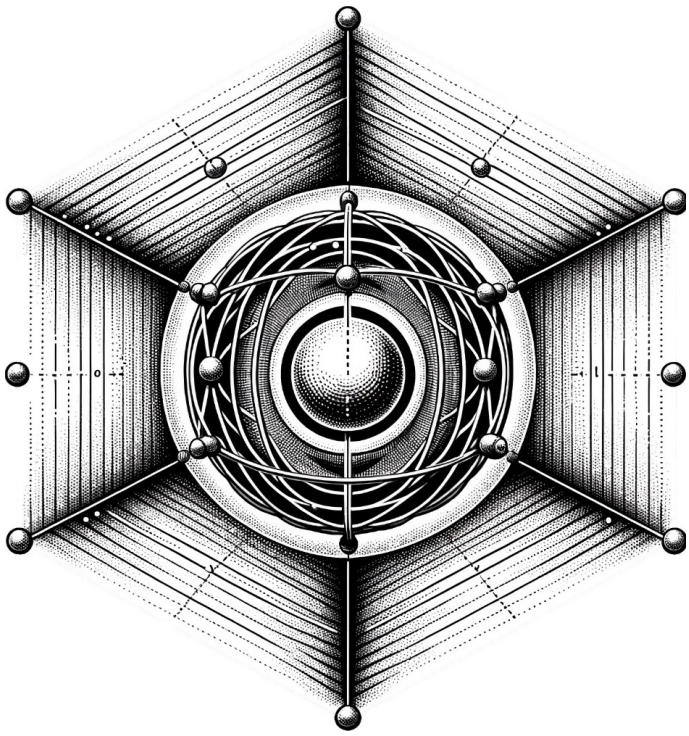

qlanth

version $|\alpha\rangle^{(5)}$

Juan David Lizarazo Ferro
& Christopher Dodson

Under the advisory of Dr. Rashid Zia

Contents

qlanth is a tool that can be used to estimate the electronic structure of lanthanide ions 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, Richard Leavitt, Brian Wybourne, and Katherine Rajnak among others. The goal of this tool is to provide a modern implementation of the methods that resulted from their work. This code is written in Wolfram language.

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 use elsewhere.

The included Mathematica notebook `qlanth.nb` has examples of the capabilities that this tool offers, and the `/examples` folder includes a series of notebooks for most of the trivalent lanthanide ions in lanthanum fluoride. $LaF_3$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 for their research into magnetic dipole transitions in lanthanide ions [DZ12]. Here it has been modified and rewritten by David Lizarazo. It has also benefited from conversations with Tharnier Puel at the University of Iowa.

This document has 12 sections. **Section 1** explains the details of the basis in which the Hamiltonian is evaluated. **Section 2** provides a brief explanation of the coefficients of fractional parentage. **Section 3** explains how the Hamiltonian is put together by first having calculated "JJ' blocks". **Section 4** is dedicated to a theoretical exposition of the effective Hamiltonian with subsections for each of the terms that it contains. **Section 5** is about the calculation of magnetic dipole transitions. **Sections 6 and 8** list additional data included in **qlanth**. **Section 7** has additional information about data fitting. **Section 9** has a brief comment on units. **Sections 9 and 10** include a summary of notation and definitions use throughout this document. Finally, **section 12** contains a printout of the code included in **qlanth**.

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 $\underline{f}^n$ are obtained from tables that originate from the original 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.

The following are all the LS terms in the $\underline{f}^n$ 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 ($S\to1, P\to2, D\to3, F\to4, G\to5, H\to6, I\to7, K\to8, L\to9, M\to10, N\to11, O\to12, Q\to3, R\to14, T\to15, U\to16, V\to17$), 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

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

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

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

^2F

$\underline{f}^{14}$
(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"};
```

In addition to LS the basis vector are also specified by the total angular momentum J (which may go from $|L-S|$ to $|L+S|$). Then for each J there are $2J+1$ projections on the z-axis. For example, the ordered $|LSJM_J\rangle$ basis for $\underline{f}^2$ is the one below. Where the first element is the LS term given as a string, the second equal to J, and the third one equal to M_J.

$(J=0)$
(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)
$\vert^3F, 3, -3\rangle$, $\vert^3F, 3, -2\rangle$, $\vert^3F, 3, -1\rangle$, $\vert^3F, 3, 0\rangle$, $\vert^3F, 3, 1\rangle$, $\vert^3F, 3, 2\rangle$, $\vert^3F, 3, 3\rangle$

$(J = 4)$ (27 kets)
$\vert^3F, 4, -4\rangle$, $\vert^3F, 4, -3\rangle$, $\vert^3F, 4, -2\rangle$, $\vert^3F, 4, -1\rangle$, $\vert^3F, 4, 0\rangle$, $\vert^3F, 4, 1\rangle$, $\vert^3F, 4, 2\rangle$, $\vert^3F, 4, 3\rangle$, $\vert^3F, 4, 4\rangle$, $\vert^3H, 4, -4\rangle$, $\vert^3H, 4, -3\rangle$, $\vert^3H, 4, -2\rangle$, $\vert^3H, 4, -1\rangle$, $\vert^3H, 4, 0\rangle$, $\vert^3H, 4, 1\rangle$, $\vert^3H, 4, 2\rangle$, $\vert^3H, 4, 3\rangle$, $\vert^3H, 4, 4\rangle$, $\vert^1G, 4, -4\rangle$, $\vert^1G, 4, -3\rangle$, $\vert^1G, 4, -2\rangle$, $\vert^1G, 4, -1\rangle$, $\vert^1G, 4, 0\rangle$, $\vert^1G, 4, 1\rangle$, $\vert^1G, 4, 2\rangle$, $\vert^1G, 4, 3\rangle$, $\vert^1G, 4, 4\rangle$

$(J = 5)$ (11 kets)
$\vert^3H, 5, -5\rangle$, $\vert^3H, 5, -4\rangle$, $\vert^3H, 5, -3\rangle$, $\vert^3H, 5, -2\rangle$, $\vert^3H, 5, -1\rangle$, $\vert^3H, 5, 0\rangle$, $\vert^3H, 5, 1\rangle$, $\vert^3H, 5, 2\rangle$, $\vert^3H, 5, 3\rangle$, $\vert^3H, 5, 4\rangle$, $\vert^3H, 5, 5\rangle$

$(J = 6)$ (26 kets)
$\vert^3H, 6, -6\rangle$, $\vert^3H, 6, -5\rangle$, $\vert^3H, 6, -4\rangle$, $\vert^3H, 6, -3\rangle$, $\vert^3H, 6, -2\rangle$, $\vert^3H, 6, -1\rangle$, $\vert^3H, 6, 0\rangle$, $\vert^3H, 6, 1\rangle$, $\vert^3H, 6, 2\rangle$, $\vert^3H, 6, 3\rangle$, $\vert^3H, 6, 4\rangle$, $\vert^3H, 6, 5\rangle$, $\vert^3H, 6, 6\rangle$, $\vert^1I, 6, -6\rangle$, $\vert^1I, 6, -5\rangle$, $\vert^1I, 6, -4\rangle$, $\vert^1I, 6, -3\rangle$, $\vert^1I, 6, -2\rangle$, $\vert^1I, 6, -1\rangle$, $\vert^1I, 6, 0\rangle$, $\vert^1I, 6, 1\rangle$, $\vert^1I, 6, 2\rangle$, $\vert^1I, 6, 3\rangle$, $\vert^1I, 6, 4\rangle$, $\vert^1I, 6, 5\rangle$, $\vert^1I, 6, 6\rangle$

The order above is exemplar of the ordering 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. Within each group for a given J the basis kets are then ordered in decreasing S, then ordered in increasing L, and then according to M_J.

In **qlanth** the ordered basis used for a given $\underline{f}^n$ 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}.
The option \"AsAssociation\" can be set to True to return the basis as an association with the keys being the allowed J values. The default is False.
";
Options[BasisLSJMJ] = {"AsAssociation" -> False};
BasisLSJMJ[numE_, OptionsPattern[]] := Module[{energyStatesTable, basis, idx1},
  (
    energyStatesTable = BasisTableGenerator[numE];
```

```
    basis = Table[
      energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
      {idx1, 1, Length[AllowedJ[numE]]}];
    basis = Flatten[basis, 1];
    If[OptionValue["AsAssociation"],
      (
        Js    = AllowedJ[numE];
        basis = Table[(J -> Select[basis, #[[2]] == J &]), {J, Js}];
        basis = Association[basis];
      )
    ];
    Return[basis]
  )
  ];
```

1.1 More quantun numbers

Besides using an integer that simply solves the problem of discriminating between degenerate LS terms by enumerating them, 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 $\mathcal{G}_2$.

1.1.1 Seniority ν

The seniority number connects different LS terms between configurations, so that a term below can be seen as the *senior* of a term above. To determine the seniority of a given term in configuration $\underline{\mathrm{f}}^n$ one must first find the configuration $\underline{\mathrm{f}}^{\tilde{n}}$ in which this term appeared. For example $\underline{\mathrm{f}}^5$ contains six degenerate ${}^2G$ terms. The first time this term appeared was in $\underline{\mathrm{f}}^3$, where it had a degeneracy of 2. The 2 degenerate terms in $\underline{\mathrm{f}}^3$ would then both have a seniority of $\boldsymbol{\nu} = 3$ since they first appeared in $\underline{\mathrm{f}}^3$. In consequency two of the six degenerate terms in $\underline{\mathrm{f}}^5$ would have the same degeneracy those two in $\underline{\mathrm{f}}^3$, and are therefore linked to those previous two. The four remaining ones, are considered to be *born* in $\underline{\mathrm{f}}^5$, and therefore have a seniority $\boldsymbol{\nu} = 5$.

These rules seem to be ad-hoc, but they are useful in dealing with the degeneracies in the LS terms as the arrive going up the configurations.

There is, however, a much deeper meaning to the seniority number. It can be shown that the seniority number (more exactly a quantity related to it) is a sort of spin, a quasi-spin, where the spin projections along the 'z-axis' correspond to different number of electrons in $\underline{\mathrm{f}}^n$ configurations. This is a consequence of the Pauli exclusion principle. It is also useful to relate matrix elements of operators in one configuration to those in another, through the use of the Wigner-Eckart theorem. This is an interesting an useful theoretical construct, but the method of fractional parentage (which is what is implemented in **qlanth** is well suited also, albeit being somewhat less parsimonious than what the quasi-spin view that seniority can provide. As such **qlanth** does not use the seniority numbers that are associated with each LS term.

1.1.2 $\mathcal{U}$ and $\mathcal{W}$

Much as L tells us how a rotation acts on a L wavefunction by mixing different M_L components, these other two labels specify how the wavefunctions transform under the operations of these other two groups. The $\mathcal{W}$ label determines how a wavefunction transforms under a rotation in 7-dimensional space, and $\mathcal{U}$ how they transform under an operator of group $\mathcal{G}_2$. Without going into the group theoretical details, the irreducible representations of $\mathcal{SO}(7)$ can be represented by triples of integer numbers, and those of $\mathcal{G}_2$ as pairs of two integers.

In **qlanth** the $\mathcal{W}$ and $\mathcal{U}$ are used in order to determine the matrix elements of the $\mathcal{C}(\mathcal{SO}(7))$ and $\mathcal{C}(\mathcal{G}_2)$ operators.

2 The coefficients of fractional parentage

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

In 1943 Giulio Racah [Rac43] provided a useful extension to this idea. In addition of using the energies of one spectrum to span the energies of another, Racah extended this idea to the wavefunctions themselves, such that from configuration $\underline{\mathrm{f}}^{n-1}$ one can create the wavefunctions for $\underline{\mathrm{f}}^{n-1}$ with all the required antisymmetry and normalization conditions. In this approach, a given *daughter* term in $\underline{\mathrm{f}}^{n}$ has a number of *parent* terms in $\underline{\mathrm{f}}^{n-1}$, with the coefficients of fractional parentage determining how much of each parent is in the daughter as a sum over parents

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

More importantly for **qlanth**, the coefficients of fractional parentage can be used to evaluate matrix elements of operators, such as in &qn-28, &qn-50, &qn-63, and &qn-40. These formulas realize a convenient calculation 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 6-j and 3-j coefficients, the LSUW labels for the terms in the $\underline{\mathrm{f}}^{n}$ configurations, reduced matrix elements in $\underline{\mathrm{f}}^{3}$ for the three-body operators, and reduced matrix elements in $\underline{\mathrm{f}}^{2}$ for the magnetic interactions.

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 lists which contain all the parents terms, together with the corresponding coefficients of fractional parentage.

```
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];
)
]
```

The coefficients of fractional parentage are traditionally only provided up to $\underline{f}^7$ (such is the case in `B1f_all.txt`), tabulating these beyond $\underline{f}^7$ would be redundant since the coefficients of fractional

parentage beyond $\underline{\mathrm{f}}^7$ satisfy relationships with those below $\underline{\mathrm{f}}^7$. According to [NK63]

$$\left(\underline{\ell}^{(14-n)-1}\bar{\alpha}\bar{L}\bar{S}\right\}\underline{\ell}^{(14-n)}\alpha LS\Big) = \xi(-1)^{S+\bar{S}+L+\bar{L}-7/2}\sqrt{\frac{(n+1)[\![\bar{S}]\!][\![\bar{L}]\!]}{(14-n)[\![S]\!][\![L]\!]}}\left(\underline{\ell}^{n-1}\alpha LS\right\}\underline{\ell}^{n}\bar{\alpha}\bar{L}\bar{S}\Big)$$

$$\text{with } \xi = \begin{cases} 1 & \text{if } n \neq 6 \\ (-1)^{(\bar{\nu}-1)/2} & \text{if } n = 6 \end{cases}, \text{ and where } \bar{\nu} \text{ is the seniority of } |\bar{\alpha}\bar{L}\bar{S}\rangle. \quad (2)$$

Under this relationship and phase convention, the matrix elements of operators pick up a global phase which depends on the rank of the operator, namely [NK63]:

$$\langle \underline{\mathrm{f}}^{14-n}\alpha SL\|\hat{U}^{(K)}\|\underline{\mathrm{f}}^{14-n}\alpha' S'L'\rangle = -(-1)^K\langle \underline{\mathrm{f}}^{n}\alpha SL\|\hat{U}^{(K)}\|\underline{\mathrm{f}}^{n}\alpha' S'L'\rangle \quad (3)$$

for a single tensor operator $\hat{U}^{(K)}$ of rank K, and

$$\langle \underline{\mathrm{f}}^{14-n}\alpha SL\|\hat{V}^{(1K)}\|\underline{\mathrm{f}}^{14-n}\alpha' S'L'\rangle = (-1)^K\langle \underline{\mathrm{f}}^{n}\alpha SL\|\hat{V}^{(1K)}\|\underline{\mathrm{f}}^{n}\alpha' S'L'\rangle \quad (4)$$

for a double tensor operator $\hat{V}^{(1K)}$ of rank 1 for spin and rank K for orbit.

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 in the $|LSJM_J\rangle$ basis 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 $\{J, J'\}$ is 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 =
      Table[
        (*Condition for a scalar matrix op*)
        SLterm   =  NKSLJM[[1]];
```

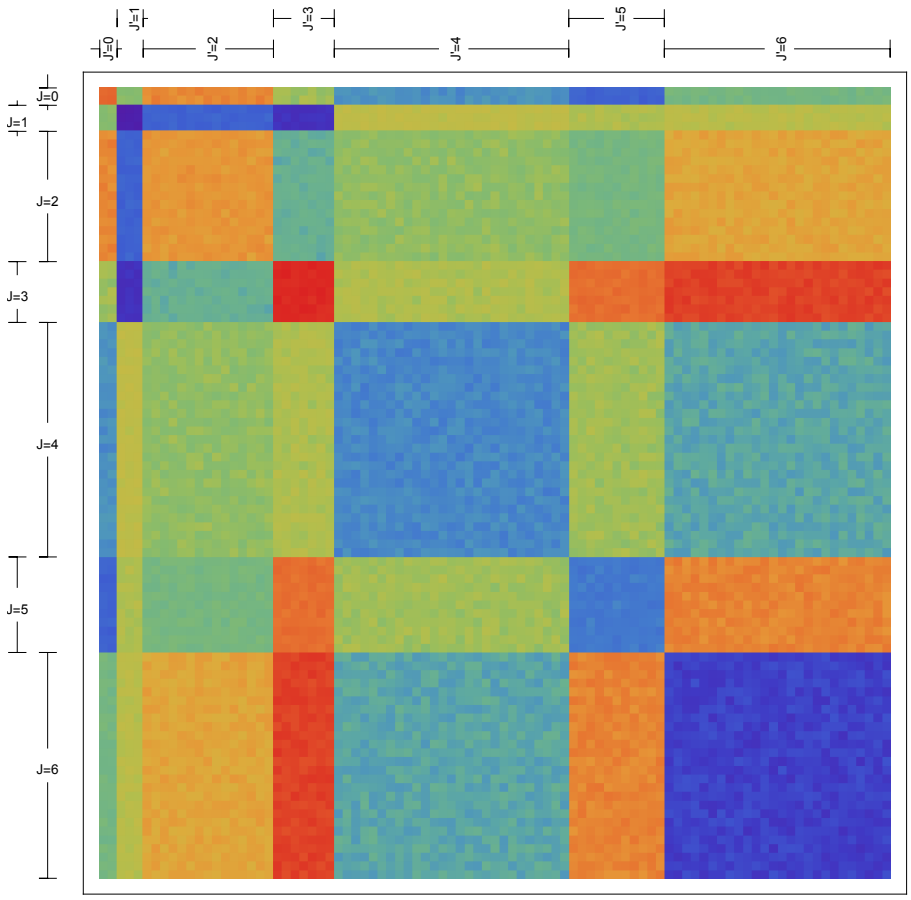

Figure 1: The block structure for $\underline{f}^2$

```
      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 calculated 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.
The option \"Set t2Switch\" can be used to toggle on or off setting the t2 selector automatically or not, the default is True, which replaces the parameter according to numE.
The option \"ReturnInBlocks\" can be use to return the matrix in block or flattened form. The default is to return it in flattened form.";
Options[HamMatrixAssembly] = {
      "FilenameAppendix"->"",
      "IncludeZeeman"->False,
      "Set t2Switch"->True,
      "ReturnInBlocks"->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];
      If[OptionValue["Set t2Switch"], params[t2Switch] = 0];
    ),
    params = params0;
    If[OptionValue["Set t2Switch"], params[t2Switch] = 1];
  ];
  (* 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 *)
  If[Not[OptionValue["ReturnInBlocks"]],
    (blockHam = ArrayFlatten[blockHam];
    blockHam  = ReplaceInSparseArray[blockHam, params];
    ),
    (blockHam  = Map[ReplaceInSparseArray[#, params]&,blockHam,{2}];)
  ];

  If[OptionValue["IncludeZeeman"],
    (
      PrintTemporary["Including Zeeman terms ..."];
      {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "
    ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
      blockHam += - TeslaToKayser * (Bx * magx + By * magy + Bz *
    magz);
    )
  ];
  Return[blockHam];
  ]
```

4 The effective Hamiltonian

Electrons in a multi-electron ion are subject to a number of interactions. They are attracted to the nucleus around which they orbit. Being bundled together with other electrons, they experience repulsion from all of them. Possesing spin, they are also subject 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. And between pairs of electrons, the spin of one can influence the other through the interaction of their respective magnetic dipoles.

To describe the effect of the charges in the lattice surrounding the lattice, the crystal field is

introduced. In the simplest of embodiments, the crystal field is simply seen as the electrostatic field due to the surrounding charges. This model, however, has some limitations, but it gives way to a much broader validity based solely on symmetry arguments.

This framework sufficiently describes the interactions within a free ion. However, to extend this model to ions within a crystal, one incorporates this through what is called 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 an 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 is called *configuration-interaction*.

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. For a detailed analysis of this see Rudzikas' [Rud07] book on theoretical atomic spectroscopy or this article [Lin74] by Lindgren. What results from this are operators that now act solely within the single configuration but with a convoluted coefficient that depends on overlap integrals between different configurations. It is from *configuration-interaction* that the parameters $\alpha, \beta, \gamma, P^{(k)}, T^{(k)}$ enter into the description.

The coefficients that result in the Hamiltonian one could try to evaluate, however within the **semi-empirical** approach these parameters are left to be fitted against experimental data, and perhaps approximated through Hartree-Fock analysis. This approach is only *semi* empirical in the sense that the model parameters are fitted from experimental data, but the model Hamilonian that is fitted is based on a clear physical picture inherited from atomic physics.

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

$$\hat{\mathscr{H}} = \underbrace{\hat{\mathscr{H}}_{\mathrm{k}}}_{\text{kinetic}} + \underbrace{\hat{\mathscr{H}}_{\mathrm{e:sn}}}_{\text{e:shielded nuc}} + \underbrace{\hat{\mathscr{H}}_{\mathrm{e:e}}}_{\text{e:e}} + \underbrace{\hat{\mathscr{H}}_{\mathrm{s:o}}}_{\text{spin-orbit}} + \underbrace{\hat{\mathscr{H}}_{\mathrm{s:s}}}_{\substack{\text{spin:spin}\\ \text{and spin:other-orbit}}} + \underbrace{\hat{\mathscr{H}}_{\mathrm{s:oo\oplus ecs:o}}}_{\substack{\text{spin:other-orbit}\\ \text{ec-correlated-spin:orbit}}} + \tag{5}$$

$$\underbrace{\hat{\mathscr{H}}_{\mathscr{SO}(3)}}_{\text{Trees effective op}} + \underbrace{\hat{\mathscr{H}}_{\mathrm{G}_2}}_{\mathrm{G}_2\text{ effective op}} + \underbrace{\hat{\mathscr{H}}_{\mathscr{SO}(7)}}_{\mathscr{SO}(7)\text{ effective op}} + \underbrace{\hat{\mathscr{H}}_{\curlywedge}}_{\substack{\text{effective}\\ \text{three-body}}} + \underbrace{\hat{\mathscr{H}}_{\mathrm{cf}}}_{\text{crystal field}} + \underbrace{\hat{\mathscr{H}}_{\mathrm{Z}}}_{\text{Zeeman}} \tag{6}$$

$$\hat{\mathscr{H}}_{\mathrm{k}} = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{n}\nabla_i^2 \text{ (kinetic energy of } n \text{ v-electrons)} \tag{7}$$

$$\hat{\mathscr{H}}_{\mathrm{e:sn}} = \sum_{i=1}^{n} V_{\mathrm{sn}}(r_i) \text{ (interaction of v-electrons with shielded nuclear charge)} \tag{8}$$

$$\hat{\mathscr{H}}_{\mathrm{e:e}} = \sum_{i>j}^{n,n}\frac{e^2}{\|\vec{r}_i - \vec{r}_j\|} = \sum_{k=0,2,4,6} F^{(k)}\hat{f}_k \text{ (v-electron:v-electron repulsion)} \tag{9}$$

$$\hat{\mathscr{H}}_{\mathrm{s:o}} = \begin{cases} \sum_{i=1}^{n}\xi(r_i)\left(\hat{\underline{s}}_i \cdot \hat{\underline{\ell}}_i\right) \text{ with } \xi(r_i) = \frac{\hbar^2}{2m^2c^2r_i}\frac{\mathrm{d}V_{\mathrm{sn}}(r_i)}{\mathrm{d}r_i} \\ \sum_{i=1}^{n}\zeta\left(\hat{\underline{s}}_i \cdot \hat{\underline{\ell}}_i\right) \text{ with } \zeta \text{ the radial average of } \xi(r_i) \text{ or used as phenomenological parameter} \end{cases} \tag{10}$$

$$\hat{\mathscr{H}}_{\mathrm{s:s}} = \sum_{k=0,2,4} M^{(k)}\hat{m}_k^{ss} \tag{11}$$

$$\hat{\mathscr{H}}_{\mathrm{s:oo\oplus ecs:o}} = \sum_{k=2,4,6} P^{(k)}\hat{p}_k + \sum_{k=0,2,4} M^{(k)}\hat{m}_k \tag{12}$$

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

$$\hat{\mathscr{H}}_{\mathscr{SO}(3)} = \alpha\, \mathcal{C}(\mathscr{SO}(3)) = \alpha \hat{L}^2 \text{ (Trees effective operator)} \tag{13}$$

$$\hat{\mathscr{H}}_{\mathrm{G}_2} = \beta\, \mathcal{C}(\mathcal{G}_2) \tag{14}$$

$$\hat{\mathscr{H}}_{\mathscr{SO}(7)} = \gamma\, \mathcal{C}(\mathscr{SO}(7)) \tag{15}$$

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

$$\hat{\mathscr{H}}_{\mathrm{cf}} = \sum_{i=1}^{n} V_{\mathrm{CF}}(\hat{r}_i) = \sum_{i=1}^{n}\sum_{k=2,4,6}\sum_{q=-k}^{k} \mathcal{B}_q^{(k)} C_q^{(k)}(i) \text{ (crystal field interaction of v-electrons with electrostatic field due to surroundings)} \tag{17}$$

$$\hat{\mathscr{H}}_{\mathrm{Z}} = -\vec{B}\cdot\hat{\mu} = \mu_B \vec{B}\cdot\left(\hat{L} + g_s\hat{S}\right) \text{ (interaction with a magnetic field)} \tag{18}$$

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.

4.1 $\hat{\mathscr{H}}_{\mathrm{k}}$: kinetic energy

$$\hat{\mathscr{H}}_{\mathrm{k}} = -\frac{\hbar^2}{2m}\sum_{i=1}^{N}\nabla_i^2 \text{ (kinetic energy of N v-electrons)} \tag{19}$$

Since our description is limited to a single configuration, the kinetic energy simply contributes a constant energy shift, and since all we care about are energy differences, then this term can be omitted from the analysis.

To interpret the range of energies that result from diagonalizing the Hamiltonian, tt might be instructive, however, to note that this term imparts an energy of about $10\,\mathrm{eV} = 10^6\mathrm{K}$ to each electron

4.2 $\hat{\mathcal{H}}_{\mathbf{e:sn}}$: the central field potential

In principle the sum over the Coulomb potential should extend over the nuclear charge and over all the electrons in the atom (not just the valence electrons). However, given the shell structure of the atom, the lanthanide ions "see" the nuclear charge as shielded by a xenon core. Since every closed shell is a singlet, having spherical symmetry, these shields are literally like spherical shells surrounding the nucleus.

$$\hat{\mathcal{H}}_{\mathrm{e:sn}} = -e^2 \sum_{i=1}^{Z} \frac{1}{r_i} + e^2 \underbrace{\sum_{i=1}^{n} \sum_{j=1}^{Z-n} \frac{1}{r_{ij}}}_{\substack{\text{Repulsion between valence} \\ \text{and inner shell} \\ \text{electrons}}} \approx \sum_{i=1}^{n} V_{\mathrm{sn}}(r_i) \text{ (with Z = atomic No.)} \tag{20}$$

The precise form of $V_{\mathrm{sn}}(r_i)$ is not of our concern here, all that matters is that we assume that it is spherically symmetric so that we can justify the separation of radial and angular parts of the wavefunctions.

4.3 $\hat{\mathcal{H}}_{\mathbf{e:e}}$: e:e repulsion

$$\hat{\mathcal{H}}_{\mathrm{e:e}} = \sum_{i>j}^{n,n} \frac{e^2}{\|\vec{r}_i - \vec{r}_j\|} = \sum_{k=0,2,4,6} F^{(k)} \hat{f}_k = \sum_{k=0,1,2,3} E_k \hat{e}^k \tag{21}$$

This term is the first we will not discard. Calculating this term for the $\underline{\mathrm{f}}^n$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 $\underline{\mathrm{f}}^n$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 $\underline{\mathrm{f}}^n$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.

$$\langle \underline{\mathrm{f}}^n \alpha^{2S+1} L \| \hat{\mathcal{H}}_{\mathrm{e:e}} \| \underline{\mathrm{f}}^n \alpha'^{2S'+1} L' \rangle = \sum_{k=0,2,4,6} F^{(k)} f_k(n, \alpha LS, \alpha' L' S') \tag{22}$$

where

$$f_k(n,\alpha LS,\alpha' L' S') = \frac{1}{2}\delta(S,S')\delta(L,L')\langle \underline{\mathrm{f}}\|\hat{C}^{(k)}\|\underline{\mathrm{f}}\rangle^2 \times$$

$$\left\{ \frac{1}{2L+1} \sum_{\alpha'' L''} \langle \underline{\mathrm{f}}^n \alpha'' L'' S\|\hat{U}^{(k)}\|\underline{\mathrm{f}}^n \alpha LS\rangle \langle \underline{\mathrm{f}}^n \alpha'' L'' S\|\hat{U}^{(k)}\|\underline{\mathrm{f}}^n \alpha' LS\rangle - \delta(\alpha,\alpha') \frac{n\,(4\underline{\mathrm{f}}+2-n)}{(2\underline{\mathrm{f}}+1)(4\underline{\mathrm{f}}+1)} \right\} \quad (23)$$

```
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}];
]
);
```

4.4 $\hat{\mathscr{H}}_{\text{s:o}}$: spin-orbit

The spin-orbit interaction arises from the interaction of the magnetic moment of the electron and the magnetic field that its orbital motion generates. In terms of the central potential $V_{\text{s:n}}$ the spin-orbit term for a single electron is

$$\hat{h}_{\text{s:o}} = \frac{\hbar^2}{2m_e^2c^2}\left(\frac{1}{r}\frac{\mathrm{d}V_{\text{s:n}}}{\mathrm{d}r}\right)\hat{l}\cdot\hat{s} := \zeta(r)\hat{l}\cdot\hat{s}. \tag{24}$$

Adding this term for all the n valence electrons, and replacing $\zeta(r)$ by it's radial average ζ then gives

$$\hat{\mathscr{H}}_{\text{s:o}} = \sum_i^n \zeta\hat{l}_i\cdot\hat{s}_i. \tag{25}$$

From equations 2-106 to 2-109 in Wybourne [Wyb63] the matrix elements we need are given by

$$\begin{aligned}\langle\alpha LSJM_J|\hat{\mathscr{H}}_{\text{s:o}}|\alpha' L'S'J'M_{J'}\rangle &= \zeta\delta(J,J')\delta(M_J,M_{J'})\langle\alpha LSJM_J|\sum_i^n \hat{l}_i\cdot\hat{s}_i|\alpha' L'S'JM_J\rangle \\ &= \zeta\delta(J,J')\delta(M_J,M_{J'})(-1)^{J+L+S'}\begin{Bmatrix} L & L' & 1\\ S' & S & J\end{Bmatrix}\langle\alpha LS|\sum_i^n \hat{l}_i\cdot\hat{s}_i|\alpha' L'S'\rangle \\ &= \zeta\delta(J,J')\delta(M_J,M_{J'})(-1)^{J+L+S'}\begin{Bmatrix} L & L' & 1\\ S' & S & J\end{Bmatrix}\sqrt{\underline{\ell}(\underline{\ell}+1)(2\underline{\ell}+1)}\langle\alpha LS\|\hat{V}^{(11)}\|\alpha' L'S'\rangle.\end{aligned} \tag{26}$$

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

$$\hat{V}^{(11)} = \sum_{i=1}^n \left(\hat{s}\hat{u}^{(1)}\right)_i, \tag{27}$$

where the rank on the spin operator $\hat{s}$ has been omitted, and the rank of the orbital tensor operator 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 [Wyb65]:

$$\begin{aligned}\langle\underline{\ell}^n\psi\|\hat{V}^{(1k)}\|\underline{\ell}^n\psi'\rangle = \langle\underline{\ell}^n\alpha LS\|\hat{V}^{(1k)}\|\underline{\ell}^n\alpha' L'S'\rangle = n\sqrt{\underline{\jmath}(\underline{\jmath}+1)(2\underline{\jmath}+1)}\sqrt{[\![S]\!][\![L]\!][\![S']\!][\![L']\!]}\times \\ \sum_{\bar{\psi}}(-1)^{\bar{S}+\bar{L}+S+L+\underline{\ell}+\underline{\jmath}+k+1}\left(\psi\left\{\bar{\psi}\right)\left(\bar{\psi}\right\}\psi'\right)\begin{Bmatrix} S & S' & 1\\ \underline{\jmath} & \underline{\jmath} & \bar{S}\end{Bmatrix}\begin{Bmatrix} L & L' & k\\ \underline{\ell} & \underline{\ell} & \bar{L}\end{Bmatrix}\end{aligned} \tag{28}$$

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];
  ]
```

These reduced matrix elements are then used by the function SpinOrbit.

```
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];
]
```

4.5 $\hat{\mathcal{H}}_{\mathcal{SO}(3)}, \hat{\mathcal{H}}_{\mathbf{G}_2}, \hat{\mathcal{H}}_{\mathcal{SO}(7)}$: electrostatic configuration interaction

This is a first term where we take into account the contributions from *configuration-interaction*. Rajnak and Wybourne [RW63] showed that *configuration-interaction* of the electrostatic interactions corresponding to two-electron excitations from $\underline{f}^n$ can be represented through the Casimir operators of the groups $\mathcal{SO}(3)$, $\mathcal{G}_2$, and $\mathcal{SO}(7)$. This borrowed from an earlier insight of Trees[Tre52], who realized that an addition of a term proportional to $L(L+1)$ improved the energy calculations for the second spectrum of manganese (MII) and the third spectrum of iron (FeIII).

