \\ 0 & \text{otherwise} \end{cases}} \tag{84}$$

10 qlanth.m

This file encapsulates the main functions in **qlanth** and contains all the Physics related functions.

```
(* ------------------------------------------------------------------
+--------------------------------------------------------------------+
|                                                                    |
|                                                                    |
|                      __                          __      __        |
|            ____ _   / /  ____ _   ____     / /_   / /_            |
|           / __ `/  / /  / __ `/  / __ \   / __/  / __ \           |
|          / /_/ /  / /  / /_/ /  / / / /  / /_   / / / /           |
|          \__, /  /_/   \__, _/  /_/ /_/   \__/  /_/ /_/           |
|            /_/                                                     |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
This code was initially authored by Christopher Dodson and Rashid
Zia and then rewritten by David Lizarazo in the years 2022-2024
under the advisory of Dr. Zia. It has also benefited from the
discussions with Tharnier Puel.

It uses an effective Hamiltonian to describe the electronic
structure of lanthanide ions in crystals. This effective Hamiltonian
includes terms representing the following interactions/relativistic
corrections: spin-orbit, electrostatic repulsion, spin-spin, crystal
field and spin-other- orbit.

The Hilbert space used in this effective Hamiltonian is limited to
single f^n configurations. The inaccuracy of this single
configuration description is partially compensated by the inclusion
of configuration interaction terms as parametrized by the Casimir
operators of SO(3), G(2), and SO(7), and by three-body effective
operators ti.

The parameters included in this model are listed in the string
paramAtlas.

The notebook "qlanth.nb" contains a gallery with all the functions
included in this module with some simple use cases.

The notebook "The Lanthanides in LaF3.nb" is an example in which the
results from this code are compared against the published results by
Carnall et. al for the energy levels of lanthinde ions in crystals
of lanthanum fluoride.

REFERENCES:

+ Condon, E U, and G H Shortley. The Theory of Atomic Spectra, 1935.

+ Racah, Giulio. "Theory of Complex Spectra. III." Physical Review
63, no. 9-10 (May 1, 1943): 367-82.
https://doi.org/10.1103/PhysRev.63.367.

+ Racah, Giulio. "Theory of Complex Spectra. II." Physical Review
62, no. 9-10 (November 1, 1942): 438-62.
https://doi.org/10.1103/PhysRev.62.438.

+ Rajnak, K, and BG Wybourne. "Configuration Interaction Effects in
l^N Configurations." Physical Review 132, no. 1 (1963): 280.
https://doi.org/10.1103/PhysRev.132.280.

+ Wybourne, Brian G. Spectroscopic Properties of Rare Earths, 1965.

+ Judd, BR. "Three-Particle Operators for Equivalent Electrons."
Physical Review 141, no. 1 (1966): 4.
https://doi.org/10.1103/PhysRev.141.4.

+ Nielson, C. W., and George F Koster. "Spectroscopic Coefficients
for the p^n, d^n, and f^n Configurations", 1963.

+ Judd, BR, HM Crosswhite, and Hannah Crosswhite. "Intra-Atomic
Magnetic Interactions for f Electrons." Physical Review 169, no. 1
(1968): 130. https://doi.org/10.1103/PhysRev.169.130.

+ (TASS) Cowan, Robert Duane. The Theory of Atomic Structure and
Spectra. Los Alamos Series in Basic and Applied Sciences 3.
Berkeley: University of California Press, 1981.
```

```
74
75 + Judd, BR, and MA Suskin. "Complete Set of Orthogonal Scalar
76 Operators for the Configuration f^3." JOSA B 1, no. 2 (1984):
77 261-65. https://doi.org/10.1364/JOSAB.1.000261.
78
79 + Carnall, W. T., G. L. Goodman, K. Rajnak, and R. S. Rana. "A
80 Systematic Analysis of the Spectra of the Lanthanides Doped into
81 Single Crystal LaF3." The Journal of Chemical Physics 90, no. 7
82 (1989): 3443-57. https://doi.org/10.1063/1.455853.
83
84 + Hansen, JE, BR Judd, and Hannah Crosswhite. "Matrix Elements of
85 Scalar Three-Electron Operators for the Atomic f-Shell." Atomic Data
86 and Nuclear Data Tables 62, no. 1 (1996): 1-49.
87 https://doi.org/10.1006/adnd.1996.0001.
88
89 + Velkov, Dobromir. "Multi-Electron Coefficients of Fractional
90 Parentage for the p, d, and f Shells." John Hopkins University,
91 2000. The B1F_ALL.TXT file is from this thesis.
92
93 + Dodson, Christopher M., and Rashid Zia. "Magnetic Dipole and
94 Electric Quadrupole Transitions in the Trivalent Lanthanide Series:
95 Calculated Emission Rates and Oscillator Strengths." Physical Review
96 B 86, no. 12 (September 5, 2012): 125102.
97 https://doi.org/10.1103/PhysRevB.86.125102.
98
99
100 ------------------------------------------------------------------ *)
101
102 BeginPackage["qlanth`"];
103 Needs["qonstants`"];
104 Needs["qplotter`"];
105 Needs["misc`"];
106
107 paramAtlas = "
108 E0: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
109 E1: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
110 E2: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
111 E3: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
112
113 ζ: spin-orbit strength parameter.
114
115 F0: Direct Slater integral F^0, produces an overall shift of all energy
     levels.
116 F2: Direct Slater integral F^2
117 F4: Direct Slater integral F^4, possibly constrained by ratio to F^2
118 F6: Direct Slater integral F^6, possibly constrained by ratio to F^2
119
120 M0: 0th Marvin integral
121 M2: 2nd Marvin integral
122 M4: 4th Marvin integral
123 \[Sigma]SS: spin-spin override, if 0 spin-spin is omitted, if 1 then
    spin-spin is included
124
125 T2:  three-body effective operator parameter T^2 (non-orthogonal)
126 T2p: three-body effective operator parameter T^2' (orthogonalized T2)
127 T3:  three-body effective operator parameter T^3
128 T4:  three-body effective operator parameter T^4
129 T6:  three-body effective operator parameter T^6
130 T7:  three-body effective operator parameter T^7
131 T8:  three-body effective operator parameter T^8
132
133 T11:  three-body effective operator parameter T^11
134 T11p: three-body effective operator parameter T^11'
135 T12:  three-body effective operator parameter T^12
136 T14:  three-body effective operator parameter T^14
137 T15:  three-body effective operator parameter T^15
138 T16:  three-body effective operator parameter T^16
139 T17:  three-body effective operator parameter T^17
140 T18:  three-body effective operator parameter T^18
141 T19:  three-body effective operator parameter T^19
142
143 P0: 0th parameter for the two-body electrostatically correlated spin-
    orbit interaction
144 P2: 2nd parameter for the two-body electrostatically correlated spin-
    orbit interaction
145 P4: 4th parameter for the two-body electrostatically correlated spin-
```

```
    orbit interaction
P6: 6th parameter for the two-body electrostatically correlated spin-
    orbit interaction

gs: electronic gyromagnetic ratio

α: Trees' parameter α describing configuration interaction via the
    Casimir operator of SO(3)
β: Trees' parameter β describing configuration interaction via the
    Casimir operator of G(2)
γ: Trees' parameter γ describing configuration interaction via the
    Casimir operator of SO(7)

B02: crystal field parameter B_0^2 (real)
B04: crystal field parameter B_0^4 (real)
B06: crystal field parameter B_0^6 (real)
B12: crystal field parameter B_1^2 (real)
B14: crystal field parameter B_1^4 (real)

B16: crystal field parameter B_1^6 (real)
B22: crystal field parameter B_2^2 (real)
B24: crystal field parameter B_2^4 (real)
B26: crystal field parameter B_2^6 (real)
B34: crystal field parameter B_3^4 (real)

B36: crystal field parameter B_3^6 (real)
B44: crystal field parameter B_4^4 (real)
B46: crystal field parameter B_4^6 (real)
B56: crystal field parameter B_5^6 (real)
B66: crystal field parameter B_6^6 (real)

S12: crystal field parameter S_1^2 (real)
S14: crystal field parameter S_1^4 (real)
S16: crystal field parameter S_1^6 (real)
S22: crystal field parameter S_2^2 (real)

S24: crystal field parameter S_2^4 (real)
S26: crystal field parameter S_2^6 (real)
S34: crystal field parameter S_3^4 (real)
S36: crystal field parameter S_3^6 (real)

S44: crystal field parameter S_4^4 (real)
S46: crystal field parameter S_4^6 (real)
S56: crystal field parameter S_5^6 (real)
S66: crystal field parameter S_6^6 (real)

\[Epsilon]: ground level baseline shift
t2Switch: controls the usage of the t2 operator beyond f7
wChErrA: If 1 then the type-A errors in Chen are used, if 0 then not.
wChErrB: If 1 then the type-B errors in Chen are used, if 0 then not.

Bx: x component of external magnetic field (in T)
By: y component of external magnetic field (in T)
Bz: z component of external magnetic field (in T)
";
paramSymbols = StringSplit[paramAtlas, "\n"];
paramSymbols = Select[paramSymbols, # != ""& ];
paramSymbols = ToExpression[StringSplit[#, ":"][[1]]] & /@ paramSymbols
    ;
Protect /@ paramSymbols;
paramLines = Select[StringSplit[paramAtlas, "\n"], # != "" &];
usageTemplate = StringTemplate["`paramSymbol`::usage=\"`paramSymbol` :
    `paramUsage`\";"];
Do[(
  {paramString, paramUsage} = StringSplit[paramLine, ":"];
  paramUsage = StringTrim[paramUsage];
  expressionString = usageTemplate[<|"paramSymbol" -> paramString, "
    paramUsage" -> paramUsage|>];
  ToExpression[usageTemplate[<|"paramSymbol" -> paramString,
     "paramUsage" -> paramUsage|>]]
),
{paramLine, paramLines}
];

(* Parameter families*)
cfSymbols = {B02, B04, B06, B12, B14, B16, B22, B24, B26, B34, B36,
```

```
    B44, B46, B56, B66, S12, S14, S16, S22, S24, S26, S34, S36, S44,
    S46, S56, S66};

TSymbols = {T2, T2p, T3, T4, T6, T7, T8, T11, T11p, T12, T14, T15, T16,
     T17, T18, T19};

AllowedJ;
AllowedMforJ;
AllowedNKSLJMforJMTerms;
AllowedNKSLJMforJTerms;

AllowedNKSLJTerms;
AllowedNKSLTerms;
AllowedNKSLforJTerms;
AllowedSLJMTerms;
AllowedSLJTerms;

AllowedSLTerms;
BasisLSJMJ;
Bqk;
CFP;
CFPAssoc;

CFPTable;
CFPTerms;
Carnall;
CasimirG2;
CasimirSO3;
CasimirSO7;

Cqk;
CrystalField;
Dk;
ElectrostaticConfigInteraction;
Electrostatic;

ElectrostaticTable;
EnergyLevelDiagram;
EnergyStates;
ExportMZip;
BasisTableGenerator;
EtoF;
ExportmZip;
fsubk;
fsupk;

FindNKLSTerm;
FindSL;

FtoE;
GG2U;
GSO7W;
GenerateCFP;
GenerateCFPAssoc;

GenerateCFPTable;
GenerateCrystalFieldTable;
GenerateElectrostaticTable;
GenerateReducedUkTable;
GenerateReducedV1kTable;

GenerateSOOandECSOLSTable;
GenerateSOOandECSOTable;
GenerateSpinOrbitTable;
GenerateSpinSpinTable;
GenerateT22Table;

GenerateThreeBodyTables;
GenerateThreeBodyTables;
Generator;
GroundStateOscillatorStrength;
HamMatrixAssembly;
HamMatrixAssemblyALT;
HamiltonianForm;

HamiltonianMatrixPlot;
```

```
HoleElectronConjugation;
IonSolver;
ImportMZip;
JJBlockMatrix;
JJBlockMagDip;
JJBlockMatrixFileName;

JJBlockMatrixTable;
LabeledGrid;
LoadAll;
LoadCFP;
LoadCarnall;

LoadChenDeltas;
LoadElectrostatic;
LoadGuillotParameters;
LoadParameters;
LoadSOOandECSO;

LoadSOOandECSOLS;
LoadSpinOrbit;
LoadSpinSpin;
LoadSymbolicHamiltonians;
LoadT11;

LoadT22;
LoadTermLabels;
LoadThreeBody;
LoadUk;
LoadV1k;

MagneticInteractions;
MagDipoleMatrixAssembly;
MagDipLineStrength;
MapToSparseArray;
MaxJ;
MinJ;
NKCFPPhase;

ParamPad;
ParseStates;
ParseStatesByNumBasisVecs;
ParseStatesByProbabilitySum;
ParseTermLabels;

Phaser;
PrettySaunders;
PrettySaundersSLJ;
PrettySaundersSLJmJ;
PrintL;

PrintSLJ;
PrintSLJM;
ReducedSOOandECSOinf2;
ReducedSOOandECSOinfn;
ReducedT11inf2;

ReducedT22inf2;
ReducedUk;
ReducedUkTable;
ReducedV1kTable;
Reducedt11inf2;

ReplaceInSparseArray;
SimplerSymbolicHamMatrix;
SOOandECSO;
SOOandECSOTable;
Seniority;

ShiftedLevels;
SixJay;
SpinOrbit;
SpinSpin;
SpinSpinTable;

Sqk;
```

```
SquarePrimeToNormal;
ReducedT22infn;
TPO;

TabulateJJBlockMatrixTable;
TabulateJJBlockMagDipTable;
TabulateManyJJBlockMatrixTables;
TabulateManyJJBlockMagDipTables;
ScalarOperatorProduct;
ThreeBodyTable;

ThreeBodyTables;
ThreeJay;
TotalCFIters;
MagDipoleRates;
chenDeltas;
fK;

fnTermLabels;
moduleDir;
symbolicHamiltonians;

(* this selects the function that is applied
to calculated matrix elements *)
SimplifyFun = Expand;

Begin["`Private`"]

  moduleDir = DirectoryName[$InputFileName];
  frontEndAvailable = (Head[$FrontEnd] === FrontEndObject);

  (* ############################################################ *)
  (* ########################## MISC ############################ *)

  TPO::usage="Two plus one.";
  TPO[args__] := Times @@ ((2*# + 1) & /@ {args});

  Phaser::usage = "Phaser[x] returns (-1)^x";
  Phaser[exponent_] := ((-1)^exponent);

  TriangleCondition::usage = "TriangleCondition[a, b, c] returns True
    if a, b, and c satisfy the triangle condition.";
  TriangleCondition[a_, b_, c_] := (Abs[b - c] <= a <= (b + c));

  TriangleAndSumCondition::usage = "TriangleAndSumCondition[a, b, c]
    returns True if a, b, and c satisfy the triangle and sum conditions
    .";
  TriangleAndSumCondition[a_, b_, c_] := (And[Abs[b - c] <= a <= (b + c
    ), IntegerQ[a + b + c]]);

  SquarePrimeToNormal::usage = "Given a list with the parts
    corresponding to the squared prime representation of a number, this
     function parses the result into standard notation.";
  SquarePrimeToNormal[squarePrime_] :=
  (
    radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {idx1,
     2, Length[squarePrime]}];
    radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" -> 13};
    val = squarePrime[[1]] * Sqrt[radical];
    Return[val];
  );

  ParamPad::usage = "ParamPad[params] takes an association params whose
     keys are a subset of paramSymbols. The function returns a new
    association where all the keys not present in paramSymbols, will
    now be included in the returned association with their values set
    to zero.
  The function additionally takes an option \"Print\" that if set to
    True, will print the symbols that were not present in the given
    association.";
  Options[ParamPad] = {"Print" -> True}
  ParamPad[params_, OptionsPattern[]] := (
    notPresentSymbols = Complement[paramSymbols, Keys[params]];
    If[OptionValue["Print"],
      Print["Following symbols were not given and are being set to 0: "
    ,
```

```
427       notPresentSymbols]
428     ];
429     newParams = Transpose[{paramSymbols, ConstantArray[0, Length[
        paramSymbols]]}];
430     newParams = (#[[1]] -> #[[2]]) & /@ newParams;
431     newParams = Association[newParams];
432     newParams = Join[newParams, params];
433     Return[newParams];
434     )
435
436   (* ############################################################ *)
437   (* ###################### Racah Algebra ###################### *)
438
439   ReducedUk::usage = "ReducedUk[n, l, SL, SpLp, k] gives the reduced
        matrix element of the symmetric unit tensor operator U^(k). See
        equation 11.53 in TASS.";
440   ReducedUk[numE_, l_, SL_, SpLp_, k_] :=
441     Module[{spin, orbital, Uk,
442       S, L, Sp, Lp, Sb, Lb,
443       parentSL, cfpSL, cfpSpLp, Ukval, SLparents, SLpparents,
        commonParents, phase},
444       {spin, orbital} = {1/2, 3};
445       {S, L}          = FindSL[SL];
446       {Sp, Lp}        = FindSL[SpLp];
447       If[Not[S == Sp],
448         Return[0]
449       ];
450       cfpSL       = CFP[{numE, SL}];
451       cfpSpLp     = CFP[{numE, SpLp}];
452       SLparents   = First /@ Rest[cfpSL];
453       SLpparents  = First /@ Rest[cfpSpLp];
454       commonParents = Intersection[SLparents, SLpparents];
455       Uk = Sum[(
456         {Sb, Lb} = FindSL[\[Psi]b];
457         Phaser[Lb] *
458           CFPAssoc[{numE, SL, \[Psi]b}] *
459           CFPAssoc[{numE, SpLp, \[Psi]b}] *
460           SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
461       ),
462       {\[Psi]b, commonParents}
463       ];
464       phase     = Phaser[orbital + L + k];
465       prefactor = numE * phase * Sqrt[TPO[L,Lp]];
466       Ukval     = prefactor*Uk;
467       Return[Ukval];
468   ]
469
470   Ck::usage = "Diagonal reduced matrix element <l||C^(k)||l> where the
        Subscript[C, q]^(k) are reduced spherical harmonics. See equation
        11.23 in TASS with l=l'.";
471   Ck[orbital_, k_] := (-1)^orbital * TPO[orbital] * ThreeJay[{orbital,
        0}, {k, 0}, {orbital, 0}]
472
473   SixJay::usage = "SixJay[{j1, j2, j3}, {j4, j5, j6}] provides the
        value for SixJSymbol[{j1, j2, j3}, {j4, j5, j6}] with memorization
        of computed values.";
474   SixJay[{j1_, j2_, j3_}, {j4_, j5_, j6_}] := (
475     sixJayval =
476       Which[
477       Not[TriangleAndSumCondition[j1, j2, j3]],
478       0,
479       Not[TriangleAndSumCondition[j1, j5, j6]],
480       0,
481       Not[TriangleAndSumCondition[j4, j2, j6]],
482       0,
483       Not[TriangleAndSumCondition[j4, j5, j3]],
484       0,
485       True,
486       SixJSymbol[{j1, j2, j3}, {j4, j5, j6}]];
487     SixJay[{j1, j2, j3}, {j4, j5, j6}] = sixJayval);
488
489   ThreeJay::usage = "ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] gives the
        value of the Wigner 3j-symbol and memorizes the computed value.";
490   ThreeJay[{j1_, m1_}, {j2_, m2_}, {j3_, m3_}] := (
491    threejval = Which[
492      Not[(m1 + m2 + m3) == 0],
```

```
    0,
    Not[TriangleCondition[j1,j2,j3]],
    0,
    True,
    ThreeJSymbol[{j1, m1}, {j2, m2}, {j3, m3}]
    ];
 ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] = threejval);

ReducedV1k::usage = "ReducedV1k[n, l, SL, SpLp, k] gives the reduced
  matrix element of the spherical tensor operator V^(1k). See
  equation 2-101 in Wybourne 1965.";
ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
  {Vk1, S, L, Sp, Lp, Sb, Lb, spin, orbital, cfpSL, cfpSpLp,
  SLparents, SpLpparents, commonParents, prefactor},
  {spin, orbital} = {1/2, 3};
  {S, L}          = FindSL[SL];
  {Sp, Lp}        = FindSL[SpLp];
  cfpSL           = CFP[{numE, SL}];
  cfpSpLp         = CFP[{numE, SpLp}];
  SLparents       = First /@ Rest[cfpSL];
  SpLpparents     = First /@ Rest[cfpSpLp];
  commonParents = Intersection[SLparents, SpLpparents];
  Vk1 = Sum[(
      {Sb, Lb} = FindSL[\[Psi]b];
      Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
      CFPAssoc[{numE, SL, \[Psi]b}] *
      CFPAssoc[{numE, SpLp, \[Psi]b}] *
      SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
      SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
    ),
  {\[Psi]b, commonParents}
  ];
  prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp, Lp]
  ];
  Return[prefactor * Vk1];
  ]

GenerateReducedUkTable::usage = "GenerateReducedUkTable[numEmax] can
  be used to generate the association of reduced matrix elements for
  the unit tensor operators Uk from f^1 up to f^numEmax. If the
  option \"Export\" is set to True then the resulting data is saved
  to ./data/ReducedUkTable.m.";
Options[GenerateReducedUkTable] = {"Export" -> True, "Progress" ->
  True};
GenerateReducedUkTable[numEmax_Integer:7, OptionsPattern[]]:= (
  numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
  AllowedNKSLTerms[#]]&/@Range[1, numEmax]] * 4;
  Print["Calculating " <> ToString[numValues] <> " values for Uk k
  =0,2,4,6."];
  counter = 1;
  If[And[OptionValue["Progress"], frontEndAvailable],
  progBar = PrintTemporary[
      Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
        counter}]]]
    ];
  ReducedUkTable = Table[
    (
      counter = counter+1;
      {numE, 3, SL, SpLp, k} -> SimplifyFun[ReducedUk[numE, 3, SL,
  SpLp, k]]
    ),
    {numE, 1, numEmax},
    {SL,   AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]},
    {k, {0, 2, 4, 6}}
  ];
  ReducedUkTable = Association[Flatten[ReducedUkTable]];
  ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "
  ReducedUkTable.m"}];
  If[And[OptionValue["Progress"], frontEndAvailable],
    NotebookDelete[progBar]
  ];
  If[OptionValue["Export"],
    (
      Print["Exporting to file " <> ToString[ReducedUkTableFname]];
      Export[ReducedUkTableFname, ReducedUkTable];
```

```
      )
    ];
    Return[ReducedUkTable];
  )

  GenerateReducedV1kTable::usage = "GenerateReducedV1kTable[nmax,
    export calculates values for Vk1 and returns an association where
    the keys are lists of the form {n, SL, SpLp, 1}. If the option \"
    Export\" is set to True then the resulting data is saved to ./data/
    ReducedV1kTable.m."
  Options[GenerateReducedV1kTable] = {"Export" -> True, "Progress" ->
    True};
  GenerateReducedV1kTable[numEmax_Integer:7, OptionsPattern[]]:= (
    numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
    AllowedNKSLTerms[#]]&/@Range[1, numEmax]];
    Print["Calculating " <> ToString[numValues] <> " values for Vk1."];
    counter = 1;
    If[And[OptionValue["Progress"], frontEndAvailable],
    progBar = PrintTemporary[
        Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
          counter}]]]
      ];
    ReducedV1kTable = Table[
      (
        counter = counter+1;
        {n, SL, SpLp, 1} -> SimplifyFun[ReducedV1k[n, SL, SpLp, 1]]
      ),
      {n, 1, numEmax},
      {SL, AllowedNKSLTerms[n]},
      {SpLp, AllowedNKSLTerms[n]}
    ];
    ReducedV1kTable = Association[ReducedV1kTable];
    If[And[OptionValue["Progress"], frontEndAvailable],
      NotebookDelete[progBar]
    ];
    exportFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m"
    }];
    If[OptionValue["Export"],
      (
        Print["Exporting to file "<>ToString[exportFname]];
        Export[exportFname, ReducedV1kTable];
      )
    ];
    Return[ReducedV1kTable];
  )

  (* ######################## Racah Algebra ######################## *)
  (* ############################################################## *)

  (* ############################################################## *)
  (* ######################## Electrostatic ######################## *)

  fsubk::usage = "Slater integral f_k. See equation 12.17 in TASS.";
  fsubk[numE_, orbital_, NKSL_, NKSLp_, k_] := Module[
    {terms, S, L, Sp, Lp, termsWithSameSpin, SL, fsubkVal,
    spinMultiplicity,
    prefactor, summand1, summand2},
    {S, L}   = FindSL[NKSL];
    {Sp, Lp} = FindSL[NKSLp];
    terms = AllowedNKSLTerms[numE];
    (* sum for summand1 is over terms with same spin *)
    spinMultiplicity  = 2*S + 1;
    termsWithSameSpin = StringCases[terms, ToString[spinMultiplicity]
    ~~ __];
    termsWithSameSpin = Flatten[termsWithSameSpin];
    If[Not[{S, L} == {Sp, Lp}],
      Return[0]
    ];
    prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
    summand1 = Sum[(
        ReducedUkTable[{numE, orbital, SL, NKSL,  k}] *
        ReducedUkTable[{numE, orbital, SL, NKSLp, k}]
        ),
      {SL, termsWithSameSpin}
    ];
    summand1 = 1 / TPO[L] * summand1;
```

```
    summand2 = (
      KroneckerDelta[NKSL, NKSLp] *
        (numE *(4*orbital + 2 - numE)) /
        ((2*orbital + 1) * (4*orbital + 1))
      );
    fsubkVal = prefactor*(summand1 - summand2);
    Return[fsubkVal];
  ]

  fsupk::usage = "Super-script Slater integral f^k = Subscript[f, k] *
    Subscript[D, k]";
  fsupk[numE_, orbital_, NKSL_, NKSLp_ ,k_]:= (Dk[k] * fsubk[numE,
    orbital, NKSL, NKSLp, k])

  Dk::usage = "Ratio between the super-script and sub-scripted Slater
    integrals (F^k /F_k). k must be even. See table 6-3 in TASS, and
    also section 2-7 of Wybourne (1965). See also equation 6.41 in TASS
    .";
  Dk[k_] := {1, 225, 1089, 184041/25}[[k/2+1]]

  FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the corresponding {E0,
     E1, E2, E3} values.
  See eqn. 2-80 in Wybourne. Note that in that equation the subscripted
     Slater integrals are used but since this function assumes the the
    input values are superscripted Slater integrals, it is necessary to
     convert them using Dk.";
  FtoE[F0_, F2_, F4_, F6_] := (Module[
    {E0, E1, E2, E3},
    E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
    E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
    E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;
    E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
    Return[{E0, E1, E2, E3}];
  ]
  );

  EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the corresponding {F0,
     F2, F4, F6} values. The inverse of FtoE.";
  EtoF[E0_, E1_, E2_, E3_] := (Module[
    {F0, F2, F4, F6},
    F0 = 1/7       (7 E0 + 9 E1);
    F2 = 75/14     (E1 + 143 E2 + 11 E3);
    F4 = 99/7      (E1 - 130 E2 + 4 E3);
    F6 = 5577/350 (E1 + 35 E2 - 7 E3);
    Return[{F0, F2, F4, F6}];
  ]
  );

  Electrostatic::usage = "Electrostatic[{numE, NKSL, NKSLp}] returns
    the LS reduced matrix element for repulsion matrix element for
    equivalent electrons. See equation 2-79 in Wybourne (1965). The
    option \"Coefficients\" can be set to \"Slater\" or \"Racah\". If
    set to \"Racah\" then E_k parameters and e^k operators are assumed,
     otherwise the Slater integrals F^k and operators f_k. The default
    is \"Slater\".";
  Options[Electrostatic] = {"Coefficients" -> "Slater"};
  Electrostatic[{numE_, NKSL_, NKSLp_}, OptionsPattern[]]:= Module[
    {fsub0, fsub2, fsub4, fsub6,
     esub0, esub1, esub2, esub3,
     fsup0, fsup2, fsup4, fsup6,
     eMatrixVal, orbital},
    orbital = 3;
    Which[
      OptionValue["Coefficients"] == "Slater",
      (
        fsub0 = fsubk[numE, orbital, NKSL, NKSLp, 0];
        fsub2 = fsubk[numE, orbital, NKSL, NKSLp, 2];
        fsub4 = fsubk[numE, orbital, NKSL, NKSLp, 4];
        fsub6 = fsubk[numE, orbital, NKSL, NKSLp, 6];
        eMatrixVal = fsub0*F0 + fsub2*F2 + fsub4*F4 + fsub6*F6;
      ),
      OptionValue["Coefficients"] == "Racah",
      (
        fsup0 = fsupk[numE, orbital, NKSL, NKSLp, 0];
        fsup2 = fsupk[numE, orbital, NKSL, NKSLp, 2];
        fsup4 = fsupk[numE, orbital, NKSL, NKSLp, 4];
```

```
        fsup6 = fsupk[numE, orbital, NKSL, NKSLp, 6];
        esub0 = fsup0;
        esub1 = 9/7*fsup0 +   1/42*fsup2 +   1/77*fsup4 +  1/462*fsup6;
        esub2 =             143/42*fsup2 - 130/77*fsup4 + 35/462*fsup6;
        esub3 =              11/42*fsup2 + 4/77*fsup4   -  7/462*fsup6;
        eMatrixVal = esub0*E0 + esub1*E1 + esub2*E2 + esub3*E3;
      )
    ];
    Return[eMatrixVal];
  ]

  GenerateElectrostaticTable::usage = "GenerateElectrostaticTable[
    numEmax] can be used to generate the table for the electrostatic
    interaction from f^1 to f^numEmax. If the option \"Export\" is set
    to True then the resulting data is saved to ./data/
    ElectrostaticTable.m.";
  Options[GenerateElectrostaticTable] = {"Export" -> True, "
    Coefficients" -> "Slater"};
  GenerateElectrostaticTable[numEmax_Integer:7, OptionsPattern[]]:= (
    ElectrostaticTable = Table[
      {numE, SL, SpLp} -> SimplifyFun[Electrostatic[{numE, SL, SpLp}, "
    Coefficients" -> OptionValue["Coefficients"]]],
      {numE, 1, numEmax},
      {SL, AllowedNKSLTerms[numE]},
      {SpLp, AllowedNKSLTerms[numE]}
    ];
    ElectrostaticTable = Association[Flatten[ElectrostaticTable]];
    If[OptionValue["Export"],
      Export[FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}],
      ElectrostaticTable];
    ];
    Return[ElectrostaticTable];
  )

  (* ###################### Electrostatic ###################### *)
  (* ########################################################## *)

  (* ########################################################## *)
  (* ########################## Bases ######################### *)

  BasisTableGenerator::usage = "BasisTableGenerator[numE] returns an
    association whose keys are triples of the form {numE, J} and whose
    values are lists having the basis elements that correspond to {numE
    , J}.";
  BasisTableGenerator[numE_] := Module[{energyStatesTable, allowedJ, J,
     Jp},
    (
      energyStatesTable = <||>;
      allowedJ = AllowedJ[numE];
      Do[
      (
        energyStatesTable[{numE, J}] = EnergyStates[numE, J];
      ),
      {Jp, allowedJ},
      {J,  allowedJ}];
      Return[energyStatesTable]
    )
    ];

  BasisLSJMJ::usage = "BasisLSJMJ[numE] returns the ordered basis in L-
    S-J-MJ with the total orbital angular momentum L and total spin
    angular momentum S coupled together to form J. The function returns
     a list with each element representing the quantum numbers for each
     basis vector. Each element is of the form {SL (string in
    spectroscopic notation),J,MJ}.";
  BasisLSJMJ[numE_] := Module[{energyStatesTable, basis, idx1},
    (
      energyStatesTable = BasisTableGenerator[numE];
      basis = Table[
        energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
        {idx1, 1, Length[AllowedJ[numE]]}];
      basis = Flatten[basis, 1];
      Return[basis]
    )
    ];

