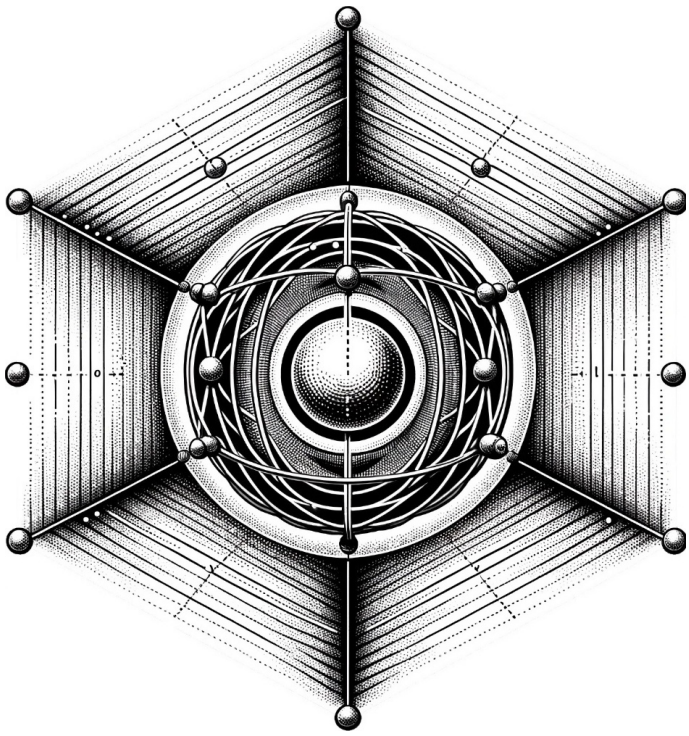

qlanth

version $|\alpha\rangle^{(2)}$

Juan David Lizarazo Ferro
& Christopher Dodson

Under the advisory of Dr. Rashid Zia

Contents

1 The $|LSJM_J\rangle$ Basis **4**

2 The coefficients of fractional parentage **8**

3 The JJ' block structure **9**

4 The effective Hamiltonian **12**
- 4.1 $\hat{\mathcal{H}}_{\mathrm{k}}$: kinetic energy . . . 14
- 4.2 $\hat{\mathcal{H}}_{\mathrm{e:sn}}$: the central field potential . . . 15
- 4.3 $\hat{\mathcal{H}}_{\mathrm{e:e}}$: e:e repulsion . . . 15
- 4.4 $\hat{\mathcal{H}}_{\mathrm{s:o}}$: spin-orbit . . . 18
- 4.5 $\hat{\mathcal{H}}_{\mathcal{SO}(3)}, \hat{\mathcal{H}}_{\mathrm{G_2}}, \hat{\mathcal{H}}_{\mathcal{SO}(7)}$: electrostatic configuration interaction . . . 19
- 4.6 $\hat{\mathcal{H}}_{\mathrm{s:s\text{-}s:oo}}$: spin-spin and spin-other-orbit . . . 20
- 4.7 $\hat{\mathcal{H}}_{\mathrm{ecs:o}}$: electrostatically-correlated-spin-orbit and a note about CI . . . 26
- 4.8 $\hat{\mathcal{H}}_3$: three-body effective operators . . . 36
- 4.9 $\hat{\mathcal{H}}_{\mathrm{cf}}$: crystal-field . . . 39
- 4.10 $\hat{\mu}$ and $\hat{\mathcal{H}}_{\mathrm{Z}}$: the magnetic dipole operator and the Zeeman term . . . 43
- 4.11 Going beyond f^7 . . . 45

5 Magnetic Dipole Transitions **46**

6 Accompanying notebooks **49**

7 Additional data **50**
- 7.1 Carnall et al data on Ln:LaF3 . . . 50
- 7.2 `sparsefn.py` . . . 51

8 Units **52**

9 Notation **53**

10 Definitions **54**

11 code **55**
- 11.1 qlanth.m . . . 55
- 11.2 qonstants.m . . . 138
- 11.3 qplotter.m . . . 139
- 11.4 misc.m . . . 145
- 11.5 qalculations.m . . . 154

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

This picture is one that developed and mostly matured in the second half of the last century by the efforts of Giulio Racah, Brian Judd, Hannah Crosswhite, Robert Cowan, Michael Reid, Bill Carnall, Clyde Morrison, Brian Wybourne, and Katherine Rajnak among others. The goal of this code is to provide a modern implementation of the methods that resulted from their work.

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

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

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

This document has 11 sections. In the first section the basis in which the Hamiltonian is evaluated is explained. In the second section a brief explanation of the coefficients of fractional parentage is given. Afther that the third section explains how the Hamiltonian is put together by first having calculated JJ' blocks.

The four section 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 7 list additional data included in **qlanth** . While section 8 has a brief comment on units.

Section 9 includes a brief of notation use throught this document and section 11 prints out 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 $\underline{\mathrm{f}}^n$ are obtained from tables that originate from the orginal work by Nielson and Koster [NK63]. In **qlanth** these are parsed from the file `B1F_ALL.TXT` which is part of the doctoral research of Dobromir Velkov [Vel00] in which he recomputed coefficients of fractional parentage under the advisory of Brian Judd.

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

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

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

$\underline{\mathrm{f}}^1$ (1 LS term)
${}^2F$

$\underline{\mathrm{f}}^2$ (7 LS terms)
${}^3P$, ${}^3F$, ${}^3H$, ${}^1S$, ${}^1D$, ${}^1G$, ${}^1I$

$\underline{\mathrm{f}}^3$ (17 LS terms)
${}^4S$, ${}^4D$, ${}^4F$, ${}^4G$, ${}^4I$, ${}^2P$, ${}^2D1$, ${}^2D2$, ${}^2F1$, ${}^2F2$, ${}^2G1$, ${}^2G2$, ${}^2H1$, ${}^2H2$, ${}^2I$, ${}^2K$, ${}^2L$

$\underline{\mathrm{f}}^4$ (47 LS terms)
${}^5S$, ${}^5D$, ${}^5F$, ${}^5G$, ${}^5I$, ${}^3P1$, ${}^3P2$, ${}^3P3$, ${}^3D1$, ${}^3D2$, ${}^3F1$, ${}^3F2$, ${}^3F3$, ${}^3F4$, ${}^3G1$, ${}^3G2$, ${}^3G3$, ${}^3H1$, ${}^3H2$, ${}^3H3$, ${}^3H4$, ${}^3I1$, ${}^3I2$, ${}^3K1$, ${}^3K2$, ${}^3L$, ${}^3M$, ${}^1S1$, ${}^1S2$, ${}^1D1$, ${}^1D2$, ${}^1D3$, ${}^1D4$, ${}^1F$, ${}^1G1$, ${}^1G2$, ${}^1G3$,

$^1G4$, $^1H1$, $^1H2$, $^1I1$, $^1I2$, $^1I3$, ^1K, $^1L1$, $^1L2$, ^1N

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

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

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

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

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

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

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

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

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

---

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

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

---

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

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

---

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

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

---

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

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

---

^{2}F

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

---

^{1}S

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

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

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

$(J=0)$ (2 kets)
$\|^3P,0,0\rangle$, $\|^1S,0,0\rangle$

$(J=1)$ (3 kets)
$\|^3P,1,-1\rangle$, $\|^3P,1,0\rangle$, $\|^3P,1,1\rangle$

$(J=2)$ (15 kets)
$\|^3P,2,-2\rangle$, $\|^3P,2,-1\rangle$, $\|^3P,2,0\rangle$, $\|^3P,2,1\rangle$, $\|^3P,2,2\rangle$, $\|^3F,2,-2\rangle$, $\|^3F,2,-1\rangle$, $\|^3F,2,0\rangle$, $\|^3F,2,1\rangle$, $\|^3F,2,2\rangle$, $\|^1D,2,-2\rangle$, $\|^1D,2,-1\rangle$, $\|^1D,2,0\rangle$, $\|^1D,2,1\rangle$, $\|^1D,2,2\rangle$

$(J=3)$ (7 kets)
$\|^3F,3,-3\rangle$, $\|^3F,3,-2\rangle$, $\|^3F,3,-1\rangle$, $\|^3F,3,0\rangle$, $\|^3F,3,1\rangle$, $\|^3F,3,2\rangle$, $\|^3F,3,3\rangle$

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

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

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

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

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

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

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

2 The coefficients of fractional parentage

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

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

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

Most importantly for us, in **qlanth** the coefficients of fractional parentage can be used to evaluate matrix elements of operators, such as in &qn-25, &qn-47, &qn-60, and &qn-37. These formulas realize a convenient calculational advantage: if one knows matrix elements in one configuration, then one can immediately calculate them in any other configuration.

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

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

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

3 The JJ' block structure

Now that we know how the bases are ordered, we can already understand the structure of how the final Hamiltonian matrix representation 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 $\{J, J'\}$ are created. The values being rectangular rank-2 arrays.

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

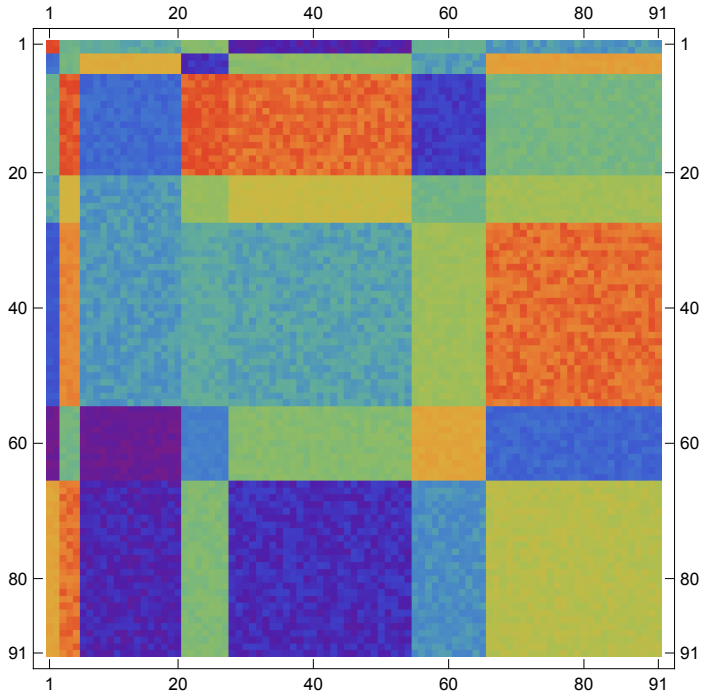

Figure 1: The JJ block structure for $\underline{\mathrm{f}}^2$

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]];
        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;";
Options[HamMatrixAssembly] = {"FilenameAppendix"->"","IncludeZeeman"->
    False};
HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
```

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

4 The effective Hamiltonian

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

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

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

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

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

These effects can be brought within the simplified description only through the help of perturbation theory. The task not the usual one of correcting for the energies/eigenvectors given an added perturbation, but rather to consider the effects of using a truncated Hilbert space due to a known interaction. For a detailed analysis of this see Rudzikas' [Rud07] book on theoretical atomic spectroscopy or this article by Lindgren [Lin74]. 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. These coefficients one could try to evaluate, and there are some that have trodden this road. Others simply label that complex expression with an unassuming symbol, and leave it as a parameter that one can hope to fit against experimental data. It is from CI that the parameters $\alpha, \beta, \gamma, P^{(k)}, T^{(k)}$ enter into the description.

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

When finding the matrix elements of the Hamiltonian defined by these terms, one also requires the specification of the basis in which the matrix elements will be computed. What we shall use

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

There is one way in which this term does influence the analysis, which is that for the lanthanide ions, this term certainly makes relativistic corrections a meanigful concern. In the *trivalent* lanthanide ions that are often considered, each electron in a *4f* has a kinetic energy of 10 eV which in the energy units we use here is about 10^6K and which corresponds to a speed of about 1% the speed of light.

4.2 $\hat{\mathcal{H}}_{\text{e:sn}}$: 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.

$$\hat{\mathcal{H}}_{\text{e:sn}} = -e^2 \sum_{i=1}^{Z} \frac{1}{r_i} + e^2 \underbrace{\sum_{i=1}^{n} \sum_{j=1}^{Z-n} \frac{1}{r_{ij}}}_{\substack{\text{Repulsion between valence} \\ \text{and inner shell} \\ \text{electrons}}} \approx \sum_{i=1}^{n} V_{\text{sn}}(r_i) \text{ (with Z = atomic No.)} \tag{17}$$

The precise form of $V_{\text{sn}}(r_i)$ 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 to the wavefunctions.

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

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

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

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

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

where

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

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

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

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

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

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

4.4 $\hat{\mathcal{H}}_{\text{s:o}}$: 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

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

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

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

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

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

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

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

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

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

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

```
ReducedV1k::usage = "ReducedV1k[n, l, SL, SpLp, k] gives the reduced
    matrix element of the spherical tensor operator V^(1k). See
    equation 2-101 in Wybourne 1965.";
ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
  {Vk1, S, L, Sp, Lp, Sb, Lb, spin, orbital, cfpSL, cfpSpLp,
  SLparents, SpLpparents, commonParents, prefactor},
  {spin, orbital} = {1/2, 3};
  {S, L}          = FindSL[SL];
```

```
  {Sp, Lp}       = FindSL[SpLp];
  cfpSL          = CFP[{numE, SL}];
  cfpSpLp        = CFP[{numE, SpLp}];
  SLparents      = First /@ Rest[cfpSL];
  SpLpparents    = First /@ Rest[cfpSpLp];
  commonParents = Intersection[SLparents, SpLpparents];
  Vk1 = Sum[(
      {Sb, Lb} = FindSL[\[Psi]b];
      Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
      CFPAssoc[{numE, SL, \[Psi]b}] *
      CFPAssoc[{numE, SpLp, \[Psi]b}] *
      SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
      SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
    ),
  {\[Psi]b, commonParents}
  ];
  prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp, Lp]
    ];
  Return[prefactor * Vk1];
  ]
```

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 very important contributions from configuration interaction. When the interaction with configurations ?? and ?? it was realized that the way the omission of these configurations in the single configuration description was to relax the previous restriction that $F^{(k)}$ should only have even values for k. Parallel to this Trees noticed an interesting fact which is that a fair amount of correction to the calculated spectrum of would benefit if one added to all of the LS energies a term quadratic in L. Soon after this it was acknowledged that the inclusion of odd $F^{(k)}$ was equivalent to adding three terms related to the Casimir operators of the groups $\mathcal{SO}(()3)$, G_2, and $\mathcal{SO}(()7)$ [Wyb63]. In addition to this, the configuration interaction analysis, also showed that the contributions from other configuration would also overlap with the already present even $F^{(k)}$.

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

$$\langle \underline{\ell}^n \alpha S L J M_J | \hat{\mathcal{H}}_{\mathcal{SO}(3)} | \underline{\ell}^n \alpha' S' L' J' M'_J \rangle = \alpha \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) L(L+1) \tag{26}$$

$$\langle \underline{\ell}^n U \alpha S L J M_J | \hat{\mathcal{H}}_{\mathrm{G}_2} | \underline{\ell}^n U \alpha' S' L' J' M'_J \rangle = \beta \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) \Lambda_{\mathrm{G}_2}(U) \tag{27}$$

$$\langle \underline{\ell}^n W \alpha S L J M_J | \hat{\mathcal{H}}_{\mathcal{SO}(7)} | \underline{\ell}^n W \alpha' S' L' J' M_J \rangle = \gamma \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) \Lambda_{\mathcal{SO}(7)}(W) \tag{28}$$

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

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

retardation of the electromagnetic field

$$\hat{\mathcal{H}}_B = -\frac{1}{2}e^2 \sum_{i>j} \left[(\alpha_i \cdot \alpha_j) \frac{1}{r_{ij}} + (\alpha_i \cdot \vec{r}_{ij})(\alpha_j \cdot \vec{r}_{ij}) \frac{1}{r_{ij}^3} \right]. \tag{29}$$

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

$$M^{(k)} := \frac{e^2\hbar^2}{8m^2c^2} \langle (nl)^2 | \frac{r_<^k}{r_>^{k+3}} | (nl)^2 \rangle \tag{30}$$

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

$$\hat{\mathcal{H}}_{s:s} = -2 \sum_{i \neq j} \sum_k M^{(k)} \sqrt{(k+1)(k+2)(2k+3)} \langle \underline{\ell} \| C^{(k)} \| \underline{\ell} \rangle \langle \underline{\ell} \| C^{(k+2)} \| \underline{\ell} \rangle \left\{ \hat{w}_i^{(1,k)} \hat{w}_j^{(1,k+2)} \right\}^{(2,2)0} \tag{31}$$

and the one for spin-other-orbit

$$\begin{aligned} \hat{\mathcal{H}}_{s:oo} = \sum_{i \neq j} \sum_k & \sqrt{(k+1)(2\underline{\ell}+k+2)(2\underline{\ell}-k)} \times \\ & \left[\left\{ \hat{w}_i^{(0,k+1)} \hat{w}_j^{(1,k)} \right\}^{(11)0} \left\{ M^{(k-1)} \langle \underline{\ell} \| C^{(k+1)} \| \underline{\ell} \rangle^2 + 2M^{(k)} \langle \underline{\ell} \| C^{(k)} \| \underline{\ell} \rangle^2 \right\} + \right. \\ & \left. \left\{ \hat{w}_i^{(0,k)} \hat{w}_j^{(1,k+1)} \right\}^{(11)0} \left\{ M^{(k)} \langle \underline{\ell} \| C^{(k)} \| \underline{\ell} \rangle^2 + 2M^{(k-1)} \langle \underline{\ell} \| C^{(k+1)} \| \underline{\ell} \rangle^2 \right\} \right]. \end{aligned} \tag{32}$$

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

$$\langle \underline{\ell} \| \hat{w}^{(\kappa,k)} \| \underline{\ell} \rangle = \sqrt{[\![\kappa]\!]\,[\![k]\!]} \tag{33}$$

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

$$\sqrt{5}\hat{\mathcal{T}}_0^{(22)} := \hat{\mathcal{H}}_{\text{s:s}} \tag{34}$$

$$-\sqrt{3}\hat{\mathcal{T}}_0^{(11)} := \hat{\mathcal{H}}_{\text{s:oo}} \tag{35}$$

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

$$\langle \gamma SLJ | \hat{\mathcal{H}} | \gamma' S' L' J' \rangle = \delta(J,J') \begin{Bmatrix} S' & L' & J \\ L & S & t \end{Bmatrix} \langle \gamma SL \| \hat{\mathcal{T}}^{(tt)} \| \gamma' S' L' \rangle. \tag{36}$$

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]

$$\langle \underline{\mathrm{f}}^n \psi \| \hat{\mathcal{T}}^{(tt)} \| \underline{\mathrm{f}}^n \psi' \rangle = \frac{n}{n-2} \sum_{\bar{\psi},\bar{\psi}'} (-1)^{\bar{S}+\bar{L}+\underline{s}+\underline{\ell}+S'+L'} \sqrt{[\![S]\!]\,[\![S']\!]\,[\![L]\!]\,[\![L']\!]} \times$$
$$\left(\psi\{|\bar{\psi}\right)\left(\psi'\{|\bar{\psi}'\right) \begin{Bmatrix} S & t & S' \\ \bar{S}' & \underline{s} & \bar{S} \end{Bmatrix} \begin{Bmatrix} L & t & L' \\ \bar{L}' & \underline{\ell} & \bar{L} \end{Bmatrix} \langle \underline{\mathrm{f}}^{n-1} \bar{\psi} \| \hat{\mathcal{T}}^{(tt)} \| \underline{\mathrm{f}}^{n-1}] \bar{\psi}' \rangle. \quad (37)$$

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-36 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 $\underline{\mathrm{f}}^2$ as provided in table II of [JCC68].

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

```
  T22Table = <||>;
  startTime = Now;
  numiter = 1;
  Do[
    (
      numiter+= 1;
      T22Table[{numE, SL, SpLp}] = Which[
        numE==1,
        0,
        numE==2,
        SimplifyFun[ReducedT22inf2[SL, SpLp]],
        True,
        SimplifyFun[ReducedT22infn[numE,  SL, SpLp]]
      ];
    ),
  {numE, 1, nmax},
  {SL,   AllowedNKSLTerms[numE]},
  {SpLp, AllowedNKSLTerms[numE]}
  ];
  If[And[OptionValue["Progress"],frontEndAvailable],
    NotebookDelete[progBar]
  ];
  If[OptionValue["Export"],
    (
      fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
      Export[fname, T22Table];
    )
  ];
  Return[T22Table];
);
```

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

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

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

```
Reducedt11inf2::usage="Reducedt11inf2[SL, SpLp] returns the reduced
    matrix element in f^2 of the double tensor operator t11 for the
    corresponding given terms {SL, SpLp}.
Values given here are those from Table VII of \"Judd, BR, HM Crosswhite
    , and Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
     Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
";
Reducedt11inf2[SL_, SpLp_]:= Module[
  {t11inf2},
  t11inf2 = <|
    {"1S", "3P"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
    {"3P", "3P"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
```

```
    {"3P", "1D"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
    {"1D", "3F"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
    {"3F", "3F"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
    {"3F", "1G"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
    {"1G", "3H"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
    {"3H", "3H"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
    {"3H", "1I"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
    |>;
  Which[
    MemberQ[Keys[t11inf2],{SL,SpLp}],
      Return[t11inf2[{SL,SpLp}]],
    MemberQ[Keys[t11inf2],{SpLp,SL}],
      Return[t11inf2[{SpLp,SL}]],
    True,
      Return[0]
  ]
]
```

4.7 $\hat{\mathcal{H}}_{\mathbf{ecs:o}}$: electrostatically-correlated-spin-orbit and a note about CI

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

$$\hat{\mathcal{H}}_{\text{ci}} = -\sum_{\chi}\sum_{i}\frac{1}{E_\chi}\xi(r_i)\left(\hat{\underline{s}}_i\cdot\hat{\underline{\ell}}_i\right)|\chi\rangle\langle\chi|\hat{\mathfrak{C}} - \frac{1}{E_\chi}\hat{\mathfrak{C}}|\chi\rangle\langle\chi|\xi(r_i)\left(\hat{\underline{s}}_i\cdot\hat{\underline{\ell}}_i\right) \tag{38}$$

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

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

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

$$\hat{\mathcal{H}}_{\text{ci}} = -\sqrt{3}\,\hat{t}_0^{(11)} \tag{39}$$

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 is also necessary

$$P^{(k)} = 6\sum_{f'}\frac{\zeta_{ff'}}{E_{ff'}}R^{(k)}(ff,ff') \text{ for } k=0,2,4,6. \tag{40}$$

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

$$\zeta_{ff'} \coloneqq \langle f|\xi(r)|f'\rangle \tag{41}$$

$$R^{(k)}(ff, ff') \coloneqq e^2\langle f_1 f_2|\frac{r_<^k}{r_>^{k+1}}|f_1 f_2'\rangle. \tag{42}$$

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

$$\langle\psi\|\hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)}\|\psi'\rangle = \sum a_i\langle\psi\|\hat{z}_i\|\psi'\rangle. \tag{43}$$

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:

$$\hat{\mathcal{H}}_{\text{s:oo}} + \hat{\mathcal{H}}_{\text{ecs:o}} = \hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)} - \frac{1}{6}a_{13}\hat{z}_{13} \tag{44}$$

where

$$a_{13} = -33M^{(0)} + 3M^{(2)} + 15/11M^{(4)} - 6P^{(0)} + 3/2(35P^{(2)} + 77P^{(4)} + 143P^{(6)}) \tag{45}$$

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 pre-computed 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 use to include or exclude the Chen
   deltas from the calculation. The default is to exclude them.";
Options[MagneticInteractions] = {"ChenDeltas" -> False};
```

```
MagneticInteractions[{numE_, SLJ_, SLJp_, J_}, OptionsPattern[]] :=
  (
    key = {numE, SLJ, SLJp, J};
    ss = \[Sigma]SS * SpinSpinTable[key];
    sooandecso = SOOandECSOTable[key];
    total = ss + sooandecso;
    total = SimplifyFun[total];
    If[
      Not[OptionValue["ChenDeltas"]],
      Return[total]
    ];
    (* In the type A errors the wrong values are different *)
    If[MemberQ[Keys[chenDeltas["A"]], {numE, SLJ, SLJp}],
      (
        {S, L} = FindSL[SLJ];
        {Sp, Lp} = FindSL[SLJp];
        phase   = Phaser[Sp + L + J];
        Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
        Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
        {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ,
    SLJp}]["wrong"];
        total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
        total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
        total  = total /. Prescaling;
        total  = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
      )
    ];
    (* In the type B errors the wrong values are zeros all around *)
    If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
      (
        {S, L} = FindSL[SLJ];
        {Sp, Lp} = FindSL[SLJp];
        phase = Phaser[Sp + L + J];
        Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
        Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
        {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
        total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
        total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
        total  = total /. Prescaling;
        total  = wChErrB * total + (1 - wChErrB) * (ss + sooandecso)
      )
    ];
    Return[total];
  )
```

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

```
(
  SpinOrbitTable =
    Table[
      {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
    {numE, 1, nmax},
    {J, MinJ[numE], MaxJ[numE]},
    {SL,   Map[First, AllowedNKSLforJTerms[numE, J]]},
    {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
    ];
  SpinOrbitTable = Association[SpinOrbitTable];
```

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

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

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

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

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

The association `T22Table` is computed by the function `GenerateT22Table`. This function populates `T22Table` with keys of the form $\{n, SL, S'L'\}$. It does this by using the function `ReducedT22inf2` in the base case of $\underline{f}^2$, and `ReducedT22infn` for configurations above $\underline{f}^2$. When `ReducedT22infn` is called the sum in &qn-37 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-44 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-37 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 $\hat{\mathscr{H}}_{\lambda}$: three-body effective operators

Three body model[1] interactions arise in the lanthanides due to configuration interaction between configurations $(4\underline{f})^n$ and $(4\underline{f})^{n\pm1}(n'\underline{\ell}')^{\mp1}$. As such they describe the effects of configuration interac-

[1] *model interactions* are interactions that appear because of the simplifications in our models, they are not fundamental, but originate from them.

tion of single-electron excitations from the ground configuration to excited ones.