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

$\Lambda_{\mathcal{G}_2}(U)$ is used to note the eigenvalue of the Casimir operator of $\mathcal{G}_2$ 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

$$\langle \underline{\ell}^n \alpha S L J M_J | \hat{\mathcal{H}}_{\mathcal{SO}(3)} | \underline{\ell}^n \alpha' S' L' J' M'_J \rangle = \alpha \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) L(L+1) \tag{29}$$

$$\langle \underline{\ell}^n U \alpha S L J M_J | \hat{\mathcal{H}}_{\mathrm{G}_2} | \underline{\ell}^n U \alpha' S' L' J' M'_J \rangle = \beta \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) \Lambda_{\mathcal{G}_2}(U) \tag{30}$$

$$\langle \underline{\ell}^n W \alpha S L J M_J | \hat{\mathcal{H}}_{\mathcal{SO}(7)} | \underline{\ell}^n W \alpha' S' L' J' M_J \rangle = \gamma \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) \Lambda_{\mathcal{SO}(7)}(W) \tag{31}$$

In **qlanth** the role of $\Lambda_{\mathcal{SO}(7)}(W)$ is played by the function `GSO7W`, the role of $\Lambda_{\mathcal{G}_2}(U)$ 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.

```
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];
  ]
```

4.6 $\hat{\mathcal{H}}_{\text{s:s-s:oo}}$: *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 use values from literature 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 used in **qlanth** is that of Judd, Crosswhite, and Crosswhite [JCC68]. 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

$$\hat{\mathcal{H}}_B = -\frac{1}{2} e^2 \sum_{i>j} \left[(\alpha_i \cdot \alpha_j) \frac{1}{r_{ij}} + (\alpha_i \cdot \vec{r}_{ij}) (\alpha_j \cdot \vec{r}_{ij}) \frac{1}{r_{ij}^3} \right]. \tag{32}$$

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

$$M^{(k)} := \frac{e^2\hbar^2}{8m^2c^2}\langle (nl)^2|\frac{r_<^k}{r_>^{k+3}}|(nl)^2\rangle \tag{33}$$

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

$$\hat{\mathcal{H}}_{s:s} = -2\sum_{i\neq j}\sum_k M^{(k)}\sqrt{(k+1)(k+2)(2k+3)}\langle \underline{\ell}\|C^{(k)}\|\underline{\ell}\rangle\langle \underline{\ell}\|C^{(k+2)}\|\underline{\ell}\rangle\left\{\hat{w}_i^{(1,k)}\hat{w}_j^{(1,k+2)}\right\}^{(2,2)0} \tag{34}$$

and the one for *spin-other-orbit*

$$\begin{aligned}\hat{\mathcal{H}}_{s:oo} = \sum_{i\neq j}\sum_k \sqrt{(k+1)(2\underline{\ell}+k+2)(2\underline{\ell}-k)}\times \\ \left[\left\{\hat{w}_i^{(0,k+1)}\hat{w}_j^{(1,k)}\right\}^{(11)0}\left\{M^{(k-1)}\langle \underline{\ell}\|C^{(k+1)}\|\underline{\ell}\rangle^2 + 2M^{(k)}\langle \underline{\ell}\|C^{(k)}\|\underline{\ell}\rangle^2\right\} + \right. \\ \left.\left\{\hat{w}_i^{(0,k)}\hat{w}_j^{(1,k+1)}\right\}^{(11)0}\left\{M^{(k)}\langle \underline{\ell}\|C^{(k)}\|\underline{\ell}\rangle^2 + 2M^{(k-1)}\langle \underline{\ell}\|C^{(k+1)}\|\underline{\ell}\rangle^2\right\}\right].\end{aligned} \tag{35}$$

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

$$\langle \underline{\ell}\|\hat{w}^{(\kappa,k)}\|\underline{\ell}\rangle = \sqrt{[\![\kappa]\!][\![k]\!]} \tag{36}$$

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

$$\sqrt{5}\hat{\mathcal{T}}_0^{(22)} := \hat{\mathcal{H}}_{\text{s:s}} \tag{37}$$

$$-\sqrt{3}\hat{\mathcal{T}}_0^{(11)} := \hat{\mathcal{H}}_{\text{s:oo}}. \tag{38}$$

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

$$\langle \gamma SLJ|\hat{\mathcal{H}}|\gamma' S'L'J'\rangle = \delta(J,J')\begin{Bmatrix} S' & L' & J \\ L & S & t\end{Bmatrix}\langle \gamma SL\|\hat{\mathcal{T}}^{(tt)}\|\gamma' S'L'\rangle. \tag{39}$$

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 $\underline{\mathrm{f}}^n$ are related to those in $\underline{\mathrm{f}}^{n-1}$ through equation 4 in Judd et al [JCC68]

$$\langle \underline{\mathrm{f}}^n \psi \| \hat{\mathcal{T}}^{(tt)} \| \underline{\mathrm{f}}^n \psi' \rangle = \frac{n}{n-2} \sum_{\bar{\psi},\bar{\psi}'} (-1)^{\bar{S}+\bar{L}+\underline{\jmath}+\underline{\ell}+S'+L'} \sqrt{[\![S]\!][\![S']\!][\![L]\!][\![L']\!]} \times$$
$$\left(\psi\{\|\bar{\psi}\}\right)\left(\psi'\{\|\bar{\psi}'\}\right) \begin{Bmatrix} S & t & S' \\ \bar{S}' & \underline{\jmath} & \bar{S} \end{Bmatrix} \begin{Bmatrix} L & t & L' \\ \bar{L}' & \underline{\ell} & \bar{L} \end{Bmatrix} \langle \underline{\mathrm{f}}^{n-1} \bar{\psi} \| \hat{\mathcal{T}}^{(tt)} \| \underline{\mathrm{f}}^{n-1}] \bar{\psi}' \rangle. \quad (40)$$

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 &qn-39 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 (with a slight modification) 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 $\underline{f}^2$ 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]
  ]
]
```

4.7 $\hat{\mathcal{H}}_{\mathbf{ecs:o}}$: *electrostatically-correlated-spin-orbit*

In the same paper [JCC68] that describes the *spin-spin* and *spin-other-orbit* interactions, consideration is also given to the emergence of additional corrections due to configuration interaction as described by the following operator (which is what results from the application of perturbation theory to *second* order) (page. 134 of [JCC68])

$$\hat{\mathcal{H}}_{\text{ci}} = -\sum_{\chi}\sum_{i}\frac{1}{E_\chi}\xi(r_i)\left(\hat{\underline{\jmath}}_i\cdot\hat{\underline{\ell}}_i\right)|\chi\rangle\langle\chi|\hat{\mathfrak{C}} - \frac{1}{E_\chi}\hat{\mathfrak{C}}|\chi\rangle\langle\chi|\xi(r_i)\left(\hat{\underline{\jmath}}_i\cdot\hat{\underline{\ell}}_i\right) \tag{41}$$

where $\xi(r_h)(\hat{\underline{\jmath}}_h\cdot\hat{\underline{\ell}}_h)$ is the customary spin-orbit interaction, E_χ is the energy of state $|\chi\rangle$, i is a label for the valence electrons, $\hat{\mathfrak{C}}$ stands for the Coulomb interaction, and $|\chi\rangle$ are states in the configurations to which one is "interacting" with. Since this term includes both the electrostatic term and the spin-orbit one, this is called the *electrostatically-correlated-spin-orbit* interaction.

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

$$\hat{\mathcal{H}}_{\text{ci}} = -\sqrt{3}\,\hat{t}_0^{(11)} \tag{42}$$

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

$$P^{(k)} = 6\sum_{f'}\frac{\zeta_{ff'}}{E_{ff'}}R^{(k)}(ff,ff') \text{ for } k=0,2,4,6. \tag{43}$$

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

$$\zeta_{ff'} \coloneqq \langle f|\xi(r)|f'\rangle \tag{44}$$

$$R^{(k)}(ff,ff') \coloneqq e^2\langle f_1 f_2|\frac{r_<^k}{r_>^{k+1}}|f_1 f_2'\rangle. \tag{45}$$

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. They also developed 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

$$\langle\psi\|\hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)}\|\psi'\rangle = \sum a_i \langle\psi\|\hat{z}_i\|\psi'\rangle. \tag{46}$$

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

$$\hat{\mathcal{H}}_{\text{s:oo}} + \hat{\mathcal{H}}_{\text{ecs:o}} = \hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)} - \frac{1}{6} a_{13} \hat{z}_{13} \tag{47}$$

where

$$a_{13} = -33M^{(0)} + 3M^{(2)} + \frac{15}{11}M^{(4)} - 6P^{(0)} + \frac{3}{2}\left(\frac{35}{225}P^{(2)} + \frac{77}{1089}P^{(4)} + \frac{25}{1287}P^{(6)}\right). \tag{48}$$

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} (1 for including, 0 for excluding).

```
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 used 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];

  exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"
    }];
  If[OptionValue["Export"],
    (
      Print["Exporting to file "<>ToString[exportFname]];
      Export[exportFname, SpinOrbitTable];
    )
  ];
  Return[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-40 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-47 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-40 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];
  ];
```

4.8 $\hat{\mathcal{H}}_{\curlywedge}$: three-body effective operators

The three-body operators arise in the Hamiltonian due to the configuration interaction effects of the Coulomb repulsion. More specifically, they originate from configuration interaction between the ground configuration $(4\underline{f})^n$ and single electron excitations to the $(4\underline{f})^{n\pm1}(n'\underline{\ell}')^{\mp1}$ configurations.

The operators that can be used to span the resulting effects were initially studied by Wybourne and Rajnak in 1963 [RW63], their analysis was complemented soon after by Judd [Jud66], and revisited again by Judd in 1984 [JS84].

This model interaction is spanned by a set of 14 $\hat{t}_i$ of operators ($\hat{t}$ fron *t*hree)

$$\hat{\mathcal{H}}_{\curlywedge} = T'^{(2)}\hat{t}'_2 + T'^{(11)}\hat{t}'_{11} \sum_{\substack{k=2,3,4,6,7,8,\\ 11,12,14,15,\\ 16,17,18,19}} T^{(k)}\hat{t}_k, \tag{49}$$

where $\hat{t}'_2$ and $\hat{t}'_{11}$ are operators that were initially included but which were later replaced by $\hat{t}_2$ and $\hat{t}_{11}$ (see [JS84]). **qlanth** includes the legacy operator $\hat{t}'_2$ since it was used for important work during and before the 1980s.

The omission of some indices in this sum has to do with the fact that the way in which these are defined in terms of their index (see [Jud66]) gives rise to two-body operators which can be absorbed by the two-body terms in the Hamiltonian. As such, it is not so much that they are not included, but rather that their effects are considered to be accounted for elsewhere. This is representative of a common feature of configuration interaction: it gives rise to new intra-configuration operators, but it also contributes to already present operators; this makes it harder to approximate the model parameters *ab-initio*, but is not a practical obstacle for the semi-empirical approach (although it certainly complicates the physical interpretation that each parameter has).

Furtermore, it is often the case that the operator set is limited to the subset {2,3,4,6,7,8}; a practice that is justified *post-facto* after seeing that these are sufficient to describe the data.

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

$$\langle \underline{f}^n \psi | \hat{t}_i | \underline{f}^n \psi' \rangle = \delta(J,J')\delta(M_J,M'_J)\frac{n}{n-3}\sum_{\bar{\psi}\bar{\psi}'} \left(\psi\middle\{\middle|\bar{\psi}\right)\left(\psi'\middle\{\middle|\bar{\psi}'\right)\langle \underline{f}^{n-1}\bar{\psi}|\hat{t}_i|\underline{f}^{n-1}\bar{\psi}'\rangle. \tag{50}$$

The sum in this expression 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, M_J as is due to a scalar operator.

In **qlanth** this is all implemented in the function `GenerateThreeBodyTables`. Where the matrix elements in $\underline{f}^3$ are 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 $\hat{t}_2$ should not be used and vice versa.

One additional feature of $\hat{t}'_2$ that needs to be accounted for, is that it doesn't have the simple relationship for conjugate configurations that all the other $\hat{t}_i$ operators have. For the sake of

parsimony, and to avoid having to explicitly store matrix elements beyond $\underline{f}^7$ **qlanth** takes the approach of adding a control parameter t2Switch which needs to be set to 1 if below or at $\underline{f}^7$ and set to 0 if above $\underline{f}^7$.

```
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}];)
```

```
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]);

  (* Parse table 1 from Judd 1984 *)
```

```
judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"}];
data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
headers = data[[1]];
data = data[[2 ;;]];
data = Transpose[data];
\[Psi] = Select[data[[1]], # != "" &];
\[Psi]p = Select[data[[2]], # != "" &];
matrixKeys = Transpose[{\[Psi], \[Psi]p}];
data = data[[3 ;;]];
cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data}];
cols = Select[cols, Length[#] == 21 &];
tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
tab1 = Transpose[Prepend[Transpose[tab1], headers]];

(* Parse table 2 from Judd 1984 *)
judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"}];
data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
headers = data[[1]];
data = data[[2 ;;]];
data = Transpose[data];
{operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} = data[[;; 4]];
multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
multiFactorValues = AssociationThread[multiFactorSymbols -> multiFactorValues];

(*scale values of table 1 given the values in table 2*)
oppyS = {};
normalTable =
  Table[header = col[[1]];
    If[StringContainsQ[header, " "],
      (
        multiplierSymbol = StringSplit[header, " "][[1]];
        multiplierValue = multiFactorValues[multiplierSymbol];
        operatorSymbol = StringSplit[header, " "][[2]];
        oppyS = Append[oppyS, operatorSymbol];
      ),
      (
        multiplierValue = 1;
        operatorSymbol = header;
      )
    ];
    normalValues = 1/multiplierValue*col[[2 ;;]];
    Join[{operatorSymbol}, normalValues], {col, tab1[[3 ;;]]}
  ];

(*Create an association for the matrix elements in the f^3 config*)
juddOperators = Association[];
Do[(
  col      = normalTable[[colIndex]];
  opLabel  = col[[1]];
  opValues = col[[2 ;;]];
  opMatrix = AssociationThread[matrixKeys -> opValues];
```

```
    Do[(
      opMatrix[Reverse[mKey]] = opMatrix[mKey]
      ),
    {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];
)
```

4.9 $\hat{\mathcal{H}}_{\mathbf{cf}}$: crystal-field

The crystal-field partially accounts for the influence of the surrounding lattice on the ion. The simplest picture of this influence imagines the lattice as responsible for an electric field felt at the position of the ion. This electric field corresponding to an electrostatic potential described as a multipolar sum of the form:

$$V(r_i, \theta_i, \phi_i) = \sum_{k=1}^{\infty} \sum_{q=-k}^{k} \mathcal{A}_q^{(k)} r_i^k C_q^{(k)}(\theta_i, \phi_i) \tag{51}$$

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

$$\hat{\mathscr{H}}_{\rm cf}(\vec{r}) = \sum_{i=1}^{n}\sum_{k=0}^{\infty}\sum_{q=-k}^{k} \mathscr{A}_q^{(k)} r_i^k \mathcal{C}_q^{(k)}(\theta_i,\phi_i). \tag{52}$$

And if we average the radial coordinate,

$$\hat{\mathscr{H}}_{\rm cf} = \sum_{i=1}^{n}\sum_{k=1}^{\infty}\sum_{q=-k}^{k} \mathcal{B}_q^{(k)} \mathcal{C}_q^{(k)}(i) \tag{53}$$

where the radial average is included as

$$\mathcal{B}_q^{(k)} := \mathscr{A}_q^{(k)} \langle 4f|r^k|4f\rangle. \tag{54}$$

$\mathcal{B}_q^{(k)}$ may be complex in general. However, since the sum in &eqn-52 needs to result in a real and Hermitian operator, there are restrictions on $\mathcal{B}_q^{(k)}$ that need to be accounted for. Once the behavior of $\mathcal{C}_q^{(k)}$ under complex conjugation is considered, $\mathcal{C}_q^{(k)*} = (-1)^q \mathcal{C}_{-q}^{(k)}$, it is necessary that

$$\mathcal{B}_q^{(k)} = (-1)^q \mathcal{B}_{-q}^{(k)*}. \tag{55}$$

Presently the sum over q spans both its negative and positive values. This can be limited to only the non-negative values of q. Separating the real and imaginary parts of $\mathcal{B}_q^{(k)}$ such that $\mathcal{B}_q^{(k)} = B_q^{(k)} + iS_q^{(k)}$ for $q \neq 0$ and $\mathcal{B}_0^{(k)} = 2B_0^{(k)}$ the sum for the crystal field can then be written as

$$\hat{\mathscr{H}}_{\rm cf}(\vec{r}) = \sum_{i=1}^{n}\sum_{k=0}^{\infty}\sum_{q=0}^{k} B_q^{(k)}\left(\mathcal{C}_q^{(k)} + (-1)^q \mathcal{C}_{-q}^{(k)}\right) + iS_q^{(k)}\left(\mathcal{C}_q^{(k)} - (-1)^q \mathcal{C}_{-q}^{(k)}\right). \tag{56}$$

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):

$$\langle \ell \| \hat{u}^{(k)} \| \ell' \rangle = 1. \tag{57}$$

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

$$\hat{U}^{(k)} = \sum_{i}^{n} \hat{u}_i^{(k)}. \tag{58}$$

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

$$\mathcal{C}_q^{(k)} = \langle \underline{\ell} \| C^{(k)} \| \underline{\ell}' \rangle \hat{u}_q^{(k)} = (-1)^{\underline{\ell}} \sqrt{[\![\underline{\ell}]\!][\![\underline{\ell}']\!]} \begin{pmatrix} \underline{\ell} & k & \underline{\ell}' \\ 0 & 0 & 0 \end{pmatrix} \hat{u}_q^{(k)}. \tag{59}$$

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

Wybourne eqn. 6-3

$$\langle \underline{\ell}^n \alpha SLJM_J | \hat{\mathscr{H}}_{\rm cf} | \underline{\ell}^n \alpha' SL'J'M_{J'} \rangle = \sum_{k=1}^{\infty}\sum_{q=-k}^{k} \mathcal{B}_q^{(k)} \langle \underline{\ell}^n \alpha SLJM_J | \hat{U}_q^{(k)} | \underline{\ell}^n \alpha' SL'J'M_{J'} \rangle \langle \underline{\ell} \| \hat{C}^{(k)} \| \underline{\ell} \rangle \tag{60}$$

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

$$\overbrace{\langle \underline{\ell}^n \alpha SLJM_J|\hat{U}_q^{(k)}|\underline{\ell}^n \alpha' S'L'J'M_{J'}\rangle}^{\text{Wybourne eqn. 6-4}} = (-1)^{J-M_J}\begin{pmatrix} J & k & J' \\ -M_J & q & M_{J'}\end{pmatrix}\langle \underline{\ell}^n \alpha SLJ\|\hat{U}^{(k)}\|\underline{\ell}^n \alpha' S'L'\rangle \quad (61)$$

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

$$\overbrace{\langle \underline{\ell}^n \alpha SLJ\|\hat{U}^{(k)}\|\underline{\ell}^n \alpha' S'L'\rangle}^{\text{Wybourne eqn. 6-5}} = (-1)^{S+L+J'+k}\sqrt{[\![J]\!][\![J']\!]}\times \begin{Bmatrix} J & J' & k \\ L' & L & S\end{Bmatrix}\langle \underline{\ell}^n \alpha SL\|\hat{U}^{(k)}\|\underline{\ell}^n \alpha' S'L'\rangle. \quad (62)$$

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

$$\overbrace{\langle \underline{\ell}^n \alpha SL\|\hat{U}^{(k)}\|\underline{\ell}^n \alpha' S'L'\rangle}^{\text{Cowan eqn. 11.53}} = \delta(S,S')n(-1)^{\underline{\ell}+L+k}\sqrt{[\![L]\!][\![L']\!]}\times \sum_{\bar{\alpha}\bar{L}\bar{S}}(-1)^{\bar{L}}\begin{Bmatrix} \underline{\ell} & k & \underline{\ell} \\ L & \bar{L} & L'\end{Bmatrix}\left(\underline{\ell}^n \alpha LS\left\{\underline{\ell}^{n-1}\bar{\alpha}\bar{L}\bar{S}\right)\left(\underline{\ell}^{n-1}\bar{\alpha}\bar{L}\bar{S}\right\}\underline{\ell}^n \alpha' L'S'\right). \quad (63)$$

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 $\triangleleft(3,k,3)$ 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 $\langle \underline{\ell}^n \alpha SLJM_J|\hat{\mathscr{H}}_{\mathrm{cf}}|\underline{\ell}^n \alpha' SL'J'M_{J'}\rangle$ 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. Putting everything together:

$$\hat{\mathscr{H}}_{\mathrm{cf}}(\vec{r}) = \sum_{i=1}^{n}\sum_{k=2,4,6}\sum_{q=0}^{k} B_q^{(k)}\left(C_q^{(k)} + (-1)^q C_{-q}^{(k)}\right) + iS_q^{(k)}\left(C_q^{(k)} - (-1)^q C_{-q}^{(k)}\right). \quad (64)$$

The above equations are implemented in **qlanth** by the function `CrystalField`. This function puts together the symbolic sum in Eqn-60 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.

```
Bqk::usage="Real part of the Bqk coefficients.";
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::usage="Imaginary part of the Bqk coefficients.";
Sqk[q_, 2] := {0, S12, S22}[[q + 1]];
Sqk[q_, 4] := {0, S14, S24, S34, S44}[[q + 1]];
Sqk[q_, 6] := {0, S16, S26, S36, S46, S56, S66}[[q + 1]];
```

```
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
  ]
```

```
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}
  ]
)
```

```
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.";
ReducedUk[numE_, l_, SL_, SpLp_, k_] :=
  Module[{spin, orbital, Uk,
    S, L, Sp, Lp, Sb, Lb,
    parentSL, cfpSL, cfpSpLp, Ukval, SLparents, SLpparents,
    commonParents, phase},
    {spin, orbital} = {1/2, 3};
    {S, L}          = FindSL[SL];
    {Sp, Lp}        = FindSL[SpLp];
    If[Not[S == Sp],
      Return[0]
    ];
    cfpSL       = CFP[{numE, SL}];
    cfpSpLp     = CFP[{numE, SpLp}];
    SLparents   = First /@ Rest[cfpSL];
    SLpparents  = First /@ Rest[cfpSpLp];
    commonParents = Intersection[SLparents, SLpparents];
    Uk = Sum[(
      {Sb, Lb} = FindSL[\[Psi]b];
      Phaser[Lb] *
        CFPAssoc[{numE, SL, \[Psi]b}] *
        CFPAssoc[{numE, SpLp, \[Psi]b}] *
        SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
    ),
    {\[Psi]b, commonParents}
    ];
    phase     = Phaser[orbital + L + k];
    prefactor = numE * phase * Sqrt[TPO[L,Lp]];
    Ukval     = prefactor*Uk;
    Return[Ukval];
]
```

4.10 $\hat{\mu}$ and $\hat{\mathscr{H}}_{\mathbf{Z}}$: the magnetic dipole operator and the Zeeman term

In Hartree atomic units, the operator associated with the magnetic dipole operator for an electron is

$$\hat{\mu} = -\mu_{\mathrm{B}} \left(\hat{L} + g_s \hat{S} \right)^{(1)}, \text{with } \mu_{\mathrm{B}} = 1/2. \tag{65}$$

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 equation 15.7 in [Cow81]

$$\langle \alpha LSJ \| \left(\hat{L} + g_s \hat{S} \right)^{(1)} \| \alpha' L' S' J' \rangle = \delta(\alpha LSJ, \alpha' L' S' J') \sqrt{J(J+1)(2J+1)} + \delta(\alpha LS, \alpha' L' S') (-1)^{L+S+J+1} \sqrt{[\![J]\!][\![J]\!]} \begin{Bmatrix} L & S & J \\ 1 & J' & S \end{Bmatrix} \quad (66)$$

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

$$\langle \alpha LSJ M_J | \left(\hat{L} + g_s \hat{S} \right)^{(1)}_q | \alpha' L' S' J' M_{J'} \rangle = (-1)^{J - M_J} \begin{pmatrix} J & 1 & J' \\ -M_J & q & M_J' \end{pmatrix} \langle \alpha LSJ \| \left(\hat{L} + g_s \hat{S} \right)^{(1)} \| \alpha' L' S' J' \rangle \quad (67)$$

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

$$\hat{\mu}_x = \frac{\hat{\mu}^{(1)}_{-1} - \hat{\mu}^{(1)}_{+1}}{\sqrt{2}} \tag{68}$$

$$\hat{\mu}_y = i\frac{\hat{\mu}^{(1)}_{-1} + \hat{\mu}^{(1)}_{+1}}{\sqrt{2}} \tag{69}$$

$$\hat{\mu}_z = \hat{\mu}^{(1)}_0 \tag{70}$$

```
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. The option \"
    ReturnInBlocks\" can be use to return the matrices in blocks. The
    default is to return the matrices in flattened form.";
Options[MagDipoleMatrixAssembly]={
  "FilenameAppendix"->"",
  "ReturnInBlocks"->False};
MagDipoleMatrixAssembly[nf_Integer, OptionsPattern[]]:=Module[
  {ImportFun, numE, appendTo, emFname, JJBlockMagDipTable, Js,
    howManyJs, blockOp, rowIdx, colIdx},
  (
  ImportFun = ImportMZip;
  numE      = nf;
  numH      = 14 - numE;
```

```
  numE       = Min[numE,numH];

  appendTo   = (OptionValue["FilenameAppendix"]<>"-magDip");
  emFname    = JJBlockMatrixFileName[numE,"FilenameAppendix"->appendTo
    ];
  JJBlockMagDipTable = ImportFun[emFname];

  Js         = AllowedJ[numE];
  howManyJs  = Length[Js];
  blockOp    = ConstantArray[0,{howManyJs,howManyJs}];
  Do[
    blockOp[[rowIdx,colIdx]] = JJBlockMagDipTable[{numE,Js[[rowIdx]],
    Js[[colIdx]]}],
    {rowIdx,1,howManyJs},
    {colIdx,1,howManyJs}
  ];
  If[OptionValue["ReturnInBlocks"],
    (
      opMinus = Map[#[[1]]&, blockOp, {4}];
      opZero  = Map[#[[2]]&, blockOp, {4}];
      opPlus  = Map[#[[3]]&, blockOp, {4}];
      opX =   (opMinus - opPlus)/Sqrt[2];
      opY = I (opPlus + opMinus)/Sqrt[2];
      opZ = opZero;
    ),
      blockOp = ArrayFlatten[blockOp];
      opMinus = blockOp[[;; , ;; , 1]];
      opZero  = blockOp[[;; , ;; , 2]];
      opPlus  = blockOp[[;; , ;; , 3]];
      opX =   (opMinus - opPlus)/Sqrt[2];
      opY = I (opPlus + opMinus)/Sqrt[2];
      opZ = opZero;
  ];
  Return[{opX, opY, opZ}];
)
];
```

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 `TeslaToKayser` in the file `qonstants.m`.

4.11 Going beyond f^7

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 $\underline{f}^{14-n}$ as given by &qn-4 and &qn-3.

$$\hat{\mathcal{H}} = \underbrace{\hat{\mathcal{H}}_{\mathrm{k}}}_{\text{kinetic}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{e:sn}}}_{\text{e:shielded nuc}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{e:e}}}_{\text{e:e}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{s:o}}}_{\text{spin-orbit}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{s:s}}}_{\text{spin:spin and spin:other-orbit}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{s:oo\oplus ecs:o}}}_{\text{spin:other-orbit ec-correlated-spin:orbit}} + \tag{71}$$

$$\underbrace{\hat{\mathcal{H}}_{\mathcal{SO}(3)}}_{\text{Trees effective op}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{G}_2}}_{\mathrm{G}_2\text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{\mathcal{SO}(7)}}_{\mathcal{SO}(7)\text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{\triangle}}_{\text{effective three-body}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{cf}}}_{\text{crystal field}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{Z}}}_{\text{Zeeman}} \tag{72}$$

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):

$$\zeta, T^{(2)}, T^{(3)}, T^{(4)}, T^{(6)}, T^{(7)}, T^{(8)}, B_q^{(k)} \tag{73}$$

In **qlanth** this symmetry is taken into account when the function `HamMatrixAssembly` is called, which uses `HoleElectronConjugation` to enforce the necessary sign changes.

```
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];
    )
  ]
```

5 Magnetic Dipole Transitions

qlanth can also calculate magnetic dipole transitions. With $\hat{\mu} = \{\hat{\mu}_x, \hat{\mu}_y, \hat{\mu}_z\}$ the magnetic dipole operator, the line strength between two eigenstates $|\nu\rangle$ and $|\nu'\rangle$ is defined as (see for example equation 14.31 in [Cow81])

$$\hat{S}(\psi, \psi') := |\langle\psi|\hat{\mu}|\psi'\rangle|^2 = |\langle\psi|\hat{\mu}_x|\psi'\rangle|^2 + |\langle\psi|\hat{\mu}_y|\psi'\rangle|^2 + |\langle\psi|\hat{\mu}_z|\psi'\rangle|^2 \tag{74}$$

In **qlanth** this is computed with the function MagDipLineStrength, which given a set of eigenvectors computes the sum above, and returns an array that contains all possible pairings of $|\psi\rangle$ and $|\psi'\rangle$ in $\hat{S}(\psi,\psi')$.

```
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[];
    )
  ];
)
]
```

Using the line strength $\hat{S}$ the rate $\mathrm{A_{MD}}$ for the spontaneous transition $|\psi_i\rangle \to |\psi_f\rangle$ is then given by (from table 7.3 of [TLJ99])

$$\mathrm{A_{MD}}(|\psi_i\rangle \to |\psi_f\rangle) = \frac{16\pi^3\mu_0}{3h}\frac{n^3}{\lambda^3}\frac{\hat{S}(\psi_i,\psi_f)}{g_i}, \tag{75}$$

where λ is the vacuum-equivalent wavelength of the transition between $|\nu\rangle$ and $|\nu'\rangle$, n the refractive index of the medium containing the ion, and g_i the degeneracy of the initial state $|\psi_i\rangle$. At the fine-grained description that **qlanth** uses, J is no longer a good quantum number so $g_i = 1$.

```
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. eigenSys is a list of
     lists with two elements, in each list the first element is the
    energy and the second one the corresponding eigenvector.
Based on table 7.3 of Thorne 1999, using g2=1.
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>.
By default this assumes that the refractive index is unity, this may
    be changed by setting the option \"RefractiveIndex\" to the
    desired value.
The option \"Lifetime\" can be used to return the reciprocal of the
    transition rates. The default is to return the transition rates.";
Options[MagDipoleRates]={"Units"->"SI", "Lifetime"->False, "
    RefractiveIndex"->1};
MagDipoleRates[eigenSys_List, numE0_Integer,OptionsPattern[]]:=Module
    [
  {AMD,Stot,eigenEnergies,transitionWaveLengthsInMeters,nRefractive
    },(
  nRefractive    = OptionValue["RefractiveIndex"];
  numE           = Min[14-numE0, numE0];
  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;

  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 / transitionWaveLengthsInMeters^3 * Stot *
    nRefractive^3;
  Which[OptionValue["Lifetime"],
    Return[1/AMD],
    True,
    Return[AMD]
  ]
  )
]
```

A final quantity of interest is the oscillator strength for the transition between the ground state $|\psi_g\rangle$ and an excited state $|\psi_e\rangle$. The oscillator strength is a dimensionless quantity which is indicative of how strong absorption is. The oscillator strength may be defined for other initial state than the ground state, but since this is the state most likely to be populated in ordinary experimental conditions, this is the initial state that is of more frequent interest. The oscillator strength is given by [CFW65]

$$\mathrm{f}_{\mathrm{MD}}\left(|\psi_g\rangle \to |\psi_e\rangle\right) = \frac{8\pi^2 m_e}{3\,hc\,e^2}\frac{n}{\lambda}\frac{\hat{S}(\psi_g,\psi_e)}{g_g} \tag{76}$$

where g_g is the degeneracy of the ground state. At the level of detail that the eigenstates are described in **qlanth** where J is no longer a good quantum number, $g_g = 1$.

```
GroundStateOscillatorStrength::usage="GroundStateOscillatorStrength[
    eigenSys, numE] calculates the oscillator strengths between the
    ground state and the excited states as given by eigenSys.
Based on equation 8 of Carnall 1965, removing the 2J+1 factor since
    this degeneracy has been removed by the crystal field.
eigenSys is a list of lists with two elements, in each list the first
     element is the energy and the second one the corresponding
    eigenvector.
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>.
By default this assumes that the refractive index is unity, this may
    be changed by setting the option \"RefractiveIndex\" to the
    desired value.";
Options[GroundStateOscillatorStrength]={"RefractiveIndex"->1};
```

```
GroundStateOscillatorStrength[eigenSys_List, numE_Integer,
    OptionsPattern[]]:=Module[
{eigenEnergies, SMDGS, GSEnergy, energyDiffs,
    transitionWaveLengthsInMeters, unitFactor, nRefractive},
(
  eigenEnergies     = First/@eigenSys;
  nRefractive       = OptionValue["RefractiveIndex"];
  SMDGS             = MagDipLineStrength[eigenSys,numE, "Units"->"SI",
     "States"->1];
  GSEnergy          = eigenSys[[1,1]];
  energyDiffs       = eigenEnergies-GSEnergy;
  energyDiffs[[1]] = Indeterminate;
  transitionWaveLengthsInMeters = 0.01/energyDiffs;
  unitFactor        = (8\[Pi]^2 me)/(3 hPlanck eCharge^2 cLight);
  fMDGS             = unitFactor / transitionWaveLengthsInMeters *
    SMDGS * nRefractive;
  Return[fMDGS];
)
]
```

6 Data fitting

qlanth also has the capacity to fit the Hamiltonian to experimental data. This is included in the sub-module `fittings.m`.

This sub-module includes the function `ClassicalFit` which uses a truncated Hamiltonian (based on free-ion energies) to fit a given subset of the model parameters to given experimental data. It yields an extensive set of results, including fitted parameters and uncertainties.

It requires the following parameters:

- `numE`: number of electrons in the system, specifying the electronic configuration.
- `expData`: experimental data, a list of lists where each sublist represents an energy level and associated parameters. The first element of the sublists must represent energies, the other elements in the sublists are ignored but can be given to be kept together with the fitted data. The data must be ordered in increasing order of energy. **IMPORTANT.** If there are known unknown levels, these should be made explicit, anything other than a number will be interpreted as a level of undetermined energy in the corresponding gap. **ALSO IMPORTANT.** In the case of odd electron cases, `expData` needs to explicitly include the duplicate energies corresponding to Kramer's degeneracy; the gaps also need to be adequately duplicated in these cases.
- `excludeDataIndices`: indices in `expData` to be excluded from the fitting process. This can be used to exclude experiemental data which is present, but which is considered dubious. In the case of odd electron configurations these indices need to implicitly include the double degeneracy of Kramer's doublets.
- `problemVars`: symbols representing the parameters to be fitted, some of which may be constrained (set fixed or proportional to others). **IMPORTANT**. If `problemVars` is a proper subset of all the parameters needed to evaluate the simplified Hamiltonian, the values for the other necessary parameters are taken from Carnall's systematic study of LaF_3.

- `startValues`: an assocciation with the initial values for the independent parameters (given in `problemVars`. Independent parameters are those that remain once the constraints have been accounted for.
- σ`exp`: estimated uncertainty in the energy level differences between experimental and calculated values.
- `constraints`: a list of replacement rules defining constraints on the parameters. These contraints can either pin down a value, or apply proportionality ratios between them. If constrained by proportionally factors, these ratios are usually taken from Hartree-Fock calculations.

Here is a description of the different steps that this algorithm implements.

1. **Initialization**: Sets initial conditions, processes options, and prepares data structures. Manages settings like the truncation energy, logging preferences, and computational accuracy goals.
2. **Data Preparation**: Determines valid data points, excluding specified indices, and establishes truncation energy for the model.
3. **Hamiltonian Assembly and Simplification**: Constructs the Hamiltonian while preserving its block structure, applies simplification rules, and processes the diagonal blocks to retain only free-ion parameters.
4. **Intermediate Coupling Basis Calculation**: Computes an intermediate coupling basis using free-ion parameters, aiding in the energy level analysis of the system.
5. **Compilation and Truncation of Hamiltonian**: Compiles the Hamiltonian and truncates it based on the set truncation energy, optimizing for computational efficiency.
6. **Fitting Process Initialization**: Prepares variables and functions for optimization, including eigenvalue calculations and difference evaluations.
7. **Optimization**: Employs the Levenberg-Marquardt method to optimize parameters, minimizing the discrepancy between calculated and experimental energy levels.
8. **Post-Processing**: Calculates the Hamiltonian's eigensystem at the solution, deriving statistics like RMS deviation, parameter uncertainties, and covariance matrix.
9. **Output Compilation**: Aggregates all relevant data and results into the output association `solCompendium`, documenting the fitting process and outcomes.
10. **Logging and Return**: Saves the comprehensive fitting results to a log file and returns the detailed output data.

This function admits several options. Importantly here one may permit the model to have a constant shift to all the levels and the truncation energy can be set. Here one can also provide simplification rules that are applied to the compiled version of the Hamiltonian.

- `TruncationEnergy`: Determines the energy level at which the Hamiltonian is truncated. If set to `Automatic`, the truncation energy is derived from the maximum energy present in the experimental data (`expData`). Otherwise, it can be manually set to a specific value.
- `MagneticSimplifier`: Provides a list of replacement rules to simplify the magnetic parameters in the Hamiltonian, aiding in the reduction of computational complexity.

- `MagFieldSimplifier`: Offers a list of replacement rules to specify a magnetic field, enhancing the flexibility in modeling magnetic effects within the system.
- `SymmetrySimplifier`: A list of replacement rules used to simplify the crystal field components of the Hamiltonian, facilitating a more efficient fitting process.
- `OtherSimplifier`: An additional list of replacement rules applied to the Hamiltonian before computation, allowing for further customization and simplification of the model, such as disabling specific interactions or effects. **IMPORTANT**. Here the default is that the spin-spin contribution (as controlled by the σSS parameter) for the Marvin integrals is *not* included.
- `MaxHistory`: This option controls the length of the logs for the solver, enabling users to adjust the amount of log data retained during the fitting process.
- `MaxIterations`: Sets the maximum number of iterations that the fitting algorithm (`NMinimize`) will execute, allowing control over the computational effort spent on the fitting.
- `FilePrefix`: Specifies the prefix for the filenames under which the fitting results are saved. By default, the prefix is set to "calcs", and the files are saved in the "log/calcs" directory.
- `AddConstantShift`: If set to `True`, this option allows for a constant shift in the energy levels during the fitting process. This is particularly useful for fine-tuning the model to better match experimental data.
- `AccuracyGoal`: Defines the accuracy goal for the `NMinimize` function used in the fitting process, allowing users to set the desired level of precision for the fit.
- `PrintFun`: Specifies the function used to print progress messages during the fitting process. The default is `PrintTemporary`, which displays temporary output that can be useful for monitoring the fitting's progress.
- `SlackChannel`: Names the Slack channel to which progress messages will be sent. If set to `None`, this feature is disabled, and no messages are sent to Slack.
- `ProgressView`: Controls whether a progress window is displayed during the fitting process. When set to `True`, it provides an auxiliary notebook is created automatically whith plots showing the progress of `NMinimize`.
- `SignatureCheck`: If `True`, the function ends prematurely and prints the list of the symbols that define the Hamiltonian after all basic simplifications have been applied without considering the given constraints.
- `SaveEigenvectors`: Determines whether both the eigenvectors and eigenvalues of the fitted model are saved. If set to `False`, only the energies are saved.
- `AppendToLogFile`: what is provided here is appended to the log file under the "Appendix" key, enabling additional data to be stored alongside the fitting results.

The function returns an association with the following keys.

- `bestRMS`: the best root mean square deviation found during the fitting process.
- `bestParams`: the optimal set of parameters found through the fitting process.
- `paramSols`: a list of the parameter solutions at each step of the fitting algorithm.
- `timeTaken/s`: the total time taken to complete the fitting process, measured in seconds.

- `simplifier`: the replacement rules used to reduce the define the free-ion Hamiltonian.
- `excludeDataIndices`: the indices that were excluded from the fitting process as specified in the input.
- `startValues`: the initial values for the problem variables as given in the input.
- `freeIonSymbols`: symbols used in the intermediate coupling basis.
- `truncationEnergy`: the energy level at which the Hamiltonian was truncated.
- `numE`: the number of electrons in the f^{numE} configuration.
- `expData`: the experimental data used for the fitting process.
- `problemVars`: the variables considered during the fitting process.
- `maxIterations`: the maximum number of iterations used in the fitting process.
- `hamDim`: the dimension of the full Hamiltonian before simplifications or truncations.
- `allVars`: all the symbols defining the Hamiltonian under the applied simplifications.
- `freeBies`: the free-ion parameters used to define the intermediate coupling basis.
- `truncatedDim`: the dimension of the truncated Hamiltonian.
- `compiledIntermediateFname`: the file name of the compiled function used for the truncated Hamiltonian.
- `fittedLevels`: the number of levels that were fitted.
- `actualSteps`: the actual number of steps taken by the fitting algorithm.
- `solWithUncertainty`: a list of replacement rules showing the best fit value and its uncertainty for each parameter.
- `rmsHistory`: Aa list of the RMS values found during the fitting process.
- `Appendix`: an association appended to the log file under the "Appendix" key.
- `presentDataIndices`: the indices in `expData` that were used for fitting.
- `states`: a list of eigenvalues and eigenvectors for the fitted model, available if eigenvectors were saved.
- `energies`: a list of the energies of the fitted levels, adjusted if an energy shift was included in the fitting.

Table Fig-2 shows the result of fitting the experimental data included in Carnall, in which certain parameters are held fixed, others made proportional to one another, and the other fitted through the Levenberg-Marquard method.

	Ce	Pr	Nd	Pm	Sm	Eu	Gd	Tb	Dy	Ho	Er	Tm	Yb
F2	---	68860.±20.	73020.±10.	[76400.]	79700.±30.	83080.±30.	85640.±10.	88870.±20.	91830.±40.	94560.±30.	97570.±20.	100130.±20.	---
F4	---	50400.±70.	52770.±40.	[54900.]	57260.±30.	[59240.2]	[60809.]	[62834.6]	64350.±30.	66480.±40.	68050.±40.	69660.±90.	---
F6	---	32880.±60.	35750.±50.	[37700.]	40200.±20.	[42539.9]	44790.±10.	47190.±10.	49260.±20.	51900.±50.	54180.±60.	56030.±80.	---
ζ	647.±1.	749.±1.	885.1±0.8	[1025.]	1175.±1.	1332.±2.	1509.±3.	1705.±2.	1908.±1.	2139.±1.	2377.±1.	2634.±1.	2915.±1.
α	---	16.1±0.2	21.3±0.1	[20.5]	20.3±0.1	[20.16]	18.8±0.1	18.5±0.1	18.2±0.1	17.7±0.1	17.4±0.1	16.9±0.2	---
β	---	−550.±10.	−580.±10.	[−560.]	−572.±5.	[−566.9]	[−600.]	−586.±4.	−638.±6.	−615.±8.	−580.±10.	−610.±10.	---
γ	---	1360.±10.	1430.±10.	[1475.]	[1500.]	[1500.]	[1575.]	[1650.]	1802.±5.	[1800.]	[1800.]	[1820.]	---
T2	---	---	293.±4.	[300.]	[300.]	[300.]	[300.]	[320.]	315.±5.	[400.]	[400.]	[400.]	---
T3	---	---	35.±9.	[35.]	[36.]	[40.]	[42.]	[40.]	30.±10.	36.±8.	40.±10.	---	---
T4	---	---	59.±8.	[58.]	[56.]	[60.]	[62.]	[50.]	90.±40.	96.±7.	63.±9.	---	---
T6	---	---	−280.±20.	[−310.]	−330.±30.	[−300.]	[−295.]	−350.±40.	−290.±40.	−260.±50.	−280.±20.	---	---
T7	---	---	330.±20.	[350.]	360.±20.	[370.]	[350.]	320.±30.	370.±20.	300.±40.	330.±20.	---	---
T8	---	---	300.±20.	[320.]	340.±10.	[320.]	[310.]	330.±10.	320.±20.	340.±20.	360.±20.	---	---
M0	---	1.8±0.3	2.1±0.1	[2.4]	2.52±0.06	[2.1]	3.3±0.1	2.4±0.09	3.22±0.06	2.61±0.08	3.8±0.1	3.9±0.2	---
P2	---	−30.±30.	210.±10.	[275.]	330.±10.	[360.]	720.±40.	390.±20.	620.±10.	550.±20.	680.±20.	670.±40.	---
B02	[−218.]	−220.±20.	−250.±40.	[−245.]	−210.±30.	−210.±60.	[−231.]	−240.±40.	−230.±20.	[−240.]	−230.±30.	−250.±30.	[−249.]
B04	[738.]	730.±30.	500.±100.	[470.]	300.±200.	400.±100.	[604.]	600.±100.	560.±70.	500.±100.	300.±100.	450.±60.	[457.]
B06	[679.]	670.±40.	640.±40.	[640.]	600.±100.	500.±100.	[280.]	300.±100.	170.±90.	300.±100.	440.±70.	300.±60.	[282.]
B22	[−50.]	−120.±20.	−50.±30.	[−50.]	[−50.]	[−50.]	[−99.]	−100.±50.	−60.±10.	−100.±20.	−90.±30.	−100.±20.	[−105.]
B24	[431.]	420.±50.	500.±80.	[525.]	620.±50.	[597.]	[340.]	260.±70.	190.±70.	240.±60.	350.±90.	300.±40.	[320.]
B26	[−921.]	−910.±50.	−830.±40.	[−750.]	−680.±90.	[−706.]	[−721.]	−730.±80.	−670.±50.	−550.±60.	−480.±20.	−450.±20.	[−482.]
B44	[616.]	600.±30.	570.±50.	[490.]	430.±60.	[408.]	[452.]	480.±30.	550.±30.	460.±40.	300.±100.	430.±40.	[428.]
B46	[−348.]	−350.±20.	−400.±40.	[−450.]	−400.±100.	[−508.]	[−204.]	−240.±80.	−100.±100.	−200.±30.	−230.±30.	−240.±60.	[−234.]
B66	[−788.]	−780.±60.	−830.±30.	[−760.]	−730.±80.	[−692.]	[−509.]	−520.±90.	−540.±40.	−580.±30.	−500.±20.	−500.±30.	[−492.]
ϵ	−2.9	−2.1	−4.8	−8.2	−16.4	−12.5	20.8	−6.	−6.7	−4.9	−7.3	−10.4	−32.8
σ	47	16	13	4	13	17	10	9	11	9	14	11	61
σBill	51	16	14	0	13	16	10	12	12	10	19	10	38
n	7	75	146	284	233	29	70	146	198	204	127	56	5
nBill	7	75	146	0	232	29	70	146	198	204	127	56	5

Figure 2: Fitting the data from Carnall et. al using **qlanth**