```

```
(* ########################## Bases ########################## *)
(* ########################################################### *)

(* ########################################################### *)
(* ############ Coefficients of Fracional Parentage ############ *)

GenerateCFP::usage = "GenerateCFP[] generates the association for the coefficients of fractional parentage. Result is exported to the file ./data/CFP.m. The coefficients of fractional parentage are taken beyond the half-filled shell using the phase convention determined by the option \"PhaseFunction\". The default is \"NK\" which corresponds to the phase convention of Nielson and Koster. The other option is \"Judd\" which corresponds to the phase convention of Judd.";
Options[GenerateCFP] = {"Export" -> True, "PhaseFunction"-> "NK"};
GenerateCFP[OptionsPattern[]]:= (
  CFP = Table[
    {numE, NKSL} -> First[CFPTerms[numE, NKSL]],
    {numE, 1, 7},
    {NKSL, AllowedNKSLTerms[numE]}];
  CFP = Association[CFP];
  (* Go all the way to f14 *)
  CFP = CFPExpander["Export" -> False, "PhaseFunction"-> OptionValue["PhaseFunction"]];
  If[OptionValue["Export"],
    Export[FileNameJoin[{moduleDir, "data", "CFPs.m"}], CFP];
  ];
  Return[CFP];
)

JuddCFPPhase::usage="Phase between conjugate coefficients of fractional parentage according to Velkov's thesis, page 40.";
JuddCFPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_, parentSeniority_, daughterSeniority_] := Module[
  {spin, orbital, expo, phase},
(
    {spin, orbital} = {1/2, 3};
    expo = (
        (parentS + parentL + daughterS + daughterL) -
        (orbital + spin) +
        1/2 * (parentSeniority + daughterSeniority - 1)
    );
    phase = Phaser[-expo];
    Return[phase];
)
]

NKCFPPhase::usage="Phase between conjugate coefficients of fractional parentage according to Nielson and Koster page viii. Note that there is a typo on there the expression for zeta should be (-1)^((v-1)/2) instead of (-1)^(v - 1/2).";
NKCFPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_, parentSeniority_, daughterSeniority_] := Module[{spin, orbital, expo, phase},
  (
    {spin, orbital} = {1/2, 3};
    expo = (
        (parentS + parentL + daughterS + daughterL) -
        (orbital + spin)
    );
    phase = Phaser[-expo];
    If[parent == 2*orbital,
        phase = phase * Phaser[(daughterSeniority-1)/2]];
    Return[phase];
  )
]

Options[CFPExpander] = {"Export" -> True, "PhaseFunction" -> "NK"};
CFPExpander::usage="Using the coefficients of fractional parentage up to f7 this function calculates them up to f14.
The coefficients of fractional parentage are taken beyond the half-filled shell using the phase convention determined by the option \"PhaseFunction\". The default is \"NK\" which corresponds to the phase convention of Nielson and Koster. The other option is \"Judd\" which corresponds to the phase convention of Judd. The result is exported to the file ./data/CFPs_extended.m.";
```

```
CFPExpander[OptionsPattern[]]:=Module[
  {orbital, halfFilled, fullShell, parentMax, PhaseFun,
  complementaryCFPs, daughter, conjugateDaughter,
  conjugateParent, parentTerms, daughterTerms,
  parentCFPs, daughterSeniority, daughterS, daughterL,
  parentCFP, parentTerm, parentCFPval,
  parentS, parentL, parentSeniority, phase, prefactor,
  newCFPval, key, extendedCFPs, exportFname},
(
    orbital    = 3;
    halfFilled = 2 * orbital + 1;
    fullShell  = 2 * halfFilled;
    parentMax  = 2 * orbital;

    PhaseFun  = <|
        "Judd" -> JuddCFPPhase,
        "NK" -> NKCFPPhase|>[OptionValue["PhaseFunction"]];
    PrintTemporary["Calculating CFPs using the phase system from ",
 PhaseFun];
    (* Initialize everything with lists to be filled in the next Do*)
    complementaryCFPs =
        Table[
        ({numE, term} -> {term}),
        {numE, halfFilled + 1, fullShell - 1, 1},
        {term, AllowedNKSLTerms[numE]
        }];
    complementaryCFPs = Association[Flatten[complementaryCFPs]];
    Do[(
        daughter          = parent + 1;
        conjugateDaughter = fullShell - parent;
        conjugateParent   = conjugateDaughter - 1;
        parentTerms       = AllowedNKSLTerms[parent];
        daughterTerms     = AllowedNKSLTerms[daughter];
        Do[
        (
            parentCFPs             = Rest[CFP[{daughter, daughterTerm
}]];
            daughterSeniority      = Seniority[daughterTerm];
            {daughterS, daughterL} = FindSL[daughterTerm];
            Do[
            (
                {parentTerm, parentCFPval} = parentCFP;
                {parentS, parentL}         = FindSL[parentTerm];
                parentSeniority            = Seniority[parentTerm];
                phase = PhaseFun[parent, parentS, parentL,
                                 daughterS, daughterL,
                                 parentSeniority, daughterSeniority];
                prefactor = (daughter * TPO[daughterS, daughterL]) /
                            (conjugateDaughter * TPO[parentS, parentL
]);
                prefactor = Sqrt[prefactor];
                newCFPval = phase * prefactor * parentCFPval;
                key = {conjugateDaughter, parentTerm};
                complementaryCFPs[key] = Append[complementaryCFPs[key
], {daughterTerm, newCFPval}]
            ),
            {parentCFP, parentCFPs}
            ]
        ),
        {daughterTerm, daughterTerms}
        ]
        ),
    {parent, 1, parentMax}
    ];

    complementaryCFPs[{14, "1S"}] = {"1S", {"2F",1}};
    extendedCFPs         = Join[CFP, complementaryCFPs];
    If[OptionValue["Export"];,
    (
        exportFname = FileNameJoin[{moduleDir, "data", "CFPs_extended
.m"}];
        Print["Exporting to ", exportFname];
        Export[exportFname, extendedCFPs];
    )
    ];
    Return[extendedCFPs];
```

```
)
]

GenerateCFPTable::usage = "GenerateCFPTable[] generates the table for the coefficients of fractional parentage. If the optional parameter \"Export\" is set to True then the resulting data is saved to ./data/CFPTable.m.
The data being parsed here is the file attachment B1F_ALL.TXT which comes from Velkov's thesis.";
Options[GenerateCFPTable] = {"Export" -> True};
GenerateCFPTable[OptionsPattern[]]:=Module[
  {rawText, rawLines, leadChar, configIndex,
  line, daughter, lineParts, numberCode, parsedNumber, toAppend,
  CFPTablefname},
(
  CleanWhitespace[string_]     := StringReplace[string, RegularExpression["\\s+"]->" "];
  AddSpaceBeforeMinus[string_] := StringReplace[string, RegularExpression["(?<!\\s)-"]->" -"];
  ToIntegerOrString[list_]     := Map[If[StringMatchQ[#, NumberString], ToExpression[#], #] &, list];
  CFPTable      = ConstantArray[{},7];
  CFPTable[[1]] = {{"2F",{"1S",1}}};

  (* Cleaning before processing is useful *)
  rawText  = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
  rawLines = StringTrim/@StringSplit[rawText,"\n"];
  rawLines = Select[rawLines,#!=""&];
  rawLines = CleanWhitespace/@rawLines;
  rawLines = AddSpaceBeforeMinus/@rawLines;

  Do[(
    (* the first character can be used to identify the start of a block *)
    leadChar=StringTake[line,{1}];
    (* ..FN, N is at position 50 in that line *)
    If[leadChar=="[",
    (
      configIndex=ToExpression[StringTake[line,{50}]];
      Continue[];
    )
    ];
    (* Identify which daughter term is being listed *)
    If[StringContainsQ[line,"[DAUGHTER TERM]"],
      daughter=StringSplit[line,"["][[1]];
      CFPTable[[configIndex]]=Append[CFPTable[[configIndex]],{daughter}];
      Continue[];
    ];
    (* Once we get here we are already parsing a row with coefficient data *)
    lineParts    = StringSplit[line," "];
    parent       = lineParts[[1]];
    numberCode   = ToIntegerOrString[lineParts[[3;;]]];
    parsedNumber = SquarePrimeToNormal[numberCode];
    toAppend     = {parent,parsedNumber};
    CFPTable[[configIndex]][[-1]] = Append[CFPTable[[configIndex]][[-1]], toAppend]
  ),
  {line,rawLines}];
  If[OptionValue["Export"],
    (
    CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
    Export[CFPTablefname, CFPTable];
    )
  ];
  Return[CFPTable];
)
]

GenerateCFPAssoc::usage = "GenerateCFPAssoc[export] converts the coefficients of fractional parentage into an association in which zero values are explicit. If \"Export\" is set to True, the association is exported to the file /data/CFPAssoc.m. This function requires that the association CFP be defined.";
```

```
Options[GenerateCFPAssoc] = {"Export" -> True};
GenerateCFPAssoc[OptionsPattern[]]:= (
  CFPAssoc = Association[];
  Do[
    (daughterTerms = AllowedNKSLTerms[numE];
    parentTerms    = AllowedNKSLTerms[numE - 1];
    Do[
      (
      cfps = CFP[{numE, daughter}];
      cfps = cfps[[2 ;;]];
      parents = First /@ cfps;
      Do[
        (
        key = {numE, daughter, parent};
        cfp = If[
          MemberQ[parents, parent],
          (
            idx = Position[parents, parent][[1, 1]];
            cfps[[idx]][[2]]
          ),
          0
          ];
        CFPAssoc[key] = cfp;
        ),
        {parent, parentTerms}
        ]
      ),
      {daughter, daughterTerms}
      ]
    ),
    {numE, 1, 14}
    ];
  If[OptionValue["Export"],
    (
    CFPAssocfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
    Export[CFPAssocfname, CFPAssoc];
    )
  ];
  Return[CFPAssoc];
)

CFPTerms::usage = "CFPTerms[numE] gives all the daughter and parent
  terms, together with the corresponding coefficients of fractional
  parentage, that correspond to the the f^n configuration.
CFPTerms[numE, SL] gives all the daughter and parent terms, together
  with the corresponding coefficients of fractional parentage, that
  are compatible with the given string SL in the f^n configuration.
CFPTerms[numE, L, S] gives all the daughter and parent terms,
  together with the corresponding coefficients of fractional
  parentage, that correspond to the given total orbital angular
  momentum L and total spin S n the f^n configuration. L being an
  integer, and S being integer or half-integer.
In all cases the output is in the shape of a list with enclosed lists
   having the format {daughter_term, {parent_term_1, CFP_1}, {
  parent_term_2, CFP_2}, ...}.
Only the one-body coefficients for f-electrons are provided.
In all cases it must be that 1 <= n <= 7.
";
CFPTerms[numE_] := Part[CFPTable, numE]
CFPTerms[numE_, SL_] :=
  Module[
    {NKterms, CFPconfig},
    NKterms   = {{}};
    CFPconfig = CFPTable[[numE]];
    Map[
      If[StringFreeQ[First[#], SL],
        Null,
        NKterms = Join[NKterms, {#}, 1]
      ] &,
    CFPconfig
    ];
    NKterms = DeleteCases[NKterms, {}]
  ]
CFPTerms[numE_, L_, S_] :=
Module[
  {NKterms, SL, CFPconfig},
```

```
    SL = StringJoin[ToString[2 S + 1], PrintL[L]];
    NKterms = {{}};
    CFPconfig = Part[CFPTable, numE];
    Map[
      If[StringFreeQ[First[#], SL],
        Null,
        NKterms = Join[NKterms, {#}, 1]
      ]&,
    CFPconfig
    ];
    NKterms = DeleteCases[NKterms, {}]
  ]

  (* ########### Coefficients of Fracional Parentage ########### *)
  (* ############################################################ *)

  (* ############################################################ *)
  (* ###################### Spin Orbit ######################### *)

  SpinOrbit::usage = "SpinOrbit[numE, SL, SpLp, J] returns the LSJ
    reduced matrix element ζ <SL, J|L.S|SpLp, J>. These are given as a
    function of ζ. This function requires that the association
    ReducedV1kTable be defined.
  See equations 2-106 and 2-109 in Wybourne (1965). Equivalently see
    eqn. 12.43 in TASS.";
  SpinOrbit[numE_, SL_, SpLp_, J_]:= Module[
    {S, L, Sp, Lp, orbital, sign, prefactor, val},
    orbital   = 3;
    {S, L}    = FindSL[SL];
    {Sp, Lp}  = FindSL[SpLp];
    prefactor = Sqrt[orbital * (orbital+1) * (2*orbital+1)] *
                SixJay[{L, Lp, 1}, {Sp, S, J}];
    sign      = Phaser[J + L + Sp];
    val       = sign * prefactor * ζ * ReducedV1kTable[{numE, SL, SpLp,
     1}];
    Return[val];
  ]

  GenerateSpinOrbitTable::usage = "GenerateSpinOrbitTable[nmax]
    computes the matrix values for the spin-orbit interaction for f^n
    configurations up to n = nmax. The function returns an association
    whose keys are lists of the form {n, SL, SpLp, J}. If export is set
     to True, then the result is exported to the data subfolder for the
     folder in which this package is in. It requires ReducedV1kTable to
     be defined.";
  Options[GenerateSpinOrbitTable] = {"Export" -> True};
  GenerateSpinOrbitTable[nmax_Integer:7, OptionsPattern[]]:= Module[
    {numE, J, SL, SpLp, exportFname},
  (
    SpinOrbitTable =
      Table[
        {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
      {numE, 1, nmax},
      {J, MinJ[numE], MaxJ[numE]},
      {SL,   Map[First, AllowedNKSLforJTerms[numE, J]]},
      {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
      ];
    SpinOrbitTable = Association[SpinOrbitTable];

    exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"
   }];
    If[OptionValue["Export"],
      (
        Print["Exporting to file "<>ToString[exportFname]];
        Export[exportFname, SpinOrbitTable];
      )
    ];
    Return[SpinOrbitTable];
  )
  ]

  (* ###################### Spin Orbit ######################### *)
  (* ############################################################ *)

  (* ############################################################ *)
  (* ################## Three Body Operators ################### *)
```

```
1060
1061  ParseJudd1984::usage="This function parses the data from tables 1 and
        2 of Judd from Judd, BR, and MA Suskin. \"Complete Set of
       Orthogonal Scalar Operators for the Configuration f^3\". JOSA B 1,
       no. 2 (1984): 261-65.\"";
1062  Options[ParseJudd1984] = {"Export" -> False};
1063  ParseJudd1984[OptionsPattern[]]:=(
1064    ParseJuddTab1[str_] := (
1065      strR = ToString[str];
1066      strR = StringReplace[strR, ".5" -> "^(1/2)"];
1067      num = ToExpression[strR];
1068      sign = Sign[num];
1069      num = sign*Simplify[Sqrt[num^2]];
1070      If[Round[num] == num, num = Round[num]];
1071      Return[num]);
1072
1073    (* Parse table 1 from Judd 1984 *)
1074    judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"
       }];
1075    data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
1076    headers = data[[1]];
1077    data = data[[2 ;;]];
1078    data = Transpose[data];
1079    \[Psi] = Select[data[[1]], # != "" &];
1080    \[Psi]p = Select[data[[2]], # != "" &];
1081    matrixKeys = Transpose[{\[Psi], \[Psi]p}];
1082    data = data[[3 ;;]];
1083    cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data}];
1084    cols = Select[cols, Length[#] == 21 &];
1085    tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
1086    tab1 = Transpose[Prepend[Transpose[tab1], headers]];
1087
1088    (* Parse table 2 from Judd 1984 *)
1089    judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"
       }];
1090    data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
1091    headers = data[[1]];
1092    data = data[[2 ;;]];
1093    data = Transpose[data];
1094    {operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} =
        data[[;; 4]];
1095    multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
1096    multiFactorValues = AssociationThread[multiFactorSymbols ->
       multiFactorValues];
1097
1098    (*scale values of table 1 given the values in table 2*)
1099    oppyS = {};
1100    normalTable =
1101      Table[header = col[[1]];
1102        If[StringContainsQ[header, " "],
1103          (
1104            multiplierSymbol = StringSplit[header, " "][[1]];
1105            multiplierValue = multiFactorValues[multiplierSymbol];
1106            operatorSymbol = StringSplit[header, " "][[2]];
1107            oppyS = Append[oppyS, operatorSymbol];
1108          ),
1109          (
1110            multiplierValue = 1;
1111            operatorSymbol = header;
1112          )
1113        ];
1114        normalValues = 1/multiplierValue*col[[2 ;;]];
1115        Join[{operatorSymbol}, normalValues], {col, tab1[[3 ;;]]}
1116      ];
1117
1118    (*Create an association for the matrix elements in the f^3 config*)
1119    juddOperators = Association[];
1120    Do[(
1121      col      = normalTable[[colIndex]];
1122      opLabel  = col[[1]];
1123      opValues = col[[2 ;;]];
1124      opMatrix = AssociationThread[matrixKeys -> opValues];
1125      Do[(
1126        opMatrix[Reverse[mKey]] = opMatrix[mKey]
1127        ),
1128      {mKey, matrixKeys}
```

```
      ];
      juddOperators[{3, opLabel}] = opMatrix),
      {colIndex, 1, Length[normalTable]}
    ];

    (* special case of t2 in f3 *)
    (* this is the same as getting the matrix elements from Judd 1966
    *)
    numE = 3;
    e3Op      = juddOperators[{3, "e_{3}"}];
    t2prime   = juddOperators[{3, "t_{2}^{'}"}];
    prefactor = 1/(70 Sqrt[2]);
    t2Op = (# -> (t2prime[#] + prefactor*e3Op[#])) & /@ Keys[t2prime];
    t2Op = Association[t2Op];
    juddOperators[{3, "t_{2}"}] = t2Op;

    (*Special case of t11 in f3*)
    t11 = juddOperators[{3, "t_{11}"}];
    eβprimeOp = juddOperators[{3, "e_{\\beta}^{'}"}];
    t11primeOp = (# -> (t11[#] + Sqrt[3/385] eβprimeOp[#])) & /@ Keys[
    t11];
    t11primeOp = Association[t11primeOp];
    juddOperators[{3, "t_{11}^{'}"}] = t11primeOp;
    If[OptionValue["Export"],
      (
        (*export them*)
        PrintTemporary["Exporting ..."];
        exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.m
    "}];
        Export[exportFname, juddOperators];
      )
    ];
    Return[juddOperators];
  )

GenerateThreeBodyTables::usage="This function generates the matrix
    elements for the three body operators using the coefficients of
    fractional parentage, including those beyond f^7.";
Options[GenerateThreeBodyTables] = {"Export" -> False};
GenerateThreeBodyTables[nmax_Integer : 14, OptionsPattern[]] := (
  tiKeys = {"t_{2}", "t_{2}^{'}", "t_{3}", "t_{4}", "t_{6}", "t_{7}",
    "t_{8}", "t_{11}", "t_{11}^{'}", "t_{12}", "t_{14}", "t_{15}",
    "t_{16}", "t_{17}", "t_{18}", "t_{19}"};
  TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
  juddOperators = ParseJudd1984[];
  (* op3MatrixElement[SL, SpLp, opSymbol] returns the value for the
    reduced matrix element of the operator opSymbol for the terms {SL,
    SpLp} in the f^3 configuration. *)
  op3MatrixElement[SL_, SpLp_, opSymbol_] := (
    jOP = juddOperators[{3, opSymbol}];
    key = {SL, SpLp};
    val = If[MemberQ[Keys[jOP], key],
      jOP[key],
      0];
    Return[val];
    );
  (*ti: This is the implementation of formula (2) in Judd & Suskin
    1984. It computes the matrix elements of ti in f^n by using the
    matrix elements in f3 and the coefficients of fractional parentage.
     If the option \"Fast\" is set to True then the values for n>7 are
    simply computed as the negatives of the values in the complementary
     configuration; this except for t2 and t11 which are treated as
    special cases. *)
  Options[ti] = {"Fast" -> True};
  ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
   Module[{nn, S, L, Sp, Lp,
      cfpSL, cfpSpLp,
      parentSL, parentSpLp, tnk, tnks},
    {S, L}   = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    fast     = OptionValue["Fast"];
    numH = 14 - nE;
    If[fast && Not[MemberQ[{"t_{2}","t_{11}"},tiKey]] && nE > 7,
      Return[-tktable[{numH, SL, SpLp, tiKey}]]
    ];
    If[(S == Sp && L == Lp),
```

```
    (
     cfpSL   = CFP[{nE, SL}];
     cfpSpLp = CFP[{nE, SpLp}];
     tnks = Table[(
         parentSL   = cfpSL[[nn, 1]];
         parentSpLp = cfpSpLp[[mm, 1]];
         cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
         tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
         ),
         {nn, 2, Length[cfpSL]},
         {mm, 2, Length[cfpSpLp]}
         ];
     tnk = Total[Flatten[tnks]];
     ),
    tnk = 0;
    ];
  Return[ nE / (nE - opOrder) * tnk];];
(*Calculate the matrix elements of t^i for n up to nmax*)
tktable = <||>;
Do[(
  Do[(
    tkValue = Which[numE <= 2,
      (*Initialize n=1,2 with zeros*)
      0,
      numE == 3,
      (*Grab matrix elem in f^3 from Judd 1984*)
      SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
      True,
      SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2,
 3]]]
      ];
    tktable[{numE, SL, SpLp, opKey}] = tkValue;
    ),
  {SL, AllowedNKSLTerms[numE]},
  {SpLp, AllowedNKSLTerms[numE]},
  {opKey, Append[tiKeys, "e_{3}"]}
  ];
  PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], "
 configuration complete"]];
  ),
{numE, 1, nmax}
];

(* Now use those matrix elements to determine their sum as weighted
 by their corresponding strengths Ti *)
ThreeBodyTable = <||>;
Do[
  Do[
  (
    ThreeBodyTable[{numE, SL, SpLp}] = (
      Sum[(
        If[tiKey == "t_{2}", t2Switch, 1] *
        tktable[{numE, SL, SpLp, tiKey}] *
        TSymbolsAssoc[tiKey] +
        If[tiKey == "t_{2}", 1 - t2Switch, 0] *
        (-tktable[{14 - numE, SL, SpLp, tiKey}]) *
        TSymbolsAssoc[tiKey]
        ),
      {tiKey, tiKeys}
      ]
    );
  ),
  {SL, AllowedNKSLTerms[numE]},
  {SpLp, AllowedNKSLTerms[numE]}
  ];
PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix
 complete"]];,
{numE, 1, 7}
];

ThreeBodyTables = Table[(
  terms = AllowedNKSLTerms[numE];
  singleThreeBodyTable =
    Table[
      {SL, SLp} -> ThreeBodyTable[{numE, SL, SLp}],
      {SL, terms},
```

```
        {SLp, terms}
      ];
    singleThreeBodyTable  = Flatten[singleThreeBodyTable];
    singleThreeBodyTables = Table[(
        notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
        reps = ConstantArray[0, Length[TSymbols]];
        reps[[notNullPosition]] = 1;
        rep = AssociationThread[TSymbols -> reps];
        notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
        ),
      {notNullSymbol, TSymbols}
      ];
    singleThreeBodyTables = Association[singleThreeBodyTables];
    numE -> singleThreeBodyTables),
    {numE, 1, 7}];

  ThreeBodyTables = Association[ThreeBodyTables];
  If[OptionValue["Export"], (
    threeBodyTablefname = FileNameJoin[{moduleDir, "data", "
    ThreeBodyTable.m"}];
    Export[threeBodyTablefname, ThreeBodyTable];
    threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "
    ThreeBodyTables.m"}];
    Export[threeBodyTablesfname, ThreeBodyTables];
    )
   ];
  Return[{ThreeBodyTable, ThreeBodyTables}];)

  ScalarOperatorProduct::usage="ScalarOperatorProduct[op1, op2, numE]
    calculated the innerproduct between the two scalar operators op1
    and op2.";
  ScalarOperatorProduct[op1_, op2_, numE_] := Module[
    {terms, S, L, factor, term1, term2},
    (
    terms = AllowedNKSLTerms[numE];
    Simplify[
      Sum[(
        {S, L} = FindSL[term1];
        factor = TPO[S, L];
        factor * op1[{term1, term2}] * op2[{term2, term1}]
        ),
      {term1, terms},
      {term2, terms}
      ]
    ]
    )
  ];

  (* ################## Three Body Operators ##################### *)
  (* ############################################################ *)

  (* ############################################################ *)
  (* ################### Reduced SOO and ECSO ################### *)

  ReducedT11inf2::usage="ReducedT11inf2[SL, SpLp] returns the reduced
    matrix element of the scalar component of the double tensor T11 for
     the given SL terms SL, SpLp.
  Data used here for m0, m2, m4 is from Table II of Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
    Interactions for f Electrons. Physical Review 169, no. 1 (1968):
    130.
  ";
  ReducedT11inf2[SL_, SpLp_] :=
    Module[{T11inf2},
    T11inf2 = <|
      {"1S", "3P"} -> 6 M0 + 2 M2 + 10/11 M4,
      {"3P", "3P"} -> -36 M0 - 72 M2 - 900/11 M4,
      {"3P", "1D"} -> -Sqrt[(2/15)] (27 M0 + 14 M2 + 115/11 M4),
      {"1D", "3F"} -> Sqrt[2/5] (23 M0 + 6 M2 - 195/11 M4),
      {"3F", "3F"} -> 2 Sqrt[14] (-15 M0 - M2 + 10/11 M4),
      {"3F", "1G"} -> Sqrt[11] (-6 M0 + 64/33 M2 - 1240/363 M4),
      {"1G", "3H"} -> Sqrt[2/5] (39 M0 - 728/33 M2 - 3175/363 M4),
      {"3H", "3H"} -> 8/Sqrt[55] (-132 M0 + 23 M2 + 130/11 M4),
      {"3H", "1I"} -> Sqrt[26] (-5 M0 - 30/11 M2 - 375/1573 M4)
      |>;
    Which[
```

```
      MemberQ[Keys[T11inf2],{SL,SpLp}],
        Return[T11inf2[{SL,SpLp}]],
      MemberQ[Keys[T11inf2],{SpLp,SL}],
        Return[T11inf2[{SpLp,SL}]],
      True,
        Return[0]
    ]
    ];

Reducedt11inf2::usage="Reducedt11inf2[SL, SpLp] returns the reduced
  matrix element in f^2 of the double tensor operator t11 for the
  corresponding given terms {SL, SpLp}.
Values given here are those from Table VII of \"Judd, BR, HM
  Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
  Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
  130.\"
";
Reducedt11inf2[SL_, SpLp_]:= Module[
  {t11inf2},
  t11inf2 = <|
    {"1S", "3P"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
    {"3P", "3P"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
    {"3P", "1D"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
    {"1D", "3F"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
    {"3F", "3F"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
    {"3F", "1G"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
    {"1G", "3H"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
    {"3H", "3H"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
    {"3H", "1I"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
    |>;
  Which[
    MemberQ[Keys[t11inf2],{SL,SpLp}],
      Return[t11inf2[{SL,SpLp}]],
    MemberQ[Keys[t11inf2],{SpLp,SL}],
      Return[t11inf2[{SpLp,SL}]],
    True,
      Return[0]
  ]
]

ReducedSOOandECSOinf2::usage="ReducedSOOandECSOinf2[SL, SpLp] returns
   the reduced matrix element corresponding to the operator (T11 +
  t11 - a13 * z13 / 6) for the terms {SL, SpLp}. This combination of
  operators corresponds to the spin-other-orbit plus ECSO interaction
  .
The T11 operator corresponds to the spin-other-orbit interaction, and
   the t11 operator (associated with electrostatically-correlated
  spin-orbit) originates from configuration interaction analysis. To
  their sum the a facor proportional to operator z13 is subtracted
  since its effect is seen as redundant to the spin-orbit interaction
  . The factor of 1/6 is not on Judd's 1966 paper, but it is on \"
  Chen, Xueyuan, Guokui Liu, Jean Margerie, and Michael F Reid. \"A
  Few Mistakes in Widely Used Data Files for Fn Configurations
  Calculations.\" Journal of Luminescence 128, no. 3 (2008): 421-27\"
  .

The values for the reduced matrix elements of z13 are obtained from
  Table IX of the same paper. The value for a13 is from table VIII.";
ReducedSOOandECSOinf2[SL_, SpLp_] :=
Module[{a13, z13, z13inf2, matElement, redSOOandECSOinf2},
  a13 = (-33 M0 + 3 M2 + 15/11 M4 -
        6 P0 + 3/2 (35 P2 + 77 P4 + 143 P6));
  z13inf2 = <|
      {"1S","3P"} -> 2,
      {"3P","3P"} -> 1,
      {"3P","1D"} -> -Sqrt[(15/2)],
      {"1D","3F"} -> Sqrt[10],
      {"3F","3F"} -> Sqrt[14],
      {"3F","1G"} -> -Sqrt[11],
      {"1G","3H"} -> Sqrt[10],
      {"3H","3H"} -> Sqrt[55],
      {"3H","1I"} -> -Sqrt[(13/2)]
      |>;
  matElement = Which[
    MemberQ[Keys[z13inf2],{SL,SpLp}],
      z13inf2[{SL,SpLp}],
```

```
      MemberQ[Keys[z13inf2],{SpLp,SL}],
        z13inf2[{SpLp,SL}],
      True,
        0
    ];
    redSOOandECSOinf2 = (
        ReducedT11inf2[SL, SpLp] +
        Reducedt11inf2[SL, SpLp] -
        a13 / 6 * matElement
    );
    redSOOandECSOinf2 = SimplifyFun[redSOOandECSOinf2];
    Return[redSOOandECSOinf2];
    ];

  ReducedSOOandECSOinfn::usage="ReducedSOOandECSOinfn[numE, SL, SpLp]
    calculates the reduced matrix elements of the (spin-other-orbit +
    ECSO) operator for the f^n configuration corresponding to the terms
     SL and SpLp. This is done recursively, starting from tabulated
    values for f^2 from \"Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
     Physical Review 169, no. 1 (1968): 130.\", and by using equation
    (4) of that same paper.
  ";
  ReducedSOOandECSOinfn[numE_, SL_, SpLp_]:= Module[
    {spin, orbital, t, S, L, Sp, Lp, idx1, idx2, cfpSL, cfpSpLp,
    parentSL, Sb, Lb, Sbp, Lbp, parentSpLp, funval},
    {spin, orbital} = {1/2, 3};
    {S, L}   = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    t = 1;
    cfpSL    = CFP[{numE, SL}];
    cfpSpLp  = CFP[{numE, SpLp}];
    funval =
      Sum[
        (
          parentSL = cfpSL[[idx2, 1]];
          parentSpLp = cfpSpLp[[idx1, 1]];
          {Sb, Lb}   = FindSL[parentSL];
          {Sbp, Lbp} = FindSL[parentSpLp];
          phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
          (
            phase *
            cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
            SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
            SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
            SOOandECSOLSTable[{numE - 1, parentSL, parentSpLp}]
          )
        ),
      {idx1, 2, Length[cfpSpLp]},
      {idx2, 2, Length[cfpSL]}
      ];
    funval *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
    Return[funval];
  ];

  GenerateSOOandECSOLSTable::usage="GenerateSOOandECSOLSTable[nmax]
    generates the LS reduced matrix elements of the spin-other-orbit +
    ECSO for the f^n configurations up to n=nmax. The values for n=1
    and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
     Physical Review 169, no. 1 (1968): 130.\", and the values for n>2
    are calculated recursively using equation (4) of that same paper.
    The values are then exported to a file \"ReducedSOOandECSOLSTable.m
    \" in the data folder of this module. The values are also returned
    as an association.";
  Options[GenerateSOOandECSOLSTable] = {"Progress" -> True, "Export" ->
     True};
  GenerateSOOandECSOLSTable[nmax_Integer, OptionsPattern[]]:= (
    If[And[OptionValue["Progress"], frontEndAvailable],
      (
        numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
    numE]]^2, {numE, 1, nmax}]];
        counters   = Association[Table[numE->0, {numE, 1, nmax}]];
        totalIters = Total[Values[numItersai[[1;;nmax]]]];
        template1  = StringTemplate["Iteration `numiter` of `totaliter`
    "];
```

```
        template2  = StringTemplate["`remtime` min remaining"];
  template3 = StringTemplate["Iteration speed = `speed` ms/it"];
        template4  = StringTemplate["Time elapsed = `runtime` min"];
        progBar = PrintTemporary[
          Dynamic[
            Pane[
              Grid[{
                    {Superscript["f", numE]},
                    {template1[<|"numiter"->numiter, "totaliter"->
  totalIters|>]},
                    {template4[<|"runtime"->Round[QuantityMagnitude[
  UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                    {template2[<|"remtime"->Round[QuantityMagnitude[
  UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
  ]], 0.1]|>]},
                    {template3[<|"speed"->Round[QuantityMagnitude[Now-
  startTime, "ms"]/(numiter), 0.01]|>]}, {ProgressIndicator[Dynamic[
  numiter], {1, totalIters}]}
                  },
                  Frame->All
              ],
              Full,
              Alignment->Center
            ]
          ]
        ];
      )
    ];
    SOOandECSOLSTable = <||>;
    numiter   = 1;
    startTime = Now;
    Do[
      (
        numiter+= 1;
        SOOandECSOLSTable[{numE, SL, SpLp}] = Which[
          numE==1,
          0,
          numE==2,
          SimplifyFun[ReducedSOOandECSOinf2[SL, SpLp]],
          True,
          SimplifyFun[ReducedSOOandECSOinfn[numE,  SL, SpLp]]
        ];
      ),
    {numE, 1, nmax},
    {SL, AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]}
    ];
    If[And[OptionValue["Progress"], frontEndAvailable],
      NotebookDelete[progBar]];
    If[OptionValue["Export"],
      (fname = FileNameJoin[{moduleDir, "data", "
  ReducedSOOandECSOLSTable.m"}];
      Export[fname, SOOandECSOLSTable];
      )
    ];
    Return[SOOandECSOLSTable];
  );

  (* ################## Reduced SOO and ECSO ################### *)
  (* ############################################################ *)

  (* ############################################################ *)
  (* ###################### Spin-Spin ########################### *)

  ReducedT22inf2::usage="ReducedT22inf2[SL, SpLp] returns the reduced
    matrix element of the scalar component of the double tensor T22 for
     the terms SL, SpLp in f^2.
  Data used here for m0, m2, m4 is from Table I of Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
    Interactions for f Electrons. Physical Review 169, no. 1 (1968):
    130.
  ";
  ReducedT22inf2[SL_, SpLp_] :=
    Module[{statePosition, PsiPsipStates, m0, m2, m4, Tkk2m},
    T22inf2 = <|
    {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
```

```
    {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
    {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
    {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
    {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
    |>;
    Which[
      MemberQ[Keys[T22inf2],{SL,SpLp}],
        Return[T22inf2[{SL,SpLp}]],
      MemberQ[Keys[T22inf2],{SpLp,SL}],
        Return[T22inf2[{SpLp,SL}]],
      True,
        Return[0]
    ]
    ];

  ReducedT22infn::usage="ReducedT22infn[n, SL, SpLp] calculates the
    reduced matrix element of the T22 operator for the f^n
    configuration corresponding to the terms SL and SpLp. This is the
    operator corresponding to the inter-electron between spin.
  It does this by using equation (4) of \"Judd, BR, HM Crosswhite, and
    Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
    Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
  ";
  ReducedT22infn[numE_, SL_, SpLp_]:= Module[
    {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp,
    parentSL, parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
    {spin, orbital} = {1/2, 3};
    {S, L}   = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    t = 2;
    cfpSL    = CFP[{numE, SL}];
    cfpSpLp  = CFP[{numE, SpLp}];
    Tnkk =
      Sum[(
        parentSL = cfpSL[[idx2, 1]];
        parentSpLp = cfpSpLp[[idx1, 1]];
        {Sb, Lb} = FindSL[parentSL];
        {Sbp, Lbp} = FindSL[parentSpLp];
        phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
        (
          phase *
          cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
          SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
          SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
          T22Table[{numE - 1, parentSL, parentSpLp}]
        )
      ),
      {idx1, 2, Length[cfpSpLp]},
      {idx2, 2, Length[cfpSL]}
      ];
    Tnkk *= numE / (numE - 2) * Sqrt[TPO[S,Sp,L,Lp]];
    Return[Tnkk];
    ];

  GenerateT22Table::usage="GenerateT22Table[nmax] generates the LS
    reduced matrix elements for the double tensor operator T22 in f^n
    up to n=nmax. If the option \"Export\" is set to true then the
    resulting association is saved to the data folder. The values for n
    =1 and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
     Physical Review 169, no. 1 (1968): 130.\", and the values for n>2
    are calculated recursively using equation (4) of that same paper.
  This is an intermediate step to the calculation of the reduced matrix
     elements of the spin-spin operator.";
  Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
  GenerateT22Table[nmax_Integer, OptionsPattern[]]:= (
    If[And[OptionValue["Progress"], frontEndAvailable],
      (
        numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
    numE]]^2, {numE, 1, nmax}]];
        counters = Association[Table[numE->0, {numE, 1, nmax}]];
        totalIters = Total[Values[numItersai[[1;;nmax]]]];
        template1 = StringTemplate["Iteration `numiter` of `totaliter`"
    ];
        template2 = StringTemplate["`remtime` min remaining"];template3
     = StringTemplate["Iteration speed = `speed` ms/it"];
```

```
        template4 = StringTemplate["Time elapsed = `runtime` min"];
        progBar = PrintTemporary[
          Dynamic[
            Pane[
              Grid[{{Superscript["f", numE]},
                    {template1[<|"numiter"->numiter, "totaliter"->totalIters|>]},
                    {template4[<|"runtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                    {template2[<|"remtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"]], 0.1]|>]},
                    {template3[<|"speed"->Round[QuantityMagnitude[Now-startTime, "ms"]/(numiter), 0.01]|>]},
                    {ProgressIndicator[Dynamic[numiter], {1, totalIters}]}},
                    Frame->All],
                    Full,
                    Alignment->Center]
                ]
              ];
      )
    ];
    T22Table = <||>;
    startTime = Now;
    numiter = 1;
    Do[
      (
        numiter+= 1;
        T22Table[{numE, SL, SpLp}] = Which[
          numE==1,
          0,
          numE==2,
          SimplifyFun[ReducedT22inf2[SL, SpLp]],
          True,
          SimplifyFun[ReducedT22infn[numE,  SL, SpLp]]
        ];
      ),
    {numE, 1, nmax},
    {SL,   AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]}
    ];
    If[And[OptionValue["Progress"],frontEndAvailable],
      NotebookDelete[progBar]
    ];
    If[OptionValue["Export"],
      (
        fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
        Export[fname, T22Table];
      )
    ];
    Return[T22Table];
  );

  SpinSpin::usage="SpinSpin[n, SL, SpLp, J] returns the matrix element
    <|SL,J|spin-spin|SpLp,J|> for the spin-spin operator within the
    configuration f^n. This matrix element is independent of MJ. This
    is obtained by querying the relevant reduced matrix element by
    querying the association T22Table and putting in the adequate phase
     and 6-j symbol.
  This is calculated according to equation (3) in \"Judd, BR, HM
    Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
    Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
    130.\"
  \".
  ";
  SpinSpin[numE_, SL_, SpLp_, J_] := Module[
    {S, L, Sp, Lp, α, val},
    α = 2;
    {S, L} = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    val = (
            Phaser[Sp + L + J] *
            SixJay[{Sp, Lp, J}, {L, S, α}] *
            T22Table[{numE, SL, SpLp}]
          );
```

```
    Return[val]
    ];

GenerateSpinSpinTable::usage="GenerateSpinSpinTable[nmax] generates
  the matrix elements in the |LSJ> basis for the (spin-other-orbit +
  electrostatically-correlated-spin-orbit) operator. It returns an
  association where the keys are of the form {numE, SL, SpLp, J}. If
  the option \"Export\" is set to True then the resulting object is
  saved to the data folder. Since this is a scalar operator, there is
   no MJ dependence. This dependence only comes into play when the
  crystal field contribution is taken into account.";
Options[GenerateSpinSpinTable] = {"Export"->False};
GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
(
  SpinSpinTable = <||>;
  PrintTemporary[Dynamic[numE]];
  Do[
    SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp, J
  ]);,
  {numE, 1, nmax},
  {J, MinJ[numE], MaxJ[numE]},
  {SL,   First /@ AllowedNKSLforJTerms[numE, J]},
  {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
  ];
  If[OptionValue["Export"],
  (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
    Export[fname, SpinSpinTable];
    )
  ];
  Return[SpinSpinTable];
  );

(* ############################################################## *)
(* ######################## Spin-Spin ########################### *)

(* ############################################################## *)
(* ## Spin-Other-Orbit and Electrostatically-Correlated-Spin-Orbit ##
   *)

SOOandECSO::usage="SOOandECSO[n, SL, SpLp, J] returns the matrix
  element <|SL,J|spin-spin|SpLp,J|> for the combined effects of the
  spin-other-orbit interaction and the electrostatically-correlated-
  spin-orbit (which originates from configuration interaction effects
  ) within the configuration f^n. This matrix element is independent
  of MJ. This is obtained by querying the relevant reduced matrix
  element by querying the association SOOandECSOLSTable and putting
  in the adequate phase and 6-j symbol. The SOOandECSOLSTable puts
  together the reduced matrix elements from three operators.
This is calculated according to equation (3) in \"Judd, BR, HM
  Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
  Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
  130.\".
";
SOOandECSO[numE_, SL_, SpLp_, J_]:= Module[
  {S, Sp, L, Lp, \[Alpha], val},
  \[Alpha] = 1;
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  val = (
           Phaser[Sp + L + J] *
           SixJay[{Sp, Lp, J}, {L, S, \[Alpha]}] *
           SOOandECSOLSTable[{numE, SL, SpLp}]
         );
  Return[val];
]

Prescaling = {P2 -> P2/225, P4 -> P4/1089, P6 -> 25 * P6 / 184041};

GenerateSOOandECSOTable::usage="GenerateSOOandECSOTable[nmax]
  generates the matrix elements in the |LSJ> basis for the (spin-