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

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

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

$$\hat{\mathscr{H}}_{\curlywedge} = T'^{(2)}\hat{t}'_2 + \sum_{\substack{k=2,3,4,6,7,8,\\ 11,12,14,15,\\ 16,17,18,19}} T^{(k)}\hat{t}_k. \tag{46}$$

Where $\hat{t}'_2$ is a fifteenth legacy operator that is needed to reproduce old results from literature. The omission of some indices has to do with the fact that they way in which these are defined, lead to some of them being two-body operators, and as such they are excluded from the sum.

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

$$\langle \underline{f}^n\psi|\hat{t}_i|\underline{f}^n\psi'\rangle = \delta(J,J')\delta(M_J,M'_J)\frac{n}{n-3}\sum_{\bar{\psi}\bar{\psi}'}\left(\psi\left\{\left|\bar{\psi}\right.\right)\left(\psi'\left\{\left|\bar{\psi}'\right.\right)\langle \underline{f}^{n-1}\bar{\psi}|\hat{t}_i|\underline{f}^{n-1}\bar{\psi}'\rangle. \right. \tag{47}$$

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

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

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

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 avoid having to explicitly store matrix elements beyond 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}
  ];
```

```
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]);
```

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 reponsible 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:

$$V(r_i, \theta_i, \phi_i) = \sum_{k=1}^{\infty} \sum_{q=-k}^{k} \mathcal{A}_q^{(k)} r^k C_q^{(k)}(\theta_i, \phi_i) \tag{48}$$

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

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

$$\hat{\mathcal{H}}_{\mathrm{cf}}(\vec{r}) = \sum_{i=1}^{n} \sum_{k=0}^{\infty} \sum_{q=-k}^{k} \mathcal{A}_q^{(k)} r_i^k C_q^{(k)}(\theta_i, \phi_i). \tag{49}$$

And if we average the radial coordinate,

$$\hat{\mathscr{H}}_{\rm cf} = \sum_{i=1}^{n}\sum_{k=1}^{\infty}\sum_{q=-k}^{k} \mathcal{B}_q^{(k)} C_q^{(k)}(i) \tag{50}$$

where the radial average is included as

$$\mathcal{B}_q^{(k)} := \mathcal{A}_q^{(k)} \langle r^k \rangle. \tag{51}$$

$\mathcal{B}_q^{(k)}$ may be complex in general. However, since the sum in &eqn-49 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 behaviour of $C_q^{(k)}$ under complex conjugation is considered it is seen that

$$\mathcal{B}_q^{(k)} = (-1)^q \mathcal{B}_{-q}^{(k)*}. \tag{52}$$

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 imaginagy 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

$$\hat{\mathscr{H}}_{\rm cf}(\vec{r}) = \sum_{i=1}^{n}\sum_{k=0}^{\infty}\sum_{q=0}^{k} B_q^{(k)} \left(C_q^{(k)} + (-1)^q C_{-q}^{(k)} \right) + iS_q^{(k)} \left(C_q^{(k)} - (-1)^q C_{-q}^{(k)} \right). \tag{53}$$

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):

$$\langle \ell \| \hat{u}^{(k)} \| \ell' \rangle = 1. \tag{54}$$

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

$$\hat{U}^{(k)} = \sum_{i}^{n} \hat{u}_i^{(k)}. \tag{55}$$

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

$$C_q^{(k)} = \langle \underline{\ell} \| C^{(k)} \| \underline{\ell}' \rangle \hat{u}_q^{(k)} = (-1)^{\underline{\ell}} \sqrt{[\![\underline{\ell}]\!]\,[\![\underline{\ell}']\!]} \begin{pmatrix} \underline{\ell} & k & \underline{\ell}' \\ 0 & 0 & 0 \end{pmatrix} \hat{u}_q^{(k)}. \tag{56}$$

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

$$\overbrace{\langle \underline{\ell}^n \alpha SLJM_J | \hat{\mathscr{H}}_{\rm cf} | \underline{\ell}^n \alpha' SL'J'M_{J'} \rangle}^{\text{Wybourne eqn. 6-3}} = \sum_{k=1}^{\infty}\sum_{q=-k}^{k} \mathcal{B}_q^{(k)} \langle \underline{\ell}^n \alpha SLJM_J | \hat{U}_q^{(k)} | \underline{\ell}^n \alpha' SL'J'M_{J'} \rangle \langle \underline{\ell} \| \hat{C}^{(k)} \| \underline{\ell} \rangle \tag{57}$$

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

$$\overbrace{\langle \underline{\ell}^n \alpha SLJM_J | \hat{U}_q^{(k)} | \underline{\ell}^n \alpha' S'L'J'M_{J'} \rangle}^{\text{Wybourne eqn. 6-4}} = (-1)^{J-M_J} \begin{pmatrix} J & k & J' \\ -M_J & q & M_{J'} \end{pmatrix} \langle \underline{\ell}^n \alpha SLJ \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S'L' \rangle \tag{58}$$

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

$$\overbrace{\langle \underline{\ell}^n \alpha SLJ \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S'L' \rangle}^{\text{Wybourne eqn. 6-5}} = (-1)^{S+L+J'+k} \sqrt{[\![J]\!]\,[\![J']\!]} \times \begin{Bmatrix} J & J' & k \\ L' & L & S \end{Bmatrix} \langle \underline{\ell}^n \alpha SL \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S'L' \rangle. \tag{59}$$

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

$$\overbrace{\langle \underline{\ell}^n \alpha SL \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S'L' \rangle}^{\text{Cowan eqn. 11.53}} = \delta(S,S') n (-1)^{\underline{\ell}+L+k} \sqrt{[\![L]\!]\,[\![L']\!]} \times \sum_{\bar{\alpha}\bar{L}\bar{S}} (-1)^{\bar{L}} \begin{Bmatrix} \underline{\ell} & k & \underline{\ell} \\ L & \bar{L} & L' \end{Bmatrix} \left(\underline{\ell}^n \alpha LS \left\{ \underline{\ell}^{n-1} \bar{\alpha}\bar{L}\bar{S} \right) \left(\underline{\ell}^{n-1} \bar{\alpha}\bar{L}\bar{S} \right\} \underline{\ell}^n \alpha' L'S' \right). \tag{60}$$

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:

$$\hat{\mathscr{H}}_{\mathrm{cf}}(\vec{r}) = \sum_{i=1}^{n} \sum_{k=2,4,6} \sum_{q=0}^{k} B_q^{(k)} \left(C_q^{(k)} + (-1)^q C_{-q}^{(k)} \right) + i S_q^{(k)} \left(C_q^{(k)} - (-1)^q C_{-q}^{(k)} \right). \tag{61}$$

The above equations are implemented in **qlanth** by the function `CrystalField`. This function puts together the symbolic sum in &qn-57 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 atomic units, the operator associated with the magnetic dipole is

$$\hat{\mu} = -\mu_{\mathrm{B}} \left(\hat{L} + g_s \hat{S}\right)^{(1)}, \text{with } \mu_{\mathrm{B}} = 1/2. \tag{62}$$

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]

$$\langle \alpha LSJ \| \left(\hat{L} + g_s \hat{S}\right)^{(1)} \| \alpha' L'S'J' \rangle = \delta(\alpha LSJ, \alpha' L'S'J') \sqrt{J(J+1)(2J+1)} + \\ \delta(\alpha LS, \alpha' L'S') (-1)^{L+S+J+1} \sqrt{[\![J]\!]\,[\![J]\!]} \begin{Bmatrix} L & S & J \\ 1 & J' & S \end{Bmatrix} \tag{63}$$

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

through Wigner-Eckart

$$\langle \alpha LSJM_J | \left(\hat{L} + g_s \hat{S} \right)^{(1)}_q | \alpha' L' S' J' M_{J'} \rangle = (-1)^{J-M_J} \begin{pmatrix} J & 1 & J' \\ -M_J & q & M'_J \end{pmatrix} \langle \alpha LSJ \| \left(\hat{L} + g_s \hat{S} \right)^{(1)} \| \alpha' L' S' J' \rangle \quad (64)$$

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

$$\hat{\mu}_x = \frac{\hat{\mu}^{(1)}_{-1} - \hat{\mu}^{(1)}_{+1}}{\sqrt{2}} \tag{65}$$

$$\hat{\mu}_y = i\frac{\hat{\mu}^{(1)}_{-1} + \hat{\mu}^{(1)}_{+1}}{\sqrt{2}} \tag{66}$$

$$\hat{\mu}_z = \hat{\mu}^{(1)}_0 \tag{67}$$

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

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

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 $\underline{\mathrm{f}}^n$ and $\underline{\mathrm{f}}^{14-n}$. This is enforced when the function `HamMatrixAssembly` is called. In there `HoleElectronConjugation` is the function responsible for enforcing a global sign flip for the following operators (or equivalently to their accompanying coefficients):

$$\zeta, T^{(2)}, T^{(3)}, T^{(4)}, T^{(6)}, T^{(7)}, T^{(8)}, B_q^{(k)} \tag{68}$$

```
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])

$$\langle\nu|\hat{\mathcal{S}}|\nu'\rangle := |\langle\nu|\hat{\mu}|\nu'\rangle|^2 = |\langle\nu|\hat{\mu}_x|\nu'\rangle|^2 + |\langle\nu|\hat{\mu}_y|\nu'\rangle|^2 + |\langle\nu|\hat{\mu}_z|\nu'\rangle|^2 \tag{69}$$

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 $|\nu\rangle$ and $|\nu'\rangle$.

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

To this end are the functions MagDipLineStrength, MagDipoleRates, and GroundStateOscillatorStrength.

```
MagDipLineStrength::usage="MagDipLineStrength[theEigensys, numE] takes
    the eigensystem of an ion and the number numE of f-electrons that
    correspond to it and it calculates the line strength array Stot.
```

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

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

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

6 Accompanying notebooks

qlanth is accompanied by the following auxiliary Mathematica notebooks:

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

7 Additional data

7.1 Carnall et al data on Ln:LaF3

The study of Carnall et al [Car+89] on lanthanum fluoride was a systematic review of trivalent lanthanide ions in LaF3. 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 accesed 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 LaF3.

```
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] = {
```

```
    "Source"->"Carnall",
    "Free Ion"->False,
    "gs"->2.002319304386
    };
LoadParameters[Ln_String, OptionsPattern[]]:=
  Module[{source, params},
  (
    source = OptionValue["Source"];
    params = Which[source=="Carnall",
            (Association[Carnall["data"][Ln]])
            ];
    (*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;
    Return[params];
  )
];
```

7.2 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-68 and setting t2Switch to 0. For configurations at or below $\underline{f}^7$ it is necessary to set t2Switch to 1.

8 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[];
```

9 Notation

orbital angular momentum operator of a single electron

$$\hat{l} \tag{70}$$

total orbital angular momentum operator

$$\hat{L} \tag{71}$$

spin angular momentum operator of a single electron

$$\hat{s} \tag{72}$$

total spin angular momentum operator

$$\hat{S} \tag{73}$$

Shorthand for all other quantum numbers

$$\Lambda \tag{74}$$

orbital angular momentum number

$$\underline{\ell} \tag{75}$$

spinning angular momentum number

$$\underline{s} \tag{76}$$

Coulomb non-central potential

$$\hat{\mathfrak{C}} \tag{77}$$

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

$$\langle \Lambda LS \| \hat{O} \| \Lambda' L' S' \rangle \tag{78}$$

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

$$\langle \Lambda LSJ \| \hat{O} \| \Lambda' L' S' J' \rangle \tag{79}$$

Spectroscopic term αLS in Russel-Saunders notation

$$^{2S+1}\alpha L \equiv |\alpha LS\rangle \tag{80}$$

spherical tensor operator of rank k

$$\hat{X}^{(k)} \tag{81}$$

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

$$\hat{X}^{(k)}_q \tag{82}$$

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

$$\left(\underline{\ell}^{n-1} \alpha' L' S' \middle\} \underline{\ell}^n \alpha LS \right) \tag{83}$$

10 Definitions

$$\overbrace{[\![x]\!]}^{\text{two plus one}} \coloneqq 2x+1 \tag{84}$$

$$\overbrace{\hat{u}^{(k)}}^{\text{irreducible unit tensor operator of rank k}} \tag{85}$$

$$\overbrace{\hat{U}^{(k)} \coloneqq \sum_{i=1}^{n} \hat{u}^{(k)}}^{\text{symmetric unit tensor operator for n equivalent electrons}} \tag{86}$$

$$\overbrace{\mathcal{C}_q^{(k)} \coloneqq \sqrt{\frac{4\pi}{2k+1}}\mathrm{Y}_q^{(k)}}^{\text{Renormalized spherical harmonics}} \tag{87}$$

$$\overbrace{\triangle(j_1,j_2,j_3) \coloneqq \begin{cases} 1 & \text{if } j_1 = (j_2+j_3),(j_2+j_3-1),\ldots,|j_2-j_3| \\ 0 & \text{otherwise} \end{cases}}^{\text{Triangle ``delta'' between } j_1,j_2,j_3} \tag{88}$$

11 code

11.1 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.

+ 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.


------------------------------------------------------------------ *)

BeginPackage["qlanth`"];
Needs["qonstants`"];
Needs["qplotter`"];
```

```
Needs["misc`"];

paramAtlas = "
E0: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
E1: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
E2: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
E3: linear combination of F_k, see eqn. (2-80) in Wybourne 1965

ζ: spin-orbit strength parameter.

F0: Direct Slater integral F^0, produces an overall shift of all energy
    levels.
F2: Direct Slater integral F^2
F4: Direct Slater integral F^4, possibly constrained by ratio to F^2
F6: Direct Slater integral F^6, possibly constrained by ratio to F^2

M0: 0th Marvin integral
M2: 2nd Marvin integral
M4: 4th Marvin integral
\[Sigma]SS: spin-spin override, if 0 spin-spin is omitted, if 1 then
    spin-spin is included

T2:  three-body effective operator parameter T^2 (non-orthogonal)
T2p: three-body effective operator parameter T^2' (orthogonalized T2)
T3:  three-body effective operator parameter T^3
T4:  three-body effective operator parameter T^4
T6:  three-body effective operator parameter T^6
T7:  three-body effective operator parameter T^7
T8:  three-body effective operator parameter T^8

T11:  three-body effective operator parameter T^11
T11p: three-body effective operator parameter T^11'
T12:  three-body effective operator parameter T^12
T14:  three-body effective operator parameter T^14
T15:  three-body effective operator parameter T^15
T16:  three-body effective operator parameter T^16
T17:  three-body effective operator parameter T^17
T18:  three-body effective operator parameter T^18
T19:  three-body effective operator parameter T^19

P0: 0th parameter for the two-body electrostatically correlated spin-
    orbit interaction
P2: 2nd parameter for the two-body electrostatically correlated spin-
    orbit interaction
P4: 4th parameter for the two-body electrostatically correlated spin-
    orbit interaction
P6: 6th parameter for the two-body electrostatically correlated spin-
    orbit interaction

gs: electronic gyromagnetic ratio

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

```
γ: Trees' parameter γ describing configuration interaction via the
    Casimir operator of SO(7)

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

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

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

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

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

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

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

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

```
  paramUsage = StringTrim[paramUsage];
  expressionString = usageTemplate[<|"paramSymbol" -> paramString, "
    paramUsage" -> paramUsage|>];
  ToExpression[usageTemplate[<|"paramSymbol" -> paramString,
     "paramUsage" -> paramUsage|>]]
),
{paramLine, paramLines}
];

(* Parameter families*)
cfSymbols = {B02, B04, B06, B12, B14, B16, B22, B24, B26, B34, B36,
   B44, B46, B56, B66, S12, S14, S16, S22, S24, S26, S34, S36, S44,
   S46, S56, S66};

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

AllowedJ;
AllowedMforJ;
AllowedNKSLJMforJMTerms;
AllowedNKSLJMforJTerms;

AllowedNKSLJTerms;
AllowedNKSLTerms;
AllowedNKSLforJTerms;
AllowedSLJMTerms;
AllowedSLJTerms;

AllowedSLTerms;
BasisLSJMJ;
Bqk;
CFP;
CFPAssoc;

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

Cqk;
CrystalField;
Dk;
ElectrostaticConfigInteraction;
Electrostatic;

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

```
fsupk;

FindNKLSTerm;
FindSL;

FtoE;
GG2U;
GSO7W;
GenerateCFP;
GenerateCFPAssoc;

GenerateCFPTable;
GenerateCrystalFieldTable;
GenerateElectrostaticTable;
GenerateReducedUkTable;
GenerateReducedV1kTable;

GenerateSOOandECSOLSTable;
GenerateSOOandECSOTable;
GenerateSpinOrbitTable;
GenerateSpinSpinTable;
GenerateT22Table;

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

HamiltonianMatrixPlot;
HoleElectronConjugation;
IonSolver;
ImportMZip;
JJBlockMatrix;
JJBlockMagDip;
JJBlockMatrixFileName;

JJBlockMatrixTable;
LabeledGrid;
LoadAll;
LoadCFP;
LoadCarnall;

LoadChenDeltas;
LoadElectrostatic;
LoadGuillotParameters;
LoadParameters;
LoadSOOandECSO;

LoadSOOandECSOLS;
LoadSpinOrbit;
LoadSpinSpin;
LoadSymbolicHamiltonians;
```

```
LoadT11;

LoadT22;
LoadTermLabels;
LoadThreeBody;
LoadUk;
LoadV1k;

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

ParamPad;
ParseStates;
ParseStatesByNumBasisVecs;
ParseStatesByProbabilitySum;
ParseTermLabels;

Phaser;
PrettySaunders;
PrettySaundersSLJ;
PrettySaundersSLJmJ;
PrintL;

PrintSLJ;
PrintSLJM;
ReducedSOOandECSOinf2;
ReducedSOOandECSOinfn;
ReducedT11inf2;

ReducedT22inf2;
ReducedUk;
ReducedUkTable;
ReducedV1kTable;
Reducedt11inf2;

ReplaceInSparseArray;
SimplerSymbolicHamMatrix;
SOOandECSO;
SOOandECSOTable;
Seniority;

ShiftedLevels;
SixJay;
SpinOrbit;
SpinSpin;
SpinSpinTable;

Sqk;
SquarePrimeToNormal;
ReducedT22infn;
```

```
TPO;

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

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

fnTermLabels;
moduleDir;
symbolicHamiltonians;

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

Begin["`Private`"]

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

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

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

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

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

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

  SquarePrimeToNormal::usage = "Given a list with the parts
    corresponding to the squared prime representation of a number, this
     function parses the result into standard notation.";
  SquarePrimeToNormal[squarePrime_] :=
  (
    radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {idx1,
    2, Length[squarePrime]}];
```

```
415     radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" -> 13};
416     val = squarePrime[[1]] * Sqrt[radical];
417     Return[val];
418   );
419
420   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.
421   The function additionally takes an option \"Print\" that if set to
        True, will print the symbols that were not present in the given
        association.";
422   Options[ParamPad] = {"Print" -> True}
423   ParamPad[params_, OptionsPattern[]] := (
424     notPresentSymbols = Complement[paramSymbols, Keys[params]];
425     If[OptionValue["Print"],
426       Print["Following symbols were not given and are being set to 0: "
      ,
427       notPresentSymbols]
428     ];
429     newParams = Transpose[{paramSymbols, ConstantArray[0, Length[
      paramSymbols]]}];
430     newParams = (#[[1]] -> #[[2]]) & /@ newParams;
431     newParams = Association[newParams];
432     newParams = Join[newParams, params];
433     Return[newParams];
434     )
435
436   (* ############################################################ *)
437   (* ###################### Racah Algebra ###################### *)
438
439   ReducedUk::usage = "ReducedUk[n, l, SL, SpLp, k] gives the reduced
        matrix element of the symmetric unit tensor operator U^(k). See
        equation 11.53 in TASS.";
440   ReducedUk[numE_, l_, SL_, SpLp_, k_] :=
441     Module[{spin, orbital, Uk,
442       S, L, Sp, Lp, Sb, Lb,
443       parentSL, cfpSL, cfpSpLp, Ukval, SLparents, SLpparents,
      commonParents, phase},
444       {spin, orbital} = {1/2, 3};
445       {S, L}          = FindSL[SL];
446       {Sp, Lp}        = FindSL[SpLp];
447       If[Not[S == Sp],
448         Return[0]
449       ];
450       cfpSL      = CFP[{numE, SL}];
451       cfpSpLp    = CFP[{numE, SpLp}];
452       SLparents  = First /@ Rest[cfpSL];
453       SLpparents = First /@ Rest[cfpSpLp];
454       commonParents = Intersection[SLparents, SLpparents];
455       Uk = Sum[(
456         {Sb, Lb} = FindSL[\[Psi]b];
457         Phaser[Lb] *
458           CFPAssoc[{numE, SL, \[Psi]b}] *
```

