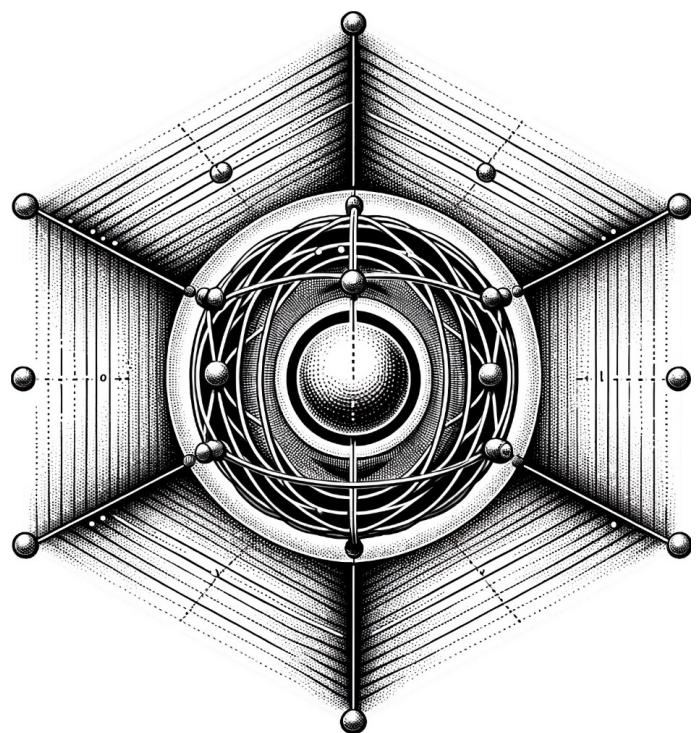


**qlanth**  
v1.2.2



Juan David Lizarazo Ferro,  
Christopher Dodson  
& Rashid Zia

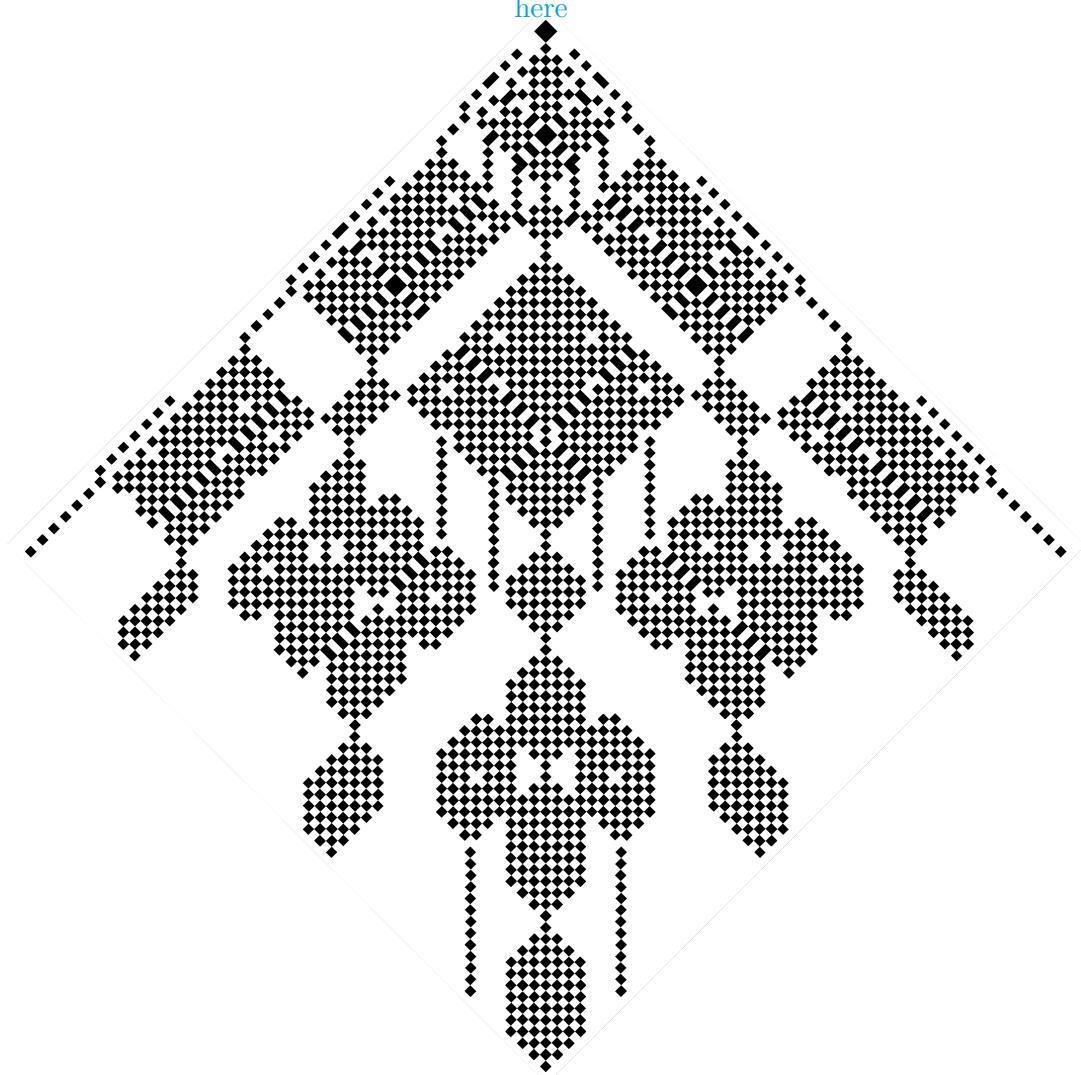
Brown University,  
Department of Physics

---

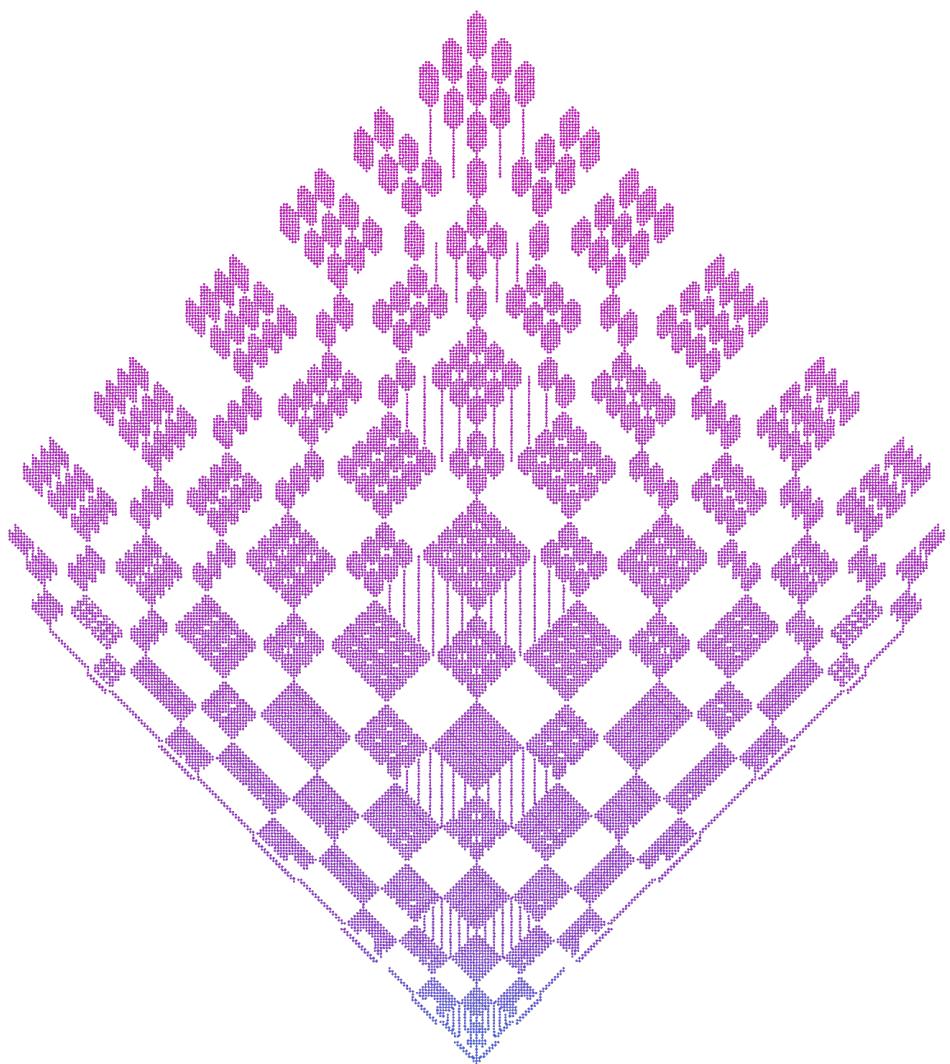
Providence, Rhode Island  
2025 AD

**qlanth** may be downloaded

[here](#)







This work was sponsored by the  
**National Science Foundation**  
Grant No. DMR-1922025



**qlanth** is a tool that can be used to estimate the electronic structure of lanthanide ions in crystals. It uses an effective Hamiltonian limited to a single-configuration with included configuration-interaction corrections. This Hamiltonian aims to describe the observed properties of ions embedded in solids in a picture that imagines them as free-ions modified by the influence of the lattice in which they find themselves in.

This picture of lanthanide ions is one that developed and mostly matured in the second half of the last century by the efforts of John Slater,<sup>1</sup> Giulio Racah,<sup>2</sup> Brian Judd,<sup>3</sup> Gerhard Dieke,<sup>4</sup> Hannah Crosswhite,<sup>5</sup> Robert Cowan,<sup>6</sup> Michael Reid,<sup>7</sup> William Carnall,<sup>8</sup> Clyde Morrison,<sup>9</sup> Richard Leavitt,<sup>10</sup> Brian Wybourne,<sup>11</sup> Richard Trees,<sup>12</sup> and Katherine Rajnak<sup>13</sup> among others. The goal of this tool is to provide a modern implementation of the methods that resulted from their work. This code is written in Wolfram language.

Separate to their specific use in this code, **qlanth** also includes data that might be of use to those interested in the single-configuration description of lanthanide ions. These data include the coefficients of fractional parentage (as calculated by Velkov and parsed here), and reduced matrix elements for all the operators in the effective Hamiltonian. These are provided as standard *Mathematica* associations that should be simple to use elsewhere. One feature of **qlanth** is that symbolic expressions are maintained up to the very last moment where numerical approximations are inevitable. As such, the symbolic expressions that result for the matrix representation of the Hamiltonian, result in linear combinations of the model parameters with symbolic coefficients.

The included *Mathematica* notebook **qlanth.nb** lists most of the functions included in **qlanth** and should be considered complementary to this document. The **/examples** folder includes notebooks containing the result of this description for most of the trivalent lanthanide ions in lanthanum fluoride. LaF<sub>3</sub> is remarkable in that it was one of the systems in which a systematic study [Car+89] of all of the trivalent lanthanide ions were studied.

This code was originally authored by Christopher Dodson and Rashid Zia for their research into magnetic dipole transitions in lanthanide ions [DZ12]. Here it has been rewritten and expanded by David Lizarazo. It has also benefited from conversations with Tharnier Puel at the University of Iowa.

This document has 18 sections. Section 1 gives an overview the semi-empirical Hamiltonian. Section 2 explains the details of the basis in which the semi-empirical Hamiltonian is evaluated, together with the method of fractional parentage, additional quantum numbers, Kramer's degeneracy, and the JJ' block structure of the semi-empirical Hamiltonian. Section 3 gives a detailed explanation of each of the interactions include in the semi-empirical Hamiltonian. Section 4 gives explains the implicit assumptions in the orientation of the coordinate system. Section 5 gives an overview of the attendant experimental setups and considerations about uncertainty. Section 6 is about the calculation of magnetic and forced electric dipole transitions. Section 7 explain certain constraints often used for the parameters in the semi-empirical Hamiltonian.

Section 8 explains the details of fitting the Hamiltonian to experimental data. Section 9 lists included auxiliary *Mathematica* notebooks. Section 11 explains the details of an abbreviated Python extension to **qlanth**. Section 12 explains some of the included experimental data. Section 13 contains a few assorted details on running **qlanth**. Section 14 has a brief comment on units. Section 15 and Section 17 include a summary of notation and definitions used throughout this document. Finally, Section 18 contains a printout of the code included in **qlanth**.

Besides being a fully functional code that works out of the box, **qlanth** is unique in that it also includes computational routines that can generate from scratch (or close to scratch) the necessary reduced matrix elements which in other codes are simply loaded from other vintages. Great care was taken to comment every loop, variable, procedure, and data provenance. To highlight this, the code relevant to the different functions has been interspersed in the parts where they are mentioned.

---

<sup>1</sup> [Sla29]    <sup>2</sup> [Rac42a; Rac42b; Rac43; Rac49]    <sup>3</sup> [Jud62; Jud63b; Jud63a; Jud66; Jud67; JCC68; CCJ68; Jud82; Jud83; JS84; JC84; Jud85; Jud86; Jud88; Jud89; JL93; Jud96; Jud05]    <sup>4</sup> [DC63; PDC67; Die68]    <sup>5</sup> [CCJ68; Cro71; Cro+76; Cro+77; DC63; JCC68; JC84]    <sup>6</sup> [Cow81]    <sup>7</sup> [Rei81]    <sup>8</sup> [CFW65; Car+89; Car92; CFR68b; CFR68e; CFR68c; CFR68d; CFR68a; Car+70; Car+76; GW+91]    <sup>9</sup> [MWK76; ML79; MW94; Mor80; MT87; MKW77b; MKW77a; ML82; Mor+83]    <sup>10</sup> [Lea87; Lea82; LM80; ML79; ML82]    <sup>11</sup> [CFW65; CW63; RW63; RW64b; RW64a; Wyb64a; Wyb64b; Wyb65; Wyb70; WS07]    <sup>12</sup> [Tre52; Tre51; Tre58]    <sup>13</sup> [RW63; RW64b; RW64a; Raj65]

## Contents

<b>1</b>	<b>The semi-empirical Hamiltonian</b>	<b>1</b>
<b>2</b>	<b>LS coupling basis</b>	<b>3</b>
2.1	$ LSJM\rangle$ states	4
2.2	More quantum numbers	8
2.2.1	Seniority $\nu$	8
2.2.2	$\mathcal{U}$ and $\mathcal{W}$	8
2.3	$ LSJ\rangle$ levels	9
2.4	The coefficients of fractional parentage	15
2.5	Going beyond $f^7$	17
2.6	The J-J' block structure	18
2.7	Kramers' degeneracy	22
<b>3</b>	<b>Interactions</b>	<b>22</b>
3.1	$\hat{\mathcal{H}}_k$ : kinetic energy	22
3.2	$\hat{\mathcal{H}}_{e:sn}$ : the central field potential	22
3.3	$\hat{\mathcal{H}}_{e:e}$ : e:e repulsion	24
3.4	$\hat{\mathcal{H}}_{s:o}$ : spin-orbit	26
3.5	$\hat{\mathcal{H}}_{SO(3)}$ , $\hat{\mathcal{H}}_{G_2}$ , $\hat{\mathcal{H}}_{SO(7)}$ : electrostatic configuration interaction	27
3.6	$\hat{\mathcal{H}}_{s:s-s:oo}$ : spin-spin and spin-other-orbit	28
3.7	$\hat{\mathcal{H}}_{ecs:o}$ : electrostatically-correlated-spin-orbit	32
3.8	$\hat{\mathcal{H}}_3$ : three-body effective operators	41
3.9	$\hat{\mathcal{H}}_{cf}$ : crystal-field	45
3.10	$\hat{\mu}$ and $\hat{\mathcal{H}}_Z$ : the magnetic dipole operator and the Zeeman term	51
3.11	Alternative operator bases	53
<b>4</b>	<b>Coordinate system</b>	<b>57</b>
<b>5</b>	<b>Spectroscopic measurements and uncertainty</b>	<b>57</b>
<b>6</b>	<b>Transitions</b>	<b>59</b>
6.1	State description	59
6.1.1	Magnetic dipole transitions	59
6.2	Level description	62
6.2.1	Forced electric dipole transitions	62
6.2.2	Magnetic dipole transitions	66
<b>7</b>	<b>Parameter constraints</b>	<b>69</b>
<b>8</b>	<b>Fitting experimental data</b>	<b>69</b>
<b>9</b>	<b>Accompanying notebooks</b>	<b>85</b>
<b>10</b>	<b>Compiled data for <math>\text{LaF}_3:\text{Ln}^{3+}</math> and <math>\text{LiYF}_4:\text{Ln}^{3+}</math></b>	<b>85</b>
<b>11</b>	<b>sparsefn.py</b>	<b>87</b>
<b>12</b>	<b>Data sources</b>	<b>88</b>
<b>13</b>	<b>Other details</b>	<b>88</b>
<b>14</b>	<b>Units</b>	<b>88</b>
<b>15</b>	<b>Notation</b>	<b>89</b>
<b>16</b>	<b>Mathematica paclet</b>	<b>89</b>
<b>17</b>	<b>Definitions</b>	<b>91</b>

<b>18 code</b>	<b>92</b>
18.1 qlanth.m . . . . .	92
18.2 fittings.m . . . . .	176
18.3 qplotter.m . . . . .	218
18.4 misc.m . . . . .	222

# 1 The semi-empirical Hamiltonian

Electrons in a multi-electron ion are subject to a number of interactions. They are attracted to the nucleus about which they orbit. Being bundled together with other electrons, they experience repulsion from all of them. Having spin, they are also subject to various magnetic interactions. The spin of each electron interacts with the magnetic field generated by its own orbital angular momentum and of other electrons. And between pairs of electrons, the spin of one can influence the others spin through the interaction of their respective magnetic dipoles.

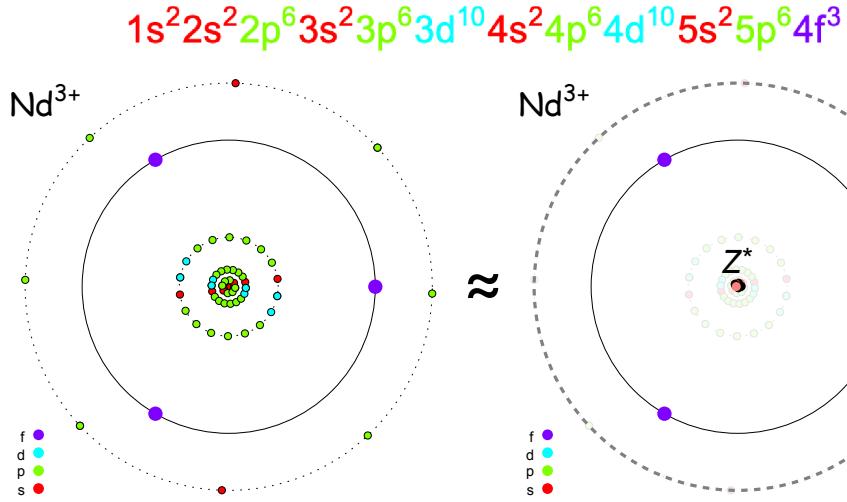


Figure 1: The trivalent neodymium ion shielded by  $5s^2$  and  $5p^6$  closed shells.

To describe the effect of the charges in the lattice surrounding the ion, the crystal field is introduced. In the simplest of embodiments, the crystal field is simply seen as the electrostatic field due to surrounding charges. This model is of limited applicability if taken too literally; however, if only symmetry considerations are assumed, the model is seen to have greater validity but a somewhat less clear physical origin.

The Hilbert space of a multi-electron ion is a vast stage. In principle, a basis for it should have a countable infinity of bound states and an uncountable infinity of 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 (or trivalent) neodymium atom, as depicted in Fig-1. In principle, this gives us the daunting task of dealing with the enormous Hilbert space of 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 57. 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 will all be considered to be located in f-orbitals, or what is the same, that they are described by  $f^n$  wavefunctions. Table 2 shows the (ground) configuration for the trivalent lanthanide ions. This is, however, a harsh approximation, but thankfully one can make some corrections to it. The effects that arise in the single configuration description because of omitting all the other possible orbitals where the

<b>Ce<sup>3+</sup></b> [Xe] <b><math>f^1</math></b> Cerium	<b>Pr<sup>3+</sup></b> [Xe] <b><math>f^2</math></b> Praseodymium	<b>Nd<sup>3+</sup></b> [Xe] <b><math>f^3</math></b> Neodymium	<b>Pm<sup>3+</sup></b> [Xe] <b><math>f^4</math></b> Promethium	<b>Sm<sup>3+</sup></b> [Xe] <b><math>f^5</math></b> Samarium	<b>Eu<sup>3+</sup></b> [Xe] <b><math>f^6</math></b> Europium	<b>Gd<sup>3+</sup></b> [Xe] <b><math>f^7</math></b> Gadolinium	<b>Tb<sup>3+</sup></b> [Xe] <b><math>f^8</math></b> Terbium	<b>Dy<sup>3+</sup></b> [Xe] <b><math>f^9</math></b> Dysprosium	<b>Ho<sup>3+</sup></b> [Xe] <b><math>f^{10}</math></b> Holmium	<b>Er<sup>3+</sup></b> [Xe] <b><math>f^{11}</math></b> Erbium	<b>Tm<sup>3+</sup></b> [Xe] <b><math>f^{12}</math></b> Thulium	<b>Yb<sup>3+</sup></b> [Xe] <b><math>f^{13}</math></b> Ytterbium
--	--	---	--	--	--	--	---	--	--	---	--	--

Figure 2: The trivalent lanthanide row and their ground configurations.

electrons might find themselves, this is what is called *configuration-interaction*.

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

$$\hat{\mathcal{H}} = \underbrace{\hat{\mathcal{H}}_k}_{\text{kinetic}} + \underbrace{\hat{\mathcal{H}}_{e:\text{sn}}}_{\text{e:shielded nuc}} + \underbrace{\hat{\mathcal{H}}_{s:o}}_{\substack{\text{spin-orbit} \\ \text{and spin:other-orbit}}} + \underbrace{\hat{\mathcal{H}}_{s:s}}_{\text{spin:spin}} + \underbrace{\hat{\mathcal{H}}_{s:oo \oplus \text{ecs:o}}}_{\substack{\text{spin:other-orbit} \\ \text{ec-correlated-spin:orbit}}} + \underbrace{\hat{\mathcal{H}}_Z}_{\text{Zeeman}} + \underbrace{\hat{\mathcal{H}}_{e:e} + \hat{\mathcal{H}}_{SO(3)} + \hat{\mathcal{H}}_{G_2} + \hat{\mathcal{H}}_{SO(7)} + \hat{\mathcal{H}}_{\text{3-body}} + \hat{\mathcal{H}}_{\text{cf}}}}_{\text{electric interactions}} \quad (1)$$

$$\hat{\mathcal{H}}_k = -\frac{\hbar^2}{2m_e} \sum_{i=1}^n \nabla_i^2 \quad (\text{kinetic energy of } n \text{ valence electrons}) \quad (2)$$

$$\hat{\mathcal{H}}_{e:\text{sn}} = \sum_{i=1}^n V_{\text{sn}}(r_i) \quad (\text{valence-electrons interaction with shielded nuc. charge}) \quad (3)$$

$$\hat{\mathcal{H}}_{s:o} = \begin{cases} \sum_{i=1}^n \xi(r_i) (\hat{s}_i \cdot \hat{l}_i) & \text{with } \xi(r_i) = \frac{\hbar^2}{2m^2c^2r_i} \frac{dV_{\text{sn}}(r_i)}{dr_i} \\ \sum_{i=1}^n \zeta (\hat{s}_i \cdot \hat{l}_i) & \text{with } \zeta \text{ the radial average of } \xi(r_i) \end{cases} \quad (4)$$

$$\hat{\mathcal{H}}_{s:s} = \sum_{k=0,2,4} m^{(k)} \hat{m}_k^{ss} \quad (5)$$

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

$$\hat{\mathcal{H}}_Z = -\vec{B} \cdot \hat{\mu} = \mu_B \vec{B} \cdot (\hat{L} + g_s \hat{S}) \quad (\text{interaction with a magnetic field}) \quad (7)$$

$$\hat{\mathcal{H}}_{e:e} = \sum_{i>j}^{n,n} \frac{e^2}{\|\vec{r}_i - \vec{r}_j\|} = \sum_{k=0,2,4,6} F^{(k)} \hat{f}_k \quad (\text{repulsion between valence electrons}) \quad (8)$$

Let  $\hat{C}(G) :=$  The Casimir operator of group  $G$ .

$$\hat{\mathcal{H}}_{SO(3)} = \alpha \hat{C}(SO(3)) = \alpha \hat{L}^2 \quad (\text{Trees effective operator}) \quad (9)$$

$$\hat{\mathcal{H}}_{G_2} = \beta \hat{C}(G_2) \quad (10)$$

$$\hat{\mathcal{H}}_{SO(7)} = \gamma \hat{C}(SO(7)) \quad (11)$$

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

$$\hat{\mathcal{H}}_{\text{cf}} = \sum_{i=1}^n V_{\text{CF}}(\hat{r}_i) = \sum_{i=1}^n \sum_{k=2,4,6} \sum_{q=-k}^k \mathcal{B}_q^{(k)} \mathcal{C}_q^{(k)}(i) \quad (\text{crystal field interaction with surroundings}) \quad (13)$$

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

Putting all of this together leads to the following effective Hamiltonian as show in [Eqn-1](#), where “v-electrons” is shorthand for valence electrons. It is important 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 having been integrated in the parameters of the effective Hamiltonian. The resulting wavefunctions being solely concerned with the angular dependence of the wavefunctions, but modulated by the effects of the radial dependence.

Once all the parameters in this semi-empirical Hamiltonian have been fitted to experimental data what results is a Hamiltonian such as the one for  $\text{Pr}^{3+}$  in  $\text{LaF}_3$  shown

in [Fig-3](#). Before we go on to explain in some detail each of the terms included in this Hamiltonian, let us continue to explain the basis used in calculations.

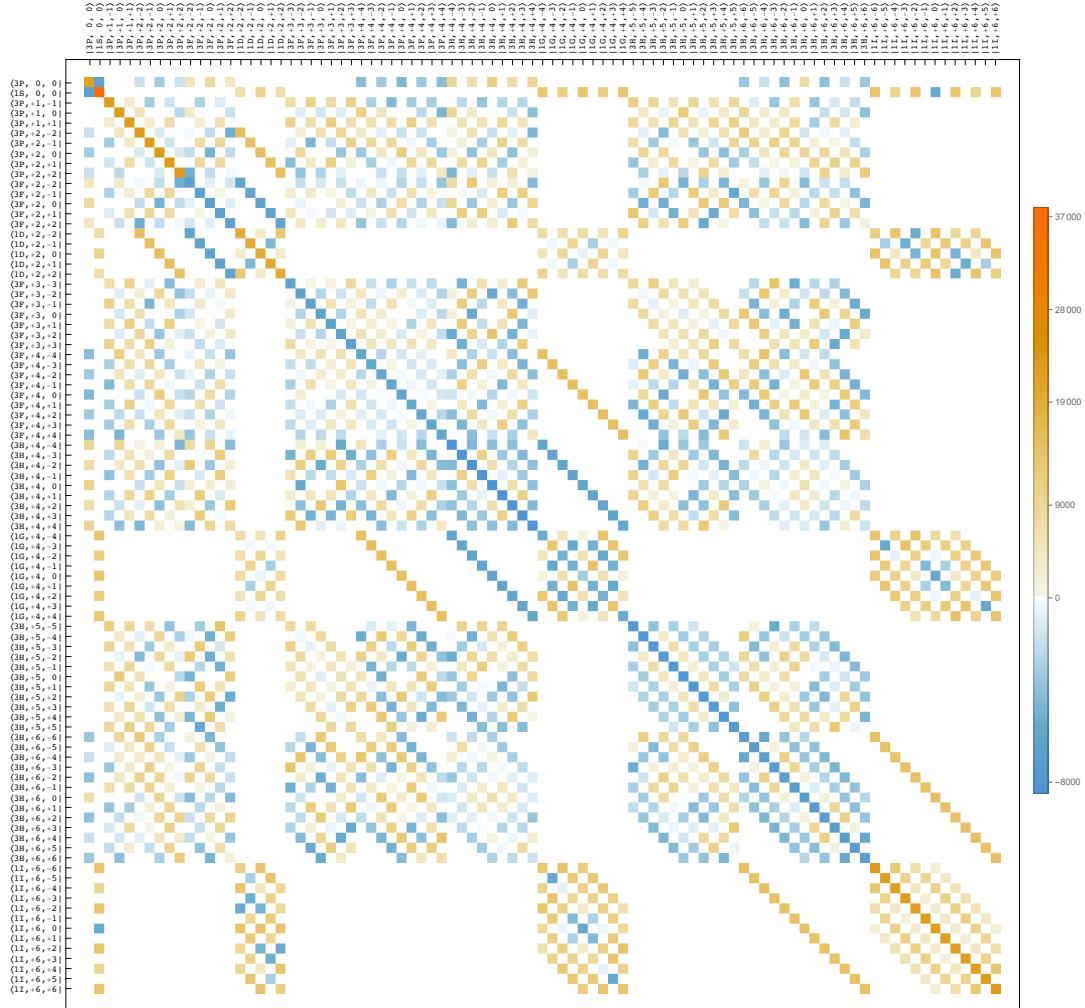


Figure 3: The matrix representation of  $\hat{\mathcal{H}}$  for  $\text{Pr}^{3+}$  in  $\text{LaF}_3$  in the  $|LSJM_J\rangle$  basis.

## 2 LS coupling basis

In choosing a coupling scheme (or equivalently, choosing a basis in which to represent the Hamiltonian), there are a myriad options; all of them legitimate in their own right. The art of choosing a useful coupling scheme is that of proposing a basis for the angular part of the wavefunctions that will be close to the actual eigenstates of the system. It being necessary to calculate the matrix elements of the relevant operators, choosing a coupling scheme may also be justified by the ease by which these can be calculated.

`qlanth` uses *LS* coupling for its calculations. In *LS* coupling all the orbital angular momenta are added to form the total orbital angular momentum  $L$ , all the spin angular momenta are added to form the total spin angular momentum  $S$ , and finally these two angular momenta are then added together to form the total angular momentum  $J$ . The exclusion principle is taken into account in limiting the possible *LS* terms, and demands no further restrictions. Finally this total angular momentum  $J$  is complemented with the quantum number<sup>14</sup>  $M_J$  describing the projection of  $J$  along the z-axis.

It is worthwhile remembering here the spectroscopic hierarchy of descriptive elements: **terms** correspond to  $|LS\rangle$  (also noted as  ${}^{2S+1}\text{L}$ ), **levels** correspond to  $|LSJ\rangle$  (also noted as  ${}^{2S+1}\text{L}_J$ ), and **states** correspond to  $|LSJM_J\rangle$  (also noted as  ${}^{2S+1}\text{L}_{J,M_J}$ ). [Fig-4](#) shows an example of the relationship between a term and its associated levels and states.

In principle the  $|LSJM_J\rangle$  description is the primordial one, the  $|LSJ\rangle$  resulting from neglecting all parts of the Hamiltonian that have no spherical symmetry, and the  $|LS\rangle$  resulting from further neglecting all terms that couple the spin and orbital angular momenta. Note that a *state* is not an *eigen-state*; all of these are assumed to be basis vectors in the type of description attached to them.

<sup>14</sup> A *good* quantum number is any eigenvalue of an operator that commutes with the Hamiltonian; in other words, they are conserved quantities.

Whereas all four quantum numbers  $|LSJM_J\rangle$  are required to specify a state, one may, however, use two simpler descriptions as the situation merits. When all the parts of the Hamiltonian without spherical symmetry are excluded, then a description in terms of  $|LSJ\rangle$  levels is sufficient, the  $M_J$  quantum numbers being redundant and with  $J$  being a good quantum number. In a second scenario, when in addition to neglecting all parts without spherical symmetry, one also neglects all parts of the Hamiltonian that couple the spin and orbital degrees of freedom, then the  $|LS\rangle$  terms constitute the most parsimonious description, with  $L$  and  $S$  being separately conserved quantities.

When a certain level of description has been adopted one can then assume (at one's own peril) that single states, levels, or terms are actual *eigen*-states/levels/terms of the system at hand. This assumption results in simple transition rules between states/levels/terms. One may, however, within each level of description, take an alternate route, the *intermediate coupling* route, of seeing how the different states/levels/terms mix in the eigenstates found by diagonalizing the appropriate Hamiltonian. This results in a more detailed description at the cost of increased complexity.

## 2.1 $|LSJM_J\rangle$ states

The basis vectors of the  $|LSJM_J\rangle$  basis are common eigenvectors of the operators  $\hat{L}^2$ ,  $\hat{S}^2$ ,  $\hat{J}^2$ , and  $\hat{J}_z$ . They are formed starting from the allowed  $LS$  terms in a given configuration, and are then completed with attendant  $J$  and  $M_J$  quantum numbers. The  $LS$  terms allowed in each configuration  $f^n$  are obtained from tables that originate from the original work by Nielson and Koster [NK63]. In **qlanth** these terms are parsed from the file **B1F\_ALL.TXT** which is part of the doctoral research of Dobromir Velkov (under the advisory of Brian Judd) [Vel00] in which he calculated anew the coefficients of fractional parentage.

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 paths 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 further meaning to these integers, except that of discriminating between degenerate terms.

The following are all the  $LS$  terms in the  $f^n$  configurations. In the notation used, the superscript index before the letter notes the spin multiplicity  $2S + 1$ , the roman letter indicates the value of  $L$  in spectroscopic notation ( $S \rightarrow 1, P \rightarrow 2, D \rightarrow 3, F \rightarrow 4, G \rightarrow 5, H \rightarrow 6, I \rightarrow 7, K \rightarrow 8, L \rightarrow 9, M \rightarrow 10, N \rightarrow 11, O \rightarrow 12, Q \rightarrow 3, R \rightarrow 14, T \rightarrow 15, U \rightarrow 16, V \rightarrow 17$ ), and the final integer (if present) is the label that discriminates between several degenerate  $LS$  and must not be confused with a value of  $J$ . This last index we frequently label in the equations contained in this document with the greek letter  $\alpha$  (sadly, for historical reasons, we prepend it, rather than append it).

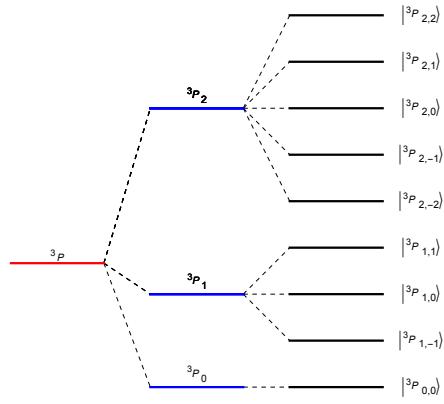


Figure 4: Levels and states associated with the  ${}^3P$  term in  $f^2$ .

$f^0$ (1 LS term)
${}^1S$
$f^1$ (1 LS term)
${}^2F$
$f^2$ (7 LS terms)

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

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

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

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

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

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

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

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

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

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

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

$\underline{f}^8$   
(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)

---

$^6P, ^6F, ^6H, ^4S, ^4P_1, ^4P_2, ^4D_1, ^4D_2, ^4D_3, ^4F_1, ^4F_2, ^4F_3, ^4F_4, ^4G_1, ^4G_2, ^4G_3, ^4G_4, ^4H_1, ^4H_2, ^4H_3, ^4I_1, ^4I_2, ^4I_3, ^4K_1, ^4K_2, ^4L, ^4M, ^2P_1, ^2P_2, ^2P_3, ^2P_4, ^2D_1, ^2D_2, ^2D_3, ^2D_4, ^2D_5, ^2F_1, ^2F_2, ^2F_3, ^2F_4, ^2F_5, ^2F_6, ^2F_7, ^2G_1, ^2G_2, ^2G_3, ^2G_4, ^2G_5, ^2G_6, ^2H_1, ^2H_2, ^2H_3, ^2H_4, ^2H_5, ^2H_6, ^2H_7, ^2I_1, ^2I_2, ^2I_3, ^2I_4, ^2I_5, ^2K_1, ^2K_2, ^2K_3, ^2K_4, ^2K_5, ^2L_1, ^2L_2, ^2L_3, ^2M_1, ^2M_2, ^2N, ^2O$

---

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

---

$^5S, ^5D, ^5F, ^5G, ^5I, ^3P_1, ^3P_2, ^3P_3, ^3D_1, ^3D_2, ^3F_1, ^3F_2, ^3F_3, ^3F_4, ^3G_1, ^3G_2, ^3G_3, ^3H_1, ^3H_2, ^3H_3, ^3H_4, ^3I_1, ^3I_2, ^3K_1, ^3K_2, ^3L, ^3M, ^1S_1, ^1S_2, ^1D_1, ^1D_2, ^1D_3, ^1D_4, ^1F, ^1G_1, ^1G_2, ^1G_3, ^1G_4, ^1H_1, ^1H_2, ^1I_1, ^1I_2, ^1I_3, ^1K, ^1L_1, ^1L_2, ^1N$

---

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

---

$^4S, ^4D, ^4F, ^4G, ^4I, ^2P, ^2D_1, ^2D_2, ^2F_1, ^2F_2, ^2G_1, ^2G_2, ^2H_1, ^2H_2, ^2I, ^2K, ^2L$

---

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

---

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

---

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

---

$^2F$

---

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

---

$^1S$

---

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

```

1 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.";
2 AllowedNKSLTerms[numE_] := Map[First, CFPTerms[Min[numE, 14-numE]]];
3 AllowedNKSLTerms[0] = {"1S"};
4 AllowedNKSLTerms[14] = {"1S"};
```

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

$(J = 0)$   
(2 states)

---

$|^3P_{0,0}\rangle, |^1S_{0,0}\rangle$

---

$(J = 1)$ (3 states)
$ ^3P_{1,-1}\rangle,  ^3P_{1,0}\rangle,  ^3P_{1,1}\rangle$

$(J = 2)$ (15 states)
$ ^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 states)
$ ^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 states)
$ ^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 states)
$ ^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 states)
$ ^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 an example of the bases ordering used in **qlanth**. Notice how the basis vectors are sorted in order of increasing  $J$ , so that for instance not all of the basis states associated with the  ${}^3P$   $LS$  term are contiguous. Within each group for a given  $J$  the basis kets are then ordered in decreasing  $S$ , then ordered in increasing  $L$ , and then according to  $M_J$ .