```
ClassicalFit::usage="Classical[numE, expData, excludeDataIndices,
    problemVars, startValues, \[Sigma]exp, constraints_List, Options]
    fits the given expData in an f^numE configuration, by using the
    symbols in problemVars. The symbols given in problemVars may be
    constrained or held constant, this being controlled by constraints
     list which is a list of replacement rules expressing desired
    constraints. The constraints list additional constraints imposed
    upon the model parameters that remain once other simplifications
    have been \"baked\" into the compiled Hamiltonians that are used
    to increase the speed of the calculation.

Important, note that in the case of odd number of electrons the given
     data must explicitly include the Kramers degeneracy;
    excludeDataIndices must be compatible with this.

The list expData needs to be a list of lists with the only
    restriction that the first element of them corresponds to energies
     of levels. In this list, an empty value can be used to indicate
    known gaps in the data. Even if the energy value for a level is
    known (and given in expData) certain values can be omitted from
    the fitting procedure through the list excludeDataIndices, which
    correspond to indices in expData that should be skipped over.

The Hamiltonian used for fitting is version that has been truncated
    either by using the maximum energy given in expData or by manually
     setting a truncation energy using the option \"TruncationEnergy\"
```

```
    .

The argument \[Sigma]exp is the estimated uncertainty in the
    differences between the calculated and the experimental energy
    levels. This is used to estimate the uncertainty in the fitted
    parameters. Admittedly this will be a rough estimate (at least on
    the contribution of the calculated uncertainty), but it is better
    than nothing and may at least provide a lower bound to the
    uncertainty in the fitted parameters. It is assumed that the
    uncertainty in the differences between the calculated and the
    experimental energy levels is the same for all of them.

The list startValues is a list with all of the parameters needed to
    define the Hamiltonian (including the initial values for
    problemVars).

The function saves the solution to a file. The file is named with a
    prefix (controlled by the option \"FilePrefix\") and a UUID. The
    file is saved in the log sub-directory as a .m file.

Here's a description of the different parts of this function: first
    the Hamiltonian is assembled and simplified using the given
    simplifications. Then the intermediate coupling basis is
    calculated using the free-ion parameters for the given lanthanide.
     The Hamiltonian is then changed to the intermediate coupling
    basis and truncated. The truncated Hamiltonian is then compiled
    into a function that can be used to calculate the energy levels of
     the truncated Hamiltonian. The function that calculates the
    energy levels is then used to fit the experimental data. The
    fitting is done using FindMinimum with the Levenberg-Marquardt
    method.

The function returns an association with the following keys:

\"bestRMS\" which is the best \[Sigma] value found.
\"bestParams\" which is the best set of parameters found.
\"paramSols\" which is a list of the parameters during the stepping
    of the fitting algorithm.
\"timeTaken/s\" which is the time taken to find the best fit.
\"simplifier\" which is the simplifier used to simplify the
    Hamiltonian.
\"excludeDataIndices\" as given in the input.
\"starValues\" as given in the input.

\"freeIonSymbols\" which are the symbols used in the intermediate
    coupling basis.
\"truncationEnergy\" which is the energy used to truncate the
    Hamiltonian.
\"numE\" which is the number of electrons in the f^numE configuration
    .
\"expData\" which is the experimental data used for fitting.
\"problemVars\" which are the symbols considered for fitting

\"maxIterations\" which is the maximum number of iterations used by
    NMinimize.
```

```
34 \"hamDim\" which is the dimension of the full Hamiltonian.
35 \"allVars\" which are all the symbols defining the Hamiltonian under
       the aggregate simplifications.
36 \"freeBies\" which are the free-ion parameters used to define the
       intermediate coupling basis.
37 \"truncatedDim\" which is the dimension of the truncated Hamiltonian.
38 \"compiledIntermediateFname\" the file name of the compiled function
       used for the truncated Hamiltonian.
39
40 \"fittedLevels\" which is the number of levels fitted for.
41 \"actualSteps\" the number of steps that FindMiniminum actually took.
42 \"solWithUncertainty\" which is a list of replacement rules whose
       left hand sides are symbols for the used parameters and whose's
       right hand sides are lists with the best fit value and the
       uncertainty in that value.
43 \"rmsHistory\" which is a list of the \[Sigma] values found during
       the fitting.
44 \"Appendix\" which is an association appended to the log file under
       the key \"Appendix\".
45 \"presentDataIndices\" which is the list of indices in expData that
       were used for fitting, this takes into account both the empty
       indices in expData and also the indices in excludeDataIndices.
46
47 \"states\" which contains a list of eigenvalues and eigenvectors for
       the fitted model, this is only available if the option \"
       SaveEigenvectors\" is set to True; if a general shift of energy
       was allowed for in the fitting, then the energies are shifted
       accordingly.
48 \"energies\" which is a list of the energies of the fitted levels,
       this is only available if the option \"SaveEigenvectors\" is set
       to False. If a general shift of energy was allowed for in the
       fitting, then the energies are shifted accordingly.
49
50 The function admits the following options with default values:
51   \"MaxHistory\" : determines how long the logs for the solver can be
       .
52   \"MaxIterations\": determines the maximum number of iterations used
        by NMinimize.
53   \"FilePrefix\" : the prefix to use for the subfolder in the log
       filder, in which the solution files are saved, by default this is
       \"calcs\" so that the calculation files are saved under the
       directory \"log/calcs\".
54   \"AddConstantShift\" : if True then a constant shift is allowed in
       the fitting, default is False. If this is the case the variable \"
       \[Epsilon]\" is added to the list of variables to be fitted for,
       it must not be included in problemVars.
55
56   \"AccuracyGoal\": the accuracy goal used by NMinimize, default of
       5.
57   \"TrucationEnergy\": if Automatic then the maximum energy in
       expData is taken, else it takes the value set by this option. In
       all cases the energies in expData are only considered up to this
       value.
58   \"PrintFun\": the function used to print progress messages, the
       default is PrintTemporary.
```

```
59
60    \"SlackChannel\": name of the Slack channel to which to dump
        progress messaages, the default is None which disables this option
        .
61    \"ProgressView\": whether or not a progress window will be opened
        to show the progress of the solver, the default is True.
62    \"SignatureCheck\": if True then then the function returns
        prematurely, returning a list with the symbols that would have
        defined the Hamiltonian after all simplifications have been
        applied. Useful to check the entire parameter set that the
        Hamiltonian has, which has to match one-to-one what is provided by
         startingValues.
63    \"SaveEigenvectors\": if True then the both the eigenvectors and
        eigenvalues are saved under the \"states\" key of the returned
        association. If False then only the energies are saved, the
        default is False.
64
65    \"AppendToLogFile\": an association appended to the log file under
        the key \"Appendix\".
66    \"MagneticSimplifier\": a list of replacement rules to simplify the
         Marvin and pesudo-magnetic paramters. Here the ratios of the
        Marvin parameters and the pseudo-magnetic parameters are defined
        to simplify the magnetic part of the Hamiltonian.
67    \"MagFieldSimplifier\": a list of replacement rules to specify a
        magnetic field (in T), if set to {}, then {Bx, By, Bz} can also be
         used as variables to be fitted for.
68
69    \"SymmetrySimplifier\": a list of replacements rules to simplify
        the crystal field.
70    \"OtherSimplifier\": an additional list of replacement rules that
        are applied to the Hamiltonian before computing with it. Here the
        spin-spin contribution can be turned off by setting \[Sigma]SS->0,
         which is the default.
71  ";
72  Options[ClassicalFit] = {
73    "MaxHistory"       -> 200,
74    "MaxIterations"    -> 100,
75    "FilePrefix"       -> "calcs",
76    "ProgressView"     -> True,
77    "TruncationEnergy" -> Automatic,
78    "AccuracyGoal"     -> 5,
79    "PrintFun"         -> PrintTemporary,
80    "SlackChannel"     -> None,
81    "ProgressView"     -> True,
82    "SignatureCheck"   -> False,
83    "AddConstantShift" -> False,
84    "SaveEigenvectors" -> False,
85    "AppendToLogFile"  -> <||>,
86    "MagneticSimplifier" -> {
87      M2 -> 56/100 M0,
88      M4 -> 31/100 M0,
89      P4 -> 1/2 P2,
90      P6 -> 1/10 P2
91      },
92    "MagFieldSimplifier" -> {
```

```
    Bx -> 0,
    By -> 0,
    Bz -> 0
    },
  "SymmetrySimplifier" -> {
    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
    },
  "OtherSimplifier" -> {
    F0->0,
    P0->0,
    \[Sigma]SS->0,
    T11p->0, T11->0, T12->0, T14->0, T15->0,
    T16->0, T18->0, T17->0, T19->0, T2p->0
    },
  "ThreeBodySimplifier" -> <|
    1  -> {
      T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
      T11p->0, T11->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17
    ->0, T19->0,
      T2p->0},
    2 -> {
      T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
      T11p->0, T11->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17
    ->0, T19->0,
      T2p->0
      },
    3  -> {},
    4  -> {},
    5  -> {},
    6  -> {},
    7  -> {},
    8  -> {},
    9  -> {},
    10 -> {},
    11 -> {},
    12 -> {
      T3->0, T4->0, T6->0, T7->0, T8->0,
      T11p->0, T11->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17
    ->0, T19->0,
      T2p->0
      },
    13->{
      T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
      T11p->0, T11->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17
    ->0, T19->0,
      T2p->0
      }
    |>,
    "FreeIonSymbols" -> {F0, F2, F4, F6, ζ}
};
ClassicalFit[numE_Integer, expData_List, excludeDataIndices_List,
    problemVars_List, startValues_Association, \[Sigma]exp_Real,
    constraints_List, OptionsPattern[]]:=
```

```
Module[{accuracyGoal, activeVarIndices, activeVars,
  activeVarsString, activeVarsWithRange, allFreeEnergies,
  allFreeEnergiesSorted, allVars, allVarsVec,
  argsForEvalInsideOfTheIntermediateSystems,
  argsOfTheIntermediateEigensystems, aVar, aVarPosition, basis,
  basisChanger, basisChangerBlocks, bestError, bestParams, bestRMS,
  blockShifts, blockSizes, colIdx, compiledDiagonal,
  compiledIntermediateFname, constrainedProblemVars,
  constrainedProblemVarsList, covMat, currentRMS, degressOfFreedom,
  dependentVars, diagonalBlocks, diagonalScalarBlocks, diff,
  eigenEnergies, eigenvalueDispenserTemplate, eigenVectors,
  elevatedIntermediateEigensystems, endTime, fmSol, fmSolAssoc,
  fractionalWidth, freeBies, freeIenergiesAndMultiplets,
  freeionSymbols, fullHam, fullSolVec, funcString, ham, hamDim,
  hamEigenvaluesTemplate, hamString, hess, indepSolVecVec, indepVars
  , intermediateHam, isolationValues, jobVars, lin, linMat, ln,
  lnParams, logFilePrefix, logFname, magneticSimplifier,
  maxFreeEnergy, maxHistory, maxIterations, methodString,
  methodStringTemplate, minFreeEnergy, minpoly, modelSymbols,
  multipletAssingments, numBlocks, numQSignature, numReps,
  solCompendium, openNotebooks, ordering, othersFixed,
  otherSimplifier, p0, paramBest, paramSigma, perHam, polySols,
  presentDataIndices, PrintFun, problemVarsPositions, problemVarsQ,
  problemVarsQString, problemVarsVec, problemVarsWithStartValues,
  reducedModelSymbols, resultMessage, roundedTruncationEnergy,
  rowIdx, runningInteractive, shiftToggle, simplifier, slackChan,
  sol, solAssoc, sols, solWithUncertainty, sortedTruncationIndex,
  sqdiff, standardValues, starTime, startingValues, startTime,
  startVarValues, states, steps, symmetrySimplifier,
  theIntermediateEigensystems, TheIntermediateEigensystems,
  TheTruncatedAndSignedPathGenerator, thisPoly, threadHeaderTemplate
  , threadMessage, threadTS, timeTaken, totalVariance,
  truncadedFname, truncatedIntermediateBasis,
  truncatedIntermediateHam, truncationEnergy, truncationIndices,
  truncationUmbral, usingInitialRange, varHash, varIdx,
  varsWithConstants, varWithValsSignature, \[Lambda]0Vec, \[Lambda]
  exp},
(
  solCompendium   = <||>;
  addShift        = OptionValue["AddConstantShift"];
  ln              = theLanthanides[[numE]];
  maxHistory      = OptionValue["MaxHistory"];
  maxIterations   = OptionValue["MaxIterations"];
  logFilePrefix   = If[OptionValue["FilePrefix"] == "",
                       ToString[theLanthanides[[numE]]],
                       OptionValue["FilePrefix"]
                    ];
  accuracyGoal    = OptionValue["AccuracyGoal"];
  slackChan       = OptionValue["SlackChannel"];
  PrintFun        = OptionValue["PrintFun"];
  freeIonSymbols       = OptionValue["FreeIonSymbols"];
  runningInteractive   = (Head[$ParentLink] === LinkObject);
  magneticSimplifier   = OptionValue["MagneticSimplifier"];
  magFieldSimplifier   = OptionValue["MagFieldSimplifier"];
  symmetrySimplifier   = OptionValue["SymmetrySimplifier"];
```

```
161     otherSimplifier      = OptionValue["OtherSimplifier"];
162     threeBodySimplifier  = If[Head[OptionValue["ThreeBodySimplifier"]]
        == Association,
163                             OptionValue["ThreeBodySimplifier"][numE],
164                             OptionValue["ThreeBodySimplifier"]
165                           ];
166
167     truncationEnergy = If[OptionValue["TruncationEnergy"]===Automatic
        ,
168       PrintFun["Truncation energy set to Automatic, using the maximum
         energy in the data ..."];
169       Max[Select[First /@ expData, NumericQ[#] &]],
170       OptionValue["TruncationEnergy"]
171       ];
172     PrintFun["Using a truncation energy of ", truncationEnergy, " K"
        ];
173
174     simplifier           = Join[magneticSimplifier,
175                             magFieldSimplifier,
176                             symmetrySimplifier,
177                             threeBodySimplifier,
178                             otherSimplifier];
179
180     PrintFun["Determining gaps in the data ..."];
181     (* the indices that are numeric in expData whatever is non-
        numeric is assumed as a known gap *)
182     presentDataIndices = Flatten[Position[expData, {_?(NumericQ[#] &)
        , _, _}]];
183     (* some indices omitted here based on the excludeDataIndices
        argument *)
184     presentDataIndices = Complement[presentDataIndices,
        excludeDataIndices];
185
186     hamDim                              = Binomial[14, numE];
187     solCompendium["simplifier"]         = simplifier;
188     solCompendium["excludeDataIndices"] = excludeDataIndices;
189     solCompendium["startValues"]        = startValues;
190     solCompendium["freeIonSymbols"]     = freeIonSymbols;
191     solCompendium["truncationEnergy"]   = truncationEnergy;
192     solCompendium["numE"]               = numE;
193     solCompendium["expData"]            = expData;
194     solCompendium["problemVars"]        = problemVars;
195     solCompendium["maxIterations"]      = maxIterations;
196     solCompendium["hamDim"]             = hamDim;
197     modelSymbols  = Sort[Select[paramSymbols, Not[MemberQ[Join[
        racahSymbols, chenSymbols,{t2Switch, \[Epsilon],gs}],#]]&]];
198     (* remove the symbols that will be removed by the simplifier, no
        symbol should remain here that is not in the symbolic Hamiltonian
        *)
199     reducedModelSymbols = Select[modelSymbols, Not[MemberQ[Keys[
        simplifier],#]]&];
200
201     (* this is useful to understand what are the arguments of the
        truncated compiled Hamiltonian *)
202     If[OptionValue["SignatureCheck"],
```

```
    (
      Print["Given the model parameters and the simplifying
    assumptions, the resultant model parameters are:"];
      Print[{reducedModelSymbols}];
      Print["The ordering in these needs to be respected in the
    startValues parameter ..."];
      Print["Exiting ..."];
      Return[""];
    )
  ];

  (* calculate the basis *)
  PrintFun["Retrieving the LSJMJ basis for f^", numE, " ..."];
  basis    = BasisLSJMJ[numE];

  (* get the reference parameters from LaF3 *)
  PrintFun["Getting reference free-ion parameters for ",ln," using
  LaF3 ..."];
  lnParams = LoadParameters[ln];
  freeBies = Prepend[Values[(#->(#/.lnParams)) &/@ freeIonSymbols],
    numE];
  (* a more explicit alias *)
  allVars          = reducedModelSymbols;
  standardValues   = allVars /. lnParams;
  solCompendium["allVars"]  = allVars;
  solCompendium["freeBies"] = freeBies;

  (* reload compiled version if found *)
  varHash                  = Hash[{numE, allVars, freeBies,
  truncationEnergy, simplifier}];
  compiledIntermediateFname = ln<>"-compiled-intermediate-truncated
  -ham-"<>ToString[varHash]<>".mx";
  compiledIntermediateFname = FileNameJoin[{moduleDir, "compiled",
  compiledIntermediateFname}];
  solCompendium["compiledIntermediateFname"] =
  compiledIntermediateFname;

  If[FileExistsQ[compiledIntermediateFname],
    PrintFun["This ion, free-ion params, and full set of variables
  have been used before (as determined by {numE, allVars, freeBies,
  truncationEnergy,simplifier}). Loading the previously saved
  compiled function and intermediate coupling basis ..."];
    PrintFun["Using : ", compiledIntermediateFname];
    {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =
   Import[compiledIntermediateFname];,
  (
    (* grab the Hamiltonian preserving its block structure *)
    PrintFun["Assembling the Hamiltonian for f^",numE," keeping the
   block structure ..."];
    ham          = HamMatrixAssembly[numE, "ReturnInBlocks"->True];
    (* apply the simplifier *)
    PrintFun["Simplifying using the aggregate set of simplification
   rules ..."];
    ham          = Map[ReplaceInSparseArray[#, simplifier]&, ham,
  {2}];
```

```
243       PrintFun["Zeroing out every symbol in the Hamiltonian that is
    not a free-ion parameter ..."];
244       (* Get the free ion symbols *)
245       freeIonSimplifier = (#->0) & /@ Complement[reducedModelSymbols,
     freeIonSymbols];
246       (* Take the diagonal blocks for the intermediate analysis *)
247       PrintFun["Grabbing the diagonal blocks of the Hamiltonian ..."
    ];
248       diagonalBlocks       = Diagonal[ham];
249       (* simplify them to only keep the free ion symbols *)
250       PrintFun["Simplifying the diagonal blocks to only keep the free
     ion symbols ..."];
251       diagonalScalarBlocks = ReplaceInSparseArray[#,freeIonSimplifier
    ]&/@diagonalBlocks;
252       (* these include the MJ quantum numbers, remove that *)
253       PrintFun["Contracting the basis vectors by removing the MJ
    quantum numbers from the diagonal blocks ..."];
254       diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
255
256       argsOfTheIntermediateEigensystems         = StringJoin[Riffle[
    Prepend[(ToString[#]<>"v_") & /@ freeIonSymbols,"numE_"],", "]];
257       argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle[(
    ToString[#]<>"v") & /@ freeIonSymbols,", "]];
258       PrintFun["argsOfTheIntermediateEigensystems = ",
    argsOfTheIntermediateEigensystems];
259       PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
    argsForEvalInsideOfTheIntermediateSystems];
260       PrintFun["(if the following fails, it might help to see if the
    arguments of TheIntermediateEigensystems match the ones shown
    above)"];
261
262       (* compile a function that will effectively calculate the
    spectrum of all of the scalar blocks given the parameters of the
    free-ion part of the Hamiltonian *)
263       (* compile one function for each of the blocks *)
264       PrintFun["Compiling functions for the diagonal blocks of the
    Hamiltonian ..."];
265       compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate[N
    [Normal[#]]]]&/@diagonalScalarBlocks;
266       (* use that to create a function that will calculate the free-
    ion eigensystem *)
267       TheIntermediateEigensystems[numEv_, F0v_, F2v_, F4v_, F6v_, ζv_
    ] := (
268         theNumericBlocks = (#[F0v, F2v, F4v, F6v, ζv]&)/
    @compiledDiagonal;
269         theIntermediateEigensystems = Eigensystem/@theNumericBlocks;
270         Js     = AllowedJ[numEv];
271         basisJ = BasisLSJMJ[numEv,"AsAssociation"->True];
272         (* having calculated the eigensystems with the removed
    degeneracies, put the degeneracies back in explicitly *)
273         elevatedIntermediateEigensystems = MapIndexed[EigenLever[#1,2
    Js[[#2[[1]]]]+1]&, theIntermediateEigensystems];
274         (* Identify a single MJ to keep *)
275         pivot = If[EvenQ[numEv],0,-1/2];
276         LSJmultiplets = (#[[1]]<>ToString[InputForm[#[[2]]]])&/
```

```
@Select[BasisLSJMJ[numEv],#[[-1]]== pivot &];
277       (* calculate the multiplet assignments that the intermediate
    basis eigenvectors have *)
278       multipletAssingments = Table[
279         (
280           J         = Js[[idx]];
281           eigenVecs = theIntermediateEigensystems[[idx]][[2]];
282           majorComponentIndices        = Ordering[Abs[#]][[1]]&/
    @eigenVecs;
283           majorComponentAssignments    = LSJmultiplets[[#]]&/
    @majorComponentIndices;
284           (* All of the degenerate eigenvectors belong to the same
    multiplet*)
285           elevatedMultipletAssignments = ListRepeater[
    majorComponentAssignments,2J+1];
286           elevatedMultipletAssignments
287         ),
288       {idx, 1, Length[Js]}
289       ];
290       (* put together the multiplet assignments and the energies *)
291       freeIenergiesAndMultiplets = Transpose/@Transpose[{First/
    @elevatedIntermediateEigensystems, multipletAssingments}];
292       freeIenergiesAndMultiplets = Flatten[
    freeIenergiesAndMultiplets, 1];
293       (* calculate the change of basis matrix using the
    intermediate coupling eigenvectors *)
294       basisChanger = BlockDiagonalMatrix[Transpose/@Last/
    @elevatedIntermediateEigensystems];
295       basisChanger = SparseArray[basisChanger];
296       Return[{theIntermediateEigensystems, multipletAssingments,
    elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
    basisChanger}]
297     );
298
299     PrintFun["Calculating the intermediate eigensystems for ",ln,"
    using free-ion params from LaF3 ..."];
300     (* calculate intermediate coupling basis using the free-ion
    params for LaF3 *)
301     {theIntermediateEigensystems, multipletAssingments,
    elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
    basisChanger} = TheIntermediateEigensystems@@freeBies;
302
303     (* use that intermediate coupling basis to compile a function
    for the full Hamiltonian *)
304     allFreeEnergies = Flatten[First/
    @elevatedIntermediateEigensystems];
305     (* important that the intermediate coupling basis have attached
     energies, which make possible the truncation *)
306     ordering = Ordering[allFreeEnergies];
307     (* sort the free ion energies and determine which indices
    should be included in the truncation *)
308     allFreeEnergiesSorted          = Sort[allFreeEnergies];
309     {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];
310     (* determine the index at which the energy is equal or larger
    than the truncation energy *)
```

```
311       sortedTruncationIndex = Which[
312         truncationEnergy > (maxFreeEnergy-minFreeEnergy),
313         hamDim,
314         True,
315         FirstPosition[allFreeEnergiesSorted-Min[allFreeEnergiesSorted
    ],x_/;x>truncationEnergy,{0},1][[1]]
316       ];
317       (* the actual energy at which the truncation is made *)
318       roundedTruncationEnergy = allFreeEnergiesSorted[[
    sortedTruncationIndex]];
319
320       (* the indices that enact the truncation *)
321       truncationIndices = ordering[[;;sortedTruncationIndex]];
322       (* Return[{basisChanger, ham, truncationIndices}]; *)
323       PrintFun["Computing the block structure of the change of basis
    array ..."];
324       blockSizes = BlockArrayDimensionsArray[ham];
325       basisChangerBlocks = ArrayBlocker[basisChanger, blockSizes];
326       blockShifts = First /@ Diagonal[blockSizes];
327       numBlocks = Length[blockSizes];
328       (* using the ham (with all the symbols) change the basis to the
     computed one *)
329       PrintFun["Changing the basis of the Hamiltonian to the
    intermediate coupling basis ..."];
330       (* intermediateHam            = Transpose[basisChanger].ham.
    basisChanger; *)
331       (* Return[{basisChangerBlocks, ham}]; *)
332       intermediateHam = BlockMatrixMultiply[ham, basisChangerBlocks];
333       PrintFun["Distributing products inside of symbolic matrix
    elements to keep complexity in check ..."];
334       Do[
335         intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
    intermediateHam[[rowIdx, colIdx]], Distribute /@ # &];,
336         {rowIdx, 1, numBlocks},
337         {colIdx, 1, numBlocks}
338       ];
339       intermediateHam = BlockMatrixMultiply[BlockTranspose[
    basisChangerBlocks],intermediateHam];
340       PrintFun["Distributing products inside of symbolic matrix
    elements to keep complexity in check ..."];
341       Do[
342         intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
    intermediateHam[[rowIdx, colIdx]], Distribute /@ # &];,
343         {rowIdx, 1, numBlocks},
344         {colIdx, 1, numBlocks}
345       ];
346       (* using the truncation indices truncate that one *)
347       PrintFun["Truncating the Hamiltonian ..."];
348       truncatedIntermediateHam = TruncateBlockArray[intermediateHam,
    truncationIndices, blockShifts];
349       (* these are the basis vectors for the truncated hamiltonian *)
350       PrintFun["Saving the truncated intermediate basis ..."];
351       truncatedIntermediateBasis = basisChanger[[All,
    truncationIndices]];
352
```

```
353       PrintFun["Compiling a function for the truncated Hamiltonian
      ..."];
354       (* compile a function that will calculate the truncated
      Hamiltonian given the parameters in allVars, this is the function
      to be use in fitting *)
355       compileIntermediateTruncatedHam = Compile[Evaluate[allVars],
      Evaluate[truncatedIntermediateHam]];
356       (* save the compiled function *)
357       PrintFun["Saving the compiled function for the truncated
      Hamiltonian and the truncatedIntermediateBasis..."];
358       Export[compiledIntermediateFname, {
      compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
359      )
360      ];
361
362      truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
363      PrintFun["The truncated Hamiltonian has a dimension of ",
      truncationUmbral, "x", truncationUmbral, " ..."];
364      presentDataIndices = Select[presentDataIndices, # <=
      truncationUmbral &];
365      solCompendium["presentDataIndices"] = presentDataIndices;
366
367      (* the problemVars are the symbols that will be fitted for *)
368
369      PrintFun["Starting up the fitting process using the Levenberg-
      Marquardt method ..."];
370      (* using the problemVars I need to create the argument list
      including _?NumericQ *)
371      problemVarsQ = (ToString[#] <> "_?NumericQ") & /@ problemVars;
372      problemVarsQString = StringJoin[Riffle[problemVarsQ, ", "]];
373      (* we also need to have the padded arguments with the variables
      in the right position and the fixed values in the remaining ones
      *)
374      problemVarsPositions = Position[allVars, #][[1, 1]] & /@
      problemVars;
375      problemVarsString = StringJoin[Riffle[ToString /@ problemVars, ",
       "]];
376      (* to feed parameters to the Hamiltonian, which includes all
      parameters, we need to form the rist set of arguments, with fixed
      values where needed, and the variables in the right position *)
377      varsWithConstants = standardValues;
378      varsWithConstants[[problemVarsPositions]] = problemVars;
379      varsWithConstantsString = ToString[varsWithConstants];
380
381      (* this following function serves eigenvalues from the
      Hamiltonian, has memoization so it might grow to use a lot of RAM
      *)
382      Clear[HamSortedEigenvalues];
383      hamEigenvaluesTemplate = StringTemplate["
384      HamSortedEigenvalues[`problemVarsQ`]:=(
385        ham          = compileIntermediateTruncatedHam@@`
      varsWithConstants`;
386        eigenValues = Sort@Eigenvalues@ham;
387        eigenValues = eigenValues - Min[eigenValues];
388        HamSortedEigenvalues[`problemVarsString`] = eigenValues;
```

```
      Return[eigenValues]
    )"];
    hamString = hamEigenvaluesTemplate[<|
        "problemVarsQ" -> problemVarsQString,
        "varsWithConstants" -> varsWithConstantsString,
        "problemVarsString" -> problemVarsString
        |>];
    ToExpression[hamString];

    (* we also need a function that will pick the i-th eigenvalue,
    this seems unnecessary but it's needed to form the right
    functional form expected by the Levenberg-Marquardt method *)
    eigenvalueDispenserTemplate = StringTemplate["
    PartialHamEigenvalues[`problemVarsQ`][i_]:=(
      eigenVals = HamSortedEigenvalues[`problemVarsString`];
      eigenVals[[i]]
    )
    "];
    eigenValueDispenserString =
      eigenvalueDispenserTemplate[<|
        "problemVarsQ"      -> problemVarsQString,
        "problemVarsString" -> problemVarsString
        |>];
    ToExpression[eigenValueDispenserString];

    PringFun["Determining the free variables after constraints ..."];
    constrainedProblemVars = (problemVars /. constraints);
    constrainedProblemVarsList = Variables[constrainedProblemVars];
    If[addShift,
      PrintFun["Adding a constant shift to the fitting parameters ...
    "];
      constrainedProblemVarsList = Append[constrainedProblemVarsList,
     \[Epsilon]]
    ];

    indepVars = Complement[pVars, #[[1]] & /@ constraints];
    stringPartialVars = ToString/@constrainedProblemVarsList;

    paramSols  = {};
    rmsHistory = {};
    steps      = 0;
    problemVarsWithStartValues = KeyValueMap[{#1,#2} &, startValues];
    If[addShift,
      problemVarsWithStartValues = Append[problemVarsWithStartValues,
     {\[Epsilon] ,0}];
    ];
    openNotebooks     = If[runningInteractive,
              ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks
    [],
              {}];
    If[Not[MemberQ[openNotebooks,"Solver Progress"]] && OptionValue["
    ProgressView"],
      ProgressNotebook["Basic"->False]
    ];
    degressOfFreedom = Length[presentDataIndices] - Length[
```

```
    problemVars] - 1;
437 PrintFun["Fitting for ", Length[presentDataIndices], " data
    points with ", Length[problemVars], " free parameters.", " The
    effective degrees of freedom are ", degressOfFreedom, " ..."];
438
439 PrintFun["Starting the fitting process ..."];
440 startTime=Now;
441 shiftToggle = If[addShift, 1, 0];
442 sol = FindMinimum[
443   Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@
    constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
444   {j, presentDataIndices}],
445   problemVarsWithStartValues,
446   Method        -> "LevenbergMarquardt",
447   MaxIterations -> OptionValue["MaxIterations"],
448   AccuracyGoal  -> OptionValue["AccuracyGoal"],
449   StepMonitor :> (
450     steps      += 1;
451     currentRMS = Sum[(expData[[j]][[1]] - (PartialHamEigenvalues
    @@ constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2, {j,
    presentDataIndices}];
452     currentRMS = Sqrt[currentRMS / degressOfFreedom];
453     paramSols  = AddToList[paramSols, constrainedProblemVarsList,
     maxHistory];
454     rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
455     )
456   ];
457 endTime = Now;
458 timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
459 PrintFun["Solution found in ", timeTaken, "s"];
460
461 solVec = constrainedProblemVars /. sol[[-1]];
462 indepSolVec = indepVars /. sol[[-1]];
463 If[addShift,
464   \[Epsilon]Best = \[Epsilon]/. sol[[-1]],
465   \[Epsilon]Best = 0
466 ];
467 fullSolVec = standardValues;
468 fullSolVec[[problemVarsPositions]] = solVec;
469 PrintFun["Calculating the numerical Hamiltonian corresponding to
    the solution ..."];
470 fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
471 PrintFun["Calculating energies and eigenvectors ..."];
472 {eigenEnergies, eigenVectors} = Eigensystem[fullHam];
473 states = Transpose[{eigenEnergies, eigenVectors}];
474 states = SortBy[states, First];
475 eigenEnergies = First /@ states;
476 PrintFun["Shifting energies to make ground state zero of energy
    ..."];
477 eigenEnergies = eigenEnergies - eigenEnergies[[1]];
478 PrintFun["Calculating the linear approximant to each eigenvalue
    ..."];
479 allVarsVec = Transpose[{allVars}];
480 p0 = Transpose[{fullSolVec}];
481 linMat = {};
```

```
    Do[
      (
        aVarPosition = Position[allVars, aVar][[1, 1]];
        isolationValues = ConstantArray[0, Length[allVars]];
        isolationValues[[aVarPosition]] = 1;
        dependentVars = KeyValueMap[{#1, D[#2, aVar]} &, Association[
    constraints]];
        Do[
          isolationValues[[Position[allVars, dVar[[1]]][[1, 1]]]] =
    dVar[[2]],
          {dVar, dependentVars}
        ];
        perHam  = compileIntermediateTruncatedHam @@ isolationValues;
        lin     = FirstOrderPerturbation[Last /@ states, perHam];
        linMat  = Append[linMat, lin];
      ),
      {aVar, constrainedProblemVarsList[[;;-2]]}
    ];
    PrintFun["Removing the gradient of the ground state ..."];
    linMat = (# - #[[1]] & /@ linMat);
    PrintFun["Transposing derivative matrices into columns ..."];
    linMat  = Transpose[linMat];

    PrintFun["Calculating the eigenvalue vector at solution ..."];
    \[Lambda]0Vec = Transpose[{eigenEnergies[[presentDataIndices]]}];
    PrintFun["Putting together the experimental vector ..."];
    \[Lambda]exp = Transpose[{First /@ expData[[presentDataIndices
    ]]}];

    problemVarsVec = Transpose[{constrainedProblemVarsList[[;;-2]]}];
    indepSolVecVec = Transpose[{indepSolVec}];
    PrintFun["Calculating the difference between eigenvalues at
    solution ..."];
    diff = If[linMat=={},
      (\[Lambda]0Vec - \[Lambda]exp)  + \[Epsilon]Best,
      (\[Lambda]0Vec - \[Lambda]exp)  + \[Epsilon]Best + linMat[[
    presentDataIndices]].(problemVarsVec - indepSolVecVec)
    ];
    PrintFun["Calculating the sum of squares of differences around
    solution ... "];
    sqdiff        = Expand[(Transpose[diff] . diff)[[1, 1]]];
    PrintFun["Calculating the minimum (which should coincide with sol
     ..."];
    minpoly       = sqdiff /. AssociationThread[problemVars -> solVec
    ];
    fmSolAssoc    = Association[sol[[2]]];
    totalVariance = Length[presentDataIndices]*\[Sigma]exp^2;
    PrintFun["Calculating the uncertainty in the parameters ..."];
    solWithUncertainty = Table[
      (
        aVar        = constrainedProblemVarsList[[varIdx]];
        paramBest   = aVar /. fmSolAssoc;
        othersFixed = AssociationThread[Delete[
    constrainedProblemVarsList[[;;-2]], varIdx] -> Delete[indepSolVec,
     varIdx]];
```

```
527         thisPoly    = sqdiff /. othersFixed;
528         polySols    = Last /@ Last /@ Solve[thisPoly == minpoly + 1*
    totalVariance];
529         polySols    = Select[polySols, Im[#] == 0 &];
530         paramSigma  = Max[polySols] - Min[polySols];
531         (aVar -> {paramBest, paramSigma})
532       ),
533     {varIdx, 1, Length[constrainedProblemVarsList]-shiftToggle}
534     ];
535     PrintFun["Calculating the covariance matrix ..."];
536     hess = If[linmat=={},
537       {{Infinity}},
538       2 * Transpose[linMat[[presentDataIndices]]] . linMat[[
    presentDataIndices]]
539     ];
540     covMat  = If[linmat=={},
541       {{0}},
542       \[Sigma]exp^2 * Inverse[hess]
543     ];
544     bestRMS = Sqrt[minpoly / degressOfFreedom];
545     solCompendium["truncatedDim"] = truncationUmbral;
546     solCompendium["fittedLevels"] = Length[presentDataIndices];
547     solCompendium["actualSteps"] = steps;
548     solCompendium["bestRMS"]     = bestRMS;
549     solCompendium["solWithUncertainty"] = solWithUncertainty;
550     solCompendium["problemVars"] = problemVars;
551     solCompendium["paramSols"]   = paramSols;
552     solCompendium["rmsHistory"]  = rmsHistory;
553     solCompendium["Appendix"]    = OptionValue["AppendToLogFile"];
554     solCompendium["timeTaken/s"] = timeTaken;
555     solCompendium["bestParams"]  = sol[[2]];
556     If[OptionValue["SaveEigenvectors"],
557         solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]}
    &/@ (Chop /@ ShiftedLevels[states]),
558         (
559           finalEnergies = Sort[First /@ states];
560           finalEnergies = finalEnergies - finalEnergies[[1]];
561           finalEnergies = finalEnergies + \[Epsilon]Best;
562           finalEnergies  = Chop /@ finalEnergies;
563           solCompendium["energies"] = finalEnergies;
564       )
565     ];
566     logFname = LogSol[solCompendium, logFilePrefix];
567     Return[solCompendium];
568   )
569 ];
```

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 $LaF_3$and compares the results against the published values from Carnall. It also calculates magnetic dipole transition rates and oscillator strengths.

8 Additional data

8.1 Carnall et al data on Ln:LaF_3

The study of Carnall et al [Car+89] on lanthanum fluoride was a systematic review of trivalent lanthanide ions in LaF_3. In this work they fitted the experimental data for all of the lanthanide ions using the single-configuration effective Hamiltonian. In their appendices one can find their calculated values, together with the experimental values that they used for their least squares fittings. In **qlanth** this data can be accessed by invoking the command `LoadCarnall` which brings into the session an Association that has keys that have as values the tables and appendices from this article. Additionally the function `LoadParameters` can be used to query the data for the fitted parameters, which may serve as a useful starting point for the description of the lanthanides ions in hosts other than LaF_3.