  other-orbit + electrostatically-correlated-spin-orbit) operator. It
   returns an association where the keys are of the form {n, SL, SpLp
  , J}. If the option \"Export\" is set to True then the resulting
  object is saved to the data folder. Since this is a scalar operator
  , there is no MJ dependence. This dependence only comes into play
  when the crystal field contribution is taken into account.";
```

```
Options[GenerateSOOandECSOTable] = {"Export"->False}
GenerateSOOandECSOTable[nmax_, OptionsPattern[]]:= (
  SOOandECSOTable = <||>;
  Do[
    SOOandECSOTable[{numE, SL, SpLp, J}] = (SOOandECSO[numE, SL, SpLp
  , J] /. Prescaling);,
  {numE, 1, nmax},
  {J, MinJ[numE], MaxJ[numE]},
  {SL,   First /@ AllowedNKSLforJTerms[numE, J]},
  {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
  ];
  If[OptionValue["Export"],
  (
    fname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
    Export[fname, SOOandECSOTable];
  )
  ];
  Return[SOOandECSOTable];
);

(* ## Spin-Other-Orbit and Electrostatically-Correlated-Spin-Orbit ##
   *)
(* ############################################################ *)

(* ############################################################ *)
(* ################### Magnetic Interactions ################## *)

MagneticInteractions::usage="MagneticInteractions[{numE, SLJ, SLJp, J
  }] returns the matrix element of the magnetic interaction between
  the terms SLJ and SLJp in the f^n configuration. The interaction is
   given by the sum of the spin-spin interaction and the SOO and ECSO
   interactions. The spin-spin interaction is given by the function
  SpinSpin[{numE, SLJ, SLJp, J}]. The SOO and ECSO interactions are
  given by the function SOOandECSO[{numE, SLJ, SLJp, J}]. The
  function requires chenDeltas to be loaded into the session. The
  option \"ChenDeltas\" can be use to include or exclude the Chen
  deltas from the calculation. The default is to exclude them.";
Options[MagneticInteractions] = {"ChenDeltas" -> False};
MagneticInteractions[{numE_, SLJ_, SLJp_, J_}, OptionsPattern[]] :=
  (
    key = {numE, SLJ, SLJp, J};
    ss = \[Sigma]SS * SpinSpinTable[key];
    sooandecso = SOOandECSOTable[key];
    total = ss + sooandecso;
    total = SimplifyFun[total];
    If[
      Not[OptionValue["ChenDeltas"]],
      Return[total]
    ];
    (* In the type A errors the wrong values are different *)
    If[MemberQ[Keys[chenDeltas["A"]], {numE, SLJ, SLJp}],
      (
        {S, L} = FindSL[SLJ];
        {Sp, Lp} = FindSL[SLJp];
        phase   = Phaser[Sp + L + J];
        Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
        Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
        {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ,
  SLJp}]["wrong"];
        total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
        total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
        total  = total /. Prescaling;
        total  = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
      )
    ];
    (* In the type B errors the wrong values are zeros all around *)
    If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
      (
        {S, L} = FindSL[SLJ];
        {Sp, Lp} = FindSL[SLJp];
        phase = Phaser[Sp + L + J];
        Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
        Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
        {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
        total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
        total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
```

```
          total  = total /. Prescaling;
          total  = wChErrB * total + (1 - wChErrB) * (ss + sooandecso)
        )
      ];
      Return[total];
    )

  (* ################## Magnetic Interactions ################## *)
  (* ########################################################### *)

  (* ########################################################### *)
  (* ###################### Crystal Field ###################### *)

  Cqk::usage = "Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp]. In Wybourne
     (1965) see equations 6-3, 6-4, and 6-5. Also in TASS see equation
    11.53.";
  Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
    {S, Sp, L, Lp, orbital, val},
    orbital = 3;
    {S, L}   = FindSL[NKSL];
    {Sp, Lp} = FindSL[NKSLp];
    f1  = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
    val =
      If[f1==0,
        0,
        (
          f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
          If[f2==0,
            0,
            (
              f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
              If[f3==0,
                0,
                (
                  (
                    Phaser[J - M + S + Lp + J + k] *
                    Sqrt[TPO[J, Jp]] *
                    f1 *
                    f2 *
                    f3 *
                    Ck[orbital, k]
                  )
                )
              ]
            )
          ]
        )
      ];
    val
    ]

  Bqk[q_, 2] := {B02/2, B12, B22}[[q + 1]];
  Bqk[q_, 4] := {B04/2, B14, B24, B34, B44}[[q + 1]];
  Bqk[q_, 6] := {B06/2, B16, B26, B36, B46, B56, B66}[[q + 1]];

  Sqk[q_, 2] := {Sm22, Sm12, S02,  S12,  S22}[[q + 3]];
  Sqk[q_, 4] := {Sm44, Sm34, Sm24, Sm14, S04,  S14,  S24, S34, S44}[[q
    + 5]];
  Sqk[q_, 6] := {Sm66, Sm56, Sm46, Sm36, Sm26, Sm16, S06, S16, S26, S36
    , S46, S56, S66}[[q + 7]];

  CrystalField::usage = "CrystalField[n, NKSL, J, M, NKSLp, Jp, Mp]
    gives the general expression for the matrix element of the crystal
    field Hamiltonian parametrized with Bqk and Sqk coefficients as a
    sum over spherical harmonics Cqk.
  Sometimes this expression only includes Bqk coefficients, see for
    example eqn 6-2 in Wybourne (1965), but one may also split the
    coefficient into real and imaginary parts as is done here, in an
    expression that is patently Hermitian.";
  CrystalField[numE_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := (
    Sum[
      (
        cqk  = Cqk[numE,  q, k, NKSL, J, M, NKSLp, Jp, Mp];
        cmqk = Cqk[numE, -q, k, NKSL, J, M, NKSLp, Jp, Mp];
        Bqk[q, k]   * (cqk + (-1)^q * cmqk) +
        I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
```

```
        ),
    {k, {2, 4, 6}},
    {q, 0, k}
    ]
  )

  TotalCFIters::usage = "TotalIters[i, j] returns total number of
    function evaluations for calculating all the matrix elements for
    the \!\(\*SuperscriptBox[\(f\), \(i\)]\) to the \!\(\*
    SuperscriptBox[\(f\), \(j\)]\) configurations.";
  TotalCFIters[i_, j_] := (
    numIters = {196, 8281, 132496, 1002001, 4008004, 9018009,
    11778624};
    Return[Total[numIters[[i ;; j]]]];
    )

  GenerateCrystalFieldTable::usage = "GenerateCrystalFieldTable[{numEs
    ]] computes the matrix values for the crystal field interaction for
     f^n configurations the given list of numE in  numEs. The function
    calculates the association CrystalFieldTable with keys of the form
    {numE, NKSL, J, M, NKSLp, Jp, Mp}. If the option \"Export\" is set
    to True, then the result is exported to the data subfolder for the
    folder in which this package is in. If the option \"Progress\" is
    set to True then an interactive progress indicator is shown. If \"
    Compress\" is set to true the exported values are compressed when
    exporting.";
  Options[GenerateCrystalFieldTable] = {"Export" -> False, "Progress"
    -> True, "Compress" -> True}
  GenerateCrystalFieldTable[numEs_List:{1,2,3,4,5,6,7}, OptionsPattern
    []]:= (
    ExportFun =
    If[OptionValue["Compress"],
      ExportMZip,
      Export
    ];
    numiter = 1;
    template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
    template2 = StringTemplate["`remtime` min remaining"];
    template3 = StringTemplate["Iteration speed = `speed` ms/it"];
    template4 = StringTemplate["Time elapsed = `runtime` min"];
    totalIter = Total[TotalCFIters[#, #] & /@ numEs];
    freebies = 0;
    startTime = Now;
    If[And[OptionValue["Progress"], frontEndAvailable],
      progBar = PrintTemporary[
        Dynamic[
          Pane[
            Grid[
              {
                {Superscript["f", numE]},
                {template1[<|"numiter" -> numiter, "totaliter" ->
    totalIter|>]},
                {template4[<|"runtime" -> Round[QuantityMagnitude[
    UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
                {template2[<|"remtime" -> Round[QuantityMagnitude[
    UnitConvert[(Now - startTime)/(numiter - freebies) * (totalIter -
    numiter), "min"]], 0.1]|>]},
                {template3[<|"speed" -> Round[QuantityMagnitude[Now -
    startTime, "ms"]/(numiter-freebies), 0.01]|>]},
                {ProgressIndicator[Dynamic[numiter], {1, totalIter}]}
              },
            Frame -> All
            ],
          Full,
          Alignment -> Center
          ]
        ]
      ];
    ];
    Do[
      (
        exportFname = FileNameJoin[{moduleDir, "data", "
    CrystalFieldTable_f"<>ToString[numE]<>".m"}];
        If[FileExistsQ[exportFname],
          Print["File exists, skipping ..."];
          numiter+=  TotalCFIters[numE, numE];
```

```
          freebies+= TotalCFIters[numE, numE];
          Continue[];
        ];
        CrystalFieldTable = <||>;
        Do[
          (
            numiter+= 1;
            CrystalFieldTable[{numE, NKSL, J, M, NKSLp, Jp, Mp}] =
  CrystalField[numE, NKSL, J, M, NKSLp, Jp, Mp];
          ),
        {J, MinJ[numE], MaxJ[numE]},
        {Jp, MinJ[numE], MaxJ[numE]},
        {M, AllowedMforJ[J]},
        {Mp, AllowedMforJ[Jp]},
        {NKSL , First /@ AllowedNKSLforJTerms[numE, J]},
        {NKSLp, First /@ AllowedNKSLforJTerms[numE, Jp]}
        ];
        If[And[OptionValue["Progress"],frontEndAvailable],
          NotebookDelete[progBar]
        ];
        If[OptionValue["Export"],
          (
            Print["Exporting to file "<>ToString[exportFname]];
            ExportFun[exportFname, CrystalFieldTable];
          )
        ];
      ),
    {numE, numEs}
    ]
  )

  (* ###################### Crystal Field ######################### *)
  (* ################################################################ *)

  (* ################################################################ *)
  (* #### Configuration-Interaction via Casimir Operators ###### *)

  CasimirSO3::usage = "CasimirSO3[SL, SpLp] returns LS reduced matrix
    element of the configuration interaction term corresponding to the
    Casimir operator of R3.";
  CasimirSO3[{SL_, SpLp_}] := (
    {S, L} = FindSL[SL];
    If[SL == SpLp,
      α * L * (L + 1),
      0
    ]
  )

  GG2U::usage = "GG2U is an association whose keys are labels for the
    irreducible representations of group G2 and whose values are the
    eigenvalues of the corresponding Casimir operator.
  Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\",
    table 2-6.";
  GG2U = Association[{
      "00" -> 0,
      "10" -> 6/12 ,
      "11" -> 12/12,
      "20" -> 14/12,
      "21" -> 21/12,
      "22" -> 30/12,
      "30" -> 24/12,
      "31" -> 32/12,
      "40" -> 36/12}
    ];

  CasimirG2::usage = "CasimirG2[SL, SpLp] returns LS reduced matrix
    element of the configuration interaction term corresponding to the
    Casimir operator of G2.";
  CasimirG2[{SL_, SpLp_}] := (
    Ulabel = FindNKLSTerm[SL][[1]][[4]];
    If[SL==SpLp,
      β * GG2U[Ulabel],
      0
    ]
  )
```

```
GSO7W::usage = "GSO7W is an association whose keys are labels for the
   irreducible representations of group R7 and whose values are the
  eigenvalues of the corresponding Casimir operator.
Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\",
  table 2-7.";
GSO7W := Association[
  {
    "000" -> 0,
    "100" -> 3/5,
    "110" -> 5/5,
    "111" -> 6/5,
    "200" -> 7/5,
    "210" -> 9/5,
    "211" -> 10/5,
    "220" -> 12/5,
    "221" -> 13/5,
    "222" -> 15/5
  }
];

CasimirSO7::usage = "CasimirSO7[SL, SpLp] returns the LS reduced
  matrix element of the configuration interaction term corresponding
  to the Casimir operator of R7.";
CasimirSO7[{SL_, SpLp_}] := (
  Wlabel = FindNKLSTerm[SL][[1]][[3]];
  If[SL==SpLp,
    γ * GSO7W[Wlabel],
    0
  ]
)

ElectrostaticConfigInteraction::usage = "
  ElectrostaticConfigInteraction[{SL, SpLp}] returns the matrix
  element for configuration interaction as approximated by the
  Casimir operators of the groups R3, G2, and R7. SL and SpLp are
  strings that represent terms under LS coupling.";
ElectrostaticConfigInteraction[{SL_, SpLp_}] := Module[
  {S, L, val},
  {S, L} = FindSL[SL];
  val = (
    If[SL == SpLp,
      CasimirSO3[{SL, SL}] +
      CasimirSO7[{SL, SL}] +
       CasimirG2[{SL, SL}],
      0
    ]
    );
  ElectrostaticConfigInteraction[{S, L}] = val;
  Return[val];
  ]

(* #### Configuration-Interaction via Casimir Operators ###### *)
(* ################################################################ *)

(* ################################################################ *)
(* ###################### Block assembly ###################### *)

JJBlockMatrix::usage = "For given J, J' in the f^n configuration
  JJBlockMatrix[numE, J, J'] determines all the SL S'L' terms that
  may contribute to them and using those it provides the matrix
  elements <J, LS | H | J', LS'>. H having contributions from the
  following interactions: Coulomb, spin-orbit, spin-other-orbit,
  electrostatically-correlated-spin-orbit, spin-spin, three-body
  interactions, and crystal-field.";
Options[JJBlockMatrix] = {"Sparse"->True, "ChenDeltas"->False};
JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]]:= Module[
  {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
  SLterm, SpLpterm,
  MJ, MJp,
  subKron, matValue, eMatrix},
  (
    NKSLJMs  = AllowedNKSLJMforJTerms[numE, J];
    NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
    eMatrix =
      Table[
        (*Condition for a scalar matrix op*)
```

```
1991            SLterm    =  NKSLJM[[1]];
1992            SpLpterm  = NKSLJMp[[1]];
1993            MJ        =  NKSLJM[[3]];
1994            MJp       = NKSLJMp[[3]];
1995            subKron   =
1996              (
1997                KroneckerDelta[J, Jp] *
1998                KroneckerDelta[MJ, MJp]
1999              );
2000            matValue =
2001              If[subKron==0,
2002                0,
2003                  (
2004                    ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2005                    ElectrostaticConfigInteraction[{SLterm, SpLpterm}] +
2006                    SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2007                    MagneticInteractions[{numE, SLterm, SpLpterm, J}, "
     ChenDeltas" -> OptionValue["ChenDeltas"]] +
2008                    ThreeBodyTable[{numE, SLterm, SpLpterm}]
2009                  )
2010              ];
2011            matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp
     }];
2012            matValue,
2013          {NKSLJMp, NKSLJMps},
2014          {NKSLJM , NKSLJMs}
2015          ];
2016       If[OptionValue["Sparse"],
2017         eMatrix = SparseArray[eMatrix]
2018       ];
2019       Return[eMatrix]
2020     )
2021     ];
2022
2023     EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some
       point may be used to redefine states used in basis.";
2024     EnergyStates[numE_, J_]:= AllowedNKSLJMforJTerms[numE, J];
2025
2026     JJBlockMatrixFileName::usage = "JJBlockMatrixFileName[numE] gives the
        filename for the energy matrix table for an atom with numE f-
       electrons. The function admits an optional parameter \"
       FilenameAppendix\" which can be used to modify the filename.";
2027     Options[JJBlockMatrixFileName] = {"FilenameAppendix" -> ""}
2028     JJBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
2029       fileApp = OptionValue["FilenameAppendix"];
2030       fname = FileNameJoin[{moduleDir,
2031           "hams",
2032           StringJoin[{"f", ToString[numE], "_JJBlockMatrixTable", fileApp
        ,".m"}]}];
2033       Return[fname];
2034       );
2035
2036     TabulateJJBlockMatrixTable::usage = "TabulateJJBlockMatrixTable[numE,
        I] returns a list with three elements {JJBlockMatrixTable,
       EnergyStatesTable, AllowedM}. JJBlockMatrixTable is an association
       with keys equal to lists of the form {numE, J, Jp}.
       EnergyStatesTable is an association with keys equal to lists of the
        form {numE, J}. AllowedM is another association with keys equal to
        lists of the form {numE, J} and values equal to lists equal to the
        corresponding values of MJ. It's unnecessary (and it won't work in
        this implementation) to give numE > 7 given the equivalency
       between electron and hole configurations.";
2037     Options[TabulateJJBlockMatrixTable] = {"Sparse"->True, "ChenDeltas"->
       False};
2038     TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]]:= (
2039       JJBlockMatrixTable = <||>;
2040       totalIterations = Length[AllowedJ[numE]]^2;
2041       template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
2042       template2 = StringTemplate["`remtime` min remaining"];
2043       template4 = StringTemplate["Time elapsed = `runtime` min"];
2044       numiter   = 0;
2045       startTime = Now;
2046       If[$FrontEnd =!= Null,
2047         (
2048           temp = PrintTemporary[
2049             Dynamic[
```

```
            Grid[
              {
                {template1[<|"numiter"->numiter, "totaliter"->totalIterations|>]},
                {template2[<|"remtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime)/(Max[1,numiter])*(totalIterations-numiter), "min"]], 0.1]|>]},
                {template4[<|"runtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                {ProgressIndicator[numiter, {1, totalIterations}]}
              }
            ]
        ]
        ];
      )
    ];
    Do[
      (
        JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp, CFTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas" -> OptionValue["ChenDeltas"]];
        numiter += 1;
      ),
    {Jp, AllowedJ[numE]},
    {J, AllowedJ[numE]}
    ];
    If[$FrontEnd =!= Null,
      NotebookDelete[temp]
    ];
    Return[JJBlockMatrixTable];
  )

  TabulateManyJJBlockMatrixTables::usage = "
    TabulateManyJJBlockMatrixTables[{n1, n2, ...}] calculates the
    tables of matrix elements for the requested f^n_i configurations.
    The function does not return the matrices themselves. It instead
    returns an association whose keys are numE and whose values are the
     filenames where the output of TabulateJJBlockMatrixTables was
    saved to.The output consists of an association whose keys are of
    the form {n, J, Jp} and whose values are rectangular arrays given
    the values of <|LSJMJa|H|L'S'J'MJ'a'|>.";
  Options[TabulateManyJJBlockMatrixTables] = {"Overwrite"->False, "Sparse"->True, "ChenDeltas"->False, "FilenameAppendix"-> "", "Compressed" -> False};
  TabulateManyJJBlockMatrixTables[ns_, OptionsPattern[]]:= (
    overwrite = OptionValue["Overwrite"];
    fNames = <||>;
    fileApp = OptionValue["FilenameAppendix"];
    ExportFun = If[OptionValue["Compressed"], ExportMZip, Export];
    Do[
      (
        CFdataFilename = FileNameJoin[{moduleDir, "data", "CrystalFieldTable_f"<>ToString[numE]<>".zip"}];
        PrintTemporary["Importing CrystalFieldTable from ", CFdataFilename, " ..."];
        CrystalFieldTable = ImportMZip[CFdataFilename];

        PrintTemporary["#------- numE = ", numE, " -------#"];
        exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix" -> fileApp];
        fNames[numE] = exportFname;
        If[FileExistsQ[exportFname] && Not[overwrite],
          Continue[]
        ];
        JJBlockMatrixTable = TabulateJJBlockMatrixTable[numE, CrystalFieldTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas" -> OptionValue["ChenDeltas"]];
        If[FileExistsQ[exportFname]&&overwrite,
          DeleteFile[exportFname]
        ];
        ExportFun[exportFname, JJBlockMatrixTable];

        ClearAll[CrystalFieldTable];
      ),
    {numE, ns}
    ];
```

```
  Return[fNames];
  )

  HamMatrixAssembly::usage="HamMatrixAssembly[numE] returns the
    Hamiltonian matrix for the f^n_i configuration. The matrix is
    returned as a SparseArray. The function admits an optional
    parameter \"FilenameAppendix\" which can be used to modify the
    filename to which the resulting array is exported to. It also
    admits an optional parameter \"IncludeZeeman\" which can be used to
     include the Zeeman interaction;";
  Options[HamMatrixAssembly] = {"FilenameAppendix"->"","IncludeZeeman"
    ->False};
  HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
    {numE, ii, jj, howManyJs, Js, blockHam},
    (*########################################*)
    ImportFun = ImportMZip;
    (*########################################*)
    (*hole-particle equivalence enforcement*)
    numE = nf;
    allVars = {E0, E1, E2, E3, ζ, F0, F2, F4, F6, M0, M2, M4, T2, T2p,
      T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
      α, β, γ, B02, B04, B06, B12, B14, B16,
      B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16, S22,
      S24, S26, S34, S36, S44, S46, S56, S66, T11, T11p, T12, T14, T15,
     T16,
      T17, T18, T19, Bx, By, Bz};
    params0 = AssociationThread[allVars, allVars];
    If[nf > 7,
      (
        numE = 14 - nf;
        params = HoleElectronConjugation[params0];
      ),
      params = params0;
    ];
    (* Load symbolic expressions for LS,J,J' energy sub-matrices. *)
    emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
    OptionValue["FilenameAppendix"]];
    JJBlockMatrixTable = ImportFun[emFname];
    (*Patch together the entire matrix representation using J,J' blocks
    .*)
    PrintTemporary["Patching JJ blocks ..."];
    Js         = AllowedJ[numE];
    howManyJs = Length[Js];
    blockHam  = ConstantArray[0, {howManyJs, howManyJs}];
    Do[
      blockHam[[jj, ii]] = JJBlockMatrixTable[{numE, Js[[ii]], Js[[jj
    ]]}];,
    {ii, 1, howManyJs},
    {jj, 1, howManyJs}
    ];
    (* Once the block form is created flatten it *)
    blockHam = ArrayFlatten[blockHam];
    blockHam = ReplaceInSparseArray[blockHam, params];
    If[OptionValue["IncludeZeeman"],
      (
        PrintTemporary["Including Zeeman terms ..."];
        {magx, magy, magz} = MagDipoleMatrixAssembly[numE];
        blockHam += - TeslaToKayser * (Bx * magx + By * magy + Bz *
    magz);
      )
    ];
    Return[blockHam];
    ]

SimplerSymbolicHamMatrix::usage="SimplerSymbolicHamMatrix[numE,
    simplifier] is a simple addition to HamMatrixAssembly that applies
    a given simplification to the full hamiltonian. Simplifier is a
    list of replacement rules. If the option \"Export\" is set to True,
     then the function also exports the resulting sparse array to the
    ./hams/ folder. The option \"PrependToFilename\" can be used to
    append a string to the filename to which the function may exports
    to. The option \"Return\" can be used to choose whether the
    function returns the matrix or not. The option \"Overwrite\" can be
     used to overwrite the file if it already exists. The option \"
    IncludeZeeman\" can be used to toggle the inclusion of the Zeeman
    interaction with an external magnetic field.";
```

```
Options[SimplerSymbolicHamMatrix]={
  "Export"->True,
  "PrependToFilename"->"",
  "EorF"->"F",
  "Overwrite" -> False,
  "Return" -> True,
  "IncludeZeeman" -> False};
SimplerSymbolicHamMatrix[numE_Integer, simplifier_List, OptionsPattern[]]:=Module[
  {thisHam,eTofs,fname},
  (
    If[Not[ValueQ[ElectrostaticTable]],
      LoadElectrostatic[]
      ];
    If[Not[ValueQ[SOOandECSOTable]],
      LoadSOOandECSO[]
    ];
    If[Not[ValueQ[SpinOrbitTable]],
      LoadSpinOrbit[]
    ];
    If[Not[ValueQ[SpinSpinTable]],
      LoadSpinSpin[]
    ];
    If[Not[ValueQ[ThreeBodyTable]],
      LoadThreeBody[]
    ];

    fname=FileNameJoin[{moduleDir,"hams",OptionValue["PrependToFilename"]<>"SymbolicMatrix-f"<>ToString[numE]<>".m"}];
    If[FileExistsQ[fname] && Not[OptionValue["Overwrite"]],
      (
        If[OptionValue["Return"],
          (
            Print["File ",fname," already exists, and option \"Overwrite\" is set to False, loading file ..."];
            thisHam = Import[fname];
            Return[thisHam];
          ),
          (
            Print["File ",fname," already exists, skipping ..."];
          Return[Null];
          )
        ]
      )
    ];

    thisHam=HamMatrixAssembly[numE, "IncludeZeeman"->OptionValue["IncludeZeeman"]];
    thisHam=ReplaceInSparseArray[thisHam, simplifier];
    If[OptionValue["Export"],
    (
      Print["Exporting to file ",fname];
      Export[fname,thisHam]
    )
    ];
    If[OptionValue["Return"],
      Return[thisHam],
      Return[Null]
    ];
  )
]

  (* ###################### Block assembly ###################### *)
  (* ########################################################### *)

  (* ########################################################### *)
  (* ##################### Optical Operators ##################### *)

  magOp = <||>;

  JJBlockMagDip::usage="JJBlockMagDip[numE, J, Jp] returns the LSJ-reduced matrix element of the magnetic dipole operator between the states with given J and Jp. The option \"Sparse\" can be used to return a sparse matrix. The default is to return a sparse matrix.
  See eqn 15.7 in TASS.
  Here it is provided in atomic units in which the Bohr magneton is
```

```
  1/2.
\[Mu] = -(1/2) (L + gs S)
We are using the Racah convention for the reduced matrix elements in
  the Wigner-Eckart theorem. See TASS eqn 11.15.
";
Options[JJBlockMagDip]={"Sparse"->True};
JJBlockMagDip[numE_, braJ_, ketJ_, OptionsPattern[]]:=Module[
  {braSLJs,ketSLJs,
  braSLJ,ketSLJ,
  braSL,ketSL,
  braS, braL,
  braMJ, ketMJ,
  matValue,magMatrix},(
  braSLJs  = AllowedNKSLJMforJTerms[numE,braJ];
  ketSLJs  = AllowedNKSLJMforJTerms[numE,ketJ];
  magMatrix  = Table[
    braSL     = braSLJ[[1]];
    ketSL     = ketSLJ[[1]];
    {braS, braL}   = FindSL[braSL];
    {ketS, ketL}   = FindSL[ketSL];
    braMJ       = braSLJ[[3]];
    ketMJ       = ketSLJ[[3]];
    summand1    = If[Or[braJ  != ketJ,
                        braSL != ketSL],
      0,
      Sqrt[braJ(braJ+1)TPO[braJ]]
    ];
    (* looking at the string includes checking L=L' S=S' \alpha=\
  alpha'*)
    summand2 = If[braSL!= ketSL,
      0,
      (gs-1) *
        Phaser[braS+braL+ketJ+1] *
          Sqrt[TPO[braJ]*TPO[ketJ]] *
            SixJay[{braJ,1,ketJ},{braS,braL,braS}] *
              Sqrt[braS(braS+1)TPO[braS]]
    ];
    matValue = summand1 + summand2;
    (* We are using the Racah convention for red matrix elements in
  Wigner-Eckart *)
    threejays = (ThreeJay[{braJ, -braMJ}, {1, #}, {ketJ, ketMJ}] &) /
  @ {-1,0,1};
    threejays *= Phaser[braJ-braMJ];
    matValue  = - 1/2 * threejays * matValue;
    matValue,
  {braSLJ, braSLJs},
  {ketSLJ, ketSLJs}
  ];
  If[OptionValue["Sparse"],
    magMatrix= SparseArray[magMatrix]
  ];
  Return[magMatrix])
];

Options[TabulateJJBlockMagDipTable]={"Sparse"->True};
TabulateJJBlockMagDipTable[numE_,OptionsPattern[]]:=(
  JJBlockMagDipTable=<||>;
  Js=AllowedJ[numE];
  Do[
    (
      JJBlockMagDipTable[{numE,braJ,ketJ}]=
        JJBlockMagDip[numE,braJ,ketJ,"Sparse"->OptionValue["Sparse"]]
    ),
  {braJ, Js},
  {ketJ, Js}
  ];
  Return[JJBlockMagDipTable]
);

TabulateManyJJBlockMagDipTables::usage = "
  TabulateManyJJBlockMagDipTables[{n1, n2, ...}] calculates the
  tables of matrix elements for the requested f^n_i configurations.
  The function does not return the matrices themselves. It instead
  returns an association whose keys are numE and whose values are the
   filenames where the output of TabulateManyJJBlockMagDipTables was
  saved to. The output consists of an association whose keys are of
```

```
    the form {n, J, Jp} and whose values are rectangular arrays given
    the values of <|LSJMJa|H_dip|L'S'J'MJ'a'|>.";
2292 Options[TabulateManyJJBlockMagDipTables]={"FilenameAppendix"->"","
    Overwrite"->False,"Compressed"->True};
2293 TabulateManyJJBlockMagDipTables[ns_,OptionsPattern[]]:=(
2294   fnames=<||>;
2295   Do[
2296   (
2297     ExportFun=If[OptionValue["Compressed"],ExportMZip,Export];
2298     PrintTemporary["#------- numE = ",numE," -------#"];
2299     appendTo     = (OptionValue["FilenameAppendix"]<>"-magDip");
2300     exportFname  = JJBlockMatrixFileName[numE,"FilenameAppendix"->
    appendTo];
2301     fnames[numE] = exportFname;
2302     If[FileExistsQ[exportFname]&&Not[OptionValue["Overwrite"]],
2303       Continue[]
2304     ];
2305     JJBlockMatrixTable = TabulateJJBlockMagDipTable[numE];
2306     If[FileExistsQ[exportFname]&&OptionValue["Overwrite"],
2307       DeleteFile[exportFname]
2308     ];
2309     ExportFun[exportFname,  JJBlockMatrixTable];
2310   ),
2311   {numE,ns}
2312   ];
2313   Return[fnames];
2314 )
2315
2316 MagDipoleMatrixAssembly::usage="MagDipoleMatrixAssembly[numE] returns
     the matrix representation of the operator - 1/2 (L + gs S) in the
    f^numE configuration. The function returns a list with three
    elements corresponding to the x,y,z components of this operator.
    The option \"FilenameAppendix\" can be used to append a string to
    the filename from which the function imports from in order to patch
     together the array. For numE beyond 7 the function returns the
    same as for the complementary configuration.";
2317 Options[MagDipoleMatrixAssembly]={"FilenameAppendix"->""};
2318 MagDipoleMatrixAssembly[nf_,OptionsPattern[]]:=Module[
2319   {ImportFun, numE, appendTo, emFname, JJBlockMagDipTable, Js,
    howManyJs, blockOp, rowIdx, colIdx},
2320   (
2321   ImportFun = ImportMZip;
2322   numE      = nf;
2323   numH      = 14 - numE;
2324   numE      = Min[numE,numH];
2325
2326   appendTo  = (OptionValue["FilenameAppendix"]<>"-magDip");
2327   emFname   = JJBlockMatrixFileName[numE,"FilenameAppendix"->appendTo
    ];
2328   JJBlockMagDipTable = ImportFun[emFname];
2329
2330   Js        = AllowedJ[numE];
2331   howManyJs = Length[Js];
2332   blockOp   = ConstantArray[0,{howManyJs,howManyJs}];
2333   Do[
2334     blockOp[[rowIdx,colIdx]] = JJBlockMagDipTable[{numE,Js[[rowIdx]],
    Js[[colIdx]]}],
2335     {rowIdx,1,howManyJs},
2336     {colIdx,1,howManyJs}
2337   ];
2338   blockOp = ArrayFlatten[blockOp];
2339   opMinus = blockOp[[;; , ;; , 1]];
2340   opZero =  blockOp[[;; , ;; , 2]];
2341   opPlus =  blockOp[[;; , ;; , 3]];
2342   opX =   (opMinus - opPlus)/Sqrt[2];
2343   opY = I (opPlus + opMinus)/Sqrt[2];
2344   opZ = opZero;
2345   Return[{opX, opY, opZ}];
2346 )
2347 ];
2348
2349 MagDipLineStrength::usage="MagDipLineStrength[theEigensys, numE]
    takes the eigensystem of an ion and the number numE of f-electrons
    that correspond to it and it calculates the line strength array
    Stot.
2350 The option \"Units\" can be set to either \"SI\" (so that the units
```

```
    of the returned array are A/m^2) or to \"Hartree\".
2351 The option \"States\" can be used to limit the states for which the
    line strength is calculated. The default, All, calculates the line
    strength for all states. A second option for this is to provide an
    index labelling a specific state, in which case only the line
    strengths between that state and all the others are computed.
2352 The returned array should be interpreted in the eigenbasis of the
    Hamiltonian. As such the element Stot[[i,i]] corresponds to the
    line strength states |i> and |j>.";
2353 Options[MagDipLineStrength]={"Reload MagOp" -> False, "Units"->"SI",
    "States" -> All};
2354 MagDipLineStrength[theEigensys_List, numE0_Integer, OptionsPattern
    []]:=Module[
2355   {allEigenvecs, Sx, Sy, Sz, Stot ,factor},
2356 (
2357   numE = Min[14-numE0, numE0];
2358   (*If not loaded then load it, *)
2359   If[Or[
2360       Not[MemberQ[Keys[magOp], numE]],
2361       OptionValue["Reload MagOp"]],
2362   (
2363     magOp[numE] = ReplaceInSparseArray[#,{gs->2}]& /@
    MagDipoleMatrixAssembly[numE];
2364   )
2365   ];
2366   allEigenvecs = Transpose[Last/@theEigensys];
2367   Which[OptionValue["States"] === All,
2368     (
2369       {Sx,Sy,Sz}   = (ConjugateTranspose[allEigenvecs].#.allEigenvecs
    ) & /@ magOp[numE];
2370       Stot         = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
2371     ),
2372     IntegerQ[OptionValue["States"]],
2373     (
2374       singleState  = theEigensys[[OptionValue["States"],2]];
2375       {Sx,Sy,Sz}   = (ConjugateTranspose[allEigenvecs].#.singleState)
     & /@ magOp[numE];
2376       Stot         = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
2377     )
2378   ];
2379   Which[
2380     OptionValue["Units"] == "SI",
2381       Return[4 \[Mu]B^2 * Stot],
2382     OptionValue["Units"] == "Hartree",
2383       Return[Stot],
2384     True,
2385     (
2386       Print["Invalid option for \"Units\". Options are \"SI\" and \"
    Hartree\"."];
2387       Abort[];
2388     )
2389   ];
2390 )
2391 ]
2392
2393 MagDipoleRates::usage="MagDipoleRates[eigenSys, numE] calculates the
    magnetic dipole transition rate array for the provided eigensystem.
     The option \"Units\" can be set to \"SI\" or to \"Hartree\". If
    the option \"Natural Radiative Lifetimes\" is set to true then the
    reciprocal of the rate is returned instead. The energy unit assumed
     in eigenSys is kayser. The returned array should be interpreted in
     the eigenbasis of the Hamiltonian. As such the element AMD[[i,i]]
    corresponds to the transition rate (or the radiative lifetime,
    depending on options) between eigenstates |i> and |j>.";
2394 Options[MagDipoleRates]={"Units"->"SI", "Lifetime"->False};
2395 MagDipoleRates[eigenSys_List, numE0_Integer,OptionsPattern[]]:=Module
    [
2396   {AMD,gKramers,Stot,eigenEnergies,transitionWaveLengthsInMeters},(
2397   numE          = Min[14-numE0, numE0];
2398   gKramers      = If[OddQ[numE],2,1];
2399   Stot          = MagDipLineStrength[eigenSys, numE, "Units"->
    OptionValue["Units"]];
2400   eigenEnergies = Chop[First/@eigenSys];
2401   energyDiffs   = Outer[Subtract,eigenEnergies,eigenEnergies];
2402   energyDiffs   = ReplaceDiagonal[energyDiffs, Indeterminate];
2403   (* Energies assumed in pseudo-energy unit kayser.*)
```

```
    transitionWaveLengthsInMeters = 0.01/energyDiffs;

    unitFactor    = Which[
    OptionValue["Units"]=="Hartree",
    (
      (* The bohrRadius factor in SI neede to convert the wavelengths
     which are assumed in m*)
      16 \[Pi]^3 (\[Mu]0Hartree /(3 hPlanckFine)) * bohrRadius^3
    ),
    OptionValue["Units"]=="SI",
    (
      16 \[Pi]^3 \[Mu]0/(3 hPlanck)
    ),
    True,
    (
      Print["Invalid option for \"Units\". Options are \"SI\" and \"
     Hartree\"."];
      Abort[];
    )
    ];
    AMD = unitFactor/gKramers (1/transitionWaveLengthsInMeters^3)*Stot;
    Which[OptionValue["Lifetime"],
      Return[1/AMD],
      True,
      Return[AMD]
    ]
    )
  ]

  GroundStateOscillatorStrength::usage="GroundStateOscillatorStrength[
    eigenSys, numE] calculates the oscillator strength between the
    ground state and the excited states as given by eigenSys. The
    energy unit assumed in eigenSys is kayser. The returned array
    should be interpreted in the eigenbasis of the Hamiltonian. As such
     the element fMDGS[[i]] corresponds to the oscillator strength
    between ground state and eigenstate |i>.";
  GroundStateOscillatorStrength[eigenSys_, numE_]:=Module[
  {eigenEnergies, SMDGS, GSEnergy, gKramers, energyDiffs,
    transitionWaveLengthsInMeters, factor},
  (
    eigenEnergies     = First/@eigenSys;
    SMDGS             = MagDipLineStrength[eigenSys,numE, "Units"->"SI",
     "States"->1];
    GSEnergy          = eigenSys[[1,1]];
    gKramers          = If[OddQ[numE],2,1];
    energyDiffs       = eigenEnergies-GSEnergy;
    energyDiffs[[1]] = Indeterminate;
    transitionWaveLengthsInMeters = 0.01/energyDiffs;
    factor = (8\[Pi]^2 me)       /(3 hPlanck eCharge^2 cLight);
    fMDGS=unitFactor/gKramers/transitionWaveLengthsInMeters*SMDGS;
    Return[fMDGS];
  )
  ]

  (* ##################### Optical Operators ##################### *)
  (* ############################################################## *)

  (* ############################################################## *)
  (* ##################### Printers and Labels ##################### *)

  PrintL::usage = "PrintL[L] give the string representation of a given
    angular momentum.";
  PrintL[L_] := If[StringQ[L], L, StringTake[specAlphabet, {L + 1}]]

  FindSL::usage = "FindSL[LS] gives the spin and orbital angular
    momentum that corresponds to the provided string LS.";
  FindSL[SL_]:= (
    FindSL[SL] =
    If[StringQ[SL],
      {
        (ToExpression[StringTake[SL, 1]]-1)/2,
        StringPosition[specAlphabet, StringTake[SL, {2}]][[1, 1]]-1
      },
      SL
    ]
  )
```

```
2468
2469  PrintSLJ::usage = "Given a list with three elements {S, L, J} this
        function returns a symbol where the spin multiplicity is presented
        as a superscript, the orbital angular momentum as its corresponding
         spectroscopic letter, and J as a subscript. Function does not
        check to see if the given J is compatible with the given S and L.";
2470  PrintSLJ[SLJ_] :=
2471    RowBox[{SuperscriptBox[" ", 2 SLJ[[1]] + 1],
2472      SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]}] // DisplayForm;
2473
2474  PrintSLJM::usage = "Given a list with four elements {S, L, J, MJ}
        this function returns a symbol where the spin multiplicity is
        presented as a superscript, the orbital angular momentum as its
        corresponding spectroscopic letter, and {J, MJ} as a subscript. No
        attempt is made to guarantee that the given input is consistent.";
2475  PrintSLJM[SLJM_] :=
2476    RowBox[{SuperscriptBox[" ", 2 SLJM[[1]] + 1],
2477      SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]}] //
2478    DisplayForm;
2479
2480  (* ################### Printers and Labels ################### *)
2481  (* ########################################################### *)
2482
2483  (* ########################################################### *)
2484  (* ###################### Term management ###################### *)
2485
2486  AllowedSLTerms::usage = "AllowedSLTerms[numE] returns a list with the
         allowed terms in the f^numE configuration, the terms are given as
        lists in the format {S, L}. This list may have redundancies which
        are compatible with the degeneracies that might correspond to the
        given case.";
2487  AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE,
        14-numE]]]
2488
2489  AllowedNKSLTerms::usage = "AllowedNKSLTerms[numE] returns a list with
         the allowed terms in the f^numE configuration, the terms are given
         as strings in spectroscopic notation. The integers in the last
        positions are used to distinguish cases with degeneracy.";
2490  AllowedNKSLTerms[numE_] := (Map[First, CFPTerms[Min[numE, 14-numE]]])
2491  AllowedNKSLTerms[0] = {"1S"};
2492  AllowedNKSLTerms[14] = {"1S"};
2493
2494  MaxJ::usage = "MaxJ[numE] gives the maximum J = S+L that corresponds
        to the configuration f^numE.";
2495  MaxJ[numE_] := Max[Map[Total, AllowedSLTerms[Min[numE, 14-numE]]]]
2496
2497  MinJ::usage = "MinJ[numE] gives the minimum J = S+L that corresponds
        to the configuration f^numE.";
2498  MinJ[numE_] := Min[Map[Abs[Part[#, 1] - Part[#, 2]] &, AllowedSLTerms
        [Min[numE, 14-numE]]]]
2499
2500  AllowedSLJTerms::usage = "AllowedSLJTerms[numE] returns a list with
        the allowed {S, L, J} terms in the f^n configuration, the terms are
         given as lists in the format {S, L, J}. This list may have
        repeated elements which account for possible degeneracies of the
        related term.";
2501  AllowedSLJTerms[numE_] :=
2502    Module[{idx1, allowedSL, allowedSLJ},
2503      allowedSL = AllowedSLTerms[numE];
2504      allowedSLJ = {};
2505      For[
2506        idx1 = 1,
2507        idx1 <= Length[allowedSL],
2508        termSL = allowedSL[[idx1]];
2509        termsSLJ =
2510          Table[
2511            {termSL[[1]], termSL[[2]], J},
2512          {J,Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2513          ];
2514        allowedSLJ = Join[allowedSLJ, termsSLJ];
2515        idx1++
2516      ];
2517      SortBy[allowedSLJ, Last]
2518    ]
2519
2520  AllowedNKSLJTerms::usage = "AllowedNKSLJTerms[numE] returns a list
```

```
    with the allowed {SL, J} terms in the f^n configuration, the terms
    are given as lists in the format {SL, J} where SL is a string in
    spectroscopic notation.";
AllowedNKSLJTerms[numE_] :=
  Module[{allowedSL, allowedNKSL, allowedSLJ, nn},
    allowedNKSL = AllowedNKSLTerms[numE];
    allowedSL = AllowedSLTerms[numE];
    allowedSLJ = {};
    For[
      nn = 1,
      nn <= Length[allowedSL],
      (
        termSL = allowedSL[[nn]];
        termNKSL = allowedNKSL[[nn]];
        termsSLJ =
          Table[{termNKSL, J},