In **qlanth** the ordered basis used for a given  $f^n$  is provided by **BasisLSJMJ** which provides a list with  $\binom{14}{n}$  elements.

```

1 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}.
2 The option ''AsAssociation'' can be set to True to return the basis
as an association with the keys corresponding to values of J and
the values lists with the corresponding {L, S, J, MJ} list. The
default of this option is False.
3 ";
4 Options[BasisLSJMJ] = {"AsAssociation" -> False};
5 BasisLSJMJ[numE_, OptionsPattern[]] := Module[
6   {energyStatesTable, basis, idx1},
7   (
8     energyStatesTable = BasisTableGenerator[numE];
9     basis = Table[
10       energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
```

```

11 {idx1, 1, Length[AllowedJ[numE]]}] ;
12 basis = Flatten[basis, 1];
13 If[OptionValue["AsAssociation"],
14 (
15 Js = AllowedJ[numE];
16 basis = Table[(J -> Select[basis, #[[2]] == J &]), {J, Js}];
17 basis = Association[basis];
18 )
19 ];
20 Return[basis];
21 )
22 ];

```

## 2.2 More quantum numbers

Besides using an integer which solves the problem of discriminating between degenerate  $LS$  terms by enumerating them, it is also possible to add more useful labels that reflect additional symmetries that the f-electron basis states have in the groups  $S\mathcal{O}(7)$  (the Lie group of rotations in seven dimensions) and  $\mathcal{G}_2$  (the rank-2 exceptional simple Lie group).

### 2.2.1 Seniority $\nu$

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

These rules seem to be ad-hoc, but they are useful in dealing with the degeneracies in the  $LS$  terms as they arrive going up the configurations. It provides a useful way of tracking what happens to each *branch* of the coupling tree as it grows and withers with increasing number of electrons.

There is, however, a deeper meaning to the seniority number. It can be shown that the seniority number (more exactly a quantity related to it) is a sort of spin, a *quasi-spin*, where the spin projections along the ‘z-axis’ correspond to different number of electrons in  $f^n$  configurations [Jud67]. This is a consequence of the exclusion principle. It is also useful to relate matrix elements of operators in one configuration to those in another, through the use of the Wigner-Eckart theorem. This is an interesting and useful theoretical construct, but the method of fractional parentage (which is what is implemented in **qlanth**) is equally adequate, albeit being somewhat less parsimonious than what the quasi-spin view that seniority can provide. As such **qlanth** does not use the seniority numbers that are associated with each  $LS$  term<sup>15</sup>. However, in **qlanth** the seniority of a given  $LS$  term can be obtained using the function **Seniority**.

```

1 Seniority::usage = "Seniority[LS] returns the seniority of the given
                     term.";
2 Seniority[LS_] := FindNKLSTerm[LS][[1, 2]];

```

### 2.2.2 $\mathcal{U}$ and $\mathcal{W}$

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

In **qlanth** the  $\mathcal{W}$  and  $\mathcal{U}$  are used in order to determine the matrix elements of the  $\hat{C}(S\mathcal{O}(7))$  and  $\hat{C}(\mathcal{G}_2)$  Casimir operators. These labels can be retrieved, for a given  $LS$  string, using the function **FindNKLSTerm**.

<sup>15</sup> Except for calculating the coefficients of fractional parentage beyond  $f^7$ , which are useful, but not essential to the calculations of **qlanth**.

```

1 FindNKLSTerm::usage = "Given the string LS FindNKLSTerm[SL] returns
2   all the terms that are compatible with it. This is only for f^n
3   configurations. The provided terms might belong to more than one
4   configuration. The function returns a list with elements of the
5   form {LS, seniority, W, U}.";
6
7 FindNKLSTerm[SL_] := Module[
8   {NKterms, n},
9   (
10   n = 7;
11   NKterms = {};
12   Map[
13     If[! StringFreeQ[First[#], SL],
14       If[ToExpression[Part[#, 2]] <= n,
15         NKterms = Join[NKterms, {#}, 1]
16       ]
17     ] &,
18   fnTermLabels
19   ];
20   NKterms = DeleteCases[NKterms, {}];
21   NKterms
22 )
23 ];

```

### 2.3 $|LSJ\rangle$ levels

When the Hamiltonian only includes spherically symmetric terms (or what is the same, when the crystal field is neglected) then the  $M_J$  quantum numbers in the  $|LSJM_J\rangle$  basis states are redundant. This permits a simplified description in terms of  $|LSJ\rangle$  levels. The following are the different  $^{2S+1}L_J$  levels that span the eigenvectors that result from diagonalizing the Hamiltonian in the level description, these may also be termed *multiplets*. (In these we have excluded the indices that distinguish between degenerate LS terms)

$\underline{f}^1$  (2 LSJ levels)

---

${}^2F_{5/2}, {}^2F_{7/2}$

$\underline{f}^2$  (13 LSJ levels)

---

${}^3P_0, {}^1S_0, {}^3P_1, {}^3P_2, {}^3F_2, {}^1D_2, {}^3F_3, {}^3F_4, {}^3H_4, {}^1G_4, {}^3H_5, {}^3H_6, {}^1I_6$

$\underline{f}^3$  (41 LSJ levels)

---

${}^4D_{1/2}, {}^2P_{1/2}, {}^4S_{3/2}, {}^4D_{3/2}, {}^4F_{3/2}, {}^2P_{3/2}, {}^2D_{3/2}, {}^2D_{3/2}, {}^4D_{5/2}, {}^4F_{5/2}, {}^4G_{5/2}, {}^2D_{5/2}, {}^2D_{5/2},$   
 ${}^2F_{5/2}, {}^2F_{5/2}, {}^4D_{7/2}, {}^4F_{7/2}, {}^4G_{7/2}, {}^2F_{7/2}, {}^2F_{7/2}, {}^2G_{7/2}, {}^4F_{9/2}, {}^4G_{9/2}, {}^4I_{9/2}, {}^2G_{9/2},$   
 ${}^2G_{9/2}, {}^2H_{9/2}, {}^2H_{9/2}, {}^4G_{11/2}, {}^4I_{11/2}, {}^2H_{11/2}, {}^2H_{11/2}, {}^2I_{11/2}, {}^4I_{13/2}, {}^2I_{13/2}, {}^2K_{13/2}, {}^4I_{15/2},$   
 ${}^2K_{15/2}, {}^2L_{15/2}, {}^2L_{17/2}$

$\underline{f}^4$  (107 LSJ levels)

---

${}^5D_0, {}^3P_0, {}^3P_0, {}^3P_0, {}^1S_0, {}^1S_0, {}^5D_1, {}^5F_1, {}^3P_1, {}^3P_1, {}^3P_1, {}^3D_1, {}^5S_2, {}^5D_2, {}^5F_2, {}^5G_2, {}^3P_2,$   
 ${}^3P_2, {}^3P_2, {}^3D_2, {}^3D_2, {}^3F_2, {}^3F_2, {}^3F_2, {}^3F_2, {}^1D_2, {}^1D_2, {}^1D_2, {}^1D_2, {}^5D_3, {}^5F_3, {}^5G_3, {}^3D_3, {}^3D_3,$   
 ${}^3F_3, {}^3F_3, {}^3F_3, {}^3F_3, {}^3G_3, {}^3G_3, {}^3G_3, {}^3G_3, {}^1F_3, {}^5D_4, {}^5F_4, {}^5G_4, {}^5I_4, {}^3F_4, {}^3F_4, {}^3F_4, {}^3G_4, {}^3G_4,$   
 ${}^3G_4, {}^3H_4, {}^3H_4, {}^3H_4, {}^3H_4, {}^1G_4, {}^1G_4, {}^1G_4, {}^1G_4, {}^5F_5, {}^5G_5, {}^5I_5, {}^3G_5, {}^3G_5, {}^3G_5, {}^3H_5, {}^3H_5,$   
 ${}^3H_5, {}^3H_5, {}^3I_5, {}^3I_5, {}^1H_5, {}^1H_5, {}^5G_6, {}^5I_6, {}^3H_6, {}^3H_6, {}^3H_6, {}^3I_6, {}^3K_6, {}^3K_6, {}^1I_6, {}^1I_6, {}^1I_6,$   
 ${}^5I_7, {}^3I_7, {}^3I_7, {}^3K_7, {}^3K_7, {}^3L_7, {}^1K_7, {}^5I_8, {}^3K_8, {}^3K_8, {}^3L_8, {}^3M_8, {}^1L_8, {}^3L_9, {}^3M_9, {}^3M_{10}, {}^1N_{10}$

$\underline{f}^5$  (198 LSJ levels)

---

${}^6F_{1/2}, {}^4P_{1/2}, {}^4P_{1/2}, {}^4D_{1/2}, {}^4D_{1/2}, {}^4D_{1/2}, {}^2P_{1/2}, {}^2P_{1/2}, {}^2P_{1/2}, {}^6P_{3/2}, {}^6F_{3/2}, {}^4S_{3/2},$   
 ${}^4P_{3/2}, {}^4P_{3/2}, {}^4D_{3/2}, {}^4D_{3/2}, {}^4D_{3/2}, {}^4F_{3/2}, {}^4F_{3/2}, {}^4F_{3/2}, {}^4F_{3/2}, {}^2P_{3/2}, {}^2P_{3/2}, {}^2P_{3/2},$   
 ${}^2D_{3/2}, {}^2D_{3/2}, {}^2D_{3/2}, {}^2D_{3/2}, {}^6P_{5/2}, {}^6F_{5/2}, {}^6H_{5/2}, {}^4P_{5/2}, {}^4P_{5/2}, {}^4D_{5/2},$   
 ${}^4D_{5/2}, {}^4F_{5/2}, {}^4F_{5/2}, {}^4F_{5/2}, {}^4F_{5/2}, {}^4G_{5/2}, {}^4G_{5/2}, {}^4G_{5/2}, {}^2D_{5/2}, {}^2D_{5/2}, {}^2D_{5/2},$

$^2D_{5/2}, ^2D_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^6P_{7/2}, ^6H_{7/2}, ^4D_{7/2},$
$^4D_{7/2}, ^4D_{7/2}, ^4F_{7/2}, ^4F_{7/2}, ^4F_{7/2}, ^4G_{7/2}, ^4G_{7/2}, ^4G_{7/2}, ^4H_{7/2}, ^4H_{7/2}, ^4H_{7/2},$
$^2F_{7/2}, ^2F_{7/2}, ^2F_{7/2}, ^2F_{7/2}, ^2F_{7/2}, ^2F_{7/2}, ^2G_{7/2}, ^2G_{7/2}, ^2G_{7/2}, ^2G_{7/2}, ^2G_{7/2},$
$^6F_{9/2}, ^6H_{9/2}, ^4F_{9/2}, ^4F_{9/2}, ^4F_{9/2}, ^4F_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4H_{9/2},$
$^4I_{9/2}, ^4I_{9/2}, ^4I_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^2H_{9/2},$
$^2H_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^6H_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4H_{11/2},$
$^4H_{11/2}, ^4I_{11/2}, ^4I_{11/2}, ^4K_{11/2}, ^4K_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2},$
$^2H_{11/2}, ^2H_{11/2}, ^2I_{11/2}, ^2I_{11/2}, ^2I_{11/2}, ^6H_{13/2}, ^4H_{13/2}, ^4H_{13/2}, ^4I_{13/2},$
$^4I_{13/2}, ^4I_{13/2}, ^4K_{13/2}, ^4K_{13/2}, ^4L_{13/2}, ^2I_{13/2}, ^2I_{13/2}, ^2I_{13/2}, ^2I_{13/2}, ^2K_{13/2},$
$^2K_{13/2}, ^2K_{13/2}, ^4I_{15/2}, ^4I_{15/2}, ^4I_{15/2}, ^4I_{15/2}, ^4K_{15/2}, ^4K_{15/2}, ^4L_{15/2}, ^4M_{15/2},$
$^2K_{15/2}, ^2K_{15/2}, ^2K_{15/2}, ^2K_{15/2}, ^2L_{15/2}, ^2L_{15/2}, ^4K_{17/2}, ^4K_{17/2}, ^4L_{17/2},$
$^4M_{17/2}, ^2L_{17/2}, ^2L_{17/2}, ^2M_{17/2}, ^2M_{17/2}, ^4L_{19/2}, ^4M_{19/2}, ^2M_{19/2}, ^2M_{19/2}, ^2N_{19/2},$
$^4M_{21/2}, ^2N_{21/2}, ^2O_{21/2}, ^2O_{23/2}$

### $\underline{f}^6$ (295 LSJ levels)

$^7F_0, ^5D_0, ^5D_0, ^3P_0, ^3P_0, ^3P_0, ^3P_0, ^1S_0, ^1S_0, ^1S_0, ^7F_1, ^5P_1, ^5D_1,$
$^5D_1, ^5F_1, ^5F_1, ^3P_1, ^3P_1, ^3P_1, ^3P_1, ^3P_1, ^3D_1, ^3D_1, ^3D_1, ^3D_1, ^1P_1, ^7F_2,$
$^5S_2, ^5P_2, ^5D_2, ^5D_2, ^5F_2, ^5F_2, ^5G_2, ^5G_2, ^5G_2, ^3P_2, ^3P_2, ^3P_2, ^3P_2, ^3D_2,$
$^3D_2, ^3D_2, ^3F_2, ^3F_2, ^3F_2, ^3F_2, ^3F_2, ^3F_2, ^3F_2, ^1D_2, ^1D_2, ^1D_2, ^1D_2, ^1D_2,$
$^7F_3, ^5P_3, ^5D_3, ^5D_3, ^5F_3, ^5F_3, ^5G_3, ^5G_3, ^5H_3, ^5H_3, ^3D_3, ^3D_3, ^3D_3, ^3D_3,$
$^3F_3, ^3F_3, ^3F_3, ^3F_3, ^3F_3, ^3F_3, ^3F_3, ^3G_3, ^3G_3, ^3G_3, ^3G_3, ^3G_3, ^3G_3, ^1F_3,$
$^1F_3, ^1F_3, ^7F_4, ^5D_4, ^5D_4, ^5D_4, ^5F_4, ^5F_4, ^5G_4, ^5G_4, ^5G_4, ^5H_4, ^5H_4, ^5I_4, ^5I_4,$
$^3F_4, ^3F_4, ^3F_4, ^3F_4, ^3F_4, ^3G_4, ^3G_4, ^3G_4, ^3G_4, ^3G_4, ^3G_4, ^3G_4, ^3H_4, ^3H_4,$
$^3H_4, ^3H_4, ^3H_4, ^1G_4, ^1G_4, ^1G_4, ^1G_4, ^1G_4, ^1G_4, ^1G_4, ^7F_5, ^5F_5, ^5F_5, ^5G_5,$
$^5G_5, ^5H_5, ^5H_5, ^5I_5, ^5I_5, ^5K_5, ^3G_5, ^3G_5, ^3G_5, ^3G_5, ^3G_5, ^3H_5, ^3H_5, ^3H_5,$
$^3H_5, ^3H_5, ^3H_5, ^3H_5, ^3I_5, ^3I_5, ^3I_5, ^3I_5, ^1I_5, ^1H_5, ^1H_5, ^1H_5, ^1H_5, ^7F_6,$
$^5G_6, ^5H_6, ^5H_6, ^5I_6, ^5I_6, ^5K_6, ^5L_6, ^3H_6, ^3H_6, ^3H_6, ^3H_6, ^3H_6, ^3H_6, ^3I_6,$
$^3I_6, ^3I_6, ^3I_6, ^3K_6, ^3K_6, ^3K_6, ^3K_6, ^3K_6, ^1I_6, ^1I_6, ^1I_6, ^1I_6, ^1I_6, ^1I_6, ^5H_7,$
$^5I_7, ^5I_7, ^5K_7, ^5L_7, ^3I_7, ^3I_7, ^3I_7, ^3I_7, ^3I_7, ^3K_7, ^3K_7, ^3K_7, ^3K_7, ^3L_7,$
$^3L_7, ^3L_7, ^1K_7, ^1K_7, ^1K_7, ^5I_8, ^5I_8, ^5K_8, ^5L_8, ^3K_8, ^3K_8, ^3K_8, ^3K_8, ^3L_8, ^3L_8,$
$^3M_8, ^3M_8, ^1L_8, ^1L_8, ^1L_8, ^1L_8, ^5K_9, ^5L_9, ^3L_9, ^3L_9, ^3M_9, ^3M_9, ^3N_9, ^1M_9,$
$^1M_9, ^5L_{10}, ^3M_{10}, ^3M_{10}, ^3N_{10}, ^3O_{10}, ^1N_{10}, ^1N_{10}, ^3N_{11}, ^3O_{12}, ^1Q_{12}$

### $\underline{f}^7$ (327 LSJ levels)

$^6D_{1/2}, ^6F_{1/2}, ^4P_{1/2}, ^4P_{1/2}, ^4D_{1/2}, ^4D_{1/2}, ^4D_{1/2}, ^4D_{1/2}, ^2S_{1/2}, ^2S_{1/2},$
$^2P_{1/2}, ^2P_{1/2}, ^2P_{1/2}, ^2P_{1/2}, ^6P_{3/2}, ^6D_{3/2}, ^6F_{3/2}, ^6G_{3/2}, ^4S_{3/2}, ^4P_{3/2},$
$^4P_{3/2}, ^4D_{3/2}, ^4D_{3/2}, ^4D_{3/2}, ^4D_{3/2}, ^4F_{3/2}, ^4F_{3/2}, ^4F_{3/2}, ^4F_{3/2}, ^2P_{3/2},$
$^2P_{3/2}, ^2P_{3/2}, ^2D_{3/2}, ^2D_{3/2}, ^2D_{3/2}, ^2D_{3/2}, ^2D_{3/2}, ^6P_{5/2}, ^6D_{5/2},$
$^6F_{5/2}, ^6G_{5/2}, ^6H_{5/2}, ^4P_{5/2}, ^4P_{5/2}, ^4D_{5/2}, ^4D_{5/2}, ^4D_{5/2}, ^4D_{5/2},$
$^4F_{5/2}, ^4F_{5/2}, ^4G_{5/2}, ^4G_{5/2}, ^4G_{5/2}, ^4G_{5/2}, ^4G_{5/2}, ^2D_{5/2}, ^2D_{5/2},$
$^2D_{5/2}, ^2D_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2}, ^2F_{5/2},$
$^8S_{7/2}, ^6P_{7/2}, ^6D_{7/2}, ^6F_{7/2}, ^6H_{7/2}, ^6I_{7/2}, ^4D_{7/2}, ^4D_{7/2}, ^4D_{7/2},$
$^4D_{7/2}, ^4D_{7/2}, ^4D_{7/2}, ^4D_{7/2}, ^4D_{7/2}, ^4D_{7/2}, ^4D_{7/2}, ^4D_{7/2}, ^4D_{7/2},$
$^4F_{7/2}, ^4F_{7/2}, ^4F_{7/2}, ^4F_{7/2}, ^4F_{7/2}, ^4G_{7/2}, ^4G_{7/2}, ^4G_{7/2}, ^4G_{7/2},$
$^4G_{7/2}, ^4H_{7/2}, ^4H_{7/2}, ^4H_{7/2}, ^4H_{7/2}, ^2F_{7/2}, ^2F_{7/2}, ^2F_{7/2}, ^2F_{7/2},$
$^2F_{7/2}, ^2F_{7/2}, ^2G_{7/2}, ^2G_{7/2}, ^2G_{7/2}, ^2G_{7/2}, ^2G_{7/2}, ^2G_{7/2}, ^2G_{7/2},$
$^6D_{9/2}, ^6F_{9/2}, ^6G_{9/2}, ^6H_{9/2}, ^6I_{9/2}, ^4F_{9/2}, ^4F_{9/2}, ^4F_{9/2}, ^4F_{9/2},$
$^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2}, ^4G_{9/2},$
$^4G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2}, ^2G_{9/2},$
$^2H_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^6F_{11/2},$
$^6G_{11/2}, ^6H_{11/2}, ^6I_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2},$
$^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2}, ^4G_{11/2},$
$^4I_{11/2}, ^4I_{11/2}, ^4I_{11/2}, ^4I_{11/2}, ^4I_{11/2}, ^4K_{11/2}, ^4K_{11/2}, ^4K_{11/2},$
$^4K_{11/2}, ^4K_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2},$
$^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2H_{11/2},$
$^2I_{11/2}, ^2I_{11/2}, ^2I_{11/2}, ^2I_{11/2}, ^2I_{11/2}, ^2I_{11/2}, ^2I_{11/2}, ^2I_{11/2},$
$^4I_{13/2}, ^4I_{13/2}, ^4I_{13/2}, ^4I_{13/2}, ^4I_{13/2}, ^4K_{13/2}, ^4K_{13/2}, ^4K_{13/2},$
$^4L_{13/2}, ^4L_{13/2}, ^4L_{13/2}, ^4L_{13/2}, ^4L_{13/2}, ^2I_{13/2}, ^2I_{13/2},$
$^2I_{13/2}, ^2I_{13/2}, ^2I_{13/2}, ^2I_{13/2}, ^2I_{13/2}, ^2K_{13/2}, ^2K_{13/2},$
$^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2},$
$^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2}, ^2K_{13/2},$
$^4K_{15/2}, ^4L_{15/2}, ^4L_{15/2}, ^4L_{15/2}, ^4M_{15/2}, ^4M_{15/2}, ^2K_{15/2},$
$^2K_{15/2}, ^2K_{15/2}, ^2K_{15/2}, ^2K_{15/2}, ^2K_{15/2}, ^2K_{15/2},$
$^2K_{15/2}, ^2L_{15/2}, ^2L_{15/2}, ^2L_{15/2}, ^2L_{15/2}, ^6I_{17/2}, ^4K_{17/2},$
$^4K_{17/2}, ^4L_{17/2}, ^4L_{17/2}, ^4L_{17/2},$

$^4L_{17/2}$ , $^4L_{17/2}$ , $^4M_{17/2}$ , $^4N_{17/2}$ , $^2L_{17/2}$ , $^2L_{17/2}$ , $^2L_{17/2}$ , $^2L_{17/2}$ , $^2M_{17/2}$ , $^2M_{17/2}$ , $^2M_{17/2}$ , $^2M_{17/2}$ , $^4L_{19/2}$ , $^4L_{19/2}$ , $^4M_{19/2}$ , $^4N_{19/2}$ , $^2M_{19/2}$ , $^2M_{19/2}$ , $^2M_{19/2}$ , $^2N_{19/2}$ , $^2N_{19/2}$ , $^4M_{21/2}$ , $^4N_{21/2}$ , $^2N_{21/2}$ , $^2N_{21/2}$ , $^2O_{21/2}$ , $^4N_{23/2}$ , $^2O_{23/2}$ , $^2Q_{23/2}$ , $^2Q_{25/2}$
---

### $\underline{f}^8$ (295 LSJ levels)

$^7F_0$ , $^5D_0$ , $^5D_0$ , $^3P_0$ , $^3P_0$ , $^3P_0$ , $^3P_0$ , $^3P_0$ , $^1S_0$ , $^1S_0$ , $^1S_0$ , $^1S_0$ , $^7F_1$ , $^5P_1$ , $^5D_1$ , $^5D_1$ , $^5D_1$ , $^5F_1$ , $^5F_1$ , $^3P_1$ , $^3P_1$ , $^3P_1$ , $^3P_1$ , $^3P_1$ , $^3D_1$ , $^3D_1$ , $^3D_1$ , $^3D_1$ , $^1P_1$ , $^7F_2$ , $^5S_2$ , $^5P_2$ , $^5D_2$ , $^5D_2$ , $^5D_2$ , $^5F_2$ , $^5F_2$ , $^5G_2$ , $^5G_2$ , $^3P_2$ , $^3P_2$ , $^3P_2$ , $^3P_2$ , $^3D_2$ , $^3D_2$ , $^3D_2$ , $^3D_2$ , $^3D_2$ , $^3F_2$ , $^1D_2$ , $^1D_2$ , $^1D_2$ , $^1D_2$ , $^1D_2$ , $^1D_2$ , $^7F_3$ , $^5P_3$ , $^5D_3$ , $^5D_3$ , $^5D_3$ , $^5F_3$ , $^5F_3$ , $^5G_3$ , $^5G_3$ , $^5G_3$ , $^5H_3$ , $^5H_3$ , $^3D_3$ , $^3D_3$ , $^3D_3$ , $^3D_3$ , $^3D_3$ , $^3F_3$ , $^3F_3$ , $^3F_3$ , $^3F_3$ , $^3F_3$ , $^3F_3$ , $^3F_3$ , $^3G_3$ , $^3G_3$ , $^3G_3$ , $^3G_3$ , $^3G_3$ , $^3G_3$ , $^1F_3$ , $^1F_3$ , $^1F_3$ , $^1F_3$ , $^7F_4$ , $^5D_4$ , $^5D_4$ , $^5F_4$ , $^5F_4$ , $^5G_4$ , $^5G_4$ , $^5H_4$ , $^5H_4$ , $^5I_4$ , $^3F_4$ , $^3F_4$ , $^3F_4$ , $^3F_4$ , $^3F_4$ , $^3F_4$ , $^3F_4$ , $^3G_4$ , $^3G_4$ , $^3G_4$ , $^3G_4$ , $^3G_4$ , $^3H_4$ , $^3H_4$ , $^3H_4$ , $^3H_4$ , $^3H_4$ , $^3H_4$ , $^3H_4$ , $^3H_4$ , $^1G_4$ , $^7F_5$ , $^5F_5$ , $^5F_5$ , $^5G_5$ , $^5G_5$ , $^5G_5$ , $^5H_5$ , $^5H_5$ , $^5I_5$ , $^5I_5$ , $^5K_5$ , $^3G_5$ , $^3G_5$ , $^3G_5$ , $^3G_5$ , $^3G_5$ , $^3G_5$ , $^3H_5$ , $^3H_5$ , $^3H_5$ , $^3H_5$ , $^3H_5$ , $^3H_5$ , $^3H_5$ , $^3H_5$ , $^3I_5$ , $^3I_5$ , $^3I_5$ , $^3I_5$ , $^1H_5$ , $^1H_5$ , $^1H_5$ , $^1H_5$ , $^1H_5$ , $^7F_6$ , $^5G_6$ , $^5G_6$ , $^5G_6$ , $^5H_6$ , $^5H_6$ , $^5I_6$ , $^5I_6$ , $^5K_6$ , $^5L_6$ , $^3H_6$ , $^3I_6$ , $^3I_6$ , $^3I_6$ , $^3I_6$ , $^3I_6$ , $^3K_6$ , $^3K_6$ , $^3K_6$ , $^3K_6$ , $^3K_6$ , $^1I_6$ , $^5H_7$ , $^5H_7$ , $^5I_7$ , $^5I_7$ , $^5K_7$ , $^5L_7$ , $^3I_7$ , $^3I_7$ , $^3I_7$ , $^3I_7$ , $^3I_7$ , $^3K_7$ , $^3K_7$ , $^3K_7$ , $^3K_7$ , $^3L_7$ , $^3L_7$ , $^3L_7$ , $^3L_7$ , $^1K_7$ , $^1K_7$ , $^1K_7$ , $^5I_8$ , $^5I_8$ , $^5K_8$ , $^5L_8$ , $^3K_8$ , $^3K_8$ , $^3K_8$ , $^3K_8$ , $^3L_8$ , $^3L_8$ , $^3L_8$ , $^3M_8$ , $^3M_8$ , $^3M_8$ , $^1L_8$ , $^1L_8$ , $^1L_8$ , $^1L_8$ , $^5K_9$ , $^5L_9$ , $^3L_9$ , $^3L_9$ , $^3M_9$ , $^3M_9$ , $^3N_9$ , $^3N_9$ , $^1M_9$ , $^1M_9$ , $^5L_{10}$ , $^3M_{10}$ , $^3M_{10}$ , $^3M_{10}$ , $^3N_{10}$ , $^3O_{10}$ , $^1N_{10}$ , $^1N_{10}$ , $^3N_{11}$ , $^3O_{11}$ , $^1Q_{12}$
--

### $\underline{f}^9$ (198 LSJ levels)

$^6F_{1/2}$ , $^4P_{1/2}$ , $^4P_{1/2}$ , $^4D_{1/2}$ , $^4D_{1/2}$ , $^2P_{1/2}$ , $^2P_{1/2}$ , $^2P_{1/2}$ , $^6P_{3/2}$ , $^4S_{3/2}$ , $^4P_{3/2}$ , $^4P_{3/2}$ , $^4D_{3/2}$ , $^4D_{3/2}$ , $^4D_{3/2}$ , $^4F_{3/2}$ , $^4F_{3/2}$ , $^4F_{3/2}$ , $^2P_{3/2}$ , $^2P_{3/2}$ , $^2P_{3/2}$ , $^2P_{3/2}$ , $^2D_{3/2}$ , $^2D_{3/2}$ , $^2D_{3/2}$ , $^2D_{3/2}$ , $^2D_{3/2}$ , $^6P_{5/2}$ , $^6H_{5/2}$ , $^4P_{5/2}$ , $^4P_{5/2}$ , $^4D_{5/2}$ , $^4D_{5/2}$ , $^4D_{5/2}$ , $^4F_{5/2}$ , $^4F_{5/2}$ , $^4F_{5/2}$ , $^4G_{5/2}$ , $^4G_{5/2}$ , $^4G_{5/2}$ , $^2D_{5/2}$ , $^2D_{5/2}$ , $^2D_{5/2}$ , $^2D_{5/2}$ , $^2D_{5/2}$ , $^2F_{5/2}$ , $^2F_{5/2}$ , $^2F_{5/2}$ , $^2F_{5/2}$ , $^2F_{5/2}$ , $^2F_{5/2}$ , $^6P_{7/2}$ , $^6F_{7/2}$ , $^6H_{7/2}$ , $^4D_{7/2}$ , $^4D_{7/2}$ , $^4D_{7/2}$ , $^4F_{7/2}$ , $^4F_{7/2}$ , $^4F_{7/2}$ , $^4F_{7/2}$ , $^4G_{7/2}$ , $^4G_{7/2}$ , $^4G_{7/2}$ , $^4H_{7/2}$ , $^4H_{7/2}$ , $^4H_{7/2}$ , $^2F_{7/2}$ , $^2F_{7/2}$ , $^2F_{7/2}$ , $^2F_{7/2}$ , $^2F_{7/2}$ , $^2F_{7/2}$ , $^2F_{7/2}$ , $^2G_{7/2}$ , $^2G_{7/2}$ , $^2G_{7/2}$ , $^2G_{7/2}$ , $^2G_{7/2}$ , $^2G_{7/2}$ , $^6F_{9/2}$ , $^6H_{9/2}$ , $^4F_{9/2}$ , $^4F_{9/2}$ , $^4F_{9/2}$ , $^4F_{9/2}$ , $^4G_{9/2}$ , $^4G_{9/2}$ , $^4G_{9/2}$ , $^4H_{9/2}$ , $^4H_{9/2}$ , $^4H_{9/2}$ , $^4I_{9/2}$ , $^4I_{9/2}$ , $^4I_{9/2}$ , $^2G_{9/2}$ , $^2G_{9/2}$ , $^2G_{9/2}$ , $^2G_{9/2}$ , $^2G_{9/2}$ , $^2G_{9/2}$ , $^2H_{9/2}$ , $^2H_{9/2}$ , $^2H_{9/2}$ , $^2H_{9/2}$ , $^2H_{9/2}$ , $^2H_{9/2}$ , $^2H_{9/2}$ , $^2H_{9/2}$ , $^6F_{11/2}$ , $^6H_{11/2}$ , $^4G_{11/2}$ , $^4G_{11/2}$ , $^4G_{11/2}$ , $^4G_{11/2}$ , $^4H_{11/2}$ , $^4H_{11/2}$ , $^4H_{11/2}$ , $^4H_{11/2}$ , $^4I_{11/2}$ , $^4I_{11/2}$ , $^4I_{11/2}$ , $^4K_{11/2}$ , $^4K_{11/2}$ , $^2H_{11/2}$ , $^2H_{11/2}$ , $^2H_{11/2}$ , $^2H_{11/2}$ , $^2H_{11/2}$ , $^2H_{11/2}$ , $^2H_{11/2}$ , $^2I_{11/2}$ , $^2I_{11/2}$ , $^2I_{11/2}$ , $^2I_{11/2}$ , $^2I_{11/2}$ , $^6H_{13/2}$ , $^4H_{13/2}$ , $^4H_{13/2}$ , $^4H_{13/2}$ , $^4I_{13/2}$ , $^4I_{13/2}$ , $^4I_{13/2}$ , $^4K_{13/2}$ , $^4K_{13/2}$ , $^4L_{13/2}$ , $^2I_{13/2}$ , $^2I_{13/2}$ , $^2I_{13/2}$ , $^2I_{13/2}$ , $^2K_{13/2}$ , $^2K_{13/2}$ , $^2K_{13/2}$ , $^2K_{13/2}$ , $^2K_{13/2}$ , $^6H_{15/2}$ , $^4I_{15/2}$ , $^4I_{15/2}$ , $^4K_{15/2}$ , $^4K_{15/2}$ , $^4L_{15/2}$ , $^4M_{15/2}$ , $^2K_{15/2}$ , $^2K_{15/2}$ , $^2K_{15/2}$ , $^2K_{15/2}$ , $^2K_{15/2}$ , $^2L_{15/2}$ , $^2L_{15/2}$ , $^2L_{15/2}$ , $^2L_{15/2}$ , $^4K_{17/2}$ , $^4K_{17/2}$ , $^4L_{17/2}$ , $^4M_{17/2}$ , $^2L_{17/2}$ , $^2L_{17/2}$ , $^2L_{17/2}$ , $^2M_{17/2}$ , $^2M_{17/2}$ , $^4L_{19/2}$ , $^4M_{19/2}$ , $^2M_{19/2}$ , $^2M_{19/2}$ , $^2N_{19/2}$ , $^4M_{21/2}$ , $^2N_{21/2}$ , $^2O_{21/2}$ , $^2O_{23/2}$
---

### $\underline{f}^{10}$ (107 LSJ levels)

$^5D_0$ , $^3P_0$ , $^3P_0$ , $^3P_0$ , $^1S_0$ , $^1S_0$ , $^5D_1$ , $^5F_1$ , $^3P_1$ , $^3P_1$ , $^3P_1$ , $^3D_1$ , $^3D_1$ , $^5S_2$ , $^5D_2$ , $^5F_2$ , $^5G_2$ , $^3P_2$ , $^3P_2$ , $^3D_2$ , $^3D_2$ , $^3F_2$ , $^3F_2$ , $^3F_2$ , $^1D_2$ , $^1D_2$ , $^1D_2$ , $^5D_3$ , $^5F_3$ , $^5G_3$ , $^3D_3$ , $^3D_3$ , $^3F_3$ , $^3F_3$ , $^3F_3$ , $^3G_3$ , $^3G_3$ , $^3G_3$ , $^1F_3$ , $^5D_4$ , $^5F_4$ , $^5G_4$ , $^5I_4$ , $^3F_4$ , $^3F_4$ , $^3F_4$ , $^3G_4$ , $^3G_4$ , $^3G_4$ , $^3H_4$ , $^3H_4$ , $^3H_4$ , $^1G_4$ , $^1G_4$ , $^1G_4$ , $^1G_4$ , $^5F_5$ , $^5G_5$ , $^5I_5$ , $^3G_5$ , $^3G_5$ , $^3G_5$ , $^3H_5$ , $^3H_5$ , $^3H_5$ , $^3H_5$ , $^3I_5$ , $^1H_5$ , $^1H_5$ , $^5G_6$ , $^5I_6$ , $^3H_6$ , $^3H_6$ , $^3H_6$ , $^3I_6$ , $^3I_6$ , $^3K_6$ , $^3K_6$ , $^1I_6$ , $^1I_6$ , $^1I_6$ , $^5L_7$ , $^3I_7$ , $^3I_7$ , $^3K_7$ , $^3K_7$ , $^1K_7$ , $^3I_8$ , $^3K_8$ , $^3L_8$ , $^3M_8$ , $^1L_8$ , $^1L_8$ , $^3L_9$ , $^3M_9$ , $^3M_9$ , $^3M_9$ , $^1N_{10}$
--

### $\underline{f}^{11}$ (41 LSJ levels)

$^4D_{1/2}$ , $^2P_{1/2}$ , $^4S_{3/2}$ , $^4D_{3/2}$ , $^4F_{3/2}$ , $^2P_{3/2}$ , $^2D_{3/2}$ , $^2D_{3/2}$ , $^4D_{5/2}$ , $^4F_{5/2}$ , $^4G_{5/2}$ , $^2D_{5/2}$ , $^2D_{5/2}$ , $^2F_{5/2}$ , $^2F_{5/2}$ , $^4D_{7/2}$ , $^4F_{7/2}$ , $^4G_{7/2}$ , $^2F_{7/2}$ , $^2F_{7/2}$ , $^2G_{7/2}$ , $^2G_{7/2}$ , $^4F_{9/2}$ , $^4G_{9/2}$ , $^4I_{9/2}$ , $^2G_{9/2}$
--

$$^2G_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^4G_{11/2}, ^4I_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2I_{11/2}, ^4I_{13/2}, ^2I_{13/2}, ^2K_{13/2}, ^4I_{15/2},$$

$$^2K_{15/2}, ^2L_{15/2}, ^2L_{17/2}$$

$f^{12}$  (13 LSJ levels)

$^3P_0, ^1S_0, ^3P_1, ^3P_2, ^3F_2, ^1D_2, ^3F_3, ^3F_4, ^3H_4, ^1G_4, ^3H_5, ^3H_6, ^1I_6$

$f^{13}$  (2 LSJ levels)

$^2F_{5/2}, ^2F_{7/2}$

The level picture is a much more frugal description of the eigenstates. Not only are the number of basis elements that need to be considered much less than otherwise, but also the diagonalization is more efficient since it can be carried out within subspaces of shared  $J$ . One needs, however, to use adequate degeneracy factors in the relevant calculations.