```
459             CFPAssoc[{numE, SpLp, \[Psi]b}] *
460             SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
461         ),
462         {\[Psi]b, commonParents}
463         ];
464         phase     = Phaser[orbital + L + k];
465         prefactor = numE * phase * Sqrt[TPO[L,Lp]];
466         Ukval     = prefactor*Uk;
467         Return[Ukval];
468     ]
469
470     Ck::usage = "Diagonal reduced matrix element <l||C^(k)||l> where the
      Subscript[C, q]^(k) are reduced spherical harmonics. See equation
      11.23 in TASS with l=l'.";
471     Ck[orbital_, k_] := (-1)^orbital * TPO[orbital] * ThreeJay[{orbital,
      0}, {k, 0}, {orbital, 0}]
472
473     SixJay::usage = "SixJay[{j1, j2, j3}, {j4, j5, j6}] provides the
      value for SixJSymbol[{j1, j2, j3}, {j4, j5, j6}] with memorization
      of computed values.";
474     SixJay[{j1_, j2_, j3_}, {j4_, j5_, j6_}] := (
475       sixJayval =
476         Which[
477         Not[TriangleAndSumCondition[j1, j2, j3]],
478         0,
479         Not[TriangleAndSumCondition[j1, j5, j6]],
480         0,
481         Not[TriangleAndSumCondition[j4, j2, j6]],
482         0,
483         Not[TriangleAndSumCondition[j4, j5, j3]],
484         0,
485         True,
486         SixJSymbol[{j1, j2, j3}, {j4, j5, j6}]];
487       SixJay[{j1, j2, j3}, {j4, j5, j6}] = sixJayval);
488
489     ThreeJay::usage = "ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] gives the
      value of the Wigner 3j-symbol and memorizes the computed value.";
490     ThreeJay[{j1_, m1_}, {j2_, m2_}, {j3_, m3_}] := (
491      threejval = Which[
492        Not[(m1 + m2 + m3) == 0],
493        0,
494        Not[TriangleCondition[j1,j2,j3]],
495        0,
496        True,
497        ThreeJSymbol[{j1, m1}, {j2, m2}, {j3, m3}]
498        ];
499      ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] = threejval);
500
501     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.";
502     ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
503       {Vk1, S, L, Sp, Lp, Sb, Lb, spin, orbital, cfpSL, cfpSpLp,
504       SLparents, SpLpparents, commonParents, prefactor},
505       {spin, orbital} = {1/2, 3};
```

```
    {S, L}         = FindSL[SL];
    {Sp, Lp}       = FindSL[SpLp];
    cfpSL          = CFP[{numE, SL}];
    cfpSpLp        = CFP[{numE, SpLp}];
    SLparents      = First /@ Rest[cfpSL];
    SpLpparents    = First /@ Rest[cfpSpLp];
    commonParents = Intersection[SLparents, SpLpparents];
    Vk1 = Sum[(
        {Sb, Lb} = FindSL[\[Psi]b];
        Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
        CFPAssoc[{numE, SL, \[Psi]b}] *
        CFPAssoc[{numE, SpLp, \[Psi]b}] *
        SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
        SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
      ),
    {\[Psi]b, commonParents}
    ];
    prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp, Lp]
    ];
    Return[prefactor * Vk1];
    ]

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

```
      NotebookDelete[progBar]
    ];
    If[OptionValue["Export"],
      (
        Print["Exporting to file " <> ToString[ReducedUkTableFname]];
        Export[ReducedUkTableFname, ReducedUkTable];
      )
    ];
    Return[ReducedUkTable];
  )

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

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

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

fsubk::usage = "Slater integral f_k. See equation 12.17 in TASS.";
fsubk[numE_, orbital_, NKSL_, NKSLp_, k_] := Module[
  {terms, S, L, Sp, Lp, termsWithSameSpin, SL, fsubkVal,
  spinMultiplicity,
   prefactor, summand1, summand2},
   {S, L}   = FindSL[NKSL];
   {Sp, Lp} = FindSL[NKSLp];
   terms = AllowedNKSLTerms[numE];
   (* sum for summand1 is over terms with same spin *)
   spinMultiplicity  = 2*S + 1;
   termsWithSameSpin = StringCases[terms, ToString[spinMultiplicity]
  ~~ __];
   termsWithSameSpin = Flatten[termsWithSameSpin];
   If[Not[{S, L} == {Sp, Lp}],
     Return[0]
   ];
   prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
   summand1 = Sum[(
       ReducedUkTable[{numE, orbital, SL, NKSL,  k}] *
       ReducedUkTable[{numE, orbital, SL, NKSLp, k}]
       ),
     {SL, termsWithSameSpin}
   ];
   summand1 = 1 / TPO[L] * summand1;
   summand2 = (
     KroneckerDelta[NKSL, NKSLp] *
       (numE *(4*orbital + 2 - numE)) /
       ((2*orbital + 1) * (4*orbital + 1))
     );
   fsubkVal = prefactor*(summand1 - summand2);
   Return[fsubkVal];
]

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

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

FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the corresponding {E0,
   E1, E2, E3} values.
See eqn. 2-80 in Wybourne. Note that in that equation the subscripted
   Slater integrals are used but since this function assumes the the
  input values are superscripted Slater integrals, it is necessary to
   convert them using Dk.";
FtoE[F0_, F2_, F4_, F6_] := (Module[
  {E0, E1, E2, E3},
```

```
643     E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
644     E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
645     E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;
646     E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
647     Return[{E0, E1, E2, E3}];
648   ]
649   );
650
651   EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the corresponding {F0,
        F2, F4, F6} values. The inverse of FtoE.";
652   EtoF[E0_, E1_, E2_, E3_] := (Module[
653     {F0, F2, F4, F6},
654     F0 = 1/7      (7 E0 + 9 E1);
655     F2 = 75/14    (E1 + 143 E2 + 11 E3);
656     F4 = 99/7     (E1 - 130 E2 + 4 E3);
657     F6 = 5577/350 (E1 + 35 E2 - 7 E3);
658     Return[{F0, F2, F4, F6}];
659   ]
660   );
661
662   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\".";
663   Options[Electrostatic] = {"Coefficients" -> "Slater"};
664   Electrostatic[{numE_, NKSL_, NKSLp_}, OptionsPattern[]]:= Module[
665     {fsub0, fsub2, fsub4, fsub6,
666      esub0, esub1, esub2, esub3,
667      fsup0, fsup2, fsup4, fsup6,
668      eMatrixVal, orbital},
669     orbital = 3;
670     Which[
671       OptionValue["Coefficients"] == "Slater",
672       (
673         fsub0 = fsubk[numE, orbital, NKSL, NKSLp, 0];
674         fsub2 = fsubk[numE, orbital, NKSL, NKSLp, 2];
675         fsub4 = fsubk[numE, orbital, NKSL, NKSLp, 4];
676         fsub6 = fsubk[numE, orbital, NKSL, NKSLp, 6];
677         eMatrixVal = fsub0*F0 + fsub2*F2 + fsub4*F4 + fsub6*F6;
678       ),
679       OptionValue["Coefficients"] == "Racah",
680       (
681         fsup0 = fsupk[numE, orbital, NKSL, NKSLp, 0];
682         fsup2 = fsupk[numE, orbital, NKSL, NKSLp, 2];
683         fsup4 = fsupk[numE, orbital, NKSL, NKSLp, 4];
684         fsup6 = fsupk[numE, orbital, NKSL, NKSLp, 6];
685         esub0 = fsup0;
686         esub1 = 9/7*fsup0 +   1/42*fsup2 +   1/77*fsup4 +  1/462*fsup6;
687         esub2 =             143/42*fsup2 - 130/77*fsup4 + 35/462*fsup6;
688         esub3 =              11/42*fsup2 + 4/77*fsup4   -  7/462*fsup6;
689         eMatrixVal = esub0*E0 + esub1*E1 + esub2*E2 + esub3*E3;
690       )
```

```
691       ];
692       Return[eMatrixVal];
693     ]
694
695     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.";
696     Options[GenerateElectrostaticTable] = {"Export" -> True, "
          Coefficients" -> "Slater"};
697     GenerateElectrostaticTable[numEmax_Integer:7, OptionsPattern[]]:= (
698       ElectrostaticTable = Table[
699         {numE, SL, SpLp} -> SimplifyFun[Electrostatic[{numE, SL, SpLp}, "
          Coefficients" -> OptionValue["Coefficients"]]],
700         {numE, 1, numEmax},
701         {SL, AllowedNKSLTerms[numE]},
702         {SpLp, AllowedNKSLTerms[numE]}
703       ];
704       ElectrostaticTable = Association[Flatten[ElectrostaticTable]];
705       If[OptionValue["Export"],
706         Export[FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}],
707         ElectrostaticTable];
708       ];
709       Return[ElectrostaticTable];
710     )
711
712     (* ###################### Electrostatic ###################### *)
713     (* ########################################################## *)
714
715     (* ########################################################## *)
716     (* ########################## Bases ######################### *)
717
718     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}.";
719     BasisTableGenerator[numE_] := Module[{energyStatesTable, allowedJ, J,
           Jp},
720       (
721         energyStatesTable = <||>;
722         allowedJ = AllowedJ[numE];
723         Do[
724         (
725           energyStatesTable[{numE, J}] = EnergyStates[numE, J];
726         ),
727         {Jp, allowedJ},
728         {J,  allowedJ}];
729         Return[energyStatesTable]
730       )
731       ];
732
733     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}.";
734 BasisLSJMJ[numE_] := Module[{energyStatesTable, basis, idx1},
735   (
736     energyStatesTable = BasisTableGenerator[numE];
737     basis = Table[
738       energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
739       {idx1, 1, Length[AllowedJ[numE]]}];
740     basis = Flatten[basis, 1];
741     Return[basis]
742   )
743   ];
744
745 (* ########################## Bases ########################### *)
746 (* ############################################################ *)
747
748 (* ############################################################ *)
749 (* ########### Coefficients of Fracional Parentage ########### *)
750
751 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.";
752 Options[GenerateCFP] = {"Export" -> True, "PhaseFunction"-> "NK"};
753 GenerateCFP[OptionsPattern[]]:= (
754   CFP = Table[
755     {numE, NKSL} -> First[CFPTerms[numE, NKSL]],
756     {numE, 1, 7},
757     {NKSL, AllowedNKSLTerms[numE]}];
758   CFP = Association[CFP];
759   (* Go all the way to f14 *)
760   CFP = CFPExpander["Export" -> False, "PhaseFunction"-> OptionValue[
   "PhaseFunction"]];
761   If[OptionValue["Export"],
762     Export[FileNameJoin[{moduleDir, "data", "CFPs.m"}], CFP];
763   ];
764   Return[CFP];
765 )
766
767 JuddCFPPhase::usage="Phase between conjugate coefficients of
   fractional parentage according to Velkov's thesis, page 40.";
768 JuddCFPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
   parentSeniority_, daughterSeniority_] := Module[
769   {spin, orbital, expo, phase},
770 (
771     {spin, orbital} = {1/2, 3};
772     expo = (
773         (parentS + parentL + daughterS + daughterL) -
774         (orbital + spin) +
775         1/2 * (parentSeniority + daughterSeniority - 1)
```

```
776       );
777       phase = Phaser[-expo];
778       Return[phase];
779   )
780   ]
781
782   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).";
783   NKCFPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
      parentSeniority_, daughterSeniority_] := Module[{spin, orbital,
      expo, phase},
784    (
785       {spin, orbital} = {1/2, 3};
786       expo = (
787           (parentS + parentL + daughterS + daughterL) -
788           (orbital + spin)
789       );
790       phase = Phaser[-expo];
791       If[parent == 2*orbital,
792           phase = phase * Phaser[(daughterSeniority-1)/2]];
793       Return[phase];
794    )
795   ]
796
797   Options[CFPExpander] = {"Export" -> True, "PhaseFunction" -> "NK"};
798   CFPExpander::usage="Using the coefficients of fractional parentage up
        to f7 this function calculates them up to f14.
799   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.";
800   CFPExpander[OptionsPattern[]]:=Module[
801     {orbital, halfFilled, fullShell, parentMax, PhaseFun,
802     complementaryCFPs, daughter, conjugateDaughter,
803     conjugateParent, parentTerms, daughterTerms,
804     parentCFPs, daughterSeniority, daughterS, daughterL,
805     parentCFP, parentTerm, parentCFPval,
806     parentS, parentL, parentSeniority, phase, prefactor,
807     newCFPval, key, extendedCFPs, exportFname},
808   (
809       orbital    = 3;
810       halfFilled = 2 * orbital + 1;
811       fullShell  = 2 * halfFilled;
812       parentMax  = 2 * orbital;
813
814       PhaseFun  = <|
815           "Judd" -> JuddCFPPhase,
816           "NK" -> NKCFPPhase|>[OptionValue["PhaseFunction"]];
817       PrintTemporary["Calculating CFPs using the phase system from ",
      PhaseFun];
818       (* Initialize everything with lists to be filled in the next Do*)
```

```
819       complementaryCFPs =
820           Table[
821           ({numE, term} -> {term}),
822           {numE, halfFilled + 1, fullShell - 1, 1},
823           {term, AllowedNKSLTerms[numE]
824           }];
825       complementaryCFPs = Association[Flatten[complementaryCFPs]];
826       Do[(
827           daughter          = parent + 1;
828           conjugateDaughter = fullShell - parent;
829           conjugateParent   = conjugateDaughter - 1;
830           parentTerms       = AllowedNKSLTerms[parent];
831           daughterTerms     = AllowedNKSLTerms[daughter];
832           Do[
833           (
834               parentCFPs             = Rest[CFP[{daughter, daughterTerm
    }]];
835               daughterSeniority      = Seniority[daughterTerm];
836               {daughterS, daughterL} = FindSL[daughterTerm];
837               Do[
838               (
839                   {parentTerm, parentCFPval} = parentCFP;
840                   {parentS, parentL}         = FindSL[parentTerm];
841                   parentSeniority            = Seniority[parentTerm];
842                   phase = PhaseFun[parent, parentS, parentL,
843                                    daughterS, daughterL,
844                                    parentSeniority, daughterSeniority];
845                   prefactor = (daughter * TPO[daughterS, daughterL]) /
846                               (conjugateDaughter * TPO[parentS, parentL
    ]);
847                   prefactor = Sqrt[prefactor];
848                   newCFPval = phase * prefactor * parentCFPval;
849                   key = {conjugateDaughter, parentTerm};
850                   complementaryCFPs[key] = Append[complementaryCFPs[key
    ], {daughterTerm, newCFPval}]
851               ),
852               {parentCFP, parentCFPs}
853               ]
854           ),
855           {daughterTerm, daughterTerms}
856           ]
857           ),
858       {parent, 1, parentMax}
859       ];
860
861       complementaryCFPs[{14, "1S"}] = {"1S", {"2F",1}};
862       extendedCFPs         = Join[CFP, complementaryCFPs];
863       If[OptionValue["Export"];,
864       (
865           exportFname = FileNameJoin[{moduleDir, "data", "CFPs_extended
    .m"}];
866           Print["Exporting to ", exportFname];
867           Export[exportFname, extendedCFPs];
868       )
869       ];
```

```
870         Return[extendedCFPs];
871     )
872     ]
873
874     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.
875     The data being parsed here is the file attachment B1F_ALL.TXT which
        comes from Velkov's thesis.";
876     Options[GenerateCFPTable] = {"Export" -> True};
877     GenerateCFPTable[OptionsPattern[]]:=Module[
878       {rawText, rawLines, leadChar, configIndex,
879       line, daughter, lineParts, numberCode, parsedNumber, toAppend,
        CFPTablefname},
880     (
881       CleanWhitespace[string_]      := StringReplace[string,
        RegularExpression["\\s+"]->" "];
882       AddSpaceBeforeMinus[string_] := StringReplace[string,
        RegularExpression["(?<!\\s)-"]->" -"];
883       ToIntegerOrString[list_]      := Map[If[StringMatchQ[#, NumberString
        ], ToExpression[#], #] &, list];
884       CFPTable       = ConstantArray[{},7];
885       CFPTable[[1]] = {{"2F",{"1S",1}}};
886
887       (* Cleaning before processing is useful *)
888       rawText   = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"
        }]];
889       rawLines = StringTrim/@StringSplit[rawText,"\n"];
890       rawLines = Select[rawLines,#!=""&];
891       rawLines = CleanWhitespace/@rawLines;
892       rawLines = AddSpaceBeforeMinus/@rawLines;
893
894       Do[(
895         (* the first character can be used to identify the start of a
        block *)
896         leadChar=StringTake[line,{1}];
897         (* ..FN, N is at position 50 in that line *)
898         If[leadChar=="[",
899         (
900           configIndex=ToExpression[StringTake[line,{50}]];
901           Continue[];
902         )
903         ];
904         (* Identify which daughter term is being listed *)
905         If[StringContainsQ[line,"[DAUGHTER TERM]"],
906           daughter=StringSplit[line,"["][[1]];
907           CFPTable[[configIndex]]=Append[CFPTable[[configIndex]],{
        daughter}];
908           Continue[];
909         ];
910         (* Once we get here we are already parsing a row with coefficient
         data *)
911         lineParts     = StringSplit[line," "];
912         parent        = lineParts[[1]];
```

```
913       numberCode   = ToIntegerOrString[lineParts[[3;;]]];
914       parsedNumber = SquarePrimeToNormal[numberCode];
915       toAppend     = {parent,parsedNumber};
916       CFPTable[[configIndex]][[-1]] = Append[CFPTable[[configIndex
        ]][[-1]], toAppend]
917     ),
918     {line,rawLines}];
919     If[OptionValue["Export"],
920       (
921       CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
922       Export[CFPTablefname, CFPTable];
923       )
924     ];
925     Return[CFPTable];
926   )
927   ]
928
929   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.";
930   Options[GenerateCFPAssoc] = {"Export" -> True};
931   GenerateCFPAssoc[OptionsPattern[]]:= (
932     CFPAssoc = Association[];
933     Do[
934       (daughterTerms = AllowedNKSLTerms[numE];
935       parentTerms    = AllowedNKSLTerms[numE - 1];
936       Do[
937         (
938         cfps = CFP[{numE, daughter}];
939         cfps = cfps[[2 ;;]];
940         parents = First /@ cfps;
941         Do[
942           (
943           key = {numE, daughter, parent};
944           cfp = If[
945             MemberQ[parents, parent],
946             (
947               idx = Position[parents, parent][[1, 1]];
948               cfps[[idx]][[2]]
949             ),
950             0
951             ];
952           CFPAssoc[key] = cfp;
953           ),
954           {parent, parentTerms}
955           ]
956         ),
957         {daughter, daughterTerms}
958         ]
959       ),
960       {numE, 1, 14}
961       ];
962     If[OptionValue["Export"],
```

```
963         (
964         CFPAssocfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
965         Export[CFPAssocfname, CFPAssoc];
966         )
967       ];
968       Return[CFPAssoc];
969     )
970
971     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.
972     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.
973     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.
974     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}, ...}.
975     Only the one-body coefficients for f-electrons are provided.
976     In all cases it must be that 1 <= n <= 7.
977     ";
978     CFPTerms[numE_] := Part[CFPTable, numE]
979     CFPTerms[numE_, SL_] :=
980       Module[
981         {NKterms, CFPconfig},
982         NKterms   = {{}};
983         CFPconfig = CFPTable[[numE]];
984         Map[
985           If[StringFreeQ[First[#], SL],
986             Null,
987             NKterms = Join[NKterms, {#}, 1]
988           ] &,
989         CFPconfig
990         ];
991         NKterms = DeleteCases[NKterms, {}]
992       ]
993     CFPTerms[numE_, L_, S_] :=
994     Module[
995       {NKterms, SL, CFPconfig},
996       SL = StringJoin[ToString[2 S + 1], PrintL[L]];
997       NKterms = {{}};
998       CFPconfig = Part[CFPTable, numE];
999       Map[
1000        If[StringFreeQ[First[#], SL],
1001          Null,
1002          NKterms = Join[NKterms, {#}, 1]
1003        ]&,
1004      CFPconfig
1005      ];
1006      NKterms = DeleteCases[NKterms, {}]
1007    ]
```

```
1008
1009  (* ########### Coefficients of Fracional Parentage ########### *)
1010  (* ############################################################ *)
1011
1012  (* ############################################################ *)
1013  (* ######################## Spin Orbit ########################## *)
1014
1015  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.
1016  See equations 2-106 and 2-109 in Wybourne (1965). Equivalently see
        eqn. 12.43 in TASS.";
1017  SpinOrbit[numE_, SL_, SpLp_, J_]:= Module[
1018    {S, L, Sp, Lp, orbital, sign, prefactor, val},
1019    orbital   = 3;
1020    {S, L}    = FindSL[SL];
1021    {Sp, Lp}  = FindSL[SpLp];
1022    prefactor = Sqrt[orbital * (orbital+1) * (2*orbital+1)] *
1023                SixJay[{L, Lp, 1}, {Sp, S, J}];
1024    sign      = Phaser[J + L + Sp];
1025    val       = sign * prefactor * ζ * ReducedV1kTable[{numE, SL, SpLp,
         1}];
1026    Return[val];
1027  ]
1028
1029  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.";
1030  Options[GenerateSpinOrbitTable] = {"Export" -> True};
1031  GenerateSpinOrbitTable[nmax_Integer:7, OptionsPattern[]]:= Module[
1032    {numE, J, SL, SpLp, exportFname},
1033  (
1034    SpinOrbitTable =
1035      Table[
1036        {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
1037      {numE, 1, nmax},
1038      {J, MinJ[numE], MaxJ[numE]},
1039      {SL,   Map[First, AllowedNKSLforJTerms[numE, J]]},
1040      {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
1041      ];
1042    SpinOrbitTable = Association[SpinOrbitTable];
1043
1044    exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"
       }];
1045    If[OptionValue["Export"],
1046      (
1047        Print["Exporting to file "<>ToString[exportFname]];
1048        Export[exportFname, SpinOrbitTable];
1049      )
1050    ];
```

```
    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]}
    ];

    (* 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"],
      (
```

```
1152          (*export them*)
1153          PrintTemporary["Exporting ..."];
1154          exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.m
      "}];
1155          Export[exportFname, juddOperators];
1156        )
1157      ];
1158      Return[juddOperators];
1159    )
1160
1161  GenerateThreeBodyTables::usage="This function generates the matrix
      elements for the three body operators using the coefficients of
      fractional parentage, including those beyond f^7.";
1162  Options[GenerateThreeBodyTables] = {"Export" -> False};
1163  GenerateThreeBodyTables[nmax_Integer : 14, OptionsPattern[]] := (
1164    tiKeys = {"t_{2}", "t_{2}^{'}", "t_{3}", "t_{4}", "t_{6}", "t_{7}",
1165      "t_{8}", "t_{11}", "t_{11}^{'}", "t_{12}", "t_{14}", "t_{15}",
1166      "t_{16}", "t_{17}", "t_{18}", "t_{19}"};
1167    TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
1168    juddOperators = ParseJudd1984[];
1169    (* 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. *)
1170    op3MatrixElement[SL_, SpLp_, opSymbol_] := (
1171      jOP = juddOperators[{3, opSymbol}];
1172      key = {SL, SpLp};
1173      val = If[MemberQ[Keys[jOP], key],
1174        jOP[key],
1175        0];
1176      Return[val];
1177      );
1178    (*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. *)
1179    Options[ti] = {"Fast" -> True};
1180    ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
1181     Module[{nn, S, L, Sp, Lp,
1182        cfpSL, cfpSpLp,
1183        parentSL, parentSpLp, tnk, tnks},
1184      {S, L}   = FindSL[SL];
1185      {Sp, Lp} = FindSL[SpLp];
1186      fast     = OptionValue["Fast"];
1187      numH = 14 - nE;
1188      If[fast && Not[MemberQ[{"t_{2}","t_{11}"},tiKey]] && nE > 7,
1189        Return[-tktable[{numH, SL, SpLp, tiKey}]]
1190      ];
1191      If[(S == Sp && L == Lp),
1192       (
1193        cfpSL   = CFP[{nE, SL}];
1194        cfpSpLp = CFP[{nE, SpLp}];
1195        tnks = Table[(
```

```
1196            parentSL   = cfpSL[[nn, 1]];
1197            parentSpLp = cfpSpLp[[mm, 1]];
1198            cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
1199            tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
1200            ),
1201            {nn, 2, Length[cfpSL]},
1202            {mm, 2, Length[cfpSpLp]}
1203            ];
1204       tnk = Total[Flatten[tnks]];
1205       ),
1206      tnk = 0;
1207      ];
1208    Return[ nE / (nE - opOrder) * tnk];];
1209  (*Calculate the matrix elements of t^i for n up to nmax*)
1210  tktable = <||>;
1211  Do[(
1212    Do[(
1213      tkValue = Which[numE <= 2,
1214        (*Initialize n=1,2 with zeros*)
1215        0,
1216        numE == 3,
1217        (*Grab matrix elem in f^3 from Judd 1984*)
1218        SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
1219        True,
1220        SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2,
    3]]]
1221        ];
1222      tktable[{numE, SL, SpLp, opKey}] = tkValue;
1223      ),
1224    {SL, AllowedNKSLTerms[numE]},
1225    {SpLp, AllowedNKSLTerms[numE]},
1226    {opKey, Append[tiKeys, "e_{3}"]}
1227    ];
1228    PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], "
    configuration complete"]];
1229    ),
1230  {numE, 1, nmax}
1231  ];
1232
1233  (* Now use those matrix elements to determine their sum as weighted
     by their corresponding strengths Ti *)
1234  ThreeBodyTable = <||>;
1235  Do[
1236    Do[
1237    (
1238      ThreeBodyTable[{numE, SL, SpLp}] = (
1239        Sum[(
1240          If[tiKey == "t_{2}", t2Switch, 1] *
1241          tktable[{numE, SL, SpLp, tiKey}] *
1242          TSymbolsAssoc[tiKey] +
1243          If[tiKey == "t_{2}", 1 - t2Switch, 0] *
1244          (-tktable[{14 - numE, SL, SpLp, tiKey}]) *
1245          TSymbolsAssoc[tiKey]
1246          ),
1247        {tiKey, tiKeys}
```

```
        ]
      );
    ),
    {SL, AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]}
    ];
  PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix
    complete"]];,
  {numE, 1, 7}
  ];

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

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

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