```
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];
  ];
)
```

```
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] = {
```

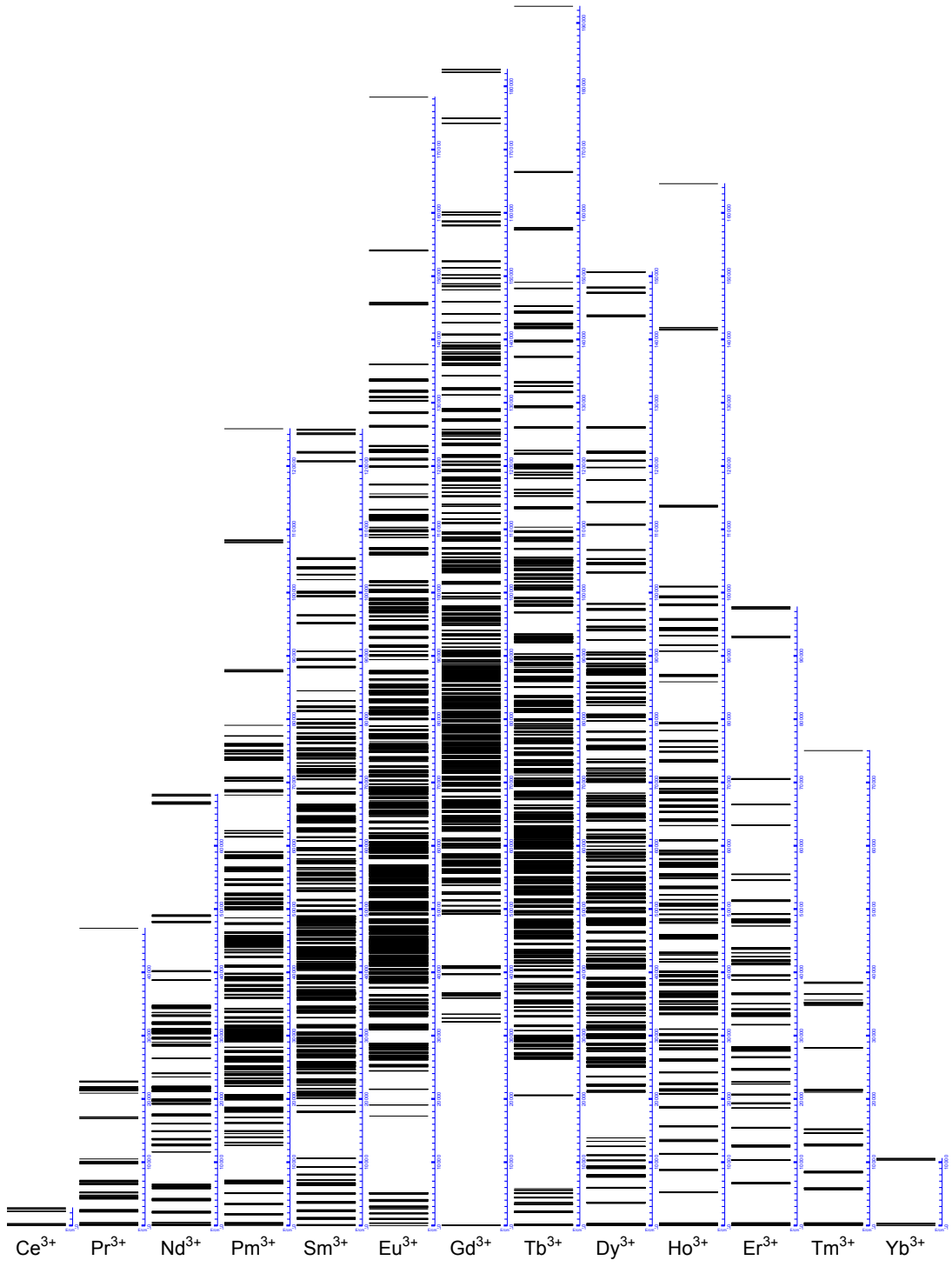

Figure 3: Dieke plot for lanthanum fluoride from data generated by **qlanth**.

```
    "Source"->"Carnall",
    "Free Ion"->False,
    "gs"->2.002319304386,
    "With Uncertainties"->False
    };
LoadParameters[Ln_String, OptionsPattern[]]:=
  Module[{source, params, uncertain, uncertainKeys, uncertainRules},
  (
    source = OptionValue["Source"];
    params = Which[source=="Carnall",
            (Association[Carnall["data"][Ln]])
            ];
    (*If a free ion then all the parameters from the crystal field
    are set to zero*)
    If[OptionValue["Free Ion"],
      Do[params[cfSymbol] = 0, {cfSymbol, cfSymbols}]
    ];
    params[F0] = 0;
    params[M2] = 0.56*params[M0]; (*See Carnall 1989,Table I,caption,
    probably fixed based on HF values*)
    params[M4] = 0.31*params[M0]; (*See Carnall 1989,Table I,caption,
    probably fixed based on HF values*)
    params[P0] = 0;
    params[P4] = 0.5*params[P2];  (*See Carnall 1989,Table I,caption,
    probably fixed based on HF values*)
    params[P6] = 0.1*params[P2];  (*See Carnall 1989,Table I,caption,
    probably fixed based on HF values*)
    params[gs] = OptionValue["gs"];
    {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[F0
    ], params[F2], params[F4], params[F6]];
    params[E0] = 0;
    If[
      Not[OptionValue["With Uncertainties"]],
      Return[params],
      (
        uncertain     = Association[Carnall["annotations"][Ln]];
        uncertainKeys = Keys[uncertain];
        uncertain     = If[# == "Not allowed to vary in fitting." ||
    # == "Interpolated",
            0., #] & /@ uncertain;
        paramKeys = Keys[params];
        uncertainVals = Sort[Intersection[paramKeys, uncertainKeys]]
    /. Association[uncertain];
        uncertainRules = MapThread[Rule, {Sort[uncertainKeys],
    uncertainVals}];
        Which[
          MemberQ[{"Ce", "Yb"}, Ln],
          (
            subsetL = {F0};
            subsetR = {0};
          ),
          True,
          (
          subsetL = {F0, M2, M4, P0, P4, P6, E0, E1, E2, E3};
          subsetR = {0, M0*0.65, M0*0.31, 0, P2*0.5, P2*0.1,
```

```
            0,
            Sqrt[(196 F2^2)/164025 + (49 F4^2)/88209 + (122500 F6^2)/134165889],
            Sqrt[F2^2/4100625 + F4^2/10673289 + (30625 F6^2)/2743558264161],
            Sqrt[F2^2/18225 + (4 F4^2)/1185921 + (30625 F6^2)/1803785841]};
        )
      ];
      uncertainRules = Join[uncertainRules, MapThread[Rule, {subsetL,subsetR} /. uncertainRules}]];
      uncertainRules = Association[uncertainRules];
      Which[
        Ln == "Eu",
        (
          uncertainRules[F4] = 12.121;
          uncertainRules[F6] = 15.872;
        ),
        Ln == "Gd",
        (
          uncertainRules[F4] = 12.07;
        ),
        Ln == "Tb",
        (
          uncertainRules[F4] = 41.006;
        )
      ];
      If[MemberQ[{"Eu","Gd","Tb"},Ln],
        (
          uncertainRules[E1] = Sqrt[(196 F2^2)/164025 + (49 F4^2)/88209 + (122500 F6^2)/134165889] /. uncertainRules;
          uncertainRules[E2] = Sqrt[F2^2/4100625 + F4^2/10673289 + (30625 F6^2)/2743558264161] /. uncertainRules;
          uncertainRules[E3] = Sqrt[F2^2/18225 + (4 F4^2)/1185921 + (30625 F6^2)/1803785841] /. uncertainRules;
        )
      ];
      uncertainKeys = First /@ Normal[uncertainRules];
      fullParams = Association[MapThread[Rule, {uncertainKeys, MapThread[Around, {uncertainKeys /. params, uncertainKeys /. uncertainRules}]}]];
      Return[Join[params, fullParams]]
    )
    ];
  )
];
```

8.2 sparsefn.py

qlanth is also accompanied by seven Python scripts sparsef[1-7].py. Each of these contains a single function effective_hamiltonian_f[1-7] which returns a sparse array for given values for the model parameters.

There is an eight Python script called basisLSJMJ.py which contains a dictionary whose keys are f1, f2, f3, f4, f5, f6, and f7, and whose values are lists that contain the ordered basis in which

the array produced by the sparsefn.py should be understood to be in. Each basis vector is a list with three elements {LS string in NK notation, J, M_J}.

In those it is left up to the user to make the adequate change of signs in the parameters for configurations above $\underline{f}^7$. These include changing the signs of all in &qn-73 and setting t2Switch to 0. For configurations at or below $\underline{f}^7$ it is necessary to set t2Switch to 1.

9 Units

All of the matrix elements of the Hamiltonian are calculated using the Kayser (K $\equiv$ 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 $\hbar = 1$.

Some constants and conversion values are included in the file qonstants.m.

```
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[];
```

10 Notation

orbital angular momentum operator of a single electron

$$\overbrace{\hat{l}} \tag{77}$$

total orbital angular momentum operator

$$\overbrace{\hat{L}} \tag{78}$$

spin angular momentum operator of a single electron

$$\overbrace{\hat{s}} \tag{79}$$

total spin angular momentum operator

$$\overbrace{\hat{S}} \tag{80}$$

Shorthand for all other quantum numbers

$$\overbrace{\Lambda} \tag{81}$$

orbital angular momentum number

$$\overbrace{\underline{\ell}} \tag{82}$$

spinning angular momentum number

$$\overbrace{\underline{s}} \tag{83}$$

Coulomb non-central potential

$$\overbrace{\hat{\mathfrak{C}}} \tag{84}$$

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

$$\overbrace{\langle \Lambda LS \| \hat{O} \| \Lambda' L' S' \rangle} \tag{85}$$

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

$$\overbrace{\langle \Lambda LSJ \| \hat{O} \| \Lambda' L' S' J' \rangle} \tag{86}$$

Spectroscopic term αLS in Russel-Saunders notation

$$\overbrace{{}^{2S+1}\alpha L \equiv |\alpha LS\rangle} \tag{87}$$

spherical tensor operator of rank k

$$\overbrace{\hat{X}^{(k)}} \tag{88}$$

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

$$\overbrace{\hat{X}^{(k)}_q} \tag{89}$$

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

$$\overbrace{\left(\underline{\ell}^{n-1}\alpha' L' S' \middle\} \underline{\ell}^n \alpha LS\right)} \tag{90}$$

11 Definitions

$$\overbrace{[\![x]\!]}^{\text{two plus one}} \coloneqq 2x+1 \tag{91}$$

$$\overbrace{\hat{u}^{(k)}}^{\text{irreducible unit tensor operator of rank k}} \tag{92}$$

$$\overbrace{\hat{U}^{(k)} \coloneqq \sum_{i=1}^{n} \hat{u}^{(k)}}^{\text{symmetric unit tensor operator for n equivalent electrons}} \tag{93}$$

$$\overbrace{\mathcal{C}_q^{(k)} \coloneqq \sqrt{\frac{4\pi}{2k+1}}\mathrm{Y}_q^{(k)}}^{\text{Renormalized spherical harmonics}} \tag{94}$$

$$\overbrace{\triangle(j_1,j_2,j_3) \coloneqq \begin{cases} 1 & \text{if } j_1 = (j_2+j_3),(j_2+j_3-1),\ldots,|j_2-j_3| \\ 0 & \text{otherwise} \end{cases}}^{\text{Triangle ``delta'' between } j_1,j_2,j_3} \tag{95}$$

12 code

12.1 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.

+ Thorne, Anne, Ulf Litz n , and Sveneric Johansson. Spectrophysics:
Principles and Applications. Springer Science & Business Media,
1999.

+ 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.

+ Judd, BR, and MA Suskin. "Complete Set of Orthogonal Scalar
Operators for the Configuration f^3." JOSA B 1, no. 2 (1984):
261-65. https://doi.org/10.1364/JOSAB.1.000261.

+ Carnall, W. T., G. L. Goodman, K. Rajnak, and R. S. Rana. "A
Systematic Analysis of the Spectra of the Lanthanides Doped into
Single Crystal LaF3." The Journal of Chemical Physics 90, no. 7
(1989): 3443-57. https://doi.org/10.1063/1.455853.

+ Hansen, JE, BR Judd, and Hannah Crosswhite. "Matrix Elements of
Scalar Three-Electron Operators for the Atomic f-Shell." Atomic Data
and Nuclear Data Tables 62, no. 1 (1996): 1-49.
https://doi.org/10.1006/adnd.1996.0001.

+ Velkov, Dobromir. "Multi-Electron Coefficients of Fractional
Parentage for the p, d, and f Shells." John Hopkins University,
2000. The B1F_ALL.TXT file is from this thesis.

+ Dodson, Christopher M., and Rashid Zia. "Magnetic Dipole and
Electric Quadrupole Transitions in the Trivalent Lanthanide Series:
Calculated Emission Rates and Oscillator Strengths." Physical Review
B 86, no. 12 (September 5, 2012): 125102.
https://doi.org/10.1103/PhysRevB.86.125102.


---------------------------------------------------------------- *)
```

```
105
106 BeginPackage["qlanth`"];
107 Needs["qonstants`"];
108 Needs["qplotter`"];
109 Needs["misc`"];
110
111 paramAtlas = "
112 E0: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
113 E1: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
114 E2: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
115 E3: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
116
117 ζ: spin-orbit strength parameter.
118
119 F0: Direct Slater integral F^0, produces an overall shift of all
        energy levels.
120 F2: Direct Slater integral F^2
121 F4: Direct Slater integral F^4, possibly constrained by ratio to F^2
122 F6: Direct Slater integral F^6, possibly constrained by ratio to F^2
123
124 M0: 0th Marvin integral
125 M2: 2nd Marvin integral
126 M4: 4th Marvin integral
127 \[Sigma]SS: spin-spin override, if 0 spin-spin is omitted, if 1 then
        spin-spin is included
128
129 T2:  three-body effective operator parameter T^2 (non-orthogonal)
130 T2p: three-body effective operator parameter T^2' (orthogonalized T2)
131 T3:  three-body effective operator parameter T^3
132 T4:  three-body effective operator parameter T^4
133 T6:  three-body effective operator parameter T^6
134 T7:  three-body effective operator parameter T^7
135 T8:  three-body effective operator parameter T^8
136
137 T11:  three-body effective operator parameter T^11
138 T11p: three-body effective operator parameter T^11'
139 T12:  three-body effective operator parameter T^12
140 T14:  three-body effective operator parameter T^14
141 T15:  three-body effective operator parameter T^15
142 T16:  three-body effective operator parameter T^16
143 T17:  three-body effective operator parameter T^17
144 T18:  three-body effective operator parameter T^18
145 T19:  three-body effective operator parameter T^19
146
147 P0: pseudo-magnetic parameter P^0
148 P2: pseudo-magnetic parameter P^2
149 P4: pseudo-magnetic parameter P^4
150 P6: pseudo-magnetic parameter P^6
151
152 gs: electronic gyromagnetic ratio
153
154 α: Trees' parameter α describing configuration interaction via the
        Casimir operator of SO(3)
155 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 (1 for f7
    or below, 0 for f8 or above)
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;

(* Parameter families *)

racahSymbols    = {E0, E1, E2, E3};
```

```
chenSymbols     = {wChErrA, wChErrB};
slaterSymbols   = {F0, F2, F4, F6};
controlSymbols  = {t2Switch, \[Sigma]SS};
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};
pseudoMagneticSymbols = {P0, P2, P4, P6};
marvinSymbols          = {M0, M2, M4};
magneticSymbols        = {Bx, By, Bz, gs, ζ};
casimirSymbols         = {α, β, γ};
paramFamilies          = Hold[{racahSymbols, chenSymbols,
    slaterSymbols, controlSymbols, cfSymbols, TSymbols,
    pseudoMagneticSymbols, marvinSymbols, casimirSymbols,
    magneticSymbols}];
ReleaseHold[Protect /@ paramFamilies];

(* Parameter usage *)
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}
];

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;
EigenLever;
EtoF;
ExportmZip;
fsubk;
fsupk;

FindNKLSTerm;
FindSL;

FreeHam;
FtoE;
GG2U;
GSO7W;
GenerateCFP;
GenerateCFPAssoc;

GenerateCFPTable;
GenerateCrystalFieldTable;
GenerateElectrostaticTable;
GenerateReducedUkTable;
GenerateReducedV1kTable;

GenerateSOOandECSOLSTable;
GenerateSOOandECSOTable;
GenerateSpinOrbitTable;
GenerateSpinSpinTable;
GenerateT22Table;

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

HamiltonianMatrixPlot;
HoleElectronConjugation;
IonSolver;
ImportMZip;
JJBlockMatrix;
```

```
JJBlockMagDip;
JJBlockMatrixFileName;

JJBlockMatrixTable;
LabeledGrid;
ListRepeater;
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 which helps keep down the complexity of the resulting
   expressions *)
SimplifyFun = Expand;

Begin["`Private`"]

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

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

  TPO::usage = "TPO[x, y, ...] gives the product of 2x+1, 2y+1, ...";
  TPO[args__] := Times @@ ((2*# + 1) & /@ {args});
```

```
417
418  Phaser::usage = "Phaser[x] gives (-1)^x.";
419  Phaser[exponent_] := ((-1)^exponent);
420
421  TriangleCondition::usage = "TriangleCondition[a, b, c] evaluates
       the triangle condition on a, b, and c.";
422  TriangleCondition[a_, b_, c_] := (Abs[b - c] <= a <= (b + c));
423
424  TriangleAndSumCondition::usage = "TriangleAndSumCondition[a, b, c]
       evaluates the joint satisfaction of the triangle and sum
       conditions.";
425  TriangleAndSumCondition[a_, b_, c_] := (
426    And[
427      Abs[b - c] <= a <= (b + c),
428      IntegerQ[a + b + c]
429      ]
430    );
431
432  SquarePrimeToNormal::usage = "SquarePrimeToNormal[squarePrime]
       evaluates the standard representation of a number from the squared
        prime representation given in the list squarePrime. For
       squarePrime of the form {c0, c1, c2, c3, ...} this function
       returns the number c0 * Sqrt[p1^c1 * p2^c2 * p3^c3 * ...] where pi
        is the ith prime number. Exceptionally some of the ci might be
       letters in which case they have to be one of \"A\", \"B\", \"C\",
       \"D\" with them corresponding to 10, 11, 12, and 13, respectively.
       ";
433  SquarePrimeToNormal[squarePrime_] :=
434  (
435    radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {
       idx1, 2, Length[squarePrime]}];
436    radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" ->
       13};
437    val = squarePrime[[1]] * Sqrt[radical];
438    Return[val];
439  );
440
441  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.
442  The function additionally takes an option \"Print\" that if set to
       True, will print the symbols that were not present in the given
       association.";
443  Options[ParamPad] = {"Print" -> True}
444  ParamPad[params_, OptionsPattern[]] := (
445    notPresentSymbols = Complement[paramSymbols, Keys[params]];
446    If[OptionValue["Print"],
447      Print["Following symbols were not given and are being set to 0:
       ",
448      notPresentSymbols]
449    ];
450    newParams = Transpose[{paramSymbols, ConstantArray[0, Length[
       paramSymbols]]}];
```

```
    newParams = (#[[1]] -> #[[2]]) & /@ newParams;
    newParams = Association[newParams];
    newParams = Join[newParams, params];
    Return[newParams];
    )

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

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.";
ReducedUk[numE_, l_, SL_, SpLp_, k_]:=Module[
  {spin, orbital, Uk, S, L, Sp, Lp, Sb, Lb, parentSL, cfpSL,
  cfpSpLp, Ukval, SLparents, SLpparents, commonParents, phase},
    {spin, orbital} = {1/2, 3};
    {S, L}          = FindSL[SL];
    {Sp, Lp}        = FindSL[SpLp];
    If[Not[S == Sp],
      Return[0]
    ];
    cfpSL       = CFP[{numE, SL}];
    cfpSpLp     = CFP[{numE, SpLp}];
    SLparents   = First /@ Rest[cfpSL];
    SLpparents  = First /@ Rest[cfpSpLp];
    commonParents = Intersection[SLparents, SLpparents];
    Uk = Sum[(
      {Sb, Lb} = FindSL[\[Psi]b];
      Phaser[Lb] *
        CFPAssoc[{numE, SL, \[Psi]b}] *
        CFPAssoc[{numE, SpLp, \[Psi]b}] *
        SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
    ),
    {\[Psi]b, commonParents}
    ];
    phase      = Phaser[orbital + L + k];
    prefactor  = numE * phase * Sqrt[TPO[L,Lp]];
    Ukval      = prefactor*Uk;
    Return[Ukval];
]

Ck::usage = "Ck[orbital, k] gives the diagonal reduced matrix
  element <l||C^(k)||l> where the Subscript[C, q]^(k) are
  renormalized spherical harmonics. See equation 11.23 in TASS with
  l=l'.";
Ck[orbital_, k_] := (-1)^orbital * TPO[orbital] * ThreeJay[{orbital
  , 0}, {k, 0}, {orbital, 0}];

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 and short-circuiting values based on triangle
  conditions.";
SixJay[{j1_, j2_, j3_}, {j4_, j5_, j6_}] := (
  sixJayval = Which[
    Not[TriangleAndSumCondition[j1, j2, j3]],
```

```
      0,
      Not[TriangleAndSumCondition[j1, j5, j6]],
      0,
      Not[TriangleAndSumCondition[j4, j2, j6]],
      0,
      Not[TriangleAndSumCondition[j4, j5, j3]],
      0,
      True,
      SixJSymbol[{j1, j2, j3}, {j4, j5, j6}]];
   SixJay[{j1, j2, j3}, {j4, j5, j6}] = sixJayval);

ThreeJay::usage = "ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] gives the
   value of the Wigner 3j-symbol and memorizes the computed value.";
ThreeJay[{j1_, m1_}, {j2_, m2_}, {j3_, m3_}] := (
 threejval = Which[
   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]
  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."
```

```
    ];
587   counter = 1;
588   If[And[OptionValue["Progress"], frontEndAvailable],
589   progBar = PrintTemporary[
590       Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
591         counter}]]]
592     ];
593   ReducedV1kTable = Table[
594     (
595       counter = counter+1;
596       {n, SL, SpLp, 1} -> SimplifyFun[ReducedV1k[n, SL, SpLp, 1]]
597     ),
598     {n, 1, numEmax},
599     {SL, AllowedNKSLTerms[n]},
600     {SpLp, AllowedNKSLTerms[n]}
601   ];
602   ReducedV1kTable = Association[ReducedV1kTable];
603   If[And[OptionValue["Progress"], frontEndAvailable],
604     NotebookDelete[progBar]
605   ];
606   exportFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m
    "}];
607   If[OptionValue["Export"],
608     (
609       Print["Exporting to file "<>ToString[exportFname]];
610       Export[exportFname, ReducedV1kTable];
611     )
612   ];
613   Return[ReducedV1kTable];
614 )
615
616 (* ###################### Racah Algebra ###################### *)
617 (* ########################################################### *)
618
619 (* ########################################################### *)
620 (* ###################### Electrostatic ###################### *)
621
622 fsubk::usage = "fsubk[numE, orbital, SL, SLp, k] gives the Slater
    integral f_k for the given configuration and pair of SL terms. See
     equation 12.17 in TASS.";
623 fsubk[numE_, orbital_, NKSL_, NKSLp_, k_]:=Module[
624   {terms, S, L, Sp, Lp, termsWithSameSpin, SL, fsubkVal,
    spinMultiplicity, prefactor, summand1, summand2},
625   (
626     {S, L}   = FindSL[NKSL];
627     {Sp, Lp} = FindSL[NKSLp];
628     terms = AllowedNKSLTerms[numE];
629     (* sum for summand1 is over terms with same spin *)
630     spinMultiplicity  = 2*S + 1;
631     termsWithSameSpin = StringCases[terms, ToString[
    spinMultiplicity] ~~ __];
632     termsWithSameSpin = Flatten[termsWithSameSpin];
633     If[Not[{S, L} == {Sp, Lp}],
634       Return[0]
635     ];
```

```
636       prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
637       summand1 = Sum[(
638           ReducedUkTable[{numE, orbital, SL, NKSL,  k}] *
639           ReducedUkTable[{numE, orbital, SL, NKSLp, k}]
640           ),
641         {SL, termsWithSameSpin}
642       ];
643       summand1 = 1 / TPO[L] * summand1;
644       summand2 = (
645         KroneckerDelta[NKSL, NKSLp] *
646           (numE *(4*orbital + 2 - numE)) /
647           ((2*orbital + 1) * (4*orbital + 1))
648         );
649       fsubkVal = prefactor*(summand1 - summand2);
650       Return[fsubkVal];
651     )
652   ];
653
654   fsupk::usage = "fsupk[numE, orbital, SL, SLp, k] gives the
        superscripted Slater integral f^k = Subscript[f, k] * Subscript[D,
        k].";
655   fsupk[numE_, orbital_, NKSL_, NKSLp_ ,k_]:= (
656     Dk[k] * fsubk[numE, orbital, NKSL, NKSLp, k]
657     )
658
659   Dk::usage = "D[k] gives the 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.";
660   Dk[k_] := {1, 225, 1089, 184041/25}[[k/2+1]];
661
662   FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the Racah parameters
        {E0, E1, E2, E3} corresponding to the given Slater integrals.
663   See eqn. 2-80 in Wybourne.
664   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.";
665   FtoE[F0_, F2_, F4_, F6_]:=Module[
666     {E0, E1, E2, E3},
667     (
668       E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
669       E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
670       E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;
671       E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
672       Return[{E0, E1, E2, E3}];
673     )
674   ];
675
676   EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the Slater integral
        parameters {F0, F2, F4, F6} corresponding to the given Racah
        parameters {E0, E1, E2, E3}. This is the inverse of the FtoE
        function.";
677   EtoF[E0_, E1_, E2_, E3_]:=Module[
678     {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.";
724 Options[GenerateElectrostaticTable] = {"Export" -> True, "
    Coefficients" -> "Slater"};
725 GenerateElectrostaticTable[numEmax_Integer:7, OptionsPattern[]]:= (
726   ElectrostaticTable = Table[
727     {numE, SL, SpLp} -> SimplifyFun[Electrostatic[{numE, SL, SpLp},
     "Coefficients" -> OptionValue["Coefficients"]]],
728     {numE, 1, numEmax},
729     {SL, AllowedNKSLTerms[numE]},
730     {SpLp, AllowedNKSLTerms[numE]}
731   ];
732   ElectrostaticTable = Association[Flatten[ElectrostaticTable]];
733   If[OptionValue["Export"],
734     Export[FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"
    }],
735     ElectrostaticTable];
736   ];
737   Return[ElectrostaticTable];
738 );
739
740 (* ###################### Electrostatic ###################### *)
741 (* ########################################################### *)
742
743 (* ########################################################### *)
744 (* ######################### Bases ########################### *)
745
746 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}.";
747 BasisTableGenerator[numE_]:=Module[
748   {energyStatesTable, allowedJ, J, Jp},
749   (
750     energyStatesTable = <||>;
751     allowedJ = AllowedJ[numE];
752     Do[
753     (
754       energyStatesTable[{numE, J}] = EnergyStates[numE, J];
755     ),
756     {Jp, allowedJ},
757     {J,  allowedJ}];
758     Return[energyStatesTable]
759   )
760 ];
761
762 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}.
763 The option \"AsAssociation\" can be set to True to return the basis
     as an association with the keys being the allowed J values. The
```

```
  default is False.
";
Options[BasisLSJMJ] = {"AsAssociation" -> False};
BasisLSJMJ[numE_, OptionsPattern[]]:=Module[
  {energyStatesTable, basis, idx1},
  (
    energyStatesTable = BasisTableGenerator[numE];
    basis = Table[
      energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
      {idx1, 1, Length[AllowedJ[numE]]}];
    basis = Flatten[basis, 1];
    If[OptionValue["AsAssociation"],
      (
        Js    = AllowedJ[numE];
        basis = Table[(J -> Select[basis, #[[2]] == J &]), {J, Js
  }];
        basis = Association[basis];
      )
    ];
    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;
```

```
852       fullShell  = 2 * halfFilled;
853       parentMax  = 2 * orbital;
854
855       PhaseFun  = <|
856           "Judd" -> JuddCFPPhase,
857           "NK" -> NKCFPPhase|>[OptionValue["PhaseFunction"]];
858       PrintTemporary["Calculating CFPs using the phase system from ",
      PhaseFun];
859       (* Initialize everything with lists to be filled in the next Do
     *)
860       complementaryCFPs =
861           Table[
862           ({numE, term} -> {term}),
863           {numE, halfFilled + 1, fullShell - 1, 1},
864           {term, AllowedNKSLTerms[numE]
865           }];
866       complementaryCFPs = Association[Flatten[complementaryCFPs]];
867       Do[(
868           daughter          = parent + 1;
869           conjugateDaughter = fullShell - parent;
870           conjugateParent   = conjugateDaughter - 1;
871           parentTerms       = AllowedNKSLTerms[parent];
872           daughterTerms     = AllowedNKSLTerms[daughter];
873           Do[
874           (
875               parentCFPs             = Rest[CFP[{daughter,
     daughterTerm}]];
876               daughterSeniority      = Seniority[daughterTerm];
877               {daughterS, daughterL} = FindSL[daughterTerm];
878               Do[
879               (
880                   {parentTerm, parentCFPval} = parentCFP;
881                   {parentS, parentL}         = FindSL[parentTerm];
882                   parentSeniority            = Seniority[parentTerm];
883                   phase = PhaseFun[parent, parentS, parentL,
884                                    daughterS, daughterL,
885                                    parentSeniority, daughterSeniority
     ];
886                   prefactor = (daughter * TPO[daughterS, daughterL])
     /
887                               (conjugateDaughter * TPO[parentS,
     parentL]);
888                   prefactor = Sqrt[prefactor];
889                   newCFPval = phase * prefactor * parentCFPval;
890                   key = {conjugateDaughter, parentTerm};
891                   complementaryCFPs[key] = Append[complementaryCFPs[
     key], {daughterTerm, newCFPval}]
892               ),
893               {parentCFP, parentCFPs}
894               ]
895           ),
896           {daughterTerm, daughterTerms}
897           ]
898           ),
899       {parent, 1, parentMax}
```

```
900         ];
901
902         complementaryCFPs[{14, "1S"}] = {"1S", {"2F",1}};
903         extendedCFPs          = Join[CFP, complementaryCFPs];
904         If[OptionValue["Export"];,
905         (
906             exportFname = FileNameJoin[{moduleDir, "data", "
      CFPs_extended.m"}];
907             Print["Exporting to ", exportFname];
908             Export[exportFname, extendedCFPs];
909         )
910         ];
911         Return[extendedCFPs];
912     )
913     ];
914
915     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.
916     The data being parsed here is the file attachment B1F_ALL.TXT which
         comes from Velkov's thesis.";
917     Options[GenerateCFPTable] = {"Export" -> True};
918     GenerateCFPTable[OptionsPattern[]]:=Module[
919       {rawText, rawLines, leadChar, configIndex, line, daughter,
920       lineParts, numberCode, parsedNumber, toAppend, CFPTablefname},
921       (
922         CleanWhitespace[string_]     := StringReplace[string,
      RegularExpression["\\s+"]->" "];
923         AddSpaceBeforeMinus[string_] := StringReplace[string,
      RegularExpression["(?<!\\s)-"]->" -"];
924         ToIntegerOrString[list_]     := Map[If[StringMatchQ[#,
      NumberString], ToExpression[#], #] &, list];
925         CFPTable       = ConstantArray[{},7];
926         CFPTable[[1]] = {{"2F",{"1S",1}}};
927
928         (* Cleaning before processing is useful *)
929         rawText   = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT
      "}]];
930         rawLines = StringTrim/@StringSplit[rawText,"\n"];
931         rawLines = Select[rawLines,#!=""&];
932         rawLines = CleanWhitespace/@rawLines;
933         rawLines = AddSpaceBeforeMinus/@rawLines;
934
935         Do[(
936           (* the first character can be used to identify the start of a
       block *)
937           leadChar=StringTake[line,{1}];
938           (* ..FN, N is at position 50 in that line *)
939           If[leadChar=="[",
940           (
941             configIndex=ToExpression[StringTake[line,{50}]];
942             Continue[];
943           )
944           ];
```

```
945         (* Identify which daughter term is being listed *)
946         If[StringContainsQ[line,"[DAUGHTER TERM]"],
947           daughter=StringSplit[line,"["][[1]];
948           CFPTable[[configIndex]]=Append[CFPTable[[configIndex]],{
    daughter}];
949           Continue[];
950         ];
951         (* Once we get here we are already parsing a row with
    coefficient data *)
952         lineParts    = StringSplit[line," "];
953         parent       = lineParts[[1]];
954         numberCode   = ToIntegerOrString[lineParts[[3;;]]];
955         parsedNumber = SquarePrimeToNormal[numberCode];
956         toAppend     = {parent,parsedNumber};
957         CFPTable[[configIndex]][[-1]] = Append[CFPTable[[configIndex
    ]][[-1]], toAppend]
958       ),
959       {line,rawLines}];
960       If[OptionValue["Export"],
961         (
962         CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"
    }];
963         Export[CFPTablefname, CFPTable];
964         )
965       ];
966       Return[CFPTable];
967     )
968   ];
969
970   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.";
971   Options[GenerateCFPAssoc] = {"Export" -> True};
972   GenerateCFPAssoc[OptionsPattern[]]:= (
973     CFPAssoc = Association[];
974     Do[
975       (daughterTerms = AllowedNKSLTerms[numE];
976       parentTerms    = AllowedNKSLTerms[numE - 1];
977       Do[
978         (
979         cfps = CFP[{numE, daughter}];
980         cfps = cfps[[2 ;;]];
981         parents = First /@ cfps;
982         Do[
983           (
984           key = {numE, daughter, parent};
985           cfp = If[
986             MemberQ[parents, parent],
987             (
988               idx = Position[parents, parent][[1, 1]];
989               cfps[[idx]][[2]]
990             ),
991             0
```

```
992             ];
993           CFPAssoc[key] = cfp;
994           ),
995           {parent, parentTerms}
996           ]
997         ),
998         {daughter, daughterTerms}
999         ]
1000      ),
1001      {numE, 1, 14}
1002      ];
1003    If[OptionValue["Export"],
1004      (
1005      CFPAssocfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"
      }];
1006      Export[CFPAssocfname, CFPAssoc];
1007      )
1008    ];
1009    Return[CFPAssoc];
1010  );
1011
1012  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.
1013  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.
1014  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.
1015  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}, ...}.
1016  Only the one-body coefficients for f-electrons are provided.
1017  In all cases it must be that 1 <= n <= 7.
1018  ";
1019  CFPTerms[numE_] := Part[CFPTable, numE]
1020  CFPTerms[numE_, SL_]:=Module[
1021    {NKterms, CFPconfig},
1022    (
1023      NKterms   = {{}};
1024      CFPconfig = CFPTable[[numE]];
1025      Map[
1026        If[StringFreeQ[First[#], SL],
1027          Null,
1028          NKterms = Join[NKterms, {#}, 1]
1029        ] &,
1030      CFPconfig
1031      ];
1032      NKterms = DeleteCases[NKterms, {}]
1033    )
1034  ];
```

```
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 ###################### *)

  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]);

    (* Parse table 1 from Judd 1984 *)
    judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv
    "}];
    data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
    headers = data[[1]];
    data = data[[2 ;;]];
    data = Transpose[data];
    \[Psi] = Select[data[[1]], # != "" &];
    \[Psi]p = Select[data[[2]], # != "" &];
    matrixKeys = Transpose[{\[Psi], \[Psi]p}];
    data = data[[3 ;;]];
    cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data
```

```
}];
cols = Select[cols, Length[#] == 21 &];
tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
tab1 = Transpose[Prepend[Transpose[tab1], headers]];

(* Parse table 2 from Judd 1984 *)
judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"}];
data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
headers = data[[1]];
data = data[[2 ;;]];
data = Transpose[data];
{operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} = data[[;; 4]];
multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
multiFactorValues = AssociationThread[multiFactorSymbols -> multiFactorValues];

(*scale values of table 1 given the values in table 2*)
oppyS = {};
normalTable =
  Table[header = col[[1]];
    If[StringContainsQ[header, " "],
      (
        multiplierSymbol = StringSplit[header, " "][[1]];
        multiplierValue = multiFactorValues[multiplierSymbol];
        operatorSymbol = StringSplit[header, " "][[2]];
        oppyS = Append[oppyS, operatorSymbol];
      ),
      (
        multiplierValue = 1;
        operatorSymbol = header;
      )
    ];
    normalValues = 1/multiplierValue*col[[2 ;;]];
    Join[{operatorSymbol}, normalValues], {col, tab1[[3 ;;]]}
  ];

(*Create an association for the matrix elements in the f^3 config*)
juddOperators = Association[];
Do[(
  col      = normalTable[[colIndex]];
  opLabel  = col[[1]];
  opValues = col[[2 ;;]];
  opMatrix = AssociationThread[matrixKeys -> opValues];
  Do[(
    opMatrix[Reverse[mKey]] = opMatrix[mKey]
    ),
  {mKey, matrixKeys}
  ];
  juddOperators[{3, opLabel}] = opMatrix),
  {colIndex, 1, Length[normalTable]}
];