          {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
          ];
        allowedSLJ = Join[allowedSLJ, termsSLJ];
        nn++
      )
    ];
    SortBy[allowedSLJ, Last]
  ]

AllowedNKSLforJTerms::usage = "AllowedNKSLforJTerms[numE, J] gives
    the terms that correspond to the given total angular momentum J in
    the f^n configuration. The result is a list whose elements are
    lists of length 2, the first element being the SL term in
    spectroscopic notation, and the second element being J.";
AllowedNKSLforJTerms[numE_, J_] := Module[
    {allowedSL, allowedNKSL, allowedSLJ, nn, termSL, termNKSL,
  termsSLJ},
    allowedNKSL = AllowedNKSLTerms[numE];
    allowedSL = AllowedSLTerms[numE];
    allowedSLJ = {};
    For[
      nn = 1,
      nn <= Length[allowedSL],
      (
        termSL = allowedSL[[nn]];
        termNKSL = allowedNKSL[[nn]];
        termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[
  termSL],
          {{termNKSL, J}},
          {}
          ];
        allowedSLJ = Join[allowedSLJ, termsSLJ];
        nn++
      )
    ];
    Return[allowedSLJ]
  ];

AllowedSLJMTerms::usage = "AllowedSLJMTerms[numE] returns a list with
    all the states that correspond to the configuration f^n. A list is
    returned whose elements are lists of the form {S, L, J, MJ}.";
AllowedSLJMTerms[numE_] := Module[
    {allowedSLJ, allowedSLJM, termSLJ, termsSLJM, nn},
    allowedSLJ = AllowedSLJTerms[numE];
    allowedSLJM = {};
    For[
      nn = 1,
      nn <= Length[allowedSLJ],
      nn++,
      (
        termSLJ = allowedSLJ[[nn]];
        termsSLJM =
          Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
          {M, - termSLJ[[3]], termSLJ[[3]]}
          ];
        allowedSLJM = Join[allowedSLJM, termsSLJM];
      )
    ];
    Return[SortBy[allowedSLJM, Last]];
    ]
```

```
2586
2587  AllowedNKSLJMforJMTerms::usage = "AllowedNKSLJMforJMTerms[numE, J, MJ
       ] returns a list with all the terms that contain states of the f^n
       configuration that have a total angular momentum J, and a
       projection along the z-axis MJ. The returned list has elements of
       the form {SL (string in spectroscopic notation), J, MJ}.";
2588  AllowedNKSLJMforJMTerms[numE_, J_, MJ_] :=
2589    Module[{allowedSL, allowedNKSL, allowedSLJM, nn},
2590      allowedNKSL = AllowedNKSLTerms[numE];
2591      allowedSL   = AllowedSLTerms[numE];
2592      allowedSLJM = {};
2593      For[
2594        nn = 1,
2595        nn <= Length[allowedSL],
2596        termSL = allowedSL[[nn]];
2597        termNKSL = allowedNKSL[[nn]];
2598        termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
2599                       <= J
2600                       <= Total[termSL]
2601                       && (Abs[MJ] <= J)
2602                       ),
2603                       {{termNKSL, J, MJ}},
2604                       {}];
2605        allowedSLJM = Join[allowedSLJM, termsSLJ];
2606        nn++
2607      ];
2608      Return[allowedSLJM];
2609    ]
2610
2611  AllowedNKSLJMforJTerms::usage = "AllowedNKSLJMforJTerms[numE, J]
       returns a list with all the states that have a total angular
       momentum J. The returned list has elements of the form {{SL (string
        in spectroscopic notation), J}, MJ}, and if the option \"Flat\" is
        set to True then the returned list has element of the form {SL (
       string in spectroscopic notation), J, MJ}.";
2612  AllowedNKSLJMforJTerms[numE_, J_] :=
2613  Module[{MJs, labelsAndMomenta, termsWithJ},
2614    (
2615    MJs = AllowedMforJ[J];
2616    (* Pair LS labels and their {S,L} momenta *)
2617    labelsAndMomenta = ({#, FindSL[#]}) & /@ AllowedNKSLTerms[numE];
2618    (* A given term will contain J if |L-S|<=J<=L+S *)
2619    ContainsJ[{SL_String, {S_, L_}}] := (Abs[S - L] <= J <= (S + L));
2620    (* Keep just the terms that satisfy this condition *)
2621    termsWithJ = Select[labelsAndMomenta, ContainsJ];
2622    (* We don't want to keep the {S,L} *)
2623    termsWithJ = {#[[1]], J} & /@ termsWithJ;
2624    (* This is just a quick way of including up all the MJ values *)
2625    Return[Flatten /@ Tuples[{termsWithJ, MJs}]]
2626    )
2627    ]
2628
2629  AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J,
       1].";
2630  AllowedMforJ[J_] := Range[-J, J, 1];
2631
2632  AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta J
        that appear in the f^numE configuration.";
2633  AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}];
2634
2635  Seniority::usage="Seniority[LS] returns the seniority of the given
       term."
2636  Seniority[LS_] := FindNKLSTerm[LS][[1, 2]]
2637
2638  FindNKLSTerm::usage = "Given the string LS FindNKLSTerm[SL] returns
       all the terms that are compatible with it. This is only for f^n
       configurations. The provided terms might belong to more than one
       configuration. The function returns a list with elements of the
       form {LS, seniority, W, U}.";
2639  FindNKLSTerm[SL_] := Module[
2640    {NKterms, n},
2641    n = 7;
2642    NKterms = {{}};
2643    Map[
2644      If[! StringFreeQ[First[#], SL],
2645        If[ToExpression[Part[#, 2]] <= n,
```

```
          NKterms = Join[NKterms, {#}, 1]
        ]
      ] &,
    fnTermLabels
    ];
    NKterms = DeleteCases[NKterms, {}];
    NKterms]

ParseTermLabels::usage="ParseTermLabels[] parses the labels for the
  terms in the f^n configurations based on the labels for the f6 and
  f7 configurations. The function returns a list whose elements are
  of the form {LS, seniority, W, U}.";
Options[ParseTermLabels] = {"Export" -> True};
ParseTermLabels[OptionsPattern[]] := Module[
  {labelsTextData, fNtextLabels, nielsonKosterLabels, seniorities,
  RacahW, RacahU},
(
  labelsTextData = FileNameJoin[{moduleDir, "data", "
  NielsonKosterLabels_f6_f7.txt"}];
  fNtextLabels   = Import[labelsTextData];
  nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
  termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
  seniorities = Map[ToExpression[Part[# , {2}]] &,
  nielsonKosterLabels];
  racahW =
    Map[
      StringTake[
        Flatten[StringCases[Part[# , {3}],
          "(" ~~ DigitCharacter ~~ DigitCharacter ~~ DigitCharacter
  ~~ ")"]],
        {2, 4}
      ] &,
    nielsonKosterLabels];
  racahU =
    Map[
      StringTake[
        Flatten[StringCases[Part[# , {3}],
          "(" ~~ DigitCharacter ~~ DigitCharacter ~~ ")"]],
        {2, 3}
      ] &,
    nielsonKosterLabels];
  fnTermLabels = Join[termLabels, seniorities, racahW, racahU, 2];
  fnTermLabels = Sort[fnTermLabels];
  If[OptionValue["Export"],
    (
      broadFname = FileNameJoin[{moduleDir,"data","fnTerms.m"}];
      Export[broadFname, fnTermLabels];
    )
  ];
  Return[fnTermLabels];
)
]

(* ##################### Term management ##################### *)
(* ############################################################ *)

LoadParameters::usage="LoadParameters[ln] takes a string with the
  symbol the element of a trivalent lanthanide ion and returns model
  parameters for it. It is based on the data for LaF3. If the option
  \"Free Ion\" is set to True then the function sets all crystal
  field parameters to zero. Through the option \"gs\" it allows
  modyfing the electronic gyromagnetic ratio. For completeness this
  function also computes the E parameters using the F parameters
  quoted on Carnall.";
Options[LoadParameters] = {
    "Source"->"Carnall",
    "Free Ion"->False,
    "gs"->2.002319304386
    };
LoadParameters[Ln_String, OptionsPattern[]]:=
  Module[{source, params},
  (
    source = OptionValue["Source"];
    params = Which[source=="Carnall",
             (Association[Carnall["data"][Ln]])
             ];
```

```
2708      (*If a free ion then all the parameters from the crystal field
          are set to zero*)
2709      If[OptionValue["Free Ion"],
2710        Do[params[cfSymbol] = 0,
2711        {cfSymbol, cfSymbols}
2712        ]
2713      ];
2714      params[F0] = 0;
2715      params[M2] = 0.56 * params[M0]; (* See Carnall 1989, Table I,
          caption, probably fixed based on HF values*)
2716      params[M4] = 0.31 * params[M0]; (* See Carnall 1989, Table I,
          caption, probably fixed based on HF values*)
2717      params[P0] = 0;
2718      params[P4] = 0.5 * params[P2];  (* See Carnall 1989, Table I,
          caption, probably fixed based on HF values*)
2719      params[P6] = 0.1 * params[P2];  (* See Carnall 1989, Table I,
          caption, probably fixed based on HF values*)
2720      params[gs] = OptionValue["gs"];
2721      {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[F0
          ], params[F2], params[F4], params[F6]];
2722      params[E0] = 0;
2723      Return[params];
2724    )
2725  ];
2726
2727  HoleElectronConjugation::usage = "HoleElectronConjugation[params]
       takes the parameters (as an association) that define a
       configuration and converts them so that they may be interpreted as
       corresponding to a complentary hole configuration. Some of this can
        be simply done by changing the sign of the model parameters. In
       the case of the effective three body interaction the relationship
       is more complex and is controlled by the value of the isE variable.
       ";
2728  HoleElectronConjugation[params_] :=
2729    Module[{newparams = params},
2730      (
2731        flipSignsOf = {ζ, T2, T3, T4, T6, T7, T8};
2732        flipSignsOf = Join[flipSignsOf, cfSymbols];
2733        flipped =
2734          Table[(flipper -> - newparams[flipper]),
2735          {flipper, flipSignsOf}
2736          ];
2737        nonflipped =
2738          Table[(flipper -> newparams[flipper]),
2739          {flipper, Complement[Keys[newparams], flipSignsOf]}
2740          ];
2741        flippedParams = Association[Join[nonflipped, flipped]];
2742        flippedParams = Select[flippedParams, FreeQ[#, Missing]&];
2743        Return[flippedParams];
2744      )
2745    ]
2746
2747  IonSolver::usage="IonSolver[numE, params, host] puts together (or
       retrieves from disk) the symbolic Hamiltonian for the f^numE
       configuration and solves it for the given params.
2748  params is an Association with keys equal to parameter symbols and
       values their numerical values. The function will replace the
       symbols in the symbolic Hamiltonian with their numerical values and
        then diagonalize the resulting matrix. Any parameter that is not
       defined in the params Association is assumed to be zero.
2749  host is an optional string that may be used to prepend the filename
       of the symbolic Hamiltonian that is saved to disk. The default is \
       "Ln\".
2750  The function returns the eigensystem as a list of lists where in each
        list the first element is the energy and the second element the
       corresponding eigenvector.
2751  Tha ordered basis in which this eigenvector is to be interpreted is
       the one corresponding to BasisLSJMJ[numE].
2752  The function admits the following options:
2753  \"Include Spin-Spin\" (bool) : If True then the spin-spin interaction
        is included as a contribution to the m_k operators. The default is
        True.
2754  \"Overwrite Hamiltonian\" (bool) : If True then the function will
       overwrite the symbolic Hamiltonian that is saved to disk to
       expedite calculations. The default is False. The symbolic
       Hamiltonian is saved to disk to the ./hams/ folder preceded by the
```

```
    string host.
2755  \"Zeroes\" (list) : A list with symbols assumed to be zero.
2756  ";
2757  Options[IonSolver] = {"Include Spin-Spin" -> True,
2758    "Overwrite Hamiltonian" -> False,
2759    "Zeroes" -> {}};
2760  IonSolver[numE_Integer, params0_Association, host_String:"Ln",
      OptionsPattern[]] := Module[
2761    {ln, simplifier, simpleHam, numHam, eigensys,
2762    startTime, endTime, diagonalTime, params=params0, zeroSymbols},
2763  (
2764    ln = StringSplit["Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb"][[numE]];
2765
2766    (* This could be done when replacing values, but this produces
      smaller saved arrays. *)
2767    simplifier = (#-> 0) & /@ OptionValue["Zeroes"];
2768    simpleHam = SimplerSymbolicHamMatrix[numE,
2769      simplifier,
2770      "PrependToFilename" -> host,
2771      "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
2772    ];
2773
2774    (* Note that we don't have to flip signs of parameters for fn
      beyond f7 since the matrix produced
2775    by SimplerSymbolicHamMatrix has already accounted for this. *)
2776
2777    (* Everything that is not given is set to zero *)
2778    params = ParamPad[params, "Print" -> True];
2779    PrintFun[params];
2780
2781    (* Enforce the override to the spin-spin contribution to the
      magnetic interactions *)
2782    params[\[Sigma]SS] = If[OptionValue["Include Spin-Spin"], 1, 0];
2783
2784    (* Create the numeric hamiltonian *)
2785    numHam = ReplaceInSparseArray[simpleHam, params];
2786    Clear[simpleHam];
2787
2788    (* Eigensolver *)
2789    PrintFun["> Diagonalizing the numerical Hamiltonian ..."];
2790    startTime = Now;
2791    eigensys  = Eigensystem[numHam];
2792    endTime   = Now;
2793    diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
2794    PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
2795    eigensys = Chop[eigensys];
2796    eigensys = Transpose[eigensys];
2797
2798    (* Shift the baseline energy *)
2799    eigensys = ShiftedLevels[eigensys];
2800    (* Sort according to energy *)
2801    eigensys = SortBy[eigensys, First];
2802    Return[eigensys];
2803    )
2804  ]
2805
2806
2807  ShiftedLevels::usage = "
2808  ShiftedLevels[originalLevels] takes a list of levels of the form
2809  {{energy_1, coeff_vector_1},
2810  {energy_2, coeff_vector_2},
2811  ...}}
2812  and returns the same input except that now to every energy the
      minimum of all of them has been subtracted.";
2813  ShiftedLevels[originalLevels_] :=
2814    Module[{groundEnergy, shifted},
2815      groundEnergy = Sort[originalLevels][[1,1]];
2816      shifted      = Map[{#[[1]] - groundEnergy, #[[2]]} &,
      originalLevels];
2817      Return[shifted];
2818    ]
2819
2820  (* ############################################################ *)
2821  (* ################### Eigensystem analysis ################### *)
2822
2823  PrettySaundersSLJmJ::usage = "PrettySaundersSLJmJ[{SL, J, mJ}]
```

```
    produces a human-redeable symbol for the given basis vector {SL, J,
     mJ}."
Options[PrettySaundersSLJmJ] = {"Representation" -> "Ket"};
PrettySaundersSLJmJ[{SL_, J_, mJ_}, OptionsPattern[]] := (If[
  StringQ[SL],
  ({S, L} = FindSL[SL];
    L = StringTake[SL, {2, -1}];
    ),
  {S, L} = SL];
  pretty = RowBox[{AdjustmentBox[Style[2*S + 1, Smaller],
      BoxBaselineShift -> -1, BoxMargins -> 0],
      AdjustmentBox[PrintL[L], BoxMargins -> -0.2],
      AdjustmentBox[
      Style[{InputForm[J], mJ}, Small, FontTracking -> "Narrow"],
      BoxBaselineShift -> 1,
      BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]}];
  pretty = DisplayForm[pretty];
  If[OptionValue["Representation"] == "Ket",
    pretty = Ket[pretty]
  ];
  Return[pretty]
  )

BasisVecInRusselSaunders::usage = "BasisVecInRusselSaunders[basisVec]
   takes a basis vector in the format {LSstring, Jval, mJval} and
  returns a human-readable symbol for the corresponding Russel-
  Saunders term."
BasisVecInRusselSaunders[basisVec_] := (
  {LSstring, Jval, mJval} = basisVec;
  Ket[PrettySaundersSLJmJ[basisVec]]
  )

LSJMJTemplate =
  StringTemplate[
  "\!\(\*TemplateBox[{\nRowBox[{\"‘LS‘\", \",\", \nRowBox[{\"J\", \
\"=\", \"‘J‘\"}], \",\", \nRowBox[{\"mJ\", \"=\", \"‘mJ‘\"}]}]},\n\
\"Ket\"]\)"];

BasisVecInLSJMJ::usage = "BasisVecInLSJMJ[basisVec] takes a basis
  vector in the format {{{LSstring, Jval}, mJval}, nucSpin} and
  returns a human-readable symbol for the corresponding LSJMJ term in
   the form |LS, J=..., mJ=...>."
BasisVecInLSJMJ[basisVec_] := (
  {LSstring, Jval, mJval} = basisVec;
  LSJMJTemplate[<|
    "LS" -> LSstring,
    "J" -> ToString[Jval, InputForm],
    "mJ" -> ToString[mJval, InputForm]|>]
  );

ParseStates::usage = "ParseStates[states, basis] takes a list of
  eigenstates in terms of their coefficients in the given basis and
  returns a list of the same states in terms of their energy, LSJMJ
  symbol, J, mJ, S, L, LSJ symbol, and LS symbol. The LS symbol
  returned corresponds to the term with the largest coefficient in
  the given basis.";
ParseStates[states_, basis_, OptionsPattern[]] := Module[{
  parsedStates},
(
  parsedStates = Table[(
    {energy, eigenVec}      = state;
    maxTermIndex            = Ordering[Abs[eigenVec]][[-1]];
    {LSstring, Jval, mJval} = basis[[maxTermIndex]];
    LSJsymbol               = Subscript[LSstring, {Jval, mJval}];
    LSJMJsymbol             = LSstring <> ToString[Jval, InputForm];
    {S, L}                  = FindSL[LSstring];
    {energy, LSstring, Jval, mJval, S, L, LSJsymbol, LSJMJsymbol}
  ),
  {state, states}
  ];
  Return[parsedStates]
  )
]

ParseStatesByNumBasisVecs::usage = "ParseStatesByNumBasisVecs[states,
   basis, numBasisVecs, roundTo] takes a list of eigenstates in terms
```

```
    of their coefficients in the given basis and returns a list of the
    same states in terms of their energy and the coefficients at most
   numBasisVecs basis vectors. By default roundTo is 0.01 and this is
   the value used to round the amplitude coefficients.
The option \"Coefficients\" can be used to specify whether the
   coefficients are given as \"Amplitudes\" or \"Probabilities\". The
   default is \"Amplitudes\".
";
Options[ParseStatesByNumBasisVecs] = {"Coefficients" -> "Amplitudes",
    "Representation" -> "Ket"};
ParseStatesByNumBasisVecs[eigensys_List, basis_List,
  numBasisVecs_Integer, roundTo_Real : 0.01, OptionsPattern[]] :=
  Module[
   {parsedStates, energy, eigenVec,
   probs, amplitudes, ordering,
   chosenIndices, majorComponents,
   majorAmplitudes, majorRep},
(
   parsedStates = Table[(
     {energy, eigenVec} = state;
     energy              = Chop[energy];
     probs               = Round[Abs[eigenVec^2], roundTo];
     amplitudes          = Round[eigenVec, roundTo];
     ordering            = Ordering[probs];
     chosenIndices       = ordering[[-numBasisVecs ;;]];
     majorComponents     = basis[[chosenIndices]];
     majorThings = If[OptionValue["Coefficients"] == "Probabilities",
       (
         probs[[chosenIndices]]
       ),
       (
         amplitudes[[chosenIndices]]
       )
     ];
     majorComponents     = PrettySaundersSLJmJ[#, "Representation" ->
  OptionValue["Representation"]] & /@ majorComponents;
     nonZ                = (# != 0.) & /@ majorThings;
     majorThings         = Pick[majorThings, nonZ];
     majorComponents     = Pick[majorComponents, nonZ];
     If[OptionValue["Coefficients"] == "Probabilities",
       (
         majorThings = majorThings * 100*"%"
       )
     ];
     majorRep            = majorThings . majorComponents;
     {energy, majorRep}
     ),
     {state, eigensys}];
   Return[parsedStates]
   )
];

FindThresholdPosition::usage = "FindThresholdPosition[list, threshold
  ] returns the position of the first element in list that is greater
    than threshold. If no such element exists, it returns the length
   of list. The elements of the given list must be in ascending order.
   ";
FindThresholdPosition[list_, threshold_] :=
Module[{position},
   position = Position[list, _?(# > threshold &), 1, 1];
   thrPos = If[Length[position] > 0,
     position[[1, 1]],
     Length[list]];
   If[thrPos == 0, Return[1], Return[thrPos+1]]
   ]

ParseStateByProbabilitySum[{energy_, eigenVec_}, probSum_, roundTo_
  :0.01, maxParts_:20] := Compile[
   {{energy, _Real, 0},{eigenVec, _Complex, 1}, {probSum, _Real, 0}, {
  roundTo, _Real, 0}, {maxParts, _Integer, 0}},
   Module[
   {numStates, state, amplitudes, probs, ordering,
   orderedProbs, truncationIndex, accProb, thresholdIndex,
  chosenIndices, majorComponents,
   majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
(
```

```
    numStates    = Length[eigenVec];
    (*Round them up*)
    amplitudes         = Round[eigenVec, roundTo];
    probs              = Round[Abs[eigenVec^2], roundTo];
    ordering           = Reverse[Ordering[probs]];
    (*Order the probabilities from high to low*)
    orderedProbs       = probs[[ordering]];
    (*To speed up Accumulate, assume that only as much as maxParts will
     be needed*)
    truncationIndex    = Min[maxParts, Length[orderedProbs]];
    orderedProbs       = orderedProbs[[;;truncationIndex]];
    (*Accumulate the probabilities*)
    accProb            = Accumulate[orderedProbs];
    (*Find the index of the first element in accProb that is greater
    than probSum*)
    thresholdIndex     = Min[Length[accProb], FindThresholdPosition[
    accProb, probSum]];
    (*Grab all the indicees up till that one*)
    chosenIndices      = ordering[[;; thresholdIndex]];
    (*Select the corresponding elements from the basis*)
    majorComponents    = basis[[chosenIndices]];
    (*Select the corresponding amplitudes*)
    majorAmplitudes    = amplitudes[[chosenIndices]];
    (*Take their absolute value*)
    absMajorAmplitudes = Abs[majorAmplitudes];
    (*Make sure that there are no effectively zero contributions*)
    notnullAmplitudes  = Flatten[Position[absMajorAmplitudes, x_ /; x
    != 0]];
    (* majorComponents    = PrettySaundersSLJmJ[{#[[1]],#[[2]],#[[3]]}]
     & /@ majorComponents; *)
    majorComponents    = PrettySaundersSLJmJ /@ majorComponents;
    majorAmplitudes    = majorAmplitudes[[notnullAmplitudes]];
    (*Make them into Kets*)
    majorComponents    = Ket /@ majorComponents[[notnullAmplitudes]];
    (*Multiply and add to build the final Ket*)
    majorRep           = majorAmplitudes . majorComponents;
      );
  Return[{energy, majorRep}]
  ],
    CompilationTarget -> "C",
    RuntimeAttributes -> {Listable},
    Parallelization -> True,
    RuntimeOptions -> "Speed"
  ];

  ParseStatesByProbabilitySum::usage = "ParseStatesByProbabilitySum[
    eigensys, basis, probSum] takes a list of eigenstates in terms of
    their coefficients in the given basis and returns a list of the
    same states in terms of their energy and the coefficients of the
    basis vectors that sum to at least probSum.";
  ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ :
    0.01, maxParts_: 20] := Module[
    {parsedByProb, numStates, state, energy, eigenVec, amplitudes,
    probs, ordering,
    orderedProbs, truncationIndex, accProb, thresholdIndex,
    chosenIndices, majorComponents,
    majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
  (
    numStates    = Length[eigensys];
    parsedByProb = Table[(
      state              = eigensys[[idx]];
      {energy, eigenVec} = state;
      (*Round them up*)
      amplitudes         = Round[eigenVec, roundTo];
      probs              = Round[Abs[eigenVec^2], roundTo];
      ordering           = Reverse[Ordering[probs]];
      (*Order the probabilities from high to low*)
      orderedProbs       = probs[[ordering]];
      (*To speed up Accumulate, assume that only as much as maxParts
    will be needed*)
      truncationIndex    = Min[maxParts, Length[orderedProbs]];
      orderedProbs       = orderedProbs[[;;truncationIndex]];
      (*Accumulate the probabilities*)
      accProb            = Accumulate[orderedProbs];
      (*Find the index of the first element in accProb that is greater
    than probSum*)
```

```
3006        thresholdIndex      = Min[Length[accProb], FindThresholdPosition[
    accProb, probSum]];
3007        (*Grab all the indicees up till that one*)
3008        chosenIndices       = ordering[[;; thresholdIndex]];
3009        (*Select the corresponding elements from the basis*)
3010        majorComponents     = basis[[chosenIndices]];
3011        (*Select the corresponding amplitudes*)
3012        majorAmplitudes     = amplitudes[[chosenIndices]];
3013        (*Take their absolute value*)
3014        absMajorAmplitudes = Abs[majorAmplitudes];
3015        (*Make sure that there are no effectively zero contributions*)
3016        notnullAmplitudes   = Flatten[Position[absMajorAmplitudes, x_ /; x
     != 0]];
3017        (* majorComponents     = PrettySaundersSLJmJ
    [{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents; *)
3018        majorComponents     = PrettySaundersSLJmJ /@ majorComponents;
3019        majorAmplitudes     = majorAmplitudes[[notnullAmplitudes]];
3020        majorComponents     = majorComponents[[notnullAmplitudes]];
3021        (*Multiply and add to build the final Ket*)
3022        majorRep            = majorAmplitudes . majorComponents;
3023        {energy, majorRep}
3024        ), {idx, numStates}];
3025    Return[parsedByProb]
3026    )
3027    ];
3028
3029
3030    (* ################## Eigensystem analysis ################### *)
3031    (* ############################################################ *)
3032
3033    (* ############################################################ *)
3034    (* ########################### Misc ########################### *)
3035
3036    SymbToNum::usage = "SymbToNum[expr, numAssociation] takes an
      expression expr and returns what results after making the
      replacements defined in the given replacementAssociation. If
      replacementAssociation doesn't define values for expected keys,
      they are taken to be zero.";
3037    SymbToNum[expr_, replacementAssociation_]:= (
3038      includedKeys = Keys[replacementAssociation];
3039      (*If a key is not defined, make its value zero.*)
3040      fullAssociation = Table[(
3041        If[MemberQ[includedKeys, key],
3042          ToExpression[key]->replacementAssociation[key],
3043          ToExpression[key]->0
3044        ]
3045      ),
3046      {key, paramSymbols}];
3047      Return[expr/.fullAssociation];
3048    )
3049
3050    SimpleConjugate::usage = "SimpleConjugate[expr] takes an expression
      and applies a simplified version of the conjugate in that all it
      does is that it replaces the imaginary unit I with -I. It assumes
      that every other symbol is real so that it remains the same under
      complex conjugation. Among other expressions it is valid for any
      rational or polynomial expression with complex coefficients and
      real variables.";
3051    SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;
3052
3053    ExportMZip::usage="ExportMZip[\"dest.[zip,m]\"] saves a compressed
      version of expr to the given destination.";
3054    ExportMZip[filename_, expr_]:=Module[{baseName, exportName,
      mImportName, zipImportName},
3055    (
3056      baseName     = FileBaseName[filename];
3057      exportName   = StringReplace[filename,".m"->".zip"];
3058      mImportName  = StringReplace[exportName,".zip"->".m"];
3059      If[FileExistsQ[mImportName],
3060      (
3061        PrintTemporary[mImportName<>" exists already, deleting"];
3062        DeleteFile[mImportName];
3063        Pause[2];
3064      )
3065      ];
3066      Export[exportName, (baseName<>".m") -> expr]
```

```
3067  )
3068  ];
3069
3070  ImportMZip::usage="ImportMZip[filename] imports a .m file inside a .
        zip file with corresponding filename. If the Option \"Leave
        Uncompressed\" is set to True (the default) then this function also
         leaves an umcompressed version of the object in the same folder of
         filename";
3071  Options[ImportMZip]={"Leave Uncompressed" -> True};
3072  ImportMZip[filename_String, OptionsPattern[]] := Module[
3073    {baseName, importKey, zipImportName, mImportName, imported},
3074  (
3075    baseName      = FileBaseName[filename];
3076    (*Function allows for the filename to be .m or .zip*)
3077    importKey     = baseName <> ".m";
3078    zipImportName = StringReplace[filename, ".m"->".zip"];
3079    mImportName   = StringReplace[zipImportName, ".zip"->".m"];
3080    If[FileExistsQ[mImportName],
3081    (
3082      PrintTemporary[".m version exists already, importing that instead
         ..."];
3083      Return[Import[mImportName]];
3084    )
3085    ];
3086    imported = Import[zipImportName, importKey];
3087    If[OptionValue["Leave Uncompressed"],
3088      Export[mImportName, imported]
3089    ];
3090    Return[imported]
3091  )
3092  ];
3093
3094  ReplaceInSparseArray::usage = "ReplaceInSparseArray[sparseArray,
        rules] takes a sparse array that may contain symbolic quantities
        and returns a sparse array in which the given replacement rules
        have been used.";
3095  ReplaceInSparseArray[s_SparseArray, rule_] := (With[{
3096      elem = s["NonzeroValues"]/.rule,
3097      def  = s["Background"]/.rule
3098      },
3099      srep = SparseArray[Automatic,
3100        s["Dimensions"],
3101        def,
3102        {1, {s["RowPointers"], s["ColumnIndices"]}, elem}
3103        ];
3104    ];
3105    Return[srep];
3106    );
3107
3108  MapToSparseArray::usage = "MapToSparseArray[sparseArray, function]
        takes a sparse array and returns a sparse array after the function
        has been applied to it.";
3109  MapToSparseArray[sparsey_SparseArray, func_] := Module[{
3110      nonZ, backg, mapped
3111      },
3112     (
3113      nonZ   = func/@ sparsey["NonzeroValues"];
3114      backg  = func[sparsey["Background"]];
3115      mapped = SparseArray[Automatic,
3116        sparsey["Dimensions"],
3117        backg,
3118        {1, {sparsey["RowPointers"], sparsey["ColumnIndices"]}, nonZ}
3119        ];
3120      Return[mapped];
3121     )
3122    ];
3123
3124  ParseTeXLikeSymbol::usage = "ParseTeXLikeSymbol[string] parses a
        string for a symbol given in LaTeX notation and returns a
        corresponding mathematica symbol. The string may have expressions
        for several symbols, they need to be separated by single spaces. In
         addition the _ and ^ symbols used in LaTeX notation need to have
        arguments that are enclosed in parenthesis, for example \"x_2\" is
        invalid, instead \"x_{2}\" should have been given.";
3125  Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
3126  ParseTeXLikeSymbol[bigString_, OptionsPattern[]] := (
```

```
3127      form = OptionValue["Form"];
3128      (*parse greek*)
3129      symbols = Table[(
3130          str = StringReplace[string, {"\\alpha" -> "α",
3131            "\\beta" -> "β",
3132            "\\gamma" -> "γ",
3133            "\\psi" -> "\[Psi]"}];
3134          symbol = Which[
3135            StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"]],
3136            (
3137            (*yes sub no sup*)
3138            mainSymbol = StringSplit[str, "_"][[1]];
3139            mainSymbol = ToExpression[mainSymbol];
3140
3141            subPart =
3142              StringCases[str,
3143                RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
3144            Subscript[mainSymbol, subPart]
3145            ),
3146            Not[StringContainsQ[str, "_"]] && StringContainsQ[str, "^"],
3147            (
3148            (*no sub yes sup*)
3149            mainSymbol = StringSplit[str, "^"][[1]];
3150            mainSymbol = ToExpression[mainSymbol];
3151
3152            supPart =
3153              StringCases[str,
3154                RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
3155            Superscript[mainSymbol, supPart]),
3156            StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
3157            (
3158            (*yes sub yes sup*)
3159            mainSymbol = StringSplit[str, "_"][[1]];
3160            mainSymbol = ToExpression[mainSymbol];
3161            {subPart, supPart} =
3162              StringCases[str, RegularExpression@"\\{(.*?)\\}" -> "$1"];
3163            Subsuperscript[mainSymbol, subPart, supPart]
3164            ),
3165            True,
3166            ((*no sup or sub*)
3167            str)
3168            ];
3169          symbol
3170          ),
3171        {string, StringSplit[bigString, " "]}];
3172      Which[
3173        form == "Row",
3174        Return[Row[symbols]],
3175        form == "List",
3176        Return[symbols]
3177        ]
3178      );
3179
3180    (* ########################## Misc ########################### *)
3181    (* ############################################################ *)
3182
3183    (* ############################################################ *)
3184    (* ################## Some Plotting Routines ################## *)
3185
3186    EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes states
        and produces a visualization of its energy spectrum.
3187    The resultant visualization can be navigated by clicking and dragging
         to zoom in on a region, or by clicking and dragging horizontally
        while pressing Ctrl. Double-click to reset the view.";
3188    Options[EnergyLevelDiagram] = {
3189      "Title"->"",
3190      "ImageSize"->1000,
3191      "AspectRatio" -> 1/8,
3192      "Background"->"Automatic",
3193      "Epilog"->{},
3194      "Explorer"->True
3195      };
3196    EnergyLevelDiagram[states_, OptionsPattern[]]:= (
3197      energies = First/@states;
3198      epi = OptionValue["Epilog"];
3199      explora = If[OptionValue["Explorer"],
```

```
      ExploreGraphics,
      Identity
    ];
    explora@ListPlot[Tooltip[{{#, 0}, {#, 1}}, {Quantity[#/8065.54429,
    "eV"], Quantity[#, 1/"Centimeters"]}] &/@ energies,
      Joined       -> True,
      PlotStyle    -> Black,
      AspectRatio  -> OptionValue["AspectRatio"],
      ImageSize    -> OptionValue["ImageSize"],
      Frame        -> True,
      PlotRange    -> {All, {0, 1}},
      FrameTicks   -> {{None, None}, {Automatic, Automatic}},
      FrameStyle   -> Directive[15, Dashed, Thin],
      PlotLabel    -> Style[OptionValue["Title"], 15, Bold],
      Background   -> OptionValue["Background"],
      FrameLabel   -> {"\!\(\*FractionBox[\(E\), SuperscriptBox[\(cm\),
    \(-1\)]]\)"},
      Epilog       -> epi]
  )

  ExploreGraphics::usage =
    "Pass a Graphics object to explore it. Zoom by clicking and
    dragging a rectangle. Pan by clicking and dragging while pressing
    Ctrl. Click twice to reset view.
    Based on ZeitPolizei @ https://mathematica.stackexchange.com/
    questions/7142/how-to-manipulate-2d-plots";

  OptAxesRedraw::usage =
    "Option for ExploreGraphics to specify redrawing of axes. Default
    False.";
  Options[ExploreGraphics] = {OptAxesRedraw -> False};

  ExploreGraphics[graph_Graphics, opts : OptionsPattern[]] := With[
    {gr  = First[graph],
      opt = DeleteCases[Options[graph],
            PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _],
      plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
      ar  = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
      ao  = AbsoluteOptions[AxesOrigin],
      rectangle = {Dashing[Small],
        Line[{#1,
              {First[#2], Last[#1]},
              #2,
              {First[#1], Last[#2]},
              #1}]} &,
      optAxesRedraw = OptionValue[OptAxesRedraw]},
    DynamicModule[
        {dragging=False, first, second, rx1, rx2, ry1, ry2,
        range = plr},
        {{rx1, rx2}, {ry1, ry2}} = plr;
      Panel@
      EventHandler[
        Dynamic@Graphics[
          If[dragging, {gr, rectangle[first, second]}, gr],
          PlotRange -> Dynamic@range,
          AspectRatio -> ar,
          AxesOrigin -> If[optAxesRedraw,
            Dynamic@Mean[range\[Transpose]], ao],
          Sequence @@ opt],
        {{"MouseDown", 1} :> (
          first = MousePosition["Graphics"]
          ),
        {"MouseDragged", 1} :> (
          dragging = True;
          second = MousePosition["Graphics"]
          ),
        "MouseClicked" :> (
          If[CurrentValue@"MouseClickCount"==2,
            range = plr];
          ),
        {"MouseUp", 1} :> If[dragging,
          dragging = False;

          range = {{rx1, rx2}, {ry1, ry2}} =
            Transpose@{first, second};
          range[[2]] = {0, 1}],
```

```
        {"MouseDown", 2} :> (
          first = {sx1, sy1} = MousePosition["Graphics"]
          ),
        {"MouseDragged", 2} :> (
          second = {sx2, sy2} = MousePosition["Graphics"];
          rx1 = rx1 - (sx2 - sx1);
          rx2 = rx2 - (sx2 - sx1);
          ry1 = ry1 - (sy2 - sy1);
          ry2 = ry2 - (sy2 - sy1);
          range = {{rx1, rx2}, {ry1, ry2}};
          range[[2]] = {0, 1};
          )}]]];

Options[LabeledGrid]={
    ItemSize->Automatic,
    Alignment->Center,
    Frame->All,
    "Separator"->",",
    "Pivot"->""
};
LabeledGrid::usage="LabeledGrid[data, rowHeaders, columnHeaders]
  provides a grid of given data interpreted as a matrix of values
  whose rows are labeled by rowHeaders and whose columns are labeled
  by columnHeaders. When hovering with the mouse over the grid
  elements, the row and column labels are displayed with the given
  separator between them.";
LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]]:=
  Module[
    {gridList=data,rowHeads=rowHeaders,colHeads=columnHeaders},
    (
    separator=OptionValue["Separator"];
    pivot=OptionValue["Pivot"];
    gridList=Table[
            Tooltip[
              data[[rowIdx,colIdx]],
              DisplayForm[
                RowBox[{rowHeads[[rowIdx]],
                        separator,
                        colHeads[[colIdx]]}
                      ]
                      ]
                    ],
        {rowIdx,Dimensions[data][[1]]},
        {colIdx,Dimensions[data][[2]]}];
    gridList=Transpose[Prepend[gridList,colHeads]];
    rowHeads=Prepend[rowHeads,pivot];
    gridList=Prepend[gridList,rowHeads]//Transpose;
    Grid[gridList,
        Frame->OptionValue[Frame],
        Alignment->OptionValue[Alignment],
        Frame->OptionValue[Frame],
        ItemSize->OptionValue[ItemSize]
        ]
)
]

HamiltonianForm::usage="HamiltonianForm[hamMatrix, basisLabels] takes
   the matrix representation of a hamiltonian together with a set of
  symbols representing the ordered basis in which the operator is
  represented. With this it creates a displayed form that has
  adequately labeled row and columns together with informative values
   when hovering over the matrix elements using the mouse cursor.";
Options[HamiltonianForm]={"Separator"->","Pivot"->""}
HamiltonianForm[hamMatrix_, basisLabels_List, OptionsPattern[]]:=(
    braLabels=DisplayForm[RowBox[{"\[LeftAngleBracket]",#,"\[
  RightBracketingBar]"}]]& /@ basisLabels;
    ketLabels=DisplayForm[RowBox[{"\[LeftBracketingBar]",#,"\[
  RightAngleBracket]"}]]& /@ basisLabels;
    LabeledGrid[hamMatrix,braLabels,ketLabels,"Separator"->
  OptionValue["Separator"],"Pivot"->OptionValue["Pivot"]]
    )

Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover"
  -> True, "Overlay Values" -> True}];
HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts : OptionsPattern
  []] := (
```

```
    braLabels = DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[
    RightBracketingBar]"}]] & /@ basisLabels;
    ketLabels = DisplayForm[Rotate[RowBox[{"\[LeftBracketingBar]", #, "
    \[RightAngleBracket]"}],\[Pi]/2]] & /@ basisLabels;
    ketLabelsUpright = DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "
    \[RightAngleBracket]"}]] & /@ basisLabels;
    numRows = Length[hamMatrix];
    numCols = Length[hamMatrix[[1]]];
    epiThings = Which[
      And[OptionValue["Hover"], Not[OptionValue["Overlay Values"]]],
      Flatten[
        Table[
          Tooltip[
            {
              Transparent,
              Rectangle[
              {j - 1, numRows - i},
              {j - 1, numRows - i} + {1, 1}
              ]
            },
          Row[{braLabels[[i]],ketLabelsUpright[[j]],"=",hamMatrix[[i, j
    ]]}]
          ],
        {i, 1, numRows},
        {j, 1, numCols}
        ]
      ],
      And[OptionValue["Hover"], OptionValue["Overlay Values"]],
      Flatten[
        Table[
          Tooltip[
            {
              Transparent,
              Rectangle[
              {j - 1, numRows - i},
              {j - 1, numRows - i} + {1, 1}
              ]
            },
          DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i]],
    "\[LeftBracketingBar]", basisLabels[[j]], "\[RightAngleBracket]"}]]
          ],
        {i, numRows},
        {j, numCols}
        ]
      ],
      True,
      {}
      ];
    textOverlay = If[OptionValue["Overlay Values"],
      (
        Flatten[
        Table[
          Text[hamMatrix[[i, j]],
            {j - 1/2, numRows - i + 1/2}
          ],
        {i, 1, numRows},
        {j, 1, numCols}
        ]
        ]
      ),
      {}
      ];
    epiThings = Join[epiThings, textOverlay];
    MatrixPlot[hamMatrix,
      FrameTicks -> {
        {Transpose[{Range[Length[braLabels]], braLabels}], None},
        {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}
        },
      Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
      Epilog -> epiThings
    ]
    );

  (* ################## Some Plotting Routines ################## *)
  (* ############################################################ *)