```
        factor = TPO[S, L];
        factor * op1[{term1, term2}] * op2[{term2, term1}]
        ),
      {term1, terms},
      {term2, terms}
      ]
    ]
    )
];

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

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

ReducedT11inf2::usage="ReducedT11inf2[SL, SpLp] returns the reduced
  matrix element of the scalar component of the double tensor T11 for
   the given SL terms SL, SpLp.
Data used here for m0, m2, m4 is from Table II of Judd, BR, HM
  Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
  Interactions for f Electrons. Physical Review 169, no. 1 (1968):
  130.
";
ReducedT11inf2[SL_, SpLp_] :=
  Module[{T11inf2},
  T11inf2 = <|
    {"1S", "3P"} -> 6 M0 + 2 M2 + 10/11 M4,
    {"3P", "3P"} -> -36 M0 - 72 M2 - 900/11 M4,
    {"3P", "1D"} -> -Sqrt[(2/15)] (27 M0 + 14 M2 + 115/11 M4),
    {"1D", "3F"} -> Sqrt[2/5] (23 M0 + 6 M2 - 195/11 M4),
    {"3F", "3F"} -> 2 Sqrt[14] (-15 M0 - M2 + 10/11 M4),
    {"3F", "1G"} -> Sqrt[11] (-6 M0 + 64/33 M2 - 1240/363 M4),
    {"1G", "3H"} -> Sqrt[2/5] (39 M0 - 728/33 M2 - 3175/363 M4),
    {"3H", "3H"} -> 8/Sqrt[55] (-132 M0 + 23 M2 + 130/11 M4),
    {"3H", "1I"} -> Sqrt[26] (-5 M0 - 30/11 M2 - 375/1573 M4)
    |>;
  Which[
    MemberQ[Keys[T11inf2],{SL,SpLp}],
      Return[T11inf2[{SL,SpLp}]],
    MemberQ[Keys[T11inf2],{SpLp,SL}],
      Return[T11inf2[{SpLp,SL}]],
    True,
      Return[0]
  ]
  ];

Reducedt11inf2::usage="Reducedt11inf2[SL, SpLp] returns the reduced
  matrix element in f^2 of the double tensor operator t11 for the
  corresponding given terms {SL, SpLp}.
Values given here are those from Table VII of \"Judd, BR, HM
  Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
  Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
  130.\"
";
```

```
Reducedt11inf2[SL_, SpLp_]:= Module[
  {t11inf2},
  t11inf2 = <|
    {"1S", "3P"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
    {"3P", "3P"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
    {"3P", "1D"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
    {"1D", "3F"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
    {"3F", "3F"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
    {"3F", "1G"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
    {"1G", "3H"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
    {"3H", "3H"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
    {"3H", "1I"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
    |>;
  Which[
    MemberQ[Keys[t11inf2],{SL,SpLp}],
      Return[t11inf2[{SL,SpLp}]],
    MemberQ[Keys[t11inf2],{SpLp,SL}],
      Return[t11inf2[{SpLp,SL}]],
    True,
      Return[0]
  ]
]

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

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

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

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

```
    funval *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
    Return[funval];
  ];

  GenerateSOOandECSOLSTable::usage="GenerateSOOandECSOLSTable[nmax]
    generates the LS reduced matrix elements of the spin-other-orbit +
    ECSO for the f^n configurations up to n=nmax. The values for n=1
    and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
     Physical Review 169, no. 1 (1968): 130.\", and the values for n>2
    are calculated recursively using equation (4) of that same paper.
    The values are then exported to a file \"ReducedSOOandECSOLSTable.m
    \" in the data folder of this module. The values are also returned
    as an association.";
  Options[GenerateSOOandECSOLSTable] = {"Progress" -> True, "Export" ->
     True};
  GenerateSOOandECSOLSTable[nmax_Integer, OptionsPattern[]]:= (
    If[And[OptionValue["Progress"], frontEndAvailable],
      (
        numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
    numE]]^2, {numE, 1, nmax}]];
        counters   = Association[Table[numE->0, {numE, 1, nmax}]];
        totalIters = Total[Values[numItersai[[1;;nmax]]]];
        template1  = StringTemplate["Iteration `numiter` of `totaliter`
    "];
        template2  = StringTemplate["`remtime` min remaining"];
    template3 = StringTemplate["Iteration speed = `speed` ms/it"];
        template4  = StringTemplate["Time elapsed = `runtime` min"];
        progBar = PrintTemporary[
          Dynamic[
            Pane[
              Grid[{
                    {Superscript["f", numE]},
                    {template1[<|"numiter"->numiter, "totaliter"->
    totalIters|>]},
                    {template4[<|"runtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                    {template2[<|"remtime"->Round[QuantityMagnitude[
    UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
    ]], 0.1]|>]},
                    {template3[<|"speed"->Round[QuantityMagnitude[Now-
    startTime, "ms"]/(numiter), 0.01]|>]}, {ProgressIndicator[Dynamic[
    numiter], {1, totalIters}]}
                  },
                  Frame->All
              ],
              Full,
              Alignment->Center
            ]
          ]
        ];
      )
    ];
    SOOandECSOLSTable = <||>;
    numiter   = 1;
```

```
1467      startTime = Now;
1468      Do[
1469        (
1470          numiter+= 1;
1471          SOOandECSOLSTable[{numE, SL, SpLp}] = Which[
1472            numE==1,
1473            0,
1474            numE==2,
1475            SimplifyFun[ReducedSOOandECSOinf2[SL, SpLp]],
1476            True,
1477            SimplifyFun[ReducedSOOandECSOinfn[numE,  SL, SpLp]]
1478          ];
1479        ),
1480      {numE, 1, nmax},
1481      {SL, AllowedNKSLTerms[numE]},
1482      {SpLp, AllowedNKSLTerms[numE]}
1483      ];
1484      If[And[OptionValue["Progress"], frontEndAvailable],
1485        NotebookDelete[progBar]];
1486      If[OptionValue["Export"],
1487        (fname = FileNameJoin[{moduleDir, "data", "
      ReducedSOOandECSOLSTable.m"}];
1488        Export[fname, SOOandECSOLSTable];
1489        )
1490      ];
1491      Return[SOOandECSOLSTable];
1492    );
1493
1494    (* ################## Reduced SOO and ECSO ################## *)
1495    (* ######################################################### *)
1496
1497    (* ######################################################### *)
1498    (* ###################### Spin-Spin ######################## *)
1499
1500    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.
1501    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.
1502    ";
1503    ReducedT22inf2[SL_, SpLp_] :=
1504      Module[{statePosition, PsiPsipStates, m0, m2, m4, Tkk2m},
1505      T22inf2 = <|
1506      {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
1507      {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
1508      {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
1509      {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
1510      {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
1511      |>;
1512      Which[
1513        MemberQ[Keys[T22inf2],{SL,SpLp}],
1514          Return[T22inf2[{SL,SpLp}]],
1515        MemberQ[Keys[T22inf2],{SpLp,SL}],
```

```
        Return[T22inf2[{SpLp,SL}]],
      True,
        Return[0]
    ]
    ];

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

GenerateT22Table::usage="GenerateT22Table[nmax] generates the LS
  reduced matrix elements for the double tensor operator T22 in f^n
  up to n=nmax. If the option \"Export\" is set to true then the
  resulting association is saved to the data folder. The values for n
  =1 and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
  Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
   Physical Review 169, no. 1 (1968): 130.\", and the values for n>2
  are calculated recursively using equation (4) of that same paper.
This is an intermediate step to the calculation of the reduced matrix
   elements of the spin-spin operator.";
```

```
Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
GenerateT22Table[nmax_Integer, OptionsPattern[]]:= (
  If[And[OptionValue["Progress"], frontEndAvailable],
    (
      numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
numE]]^2, {numE, 1, nmax}]];
      counters = Association[Table[numE->0, {numE, 1, nmax}]];
      totalIters = Total[Values[numItersai[[1;;nmax]]]];
      template1 = StringTemplate["Iteration `numiter` of `totaliter`"
];
      template2 = StringTemplate["`remtime` min remaining"];template3
 = StringTemplate["Iteration speed = `speed` ms/it"];
      template4 = StringTemplate["Time elapsed = `runtime` min"];
      progBar = PrintTemporary[
        Dynamic[
          Pane[
            Grid[{{Superscript["f", numE]},
                  {template1[<|"numiter"->numiter, "totaliter"->
totalIters|>]},
                  {template4[<|"runtime"->Round[QuantityMagnitude[
UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                  {template2[<|"remtime"->Round[QuantityMagnitude[
UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
]], 0.1]|>]},
                  {template3[<|"speed"->Round[QuantityMagnitude[Now-
startTime, "ms"]/(numiter), 0.01]|>]},
                  {ProgressIndicator[Dynamic[numiter], {1, totalIters
}]}},
                  Frame->All],
                  Full,
                  Alignment->Center]
              ]
            ];
    )
  ];
  T22Table = <||>;
  startTime = Now;
  numiter = 1;
  Do[
    (
      numiter+= 1;
      T22Table[{numE, SL, SpLp}] = Which[
        numE==1,
        0,
        numE==2,
        SimplifyFun[ReducedT22inf2[SL, SpLp]],
        True,
        SimplifyFun[ReducedT22infn[numE,  SL, SpLp]]
      ];
    ),
  {numE, 1, nmax},
  {SL,   AllowedNKSLTerms[numE]},
  {SpLp, AllowedNKSLTerms[numE]}
  ];
  If[And[OptionValue["Progress"],frontEndAvailable],
```

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

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

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

```
    {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
    ];
    If[OptionValue["Export"],
    (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
      Export[fname, SpinSpinTable];
      )
    ];
    Return[SpinSpinTable];
    );

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

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

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

```
1678  GenerateSOOandECSOTable[nmax_, OptionsPattern[]]:= (
1679    SOOandECSOTable = <||>;
1680    Do[
1681      SOOandECSOTable[{numE, SL, SpLp, J}] = (SOOandECSO[numE, SL, SpLp
        , J] /. Prescaling);,
1682    {numE, 1, nmax},
1683    {J, MinJ[numE], MaxJ[numE]},
1684    {SL,   First /@ AllowedNKSLforJTerms[numE, J]},
1685    {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
1686    ];
1687    If[OptionValue["Export"],
1688    (
1689      fname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
1690      Export[fname, SOOandECSOTable];
1691    )
1692    ];
1693    Return[SOOandECSOTable];
1694  );
1695
1696  (* ## Spin-Other-Orbit and Electrostatically-Correlated-Spin-Orbit ##
       *)
1697  (* ################################################################ *)
1698
1699  (* ################################################################ *)
1700  (* ################### Magnetic Interactions ################### *)
1701
1702  MagneticInteractions::usage="MagneticInteractions[{numE, SLJ, SLJp, J
      }] returns the matrix element of the magnetic interaction between
      the terms SLJ and SLJp in the f^n configuration. The interaction is
       given by the sum of the spin-spin interaction and the SOO and ECSO
       interactions. The spin-spin interaction is given by the function
      SpinSpin[{numE, SLJ, SLJp, J}]. The SOO and ECSO interactions are
      given by the function SOOandECSO[{numE, SLJ, SLJp, J}]. The
      function requires chenDeltas to be loaded into the session. The
      option \"ChenDeltas\" can be use to include or exclude the Chen
      deltas from the calculation. The default is to exclude them.";
1703  Options[MagneticInteractions] = {"ChenDeltas" -> False};
1704  MagneticInteractions[{numE_, SLJ_, SLJp_, J_}, OptionsPattern[]] :=
1705    (
1706      key = {numE, SLJ, SLJp, J};
1707      ss = \[Sigma]SS * SpinSpinTable[key];
1708      sooandecso = SOOandECSOTable[key];
1709      total = ss + sooandecso;
1710      total = SimplifyFun[total];
1711      If[
1712        Not[OptionValue["ChenDeltas"]],
1713        Return[total]
1714      ];
1715      (* In the type A errors the wrong values are different *)
1716      If[MemberQ[Keys[chenDeltas["A"]], {numE, SLJ, SLJp}],
1717        (
1718          {S, L} = FindSL[SLJ];
1719          {Sp, Lp} = FindSL[SLJp];
1720          phase    = Phaser[Sp + L + J];
1721          Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
```

```
1722            Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
1723            {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ,
    SLJp}]["wrong"];
1724            total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1725            total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1726            total  = total /. Prescaling;
1727            total  = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
1728          )
1729        ];
1730        (* In the type B errors the wrong values are zeros all around *)
1731        If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
1732          (
1733            {S, L} = FindSL[SLJ];
1734            {Sp, Lp} = FindSL[SLJp];
1735            phase = Phaser[Sp + L + J];
1736            Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
1737            Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
1738            {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
1739            total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1740            total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1741            total  = total /. Prescaling;
1742            total  = wChErrB * total + (1 - wChErrB) * (ss + sooandecso)
1743          )
1744        ];
1745        Return[total];
1746      )
1747
1748    (* ################## Magnetic Interactions ################## *)
1749    (* ########################################################### *)
1750
1751    (* ########################################################### *)
1752    (* ###################### Crystal Field ###################### *)
1753
1754    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.";
1755    Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
1756      {S, Sp, L, Lp, orbital, val},
1757      orbital = 3;
1758      {S, L}   = FindSL[NKSL];
1759      {Sp, Lp} = FindSL[NKSLp];
1760      f1  = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
1761      val =
1762        If[f1==0,
1763          0,
1764          (
1765            f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
1766            If[f2==0,
1767              0,
1768              (
1769                f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
1770                If[f3==0,
1771                  0,
1772                  (
1773                    (
```

```
                    Phaser[J - M + S + Lp + J + k] *
                    Sqrt[TPO[J, Jp]] *
                    f1 *
                    f2 *
                    f3 *
                    Ck[orbital, k]
                  )
                )
              ]
            )
          ]
        )
      ];
    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.";
  TotalCFIters[i_, j_] := (
    numIters = {196, 8281, 132496, 1002001, 4008004, 9018009,
    11778624};
    Return[Total[numIters[[i ;; j]]]];
```

```
1819     )
1820
1821   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.";
1822   Options[GenerateCrystalFieldTable] = {"Export" -> False, "Progress"
         -> True, "Compress" -> True}
1823   GenerateCrystalFieldTable[numEs_List:{1,2,3,4,5,6,7}, OptionsPattern
         []]:= (
1824     ExportFun =
1825     If[OptionValue["Compress"],
1826       ExportMZip,
1827       Export
1828     ];
1829     numiter = 1;
1830     template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
1831     template2 = StringTemplate["`remtime` min remaining"];
1832     template3 = StringTemplate["Iteration speed = `speed` ms/it"];
1833     template4 = StringTemplate["Time elapsed = `runtime` min"];
1834     totalIter = Total[TotalCFIters[#, #] & /@ numEs];
1835     freebies = 0;
1836     startTime = Now;
1837     If[And[OptionValue["Progress"], frontEndAvailable],
1838       progBar = PrintTemporary[
1839         Dynamic[
1840           Pane[
1841             Grid[
1842               {
1843                 {Superscript["f", numE]},
1844                 {template1[<|"numiter" -> numiter, "totaliter" ->
         totalIter|>]},
1845                 {template4[<|"runtime" -> Round[QuantityMagnitude[
         UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
1846                 {template2[<|"remtime" -> Round[QuantityMagnitude[
         UnitConvert[(Now - startTime)/(numiter - freebies) * (totalIter -
         numiter), "min"]], 0.1]|>]},
1847                 {template3[<|"speed" -> Round[QuantityMagnitude[Now -
         startTime, "ms"]/(numiter-freebies), 0.01]|>]},
1848                 {ProgressIndicator[Dynamic[numiter], {1, totalIter}]}
1849               },
1850             Frame -> All
1851             ],
1852           Full,
1853           Alignment -> Center
1854           ]
1855         ]
1856       ];
1857     ];
```

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

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

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

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

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

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

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

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

```
      0
    ]
  )

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

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

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

  JJBlockMatrix::usage = "For given J, J' in the f^n configuration
    JJBlockMatrix[numE, J, J'] determines all the SL S'L' terms that
    may contribute to them and using those it provides the matrix
    elements <J, LS | H | J', LS'>. H having contributions from the
    following interactions: Coulomb, spin-orbit, spin-other-orbit,
    electrostatically-correlated-spin-orbit, spin-spin, three-body
    interactions, and crystal-field.";
  Options[JJBlockMatrix] = {"Sparse"->True, "ChenDeltas"->False};
  JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]]:= Module[
    {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
    SLterm, SpLpterm,
    MJ, MJp,
    subKron, matValue, eMatrix},
    (
      NKSLJMs  = AllowedNKSLJMforJTerms[numE, J];
      NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
      eMatrix =
        Table[
          (*Condition for a scalar matrix op*)
          SLterm   =  NKSLJM[[1]];
          SpLpterm = NKSLJMp[[1]];
          MJ       =  NKSLJM[[3]];
          MJp      = NKSLJMp[[3]];
          subKron  =
            (
```

```
1999                KroneckerDelta[J, Jp] *
2000                KroneckerDelta[MJ, MJp]
2001              );
2002            matValue =
2003              If[subKron==0,
2004                0,
2005                  (
2006                    ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2007                    ElectrostaticConfigInteraction[{SLterm, SpLpterm}] +
2008                    SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2009                    MagneticInteractions[{numE, SLterm, SpLpterm, J}, "
    ChenDeltas" -> OptionValue["ChenDeltas"]] +
2010                    ThreeBodyTable[{numE, SLterm, SpLpterm}]
2011                  )
2012              ];
2013            matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp
    }];
2014            matValue,
2015          {NKSLJMp, NKSLJMps},
2016          {NKSLJM , NKSLJMs}
2017          ];
2018       If[OptionValue["Sparse"],
2019         eMatrix = SparseArray[eMatrix]
2020       ];
2021       Return[eMatrix]
2022     )
2023     ];
2024
2025     EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some
       point may be used to redefine states used in basis.";
2026     EnergyStates[numE_, J_]:= AllowedNKSLJMforJTerms[numE, J];
2027
2028     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.";
2029     Options[JJBlockMatrixFileName] = {"FilenameAppendix" -> ""}
2030     JJBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
2031       fileApp = OptionValue["FilenameAppendix"];
2032       fname = FileNameJoin[{moduleDir,
2033           "hams",
2034           StringJoin[{"f", ToString[numE], "_JJBlockMatrixTable", fileApp
        ,".m"}]}];
2035       Return[fname];
2036       );
2037
2038     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.";
Options[TabulateJJBlockMatrixTable] = {"Sparse"->True, "ChenDeltas"->
  False};
TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]]:= (
  JJBlockMatrixTable = <||>;
  totalIterations = Length[AllowedJ[numE]]^2;
  template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
  template2 = StringTemplate["`remtime` min remaining"];
  template4 = StringTemplate["Time elapsed = `runtime` min"];
  numiter   = 0;
  startTime = Now;
  If[$FrontEnd =!= Null,
    (
      temp = PrintTemporary[
        Dynamic[
          Grid[
            {
              {template1[<|"numiter"->numiter, "totaliter"->
  totalIterations|>]},
              {template2[<|"remtime"->Round[QuantityMagnitude[
  UnitConvert[(Now-startTime)/(Max[1,numiter])*(totalIterations-
  numiter), "min"]], 0.1]|>]},
              {template4[<|"runtime"->Round[QuantityMagnitude[
  UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
              {ProgressIndicator[numiter, {1, totalIterations}]}
            }
          ]
      ]
      ];
    )
  ];
  Do[
    (
      JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp,
  CFTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas" ->
  OptionValue["ChenDeltas"]];
      numiter += 1;
    ),
  {Jp, AllowedJ[numE]},
  {J, AllowedJ[numE]}
  ];
  If[$FrontEnd =!= Null,
    NotebookDelete[temp]
  ];
  Return[JJBlockMatrixTable];
)

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

```
    the values of <|LSJMJa|H|L'S'J'MJ'a'|>.";
2079  Options[TabulateManyJJBlockMatrixTables] = {"Overwrite"->False, "
    Sparse"->True, "ChenDeltas"->False, "FilenameAppendix"-> "", "
    Compressed" -> False};
2080  TabulateManyJJBlockMatrixTables[ns_, OptionsPattern[]]:= (
2081    overwrite = OptionValue["Overwrite"];
2082    fNames = <||>;
2083    fileApp = OptionValue["FilenameAppendix"];
2084    ExportFun = If[OptionValue["Compressed"], ExportMZip, Export];
2085    Do[
2086      (
2087        CFdataFilename = FileNameJoin[{moduleDir, "data", "
    CrystalFieldTable_f"<>ToString[numE]<>".zip"}];
2088        PrintTemporary["Importing CrystalFieldTable from ",
    CFdataFilename, " ..."];
2089        CrystalFieldTable = ImportMZip[CFdataFilename];
2090
2091        PrintTemporary["#------- numE = ", numE, " -------#"];
2092        exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
     fileApp];
2093        fNames[numE] = exportFname;
2094        If[FileExistsQ[exportFname] && Not[overwrite],
2095          Continue[]
2096        ];
2097        JJBlockMatrixTable = TabulateJJBlockMatrixTable[numE,
    CrystalFieldTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas" ->
     OptionValue["ChenDeltas"]];
2098        If[FileExistsQ[exportFname]&&overwrite,
2099          DeleteFile[exportFname]
2100        ];
2101        ExportFun[exportFname, JJBlockMatrixTable];
2102
2103        ClearAll[CrystalFieldTable];
2104      ),
2105    {numE, ns}
2106    ];
2107  Return[fNames];
2108  )
2109
2110  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;";
2111  Options[HamMatrixAssembly] = {
2112        "FilenameAppendix"->"",
2113        "IncludeZeeman"->False,
2114        "Set t2Switch"->False};
2115  HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
2116    {numE, ii, jj, howManyJs, Js, blockHam},
2117    (*######################################*)
2118    ImportFun = ImportMZip;
2119    (*######################################*)
```

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

SimplerSymbolicHamMatrix::usage="SimplerSymbolicHamMatrix[numE,
    simplifier] is a simple addition to HamMatrixAssembly that applies
    a given simplification to the full hamiltonian. Simplifier is a
    list of replacement rules. If the option \"Export\" is set to True,
     then the function also exports the resulting sparse array to the
    ./hams/ folder. The option \"PrependToFilename\" can be used to
```