```

```
1179      (* special case of t2 in f3 *)
1180      (* this is the same as getting the matrix elements from Judd 1966
           *)
1181      numE = 3;
1182      e3Op      = juddOperators[{3, "e_{3}"}];
1183      t2prime   = juddOperators[{3, "t_{2}^{'}"}];
1184      prefactor = 1/(70 Sqrt[2]);
1185      t2Op = (# -> (t2prime[#] + prefactor*e3Op[#])) & /@ Keys[t2prime
          ];
1186      t2Op = Association[t2Op];
1187      juddOperators[{3, "t_{2}"}] = t2Op;
1188
1189      (*Special case of t11 in f3*)
1190      t11 = juddOperators[{3, "t_{11}"}];
1191      eβprimeOp = juddOperators[{3, "e_{\\beta}^{'}"}];
1192      t11primeOp = (# -> (t11[#] + Sqrt[3/385] eβprimeOp[#])) & /@ Keys
          [t11];
1193      t11primeOp = Association[t11primeOp];
1194      juddOperators[{3, "t_{11}^{'}"}] = t11primeOp;
1195      If[OptionValue["Export"],
1196        (
1197          (*export them*)
1198          PrintTemporary["Exporting ..."];
1199          exportFname = FileNameJoin[{moduleDir, "data", "juddOperators
          .m"}];
1200          Export[exportFname, juddOperators];
1201        )
1202      ];
1203      Return[juddOperators];
1204    );
1205
1206  GenerateThreeBodyTables::usage="This function generates the matrix
        elements for the three body operators using the coefficients of
        fractional parentage, including those beyond f^7.";
1207  Options[GenerateThreeBodyTables] = {"Export" -> False};
1208  GenerateThreeBodyTables[nmax_Integer : 14, OptionsPattern[]] := (
1209    tiKeys = {"t_{2}", "t_{2}^{'}", "t_{3}", "t_{4}", "t_{6}", "t_{7}",
1210      "t_{8}", "t_{11}", "t_{11}^{'}", "t_{12}", "t_{14}", "t_{15}",
1211      "t_{16}", "t_{17}", "t_{18}", "t_{19}"};
1212    TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
1213    juddOperators = ParseJudd1984[];
1214    (* 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. *)
1215    op3MatrixElement[SL_, SpLp_, opSymbol_] := (
1216      jOP = juddOperators[{3, opSymbol}];
1217      key = {SL, SpLp};
1218      val = If[MemberQ[Keys[jOP], key],
1219        jOP[key],
1220        0];
1221      Return[val];
1222      );
1223    (* 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. *)
1224  Options[ti] = {"Fast" -> True};
1225  ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]]:=Module
      [
1226    {
1227      nn, S, L, Sp, Lp,
1228      cfpSL, cfpSpLp,
1229      parentSL, parentSpLp, tnk, tnks
1230    },
1231    (
1232      {S, L}   = FindSL[SL];
1233      {Sp, Lp} = FindSL[SpLp];
1234      fast     = OptionValue["Fast"];
1235      numH = 14 - nE;
1236      If[fast && Not[MemberQ[{"t_{2}","t_{11}"},tiKey]] && nE > 7,
1237        Return[-tktable[{numH, SL, SpLp, tiKey}]]
1238      ];
1239      If[(S == Sp && L == Lp),
1240      (
1241        cfpSL   = CFP[{nE, SL}];
1242        cfpSpLp = CFP[{nE, SpLp}];
1243        tnks = Table[(
1244            parentSL   = cfpSL[[nn, 1]];
1245            parentSpLp = cfpSpLp[[mm, 1]];
1246            cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
1247            tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
1248            ),
1249            {nn, 2, Length[cfpSL]},
1250            {mm, 2, Length[cfpSpLp]}
1251            ];
1252        tnk = Total[Flatten[tnks]];
1253        ),
1254      tnk = 0;
1255      ];
1256      Return[ nE / (nE - opOrder) * tnk];
1257    )
1258  ];
1259  (*Calculate the matrix elements of t^i for n up to nmax*)
1260  tktable = <||>;
1261  Do[(
1262    Do[(
1263      tkValue = Which[numE <= 2,
1264        (*Initialize n=1,2 with zeros*)
1265        0,
1266        numE == 3,
1267        (*Grab matrix elem in f^3 from Judd 1984*)
1268        SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
1269        True,
1270        SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2,
     3]]]
1271        ];
1272      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 a factor proportional to the operator z13 is
  subtracted since its effect is 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.
Rigurously speaking the Pk parameters here are subscripted. The
  conversion to superscripted parameters is performed elsewhere with
   the Prescaling replacement rules.
";
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^numE 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};
1498  GenerateSOOandECSOLSTable[nmax_Integer, OptionsPattern[]]:= (
1499    If[And[OptionValue["Progress"], frontEndAvailable],
1500      (
1501        numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
      numE]]^2, {numE, 1, nmax}]];
1502        counters   = Association[Table[numE->0, {numE, 1, nmax}]];
1503        totalIters = Total[Values[numItersai[[1;;nmax]]]];
1504        template1  = StringTemplate["Iteration `numiter` of `
      totaliter`"];
1505        template2  = StringTemplate["`remtime` min remaining"];
      template3 = StringTemplate["Iteration speed = `speed` ms/it"];
1506        template4  = StringTemplate["Time elapsed = `runtime` min"];
1507        progBar = PrintTemporary[
1508          Dynamic[
1509            Pane[
1510              Grid[{
1511                    {Superscript["f", numE]},
1512                    {template1[<|"numiter"->numiter, "totaliter"->
      totalIters|>]},
1513                    {template4[<|"runtime"->Round[QuantityMagnitude[
      UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
1514                    {template2[<|"remtime"->Round[QuantityMagnitude[
      UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
      ]], 0.1]|>]},
1515                    {template3[<|"speed"->Round[QuantityMagnitude[Now
      -startTime, "ms"]/(numiter), 0.01]|>]}, {ProgressIndicator[Dynamic
      [numiter], {1, totalIters}]}
1516                  },
1517                  Frame->All
1518              ],
1519              Full,
1520              Alignment->Center
1521            ]
1522          ]
1523        ];
1524      )
1525    ];
1526    SOOandECSOLSTable = <||>;
1527    numiter   = 1;
1528    startTime = Now;
1529    Do[
1530      (
1531        numiter+= 1;
1532        SOOandECSOLSTable[{numE, SL, SpLp}] = Which[
1533          numE==1,
1534          0,
1535          numE==2,
1536          SimplifyFun[ReducedSOOandECSOinf2[SL, SpLp]],
1537          True,
1538          SimplifyFun[ReducedSOOandECSOinfn[numE,  SL, SpLp]]
1539        ];
1540      ),
1541    {numE, 1, nmax},
1542    {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"];
1630        template4 = StringTemplate["Time elapsed = `runtime` min"];
1631        progBar = PrintTemporary[
1632          Dynamic[
1633            Pane[
1634              Grid[{{Superscript["f", numE]},
1635                    {template1[<|"numiter"->numiter, "totaliter"->
    totalIters|>]},
1636                    {template4[<|"runtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
1637                    {template2[<|"remtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
    ]], 0.1]|>]},
1638                    {template3[<|"speed"->Round[QuantityMagnitude[Now
    -startTime, "ms"]/(numiter), 0.01]|>]},
1639                    {ProgressIndicator[Dynamic[numiter], {1,
    totalIters}]}},
1640                    Frame->All],
1641                    Full,
1642                    Alignment->Center]
1643                ]
1644              ];
1645      )
1646    ];
1647    T22Table = <||>;
1648    startTime = Now;
1649    numiter = 1;
1650    Do[
1651      (
1652        numiter+= 1;
1653        T22Table[{numE, SL, SpLp}] = Which[
1654          numE==1,
1655          0,
1656          numE==2,
1657          SimplifyFun[ReducedT22inf2[SL, SpLp]],
1658          True,
1659          SimplifyFun[ReducedT22infn[numE,  SL, SpLp]]
1660        ];
1661      ),
1662    {numE, 1, nmax},
1663    {SL,   AllowedNKSLTerms[numE]},
1664    {SpLp, AllowedNKSLTerms[numE]}
1665    ];
1666    If[And[OptionValue["Progress"],frontEndAvailable],
1667      NotebookDelete[progBar]
1668    ];
1669    If[OptionValue["Export"],
1670      (
1671        fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"
    }];
1672        Export[fname, T22Table];
1673      )
1674    ];
1675    Return[T22Table];
1676  );
```

```
1677
1678  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 from the association T22Table, putting in the adequate
        phase, and 6-j symbol.
1679  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.\"
1680  \".
1681  ";
1682  SpinSpin[numE_, SL_, SpLp_, J_]:=Module[
1683    {S, L, Sp, Lp, α, val},
1684    (
1685      α = 2;
1686      {S, L} = FindSL[SL];
1687      {Sp, Lp} = FindSL[SpLp];
1688      val = (
1689               Phaser[Sp + L + J] *
1690               SixJay[{Sp, Lp, J}, {L, S, α}] *
1691               T22Table[{numE, SL, SpLp}]
1692             );
1693      Return[val]
1694    )
1695  ];
1696
1697  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.";
1698  Options[GenerateSpinSpinTable] = {"Export"->False};
1699  GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
1700  (
1701    SpinSpinTable = <||>;
1702    PrintTemporary[Dynamic[numE]];
1703    Do[
1704      SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp,
       J]);,
1705    {numE, 1, nmax},
1706    {J, MinJ[numE], MaxJ[numE]},
1707    {SL,   First /@ AllowedNKSLforJTerms[numE, J]},
1708    {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
1709    ];
1710    If[OptionValue["Export"],
1711    (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
1712      Export[fname, SpinSpinTable];
1713      )
1714    ];
1715    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, 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];
  )
];

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^numE configuration. The
    interaction is given by the sum of the spin-spin, the spin-other-
    orbit,  and the electrostatically-correlated-spin-orbit
    interactions.
  The part corresponding to the spin-spin interaction is provided by
    SpinSpin[{numE, SLJ, SLJp, J}].
  The part corresponding to SOO and ECSO is provided by the function
    SOOandECSO[{numE, SLJ, SLJp, J}].
  The function requires chenDeltas to be loaded into the session.
  The option \"ChenDeltas\" can be used 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}];
```

```
1794            Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
1795            {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ
    , SLJp}]["wrong"];
1796            total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1797            total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1798            total  = total /. Prescaling;
1799            total  = wChErrA * total + (1 - wChErrA) * (ss + sooandecso
    )
1800          )
1801        ];
1802        (* In the type B errors the wrong values are zeros all around
    *)
1803        If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
1804          (
1805            {S, L} = FindSL[SLJ];
1806            {Sp, Lp} = FindSL[SLJp];
1807            phase = Phaser[Sp + L + J];
1808            Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
1809            Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
1810            {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
1811            total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1812            total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1813            total  = total /. Prescaling;
1814            total  = wChErrB * total + (1 - wChErrB) * (ss + sooandecso
    )
1815          )
1816        ];
1817        Return[total];
1818      )
1819
1820    (* ################### Magnetic Interactions ################### *)
1821    (* ############################################################ *)
1822
1823    (* ############################################################ *)
1824    (* ###################### Crystal Field ######################## *)
1825
1826    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.";
1827    Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_]:=Module[
1828      {S, Sp, L, Lp, orbital, val},
1829      (
1830        orbital = 3;
1831        {S, L}   = FindSL[NKSL];
1832        {Sp, Lp} = FindSL[NKSLp];
1833        f1  = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
1834        val =
1835          If[f1==0,
1836            0,
1837            (
1838              f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
1839              If[f2==0,
1840                0,
1841                (
1842                  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]
                    )
                  )
                ]
              )
            ]
          )
        ];
      Return[val];
    )
  ];

  Bqk::usage="Real part of the Bqk coefficients.";
  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::usage="Imaginary part of the Bqk coefficients.";
  Sqk[q_, 2] := {0, S12, S22}[[q + 1]];
  Sqk[q_, 4] := {0, S14, S24, S34, S44}[[q + 1]];
  Sqk[q_, 6] := {0, S16, S26, S36, S46, S56, S66}[[q + 1]];

  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.";
1890  TotalCFIters[i_, j_] := (
1891    numIters = {196, 8281, 132496, 1002001, 4008004, 9018009,
      11778624};
1892    Return[Total[numIters[[i ;; j]]]];
1893    )
1894
1895  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.";
1896  Options[GenerateCrystalFieldTable] = {"Export" -> False, "Progress"
       -> True, "Compress" -> True}
1897  GenerateCrystalFieldTable[numEs_List:{1,2,3,4,5,6,7},
      OptionsPattern[]]:= (
1898    ExportFun =
1899    If[OptionValue["Compress"],
1900      ExportMZip,
1901      Export
1902    ];
1903    numiter = 1;
1904    template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
1905    template2 = StringTemplate["`remtime` min remaining"];
1906    template3 = StringTemplate["Iteration speed = `speed` ms/it"];
1907    template4 = StringTemplate["Time elapsed = `runtime` min"];
1908    totalIter = Total[TotalCFIters[#, #] & /@ numEs];
1909    freebies = 0;
1910    startTime = Now;
1911    If[And[OptionValue["Progress"], frontEndAvailable],
1912      progBar = PrintTemporary[
1913        Dynamic[
1914          Pane[
1915            Grid[
1916              {
1917                {Superscript["f", numE]},
1918                {template1[<|"numiter" -> numiter, "totaliter" ->
      totalIter|>]},
1919                {template4[<|"runtime" -> Round[QuantityMagnitude[
      UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
1920                {template2[<|"remtime" -> Round[QuantityMagnitude[
      UnitConvert[(Now - startTime)/(numiter - freebies) * (totalIter -
      numiter), "min"]], 0.1]|>]},
1921                {template3[<|"speed" -> Round[QuantityMagnitude[Now -
       startTime, "ms"]/(numiter-freebies), 0.01]|>]},
1922                {ProgressIndicator[Dynamic[numiter], {1, totalIter}]}
1923              },
1924            Frame -> All
1925            ],
1926          Full,
```

```
1927            Alignment -> Center
1928            ]
1929          ]
1930        ];
1931      ];
1932      Do[
1933        (
1934          exportFname = FileNameJoin[{moduleDir, "data", "
      CrystalFieldTable_f"<>ToString[numE]<>".m"}];
1935          If[FileExistsQ[exportFname],
1936            Print["File exists, skipping ..."];
1937            numiter+=  TotalCFIters[numE, numE];
1938            freebies+= TotalCFIters[numE, numE];
1939            Continue[];
1940          ];
1941          CrystalFieldTable = <||>;
1942          Do[
1943            (
1944              numiter+= 1;
1945              CrystalFieldTable[{numE, NKSL, J, M, NKSLp, Jp, Mp}] =
      CrystalField[numE, NKSL, J, M, NKSLp, Jp, Mp];
1946            ),
1947          {J, MinJ[numE], MaxJ[numE]},
1948          {Jp, MinJ[numE], MaxJ[numE]},
1949          {M, AllowedMforJ[J]},
1950          {Mp, AllowedMforJ[Jp]},
1951          {NKSL , First /@ AllowedNKSLforJTerms[numE, J]},
1952          {NKSLp, First /@ AllowedNKSLforJTerms[numE, Jp]}
1953          ];
1954          If[And[OptionValue["Progress"],frontEndAvailable],
1955            NotebookDelete[progBar]
1956          ];
1957          If[OptionValue["Export"],
1958            (
1959              Print["Exporting to file "<>ToString[exportFname]];
1960              ExportFun[exportFname, CrystalFieldTable];
1961            )
1962          ];
1963        ),
1964      {numE, numEs}
1965      ]
1966    )
1967
1968    (* ###################### Crystal Field ######################### *)
1969    (* ################################################################ *)
1970
1971    (* ################################################################ *)
1972    (* #### Configuration-Interaction via Casimir Operators ###### *)
1973
1974    CasimirSO3::usage = "CasimirSO3[SL, SpLp] returns LS reduced matrix
        element of the configuration interaction term corresponding to
      the Casimir operator of R3.";
1975    CasimirSO3[{SL_, SpLp_}] := (
1976      {S, L} = FindSL[SL];
1977      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.";
2024  CasimirSO7[{SL_, SpLp_}] := (
2025    Wlabel = FindNKLSTerm[SL][[1]][[3]];
2026    If[SL==SpLp,
2027      γ * GSO7W[Wlabel],
2028      0
2029    ]
2030  )
2031
2032  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.";
2033  ElectrostaticConfigInteraction[{SL_, SpLp_}]:=Module[
2034    {S, L, val},
2035    (
2036      {S, L} = FindSL[SL];
2037      val = (
2038        If[SL == SpLp,
2039          CasimirSO3[{SL, SL}] +
2040          CasimirSO7[{SL, SL}] +
2041          CasimirG2[{SL, SL}],
2042          0
2043        ]
2044        );
2045      ElectrostaticConfigInteraction[{S, L}] = val;
2046      Return[val];
2047    )
2048  ];
2049
2050  (* #### Configuration-Interaction via Casimir Operators ###### *)
2051  (* ############################################################ *)
2052
2053  (* ############################################################ *)
2054  (* ##################### Block assembly ####################### *)
2055
2056  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.";
2057  Options[JJBlockMatrix] = {"Sparse"->True, "ChenDeltas"->False};
2058  JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]]:=Module[
2059    {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
2060    SLterm, SpLpterm,
2061    MJ, MJp,
2062    subKron, matValue, eMatrix},
2063    (
2064      NKSLJMs  = AllowedNKSLJMforJTerms[numE, J];
2065      NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
2066      eMatrix =
2067        Table[
```

```
2068            (*Condition for a scalar matrix op*)
2069            SLterm    =  NKSLJM[[1]];
2070            SpLpterm  = NKSLJMp[[1]];
2071            MJ        =  NKSLJM[[3]];
2072            MJp       = NKSLJMp[[3]];
2073            subKron   =
2074              (
2075                KroneckerDelta[J, Jp] *
2076                KroneckerDelta[MJ, MJp]
2077              );
2078            matValue =
2079              If[subKron==0,
2080                0,
2081                  (
2082                    ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2083                    ElectrostaticConfigInteraction[{SLterm, SpLpterm}]
    +
2084                    SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2085                    MagneticInteractions[{numE, SLterm, SpLpterm, J}, "
    ChenDeltas" -> OptionValue["ChenDeltas"]] +
2086                    ThreeBodyTable[{numE, SLterm, SpLpterm}]
2087                  )
2088              ];
2089            matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp
    }];
2090            matValue,
2091          {NKSLJMp, NKSLJMps},
2092          {NKSLJM , NKSLJMs}
2093          ];
2094       If[OptionValue["Sparse"],
2095         eMatrix = SparseArray[eMatrix]
2096       ];
2097       Return[eMatrix]
2098     )
2099     ];
2100
2101     EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some
        point may be used to redefine states used in basis.";
2102     EnergyStates[numE_, J_]:= AllowedNKSLJMforJTerms[numE, J];
2103
2104     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.";
2105     Options[JJBlockMatrixFileName] = {"FilenameAppendix" -> ""}
2106     JJBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
2107       fileApp = OptionValue["FilenameAppendix"];
2108       fname = FileNameJoin[{moduleDir,
2109           "hams",
2110           StringJoin[{"f", ToString[numE], "_JJBlockMatrixTable",
    fileApp ,".m"}]}];
2111       Return[fname];
2112     );
2113
2114     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.";
2115  Options[TabulateJJBlockMatrixTable] = {"Sparse"->True, "ChenDeltas"
    ->False};
2116  TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]]:= (
2117    JJBlockMatrixTable = <||>;
2118    totalIterations = Length[AllowedJ[numE]]^2;
2119    template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
2120    template2 = StringTemplate["`remtime` min remaining"];
2121    template4 = StringTemplate["Time elapsed = `runtime` min"];
2122    numiter   = 0;
2123    startTime = Now;
2124    If[$FrontEnd =!= Null,
2125      (
2126        temp = PrintTemporary[
2127          Dynamic[
2128            Grid[
2129              {
2130                {template1[<|"numiter"->numiter, "totaliter"->
    totalIterations|>]},
2131                {template2[<|"remtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime)/(Max[1,numiter])*(totalIterations-
    numiter), "min"]], 0.1]|>]},
2132                {template4[<|"runtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
2133                {ProgressIndicator[numiter, {1, totalIterations}]}
2134              }
2135            ]
2136        ]
2137        ];
2138      )
2139    ];
2140    Do[
2141      (
2142        JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp
    , CFTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas" ->
    OptionValue["ChenDeltas"]];
2143        numiter += 1;
2144      ),
2145    {Jp, AllowedJ[numE]},
2146    {J, AllowedJ[numE]}
2147    ];
2148    If[$FrontEnd =!= Null,
2149      NotebookDelete[temp]
2150    ];
2151    Return[JJBlockMatrixTable];
2152  );
2153
```

```
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.
```

```
2189  The option \"Set t2Switch\" can be used to toggle on or off setting
        the t2 selector automatically or not, the default is True, which
      replaces the parameter according to numE.
2190  The option \"ReturnInBlocks\" can be use to return the matrix in
      block or flattened form. The default is to return it in flattened
      form.";
2191  Options[HamMatrixAssembly] = {
2192        "FilenameAppendix"->"",
2193        "IncludeZeeman"->False,
2194        "Set t2Switch"->True,
2195        "ReturnInBlocks"->False};
2196  HamMatrixAssembly[nf_, OptionsPattern[]]:=Module[
2197    {numE, ii, jj, howManyJs, Js, blockHam},
2198    (
2199      (*######################################*)
2200      ImportFun = ImportMZip;
2201      (*######################################*)
2202      (*hole-particle equivalence enforcement*)
2203      numE = nf;
2204      allVars = {E0, E1, E2, E3, ζ, F0, F2, F4, F6, M0, M2, M4, T2,
      T2p,
2205           T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
2206           α, β, γ, B02, B04, B06, B12, B14, B16,
2207           B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16,
      S22,
2208           S24, S26, S34, S36, S44, S46, S56, S66, T11, T11p, T12, T14,
      T15, T16,
2209           T17, T18, T19, Bx, By, Bz};
2210      params0 = AssociationThread[allVars, allVars];
2211      If[nf > 7,
2212        (
2213          numE = 14 - nf;
2214          params = HoleElectronConjugation[params0];
2215          If[OptionValue["Set t2Switch"], params[t2Switch] = 0];
2216        ),
2217        params = params0;
2218        If[OptionValue["Set t2Switch"], params[t2Switch] = 1];
2219      ];
2220      (* Load symbolic expressions for LS,J,J' energy sub-matrices.
      *)
2221      emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
      OptionValue["FilenameAppendix"]];
2222      JJBlockMatrixTable = ImportFun[emFname];
2223      (*Patch together the entire matrix representation using J,J'
      blocks.*)
2224      PrintTemporary["Patching JJ blocks ..."];
2225      Js         = AllowedJ[numE];
2226      howManyJs = Length[Js];
2227      blockHam  = ConstantArray[0, {howManyJs, howManyJs}];
2228      Do[
2229        blockHam[[jj, ii]] = JJBlockMatrixTable[{numE, Js[[ii]], Js[[
      jj]]}];,
2230      {ii, 1, howManyJs},
2231      {jj, 1, howManyJs}
2232      ];
```

```
2233
2234        (* Once the block form is created flatten it *)
2235        If[Not[OptionValue["ReturnInBlocks"]],
2236          (blockHam = ArrayFlatten[blockHam];
2237          blockHam  = ReplaceInSparseArray[blockHam, params];
2238          ),
2239          (blockHam  = Map[ReplaceInSparseArray[#, params]&,blockHam
      ,{2}];)
2240        ];
2241
2242        If[OptionValue["IncludeZeeman"],
2243          (
2244            PrintTemporary["Including Zeeman terms ..."];
2245            {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "
      ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
2246            blockHam += - TeslaToKayser * (Bx * magx + By * magy + Bz *
       magz);
2247          )
2248        ];
2249        Return[blockHam];
2250      )
2251    ];
2252
2253    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.";
2254    Options[SimplerSymbolicHamMatrix]={
2255      "Export"->True,
2256      "PrependToFilename"->"",
2257      "EorF"->"F",
2258      "Overwrite" -> False,
2259      "Return" -> True,
2260      "Set t2Switch" -> False,
2261      "IncludeZeeman" -> False};
2262    SimplerSymbolicHamMatrix[numE_Integer, simplifier_List,
      OptionsPattern[]]:=Module[
2263      {thisHam,eTofs,fname},
2264      (
2265        If[Not[ValueQ[ElectrostaticTable]],
2266          LoadElectrostatic[]
2267          ];
2268        If[Not[ValueQ[SOOandECSOTable]],
2269          LoadSOOandECSO[]
2270        ];
2271        If[Not[ValueQ[SpinOrbitTable]],
2272          LoadSpinOrbit[]
```

```
2273        ];
2274        If[Not[ValueQ[SpinSpinTable]],
2275          LoadSpinSpin[]
2276        ];
2277        If[Not[ValueQ[ThreeBodyTable]],
2278          LoadThreeBody[]
2279        ];
2280
2281        fname = FileNameJoin[{moduleDir,"hams",OptionValue["
    PrependToFilename"]<>"SymbolicMatrix-f"<>ToString[numE]<>".m"}];
2282        fnamemx = FileNameJoin[{moduleDir,"hams",OptionValue["
    PrependToFilename"]<>"SymbolicMatrix-f"<>ToString[numE]<>".mx"}];
2283        If[Or[FileExistsQ[fname], FileExistsQ[fnamemx]] && Not[
    OptionValue["Overwrite"]],
2284          (
2285            If[OptionValue["Return"],
2286              (
2287                Which[
2288                  FileExistsQ[fnamemx],
2289                  (
2290                    Print["File ",fnamemx," already exists, and option
    \"Overwrite\" is set to False, loading file ..."];
2291                    thisHam = Import[fnamemx];
2292                    Return[thisHam];
2293                  ),
2294                  FileExistsQ[fname],
2295                  (
2296                    Print["File ",fname," already exists, and option \"
    Overwrite\" is set to False, loading file ..."];
2297                    thisHam = Import[fname];
2298                    Print["Exporting to file ",fnamemx, " for quicker
    loading."];
2299                    Export[fnamemx,thisHam];
2300                    Return[thisHam];
2301                  )
2302                ]
2303              ),
2304              (
2305                Print["File ",fname," already exists, skipping ..."];
2306                Return[Null];
2307              )
2308            ]
2309          )
2310        ];
2311
2312        thisHam = HamMatrixAssembly[numE, "Set t2Switch" -> OptionValue
    ["Set t2Switch"], "IncludeZeeman"->OptionValue["IncludeZeeman"]];
2313        thisHam = ReplaceInSparseArray[thisHam, simplifier];
2314        (* This removes zero entries from being included in the sparse
    array *)
2315        thisHam = SparseArray[thisHam];
2316        If[OptionValue["Export"],
2317        (
2318          Print["Exporting to file ",fname, " and to ", fnamemx];
2319          Export[fname,thisHam];
```

```
        Export[fnamemx,thisHam];
      )
      ];
      If[OptionValue["Return"],
        Return[thisHam],
        Return[Null]
      ];
    )
  ];


  (* ############################################################ *)
  (* ################ To Intermediate Coupling ################## *)

  FreeHam::usage = "FreeHam[JJBlocks, numE] given the JJ blocks of
    the Hamiltonian for f^n, this function returns a list with all the
     scalar-simplified versions of the blocks.";
  FreeHam[JJBlocks_List, numE_Integer]:=Module[
    {Js, basisJ, pivot, freeHam, idx, J, thisJbasis,
    shrunkBasisPositions, theBlock},
    (
      Js       = AllowedJ[numE];
      basisJ   = BasisLSJMJ[numE, "AsAssociation" -> True];
      pivot    = If[OddQ[numE], 1/2, 0];
      freeHam  = Table[(
        J = Js[[idx]];
        theBlock = JJBlocks[[idx]];
        thisJbasis = basisJ[J];
        (* find the basis vectors that end with pivot *)
        shrunkBasisPositions = Flatten[Position[thisJbasis, {_ ..., pivot}]];
        (* take only those rows and columns *)
        theBlock[[shrunkBasisPositions, shrunkBasisPositions]]
      ),
      {idx, 1, Length[Js]}
      ];
      Return[freeHam];
    )
  ];

  ListRepeater::usage="ListRepeater[list, reps] repeats each element
    of list reps times.";
  ListRepeater[list_List, repeats_Integer] := (
    Flatten[ConstantArray[#, repeats] & /@ list]
  );

  ListLever::usage="ListLever[vecs, multiplicity] takes a list of
    vectors and returns all interleaved shifted versions of them.";
  ListLever[vecs_, multiplicity_]:=Module[
  {uppityVecs, uppityVec},
  (
    uppityVecs = Table[(
      uppityVec = PadRight[{#}, multiplicity] & /@ vec;
      uppityVec = Permutations /@ uppityVec;
      uppityVec = Transpose[uppityVec];
```

```
2369        uppityVec = Flatten /@ uppityVec
2370        ),
2371      {vec, vecs}
2372      ];
2373      Return[Flatten[uppityVecs, 1]];
2374    )
2375    ];
2376
2377    EigenLever::usage="EigenLever[eigenSys, multiplicity] takes a list
          eigenSys of the form {eigenvalues, eigenvectors} and returns the
          eigenvalues repeated multiplicity times and the eigenvectors
          interleaved and shifted accordingly.";
2378    EigenLever[eigenSys_, multiplicity_]:=Module[
2379      {eigenVals, eigenVecs, leveledEigenVecs, leveledEigenVals},
2380      (
2381        {eigenVals, eigenVecs} = eigenSys;
2382        leveledEigenVals       = ListRepeater[eigenVals, multiplicity];
2383        leveledEigenVecs       = ListLever[eigenVecs, multiplicity];
2384        Return[{Flatten[leveledEigenVals], leveledEigenVecs}]
2385      )
2386    ];
2387
2388    (* ################# To Intermediate Coupling ################# *)
2389    (* ############################################################ *)
2390
2391    (* ##################### Block assembly ####################### *)
2392    (* ############################################################ *)
2393
2394    (* ############################################################ *)
2395    (* ##################### Optical Operators #################### *)
2396
2397    magOp = <||>;
2398
2399    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.
2400    Here it is provided in atomic units in which the Bohr magneton is
          1/2.
2401    \[Mu] = -(1/2) (L + gs S)
2402    We are using the Racah convention for the reduced matrix elements
          in the Wigner-Eckart theorem. See TASS eqn 11.15.
2403    ";
2404    Options[JJBlockMagDip]={"Sparse"->True};
2405    JJBlockMagDip[numE_, braJ_, ketJ_, OptionsPattern[]]:=Module[
2406      {braSLJs, ketSLJs,
2407      braSLJ,   ketSLJ,
2408      braSL,    ketSL,
2409      braS,     braL,
2410      braMJ,    ketMJ,
2411      matValue, magMatrix},
2412      (
2413        braSLJs  = AllowedNKSLJMforJTerms[numE,braJ];
2414        ketSLJs  = AllowedNKSLJMforJTerms[numE,ketJ];
2415
```

```
      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]
  );
```

```
2467
2468  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'|>.";
2469  Options[TabulateManyJJBlockMagDipTables]={"FilenameAppendix"->"","
        Overwrite"->False,"Compressed"->True};
2470  TabulateManyJJBlockMagDipTables[ns_,OptionsPattern[]]:=(
2471    fnames=<||>;
2472    Do[
2473    (
2474      ExportFun=If[OptionValue["Compressed"],ExportMZip,Export];
2475      PrintTemporary["#------- numE = ",numE," -------#"];
2476      appendTo     = (OptionValue["FilenameAppendix"]<>"-magDip");
2477      exportFname  = JJBlockMatrixFileName[numE,"FilenameAppendix"->
        appendTo];
2478      fnames[numE] = exportFname;
2479      If[FileExistsQ[exportFname]&&Not[OptionValue["Overwrite"]],
2480        Continue[]
2481      ];
2482      JJBlockMatrixTable = TabulateJJBlockMagDipTable[numE];
2483      If[FileExistsQ[exportFname]&&OptionValue["Overwrite"],
2484        DeleteFile[exportFname]
2485      ];
2486      ExportFun[exportFname,  JJBlockMatrixTable];
2487    ),
2488    {numE,ns}
2489    ];
2490    Return[fnames];
2491  );
2492
2493  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. The
        option \"ReturnInBlocks\" can be use to return the matrices in
        blocks. The default is to return the matrices in flattened form.";
2494  Options[MagDipoleMatrixAssembly]={
2495    "FilenameAppendix"->"",
2496    "ReturnInBlocks"->False};
2497  MagDipoleMatrixAssembly[nf_Integer, OptionsPattern[]]:=Module[
2498    {ImportFun, numE, appendTo, emFname, JJBlockMagDipTable,
2499    Js, howManyJs, blockOp, rowIdx, colIdx},
2500    (
2501      ImportFun = ImportMZip;
2502      numE      = nf;
```

```
2503        numH       = 14 - numE;
2504        numE       = Min[numE,numH];
2505
2506        appendTo   = (OptionValue["FilenameAppendix"]<>"-magDip");
2507        emFname    = JJBlockMatrixFileName[numE,"FilenameAppendix"->
      appendTo];
2508        JJBlockMagDipTable = ImportFun[emFname];
2509
2510        Js         = AllowedJ[numE];
2511        howManyJs  = Length[Js];
2512        blockOp    = ConstantArray[0,{howManyJs,howManyJs}];
2513        Do[
2514          blockOp[[rowIdx,colIdx]] = JJBlockMagDipTable[{numE,Js[[
      rowIdx]],Js[[colIdx]]}],
2515            {rowIdx,1,howManyJs},
2516            {colIdx,1,howManyJs}
2517        ];
2518        If[OptionValue["ReturnInBlocks"],
2519          (
2520            opMinus = Map[#[[1]]&, blockOp, {4}];
2521            opZero  = Map[#[[2]]&, blockOp, {4}];
2522            opPlus  = Map[#[[3]]&, blockOp, {4}];
2523            opX =   (opMinus - opPlus)/Sqrt[2];
2524            opY = I (opPlus + opMinus)/Sqrt[2];
2525            opZ = opZero;
2526          ),
2527            blockOp = ArrayFlatten[blockOp];
2528            opMinus = blockOp[[;; , ;; , 1]];
2529            opZero  = blockOp[[;; , ;; , 2]];
2530            opPlus  = blockOp[[;; , ;; , 3]];
2531            opX =   (opMinus - opPlus)/Sqrt[2];
2532            opY = I (opPlus + opMinus)/Sqrt[2];
2533            opZ = opZero;
2534        ];
2535        Return[{opX, opY, opZ}];
2536      )
2537    ];
2538
2539    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.
2540    The option \"Units\" can be set to either \"SI\" (so that the units
         of the returned array are A/m^2) or to \"Hartree\".
2541    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.
2542    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>.";
2543    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.
eigenSys is a list of lists with two elements, in each list the first element is the energy and the second one the corresponding eigenvector.
Based on table 7.3 of Thorne 1999, using g2=1.
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>.
2587  By default this assumes that the refractive index is unity, this
    may be changed by setting the option \"RefractiveIndex\" to the
    desired value.
2588  The option \"Lifetime\" can be used to return the reciprocal of the
     transition rates. The default is to return the transition rates."
    ;
2589  Options[MagDipoleRates]={"Units"->"SI", "Lifetime"->False, "
    RefractiveIndex"->1};
2590  MagDipoleRates[eigenSys_List, numE0_Integer,OptionsPattern[]]:=
    Module[
2591    {AMD,Stot,eigenEnergies,transitionWaveLengthsInMeters,nRefractive
    },
2592    (
2593      nRefractive    = OptionValue["RefractiveIndex"];
2594      numE           = Min[14-numE0, numE0];
2595      Stot           = MagDipLineStrength[eigenSys, numE, "Units"->
    OptionValue["Units"]];
2596      eigenEnergies = Chop[First/@eigenSys];
2597      energyDiffs    = Outer[Subtract,eigenEnergies,eigenEnergies];
2598      energyDiffs    = ReplaceDiagonal[energyDiffs, Indeterminate];
2599      (* Energies assumed in pseudo-energy unit kayser.*)
2600      transitionWaveLengthsInMeters = 0.01/energyDiffs;
2601
2602      unitFactor     = Which[
2603      OptionValue["Units"]=="Hartree",
2604      (
2605        (* The bohrRadius factor in SI neede to convert the
    wavelengths which are assumed in m*)
2606        16 \[Pi]^3 (\[Mu]0Hartree /(3 hPlanckFine)) * bohrRadius^3
2607      ),
2608      OptionValue["Units"]=="SI",
2609      (
2610        16 \[Pi]^3 \[Mu]0/(3 hPlanck)
2611      ),
2612      True,
2613      (
2614        Print["Invalid option for \"Units\". Options are \"SI\" and \
    "Hartree\"."];
2615        Abort[];
2616      )
2617      ];
2618      AMD = unitFactor / transitionWaveLengthsInMeters^3 * Stot *
    nRefractive^3;
2619      Which[OptionValue["Lifetime"],
2620        Return[1/AMD],
2621        True,
2622        Return[AMD]
2623      ]
2624    )
2625  ];
2626
2627  GroundStateOscillatorStrength::usage="GroundStateOscillatorStrength
```

```
    [eigenSys, numE] calculates the oscillator strengths between the
    ground state and the excited states as given by eigenSys.
2628 Based on equation 8 of Carnall 1965, removing the 2J+1 factor since
     this degeneracy has been removed by the crystal field.
2629 eigenSys is a list of lists with two elements, in each list the
    first element is the energy and the second one the corresponding
    eigenvector.
2630 The energy unit assumed in eigenSys is Kayser.
2631 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>.
2632 By default this assumes that the refractive index is unity, this
    may be changed by setting the option \"RefractiveIndex\" to the
    desired value.";
2633 Options[GroundStateOscillatorStrength]={"RefractiveIndex"->1};
2634 GroundStateOscillatorStrength[eigenSys_List, numE_Integer,
    OptionsPattern[]]:=Module[
2635   {eigenEnergies, SMDGS, GSEnergy, energyDiffs,
    transitionWaveLengthsInMeters, unitFactor, nRefractive},
2636   (
2637     eigenEnergies    = First/@eigenSys;
2638     nRefractive      = OptionValue["RefractiveIndex"];
2639     SMDGS            = MagDipLineStrength[eigenSys,numE, "Units"->"
    SI", "States"->1];
2640     GSEnergy         = eigenSys[[1,1]];
2641     energyDiffs      = eigenEnergies-GSEnergy;
2642     energyDiffs[[1]] = Indeterminate;
2643     transitionWaveLengthsInMeters = 0.01/energyDiffs;
2644     unitFactor       = (8\[Pi]^2 me)/(3 hPlanck eCharge^2 cLight);
2645     fMDGS            = unitFactor / transitionWaveLengthsInMeters *
     SMDGS * nRefractive;
2646     Return[fMDGS];
2647   )
2648 ];
2649
2650 (* ##################### Optical Operators ##################### *)
2651 (* ############################################################ *)
2652
2653 (* ############################################################ *)
2654 (* ################### Printers and Labels ################### *)
2655
2656 PrintL::usage = "PrintL[L] give the string representation of a