```

```
(* ############################################################ *)
(* ###################### Load Functions ###################### *)

LoadAll::usage="LoadAll[] executes all Load* functions.";
LoadAll[]:=(
    LoadTermLabels[];
    LoadCFP[];
    LoadUk[];
    LoadV1k[];
    LoadT22[];
    LoadSOOandECSOLS[];

    LoadElectrostatic[];
    LoadSpinOrbit[];
    LoadSOOandECSO[];
    LoadSpinSpin[];
    LoadThreeBody[];
    LoadChenDeltas[];
    LoadCarnall[];
  )

fnTermLabels::usage = "This list contains the labels of f^n
  configurations. Each element of the list has four elements {LS,
  seniority, W, U}. At first sight this seems to only include the
  labels for the f^6 and f^7 configuration, however, all is included
  in these two.";

LoadTermLabels::usage="LoadTermLabels[] loads into the session the
  labels for the terms in the f^n configurations.";
LoadTermLabels[]:= (
  If[ValueQ[fnTermLabels], Return[]];
  PrintTemporary["Loading data for state labels in the f^n
  configurations..."];
  fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];

  If[!FileExistsQ[fnTermsFname],
    (PrintTemporary[">> fnTerms.m not found, generating ..."];
      fnTermLabels = ParseTermLabels["Export"->True];
    ),
    fnTermLabels = Import[fnTermsFname];
  ];
)


Carnall::usage = "Association of data from Carnall et al (1989) with
  the following keys: {data, annotations, paramSymbols, elementNames,
   rawData, rawAnnotations, annnotatedData, appendix:Pr:Association,
  appendix:Pr:Calculated, appendix:Pr:RawTable, appendix:Headings}";
LoadCarnall::usage="LoadCarnall[] loads data for trivalent
  lanthanides in LaF3 using the data from Bill Carnall's 1989 paper."
  ;
LoadCarnall[]:=(
  If[ValueQ[Carnall], Return[]];
  carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
  If[!FileExistsQ[carnallFname],
    (PrintTemporary[">> Carnall.m not found, generating ..."];
      Carnall = ParseCarnall[];
    ),
    Carnall = Import[carnallFname];
  ];
)

LoadChenDeltas::usage="LoadChenDeltas[] loads the differences noted
  by Chen.";
LoadChenDeltas[]:=(
  If[ValueQ[chenDeltas], Return[]];
  PrintTemporary["Loading the association of discrepancies found by
  Chen ..."];
  chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"
  }];
  If[!FileExistsQ[chenDeltasFname],
    (PrintTemporary[">> chenDeltas.m not found, generating ..."];
      chenDeltas = ParseChenDeltas[];
    ),
    chenDeltas = Import[chenDeltasFname];
  ];
```

```
);

ParseChenDeltas::usage="ParseChenDeltas[] parses the data found in ./data/the-chen-deltas-A.csv and ./data/the-chen-deltas-B.csv. If the option \"Export\" is set to True (True is the default), then the parsed data is saved to ./data/chenDeltas.m";
Options[ParseChenDeltas] = {"Export" -> True};
ParseChenDeltas[OptionsPattern[]]:=(
  chenDeltasRaw = Import[FileNameJoin[{moduleDir, "data", "the-chen-deltas-A.csv"}]];
  chenDeltasRaw = chenDeltasRaw[[2 ;;]];
  chenDeltas = <||>;
  chenDeltasA = <||>;
  Off[Power::infy];
  Do[
    ({right, wrong} = {chenDeltasRaw[[row]][[4 ;;]],
      chenDeltasRaw[[row + 1]][[4 ;;]]};
    key = chenDeltasRaw[[row]][[1 ;; 3]];
    repRule = (#[[1]] -> #[[2]]*#[[1]]) & /@
      Transpose[{{M0, M2, M4, P2, P4, P6}, right/wrong}];
    chenDeltasA[key] = <|"right" -> right, "wrong" -> wrong,
      "repRule" -> repRule|>;
    chenDeltasA[{key[[1]], key[[3]], key[[2]]}] = <|"right" -> right,
      "wrong" -> wrong, "repRule" -> repRule|>;
    ),
    {row, 1, Length[chenDeltasRaw], 2}];
  chenDeltas["A"] = chenDeltasA;

  chenDeltasRawB = Import[FileNameJoin[{moduleDir, "data", "the-chen-deltas-B.csv"}], "Text"];
  chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
  chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
  chenDeltasB = {ToExpression[StringTake[#[[1]], {2}]], #[[2]], #[[3]]} & /@ chenDeltasB;
  chenDeltas["B"] = chenDeltasB;
  On[Power::infy];
  If[OptionValue["Export"],
    (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"}];
    Export[chenDeltasFname, chenDeltas];
    )
    ];
  Return[chenDeltas];
)

ParseCarnall::usage="ParseCarnall[] parses the data found in ./data/Carnall.xls. If the option \"Export\" is set to True (True is the default), then the parsed data is saved to ./data/Carnall. This data is from the tables and appendices of Carnall et al (1989).";
Options[ParseCarnall] = {"Export" -> True};
ParseCarnall[] := (
  ions         = {"Pr","Nd","Pm","Sm","Eu","Gd","Tb","Dy","Ho","Er","Tm"};
  templates    = StringTemplate/@StringSplit["appendix:`ion`:Association appendix:`ion`:Calculated appendix:`ion`:RawTable appendix:`ion`:Headings"," "];

  (* How many unique eigenvalues, after removing Kramer's degeneracy *)
  fullSizes    = AssociationThread[ions, {91, 182, 1001, 1001, 3003, 1716, 3003, 1001, 1001, 182, 91}];
  carnall      = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[2]];
  carnallErr   = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[3]];

  elementNames = carnall[[1]][[2;;]];
  carnall      = carnall[[2;;]];
  carnallErr   = carnallErr[[2;;]];
  carnall      = Transpose[carnall];
  carnallErr   = Transpose[carnallErr];
  paramNames   = ToExpression/@carnall[[1]][[1;;]];
  carnall      = carnall[[2;;]];
  carnallErr   = carnallErr[[2;;]];
  carnallData  = Table[(
                   data         = carnall[[i]];
```

```
                    data          = (#[[1]]->#[[2]])&/@Select[Transpose
    [{paramNames,data}],#[[2]]!=""&];
                    elementNames[[i]]->data
                    ),
                    {i,1,13}
                    ];
    carnallData  = Association[carnallData];
    carnallNotes = Table[(
                    data        = carnallErr[[i]];
                    elementName = elementNames[[i]];
                    dataFun     = (
                        #[[1]] -> If[#[[2]]=="[]",
                        "Not allowed to vary in fitting.",
                        If[#[[2]]=="[R]",
                            "Ratio constrained by: " <> <|"Eu"->"F4/F2
    =0.713; F6/F2=0.512",
                                "Gd"->"F4/F2=0.710]",
                                "Tb"->"F4/F2=0.707"|>[elementName],
                            If[#[[2]]=="i",
                                "Interpolated",
                                #[[2]]
                            ]
                        ]
                        ]) &;
                    data = dataFun /@ Select[Transpose[{paramNames,data
    }],#[[2]]!=""&];
                    elementName->data
                      ),
                  {i,1,13}
                  ];
    carnallNotes  = Association[carnallNotes];

    annotatedData = Table[
                    If[NumberQ[#[[1]]],Tooltip[#[[1]],#[[2]]],""] & /@
    Transpose[{paramNames/.carnallData[element],
                      paramNames/.carnallNotes[element]
                      }],
                    {element,elementNames}
                    ];
    annotatedData = Transpose[annotatedData];

    Carnall = <|"data"       -> carnallData,
        "annotations"        -> carnallNotes,
        "paramSymbols"       -> paramNames,
        "elementNames"       -> elementNames,
        "rawData"            -> carnall,
        "rawAnnotations"     -> carnallErr,
        "includedTableIons" -> ions,
        "annnotatedData"     -> annotatedData
    |>;

    Do[(
        carnallData    = Import[FileNameJoin[{moduleDir,"data","Carnall.
    xls"}]][[i]];
        headers        = carnallData[[1]];
        calcIndex      = Position[headers,"Calc (1/cm)"][[1,1]];
        headers        = headers[[2;;]];
        carnallLabels  = carnallData[[1]];
        carnallData    = carnallData[[2;;]];
        carnallTerms   = DeleteDuplicates[First/@carnallData];
        parsedData     = Table[(
                            rows = Select[carnallData,#[[1]]==term&];
                            rows = #[[2;;]]&/@rows;
                            rows = Transpose[rows];
                            rows = Transpose[{headers,rows}];
                            rows = Association[(#[[1]]->#[[2]])&/@rows];
                            term->rows
                            ),
                          {term,carnallTerms}
                          ];
        carnallAssoc         = Association[parsedData];
        carnallCalcEnergies  = #[[calcIndex]]&/@carnallData;
        carnallCalcEnergies  = If[NumberQ[#],#,Missing[]]&/
    @carnallCalcEnergies;
        ion                  = ions[[i-3]];
        carnallCalcEnergies  = PadRight[carnallCalcEnergies, fullSizes[
```

```
  ion], Missing[]];
3592        keys                = #[<|"ion"->ion|>]&/@templates;
3593        Carnall[keys[[1]]]  = carnallAssoc;
3594        Carnall[keys[[2]]]  = carnallCalcEnergies;
3595        Carnall[keys[[3]]]  = carnallData;
3596        Carnall[keys[[4]]]  = headers;
3597        ),
3598    {i,4,14}
3599    ];
3600
3601    goodions = Select[ions,#!="Pm"&];
3602    expData  = Select[Transpose[Carnall["appendix:"<>#<>":RawTable"
    ]][[1+Position[Carnall["appendix:"<>#<>":Headings"],"Exp (1/cm)"
    ]][[1,1]]]],NumberQ]&/@goodions;
3603    Carnall["All Experimental Data"]=AssociationThread[goodions,expData
    ];
3604    If[OptionValue["Export"],
3605      (
3606        carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
3607        Print["Exporting to "<>exportFname];
3608        Export[carnallFname, Carnall];
3609      )
3610      ];
3611    Return[Carnall];
3612  )
3613
3614  CFP::usage = "CFP[{n, NKSL}] provides a list whose first element
      echoes NKSL and whose other elements are lists with two elements
      the first one being the symbol of a parent term and the second
      being the corresponding coefficient of fractional parentage. n must
       satisfy 1 <= n <= 7";
3615
3616  CFPAssoc::usage = " CFPAssoc is an association where keys are of
      lists of the form {num_electrons, daugherTerm, parentTerm} and
      values are the corresponding coefficients of fractional parentage.
      The terms given in string-spectroscopic notation. If a certain
      daughter term does not have a parent term, the value is 0. Loaded
      using LoadCFP[].";
3617
3618  LoadCFP::usage="LoadCFP[] loads CFP, CFPAssoc, and CFPTable into the
      session.";
3619  LoadCFP[]:=(
3620    If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]],Return[]];
3621
3622    PrintTemporary["Loading CFPtable ..."];
3623    CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
3624    If[!FileExistsQ[CFPTablefname],
3625      (PrintTemporary[">> CFPTable.m not found, generating ..."];
3626        CFPTable = GenerateCFPTable["Export"->True];
3627      ),
3628      CFPTable = Import[CFPTablefname];
3629    ];
3630
3631    PrintTemporary["Loading CFPs.m ..."];
3632    CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
3633    If[!FileExistsQ[CFPfname],
3634      (PrintTemporary[">> CFPs.m not found, generating ..."];
3635        CFP = GenerateCFP["Export"->True];
3636      ),
3637      CFP = Import[CFPfname];
3638    ];
3639
3640    PrintTemporary["Loading CFPAssoc.m ..."];
3641    CFPAfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
3642    If[!FileExistsQ[CFPAfname],
3643      (PrintTemporary[">> CFPAssoc.m not found, generating ..."];
3644        CFPAssoc = GenerateCFPAssoc["Export"->True];
3645      ),
3646      CFPAssoc = Import[CFPAfname];
3647    ];
3648  );
3649
3650  ReducedUkTable::usage = "ReducedUkTable[{n, l = 3, SL, SpLp, k}]
      provides reduced matrix elements of the spherical tensor operator
      Uk. See TASS section 11-9 \"Unit Tensor Operators\". Loaded using
      LoadUk[].";
```

```
3651
3652  LoadUk::usage="LoadUk[] loads into session the reduced matrix
        elements for unit tensor operators.";
3653  LoadUk[]:=(
3654    If[ValueQ[ReducedUkTable], Return[]];
3655    PrintTemporary["Loading the association of reduced matrix elements
       for unit tensor operators ..."];
3656    ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "
       ReducedUkTable.m"}];
3657    If[!FileExistsQ[ReducedUkTableFname],
3658      (PrintTemporary[">> ReducedUkTable.m not found, generating ..."];
3659        ReducedUkTable = GenerateReducedUkTable[7];
3660      ),
3661      ReducedUkTable = Import[ReducedUkTableFname];
3662    ];
3663  );
3664
3665  ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}]
       provides the calculated result of Electrostatic[{n, SL, SpLp}].
       Load using LoadElectrostatic[].";
3666
3667  LoadElectrostatic::usage="LoadElectrostatic[] loads the reduced
       matrix elements for the electrostatic interaction.";
3668  LoadElectrostatic[]:=(
3669    If[ValueQ[ElectrostaticTable], Return[]];
3670    PrintTemporary["Loading the association of matrix elements for the
       electrostatic interaction ..."];
3671    ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "
       ElectrostaticTable.m"}];
3672    If[!FileExistsQ[ElectrostaticTablefname],
3673      (PrintTemporary[">> ElectrostaticTable.m not found, generating
       ..."];
3674        ElectrostaticTable = GenerateElectrostaticTable[7];
3675      ),
3676      ElectrostaticTable = Import[ElectrostaticTablefname];
3677    ];
3678  );
3679
3680  LoadV1k::usage="LoadV1k[] loads into session the matrix elements of
       V1k.";
3681  LoadV1k[]:=(
3682    If[ValueQ[ReducedV1kTable], Return[]];
3683    PrintTemporary["Loading the association of matrix elements for V1k
       ..."];
3684    ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "
       ReducedV1kTable.m"}];
3685    If[!FileExistsQ[ReducedV1kTableFname],
3686      (PrintTemporary[">> ReducedV1kTable.m not found, generating ..."
       ];
3687        ReducedV1kTable = GenerateReducedV1kTable[7];
3688      ),
3689      ReducedV1kTable = Import[ReducedV1kTableFname];
3690    ]
3691  );
3692
3693  LoadSpinOrbit::usage="LoadSpinOrbit[] loads into session the matrix
       elements of the spin-orbit interaction.";
3694  LoadSpinOrbit[]:=(
3695    If[ValueQ[SpinOrbitTable], Return[]];
3696    PrintTemporary["Loading the association of matrix elements for spin
       -orbit ..."];
3697    SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "
       SpinOrbitTable.m"}];
3698    If[!FileExistsQ[SpinOrbitTableFname],
3699      (PrintTemporary[">> SpinOrbitTable.m not found, generating ..."];
3700        SpinOrbitTable = GenerateSpinOrbitTable[7, "Export" -> True];
3701      ),
3702      SpinOrbitTable = Import[SpinOrbitTableFname];
3703    ]
3704  );
3705
3706  LoadSOOandECSOLS::usage="LoadSOOandECSOLS[] loads into session the LS
       reduced matrix elements of the SOO-ECSO interaction.";
3707  LoadSOOandECSOLS[]:=(
3708    If[ValueQ[SOOandECSOLSTable], Return[]];
3709    PrintTemporary["Loading the association of LS reduced matrix
```

```
    elements for SOO-ECSO ..."];
3710    SOOandECSOLSTableFname = FileNameJoin[{moduleDir, "data", "
    ReducedSOOandECSOLSTable.m"}];
3711    If[!FileExistsQ[SOOandECSOLSTableFname],
3712      (PrintTemporary[">> ReducedSOOandECSOLSTable.m not found,
    generating ..."];
3713        SOOandECSOLSTable = GenerateSOOandECSOLSTable[7];
3714      ),
3715      SOOandECSOLSTable = Import[SOOandECSOLSTableFname];
3716    ];
3717  );
3718
3719  LoadSOOandECSO::usage="LoadSOOandECSO[] loads into session the LSJ
    reduced matrix elements of spin-other-orbit and electrostatically-
    correlated-spin-orbit.";
3720  LoadSOOandECSO[]:=(
3721    If[ValueQ[SOOandECSOTableFname], Return[]];
3722    PrintTemporary["Loading the association of matrix elements for spin
    -other-orbit and electrostatically-correlated-spin-orbit ..."];
3723    SOOandECSOTableFname = FileNameJoin[{moduleDir, "data", "
    SOOandECSOTable.m"}];
3724    If[!FileExistsQ[SOOandECSOTableFname],
3725      (PrintTemporary[">> SOOandECSOTable.m not found, generating ..."
    ];
3726        SOOandECSOTable = GenerateSOOandECSOTable[7, "Export"->True];
3727      ),
3728      SOOandECSOTable = Import[SOOandECSOTableFname];
3729    ];
3730  );
3731
3732  LoadT22::usage="LoadT22[] loads into session the matrix elements of
    T22.";
3733  LoadT22[]:=(
3734    If[ValueQ[T22Table], Return[]];
3735    PrintTemporary["Loading the association of reduced T22 matrix
    elements ..."];
3736    T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m
    "}];
3737    If[!FileExistsQ[T22TableFname],
3738      (PrintTemporary[">> ReducedT22Table.m not found, generating ..."
    ];
3739        T22Table = GenerateT22Table[7];
3740      ),
3741      T22Table = Import[T22TableFname];
3742    ];
3743  );
3744
3745  LoadSpinSpin::usage="LoadSpinSpin[] loads into session the matrix
    elements of spin-spin.";
3746  LoadSpinSpin[]:=(
3747    If[ValueQ[SpinSpinTable], Return[]];
3748    PrintTemporary["Loading the association of matrix elements for spin
    -spin ..."];
3749    SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "
    SpinSpinTable.m"}];
3750    If[!FileExistsQ[SpinSpinTableFname],
3751      (PrintTemporary[">> SpinSpinTable.m not found, generating ..."];
3752        SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
3753      ),
3754      SpinSpinTable = Import[SpinSpinTableFname];
3755    ];
3756  );
3757
3758  LoadThreeBody::usage="LoadThreeBody[] loads into session the matrix
    elements of three-body configuration-interaction effects.";
3759  LoadThreeBody[]:=(
3760    If[ValueQ[ThreeBodyTable], Return[]];
3761    PrintTemporary["Loading the association of matrix elements for
    three-body configuration-interaction effects ..."];
3762    ThreeBodyFname   = FileNameJoin[{moduleDir, "data", "ThreeBodyTable
    .m"}];
3763    ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "
    ThreeBodyTables.m"}];
3764    If[!FileExistsQ[ThreeBodyFname],
3765      (PrintTemporary[">> ThreeBodyTable.m not found, generating ..."];
3766        {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTables[14,
```

```
    "Export" -> True];
      ),
      ThreeBodyTable = Import[ThreeBodyFname];
      ThreeBodyTables = Import[ThreeBodiesFname];
    ];
  );

  (* ##################### Load Functions ###################### *)
  (* ############################################################ *)

End[]

LoadTermLabels[];
LoadCFP[];

EndPackage[]
```