```
    append a string to the filename to which the function may exports
    to. The option \"Return\" can be used to choose whether the
    function returns the matrix or not. The option \"Overwrite\" can be
     used to overwrite the file if it already exists. The option \"
    IncludeZeeman\" can be used to toggle the inclusion of the Zeeman
    interaction with an external magnetic field.";
Options[SimplerSymbolicHamMatrix]={
  "Export"->True,
  "PrependToFilename"->"",
  "EorF"->"F",
  "Overwrite" -> False,
  "Return" -> True,
  "IncludeZeeman" -> False};
SimplerSymbolicHamMatrix[numE_Integer, simplifier_List, OptionsPattern
    []]:=Module[
  {thisHam,eTofs,fname},
  (
    If[Not[ValueQ[ElectrostaticTable]],
      LoadElectrostatic[]
      ];
    If[Not[ValueQ[SOOandECSOTable]],
      LoadSOOandECSO[]
    ];
    If[Not[ValueQ[SpinOrbitTable]],
      LoadSpinOrbit[]
    ];
    If[Not[ValueQ[SpinSpinTable]],
      LoadSpinSpin[]
    ];
    If[Not[ValueQ[ThreeBodyTable]],
      LoadThreeBody[]
    ];

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

    thisHam=HamMatrixAssembly[numE, "IncludeZeeman"->OptionValue["
    IncludeZeeman"]];
    thisHam=ReplaceInSparseArray[thisHam, simplifier];
```

```
    If[OptionValue["Export"],
    (
      Print["Exporting to file ",fname];
      Export[fname,thisHam]
    )
    ];
    If[OptionValue["Return"],
      Return[thisHam],
      Return[Null]
    ];
  )
]

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

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

  magOp = <||>;

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

```
    alpha'*)
      summand2 = If[braSL!= ketSL,
        0,
        (gs-1) *
          Phaser[braS+braL+ketJ+1] *
            Sqrt[TPO[braJ]*TPO[ketJ]] *
              SixJay[{braJ,1,ketJ},{braS,braL,braS}] *
                Sqrt[braS(braS+1)TPO[braS]]
      ];
      matValue = summand1 + summand2;
      (* We are using the Racah convention for red matrix elements in
    Wigner-Eckart *)
      threejays = (ThreeJay[{braJ, -braMJ}, {1, #}, {ketJ, ketMJ}] &) /
    @ {-1,0,1};
      threejays *= Phaser[braJ-braMJ];
      matValue  = - 1/2 * threejays * matValue;
      matValue,
    {braSLJ, braSLJs},
    {ketSLJ, ketSLJs}
    ];
    If[OptionValue["Sparse"],
      magMatrix= SparseArray[magMatrix]
    ];
    Return[magMatrix])
  ];

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

  TabulateManyJJBlockMagDipTables::usage = "
    TabulateManyJJBlockMagDipTables[{n1, n2, ...}] calculates the
    tables of matrix elements for the requested f^n_i configurations.
    The function does not return the matrices themselves. It instead
    returns an association whose keys are numE and whose values are the
     filenames where the output of TabulateManyJJBlockMagDipTables was
    saved to. The output consists of an association whose keys are of
    the form {n, J, Jp} and whose values are rectangular arrays given
    the values of <|LSJMJa|H_dip|L'S'J'MJ'a'|>.";
  Options[TabulateManyJJBlockMagDipTables]={"FilenameAppendix"->"","
    Overwrite"->False,"Compressed"->True};
  TabulateManyJJBlockMagDipTables[ns_,OptionsPattern[]]:=(
    fnames=<||>;
    Do[
```

```
    (
      ExportFun=If[OptionValue["Compressed"],ExportMZip,Export];
      PrintTemporary["#------- numE = ",numE," -------#"];
      appendTo     = (OptionValue["FilenameAppendix"]<>"-magDip");
      exportFname  = JJBlockMatrixFileName[numE,"FilenameAppendix"->
    appendTo];
      fnames[numE] = exportFname;
      If[FileExistsQ[exportFname]&&Not[OptionValue["Overwrite"]],
        Continue[]
      ];
      JJBlockMatrixTable = TabulateJJBlockMagDipTable[numE];
      If[FileExistsQ[exportFname]&&OptionValue["Overwrite"],
        DeleteFile[exportFname]
      ];
      ExportFun[exportFname,  JJBlockMatrixTable];
    ),
    {numE,ns}
    ];
    Return[fnames];
  )

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

    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}
    ];
    blockOp = ArrayFlatten[blockOp];
    opMinus = blockOp[[;; , ;; , 1]];
```

```
2347      opZero =  blockOp[[;; , ;; , 2]];
2348      opPlus =  blockOp[[;; , ;; , 3]];
2349      opX =   (opMinus - opPlus)/Sqrt[2];
2350      opY = I (opPlus + opMinus)/Sqrt[2];
2351      opZ = opZero;
2352      Return[{opX, opY, opZ}];
2353    )
2354    ];
2355
2356    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.
2357    The option \"Units\" can be set to either \"SI\" (so that the units
        of the returned array are A/m^2) or to \"Hartree\".
2358    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.
2359    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>.";
2360    Options[MagDipLineStrength]={"Reload MagOp" -> False, "Units"->"SI",
        "States" -> All};
2361    MagDipLineStrength[theEigensys_List, numE0_Integer, OptionsPattern
        []]:=Module[
2362      {allEigenvecs, Sx, Sy, Sz, Stot ,factor},
2363    (
2364      numE = Min[14-numE0, numE0];
2365      (*If not loaded then load it, *)
2366      If[Or[
2367          Not[MemberQ[Keys[magOp], numE]],
2368          OptionValue["Reload MagOp"]],
2369      (
2370        magOp[numE] = ReplaceInSparseArray[#,{gs->2}]& /@
        MagDipoleMatrixAssembly[numE];
2371      )
2372      ];
2373      allEigenvecs = Transpose[Last/@theEigensys];
2374      Which[OptionValue["States"] === All,
2375        (
2376          {Sx,Sy,Sz}    = (ConjugateTranspose[allEigenvecs].#.allEigenvecs
        ) & /@ magOp[numE];
2377          Stot          = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
2378        ),
2379        IntegerQ[OptionValue["States"]],
2380        (
2381          singleState   = theEigensys[[OptionValue["States"],2]];
2382          {Sx,Sy,Sz}    = (ConjugateTranspose[allEigenvecs].#.singleState)
         & /@ magOp[numE];
2383          Stot          = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
2384        )
2385      ];
2386      Which[
```

```
      OptionValue["Units"] == "SI",
        Return[4 \[Mu]B^2 * Stot],
      OptionValue["Units"] == "Hartree",
        Return[Stot],
      True,
      (
        Print["Invalid option for \"Units\". Options are \"SI\" and \"Hartree\"."];
        Abort[];
      )
    ];
  )
  ]

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

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

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

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

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

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

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

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

```
2474  )
2475
2476  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.";
2477  PrintSLJ[SLJ_] :=
2478    RowBox[{SuperscriptBox[" ", 2 SLJ[[1]] + 1],
2479      SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]}] // DisplayForm;
2480
2481  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.";
2482  PrintSLJM[SLJM_] :=
2483    RowBox[{SuperscriptBox[" ", 2 SLJM[[1]] + 1],
2484      SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]}] //
2485    DisplayForm;
2486
2487  (* ################### Printers and Labels ################### *)
2488  (* ############################################################ *)
2489
2490  (* ############################################################ *)
2491  (* ##################### Term management ###################### *)
2492
2493  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.";
2494  AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE,
        14-numE]]]
2495
2496  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.";
2497  AllowedNKSLTerms[numE_] := (Map[First, CFPTerms[Min[numE, 14-numE]]])
2498  AllowedNKSLTerms[0] = {"1S"};
2499  AllowedNKSLTerms[14] = {"1S"};
2500
2501  MaxJ::usage = "MaxJ[numE] gives the maximum J = S+L that corresponds
        to the configuration f^numE.";
2502  MaxJ[numE_] := Max[Map[Total, AllowedSLTerms[Min[numE, 14-numE]]]]
2503
2504  MinJ::usage = "MinJ[numE] gives the minimum J = S+L that corresponds
        to the configuration f^numE.";
2505  MinJ[numE_] := Min[Map[Abs[Part[#, 1] - Part[#, 2]] &, AllowedSLTerms
        [Min[numE, 14-numE]]]]
2506
2507  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.";
2508  AllowedSLJTerms[numE_] :=
2509    Module[{idx1, allowedSL, allowedSLJ},
2510      allowedSL = AllowedSLTerms[numE];
2511      allowedSLJ = {};
2512      For[
2513        idx1 = 1,
2514        idx1 <= Length[allowedSL],
2515        termSL = allowedSL[[idx1]];
2516        termsSLJ =
2517          Table[
2518            {termSL[[1]], termSL[[2]], J},
2519          {J,Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2520          ];
2521        allowedSLJ = Join[allowedSLJ, termsSLJ];
2522        idx1++
2523      ];
2524      SortBy[allowedSLJ, Last]
2525    ]
2526
2527  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.";
2528  AllowedNKSLJTerms[numE_] :=
2529    Module[{allowedSL, allowedNKSL, allowedSLJ, nn},
2530      allowedNKSL = AllowedNKSLTerms[numE];
2531      allowedSL = AllowedSLTerms[numE];
2532      allowedSLJ = {};
2533      For[
2534        nn = 1,
2535        nn <= Length[allowedSL],
2536        (
2537          termSL = allowedSL[[nn]];
2538          termNKSL = allowedNKSL[[nn]];
2539          termsSLJ =
2540            Table[{termNKSL, J},
2541            {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2542            ];
2543          allowedSLJ = Join[allowedSLJ, termsSLJ];
2544          nn++
2545        )
2546      ];
2547      SortBy[allowedSLJ, Last]
2548    ]
2549
2550  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.";
2551  AllowedNKSLforJTerms[numE_, J_] := Module[
2552      {allowedSL, allowedNKSL, allowedSLJ, nn, termSL, termNKSL,
    termsSLJ},
```

```
2553        allowedNKSL = AllowedNKSLTerms[numE];
2554        allowedSL = AllowedSLTerms[numE];
2555        allowedSLJ = {};
2556        For[
2557          nn = 1,
2558          nn <= Length[allowedSL],
2559          (
2560            termSL = allowedSL[[nn]];
2561            termNKSL = allowedNKSL[[nn]];
2562            termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[
    termSL],
2563              {{termNKSL, J}},
2564              {}
2565              ];
2566            allowedSLJ = Join[allowedSLJ, termsSLJ];
2567            nn++
2568          )
2569        ];
2570        Return[allowedSLJ]
2571      ];
2572
2573    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}.";
2574    AllowedSLJMTerms[numE_] := Module[
2575        {allowedSLJ, allowedSLJM, termSLJ, termsSLJM, nn},
2576        allowedSLJ = AllowedSLJTerms[numE];
2577        allowedSLJM = {};
2578        For[
2579          nn = 1,
2580          nn <= Length[allowedSLJ],
2581          nn++,
2582          (
2583            termSLJ = allowedSLJ[[nn]];
2584            termsSLJM =
2585              Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
2586              {M, - termSLJ[[3]], termSLJ[[3]]}
2587              ];
2588            allowedSLJM = Join[allowedSLJM, termsSLJM];
2589          )
2590        ];
2591        Return[SortBy[allowedSLJM, Last]];
2592        ]
2593
2594    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}.";
2595    AllowedNKSLJMforJMTerms[numE_, J_, MJ_] :=
2596      Module[{allowedSL, allowedNKSL, allowedSLJM, nn},
2597        allowedNKSL = AllowedNKSLTerms[numE];
2598        allowedSL   = AllowedSLTerms[numE];
2599        allowedSLJM = {};
2600        For[
```

```
2601          nn = 1,
2602          nn <= Length[allowedSL],
2603          termSL = allowedSL[[nn]];
2604          termNKSL = allowedNKSL[[nn]];
2605          termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
2606                         <= J
2607                         <= Total[termSL]
2608                         && (Abs[MJ] <= J)
2609                         ),
2610                         {{termNKSL, J, MJ}},
2611                         {}];
2612          allowedSLJM = Join[allowedSLJM, termsSLJ];
2613          nn++
2614        ];
2615        Return[allowedSLJM];
2616      ]
2617
2618    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}.";
2619    AllowedNKSLJMforJTerms[numE_, J_] :=
2620    Module[{MJs, labelsAndMomenta, termsWithJ},
2621      (
2622      MJs = AllowedMforJ[J];
2623      (* Pair LS labels and their {S,L} momenta *)
2624      labelsAndMomenta = ({#, FindSL[#]}) & /@ AllowedNKSLTerms[numE];
2625      (* A given term will contain J if |L-S|<=J<=L+S *)
2626      ContainsJ[{SL_String, {S_, L_}}] := (Abs[S - L] <= J <= (S + L));
2627      (* Keep just the terms that satisfy this condition *)
2628      termsWithJ = Select[labelsAndMomenta, ContainsJ];
2629      (* We don't want to keep the {S,L} *)
2630      termsWithJ = {#[[1]], J} & /@ termsWithJ;
2631      (* This is just a quick way of including up all the MJ values *)
2632      Return[Flatten /@ Tuples[{termsWithJ, MJs}]]
2633      )
2634      ]
2635
2636    AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J,
       1].";
2637    AllowedMforJ[J_] := Range[-J, J, 1];
2638
2639    AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta J
        that appear in the f^numE configuration.";
2640    AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}];
2641
2642    Seniority::usage="Seniority[LS] returns the seniority of the given
       term."
2643    Seniority[LS_] := FindNKLSTerm[LS][[1, 2]]
2644
2645    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}.";
2646  FindNKLSTerm[SL_] := Module[
2647    {NKterms, n},
2648    n = 7;
2649    NKterms = {{}};
2650    Map[
2651      If[! StringFreeQ[First[#], SL],
2652        If[ToExpression[Part[#, 2]] <= n,
2653          NKterms = Join[NKterms, {#}, 1]
2654        ]
2655      ] &,
2656    fnTermLabels
2657    ];
2658    NKterms = DeleteCases[NKterms, {}];
2659    NKterms]
2660
2661  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}.";
2662  Options[ParseTermLabels] = {"Export" -> True};
2663  ParseTermLabels[OptionsPattern[]] := Module[
2664    {labelsTextData, fNtextLabels, nielsonKosterLabels, seniorities,
    RacahW, RacahU},
2665  (
2666    labelsTextData = FileNameJoin[{moduleDir, "data", "
    NielsonKosterLabels_f6_f7.txt"}];
2667    fNtextLabels   = Import[labelsTextData];
2668    nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
2669    termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
2670    seniorities = Map[ToExpression[Part[# , {2}]] &,
    nielsonKosterLabels];
2671    racahW =
2672      Map[
2673        StringTake[
2674          Flatten[StringCases[Part[# , {3}],
2675            "(" ~~ DigitCharacter ~~ DigitCharacter ~~ DigitCharacter
    ~~ ")"]],
2676          {2, 4}
2677        ] &,
2678      nielsonKosterLabels];
2679    racahU =
2680      Map[
2681        StringTake[
2682          Flatten[StringCases[Part[# , {3}],
2683            "(" ~~ DigitCharacter ~~ DigitCharacter ~~ ")"]],
2684          {2, 3}
2685        ] &,
2686      nielsonKosterLabels];
2687    fnTermLabels = Join[termLabels, seniorities, racahW, racahU, 2];
2688    fnTermLabels = Sort[fnTermLabels];
2689    If[OptionValue["Export"],
2690      (
2691        broadFname = FileNameJoin[{moduleDir,"data","fnTerms.m"}];
```

```
2692          Export[broadFname, fnTermLabels];
2693        )
2694      ];
2695      Return[fnTermLabels];
2696    )
2697    ]
2698
2699    (* ###################### Term management ###################### *)
2700    (* ############################################################ *)
2701
2702    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.";
2703    Options[LoadParameters] = {
2704        "Source"->"Carnall",
2705        "Free Ion"->False,
2706        "gs"->2.002319304386
2707        };
2708    LoadParameters[Ln_String, OptionsPattern[]]:=
2709      Module[{source, params},
2710      (
2711        source = OptionValue["Source"];
2712        params = Which[source=="Carnall",
2713                (Association[Carnall["data"][Ln]])
2714                ];
2715        (*If a free ion then all the parameters from the crystal field
        are set to zero*)
2716        If[OptionValue["Free Ion"],
2717          Do[params[cfSymbol] = 0,
2718          {cfSymbol, cfSymbols}
2719          ]
2720        ];
2721        params[F0] = 0;
2722        params[M2] = 0.56 * params[M0]; (* See Carnall 1989, Table I,
        caption, probably fixed based on HF values*)
2723        params[M4] = 0.31 * params[M0]; (* See Carnall 1989, Table I,
        caption, probably fixed based on HF values*)
2724        params[P0] = 0;
2725        params[P4] = 0.5 * params[P2];  (* See Carnall 1989, Table I,
        caption, probably fixed based on HF values*)
2726        params[P6] = 0.1 * params[P2];  (* See Carnall 1989, Table I,
        caption, probably fixed based on HF values*)
2727        params[gs] = OptionValue["gs"];
2728        {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[F0
        ], params[F2], params[F4], params[F6]];
2729        params[E0] = 0;
2730        Return[params];
2731      )
2732    ];
2733
```

```
2734  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.
        ";
2735  HoleElectronConjugation[params_] :=
2736    Module[{newparams = params},
2737      (
2738        flipSignsOf = {ζ, T2, T3, T4, T6, T7, T8};
2739        flipSignsOf = Join[flipSignsOf, cfSymbols];
2740        flipped =
2741          Table[(flipper -> - newparams[flipper]),
2742          {flipper, flipSignsOf}
2743          ];
2744        nonflipped =
2745          Table[(flipper -> newparams[flipper]),
2746          {flipper, Complement[Keys[newparams], flipSignsOf]}
2747          ];
2748        flippedParams = Association[Join[nonflipped, flipped]];
2749        flippedParams = Select[flippedParams, FreeQ[#, Missing]&];
2750        Return[flippedParams];
2751      )
2752    ]
2753
2754  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.
2755  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.
2756  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\".
2757  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.
2758  Tha ordered basis in which this eigenvector is to be interpreted is
        the one corresponding to BasisLSJMJ[numE].
2759  The function admits the following options:
2760  \"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.
2761  \"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.
2762  \"Zeroes\" (list) : A list with symbols assumed to be zero.
2763  ";
2764  Options[IonSolver] = {"Include Spin-Spin" -> True,
```

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

```
{{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.";
ShiftedLevels[originalLevels_] :=
  Module[{groundEnergy, shifted},
    groundEnergy = Sort[originalLevels][[1,1]];
    shifted      = Map[{#[[1]] - groundEnergy, #[[2]]} &,
  originalLevels];
    Return[shifted];
  ]

(* ############################################################ *)
(* ################## Eigensystem analysis ################### *)

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

```

```
2864  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=...>."
2865  BasisVecInLSJMJ[basisVec_] := (
2866    {LSstring, Jval, mJval} = basisVec;
2867    LSJMJTemplate[<|
2868      "LS" -> LSstring,
2869      "J" -> ToString[Jval, InputForm],
2870      "mJ" -> ToString[mJval, InputForm]|>]
2871    );
2872
2873  ParseStates::usage = "ParseStates[states, basis] takes a list of
        eigenstates in terms of their coefficients in the given basis and
        returns a list of the same states in terms of their energy, LSJMJ
        symbol, J, mJ, S, L, LSJ symbol, and LS symbol. The LS symbol
        returned corresponds to the term with the largest coefficient in
        the given basis.";
2874  ParseStates[states_, basis_, OptionsPattern[]] := Module[{
        parsedStates},
2875  (
2876    parsedStates = Table[(
2877      {energy, eigenVec}      = state;
2878      maxTermIndex            = Ordering[Abs[eigenVec]][[-1]];
2879      {LSstring, Jval, mJval} = basis[[maxTermIndex]];
2880      LSJsymbol               = Subscript[LSstring, {Jval, mJval}];
2881      LSJMJsymbol             = LSstring <> ToString[Jval, InputForm];
2882      {S, L}                  = FindSL[LSstring];
2883      {energy, LSstring, Jval, mJval, S, L, LSJsymbol, LSJMJsymbol}
2884    ),
2885    {state, states}
2886    ];
2887    Return[parsedStates]
2888    )
2889  ]
2890
2891  ParseStatesByNumBasisVecs::usage = "ParseStatesByNumBasisVecs[states,
         basis, numBasisVecs, roundTo] takes a list of eigenstates in terms
         of their coefficients in the given basis and returns a list of the
         same states in terms of their energy and the coefficients at most
        numBasisVecs basis vectors. By default roundTo is 0.01 and this is
        the value used to round the amplitude coefficients.
2892  The option \"Coefficients\" can be used to specify whether the
        coefficients are given as \"Amplitudes\" or \"Probabilities\". The
        default is \"Amplitudes\".
2893  ";
2894  Options[ParseStatesByNumBasisVecs] = {"Coefficients" -> "Amplitudes",
         "Representation" -> "Ket"};
2895  ParseStatesByNumBasisVecs[eigensys_List, basis_List,
        numBasisVecs_Integer, roundTo_Real : 0.01, OptionsPattern[]] :=
        Module[
2896    {parsedStates, energy, eigenVec,
2897    probs, amplitudes, ordering,
2898    chosenIndices, majorComponents,
2899    majorAmplitudes, majorRep},
```

```
(
  parsedStates = Table[(
    {energy, eigenVec} = state;
    energy              = Chop[energy];
    probs               = Round[Abs[eigenVec^2], roundTo];
    amplitudes          = Round[eigenVec, roundTo];
    ordering            = Ordering[probs];
    chosenIndices       = ordering[[-numBasisVecs ;;]];
    majorComponents     = basis[[chosenIndices]];
    majorThings = If[OptionValue["Coefficients"] == "Probabilities",
      (
        probs[[chosenIndices]]
      ),
      (
        amplitudes[[chosenIndices]]
      )
    ];
    majorComponents     = PrettySaundersSLJmJ[#, "Representation" ->
  OptionValue["Representation"]] & /@ majorComponents;
    nonZ                = (# != 0.) & /@ majorThings;
    majorThings         = Pick[majorThings, nonZ];
    majorComponents     = Pick[majorComponents, nonZ];
    If[OptionValue["Coefficients"] == "Probabilities",
      (
        majorThings = majorThings * 100*"%"
      )
    ];
    majorRep            = majorThings . majorComponents;
    {energy, majorRep}
    ),
    {state, eigensys}];
  Return[parsedStates]
  )
];

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

ParseStateByProbabilitySum[{energy_, eigenVec_}, probSum_, roundTo_
  :0.01, maxParts_:20] := Compile[
  {{energy, _Real, 0},{eigenVec, _Complex, 1}, {probSum, _Real, 0}, {
  roundTo, _Real, 0}, {maxParts, _Integer, 0}},
  Module[
  {numStates, state, amplitudes, probs, ordering,
```