In `qlanth` the function `BasisLSJ` can be used to retrieve the ordered basis that is used for the intermediate coupling description in terms of levels.

```

1 BasisLSJ::usage = "BasisLSJ[numE] returns the level basis LSJ. 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}.
2 The option ''AsAssociation'' can be set to True to return the basis
   as an association with the keys being the allowed J values. The
   default is False.
3 ";
4 Options[BasisLSJ]={ "AsAssociation" -> False};
5 BasisLSJ[numE_,OptionsPattern[]]:=Module[
6   {Js,basis},
7   (
8     Js= AllowedJ[numE];
9     basis=BasisLSJMJ[numE,"AsAssociation" -> False];
10    basis=DeleteDuplicates[{#[[1]],#[[2]]} & /@ basis];
11    If[OptionValue["AsAssociation"],
12      (
13        basis= Association @ Table[(J->Select[basis, #[[2]]==J]),{J,
14          Js}]
15      )
16    ];
17    Return[basis];
18  );
19 ];
```

To obtain the blocks (indexed by  $J$ ) representing the Hamiltonian in the level description, the function `LevelSimplerEffectiveHamiltonian` is provided in `qlanth`.

```

1 LevelSimplerEffectiveHamiltonian::usage =
  LevelSimplerEffectiveHamiltonian[numE] is a variation of
  EffectiveHamiltonian that returns the diagonal JJ Hamiltonian
  blocks applying a simplifier and with simplifications adequate for
  the level description. The keys of the given association
  correspond to the different values of J that are possible for f^
  numE, the values are sparse array that are meant to be interpreted
  in the basis provided by BasisLSJ.
2 The option ''Simplifier'' is a list of symbols that are set to zero.
  At a minimum this has to include the crystal field parameters. By
  default this includes everything except the Slater parameters Fk
  and the spin orbit coupling  $\zeta$ .
3 The option ''Export'' controls whether the resulting association is
  saved to disk, the default is True and the resulting file is saved
  to the ./hams/ folder. A hash is appended to the filename that
  corresponds to the simplifier used in the resulting expression. If
  the option ''Overwrite'' is set to False then these files may be
  used to quickly retrieve a previously computed case. The file is
  saved both in .m and .mx format.
4 The option ''PrependToFilename'' can be used to append a string to
  the filename to which the function may export to.
5 The option ''Return'' can be used to choose whether the function
  returns the matrix or not.
6 The option ''Overwrite'' can be used to overwrite the file if it
  already exists.";
```

```

7 Options[LevelSimplerEffectiveHamiltonian] = {
8   "Export" -> True,
9   "PrependToFilename" -> "",
10  "Overwrite" -> False,
11  "Return" -> True,
12  "Simplifier" -> Join[
13    {FO, \[Sigma]SS},
14    cfSymbols,
15    TSymbols,
16    casimirSymbols,
17    pseudoMagneticSymbols,
18    marvinSymbols,
19    DeleteCases[magneticSymbols, \[Zeta]]
20  ]
21 };
22 LevelSimplerEffectiveHamiltonian[numE_Integer, OptionsPattern[]] :=
23   Module[
24     {thisHamAssoc, Js, fname,
25      fnamemx, hash, simplifier},
26     (
27       simplifier = (#->0)&/@Sort[OptionValue["Simplifier"]];
28       hash = Hash[simplifier];
29       If[Not[ValueQ[ElectrostaticTable]], LoadElectrostatic[]];
30       If[Not[ValueQ[S00andECSOTable]], LoadS00andECSO[]];
31       If[Not[ValueQ[SpinOrbitTable]], LoadSpinOrbit[]];
32       If[Not[ValueQ[SpinSpinTable]], LoadSpinSpin[]];
33       If[Not[ValueQ[ThreeBodyTable]], LoadThreeBody[]];
34       fname = FileNameJoin[{moduleDir, "hams", OptionValue[
35         "PrependToFilename"] <> "Level-SymbolicMatrix-f" <> ToString[numE] <> "-" <> ToString[hash] <> ".m"}];
36       fnamemx = FileNameJoin[{moduleDir, "hams", OptionValue[
37         "PrependToFilename"] <> "Level-SymbolicMatrix-f" <> ToString[numE] <> "-" <> ToString[hash] <> ".mx"}];
38       If[Or[FileExistsQ[fname], FileExistsQ[fnamemx]] && Not[OptionValue[
39         "Overwrite"]],
40       (
41         If[OptionValue["Return"],
42           (
43             Which[FileExistsQ[fnamemx],
44               (
45                 Echo["File " <> fnamemx <> " already exists, and option '" Overwrite "' is set to False, loading file ..."];
46                 thisHamAssoc=Import[fnamemx];
47                 Return[thisHamAssoc];
48               ),
49               FileExistsQ[fname],
50               (
51                 Echo["File " <> fname <> " already exists, and option '" Overwrite "' is set to False, loading file ..."];
52                 thisHamAssoc=Import[fname];
53                 Echo["Exporting to file " <> fnamemx <> " for quicker loading."];
54                 Export[fnamemx,thisHamAssoc];
55                 Return[thisHamAssoc];
56               )
57             ]
58           ],
59           (
60             Echo["File " <> fname <> " already exists, skipping ..."];
61             Return[Null];
62           )
63         ]
64       ];
65       Js = AllowedJ[numE];
66       thisHamAssoc = EffectiveHamiltonian[numE,
67         "Set t2Switch"->True,
68         "IncludeZeeman"->False,
69         "ReturnInBlocks"->True
70       ];
71       thisHamAssoc = Diagonal[thisHamAssoc];
72       thisHamAssoc = Map[SparseArray[ReplaceInSparseArray[#, simplifier]]&, thisHamAssoc,{1}];
73       thisHamAssoc = FreeHam[thisHamAssoc,numE];
74       thisHamAssoc = AssociationThread[Js->thisHamAssoc];
75       If[OptionValue["Export"],
```

```

73      (
74        Echo["Exporting to file " <> fname <> " and to " <> fnamemx];
75        Export[fname, thisHamAssoc];
76        Export[fnamemx, thisHamAssoc];
77      )
78    ];
79    If[OptionValue["Return"],
80      Return[thisHamAssoc],
81      Return[Null]
82    ];
83  );
84 ];

```

Whereas this description may be calculated without ever making explicit reference to  $M_J$ , in `qlanth` what is done is that the more verbose description associated with the  $|LSJM_J\rangle$  basis is “downsized” to obtain the description in terms of levels. For this aim the following functions in `qlanth` are instrumental: `EigenLever`, `FreeHam`, `ListRepeater`, and `ListLever`.

The function `LevelSolver` can be used to calculate the level structure for given values of the parameters that one wishes to keep for the level description, which is often simply termed the *free-ion* part of the Hamiltonian.

```

1 LevelSolver::usage = "LevelSolver[numE, params] puts together (or
2   retrieves from disk) the symbolic level Hamiltonian for the f^numE
3   configuration and solves it for the given params returning the
4   resultant energies and eigenstates.
5 If the option ''Return as states'' is set to False, then the function
6   returns an association whose keys are values for J in f^numE, and
7   whose values are lists with two elements. The first element being
8   equal to the ordered basis for the corresponding subspace, given
9   as a list of lists of the form {LS string, J}. The second element
10  being another list of two elements, the first element being equal
11  to the energies and the second being equal to the corresponding
12  normalized eigenvectors. The energies given have been subtracted
13  the energy of the ground state.
14 If the option ''Return as states'' is set to True, then the function
15  returns a list with three elements. The first element is the
16  global level basis for the f^numE configuration, given as a list
17  of lists of the form {LS string, J}. The second element are the
18  major LSJ components in the returned eigenstates. The third
19  element is a list of lists with three elements, in each list the
20  first element being equal to the energy, the second being equal to
21  the value of J, and the third being equal to the corresponding
22  normalized eigenvector (given as a row). The energies given have
23  been subtracted the energy of the ground state, and the states
24  have been sorted in order of increasing energy.
25 The following options are admitted:
26 - ''Overwrite Hamiltonian'', if set to True the function will
27   overwrite the symbolic Hamiltonian. Default is False.
28 - ''Return as states'', see description above. Default is True.
29 - ''Simplifier'', this is a list with symbols that are set to zero
   for defining the parameters kept in the level description.
";
Options[LevelSolver] = {
  "Overwrite Hamiltonian" -> False,
  "Return as states" -> True,
  "Simplifier" -> Join[
    cfSymbols,
    TSymbols,
    casimirSymbols,
    pseudoMagneticSymbols,
    marvinSymbols,
    DeleteCases[magneticSymbols, \[Zeta]]
  ],
  "PrintFun" -> PrintTemporary
};
LevelSolver[numE_Integer, params0_Association, OptionsPattern[]] :=
  Module[
{ln, simplifier, simpleHam, basis,
 numHam, eigensys, startTime, endTime,
 diagonalTime, params=params0, globalBasis,
 eigenVectors, eigenEnergies, eigenJs,
 states, groundEnergy, allEnergies, PrintFun},
(
  ln           = theLanthanides[[numE]];

```

```

30 basis      = BasisLSJ[numE, "AsAssociation" -> True];
31 simplifier = OptionValue["Simplifier"];
32 PrintFun   = OptionValue["PrintFun"];
33 PrintFun["> LevelSolver for ", ln, " with ", numE, " f-electrons."];
34 PrintFun["> Loading the symbolic level Hamiltonian ..."];
35 simpleHam = LevelSimplerEffectiveHamiltonian[numE,
36     "Simplifier" -> simplifier,
37     "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
38 ];
39 (* Everything that is not given is set to zero *)
40 PrintFun["> Setting to zero every parameter not given ..."];
41 params    = ParamPad[params, "PrintFun" -> PrintFun];
42 PrintFun[params];
43 (* Create the numeric hamiltonian *)
44 PrintFun["> Replacing parameters in the J-blocks of the
45 Hamiltonian to produce numeric arrays ..."];
46 numHam    = N /@ Map[ReplaceInSparseArray[#, params] &, simpleHam
47 ];
48 Clear[simpleHam];
49 (* Eigensolver *)
50 PrintFun["> Diagonalizing the numerical Hamiltonian within each
51 separate J-subspace ..."];
52 startTime = Now;
53 eigensys = Eigensystem /@ numHam;
54 endTime   = Now;
55 diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
56 allEnergies = Flatten[First /@ Values[eigensys]];
57 groundEnergy = Min[allEnergies];
58 eigensys = Map[Chop[{#[[1]] - groundEnergy, #[[2]]}] &, eigensys];
59 eigensys = Association@KeyValueMap[#1 -> {basis[#1], #2} &,
eigensys];
60 PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
61 If[OptionValue["Return as states"],
62 (
63     PrintFun["> Padding the eigenvectors to correspond to the
64 level basis ..."];
65     eigenVectors = SparseArray @ BlockDiagonalMatrix[Values
66     #[[2, 2]] & /@ eigensys];
67     globalBasis  = Flatten[Values[basis], 1];
68     eigenEnergies = Flatten[Values[#[[2, 1]] & /@ eigensys]];
69     eigenJs    = Flatten[KeyValueMap[ConstantArray[#1, Length
70     #[[2, 2]]]] &, eigensys]];
71     states      = Transpose[{eigenEnergies, eigenJs,
eigenVectors}];
72     states      = SortBy[states, First];
73     eigenVectors = Last /@ states;
74     LSJmultiplets = (RemoveTrailingDigits[#[[1]]] <> ToString[
75     InputForm[#[[2]]]]) & /@ globalBasis;
76     majorComponentIndices = Ordering[Abs[#[[-1]]] & /@
eigenVectors;
77     levelLabels       = LSJmultiplets[[majorComponentIndices
78 ]];
79     Return[{globalBasis, levelLabels, states}];
80   ),
81   Return[{basis, eigensys}]
82 ];
83 );
84 ]
85 ];

```

## 2.4 The coefficients of fractional parentage

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

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

as a sum over parents

$$|\underline{\ell}^n \alpha LS\rangle = \sum_{\bar{\alpha} \bar{L} \bar{S}} \underbrace{(\underline{\ell}^{n-1} \bar{\alpha} \bar{L} \bar{S})}_{\text{How much of parent } |\underline{\ell}^{n-1} \bar{\alpha} \bar{L} \bar{S}\rangle \text{ is in daughter } |\underline{\ell}^n \alpha LS\rangle} \underbrace{|\underline{\ell}^{n-1} \bar{\alpha} \bar{L} \bar{S}, \underline{\ell}\rangle}_{\text{Couple an additional } \underline{\ell} \text{ to the parent } |\underline{\ell}^{n-1} \bar{\alpha} \bar{L} \bar{S}\rangle} \alpha LS\rangle. \quad (14)$$

More importantly for **qlanth**, the coefficients of fractional parentage can be used to evaluate matrix elements of operators, such as in [Eqn-29](#), [Eqn-51](#), [Eqn-65](#), and [Eqn-41](#). These formulas realize a convenient calculation advantage: if one knows matrix elements in one configuration, then one can immediately calculate them in any other configuration.

In principle all the data that is needed in order to evaluate the matrix elements that **qlanth** uses can all be derived from coefficients of fractional parentage, tables of 6-j and 3-j coefficients, the LSUW labels for the terms in the  $\underline{f}^n$  configurations, reduced matrix elements in  $\underline{f}^3$  for the three-body operators, and reduced matrix elements in  $\underline{f}^2$  for the magnetic interactions.

The data for the coefficients of fractional parentage we owe to [\[Vel00\]](#) from which the file `B1F_all.txt` originates, and which we use here to extract this useful “escalator” up the  $\underline{f}^n$  configurations.

In **qlanth** the function `GenerateCFPTable` is used to parse the data contained in this file. From this data the association `CFP` is generated. This association has keys that represent  $LS$  terms for a configuration  $\underline{f}^n$  and values that are lists which contain all the parent terms, together with the corresponding coefficients of fractional parentage.

```

1 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.
2 The data being parsed here is the file attachment B1F_ALL.TXT which
   comes from Velkov's thesis.";
3 Options[GenerateCFPTable] = {"Export" -> True};
4 GenerateCFPTable[OptionsPattern[]] := Module[
5   {rawText, rawLines, leadChar, configIndex, line, daughter,
6   lineParts, numberCode, parsedNumber, toAppend, CFPTablefname},
7   (
8     CleanWhitespace[string_]      := StringReplace[string,
9       RegularExpression["\\s+"]->" "];
10    AddSpaceBeforeMinus[string_] := StringReplace[string,
11      RegularExpression["(?<!\\s)-"]->" -"];
12    ToIntegerOrString[list_]     := Map[If[StringMatchQ[#, 
13      NumberString], ToExpression[#, #] &, list];
14    CFPTable      = ConstantArray[{}, 7];
15    CFPTable[[1]] = {{"2F", {"1S", 1}}};
16
17 (* Cleaning before processing is useful *)
18 rawText = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
19 rawLines = StringTrim/@StringSplit[rawText, "\n"];
20 rawLines = Select[rawLines, # != "" &];
21 rawLines = CleanWhitespace/@rawLines;
22 rawLines = AddSpaceBeforeMinus/@rawLines;
23
24 Do[(
25   (* the first character can be used to identify the start of a
26   block *)
27   leadChar=StringTake[line,{1}];
28   (* ...FN, N is at position 50 in that line *)
29   If[leadChar=="[",
30   (
31     configIndex=ToExpression[StringTake[line,{50}]];
32     Continue[];
33   )
34   ];
35   (* Identify which daughter term is being listed *)
36   If[StringContainsQ[line, "[DAUGHTER TERM]"],
37     daughter=StringSplit[line, "["[[1]];
38     CFPTable[[configIndex]]=Append[CFPTable[[configIndex]], {
daughter}];
39     Continue[];
40   ];
41   (* Once we get here we are already parsing a row with
42   coefficient data *)
43   lineParts = StringSplit[line, " "];

```

```

39     parent      = lineParts[[1]];
40     numberCode = ToIntegerOrString[lineParts[[3;;]]];
41     parsedNumber = SquarePrimeToNormal[numberCode];
42     toAppend    = {parent, parsedNumber};
43     CFPTable[[configIndex]][[-1]] = Append[CFPTable[[configIndex
44   ]][[-1]], toAppend]
45   ),
46   {line, rawLines}];
47   If[OptionValue["Export"],
48   (
49     CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"
50   }];
51     Export[CFPTablefname, CFPTable];
52   )
53 ];
54 ]

```

The coefficients of fractional parentage are traditionally only provided up to  $\underline{f}^7$  (such is the case in [B1f\\_all.txt](#)), tabulating these beyond  $\underline{f}^7$  would be redundant since the coefficients of fractional parentage beyond  $\underline{f}^7$  satisfy relationships with those below  $\underline{f}^7$ . According to [NK63]

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

with  $\xi = \begin{cases} 1 & \text{if } n \neq 6 \\ (-1)^{(\bar{\nu}-1)/2} & \text{if } n = 6 \end{cases}$ , and where  $\bar{\nu}$  is the seniority of  $|\bar{\alpha} \bar{L} \bar{S}\rangle$ . (15)

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

$$\langle \underline{f}^{14-n} \alpha S L \| \hat{U}^{(K)} \| \underline{f}^{14-n} \alpha' S' L' \rangle = -(-1)^K \langle \underline{f}^n \alpha S L \| \hat{U}^{(K)} \| \underline{f}^n \alpha' S' L' \rangle \quad (16)$$

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

$$\langle \underline{f}^{14-n} \alpha S L \| \hat{V}^{(1K)} \| \underline{f}^{14-n} \alpha' S' L' \rangle = (-1)^K \langle \underline{f}^n \alpha S L \| \hat{V}^{(1K)} \| \underline{f}^n \alpha' S' L' \rangle \quad (17)$$

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

## 2.5 Going beyond $\underline{f}^7$

In most cases all matrix elements in `qlanth` are only calculated up to and including  $\underline{f}^7$ . Beyond  $\underline{f}^7$  adequate changes of sign are enforced to take into account the equivalence that can be made between  $\underline{f}^n$  and  $\underline{f}^{14-n}$  as given by [Eqn-17](#) and [Eqn-16](#).

This is enforced when the function `EffectiveHamiltonian` is called. In there `Hole-ElectronConjugation` is the function responsible for enforcing a global sign flip for the following operators (or alternatively, to their accompanying coefficients):

$$\begin{aligned} & \zeta, B_q^{(k)} \\ & T^{(2)\prime}, T^{(3)}, T^{(4)}, T^{(6)}, T^{(7)}, T^{(8)} \\ & T^{(11)\prime}, T^{(12)}, T^{(13)}, T^{(14)}, T^{(15)}, T^{(16)}, T^{(17)}, T^{(18)}, T^{(19)}, \end{aligned} \quad (18)$$

$T^{(2)}$  and  $T^{(11)}$  must be treated separately since they have a part that changes sign, and another that doesn't.

```

1 HoleElectronConjugation::usage = "HoleElectronConjugation[params]
2   takes the parameters (as an association) that define a
3   configuration and converts them so that they may be interpreted as
4   corresponding to a complementary hole configuration. Some of this
5   can be simply done by changing the sign of the model parameters.
6   In the case of the effective three body interaction of T2 the
7   relationship is more complex and is controlled by the value of the
8   t2Switch variable.";
9 HoleElectronConjugation[params_] := Module[
10   {newparams = params},
11   (
12     flipSignsOf = Join[{\zeta}, cfSymbols, TSymbols];
13     flipped = Table[

```

```

7   (
8     flipper -> - newparams[flipper]
9   ),
10  {flipper, flipSignsOf}
11  ];
12 nonflipped = Table[
13  (
14    flipper -> newparams[flipper]
15  ),
16  {flipper, Complement[Keys[newparams], flipSignsOf]}
17  ];
18 flippedParams = Association[Join[nonflipped, flipped]];
19 flippedParams = Select[flippedParams, FreeQ[#, Missing]&];
20 Return[flippedParams];
21 )
22 ];

```

## 2.6 The J-J' 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  $f^n$  and for each term  $\hat{h}$  in the Hamiltonian, **qlanth** first calculates the matrix elements  $\langle \alpha L S J M_J | \hat{h} | \alpha' L' S' J' M'_J \rangle$  so that for each interaction an association with keys of the form  $\{J, J'\}$  is created. The values being rectangular arrays.

**Fig-5** shows roughly this block structure for  $f^2$ . 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  $(2 \times 2)$ ,  $(2 \times 3)$ ,  $(2 \times 15)$ ,  $(2 \times 7)$ ,  $(2 \times 27)$ ,  $(2 \times 11)$ , and  $(2 \times 26)$ .

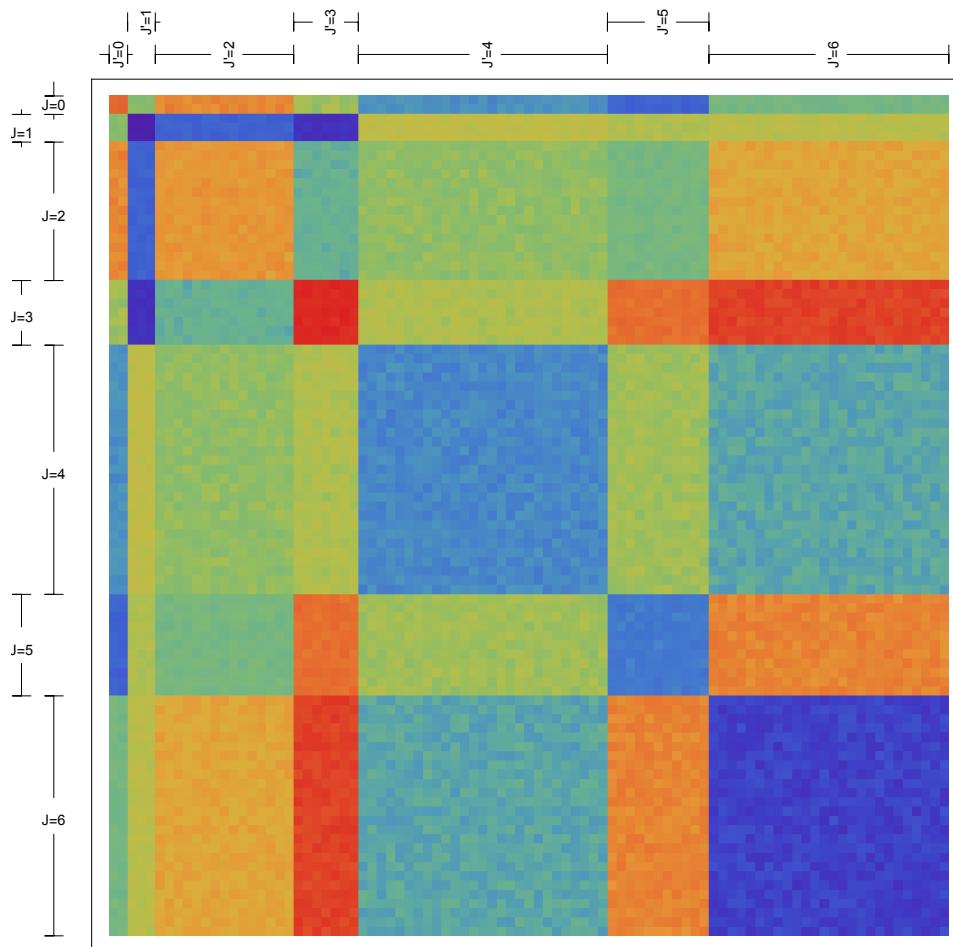


Figure 5: The J-J' block structure for  $f^2$

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

```

1 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."];
2 Options[JJBlockMatrix] = {"Sparse" -> True, "ChenDeltas" -> False};
3 JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]] := Module[
4 {NKSLJMs, NKSLJMp, NKSLJM, NKSLJMP,
5 SLterm, SpLpterm,
6 MJ, MJp,
7 subKron, matValue, eMatrix},
8 (
9   NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
10  NKSLJMp = AllowedNKSLJMforJTerms[numE, Jp];
11  eMatrix =
12   Table[
13     (*Condition for a scalar matrix op*)
14     SLterm = NKSLJM[[1]];
15     SpLpterm = NKSLJMP[[1]];
16     MJ = NKSLJM[[3]];
17     MJp = NKSLJMP[[3]];
18     subKron = (
19       KroneckerDelta[J, Jp] *
20       KroneckerDelta[MJ, MJp]
21     );
22     matValue =
23     If[subKron == 0,
24       0,
25       (
26         ElectrostaticTable[{numE, SLterm, SpLpterm}] +
27         ElectrostaticConfigInteraction[numE, {SLterm,
28           SpLpterm}] +
29         SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
30         MagneticInteraction[{numE, SLterm, SpLpterm, J},
31           "ChenDeltas" -> OptionValue["ChenDeltas"]] +
32         ThreeBodyTable[{numE, SLterm, SpLpterm}]
33       )
34     ];
35     matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp
36   }];
37     matValue,
38     {NKSLJMP, NKSLJMp},
39     {NKSLJM, NKSLJMs}
40   ];
41   If[OptionValue["Sparse"],
42     eMatrix = SparseArray[eMatrix]
43   ];
44   Return[eMatrix]
45 )
46 ];

```

Once these blocks have been calculated and saved to disk (in the folder `./hams/`) the function `EffectiveHamiltonian` takes them, assembles the arrays in block form, and finally flattens them to provide a sparse rank-2 array. These are the arrays that are finally diagonalized to find energies and eigenstates. Through options, this function can also return the Hamiltonian in block form, which is useful for the level description of the eigenstates.

```

1 EffectiveHamiltonian::usage = "EffectiveHamiltonian[numE] returns the
2   Hamiltonian matrix for the f^numE configuration. The matrix is
3   returned as a SparseArray.
4 The function admits an optional parameter ''FilenameAppendix'', which
5   can be used to control which variant of the JJBlocks is used to
6   assemble the matrix.
7 It also admits an optional parameter ''IncludeZeeman'', which can be
8   used to include the Zeeman interaction. The default is False.
9 The option ''Set t2Switch'' can be used to toggle on or off setting
10  the t2 selector automatically or not, the default is True, which
11  replaces the parameter according to numE.
12 The option ''ReturnInBlocks'' can be used to return the matrix in
13  block or flattened form. The default is to return it in flattened
14  form.";
15 Options[EffectiveHamiltonian] = {
16   "FilenameAppendix" -> "",
17   "IncludeZeeman" -> False,
18   "Set t2Switch" -> True,
19   "ReturnInBlocks" -> False,
20   "OperatorBasis" -> "Legacy"};
21 EffectiveHamiltonian[nf_, OptionsPattern[]] := Module[
22

```

```

13 {numE, ii, jj, howManyJs, Js, blockHam, opBasis},
14 (
15 (*#####
16 ImportFun = ImportMZip;
17 opBasis = OptionValue["OperatorBasis"];
18 If[Not[MemberQ>{"Legacy", "MostlyOrthogonal", "Orthogonal"}, opBasis]],
19     Echo["Operator basis " <> opBasis <> " not recognized, using ",
20 Legacy' basis."];
21     opBasis = "Legacy";
22 ];
23 If[opBasis == "Orthogonal",
24     Echo["Operator basis 'Orthogonal' not implemented yet,
25 aborting ..."];
26     Return[Null];
27 ];
28 (*#####
29 If[opBasis == "MostlyOrthogonal",
30 (
31     blockHam = EffectiveHamiltonian[nf,
32         "OperatorBasis" -> "Legacy",
33         "FilenameAppendix" -> OptionValue["FilenameAppendix"],
34         "IncludeZeeman" -> OptionValue["IncludeZeeman"],
35         "Set t2Switch" -> OptionValue["Set t2Switch"],
36         "ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
37     paramChanger = Which[
38         nf < 7,
39         <|
40             F0 -> 1/91 (54 E1p+91 E0p+78 γp),
41             F2 -> (15/392 *
42                 (
43                     140 E1p +
44                     20020 E2p +
45                     1540 E3p +
46                     770 αp -
47                     70 γp +
48                     22 Sqrt[2] T2p -
49                     11 Sqrt[2] nf T2p t2Switch -
50                     11 Sqrt[2] (14-nf) T2p (1 - t2Switch)
51                 )
52             ),
53             F4 -> (99/490 *
54                 (
55                     70 E1p -
56                     9100 E2p +
57                     280 E3p +
58                     140 αp -
59                     35 γp +
60                     4 Sqrt[2] T2p -
61                     2 Sqrt[2] nf T2p t2Switch -
62                     2 Sqrt[2] (14-nf) T2p (1-t2Switch)
63                 )
64             ),
65             F6 -> (5577/7000 *
66                 (
67                     20 E1p +
68                     700 E2p -
69                     140 E3p -
70                     70 αp -
71                     10 γp -
72                     2 Sqrt[2] T2p +
73                     Sqrt[2] nf T2p t2Switch +
74                     Sqrt[2] (14-nf) T2p (1-t2Switch)
75                 )
76             ),
77             ζ -> ζ,
78             α -> (5 αp)/4,
79             β -> -6 (5 αp + βp),
80             γ -> 5/2 (2 βp + 5 γp),
81             T2 -> 0
82             |>,
83             nf >= 7,
84             <|
85                 F0 -> 1/91 (54 E1p+91 E0p+78 γp),
86                 F2 -> (15/392 *
87                     (

```

```

86          140   E1p  +
87          20020  E2p  +
88          1540   E3p  +
89          770   αp  -
90          70   γp  +
91          22  Sqrt[2]  T2p  -
92          11  Sqrt[2]  nf  T2p
93      )
94  ),
95  F4 -> (99/490 *
96  (
97      70   E1p  -
98      9100  E2p  +
99      280   E3p  +
100     140   αp  -
101     35   γp  +
102     4  Sqrt[2]  T2p  -
103     2  Sqrt[2]  nf  T2p
104  )
105  ),
106  F6 -> (5577/7000 *
107  (
108      20   E1p  +
109      700  E2p  -
110      140  E3p  -
111      70   αp  -
112      10   γp  -
113      2  Sqrt[2]  T2p  +
114      Sqrt[2]  nf  T2p
115  )
116  ),
117  ζ -> ζ,
118  α -> (5 αp)/4,
119  β -> -6 (5 αp + βp),
120  γ -> 5/2 (2 βp + 5 γp),
121  T2 -> 0
122 |>
123 ];
124 blockHamM0 = Which[
125   OptionValue["ReturnInBlocks"] == False,
126   ReplaceInSparseArray[blockHam, paramChanger],
127   OptionValue["ReturnInBlocks"] == True,
128   Map[ReplaceInSparseArray[#, paramChanger]&, blockHam, {2}]
129 ];
130 Return[blockHamM0];
131 )
132 ];
133 (*#####
134 (*hole-particle equivalence enforcement*)
135 numE = nf;
136 allVars = {E0, E1, E2, E3, ζ, F0, F2, F4, F6, M0, M2, M4, T2, T2p
137 ,
138   T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
139   α, β, γ, B02, B04, B06, B12, B14, B16,
140   B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16, S22
141 ,
142   S24, S26, S34, S36, S44, S46, S56, S66, T11p, T12, T14, T15,
143   T16,
144   T17, T18, T19, Bx, By, Bz};
145 params0 = AssociationThread[allVars, allVars];
146 If[nf > 7,
147   (
148     numE = 14 - nf;
149     params = HoleElectronConjugation[params0];
150     If[OptionValue["Set t2Switch"], params[t2Switch] = 0];
151   ),
152   params = params0;
153   If[OptionValue["Set t2Switch"], params[t2Switch] = 1];
154 ];
155 (* Load symbolic expressions for LS,J,J' energy sub-matrices. *)
156 emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
157 OptionValue["FilenameAppendix"]];
158 JJBlockMatrixTable = ImportFun[emFname];
159 (*Patch together the entire matrix representation using J,J'
160 blocks.*)

```

```

157 PrintTemporary["Patching JJ blocks ..."];
158 Js = AllowedJ[numE];
159 howManyJs = Length[Js];
160 blockHam = ConstantArray[0, {howManyJs, howManyJs}];
161 Do[
162   blockHam[[jj, ii]] = JJBlockMatrixTable[{numE, Js[[ii]], Js[[jj]]}];,
163   {ii, 1, howManyJs},
164   {jj, 1, howManyJs}
165 ];
166 (* Once the block form is created flatten it *)
167 If[Not[OptionValue["ReturnInBlocks"]],
168   (blockHam = ArrayFlatten[blockHam];
169   blockHam = ReplaceInSparseArray[blockHam, params];
170   ),
171   (blockHam = Map[ReplaceInSparseArray[#, params]&, blockHam
172 ,{2}]);
173 ];
174 If[OptionValue["IncludeZeeman"],
175 (
176   PrintTemporary["Including Zeeman terms ..."];
177   {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "
178 ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
179   blockHam += - teslaToKayser * (Bx * magx + By * magy + Bz *
180   magz);
181   )
182 ];
183 Return[blockHam];
184 ]
185 ];

```

In `qlanth` the reduced matrix elements of all operators, and the subsequent matrix elements of  $\hat{\mathcal{H}}$  are calculated exactly. This is in contrast to what is done in older alternatives to `qlanth` such as `linuxemp`, in which calculations of reduced matrix elements were obtained from tables calculated with finite precision. To underscore this fact, `&qn-??` shows an example of a J-J block as contained in `qlanth`.

## 2.7 Kramers' degeneracy

In the odd-electron cases, every energy is at least doubly degenerate. In `qlanth`, except in the case of the experimental data compiled for LaF<sub>3</sub>, Kramers' degeneracy is given/expected explicitly.

## 3 Interactions

### 3.1 $\hat{\mathcal{H}}_k$ : kinetic energy

$$\hat{\mathcal{H}}_k = -\frac{\hbar^2}{2m} \sum_{i=1}^N \nabla_i^2 \text{ (kinetic energy of N v-electrons)} \quad (20)$$

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

To interpret the range of energies that result from diagonalizing the semi-empirical Hamiltonian, it might be instructive, however, to note that this term imparts an energy of about 10 eV  $\approx 10^6 \mathcal{K}$ <sup>16</sup> to each electron.

### 3.2 $\hat{\mathcal{H}}_{e:sn}$ : the central field potential

$$\hat{\mathcal{H}}_{e:sn} = -e^2 \sum_{i=1}^Z \frac{1}{r_i} + e^2 \sum_{i=1}^n \sum_{j=1}^{Z-n} \frac{1}{r_{ij}} \approx \sum_{i=1}^n V_{sn}(r_i) \text{ (with Z = atomic No.)} \quad (21)$$

Repulsion between valence and inner shell electrons

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

<sup>16</sup> Note, (Kayser)  $\mathcal{K} \equiv \text{cm}^{-1}$ , see section on units.

	$ {}^4F_{1,-1}\rangle$	$ {}^4F_{1,0}\rangle$	$ {}^4F_{1,1}\rangle$
$\langle {}^4F_{1,-1} $	$\begin{aligned} & 2\alpha + \beta - \frac{B_0^{(2)}}{10} + \gamma \\ & -\frac{\zeta}{2} + \frac{14F^{(0)}}{13} + \frac{43F^{(2)}}{195} + \frac{19F^{(4)}}{429} \\ & -\frac{875F^{(6)}}{5577} + 2m^{(0)}\sigma_{SS} + \frac{61m^{(0)}}{12} \\ & + 4m^{(2)}\sigma_{SS} + \frac{145m^{(2)}}{12} + \frac{50m^{(4)}\sigma_{SS}}{11} \\ & + \frac{1805m^{(4)}}{132} + \frac{43P^{(2)}}{1080} + \frac{19P^{(4)}}{2376} \\ & - \frac{875P^{(6)}}{30888} \end{aligned}$	$\begin{aligned} & -\frac{\sqrt{3}B_1^{(2)}}{10} - \frac{1}{10}i\sqrt{3}S_2^{(2)} \\ & -\frac{1}{5}\sqrt{\frac{3}{2}}B_2^{(2)} - \frac{1}{5}i\sqrt{\frac{3}{2}}S_2^{(2)} \end{aligned}$	
$\langle {}^4F_{1,0} $	$\begin{aligned} & 2\alpha + \beta + \frac{B_0^{(2)}}{5} + \gamma \\ & -\frac{\zeta}{2} + \frac{14F^{(0)}}{13} + \frac{43F^{(2)}}{195} + \frac{19F^{(4)}}{429} \\ & -\frac{875F^{(6)}}{5577} + 2m^{(0)}\sigma_{SS} + \frac{61m^{(0)}}{12} \\ & + 4m^{(2)}\sigma_{SS} + \frac{145m^{(2)}}{12} + \frac{50m^{(4)}\sigma_{SS}}{11} \\ & + \frac{1805m^{(4)}}{132} + \frac{43P^{(2)}}{1080} + \frac{19P^{(4)}}{2376} \\ & - \frac{875P^{(6)}}{30888} \end{aligned}$	$\begin{aligned} & \frac{\sqrt{3}B_1^{(2)}}{10} + \frac{1}{10}i\sqrt{3}S_2^{(2)} \\ & -\frac{1}{5}\sqrt{\frac{3}{2}}B_2^{(2)} + \frac{1}{5}i\sqrt{\frac{3}{2}}S_2^{(2)} \end{aligned}$	$\begin{aligned} & 2\alpha + \beta - \frac{B_0^{(2)}}{10} + \gamma \\ & -\frac{\zeta}{2} + \frac{14F^{(0)}}{13} + \frac{43F^{(2)}}{195} + \frac{19F^{(4)}}{429} \\ & -\frac{875F^{(6)}}{5577} + 2m^{(0)}\sigma_{SS} + \frac{61m^{(0)}}{12} \\ & + 4m^{(2)}\sigma_{SS} + \frac{145m^{(2)}}{12} + \frac{50m^{(4)}\sigma_{SS}}{11} \\ & + \frac{1805m^{(4)}}{132} + \frac{43P^{(2)}}{1080} + \frac{19P^{(4)}}{2376} \\ & - \frac{875P^{(6)}}{30888} \end{aligned}$

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 like spherical shells surrounding the nucleus.

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 of the wavefunctions.

### 3.3 $\hat{\mathcal{H}}_{\text{e:e}}$ : e:e repulsion

$$\hat{\mathcal{H}}_{\text{e:e}} = \sum_{i>j}^{n,n} \frac{e^2}{\|\vec{r}_i - \vec{r}_j\|} = \sum_{k=0,2,4,6} F^{(k)} \hat{f}_k = \sum_{k=0,1,2,3} E_k \hat{e}_k \quad (22)$$

This term is the first we will not discard. Calculating this term for the  $f^n$  configurations was one of the contributions 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 [Rac49]. The insight that Racah had was that if in a given operator one identifies the parts in it that transform accordingly to the different symmetry groups present in the problem, then calculating the necessary matrix element in all  $f^n$  configurations can be greatly simplified.