11 qonstants.m

This file has a few constants and conversion factors.

```
BeginPackage["qonstants`"];

(* Physical Constants*)
bohrRadius = 5.29177210903 * 10^-9;
ee         = 1.602176634 * 10^-19;

(* Spectroscopic niceties*)
theLanthanides = {"Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy",
    "Ho", "Er", "Tm", "Yb"};
theActinides   = {"Ac", "Th", "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk",
   "Cf", "Es", "Fm", "Md", "No", "Lr"};
theTrivalents  = {"Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho",
    "Er", "Tm"};
specAlphabet   = "SPDFGHIKLMNOQRTUV"

(* SI *)
\[Mu]0  = 4 \[Pi]*10^-7;             (* magnetic permeability in vacuum
    in SI *)
hPlanck = 6.62607015*10^-34;         (* Planck's constant in SI *)
\[Mu]B  = 9.2740100783*10^-24;       (* Bohr magneton in SI *)
me      = 9.1093837015*10^-31;       (* electron mass in SI *)
cLight  = 2.99792458*10^8;           (* speed of light in SI *)
eCharge = 1.602176634*10^-19;        (* elementary charge in SI *)
αFine = 1/137.036;             (* fine structure constant in SI *)
bohrRadius   = 5.29177210903*10^-11; (* Bohr radius in SI *)

(* Hartree atomic units *)
hPlanckHartree = 2 \[Pi]; (* Planck's constant in Hartree *)
meHartree      = 1;       (* electron mass in Hartree *)
cLightHartree  = 137.036; (* speed of light in Hartree *)
eChargeHartree = 1;       (* elementary charge in Hartree *)
\[Mu]0Hartree  = αFine^2; (* magnetic permeability in vacuum in Hartree
    *)

(* some conversion factors *)
eVtoKayser    = 8065.54429;   (* 1 eV = 8065.54429 cm^-1 *)
KaysertoeV    = 1/eVtoKayser; (* 1 cm^-1 = 1/8065.54429 eV *)
TeslaToKayser = 2 * \[Mu]B / hPlanck / cLight / 100;

EndPackage[];
```