```
2948      orderedProbs, truncationIndex, accProb, thresholdIndex,
          chosenIndices, majorComponents,
2949      majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
2950    (
2951      numStates     = Length[eigenVec];
2952      (*Round them up*)
2953      amplitudes          = Round[eigenVec, roundTo];
2954      probs               = Round[Abs[eigenVec^2], roundTo];
2955      ordering            = Reverse[Ordering[probs]];
2956      (*Order the probabilities from high to low*)
2957      orderedProbs        = probs[[ordering]];
2958      (*To speed up Accumulate, assume that only as much as maxParts will
          be needed*)
2959      truncationIndex     = Min[maxParts, Length[orderedProbs]];
2960      orderedProbs        = orderedProbs[[;;truncationIndex]];
2961      (*Accumulate the probabilities*)
2962      accProb             = Accumulate[orderedProbs];
2963      (*Find the index of the first element in accProb that is greater
          than probSum*)
2964      thresholdIndex      = Min[Length[accProb], FindThresholdPosition[
          accProb, probSum]];
2965      (*Grab all the indicees up till that one*)
2966      chosenIndices       = ordering[[;; thresholdIndex]];
2967      (*Select the corresponding elements from the basis*)
2968      majorComponents     = basis[[chosenIndices]];
2969      (*Select the corresponding amplitudes*)
2970      majorAmplitudes     = amplitudes[[chosenIndices]];
2971      (*Take their absolute value*)
2972      absMajorAmplitudes = Abs[majorAmplitudes];
2973      (*Make sure that there are no effectively zero contributions*)
2974      notnullAmplitudes  = Flatten[Position[absMajorAmplitudes, x_ /; x
          != 0]];
2975      (* majorComponents     = PrettySaundersSLJmJ[{#[[1]],#[[2]],#[[3]]}]
          & /@ majorComponents; *)
2976      majorComponents     = PrettySaundersSLJmJ /@ majorComponents;
2977      majorAmplitudes     = majorAmplitudes[[notnullAmplitudes]];
2978      (*Make them into Kets*)
2979      majorComponents     = Ket /@ majorComponents[[notnullAmplitudes]];
2980      (*Multiply and add to build the final Ket*)
2981      majorRep            = majorAmplitudes . majorComponents;
2982        );
2983    Return[{energy, majorRep}]
2984    ],
2985      CompilationTarget -> "C",
2986      RuntimeAttributes -> {Listable},
2987      Parallelization -> True,
2988      RuntimeOptions -> "Speed"
2989    ];
2990
2991    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.";
2992    ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ :
```

```
    0.01, maxParts_: 20] := Module[
2993    {parsedByProb, numStates, state, energy, eigenVec, amplitudes,
    probs, ordering,
2994     orderedProbs, truncationIndex, accProb, thresholdIndex,
    chosenIndices, majorComponents,
2995     majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
2996   (
2997     numStates     = Length[eigensys];
2998     parsedByProb = Table[(
2999       state               = eigensys[[idx]];
3000       {energy, eigenVec} = state;
3001       (*Round them up*)
3002       amplitudes          = Round[eigenVec, roundTo];
3003       probs               = Round[Abs[eigenVec^2], roundTo];
3004       ordering            = Reverse[Ordering[probs]];
3005       (*Order the probabilities from high to low*)
3006       orderedProbs        = probs[[ordering]];
3007       (*To speed up Accumulate, assume that only as much as maxParts
    will be needed*)
3008       truncationIndex     = Min[maxParts, Length[orderedProbs]];
3009       orderedProbs        = orderedProbs[[;;truncationIndex]];
3010       (*Accumulate the probabilities*)
3011       accProb             = Accumulate[orderedProbs];
3012       (*Find the index of the first element in accProb that is greater
    than probSum*)
3013       thresholdIndex      = Min[Length[accProb], FindThresholdPosition[
    accProb, probSum]];
3014       (*Grab all the indicees up till that one*)
3015       chosenIndices       = ordering[[;; thresholdIndex]];
3016       (*Select the corresponding elements from the basis*)
3017       majorComponents     = basis[[chosenIndices]];
3018       (*Select the corresponding amplitudes*)
3019       majorAmplitudes     = amplitudes[[chosenIndices]];
3020       (*Take their absolute value*)
3021       absMajorAmplitudes = Abs[majorAmplitudes];
3022       (*Make sure that there are no effectively zero contributions*)
3023       notnullAmplitudes  = Flatten[Position[absMajorAmplitudes, x_ /; x
     != 0]];
3024       (* majorComponents     = PrettySaundersSLJmJ
    [{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents; *)
3025       majorComponents     = PrettySaundersSLJmJ /@ majorComponents;
3026       majorAmplitudes     = majorAmplitudes[[notnullAmplitudes]];
3027       majorComponents     = majorComponents[[notnullAmplitudes]];
3028       (*Multiply and add to build the final Ket*)
3029       majorRep            = majorAmplitudes . majorComponents;
3030       {energy, majorRep}
3031       ), {idx, numStates}];
3032   Return[parsedByProb]
3033   )
3034   ];
3035
3036
3037   (* ################## Eigensystem analysis ################## *)
3038   (* ########################################################### *)
3039
```

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

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.";
SymbToNum[expr_, replacementAssociation_]:= (
  includedKeys = Keys[replacementAssociation];
  (*If a key is not defined, make its value zero.*)
  fullAssociation = Table[(
    If[MemberQ[includedKeys, key],
      ToExpression[key]->replacementAssociation[key],
      ToExpression[key]->0
    ]
  ),
  {key, paramSymbols}];
  Return[expr/.fullAssociation];
)

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.";
SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;

ExportMZip::usage="ExportMZip[\"dest.[zip,m]\"] saves a compressed
  version of expr to the given destination.";
ExportMZip[filename_, expr_]:=Module[{baseName, exportName,
  mImportName, zipImportName},
(
  baseName    = FileBaseName[filename];
  exportName  = StringReplace[filename,".m"->".zip"];
  mImportName = StringReplace[exportName,".zip"->".m"];
  If[FileExistsQ[mImportName],
  (
    PrintTemporary[mImportName<>" exists already, deleting"];
    DeleteFile[mImportName];
    Pause[2];
  )
  ];
  Export[exportName, (baseName<>".m") -> expr]
)
];

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";
Options[ImportMZip]={"Leave Uncompressed" -> True};
```

```
ImportMZip[filename_String, OptionsPattern[]] := Module[
  {baseName, importKey, zipImportName, mImportName, imported},
(
  baseName      = FileBaseName[filename];
  (*Function allows for the filename to be .m or .zip*)
  importKey     = baseName <> ".m";
  zipImportName = StringReplace[filename, ".m"->".zip"];
  mImportName   = StringReplace[zipImportName, ".zip"->".m"];
  If[FileExistsQ[mImportName],
  (
    PrintTemporary[".m version exists already, importing that instead
   ..."];
    Return[Import[mImportName]];
  )
  ];
  imported = Import[zipImportName, importKey];
  If[OptionValue["Leave Uncompressed"],
    Export[mImportName, imported]
  ];
  Return[imported]
)
];

ReplaceInSparseArray::usage = "ReplaceInSparseArray[sparseArray,
  rules] takes a sparse array that may contain symbolic quantities
  and returns a sparse array in which the given replacement rules
  have been used.";
ReplaceInSparseArray[s_SparseArray, rule_] := (With[{
    elem = s["NonzeroValues"]/.rule,
    def  = s["Background"]/.rule
    },
    srep = SparseArray[Automatic,
      s["Dimensions"],
      def,
      {1, {s["RowPointers"], s["ColumnIndices"]}, elem}
      ];
  ];
  Return[srep];
  );

MapToSparseArray::usage = "MapToSparseArray[sparseArray, function]
  takes a sparse array and returns a sparse array after the function
  has been applied to it.";
MapToSparseArray[sparsey_SparseArray, func_] := Module[{
    nonZ, backg, mapped
    },
   (
    nonZ   = func/@ sparsey["NonzeroValues"];
    backg  = func[sparsey["Background"]];
    mapped = SparseArray[Automatic,
      sparsey["Dimensions"],
      backg,
      {1, {sparsey["RowPointers"], sparsey["ColumnIndices"]}, nonZ}
      ];
    Return[mapped];
```

```
3128      )
3129     ];
3130
3131   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.";
3132   Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
3133   ParseTeXLikeSymbol[bigString_, OptionsPattern[]] := (
3134     form = OptionValue["Form"];
3135     (*parse greek*)
3136     symbols = Table[(
3137         str = StringReplace[string, {"\\alpha" -> "α",
3138           "\\beta" -> "β",
3139           "\\gamma" -> "γ",
3140           "\\psi" -> "\[Psi]"}];
3141         symbol = Which[
3142           StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"]],
3143           (
3144           (*yes sub no sup*)
3145           mainSymbol = StringSplit[str, "_"][[1]];
3146           mainSymbol = ToExpression[mainSymbol];
3147
3148           subPart =
3149             StringCases[str,
3150               RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
3151           Subscript[mainSymbol, subPart]
3152           ),
3153           Not[StringContainsQ[str, "_"]] && StringContainsQ[str, "^"],
3154           (
3155           (*no sub yes sup*)
3156           mainSymbol = StringSplit[str, "^"][[1]];
3157           mainSymbol = ToExpression[mainSymbol];
3158
3159           supPart =
3160             StringCases[str,
3161               RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
3162           Superscript[mainSymbol, supPart]),
3163           StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
3164           (
3165           (*yes sub yes sup*)
3166           mainSymbol = StringSplit[str, "_"][[1]];
3167           mainSymbol = ToExpression[mainSymbol];
3168           {subPart, supPart} =
3169             StringCases[str, RegularExpression@"\\{(.*?)\\}" -> "$1"];
3170           Subsuperscript[mainSymbol, subPart, supPart]
3171           ),
3172           True,
3173           ((*no sup or sub*)
3174           str)
3175           ];
3176         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}},
    FrameTicks   -> {{None, None}, {Automatic, Automatic}},
    FrameStyle   -> Directive[15, Dashed, Thin],
    PlotLabel    -> Style[OptionValue["Title"], 15, Bold],
    Background   -> OptionValue["Background"],
    FrameLabel   -> {"\!\(\*FractionBox[\(E\), SuperscriptBox[\(cm\),
   \(-1\)]]\)"},
    Epilog       -> epi]
)

ExploreGraphics::usage =
  "Pass a Graphics object to explore it. Zoom by clicking and
```

```
    dragging a rectangle. Pan by clicking and dragging while pressing
    Ctrl. Click twice to reset view.
3227    Based on ZeitPolizei @ https://mathematica.stackexchange.com/
    questions/7142/how-to-manipulate-2d-plots";
3228
3229  OptAxesRedraw::usage =
3230    "Option for ExploreGraphics to specify redrawing of axes. Default
    False.";
3231  Options[ExploreGraphics] = {OptAxesRedraw -> False};
3232
3233  ExploreGraphics[graph_Graphics, opts : OptionsPattern[]] := With[
3234    {gr  = First[graph],
3235      opt = DeleteCases[Options[graph],
3236            PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _],
3237      plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
3238      ar  = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
3239      ao  = AbsoluteOptions[AxesOrigin],
3240      rectangle = {Dashing[Small],
3241        Line[{#1,
3242              {First[#2], Last[#1]},
3243              #2,
3244              {First[#1], Last[#2]},
3245              #1}]} &,
3246      optAxesRedraw = OptionValue[OptAxesRedraw]},
3247    DynamicModule[
3248        {dragging=False, first, second, rx1, rx2, ry1, ry2,
3249        range = plr},
3250        {{rx1, rx2}, {ry1, ry2}} = plr;
3251      Panel@
3252      EventHandler[
3253        Dynamic@Graphics[
3254          If[dragging, {gr, rectangle[first, second]}, gr],
3255          PlotRange -> Dynamic@range,
3256          AspectRatio -> ar,
3257          AxesOrigin -> If[optAxesRedraw,
3258            Dynamic@Mean[range\[Transpose]], ao],
3259          Sequence @@ opt],
3260        {{"MouseDown", 1} :> (
3261          first = MousePosition["Graphics"]
3262          ),
3263        {"MouseDragged", 1} :> (
3264          dragging = True;
3265          second = MousePosition["Graphics"]
3266          ),
3267        "MouseClicked" :> (
3268          If[CurrentValue@"MouseClickCount"==2,
3269            range = plr];
3270          ),
3271        {"MouseUp", 1} :> If[dragging,
3272          dragging = False;
3273
3274          range = {{rx1, rx2}, {ry1, ry2}} =
3275            Transpose@{first, second};
3276          range[[2]] = {0, 1}],
3277        {"MouseDown", 2} :> (
```

```
3278            first = {sx1, sy1} = MousePosition["Graphics"]
3279            ),
3280          {"MouseDragged", 2} :> (
3281            second = {sx2, sy2} = MousePosition["Graphics"];
3282            rx1 = rx1 - (sx2 - sx1);
3283            rx2 = rx2 - (sx2 - sx1);
3284            ry1 = ry1 - (sy2 - sy1);
3285            ry2 = ry2 - (sy2 - sy1);
3286            range = {{rx1, rx2}, {ry1, ry2}};
3287            range[[2]] = {0, 1};
3288            )}]]];
3289
3290  Options[LabeledGrid]={
3291      ItemSize->Automatic,
3292      Alignment->Center,
3293      Frame->All,
3294      "Separator"->",",
3295      "Pivot"->""
3296  };
3297  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.";
3298  LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]]:=
      Module[
3299     {gridList=data,rowHeads=rowHeaders,colHeads=columnHeaders},
3300     (
3301     separator=OptionValue["Separator"];
3302     pivot=OptionValue["Pivot"];
3303     gridList=Table[
3304             Tooltip[
3305               data[[rowIdx,colIdx]],
3306               DisplayForm[
3307                 RowBox[{rowHeads[[rowIdx]],
3308                         separator,
3309                         colHeads[[colIdx]]}
3310                       ]
3311                       ]
3312                     ],
3313         {rowIdx,Dimensions[data][[1]]},
3314         {colIdx,Dimensions[data][[2]]}];
3315     gridList=Transpose[Prepend[gridList,colHeads]];
3316     rowHeads=Prepend[rowHeads,pivot];
3317     gridList=Prepend[gridList,rowHeads]//Transpose;
3318     Grid[gridList,
3319         Frame->OptionValue[Frame],
3320         Alignment->OptionValue[Alignment],
3321         Frame->OptionValue[Frame],
3322         ItemSize->OptionValue[ItemSize]
3323         ]
3324  )
3325  ]
3326
```

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

Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover" -> True, "Overlay Values" -> True}];
HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts : OptionsPattern[]] := (
 braLabels = DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]] & /@ basisLabels;
 ketLabels = DisplayForm[Rotate[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"}],\[Pi]/2]] & /@ basisLabels;
 ketLabelsUpright = DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"}]] & /@ basisLabels;
 numRows = Length[hamMatrix];
 numCols = Length[hamMatrix[[1]]];
 epiThings = Which[
   And[OptionValue["Hover"], Not[OptionValue["Overlay Values"]]],
   Flatten[
     Table[
       Tooltip[
         {
           Transparent,
           Rectangle[
           {j - 1, numRows - i},
           {j - 1, numRows - i} + {1, 1}
           ]
         },
       Row[{braLabels[[i]],ketLabelsUpright[[j]],"=",hamMatrix[[i, j]]}]
       ],
     {i, 1, numRows},
     {j, 1, numCols}
     ]
   ],
   And[OptionValue["Hover"], OptionValue["Overlay Values"]],
   Flatten[
     Table[
       Tooltip[
         {
           Transparent,
           Rectangle[
           {j - 1, numRows - i},
```

```
3368                {j - 1, numRows - i} + {1, 1}
3369                ]
3370              },
3371            DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i]],
    "\[LeftBracketingBar]", basisLabels[[j]], "\[RightAngleBracket]"}]]
3372            ],
3373          {i, numRows},
3374          {j, numCols}
3375          ]
3376        ],
3377        True,
3378        {}
3379        ];
3380      textOverlay = If[OptionValue["Overlay Values"],
3381        (
3382          Flatten[
3383          Table[
3384            Text[hamMatrix[[i, j]],
3385              {j - 1/2, numRows - i + 1/2}
3386            ],
3387          {i, 1, numRows},
3388          {j, 1, numCols}
3389          ]
3390          ]
3391        ),
3392        {}
3393        ];
3394      epiThings = Join[epiThings, textOverlay];
3395      MatrixPlot[hamMatrix,
3396        FrameTicks -> {
3397          {Transpose[{Range[Length[braLabels]], braLabels}], None},
3398          {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}
3399          },
3400        Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
3401        Epilog -> epiThings
3402      ]
3403      );
3404
3405    (* ################## Some Plotting Routines ################## *)
3406    (* ############################################################ *)
3407
3408    (* ############################################################ *)
3409    (* ###################### Load Functions ###################### *)
3410
3411    LoadAll::usage="LoadAll[] executes all Load* functions.";
3412    LoadAll[]:=(
3413        LoadTermLabels[];
3414        LoadCFP[];
3415        LoadUk[];
3416        LoadV1k[];
3417        LoadT22[];
3418        LoadSOOandECSOLS[];
3419
3420        LoadElectrostatic[];
3421        LoadSpinOrbit[];
```

```
3422        LoadSOOandECSO[];
3423        LoadSpinSpin[];
3424        LoadThreeBody[];
3425        LoadChenDeltas[];
3426        LoadCarnall[];
3427      )
3428
3429    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.";
3430
3431    LoadTermLabels::usage="LoadTermLabels[] loads into the session the
          labels for the terms in the f^n configurations.";
3432    LoadTermLabels[]:= (
3433      If[ValueQ[fnTermLabels], Return[]];
3434      PrintTemporary["Loading data for state labels in the f^n
          configurations..."];
3435      fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
3436
3437      If[!FileExistsQ[fnTermsFname],
3438        (PrintTemporary[">> fnTerms.m not found, generating ..."];
3439          fnTermLabels = ParseTermLabels["Export"->True];
3440        ),
3441        fnTermLabels = Import[fnTermsFname];
3442      ];
3443    )
3444
3445
3446    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}";
3447
3448    LoadCarnall::usage="LoadCarnall[] loads data for trivalent
          lanthanides in LaF3 using the data from Bill Carnall's 1989 paper."
          ;
3449    LoadCarnall[]:=(
3450      If[ValueQ[Carnall], Return[]];
3451      carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
3452      If[!FileExistsQ[carnallFname],
3453        (PrintTemporary[">> Carnall.m not found, generating ..."];
3454          Carnall = ParseCarnall[];
3455        ),
3456        Carnall = Import[carnallFname];
3457      ];
3458    )
3459
3460    LoadChenDeltas::usage="LoadChenDeltas[] loads the differences noted
          by Chen.";
3461    LoadChenDeltas[]:=(
3462      If[ValueQ[chenDeltas], Return[]];
3463      PrintTemporary["Loading the association of discrepancies found by
          Chen ..."];
```

```
    chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"
   }];
    If[!FileExistsQ[chenDeltasFname],
      (PrintTemporary[">> chenDeltas.m not found, generating ..."];
        chenDeltas = ParseChenDeltas[];
      ),
      chenDeltas = Import[chenDeltasFname];
    ];
  );

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

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

  ParseCarnall::usage="ParseCarnall[] parses the data found in ./data/
    Carnall.xls. If the option \"Export\" is set to True (True is the
```

```
    default), then the parsed data is saved to ./data/Carnall. This
    data is from the tables and appendices of Carnall et al (1989).";
3510  Options[ParseCarnall] = {"Export" -> True};
3511  ParseCarnall[] := (
3512    ions          = {"Pr","Nd","Pm","Sm","Eu","Gd","Tb","Dy","Ho","Er","
      Tm"};
3513    templates     = StringTemplate/@StringSplit["appendix:`ion`:
      Association appendix:`ion`:Calculated appendix:`ion`:RawTable
      appendix:`ion`:Headings"," "];
3514
3515    (* How many unique eigenvalues, after removing Kramer's degeneracy
      *)
3516    fullSizes     = AssociationThread[ions, {91, 182, 1001, 1001, 3003,
      1716, 3003, 1001, 1001, 182, 91}];
3517    carnall       = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"
      }]][[2]];
3518    carnallErr    = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"
      }]][[3]];
3519
3520    elementNames = carnall[[1]][[2;;]];
3521    carnall       = carnall[[2;;]];
3522    carnallErr    = carnallErr[[2;;]];
3523    carnall       = Transpose[carnall];
3524    carnallErr    = Transpose[carnallErr];
3525    paramNames    = ToExpression/@carnall[[1]][[1;;]];
3526    carnall       = carnall[[2;;]];
3527    carnallErr    = carnallErr[[2;;]];
3528    carnallData   = Table[(
3529                     data          = carnall[[i]];
3530                     data          = (#[[1]]->#[[2]])&/@Select[Transpose
      [{paramNames,data}],#[[2]]!=""&];
3531                     elementNames[[i]]->data
3532                     ),
3533                     {i,1,13}
3534                     ];
3535    carnallData   = Association[carnallData];
3536    carnallNotes = Table[(
3537                     data         = carnallErr[[i]];
3538                     elementName = elementNames[[i]];
3539                     dataFun      = (
3540                         #[[1]] -> If[#[[2]]=="[]",
3541                         "Not allowed to vary in fitting.",
3542                         If[#[[2]]=="[R]",
3543                             "Ratio constrained by: " <> <|"Eu"->"F4/F2
      =0.713; F6/F2=0.512",
3544                                 "Gd"->"F4/F2=0.710]",
3545                                 "Tb"->"F4/F2=0.707"|>[elementName],
3546                             If[#[[2]]=="i",
3547                                 "Interpolated",
3548                                 #[[2]]
3549                             ]
3550                         ]
3551                         ]) &;
3552                     data = dataFun /@ Select[Transpose[{paramNames,data
      }],#[[2]]!=""&];
```

```
                    elementName->data
                      ),
                  {i,1,13}
                  ];
    carnallNotes  = Association[carnallNotes];