The functions used in `qlanth` to compute these LS-reduced matrix elements <sup>17</sup> 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  $f^n$  configuration. By default this term is computed in terms of  $F^{(k)}$  Slater integrals, but it can also be computed in terms of the  $E_k$  Racah parameters, the functions `EtoF` and `FtoE` are useful for going from one representation to the other.

$$\langle f^n \alpha'^{2S+1} L \| \hat{\mathcal{H}}_{\text{e:e}} \| f^n \alpha'^{2S'+1} L' \rangle = \sum_{k=0,2,4,6} F^{(f)}_k(n, \alpha L S, \alpha' L' S') \quad (23)$$

where

$$f_k(n, \alpha L S, \alpha' L' S') = \frac{1}{2} \delta(S, S') \delta(L, L') \langle f \| \hat{C}^{(k)} \| f \rangle^2 \times \\ \left\{ \frac{1}{2L+1} \sum_{\alpha'' L''} \langle f^n \alpha'' L'' S \| \hat{U}^{(k)} \| f^n \alpha L S \rangle \langle f^n \alpha'' L'' S \| \hat{U}^{(k)} \| f^n \alpha' L S \rangle - \delta(\alpha, \alpha') \frac{n(4f+2-n)}{(2f+1)(4f+1)} \right\}. \quad (24)$$

```

1 Electrostatic::usage = "Electrostatic[{numE, NKSL, NKSLp}] returns
2   the LS reduced matrix element for repulsion matrix element for
3   equivalent electrons. See equation 2-79 in Wybourne (1965). The
4   option ''Coefficients'' can be set to ''Slater'' or ''Racah''. If
5   set to ''Racah'' then E_k parameters and e^k operators are assumed
6   , otherwise the Slater integrals F^k and operators f_k. The
7   default is ''Slater''.";
8 Options[Electrostatic] = {"Coefficients" -> "Slater"};
9 Electrostatic[{numE_, NKSL_, NKSLp_}, OptionsPattern[]] := Module[
10   {fsub0, fsub2, fsub4, fsub6,
11    esub0, esub1, esub2, esub3,
12    fsup0, fsup2, fsup4, fsup6,
13    eMatrixVal, orbital},
14   (
15     orbital = 3;
16     Which[
17       OptionValue["Coefficients"] == "Slater",
18       (
19         fsub0 = fsubk[numE, orbital, NKSL, NKSLp, 0];
20         fsub2 = fsubk[numE, orbital, NKSL, NKSLp, 2];
21         fsub4 = fsubk[numE, orbital, NKSL, NKSLp, 4];
22         fsub6 = fsubk[numE, orbital, NKSL, NKSLp, 6];
23         eMatrixVal = fsub0*F0 + fsub2*F2 + fsub4*F4 + fsub6*F6;
24       ),
25       OptionValue["Coefficients"] == "Racah",
26       (
27         fsup0 = fsupk[numE, orbital, NKSL, NKSLp, 0];
28         fsup2 = fsupk[numE, orbital, NKSL, NKSLp, 2];
29         fsup4 = fsupk[numE, orbital, NKSL, NKSLp, 4];
30         fsup6 = fsupk[numE, orbital, NKSL, NKSLp, 6];
31       )
32     ]
33   ]
34 
```

<sup>17</sup> An LS-reduced matrix element is ...

```

25      esub0 = fsup0;
26      esub1 = 9/7*fsup0 + 1/42*fsup2 + 1/77*fsup4 + 1/462*
27      fsup6;
28      esub2 = 143/42*fsup2 - 130/77*fsup4 + 35/462*
29      fsup6;
30      esub3 = 11/42*fsup2 + 4/77*fsup4 - 7/462*
31      fsup6;
32      eMatrixVal = esub0*E0 + esub1*E1 + esub2*E2 + esub3*E3;
33      )
34 ];

```

```

1 fsubk::usage = "fsubk[numE_, orbital_, SL_, SLP_, k_] gives the Slater
2     integral f_k for the given configuration and pair of SL terms. See
3     equation 12.17 in TASS.";
4 fsubk[numE_, orbital_, NKSL_, NKS LP_, k_] := Module[
5 {terms, S, L, Sp, Lp,
6 termsWithSameSpin, SL,
7 fsubkVal, spinMultiplicity,
8 prefactor, summand1, summand2},
9 (
10 {S, L} = FindSL[NKSL];
11 {Sp, Lp} = FindSL[NKS LP];
12 terms = AllowedNKSLTerms[numE];
13 (* sum for summand1 is over terms with same spin *)
14 spinMultiplicity = 2*S + 1;
15 termsWithSameSpin = StringCases[terms, ToString[spinMultiplicity]
16 ~~ __];
17 termsWithSameSpin = Flatten[termsWithSameSpin];
18 If[Not[{S, L} == {Sp, Lp}],
19     Return[0]
20 ];
21 prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
22 summand1 = Sum[((
23 ReducedUkTable[{numE, orbital, SL, NKSL, k}] *
24 ReducedUkTable[{numE, orbital, SL, NKS LP, k}]
25 ),
26 {SL, termsWithSameSpin}
27 ];
28 summand1 = 1 / TPO[L] * summand1;
29 summand2 = (
30 KroneckerDelta[NKSL, NKS LP] *
31 (numE *(4*orbital + 2 - numE)) /
32 ((2*orbital + 1) * (4*orbital + 1))
33 );
34 fsubkVal = prefactor*(summand1 - summand2);
35 Return[fsubkVal];
36 )
37 ];

```

```

1 EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the Slater integral
2     parameters {F0, F2, F4, F6} corresponding to the given Racah
3     parameters {E0, E1, E2, E3}. This is the inverse of the FtoE
4     function.";
5 EtoF[E0_, E1_, E2_, E3_] := Module[
6 {F0, F2, F4, F6},
7 (
8 F0 = 1/7 (7 E0 + 9 E1);
9 F2 = 75/14 (E1 + 143 E2 + 11 E3);
10 F4 = 99/7 (E1 - 130 E2 + 4 E3);
11 F6 = 5577/350 (E1 + 35 E2 - 7 E3);
12 Return[{F0, F2, F4, F6}];
13 )
14 ];

```

```

1 FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the Racah parameters {
2     E0, E1, E2, E3} corresponding to the given Slater integrals.
3 See eqn. 2-80 in Wybourne.
4 Note that in that equation the subscripted Slater integrals are used
5     but since this function assumes the the input values are
6     superscripted Slater integrals, it is necessary to convert them
7     using Dk.";
8 FtoE[F0_, F2_, F4_, F6_] := Module[
9 
```

```

5 {E0, E1, E2, E3},
6 (
7   E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
8   E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
9   E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;
10  E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
11  Return [{E0, E1, E2, E3}];
12 )
13 ];

```

### 3.4 $\hat{\mathcal{H}}_{\text{s:o}}$ : 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

$$\hat{\mathcal{H}}_{\text{s:o}} = \frac{\hbar^2}{2m_e^2c^2} \left( \frac{1}{r} \frac{dV_{\text{s:n}}}{dr} \right) \hat{\mathbf{l}} \cdot \hat{\mathbf{s}} := \zeta(r) \hat{\mathbf{l}} \cdot \hat{\mathbf{s}}. \quad (25)$$

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

$$\hat{\mathcal{H}}_{\text{s:o}} = \zeta \sum_i^n \hat{\mathbf{l}}_i \cdot \hat{\mathbf{s}}_i. \quad (26)$$

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

$$\begin{aligned} \langle \alpha LSJM_J | \hat{\mathcal{H}}_{\text{s:o}} | \alpha' L'S'J'M_{J'} \rangle &= \zeta \delta(J, J') \delta(M_J, M_{J'}) \langle \alpha LSJM_J | \sum_i^n \hat{\mathbf{l}}_i \cdot \hat{\mathbf{s}}_i | \alpha' L'S'JM_J \rangle \\ &= \zeta \delta(J, J') \delta(M_J, M_{J'}) (-1)^{J+L+S'} \begin{Bmatrix} L & L' & 1 \\ S' & S & J \end{Bmatrix} \langle \alpha LS | \sum_i^n \hat{\mathbf{l}}_i \cdot \hat{\mathbf{s}}_i | \alpha' L'S' \rangle \\ &= \zeta \delta(J, J') \delta(M_J, M_{J'}) (-1)^{J+L+S'} \begin{Bmatrix} L & L' & 1 \\ S' & S & J \end{Bmatrix} \sqrt{\underline{\ell}(\underline{\ell}+1)(2\underline{\ell}+1)} \langle \alpha LS \| \hat{\mathbf{V}}^{(11)} \| \alpha' L'S' \rangle, \end{aligned} \quad (27)$$

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

$$\hat{\mathbf{V}}^{(11)} = \sum_{i=1}^n \left( \hat{\mathbf{s}} \hat{\mathbf{u}}^{(1)} \right)_i, \quad (28)$$

in which the rank on the spin operator  $\hat{\mathbf{s}}$  has been omitted, and the rank of the orbital tensor operator given explicitly as 1.

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

$$\begin{aligned} \langle \underline{\ell}^n \psi \| \hat{\mathbf{V}}^{(1k)} \| \underline{\ell}^n \psi' \rangle &= \langle \underline{\ell}^n \alpha LS \| \hat{\mathbf{V}}^{(1k)} \| \underline{\ell}^n \alpha' L'S' \rangle = n \sqrt{\underline{\ell}(\underline{\ell}+1)(2\underline{\ell}+1)} \sqrt{[S][L][S'][L']} \times \\ &\quad \sum_{\bar{\psi}} (-1)^{\bar{S}+\bar{L}+S+L+\underline{\ell}+\underline{\ell}+k+1} (\psi \{ \bar{\psi} \}) (\bar{\psi} \{ \psi' \}) \begin{Bmatrix} S & S' & 1 \\ \underline{\ell} & \underline{\ell} & \bar{S} \end{Bmatrix} \begin{Bmatrix} L & L' & k \\ \underline{\ell} & \underline{\ell} & \bar{L} \end{Bmatrix} \end{aligned} \quad (29)$$

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

```

1 ReducedV1k::usage = "ReducedV1k[n, SL, SpLp, k] gives the reduced
2   matrix element of the spherical tensor operator V^(1k). See
3   equation 2-101 in Wybourne 1965.";
4 ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
5   {V1k, S, L, Sp, Lp,
6   Sb, Lb, spin, orbital,
7   cfpSL, cfpSpLp,
8   SLparents, SpLpparents,
9   commonParents, prefactor},
10  (
11    {spin, orbital} = {1/2, 3};

```

```

10 {S, L} = FindSL[SL];
11 {Sp, Lp} = FindSL[SpLp];
12 cfpSL = CFP[{numE, SL}];
13 cfpSpLp = CFP[{numE, SpLp}];
14 SLparents = First /@ Rest[cfpSL];
15 SpLpparents = First /@ Rest[cfpSpLp];
16 commonParents = Intersection[SLparents, SpLpparents];
17 Vk1 = Sum[(
18   {Sb, Lb} = FindSL[\[Psi]b];
19   Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
20   CFPAssoc[{numE, SL, \[Psi]b}] *
21   CFPAssoc[{numE, SpLp, \[Psi]b}] *
22   SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
23   SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
24 ),
25 {\[Psi]b, commonParents}
26 ];
27 prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp,
28 Lp]];
29 Return[prefactor * Vk1];
30 )
31 ];

```

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

```

1 SpinOrbit::usage = "SpinOrbit[numE, SL, SpLp, J] returns the LSJ
2   reduced matrix element  $\zeta$  <SL, J|L.S|SpLp, J>. These are given as a
3   function of  $\zeta$ . This function requires that the association
4   ReducedV1kTable be defined.
5 See equations 2-106 and 2-109 in Wybourne (1965). Equivalently see
6   eqn. 12.43 in TASS.";
7 SpinOrbit[numE_, SL_, SpLp_, J_] := Module[
8   {S, L, Sp, Lp, orbital, sign, prefactor, val},
9   (
10   orbital = 3;
11   {S, L} = FindSL[SL];
12   {Sp, Lp} = FindSL[SpLp];
13   prefactor = Sqrt[orbital * (orbital+1) * (2*orbital+1)] *
14     SixJay[{L, Lp, 1}, {Sp, S, J}];
15   sign = Phaser[J + L + Sp];
16   val = sign * prefactor *  $\zeta$  * ReducedV1kTable[{numE, SL,
17   SpLp, 1}];
18   Return[val];
19   )
20 ];

```

### 3.5 $\hat{\mathcal{H}}_{SO(3)}, \hat{\mathcal{H}}_{G_2}, \hat{\mathcal{H}}_{SO(7)}$ : electrostatic configuration interaction

These are the first terms where we take into account the contributions from *configuration-interaction*. Rajnak and Wybourne [RW63] showed that *configuration-interaction* of the electrostatic interactions corresponding to two-electron excitations from  $f^n$  can be represented through the Casimir operators of the groups  $SO(3)$ ,  $G_2$ , and  $SO(7)$ . This borrowed from an earlier insight of Trees [Tre52], who realized that an addition of a term proportional to  $L(L+1)$  improved the energy calculations for the second spectrum of manganese (Mn-II) and the third spectrum of iron (Fe-III).

One of these Casimir operators is the familiar  $\hat{L}^2$  from  $SO(3)$ . In analogy to  $\hat{L}^2$  in which the quantum number  $L$  can be used to determine the eigenvalues, in the cases of  $\hat{\mathcal{H}}_{G_2}$  the necessary state label is the  $U$  label of the  $LS$  term, and in the case of  $\hat{\mathcal{H}}_{SO(7)}$  the necessary label is  $W$ . If  $\Lambda_{G_2}(U)$  is used to note the eigenvalue of the Casimir operator of  $G_2$  corresponding to label  $U$ , and  $\Lambda_{SO(7)}(W)$  the eigenvalue corresponding to state label  $W$ , then the matrix elements of  $\hat{\mathcal{H}}_{SO(3)}$ ,  $\hat{\mathcal{H}}_{G_2}$  and  $\hat{\mathcal{H}}_{SO(7)}$  are diagonal in all quantum numbers (see Rajnak and Wybourne [RW63]) and are given by

$$\langle \ell^n \alpha S L J M_J | \hat{\mathcal{H}}_{SO(3)} | \ell'^n \alpha' S' L' J' M'_J \rangle = \alpha \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) L(L+1) \quad (30)$$

$$\langle \ell^n U \alpha S L J M_J | \hat{\mathcal{H}}_{G_2} | \ell^n U \alpha' S' L' J' M'_J \rangle = \beta \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) \Lambda_{G_2}(U) \quad (31)$$

$$\langle \ell^n W \alpha S L J M_J | \hat{\mathcal{H}}_{SO(7)} | \ell^n W \alpha' S' L' J' M'_J \rangle = \gamma \delta(\alpha S L J M_J, \alpha' S' L' J' M'_J) \Lambda_{SO(7)}(W) \quad (32)$$

In `qianth` the role of  $\Lambda_{SO(7)}(W)$  is played by the function `GS07W`, the role of  $\Lambda_{G_2}(U)$  by `GG2U`, and the role of  $\Lambda_{SO(3)}(L)$  by `CasimirS03`. These are used by `CasimirG2`, `CasimirS03`, and `CasimirS07` which find the corresponding  $U, W, L$  labels to the  $LS$  terms provided to them. Finally, the function `ElectrostaticConfigInteraction` puts them together.

```

1 ElectrostaticConfigInteraction::usage = "
2   ElectrostaticConfigInteraction[numE_, {SL, SpLp}] returns the
3   matrix element for configuration interaction as approximated by
4   the Casimir operators of the groups R3, G2, and R7. SL and SpLp
5   are strings that represent terms under LS coupling.";
6 ElectrostaticConfigInteraction[numE_, {SL_, SpLp_}] := Module[
7   {S, L, val},
8   (
9     If [
10       Or[numE == 1, numE==13],
11       Return[0];
12     ];
13     {S, L} = FindSL[SL];
14     val = (
15       If [SL == SpLp,
16         CasimirS03[{SL, SL}] +
17         CasimirS07[{SL, SL}] +
18         CasimirG2[{SL, SL}],
19         0
20       ]
21     );
22     ElectrostaticConfigInteraction[numE, {S, L}] = val;
23     Return[val];
24   )
25 ];

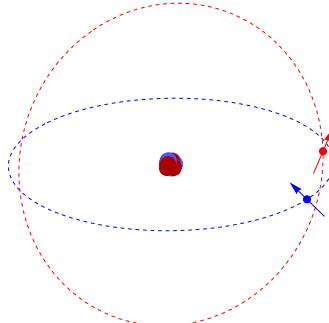
```

### 3.6 $\hat{\mathcal{H}}_{s:s-s:oo}$ : spin-spin and spin-other-orbit

The calculation of the  $\hat{\mathcal{H}}_{s:s-s:oo}$  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  $f^2$  or in  $f^3$  and then we “pull” them up for all  $f^n$  configurations with help of the coefficients of fractional parentage.

The analysis of *spin-other-orbit*, and the *spin-spin* contributions used in **qlanth** is that of Judd, Crosswhite, and Crosswhite [JCC68]. Much as the spin-orbit effect can be extracted from the Dirac equation as a relativistic correction, the multi-electron spin-orbit effects can be derived from the Breit operator  $\hat{\mathcal{H}}_B$  [BS57] which is a term added to the relativistic description of a many-particle system in order to account for retardation of the electromagnetic field

$$\hat{\mathcal{H}}_B = -\frac{1}{2}e^2 \sum_{i>j} \left[ (\alpha_i \cdot \alpha_j) \frac{1}{r_{ij}} + (\alpha_i \cdot \vec{r}_{ij}) (\alpha_j \cdot \vec{r}_{ij}) \frac{1}{r_{ij}^3} \right]. \quad (33)$$



When this operator is expanded in powers of  $v/c$ , a number of non-relativistic inter-electron interactions result. Two of them are 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

$$m^{(k)} := \frac{e^2 \hbar^2}{8m^2 c^2} \langle (nl)^2 | \frac{r_{\leq}^k}{r_{>}^{k+3}} | (nl)^2 \rangle. \quad (34)$$

With these, the expression for the *spin-spin* term within the single configuration description is [JCC68]

$$\hat{\mathcal{H}}_{s:s} = -2 \sum_{i \neq j} \sum_k m^{(k)} \sqrt{(k+1)(k+2)(2k+3)} \langle \ell | \mathcal{C}^{(k)} | \ell \rangle \langle \ell | \mathcal{C}^{(k+2)} | \ell \rangle \left\{ \hat{w}_i^{(1,k)} \hat{w}_j^{(1,k+2)} \right\}^{(2,2)0} \quad (35)$$

and the one for *spin-other-orbit*

$$\hat{\mathcal{H}}_{s:oo} = \sum_{i \neq j} \sum_k \sqrt{(k+1)(2\underline{\ell}+k+2)(2\underline{\ell}-k)} \times \\ \left[ \left\{ \hat{w}_i^{(0,k+1)} \hat{w}_j^{(1,k)} \right\}^{(11)0} \left\{ m^{(k-1)} \langle \underline{\ell} | C^{(k+1)} | \underline{\ell} \rangle^2 + 2m^{(k)} \langle \underline{\ell} | C^{(k)} | \underline{\ell} \rangle^2 \right\} + \right. \\ \left. \left\{ \hat{w}_i^{(0,k)} \hat{w}_j^{(1,k+1)} \right\}^{(11)0} \left\{ m^{(k)} \langle \underline{\ell} | C^{(k)} | \underline{\ell} \rangle^2 + 2m^{(k-1)} \langle \underline{\ell} | C^{(k+1)} | \underline{\ell} \rangle^2 \right\} \right]. \quad (36)$$

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

$$\langle \underline{\ell} | \hat{w}^{(\kappa,k)} | \underline{\ell} \rangle = \sqrt{[\kappa][k]}. \quad (37)$$

The explicit complexity of the above expressions can be somewhat reduced 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

$$\sqrt{5} \hat{\mathcal{T}}_0^{(22)} := \hat{\mathcal{H}}_{ss} \quad (38)$$

$$-\sqrt{3} \hat{\mathcal{T}}_0^{(11)} := \hat{\mathcal{H}}_{s:oo}. \quad (39)$$

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

$$\langle \gamma SLJ | \hat{\mathcal{H}} | \gamma' S' L' J' \rangle = \delta(J, J') \begin{Bmatrix} S' & L' & J \\ L & S & t \end{Bmatrix} \langle \gamma SL | \hat{\mathcal{T}}^{(tt)} | \gamma' S' L' \rangle. \quad (40)$$

This above relationship is the one effectively used in **qlanth** in the functions **SpinSpin** and **S00andECSO**.

```

1 SpinSpin::usage = "SpinSpin[n, SL, SpLp, J] returns the matrix
2   element <|SL,J|spin-spin|SpLp,J|> for the spin-spin operator
3   within the configuration f^n. This matrix element is independent
4   of MJ. This is obtained by querying the relevant reduced matrix
5   element from the association T22Table, putting in the adequate
6   phase, and 6-j symbol.
7 This is calculated according to equation (3) in ''Judd, BR, HM
8   Crosswhite, and Hannah Crosswhite. ''Intra-Atomic Magnetic
9   Interactions for f Electrons.'' Physical Review 169, no. 1 (1968):
10  130.''
11  ,,
12  ";
13 SpinSpin[numE_, SL_, SpLp_, J_] := Module[
14   {S, L, Sp, Lp, \[Alpha], val},
15   (
16     \[Alpha] = 2;
17     {S, L} = FindSL[SL];
18     {Sp, Lp} = FindSL[SpLp];
19     val = (
20       Phaser[Sp + L + J] *
21       SixJay[{Sp, Lp, J}, {L, S, \[Alpha]}] *
22       T22Table[{numE, SL, SpLp}]
23     );
24     Return[val]
25   )
26 ];
27 ]
28 
```

```

1 S00andECSO::usage = "S00andECSO[n, SL, SpLp, J] returns the matrix
2   element <|SL,J|spin-spin|SpLp,J|> for the combined effects of the
3   spin-other-orbit interaction and the electrostatically-correlated-
4   spin-orbit (which originates from configuration interaction
5   effects) within the configuration f^n. This matrix element is
6   independent of MJ. This is obtained by querying the relevant
7   reduced matrix element by querying the association
8   S00andECSOLSTable and putting in the adequate phase and 6-j symbol
9   . The S00andECSOLSTable puts together the reduced matrix elements
10  from three operators.
11 This is calculated according to equation (3) in ''Judd, BR, HM
12  Crosswhite, and Hannah Crosswhite. ''Intra-Atomic Magnetic
13  Interactions for f Electrons.'' Physical Review 169, no. 1 (1968):
14  130.''.
15 ";
16 S00andECSO[numE_, SL_, SpLp_, J_] := Module[
17 
```

```

5   {S, Sp, L, Lp, α, val},
6   (
7     α = 1;
8     {S, L} = FindSL[SL];
9     {Sp, Lp} = FindSL[SpLp];
10    val = (
11      Phaser[Sp + L + J] *
12      SixJay[{Sp, Lp, J}, {L, S, α}] *
13      SOOandECSOLSTable[{numE, SL, SpLp}]
14    );
15    Return[val];
16  )
17 ];

```

For two-electron operators such as these, the matrix elements in  $\underline{f}^n$  are related to those in  $\underline{f}^{n-1}$  by equation 4 in Judd *et al.* [JCC68]

$$\langle \underline{f}^n \psi | \hat{\mathcal{T}}^{(tt)} | \underline{f}^n \psi' \rangle = \frac{n}{n-2} \sum_{\bar{\psi}, \bar{\psi}'} (-1)^{\bar{S}+\bar{L}+\underline{s}+\underline{\ell}+S'+L'} \sqrt{[S][S'][L][L']} \times \\ (\psi \{ \bar{\psi} \} (\psi' \{ \bar{\psi}' \}) \left\{ \begin{matrix} S & t & S' \\ \bar{S}' & \underline{s} & \bar{S} \end{matrix} \right\} \left\{ \begin{matrix} L & t & L' \\ \bar{L}' & \underline{\ell} & \bar{L} \end{matrix} \right\} \langle \underline{f}^{n-1} \bar{\psi} | \hat{\mathcal{T}}^{(tt)} | \underline{f}^{n-1} ] \bar{\psi}' \rangle), \quad (41)$$

where the sum runs over the terms  $\bar{\psi}$  and  $\bar{\psi}'$  in  $\underline{f}^{n-1}$  which are parents common to  $\psi$  and  $\psi'$ . Using these the matrix elements of  $\hat{\mathcal{T}}^{(11)}$  and  $\hat{\mathcal{T}}^{(22)}$  in  $\underline{f}^2$  can be used to compute all the reduced matrix elements in  $\underline{f}^n$ . These could then be used together with Eqn-40 to obtain the matrix elements of  $\hat{\mathcal{H}}_{ss}$  and  $\hat{\mathcal{H}}_{soo}$ . This is done for  $\hat{\mathcal{H}}_{ss}$ , but not for  $\hat{\mathcal{H}}_{soo}$ , because this term is traditionally computed (with a slight modification) at the same time as the electrostatically-correlated-spin-orbit (see next section).

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

```

1 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.
2 This is an intermediate step to the calculation of the reduced matrix
  elements of the spin-spin operator.";
3 Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
4 GenerateT22Table[nmax_Integer, OptionsPattern[]] := (
5   If[And[OptionValue["Progress"], frontEndAvailable],
6     (
7       numItersai = Association[Table[numE -> Length[AllowedNKSLTerms[
8         numE]]^2, {numE, 1, nmax}]];
9       counters = Association[Table[numE -> 0, {numE, 1, nmax}]];
10      totalIters = Total[Values[numItersai[[1;; nmax]]]];
11      template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
12      template2 = StringTemplate["`remtime` min remaining"]; template3 =
13      = StringTemplate["Iteration speed = `speed` ms/it"];
14      template4 = StringTemplate["Time elapsed = `runtime` min"];
15      progBar = PrintTemporary[
16        Dynamic[
17          Pane[
18            Grid[{{Superscript["f", numE]}, {
19              template1[<|"numiter" -> numiter, "totaliter" ->
20                totalIters|>], {
21                  template4[<|"runtime" -> Round[QuantityMagnitude[
22                    UnitConvert[(Now - startTime), "min"]], 0.1]|>]}, {
23                  template2[<|"remtime" -> Round[QuantityMagnitude[
24                    UnitConvert[(Now - startTime)/(numiter)*(totalIters - numiter), "min"]
25                  ], 0.1]|>]}, {
26                  template3[<|"speed" -> Round[QuantityMagnitude[Now -
27                    startTime, "ms"]/(numiter), 0.01]|>]}, {
28                  ProgressIndicator[Dynamic[numiter], {1, totalIters
29                  }]}}, 2]
30      ];

```

```

22           Frame -> All] ,
23           Full ,
24           Alignment -> Center]
25       ];
26   ];
27 );
28 ];
29 T22Table = <||>;
30 startTime = Now;
31 numiter = 1;
32 Do [
33 (
34     numiter+= 1;
35     T22Table[{numE, SL, SpLp}] = Which[
36         numE==1,
37         0,
38         numE==2,
39         SimplifyFun[ReducedT22inf2[SL, SpLp]],
40         True,
41         SimplifyFun[ReducedT22infn[numE, SL, SpLp]]
42     ];
43 ),
44 {numE, 1, nmax},
45 {SL, AllowedNKSLTerms[numE]},
46 {SpLp, AllowedNKSLTerms[numE]}
47 ];
48 If[And[OptionValue["Progress"], frontEndAvailable],
49     NotebookDelete[progBar]
50 ];
51 If[OptionValue["Export"],
52 (
53     fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
54     Export[fname, T22Table];
55 )
56 ];
57 Return[T22Table];
58 );

```

```

1 ReducedT22infn::usage = "ReducedT22infn[n, SL, SpLp] calculates the
2      reduced matrix element of the T22 operator for the f^n
3      configuration corresponding to the terms SL and SpLp.
4 This is done by using equation (4) of 'Judd, BR, HM Crosswhite, and
5      Hannah Crosswhite. 'Intra-Atomic Magnetic Interactions for f
6      Electrons.' Physical Review 169, no. 1 (1968): 130.'
7 ";
8 ReducedT22infn[numE_, SL_, SpLp_] := Module[
9     {spin, orbital, t, idx1, idx2, S, L,
10    Sp, Lp, cfpSL, cfpSpLp, parentSL,
11    parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
12     (
13         {spin, orbital} = {1/2, 3};
14         {S, L} = FindSL[SL];
15         {Sp, Lp} = FindSL[SpLp];
16         t = 2;
17         cfpSL = CFP[{numE, SL}];
18         cfpSpLp = CFP[{numE, SpLp}];
19         Tnkk = Sum[((
20             parentSL = cfpSL[[idx2, 1]];
21             parentSpLp = cfpSpLp[[idx1, 1]];
22             {Sb, Lb} = FindSL[parentSL];
23             {Sbp, Lbp} = FindSL[parentSpLp];
24             phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
25             (
26                 phase *
27                 cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
28                 SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
29                 SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
30                 T22Table[{numE - 1, parentSL, parentSpLp}]
31             )
32         ),
33         {idx1, 2, Length[cfpSpLp]},
34         {idx2, 2, Length[cfpSL]}
35     ];
36     Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
37     Return[Tnkk];
38 )

```

35 ];

```

1 ReducedT22inf2::usage = "ReducedT22inf2[SL, SpLp] returns the reduced
2      matrix element of the scalar component of the double tensor T22
3      for the terms SL, SpLp in f^2.
4 Data used here for m0, m2, m4 is from Table I of Judd, BR, HM
5      Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
6      Interactions for f Electrons. Physical Review 169, no. 1 (1968):
7      130.
8 ";
9 ReducedT22inf2[SL_, SpLp_] := Module[
10   {statePosition, PsiPsipStates, m0, m2, m4, Tk2m},
11   (
12     T22inf2 = <|
13       {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
14       {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
15       {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
16       {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
17       {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
18     |>;
19     Which[
20       MemberQ[Keys[T22inf2], {SL, SpLp}],
21         Return[T22inf2[{SL, SpLp}]],
22       MemberQ[Keys[T22inf2], {SpLp, SL}],
23         Return[T22inf2[{SpLp, SL}]],
24       True,
25         Return[0]
26     ]
27   )
28 ];
29 
```

```

1 Reducedt11inf2::usage = "Reducedt11inf2[SL, SpLp] returns the reduced
2      matrix element in f^2 of the double tensor operator tii for the
3      corresponding given terms {SL, SpLp}.
4 Values given here are those from Table VII of ''Judd, BR, HM
5      Crosswhite, and Hannah Crosswhite. ''Intra-Atomic Magnetic
6      Interactions for f Electrons.'' Physical Review 169, no. 1 (1968):
7      130.''
8 ";
9 Reducedt11inf2[SL_, SpLp_] := Module[
10   {t11inf2},
11   (
12     t11inf2 = <|
13       {"1S", "3P"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
14       {"3P", "3P"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
15       {"3P", "1D"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
16       {"1D", "3F"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
17       {"3F", "3F"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
18       {"3F", "1G"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
19       {"1G", "3H"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
20       {"3H", "3H"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
21       {"3H", "1I"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
22     |>;
23     Which[
24       MemberQ[Keys[t11inf2], {SL, SpLp}],
25         Return[t11inf2[{SL, SpLp}]],
26       MemberQ[Keys[t11inf2], {SpLp, SL}],
27         Return[t11inf2[{SpLp, SL}]],
28       True,
29         Return[0]
30     ]
31   )
32 ];
33 
```

### 3.7 $\hat{\mathcal{H}}_{\text{ecs:o}}$ : electrostatically-correlated-spin-orbit

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

$$\hat{\mathcal{H}}_{\text{ci}} = - \sum_{\chi} \sum_i \frac{1}{E_{\chi}} \xi(r_i) (\hat{\mathbf{j}}_i \cdot \hat{\mathbf{l}}_i) |\chi\rangle \langle \chi| \hat{\mathbf{C}} - \frac{1}{E_{\chi}} \hat{\mathbf{C}} |\chi\rangle \langle \chi| \xi(r_i) (\hat{\mathbf{j}}_i \cdot \hat{\mathbf{l}}_i) \quad (42)$$

where  $\xi(r_h)(\hat{\mathbf{J}}_h \cdot \hat{\mathbf{L}}_h)$  is the customary spin-orbit interaction,  $E_\chi$  is the energy of state  $|\chi\rangle$ ,  $i$  is a label for the valence electrons,  $\hat{\mathcal{C}}$  stands for the Coulomb interaction, and  $|\chi\rangle$  are states in the configurations with which one is “interacting”. Since this term includes both the electrostatic term and the spin-orbit one, this is called the *electrostatically-correlated-spin-orbit* interaction.

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

$$\hat{\mathcal{H}}_{ci} = -\sqrt{3} \hat{t}_0^{(11)}. \quad (43)$$

Judd *et al.* [JCC68] then go on to list the reduced matrix elements of this operator in the  $f^2$  configuration. When this is done the Marvin integrals  $\mathcal{M}^{(k)}$  appear again, but a second set of parameters, the *pseudo-magnetic* parameters  $\mathcal{P}^{(k)}$ , is also necessary

$$\mathcal{P}^{(k)} = 6 \sum_{f'} \frac{\zeta_{ff'}}{E_{ff'}} R^{(k)}(ff, ff') \text{ for } k = 0, 2, 4, 6. \quad (44)$$

Where  $f$  stands for an  $f$ -electron radial eigenfunction, and  $f'$  similarly but for a configuration different from  $f^n$ . And where

$$\zeta_{ff'} := \langle f | \xi(r) | f' \rangle \quad (45)$$

$$R^{(k)}(ff, ff') := e^2 \langle f_1 f_2 | \frac{r_{<}^k}{r_{>}^{k+1}} | f_1 f'_2 \rangle. \quad (46)$$

In the semi-empirical approach embodied by **qlanth**, calculating these quantities *ab initio* is not the objective, they are instead to be defined from experiments. Nonetheless, not only these expressions give theoretical justification to the model, but they also serve to justify the ratios between different orders of these quantities, their relative importance, or their sign. Consequently, both the set of three  $\mathcal{M}^{(k)}$  and the set of  $\mathcal{P}^{(k)}$  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  $\mathcal{P}^{(0)}$  is proportional to the spin orbit operator, and as such its effect is absorbed by the standard spin-orbit parameter  $\zeta$ . 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

$$\langle \psi | \hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)} | \psi' \rangle = \sum a_i \langle \psi | \hat{z}_i | \psi' \rangle. \quad (47)$$

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

$$\hat{\mathcal{H}}_{so:oo} + \hat{\mathcal{H}}_{ecs:o} = \hat{\mathcal{T}}^{(11)} + \hat{t}^{(11)} - \frac{1}{6} a_{13} \hat{z}_{13} \quad (48)$$

where

$$a_{13} = -33\mathcal{M}^{(0)} + 3\mathcal{M}^{(2)} + \frac{15}{11}\mathcal{M}^{(4)} - 6\mathcal{P}^{(0)} + \frac{3}{2} \left( \frac{35}{225}\mathcal{P}^{(2)} + \frac{77}{1089}\mathcal{P}^{(4)} + \frac{25}{1287}\mathcal{P}^{(6)} \right). \quad (49)$$

In **qlanth** the contributions from *spin-spin*, *spin-other-orbit*, and *electrostatically-correlated-spin-orbit* are put together by the function **MagneticInteractions**. That function queries precomputed values from two associations **SpinSpinTable** and **S0OandECSTable**. In turn these two associations are generated by the functions **GenerateSpinOrbitTable** and **GenerateS0OandECSTable**. Note that both *spin-spin* and *spin-other-orbit* end up contributing through  $\mathcal{M}^{(k)}$ , 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  $\sigma_{SS}$  (1 for including, 0 for excluding).

```
1 MagneticInteractions::usage = "MagneticInteractions[{numE, SL, SLP, J
  }] returns the matrix element of the magnetic interaction between
  the terms SL and SLP in the f^numE configuration for the given
  value of J. The interaction is given by the sum of the spin-spin,
  the spin-other-orbit, and the electrostatically-correlated-spin-
  orbit interactions."
```

```

2 The part corresponding to the spin-spin interaction is provided by
3   SpinSpin[{numE, SL, SLP, J}].
4 The part corresponding to SOO and ECSO is provided by the function
5   SOOandECSO[{numE, SL, SLP, J}].
6 The option ''ChenDeltas'' can be used to include or exclude the Chen
7   deltas from the calculation. The default is to exclude them. If
8   this option is used, then the chenDeltas association needs to be
9   loaded into the session with LoadChenDeltas[].";
10 Options[MagneticInteractions] = {"ChenDeltas" -> False};
11 MagneticInteractions[{numE_, SL_, SLP_, J_}, OptionsPattern[]] :=
12   Module[
13     {key, ss, sooandecso, total,
14      S, L, Sp, Lp, phase, sixjay,
15      M0v, M2v, M4v,
16      P2v, P4v, P6v},
17     (
18       key = {numE, SL, SLP, J};
19       ss = \[Sigma]SS * SpinSpinTable[key];
20       sooandecso = SOOandECSOTable[key];
21       total = ss + sooandecso;
22       total = SimplifyFun[total];
23       If[
24         Not[OptionValue["ChenDeltas"]],
25         Return[total]
26       ];
27       (* In the type A errors the wrong values are different *)
28       If[MemberQ[Keys[chenDeltas["A"]], {numE, SL, SLP}],
29         (
30           {S, L} = FindSL[SL];
31           {Sp, Lp} = FindSL[SLP];
32           phase = Phaser[Sp + L + J];
33           sixjay = SixJay[{Sp, Lp, J}, {L, S, 1}];
34           {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SL,
35             SLP}]["wrong"];
36           total = (
37             phase * sixjay *
38             (
39               M0v*M0 + M2v*M2 + M4v*M4 +
40               P2v*P2 + P4v*P4 + P6v*P6
41             )
42           );
43           total = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
44         )
45       );
46       (* In the type B errors the wrong values are zeros all around *)
47       If[MemberQ[chenDeltas["B"], {numE, SL, SLP}],
48         (
49           total = (1 - wChErrB) * (ss + sooandecso)
50         )
51       ];
52       Return[total];
53     )
54   ];

```

```

1 GenerateSpinOrbitTable::usage = "GenerateSpinOrbitTable[nmax]
2   computes the matrix elements for the spin-orbit interaction for f^
3   n configurations up to n = nmax. The function returns an
4   association whose keys are lists of the form {n, SL, SpLp, J}. If
5   ''Export'' is set to True, then the result is exported to the data
6   folder. It requires ReducedV1kTable to be defined.";
7 Options[GenerateSpinOrbitTable] = {"Export" -> True};
8 GenerateSpinOrbitTable[nmax_Integer:7, OptionsPattern[]] := Module[
9   {numE, J, SL, SpLp, exportFname},
10   (
11     SpinOrbitTable =
12     Table[
13       {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
14       {numE, 1, nmax},
15       {J, MinJ[numE], MaxJ[numE]},
16       {SL, Map[First, AllowedNKSLforJTerms[numE, J]]},
17       {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
18     ];
19     SpinOrbitTable = Association[SpinOrbitTable];
20     exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}]
21   ];

```

```

    }];
17  If[OptionValue["Export"],
18   (
19     Echo["Exporting to file " <> ToString[exportFname]];
20     Export[exportFname, SpinOrbitTable];
21   )
22 ];
23  Return[SpinOrbitTable];
24 )
25 ];