    given angular momentum.";
2657 PrintL[L_] := If[StringQ[L], L, StringTake[specAlphabet, {L + 1}]]
2658
2659 FindSL::usage = "FindSL[LS] gives the spin and orbital angular
    momentum that corresponds to the provided string LS.";
2660 FindSL[SL_]:= (
2661   FindSL[SL] =
2662   If[StringQ[SL],
2663     {
2664       (ToExpression[StringTake[SL, 1]]-1)/2,
2665       StringPosition[specAlphabet, StringTake[SL, {2}]][[1, 1]]-1
2666     },
2667     SL
```

```
    ]
];

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.";
PrintSLJ[SLJ_] :=
  RowBox[{SuperscriptBox[" ", 2 SLJ[[1]] + 1],
    SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]}] // DisplayForm;

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."
  ;
PrintSLJM[SLJM_] :=
  RowBox[{SuperscriptBox[" ", 2 SLJM[[1]] + 1],
    SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]}] //
  DisplayForm;

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

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

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.";
AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE,
   14-numE]]];

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"};

MaxJ::usage = "MaxJ[numE] gives the maximum J = S+L that
  corresponds to the configuration f^numE.";
MaxJ[numE_] := Max[Map[Total, AllowedSLTerms[Min[numE, 14-numE]]]];

MinJ::usage = "MinJ[numE] gives the minimum J = S+L that
  corresponds to the configuration f^numE.";
MinJ[numE_] := Min[Map[Abs[Part[#, 1] - Part[#, 2]] &,
  AllowedSLTerms[Min[numE, 14-numE]]]]
```

```
2701
2702  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.";
2703  AllowedSLJTerms[numE_] := Module[
2704    {idx1, allowedSL, allowedSLJ},
2705    (
2706      allowedSL = AllowedSLTerms[numE];
2707      allowedSLJ = {};
2708      For[
2709        idx1 = 1,
2710        idx1 <= Length[allowedSL],
2711        termSL = allowedSL[[idx1]];
2712        termsSLJ =
2713          Table[
2714            {termSL[[1]], termSL[[2]], J},
2715          {J,Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2716          ];
2717        allowedSLJ = Join[allowedSLJ, termsSLJ];
2718        idx1++
2719      ];
2720      SortBy[allowedSLJ, Last]
2721    )
2722  ];
2723
2724  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.";
2725  AllowedNKSLJTerms[numE_] := Module[
2726    {allowedSL, allowedNKSL, allowedSLJ, nn},
2727    (
2728      allowedNKSL = AllowedNKSLTerms[numE];
2729      allowedSL = AllowedSLTerms[numE];
2730      allowedSLJ = {};
2731      For[
2732        nn = 1,
2733        nn <= Length[allowedSL],
2734        (
2735          termSL = allowedSL[[nn]];
2736          termNKSL = allowedNKSL[[nn]];
2737          termsSLJ =
2738            Table[{termNKSL, J},
2739            {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2740            ];
2741          allowedSLJ = Join[allowedSLJ, termsSLJ];
2742          nn++
2743        )
2744      ];
2745      SortBy[allowedSLJ, Last]
2746    )
2747  ];
2748
```

```
2749  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.";
2750  AllowedNKSLforJTerms[numE_, J_]:=Module[
2751    {allowedSL, allowedNKSL, allowedSLJ, nn, termSL, termNKSL,
       termsSLJ},
2752    (
2753      allowedNKSL = AllowedNKSLTerms[numE];
2754      allowedSL = AllowedSLTerms[numE];
2755      allowedSLJ = {};
2756      For[
2757        nn = 1,
2758        nn <= Length[allowedSL],
2759        (
2760          termSL = allowedSL[[nn]];
2761          termNKSL = allowedNKSL[[nn]];
2762          termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[
       termSL],
2763            {{termNKSL, J}},
2764            {}
2765            ];
2766          allowedSLJ = Join[allowedSLJ, termsSLJ];
2767          nn++
2768        )
2769      ];
2770      Return[allowedSLJ]
2771    )
2772  ];
2773
2774  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
       }.";
2775  AllowedSLJMTerms[numE_]:=Module[
2776    {allowedSLJ, allowedSLJM, termSLJ, termsSLJM, nn},
2777    (
2778    allowedSLJ = AllowedSLJTerms[numE];
2779    allowedSLJM = {};
2780    For[
2781      nn = 1,
2782      nn <= Length[allowedSLJ],
2783      nn++,
2784      (
2785        termSLJ = allowedSLJ[[nn]];
2786        termsSLJM =
2787          Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
2788          {M, - termSLJ[[3]], termSLJ[[3]]}
2789          ];
2790        allowedSLJM = Join[allowedSLJM, termsSLJM];
2791      )
2792    ];
2793    Return[SortBy[allowedSLJM, Last]];
2794    )
```

```
];

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}.";
AllowedNKSLJMforJMTerms[numE_, J_, MJ_] := Module[
  {allowedSL, allowedNKSL, allowedSLJM, nn},
  (
    allowedNKSL = AllowedNKSLTerms[numE];
    allowedSL   = AllowedSLTerms[numE];
    allowedSLJM = {};
    For[
      nn = 1,
      nn <= Length[allowedSL],
      termSL = allowedSL[[nn]];
      termNKSL = allowedNKSL[[nn]];
      termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
                    <= J
                    <= Total[termSL]
                    && (Abs[MJ] <= J)
                    ),
                    {{termNKSL, J, MJ}},
                    {}];
      allowedSLJM = Join[allowedSLJM, termsSLJ];
      nn++
    ];
    Return[allowedSLJM];
  )
];

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}.";
AllowedNKSLJMforJTerms[numE_, J_] := Module[
  {MJs, labelsAndMomenta, termsWithJ},
  (
    MJs = AllowedMforJ[J];
    (* Pair LS labels and their {S,L} momenta *)
    labelsAndMomenta = ({#, FindSL[#]}) & /@ AllowedNKSLTerms[numE
  ];
    (* A given term will contain J if |L-S|<=J<=L+S *)
    ContainsJ[{SL_String, {S_, L_}}] := (Abs[S - L] <= J <= (S + L)
  );
    (* Keep just the terms that satisfy this condition *)
    termsWithJ = Select[labelsAndMomenta, ContainsJ];
    (* We don't want to keep the {S,L} *)
    termsWithJ = {#[[1]], J} & /@ termsWithJ;
    (* This is just a quick way of including up all the MJ values
  *)
    Return[Flatten /@ Tuples[{termsWithJ, MJs}]]
```

```
2838      )
2839    ];
2840
2841    AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J
          , 1].";
2842    AllowedMforJ[J_] := Range[-J, J, 1];
2843
2844    AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta
           J that appear in the f^numE configuration.";
2845    AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}];
2846
2847    Seniority::usage="Seniority[LS] returns the seniority of the given
          term.";
2848    Seniority[LS_] := FindNKLSTerm[LS][[1, 2]];
2849
2850    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}.";
2851    FindNKLSTerm[SL_]:=Module[
2852      {NKterms, n},
2853      (
2854        n = 7;
2855        NKterms = {{}};
2856        Map[
2857          If[! StringFreeQ[First[#], SL],
2858            If[ToExpression[Part[#, 2]] <= n,
2859              NKterms = Join[NKterms, {#}, 1]
2860            ]
2861          ] &,
2862        fnTermLabels
2863        ];
2864        NKterms = DeleteCases[NKterms, {}];
2865        NKterms
2866      )
2867    ];
2868
2869    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}.";
2870    Options[ParseTermLabels] = {"Export" -> True};
2871    ParseTermLabels[OptionsPattern[]]:=Module[
2872      {labelsTextData, fNtextLabels, nielsonKosterLabels, seniorities,
          RacahW, RacahU},
2873      (
2874        labelsTextData = FileNameJoin[{moduleDir, "data", "
          NielsonKosterLabels_f6_f7.txt"}];
2875        fNtextLabels   = Import[labelsTextData];
2876        nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
2877        termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
2878        seniorities = Map[ToExpression[Part[# , {2}]] &,
          nielsonKosterLabels];
2879        racahW =
```

```
2880          Map[
2881            StringTake[
2882              Flatten[StringCases[Part[# , {3}],
2883                "(" ~~ DigitCharacter ~~ DigitCharacter ~~
    DigitCharacter ~~ ")"]],
2884              {2, 4}
2885            ] &,
2886          nielsonKosterLabels];
2887        racahU =
2888          Map[
2889            StringTake[
2890              Flatten[StringCases[Part[# , {3}],
2891                "(" ~~ DigitCharacter ~~ DigitCharacter ~~ ")"]],
2892              {2, 3}
2893            ] &,
2894          nielsonKosterLabels];
2895        fnTermLabels = Join[termLabels, seniorities, racahW, racahU,
    2];
2896        fnTermLabels = Sort[fnTermLabels];
2897        If[OptionValue["Export"],
2898          (
2899            broadFname = FileNameJoin[{moduleDir,"data","fnTerms.m"}];
2900            Export[broadFname, fnTermLabels];
2901          )
2902        ];
2903        Return[fnTermLabels];
2904      )
2905    ];
2906
2907    (* ###################### Term management ###################### *)
2908    (* ########################################################### *)
2909
2910    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.";
2911    Options[LoadParameters] = {
2912        "Source"->"Carnall",
2913        "Free Ion"->False,
2914        "gs"->2.002319304386,
2915        "With Uncertainties"->False
2916        };
2917    LoadParameters[Ln_String, OptionsPattern[]]:= Module[
2918      {source, params, uncertain, uncertainKeys, uncertainRules},
2919      (
2920        source = OptionValue["Source"];
2921        params = Which[source=="Carnall",
2922                 (Association[Carnall["data"][Ln]])
2923                 ];
2924        (*If a free ion then all the parameters from the crystal field
      are set to zero*)
```

```
2925        If[OptionValue["Free Ion"],
2926          Do[params[cfSymbol] = 0, {cfSymbol, cfSymbols}]
2927        ];
2928        params[F0] = 0;
2929        params[M2] = 0.56*params[M0]; (*See Carnall 1989,Table I,
    caption,probably fixed based on HF values*)
2930        params[M4] = 0.31*params[M0]; (*See Carnall 1989,Table I,
    caption,probably fixed based on HF values*)
2931        params[P0] = 0;
2932        params[P4] = 0.5*params[P2];  (*See Carnall 1989,Table I,
    caption,probably fixed based on HF values*)
2933        params[P6] = 0.1*params[P2];  (*See Carnall 1989,Table I,
    caption,probably fixed based on HF values*)
2934        params[gs] = OptionValue["gs"];
2935        {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[
    F0], params[F2], params[F4], params[F6]];
2936        params[E0] = 0;
2937        If[
2938          Not[OptionValue["With Uncertainties"]],
2939          Return[params],
2940          (
2941            uncertain     = Association[Carnall["annotations"][Ln]];
2942            uncertainKeys = Keys[uncertain];
2943            uncertain     = If[# == "Not allowed to vary in fitting."
    || # == "Interpolated",
2944              0., #] & /@ uncertain;
2945            paramKeys = Keys[params];
2946            uncertainVals = Sort[Intersection[paramKeys, uncertainKeys
    ]] /. Association[uncertain];
2947            uncertainRules = MapThread[Rule, {Sort[uncertainKeys],
    uncertainVals}];
2948            Which[
2949              MemberQ[{"Ce", "Yb"}, Ln],
2950              (
2951                subsetL = {F0};
2952                subsetR = {0};
2953              ),
2954              True,
2955              (
2956              subsetL = {F0, M2, M4, P0, P4, P6, E0, E1, E2, E3};
2957              subsetR = {0, M0*0.65, M0*0.31, 0, P2*0.5, P2*0.1,
2958                0,
2959                Sqrt[(196 F2^2)/164025 + (49 F4^2)/88209 + (122500 F6
    ^2)/134165889],
2960                Sqrt[F2^2/4100625 + F4^2/10673289 + (30625 F6^2)
    /2743558264161],
2961                Sqrt[F2^2/18225 + (4 F4^2)/1185921 + (30625 F6^2)
    /1803785841]};
2962              )
2963            ];
2964            uncertainRules = Join[uncertainRules, MapThread[Rule, {
    subsetL,subsetR /. uncertainRules}]];
2965            uncertainRules = Association[uncertainRules];
2966            Which[
2967              Ln == "Eu",
```

```
            (
              uncertainRules[F4] = 12.121;
              uncertainRules[F6] = 15.872;
            ),
            Ln == "Gd",
            (
              uncertainRules[F4] = 12.07;
            ),
            Ln == "Tb",
            (
              uncertainRules[F4] = 41.006;
            )
          ];
          If[MemberQ[{"Eu","Gd","Tb"},Ln],
            (
              uncertainRules[E1] = Sqrt[(196 F2^2)/164025 + (49 F4^2)
  /88209 + (122500 F6^2)/134165889] /. uncertainRules;
              uncertainRules[E2] = Sqrt[F2^2/4100625 + F4^2/10673289
  + (30625 F6^2)/2743558264161] /. uncertainRules;
              uncertainRules[E3] = Sqrt[F2^2/18225 + (4 F4^2)/1185921
   + (30625 F6^2)/1803785841] /. uncertainRules;
            )
          ];
          uncertainKeys = First /@ Normal[uncertainRules];
          fullParams = Association[MapThread[Rule, {uncertainKeys,
  MapThread[Around, {uncertainKeys /. params, uncertainKeys /.
  uncertainRules}]}]];
          Return[Join[params, fullParams]]
        )
        ];
    )
];

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]];
```

```
3011        flippedParams = Select[flippedParams, FreeQ[#, Missing]&];
3012        Return[flippedParams];
3013      )
3014    ];
3015
3016    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.
3017    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.
3018    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\".
3019    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.
3020    Tha ordered basis in which this eigenvector is to be interpreted is
           the one corresponding to BasisLSJMJ[numE].
3021    The function admits the following options:
3022    \"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.
3023    \"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.
3024    \"Zeroes\" (list) : A list with symbols assumed to be zero.
3025    ";
3026    Options[IonSolver] = {"Include Spin-Spin" -> True,
3027      "Overwrite Hamiltonian" -> False,
3028      "Zeroes" -> {}};
3029    IonSolver[numE_Integer, params0_Association, host_String:"Ln",
          OptionsPattern[]]:=Module[
3030      {ln, simplifier, simpleHam, numHam, eigensys,
3031      startTime, endTime, diagonalTime, params=params0, zeroSymbols},
3032      (
3033        ln = StringSplit["Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb"][[
          numE]];
3034
3035        (* This could be done when replacing values, but this produces
          smaller saved arrays. *)
3036        simplifier = (#-> 0) & /@ OptionValue["Zeroes"];
3037        simpleHam = SimplerSymbolicHamMatrix[numE,
3038          simplifier,
3039          "PrependToFilename" -> host,
3040          "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
3041        ];
3042
3043        (* Note that we don't have to flip signs of parameters for fn
          beyond f7 since the matrix produced
3044        by SimplerSymbolicHamMatrix has already accounted for this. *)
```

```
3045
3046        (* Everything that is not given is set to zero *)
3047        params = ParamPad[params, "Print" -> True];
3048        PrintFun[params];
3049
3050        (* Enforce the override to the spin-spin contribution to the
      magnetic interactions *)
3051        params[\[Sigma]SS] = If[OptionValue["Include Spin-Spin"], 1,
      0];
3052
3053        (* Create the numeric hamiltonian *)
3054        numHam = ReplaceInSparseArray[simpleHam, params];
3055        Clear[simpleHam];
3056
3057        (* Eigensolver *)
3058        PrintFun["> Diagonalizing the numerical Hamiltonian ..."];
3059        startTime = Now;
3060        eigensys  = Eigensystem[numHam];
3061        endTime   = Now;
3062        diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"
      ];
3063        PrintFun[">> Diagonalization took ", diagonalTime, " seconds."
      ];
3064        eigensys = Chop[eigensys];
3065        eigensys = Transpose[eigensys];
3066
3067        (* Shift the baseline energy *)
3068        eigensys = ShiftedLevels[eigensys];
3069        (* Sort according to energy *)
3070        eigensys = SortBy[eigensys, First];
3071        Return[eigensys];
3072      )
3073    ];
3074
3075    ShiftedLevels::usage = "ShiftedLevels[eigenSys] takes a list of
      levels of the form
3076    {{energy_1, coeff_vector_1}, {energy_2, coeff_vector_2},...}} and
      returns the same input except that now to every energy the minimum
       of all of them has been subtracted.";
3077    ShiftedLevels[originalLevels_] := Module[
3078      {groundEnergy, shifted},
3079      (
3080        groundEnergy = Sort[originalLevels][[1,1]];
3081        shifted      = Map[{#[[1]] - groundEnergy, #[[2]]} &,
      originalLevels];
3082        Return[shifted];
3083      )
3084    ];
3085
3086    (* ############################################################ *)
3087    (* ################## Eigensystem analysis ################### *)
3088
3089    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[eigenSys, 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. eigenSys is a list
   of lists with two elements, in each list the first element is the
   energy and the second one the corresponding eigenvector. 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[
  eigenSys, basis, numBasisVecs, roundTo] takes a list of
  eigenstates (given in eigenSys) 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. eigenSys is a list of lists with
   two elements, in each list the first element is the energy and
  the second one the corresponding eigenvector.
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]]
```

```
3172            ),
3173            (
3174              amplitudes[[chosenIndices]]
3175            )
3176          ];
3177          majorComponents    = PrettySaundersSLJmJ[#, "Representation"
    -> OptionValue["Representation"]] & /@ majorComponents;
3178          nonZ               = (# != 0.) & /@ majorThings;
3179          majorThings        = Pick[majorThings, nonZ];
3180          majorComponents    = Pick[majorComponents, nonZ];
3181          If[OptionValue["Coefficients"] == "Probabilities",
3182            (
3183              majorThings = majorThings * 100*"%"
3184            )
3185          ];
3186          majorRep           = majorThings . majorComponents;
3187          {energy, majorRep}
3188          ),
3189          {state, eigensys}];
3190        Return[parsedStates]
3191      )
3192    ];
3193
3194    FindThresholdPosition::usage = "FindThresholdPosition[list,
      threshold] returns the position of the first element in list that
      is greater than or equal to threshold. If no such element exists,
      it returns the length of list. The elements of the given list must
       be in ascending order.";
3195    FindThresholdPosition[list_, threshold_] := Module[
3196      {position},
3197      (
3198      position = Position[list, _?(# >= threshold &), 1, 1];
3199      thrPos = If[Length[position] > 0,
3200        position[[1, 1]],
3201        Length[list]];
3202      If[thrPos == 0,
3203        Return[1],
3204        Return[thrPos]]
3205      )
3206    ];
3207
3208    ParseStateByProbabilitySum[{energy_, eigenVec_}, probSum_, roundTo_
      :0.01, maxParts_:20] := Compile[
3209      {{energy, _Real, 0}, {eigenVec, _Complex, 1},
3210       {probSum, _Real, 0},{roundTo, _Real, 0},
3211       {maxParts, _Integer, 0}},
3212        Module[
3213          {numStates, state, amplitudes, probs, ordering,
3214          orderedProbs, truncationIndex, accProb, thresholdIndex,
    chosenIndices, majorComponents,
3215          majorAmplitudes, absMajorAmplitudes, notnullAmplitudes,
    majorRep},
3216          (
3217            numStates    = Length[eigenVec];
3218            (*Round them up*)
```

```
3219            amplitudes          = Round[eigenVec, roundTo];
3220            probs               = Round[Abs[eigenVec^2], roundTo];
3221            ordering            = Reverse[Ordering[probs]];
3222            (*Order the probabilities from high to low*)
3223            orderedProbs        = probs[[ordering]];
3224            (*To speed up Accumulate, assume that only as much as
      maxParts will be needed*)
3225            truncationIndex     = Min[maxParts, Length[orderedProbs]];
3226            orderedProbs        = orderedProbs[[;;truncationIndex]];
3227            (*Accumulate the probabilities*)
3228            accProb             = Accumulate[orderedProbs];
3229            (*Find the index of the first element in accProb that is
      greater than probSum*)
3230            thresholdIndex      = Min[Length[accProb],
      FindThresholdPosition[accProb, probSum]];
3231            (*Grab all the indicees up till that one*)
3232            chosenIndices       = ordering[[;; thresholdIndex]];
3233            (*Select the corresponding elements from the basis*)
3234            majorComponents     = basis[[chosenIndices]];
3235            (*Select the corresponding amplitudes*)
3236            majorAmplitudes     = amplitudes[[chosenIndices]];
3237            (*Take their absolute value*)
3238            absMajorAmplitudes = Abs[majorAmplitudes];
3239            (*Make sure that there are no effectively zero
      contributions*)
3240            notnullAmplitudes   = Flatten[Position[absMajorAmplitudes,
      x_ /; x != 0]];
3241            (* majorComponents     = PrettySaundersSLJmJ
      [{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents; *)
3242            majorComponents     = PrettySaundersSLJmJ /@ majorComponents
      ;
3243            majorAmplitudes     = majorAmplitudes[[notnullAmplitudes]];
3244            (*Make them into Kets*)
3245            majorComponents     = Ket /@ majorComponents[[
      notnullAmplitudes]];
3246            (*Multiply and add to build the final Ket*)
3247            majorRep            = majorAmplitudes . majorComponents;
3248            Return[{energy, majorRep}];
3249        )
3250      ],
3251     CompilationTarget -> "C",
3252     RuntimeAttributes -> {Listable},
3253     Parallelization -> True,
3254     RuntimeOptions -> "Speed"
3255   ];
3256
3257   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.";
3258   ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ :
        0.01, maxParts_: 20]:=Module[
3259     {parsedByProb, numStates, state, energy, eigenVec, amplitudes,
      probs, ordering,
```

```
3260      orderedProbs, truncationIndex, accProb, thresholdIndex,
          chosenIndices, majorComponents,
3261      majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep
          },
3262      (
3263        numStates    = Length[eigensys];
3264        parsedByProb = Table[(
3265          state              = eigensys[[idx]];
3266          {energy, eigenVec} = state;
3267          (*Round them up*)
3268          amplitudes         = Round[eigenVec, roundTo];
3269          probs              = Round[Abs[eigenVec^2], roundTo];
3270          ordering           = Reverse[Ordering[probs]];
3271          (*Order the probabilities from high to low*)
3272          orderedProbs       = probs[[ordering]];
3273          (*To speed up Accumulate, assume that only as much as
          maxParts will be needed*)
3274          truncationIndex    = Min[maxParts, Length[orderedProbs]];
3275          orderedProbs       = orderedProbs[[;;truncationIndex]];
3276          (*Accumulate the probabilities*)
3277          accProb            = Accumulate[orderedProbs];
3278          (*Find the index of the first element in accProb that is
          greater than probSum*)
3279          thresholdIndex     = Min[Length[accProb],
          FindThresholdPosition[accProb, probSum]];
3280          (*Grab all the indicees up till that one*)
3281          chosenIndices      = ordering[[;; thresholdIndex]];
3282          (*Select the corresponding elements from the basis*)
3283          majorComponents    = basis[[chosenIndices]];
3284          (*Select the corresponding amplitudes*)
3285          majorAmplitudes    = amplitudes[[chosenIndices]];
3286          (*Take their absolute value*)
3287          absMajorAmplitudes = Abs[majorAmplitudes];
3288          (*Make sure that there are no effectively zero contributions
          *)
3289          notnullAmplitudes  = Flatten[Position[absMajorAmplitudes, x_
          /; x != 0]];
3290          (* majorComponents    = PrettySaundersSLJmJ
          [{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents; *)
3291          majorComponents    = PrettySaundersSLJmJ /@ majorComponents;
3292          majorAmplitudes    = majorAmplitudes[[notnullAmplitudes]];
3293          majorComponents    = majorComponents[[notnullAmplitudes]];
3294          (*Multiply and add to build the final Ket*)
3295          majorRep           = majorAmplitudes . majorComponents;
3296          {energy, majorRep}
3297          ), {idx, numStates}];
3298        Return[parsedByProb];
3299      )
3300    ];
3301
3302    (* ################## Eigensystem analysis ################## *)
3303    (* ########################################################### *)
3304
3305    (* ########################################################### *)
3306    (* ########################## Misc ########################## *)
```

```
3307
3308  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.";
3309  SymbToNum[expr_, replacementAssociation_]:= (
3310    includedKeys = Keys[replacementAssociation];
3311    (*If a key is not defined, make its value zero.*)
3312    fullAssociation = Table[(
3313      If[MemberQ[includedKeys, key],
3314        ToExpression[key]->replacementAssociation[key],
3315        ToExpression[key]->0
3316      ]
3317    ),
3318    {key, paramSymbols}];
3319    Return[expr/.fullAssociation];
3320  );
3321
3322  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.";
3323  SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;
3324
3325  ExportMZip::usage="ExportMZip[\"dest.[zip,m]\"] saves a compressed
        version of expr to the given destination.";
3326  ExportMZip[filename_, expr_]:=Module[
3327    {baseName, exportName, mImportName, zipImportName},
3328    (
3329      baseName    = FileBaseName[filename];
3330      exportName  = StringReplace[filename,".m"->".zip"];
3331      mImportName = StringReplace[exportName,".zip"->".m"];
3332      If[FileExistsQ[mImportName],
3333      (
3334        PrintTemporary[mImportName<>" exists already, deleting"];
3335        DeleteFile[mImportName];
3336        Pause[2];
3337      )
3338      ];
3339      Export[exportName, (baseName<>".m") -> expr];
3340    )
3341  ];
3342
3343  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";
3344  Options[ImportMZip]={"Leave Uncompressed" -> True};
3345  ImportMZip[filename_String, OptionsPattern[]]:=Module[
3346    {baseName, importKey, zipImportName, mImportName, imported},
```

```
3347      (
3348        baseName      = FileBaseName[filename];
3349        (*Function allows for the filename to be .m or .zip*)
3350        importKey     = baseName <> ".m";
3351        zipImportName = StringReplace[filename, ".m"->".zip"];
3352        mImportName   = StringReplace[zipImportName, ".zip"->".m"];
3353        If[FileExistsQ[mImportName],
3354        (
3355          PrintTemporary[".m version exists already, importing that
      instead ..."];
3356          Return[Import[mImportName]];
3357        )
3358        ];
3359        imported = Import[zipImportName, importKey];
3360        If[OptionValue["Leave Uncompressed"],
3361          Export[mImportName, imported]
3362        ];
3363        Return[imported]
3364      )
3365    ];
3366
3367    ReplaceInSparseArray::usage = "ReplaceInSparseArray[sparseArray,
       rules] takes a sparse array that may contain symbolic quantities
       and returns a sparse array in which the given rules have been used
        on every element.";
3368    ReplaceInSparseArray[sparseA_SparseArray, rules_]:=(
3369      SparseArray[Automatic,
3370        sparseA["Dimensions"],
3371        sparseA["Background"] /. rules,
3372        {
3373          1,
3374          {sparseA["RowPointers"], sparseA["ColumnIndices"]},
3375          sparseA["NonzeroValues"] /. rules
3376        }
3377        ]
3378    );
3379
3380    MapToSparseArray::usage = "MapToSparseArray[sparseArray, function]
       takes a sparse array and returns a sparse array after the function
        has been applied to it.";
3381    MapToSparseArray[sparseA_SparseArray, func_]:=Module[
3382      {nonZ, backg, mapped},
3383      (
3384        nonZ   = func /@ sparseA["NonzeroValues"];
3385        backg  = func[sparseA["Background"]];
3386        mapped = SparseArray[Automatic,
3387          sparseA["Dimensions"],
3388          backg,
3389          {
3390            1,
3391            {sparseA["RowPointers"], sparseA["ColumnIndices"]},
3392            nonZ
3393          }
3394        ];
3395        Return[mapped];
```

```
    )
  ];

  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.";
  Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
  ParseTeXLikeSymbol[bigString_, OptionsPattern[]] := Module[
    {form, mainSymbol, symbols},
    (
      form = OptionValue["Form"];
      (* parse greek *)
      symbols = Table[(
          str = StringReplace[string, {"\\alpha" -> "α",
            "\\beta" -> "β",
            "\\gamma" -> "γ",
            "\\psi" -> "\[Psi]"}];
          symbol = Which[
            StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"
    ]],
            (
              (*yes sub no sup*)
              mainSymbol = StringSplit[str, "_"][[1]];
              mainSymbol = ToExpression[mainSymbol];

              subPart =
                StringCases[str,
                  RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
              Subscript[mainSymbol, subPart]
            ),
            Not[StringContainsQ[str, "_"]] && StringContainsQ[str, "^
    "],
            (
              (*no sub yes sup*)
              mainSymbol = StringSplit[str, "^"][[1]];
              mainSymbol = ToExpression[mainSymbol];

              supPart =
                StringCases[str,
                  RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
              Superscript[mainSymbol, supPart]
            ),
            StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
            (
              (*yes sub yes sup*)
              mainSymbol = StringSplit[str, "_"][[1]];
              mainSymbol = ToExpression[mainSymbol];
              {subPart, supPart} =
                StringCases[str, RegularExpression@"\\{(.*?)\\}" -> "
    $1"];
              Subsuperscript[mainSymbol, subPart, supPart]
```

```
            ),
            True,
            (
              (*no sup or sub*)
              str
            )
            ];
          symbol
          ),
        {string, StringSplit[bigString, " "]}
      ];
      Which[
        form == "Row",
        Return[Row[symbols]],
        form == "List",
        Return[symbols]
      ]
    )
  ];

  (* ########################## Misc ########################## *)
  (* ########################################################## *)

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

  EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes
    states and produces a visualization of its energy spectrum.
  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."
    ;
  Options[EnergyLevelDiagram] = {
    "Title"->"",
    "ImageSize"->1000,
    "AspectRatio" -> 1/8,
    "Background"->"Automatic",
    "Epilog"->{},
    "Explorer"->True
    };
  EnergyLevelDiagram[states_, OptionsPattern[]]:= (
    energies = First/@states;
    epi = OptionValue["Epilog"];
    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}},
```

```
3492        FrameTicks    -> {{None, None}, {Automatic, Automatic}},
3493        FrameStyle    -> Directive[15, Dashed, Thin],
3494        PlotLabel     -> Style[OptionValue["Title"], 15, Bold],
3495        Background    -> OptionValue["Background"],
3496        FrameLabel    -> {"\!\(\*FractionBox[\(E\), SuperscriptBox[\(cm
      \), \(-1\)]]\)"},
3497        Epilog        -> epi]
3498    )
3499
3500    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.
3501    Based on ZeitPolizei @ https://mathematica.stackexchange.com/
      questions/7142/how-to-manipulate-2d-plots.
3502    The option \"OptAxesRedraw\" can be used to specify whether the
      axes should be redrawn. The default is False.";
3503    Options[ExploreGraphics] = {OptAxesRedraw -> False};
3504    ExploreGraphics[graph_Graphics, opts : OptionsPattern[]] := With[
3505      {gr   = First[graph],
3506        opt = DeleteCases[Options[graph],
3507              PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _],
3508        plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
3509        ar  = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
3510        ao  = AbsoluteOptions[AxesOrigin],
3511        rectangle = {Dashing[Small],
3512          Line[{#1,
3513                {First[#2], Last[#1]},
3514                #2,
3515                {First[#1], Last[#2]},
3516                #1}]} &,
3517        optAxesRedraw = OptionValue[OptAxesRedraw]},
3518      DynamicModule[
3519          {dragging=False, first, second, rx1, rx2, ry1, ry2,
3520          range = plr},
3521          {{rx1, rx2}, {ry1, ry2}} = plr;
3522        Panel@
3523        EventHandler[
3524          Dynamic@Graphics[
3525            If[dragging, {gr, rectangle[first, second]}, gr],
3526            PlotRange -> Dynamic@range,
3527            AspectRatio -> ar,
3528            AxesOrigin -> If[optAxesRedraw,
3529              Dynamic@Mean[range\[Transpose]], ao],
3530            Sequence @@ opt],
3531          {{"MouseDown", 1} :> (
3532            first = MousePosition["Graphics"]
3533            ),
3534          {"MouseDragged", 1} :> (
3535            dragging = True;
3536            second = MousePosition["Graphics"]
3537            ),
3538          "MouseClicked" :> (
3539            If[CurrentValue@"MouseClickCount"==2,
3540              range = plr];
3541            ),
```

```
3542          {"MouseUp", 1} :> If[dragging,
3543            dragging = False;
3544
3545            range = {{rx1, rx2}, {ry1, ry2}} =
3546              Transpose@{first, second};
3547            range[[2]] = {0, 1}],
3548          {"MouseDown", 2} :> (
3549            first = {sx1, sy1} = MousePosition["Graphics"]
3550            ),
3551          {"MouseDragged", 2} :> (
3552            second = {sx2, sy2} = MousePosition["Graphics"];
3553            rx1 = rx1 - (sx2 - sx1);
3554            rx2 = rx2 - (sx2 - sx1);
3555            ry1 = ry1 - (sy2 - sy1);
3556            ry2 = ry2 - (sy2 - sy1);
3557            range = {{rx1, rx2}, {ry1, ry2}};
3558            range[[2]] = {0, 1};
3559            )}]]];
3560
3561  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.";
3562  Options[LabeledGrid]={
3563      ItemSize->Automatic,
3564      Alignment->Center,
3565      Frame->All,
3566      "Separator"->",",
3567      "Pivot"->""
3568  };
3569  LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]]:=
     Module[
3570    {gridList=data,rowHeads=rowHeaders,colHeads=columnHeaders},
3571    (
3572      separator=OptionValue["Separator"];
3573      pivot=OptionValue["Pivot"];
3574      gridList=Table[
3575              Tooltip[
3576                data[[rowIdx,colIdx]],
3577                DisplayForm[
3578                  RowBox[{rowHeads[[rowIdx]],
3579                          separator,
3580                          colHeads[[colIdx]]}
3581                        ]
3582                        ]
3583                    ],
3584          {rowIdx,Dimensions[data][[1]]},
3585          {colIdx,Dimensions[data][[2]]}];
3586      gridList=Transpose[Prepend[gridList,colHeads]];
3587      rowHeads=Prepend[rowHeads,pivot];
3588      gridList=Prepend[gridList,rowHeads]//Transpose;
3589      Grid[gridList,
3590          Frame->OptionValue[Frame],
```

```
3591            Alignment->OptionValue[Alignment],
3592            Frame->OptionValue[Frame],
3593            ItemSize->OptionValue[ItemSize]
3594            ]
3595      )
3596  ];
3597
3598  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.";
3599  Options[HamiltonianForm]={"Separator"->"","Pivot"->""}
3600  HamiltonianForm[hamMatrix_, basisLabels_List, OptionsPattern[]]:=(
3601      braLabels=DisplayForm[RowBox[{"\[LeftAngleBracket]",#,"\[
      RightBracketingBar]"}]]& /@ basisLabels;
3602      ketLabels=DisplayForm[RowBox[{"\[LeftBracketingBar]",#,"\[
      RightAngleBracket]"}]]& /@ basisLabels;
3603      LabeledGrid[hamMatrix,braLabels,ketLabels,"Separator"->
      OptionValue["Separator"],"Pivot"->OptionValue["Pivot"]]
3604      )
3605
3606  HamiltonianMatrixPlot::usage="HamiltonianMatrixPlot[hamMatrix,
      basisLabels] creates a matrix plot of the given hamiltonian matrix
       with the given basis labels. The matrix elements can be hovered
      over to display the corresponding row and column labels together
      with the value of the matrix element. The option \"Overlay Values\
      " can be used to specify whether the matrix elements should be
      displayed on top of the matrix plot.";
3607  Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover"
       -> True, "Overlay Values" -> True}];
3608  HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts :
      OptionsPattern[]] := (
3609    braLabels = DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[
      RightBracketingBar]"}]] & /@ basisLabels;
3610    ketLabels = DisplayForm[Rotate[RowBox[{"\[LeftBracketingBar]", #,
       "\[RightAngleBracket]"}],\[Pi]/2]] & /@ basisLabels;
3611    ketLabelsUpright = DisplayForm[RowBox[{"\[LeftBracketingBar]", #,
       "\[RightAngleBracket]"}]] & /@ basisLabels;
3612    numRows = Length[hamMatrix];
3613    numCols = Length[hamMatrix[[1]]];
3614    epiThings = Which[
3615      And[OptionValue["Hover"], Not[OptionValue["Overlay Values"]]],
3616      Flatten[
3617        Table[
3618          Tooltip[
3619            {
3620              Transparent,
3621              Rectangle[
3622              {j - 1, numRows - i},
3623              {j - 1, numRows - i} + {1, 1}
3624              ]
3625            },
```

```
3626            Row[{braLabels[[i]],ketLabelsUpright[[j]],"=",hamMatrix[[i,
     j]]}]
3627            ],
3628          {i, 1, numRows},
3629          {j, 1, numCols}
3630          ]
3631        ],
3632        And[OptionValue["Hover"], OptionValue["Overlay Values"]],
3633        Flatten[
3634          Table[
3635            Tooltip[
3636              {
3637                Transparent,
3638                Rectangle[
3639                {j - 1, numRows - i},
3640                {j - 1, numRows - i} + {1, 1}
3641                ]
3642              },
3643            DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i
    ]], "\[LeftBracketingBar]", basisLabels[[j]], "\[RightAngleBracket
    ]"}]]
3644            ],
3645          {i, numRows},
3646          {j, numCols}
3647          ]
3648        ],
3649        True,
3650        {}
3651        ];
3652      textOverlay = If[OptionValue["Overlay Values"],
3653        (
3654          Flatten[
3655          Table[
3656            Text[hamMatrix[[i, j]],
3657              {j - 1/2, numRows - i + 1/2}
3658            ],
3659          {i, 1, numRows},
3660          {j, 1, numCols}
3661          ]
3662          ]
3663        ),
3664        {}
3665        ];
3666      epiThings = Join[epiThings, textOverlay];
3667      MatrixPlot[hamMatrix,
3668        FrameTicks -> {
3669          {Transpose[{Range[Length[braLabels]], braLabels}], None},
3670          {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}
3671          },
3672        Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
3673        Epilog -> epiThings
3674      ]
3675    );
3676
3677    (* ################## Some Plotting Routines ################## *)
```