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

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

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

```
3604          Carnall[keys[[4]]]  = headers;
3605          ),
3606      {i,4,14}
3607      ];
3608
3609      goodions = Select[ions,#!="Pm"&];
3610      expData  = Select[Transpose[Carnall["appendix:"<>#<>":RawTable"
          ]][[1+Position[Carnall["appendix:"<>#<>":Headings"],"Exp (1/cm)"
          ][[1,1]]]],NumberQ]&/@goodions;
3611      Carnall["All Experimental Data"]=AssociationThread[goodions,expData
          ];
3612      If[OptionValue["Export"],
3613        (
3614          carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
3615          Print["Exporting to "<>exportFname];
3616          Export[carnallFname, Carnall];
3617        )
3618        ];
3619      Return[Carnall];
3620    )
3621
3622    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";
3623
3624    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[].";
3625
3626    LoadCFP::usage="LoadCFP[] loads CFP, CFPAssoc, and CFPTable into the
        session.";
3627    LoadCFP[]:=(
3628      If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]],Return[]];
3629
3630      PrintTemporary["Loading CFPtable ..."];
3631      CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
3632      If[!FileExistsQ[CFPTablefname],
3633        (PrintTemporary[">> CFPTable.m not found, generating ..."];
3634          CFPTable = GenerateCFPTable["Export"->True];
3635        ),
3636        CFPTable = Import[CFPTablefname];
3637      ];
3638
3639      PrintTemporary["Loading CFPs.m ..."];
3640      CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
3641      If[!FileExistsQ[CFPfname],
3642        (PrintTemporary[">> CFPs.m not found, generating ..."];
3643          CFP = GenerateCFP["Export"->True];
3644        ),
3645        CFP = Import[CFPfname];
```

```
3646      ];
3647
3648      PrintTemporary["Loading CFPAssoc.m ..."];
3649      CFPAfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
3650      If[!FileExistsQ[CFPAfname],
3651        (PrintTemporary[">> CFPAssoc.m not found, generating ..."];
3652          CFPAssoc = GenerateCFPAssoc["Export"->True];
3653        ),
3654        CFPAssoc = Import[CFPAfname];
3655      ];
3656    );
3657
3658    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[].";
3659
3660    LoadUk::usage="LoadUk[] loads into session the reduced matrix
       elements for unit tensor operators.";
3661    LoadUk[]:=(
3662      If[ValueQ[ReducedUkTable], Return[]];
3663      PrintTemporary["Loading the association of reduced matrix elements
       for unit tensor operators ..."];
3664      ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "
       ReducedUkTable.m"}];
3665      If[!FileExistsQ[ReducedUkTableFname],
3666        (PrintTemporary[">> ReducedUkTable.m not found, generating ..."];
3667          ReducedUkTable = GenerateReducedUkTable[7];
3668        ),
3669        ReducedUkTable = Import[ReducedUkTableFname];
3670      ];
3671    );
3672
3673    ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}]
       provides the calculated result of Electrostatic[{n, SL, SpLp}].
       Load using LoadElectrostatic[].";
3674
3675    LoadElectrostatic::usage="LoadElectrostatic[] loads the reduced
       matrix elements for the electrostatic interaction.";
3676    LoadElectrostatic[]:=(
3677      If[ValueQ[ElectrostaticTable], Return[]];
3678      PrintTemporary["Loading the association of matrix elements for the
       electrostatic interaction ..."];
3679      ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "
       ElectrostaticTable.m"}];
3680      If[!FileExistsQ[ElectrostaticTablefname],
3681        (PrintTemporary[">> ElectrostaticTable.m not found, generating
       ..."];
3682          ElectrostaticTable = GenerateElectrostaticTable[7];
3683        ),
3684        ElectrostaticTable = Import[ElectrostaticTablefname];
3685      ];
3686    );
3687
3688    LoadV1k::usage="LoadV1k[] loads into session the matrix elements of
```

```
  V1k.";
LoadV1k[]:=(
  If[ValueQ[ReducedV1kTable], Return[]];
  PrintTemporary["Loading the association of matrix elements for V1k
  ..."];
  ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "
  ReducedV1kTable.m"}];
  If[!FileExistsQ[ReducedV1kTableFname],
    (PrintTemporary[">> ReducedV1kTable.m not found, generating ..."
  ];
      ReducedV1kTable = GenerateReducedV1kTable[7];
    ),
    ReducedV1kTable = Import[ReducedV1kTableFname];
  ]
);

LoadSpinOrbit::usage="LoadSpinOrbit[] loads into session the matrix
  elements of the spin-orbit interaction.";
LoadSpinOrbit[]:=(
  If[ValueQ[SpinOrbitTable], Return[]];
  PrintTemporary["Loading the association of matrix elements for spin
  -orbit ..."];
  SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "
  SpinOrbitTable.m"}];
  If[!FileExistsQ[SpinOrbitTableFname],
    (PrintTemporary[">> SpinOrbitTable.m not found, generating ..."];
      SpinOrbitTable = GenerateSpinOrbitTable[7, "Export" -> True];
    ),
    SpinOrbitTable = Import[SpinOrbitTableFname];
  ]
);

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 ..."];
3731    SOOandECSOTableFname = FileNameJoin[{moduleDir, "data", "
    SOOandECSOTable.m"}];
3732    If[!FileExistsQ[SOOandECSOTableFname],
3733      (PrintTemporary[">> SOOandECSOTable.m not found, generating ..."
    ];
3734        SOOandECSOTable = GenerateSOOandECSOTable[7, "Export"->True];
3735      ),
3736      SOOandECSOTable = Import[SOOandECSOTableFname];
3737    ];
3738  );
3739
3740  LoadT22::usage="LoadT22[] loads into session the matrix elements of
    T22.";
3741  LoadT22[]:=(
3742    If[ValueQ[T22Table], Return[]];
3743    PrintTemporary["Loading the association of reduced T22 matrix
    elements ..."];
3744    T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m
    "}];
3745    If[!FileExistsQ[T22TableFname],
3746      (PrintTemporary[">> ReducedT22Table.m not found, generating ..."
    ];
3747        T22Table = GenerateT22Table[7];
3748      ),
3749      T22Table = Import[T22TableFname];
3750    ];
3751  );
3752
3753  LoadSpinSpin::usage="LoadSpinSpin[] loads into session the matrix
    elements of spin-spin.";
3754  LoadSpinSpin[]:=(
3755    If[ValueQ[SpinSpinTable], Return[]];
3756    PrintTemporary["Loading the association of matrix elements for spin
    -spin ..."];
3757    SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "
    SpinSpinTable.m"}];
3758    If[!FileExistsQ[SpinSpinTableFname],
3759      (PrintTemporary[">> SpinSpinTable.m not found, generating ..."];
3760        SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
3761      ),
3762      SpinSpinTable = Import[SpinSpinTableFname];
3763    ];
3764  );
3765
3766  LoadThreeBody::usage="LoadThreeBody[] loads into session the matrix
    elements of three-body configuration-interaction effects.";
3767  LoadThreeBody[]:=(
3768    If[ValueQ[ThreeBodyTable], Return[]];
3769    PrintTemporary["Loading the association of matrix elements for
    three-body configuration-interaction effects ..."];
3770    ThreeBodyFname    = FileNameJoin[{moduleDir, "data", "ThreeBodyTable
    .m"}];
3771    ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "
    ThreeBodyTables.m"}];
```

```
    If[!FileExistsQ[ThreeBodyFname],
      (PrintTemporary[">> ThreeBodyTable.m not found, generating ..."];
        {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTables[14,
     "Export" -> True];
      ),
      ThreeBodyTable = Import[ThreeBodyFname];
      ThreeBodyTables = Import[ThreeBodiesFname];
    ];
  );

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

End[]

LoadTermLabels[];
LoadCFP[];

EndPackage[]
```

11.2 qonstants.m

This file has a few constants and conversion factors.

```
BeginPackage["qonstants`"];

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

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

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

(* Hartree atomic units *)
hPlanckHartree = 2 \[Pi]; (* Planck's constant in Hartree *)
meHartree      = 1;       (* electron mass in Hartree *)
cLightHartree  = 137.036; (* speed of light in Hartree *)
```

```
eChargeHartree = 1;        (* elementary charge in Hartree *)
\[Mu]0Hartree  = αFine^2; (* magnetic permeability in vacuum in Hartree
    *)

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

EndPackage[];
```

11.3 qplotter.m

This module has a few useful plotting routines.

```

BeginPackage["qplotter`"];

GetColor;
IndexMappingPlot;
ListLabelPlot;
AutoGraphicsGrid;
SpectrumPlot;
WaveToRGB;

Begin["`Private`"];

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

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

```
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;]
60      ), {tick, fTicks}];
61    If[Or[Not[prevTickLabel === tickLabel], groupCounter > 1],
62    (
63      avgPosition = groupTally/groupCounter;
64      avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
65      )
66    ];
67    Return[avgTicks];)]
68
69  GetColor[s_Style] := s /. Style[_, c_] :> c
70  GetColor[_] := Black
71
72  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.";
73  Options[ListLabelPlot] = Append[Options[ListPlot], "TickCompression"
      ->True];
74  ListLabelPlot[data_, labels_, opts : OptionsPattern[]] := Module[
75    {uniqueLabels, pallete, groupedByTerm, groupedKeys, scatterGroups,
76    groupedColors, frameTicks, compTicks, bottomTicks, topTicks},
77    (
78    uniqueLabels  = DeleteDuplicates[labels];
79    pallete = Table[ColorData["Rainbow", i], {i, 0, 1,
80        1/(Length[uniqueLabels] - 1)}];
```

```
    uniqueLabels  = (#[[1]] -> #[[2]]) & /@ Transpose[{RandomSample[
    uniqueLabels], pallete}];
    uniqueLabels  = Association[uniqueLabels];
    groupedByTerm = GroupBy[Transpose[{labels, Range[Length[data]],
    data}], First];
    groupedKeys   = Keys[groupedByTerm];
    scatterGroups = Transpose[Transpose[#][[2 ;; 3]]] & /@ Values[
    groupedByTerm];
    groupedColors = uniqueLabels[#] & /@ groupedKeys;
    frameTicks    = {Transpose[{Range[Length[data]],
      Style[Rotate[#, 0], uniqueLabels[#]] & /@ labels}],
      Automatic};
      If[OptionValue["TickCompression"], (
          compTicks = TickCompressor[frameTicks[[1]]];
          bottomTicks =
              MapIndexed[
              If[EvenQ[First[#2]], {#1[[1]],
                  Tooltip[Style["\[SmallCircle]", GetColor
    [#1[[2]]]],#1[[2]]]
                  }, #1] &, compTicks];
          topTicks =
              MapIndexed[
              If[OddQ[First[#2]], {#1[[1]],
                  Tooltip[Style["\[SmallCircle]", GetColor
    [#1[[2]]]],#1[[2]]]
                  }, #1] &, compTicks];
          frameTicks = {{Automatic, Automatic}, {bottomTicks, topTicks
    }};)
    ];
    ListPlot[scatterGroups,
      opts,
      Frame->True,
      PlotStyle -> groupedColors,
      FrameTicks -> frameTicks]
    )
    ]

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

```
      R = ((-(wavelength - 440)/(440 - 380))*attenuation)^gamma;
      G = 0.0;
      B = (1.0*attenuation)^gamma;
      ),
    440 <= wavelength <= 490,
    (
      R = 0.0;
      G = ((wavelength - 440)/(490 - 440))^gamma;
      B = 1.0;
      ),
    490 <= wavelength <= 510,
    (
      R = 0.0;
      G = 1.0;
      B = (-(wavelength - 510)/(510 - 490))^gamma;
      ),
    510 <= wavelength <= 580,
    (
      R = ((wavelength - 510)/(580 - 510))^gamma;
      G = 1.0;
      B = 0.0;
      ),
    580 <= wavelength <= 645,
    (
      R = 1.0;
      G = (-(wavelength - 645)/(645 - 580))^gamma;
      B = 0.0;
      ),
    645 <= wavelength <= 750,
    (
      attenuation = 0.3 + 0.7*(750 - wavelength)/(750 - 645);
      R = (1.0*attenuation)^gamma;
      G = 0.0;
      B = 0.0;
      ),
    True,
    (
      R = 0;
      G = 0;
      B = 0;
      )];
    Return[RGBColor[R, G, B]]
    )

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

```
    nocolor = Directive[Opacity[0], color];
    ymax = ymin + height;
    intenseColor = Directive[Opacity[intensity], color];
    polys = {
      Polygon[{
        {xCenter - width/2, ymin},
        {xCenter, ymin},
        {xCenter, ymax},
        {xCenter - width/2, ymax}},
        VertexColors -> {
          nocolor,
          intenseColor,
          intenseColor,
          nocolor,
          nocolor}],
      Polygon[{
        {xCenter, ymin},
        {xCenter + width/2, ymin},
        {xCenter + width/2, ymax},
        {xCenter, ymax}},
        VertexColors -> {
          intenseColor,
          nocolor,
          nocolor,
          intenseColor,
          intenseColor}]
      };
    Return[polys]
    );
]

Options[SpectrumPlot] = Options[Graphics];
Options[SpectrumPlot] = Join[Options[SpectrumPlot], {"Intensities" ->
   {},"Tooltips" -> True, "Comments" -> {}, "SpectrumFunction" ->
  WaveToRGB}];
SpectrumPlot::usage="SpectrumPlot[lines, widthToHeightAspect,
  lineWidth] takes a list of spectral lines and creates a visual
  representation of them. The lines are represented as fuzzy
  rectangles with a width of lineWidth and a height that is
  determined by the overall condition that the with to height ratio
  of the resulting graph is widthToHeightAspect. The color of the
  lines is determined by the wavelength of the line. The function
  assumes that the lines are given in nm.
If the lineWidth parameter is a single number, then every line shares
   that width. If the lineWidth parameter is a list of numbers, then
  each line has a different width. The function returns a Graphics
  object. The function also accepts any options that Graphics accepts
  . The background of the plot is black by default. The plot range is
   set to the minimum and maximum wavelength of the given lines.
Besides the options for Graphics the function also admits the option
  Intensities. This option is a list of numbers that determines the
  intensity of each line. If the Intensities option is not given,
  then the lines are drawn with full intensity. If the Intensities
  option is given, then the lines are drawn with the given intensity.
   The intensity is a number between 0 and 1.
```

```
208   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.
209   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.
210   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.
211   ";
212   SpectrumPlot[lines_, widthToHeightAspect_, lineWidth_, opts :
        OptionsPattern[]] := Module[
213     {minWave, maxWave, height, fuzzyLines},
214   (
215     colorFun = OptionValue["SpectrumFunction"];
216     {minWave, maxWave} = MinMax[lines];
217     height     = (maxWave - minWave)/widthToHeightAspect;
218     fuzzyLines = Which[
219       NumberQ[lineWidth] && Length[OptionValue["Intensities"]] == 0,
220         FuzzyRectangle[#, lineWidth, 0, height, colorFun[#]] &/@ lines,
221       Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]] ==
        0,
222         MapThread[FuzzyRectangle[#1, #2, 0, height, colorFun[#1]] &, {
        lines, lineWidth}],
223       NumberQ[lineWidth] && Length[OptionValue["Intensities"]] > 0,
224         MapThread[FuzzyRectangle[#1, lineWidth, 0, height, colorFun
        [#1], #2] &, {lines, OptionValue["Intensities"]}],
225       Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]] >
        0,
226         MapThread[FuzzyRectangle[#1, #2, 0, height, colorFun[#1], #3]
        &, {lines, lineWidth, OptionValue["Intensities"]}]
227     ];
228     comments = Which[
229       Length[OptionValue["Comments"]] > 0,
230         MapThread[StringJoin[ToString[#1]<>" nm","\n",ToString[#2]]&,
231         {lines, OptionValue["Comments"]}],
232       Length[OptionValue["Comments"]] == 0,
233         ToString[#] <>" nm" & /@ lines,
234       True,
235       {}
236     ];
237     If[OptionValue["Tooltips"],
238       fuzzyLines = MapThread[Tooltip[#1, #2] &, {fuzzyLines, comments
        }];
239     ];
240     graphicsOpts = FilterRules[{opts}, Options[Graphics]];
241     Graphics[fuzzyLines,
242         graphicsOpts,
243         Background -> Black,
244         PlotRange -> {{minWave, maxWave}, {0, height}}]
245     )
246   ];
247
```

```
End[];

EndPackage[];
```

11.4 misc.m

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

```
BeginPackage["misc`"];

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

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

FirstOrderPerturbation;
SecondOrderPerturbation;
RoundValueWithUncertainty;

ToPythonSymPyExpression;
RoundToSignificantFigures;
RobustMissingQ;

Begin["`Private`"];

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

  Options[RoundValueWithUncertainty] = {"SetPrecision" -> False};
  RoundValueWithUncertainty::usage =
    "RoundValueWithUncertainty[x,dx] given a number x together with an
    \
  uncertainty dx this function rounds x to the first significant figure
     \
  of dx and also rounds dx to have a single significant figure.
```

```
40  The returned value is a list with the form {roundedX, roundedDx}.
41  The option \"SetPrecision\" can be used to control whether the \
42  Mathematica precision of x and dx is also set accordingly to these \
43  rules, otherwise the rounded numbers still have the original \
44  precision of the input values.
45  If the position of the first significant figure of x is after the \
46  position of the first significant figure of dx, the function returns
      \
47  {0,dx} with dx rounded to one significant figure.";
48  RoundValueWithUncertainty[x_, dx_, OptionsPattern[]] := Module[
49    {xExpo, dxExpo, sigFigs, roundedX, roundedDx, returning},
50    (
51      xExpo = RealDigits[x][[2]];
52      dxExpo = RealDigits[dx][[2]];
53      sigFigs = xExpo - dxExpo + 1;
54      {roundedX, roundedDx} = If[sigFigs <= 0,
55        {0., N@RoundToSignificantFigures[dx, 1]},
56        N[
57        {
58          RoundToSignificantFigures[x, xExpo - dxExpo + 1],
59          RoundToSignificantFigures[dx, 1]}
60        ]
61        ];
62      returning = If[
63        OptionValue["SetPrecision"],
64        {SetPrecision[roundedX, Max[1, sigFigs]],
65        SetPrecision[roundedDx, 1]},
66        {roundedX, roundedDx}
67        ];
68      Return[returning]
69      )
70    ];
71
72  RoundToSignificantFigures::usage =
73    "RoundToSignificantFigures[x, sigFigs] rounds x so that it only has
       \
74  sigFigs significant figures.";
75  RoundToSignificantFigures[x_, sigFigs_] :=
76    Sign[x]*N[FromDigits[RealDigits[x, 10, sigFigs]]];
77
78  RobustMissingQ[expr_] := (FreeQ[expr, _Missing] === False);
79
80  TextBasedProgressBar[progress_, totalIterations_, prefix_:""] :=
      Module[
81      {progMessage},
82      progMessage = ToString[progress] <> "/" <> ToString[
    totalIterations];
83      If[progress < totalIterations,
84          WriteString["stdout", StringJoin[prefix, progMessage, "\r"]],
85          WriteString["stdout", StringJoin[prefix, progMessage, "\n"]]
86      ];
87  ];
88
89  FirstOrderPerturbation::usage="Given the eigenValues and eigenVectors
       of a matrix A (which doesn't need to be given) together with a
```

```
    corresponding perturbation matrix perMatrix, this function
    calculates the first derivative of the eigenvalues with respect to
    the scale factor of the perturbation matrix. In the sense that the
    eigenvalues of the matrix A + β perMatrix are to first order equal
    to \[Lambda] + \[Delta]_i β, where the \[Delta]_i are the returned
    values. The eigenvalues and eigenvectors are assumed to be given in
     the same order, i.e. the ith eigenvalue corresponds to the ith
    eigenvector. This assuming that the eigenvalues are non-degenerate.
    ";
90  FirstOrderPerturbation[eigenValues_, eigenVectors_,
91    perMatrix_] := (Diagonal[
92    eigenVectors . perMatrix . Transpose[eigenVectors]])
93
94  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.";
95  SecondOrderPerturbation[eigenValues_, eigenVectors_, perMatrix_] := (
96    dim = Length[perMatrix];
97    eigenBras = Conjugate[eigenVectors];
98    eigenKets = eigenVectors;
99    matV = Abs[eigenBras . perMatrix . Transpose[eigenKets]]^2;
100   OneOver[x_, y_] := If[x == y, 0, 1/(x - y)];
101   eigenDiffs = Outer[OneOver, eigenValues, eigenValues, 1];
102   pProduct = Transpose[eigenDiffs]*matV;
103   Return[2*(Total /@ Transpose[pProduct])];
104   )
105
106 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.";
107 SuperIdentity[args___] := {args};
108
109 FlattenBasis::usage="FlattenBasis[basis] takes a basis in the
    standard representation and separates out the strings that describe
     the LS part of the labels and the additional numbers that define
    the values of J MJ and MI. It returns a list with two elements {
    flatbasisLS, flatbasisNums}. This is useful for saving the basis to
     an h5 file where the strings and numbers need to be separated.";
110 FlattenBasis[basis_] := Module[{flatbasis, flatbasisLS, flatbasisNums
    },
111   (
112     flatbasis = Flatten[basis];
113     flatbasisLS = flatbasis[[1 ;; ;; 4]];
114     flatbasisNums = Select[flatbasis, Not[StringQ[#]] &];
115     Return[{flatbasisLS, flatbasisNums}]
116     )
117   ];
```

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

```
Options[GreedyMatching] = {
    "alistLabels" -> {},
    "blistLabels" -> {}};
GreedyMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{
  aValues = aValues0,
  bValues = bValues0,
  bValuesOriginal = bValues0,
  bestLabels, bestMatches,
  bestLabel, aElement, givenLabels,
  aLabels, aLabel,
  diffs, minDiff,
  bLabels,
  minDiffPosition, bestMatch},
  (
  aLabels     = OptionValue["alistLabels"];
  bLabels     = OptionValue["blistLabels"];
  bestMatches = {};
  bestLabels  = {};
  givenLabels = (Length[aLabels] > 0);
  Do[
    (
    aElement        = aValues[[idx]];
    diffs           = Abs[bValues - aElement];
    minDiff         = Min[diffs];
    minDiffPosition = Position[diffs, minDiff][[1, 1]];
    bestMatch       = bValues[[minDiffPosition]];
    bestMatches     = Append[bestMatches, {aElement, bestMatch}];
    If[givenLabels,
      (
      aLabel     = aLabels[[idx]];
      bestLabel  = bLabels[[minDiffPosition]];
      bestLabels = Append[bestLabels, {aLabel, bestLabel}];
      bLabels    = Drop[bLabels, {minDiffPosition}];
      )
      ];
    bValues = Drop[bValues, {minDiffPosition}];
    If[Length[bValues] == 0, Break[]];
    ),
    {idx, 1, Length[aValues]}
    ];
  pairedIndices = MapIndexed[{#2[[1]], Position[bValuesOriginal,
 #1[[2]]][[1, 1]]} &, bestMatches];
  If[givenLabels,
    Return[{bestMatches, bestLabels, pairedIndices}],
    Return[{bestMatches, pairedIndices}]
    ]
  )
  ]

StochasticMatching::usage="StochasticMatching[aValues, bValues] finds
   a better assignment by randomly shuffling the elements of aValues
  and then applying the greedy assignment algorithm. The function
  prints what is the range of total absolute differences found during
   shuffling, the standard deviation of all of them, and the number
  of shuffles that were attempted. The option \"alistLabels\" can be
```

```
    used to specify labels for the elements in aValues. The option \"
    blistLabels\" can be used to specify labels for the elements in
    bValues. If these options are used, the function returns a list
    with three elements the pairs of matched elements, the pairs of
    corresponding matched labels, and the mapping of indices.";
203 Options[StochasticMatching] = {"alistLabels" -> {},
204   "blistLabels" -> {}};
205 StochasticMatching[aValues0_, bValues0_, numShuffles_ : 200,
    OptionsPattern[]] := Module[{
206   aValues = aValues0,
207   bValues = bValues0,
208   matchingLabels, ranger, matches, noShuff, bestMatch, highestCost,
    lowestCost, dev, sorter, bestValues,
209   pairedIndices, bestLabels, matchedIndices, shuffler
210   },
211   (
212   matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
213   ranger = Range[1, Length[aValues]];
214   matches = If[Not[matchingLabels], (
215       Table[(
216         shuffler = If[i == 1, ranger, RandomSample[ranger]];
217         {bestValues, matchedIndices} =
218         GreedyMatching[aValues[[shuffler]], bValues];
219         cost = Total[Abs[#[[1]] - #[[2]]] & /@ bestValues];
220         {cost, {bestValues, matchedIndices}}
221         ), {i, 1, numShuffles}]
222       ),
223     Table[(
224       shuffler = If[i == 1, ranger, RandomSample[ranger]];
225       {bestValues, bestLabels, matchedIndices} =
226         GreedyMatching[aValues[[shuffler]], bValues,
227         "alistLabels" -> OptionValue["alistLabels"][[shuffler]],
228         "blistLabels" -> OptionValue["blistLabels"]];
229       cost = Total[Abs[#[[1]] - #[[2]]] & /@ bestValues];
230       {cost, {bestValues, bestLabels, matchedIndices}}
231       ), {i, 1, numShuffles}]
232     ];
233   noShuff = matches[[1, 1]];
234   matches = SortBy[matches, First];
235   bestMatch = matches[[1, 2]];
236   highestCost = matches[[-1, 1]];
237   lowestCost = matches[[1, 1]];
238   dev = StandardDeviation[First /@ matches];
239   Print[lowestCost, " <-> ", highestCost, " | \[Sigma]=", dev,
240     " | N=", numShuffles, " | null=", noShuff];
241   If[matchingLabels,
242     (
243     {bestValues, bestLabels, matchedIndices} = bestMatch;
244     sorter = Ordering[First /@ bestValues];
245     bestValues = bestValues[[sorter]];
246     bestLabels = bestLabels[[sorter]];
247     pairedIndices =
248       MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
249       bestValues];
250     Return[{bestValues, bestLabels, pairedIndices}]
```

```
      ),
      (
      {bestValues, matchedIndices} = bestMatch;
      sorter = Ordering[First /@ bestValues];
      bestValues = bestValues[[sorter]];
      pairedIndices =
        MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
        bestValues];
      Return[{bestValues, pairedIndices}]
      )
      ];
    )
    ]

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

```
287
288     (*From all the elements of A to all the elements of B*)
289     midLayer = Table[{{"A", i} \[DirectedEdge] ({"B", j}), 1, cFun[
        aValues[[i]], bValues[[j]]]}, {i, 1, nA}, {j, 1, nB}];
290     midLayer = Flatten[midLayer, 1];
291     {midLayerEdges, midCapacities, midCosts} = Transpose[midLayer];
292
293     (*From the elements of B to the sink*)
294     edgesBtoSink = ({"B", #} \[DirectedEdge] "sink") & /@ Range[1, nB];
295     capacityBtoSink = ConstantArray[1, nB];
296     costBtoSink = ConstantArray[0, nB];
297
298     (*Put it all together*)
299     allCapacities = Join[capacitySourceToA, midCapacities,
        capacityBtoSink];
300     allCosts = Join[costSourceToA, midCosts, costBtoSink];
301     allEdges = Join[edgesSourceToA, midLayerEdges, edgesBtoSink];
302     graph = Graph[allEdges, EdgeCapacity -> allCapacities,
303       EdgeCost -> allCosts];
304
305     (*Solve it*)
306     flow = FindMinimumCostFlow[graph, "source", "sink", nA -
        OptionValue["notMatched"], "OptimumFlowData"];
307     (*Collect the pairs of matched indices*)
308     pairedIndices = Select[flow["EdgeList"], And[Not[#[[1]] === "source
        "], Not[#[[2]] === "sink"]] &];
309     pairedIndices = {#[[1, 2]], #[[2, 2]]} & /@ pairedIndices;
310     (*Collect the pairs of matched values*)
311     bestValues = {aValues[[#[[1]]]], bValues[[#[[2]]]]} & /@
        pairedIndices;
312     (*Account for having been given labels*)
313     If[matchingLabels,
314       (
315       bestLabels = {aLabels[[#[[1]]]], bLabels[[#[[2]]]]} & /@
        pairedIndices;
316       Return[{bestValues, bestLabels, pairedIndices}]
317       ),
318       (
319       Return[{bestValues, pairedIndices}]
320       )
321       ];
322     )
323     ]
324
325   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.";
326   HelperNotebook[nbName_:"OUT"] :=
327   Module[{screenDims, screenWidth, screenHeight, nbWidth, leftMargin,
328     PrintToOutputNb}, (
329     screenDims =
330       SystemInformation["Devices", "ScreenInformation"][[1, 2, 2]];
331     screenWidth = screenDims[[1, 2]];
332     screenHeight = screenDims[[2, 2]];
```

```
    nbWidth = Round[screenWidth/3];
    leftMargin = screenWidth - nbWidth;
    outputNb = CreateDocument[{}, WindowTitle -> nbName,
      WindowMargins -> {{leftMargin, Automatic}, {Automatic,
          Automatic}}, WindowSize -> {nbWidth, screenHeight}];
    PrintToOutputNb[text_] :=
      (
          SelectionMove[outputNb, After, Notebook];
          NotebookWrite[outputNb, Cell[BoxData[ToBoxes[text]], "Output"
  ]];
      );
    Return[PrintToOutputNb]
    )
    ]

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[];
```

11.5 qalculations.m

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

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

workDir = DirectoryName[$InputFileName];

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

The function returns a list with nine elements
```

```
{rmsDifference, carnallEnergies, eigenEnergies, ln, carnallAssignments,
     simplerStateLabels, eigensys, basis, truncatedStates}.