```

```

1 GenerateS0OandECSOTable::usage = "GenerateS0OandECSOTable[nmax]
generates the reduced 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.";
2 Options[GenerateS0OandECSOTable] = {"Export" -> False};
3 GenerateS0OandECSOTable[nmax_, OptionsPattern[]] :=
4   S0OandECSOTable = <||>;
5   Do[
6     S0OandECSOTable[{numE, SL, SpLp, J}] = (S0OandECSO[numE, SL, SpLp
, J] /. Prescaling);
7     {numE, 1, nmax},
8     {J, MinJ[numE], MaxJ[numE]},
9     {SL, First /@ AllowedNKSLforJTerms[numE, J]},
10    {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
11  ];
12  If[OptionValue["Export"],
13   (
14     fname = FileNameJoin[{moduleDir, "data", "S0OandECSOTable.m"}];
15     Export[fname, S0OandECSOTable];
16   )
17 ];
18  Return[S0OandECSOTable];
19 );

```

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 double tensor  $\mathcal{T}^{(22)}$  which are stored in the association `T22Table`.

```

1 GenerateSpinSpinTable::usage = "GenerateSpinSpinTable[nmax] generates
the reduced matrix elements in the |LSJ> basis for the spin-spin
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.";
2 Options[GenerateSpinSpinTable] = {"Export" -> False};
3 GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
4   (
5     SpinSpinTable = <||>;
6     PrintTemporary[Dynamic[numE]];
7     Do[
8       SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp,
J]);
9       {numE, 1, nmax},
10      {J, MinJ[numE], MaxJ[numE]},
11      {SL, First /@ AllowedNKSLforJTerms[numE, J]},
12      {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
13    ];
14    If[OptionValue["Export"],
15      (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
16        Export[fname, SpinSpinTable];
17      )
18    ];
19    Return[SpinSpinTable];
20 );

```

```

1 SpinSpin::usage = "SpinSpin[n, SL, SpLp, J] returns the matrix
2   element <|SL,J|spin-spin|SpLp,J|> for the spin-spin operator
3   within the configuration f^n. This matrix element is independent
4   of MJ. This is obtained by querying the relevant reduced matrix
5   element from the association T22Table, putting in the adequate
6   phase, and 6-j symbol.
7 This is calculated according to equation (3) in ''Judd, BR, HM
8   Crosswhite, and Hannah Crosswhite. ''Intra-Atomic Magnetic
9   Interactions for f Electrons.'' Physical Review 169, no. 1 (1968):
10  130.''
11  '';
12  ";
13 SpinSpin[numE_, SL_, SpLp_, J_] := Module[
14   {S, L, Sp, Lp, α, val},
15   (
16     α = 2;
17     {S, L} = FindSL[SL];
18     {Sp, Lp} = FindSL[SpLp];
19     val = (
20       Phaser[Sp + L + J] *
21       SixJay[{Sp, Lp, J}, {L, S, α}] *
22       T22Table[{numE, SL, SpLp}]
23     );
24     Return[val]
25   )
26 ];

```

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  $f^2$ , and `ReducedT22infn` for configurations above  $f^2$ . When `ReducedT22infn` is called, the sum in [Eqn-41](#) is carried out using  $t = 2$ . When `ReducedT22inf2` is called, the reduced matrix elements from [\[JCC68\]](#) are used.

```

1 GenerateT22Table::usage = "GenerateT22Table[nmax] generates the LS
2   reduced matrix elements for the double tensor operator T22 in f^n
3   up to n=nmax. If the option ''Export'' is set to true then the
4   resulting association is saved to the data folder. The values for
5   n=1 and n=2 are taken from ''Judd, BR, HM Crosswhite, and Hannah
6   Crosswhite. ''Intra-Atomic Magnetic Interactions for f Electrons
7   .'' Physical Review 169, no. 1 (1968): 130.'', and the values for
8   n>2 are calculated recursively using equation (4) of that same
9   paper.
10 This is an intermediate step to the calculation of the reduced matrix
11  elements of the spin-spin operator.";
12 Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
13 GenerateT22Table[nmax_Integer, OptionsPattern[]] := (
14   If[And[OptionValue["Progress"], frontEndAvailable],
15   (
16     numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
17       numE]]^2, {numE, 1, nmax}]];
18     counters = Association[Table[numE->0, {numE, 1, nmax}]];
19     totalIters = Total[Values[numItersai[[1;;nmax]]]];
20     template1 = StringTemplate["Iteration `numiter` of `totaliter`"]
21   ];
22   template2 = StringTemplate["`remtime` min remaining"];template3 =
23   StringTemplate["Iteration speed = `speed` ms/it"];
24   template4 = StringTemplate["Time elapsed = `runtime` min"];
25   progBar = PrintTemporary[
26     Dynamic[
27       Pane[
28         Grid[{{Superscript["f", numE]}, {
29           template1[<|"numiter"->numiter, "totaliter"->
30           totalIters|>]},
31           {template4[<|"runtime"->Round[QuantityMagnitude[
32             UnitConvert[(Now-startTime), "min"]], 0.1]|>}],
33           {template2[<|"remtime"->Round[QuantityMagnitude[
34             UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"]
35           ], 0.1]|>]},
36           {template3[<|"speed"->Round[QuantityMagnitude[Now-
37             startTime, "ms"]/(numiter), 0.01]|>]},
38           {ProgressIndicator[Dynamic[numiter], {1, totalIters
39           }]}},
40           Frame -> All],
41           Full,
42           Alignment -> Center]
43     ]

```

```

26           ];
27       )
28   ];
29 T22Table = <||>;
30 startTime = Now;
31 numiter = 1;
32 Do [
33 (
34     numiter+= 1;
35     T22Table[{numE, SL, SpLp}] = Which[
36         numE==1,
37         0,
38         numE==2,
39         SimplifyFun[ReducedT22inf2[SL, SpLp]],
40         True,
41         SimplifyFun[ReducedT22infn[numE, SL, SpLp]]
42     ];
43     ),
44     {numE, 1, nmax},
45     {SL, AllowedNKSLTerms[numE]},
46     {SpLp, AllowedNKSLTerms[numE]}
47   ];
48 If[And[OptionValue["Progress"], frontEndAvailable],
49   NotebookDelete[progBar]
50 ];
51 If[OptionValue["Export"],
52 (
53   fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
54   Export[fname, T22Table];
55 )
56 ];
57 Return[T22Table];
58 );

```

```

1 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.
2 This is done 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.'
3 ";
4 ReducedT22infn[numE_, SL_, SpLp_] := Module[
5   {spin, orbital, t, idx1, idx2, S, L,
6   Sp, Lp, cfpSL, cfpSpLp, parentSL,
7   parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
8   (
9     {spin, orbital} = {1/2, 3};
10    {S, L} = FindSL[SL];
11    {Sp, Lp} = FindSL[SpLp];
12    t = 2;
13    cfpSL = CFP[{numE, SL}];
14    cfpSpLp = CFP[{numE, SpLp}];
15    Tnkk = Sum[(
16      parentSL = cfpSL[[idx2, 1]];
17      parentSpLp = cfpSpLp[[idx1, 1]];
18      {Sb, Lb} = FindSL[parentSL];
19      {Sbp, Lbp} = FindSL[parentSpLp];
20      phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
21      (
22        phase *
23        cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
24        SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
25        SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
26        T22Table[{numE - 1, parentSL, parentSpLp}]
27      )
28    ),
29    {idx1, 2, Length[cfpSpLp]},
30    {idx2, 2, Length[cfpSL]}
31  ];
32  Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
33  Return[Tnkk];
34 )
35 ];

```

```

1 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.
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.
3 ";
4 ReducedT22inf2[SL_, SpLp_] := Module[
5   {statePosition, PsiPsipStates, m0, m2, m4, Tk2m},
6   (
7     T22inf2 = <|
8       {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
9       {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
10      {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
11      {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
12      {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
13     |>;
14     Which[
15       MemberQ[Keys[T22inf2], {SL, SpLp}],
16       Return[T22inf2[{SL, SpLp}]],
17       MemberQ[Keys[T22inf2], {SpLp, SL}],
18       Return[T22inf2[{SpLp, SL}]],
19       True,
20       Return[0]
21     ]
22   )
23 ];

```

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 [Eqn-48](#) as stored in the association `SOOandECSOLSTable`.

```

1 GenerateSOOandECSOTable::usage = "GenerateSOOandECSOTable[nmax]
generates the reduced 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.";
2 Options[GenerateSOOandECSOTable] = {"Export" -> False};
3 GenerateSOOandECSOTable[nmax_, OptionsPattern[]] :=
4   SOOandECSOTable = <||>;
5   Do[
6     SOOandECSOTable[{numE, SL, SpLp, J}] = (SOOandECSO[numE, SL, SpLp
   , J] /. Prescaling);
7     {numE, 1, nmax},
8     {J, MinJ[numE], MaxJ[numE]},
9     {SL, First /@ AllowedNKSLforJTerms[numE, J]},
10    {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
11  ];
12  If[OptionValue["Export"],
13  (
14    fname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
15    Export[fname, SOOandECSOTable];
16  )
17  ];
18  Return[SOOandECSOTable];
19 );

```

```

1 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.
2 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.''.
3 ";

```

```

4 S00andECSO[numE_, SL_, SpLp_, J_] := Module[
5   {S, Sp, L, Lp, α, val},
6   (
7     α = 1;
8     {S, L} = FindSL[SL];
9     {Sp, Lp} = FindSL[SpLp];
10    val = (
11      Phaser[Sp + L + J] *
12      SixJay[{Sp, Lp, J}, {L, S, α}] *
13      S00andECSOLSTable[{numE, SL, SpLp}]
14    );
15    Return[val];
16  )
17 ];

```

```

1 S00andECSO::usage = "S00andECSO[n, SL, SpLp, J] returns the matrix
2   element <|SL,J|spin-spin|SpLp,J|> for the combined effects of the
3   spin-other-orbit interaction and the electrostatically-correlated-
4   spin-orbit (which originates from configuration interaction
5   effects) within the configuration f^n. This matrix element is
6   independent of MJ. This is obtained by querying the relevant
7   reduced matrix element by querying the association
8   S00andECSOLSTable and putting in the adequate phase and 6-j symbol
9   . The S00andECSOLSTable puts together the reduced matrix elements
10  from three operators.
11 This is calculated according to equation (3) in ''Judd, BR, HM
12  Crosswhite, and Hannah Crosswhite. ''Intra-Atomic Magnetic
13  Interactions for f Electrons.'' Physical Review 169, no. 1 (1968):
14  130.''.
15 ";
16 S00andECSO[numE_, SL_, SpLp_, J_] := Module[
17   {S, Sp, L, Lp, α, val},
18   (
19     α = 1;
20     {S, L} = FindSL[SL];
21     {Sp, Lp} = FindSL[SpLp];
22     val = (
23       Phaser[Sp + L + J] *
24       SixJay[{Sp, Lp, J}, {L, S, α}] *
25       S00andECSOLSTable[{numE, SL, SpLp}]
26     );
27     Return[val];
28   )
29 ];

```

The association `S00andECSOLSTable` is computed by the function `GenerateS00andECSOLSTable`. This function populates `S00andECSOLSTable` with keys of the form  $\{n, SL, S'L'\}$ . It does this by using the function `ReducedS00andECSOinf2` in the base case of  $f^2$ , and `ReducedS00andECSOinfn` for configurations above  $f^2$ . When `ReducedS00andECSOinfn` is called the sum in [Eqn-41](#) is carried out using  $t = 1$ . When `ReducedS00andECSOinf2` is called the reduced matrix elements from [\[JCC68\]](#) are used.

```

1 ReducedS00andECSOinfn::usage = "ReducedS00andECSOinfn[numE, SL, SpLp]
2   calculates the reduced matrix elements of the (spin-other-orbit +
3   ECSO) operator for the f^numE configuration corresponding to the
4   terms SL and SpLp. This is done recursively, starting from
5   tabulated values for f^2 from ''Judd, BR, HM Crosswhite,
6   and Hannah Crosswhite. ''Intra-Atomic Magnetic Interactions for f
7   Electrons.'' Physical Review 169, no. 1 (1968): 130.'', and by
8   using equation (4) of that same paper.
9 ";
10 ReducedS00andECSOinfn[numE_, SL_, SpLp_] := Module[
11   {spin, orbital, t, S, L, Sp, Lp,
12   idx1, idx2, cfpSL, cfpSpLp, parentSL,
13   Sb, Lb, Sbp, Lbp, parentSpLp, funval},
14   (
15     {spin, orbital} = {1/2, 3};
16     {S, L} = FindSL[SL];
17     {Sp, Lp} = FindSL[SpLp];
18     t = 1;
19     cfpSL = CFP[{numE, SL}];
20     cfpSpLp = CFP[{numE, SpLp}];
21     funval = Sum[
22       (
23         parentSL = cfpSL[[idx2, 1]];
24       )
25     ];
26   );
27   Return[funval];
28 ];

```

```

17     parentSpLp = cfpSpLp[[idx1, 1]];
18     {Sb, Lb} = FindSL[parentSL];
19     {Sbp, Lbp} = FindSL[parentSpLp];
20     phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
21     (
22       phase *
23       cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
24       SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
25       SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
26       S00andECSOLSTable[{numE - 1, parentSL, parentSpLp}]
27     )
28   ),
29   {idx1, 2, Length[cfpSpLp]},
30   {idx2, 2, Length[cfpSL]}
31 ];
32   funval *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
33   Return[funval];
34 )
35 ];

```

```

1 ReducedS00andECSOinf2::usage = "ReducedS00andECSOinf2[SL, SpLp]
2   returns the reduced matrix element corresponding to the operator (
3     T11 + t11 - a13 * z13 / 6) for the terms {SL, SpLp}. This
4   combination of operators corresponds to the spin-other-orbit plus
5   ECSO interaction.
6 The T11 operator corresponds to the spin-other-orbit interaction, and
7   the t11 operator (associated with electrostatically-correlated
8   spin-orbit) originates from configuration interaction analysis. To
9   their sum a factor proportional to the operator z13 is subtracted
10  since its effect is redundant to the spin-orbit interaction. The
11  factor of 1/6 is not on Judd's 1968 paper, but it is on ''Chen,
12  Xueyuan, Guokui Liu, Jean Margerie, and Michael F Reid. ''A Few
13  Mistakes in Widely Used Data Files for Fn Configurations
14  Calculations.'' Journal of Luminescence 128, no. 3 (2008):
15  421-27''.
16 The values for the reduced matrix elements of z13 are obtained from
17  Table IX of the same paper. The value for a13 is from table VIII.
18 Rigorously speaking the Pk parameters here are subscripted. The
19  conversion to superscripted parameters is performed elsewhere with
20  the Prescaling replacement rules.
21 ";
22 ReducedS00andECSOinf2[SL_, SpLp_] := Module[
23   {a13, z13, z13inf2, matElement, redS00andECSOinf2},
24   (
25     a13 = (-33 M0 + 3 M2 + 15/11 M4 -
26             6 P0 + 3/2 (35 P2 + 77 P4 + 143 P6));
27     z13inf2 = <|
28       {"1S", "3P"} -> 2,
29       {"3P", "3P"} -> 1,
30       {"3P", "1D"} -> -Sqrt[(15/2)],
31       {"1D", "3F"} -> Sqrt[10],
32       {"3F", "3F"} -> Sqrt[14],
33       {"3F", "1G"} -> -Sqrt[11],
34       {"1G", "3H"} -> Sqrt[10],
35       {"3H", "3H"} -> Sqrt[55],
36       {"3H", "1I"} -> -Sqrt[(13/2)]
37     |>;
38     matElement = Which[
39       MemberQ[Keys[z13inf2], {SL, SpLp}],
40         z13inf2[{SL, SpLp}],
41       MemberQ[Keys[z13inf2], {SpLp, SL}],
42         z13inf2[{SpLp, SL}],
43       True,
44         0
45     ];
46     redS00andECSOinf2 = (
47       ReducedT11inf2[SL, SpLp] +
48       Reducedt11inf2[SL, SpLp] -
49       a13 / 6 * matElement
50     );
51     redS00andECSOinf2 = SimplifyFun[redS00andECSOinf2];
52     Return[redS00andECSOinf2];
53   )
54 ];

```

### 3.8 $\hat{\mathcal{H}}_{\lambda}$ : three-body effective operators

The three-body operators in the semi-empirical Hamiltonian are due to the *configuration-interaction* effects of the Coulomb repulsion. More specifically, they originate from configuration interaction between the ground configuration  $(4f)^n$  and single electron excitations to the  $(4f)^{n \pm 1}(n'\ell')^{\mp 1}$  configurations.

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

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

$$\hat{\mathcal{H}}_{\lambda} = T'^{(2)}\hat{t}_2' + T'^{(11)}\hat{t}_{11}' \sum_{\substack{k=2,3,4,6,7,8, \\ 11,12,14,15, \\ 16,17,18,19}} T^{(k)}\hat{t}_k, \quad (50)$$

where  $\hat{t}_2$  and  $\hat{t}_{11}$  are operators that have orthogonal alternatives represented by  $\hat{t}_2'$  and  $\hat{t}_{11}'$  (see [JS84]). **qlanth** includes the legacy operator  $\hat{t}_2$  since it was used for important work during and before the 1980s.

The omission of some indices in this sum has to do with the fact that the way in which these are defined in terms of their index (see [Jud66]) gives rise to two-body operators which can be absorbed by the two-body terms in the Hamiltonian. As such, it is not so much that they are not included, but rather that their effects are considered to be accounted for elsewhere. This is representative of a common feature of configuration interaction: it gives rise to new intra-configuration operators, but it also contributes to already present operators; this makes it harder to approximate the model parameters *ab initio*, but is not a practical obstacle for the semi-empirical approach (although it certainly complicates the physical interpretation that each parameter has). Furthermore, it is often the case that the operator set is limited to the subset  $\{2,3,4,6,7,8\}$ ; a practice that is justified *post-facto* after seeing that these are sufficient to describe the data.

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

$$\langle f^n \psi | \hat{t}_i | f^n \psi' \rangle = \delta(J, J') \delta(M_J, M'_J) \frac{n}{n-3} \sum_{\bar{\psi}\bar{\psi}'} (\psi \{ \bar{\psi} \}) (\psi' \{ \bar{\psi}' \}) \langle f^{n-1} \bar{\psi} | \hat{t}_i | f^{n-1} \bar{\psi}' \rangle. \quad (51)$$

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

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

In `GenerateThreeBodyTables` a special case is made for  $\hat{t}_2$  and  $\hat{t}_{11}$  which are calculated differently beyond the half-filled shell. In the case of the other  $\hat{t}_k$  operators, beyond  $f^7$  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  $f^7$  are used. This yields the unexpected result that in the  $f^{12}$  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  $\hat{t}_2'/\hat{t}_{11}'$  is used then  $\hat{t}_2/\hat{t}_{11}$  should not be used and vice versa.

One additional feature of  $\hat{t}_2$  that needs to be accounted for, is that it doesn't have the simple relationship for conjugate configurations that all the other  $\hat{t}_i$  operators have. For the sake of simplicity, and to 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  $f^7$  and set to 0 if above  $f^7$ .

```

1 GenerateThreeBodyTables::usage = "This function generates the reduced
   matrix elements for the three body operators using the
   coefficients of fractional parentage, including those beyond f^7."
;
2 Options[GenerateThreeBodyTables] = {"Export" -> False};
3 GenerateThreeBodyTables[OptionsPattern[]] := (

```

```

4   tiKeys      = (StringReplace[ToString[#], {"T" -> "t_#", "p" -> " "
5     }^{"{", "}" ] <-> "}") & /@ TSymbols;
6   TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
7   juddOperators = ParseJudd1984[];
8   (* 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. *)
9   op3MatrixElement[SL_, SpLp_, opSymbol_] := (
10    jOP = juddOperators[{3, opSymbol}];
11    key = {SL, SpLp};
12    val = If[MemberQ[Keys[jOP], key],
13      jOP[key],
14      0];
15    Return[val];
16  );
17  (* ti: This is the implementation of formula (2) in Judd & Suskin
18    1984. It computes the reduced matrix elements of ti in f^n by
19    using the reduced matrix elements in f^3 and the coefficients of
20    fractional parentage. If the option 'Fast' is set to True then
21    the values for n>7 are simply computed as the negatives of the
22    values in the complementary configuration; this except for t2 and
23    t11 which are treated as special cases. *)
24  Options[ti] = {"Fast" -> True};
25  ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
26    Module[
27      {nn, S, L, Sp, Lp,
28       cfpSL, cfpSpLp,
29       parentSL, parentSpLp,
30       tnk, tnks},
31      (
32        {S, L} = FindSL[SL];
33        {Sp, Lp} = FindSL[SpLp];
34        fast = OptionValue["Fast"];
35        numH = 14 - nE;
36        If[fast && Not[MemberQ[{t_2, t_11}, tiKey]] && nE > 7,
37          Return[-tktable[{numH, SL, SpLp, tiKey}]];
38        ];
39        If[(S == Sp && L == Lp),
40          (
41            cfpSL = CFP[{nE, SL}];
42            cfpSpLp = CFP[{nE, SpLp}];
43            tnks = Table[(
44              parentSL = cfpSL[[nn, 1]];
45              parentSpLp = cfpSpLp[[mm, 1]];
46              cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
47              tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
48            ),
49            {nn, 2, Length[cfpSL]},
50            {mm, 2, Length[cfpSpLp]}
51          ];
52          tnk = Total[Flatten[tnks]];
53        ),
54        tnk = 0;
55      ];
56      Return[nE / (nE - opOrder) * tnk];
57    )
58  ];
59  (* Calculate the reduced matrix elements of t^i for n up to 14 *)
60  tktable = <||>;
61  Do[[
62    Do[[
63      tkValue = Which[numE <= 2,
64        (* Initialize n=1,2 with zeros*)
65        0,
66        numE == 3,
67        (* Grab matrix elem in f^3 from Judd 1984 *)
68        SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
69        True,
70        SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2,
71          3]]];
72      ];
73      tktable[{numE, SL, SpLp, opKey}] = tkValue;
74    ],
75    {SL, AllowedNKSLTerms[numE]},
76    {SpLp, AllowedNKSLTerms[numE]},
77    {opKey, Append[tiKeys, "e_{3}"]}]
78  ];

```

```

69 ];
70 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " "
71 configuration complete"]];
72 ),
73 {numE, 1, 14}
74 ];
75 (* Now use those reduced matrix elements to determine their sum as
76 weighted by their corresponding strengths Ti *)
77 ThreeBodyTable = <||>;
78 Do[
79 Do[
80 (
81 ThreeBodyTable[{numE, SL, SpLp}] = (
82 Sum[(
83 If[tiKey == "t_{2}", t2Switch, 1] *
84 tktable[{numE, SL, SpLp, tiKey}] *
85 TSymbolsAssoc[tiKey] +
86 If[tiKey == "t_{2}", 1 - t2Switch, 0] *
87 (-tktable[{14 - numE, SL, SpLp, tiKey}]) *
88 TSymbolsAssoc[tiKey]
89 ),
90 {tiKey, tiKeys}
91 ]
92 );
93 {SL, AllowedNKSLTerms[numE]},
94 {SpLp, AllowedNKSLTerms[numE]}
95 ];
96 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix
97 complete"]];
98 {numE, 1, 7}
99 ];
100 ThreeBodyTables = Table[(
101 terms = AllowedNKSLTerms[numE];
102 singleThreeBodyTable =
103 Table[
104 {SL, SLP} -> ThreeBodyTable[{numE, SL, SLP}],
105 {SL, terms},
106 {SLP, terms}
107 ];
108 singleThreeBodyTable = Flatten[singleThreeBodyTable];
109 singleThreeBodyTables = Table[(
110 notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
111 reps = ConstantArray[0, Length[TSymbols]];
112 reps[[notNullPosition]] = 1;
113 rep = AssociationThread[TSymbols -> reps];
114 notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
115 ),
116 {notNullSymbol, TSymbols}
117 ];
118 singleThreeBodyTables = Association[singleThreeBodyTables];
119 numE -> singleThreeBodyTables),
120 {numE, 1, 7}
121 ];
122 ThreeBodyTables = Association[ThreeBodyTables];
123 If[OptionValue["Export"],
124 (
125 threeBodyTablefname = FileNameJoin[{moduleDir, "data", " "
126 ThreeBodyTable.m}];
127 Export[threeBodyTablefname, ThreeBodyTable];
128 threeBodyTablesfname = FileNameJoin[{moduleDir, "data", " "
129 ThreeBodyTables.m}];
130 Export[threeBodyTablesfname, ThreeBodyTables];
131 )
132 ];
133 Return[{ThreeBodyTable, ThreeBodyTables}];
134 );

```

```

1 ParseJudd1984::usage = "This function parses the data from tables 1
2 and 2 of Judd from Judd, BR, and MA Suskin. ''Complete Set of
3 Orthogonal Scalar Operators for the Configuration f^3''. JOSA B 1,
4 no. 2 (1984): 261-65.";
5 Options[ParseJudd1984] = {"Export" -> False};

```

```

3 ParseJudd1984[OptionsPattern[]] := (
4   ParseJuddTab1[str_] := (
5     strR = ToString[str];
6     strR = StringReplace[strR, ".5" -> "^(1/2)"];
7     num = ToExpression[strR];
8     sign = Sign[num];
9     num = sign*Simplify[Sqrt[num^2]];
10    If[Round[num] == num, num = Round[num]];
11    Return[num]);
12
13 (* Parse table 1 from Judd 1984 *)
14 judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"}];
15 data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
16 headers = data[[1]];
17 data = data[[2 ;;]];
18 data = Transpose[data];
19 \[Psi] = Select[data[[1]], # != "" &];
20 \[Psi]p = Select[data[[2]], # != "" &];
21 matrixKeys = Transpose[{\[Psi], \[Psi]p}];
22 data = data[[3 ;;]];
23 cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data}];
24 cols = Select[cols, Length[#] == 21 &];
25 tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
26 tab1 = Transpose[Prepend[Transpose[tab1], headers]];
27
28 (* Parse table 2 from Judd 1984 *)
29 judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"}];
30 data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
31 headers = data[[1]];
32 data = data[[2 ;;]];
33 data = Transpose[data];
34 {operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} =
35   data[[;; 4]];
36 multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
37 multiFactorValues = AssociationThread[multiFactorSymbols ->
38   multiFactorValues];
39
40 (*scale values of table 1 given the values in table 2*)
41 oppyS = {};
42 normalTable =
43   Table[header = col[[1]];
44     If[StringContainsQ[header, " "],
45       (
46         multiplierSymbol = StringSplit[header, " "][[1]];
47         multiplierValue = multiFactorValues[multiplierSymbol];
48         operatorSymbol = StringSplit[header, " "][[2]];
49         oppyS = Append[oppyS, operatorSymbol];
50       ),
51       (
52         multiplierValue = 1;
53         operatorSymbol = header;
54       )
55     ];
56     normalValues = 1/multiplierValue*col[[2 ;;]];
57     Join[{operatorSymbol}, normalValues], {col, tab1[[3 ;;]]}]
58   ];
59
60 (*Create an association for the reduced matrix elements in the f^3
61 config*)
62 juddOperators = Association[];
63 Do[(
64   col      = normalTable[[colIndex]];
65   opLabel = col[[1]];
66   opValues = col[[2 ;;]];
67   opMatrix = AssociationThread[matrixKeys -> opValues];
68   Do[((
69     opMatrix[Reverse[mKey]] = opMatrix[mKey]
70   )),
71   {mKey, matrixKeys}
72 ];
73   juddOperators[{3, opLabel}] = opMatrix,
74   {colIndex, 1, Length[normalTable]}
75 ];

```

```

74 (* special case of t2 in f3 *)
75 (* this is the same as getting the reduced matrix elements from
   Judd 1966 *)
76 numE = 3;
77 e30p = juddOperators[{3, "e_{3}"}];
78 t2prime = juddOperators[{3, "t_{2}^{'}}"];;
79 prefactor = 1/(70 Sqrt[2]);
80 t20p = (# -> (t2prime[#] + prefactor*e30p[#])) & /@ Keys[t2prime];
81 t20p = Association[t20p];
82 juddOperators[{3, "t_{2}"}] = t20p;
83
84 (*Special case of t11 in f3*)
85 t11 = juddOperators[{3, "t_{11}"}];
86 ebetaPrimeOp = juddOperators[{3, "e_{\beta}^{'}}"];;
87 t11primeOp = (# -> (t11[#] + Sqrt[3/385] ebetaPrimeOp[#])) & /@ Keys[
   t11];
88 t11primeOp = Association[t11primeOp];
89 juddOperators[{3, "t_{11}^{'}}"] = t11primeOp;
90 If[OptionValue["Export"],
  (
    (*export them*)
    PrintTemporary["Exporting ..."];
    exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.m"}];
    Export[exportFname, juddOperators];
  )
];
91
92 Return[juddOperators];
93
94
95
96
97
98
99
100
101 ThreeBodyTable::usage="ThreeBodyTable is an association containing
   the LS-reduced matrix elements for the three-body operators for f-
   n configurations. The keys are lists of the form {n, SL, SpLp}.";

```

### 3.9 $\hat{\mathcal{H}}_{\text{cf}}$ : crystal-field

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

$$V(r_i, \theta_i, \phi_i) = \sum_{k=1}^{\infty} \sum_{q=-k}^k \mathcal{A}_q^{(k)} r_i^k \mathcal{C}_q^{(k)}(\theta_i, \phi_i) \quad (52)$$

where the  $\mathcal{C}_q^{(k)}$  are spherical harmonics normalized with the Racah convention

$$\mathcal{C}_q^{(k)} = \sqrt{\frac{4\pi}{2k+1}} Y_q^{(k)}. \quad (53)$$

Here 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  $r_i$  because 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

$$\hat{\mathcal{H}}_{\text{cf}}(\vec{r}) = \sum_{i=1}^n \sum_{k=0}^{\infty} \sum_{q=-k}^k \mathcal{A}_q^{(k)} r_i^k \mathcal{C}_q^{(k)}(\theta_i, \phi_i). \quad (54)$$

And if we average the radial coordinate,

$$\hat{\mathcal{H}}_{\text{cf}} = \sum_{i=1}^n \sum_{k=1}^{\infty} \sum_{q=-k}^k \mathcal{B}_q^{(k)} \mathcal{C}_q^{(k)}(i) \quad (55)$$

where the radial average is included as

$$\mathcal{B}_q^{(k)} := \mathcal{A}_q^{(k)} \langle 4f | r^k | 4f \rangle. \quad (56)$$

$\mathcal{B}_q^{(k)}$  may be complex in general. However, since the sum in [Eqn-54](#) needs to result in a real and Hermitian operator, there are restrictions on  $\mathcal{B}_q^{(k)}$  that need to be accounted for. Once the behavior of  $C_q^{(k)}$  under complex conjugation is considered,  $C_q^{(k)*} = (-1)^q C_{-q}^{(k)}$ , it is necessary that

$$\mathcal{B}_q^{(k)} = (-1)^q \mathcal{B}_{-q}^{(k)*}. \quad (57)$$

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

$$\hat{\mathcal{H}}_{\text{cf}}(\vec{r}) = \sum_{i=1}^n \sum_{k=0}^{\infty} \sum_{q=0}^k B_q^{(k)} \left( C_q^{(k)} + (-1)^q C_{-q}^{(k)} \right) + i S_q^{(k)} \left( C_q^{(k)} - (-1)^q C_{-q}^{(k)} \right). \quad (58)$$

A staple of the Wigner-Racah algebra is writing up operators of 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):

$$\langle \ell \| \hat{u}^{(k)} \| \ell' \rangle = 1. \quad (59)$$

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

$$\hat{U}^{(k)} = \sum_i^n \hat{u}_i^{(k)}. \quad (60)$$

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

$$C_q^{(k)} = \langle \underline{\ell} \| C^{(k)} \| \underline{\ell}' \rangle \hat{u}_q^{(k)} = (-1)^{\underline{\ell}} \sqrt{[\underline{\ell}] [\underline{\ell}']} \begin{pmatrix} \underline{\ell} & k & \underline{\ell}' \\ 0 & 0 & 0 \end{pmatrix} \hat{u}_q^{(k)}. \quad (61)$$

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

$$\overline{\langle \underline{\ell}^n \alpha SLJM_J | \hat{\mathcal{H}}_{\text{cf}} | \underline{\ell}^n \alpha' SL'J'M_{J'} \rangle} = \sum_{k=1}^{\infty} \sum_{q=-k}^k \mathcal{B}_q^{(k)} \langle \underline{\ell}^n \alpha SLJM_J | \hat{U}_q^{(k)} | \underline{\ell}^n \alpha' SL'J'M_{J'} \rangle \langle \underline{\ell} \| \hat{C}^{(k)} \| \underline{\ell} \rangle \quad (62)$$

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

$$\overline{\langle \underline{\ell}^n \alpha SLJM_J | \hat{U}_q^{(k)} | \underline{\ell}^n \alpha' S'L'J'M_{J'} \rangle} = (-1)^{J-M_J} \begin{pmatrix} J & k & J' \\ -M_J & q & M_{J'} \end{pmatrix} \langle \underline{\ell}^n \alpha SLJ \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S'L' \rangle \quad (63)$$

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

$$\overline{\langle \underline{\ell}^n \alpha SLJ \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S'L' \rangle} = (-1)^{S+L+J'+k} \sqrt{[J][J']} \times \begin{Bmatrix} J & J' & k \\ L' & L & S \end{Bmatrix} \langle \underline{\ell}^n \alpha SL \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S'L' \rangle. \quad (64)$$

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

$$\overline{\langle \underline{\ell}^n \alpha SL \| \hat{U}^{(k)} \| \underline{\ell}^n \alpha' S'L' \rangle} = \delta(S, S') n(-1)^{\underline{\ell}+L+k} \sqrt{[L][L']} \times \sum_{\bar{\alpha} \bar{L} \bar{S}} (-1)^{\bar{L}} \begin{Bmatrix} \underline{\ell} & k & \underline{\ell} \\ L & \bar{L} & L' \end{Bmatrix} (\underline{\ell}^n \alpha LS \{ \underline{\ell}^{n-1} \bar{\alpha} \bar{L} \bar{S} \}) (\underline{\ell}^{n-1} \bar{\alpha} \bar{L} \bar{S} \} \underline{\ell}^n \alpha' L'S'). \quad (65)$$

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

$$\hat{\mathcal{H}}_{\text{cf}}(\vec{r}) = \sum_{i=1}^n \sum_{k=2,4,6} \sum_{q=0}^k B_q^{(k)} \left( C_q^{(k)} + (-1)^q C_{-q}^{(k)} \right) + i S_q^{(k)} \left( C_q^{(k)} - (-1)^q C_{-q}^{(k)} \right). \quad (66)$$

The above equations are implemented in `qlanth` by the function `CrystalField`. This function puts together the symbolic sum in [Eqn-62](#) 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 `ReduceUk`, which is used by `GenerateReducedUkTable` to precompute its values.

```

1 Bqk::usage = "Real part of the Bqk coefficients.";
2 Bqk[q_, 2] := {B02/2, B12, B22}[[q + 1]];
3 Bqk[q_, 4] := {B04/2, B14, B24, B34, B44}[[q + 1]];
4 Bqk[q_, 6] := {B06/2, B16, B26, B36, B46, B56, B66}[[q + 1]];

```

```

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

```

```

1 Cqk::usage = "Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_]. In Wybourne
2 (1965) see equations 6-3, 6-4, and 6-5. Also in TASS see equation
3 11.53.";
4 Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
5   {S, Sp, L, Lp, orbital, val},
6   (
7     orbital = 3;
8     {S, L} = FindSL[NKSL];
9     {Sp, Lp} = FindSL[NKSLp];
10    f1 = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
11    val =
12      If[f1 == 0,
13        0,
14        (
15          f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
16          If[f2 == 0,
17            0,
18            (
19              f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
20              If[f3 == 0,
21                0,
22                (
23                  Phaser[J - M + S + Lp + J + k] *
24                  Sqrt[TPO[J, Jp]] *
25                  f1 *
26                  f2 *
27                  f3 *
28                  Ck[orbital, k]
29                )
30              ]
31            )
32          ]
33        );
34      Return[val];
35    )
36  ];
37 ];

```

```

1 CrystalField::usage = "CrystalField[n, NKSL, J, M, NKSLp, Jp, Mp]
2   calculates the matrix element of the crystal field in terms of Bqk
3   and Sqk parameters for configuration f^numE. It is calculated as
4   an association with keys of the form {n, NKSL, J, M, NKSLp, Jp, Mp
5   }.";
6 CrystalField[numE_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := (
7   Sum[
8     (
9       cqk = Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp];
10      cmqk = Cqk[numE, -q, k, NKSL, J, M, NKSLp, Jp, Mp];
11      Bqk[q, k] * (cqk + (-1)^q * cmqk) +
12      I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
13     ),
14     {k, {2, 4, 6}},
15     {q, 0, k}
16   ]
17 )

```

```

1 ReducedUk::usage = "ReducedUk[n, l, SL, SpLp, k] gives the reduced
2   matrix element of the symmetric unit tensor operator U^(k). See
3   equation 11.53 in TASS.";
4 ReducedUk[numE_, l_, SL_, SpLp_, k_] := Module[
5   {spin, orbital, Uk, S, L,
6   Sp, Lp, Sb, Lb, parentSL,
7   cfpSL, cfpSpLp, Ukval,
8   SLparents, SLpparents,
9   commonParents, phase},
10  {spin, orbital} = {1/2, 3};
11  {S, L} = FindSL[SL];
12  {Sp, Lp} = FindSL[SpLp];
13  If[Not[S == Sp],
14    Return[0]
15  ];
16  cfpSL = CFP[{numE, SL}];
17  cfpSpLp = CFP[{numE, SpLp}];
18  SLparents = First /@ Rest[cfpSL];
19  SLpparents = First /@ Rest[cfpSpLp];
20  commonParents = Intersection[SLparents, SLpparents];
21  Uk = Sum[(
22    {Sb, Lb} = FindSL[\[Psi]b];
23    Phaser[Lb] *
24      CFPAssoc[{numE, SL, \[Psi]b}] *
25      CFPAssoc[{numE, SpLp, \[Psi]b}] *
26      SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
27  ),
28  {\[Psi]b, commonParents}
29  ];
30  phase = Phaser[orbital + L + k];
31  prefactor = numE * phase * Sqrt[TPO[L, Lp]];
32  Ukval = prefactor*Uk;
33  Return[Ukval];
34 ]

```

Each of the 32 crystallographic point groups requires only a limited number of non-zero crystal field parameters. In `qlanth` these can be queried programmatically with the use of the function `CrystalFieldForm`. These were taken from a table in Benelli and Gatteschi [BG15] and their corresponding expressions (for a single electron) are in the equations below with a table linking to the corresponding equations. Note that these expressions bring with them an implicit choice for the orientation of the coordinate system (see Section 4).