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

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

LoadAll::usage="LoadAll[] executes most 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"];
```

```
3767      chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
3768      chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
3769      chenDeltasB = {ToExpression[StringTake[#[[1]], {2}]], #[[2]],
          #[[3]]} & /@ chenDeltasB;
3770      chenDeltas["B"] = chenDeltasB;
3771      On[Power::infy];
3772      If[OptionValue["Export"],
3773        (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas
          .m"}];
3774        Export[chenDeltasFname, chenDeltas];
3775        )
3776        ];
3777      Return[chenDeltas];
3778    );
3779
3780    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).";
3781    Options[ParseCarnall] = {"Export" -> True};
3782    ParseCarnall[OptionsPattern[]] := (
3783      ions          = {"Ce","Pr","Nd","Pm","Sm","Eu","Gd","Tb","Dy","Ho"
          ,"Er","Tm","Yb"};
3784      templates     = StringTemplate/@StringSplit["appendix:`ion`:
          Association appendix:`ion`:Calculated appendix:`ion`:RawTable
          appendix:`ion`:Headings"," "];
3785
3786      (* How many unique eigenvalues, after removing Kramer's
          degeneracy *)
3787      fullSizes     = AssociationThread[ions, {7, 91, 182, 1001, 1001,
          3003, 1716, 3003, 1001, 1001, 182, 91, 7}];
3788      carnall       = Import[FileNameJoin[{moduleDir,"data","Carnall.xls
          "}]][[2]];
3789      carnallErr    = Import[FileNameJoin[{moduleDir,"data","Carnall.xls
          "}]][[3]];
3790
3791      elementNames = carnall[[1]][[2;;]];
3792      carnall      = carnall[[2;;]];
3793      carnallErr   = carnallErr[[2;;]];
3794      carnall      = Transpose[carnall];
3795      carnallErr   = Transpose[carnallErr];
3796      paramNames   = ToExpression/@carnall[[1]][[1;;]];
3797      carnall      = carnall[[2;;]];
3798      carnallErr   = carnallErr[[2;;]];
3799      carnallData  = Table[(
3800                      data         = carnall[[i]];
3801                      data         = (#[[1]]->#[[2]])&/@Select[
          Transpose[{paramNames,data}],#[[2]]!=""&];
3802                      elementNames[[i]]->data
3803                      ),
3804                      {i,1,13}
3805                      ];
3806      carnallData  = Association[carnallData];
3807      carnallNotes = Table[(
3808                      data         = carnallErr[[i]];
```

```
3809                    elementName = elementNames[[i]];
3810                    dataFun     = (
3811                        #[[1]] -> If[#[[2]]=="[]",
3812                        "Not allowed to vary in fitting.",
3813                        If[#[[2]]=="[R]",
3814                            "Ratio constrained by: " <> <|"Eu"->"F4/
    F2=0.713; F6/F2=0.512",
3815                                "Gd"->"F4/F2=0.710]",
3816                                "Tb"->"F4/F2=0.707"|>[elementName],
3817                            If[#[[2]]=="i",
3818                                "Interpolated",
3819                                #[[2]]
3820                            ]
3821                        ]
3822                        ]) &;
3823                    data = dataFun /@ Select[Transpose[{paramNames,
    data}],#[[2]]!=""&];
3824                    elementName->data
3825                      ),
3826                {i,1,13}
3827                ];
3828    carnallNotes  = Association[carnallNotes];
3829
3830    annotatedData = Table[
3831                    If[NumberQ[#[[1]]],Tooltip[#[[1]],#[[2]]],""] & /
    @ Transpose[{paramNames/.carnallData[element],
3832                      paramNames/.carnallNotes[element]
3833                        }],
3834                    {element,elementNames}
3835                    ];
3836    annotatedData = Transpose[annotatedData];
3837
3838    Carnall = <|"data"      -> carnallData,
3839        "annotations"       -> carnallNotes,
3840        "paramSymbols"      -> paramNames,
3841        "elementNames"      -> elementNames,
3842        "rawData"           -> carnall,
3843        "rawAnnotations"    -> carnallErr,
3844        "includedTableIons" -> ions,
3845        "annnotatedData"    -> annotatedData
3846    |>;
3847
3848    Do[(
3849        carnallData   = Import[FileNameJoin[{moduleDir,"data","
    Carnall.xls"}]][[sheetIdx]];
3850        headers       = carnallData[[1]];
3851        calcIndex     = Position[headers,"Calc (1/cm)"][[1,1]];
3852        headers       = headers[[2;;]];
3853        carnallLabels = carnallData[[1]];
3854        carnallData   = carnallData[[2;;]];
3855        carnallTerms  = DeleteDuplicates[First/@carnallData];
3856        parsedData    = Table[(
3857                            rows = Select[carnallData,#[[1]]==term&];
3858                            rows = #[[2;;]]&/@rows;
3859                            rows = Transpose[rows];
```

```
3860                            rows = Transpose[{headers,rows}];
3861                            rows = Association[(#[[1]]->#[[2]])&/@rows
    ];
3862                            term->rows
3863                            ),
3864                        {term,carnallTerms}
3865                        ];
3866          carnallAssoc         = Association[parsedData];
3867          carnallCalcEnergies  = #[[calcIndex]]&/@carnallData;
3868          carnallCalcEnergies  = If[NumberQ[#],#,Missing[]]&/
    @carnallCalcEnergies;
3869          ion                  = ions[[sheetIdx-3]];
3870          carnallCalcEnergies  = PadRight[carnallCalcEnergies, fullSizes
    [ion], Missing[]];
3871          keys                 = #[<|"ion"->ion|>]&/@templates;
3872          Carnall[keys[[1]]]   = carnallAssoc;
3873          Carnall[keys[[2]]]   = carnallCalcEnergies;
3874          Carnall[keys[[3]]]   = carnallData;
3875          Carnall[keys[[4]]]   = headers;
3876          ),
3877      {sheetIdx,4,16}
3878      ];
3879
3880      goodions = Select[ions,#!="Pm"&];
3881      expData  = Select[Transpose[Carnall["appendix:"<>#<>":RawTable"
    ]][[1+Position[Carnall["appendix:"<>#<>":Headings"],"Exp (1/cm)"
    ][[1,1]]]],NumberQ]&/@goodions;
3882      Carnall["All Experimental Data"]=AssociationThread[goodions,
    expData];
3883      If[OptionValue["Export"],
3884        (
3885          carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"
    }];
3886          Print["Exporting to "<>carnallFname];
3887          Export[carnallFname, Carnall];
3888        )
3889        ];
3890      Return[Carnall];
3891    );
3892
3893    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";
3894
3895    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[].";
3896
3897    LoadCFP::usage="LoadCFP[] loads CFP, CFPAssoc, and CFPTable into
      the session.";
```

```
3898  LoadCFP[]:=(
3899    If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]],Return
      []];
3900
3901    PrintTemporary["Loading CFPtable ..."];
3902    CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
3903    If[!FileExistsQ[CFPTablefname],
3904      (PrintTemporary[">> CFPTable.m not found, generating ..."];
3905        CFPTable = GenerateCFPTable["Export"->True];
3906      ),
3907      CFPTable = Import[CFPTablefname];
3908    ];
3909
3910    PrintTemporary["Loading CFPs.m ..."];
3911    CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
3912    If[!FileExistsQ[CFPfname],
3913      (PrintTemporary[">> CFPs.m not found, generating ..."];
3914        CFP = GenerateCFP["Export"->True];
3915      ),
3916      CFP = Import[CFPfname];
3917    ];
3918
3919    PrintTemporary["Loading CFPAssoc.m ..."];
3920    CFPAfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
3921    If[!FileExistsQ[CFPAfname],
3922      (PrintTemporary[">> CFPAssoc.m not found, generating ..."];
3923        CFPAssoc = GenerateCFPAssoc["Export"->True];
3924      ),
3925      CFPAssoc = Import[CFPAfname];
3926    ];
3927  );
3928
3929  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[].";
3930
3931  LoadUk::usage="LoadUk[] loads into session the reduced matrix
      elements for unit tensor operators.";
3932  LoadUk[]:=(
3933    If[ValueQ[ReducedUkTable], Return[]];
3934    PrintTemporary["Loading the association of reduced matrix
      elements for unit tensor operators ..."];
3935    ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "
      ReducedUkTable.m"}];
3936    If[!FileExistsQ[ReducedUkTableFname],
3937      (PrintTemporary[">> ReducedUkTable.m not found, generating ..."
      ];
3938        ReducedUkTable = GenerateReducedUkTable[7];
3939      ),
3940      ReducedUkTable = Import[ReducedUkTableFname];
3941    ];
3942  );
3943
3944  ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}]
```

```
    provides the calculated result of Electrostatic[{n, SL, SpLp}].
    Load using LoadElectrostatic[].";
3945
3946  LoadElectrostatic::usage="LoadElectrostatic[] loads the reduced
    matrix elements for the electrostatic interaction.";
3947  LoadElectrostatic[]:=(
3948    If[ValueQ[ElectrostaticTable], Return[]];
3949    PrintTemporary["Loading the association of matrix elements for
    the electrostatic interaction ..."];
3950    ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "
    ElectrostaticTable.m"}];
3951    If[!FileExistsQ[ElectrostaticTablefname],
3952      (PrintTemporary[">> ElectrostaticTable.m not found, generating
    ..."];
3953        ElectrostaticTable = GenerateElectrostaticTable[7];
3954      ),
3955      ElectrostaticTable = Import[ElectrostaticTablefname];
3956    ];
3957  );
3958
3959  LoadV1k::usage="LoadV1k[] loads into session the matrix elements of
     V1k.";
3960  LoadV1k[]:=(
3961    If[ValueQ[ReducedV1kTable], Return[]];
3962    PrintTemporary["Loading the association of matrix elements for
    V1k ..."];
3963    ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "
    ReducedV1kTable.m"}];
3964    If[!FileExistsQ[ReducedV1kTableFname],
3965      (PrintTemporary[">> ReducedV1kTable.m not found, generating ...
    "];
3966        ReducedV1kTable = GenerateReducedV1kTable[7];
3967      ),
3968      ReducedV1kTable = Import[ReducedV1kTableFname];
3969    ]
3970  );
3971
3972  LoadSpinOrbit::usage="LoadSpinOrbit[] loads into session the matrix
     elements of the spin-orbit interaction.";
3973  LoadSpinOrbit[]:=(
3974    If[ValueQ[SpinOrbitTable], Return[]];
3975    PrintTemporary["Loading the association of matrix elements for
    spin-orbit ..."];
3976    SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "
    SpinOrbitTable.m"}];
3977    If[!FileExistsQ[SpinOrbitTableFname],
3978      (PrintTemporary[">> SpinOrbitTable.m not found, generating ..."
    ];
3979        SpinOrbitTable = GenerateSpinOrbitTable[7, "Export" -> True];
3980      ),
3981      SpinOrbitTable = Import[SpinOrbitTableFname];
3982    ]
3983  );
3984
3985  LoadSOOandECSOLS::usage="LoadSOOandECSOLS[] loads into session the
```

```
  LS reduced matrix elements of the SOO-ECSO interaction.";
LoadSOOandECSOLS[]:=(
  If[ValueQ[SOOandECSOLSTable], Return[]];
  PrintTemporary["Loading the association of LS reduced matrix
  elements for SOO-ECSO ..."];
  SOOandECSOLSTableFname = FileNameJoin[{moduleDir, "data", "
  ReducedSOOandECSOLSTable.m"}];
  If[!FileExistsQ[SOOandECSOLSTableFname],
    (PrintTemporary[">> ReducedSOOandECSOLSTable.m not found,
  generating ..."];
      SOOandECSOLSTable = GenerateSOOandECSOLSTable[7];
    ),
    SOOandECSOLSTable = Import[SOOandECSOLSTableFname];
  ];
);

LoadSOOandECSO::usage="LoadSOOandECSO[] loads into session the LSJ
  reduced matrix elements of spin-other-orbit and electrostatically-
  correlated-spin-orbit.";
LoadSOOandECSO[]:=(
  If[ValueQ[SOOandECSOTableFname], Return[]];
  PrintTemporary["Loading the association of matrix elements for
  spin-other-orbit and electrostatically-correlated-spin-orbit ..."
  ];
  SOOandECSOTableFname = FileNameJoin[{moduleDir, "data", "
  SOOandECSOTable.m"}];
  If[!FileExistsQ[SOOandECSOTableFname],
    (PrintTemporary[">> SOOandECSOTable.m not found, generating ...
  "];
      SOOandECSOTable = GenerateSOOandECSOTable[7, "Export"->True];
    ),
    SOOandECSOTable = Import[SOOandECSOTableFname];
  ];
);

LoadT22::usage="LoadT22[] loads into session the matrix elements of
   T22.";
LoadT22[]:=(
  If[ValueQ[T22Table], Return[]];
  PrintTemporary["Loading the association of reduced T22 matrix
  elements ..."];
  T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table
  .m"}];
  If[!FileExistsQ[T22TableFname],
    (PrintTemporary[">> ReducedT22Table.m not found, generating ...
  "];
      T22Table = GenerateT22Table[7];
    ),
    T22Table = Import[T22TableFname];
  ];
);

LoadSpinSpin::usage="LoadSpinSpin[] loads into session the matrix
  elements of spin-spin.";
LoadSpinSpin[]:=(
```

```
4026      If[ValueQ[SpinSpinTable], Return[]];
4027      PrintTemporary["Loading the association of matrix elements for
          spin-spin ..."];
4028      SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "
          SpinSpinTable.m"}];
4029      If[!FileExistsQ[SpinSpinTableFname],
4030        (PrintTemporary[">> SpinSpinTable.m not found, generating ..."
          ];
4031          SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
4032        ),
4033        SpinSpinTable = Import[SpinSpinTableFname];
4034      ];
4035    );
4036
4037    LoadThreeBody::usage="LoadThreeBody[] loads into session the matrix
         elements of three-body configuration-interaction effects.";
4038    LoadThreeBody[]:=(
4039      If[ValueQ[ThreeBodyTable], Return[]];
4040      PrintTemporary["Loading the association of matrix elements for
          three-body configuration-interaction effects ..."];
4041      ThreeBodyFname   = FileNameJoin[{moduleDir, "data", "
          ThreeBodyTable.m"}];
4042      ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "
          ThreeBodyTables.m"}];
4043      If[!FileExistsQ[ThreeBodyFname],
4044        (PrintTemporary[">> ThreeBodyTable.m not found, generating ..."
          ];
4045          {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTables
          [14, "Export" -> True];
4046        ),
4047        ThreeBodyTable = Import[ThreeBodyFname];
4048        ThreeBodyTables = Import[ThreeBodiesFname];
4049      ];
4050    );
4051
4052    (* ###################### Load Functions ###################### *)
4053    (* ########################################################### *)
4054
4055  End[];
4056
4057  LoadTermLabels[];
4058  LoadCFP[];
4059
4060  EndPackage[];
```

12.2 qonstants.m

This file has a few constants and conversion factors.

```
1  BeginPackage["qonstants`"];
2
3  (* Physical Constants*)
4  bohrRadius = 5.29177210903 * 10^-9;
5  ee         = 1.602176634 * 10^-19;
```

```
6
7  (* Spectroscopic niceties*)
8  theLanthanides = {"Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy
       ", "Ho", "Er", "Tm", "Yb"};
9  theActinides   = {"Ac", "Th", "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk"
       , "Cf", "Es", "Fm", "Md", "No", "Lr"};
10 theTrivalents  = {"Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho
       ", "Er", "Tm"};
11 specAlphabet   = "SPDFGHIKLMNOQRTUV"
12
13 (* SI *)
14 \[Mu]0  = 4 \[Pi]*10^-7;              (* magnetic permeability in
       vacuum in SI *)
15 hPlanck = 6.62607015*10^-34;          (* Planck's constant in SI *)
16 \[Mu]B  = 9.2740100783*10^-24;        (* Bohr magneton in SI *)
17 me      = 9.1093837015*10^-31;        (* electron mass in SI *)
18 cLight  = 2.99792458*10^8;            (* speed of light in SI *)
19 eCharge = 1.602176634*10^-19;         (* elementary charge in SI *)
20 αFine = 1/137.036;              (* fine structure constant in SI *)
21 bohrRadius   = 5.29177210903*10^-11; (* Bohr radius in SI *)
22
23 (* Hartree atomic units *)
24 hPlanckHartree = 2 \[Pi]; (* Planck's constant in Hartree *)
25 meHartree      = 1;       (* electron mass in Hartree *)
26 cLightHartree  = 137.036; (* speed of light in Hartree *)
27 eChargeHartree = 1;       (* elementary charge in Hartree *)
28 \[Mu]0Hartree  = αFine^2; (* magnetic permeability in vacuum in
       Hartree *)
29
30 (* some conversion factors *)
31 eVtoKayser    = 8065.54429;   (* 1 eV = 8065.54429 cm^-1 *)
32 KaysertoeV    = 1/eVtoKayser; (* 1 cm^-1 = 1/8065.54429 eV *)
33 TeslaToKayser = 2 * \[Mu]B / hPlanck / cLight / 100;
34
35 EndPackage[];
```

12.3 qplotter.m

This module has a few useful plotting routines.

```
1
2  BeginPackage["qplotter`"];
3
4  GetColor;
5  IndexMappingPlot;
6  ListLabelPlot;
7  AutoGraphicsGrid;
8  SpectrumPlot;
9  WaveToRGB;
10
11 Begin["`Private`"];
12
13   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.";
14  Options[AutoGraphicsGrid] = Options[GraphicsGrid];
15  AutoGraphicsGrid[graphsList_, opts : OptionsPattern[]] :=
16    (
17      numGraphs = Length[graphsList];
18      width = Floor[Sqrt[numGraphs]];
19      height = Ceiling[numGraphs/width];
20      groupedGraphs = Partition[graphsList, width, width, 1, Null];
21      GraphicsGrid[groupedGraphs, opts]
22    )
23
24  Options[IndexMappingPlot] = Options[Graphics];
25  IndexMappingPlot::usage =
26    "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.";
27  IndexMappingPlot[pairs_, opts : OptionsPattern[]] := Module[{width,
     height}, (
28    width = Max[First /@ pairs];
29    height = width/3;
30    Return[
31      Graphics[{{Tooltip[Point[{#[[1]], 0}],#[[1]]]}, Tooltip[Point
    [{#[[2]], height}],#[[2]]],
32            Line[{{#[[1]], 0}, {#[[2]], height}}]} & /@ pairs, opts,
    ImageSize -> 800]]
33    )
34    ]
35
36  TickCompressor[fTicks_] :=
37  Module[{avgTicks, prevTickLabel, groupCounter, groupTally, idx,
38    tickPosition, tickLabel, avgPosition, groupLabel}, (avgTicks =
    {};
39    prevTickLabel = fTicks[[1, 2]];
40    groupCounter = 0;
41    groupTally = 0;
42    idx = 1;
43    Do[({tickPosition, tickLabel} = tick;
44      If[
45      tickLabel === prevTickLabel,
46      (groupCounter += 1;
47        groupTally += tickPosition;
48        groupLabel = tickLabel;),
49      (
50        avgPosition = groupTally/groupCounter;
51        avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
52        groupCounter = 1;
53        groupTally = tickPosition;
54        groupLabel = tickLabel;
55        )
56      ];
57      If[idx != Length[fTicks],
58      prevTickLabel = tickLabel;
59      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] = Join[Options[ListPlot], {"TickCompression"
  ->True,
"LabelLevels"->1}];
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[#, 90 Degree], 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}};)
104   ];
105   ListPlot[scatterGroups,
106     opts,
107     Frame->True,
108     AxesStyle -> {Directive[Black, Dotted], Automatic},
109     PlotStyle -> groupedColors,
110     FrameTicks -> frameTicks]
111   )
112   ]
113
114 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. ";
115 WaveToRGB[wave_, gamma_ : 0.8] := (
116   wavelength = (wave);
117   Which[
118     wavelength < 380,
119       wavelength = 380,
120     wavelength > 750,
121       wavelength = 750
122   ];
123   Which[380 <= wavelength <= 440,
124   (
125     attenuation = 0.3 + 0.7*(wavelength - 380)/(440 - 380);
126     R = ((-(wavelength - 440)/(440 - 380))*attenuation)^gamma;
127     G = 0.0;
128     B = (1.0*attenuation)^gamma;
129     ),
130   440 <= wavelength <= 490,
131   (
132     R = 0.0;
133     G = ((wavelength - 440)/(490 - 440))^gamma;
134     B = 1.0;
135     ),
136   490 <= wavelength <= 510,
137   (
138     R = 0.0;
139     G = 1.0;
140     B = (-(wavelength - 510)/(510 - 490))^gamma;
141     ),
142   510 <= wavelength <= 580,
143   (
144     R = ((wavelength - 510)/(580 - 510))^gamma;
145     G = 1.0;
146     B = 0.0;
147     ),
148   580 <= wavelength <= 645,
149   (
150     R = 1.0;
151     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}]
```

```
200       };
201     Return[polys]
202     );
203   ]
204
205   Options[SpectrumPlot] = Options[Graphics];
206   Options[SpectrumPlot] = Join[Options[SpectrumPlot], {"Intensities"
        -> {},"Tooltips" -> True, "Comments" -> {}, "SpectrumFunction" ->
        WaveToRGB}];
207   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.
208   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.
209   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.
210   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.
211   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.
212   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.
213   ";
214   SpectrumPlot[lines_, widthToHeightAspect_, lineWidth_, opts :
        OptionsPattern[]] := Module[
215     {minWave, maxWave, height, fuzzyLines},
216   (
217     colorFun = OptionValue["SpectrumFunction"];
218     {minWave, maxWave} = MinMax[lines];
219     height     = (maxWave - minWave)/widthToHeightAspect;
220     fuzzyLines = Which[
221       NumberQ[lineWidth] && Length[OptionValue["Intensities"]] == 0,
222         FuzzyRectangle[#, lineWidth, 0, height, colorFun[#]] &/@
        lines,
223       Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]]
        == 0,
```

```
224         MapThread[FuzzyRectangle[#1, #2, 0, height, colorFun[#1]] &,
    {lines, lineWidth}],
225       NumberQ[lineWidth] && Length[OptionValue["Intensities"]] > 0,
226         MapThread[FuzzyRectangle[#1, lineWidth, 0, height, colorFun
    [#1], #2] &, {lines, OptionValue["Intensities"]}],
227       Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]] >
     0,
228         MapThread[FuzzyRectangle[#1, #2, 0, height, colorFun[#1], #3]
     &, {lines, lineWidth, OptionValue["Intensities"]}]
229     ];
230     comments = Which[
231       Length[OptionValue["Comments"]] > 0,
232         MapThread[StringJoin[ToString[#1]<>" nm","\n",ToString[#2]]&,
233         {lines, OptionValue["Comments"]}],
234       Length[OptionValue["Comments"]] == 0,
235         ToString[#] <>" nm" & /@ lines,
236       True,
237       {}
238     ];
239     If[OptionValue["Tooltips"],
240       fuzzyLines = MapThread[Tooltip[#1, #2] &, {fuzzyLines, comments
    }];
241     ];
242     graphicsOpts = FilterRules[{opts}, Options[Graphics]];
243     Graphics[fuzzyLines,
244         graphicsOpts,
245         Background -> Black,
246         PlotRange -> {{minWave, maxWave}, {0, height}}]
247     )
248   ];
249
250 End[];
251
252 EndPackage[];
```

12.4 misc.m

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

```
1  BeginPackage["misc`"];
2
3  ExportToH5;
4  FlattenBasis;
5  RecoverBasis;
6  FlowMatching;
7  SuperIdentity;
8  RobustMissingQ;
9  ReplaceDiagonal;
10
11 GreedyMatching;
12 HelperNotebook;
13 StochasticMatching;
14 ExtractSymbolNames;
15 GetModificationDate;
```

```
TextBasedProgressBar;
ToPythonSparseFunction;

FirstOrderPerturbation;
SecondOrderPerturbation;
RoundValueWithUncertainty;

ToPythonSymPyExpression;
RoundToSignificantFigures;
RobustMissingQ;

BlockMatrixMultiply;
BlockAndIndex;
TruncateBlockArray;
BlockArrayDimensionsArray;
ArrayBlocker;
BlockTranspose;

Begin["`Private`"];

  BlockTranspose[anArray_]:=(
    Map[Transpose, Transpose[anArray], {2}]
  );

  BlockMatrixMultiply::usage="BlockMatrixMultiply[A,B] gives the
    matrix multiplication of A and B, with A and B having a compatible
     block structure that allows for matrix multiplication into a
    congruent block structure.";
  BlockMatrixMultiply[Amat_,Bmat_]:=Module[{rowIdx,colIdx,sumIdx},
  (
    Table[
      Sum[Amat[[rowIdx,sumIdx]].Bmat[[sumIdx,colIdx]],{sumIdx,1,
    Dimensions[Amat][[2]]}],
    {rowIdx,1,Dimensions[Amat][[1]]},
    {colIdx,1,Dimensions[Bmat][[2]]}
    ]
  )
  ];

  BlockAndIndex::usage="BlockAndIndex[blockSizes, index] takes a list
     of bin widths and index. The function return in which block the
    index would be, were the bins to be layed out from left to right.
    The function also returns the position within the bin in which it
    is accomodated. The function returns these two numbers as a list
    of two elements {blockIndex, blockSubIndex}";
  BlockAndIndex[blockSizes_List, index_Integer]:=Module[{
    accumulatedBlockSize,blockIndex, blockSubIndex},
  (
    accumulatedBlockSize = Accumulate[blockSizes];
    If[accumulatedBlockSize[[-1]]-index<0,
      Print["Index out of bounds"];
      Abort[]
    ];
    blockIndex    = Flatten[Position[accumulatedBlockSize-index,n_ /;
     n>=0]][[1]];
```

```
60      blockSubIndex = Mod[index-accumulatedBlockSize[[blockIndex]],
        blockSizes[[blockIndex]],1];
61      Return[{blockIndex,blockSubIndex}]
62    )
63    ];
64
65    TruncateBlockArray::usage="TruncateBlockArray[blockArray,
        truncationIndices, blockWidths] takes a an array of blocks and
        selects the columns and rows corresponding to truncationIndices.
        The indices being given in what would be the ArrayFlatten[
        blockArray] version of the array. They blocks in the given array
        may be SparseArray. This is equivalent to FlattenArray[blockArray
        ][truncationIndices, truncationIndices] but may be more efficient
        blockArray is sparse.";
66    TruncateBlockArray[blockArray_,truncationIndices_,blockWidths_]:=
        Module[
67    {truncatedArray,blockCol,blockRow,blockSubCol,blockSubRow},(
68    truncatedArray = Table[
69      {blockCol,blockSubCol} = BlockAndIndex[blockWidths,fullColIndex];
70      {blockRow,blockSubRow} = BlockAndIndex[blockWidths,fullRowIndex];
71      blockArray[[blockRow,blockCol]][[blockSubRow,blockSubCol]],
72    {fullColIndex,truncationIndices},
73    {fullRowIndex,truncationIndices}
74    ];
75    Return[truncatedArray]
76    )
77    ];
78
79    BlockArrayDimensionsArray::usage="BlockArrayDimensionsArray[
        blockArray] returns the array of block sizes in a given blocked
        array.";
80    BlockArrayDimensionsArray[blockArray_]:=(
81      Map[Dimensions,blockArray,{2}]
82    );
83
84    ArrayBlocker::usage="ArrayBlocker[anArray, blockSizes] takes a flat
         2d array and a congruent 2D array of block sizes, and with them
        it returns the original array with the block structure imposed by
        blockSizes. The resulting array satisfies ArrayFlatten[
        blockedArray] == anArray, and also Map[Dimensions, blockedArray
        ,{2}] == blockSizes.";
85    ArrayBlocker[anArray_,blockSizes_]:=Module[{rowStart,colStart,
        colEnd,numBlocks,blockedArray,blockSize,rowEnd,aBlock,idxRow,
        idxCol},(
86      rowStart  = 1;
87      colStart  = 1;
88      colEnd    = 1;
89      numBlocks = Length[blockSizes];
90      blockedArray = Table[(
91        blockSize = blockSizes[[idxRow,idxCol]];
92        rowEnd    = rowStart+blockSize[[1]]-1;
93        colEnd    = colStart+blockSize[[2]]-1;
94        aBlock    = anArray[[rowStart;;rowEnd,colStart;;colEnd]];
95        colStart  = colEnd+1;
96        If[idxCol==numBlocks,
```

```
97          rowStart=rowEnd+1;
98          colStart=1;
99        ];
100       aBlock
101     ),
102     {idxRow,1,numBlocks},
103     {idxCol,1,numBlocks}
104     ];
105     Return[blockedArray]
106   )
107   ];
108
109   ReplaceDiagonal::usage =
110     "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.";
111   ReplaceDiagonal[matrix_, repValue_] :=
112     ReplacePart[matrix,
113       Table[{i, i} -> repValue, {i, 1, Length[matrix]}]];
114
115   Options[RoundValueWithUncertainty] = {"SetPrecision" -> False};
116   RoundValueWithUncertainty::usage =
117     "RoundValueWithUncertainty[x,dx] given a number x together with
      an \
118   uncertainty dx this function rounds x to the first significant
      figure \
119   of dx and also rounds dx to have a single significant figure.
120   The returned value is a list with the form {roundedX, roundedDx}.
121   The option \"SetPrecision\" can be used to control whether the \
122   Mathematica precision of x and dx is also set accordingly to these
      \
123   rules, otherwise the rounded numbers still have the original \
124   precision of the input values.
125   If the position of the first significant figure of x is after the \
126   position of the first significant figure of dx, the function
      returns \
127   {0,dx} with dx rounded to one significant figure.";
128   RoundValueWithUncertainty[x_, dx_, OptionsPattern[]] := Module[
129     {xExpo, dxExpo, sigFigs, roundedX, roundedDx, returning},
130     (
131       xExpo = RealDigits[x][[2]];
132       dxExpo = RealDigits[dx][[2]];
133       sigFigs = xExpo - dxExpo + 1;
134       {roundedX, roundedDx} = If[sigFigs <= 0,
135         {0., N@RoundToSignificantFigures[dx, 1]},
136         N[
137         {
138           RoundToSignificantFigures[x, xExpo - dxExpo + 1],
139           RoundToSignificantFigures[dx, 1]}
140         ]
141         ];
```