Where:
1. rmsDifference is the root mean squared difference between the
    calculated values and those quoted by Carnall
2. carnallEnergies are the quoted calculated energies from Carnall;
3. eigenEnergies are the calculated energies (in the case of an odd
    number of electrons the Kramers degeneracy may have been removed
    from this list according to the option \"Remove Kramers\");
4. ln is simply a string labelling the corresponding lanthanide;
5. carnallAssignments is a list of strings providing the multiplet
    assignments that Carnall assumed;
6. simplerStateLabels is a list of strings providing the multiplet
    assignments that this function assumes;
7. eigensys is a list of tuples where the first element is the energy
    corresponding to the eigenvector given as the second element (in
    the case of an odd number of electrons the Kramers degeneracy may
    have been removed from this list according to the option \"Remove
    Kramers\");
8. basis is a list that specifies the basis in which the Hamiltonian
    was constructed and diagonalized, equal to BasisLSJMJ[numE];
9. Same as eigensys but the eigenvectors have been truncated so that
    the truncated version adds up to at least a total probability of
    eigenstateTruncationProbability.
";
Options[FastIonSolverLaF3] = {
  "MakeNotebook" -> True,
  "NotebookSave" -> True,
  "HTMLSave" -> False,
  "eigenstateTruncationProbability" -> 0.9,
  "Include spin-spin" -> True,
  "Max Eigenstates in Table" -> 100,
  "Sparse" -> True,
  "PrintFun" -> Print,
  "SaveData" -> True,
  "paramFiddle" -> {},
  "Append to Filename" -> "",
  "Remove Kramers" -> True,
  "OutputDirectory" -> "calcs",
  "Explorer" -> False
};
FastIonSolverLaF3[numE_, OptionsPattern[]] := Module[
  {makeNotebook, eigenstateTruncationProbability, host,
  ln, terms, termNames, carnallEnergies, eigenEnergies,
    simplerStateLabels,
  eigensys, basis, assignmentMatches, stateLabels, carnallAssignments},
  (
    PrintFun = OptionValue["PrintFun"];
    makeNotebook = OptionValue["MakeNotebook"];
    eigenstateTruncationProbability = OptionValue["
    eigenstateTruncationProbability"];
    maxStatesInTable = OptionValue["Max Eigenstates in Table"];
    Duplicator[aList_] := Flatten[{#, #} & /@ aList];
    host = "LaF3";
```

```
    paramFiddle = OptionValue["paramFiddle"];
    ln = theLanthanides[[numE]];
    terms = AllowedNKSLJTerms[Min[numE, 14 - numE]];
    termNames = First /@ terms;
    (* For labeling the states, the degeneracy in some of the terms is
    elided *)
    PrintFun["> Calculating simpler term labels ..."];
    termSimplifier = Table[termN -> If[StringLength[termN] == 3,
      StringTake[termN, {1, 2}],
      termN
      ],
      {termN, termNames}
    ];

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

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

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

```

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

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

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

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

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

    (*Compare against the data quoted by Bill Carnall*)
    PrintFun["> Comparing against the data from Carnall ..."];
    mainKey            = StringTemplate["appendix:`Ln`:Association"][<|
    "Ln" -> ln|>];
    lnData             = Carnall[mainKey];
    carnalKeys         = lnData // Keys;
    repetitions        = Length[lnData[#]["Calc (1/cm)"]] & /@
    carnalKeys;
    carnallAssignments = First /@ Carnall["appendix:" <> ln <> ":
    RawTable"];
    carnalKey          = StringTemplate["appendix:`Ln`:Calculated"][<|"
    Ln" -> ln|>];
    carnallEnergies    = Carnall[carnalKey];
```

```
150     If[And[OddQ[numE], Not[OptionValue["Remove Kramers"]]],
151     (
152       PrintFun[">> The number of eigenstates and the number of quoted
        states don't match, removing the last state ..."];
153       carnallAssignments = Duplicator[carnallAssignments];
154       carnallEnergies    = Duplicator[carnallEnergies];
155     )
156     ];
157
158     (* For the difference take as many energies as quoted by Bill*)
159     eigenEnergies = eigenEnergies + carnallEnergies[[1]];
160     diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] -
        carnallEnergies;
161     (* Remove the differences where the appendix tables have elided
        values*)
162     rmsDifference = Sqrt[Mean[(Select[diffs, FreeQ[#, Missing[]] &])
        ^2]];
163     titleTemplate = StringTemplate[
164         "Energy Level Diagram of \!\(\*SuperscriptBox[\(`ion`\),
        \(\(3\)\(+\)\)]\)"];
165     title = titleTemplate[<|"ion" -> In|>];
166     parsedStates = ParseStates[eigensys, basis];
167     If[And[OddQ[numE],OptionValue["Remove Kramers"]],
168       parsedStates = parsedStates[[;; ;; 2]]
169     ];
170
171     stateLabels = #[[-1]] & /@ parsedStates;
172     simplerStateLabels = ((#[[2]] /. termSimplifier) <> ToString
        [#[[3]], InputForm]) & /@ parsedStates;
173
174     PrintFun[">> Truncating eigenvectors to given probability ..."];
175     startTime = Now;
176     truncatedStates = ParseStatesByProbabilitySum[eigensys, basis,
177       eigenstateTruncationProbability,
178       0.01];
179     endTime = Now;
180     truncationTime = QuantityMagnitude[endTime - startTime, "Seconds"];
181     PrintFun[">>> Truncation took ", truncationTime, " seconds."];
182
183     If[makeNotebook,
184     (
185       PrintFun["> Putting together results in a notebook ..."];
186       energyDiagram = Framed[
187         EnergyLevelDiagram[eigensys, "Title" -> title,
188         "Explorer" -> OptionValue["Explorer"],
189         "Background" -> White]
190         , Background -> White, FrameMargins -> 50];
191       appToFname = OptionValue["Append to Filename"];
192       PrintFun[">> Comparing the term assignments between qlanth and
        Carnall ..."];
193       assignmentMatches =
194       If[StringContainsQ[#[[1]], #[[2]]], "\[Checkmark]", "X"] & /@
195         Transpose[{carnallAssignments, simplerStateLabels[[;; Length[
        carnallAssignments]]]}];
196       assignmentMatches = {{"\[Checkmark]",
```

```
197         Count[assignmentMatches, "\[Checkmark]"]}, {"X",
198         Count[assignmentMatches, "X"]}};
199       labelComparison = (If[StringContainsQ[#[[1]], #[[2]]], "\[
    Checkmark]", "X"] & /@
200         Transpose[{carnallAssignments,
201         simplerStateLabels[[;; Length[carnallAssignments]]]}]);
202       labelComparison =
203       PadRight[labelComparison, Length[simplerStateLabels], "-"];
204
205       statesTable = Grid[Prepend[{Round[#[[1]]], #[[2]]} & /@
206           truncatedStates[[;;Min[Length[eigensys],maxStatesInTable]]],
    {"Energy/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)",
207           "\[Psi]"}], Frame -> All, Spacings -> {2, 2},
208           FrameStyle -> Blue,
209           Dividers -> {{False, True, False}, {True, True}}];
210       DefaultIfMissing[expr_]:= If[FreeQ[expr, Missing[]], expr,"NA"];
211       PrintFun[">> Rounding the energy differences for table
    presentation ..."];
212       roundedDiffs = Round[diffs, 0.1];
213       roundedDiffs = PadRight[roundedDiffs, Length[simplerStateLabels],
     "-"];
214       roundedDiffs = DefaultIfMissing /@ roundedDiffs;
215       diffs = PadRight[diffs, Length[simplerStateLabels], "-"];
216       diffs = DefaultIfMissing /@ diffs;
217       diffTableData = Transpose[{simplerStateLabels, eigenEnergies,
218         labelComparison,
219         PadRight[carnallAssignments, Length[simplerStateLabels], "-"],
220         DefaultIfMissing/@PadRight[carnallEnergies, Length[
    simplerStateLabels], "-"],
221         roundedDiffs}
222       ];
223       diffTable = TableForm[diffTableData,
224         TableHeadings -> {None, {"qlanth",
225         "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)", "", "Carnall",
226         "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)",
227         "\[CapitalDelta]E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"}}
228       ];
229
230       diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] -
    carnallEnergies;
231       notBad = FreeQ[#,Missing[]]&/@diffs;
232       diffs = Pick[diffs,notBad];
233       diffHistogram = Histogram[diffs,
234         Frame -> True,
235         ImageSize -> 800,
236         AspectRatio -> 1/3, FrameStyle -> Directive[16],
237         FrameLabel -> {"(qlanth-carnall)/Ky", "Freq"}
238       ];
239       rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
240       labelTempate = StringTemplate["\!\(\*SuperscriptBox[\(`ln`\),
    \(\(3\)\(+\)\)]\)"];
241       diffData   = diffs;
242       diffLabels = simplerStateLabels[[;;Length[notBad]]];
243       diffLabels = Pick[diffLabels, notBad];
244       diffPlot   = Framed[
```

```
245         ListLabelPlot[diffData,
246         diffLabels,
247         Frame -> True,
248         PlotRange -> All,
249         ImageSize -> 1200,
250         AspectRatio -> 1/3,
251         FrameLabel -> {"",
252         "(qlanth-carnall) / \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"},
253         PlotMarkers -> "OpenMarkers",
254         PlotLabel ->
255         Style[labelTempate[<|"ln" -> ln|>] <> " | " <> "\[Sigma]=" <>
256             ToString[Round[rmsDifference, 0.01]] <>
257             " \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)\n", 20],
258         Background -> White
259         ],
260         Background -> White,
261         FrameMargins -> 50
262       ];
263       (* now place all of this in a new notebook *)
264       nb = CreateDocument[
265       {
266         TextCell[Style[
267           DisplayForm[RowBox[{SuperscriptBox[host <> ":" <> ln, "3+"],
    "(", SuperscriptBox["f", numE], ")"}]]
268           ], "Title", TextAlignment -> Center
269         ],
270         TextCell["Energy Diagram",
271           "Section",
272           TextAlignment -> Center
273         ],
274         TextCell[energyDiagram,
275           TextAlignment -> Center
276         ],
277         TextCell["Multiplet Assignments & Energy Levels",
278           "Section",
279           TextAlignment -> Center
280         ],
281         TextCell[diffHistogram, TextAlignment -> Center],
282         TextCell[diffPlot, "Output", TextAlignment -> Center],
283         TextCell[assignmentMatches, "Output", TextAlignment -> Center],
284         TextCell[diffTable, "Output", TextAlignment -> Center],
285         TextCell["Truncated Eigenstates", "Section", TextAlignment ->
    Center],
286         TextCell["These are some of the resultant eigenstates which add
     up to at least a total probability of " <> ToString[
    eigenstateTruncationProbability] <> ".", "Text", TextAlignment ->
    Center],
287         TextCell[statesTable, "Output", TextAlignment -> Center]
288       },
289       WindowSelected -> True,
290       WindowTitle -> ln <> " in " <> "LaF3" <> appToFname,
291       WindowSize -> {1600, 800}];
292       If[OptionValue["SaveData"],
293       (
294         exportFname = FileNameJoin[{workDir,OptionValue["
```

```
    OutputDirectory"], ln <> " in " <> "LaF3" <> appToFname <> ".m"}];
295       SelectionMove[nb, After, Notebook];
296       NotebookWrite[nb, Cell["Reload Data", "Section", TextAlignment
    -> Center]];
297       NotebookWrite[nb,
298         Cell[(
299           "{rmsDifference, carnallEnergies, eigenEnergies, ln,
    carnallAssignments, simplerStateLabels, eigensys, basis,
    truncatedStates} = Import[FileNameJoin[{NotebookDirectory[],\"" <>
    StringSplit[exportFname,"/"][[-1]] <> "\"}]];"
300           ),"Input"
301         ]
302       ];
303       NotebookWrite[nb,
304         Cell[(
305           "Manipulate[First[MinimalBy[truncatedStates, Abs[First[#] -
     energy] &]], {energy,0}]"
306           ),"Input"]
307       ];
308       (* Move the cursor to the top of the notebook *)
309       SelectionMove[nb, Before, Notebook];
310       Export[exportFname,
311         {rmsDifference, carnallEnergies, eigenEnergies, ln,
    carnallAssignments, simplerStateLabels, eigensys, basis,
    truncatedStates}
312       ];
313       tinyexportFname = FileNameJoin[
314         {workDir, OptionValue["OutputDirectory"], ln <> " in " <> "
    LaF3" <> appToFname <> " - tiny.m"}
315       ];
316       tinyExport = <|"ln"->ln,
317                      "carnallEnergies"->carnallEnergies,
318                      "rmsDifference"-> rmsDifference,
319                      "eigenEnergies"-> eigenEnergies,
320                      "carnallAssignments"-> carnallAssignments,
321                      "simplerStateLabels" -> simplerStateLabels|>;
322       Export[tinyexportFname, tinyExport];
323     )
324     ];
325     If[OptionValue["NotebookSave"],
326       (
327         nbFname = FileNameJoin[{workDir,OptionValue["OutputDirectory"
    ], ln <> " in " <> "LaF3" <> appToFname <> ".nb"}];
328         PrintFun[">> Saving notebook to ", nbFname, " ..."];
329         NotebookSave[nb, nbFname];
330       )
331     ];
332     If[OptionValue["HTMLSave"],
333     (
334       htmlFname = FileNameJoin[{workDir,OptionValue["OutputDirectory"
    ], "html", ln <> " in " <> "LaF3" <> appToFname <> ".html"}];
335       PrintFun[">> Saving html version to ", htmlFname, " ..."];
336       Export[htmlFname, nb];
337     )
338     ];
```

```
339     )
340     ];
341
342     Return[{rmsDifference, carnallEnergies, eigenEnergies, ln,
        carnallAssignments, simplerStateLabels, eigensys, basis,
        truncatedStates}];
343   )
344 ];
345
346 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:
347     \[Psi]i:simple, (* main contribution to the wavefuction |i>*)
348     \[Psi]f:simple, (* main contribution to the wavefuction |j>*)
349     \[Psi]i:idx,    (* index of the wavefuction |i>*)
350     \[Psi]f:idx,    (* index of the wavefuction |j>*)
351     Ei/K,           (* energy of the initial state in K *)
352     Ef/K,           (* energy of the final state in K *)
353     \[Lambda]/nm,   (* transition wavelength in nm *)
354     \[CapitalDelta]\[Lambda]/nm, (* uncertainty in the transition
        wavelength in nm *)
355     \[Tau]/s,       (* radiative lifetime in s *)
356     AMD/s^-1        (* magnetic dipole transition rate in s^-1 *)
357
358 The raw data file contains the following keys:
359     - Line Strength, (* Line strength array *)
360     - AMD, (* Magnetic dipole transition rates in 1/s *)
361     - fMD, (* Oscillator strengths from ground to excited states *)
362     - Radiative lifetimes, (* Radiative lifetimes in s *)
363     - Transition Energies / K, (* Transition energies in K *)
364     - Transition Wavelengths in nm. (* Transition wavelengths in nm *)
365
366 The CSV file contains the same information as the tabular file.
367
368 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.
369
370 The function takes the following options:
371     - \"Make Notebook\" -> True or False. If True, a notebook with a
        Manipulate is created. Default is True.
372     - \"Print Function\" -> PrintTemporary or Print. The function used
        to print the progress of the calculation. Default is PrintTemporary
        .
373     - \"Host\" -> \"LaF3\". The host material. Default is LaF3.
374     - \"Wavelength Range\" -> {50,2000}. The range of wavelengths in nm
         for the Manipulate object in the created notebook. Default is
        {50,2000}.
375
376 The function returns an association containing the following keys: Line
         Strength, AMD, fMD, Radiative lifetimes, Transition Energies / K,
        Transition Wavelengths in nm.";
377 Options[MagneticDipoleTransitions] = {
378         "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,
    fromTo,
    energyPairs,
    magIon["Transition Wavelengths in nm"],
    \[Lambda]uncertainty,
    magIon["AMD"],
    magIon["Radiative lifetimes"]
};
allTransitions = (Flatten/@Transpose[Flatten[#,1]&/@allTransitions]);
allTransitions = Select[allTransitions, #[[3]]!=#[[4]]&];
allTransitions = Select[allTransitions, #[[10]]>0&];
allTransitions = Transpose[allTransitions];

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

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

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

```
467   exportKey    = StringReplace[basename,".zip"->".m"];
468   Export[exportFname, <|exportKey->finalTable|>];
469
470   (* raw data output *)
471   basename    = ln <> " in " <> host <> " - example - " <> "MD1 - raw.
        zip";
472   rawexportFname = FileNameJoin[{"./examples",basename}];
473   PrintFun["Exporting raw data as an association to "<>exportFname<>"
        ..."];
474   rawexportKey   = StringReplace[basename,".zip"->".m"];
475   Export[rawexportFname, <|rawexportKey->magIon|>];
476
477   (* csv output *)
478   PrintFun["Formatting and exporting a CSV output..."];
479   csvOut = Table[
480       StringJoin[Riffle[ToString[#,CForm]&/@finalTable[[i]],","]],
481   {i,1,Length[finalTable]}
482   ];
483   csvOut      = StringJoin[Riffle[csvOut, "\n"]];
484   basename    = ln <> " in " <> host <> " - example - " <> "MD1.csv";
485   exportFname = FileNameJoin[{"./examples", basename}];
486   PrintFun["Exporting csv data to "<>exportFname<>" ..."];
487   Export[exportFname, csvOut, "Text"];
488
489   If[OptionValue["Make Notebook"],
490   (
491       PrintFun["Creating a notebook with a Manipulate to select a
        wavelength interval and a lifetime power of ten ..."];
492       finalTable     = Rest[finalTable];
493       finalTable     = SortBy[finalTable,#[[7]]&];
494       opticalTable   = Select[finalTable,\[Lambda]min<=#[[7]]<=\[Lambda
        ]max&];
495       pows           = Sort[DeleteDuplicates[(MantissaExponent
        [#[[9]]][[2]]-1)&/@opticalTable]];
496
497       man            = Manipulate[
498       (
499           {\[Lambda]min,\[Lambda]max} = \[Lambda]int;
500           table = Select[opticalTable,And[(\[Lambda]min<=#[[7]]<=\[
        Lambda]max),
501                       (MantissaExponent[#[[9]]][[2]]-1)==log10\[Tau
        ]]&];
502           tab    = TableForm[table,TableHeadings->{None,header}];
503           Column[{{"\[Lambda]min="<>ToString[\[Lambda]min]<>" nm","\[
        Lambda]max="<>ToString[\[Lambda]max]<>" nm",log10\[Tau]},tab}]
504       ),
505       {{\[Lambda]int,\[Lambda]Range,"\[Lambda] interval"},
506           \[Lambda]Range[[1]],
507           \[Lambda]Range[[2]],
508           50,
509           ControlType->IntervalSlider
510       },
511       {{log10\[Tau],pows[[-1]]},
512           pows
513       },
```

```
514       TrackedSymbols  :> {\[Lambda]int,log10\[Tau]},
515       SaveDefinitions -> True
516       ];
517
518       nb = CreateDocument[{
519           TextCell[Style[DisplayForm[RowBox[{"Magnetic Dipole
    Transitions", "\n", SuperscriptBox[host<>":"<>ln,"3+"],"(",
    SuperscriptBox["f",numE],")"}]]],"Title",TextAlignment->Center],
520           (* TextCell["Magnetic Dipole Transition Lifetimes","Section",
    TextAlignment->Center], *)
521           TextCell[man,"Output",TextAlignment->Center]
522       },
523       WindowSelected -> True,
524       WindowTitle    -> "MD1 - "<>ln<>" in "<>host,
525       WindowSize     -> {1600,800}
526       ];
527       SelectionMove[nb, After, Notebook];
528       NotebookWrite[nb, Cell["Reload Data", "Section", TextAlignment ->
     Center]];
529       NotebookWrite[nb, Cell[(
530             "magTransitions = Import[FileNameJoin[{NotebookDirectory
    [],\"" <> StringSplit[rawexportFname,"/"][[-1]] <> "\"}],\""<>
    rawexportKey<>"\"];"
531           ),"Input"]];
532       SelectionMove[nb, Before, Notebook];
533       nbFname = FileNameJoin[{workDir,"examples","MD1 - "<>ln<>" in "<>
    "LaF3"<>".nb"}];
534       PrintFun[">> Saving notebook to ",nbFname," ..."];
535       NotebookSave[nb, nbFname];
536       If[OptionValue["Close Notebook"],
537           NotebookClose[nb];
538       ];
539   )
540   ];
541
542   Return[magIon];
543 )
```

References

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

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

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

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

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

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

[JS84] BR Judd and MA Suskin. "Complete set of orthogonal scalar operators for the configuration 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.

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

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

[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.

[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.