$S_2$ : Eqn-67	$C_s$ : Eqn-68	$C_{1h}$ : Eqn-69	$C_2$ : Eqn-70	$C_{2h}$ : Eqn-71
$C_{2v}$ : Eqn-72	$D_2$ : Eqn-73	$D_{2h}$ : Eqn-74	$S_4$ : Eqn-75	$C_4$ : Eqn-76
$C_{4h}$ : Eqn-77	$D_{2d}$ : Eqn-78	$C_{4v}$ : Eqn-79	$D_4$ : Eqn-80	$D_{4h}$ : Eqn-81
$C_3$ : Eqn-82	$S_6$ : Eqn-83	$C_{3h}$ : Eqn-84	$C_{3v}$ : Eqn-85	$D_3$ : Eqn-86
$D_{3d}$ : Eqn-87	$D_{3h}$ : Eqn-88	$C_6$ : Eqn-89	$C_{6h}$ : Eqn-90	$C_{6v}$ : Eqn-91
$D_6$ : Eqn-92	$D_{6h}$ : Eqn-93	$\mathcal{T}$ : Eqn-94	$\mathcal{T}_h$ : Eqn-95	$\mathcal{T}_d$ : Eqn-96
$O$ : Eqn-97	$O_h$ : Eqn-98			

Table 1: Expressions for the crystal field in the 32 crystallographic point groups

$$\begin{aligned} \hat{\mathcal{H}}_{\text{cf}}(\mathcal{S}_2) = & B_0^{(2)} \mathcal{C}_0^{(2)} + B_1^{(2)} \mathcal{C}_1^{(2)} + \left(B_2^{(2)} + iS_2^{(2)}\right) \mathcal{C}_2^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} \\ & + \left(B_1^{(4)} + iS_1^{(4)}\right) \mathcal{C}_1^{(4)} + \left(B_2^{(4)} + iS_2^{(4)}\right) \mathcal{C}_2^{(4)} + \left(B_3^{(4)} + iS_3^{(4)}\right) \mathcal{C}_3^{(4)} + \left(B_4^{(4)} + iS_4^{(4)}\right) \mathcal{C}_4^{(4)} \\ & + B_0^{(6)} \mathcal{C}_0^{(6)} + \left(B_1^{(6)} + iS_1^{(6)}\right) \mathcal{C}_1^{(6)} + \left(B_2^{(6)} + iS_2^{(6)}\right) \mathcal{C}_2^{(6)} + \left(B_3^{(6)} + iS_3^{(6)}\right) \mathcal{C}_3^{(6)} \\ & + \left(B_4^{(6)} + iS_4^{(6)}\right) \mathcal{C}_4^{(6)} + \left(B_5^{(6)} + iS_5^{(6)}\right) \mathcal{C}_5^{(6)} + \left(B_6^{(6)} + iS_6^{(6)}\right) \mathcal{C}_6^{(6)} \quad (67) \end{aligned}$$

$$\begin{aligned} \hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_s) = & B_0^{(2)} \mathcal{C}_0^{(2)} + B_2^{(2)} \mathcal{C}_2^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + \left( B_2^{(4)} + iS_2^{(4)} \right) \mathcal{C}_2^{(4)} \\ & + \left( B_4^{(4)} + iS_4^{(4)} \right) \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left( B_2^{(6)} + iS_2^{(6)} \right) \mathcal{C}_2^{(6)} + \left( B_4^{(6)} + iS_4^{(6)} \right) \mathcal{C}_4^{(6)} \\ & + \left( B_6^{(6)} + iS_6^{(6)} \right) \mathcal{C}_6^{(6)} \quad (68) \end{aligned}$$

$$\begin{aligned} \hat{\mathcal{H}}_{\text{cf}}(C_{1h}) = & B_0^{(2)} C_0^{(2)} + B_2^{(2)} C_2^{(2)} + B_0^{(4)} C_0^{(4)} + \left( B_2^{(4)} + iS_2^{(4)} \right) C_2^{(4)} \\ & + \left( B_4^{(4)} + iS_4^{(4)} \right) C_4^{(4)} + B_0^{(6)} C_0^{(6)} + \left( B_2^{(6)} + iS_2^{(6)} \right) C_2^{(6)} + \left( B_4^{(6)} + iS_4^{(6)} \right) C_4^{(6)} \\ & + \left( B_6^{(6)} + iS_6^{(6)} \right) C_6^{(6)} \quad (69) \end{aligned}$$

$$\begin{aligned} \hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_2) = & B_0^{(2)} \mathcal{C}_0^{(2)} + B_2^{(2)} \mathcal{C}_2^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + \left( B_2^{(4)} + iS_2^{(4)} \right) \mathcal{C}_2^{(4)} \\ & + \left( B_4^{(4)} + iS_4^{(4)} \right) \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left( B_2^{(6)} + iS_2^{(6)} \right) \mathcal{C}_2^{(6)} + \left( B_4^{(6)} + iS_4^{(6)} \right) \mathcal{C}_4^{(6)} \\ & + \left( B_6^{(6)} + iS_6^{(6)} \right) \mathcal{C}_6^{(6)} \quad (70) \end{aligned}$$

$$\begin{aligned} \hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_{2h}) = & B_0^{(2)} \mathcal{C}_0^{(2)} + B_2^{(2)} \mathcal{C}_2^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + \left( B_2^{(4)} + iS_2^{(4)} \right) \mathcal{C}_2^{(4)} \\ & + \left( B_4^{(4)} + iS_4^{(4)} \right) \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left( B_2^{(6)} + iS_2^{(6)} \right) \mathcal{C}_2^{(6)} + \left( B_4^{(6)} + iS_4^{(6)} \right) \mathcal{C}_4^{(6)} \\ & + \left( B_6^{(6)} + iS_6^{(6)} \right) \mathcal{C}_6^{(6)} \quad (71) \end{aligned}$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_{2v}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_2^{(2)} \mathcal{C}_2^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_2^{(4)} \mathcal{C}_2^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_2^{(6)} \mathcal{C}_2^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (72)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_2) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_2^{(2)} \mathcal{C}_2^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_2^{(4)} \mathcal{C}_2^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} \\ + B_0^{(6)} \mathcal{C}_0^{(6)} + B_2^{(6)} \mathcal{C}_2^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (73)$$

$$\begin{aligned} \hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_{2h}) = & B_0^{(2)} C_0^{(2)} + B_2^{(2)} C_2^{(2)} + B_0^{(4)} C_0^{(4)} + B_2^{(4)} C_2^{(4)} + B_4^{(4)} C_4^{(4)} \\ & + B_0^{(6)} C_0^{(6)} + B_2^{(6)} C_2^{(6)} + B_4^{(6)} C_4^{(6)} + B_6^{(6)} C_6^{(6)} \quad (74) \end{aligned}$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{S}_4) = B_0^{(2)} C_0^{(2)} + B_0^{(4)} C_0^{(4)} + B_4^{(4)} C_4^{(4)} + B_0^{(6)} C_0^{(6)} + \left(B_4^{(6)} + i S_4^{(6)}\right) C_4^{(6)} \quad (75)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_4) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left(B_4^{(6)} + iS_4^{(6)}\right) \mathcal{C}_4^{(6)} \quad (76)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_{4h}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left(B_4^{(6)} + iS_4^{(6)}\right) \mathcal{C}_4^{(6)} \quad (77)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_{2d}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (78)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_{4v}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (79)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_4) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (80)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_{4h}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (81)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_3) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_3^{(4)} \mathcal{C}_3^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left( B_3^{(6)} + iS_3^{(6)} \right) \mathcal{C}_3^{(6)} + \left( B_6^{(6)} + iS_6^{(6)} \right) \mathcal{C}_6^{(6)} \quad (82)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{S}_6) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_3^{(4)} \mathcal{C}_3^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left( B_3^{(6)} + i S_3^{(6)} \right) \mathcal{C}_3^{(6)} + \left( B_6^{(6)} + i S_6^{(6)} \right) \mathcal{C}_6^{(6)} \quad (83)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_{3h}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (84)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_{3v}) = B_0^{(2)}\mathcal{C}_0^{(2)} + B_0^{(4)}\mathcal{C}_0^{(4)} + B_3^{(4)}\mathcal{C}_3^{(4)} + B_0^{(6)}\mathcal{C}_0^{(6)} + B_3^{(6)}\mathcal{C}_3^{(6)} + B_6^{(6)}\mathcal{C}_6^{(6)} \quad (85)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_3) = B_0^{(2)}C_0^{(2)} + B_0^{(4)}C_0^{(4)} + B_3^{(4)}C_3^{(4)} + B_0^{(6)}C_0^{(6)} + B_3^{(6)}C_3^{(6)} + B_6^{(6)}C_6^{(6)} \quad (86)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_{3d}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_3^{(4)} \mathcal{C}_3^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_3^{(6)} \mathcal{C}_3^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (87)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_{3h}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (88)$$

$$\mathcal{H}_{\text{cf}}(\mathcal{C}_6) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (89)$$

$$\hat{\mathcal{H}}_{\text{cf}}(C_{6h}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (90)$$

$$\mathcal{H}_{\text{cf}}(\mathcal{C}_{6v}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (91)$$

$$\mathcal{H}_{\text{cf}}(\mathcal{D}_6) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (92)$$

$$\mathcal{H}_{\text{cf}}(\mathcal{D}_{6h}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (93)$$

$$\mathcal{H}_{\text{cf}}(\mathcal{I}) = B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (94)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{I}_h) = B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (95)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{I}_d) = B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (96)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{O}) = B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (97)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{O}_h) = B_0^{(4)}C_0^{(4)} + B_4^{(4)}C_4^{(4)} + B_0^{(6)}C_0^{(6)} + B_4^{(6)}C_4^{(6)} \quad (98)$$

\_\_\_\_ (END) Crystal field expressions (END) \_\_\_\_\_

(END) Crystal field expressions (END)

```

1 CrystalFieldForm::usage = "CrystalFieldForm[symmetryGroup] returns an
   association that describes the crystal field parameters that are
   necessary to describe a crystal field for the given symmetry group
   .
2
3 The symmetry group must be given as a string in Schoenflies notation
   and must be one of C1, Ci, S2, Cs, C1h, C2, C2h, C2v, D2, D2h, S4,
   C4, C4h, D2d, C4v, D4, D4h, C3, S6, C3h, C3v, D3, D3d, D3h, C6,
   C6h, C6v, D6, D6h, T, Th, Td, O, Oh.
4
5 The returned association has three keys:
6   ''BqkSqk'' whose values is a list with the nonzero Bqk and Sqk
   parameters;
7   ''constraints'' whose value is either an empty list, or a lists of
   replacements rules that are constraints on the Bqk and Sqk
   parameters;
8   ''simplifier'' whose value is an association that can be used to
   set to zero the crystal field parameters that are zero for the
   given symmetry group;
9   ''aliases'' whose value is a list with the integer by which the
   point group is also known for and an alternate Schoenflies symbol
   if it exists.
10
11 This uses data from table 3.3 in Benelli and Gatteschi, 2015.";
12 CrystalFieldForm[symmetryGroupString_] := (
13   If[Not@ValueQ[crystalFieldFunctionalForms],
14     crystalFieldFunctionalForms = Import[FileNameJoin[{moduleDir, "data", "crystalFieldFunctionalForms.m"}]];
15   ];
16   cfForm = crystalFieldFunctionalForms[symmetryGroupString];
17   simplifier = Association[(# -> 0) &/@ Complement[cfSymbols, cfForm["BqkSqk"]]];
18   Return[Join[cfForm, <|"simplifier" -> simplifier|>]];
19 )

```

### 3.10 $\hat{\mu}$ and $\hat{\mathcal{H}}_z$ : the magnetic dipole operator and the Zeeman term

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

$$\hat{\mu} = -\mu_B \left( \hat{L} + g_s \hat{S} \right)^{(1)}, \text{ with } \mu_B = 1/2. \quad (99)$$

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

In the  $|LSJM\rangle$  basis that we use in `qlanth` the LSJ reduced-matrix elements are computed using equation 15.7 in [Cow81]

$$\langle \alpha LSJ \| \left( \hat{L} + g_s \hat{S} \right)^{(1)} \| \alpha' L' S' J' \rangle = \delta(\alpha LSJ, \alpha' L' S' J') \sqrt{J(J+1)(2J+1)} + \\ \delta(\alpha LS, \alpha' L' S') (-1)^{L+S+J+1} \sqrt{[J][J]} \begin{Bmatrix} L & S & J \\ 1 & J' & S \end{Bmatrix}. \quad (100)$$

Then these reduced matrix elements are used to resolve the  $M_J$  components for  $q = -1, 0, 1$  through Wigner-Eckart:

$$\langle \alpha LSJM_J | \left( \hat{L} + g_s \hat{S} \right)_q^{(1)} | \alpha' L' S' J' M_{J'} \rangle = \\ (-1)^{J-M_J} \begin{pmatrix} J & 1 & J' \\ -M_J & q & M'_J \end{pmatrix} \langle \alpha LSJ \| \left( \hat{L} + g_s \hat{S} \right)^{(1)} \| \alpha' L' S' J' \rangle. \quad (101)$$

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

```

1 JJBlockMagDip::usage = "JJBlockMagDip[numE, J, Jp] returns an array
   for the LSJM matrix elements of the magnetic dipole operator
   between 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.
2 See eqn 15.7 in TASS.
3 Here it is provided in atomic units in which the Bohr magneton is
   1/2.

```

```

4 \[Mu] = -(1/2) (L + gs S)
5 We are using the Racah convention for the reduced matrix elements in
   the Wigner-Eckart theorem. See TASS eqn 11.15.
6 ";
7 Options[JJBlockMagDip]= {"Sparse" -> True};
8 JJBlockMagDip[numE_, braJ_, ketJ_, OptionsPattern[]] := Module[
9   {braSLJs, ketSLJs,
10  braSLJ, ketSLJ,
11  braSL, ketSL,
12  braS, braL,
13  ketS, ketL,
14  braMJ, ketMJ,
15  matValue, magMatrix,
16  summand1, summand2,
17  threejays},
18 (
19   braSLJs = AllowedNKSLJMforJTerms[numE, braJ];
20   ketSLJs = AllowedNKSLJMforJTerms[numE, ketJ];
21   magMatrix = Table[
22     braSL = braSLJ[[1]];
23     ketSL = ketSLJ[[1]];
24     {braS, braL} = FindSL[braSL];
25     {ketS, ketL} = FindSL[ketSL];
26     braMJ = braSLJ[[3]];
27     ketMJ = ketSLJ[[3]];
28     summand1 = If[Or[braJ != ketJ,
29                       braSL != ketSL],
30                   0,
31                   Sqrt[braJ*(braJ+1)*TPO[braJ]]
32                 ];
33     (* looking at the string includes checking L=L', S=S', and \
alpha=alpha *)
34     summand2 = If[braSL != ketSL,
35                   0,
36                   (gs-1) *
37                   Phaser[braS+braL+ketJ+1] *
38                   Sqrt[TPO[braJ]*TPO[ketJ]] *
39                   SixJay[{braJ, 1, ketJ}, {braS, braL, braS}] *
40                   Sqrt[braS(braS+1)TPO[braS]]
41                 ];
42     matValue = summand1 + summand2;
43     (* We are using the Racah convention for red matrix elements in
Wigner-Eckart *)
44     threejays = (ThreeJay[{braJ, -braMJ}, {1, #}, {ketJ, ketMJ}] &
45 /@ {-1, 0, 1};
46     threejays *= Phaser[braJ-braMJ];
47     matValue = -1/2 * threejays * matValue;
48     matValue,
49     {braSLJ, braSLJs},
50     {ketSLJ, ketSLJs}
51   ];
52   If[OptionValue["Sparse"],
53     magMatrix = SparseArray[magMatrix]
54   ];
55   Return[magMatrix];
56 )
56 ];

```

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

$$\hat{\mu}_x = \frac{\hat{\mu}_{-1}^{(1)} - \hat{\mu}_{+1}^{(1)}}{\sqrt{2}}, \quad (102)$$

$$\hat{\mu}_y = i \frac{\hat{\mu}_{-1}^{(1)} + \hat{\mu}_{+1}^{(1)}}{\sqrt{2}}, \quad (103)$$

$$\hat{\mu}_z = \hat{\mu}_0^{(1)}. \quad (104)$$

```

1 MagDipoleMatrixAssembly::usage = "MagDipoleMatrixAssembly[numE]
   returns the matrix representation of the operator - 1/2 (L + gs S)
   in the f^numE configuration. The function returns a list with
   three elements corresponding to the x,y,z components of this
   operator. The option ''FilenameAppendix'' can be used to append a

```

```

    string to the filename from which the function imports from in
    order to patch together the array. For numE beyond 7 the function
    returns the same as for the complementary configuration. The
    option ''ReturnInBlocks'' can be used to return the matrices in
    blocks. The default is to return the matrices in flattened form
    and as sparse array.";
2 Options[MagDipoleMatrixAssembly] ={
3   "FilenameAppendix" -> "",
4   "ReturnInBlocks" -> False};
5 MagDipoleMatrixAssembly[nf_Integer, OptionsPattern[]] := Module[
6   {ImportFun, numE, appendTo,
7   emFname, JJBlockMagDipTable,
8   Js, howManyJs, blockOp,
9   rowIdx, colIdx},
10  (
11    ImportFun = ImportMZip;
12    numE = nf;
13    numH = 14 - numE;
14    numE = Min[numE, numH];
15
16    appendTo = (OptionValue["FilenameAppendix"] <> "-magDip");
17    emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
18      appendTo];
19    JJBlockMagDipTable = ImportFun[emFname];
20
21    Js = AllowedJ[numE];
22    howManyJs = Length[Js];
23    blockOp = ConstantArray[0, {howManyJs, howManyJs}];
24    Do[
25      blockOp[[rowIdx, colIdx]] = JJBlockMagDipTable[{numE, Js[[rowIdx
26      ]], Js[[colIdx]]}],
27      {rowIdx, 1, howManyJs},
28      {colIdx, 1, howManyJs}
29    ];
29
30    If[OptionValue["ReturnInBlocks"],
31      (
32        opMinus = Map[#[[1]] &, blockOp, {4}];
33        opZero = Map[#[[2]] &, blockOp, {4}];
34        opPlus = Map[#[[3]] &, blockOp, {4}];
35        opX = (opMinus - opPlus)/Sqrt[2];
36        opY = I (opPlus + opMinus)/Sqrt[2];
37        opZ = opZero;
38      ),
39      blockOp = ArrayFlatten[blockOp];
40      opMinus = blockOp[[;, , ;, 1]];
41      opZero = blockOp[[;, , ;, 2]];
42      opPlus = blockOp[[;, , ;, 3]];
43      opX = (opMinus - opPlus)/Sqrt[2];
44      opY = I (opPlus + opMinus)/Sqrt[2];
45      opZ = opZero;
46    ];
47    Return[{opX, opY, opZ}];
48  )
49];

```

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 `EffectiveHamiltonian`. 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 energy pseudo-unit Kayser ( $\text{cm}^{-1}$ ). The conversion factor is called `teslaToKayser` in the file `qonstants.m`.

### 3.11 Alternative operator bases

One feature from the operators used in `Eqn-1` is that when data is fit to this model, the parameters are correlated. This has the consequence that using a partial set of operators (say those describing the free-ion part) results in parameter values that change noticeably (by perhaps 10%) when additional interactions are brought into the analysis. The semi-empirical Hamiltonian as written in `Eqn-1` can be described as a linear combination of a basis set of operators. Correlations in fitted parameters may be removed by using a different operator basis, with this having the added benefit of reducing parameters uncertainties [New82]. This removal of correlations is achieved by making the basis operators

pair-wise orthogonal between themselves.<sup>18</sup>

The operators  $\hat{f}_k$ ,  $\hat{L}^2$ ,  $\hat{\mathcal{C}}(\mathcal{G}_2)$ ,  $\hat{\mathcal{C}}(\mathcal{SO}(7))$ , and  $\hat{\mathbf{t}}_2$  can be made orthogonal among themselves as prescribed by Judd, Crosswhite, and Suskin [JC84; JS84]. In there the change in the operator basis has an accompanying relationship between the coefficients in the old and the new bases, as given below. (Note the  $n$  dependence on the equation for  $E_{\perp}^{(3)}$ .)

$$\begin{aligned} E_{\perp}^{(1)} &= \frac{4\alpha}{5} + \frac{\beta}{30} + \frac{\gamma}{25} + \frac{14F^{(2)}}{405} \\ &\quad + \frac{7F^{(4)}}{297} + \frac{350F^{(6)}}{11583} \end{aligned} \quad (105)$$

$$E_{\perp}^{(2)} = \frac{F^{(2)}}{2025} - \frac{F^{(4)}}{3267} + \frac{175F^{(6)}}{1656369} \quad (106)$$

$$\begin{aligned} E_{\perp}^{(3)} &= -\frac{2\alpha}{5} + \frac{F^{(2)}}{135} + \frac{2F^{(4)}}{1089} \\ &\quad - \frac{175F^{(6)}}{42471} + \frac{nT^{(2)}}{70\sqrt{2}} - \frac{T^{(2)}}{35\sqrt{2}} \end{aligned} \quad (107)$$

$$\alpha_{\perp} = \frac{4\alpha}{5} \quad (108)$$

$$\beta_{\perp} = -4\alpha - \frac{\beta}{6} \quad (109)$$

$$\gamma_{\perp} = \frac{8\alpha}{5} + \frac{\beta}{15} + \frac{2\gamma}{25} \quad (110)$$

$$T_{\perp}^{(2)} = T^{(2)} \quad (111)$$

(In general, if a new operator basis  $\hat{O}'$  is defined by a linear transformation  $m$  of the original basis  $\hat{O}$  such that  $\hat{O}' = m\hat{O}$ , then for a given linear combination of the original operator basis,  $\mathcal{H} = \vec{\eta} \cdot \hat{O}$ , a new linear combination  $\vec{\eta}'$  of the new operator basis  $\hat{O}'$ , can be defined such that  $\vec{\eta} \cdot \hat{O} = \vec{\eta}' \cdot \hat{O}'$  by taking  $\vec{\eta}' = (m^{-1})^T \vec{\eta}$ .)

Using these coefficients and their accompanying operators, the semi-empirical Hamiltonian is now

$$\begin{aligned} \hat{\mathcal{H}}_{\text{eff}}^{\perp} &= \hat{\mathcal{H}}_0 + \sum_{k=0,2,4,6} E_{\perp}^{(k)} \hat{e}_k^{\perp} + \zeta \sum_{i=1}^N (\hat{s}_i \cdot \hat{l}_i) \\ &\quad + \alpha_{\perp} \hat{\alpha}^{\perp} + \beta_{\perp} \hat{\beta}^{\perp} + \gamma_{\perp} \hat{\gamma}^{\perp} \\ &\quad + T_{\perp}^{(2)} \hat{\mathbf{t}}_2^{\perp} + \sum_{\substack{k=2,3,4,6,7,8, \\ 11,12,14,15, \\ 16,17,18,19}} T^{(k)} \hat{\mathbf{t}}_k \\ &\quad + \sum_{k=2,4,6} P^{(k)} \hat{\mathbf{p}}_k + \sum_{k=0,2,4} m^{(k)} \hat{\mathbf{m}}_k \\ &\quad + \sum_{i=1}^N \sum_{k=2,4,6} \sum_{q=-k}^k \mathcal{B}_q^{(k)} \mathcal{C}_q^{(k)}(i). \end{aligned} \quad (112)$$

Some operators remain unchanged in this parametrization, specifically  $\hat{\mathcal{C}}_q^k$ ,  $\hat{\mathbf{t}}_k$  ( $k \neq 2$ ), and the spin-orbit term. However some operators are still non-orthogonal. Operators from different *families* are mutually orthogonal, and all pairs within each family are orthogonal as well. Nevertheless, any two operators from either  $\hat{\mathbf{m}}_k$  or  $\hat{\mathbf{p}}_k$  are non-orthogonal. This residual non-orthogonality in the Hamiltonian can be resolved using the  $\hat{z}_i$  operators, originally introduced by Judd in his discussion of intra-atomic magnetic interactions [JCC68]. This parametrization, which leaves  $\hat{\mathbf{m}}_k$  or  $\hat{\mathbf{p}}_k$  as they are, is therefore not entirely orthogonal, and we refer to it as the *mostly*-orthogonal Hamiltonian.

In `qlanth` the symbolic array that represents the *mostly*-orthogonal Hamiltonian can be obtained with the adequate setting of the option `OperatorBasis` in `EffectiveHamiltonian`. In that option, the standard operator basis (i.e. the one use by Carnall *et al.* [Car+89]) is termed the *legacy* basis.

```
1 EffectiveHamiltonian::usage = "EffectiveHamiltonian[numE] returns the
   Hamiltonian matrix for the f^numE configuration. The matrix is
   returned as a SparseArray.
2 The function admits an optional parameter ''FilenameAppendix'', which
   can be used to control which variant of the JJBlocks is used to
   assemble the matrix."
```

<sup>18</sup> Two operators  $\hat{\mathcal{O}}_1$  and  $\hat{\mathcal{O}}_2$  are orthogonal if  $\text{tr}(\hat{\mathcal{O}}_1^\dagger \hat{\mathcal{O}}_2) = 0$ .

```

3 It also admits an optional parameter ''IncludeZeeman'', which can be
4 used to include the Zeeman interaction. The default is False.
5 The option ''Set t2Switch'' can be used to toggle on or off setting
6 the t2 selector automatically or not, the default is True, which
7 replaces the parameter according to numE.
8 The option ''ReturnInBlocks'' can be used to return the matrix in
9 block or flattened form. The default is to return it in flattened
10 form.";
11 Options[EffectiveHamiltonian] = {
12     "FilenameAppendix" -> "",
13     "IncludeZeeman" -> False,
14     "Set t2Switch" -> True,
15     "ReturnInBlocks" -> False,
16     "OperatorBasis" -> "Legacy"};
17 EffectiveHamiltonian[nf_, OptionsPattern[]] := Module[
18 {numE, ii, jj, howManyJs, Js, blockHam, opBasis},
19 (
20 (*#####
21 ImportFun = ImportMZip;
22 opBasis = OptionValue["OperatorBasis"];
23 If[Not[MemberQ>{"Legacy", "MostlyOrthogonal", "Orthogonal"}, opBasis]],
24     Echo["Operator basis " <> opBasis <> " not recognized, using 'Legacy' basis."];
25     opBasis = "Legacy";
26 ];
27 If[opBasis == "Orthogonal",
28     Echo["Operator basis 'Orthogonal' not implemented yet, aborting ..."];
29     Return[Null];
30 ];
31 (*#####
32 If[opBasis == "MostlyOrthogonal",
33 (
34     blockHam = EffectiveHamiltonian[nf,
35         "OperatorBasis" -> "Legacy",
36         "FilenameAppendix" -> OptionValue["FilenameAppendix"],
37         "IncludeZeeman" -> OptionValue["IncludeZeeman"],
38         "Set t2Switch" -> OptionValue["Set t2Switch"],
39         "ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
40     paramChanger = Which[
41         nf < 7,
42             (
43                 F0 -> 1/91 (54 E1p+91 E0p+78 γp),
44                 F2 -> (15/392 *
45                     (
46                         140 E1p +
47                         20020 E2p +
48                         1540 E3p +
49                         770 αp -
50                         70 γp +
51                         22 Sqrt[2] T2p -
52                         11 Sqrt[2] nf T2p t2Switch -
53                         11 Sqrt[2] (14-nf) T2p (1 - t2Switch)
54                     )
55             ),
56             F4 -> (99/490 *
57                 (
58                     70 E1p -
59                     9100 E2p +
60                     280 E3p +
61                     140 αp -
62                     35 γp +
63                     4 Sqrt[2] T2p -
64                     2 Sqrt[2] nf T2p t2Switch -
65                     2 Sqrt[2] (14-nf) T2p (1-t2Switch)
66                 )
67             ),
68             F6 -> (5577/7000 *
69                 (
70                     20 E1p +
71                     700 E2p -
72                     140 E3p -
73                     70 αp -
74                     10 γp -
75                     2 Sqrt[2] T2p +
76                 )
77             )
78         )
79     ];
80     blockHam /. paramChanger
81 ];
82 
```

```

71          Sqrt[2] nf T2p t2Switch +
72          Sqrt[2] (14-nf) T2p (1-t2Switch)
73      )
74      ),
75       $\zeta \rightarrow \zeta$ ,
76       $\alpha \rightarrow (5 \alpha p)/4$ ,
77       $\beta \rightarrow -6 (5 \alpha p + \beta p)$ ,
78       $\gamma \rightarrow 5/2 (2 \beta p + 5 \gamma p)$ ,
79      T2  $\rightarrow 0$ 
80  | >,
81  nf  $\geq 7$ ,
82  <|
83      F0  $\rightarrow 1/91 (54 E1p + 91 E0p + 78 \gamma p)$ ,
84      F2  $\rightarrow (15/392 *$ 
85      (
86          140 E1p +
87          20020 E2p +
88          1540 E3p +
89          770  $\alpha p$  -
90          70  $\gamma p$  +
91          22 Sqrt[2] T2p -
92          11 Sqrt[2] nf T2p
93      )
94  ),
95  F4  $\rightarrow (99/490 *$ 
96  (
97      70 E1p -
98      9100 E2p +
99      280 E3p +
100     140  $\alpha p$  -
101     35  $\gamma p$  +
102     4 Sqrt[2] T2p -
103     2 Sqrt[2] nf T2p
104  )
105  ),
106  F6  $\rightarrow (5577/7000 *$ 
107  (
108      20 E1p +
109      700 E2p -
110      140 E3p -
111      70  $\alpha p$  -
112      10  $\gamma p$  -
113      2 Sqrt[2] T2p +
114      Sqrt[2] nf T2p
115  )
116  ),
117   $\zeta \rightarrow \zeta$ ,
118   $\alpha \rightarrow (5 \alpha p)/4$ ,
119   $\beta \rightarrow -6 (5 \alpha p + \beta p)$ ,
120   $\gamma \rightarrow 5/2 (2 \beta p + 5 \gamma p)$ ,
121  T2  $\rightarrow 0$ 
122  | >
123 ];
124 blockHamM0 = Which[
125   OptionValue["ReturnInBlocks"] == False,
126   ReplaceInSparseArray[blockHam, paramChanger],
127   OptionValue["ReturnInBlocks"] == True,
128   Map[ReplaceInSparseArray[#, paramChanger]&, blockHam, {2}]
129 ];
130 Return[blockHamM0];
131 )
132 ];
133 (*#####
134 (*hole-particle equivalence enforcement*)
135 numE = nf;
136 allVars = {E0, E1, E2, E3,  $\zeta$ , F0, F2, F4, F6, M0, M2, M4, T2, T2p
137 ,
138   T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
139    $\alpha$ ,  $\beta$ , B02, B04, B06, B12, B14, B16,
140   B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16, S22
141 ,
142   S24, S26, S34, S36, S44, S46, S56, S66, T11p, T12, T14, T15,
143   T16,
144   T17, T18, T19, Bx, By, Bz};
145 params0 = AssociationThread[allVars, allVars];

```

```

144 If [nf > 7,
145 (
146     numE = 14 - nf;
147     params = HoleElectronConjugation[params0];
148     If[OptionValue["Set t2Switch"], params[t2Switch] = 0];
149 ),
150 params = params0;
151 If[OptionValue["Set t2Switch"], params[t2Switch] = 1];
152 ];
153 (* Load symbolic expressions for LS,J,J' energy sub-matrices. *)
154 emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
OptionValue["FilenameAppendix"]];
155 JJBlockMatrixTable = ImportFun[emFname];
156 (*Patch together the entire matrix representation using J,J'
blocks.*)
157 PrintTemporary["Patching JJ blocks ..."];
158 Js = AllowedJ[numE];
159 howManyJs = Length[Js];
160 blockHam = ConstantArray[0, {howManyJs, howManyJs}];
161 Do[
162     blockHam[[jj, ii]] = JJBlockMatrixTable[{numE, Js[[ii]], Js[[jj]]}];,
163 {ii, 1, howManyJs},
164 {jj, 1, howManyJs}
165 ];
166 (* Once the block form is created flatten it *)
167 If[Not[OptionValue["ReturnInBlocks"]],
168 (blockHam = ArrayFlatten[blockHam];
169 blockHam = ReplaceInSparseArray[blockHam, params];
170 ),
171 (blockHam = Map[ReplaceInSparseArray[#, params]&, blockHam
,{2}])
172 ];
173
174 If[OptionValue["IncludeZeeman"],
175 (
176     PrintTemporary["Including Zeeman terms ..."];
177     {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "
ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
178     blockHam += - teslaToKayser * (Bx * magx + By * magy + Bz *
magz);
179 )
180 ];
181 Return[blockHam];
182 )
183 ];

```

## 4 Coordinate system

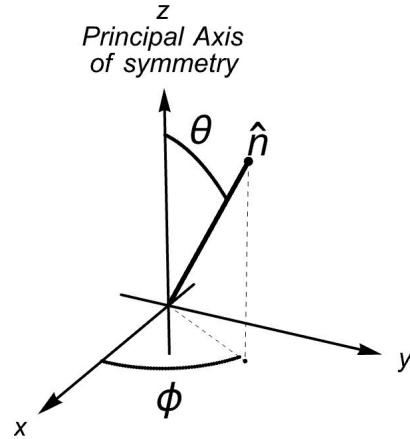
Before adding interactions lacking spherical symmetry, the orientation of the coordinate system is irrelevant. At the point when the crystal field is added, this orientation becomes relevant in the sense that only certain orientations of the coordinate system yield an expression for the crystal field potential in its simplest form<sup>19</sup>. To accomplish this the z-axis needs to be taken as one of the principal axis of symmetry<sup>20</sup>. To complete the orientation of the coordinate system, the x and y axes are chosen as symmetry axes perpendicular to the z-axis. Furthermore, certain choices for the orientation of the coordinate system also allow one to make certain crystal field parameters real, or to fix their sign.

## 5 Spectroscopic measurements and uncertainty

We may categorize the uncertainty in the parameters fitted to experimental data in three categories: experimental, model, and others.

Before listing the sources that contribute to experimental error, let's briefly recount the types of experiments that are used in order to determine level energies and state labels. The first type is absorption spectroscopy, in which a crystal, adequately doped, is illuminated with a broad spectrum light source, and the wavelength dependent absorption

<sup>19</sup> Of course, the crystal field potential can be expressed in any rotated coordinate system, but in these the potential would include additional  $C_q^{(k)}$  with linear combinations of the  $B_q^{(k)}$ <sup>20</sup> A principal axis is a symmetry axis having the most rotational symmetry in the relevant symmetry group. For example, in cubic groups, the principal axis is the 111 diagonal.



by the crystal thus determined. The crystals absorb radiation depending on the availability of a transition energy between the thermally populated low-lying states and excited levels. This data therefore provides transition energies between the ground multiplet and excited states. Furthermore, from this data one can also estimate the probability that a photon of a given wavelength is absorbed by the ions in the crystal, and from this one estimates the oscillator strength of a given transition.

In order to inform to what two multiplets the transitions belong to, here one may already count how many lines arrange themselves in groups. Alas, this type of absorption spectroscopy is lacking in that it only provides information about transitions between the ground and excited states, and may even elide such transitions that are too weak to be observed. In view of this, some variety of emission spectroscopy becomes relevant. In these, the ions inside of the crystal are excited (thermally, electrically, or radiatively) and the light produced by the relaxing transitions are then registered. This has the benefit that one has now populated other states than the ground state (perhaps with aid of non-radiative transitions inside of the crystal or energy transfer) so that now one can also have information about transitions that depart from a state different than ground. From this type of spectroscopy, given a transition, one may also determine its transition rate, given the availability of time-resolved emission; given this one may then give an upper bound on the spontaneous rate of identified transitions.

In these analyses a few things may not go according to plan:

1. **Several non-equivalent symmetry sites.** Ions may not be located in sites with the same crystal symmetry. As such it will be problematic to interpret their crystal splittings based on the assumption of a single symmetry.
2. **Non-homogeneous crystal field.** Even if they are located in sites with the same point symmetry, it may also be that the crystal field they experience has variations across the bulk of the crystal. As such, the observations would then rather be about an ensemble of crystal fields, instead of a single one.
3. **Crystal impurities.** The doped crystals may contain impurities that will lead to the false identification of transitions to the ion of study. This may be disambiguated from pooling together several experiments.
4. **Non-radiative transitions**, mediated by the crystal, will lead to shifts in transition energies, both in emission, and in absorption. This yields a confounding factor for the *radiative* transition energies that are in principle required to be valid inputs to the model Hamiltonian. Comparison of emission and absorption lines is key to determine the relevance of this.
5. **Spectrometer resolution.** The spectrometers used have a finite resolution. In the setups typically used for this, the nominal resolution might be of the order of 0.1 nm.
6. **Crystal transparency.** Observation of transitions within the ions requires that the crystal be mostly transparent at the relevant wavelengths.

In the works of Carnall and others, the nominal uncertainty in the state energies is of  $1\mathcal{K}$ , this being the precision to which the used experimental energies are quoted.

With regards to model uncertainties, the following factors may be considered to contribute to it:

- Intra-configuration transitions.** When energy levels reach a certain threshold, observations may no longer be intra-configuration transitions, but rather inter-configuration transitions. These transitions should not be included, so care must be taken to exclude them from the analysis.
- Unaccounted configuration interaction.** The model makes an attempt at describing configuration interaction effects, but this is only carried to second order in the types of considered interactions, and not all interactions are considered.

Finally, in the “others” category we have the two following:

- Numerical precision.** No longer relevant with modern computers, however, at the time at which some of these calculations were done, numerical precision might account for some of the discrepancies one finds when comparing current calculations to old ones.
- Errors in tables with reduced matrix elements.** The Crosswhite group at Argonne National Lab produced a set of tables with the reduced matrix elements of operators. However, at some point, these tables became slightly corrupted, and subsequent codes that used them carried those errors with them. In `qlanth` this problem is avoided since all reduced matrix elements are calculated from scratch.

When the model parameters are fitted to experimental data and their uncertainties are being estimated, `qlanth` offers two approaches. In the first approach a given constant uncertainty in the energy levels is assumed, this in turns determines the relevant contour of  $\chi^2$ , and from this the uncertainties in the model parameters are calculated.

In an alternative approach, the uncertainties are determined *a posteriori*. The model parameters are fit to minimize the square differences between calculated energies and the experimental ones. Then, a single experimental uncertainty is assumed in all the energy levels, and taken equal to the minimum root mean square error, as taken over the available degrees of freedom. This uncertainty  $\sigma$  together with a chosen confidence interval  $p$  is then used to determine the contours of  $\chi^2$ , which in turn determine the corresponding confidence interval in the model parameters. In a sense, the model is assumed to be valid, and the resulting uncertainties in the model parameters are adjusted to allow for this possibility.