12 qplotter.m

This module has a few useful plotting routines.

```

BeginPackage["qplotter`"];

GetColor;
IndexMappingPlot;
ListLabelPlot;
AutoGraphicsGrid;
SpectrumPlot;
```

```
WaveToRGB;

Begin["`Private`"];

  AutoGraphicsGrid::usage="AutoGraphicsGrid[graphsList] takes a list of
     graphics and creates a GraphicsGrid with them. The number of
    columns and rows is chosen automatically so that the grid has a
    squarish shape.";
  Options[AutoGraphicsGrid] = Options[GraphicsGrid];
  AutoGraphicsGrid[graphsList_, opts : OptionsPattern[]] :=
    (
      numGraphs = Length[graphsList];
      width = Floor[Sqrt[numGraphs]];
      height = Ceiling[numGraphs/width];
      groupedGraphs = Partition[graphsList, width, width, 1, Null];
      GraphicsGrid[groupedGraphs, opts]
    )

  Options[IndexMappingPlot] = Options[Graphics];
  IndexMappingPlot::usage =
    "IndexMappingPlot[pairs] take a list of pairs of integers and
    creates a visual representation of how they are paired. The first
    indices being depicted in the bottom and the second indices being
    depicted on top.";
  IndexMappingPlot[pairs_, opts : OptionsPattern[]] := Module[{width,
    height}, (
    width = Max[First /@ pairs];
    height = width/3;
    Return[
      Graphics[{{Tooltip[Point[{#[[1]], 0}],#[[1]]]}, Tooltip[Point
    [{#[[2]], height}],#[[2]]],
          Line[{{#[[1]], 0}, {#[[2]], height}}]} & /@ pairs, opts,
    ImageSize -> 800]]
    )
    ]

  TickCompressor[fTicks_] :=
  Module[{avgTicks, prevTickLabel, groupCounter, groupTally, idx,
    tickPosition, tickLabel, avgPosition, groupLabel}, (avgTicks = {};
    prevTickLabel = fTicks[[1, 2]];
    groupCounter = 0;
    groupTally = 0;
    idx = 1;
    Do[({tickPosition, tickLabel} = tick;
      If[
      tickLabel === prevTickLabel,
      (groupCounter += 1;
        groupTally += tickPosition;
        groupLabel = tickLabel;),
      (
        avgPosition = groupTally/groupCounter;
        avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
        groupCounter = 1;
        groupTally = tickPosition;
        groupLabel = tickLabel;
        )
      ];
      If[idx != Length[fTicks],
      prevTickLabel = tickLabel;
      idx += 1;]
      ), {tick, fTicks}];
    If[Or[Not[prevTickLabel === tickLabel], groupCounter > 1],
    (
      avgPosition = groupTally/groupCounter;
      avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
      )
    ];
    Return[avgTicks];)]

  GetColor[s_Style] := s /. Style[_, c_] :> c
  GetColor[_] := Black

  ListLabelPlot::usage="ListLabelPlot[data, labels] takes a list of
    numbers with corresponding labels. The data is grouped according to
     the labels and a ListPlot is created with them so that each group
    has a different color and their corresponding label is shown in the
```

```
    horizontal axis.";
Options[ListLabelPlot] = Append[Options[ListPlot], "TickCompression"
  ->True];
ListLabelPlot[data_, labels_, opts : OptionsPattern[]] := Module[
  {uniqueLabels, pallete, groupedByTerm, groupedKeys, scatterGroups,
  groupedColors, frameTicks, compTicks, bottomTicks, topTicks},
  (
  uniqueLabels  = DeleteDuplicates[labels];
  pallete = Table[ColorData["Rainbow", i], {i, 0, 1,
      1/(Length[uniqueLabels] - 1)}];
  uniqueLabels  = (#[[1]] -> #[[2]]) & /@ Transpose[{RandomSample[
  uniqueLabels], pallete}];
  uniqueLabels  = Association[uniqueLabels];
  groupedByTerm = GroupBy[Transpose[{labels, Range[Length[data]],
  data}], First];
  groupedKeys   = Keys[groupedByTerm];
  scatterGroups = Transpose[Transpose[#][[2 ;; 3]]] & /@ Values[
  groupedByTerm];
  groupedColors = uniqueLabels[#] & /@ groupedKeys;
  frameTicks    = {Transpose[{Range[Length[data]],
    Style[Rotate[#, 0], uniqueLabels[#]] & /@ labels}],
    Automatic};
    If[OptionValue["TickCompression"], (
        compTicks = TickCompressor[frameTicks[[1]]];
        bottomTicks =
            MapIndexed[
            If[EvenQ[First[#2]], {#1[[1]],
                Tooltip[Style["\[SmallCircle]", GetColor
  [#1[[2]]]],#1[[2]]]
                }, #1] &, compTicks];
        topTicks =
            MapIndexed[
            If[OddQ[First[#2]], {#1[[1]],
                Tooltip[Style["\[SmallCircle]", GetColor
  [#1[[2]]]],#1[[2]]]
                }, #1] &, compTicks];
        frameTicks = {{Automatic, Automatic}, {bottomTicks, topTicks
  }};)
  ];
  ListPlot[scatterGroups,
    opts,
    Frame->True,
    PlotStyle -> groupedColors,
    FrameTicks -> frameTicks]
  )
  ]

WaveToRGB::usage="WaveToRGB[wave, gamma] takes a wavelength in nm and
   returns the corresponding RGB color. The gamma parameter is
  optional and defaults to 0.8. The wavelength wave is assumed to be
  in nm. If the wavelength is below 380 the color will be the same as
   for 380 nm. If the wavelength is above 750 the color will be the
  same as for 750 nm. The function returns an RGBColor object. REF:
  https://www.noah.org/wiki/wave_to_rgb_in_Python. ";
WaveToRGB[wave_, gamma_ : 0.8] := (
  wavelength = (wave);
  Which[
    wavelength < 380,
      wavelength = 380,
    wavelength > 750,
      wavelength = 750
  ];
  Which[380 <= wavelength <= 440,
  (
    attenuation = 0.3 + 0.7*(wavelength - 380)/(440 - 380);
    R = ((-(wavelength - 440)/(440 - 380))*attenuation)^gamma;
    G = 0.0;
    B = (1.0*attenuation)^gamma;
    ),
  440 <= wavelength <= 490,
  (
    R = 0.0;
    G = ((wavelength - 440)/(490 - 440))^gamma;
    B = 1.0;
    ),
  490 <= wavelength <= 510,
```

```
    (
      R = 0.0;
      G = 1.0;
      B = (-(wavelength - 510)/(510 - 490))^gamma;
      ),
    510 <= wavelength <= 580,
    (
      R = ((wavelength - 510)/(580 - 510))^gamma;
      G = 1.0;
      B = 0.0;
      ),
    580 <= wavelength <= 645,
    (
      R = 1.0;
      G = (-(wavelength - 645)/(645 - 580))^gamma;
      B = 0.0;
      ),
    645 <= wavelength <= 750,
    (
      attenuation = 0.3 + 0.7*(750 - wavelength)/(750 - 645);
      R = (1.0*attenuation)^gamma;
      G = 0.0;
      B = 0.0;
      ),
    True,
    (
      R = 0;
      G = 0;
      B = 0;
      )];
    Return[RGBColor[R, G, B]]
    )

FuzzyRectangle::usage = "FuzzyRectangle[xCenter, width, ymin, height,
    color] creates a polygon with a fuzzy edge. The polygon is
  centered at xCenter and has a full horizontal width of width. The
  bottom of the polygon is at ymin and the height is height. The
  color of the polygon is color. The left edge and the right edge of
  the resulting polygon will be transparent and the middle will be
  colored. The polygon is returned as a list of polygons.";
FuzzyRectangle[xCenter_, width_, ymin_, height_, color_, intensity_
  :1] := Module[
  {intenseColor, nocolor, ymax, polys},
(
  nocolor = Directive[Opacity[0], color];
  ymax = ymin + height;
  intenseColor = Directive[Opacity[intensity], color];
  polys = {
    Polygon[{
      {xCenter - width/2, ymin},
      {xCenter, ymin},
      {xCenter, ymax},
      {xCenter - width/2, ymax}},
      VertexColors -> {
        nocolor,
        intenseColor,
        intenseColor,
        nocolor,
        nocolor}],
    Polygon[{
      {xCenter, ymin},
      {xCenter + width/2, ymin},
      {xCenter + width/2, ymax},
      {xCenter, ymax}},
      VertexColors -> {
        intenseColor,
        nocolor,
        nocolor,
        intenseColor,
        intenseColor}]
    };
  Return[polys]
  );
]

Options[SpectrumPlot] = Options[Graphics];
```

```
Options[SpectrumPlot] = Join[Options[SpectrumPlot], {"Intensities" -> {},"Tooltips" -> True, "Comments" -> {}, "SpectrumFunction" -> WaveToRGB}];
SpectrumPlot::usage="SpectrumPlot[lines, widthToHeightAspect, lineWidth] takes a list of spectral lines and creates a visual representation of them. The lines are represented as fuzzy rectangles with a width of lineWidth and a height that is determined by the overall condition that the with to height ratio of the resulting graph is widthToHeightAspect. The color of the lines is determined by the wavelength of the line. The function assumes that the lines are given in nm.
If the lineWidth parameter is a single number, then every line shares that width. If the lineWidth parameter is a list of numbers, then each line has a different width. The function returns a Graphics object. The function also accepts any options that Graphics accepts. The background of the plot is black by default. The plot range is set to the minimum and maximum wavelength of the given lines.
Besides the options for Graphics the function also admits the option Intensities. This option is a list of numbers that determines the intensity of each line. If the Intensities option is not given, then the lines are drawn with full intensity. If the Intensities option is given, then the lines are drawn with the given intensity. The intensity is a number between 0 and 1.
The function also admits the option \"Tooltips\". If this option is set to True, then the lines will have a tooltip that shows the wavelength of the line. If this option is set to False, then the lines will not have a tooltip. The default value for this option is True.
If \"Tooltips\" is set to True and the option \"Comments\" is a non-empty list, then the tooltip will append the wavelength and the values in the comments list for the tooltips.
The function also admits the option \"SpectrumFunction\". This option is a function that takes a wavelength and returns a color. The default value for this option is WaveToRGB.
";
SpectrumPlot[lines_, widthToHeightAspect_, lineWidth_, opts : OptionsPattern[]] := Module[
  {minWave, maxWave, height, fuzzyLines},
(
  colorFun = OptionValue["SpectrumFunction"];
  {minWave, maxWave} = MinMax[lines];
  height     = (maxWave - minWave)/widthToHeightAspect;
  fuzzyLines = Which[
    NumberQ[lineWidth] && Length[OptionValue["Intensities"]] == 0,
      FuzzyRectangle[#, lineWidth, 0, height, colorFun[#]] &/@ lines,
    Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]] == 0,
      MapThread[FuzzyRectangle[#1, #2, 0, height, colorFun[#1]] &, {lines, lineWidth}],
    NumberQ[lineWidth] && Length[OptionValue["Intensities"]] > 0,
      MapThread[FuzzyRectangle[#1, lineWidth, 0, height, colorFun[#1], #2] &, {lines, OptionValue["Intensities"]}],
    Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]] > 0,
      MapThread[FuzzyRectangle[#1, #2, 0, height, colorFun[#1], #3] &, {lines, lineWidth, OptionValue["Intensities"]}]
  ];
  comments = Which[
    Length[OptionValue["Comments"]] > 0,
      MapThread[StringJoin[ToString[#1]<>" nm","\n",ToString[#2]]&, {lines, OptionValue["Comments"]}],
    Length[OptionValue["Comments"]] == 0,
      ToString[#] <>" nm" & /@ lines,
    True,
    {}
  ];
  If[OptionValue["Tooltips"],
    fuzzyLines = MapThread[Tooltip[#1, #2] &, {fuzzyLines, comments}];
  ];
  graphicsOpts = FilterRules[{opts}, Options[Graphics]];
  Graphics[fuzzyLines,
      graphicsOpts,
      Background -> Black,
      PlotRange -> {{minWave, maxWave}, {0, height}}]
  )
```

```
  ];

End[];

EndPackage[];
```

13 misc.m

This module includes a few functions useful for data-handling.

```
BeginPackage["misc`"];

ExportToH5;
FlattenBasis;
RecoverBasis;
FlowMatching;
SuperIdentity;
RobustMissingQ;
ReplaceDiagonal;

GreedyMatching;
HelperNotebook;
StochasticMatching;
ExtractSymbolNames;
GetModificationDate;
TextBasedProgressBar;
ToPythonSparseFunction;

FirstOrderPerturbation;
SecondOrderPerturbation;
RoundValueWithUncertainty;

ToPythonSymPyExpression;
RoundToSignificantFigures;
RobustMissingQ;

Begin["`Private`"];

  ReplaceDiagonal::usage =
    "ReplaceDiagonal[matrix, repValue] replaces all the diagonal of the
     given array to the given value. The array is assumed to be square
    and the replacement value is assumed to be a number. The returned
    value is the array with the diagonal replaced. This function is
    useful for setting the diagonal of an array to a given value. The
    original array is not modified. The given array may be sparse.";
  ReplaceDiagonal[matrix_, repValue_] :=
    ReplacePart[matrix,
      Table[{i, i} -> repValue, {i, 1, Length[matrix]}]];

  Options[RoundValueWithUncertainty] = {"SetPrecision" -> False};
  RoundValueWithUncertainty::usage =
    "RoundValueWithUncertainty[x,dx] given a number x together with an
    \
  uncertainty dx this function rounds x to the first significant figure
    \
  of dx and also rounds dx to have a single significant figure.
  The returned value is a list with the form {roundedX, roundedDx}.
  The option \"SetPrecision\" can be used to control whether the \
  Mathematica precision of x and dx is also set accordingly to these \
  rules, otherwise the rounded numbers still have the original \
  precision of the input values.
  If the position of the first significant figure of x is after the \
  position of the first significant figure of dx, the function returns
    \
  {0,dx} with dx rounded to one significant figure.";
  RoundValueWithUncertainty[x_, dx_, OptionsPattern[]] := Module[
    {xExpo, dxExpo, sigFigs, roundedX, roundedDx, returning},
    (
      xExpo = RealDigits[x][[2]];
      dxExpo = RealDigits[dx][[2]];
      sigFigs = xExpo - dxExpo + 1;
      {roundedX, roundedDx} = If[sigFigs <= 0,
        {0., N@RoundToSignificantFigures[dx, 1]},
        N[
        {
```

```
          RoundToSignificantFigures[x, xExpo - dxExpo + 1],
          RoundToSignificantFigures[dx, 1]}
        ]
        ];
      returning = If[
        OptionValue["SetPrecision"],
        {SetPrecision[roundedX, Max[1, sigFigs]],
        SetPrecision[roundedDx, 1]},
        {roundedX, roundedDx}
        ];
      Return[returning]
      )
    ];

RoundToSignificantFigures::usage =
  "RoundToSignificantFigures[x, sigFigs] rounds x so that it only has \
sigFigs significant figures.";
RoundToSignificantFigures[x_, sigFigs_] :=
  Sign[x]*N[FromDigits[RealDigits[x, 10, sigFigs]]];

RobustMissingQ[expr_] := (FreeQ[expr, _Missing] === False);

TextBasedProgressBar[progress_, totalIterations_, prefix_:""] :=
  Module[
    {progMessage},
    progMessage = ToString[progress] <> "/" <> ToString[
  totalIterations];
    If[progress < totalIterations,
        WriteString["stdout", StringJoin[prefix, progMessage, "\r"]],
        WriteString["stdout", StringJoin[prefix, progMessage, "\n"]]
    ];
];

FirstOrderPerturbation::usage="Given the eigenValues and eigenVectors \
of a matrix A (which doesn't need to be given) together with a \
corresponding perturbation matrix perMatrix, this function \
calculates the first derivative of the eigenvalues with respect to \
the scale factor of the perturbation matrix. In the sense that the \
eigenvalues of the matrix A + β perMatrix are to first order equal \
to \[Lambda] + \[Delta]_i β, where the \[Delta]_i are the returned \
values. The eigenvalues and eigenvectors are assumed to be given in \
the same order, i.e. the ith eigenvalue corresponds to the ith \
eigenvector. This assuming that the eigenvalues are non-degenerate.\
";
FirstOrderPerturbation[eigenValues_, eigenVectors_,
  perMatrix_] := (Diagonal[
  eigenVectors . perMatrix . Transpose[eigenVectors]])

SecondOrderPerturbation::usage="Given the eigenValues and \
eigenVectors of a matrix A (which doesn't need to be given) \
together with a corresponding perturbation matrix perMatrix, this \
function calculates the second derivative of the eigenvalues with \
respect to the scale factor of the perturbation matrix. In the \
sense that the eigenvalues of the matrix A + β perMatrix are to \
second order equal to \[Lambda] + \[Delta]_i β + \[Delta]_i^{(2)}/2 \
β^2, where the \[Delta]_i^{(2)} are the returned values. The \
eigenvalues and eigenvectors are assumed to be given in the same \
order, i.e. the ith eigenvalue corresponds to the ith eigenvector. \
This assuming that the eigenvalues are non-degenerate.";
SecondOrderPerturbation[eigenValues_, eigenVectors_, perMatrix_] := (
  dim = Length[perMatrix];
  eigenBras = Conjugate[eigenVectors];
  eigenKets = eigenVectors;
  matV = Abs[eigenBras . perMatrix . Transpose[eigenKets]]^2;
  OneOver[x_, y_] := If[x == y, 0, 1/(x - y)];
  eigenDiffs = Outer[OneOver, eigenValues, eigenValues, 1];
  pProduct = Transpose[eigenDiffs]*matV;
  Return[2*(Total /@ Transpose[pProduct])];
  )

SuperIdentity::usage="SuperIdentity[args] returns the arguments \
passed to it. This is useful for defining a function that does \
nothing, but that can be used in a composition.";
SuperIdentity[args___] := {args};
```

```
109  FlattenBasis::usage="FlattenBasis[basis] takes a basis in the
       standard representation and separates out the strings that describe
        the LS part of the labels and the additional numbers that define
       the values of J MJ and MI. It returns a list with two elements {
       flatbasisLS, flatbasisNums}. This is useful for saving the basis to
        an h5 file where the strings and numbers need to be separated.";
110  FlattenBasis[basis_] := Module[{flatbasis, flatbasisLS, flatbasisNums
       },
111    (
112      flatbasis = Flatten[basis];
113      flatbasisLS = flatbasis[[1 ;; ;; 4]];
114      flatbasisNums = Select[flatbasis, Not[StringQ[#]] &];
115      Return[{flatbasisLS, flatbasisNums}]
116      )
117    ];
118
119  RecoverBasis::usage="RecoverBasis[{flatBasisLS, flatbasisNums}] takes
         the output of FlattenBasis and returns the original basis. The
       input is a list with two elements {flatbasisLS, flatbasisNums}.";
120  RecoverBasis[{flatbasisLS_, flatbasisNums_}] := Module[{recBasis},
121    (
122    recBasis = {{{#[[1]], #[[2]]}, #[[3]]}, #[[4]]} & /@ (Flatten /@
123        Transpose[{flatbasisLS,
124          Partition[Round[2*#]/2 & /@ flatbasisNums, 3]}]);
125    Return[recBasis];
126    )
127    ]
128
129  ExtractSymbolNames[expr_Hold] := Module[
130    {strSymbols},
131    strSymbols = ToString[expr, InputForm];
132    StringCases[strSymbols, RegularExpression["\\w+"]][[2 ;;]]
133    ]
134
135  ExportToH5::usage =
136    "ExportToH5[fname, Hold[{symbol1, symbol2, ...}]] takes an .h5
       filename and a held list of symbols and export to the .h5 file the
       values of the symbols with keys equal the symbol names. The values
       of the symbols cannot be arbitrary, for instance a list with mixes
       numbers and string will fail, but an Association with mixed values
       exports ok. Do give it a try.
137    If the file is already present in disk, this function will
       overwrite it by default. If the value of a given symbol contains
       symbolic numbers, e.g. \[Pi], these will be converted to floats in
       the exported file.";
138  Options[ExportToH5] = {"Overwrite" -> True};
139  ExportToH5[fname_String, symbols_Hold, OptionsPattern[]] := (
140    If[And[FileExistsQ[fname], OptionValue["Overwrite"]],
141    (
142      Print["File already exists, overwriting ..."];
143      DeleteFile[fname];
144      )
145    ];
146    symbolNames = ExtractSymbolNames[symbols];
147    Do[(Print[symbolName];
148      Export[fname, ToExpression[symbolName], {"Datasets", symbolName},
149      OverwriteTarget -> "Append"]
150      ), {symbolName, symbolNames}]
151    )
152
153  GreedyMatching::usage="GreedyMatching[aList, bList] returns a list of
         pairs of elements from aList and bList that are closest to each
       other, this is returned in a list together with a mapping of
       indices from the aList to those in bList to which they were matched
       . The option \"alistLabels\" can be used to specify labels for the
       elements in aList. The option \"blistLabels\" can be used to
       specify labels for the elements in bList. If these options are used
       , the function returns a list with three elements the pairs of
       matched elements, the pairs of corresponding matched labels, and
       the mapping of indices.";
154  Options[GreedyMatching] = {
155      "alistLabels" -> {},
156      "blistLabels" -> {}};
157  GreedyMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{
158    aValues = aValues0,
159    bValues = bValues0,
```

```
    bValuesOriginal = bValues0,
    bestLabels, bestMatches,
    bestLabel, aElement, givenLabels,
    aLabels, aLabel,
    diffs, minDiff,
    bLabels,
    minDiffPosition, bestMatch},
    (
    aLabels     = OptionValue["alistLabels"];
    bLabels     = OptionValue["blistLabels"];
    bestMatches = {};
    bestLabels  = {};
    givenLabels = (Length[aLabels] > 0);
    Do[
      (
      aElement         = aValues[[idx]];
      diffs            = Abs[bValues - aElement];
      minDiff          = Min[diffs];
      minDiffPosition = Position[diffs, minDiff][[1, 1]];
      bestMatch        = bValues[[minDiffPosition]];
      bestMatches      = Append[bestMatches, {aElement, bestMatch}];
      If[givenLabels,
        (
        aLabel      = aLabels[[idx]];
        bestLabel   = bLabels[[minDiffPosition]];
        bestLabels = Append[bestLabels, {aLabel, bestLabel}];
        bLabels     = Drop[bLabels, {minDiffPosition}];
        )
        ];
      bValues = Drop[bValues, {minDiffPosition}];
      If[Length[bValues] == 0, Break[]];
      ),
      {idx, 1, Length[aValues]}
      ];
    pairedIndices = MapIndexed[{#2[[1]], Position[bValuesOriginal,
    #1[[2]]][[1, 1]]} &, bestMatches];
    If[givenLabels,
      Return[{bestMatches, bestLabels, pairedIndices}],
      Return[{bestMatches, pairedIndices}]
      ]
    )
    ]

  StochasticMatching::usage="StochasticMatching[aValues, bValues] finds
     a better assignment by randomly shuffling the elements of aValues
    and then applying the greedy assignment algorithm. The function
    prints what is the range of total absolute differences found during
     shuffling, the standard deviation of all of them, and the number
    of shuffles that were attempted. The option \"alistLabels\" can be
    used to specify labels for the elements in aValues. The option \"
    blistLabels\" can be used to specify labels for the elements in
    bValues. If these options are used, the function returns a list
    with three elements the pairs of matched elements, the pairs of
    corresponding matched labels, and the mapping of indices.";
  Options[StochasticMatching] = {"alistLabels" -> {},
    "blistLabels" -> {}};
  StochasticMatching[aValues0_, bValues0_, numShuffles_ : 200,
    OptionsPattern[]] := Module[{
    aValues = aValues0,
    bValues = bValues0,
    matchingLabels, ranger, matches, noShuff, bestMatch, highestCost,
    lowestCost, dev, sorter, bestValues,
    pairedIndices, bestLabels, matchedIndices, shuffler
    },
    (
    matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
    ranger = Range[1, Length[aValues]];
    matches = If[Not[matchingLabels], (
        Table[(
          shuffler = If[i == 1, ranger, RandomSample[ranger]];
          {bestValues, matchedIndices} =
          GreedyMatching[aValues[[shuffler]], bValues];
          cost = Total[Abs[#[[1]] - #[[2]]] & /@ bestValues];
          {cost, {bestValues, matchedIndices}}
          ), {i, 1, numShuffles}]
        ),
```

```
      Table[(
        shuffler = If[i == 1, ranger, RandomSample[ranger]];
        {bestValues, bestLabels, matchedIndices} =
          GreedyMatching[aValues[[shuffler]], bValues,
          "alistLabels" -> OptionValue["alistLabels"][[shuffler]],
          "blistLabels" -> OptionValue["blistLabels"]];
        cost = Total[Abs[#[[1]] - #[[2]]] & /@ bestValues];
        {cost, {bestValues, bestLabels, matchedIndices}}
        ), {i, 1, numShuffles}]
      ];
    noShuff = matches[[1, 1]];
    matches = SortBy[matches, First];
    bestMatch = matches[[1, 2]];
    highestCost = matches[[-1, 1]];
    lowestCost = matches[[1, 1]];
    dev = StandardDeviation[First /@ matches];
    Print[lowestCost, " <-> ", highestCost, " | \[Sigma]=", dev,
      " | N=", numShuffles, " | null=", noShuff];
    If[matchingLabels,
      (
      {bestValues, bestLabels, matchedIndices} = bestMatch;
      sorter = Ordering[First /@ bestValues];
      bestValues = bestValues[[sorter]];
      bestLabels = bestLabels[[sorter]];
      pairedIndices =
        MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
        bestValues];
      Return[{bestValues, bestLabels, pairedIndices}]
      ),
      (
      {bestValues, matchedIndices} = bestMatch;
      sorter = Ordering[First /@ bestValues];
      bestValues = bestValues[[sorter]];
      pairedIndices =
        MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
        bestValues];
      Return[{bestValues, pairedIndices}]
      )
      ];
    )
    ]

  FlowMatching::usage="FlowMatching[aList, bList] returns a list of
    pairs of elements from aList and bList that are closest to each
    other, this is returned in a list together with a mapping of
    indices from the aList to those in bList to which they were matched
    . The option \"alistLabels\" can be used to specify labels for the
    elements in aList. The option \"blistLabels\" can be used to
    specify labels for the elements in bList. If these options are used
    , the function returns a list with three elements the pairs of
    matched elements, the pairs of corresponding matched labels, and
    the mapping of indices. This is basically a wrapper around
    Mathematica's FindMinimumCostFlow function. By default the option \
    "noMatched\" is zero, and this means that all elements of aList
    must be matched to elements of bList. If this is not the case, the
    option \"noMatched\" can be used to specify how many elements of
    aList can be left unmatched. By default the cost function is Abs
    [#1-#2]&, but this can be changed with the option \"CostFun\", this
     function needs to take two arguments.";
  Options[FlowMatching] = {"alistLabels" -> {}, "blistLabels" -> {}, "
    notMatched" -> 0, "CostFun"-> (Abs[#1-#2] &)};
  FlowMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{
    aValues = aValues0, bValues = bValues0, edgesSourceToA,
    capacitySourceToA, nA, nB,
    costSourceToA, midLayer, midLayerEdges, midCapacities,
    midCosts, edgesBtoSink, capacityBtoSink, costBtoSink,
    allCapacities, allCosts, allEdges, graph,
    flow, bestValues, bestLabels, cFun,
    aLabels, bLabels, pairedIndices, matchingLabels},
    (
    matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
    aLabels = OptionValue["alistLabels"];
    bLabels = OptionValue["blistLabels"];
    cFun = OptionValue["CostFun"];
    nA      = Length[aValues];
    nB      = Length[bValues];
```

```
    (*Build up the edges costs and capacities*)
    (*From source to the nodes representing the values of the first \
  list*)
    edgesSourceToA = ("source" \[DirectedEdge] {"A", #}) & /@ Range[1,
    nA];
    capacitySourceToA = ConstantArray[1, nA];
    costSourceToA = ConstantArray[0, nA];

    (*From all the elements of A to all the elements of B*)
    midLayer = Table[{{"A", i} \[DirectedEdge] ({"B", j}), 1, cFun[
    aValues[[i]], bValues[[j]]]}, {i, 1, nA}, {j, 1, nB}];
    midLayer = Flatten[midLayer, 1];
    {midLayerEdges, midCapacities, midCosts} = Transpose[midLayer];

    (*From the elements of B to the sink*)
    edgesBtoSink = ({"B", #} \[DirectedEdge] "sink") & /@ Range[1, nB];
    capacityBtoSink = ConstantArray[1, nB];
    costBtoSink = ConstantArray[0, nB];

    (*Put it all together*)
    allCapacities = Join[capacitySourceToA, midCapacities,
    capacityBtoSink];
    allCosts = Join[costSourceToA, midCosts, costBtoSink];
    allEdges = Join[edgesSourceToA, midLayerEdges, edgesBtoSink];
    graph = Graph[allEdges, EdgeCapacity -> allCapacities,
      EdgeCost -> allCosts];

    (*Solve it*)
    flow = FindMinimumCostFlow[graph, "source", "sink", nA -
    OptionValue["notMatched"], "OptimumFlowData"];
    (*Collect the pairs of matched indices*)
    pairedIndices = Select[flow["EdgeList"], And[Not[#[[1]] === "source
    "], Not[#[[2]] === "sink"]] &];
    pairedIndices = {#[[1, 2]], #[[2, 2]]} & /@ pairedIndices;
    (*Collect the pairs of matched values*)
    bestValues = {aValues[[#[[1]]]], bValues[[#[[2]]]]} & /@
    pairedIndices;
    (*Account for having been given labels*)
    If[matchingLabels,
      (
      bestLabels = {aLabels[[#[[1]]]], bLabels[[#[[2]]]]} & /@
    pairedIndices;
      Return[{bestValues, bestLabels, pairedIndices}]
      ),
      (
      Return[{bestValues, pairedIndices}]
      )
      ];
    )
    ]

  HelperNotebook::usage="HelperNotebook[nbName] creates a separate
    notebook and returns a function that can be used to print to the
    bottom of it. The name of the notebook, nbName, is optional and
    defaults to OUT.";
  HelperNotebook[nbName_:"OUT"] :=
  Module[{screenDims, screenWidth, screenHeight, nbWidth, leftMargin,
    PrintToOutputNb}, (
    screenDims =
      SystemInformation["Devices", "ScreenInformation"][[1, 2, 2]];
    screenWidth = screenDims[[1, 2]];
    screenHeight = screenDims[[2, 2]];
    nbWidth = Round[screenWidth/3];
    leftMargin = screenWidth - nbWidth;
    outputNb = CreateDocument[{}, WindowTitle -> nbName,
      WindowMargins -> {{leftMargin, Automatic}, {Automatic,
          Automatic}}, WindowSize -> {nbWidth, screenHeight}];
    PrintToOutputNb[text_] :=
      (
          SelectionMove[outputNb, After, Notebook];
          NotebookWrite[outputNb, Cell[BoxData[ToBoxes[text]], "Output"
    ]];
      );
    Return[PrintToOutputNb]
    )
    ]
```

```
346
347   GetModificationDate::usage="GetModificationDate[fname] returns the
        modification date of the given file.";
348   GetModificationDate[theFileName_] := FileDate[theFileName, "
        Modification"];
349
350   (*Helper function to convert Mathematica expressions to standard form
        *)
351   StandardFormExpression[expr0_] := Module[{expr=expr0}, ToString[expr,
         InputForm]];
352
353   (*Helper function to translate to Python/SymPy expressions*)
354   ToPythonSymPyExpression::usage="ToPythonSymPyExpression[expr]
        converts a Mathematica expression to a SymPy expression. This is a
        little iffy and might break if the expression includes Mathematica
        functions that haven't been given a SymPy equivalent.";
355   ToPythonSymPyExpression[expr0_] := Module[{standardForm, expr=expr0},
356     standardForm = StandardFormExpression[expr];
357     StringReplace[standardForm, {
358       "Power[" -> "Pow(",
359       "Sqrt[" -> "sqrt(",
360       "[" -> "(",
361       "]" -> ")",
362       "\\" -> "",
363       (*Remove special Mathematica backslashes*)
364       "/" -> "/" (*Ensure division is represented with a slash*)}]];
365
366   ToPythonSparseFunction[sparseArray_SparseArray, funName_] :=
367     Module[{data, rowPointers, columnIndices, dimensions, pyCode, vars,
368       varList, dataPyList,
369       colIndicesPyList},(*Extract unique symbolic variables from the \
370   SparseArray*)
371     vars = Union[Cases[Normal[sparseArray], _Symbol, Infinity]];
372     varList = StringRiffle[ToString /@ vars, ", "];
373     (*varList=ToPythonSymPyExpression/@varList;*)
374     (*Convert data to SymPy compatible strings*)
375     dataPyList =
376       StringRiffle[
377       ToPythonSymPyExpression /@ Normal[sparseArray["NonzeroValues"]],
378       ", "];
379     colIndicesPyList =
380       StringRiffle[
381       ToPythonSymPyExpression /@ (Flatten[
382           Normal[sparseArray["ColumnIndices"]] - 1]), ", "];
383     (*Extract sparse array properties*)
384     rowPointers = Normal[sparseArray["RowPointers"]];
385     dimensions = Dimensions[sparseArray];
386     (*Create Python code string*)pyCode = StringJoin[
387       "#!/usr/bin/env python3\n\n",
388       "from scipy.sparse import csr_matrix\n",
389       "from sympy import *\n",
390       "import numpy as np\n",
391       "\n",
392       "sqrt = np.sqrt\n",
393       "\n",
394       "def ", funName, "(",
395       varList,
396       "):\n",
397       "    data = np.array([", dataPyList, "])\n",
398       "    indices = np.array([",
399       colIndicesPyList,
400       "])\n",
401       "    indptr = np.array([",
402       StringRiffle[ToString /@ rowPointers, ", "], "])\n",
403       "    shape = (", StringRiffle[ToString /@ dimensions, ", "],
404       ")\n",
405       "    return csr_matrix((data, indices, indptr), shape=shape)"];
406     pyCode
407     ];
408
409 End[];
410 EndPackage[];
```