```
142       returning = If[
143         OptionValue["SetPrecision"],
144         {SetPrecision[roundedX, Max[1, sigFigs]],
145         SetPrecision[roundedDx, 1]},
146         {roundedX, roundedDx}
147         ];
148       Return[returning]
149       )
150     ];
151
152   RoundToSignificantFigures::usage =
153     "RoundToSignificantFigures[x, sigFigs] rounds x so that it only has \
154   sigFigs significant figures.";
155   RoundToSignificantFigures[x_, sigFigs_] :=
156     Sign[x]*N[FromDigits[RealDigits[x, 10, sigFigs]]];
157
158   RobustMissingQ[expr_] := (FreeQ[expr, _Missing] === False);
159
160   TextBasedProgressBar[progress_, totalIterations_, prefix_:""] :=
       Module[
161       {progMessage},
162       progMessage = ToString[progress] <> "/" <> ToString[
       totalIterations];
163       If[progress < totalIterations,
164           WriteString["stdout", StringJoin[prefix, progMessage, "\r"
       ]],
165           WriteString["stdout", StringJoin[prefix, progMessage, "\n"
       ]]
166       ];
167   ];
168
169   FirstOrderPerturbation::usage="Given the 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. This
       assuming that the eigenvalues are non-degenerate.";
170   FirstOrderPerturbation[eigenVectors_,
171     perMatrix_] := (Diagonal[
172     eigenVectors . perMatrix . Transpose[eigenVectors]])
173
174   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};

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.";
FlattenBasis[basis_] := Module[{flatbasis, flatbasisLS,
  flatbasisNums},
  (
    flatbasis = Flatten[basis];
    flatbasisLS = flatbasis[[1 ;; ;; 4]];
    flatbasisNums = Select[flatbasis, Not[StringQ[#]] &];
    Return[{flatbasisLS, flatbasisNums}]
    )
  ];

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
  }.";
RecoverBasis[{flatbasisLS_, flatbasisNums_}] := Module[{recBasis},
  (
  recBasis = {{{#[[1]], #[[2]]}, #[[3]]}, #[[4]]} & /@ (Flatten /@
      Transpose[{flatbasisLS,
        Partition[Round[2*#]/2 & /@ flatbasisNums, 3]}]);
  Return[recBasis];
  )
  ]

ExtractSymbolNames[expr_Hold] := Module[
  {strSymbols},
  strSymbols = ToString[expr, InputForm];
  StringCases[strSymbols, RegularExpression["\\w+"]][[2 ;;]]
  ]

ExportToH5::usage =
  "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.
217  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.";
218 Options[ExportToH5] = {"Overwrite" -> True};
219 ExportToH5[fname_String, symbols_Hold, OptionsPattern[]] := (
220   If[And[FileExistsQ[fname], OptionValue["Overwrite"]],
221   (
222     Print["File already exists, overwriting ..."];
223     DeleteFile[fname];
224     )
225   ];
226   symbolNames = ExtractSymbolNames[symbols];
227   Do[(Print[symbolName];
228     Export[fname, ToExpression[symbolName], {"Datasets", symbolName
    },
229     OverwriteTarget -> "Append"]
230     ), {symbolName, symbolNames}]
231   )
232
233 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.";
234 Options[GreedyMatching] = {
235     "alistLabels" -> {},
236     "blistLabels" -> {}};
237 GreedyMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{
238   aValues = aValues0,
239   bValues = bValues0,
240   bValuesOriginal = bValues0,
241   bestLabels, bestMatches,
242   bestLabel, aElement, givenLabels,
243   aLabels, aLabel,
244   diffs, minDiff,
245   bLabels,
246   minDiffPosition, bestMatch},
247   (
248   aLabels     = OptionValue["alistLabels"];
249   bLabels     = OptionValue["blistLabels"];
250   bestMatches = {};
251   bestLabels  = {};
252   givenLabels = (Length[aLabels] > 0);
253   Do[
```

```
254       (
255       aElement         = aValues[[idx]];
256       diffs            = Abs[bValues - aElement];
257       minDiff          = Min[diffs];
258       minDiffPosition = Position[diffs, minDiff][[1, 1]];
259       bestMatch        = bValues[[minDiffPosition]];
260       bestMatches      = Append[bestMatches, {aElement, bestMatch}];
261       If[givenLabels,
262         (
263         aLabel      = aLabels[[idx]];
264         bestLabel   = bLabels[[minDiffPosition]];
265         bestLabels = Append[bestLabels, {aLabel, bestLabel}];
266         bLabels     = Drop[bLabels, {minDiffPosition}];
267         )
268         ];
269       bValues = Drop[bValues, {minDiffPosition}];
270       If[Length[bValues] == 0, Break[]];
271       ),
272       {idx, 1, Length[aValues]}
273       ];
274     pairedIndices = MapIndexed[{#2[[1]], Position[bValuesOriginal,
      #1[[2]]][[1, 1]]} &, bestMatches];
275     If[givenLabels,
276       Return[{bestMatches, bestLabels, pairedIndices}],
277       Return[{bestMatches, pairedIndices}]
278       ]
279     )
280     ]
281
282   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.";
283   Options[StochasticMatching] = {"alistLabels" -> {},
284     "blistLabels" -> {}};
285   StochasticMatching[aValues0_, bValues0_, numShuffles_ : 200,
      OptionsPattern[]] := Module[{
286     aValues = aValues0,
287     bValues = bValues0,
288     matchingLabels, ranger, matches, noShuff, bestMatch, highestCost,
       lowestCost, dev, sorter, bestValues,
289     pairedIndices, bestLabels, matchedIndices, shuffler
290     },
291     (
292     matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
293     ranger = Range[1, Length[aValues]];
294     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
    \"notMatched\" 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 \"notMatched\" 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.";
346 Options[FlowMatching] = {"alistLabels" -> {}, "blistLabels" -> {},
     "notMatched" -> 0, "CostFun"-> (Abs[#1-#2] &)};
347 FlowMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{
348   aValues = aValues0, bValues = bValues0, edgesSourceToA,
    capacitySourceToA, nA, nB,
349   costSourceToA, midLayer, midLayerEdges, midCapacities,
350   midCosts, edgesBtoSink, capacityBtoSink, costBtoSink,
351   allCapacities, allCosts, allEdges, graph,
352   flow, bestValues, bestLabels, cFun,
353   aLabels, bLabels, pairedIndices, matchingLabels},
354   (
355   matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
356   aLabels = OptionValue["alistLabels"];
357   bLabels = OptionValue["blistLabels"];
358   cFun    = OptionValue["CostFun"];
359   nA      = Length[aValues];
360   nB      = Length[bValues];
361   (*Build up the edges costs and capacities*)
362   (*From source to the nodes representing the values of the first \
363 list*)
364   edgesSourceToA    = ("source" \[DirectedEdge] {"A", #}) & /@
     Range[1, nA];
365   capacitySourceToA = ConstantArray[1, nA];
366   costSourceToA     = ConstantArray[0, nA];
367
368   (*From all the elements of A to all the elements of B*)
369   midLayer = Table[{{"A", i} \[DirectedEdge] ({"B", j}), 1, cFun[
    aValues[[i]], bValues[[j]]]}, {i, 1, nA}, {j, 1, nB}];
370   midLayer = Flatten[midLayer, 1];
371   {midLayerEdges, midCapacities, midCosts} = Transpose[midLayer];
372
373   (*From the elements of B to the sink*)
374   edgesBtoSink    = ({"B", #} \[DirectedEdge] "sink") & /@ Range[1,
     nB];
375   capacityBtoSink = ConstantArray[1, nB];
376   costBtoSink     = ConstantArray[0, nB];
377
378   (*Put it all together*)
379   allCapacities = Join[capacitySourceToA, midCapacities,
    capacityBtoSink];
380   allCosts      = Join[costSourceToA, midCosts, costBtoSink];
381   allEdges      = Join[edgesSourceToA, midLayerEdges, edgesBtoSink
    ];
```

```
382     graph          = Graph[allEdges, EdgeCapacity -> allCapacities,
383       EdgeCost -> allCosts];
384
385     (*Solve it*)
386     flow           = FindMinimumCostFlow[graph, "source", "sink", nA -
        OptionValue["notMatched"], "OptimumFlowData"];
387     (*Collect the pairs of matched indices*)
388     pairedIndices = Select[flow["EdgeList"], And[Not[#[[1]] === "
      source"], Not[#[[2]] === "sink"]] &];
389     pairedIndices = {#[[1, 2]], #[[2, 2]]} & /@ pairedIndices;
390     (*Collect the pairs of matched values*)
391     bestValues     = {aValues[[#[[1]]]], bValues[[#[[2]]]]} & /@
      pairedIndices;
392     (*Account for having been given labels*)
393     If[matchingLabels,
394       (
395       bestLabels = {aLabels[[#[[1]]]], bLabels[[#[[2]]]]} & /@
      pairedIndices;
396       Return[{bestValues, bestLabels, pairedIndices}]
397       ),
398       (
399       Return[{bestValues, pairedIndices}]
400       )
401       ];
402     )
403     ]
404
405   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.";
406   HelperNotebook[nbName_:"OUT"] :=
407   Module[{screenDims, screenWidth, screenHeight, nbWidth, leftMargin,
408     PrintToOutputNb}, (
409     screenDims =
410       SystemInformation["Devices", "ScreenInformation"][[1, 2, 2]];
411     screenWidth = screenDims[[1, 2]];
412     screenHeight = screenDims[[2, 2]];
413     nbWidth = Round[screenWidth/3];
414     leftMargin = screenWidth - nbWidth;
415     outputNb = CreateDocument[{}, WindowTitle -> nbName,
416       WindowMargins -> {{leftMargin, Automatic}, {Automatic,
417           Automatic}}, WindowSize -> {nbWidth, screenHeight}];
418     PrintToOutputNb[text_] :=
419       (
420           SelectionMove[outputNb, After, Notebook];
421           NotebookWrite[outputNb, Cell[BoxData[ToBoxes[text]], "
      Output"]];
422       );
423     Return[PrintToOutputNb]
424     )
425     ]
426
427   GetModificationDate::usage="GetModificationDate[fname] returns the
      modification date of the given file.";
```

```
GetModificationDate[theFileName_] := FileDate[theFileName, "Modification"];

(*Helper function to convert Mathematica expressions to standard form*)
StandardFormExpression[expr0_] := Module[{expr=expr0}, ToString[expr, InputForm]];

(*Helper function to translate to Python/SymPy expressions*)
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.";
ToPythonSymPyExpression[expr0_] := Module[{standardForm, expr=expr0},
  standardForm = StandardFormExpression[expr];
  StringReplace[standardForm, {
    "Power[" -> "Pow(",
    "Sqrt[" -> "sqrt(",
    "[" -> "(",
    "]" -> ")",
    "\\" -> "",
    "I" -> "1j",
    (*Remove special Mathematica backslashes*)
    "/" -> "/" (*Ensure division is represented with a slash*)}]];

ToPythonSparseFunction[sparseArray_SparseArray, funName_] :=
  Module[{data, rowPointers, columnIndices, dimensions, pyCode, vars,
    varList, dataPyList,
    colIndicesPyList},(*Extract unique symbolic variables from the \
SparseArray*)
  vars = Union[Cases[Normal[sparseArray], _Symbol, Infinity]];
  varList = StringRiffle[ToString /@ vars, ", "];
  (*varList=ToPythonSymPyExpression/@varList;*)
  (*Convert data to SymPy compatible strings*)
  dataPyList =
    StringRiffle[
    ToPythonSymPyExpression /@ Normal[sparseArray["NonzeroValues"]],
    ", "];
  colIndicesPyList =
    StringRiffle[
    ToPythonSymPyExpression /@ (Flatten[
        Normal[sparseArray["ColumnIndices"]] - 1]), ", "];
  (*Extract sparse array properties*)
  rowPointers = Normal[sparseArray["RowPointers"]];
  dimensions = Dimensions[sparseArray];
  (*Create Python code string*)pyCode = StringJoin[
    "#!/usr/bin/env python3\n\n",
    "from scipy.sparse import csr_matrix\n",
    "from sympy import *\n",
    "import numpy as np\n",
```

```
      "\n",
      "sqrt = np.sqrt\n",
      "\n",
      "def ", funName, "(",
      varList,
      "):\n",
      "    data = np.array([", dataPyList, "])\n",
      "    indices = np.array([",
      colIndicesPyList,
      "])\n",
      "    indptr = np.array([",
      StringRiffle[ToString /@ rowPointers, ", "], "])\n",
      "    shape = (", StringRiffle[ToString /@ dimensions, ", "],
      ")\n",
      "    return csr_matrix((data, indices, indptr), shape=shape)"];
    pyCode
    ];

End[];
EndPackage[];
```

12.5 qalculations.m

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

```
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
    parameters in the Hamiltonian are  taken from the values quoted by
     Carnall. It can use precomputed symbolic matrices for the
    Hamiltonian if they have been loaded already and defined as a
    value of symbolicHamiltonians.

The function returns a list with nine elements {rmsDifference,
    carnallEnergies, eigenEnergies, ln, carnallAssignments,
    simplerStateLabels, eigensys, basis, truncatedStates}. The
    elements of the list are as follows:

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
    used for comparison;
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. truncatedStates is the same as eigensys but with the truncated
    eigenvectors so that the total propability add up to at least
    eigenstateTruncationProbability.

This function admits the following options:
  - \"MakeNotebook\" -> True or False. If True, a notebook with a
    summary of the data is created. Default is True.
  - \"NotebookSave\" -> True or False. If True, the results notebook
    is saved automatically. Default is True.
  - \"eigenstateTruncationProbability\" -> 0.9. The probability sum
    of the truncated eigenvectors. Default is 0.9.
  - \"Include spin-spin\" -> True or False. If True, the spin-spin
    contribution to the magnetic interactions is included. Default is
    True.
  - \"Max Eigenstates in Table\" -> 100. The maximum number of
    eigenstates to be shown in the table shown in the results notebook
    . Default is 100.
  - \"Sparse\" -> True or False. If True, the numerical Hamiltonian
    is kept in sparse form. Default is True.
  - \"PrintFun\" -> Print, PrintTemporary, or other to serve as a
    printer for progress messages. Default is Print.
  - \"SaveData\" -> True or False. If True, the resulting data is
    saved to disk. Default is True.
  - \"ParamOverride\". An association that can override parameters in
     the Hamiltonian. Default is <||>. This override cannot change the
     inclusion or exclusion of the spin-spin contribution to the
    magnetic interactions, for this purpose use the option \"Include
    spin-spin\".
  - \"Append to Filename\" -> \"\". A string to append to the
    filename of the saved notebook and data files. Default is \"\".
  - \"Remove Kramers\" -> True or False. If True, the Kramers
    degeneracy is removed from the eigenstates. Default is True.
  - \"OutputDirectory\" -> \"calcs\". The directory where the output
    files are saved. Default is \"calcs\".
  - \"Explorer\" -> True or False. If True, the energy level diagram
    is interactive. Default is False.
";
Options[FastIonSolverLaF3] = {
  "MakeNotebook"      -> True,
  "NotebookSave"      -> True,
  "Include spin-spin" -> True,
  "eigenstateTruncationProbability" -> 0.9,
  "Max Eigenstates in Table" -> 100,
  "Sparse" -> True,
```

```
  "PrintFun" -> Print,
  "SaveData" -> True,
  "ParamOverride" -> <||>,
  "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";
    ParamOverride = OptionValue["ParamOverride"];
    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;
    ];

    (* Apply the parameter override *)
    Do[params[key] = ParamOverride[key],
      {key, Keys[ParamOverride]}
    ];

    (* Import the symbolic Hamiltonian *)
```

```
95      PrintFun["> Loading the symbolic Hamiltonian for this
        configuration ..."];
96      startTime = Now;
97      numH = 14 - numE;
98      numEH = Min[numE, numH];
99      C2vsimplifier = {
100       B12 -> 0, B14 -> 0, B16 -> 0, B34 -> 0, B36 -> 0, B56 -> 0,
101       S12 -> 0, S14 -> 0, S16 -> 0, S22 -> 0, S24 -> 0, S26 -> 0, S34
         -> 0, S36 -> 0,S44 -> 0, S46 -> 0, S56 -> 0, S66 -> 0,
102       T11p -> 0, T11 -> 0, T12 -> 0, T14 -> 0, T15 -> 0, T16 -> 0,
        T18 -> 0, T17 -> 0, T19 -> 0};
103     (* If the necessary symbolicHamiltonian is define load if not
        make it *)
104     simpleHam = If[
105       ValueQ[symbolicHamiltonians[numEH]],
106       symbolicHamiltonians[numEH],
107       SimplerSymbolicHamMatrix[numE, C2vsimplifier, "
        PrependToFilename" -> "C2v-", "Overwrite" -> False]
108     ];
109     endTime  = Now;
110     loadTime = QuantityMagnitude[endTime - startTime, "Seconds"];
111     PrintFun[">> Loading the symbolic Hamiltonian took ", loadTime, "
         seconds."];
112
113     (*Enforce the override to the spin-spin contribution to the
        magnetic interactions*)
114     params[\[Sigma]SS] = If[OptionValue["Include spin-spin"], 1, 0];
115
116     (*Everything that is not given is set to zero*)
117     params = ParamPad[params, "Print" -> False];
118     PrintFun[params];
119     numHam = ReplaceInSparseArray[simpleHam, params];
120     If[Not[OptionValue["Sparse"]],
121         numHam = Normal[numHam]
122     ];
123     PrintFun["> Calculating the SLJ basis ..."];
124     basis = BasisLSJMJ[numE];
125
126     (* Eigensolver *)
127     PrintFun["> Diagonalizing the numerical Hamiltonian ..."];
128     startTime = Now;
129     eigensys  = Eigensystem[numHam];
130     endTime   = Now;
131     diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
132     PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
133     eigensys = Chop[eigensys];
134     eigensys = Transpose[eigensys];
135
136     (* Shift the baseline energy *)
137     eigensys = ShiftedLevels[eigensys];
138     (* Sort according to energy *)
139     eigensys = SortBy[eigensys, First];
140     (* Grab just the energies *)
141     eigenEnergies = First /@ eigensys;
142
```

```
143     (* Energies are doubly degenerate in the case of odd number of
        electrons, keep only one *)
144     If[And[OddQ[numE], OptionValue["Remove Kramers"]],
145         (
146         PrintFun["> Since there's an odd number of electrons energies
         come in pairs, taking just one for each pair ..."];
147         eigenEnergies = eigenEnergies[[;; ;; 2]];
148         )
149     ];
150
151     (* Compare against the data quoted by Bill Carnall *)
152     PrintFun["> Comparing against the data from Carnall ..."];
153     mainKey             = StringTemplate["appendix:`Ln`:Association"
        ][<|"Ln" -> ln|>];
154     lnData              = Carnall[mainKey];
155     carnalKeys          = lnData // Keys;
156     repetitions         = Length[lnData[#]["Calc (1/cm)"]] & /@
        carnalKeys;
157     carnallAssignments = First /@ Carnall["appendix:" <> ln <> ":
        RawTable"];
158     carnallAssignments = Select[carnallAssignments, Not[# === ""] &];
159     carnalKey           = StringTemplate["appendix:`Ln`:Calculated"
        ][<|"Ln" -> ln|>];
160     carnallEnergies     = Carnall[carnalKey];
161
162     If[And[OddQ[numE], Not[OptionValue["Remove Kramers"]]],
163     (
164       PrintFun[">> The number of eigenstates and the number of quoted
         states don't match, removing the last state ..."];
165       carnallAssignments = Duplicator[carnallAssignments];
166       carnallEnergies    = Duplicator[carnallEnergies];
167     )
168     ];
169
170     (* For the difference take as many energies as quoted by Bill *)
171     eigenEnergies = eigenEnergies + carnallEnergies[[1]];
172     diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] -
        carnallEnergies;
173     (* Remove the differences where the appendix tables have elided
        values*)
174     rmsDifference = Sqrt[Mean[(Select[diffs, FreeQ[#, Missing[]] &])
        ^2]];
175     titleTemplate = StringTemplate[
176       "Energy Level Diagram of \!\(\*SuperscriptBox[\(`ion`\),
        \(\(3\)\(+\)\)]\)"];
177     title = titleTemplate[<|"ion" -> ln|>];
178     parsedStates = ParseStates[eigensys, basis];
179     If[And[OddQ[numE],OptionValue["Remove Kramers"]],
180       parsedStates = parsedStates[[;; ;; 2]]
181     ];
182
183     stateLabels = #[[-1]] & /@ parsedStates;
184     simplerStateLabels = ((#[[2]] /. termSimplifier) <> ToString
        [#[[3]], InputForm]) & /@ parsedStates;
185
```

```
    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 ..."];
      AssignmentMatchFunc = Which[
        StringContainsQ[#[[1]], #[[2]]],
        "\[Checkmark]",
        True,
        "X"] &;
      assignmentMatches = AssignmentMatchFunc /@ Transpose[{
    carnallAssignments, simplerStateLabels[[;; Length[
    carnallAssignments]]]}];
      assignmentMatches = {{"\[Checkmark]",
        Count[assignmentMatches, "\[Checkmark]"]}, {"X",
        Count[assignmentMatches, "X"]}};
      labelComparison = (AssignmentMatchFunc /@ 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[[;;;;If[OddQ[numE],2,1]]],
          diffLabels[[;;;;If[OddQ[numE],2,1]]],
          Frame -> True,
          PlotRange -> All,
          ImageSize -> 1200,
          AspectRatio -> 1/3,
          Filling -> Axis,
          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 *)
```

```
279       nb = CreateDocument[
280       {
281         TextCell[Style[
282           DisplayForm[RowBox[{SuperscriptBox[host <> ":" <> ln, "3+"
    ], "(", SuperscriptBox["f", numE], ")"}]]
283           ], "Title", TextAlignment -> Center
284         ],
285         TextCell["Energy Diagram",
286           "Section",
287           TextAlignment -> Center
288         ],
289         TextCell[energyDiagram,
290           TextAlignment -> Center
291         ],
292         TextCell["Multiplet Assignments & Energy Levels",
293           "Section",
294           TextAlignment -> Center
295         ],
296         (* TextCell[diffHistogram, TextAlignment -> Center], *)
297         TextCell[diffPlot, "Output", TextAlignment -> Center],
298         TextCell[assignmentMatches, "Output", TextAlignment -> Center
    ],
299         TextCell[diffTable, "Output", TextAlignment -> Center],
300         TextCell["Truncated Eigenstates", "Section", TextAlignment ->
     Center],
301         TextCell["These are some of the resultant eigenstates which
    add up to at least a total probability of " <> ToString[
    eigenstateTruncationProbability] <> ".", "Text", TextAlignment ->
    Center],
302         TextCell[statesTable, "Output", TextAlignment -> Center]
303       },
304       WindowSelected -> True,
305       WindowTitle -> ln <> " in " <> "LaF3" <> appToFname,
306       WindowSize -> {1600, 800}];
307       If[OptionValue["SaveData"],
308       (
309         exportFname = FileNameJoin[{workDir,OptionValue["
    OutputDirectory"], ln <> " in " <> "LaF3" <> appToFname <> ".m"}];
310         SelectionMove[nb, After, Notebook];
311         NotebookWrite[nb, Cell["Reload Data", "Section",
    TextAlignment -> Center]];
312         NotebookWrite[nb,
313           Cell[(
314             "{rmsDifference, carnallEnergies, eigenEnergies, ln,
    carnallAssignments, simplerStateLabels, eigensys, basis,
    truncatedStates} = Import[FileNameJoin[{NotebookDirectory[],\"" <>
     StringSplit[exportFname,"/"][[-1]] <> "\"}]];"
315           ),"Input"
316           ]
317         ];
318         NotebookWrite[nb,
319           Cell[(
320             "Manipulate[First[MinimalBy[truncatedStates, Abs[First[#]
     - energy] &]], {energy,0}]"
321           ),"Input"]
```

```
322         ];
323         (* Move the cursor to the top of the notebook *)
324         SelectionMove[nb, Before, Notebook];
325         Export[exportFname,
326           {rmsDifference, carnallEnergies, eigenEnergies, ln,
    carnallAssignments, simplerStateLabels, eigensys, basis,
    truncatedStates}
327         ];
328         tinyexportFname = FileNameJoin[
329           {workDir, OptionValue["OutputDirectory"], ln <> " in " <> "
    LaF3" <> appToFname <> " - tiny.m"}
330         ];
331         tinyExport = <|"ln"->ln,
332                        "carnallEnergies"->carnallEnergies,
333                        "rmsDifference"-> rmsDifference,
334                        "eigenEnergies"-> eigenEnergies,
335                        "carnallAssignments"-> carnallAssignments,
336                        "simplerStateLabels" -> simplerStateLabels|>;
337         Export[tinyexportFname, tinyExport];
338       )
339       ];
340       If[OptionValue["NotebookSave"],
341         (
342           nbFname = FileNameJoin[{workDir,OptionValue["
    OutputDirectory"], ln <> " in " <> "LaF3" <> appToFname <> ".nb"
    }];
343           PrintFun[">> Saving notebook to ", nbFname, " ..."];
344           NotebookSave[nb, nbFname];
345         )
346       ];
347     )
348     ];
349
350     Return[{rmsDifference, carnallEnergies,
351             eigenEnergies, ln,
352             carnallAssignments, simplerStateLabels,
353             eigensys, basis,
354             truncatedStates}];
355   )
356 ];
357
358 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:
359     \[Psi]i:simple, (* main contribution to the wavefuction |i>*)
360     \[Psi]f:simple, (* main contribution to the wavefuction |j>*)
361     \[Psi]i:idx,    (* index of the wavefuction |i>*)
362     \[Psi]f:idx,    (* index of the wavefuction |j>*)
363     Ei/K,           (* energy of the initial state in K *)
364     Ef/K,           (* energy of the final state in K *)
365     \[Lambda]/nm,   (* transition wavelength in nm *)
366     \[CapitalDelta]\[Lambda]/nm, (* uncertainty in the transition
    wavelength in nm *)
367     \[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.
    - \"Print Function\" -> PrintTemporary or Print. The function
    used to print the progress of the calculation. Default is
    PrintTemporary.
    - \"Host\" -> \"LaF3\". The host material. Default is LaF3.
    - \"Wavelength Range\" -> {50,2000}. The range of wavelengths in
    nm for the Manipulate object in the created notebook. Default is
    {50,2000}.

The function returns an association containing the following keys:
    Line Strength, AMD, fMD, Radiative lifetimes, Transition Energies
    / K, Transition Wavelengths in nm.";
Options[MagneticDipoleTransitions] = {
        "Make Notebook" -> True,
        "Close Notebook" -> True,
        "Print Function" -> PrintTemporary,
        "Host" -> "LaF3",
        "Wavelength Range" -> {50,2000}};
MagneticDipoleTransitions[numE_Integer, OptionsPattern[]]:= (
  host           = OptionValue["Host"];
  \[Lambda]Range = OptionValue["Wavelength Range"];
  PrintFun       = OptionValue["Print Function"];
  {\[Lambda]min, \[Lambda]max} = OptionValue["Wavelength Range"];

  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"};
  ln        = {"Ce","Pr","Nd","Pm","Sm","Eu","Gd","Tb","Dy","Ho","Er"
    ,"Tm","Yb"}[[numE]];
  {rmsDifference,carnallEnergies,eigenEnergies,ln,
  carnallAssignments,simplerStateLabels,eigensys,basis,
    truncatedStates} = Import["./examples/"<>ln<>" in LaF3 - example.m
    "];

  (* Some of the above are not needed here *)
  Clear[truncatedStates];
```

```
Clear[basis];
Clear[rmsDifference];
Clear[carnallEnergies];
Clear[carnallAssignments];
If[OddQ[numE],
    eigenEnergies      = eigenEnergies[[;;;;2]];
    simplerStateLabels = simplerStateLabels[[;;;;2]];
    eigensys           = eigensys[[;;;;2]];
];
eigenEnergies = eigenEnergies - eigenEnergies[[1]];

magIon = <||>;
PrintFun["Calculating the magnetic dipole line strength array..."];
magIon["Line Strength"] = magIon;
MagDipLineStrength[eigensys, numE, "Reload MagOp" -> False, "Units"
   -> "SI"];

PrintFun["Calculating the M1 spontaneous transition rates ..."];
magIon["AMD"] = MagDipoleRates[eigensys, numE, "Units"->"SI","
  Lifetime"->False];
magIon["AMD"] = magIon["AMD"]/.{0.->Indeterminate};

PrintFun["Calculating the oscillator strength for transition from
  the ground state ..."];
magIon["fMD"]=GroundStateOscillatorStrength[eigensys, numE];

PrintFun["Calculating the natural radiative lifetims ..."];
magIon["Radiative lifetimes"]= 1/magIon["AMD"];

PrintFun["Calculating the transition energies in K ..."];
transitionEnergies=Outer[Subtract,First/@eigensys,First/@eigensys];
magIon["Transition Energies / K"]=ReplaceDiagonal[
  transitionEnergies,Indeterminate];

PrintFun["Calculating the transition wavelengths in nm ..."];
magIon["Transition Wavelengths in nm"] =10^7/magIon["Transition
  Energies / K"];

PrintFun["Estimating the uncertainties in \[Lambda]/nm assuming a 1
   K uncertainty in energies."];
(*Assuming an uncertainty of 1 K in both energies used to calculate
   the wavelength*)
\[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,
```

```
452         fromTo,
453         energyPairs,
454         magIon["Transition Wavelengths in nm"],
455         \[Lambda]uncertainty,
456         magIon["AMD"],
457         magIon["Radiative lifetimes"]
458     };
459     allTransitions = (Flatten/@Transpose[Flatten[#,1]&/@allTransitions
        ]);
460     allTransitions = Select[allTransitions, #[[3]]!=#[[4]]&];
461     allTransitions = Select[allTransitions, #[[10]]>0&];
462     allTransitions = Transpose[allTransitions];
463
464     (*round things up*)
465     PrintFun["Rounding wavelengths according to estimated uncertainties
          ..."];
466     {roundedWaves,roundedDeltas} = Transpose[MapThread[
        RoundValueWithUncertainty,{allTransitions[[7]],allTransitions
        [[8]]}]];
467     allTransitions[[7]]          = roundedWaves;
468     allTransitions[[8]]          = roundedDeltas;
469
470     PrintFun["Rounding lifetimes and transition rates to three
        significant figures ..."];
471     allTransitions[[9]]          = RoundToSignificantFigures[#,3]&/@(
        allTransitions[[9]]);
472     allTransitions[[10]]         = RoundToSignificantFigures[#,3]&/@(
        allTransitions[[10]]);
473     finalTable                   = Transpose[allTransitions];
474     finalTable                   = Prepend[finalTable,header];
475
476     (* tabular output *)
477     basename     = ln <> " in " <> host <> " - example - " <> "MD1 -
        tabular.zip";
478     exportFname  = FileNameJoin[{"./examples",basename}];
479     PrintFun["Exporting tabular data to "<>exportFname<>" ..."];
480     exportKey    = StringReplace[basename,".zip"->".m"];
481     Export[exportFname, <|exportKey->finalTable|>];
482
483     (* raw data output *)
484     basename    = ln <> " in " <> host <> " - example - " <> "MD1 - raw
        .zip";
485     rawexportFname = FileNameJoin[{"./examples",basename}];
486     PrintFun["Exporting raw data as an association to "<>exportFname<>"
          ..."];
487     rawexportKey   = StringReplace[basename,".zip"->".m"];
488     Export[rawexportFname, <|rawexportKey->magIon|>];
489
490     (* csv output *)
491     PrintFun["Formatting and exporting a CSV output..."];
492     csvOut = Table[
493         StringJoin[Riffle[ToString[#,CForm]&/@finalTable[[i]],","]],
494     {i,1,Length[finalTable]}
495     ];
496     csvOut       = StringJoin[Riffle[csvOut, "\n"]];
```

```
497  basename    = ln <> " in " <> host <> " - example - " <> "MD1.csv";
498  exportFname = FileNameJoin[{"./examples", basename}];
499  PrintFun["Exporting csv data to "<>exportFname<>" ..."];
500  Export[exportFname, csvOut, "Text"];
501
502  If[OptionValue["Make Notebook"],
503  (
504      PrintFun["Creating a notebook with a Manipulate to select a
       wavelength interval and a lifetime power of ten ..."];
505      finalTable     = Rest[finalTable];
506      finalTable     = SortBy[finalTable,#[[7]]&];
507      opticalTable   = Select[finalTable,\[Lambda]min<=#[[7]]<=\[
       Lambda]max&];
508      pows           = Sort[DeleteDuplicates[(MantissaExponent
       [#[[9]]][[2]]-1)&/@opticalTable]];
509
510      man            = Manipulate[
511      (
512          {\[Lambda]min,\[Lambda]max} = \[Lambda]int;
513          table = Select[opticalTable,And[(\[Lambda]min<=#[[7]]<=\[
       Lambda]max),
514                      (MantissaExponent[#[[9]]][[2]]-1)==log10\[Tau
       ]]&];
515          tab   = TableForm[table,TableHeadings->{None,header}];
516          Column[{{"\[Lambda]min="<>ToString[\[Lambda]min]<>" nm","\[
       Lambda]max="<>ToString[\[Lambda]max]<>" nm",log10\[Tau]},tab}]
517      ),
518      {{\[Lambda]int,\[Lambda]Range,"\[Lambda] interval"},
519          \[Lambda]Range[[1]],
520          \[Lambda]Range[[2]],
521          50,
522          ControlType->IntervalSlider
523      },
524      {{log10\[Tau],pows[[-1]]},
525          pows
526      },
527      TrackedSymbols  :> {\[Lambda]int,log10\[Tau]},
528      SaveDefinitions -> True
529      ];
530
531      nb = CreateDocument[{
532          TextCell[Style[DisplayForm[RowBox[{"Magnetic Dipole
       Transitions", "\n", SuperscriptBox[host<>":"<>ln,"3+"],"(",
       SuperscriptBox["f",numE],")"}]]],"Title",TextAlignment->Center],
533          (* TextCell["Magnetic Dipole Transition Lifetimes","Section
       ",TextAlignment->Center], *)
534          TextCell[man,"Output",TextAlignment->Center]
535      },
536      WindowSelected -> True,
537      WindowTitle    -> "MD1 - "<>ln<>" in "<>host,
538      WindowSize     -> {1600,800}
539      ];
540      SelectionMove[nb, After, Notebook];
541      NotebookWrite[nb, Cell["Reload Data", "Section", TextAlignment
       -> Center]];
```

```
542       NotebookWrite[nb, Cell[(
543             "magTransitions = Import[FileNameJoin[{NotebookDirectory
    []," <> StringSplit[rawexportFname,"/"][[-1]] <> "}],""<>
    rawexportKey<>"];"
544             ),"Input"]];
545       SelectionMove[nb, Before, Notebook];
546       nbFname = FileNameJoin[{workDir,"examples","MD1 - "<>ln<>" in "
    <>"LaF3"<>".nb"}];
547       PrintFun[">> Saving notebook to ",nbFname," ..."];
548       NotebookSave[nb, nbFname];
549       If[OptionValue["Close Notebook"],
550           NotebookClose[nb];
551       ];
552   )
553   ];
554
555   Return[magIon];
556 )
```

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).

[CFW65] W To Carnall, PR Fields, and BG Wybourne. "Spectral intensities of the trivalent lanthanides and actinides in solution. I. Pr3+, Nd3+, Er3+, Tm3+, and Yb3+". In: *The Journal of Chemical Physics* 42.11 (1965). Publisher: American Institute of Physics, pp. 3797–3806.

[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.

[DZ12] Christopher M. Dodson and Rashid Zia. "Magnetic dipole and electric quadrupole transitions in the trivalent lanthanide series: Calculated emission rates and oscillator strengths". en. In: *Physical Review B* 86.12 (Sept. 2012), p. 125102. ISSN: 1098-0121, 1550-235X. DOI: 10.1103/PhysRevB.86.125102. URL: https://link.aps.org/doi/10.1103/PhysRevB.86.125102 (visited on 07/02/2021).

[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 f^3". 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.

[Lin74] Ingvar Lindgren. "The Rayleigh-Schrodinger perturbation and the linked-diagram theorem for a multi-configurational model space". In: *Journal of Physics B: Atomic and Molecular Physics* 7.18 (1974). Publisher: IOP Publishing, p. 2441.

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

[Rac43] Giulio Racah. "Theory of Complex Spectra. III". en. In: *Physical Review* 63.9-10 (May 1943), pp. 367–382. ISSN: 0031-899X. DOI: 10.1103/PhysRev.63.367. URL: https://link.aps.org/doi/10.1103/PhysRev.63.367 (visited on 07/02/2021).

[Rud07] Zenonas Rudzikas. *Theoretical atomic spectroscopy*. 2007.

[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.

[TLJ99] Anne Thorne, Ulf Litzén, and Sveneric Johansson. *Spectrophysics: principles and applications*. Springer Science & Business Media, 1999.

[Tre52] R. E. Trees. "The L (L + 1) Correction to the Slater Formulas for the Energy Levels". en. In: *Physical Review* 85.2 (Jan. 1952), pp. 382–382. ISSN: 0031-899X. DOI: 10.1103/PhysRev.85.382. URL: https://link.aps.org/doi/10.1103/PhysRev.85.382 (visited on 01/18/2022).

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

[Wyb63] BG Wybourne. "Electrostatic Interactions in Complex Electron Configurations". In: *Journal of Mathematical Physics* 4.3 (1963). Publisher: American Institute of Physics, pp. 354–356.

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