In the examples included in this code the uncertainty in the experimental data was assumed to be constant and equal to  $1\text{cm}^{-1}$ . And when the data for magnetic dipole transitions was calculated, an uncertainty equal to the  $\sigma$  of the related parametric fit was assumed.

## 6 Transitions

`qlanth` can also compute magnetic dipole transition rates within states and levels, as well as forced electric dipole transition rates between levels.

### 6.1 State description

#### 6.1.1 Magnetic dipole transitions

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

$$\hat{S}(\psi, \psi') := |\langle \psi | \hat{\mu} | \psi' \rangle|^2 = |\langle \psi | \hat{\mu}_x | \psi' \rangle|^2 + |\langle \psi | \hat{\mu}_y | \psi' \rangle|^2 + |\langle \psi | \hat{\mu}_z | \psi' \rangle|^2 \quad (113)$$

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

```

1 MagDipLineStrength::usage = "MagDipLineStrength[theEigensys, numE]
    takes the eigensystem of an ion and the number numE of f-electrons
    that correspond to it and calculates the line strength array Stot"
.
2 The option ''Units'' can be set to either ''SI'' (so that the units
    of the returned array are (A m^2)^2) or to ''Hartree''.

```

```

3 The option ''States'' can be used to limit the states for which the
4 line strength is calculated. The default, All, calculates the line
5 strength for all states. A second option for this is to provide
6 an index labelling a specific state, in which case only the line
7 strengths between that state and all the others are computed.
8 The returned array should be interpreted in the eigenbasis of the
9 Hamiltonian. As such the element Stot[[i,i]] corresponds to the
10 line strength states between states  $|i\rangle$  and  $|j\rangle$ .";
11 Options[MagDipLineStrength]={"Reload MagOp" -> False, "Units" -> "SI",
12 "States" -> All};
13 MagDipLineStrength[theEigensys_List, numE0_Integer, OptionsPattern[]]
14 := Module[
15   {numE, allEigenvecs, Sx, Sy, Sz, Stot, factor},
16   (
17     numE = Min[14 - numE0, numE0];
18     (*If not loaded then load it, *)
19     If[Or[
20       Not[MemberQ[Keys[magOp], numE]],
21       OptionValue["Reload MagOp"]],
22       (
23         magOp[numE] = ReplaceInSparseArray[#, {gs -> 2}] & /@*
24         MagDipoleMatrixAssembly[numE];
25       )
26     ];
27     allEigenvecs = Transpose[Last /@ theEigensys];
28     Which[OptionValue["States"] === All,
29       (
30         {Sx, Sy, Sz} = (ConjugateTranspose[allEigenvecs].#.
31         allEigenvecs) & /@ magOp[numE];
32         Stot = Abs[Sx]^2 + Abs[Sy]^2 + Abs[Sz]^2;
33       ),
34       IntegerQ[OptionValue["States"]],
35       (
36         singleState = theEigensys[[OptionValue["States"], 2]];
37         {Sx, Sy, Sz} = (ConjugateTranspose[allEigenvecs].#.
38         singleState) & /@ magOp[numE];
39         Stot = Abs[Sx]^2 + Abs[Sy]^2 + Abs[Sz]^2;
40       )
41     ];
42   ];
43   Which[
44     OptionValue["Units"] == "SI",
45     Return[4 \[Mu]B^2 * Stot],
46     OptionValue["Units"] == "Hartree",
47     Return[Stot],
48     True,
49     (
50       Echo["Invalid option for ''Units''. Options are ''SI'' and ''"
51       Hartree'']."];
52       Abort[];
53     )
54   ];
55 ]
56 );
57
58 ];

```

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

$$A_{MD}(|\psi_i\rangle \rightsquigarrow |\psi_f\rangle) = \frac{16\pi^3\mu_0}{3h} \frac{n^3}{\lambda_{if}^3} \frac{\hat{\mathcal{S}}(\psi_i, \psi_f)}{g_i}, \quad (14)$$

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

```

1 MagDipoleRates::usage = "MagDipoleRates[eigenSys, numE] calculates
2 the magnetic dipole transition rate array for the provided
3 eigensystem. The option ''Units'' can be set to ''SI'' or to ''
4 ''Hartree''. If the option ''Natural Radiative Lifetimes'' is set to
5 true then the reciprocal of the rate is returned instead.
6 eigenSys is a list of lists with two elements, in each list the
7 first element is the energy and the second one the corresponding
8 eigenvector.
9 Based on table 7.3 of Thorne 1999, using g2=1.
10 The energy unit assumed in eigenSys is kayser.

```

```

4 The returned array should be interpreted in the eigenbasis of the
5 Hamiltonian. As such the element AMD[[i,i]] corresponds to the
6 transition rate (or the radiative lifetime, depending on options)
7 between eigenstates  $|i\rangle$  and  $|j\rangle$ .
8 By default this assumes that the refractive index is unity, this may
9 be changed by setting the option ''RefractiveIndex'' to the
10 desired value.
11 The option ''Lifetime'' can be used to return the reciprocal of the
12 transition rates. The default is to return the transition rates.";
13 Options[MagDipoleRates]={ "Units" -> "SI", "Lifetime" -> False, "
14 RefractiveIndex" -> 1};
15 MagDipoleRates[eigenSys_List, numE0_Integer, OptionsPattern[]] :=
16 Module[
17 {AMD, Stot, eigenEnergies,
18 transitionWaveLengthsInMeters, nRefractive},
19 (
20   nRefractive = OptionValue["RefractiveIndex"];
21   numE = Min[14 - numE0, numE0];
22   Stot = MagDipLineStrength[eigenSys, numE, "Units" ->
23 OptionValue["Units"]];
24   eigenEnergies = Chop[First/@eigenSys];
25   energyDiffs = Outer[Subtract, eigenEnergies, eigenEnergies];
26   energyDiffs = ReplaceDiagonal[energyDiffs, Indeterminate];
27   (* Energies assumed in kayser.*)
28   transitionWaveLengthsInMeters = 0.01/energyDiffs;
29
30   unitFactor = Which[
31     OptionValue["Units"] == "Hartree",
32     (
33       (* The bohrRadius factor in SI needed to convert the
34 wavelengths which are assumed in m*)
35       16 \[Pi]^3 (\[Mu]0Hartree /(3 hPlanckHartree)) * bohrRadius^3
36     ),
37     OptionValue["Units"] == "SI",
38     (
39       16 \[Pi]^3 \[Mu]0/(3 hPlanck)
40     ),
41     True,
42     (
43       Echo["Invalid option for ''Units''. Options are ''SI'' and ''"
44       Hartree"."];
45       Abort[];
46     )
47   ];
48   AMD = unitFactor / transitionWaveLengthsInMeters^3 * Stot *
49   nRefractive^3;
50   Which[OptionValue["Lifetime"],
51     Return[1/AMD],
52     True,
53     Return[AMD]
54   ]
55 ]
56 )
57 ];

```

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

$$f_{MD}(|\psi_g\rangle \rightsquigarrow |\psi_e\rangle) = \frac{8\pi^2 m_e}{3 h c e^2} \frac{n}{\lambda_{ge}} \frac{\hat{S}(\psi_g, \psi_e)}{g_g} \quad (115)$$

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

In **qlanth** the function **GroundMagDipoleOscillatorStrength** implements the calculation of the oscillator strengths from the ground state to all the excited ones.

```

1 GroundMagDipoleOscillatorStrength::usage =
2   GroundMagDipoleOscillatorStrength[eigenSys, numE] calculates the
3   magnetic dipole oscillator strengths between the ground state and
4   the excited states as given by eigenSys.
5 Based on equation 8 of Carnall 1965, removing the  $2J+1$  factor since
6   this degeneracy has been removed by the crystal field.

```

```

3 eigenSys is a list of lists with two elements, in each list the first
4   element is the energy and the second one the corresponding
5   eigenvector.
6 The energy unit assumed in eigenSys is Kayser.
7 The oscillator strengths are dimensionless.
8 The returned array should be interpreted in the eigenbasis of the
9   Hamiltonian. As such the element fMDGS[[i]] corresponds to the
10  oscillator strength between ground state and eigenstate |i>.
11 By default this assumes that the refractive index is unity, this may
12  be changed by setting the option ''RefractiveIndex'' to the
13  desired value.";
14 Options[GroundMagDipoleOscillatorStrength]={ "RefractiveIndex" -> 1};
15 GroundMagDipoleOscillatorStrength[eigenSys_List, numE_Integer,
16   OptionsPattern[]] := Module[
17   {eigenEnergies, SMDGS, GSEnergy, energyDiffs,
18   transitionWaveLengthsInMeters, unitFactor, nRefractive},
19   (
20     eigenEnergies = First/@eigenSys;
21     nRefractive = OptionValue["RefractiveIndex"];
22     SMDGS = MagDipLineStrength[eigenSys, numE, "Units" -> "SI",
23       "States" -> 1];
24     GSEnergy = eigenSys[[1, 1]];
25     energyDiffs = eigenEnergies - GSEnergy;
26     energyDiffs[[1]] = Indeterminate;
27     transitionWaveLengthsInMeters = 0.01/energyDiffs;
28     unitFactor = (8\[Pi]^2 me)/(3 hPlanck eCharge^2 cLight);
29     fMDGS = unitFactor / transitionWaveLengthsInMeters *
30     SMDGS * nRefractive;
31     Return[fMDGS];
32   )
33 ];

```

## 6.2 Level description

### 6.2.1 Forced electric dipole transitions

Any two eigenfunctions that are approximated within the limits of a single configuration cannot help but have the same parity as they are spanned by basis vectors with definite and shared parity. Analysis of the amplitudes for different transition operators can then inform as to what transitions are forbidden, which are those in which the product of the parity of the two participating wavefunctions and that of the transition operator results in odd parity. As such, within the single configuration approximation, since the product of the two participating wavefunctions is always even, then any transition described by an operator of odd parity is forbidden. This is the content of Laporte's parity selection rule. Since the parity of the magnetic dipole operator is even<sup>21</sup>, then this operator allows for intra-configuration transitions, and since the parity of the electric dipole operator is odd, then these types of intra-configuration transitions are forbidden.

However, much as configuration interaction is an essential component in the description of the electronic structure, it has a bearing on the energy spectrum and the intra-configuration wavefunctions themselves. Configuration interaction may also be used to bring back into the analysis the fact that the *actual* wavefunctions will also have at least a small part of them in other configurations, even if most of them may be within the ground configuration. It is therefore the case that the *actual* parity of the wavefunctions is mixed, and therefore intra-configuration<sup>22</sup> electric dipole transitions are actually allowed. These electric dipole transitions are called *forced* electric dipole transitions.

Judd [Jud62] and Ofelt [Ofe62] came separately to similar versions of this analysis, and showed after a series of approximations that the forced electric dipole transitions could be described by the intra-configuration matrix elements of the multi-electron unit operators  $\hat{U}^{(k)}$  (for  $k=2,4,6$ ) together with a set of three accompanying coefficients  $\{\Omega_{(2)}, \Omega_{(4)}, \Omega_{(6)}\}$ . These coefficients have a definite form related to the overlap between the mixed parity parts of the corrected wavefunctions, but they can also be considered as additional phenomenological parameters.

Judd-Ofelt theory is based on the level description, and its mathematical expression is the following. Given two intermediate coupling levels  $|\alpha SLJ\rangle$  and  $|\alpha' SL'J'\rangle$ , the oscillator

<sup>21</sup> The parity of the electric quadrupole operator is also even, but we haven't included it in qlanth

<sup>22</sup> Calling these *intra*-configuration transitions is somewhat of a misnomer since their nature is tied to the fact that the single-configuration description is wanting.

strength between them is approximated as [Jud62]

$$f_{\text{f-ED}}(|\alpha LSJ\rangle \rightsquigarrow |\alpha' SL'J'\rangle) = \mathcal{R} \frac{8\pi^2 m_e}{3h} \frac{\nu}{2J+1} \frac{\chi}{n} \sum_{k=2,4,6} \Omega_{(k)} \left| \langle \underline{f}^n \alpha SLJ | \hat{U}^{(k)} | \underline{f}^n \alpha' SL'J' \rangle \right|^2, \quad (116)$$

where  $\nu$  is the frequency of the transition,  $\chi$  the local field correction,  $n$  the refractive index of the crystal host, and  $\mathcal{R} = 1$  in the case of absorption and  $\mathcal{R} = n^2$  in the case of emission.

The local field correction  $\chi$  accounts for the difference between the macroscopic and microscopic electric fields, in the case of ions embedded for crystals the most common choice is

$$\chi = \frac{n^2 + 2}{3} \quad (117)$$

and for other environments (or emitters other than ions such as molecules) different alternatives are relevant (see [DR06]).

In **qlanth** Judd-Olfelt theory is implemented with help of the functions **JuddOfeltUk-Squared** and **LevelElecDipoleOscillatorStrength**.

```

1 JuddOfeltUkSquared::usage = "JuddOfeltUkSquared[numE, params]
2   calculates the matrix elements of the Uk operator in the level
3   basis. These are calculated according to equation (7) in Carnall
4   1965.
5 The function returns a list with the following elements:
6   - basis : A list with the allowed {SL, J} terms in the f^n configuration. Equal to BasisLSJ[numE].
7   - eigenSys : A list with the eigensystem of the Hamiltonian for the f^n configuration.
8   - levelLabels : A list with the labels of the major components of
9   the level eigenstates.
10  - LevelUkSquared : An association with the squared matrix elements
11   of the Uk operators in the level eigenbasis. The keys being {2, 4,
12   6} corresponding to the rank of the Uk operator. The basis in
13   which the matrix elements are given is the one corresponding to
14   the level eigenstates given in eigenSys and whose major SLJ
15   components are given in levelLabels. The matrix is symmetric and
16   given as a SymmetrizedArray.
17 The function admits the following options:
18   ''PrintFun'' : A function that will be used to print the progress
19   of the calculations. The default is PrintTemporary.";
20 Options[JuddOfeltUkSquared] = {"PrintFun" -> PrintTemporary};
21 JuddOfeltUkSquared[numE_, params_, OptionsPattern[]] := Module[
22   {eigenChanger, numEH, basis, eigenSys,
23   Js, Ukmatrix, LevelUkSquared, kRank,
24   S, L, Sp, Lp, J, Jp, phase,
25   braTerm, ketTerm, levelLabels,
26   eigenVecs, majorComponentIndices},
27   (
28     If[Not[ValueQ[ReducedUkTable]],
29      LoadUk[]
30    ];
31    numEH = Min[numE, 14-numE];
32    PrintFun = OptionValue["PrintFun"];
33    PrintFun["> Calculating the levels for the given parameters ..."];
34    {basis, majorComponents, eigenSys} = LevelSolver[numE, params];
35    (* The change of basis matrix to the eigenstate basis *)
36    eigenChanger = Transpose[Last /@ eigenSys];
37    PrintFun["Calculating the matrix elements of Uk in the physical
38    coupling basis ..."];
39    LevelUkSquared = <||>;
40    Do[(
41      Ukmatrix = Table[(
42        {S, L} = FindSL[braTerm[[1]]];
43        J = braTerm[[2]];
44        Jp = ketTerm[[2]];
45        {Sp, Lp} = FindSL[ketTerm[[1]]];
46        phase = Phaser[S + Lp + J + kRank];
47        Simplify @ (
48          phase *
49          Sqrt[TPO[J]*TPO[Jp]] *
50          SixJay[{J, Jp, kRank}, {Lp, L, S}] *
51          ReducedUkTable[{numEH, 3, braTerm[[1]], ketTerm[[1]],
52          kRank}])
53      ]
54    ]
55  ]
56
```

```

40         )
41     ),
42     {braTerm, basis},
43     {ketTerm, basis}
44 ];
45 Ukmatrix = (Transpose[eigenChanger] . Ukmatrix . eigenChanger)^2;
46 Ukmatrix = Chop@Ukmatrix;
47 LevelUkSquared[kRank] = SymmetrizedArray[Ukmatrix, Dimensions[
48 eigenChanger], Symmetric[{1, 2}]];
49     ),
50     {kRank, {2, 4, 6}}
51 ];
52 LSJmultiplets = (RemoveTrailingDigits[#[[1]]] <> ToString[
53 InputForm[#[[2]]]]) & /@ basis;
54 eigenVecs = Last /@ eigenSys;
55 majorComponentIndices = Ordering[Abs[#][[-1]] & /@ eigenVecs];
56 levelLabels = LSJmultiplets[[majorComponentIndices]];
57 Return[{basis, eigenSys, levelLabels, LevelUkSquared}];
58 )
59 ];

```

```

1 LevelElecDipoleOscillatorStrength::usage =
2   LevelElecDipoleOscillatorStrength[numE_, levelParams,
3     juddOfeltParams] uses Judd-Ofelt theory to estimate the forced
4     electric dipole oscillator strengths ions whose level description
5     is determined by levelParams.
6 The third parameter juddOfeltParams is an association with keys equal
7     to the three Judd-Ofelt intensity parameters {\[CapitalOmega]2,
8     \[CapitalOmega]4, \[CapitalOmega]6} and corresponding values in cm
9     ^-2.
10 The local field correction implemented here corresponds to the one
11     given by the virtual cavity model of Lorentz.
12 The function returns a list with the following elements:
13 - basis : A list with the allowed {SL, J} terms in the f^n configuration. Equal to BasisLSJ[numE].
14 - eigenSys : A list with the eigensystem of the Hamiltonian for the
15     f^n configuration in the level description.
16 - levelLabels : A list with the labels of the major components of
17     the calculated levels.
18 - oStrengthArray : A square array whose elements represent the
19     oscillator strengths between levels such that the element
20     oStrengthArray[[i,j]] is the oscillator strength between the
21     levels |Subscript[\[Psi], i]> and |Subscript[\[Psi], j]>. In this
22     array, the elements below the diagonal represent emission
23     oscillator strengths, and elements above the diagonal represent
24     absorption oscillator strengths.
25 The function admits the following three options:
26   ''PrintFun'' : A function that will be used to print the progress
27     of the calculations. The default is PrintTemporary.
28   ''RefractiveIndex'' : The refractive index of the medium where the
29     transitions are taking place. This may be a number or a function.
30     If a number then the oscillator strengths are calculated for
31     assuming a wavelength-independent refractive index. If a function
32     then the refractive indices are calculated accordingly to the
33     wavelength of each transition (the function must admit a single
34     argument equal to the wavelength in nm). The default is 1.
35   ''LocalFieldCorrection'' : The local field correction to be used.
36     The default is ''VirtualCavity''. The options are: ''VirtualCavity''
37     and ''EmptyCavity''.
38 The equation implemented here is the one given in eqn. 29 from the
39     review article of Hehlen (2013). See that same article for a
40     discussion on the local field correction.
41 ";
42 Options[LevelElecDipoleOscillatorStrength]={
43   "PrintFun"          -> PrintTemporary,
44   "RefractiveIndex"  -> 1,
45   "LocalFieldCorrection" -> "VirtualCavity"
46 };
47 LevelElecDipoleOscillatorStrength[numE_, levelParams_Association,
48   juddOfeltParams_Association, OptionsPattern[]] := Module[
49   {PrintFun, basis, eigenSys, levelLabels,
50   LevelUkSquared, eigenEnergies, energyDiffs,
51   oStrengthArray, nRef, \[Chi], nRefs,
52   \[Chi]OverN, groundLevel, const,
53   transitionFrequencies, wavelengthsInNM,
54   fieldCorrectionType},
55 
```

```

27  (
28      PrintFun = OptionValue["PrintFun"];
29      nRef     = OptionValue["RefractiveIndex"];
30      PrintFun["Calculating the  $U_k^2$  matrix elements for the given
31      parameters ..."];
32      {basis, eigenSys, levelLabels, LevelUkSquared} =
33      JuddOfeltUkSquared[numE, levelParams, "PrintFun" -> PrintFun];
34      eigenEnergies = First/@eigenSys;
35      (* converted to cm-2 *)
36      const        = (8\[\Pi]2/3 me/hPlanck * 10(-4);
37      energyDiffs = Transpose@Outer[Subtract, eigenEnergies,
38      eigenEnergies];
39      (* since energies are assumed in Kayser, speed of light needs to
40      be in cm/s, so that the frequencies are in 1/s *)
41      transitionFrequencies = energyDiffs*cLight*100;
42      (* grab the J for each level *)
43      levelJs       = #[[2]] & /@ eigenSys;
44      oStrengthArray = (
45          juddOfeltParams[\[CapitalOmega]2]*LevelUkSquared[2]+
46          juddOfeltParams[\[CapitalOmega]4]*LevelUkSquared[4]+
47          juddOfeltParams[\[CapitalOmega]6]*LevelUkSquared[6]
48      );
49      oStrengthArray = Abs@(const * transitionFrequencies *
50      oStrengthArray);
51      (* it is necessary to divide each oscillator strength by the
52      degeneracy of the initial level *)
53      oStrengthArray = MapIndexed[1/(2 levelJs[[#2[[1]]]]+1) #1 &,
54      oStrengthArray, {2}];
55      (* including the effects of the refractive index *)
56      fieldCorrectionType = OptionValue["LocalFieldCorrection"];
57      Which[
58          nRef === 1,
59          True,
60          NumberQ[nRef],
61          (
62              \[Chi] = Which[
63                  fieldCorrectionType == "VirtualCavity",
64                  (
65                      ( (nRef2 + 2) / 3 )2
66                  ),
67                  fieldCorrectionType == "EmptyCavity",
68                  (
69                      ( 3 * nRef2 / ( 2 * nRef2 + 1 ) )2
70                  )
71              ];
72              \[Chi]OverN = \[Chi] / nRef;
73              oStrengthArray = \[Chi]OverN * oStrengthArray;
74              (* the refractive index participates differently in
75              absorption and in emission *)
76              aFunction      = If[#2[[1]] > #2[[2]], #1 * nRef2, #1]&;
77              oStrengthArray = MapIndexed[aFunction, oStrengthArray, {2}];
78          ),
79          True,
80          (
81              wavelengthsInNM = Abs[1 / energyDiffs] * 107;
82              nRefs           = Map[nRef, wavelengthsInNM];
83              Echo["Calculating the oscillator strengths for the given
84              refractive index ..."];
85              \[Chi] = Which[
86                  fieldCorrectionType == "VirtualCavity",
87                  (
88                      ( (nRefs2 + 2) / 3 )2
89                  ),
90                  fieldCorrectionType == "EmptyCavity",
91                  (
92                      ( 3 * nRefs2 / ( 2*nRefs2 + 1 ) )2
93                  )
94              ];
95              \[Chi]OverN = \[Chi] / nRefs;
96              oStrengthArray = \[Chi]OverN * oStrengthArray
97          )
98      ];
99      Return[{basis, eigenSys, levelLabels, oStrengthArray}];
100  )
101 ];
```

### 6.2.2 Magnetic dipole transitions

In Hartree atomic units, the magnetic dipole line strength between levels  $|\alpha LSJ\rangle$  and  $|\alpha' S'L'J'\rangle$  is given by

$$\hat{\mathcal{S}}(|\alpha LSJ\rangle, |\alpha' S'L'J'\rangle) = \left| \langle \alpha LSJ | \frac{1}{2} (\hat{\mathbf{L}} + g \hat{\mathbf{S}}) | \alpha' S'L'J' \rangle \right|^2 \quad (118)$$

In **qlanth** the line strength can be calculated using the function `LevelMagDipoleLineStrength`.

```

1 LevelMagDipoleLineStrength::usage = "LevelMagDipoleLineStrength[
2   eigenSys, numE] calculates the magnetic dipole line strengths for
3   an ion whose level description is determined by levelParams. The
4   function returns a square array whose elements represent the
5   magnetic dipole line strengths between the levels given in
6   eigenSys such that the element magDipoleLineStrength[[i,j]] is the
7   line strength between the levels |Subscript[\[Psi], i]> and |Subscript[\[Psi], j]>. Eigensys must be such that it consists of a
8   lists of lists where in each list the last element corresponds to
9   the eigenvector of a level (given as a row) in the standard basis
10  for levels of the f^numE configuration.
11 The function admits the following options:
12   'Units' : The units in which the line strengths are given. The
13   default is 'SI'. The options are 'SI' and 'Hartree'. If 'SI'
14   then the unit of the line strength is (A m^2)^2 = (J/T)^2. If
15   'Hartree' then the line strength is given in units of 2 \[Mu]B."
16 ;
17 Options[LevelMagDipoleLineStrength] = {
18   "Units" -> "SI"
19 };
20 LevelMagDipoleLineStrength[theEigensys_List, numE0_Integer,
21   OptionsPattern[]] := Module[
22   {numE, levelMagOp, allEigenvecs, magDipoleLineStrength, units},
23   (
24     numE          = Min[14 - numE0, numE0];
25     levelMagOp    = LevelMagDipoleMatrixAssembly[numE];
26     allEigenvecs  = Transpose[Last /@ theEigensys];
27     units         = OptionValue["Units"];
28     magDipoleLineStrength      = Transpose[allEigenvecs].levelMagOp.allEigenvecs;
29     magDipoleLineStrength       = Abs[magDipoleLineStrength]^2;
30     Which[
31       units == "SI",
32         Return[4 \[Mu]B^2 * magDipoleLineStrength],
33       units == "Hartree",
34         Return[magDipoleLineStrength]
35     ];
36   )
37 ]
38 ];
```

In atomic units, the magnetic dipole oscillator strength for a transition between level  $|\alpha LSJ\rangle$  and an excited level  $|\alpha' S'L'J'\rangle$  is given by [Rud07]

$$f_{MD} (|\alpha LSJ\rangle \rightsquigarrow |\alpha' S'L'J'\rangle) = \frac{2n}{3} \frac{\mathcal{E}(|\alpha' S'L'J'\rangle) - \mathcal{E}(|\alpha LSJ\rangle)}{2J+1} \alpha^2 \hat{\mathcal{S}} (|\alpha LSJ\rangle, |\alpha' S'L'J'\rangle) \quad (119)$$

where  $\mathcal{E}(|\alpha LSJ\rangle)$  is the energy of level  $|\alpha LSJ\rangle$ ,  $n$  is the refractive index of the medium, and  $\alpha$  is the fine structure constant. In obtaining this expression one considers the transition from one state of the initial level into another single state of the final level. Furthermore, here it is assumed that all the states of the initial level are equally populated.

In **qlanth** the function `LevelMagDipoleOscillatorStrength` can be used to calculate these.

```

1 LevelMagDipoleOscillatorStrength::usage = "
2   LevelMagDipoleOscillatorStrength[eigenSys, numE] calculates the
3   magnetic dipole oscillator strengths for an ion whose levels are
4   described by eigenSys in configuration f^numE. The refractive
5   index of the medium is relevant, but here it is assumed to be 1,
6   this can be changed through the option 'RefractiveIndex'.
7   eigenSys must consist of a lists of lists with three elements: the
8   first element being the energy of the level, the second element
9   being the J of the level, and the third element being the
10  eigenvector of the level.
11 The function returns a list with the following elements:
```

```

3      - basis : A list with the allowed {SL, J} terms in the f^numE
4      configuration. Equal to BasisLSJ[numE].
5      - eigenSys : A list with the eigensystem of the Hamiltonian for
6      the f^n configuration in the level description.
7      - magDipole0strength : A square array whose elements represent
8      the magnetic dipole oscillator strengths between the levels given
9      in eigenSys such that the element magDipole0strength[[i,j]] is the
10     oscillator strength between the levels |Subscript[\[Psi], i]> and
11     |Subscript[\[Psi], j]>. In this array the elements below the
12     diagonal represent emission oscillator strengths, and elements
13     above the diagonal represent absorption oscillator strengths. The
14     emission oscillator strengths are negative. The oscillator
15     strength is a dimensionless quantity.
16
17 The function admits the following option:
18   ''RefractiveIndex'' : The refractive index of the medium where
19   the transitions are taking place. This may be a number or a
20   function. If a number then the oscillator strengths are calculated
21   assuming a wavelength-independent refractive index as given. If a
22   function then the refractive indices are calculated accordingly
23   to the vaccum wavelength of each transition (the function must
24   admit a single argument equal to the wavelength in nm). The
25   default is 1.
26
27 For reference see equation (27.8) in Rudzikas (2007). The
28 expression for the line strenght is the simplest when using atomic
29 units, (27.8) is missing a factor of  $\alpha^2$ .;
30 Options[LevelMagDipoleOscillatorStrength]={
31   "RefractiveIndex" -> 1
32 };
33 LevelMagDipoleOscillatorStrength[eigenSys_, numE_, OptionsPattern[]]
34   := Module[
35   {eigenEnergies, eigenVecs, levelJs,
36   energyDiffs, magDipole0strength, nRef,
37   wavelengthsInNM, nRefs, degenDivisor},
38   (
39     basis          = BasisLSJ[numE];
40     eigenEnergies = First/@eigenSys;
41     nRef          = OptionValue["RefractiveIndex"];
42     eigenVecs     = Last/@eigenSys;
43     levelJs       = #[[2]]&/@eigenSys;
44     energyDiffs   = -Outer[Subtract,eigenEnergies,eigenEnergies];
45     energyDiffs *= kayserToHartree;
46     magDipole0strength = LevelMagDipoleLineStrength[eigenSys, numE, "Units"->"Hartree"];
47     magDipole0strength = 2/3 *  $\alpha$ Fine^2 * energyDiffs *
48     magDipole0strength;
49     degenDivisor    = #1 / ( 2 * levelJs[[#2[[1]]]] + 1 ) &;
50     magDipole0strength = MapIndexed[degenDivisor, magDipole0strength,
51     {2}];
52
53     Which[nRef==1,
54       True,
55       NumberQ[nRef],
56       (
57         magDipole0strength = nRef * magDipole0strength;
58       ),
59       True,
60       (
61         wavelengthsInNM    = Abs[kayserToHartree / energyDiffs] *
62         10^7;
63         nRefs              = Map[nRef, wavelengthsInNM];
64         magDipole0strength = nRefs * magDipole0strength;
65       )
66     ];
67     Return[{basis, eigenSys, magDipole0strength}];
68   )
69 ]
70

```

A final quantity of interest is the spontaneous magnetic dipole decay rate from one level to a lower lying one. In atomic units this rate is determined by

$$\Gamma_{MD}(|\alpha LSJ\rangle \rightsquigarrow |\alpha' S'L'J'\rangle) = \frac{4n^3}{3} \frac{(\mathcal{E}(|\alpha LSJ\rangle) - \mathcal{E}(|\alpha' S'L'J'\rangle))^3}{2J+1} \alpha^5 \hat{\mathcal{S}}(|\alpha LSJ\rangle, |\alpha' S'L'J'\rangle). \quad (120)$$

In **qlanth** the spontaneous decay rates may be calculated through the function **LevelMagDipoleSpotaneousDecayRates**.

```

1 LevelMagDipoleSpotaneousDecayRates::usage = "
2   LevelMagDipoleSpotaneousDecayRates[eigenSys, numE] calculates the
3   spontaneous emission rates for the magnetic dipole transitions
4   between the levels given in eigenSys. The function returns a
5   square array whose elements represent the spontaneous emission
6   rates between the levels given in eigenSys such that the element
7   [[i,j]] of the returned array is the rate of spontaneous emission
8   from the level |Subscript[\[Psi], i]> to the level |Subscript[\[Psi], j]>. In this array the elements below the diagonal represent
9   emission rates, and elements above the diagonal are identically
10  zero.
11 The function admits two optional arguments:
12  + \"Units\" : The units in which the rates are given. The default
13    is \"SI\". The options are \"SI\" and \"Hartree\". If \"SI\" then
14    the rates are given in s^-1. If \"Hartree\" then the rates are
15    given in the atomic unit of frequency.
16  + \"RefractiveIndex\" : The refractive index of the medium where
17    the transitions are taking place. This may be a number or a
18    function. If a number then the rates are calculated assuming a
19    wavelength-independent refractive index as given. If a function
20    then the refractive indices are calculated accordingly to the
21    vacuum wavelength of each transition (the function must admit a
22    single argument equal to the wavelength in nm). The default is 1."
23 ;
24 Options[LevelMagDipoleSpotaneousDecayRates] = {
25   "Units" -> "SI",
26   "RefractiveIndex" -> 1};
27 LevelMagDipoleSpotaneousDecayRates[eigenSys_List, numE_Integer,
28   OptionsPattern[]] := Module[
29   {
30     levMDlineStrength, eigenEnergies, energyDiffs, levelJs,
31     spontaneousRatesInHartree, spontaneousRatesInSI, degenDivisor,
32     units,
33     nRef, nRefs, wavelengthsInNM
34   },
35   (
36     nRef           = OptionValue["RefractiveIndex"];
37     units          = OptionValue["Units"];
38     levMDlineStrength = LowerTriangularize@LevelMagDipoleLineStrength
39     [eigenSys, numE, "Units" -> "Hartree"];
40     levMDlineStrength = SparseArray[levMDlineStrength];
41     eigenEnergies    = First /@ eigenSys;
42     energyDiffs      = Outer[Subtract, eigenEnergies, eigenEnergies
43     ];
44     energyDiffs      = kayserToHartree * energyDiffs;
45     energyDiffs      = SparseArray[LowerTriangularize[energyDiffs]];
46     levelJs          = #[[2]] & /@ eigenSys;
47     spontaneousRatesInHartree = 4/3 αFine^5 * energyDiffs^3 *
48     levMDlineStrength;
49     degenDivisor      = #1 / (2*levelJs[[#2[[1]]]] + 1) &;
50     spontaneousRatesInHartree = MapIndexed[degenDivisor,
51     spontaneousRatesInHartree, {2}];
52     Which[nRef === 1,
53       True,
54       NumberQ[nRef],
55       (
56         spontaneousRatesInHartree = nRef^3 *
57         spontaneousRatesInHartree;
58       ),
59       True,
60       (
61         wavelengthsInNM      = Abs[kayserToHartree / energyDiffs] *
62         10^7;
63         nRefs                  = Map[nRef, wavelengthsInNM];
64         spontaneousRatesInHartree = nRefs^3 *
65         spontaneousRatesInHartree;
66       )
67     ];
68   ];
69   If[units == "SI",
70   (
71     spontaneousRatesInSI = 1/hartreeTime *
72     spontaneousRatesInHartree;
73     Return[SparseArray@spontaneousRatesInSI];
74   ),
75   Return[SparseArray@spontaneousRatesInHartree];
76 ];
77 
```

```
46 ] )
47 ] ;
```

## 7 Parameter constraints

When there is a scarcity of experimental data, one useful strategy to reduce the number of free parameters is to enforce some constraints between ratios of Slater integrals  $F^{(k)}$ , Marvin integrals  $m^{(k)}$ , and pseudo-magnetic parameters  $P^{(k)}$ .

For the Slater integrals one may leave only  $F^{(2)}$  as free parameter, and fix the ratios  $F^{(4)}/F^{(2)}$  and  $F^{(6)}/F^{(2)}$ .

For the Marvin integrals one often leaves only a single free parameter  $m^{(0)}$ , and give values to  $m^{(2)}$  and  $m^{(4)}$ , by fixing the ratios  $m^{(2)}/m^{(0)}$  and  $m^{(4)}/m^{(0)}$ .

For the pseudo-magnetic parameters again the common practice is to only leave a single free parameter  $P^{(2)}$ , and give values to  $P^{(4)}$  and  $P^{(6)}$ , by fixing the ratios  $P^{(4)}/P^{(2)}$  and  $P^{(6)}/P^{(2)}$ .

The values for all these ratios were historically obtained by using the integral expressions for the corresponding parameters, and calculating them using Hartree-Fock solutions to the radial parts of the wavefunctions. Examples of these ratios can be seen in the sections with data for LaF<sub>3</sub> and LiYF<sub>4</sub>.

## 8 Fitting experimental data

`qlanth` also has the capacity to fit the semi-empirical Hamiltonian to experimental data. This is included in the sub-module `fittings.m` (see [Appendix 18.2](#)). This sub-module includes the function `ClassicalFit` which uses a truncated Hamiltonian (based on free-ion energies) to fit a given subset of the model parameters to given experimental data. It yields an extensive set of results, including fitted parameters and uncertainties. If the truncation energy parameter is set to infinity, then the fitting is performed with no truncation.

This function, however, is specifically used for fitting data for a single ion in a specific host. In the case of fitting data for several ions, it may be necessary to use parameter trends  $\mathcal{P}(n)$ . This is necessary since not only there might be some ions where there is no data (and where one would then propose a “synthetic” solution), but also since there are cases where there are too few data points to justify varying all of the model parameters.<sup>23</sup>

In these cases where one is fitting data for all or most of the lanthanide ions in a given host, it is useful to first fit the model in cases where there are the most data points, and to build up a parameter model  $\mathcal{P}(n)$  for each parameter as the fitting of all the ions progresses. One feature often used in fitting for several ions (see Carnall *et al.* [[Car+89](#)] and Cheng *et al.* [[Che+16](#)]) is that when there is scarcity of data (as mentioned above), one can then use the trends in the  $\mathcal{P}(n)$  in order to fix some parameter values at a given column (and proceed to vary others to fit the data to the model).<sup>24</sup>

Here below is a detailed explanation of the parameters required by `ClassicalFit`. The code for this function `ClassicalFit` may be found in [Appendix 18.2](#).

- `numE`: number of electrons in the system, specifying the electronic configuration.
- `expData`: experimental data, a list of lists where each sublist represents an energy level and associated parameters. The first element of the sublists must represent energies, the other elements in the sublists are ignored but can be given to be kept together with the fitted data. The data must be ordered in increasing order of energy. **IMPORTANT:** if there are known unknown levels, these should be made explicit, anything other than a number will be interpreted as a level of undetermined energy in the corresponding gap. **ALSO IMPORTANT:** in the case of odd electron cases, `expData` needs to explicitly include the duplicate energies corresponding to Kramers’ degeneracy; the gaps also need to be adequately duplicated in these cases.