14 qalculations.m

This script encapsulates example calculations in which the level structure and magnetic dipole transitions are calculated for the lanthanide ions in lanthanum flouride.

```
Needs["qlanth`"];
Needs["misc`"];
Needs["qplotter`"];
Needs["qonstants`"]
LoadCarnall[];

workDir = DirectoryName[$InputFileName];

FastIonSolverLaF3::usage = "This function solves the energy levels of
    the given trivalent lanthanide in LaF3. The values for the
    Hamiltonian are simply taken from the values quoted by Carnall. It
    uses precomputed symbolic matrices for the Hamiltonian so it's
    faster than the previous alternatives.

The function returns a list with nine elements
{rmsDifference, carnallEnergies, eigenEnergies, ln, carnallAssignments,
     simplerStateLabels, eigensys, basis, truncatedStates}.

Where:

1. rmsDifference is the root mean squared difference between the
    calculated values and those quoted by Carnall

2. carnallEnergies are the quoted calculated energies from Carnall;

3. eigenEnergies are the calculated energies (in the case of an odd
    number of electrons the Kramers degeneracy may have been removed
    from this list according to the option \"Remove Kramers\");

4. ln is simply a string labelling the corresponding lanthanide;

5. carnallAssignments is a list of strings providing the multiplet
    assignments that Carnall assumed;

6. simplerStateLabels is a list of strings providing the multiplet
    assignments that this function assumes;

7. eigensys is a list of tuples where the first element is the energy
    corresponding to the eigenvector given as the second element (in
    the case of an odd number of electrons the Kramers degeneracy may
    have been removed from this list according to the option \"Remove
    Kramers\");

8. basis is a list that specifies the basis in which the Hamiltonian
    was constructed and diagonalized, equal to BasisLSJMJ[numE];

9. Same as eigensys but the eigenvectors have been truncated so that
    the truncated version adds up to at least a total probability of
    eigenstateTruncationProbability.

";
Options[FastIonSolverLaF3] = {
  "MakeNotebook" -> True,
  "NotebookSave" -> True,
  "HTMLSave" -> False,
  "eigenstateTruncationProbability" -> 0.9,
  "Include spin-spin" -> True,
  "Max Eigenstates in Table" -> 100,
  "Sparse" -> True,
  "PrintFun" -> Print,
  "SaveData" -> True,
  "paramFiddle" -> {},
  "Append to Filename" -> "",
  "Remove Kramers" -> True,
  "OutputDirectory" -> "calcs",
  "Explorer" -> False
};
FastIonSolverLaF3[numE_, OptionsPattern[]] := Module[
  {makeNotebook, eigenstateTruncationProbability, host,
  ln, terms, termNames, carnallEnergies, eigenEnergies,
    simplerStateLabels,
  eigensys, basis, assignmentMatches, stateLabels, carnallAssignments},
```

```
(
  PrintFun = OptionValue["PrintFun"];
  makeNotebook = OptionValue["MakeNotebook"];
  eigenstateTruncationProbability = OptionValue["
eigenstateTruncationProbability"];
  maxStatesInTable = OptionValue["Max Eigenstates in Table"];
  Duplicator[aList_] := Flatten[{#, #} & /@ aList];
  host = "LaF3";
  paramFiddle = OptionValue["paramFiddle"];
  ln = theLanthanides[[numE]];
  terms = AllowedNKSLJTerms[Min[numE, 14 - numE]];
  termNames = First /@ terms;
  (* For labeling the states, the degeneracy in some of the terms is
elided *)
  PrintFun["> Calculating simpler term labels ..."];
  termSimplifier = Table[termN -> If[StringLength[termN] == 3,
    StringTake[termN, {1, 2}],
    termN
    ],
    {termN, termNames}
  ];

  (*Load the parameters from Carnall*)
  PrintFun["> Loading the fit parameters from Carnall ..."];
  params = LoadParameters[ln, "Free Ion" -> False];
  If[numE>7,
    (
      PrintFun["> Conjugating the parameters accounting for the hole-
particle equivalence ..."];
      params = HoleElectronConjugation[params];
      params[t2Switch] = 0;
    ),
    params[t2Switch] = 1;
  ];

  Do[params[key] = paramFiddle[key],
    {key, Keys[paramFiddle]}
  ];

  (* Import the symbolic Hamiltonian *)
  PrintFun["> Loading the symbolic Hamiltonian for this configuration
 ..."];
  startTime = Now;
  numH = 14 - numE;
  numEH = Min[numE, numH];
  C2vsimplifier = {B12 -> 0, B14 -> 0, B16 -> 0, B34 -> 0, B36 -> 0,
    B56 -> 0,
    S12 -> 0, S14 -> 0, S16 -> 0, S22 -> 0, S24 -> 0, S26 -> 0,
    S34 -> 0, S36 -> 0,
    S44 -> 0, S46 -> 0, S56 -> 0, S66 -> 0, T11p -> 0, T11 -> 0,
    T12 -> 0, T14 -> 0, T15 -> 0,
    T16 -> 0, T18 -> 0, T17 -> 0, T19 -> 0};
  simpleHam = If[
    ValueQ[symbolicHamiltonians[numEH]],
    symbolicHamiltonians[numEH],
    SimplerSymbolicHamMatrix[numE, C2vsimplifier, "PrependToFilename"
 -> "C2v-", "Overwrite" -> False]
  ];
  endTime  = Now;
  loadTime = QuantityMagnitude[endTime - startTime, "Seconds"];
  PrintFun[">> Loading the symbolic Hamiltonian took ", loadTime, "
seconds."];

  (*Enforce the override to the spin-spin contribution to the
magnetic interactions*)
  params[\[Sigma]SS] = If[OptionValue["Include spin-spin"], 1, 0];

  (*Everything that is not given is set to zero*)
  params = ParamPad[params, "Print" -> False];
  PrintFun[params];
  (* numHam = simpleHam /. params; *)
  numHam = ReplaceInSparseArray[simpleHam, params];
  If[Not[OptionValue["Sparse"]],
      numHam = Normal[numHam]
  ];
  PrintFun["> Calculating the SLJ basis ..."];
```

```
    basis = BasisLSJMJ[numE];

    (* Eigensolver *)
    PrintFun["> Diagonalizing the numerical Hamiltonian ..."];
    startTime = Now;
    eigensys  = Eigensystem[numHam];
    endTime   = Now;
    diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
    PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
    eigensys = Chop[eigensys];
    eigensys = Transpose[eigensys];

    (*Shift the baseline energy*)
    eigensys = ShiftedLevels[eigensys];
    (*Sort according to energy*)
    eigensys = SortBy[eigensys, First];
    (*Grab just the energies*)
    eigenEnergies = First /@ eigensys;

    (*Energies are doubly degenerate in the case of odd number of
    electrons, keep only one*)
    If[And[OddQ[numE], OptionValue["Remove Kramers"]],
        (
        PrintFun["> Since there's an odd number of electrons energies
    come in pairs, taking just one for each pair ..."];
        eigenEnergies = eigenEnergies[[;; ;; 2]];
        )
    ];

    (*Compare against the data quoted by Bill Carnall*)
    PrintFun["> Comparing against the data from Carnall ..."];
    mainKey            = StringTemplate["appendix:`Ln`:Association"][<|
    "Ln" -> ln|>];
    lnData             = Carnall[mainKey];
    carnalKeys         = lnData // Keys;
    repetitions        = Length[lnData[#]["Calc (1/cm)"]] & /@
    carnalKeys;
    carnallAssignments = First /@ Carnall["appendix:" <> ln <> ":
    RawTable"];
    carnalKey          = StringTemplate["appendix:`Ln`:Calculated"][<|"
    Ln" -> ln|>];
    carnallEnergies    = Carnall[carnalKey];
    If[And[OddQ[numE], Not[OptionValue["Remove Kramers"]]],
    (
      PrintFun[">> The number of eigenstates and the number of quoted
    states don't match, removing the last state ..."];
      carnallAssignments = Duplicator[carnallAssignments];
      carnallEnergies    = Duplicator[carnallEnergies];
    )
    ];

    (* For the difference take as many energies as quoted by Bill*)
    eigenEnergies = eigenEnergies + carnallEnergies[[1]];
    diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] -
    carnallEnergies;
    (* Remove the differences where the appendix tables have elided
    values*)
    rmsDifference = Sqrt[Mean[(Select[diffs, FreeQ[#, Missing[]] &])
    ^2]];
    titleTemplate = StringTemplate[
        "Energy Level Diagram of \!\(\*SuperscriptBox[\(`ion`\),
    \(\(3\)\(+\)\)]\)"];
    title = titleTemplate[<|"ion" -> ln|>];
    parsedStates = ParseStates[eigensys, basis];
    If[And[OddQ[numE],OptionValue["Remove Kramers"]],
      parsedStates = parsedStates[[;; ;; 2]]
    ];

    stateLabels = #[[-1]] & /@ parsedStates;
    simplerStateLabels = ((#[[2]] /. termSimplifier) <> ToString
    [#[[3]], InputForm]) & /@ parsedStates;

    PrintFun[">> Truncating eigenvectors to given probability ..."];
    startTime = Now;
    truncatedStates = ParseStatesByProbabilitySum[eigensys, basis,
      eigenstateTruncationProbability,
```

```
      0.01];
    endTime = Now;
    truncationTime = QuantityMagnitude[endTime - startTime, "Seconds"];
    PrintFun[">>> Truncation took ", truncationTime, " seconds."];

    If[makeNotebook,
    (
      PrintFun["> Putting together results in a notebook ..."];
      energyDiagram = Framed[
        EnergyLevelDiagram[eigensys, "Title" -> title,
        "Explorer" -> OptionValue["Explorer"],
        "Background" -> White]
        , Background -> White, FrameMargins -> 50];
      appToFname = OptionValue["Append to Filename"];
      PrintFun[">> Comparing the term assignments between qlanth and
    Carnall ..."];
      assignmentMatches =
      If[StringContainsQ[#[[1]], #[[2]]], "\[Checkmark]", "X"] & /@
        Transpose[{carnallAssignments, simplerStateLabels[[;; Length[
    carnallAssignments]]]}];
      assignmentMatches = {{"\[Checkmark]",
        Count[assignmentMatches, "\[Checkmark]"]}, {"X",
        Count[assignmentMatches, "X"]}};
      labelComparison = (If[StringContainsQ[#[[1]], #[[2]]], "\[
    Checkmark]", "X"] & /@
        Transpose[{carnallAssignments,
        simplerStateLabels[[;; Length[carnallAssignments]]]}]);
      labelComparison =
      PadRight[labelComparison, Length[simplerStateLabels], "-"];

      statesTable = Grid[Prepend[{Round[#[[1]]], #[[2]]} & /@
          truncatedStates[[;;Min[Length[eigensys],maxStatesInTable]]],
    {"Energy/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)",
          "\[Psi]"}], Frame -> All, Spacings -> {2, 2},
          FrameStyle -> Blue,
          Dividers -> {{False, True, False}, {True, True}}];
      DefaultIfMissing[expr_]:= If[FreeQ[expr, Missing[]], expr,"NA"];
      PrintFun[">> Rounding the energy differences for table
    presentation ..."];
      roundedDiffs = Round[diffs, 0.1];
      roundedDiffs = PadRight[roundedDiffs, Length[simplerStateLabels],
     "-"];
      roundedDiffs = DefaultIfMissing /@ roundedDiffs;
      diffs = PadRight[diffs, Length[simplerStateLabels], "-"];
      diffs = DefaultIfMissing /@ diffs;
      diffTableData = Transpose[{simplerStateLabels, eigenEnergies,
        labelComparison,
        PadRight[carnallAssignments, Length[simplerStateLabels], "-"],
        DefaultIfMissing/@PadRight[carnallEnergies, Length[
    simplerStateLabels], "-"],
        roundedDiffs}
      ];
      diffTable = TableForm[diffTableData,
        TableHeadings -> {None, {"qlanth",
        "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)", "", "Carnall",
        "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)",
        "\[CapitalDelta]E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"}}
      ];

      diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] -
    carnallEnergies;
      notBad = FreeQ[#,Missing[]]&/@diffs;
      diffs = Pick[diffs,notBad];
      diffHistogram = Histogram[diffs,
        Frame -> True,
        ImageSize -> 800,
        AspectRatio -> 1/3, FrameStyle -> Directive[16],
        FrameLabel -> {"(qlanth-carnall)/Ky", "Freq"}
      ];
      rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
      labelTempate = StringTemplate["\!\(\*SuperscriptBox[\(`ln`\),
    \(\(3\)\(+\)\)]\)"];
      diffData   = diffs;
      diffLabels = simplerStateLabels[[;;Length[notBad]]];
      diffLabels = Pick[diffLabels, notBad];
      diffPlot   = Framed[
```

```
        ListLabelPlot[diffData,
        diffLabels,
        Frame -> True,
        PlotRange -> All,
        ImageSize -> 1200,
        AspectRatio -> 1/3,
        FrameLabel -> {"",
        "(qlanth-carnall) / \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"},
        PlotMarkers -> "OpenMarkers",
        PlotLabel ->
        Style[labelTempate[<|"ln" -> ln|>] <> " | " <> "\[Sigma]=" <>
            ToString[Round[rmsDifference, 0.01]] <>
            " \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)\n", 20],
        Background -> White
        ],
        Background -> White,
        FrameMargins -> 50
      ];
      (* now place all of this in a new notebook *)
      nb = CreateDocument[
      {
        TextCell[Style[
          DisplayForm[RowBox[{SuperscriptBox[host <> ":" <> ln, "3+"],
"(", SuperscriptBox["f", numE], ")"}]]
        ], "Title", TextAlignment -> Center
        ],
        TextCell["Energy Diagram",
          "Section",
          TextAlignment -> Center
        ],
        TextCell[energyDiagram,
          TextAlignment -> Center
        ],
        TextCell["Multiplet Assignments & Energy Levels",
          "Section",
          TextAlignment -> Center
        ],
        TextCell[diffHistogram, TextAlignment -> Center],
        TextCell[diffPlot, "Output", TextAlignment -> Center],
        TextCell[assignmentMatches, "Output", TextAlignment -> Center],
        TextCell[diffTable, "Output", TextAlignment -> Center],
        TextCell["Truncated Eigenstates", "Section", TextAlignment ->
Center],
        TextCell["These are some of the resultant eigenstates which add
 up to at least a total probability of " <> ToString[
eigenstateTruncationProbability] <> ".", "Text", TextAlignment ->
Center],
        TextCell[statesTable, "Output", TextAlignment -> Center]
      },
      WindowSelected -> True,
      WindowTitle -> ln <> " in " <> "LaF3" <> appToFname,
      WindowSize -> {1600, 800}];
      If[OptionValue["SaveData"],
      (
        exportFname = FileNameJoin[{workDir,OptionValue["
OutputDirectory"], ln <> " in " <> "LaF3" <> appToFname <> ".m"}];
        SelectionMove[nb, After, Notebook];
        NotebookWrite[nb, Cell["Reload Data", "Section", TextAlignment
-> Center]];
        NotebookWrite[nb,
          Cell[(
            "{rmsDifference, carnallEnergies, eigenEnergies, ln,
carnallAssignments, simplerStateLabels, eigensys, basis,
truncatedStates} = Import[FileNameJoin[{NotebookDirectory[],\"" <>
StringSplit[exportFname,"/"][[-1]] <> "\"}]];"
          ),"Input"
          ]
        ];
        NotebookWrite[nb,
          Cell[(
            "Manipulate[First[MinimalBy[truncatedStates, Abs[First[#] -
 energy] &]], {energy,0}]"
          ),"Input"]
        ];
        (* Move the cursor to the top of the notebook *)
        SelectionMove[nb, Before, Notebook];
```

```
        Export[exportFname,
          {rmsDifference, carnallEnergies, eigenEnergies, ln,
    carnallAssignments, simplerStateLabels, eigensys, basis,
    truncatedStates}
        ];
        tinyexportFname = FileNameJoin[
          {workDir, OptionValue["OutputDirectory"], ln <> " in " <> "
    LaF3" <> appToFname <> " - tiny.m"}
        ];
        tinyExport = <|"ln"->ln,
                       "carnallEnergies"->carnallEnergies,
                       "rmsDifference"-> rmsDifference,
                       "eigenEnergies"-> eigenEnergies,
                       "carnallAssignments"-> carnallAssignments,
                       "simplerStateLabels" -> simplerStateLabels|>;
        Export[tinyexportFname, tinyExport];
      )
      ];
      If[OptionValue["NotebookSave"],
        (
          nbFname = FileNameJoin[{workDir,OptionValue["OutputDirectory"
    ], ln <> " in " <> "LaF3" <> appToFname <> ".nb"}];
          PrintFun[">> Saving notebook to ", nbFname, " ..."];
          NotebookSave[nb, nbFname];
        )
      ];
      If[OptionValue["HTMLSave"],
      (
        htmlFname = FileNameJoin[{workDir,OptionValue["OutputDirectory"
    ], "html", ln <> " in " <> "LaF3" <> appToFname <> ".html"}];
        PrintFun[">> Saving html version to ", htmlFname, " ..."];
        Export[htmlFname, nb];
      )
      ];
    )
    ];

    Return[{rmsDifference, carnallEnergies, eigenEnergies, ln,
    carnallAssignments, simplerStateLabels, eigensys, basis,
    truncatedStates}];
  )
];

MagneticDipoleTransitions::usage = "MagneticDipoleTransitions[numE]
    calculates the magnetic dipole transitions for the lanthanide ion
    numE in LaF3. The output is a tabular file, a raw data file, and a
    CSV file. The tabular file contains the following columns:
    \[Psi]i:simple, (* main contribution to the wavefuction |i>*)
    \[Psi]f:simple, (* main contribution to the wavefuction |j>*)
    \[Psi]i:idx,    (* index of the wavefuction |i>*)
    \[Psi]f:idx,    (* index of the wavefuction |j>*)
    Ei/K,           (* energy of the initial state in K *)
    Ef/K,           (* energy of the final state in K *)
    \[Lambda]/nm,   (* transition wavelength in nm *)
    \[CapitalDelta]\[Lambda]/nm, (* uncertainty in the transition
    wavelength in nm *)
    \[Tau]/s,       (* radiative lifetime in s *)
    AMD/s^-1        (* magnetic dipole transition rate in s^-1 *)

The raw data file contains the following keys:
    - Line Strength, (* Line strength array *)
    - AMD, (* Magnetic dipole transition rates in 1/s *)
    - fMD, (* Oscillator strengths from ground to excited states *)
    - Radiative lifetimes, (* Radiative lifetimes in s *)
    - Transition Energies / K, (* Transition energies in K *)
    - Transition Wavelengths in nm. (* Transition wavelengths in nm *)

The CSV file contains the same information as the tabular file.

The function also creates a notebook with a Manipulate that allows the
    user to select a wavelength interval and a lifetime power of ten.
    The results notebook is saved in the examples directory.

The function takes the following options:
    - \"Make Notebook\" -> True or False. If True, a notebook with a
    Manipulate is created. Default is True.
```

```
382     - \"Print Function\" -> PrintTemporary or Print. The function used
        to print the progress of the calculation. Default is PrintTemporary
        .
383     - \"Host\" -> \"LaF3\". The host material. Default is LaF3.
384     - \"Wavelength Range\" -> {50,2000}. The range of wavelengths in nm
         for the Manipulate object in the created notebook. Default is
        {50,2000}.
385
386 The function returns an association containing the following keys: Line
         Strength, AMD, fMD, Radiative lifetimes, Transition Energies / K,
        Transition Wavelengths in nm.";
387 Options[MagneticDipoleTransitions] = {
388         "Make Notebook" -> True,
389         "Close Notebook" -> True,
390         "Print Function" -> PrintTemporary,
391         "Host" -> "LaF3",
392         "Wavelength Range" -> {50,2000}};
393 MagneticDipoleTransitions[numE_Integer, OptionsPattern[]]:= (
394   host            = OptionValue["Host"];
395   \[Lambda]Range  = OptionValue["Wavelength Range"];
396   PrintFun        = OptionValue["Print Function"];
397   {\[Lambda]min, \[Lambda]max} = OptionValue["Wavelength Range"];
398
399   header     = {"\[Psi]i:simple","\[Psi]f:simple","\[Psi]i:idx","\[Psi]f
        :idx","Ei/K","Ef/K","\[Lambda]/nm","\[CapitalDelta]\[Lambda]/nm","
        \[Tau]/s","AMD/s^-1"};
400   ln         = {"Ce","Pr","Nd","Pm","Sm","Eu","Gd","Tb","Dy","Ho","Er","
        Tm","Yb"}[[numE]];
401   {rmsDifference,carnallEnergies,eigenEnergies,ln,
402   carnallAssignments,simplerStateLabels,eigensys,basis,truncatedStates}
        = Import["./examples/"<>ln<>" in LaF3 - example.m"];
403
404   (* Some of the above are not needed here *)
405   Clear[truncatedStates];
406   Clear[basis];
407   Clear[rmsDifference];
408   Clear[carnallEnergies];
409   Clear[carnallAssignments];
410   If[OddQ[numE],
411       eigenEnergies      = eigenEnergies[[;;;;2]];
412       simplerStateLabels = simplerStateLabels[[;;;;2]];
413       eigensys           = eigensys[[;;;;2]];
414   ];
415   eigenEnergies = eigenEnergies - eigenEnergies[[1]];
416
417   magIon = <||>;
418   PrintFun["Calculating the magnetic dipole line strength array..."];
419   magIon["Line Strength"] = magIon;MagDipLineStrength[eigensys, numE, "
        Reload MagOp" -> False, "Units" -> "SI"];
420
421   PrintFun["Calculating the M1 spontaneous transition rates ..."];
422   magIon["AMD"] = MagDipoleRates[eigensys, numE, "Units"->"SI","
        Lifetime"->False];
423   magIon["AMD"] = magIon["AMD"]/.{0.->Indeterminate};
424
425   PrintFun["Calculating the oscillator strength for transition from the
         ground state ..."];
426   magIon["fMD"]=GroundStateOscillatorStrength[eigensys, numE];
427
428   PrintFun["Calculating the natural radiative lifetims ..."];
429   magIon["Radiative lifetimes"]= 1/magIon["AMD"];
430
431   PrintFun["Calculating the transition energies in K ..."];
432   transitionEnergies=Outer[Subtract,First/@eigensys,First/@eigensys];
433   magIon["Transition Energies / K"]=ReplaceDiagonal[transitionEnergies,
        Indeterminate];
434
435   PrintFun["Calculating the transition wavelengths in nm ..."];
436   magIon["Transition Wavelengths in nm"] =10^7/magIon["Transition
        Energies / K"];
437
438   PrintFun["Estimating the uncertainties in \[Lambda]/nm assuming a 1 K
         uncertainty in energies."];
439   (*Assuming an uncertainty of 1 K in both energies used to calculate
        the wavelength*)
440   \[Lambda]uncertainty=Sqrt[2]*magIon["Transition Wavelengths in nm"
```

```
  ]^2*10^-7;

PrintFun["Formatting a tabular output file ..."];
numEigenvecs    = Length[eigensys];
roundedEnergies = Round[eigenEnergies, 1.];
simpleFromTo    = Outer[{#1,#2}&, simplerStateLabels,
  simplerStateLabels];
fromTo          = Outer[{#1,#2}&, Range[numEigenvecs], Range[
  numEigenvecs]];
energyPairs     = Outer[{#1,#2}&, roundedEnergies,
  roundedEnergies];
allTransitions  = {simpleFromTo,
    fromTo,
    energyPairs,
    magIon["Transition Wavelengths in nm"],
    \[Lambda]uncertainty,
    magIon["AMD"],
    magIon["Radiative lifetimes"]
};
allTransitions = (Flatten/@Transpose[Flatten[#,1]&/@allTransitions]);
allTransitions = Select[allTransitions, #[[3]]!=#[[4]]&];
allTransitions = Select[allTransitions, #[[10]]>0&];
allTransitions = Transpose[allTransitions];

(*round things up*)
PrintFun["Rounding wavelengths according to estimated uncertainties
  ..."];
{roundedWaves,roundedDeltas} = Transpose[MapThread[
  RoundValueWithUncertainty,{allTransitions[[7]],allTransitions
  [[8]]}]];
allTransitions[[7]]          = roundedWaves;
allTransitions[[8]]          = roundedDeltas;

PrintFun["Rounding lifetimes and transition rates to three
  significant figures ..."];
allTransitions[[9]]          = RoundToSignificantFigures[#,3]&/@(
  allTransitions[[9]]);
allTransitions[[10]]         = RoundToSignificantFigures[#,3]&/@(
  allTransitions[[10]]);
finalTable                   = Transpose[allTransitions];
finalTable                   = Prepend[finalTable,header];

(* tabular output *)
basename     = ln <> " in " <> host <> " - example - " <> "MD1 -
  tabular.zip";
exportFname  = FileNameJoin[{"./examples",basename}];
PrintFun["Exporting tabular data to "<>exportFname<>" ..."];
exportKey    = StringReplace[basename,".zip"->".m"];
Export[exportFname, <|exportKey->finalTable|>];

(* raw data output *)
basename    = ln <> " in " <> host <> " - example - " <> "MD1 - raw.
  zip";
rawexportFname = FileNameJoin[{"./examples",basename}];
PrintFun["Exporting raw data as an association to "<>exportFname<>"
  ..."];
rawexportKey   = StringReplace[basename,".zip"->".m"];
Export[rawexportFname, <|rawexportKey->magIon|>];

(* csv output *)
PrintFun["Formatting and exporting a CSV output..."];
csvOut = Table[
    StringJoin[Riffle[ToString[#,CForm]&/@finalTable[[i]],","]],
{i,1,Length[finalTable]}
];
csvOut      = StringJoin[Riffle[csvOut, "\n"]];
basename    = ln <> " in " <> host <> " - example - " <> "MD1.csv";
exportFname = FileNameJoin[{"./examples", basename}];
PrintFun["Exporting csv data to "<>exportFname<>" ..."];
Export[exportFname, csvOut, "Text"];

If[OptionValue["Make Notebook"],
(
    PrintFun["Creating a notebook with a Manipulate to select a
  wavelength interval and a lifetime power of ten ..."];
    finalTable     = Rest[finalTable];
```

```
503       finalTable      = SortBy[finalTable,#[[7]]&];
504       opticalTable    = Select[finalTable,\[Lambda]min<=#[[7]]<=\[Lambda
    ]max&];
505       pows            = Sort[DeleteDuplicates[(MantissaExponent
    [#[[9]]][[2]]-1)&/@opticalTable]];
506
507       man             = Manipulate[
508       (
509           {\[Lambda]min,\[Lambda]max} = \[Lambda]int;
510           table = Select[opticalTable,And[(\[Lambda]min<=#[[7]]<=\[
    Lambda]max),
511                       (MantissaExponent[#[[9]]][[2]]-1)==log10\[Tau
    ]]&];
512           tab   = TableForm[table,TableHeadings->{None,header}];
513           Column[{{"\[Lambda]min="<>ToString[\[Lambda]min]<>" nm","\[
    Lambda]max="<>ToString[\[Lambda]max]<>" nm",log10\[Tau]},tab}]
514       ),
515       {{\[Lambda]int,\[Lambda]Range,"\[Lambda] interval"},
516           \[Lambda]Range[[1]],
517           \[Lambda]Range[[2]],
518           50,
519           ControlType->IntervalSlider
520       },
521       {{log10\[Tau],pows[[-1]]},
522           pows
523       },
524       TrackedSymbols  :> {\[Lambda]int,log10\[Tau]},
525       SaveDefinitions -> True
526       ];
527
528       nb = CreateDocument[{
529           TextCell[Style[DisplayForm[RowBox[{"Magnetic Dipole
    Transitions", "\n", SuperscriptBox[host<>":"<>ln,"3+"],"(",
    SuperscriptBox["f",numE],")"}]]],"Title",TextAlignment->Center],
530           (* TextCell["Magnetic Dipole Transition Lifetimes","Section",
    TextAlignment->Center], *)
531           TextCell[man,"Output",TextAlignment->Center]
532       },
533       WindowSelected -> True,
534       WindowTitle    -> "MD1 - "<>ln<>" in "<>host,
535       WindowSize     -> {1600,800}
536       ];
537       SelectionMove[nb, After, Notebook];
538       NotebookWrite[nb, Cell["Reload Data", "Section", TextAlignment ->
     Center]];
539       NotebookWrite[nb, Cell[(
540             "magTransitions = Import[FileNameJoin[{NotebookDirectory
    [],\"" <> StringSplit[rawexportFname,"/"][[-1]] <> "\"}],\""<>
    rawexportKey<>"\"];"
541           ),"Input"]];
542       SelectionMove[nb, Before, Notebook];
543       nbFname = FileNameJoin[{workDir,"examples","MD1 - "<>ln<>" in "<>
    "LaF3"<>".nb"}];
544       PrintFun[">> Saving notebook to ",nbFname," ..."];
545       NotebookSave[nb, nbFname];
546       If[OptionValue["Close Notebook"],
547           NotebookClose[nb];
548       ];
549   )
550   ];
551
552   Return[magIon];
553 )
```

References

[BG34] R. F. Bacher and S. Goudsmit. “Atomic Energy Relations. I”. In: *Phys. Rev.* 46.11 (Dec. 1934). Publisher: American Physical Society, pp. 948–969. DOI: 10.1103/PhysRev.46.948. URL: https://link.aps.org/doi/10.1103/PhysRev.46.948.

[BS57] Hans Bethe and Edwin Salpeter. *Quantum Mechanics of One- and Two-Electron Atoms*. 1957.

[Car+89] W. T. Carnall et al. “A systematic analysis of the spectra of the lanthanides doped into single crystal LaF3”. en. In: *The Journal of Chemical Physics* 90.7 (1989), pp. 3443–3457. ISSN: 0021-9606, 1089-7690. DOI: 10.1063/1.455853. URL: http://aip.scitation.org/doi/10.1063/1.455853 (visited on 07/02/2021).

[Che+08] Xueyuan Chen et al. “A few mistakes in widely used data files for fn configurations calculations”. In: *Journal of luminescence* 128.3 (2008). Publisher: Elsevier, pp. 421–427.

[Cow81] Robert Duane Cowan. *The theory of atomic structure and spectra*. en. Los Alamos series in basic and applied sciences 3. Berkeley: University of California Press, 1981. ISBN: 978-0-520-03821-9.

[JCC68] BR Judd, HM Crosswhite, and Hannah Crosswhite. “Intra-atomic magnetic interactions for f electrons”. In: *Physical Review* 169.1 (1968). Publisher: APS, p. 130. DOI: https://doi.org/10.1103/PhysRev.169.130.

[JS84] BR Judd and MA Suskin. “Complete set of orthogonal scalar operators for the configuration f3”. In: *JOSA B* 1.2 (1984). Publisher: Optica Publishing Group, pp. 261–265. DOI: https://doi.org/10.1364/JOSAB.1.000261.

[Jud66] BR Judd. “Three-particle operators for equivalent electrons”. In: *Physical Review* 141.1 (1966). Publisher: APS, p. 4. DOI: https://doi.org/10.1103/PhysRev.141.4.

[MR71] JC Morrison and K Rajnak. “Many-body calculations for the heavy atoms”. In: *Physical Review A* 4.2 (1971). Publisher: APS, p. 536.

[NK63] C. W. Nielson and George F Koster. *Spectroscopic Coefficients for the pn, dn, and fn configurations*. 1963.

[RW63] K Rajnak and BG Wybourne. “Configuration interaction effects in l^N configurations”. In: *Physical Review* 132.1 (1963). Publisher: APS, p. 280. DOI: https://doi.org/10.1103/PhysRev.132.280.

[Vel00] Dobromir Velkov. “Multi-electron coefficients of fractional parentage for the p, d, and f shells”. PhD thesis. John Hopkins University, 2000.

[Wyb65] Brian G Wybourne. *Spectroscopic Properties of Rare Earths*. 1965.