<sup>23</sup> The extreme case of this scarcity being Yb and Ce, where there are only 7 non-degenerate energies, but where the crystal-field alone might require more parameters than these (for instance in  $C_{2v}$  symmetry one needs 9  $B_q^{(k)}$  parameters). <sup>24</sup> For example, in the [Table ??](#) for LaF<sub>3</sub>, in the case of Yb, only two parameters are varied ( $\zeta$  and  $\epsilon$ ) and the values for the crystal field obtained from linear fits to the previously fitted  $B_q^{(k)}$  in Pr, Nd, Dy, Sm\*, Ho\*, Er, and Tm. (\*) not all  $B_q^{(k)}$

- `excludeDataIndices`: indices in `expData` to be excluded from the fitting process. This can be used to exclude experimental data which is present, but which is considered dubious. In the case of odd electron configurations, these indices need to explicitly include the double degeneracy of Kramers doublets.
- `problemVars`: symbols representing the parameters to be fitted, some of which may be constrained (set fixed or proportional to others). **IMPORTANT:** if `problemVars` is a proper subset of all the parameters needed to evaluate the simplified Hamiltonian, the values for the other necessary parameters are taken from the Carnall *et al.* [Car+89] systematic study of LaF<sub>3</sub>.
- `startValues`: an association with the initial values for the independent parameters given in `problemVars`. Independent parameters are those that remain once the constraints have been accounted for.
- $\sigma_{\text{exp}}$ : estimated uncertainty in the energy level differences between experimental and calculated values.
- `constraints`: a list of replacement rules defining constraints on the parameters. These constraints can either pin down a value, or apply proportionality ratios between them. If constrained by proportionally factors, these ratios are usually taken from Hartree-Fock calculations.

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

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

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

- `Energy Uncertainty in K`: used for error estimation, it can be either `Automatic` in which case the  $(\sigma$  of the fit is used as the uncertainty of the energies) or a numeric value in which case that is the value used for error propagation.

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

The function returns an association with the following keys.

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

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

```

1 ClassicalFit::usage="ClassicalFit[numE, expData, excludeDataIndices,
2   problemVars, startValues, constraints] fits the given expData in
3   an  $f^{\text{numE}}$  configuration, by using the symbols in problemVars. The
4   symbols given in problemVars may be constrained or held constant,
5   this being controlled by constraints list which is a list of
6   replacement rules expressing desired constraints. The constraints
7   list additional constraints imposed upon the model parameters that
8   remain once other simplifications have been ‘‘baked’’ into the
9   compiled Hamiltonians that are used to increase the speed of the
10  calculation.
11
12 Important, note that in the case of odd number of electrons the given
13  data must explicitly include the Kramers degeneracy;
14  excludeDataIndices must be compatible with this.
15
16 The list expData needs to be a list of lists with the only
17  restriction that the first element of them corresponds to energies
18  of levels. In this list, an empty value can be used to indicate
19  known gaps in the data. Even if the energy value for a level is
20  known (and given in expData) certain values can be omitted from
21  the fitting procedure through the list excludeDataIndices, which
22  correspond to indices in expData that should be skipped over.
23

```

```

7 The Hamiltonian used for fitting is a version that has been truncated
8 either by using the maximum energy given in expData or by
9 manually setting a truncation energy using the option ''TruncationEnergy''.
10
11 The option ''Experimental Uncertainty in K'' is the estimated
12 uncertainty in the differences between the calculated and the
13 experimental energy levels. This is used to estimate the
14 uncertainty in the fitted parameters. Admittedly this will be a
15 rough estimate (at least on the contribution of the calculated
16 uncertainty), but it is better than nothing and may at least
17 provide a lower bound to the uncertainty in the fitted parameters.
18 It is assumed that the uncertainty in the differences between the
19 calculated and the experimental energy levels is the same for all
20 of them.
21
22 The list startValues is a list with all of the parameters needed to
23 define the Hamiltonian (including the initial values for
24 problemVars).
25
26 The function saves the solution to a file. The file is named with a
27 prefix (controlled by the option ''FilePrefix'') and a UUID. The
28 file is saved in the log sub-directory as a .m file.
29
30 Here's a description of the different parts of this function: first
31 the Hamiltonian is assembled and simplified using the given
32 simplifications. Then the intermediate coupling basis is
33 calculated using the free-ion parameters for the given lanthanide.
34 The Hamiltonian is then changed to the intermediate coupling
35 basis and truncated. The truncated Hamiltonian is then compiled
36 into a function that can be used to calculate the energy levels of
37 the truncated Hamiltonian. The function that calculates the
38 energy levels is then used to fit the experimental data. The
39 fitting is done using FindMinimum with the Levenberg-Marquardt
40 method.
41
42 The function returns an association with the following keys:
43
44 - ''bestRMS'' which is the best \[Sigma] value found.
45 - ''bestParams'' which is the best set of parameters found for the
46 variables that were not constrained.
47 - ''bestParamsWithConstraints'' which has the best set of parameters
48 (from - ''bestParams'') together with the used constraints. These
49 include all the parameters in the model, even those that were not
50 fitted for.
51 - ''paramSols'' which is a list of the parameters trajectories during
52 the stepping of the fitting algorithm.
53 - ''timeTaken/s'' which is the time taken to find the best fit.
54 - ''simplifier'' which is the simplifier used to simplify the
55 Hamiltonian.
56 - ''excludeDataIndices'' as given in the input.
57 - ''startValues'' as given in the input.
58
59 - ''freeIonSymbols'' which are the symbols used in the intermediate
60 coupling basis.
61 - ''truncationEnergy'' which is the energy used to truncate the
62 Hamiltonian, if it was set to Automatic, the value here is the
63 actual energy used.
64 - ''numE'' which is the number of electrons in the f^numE
65 configuration.
66 - ''expData'' which is the experimental data used for fitting.
67 - ''problemVars'' which are the symbols considered for fitting
68
69 - ''maxIterations'' which is the maximum number of iterations used by
70 NMinimize.
71 - ''hamDim'' which is the dimension of the full Hamiltonian.
72 - ''allVars'' which are all the symbols defining the Hamiltonian
73 under the aggregate simplifications.
74 - ''freeBies'' which are the free-ion parameters used to define the
75 intermediate coupling basis.
76 - ''truncatedDim'' which is the dimension of the truncated
77 Hamiltonian.
78 - ''compiledIntermediateFname'' the file name of the compiled
79 function used for the truncated Hamiltonian.
80
81 - ''fittedLevels'' which is the number of levels fitted for.

```

```

42 - ''actualSteps'' the number of steps that FindMininum actually
43 took.
44 - ''solWithUncertainty'' which is a list of replacement rules of the
45 form (paramSymbol -> {bestEstimate, uncertainty}).
46 - ''rmsHistory'' which is a list of the  $\backslash[\Sigma]$  values found during
47 the fitting.
48 - ''Appendix'' which is an association appended to the log file under
49 the key ''Appendix''.
50 - ''presentDataIndices'' which is the list of indices in expData that
51 were used for fitting, this takes into account both the empty
52 indices in expData and also the indices in excludeDataIndices.
53 - ''states'' which contains a list of eigenvalues and eigenvectors
54 for the fitted model, this is only available if the option ''SaveEigenvectors'' is set to True; if a general shift of energy
55 was allowed for in the fitting, then the energies are shifted
56 accordingly.
57 - ''energies'' which is a list of the energies of the fitted levels,
58 this is only available if the option ''SaveEigenvectors'' is set
59 to False. If a general shift of energy was allowed for in the
60 fitting, then the energies are shifted accordingly.
61 - ''degreesOfFreedom'' which is equal to the number of fitted state
62 energies minus the number of parameters used in fitting.
63
64 The function admits the following options with corresponding default
65 values:
66
67 - ''MaxHistory'': determines how long the logs for the solver can be
68 .
69 - ''MaxIterations'': determines the maximum number of iterations used
70 by NMinimize.
71 - ''FilePrefix'': the prefix to use for the subfolder in the log
72 filder, in which the solution files are saved, by default this is
73 ''calcs'' so that the calculation files are saved under the
74 directory ''log/calcs''.
75 - ''AddConstantShift'': if True then a constant shift is allowed in
76 the fitting, default is False. If this is the case the variable
77 '' $\backslash[Epsilon]$ '' is added to the list of variables to be fitted for,
78 it must not be included in problemVars.
79
80 - ''AccuracyGoal'': the accuracy goal used by NMinimize, default of
81 5.
82 - ''TruncationEnergy'': if Automatic then the maximum energy in
83 expData is taken, else it takes the value set by this option. In
84 all cases the energies in expData are only considered up to this
85 value.
86 - ''PrintFun'': the function used to print progress messages, the
87 default is PrintTemporary.
88 - ''RefParamsVintage'': the vintage of the reference parameters to
89 use. The reference parameters are both used to determine the
90 truncated Hamiltonian, and also as starting values for the solver.
91 It may be ''LaF3'', in which case reference parameters from
92 Carnall are used. It may also be ''LiYF4'', in which case the
93 reference parameters from the LiYF4 paper are used. It may also be
94 Automatic, in which case the given experimental data is used to
95 determine starting values for  $F^k$  and  $\zeta$ . It may also be a list or
96 association that provides values for the Slater integrals and spin
97 -orbit coupling, the remaining necessary parameters complemented
98 by using ''LaF3''.
99
100 - ''ProgressView'': whether or not a progress window will be opened
101 to show the progress of the solver, the default is True.
102 - ''SignatureCheck'': if True then the function returns
103 prematurely, returning a list with the symbols that would have
104 defined the Hamiltonian after all simplifications have been
105 applied. Useful to check the entire parameter set that the
106 Hamiltonian has, which has to match one-to-one what is provided by
107 startValues.
108 - ''SaveEigenvectors'': if True then the both the eigenvectors and
109 eigenvalues are saved under the ''states'' key of the returned
110 association. If False then only the energies are saved, the
111 default is False.
112
113 - ''AppendToFile'': an association appended to the log file under
114 the key ''Appendix''.
115 - ''MagneticSimplifier'': a list of replacement rules to simplify the
116 Marvin and pesudo-magnetic paramters. Here the ratios of the
117

```

```

    Marvin parameters and the pseudo-magnetic parameters are defined
    to simplify the magnetic part of the Hamiltonian.
69 - ''MagFieldSimplifier'': a list of replacement rules to specify a
    magnetic field (in T), if set to {}, then {Bx, By, Bz} can also be
    used as variables to be fitted for.
70
71 - ''SymmetrySimplifier'': a list of replacements rules to simplify
    the crystal field.
72 - ''OtherSimplifier'': an additional list of replacement rules that
    are applied to the Hamiltonian before computing with it. Here the
    spin-spin contribution can be turned off by setting \[Sigma]SS->0,
    which is the default.
73 ";
74 Options[ClassicalFit] = {
75   "MaxHistory"      -> 200,
76   "MaxIterations"   -> 100,
77   "FilePrefix"       -> "calcs",
78   "ProgressView"    -> True,
79   "TruncationEnergy" -> Automatic,
80   "AccuracyGoal"    -> 5,
81   "PrintFun"         -> PrintTemporary,
82   "RefParamsVintage" -> "LaF3",
83   "SignatureCheck"   -> False,
84   "AddConstantShift" -> False,
85   "SaveEigenvectors" -> False,
86   "AppendToLogFile"  -> <||>,
87   "SaveToLog"        -> False,
88   "Energy Uncertainty in K" -> Automatic,
89   "MagneticSimplifier" -> {
90     M2 -> 56/100 MO,
91     M4 -> 31/100 MO,
92     P4 -> 1/2 P2,
93     P6 -> 1/10 P2
94   },
95   "MagFieldSimplifier" -> {
96     Bx -> 0,
97     By -> 0,
98     Bz -> 0
99   },
100  "SymmetrySimplifier" -> {
101    B12->0, B14->0, B16->0, B34->0, B36->0, B56->0,
102    S12->0, S14->0, S16->0, S22->0, S24->0, S26->0,
103    S34->0, S36->0, S44->0, S46->0, S56->0, S66->0
104  },
105  "OtherSimplifier" -> {
106    F0->0,
107    P0->0,
108    \[Sigma]SS->0,
109    T11p->0, T12->0, T14->0, T15->0,
110    T16->0, T18->0, T17->0, T19->0, T2p->0,
111    wChErrA ->0, wChErrB ->0
112  },
113  "ThreeBodySimplifier" -> <|
114    1 -> {
115      T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
116      T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
117      ->0,
118      T2p->0},
119    2 -> {
120      T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
121      T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
122      ->0,
123      T2p->0},
124    3 -> {},
125    4 -> {},
126    5 -> {},
127    6 -> {},
128    7 -> {},
129    8 -> {},
130    9 -> {},
131    10 -> {},
132    11 -> {},
133    12 -> {
134      T3->0, T4->0, T6->0, T7->0, T8->0,
      T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19

```

```

135      ->0,
136      T2p->0
137    },
138    13->{
139      T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
140      T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
141    ->0,
142      T2p->0
143    }
144  |>,
145  "FreeIonSymbols" -> {F0, F2, F4, F6,  $\zeta$ }
146 };
147 ClassicalFit[numE_Integer, expData_List, excludeDataIndices_List,
148   problemVars_List, startValues_Association, constraints_List,
149   OptionsPattern[]]:=Module[
150   {
151     accuracyGoal,
152     allFreeEnergies,
153     allFreeEnergiesSorted, allVars, allVarsVec,
154     argsForEvalInsideOfTheIntermediateSystems,
155     argsOfTheIntermediateEigensystems, aVar, aVarPosition,
156     basis, basisChanger, basisChangerBlocks,
157     bestParams, bestRMS, blockShifts, blockSizes,
158     compiledDiagonal, compiledIntermediateFname,
159     constrainedProblemVars, constrainedProblemVarsList,
160     currentRMS, degressOfFreedom, dependentVars,
161     diagonalBlocks, diagonalScalarBlocks, diff,
162     eigenEnergies, eigenvalueDispenserTemplate,
163     eigenVectors, elevatedIntermediateEigensystems,
164     endTime, fmSolAssoc, freeBies,
165     freeIenergiesAndMultiplets, fullHam, fullSolveC,
166     ham, hamDim, hamEigenvaluesTemplate,
167     hamString, indepSolveVec, indepVars, intermediateHam,
168     isolationValues, lin, linMat, ln, lnParams,
169     logFilePrefix, magneticSimplifier,
170     maxFreeEnergy, maxHistory, maxIterations,
171     minFreeEnergy, minpoly,
172     modelSymbols, multipletAssignments, needlePosition,
173     numBlocks, solCompendium,
174     openNotebooks, ordering, otherSimplifier, p0,
175     paramBest, perHam,
176     presentDataIndices, PrintFun, problemVarsPositions,
177     problemVarsQ, problemVarsQString, problemVarsVec,
178     problemVarsWithStartValues, reducedModelSymbols,
179     roundedTruncationEnergy,
180     runningInteractive, shiftToggle, simplifier,
181     sol, solWithUncertainty,
182     sortedTruncationIndex, sqdiff, standardValues,
183     startTime,
184     states, steps, symmetrySimplifier,
185     theIntermediateEigensystems, TheIntermediateEigensystems,
186     TheTruncatedAndSignedPathGenerator, timeTaken,
187     truncatedIntermediateBasis, truncatedIntermediateHam,
188     truncationEnergy, truncationIndices, RefParams,
189     truncationUmbral, varHash,
190     varsWithConstants, \[Lambda]0Vec,
191     \[Lambda]exp
192   },
193   (
194     \[Sigma]exp = OptionValue["Energy Uncertainty in K"];
195     solCompendium = <||>;
196     refParamsVintage = OptionValue["RefParamsVintage"];
197     RefParams = Which[
198       refParamsVintage === "LaF3",
199       LoadLaF3Parameters,
200       refParamsVintage === "LiYF4",
201       LoadLiYF4Parameters,
202       True,
203       refParamsVintage
204     ];
205     hamDim = Binomial[14, numE];
206     addShift = OptionValue["AddConstantShift"];
207     ln = theLanthanides[[numE]];
208     maxHistory = OptionValue["MaxHistory"];
209     maxIterations = OptionValue["MaxIterations"];
210     logFilePrefix = If[OptionValue["FilePrefix"] == "",
```

```

207             ToString[theLanthanides[[numE]]],  

208             OptionValue["FilePrefix"]  

209         ];  

210 accuracyGoal = OptionValue["AccuracyGoal"];  

211 PrintFun = OptionValue["PrintFun"];  

212 freeIonSymbols = OptionValue["FreeIonSymbols"];  

213 runningInteractive = (Head[$ParentLink] === LinkObject);  

214 magneticSimplifier = OptionValue["MagneticSimplifier"];  

215 magFieldSimplifier = OptionValue["MagFieldSimplifier"];  

216 symmetrySimplifier = OptionValue["SymmetrySimplifier"];  

217 otherSimplifier = OptionValue["OtherSimplifier"];  

218 threeBodySimplifier = If[Head[OptionValue["ThreeBodySimplifier"]]  

219 == Association,  

220             OptionValue["ThreeBodySimplifier"][numE],  

221             OptionValue["ThreeBodySimplifier"]  

222         ];  

223  

224 truncationEnergy = If[OptionValue["TruncationEnergy"] ===  

225 Automatic,  

226     (  

227         PrintFun["Truncation energy set to Automatic, using the  

228 maximum energy (+20%) in the data ..."];  

229         Round[1.2 * Max[Select[First /@ expData, NumericQ[#] &]]]  

230     ),  

231     OptionValue["TruncationEnergy"]  

232 ];  

233 truncationEnergy = Max[50000, truncationEnergy];  

234 PrintFun["Using a truncation energy of ", truncationEnergy, " K"]  

235 ;  

236  

237 simplifier = Join[magneticSimplifier,  

238                     magFieldSimplifier,  

239                     symmetrySimplifier,  

240                     threeBodySimplifier,  

241                     otherSimplifier];  

242  

243 PrintFun["Determining gaps in the data ..."];  

244 (* whatever is non-numeric is assumed as a known gap *)  

245 presentDataIndices = Flatten[Position[expData, {_?(NumericQ[#] &)  

246 , ___}]];  

247 (* some indices omitted here based on the excludeDataIndices  

248 argument *)  

249 presentDataIndices = Complement[presentDataIndices,  

250 excludeDataIndices];  

251  

252 solCompendium["simplifier"] = simplifier;  

253 solCompendium["excludeDataIndices"] = excludeDataIndices;  

254 solCompendium["startValues"] = startValues;  

255 solCompendium["freeIonSymbols"] = freeIonSymbols;  

256 solCompendium["truncationEnergy"] = truncationEnergy;  

257 solCompendium["numE"] = numE;  

258 solCompendium["expData"] = expData;  

259 solCompendium["problemVars"] = problemVars;  

260 solCompendium["maxIterations"] = maxIterations;  

261 solCompendium["hamDim"] = hamDim;  

262 solCompendium["constraints"] = constraints;  

263  

264 modelSymbols = Sort[Select[paramSymbols, Not[MemberQ[Join[  

265 racahSymbols, juddOfeltIntensitySymbols, chenSymbols, {t2Switch, \[Epsilon],  

266 gs, nE}], #]] &]];(* remove the symbols that will be removed by the simplifier, no  

267 symbol should remain here that is not in the symbolic Hamiltonian *)  

268 reducedModelSymbols = Select[modelSymbols, Not[MemberQ[Keys[  

269 simplifier], #]] &];  

270  

271 (* this is useful to understand what are the arguments of the  

272 truncated compiled Hamiltonian *)  

273 If[OptionValue["SignatureCheck"],  

274     (  

275         PrintFun["Given the model parameters and the simplifying  

276 assumptions, the resultant model parameters are:"];  

277         PrintFun[{reducedModelSymbols}];  

278         PrintFun["Exiting ..."];  

279         Return[""];  

280     )  

281 
```

```

269 ];
270
271 (* calculate the basis *)
272 PrintFun["Retrieving the LSJM basis for f^", numE, " ..."];
273 basis = BasisLSJM[numE];
274
275 Which[refParamsVintage === Automatic,
276 (
277   PrintFun["Using the automatic vintage with freshly fitted
278 free-ion parameters and others as in LaF3 ..."];
279   lnParams = LoadLaF3Parameters[ln];
280   freeIonSol = FreeIonSolver[expData, numE];
281   freeIonParams = freeIonSol["bestParams"];
282   lnParams = Join[lnParams, freeIonParams];
283 ), MemberQ[{List, Association}, Head[RefParams]],
284 (
285   RefParams = Association[RefParams];
286   PrintFun["Using the given parameters as a starting point ..."]
287 ];
288   lnParams = RefParams;
289   extraParams = LoadLaF3Parameters[ln];
290   lnParams = Join[extraParams, lnParams];
291 ), True,
292 (
293   (* get the reference parameters from the given vintage *)
294   PrintFun["Getting reference free-ion parameters for ", ln, " "
295 using ", refParamsVintage, " ..."];
296   lnParams = ParamPad[RefParams[ln], "PrintFun" -> PrintFun];
297 )
298 ];
299 freeBies = Prepend[Values[(# -> (#/.lnParams)) &/@ freeIonSymbols], numE];
300 (* a more explicit alias *)
301 allVars = reducedModelSymbols;
302 numericConstraints = Association@Select[constraints, NumericQ
303 #[[2]] &];
304 standardValues = allVars /. Join[lnParams, numericConstraints];
305 solCompendium["allVars"] = allVars;
306 solCompendium["freeBies"] = freeBies;
307
308 (* reload compiled version if found *)
309 varHash = Hash[{numE, allVars, freeBies,
310 truncationEnergy, simplifier}];
311 compiledIntermediateFname = ln <> "-compiled-intermediate-
312 truncated-ham-" <> ToString[varHash] <> ".mx";
313 compiledIntermediateFname = FileNameJoin[{moduleDir, "compiled",
314 compiledIntermediateFname}];
315 solCompendium["compiledIntermediateFname"] =
316 compiledIntermediateFname;
317
318 If[FileExistsQ[compiledIntermediateFname],
319   PrintFun["This ion, free-ion params, and full set of variables
320 have been used before (as determined by {numE, allVars, freeBies,
321 truncationEnergy, simplifier}). Loading the previously saved
322 compiled function and intermediate coupling basis ..."];
323   PrintFun["Using : ", compiledIntermediateFname];
324   {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =
325 Import[compiledIntermediateFname];
326 (
327   If[truncationEnergy == Infinity,
328 (
329     ham = EffectiveHamiltonian[numE, "ReturnInBlocks" -> False
330 ];
331     theSimplifier = simplifier;
332     ham = Normal@ReplaceInSparseArray[ham, simplifier];
333     PrintFun["Compiling a function for the Hamiltonian with no
334 truncation ..."];
335   (* compile a function that will calculate the truncated
336 Hamiltonian given the parameters in allVars, this is the function
337 to be use in fitting *)
338   compileIntermediateTruncatedHam = Compile[Evaluate[allVars
339 ], Evaluate[ham]];
340   truncatedIntermediateBasis = SparseArray@IdentityMatrix[
341

```

```

326 Binomial[14, numE]];
327 (* save the compiled function *)
328 PrintFun["Saving the compiled function for the Hamiltonian
with no truncation and a placeholder intermediate basis ..."];
329 Export[compiledIntermediateFname, {
330 compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
331 ),
332 (
333 (* grab the Hamiltonian preserving the block structure *)
334 PrintFun["Assembling the Hamiltonian for f^", numE, " keeping
the block structure ..."];
335 ham = EffectiveHamiltonian[numE, "ReturnInBlocks" ->
336 True];
337 (* apply the simplifier *)
338 PrintFun["Simplifying using the aggregate set of
simplification rules ..."];
339 ham = Map[ReplaceInSparseArray[#, simplifier] &, ham,
{2}];
340 PrintFun["Zeroing out every symbol in the Hamiltonian that is
not a free-ion parameter ..."];
341 (* Get the free ion symbols *)
342 freeIonSimplifier = (# -> 0) & /@ Complement[
343 reducedModelSymbols, freeIonSymbols];
344 (* Take the diagonal blocks for the intermediate analysis *)
345 PrintFun["Grabbing the diagonal blocks of the Hamiltonian ...
"];
346 diagonalBlocks = Diagonal[ham];
347 (* simplify them to only keep the free ion symbols *)
348 PrintFun["Simplifying the diagonal blocks to only keep the
free ion symbols ..."];
349 diagonalScalarBlocks = ReplaceInSparseArray[#, freeIonSimplifier] & /@ diagonalBlocks;
350 (* these include the MJ quantum numbers, remove that *)
351 PrintFun["Contracting the basis vectors by removing the MJ
quantum numbers from the diagonal blocks ..."];
352 diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
353
354 argsOfTheIntermediateEigensystems = StringJoin[Riffle[
355 Prepend[(ToString[#] <> "v_") & /@ freeIonSymbols, "numE_"], ", "]];
356 argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle[
357 ((ToString[#] <> "v") & /@ freeIonSymbols), ", "]];
358 PrintFun["argsOfTheIntermediateEigensystems = ",
359 argsOfTheIntermediateEigensystems];
360 PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
361 argsForEvalInsideOfTheIntermediateSystems];
362 PrintFun["(if the following fails, it might help to see if
the arguments of TheIntermediateEigensystems match the ones shown
above)"];
363
364 (* compile a function that will effectively calculate the
spectrum of all of the scalar blocks given the parameters of the
free-ion part of the Hamiltonian *)
365 (* compile one function for each of the blocks *)
366 PrintFun["Compiling functions for the diagonal blocks of the
Hamiltonian ..."];
367 compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate
368 [N[Normal[#]]] & /@ diagonalScalarBlocks];
369 (* use that to create a function that will calculate the free
-ion eigensystem *)
370 TheIntermediateEigensystems[numEv_, F0v_, F2v_, F4v_, F6v_,  $\zeta$ 
371 v_] := (
372 theNumericBlocks = (#[F0v, F2v, F4v, F6v,  $\zeta$ v] &) /@
373 compiledDiagonal;
374 theIntermediateEigensystems = Eigensystem /@
375 theNumericBlocks;
376 Js = AllowedJ[numEv];
377 basisJ = BasisLSJMJ[numEv, "AsAssociation" -> True];
378 (* having calculated the eigensystems with the removed
degeneracies, put the degeneracies back in explicitly *)
379 elevatedIntermediateEigensystems = MapIndexed[EigenLever
380 [#1, 2Js[[#2[[1]]]] + 1] &, theIntermediateEigensystems];
381 (* Identify a single MJ to keep *)
382 pivot = If[EvenQ[numEv], 0, -1/2];
383 LSJmultiplets = (#[[1]] <> ToString[InputForm[#[[2]]]]) & /
384 @Select[BasisLSJMJ[numEv], #[[{-1}]] == pivot &];
385 (* calculate the multiplet assignments that the

```

```

intermediate basis eigenvectors have *)
372    needlePosition = 0;
373    multipletAssignments = Table[
374      (
375        J = Js[[idx]];
376        eigenVecs = theIntermediateEigensystems[[idx]][[2]];
377        majorComponentIndices = Ordering[Abs[#]][[-1]] &/
378        @eigenVecs;
379        majorComponentIndices += needlePosition;
380        needlePosition += Length[
381          majorComponentIndices];
382        majorComponentAssignments = LSJmultiplets[[#]] &/
383        @majorComponentIndices;
384        (* All of the degenerate eigenvectors belong to the
385         same multiplet*)
386        elevatedMultipletAssignments = ListRepeater[
387          majorComponentAssignments, 2J+1];
388        elevatedMultipletAssignments
389        ),
390        {idx, 1, Length[Js]}
391      ];
392      (* put together the multiplet assignments and the energies
393       *)
394      freeIenergiesAndMultiplets = Transpose /@ Transpose[{First /
395      @elevatedIntermediateEigensystems, multipletAssignments}];
396      freeIenergiesAndMultiplets = Flatten[
397        freeIenergiesAndMultiplets, 1];
398      (* calculate the change of basis matrix using the
399       intermediate coupling eigenvectors *)
400      basisChanger = BlockDiagonalMatrix[Transpose /@ Last /
401      @elevatedIntermediateEigensystems];
402      basisChanger = SparseArray[basisChanger];
403      Return[{theIntermediateEigensystems, multipletAssignments,
404      elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
405      basisChanger}]
406    );
407
408    PrintFun["Calculating the intermediate eigensystems for ", ln,
409    " using free-ion params from LaF3 ..."];
410    (* calculate intermediate coupling basis using the free-ion
411     params for LaF3 *)
412    {theIntermediateEigensystems, multipletAssignments,
413     elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
414     basisChanger} = TheIntermediateEigensystems @@ freeBies;
415
416    (* use that intermediate coupling basis to compile a function
417     for the full Hamiltonian *)
418    allFreeEnergies = Flatten[First /
419    @elevatedIntermediateEigensystems];
420    (* important that the intermediate coupling basis have
421     attached energies, which make possible the truncation *)
422    ordering = Ordering[allFreeEnergies];
423    (* sort the free ion energies and determine which indices
424     should be included in the truncation *)
425    allFreeEnergiesSorted = Sort[allFreeEnergies];
426    {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];
427    (* determine the index at which the energy is equal or larger
428     than the truncation energy *)
429    sortedTruncationIndex = Which[
430      truncationEnergy > (maxFreeEnergy - minFreeEnergy),
431      hamDim,
432      True,
433      FirstPosition[allFreeEnergiesSorted - Min[
434        allFreeEnergiesSorted], x_ /; x > truncationEnergy, {0}, 1][[1]]
435    ];
436    (* the actual energy at which the truncation is made *)
437    roundedTruncationEnergy = allFreeEnergiesSorted[[[
438      sortedTruncationIndex]]];
439
440    (* the indices that participate in the truncation *)
441    truncationIndices = ordering[[;; sortedTruncationIndex]];
442    PrintFun["Computing the block structure of the change of
443     basis array ..."];
444    blockSizes = BlockArrayDimensionsArray[ham];
445    basisChangerBlocks = ArrayBlocker[basisChanger, blockSizes];
446    blockShifts = First /@ Diagonal[blockSizes];

```

```

423      numBlocks          = Length[blockSizes];
424      (* using the ham (with all the symbols) change the basis to
425       the computed one *)
426      PrintFun["Changing the basis of the Hamiltonian to the
427       intermediate coupling basis ..."];
428      intermediateHam = BlockMatrixMultiply[ham, basisChangerBlocks
429      ];
430      PrintFun["Distributing products inside of symbolic matrix
431       elements to keep complexity in check ..."];
432      Do[
433        intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
434        intermediateHam[[rowIdx, colIdx]], Distribute /@ # &,
435        {rowIdx, 1, numBlocks},
436        {colIdx, 1, numBlocks}
437      ];
438      intermediateHam = BlockMatrixMultiply[BlockTranspose[
439        basisChangerBlocks], intermediateHam];
440      PrintFun["Distributing products inside of symbolic matrix
441       elements to keep complexity in check ..."];
442      Do[
443        intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
444        intermediateHam[[rowIdx, colIdx]], Distribute /@ # &,
445        {rowIdx, 1, numBlocks},
446        {colIdx, 1, numBlocks}
447      ];
448      (* using the truncation indices truncate that one *)
449      PrintFun["Truncating the Hamiltonian ..."];
450      truncatedIntermediateHam = TruncateBlockArray[intermediateHam
451      , truncationIndices, blockShifts];
452      (* these are the basis vectors for the truncated hamiltonian
453      *)
454      PrintFun["Saving the truncated intermediate basis ..."];
455      truncatedIntermediateBasis = basisChanger[[All,
456      truncationIndices]];
457
458      PrintFun["Compiling a function for the truncated Hamiltonian
459      ..."];
460      (* compile a function that will calculate the truncated
461       Hamiltonian given the parameters in allVars, this is the function
462       to be use in fitting *)
463      compileIntermediateTruncatedHam = Compile[Evaluate[allVars],
464      Evaluate[truncatedIntermediateHam]];
465      (* save the compiled function *)
466      PrintFun["Saving the compiled function for the truncated
467       Hamiltonian and the truncated intermediate basis ..."];
468      Export[compiledIntermediateFname, {
469      compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
470
471      truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
472      PrintFun["The truncated Hamiltonian has a dimension of ",
473      truncationUmbral, "x", truncationUmbral, "..."];
474      presentDataIndices = Select[presentDataIndices, # <=
475      truncationUmbral &];
476      solCompendium["presentDataIndices"] = presentDataIndices;
477
478      (* the problemVars are the symbols that will be fitted for *)
479
480      PrintFun["Starting up the fitting process using the Levenberg-
481       Marquardt method ..."];
482      (* using the problemVars I need to create the argument list
483       including _?NumericQ *)
484      problemVarsQ      = (ToString[#] <> "_?NumericQ") & /@
485      problemVars;
486      problemVarsQString = StringJoin[Riffle[problemVarsQ, ", "]];
487      (* we also need to have the padded arguments with the variables
488       in the right position and the fixed values in the remaining ones
489       *)
490      problemVarsPositions = Position[allVars, #][[1, 1]] & /@
491      problemVars;
492      problemVarsString   = StringJoin[Riffle[ToString /@ problemVars,
493        ", "]];
494      (* to feed parameters to the Hamiltonian, which includes all

```

```

parameters, we need to form the set of arguments, with fixed
values where needed, and the variables in the right position *)
473 varsWithConstants = standardValues;
474 varsWithConstants[[problemVarsPositions]] = problemVars;
475 varsWithConstantsString = ToString[
476 varsWithConstants];
477
(* this following function serves eigenvalues from the
Hamiltonian, has memoization so it might grow to use a lot of RAM
*)
478 Clear[HamSortedEigenvalues];
479 hamEigenvaluesTemplate = StringTemplate["
HamSortedEigenvalues['problemVarsQ']:=(
480     ham          = compileIntermediateTruncatedHam@@'
481 varsWithConstants';
482     eigenValues = Chop[Sort@Eigenvalues@ham];
483     eigenValues = eigenValues - Min[eigenValues];
484     HamSortedEigenvalues['problemVarsString'] = eigenValues;
485     Return[eigenValues]
486 )"];
487 hamString = hamEigenvaluesTemplate[<|
488     "problemVarsQ"      -> problemVarsQString,
489     "varsWithConstants" -> varsWithConstantsString,
490     "problemVarsString" -> problemVarsString
491     |>];
492 ToExpression[hamString];
493
(* we also need a function that will pick the i-th eigenvalue,
this seems unnecessary but it's needed to form the right
functional form expected by the Levenberg-Marquardt method *)
494 eigenvalueDispenserTemplate = StringTemplate["
PartialHamEigenvalues['problemVarsQ'][i_]:=(
495     eigenVals = HamSortedEigenvalues['problemVarsString'];
496     eigenVals[[i]]
497 )
498 ];
499 eigenValueDispenserString = eigenvalueDispenserTemplate[<|
500     "problemVarsQ"      -> problemVarsQString,
501     "problemVarsString" -> problemVarsString
502     |>];
503 ToExpression[eigenValueDispenserString];
504
PrintFun["Determining the free variables after constraints ..."];
505 constrainedProblemVars = (problemVars /. constraints);
506 constrainedProblemVarsList = Variables[constrainedProblemVars];
507 If[addShift,
508     PrintFun["Adding a constant shift to the fitting parameters ...
509 "];
510     constrainedProblemVarsList = Append[constrainedProblemVarsList,
511     \[Epsilon]];
512 ];
513
514 indepVars = Complement[problemVars, #[[1]] & /@ constraints];
515 stringPartialVars = ToString/@constrainedProblemVarsList;
516
517 paramSols = {};
518 rmsHistory = {};
519 steps = 0;
520 problemVarsWithStartValues = KeyValueMap[{#1, #2} &, startValues];
521 If[addShift,
522     problemVarsWithStartValues = Append[problemVarsWithStartValues,
523     {\[Epsilon], 0}];
524 ];
525 openNotebooks = If[runningInteractive,
526     ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks
527     [],
528     {}];
529 If[Not[MemberQ[openNotebooks, "Solver Progress"]] && OptionValue["
530 ProgressView"],
531     ProgressNotebook["Basic" -> False]
532 ];
533 degressOfFreedom = Length[presentDataIndices] - Length[
534 problemVars] - 1;
535 PrintFun["Fitting for ", Length[presentDataIndices], " data
536 points with ", Length[problemVars], " free parameters.", " The
537 effective degrees of freedom are ", degressOfFreedom, " ..."];

```

```

533 PrintFun["Fitting model to data ..."];
534 startTime = Now;
535 shiftToggle = If[addShift, 1, 0];
536 sol = FindMinimum[
537   Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
538     {j, presentDataIndices}],
539   problemVarsWithStartValues,
540   Method -> "LevenbergMarquardt",
541   MaxIterations -> OptionValue["MaxIterations"],
542   AccuracyGoal -> OptionValue["AccuracyGoal"],
543   StepMonitor :> (
544     steps += 1;
545     currentSqSum = Sum[(expData[[j]][[1]] - (
546       PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle *
547       \[Epsilon])^2, {j, presentDataIndices}];
548     currentRMS = Sqrt[currentSqSum / degreesOfFreedom];
549     paramSols = AddToList[paramSols, constrainedProblemVarsList,
550     maxHistory];
551     rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
552   )
553 ];
554 endTime = Now;
555 timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
556 PrintFun["Solution found in ", timeTaken, "s"];
557
558 solVec = constrainedProblemVars /. sol[[-1]];
559 indepSolVec = indepVars /. sol[[-1]];
560 If[addShift,
561   \[Epsilon]Best = \[Epsilon]/. sol[[-1]],
562   \[Epsilon]Best = 0
563 ];
564 fullSolVec = standardValues;
565 fullSolVec[[problemVarsPositions]] = solVec;
566 PrintFun["Calculating the truncated numerical Hamiltonian
corresponding to the solution ..."];
567 fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
568 PrintFun["Calculating energies and eigenvectors ..."];
569 {eigenEnergies, eigenVectors} = Eigensystem[fullHam];
570 states = Transpose[{eigenEnergies, eigenVectors}];
571 states = SortBy[states, First];
572 eigenEnergies = First /@ states;
573 PrintFun["Shifting energies to make ground state zero of energy
..."];
574 eigenEnergies = eigenEnergies - eigenEnergies[[1]];
575 PrintFun["Calculating the linear approximant to each eigenvalue
..."];
576 allVarsVec = Transpose[{allVars}];
577 p0 = Transpose[{fullSolVec}];
578 linMat = {};
579 If[addShift,
580   tail = -2,
581   tail = -1];
582 Do[
583   (
584     aVarPosition = Position[allVars, aVar][[1, 1]];
585     isolationValues = ConstantArray[0, Length[allVars]];
586     isolationValues[[aVarPosition]] = 1;
587     dependentVars = KeyValueMap[{#1, D[#2, aVar]} &, Association[
588       constraints]];
589     Do[
590       isolationValues[[Position[allVars, dVar[[1]]][[1, 1]]]] =
591       dVar[[2]],
592       {dVar, dependentVars}
593     ];
594     perHam = compileIntermediateTruncatedHam @@ isolationValues;
595     lin = FirstOrderPerturbation[Last /@ states, perHam];
596     linMat = Append[linMat, lin];
597   ),
598   {aVar, constrainedProblemVarsList[;;tail]}
599 ];
599 PrintFun["Removing the gradient of the ground state ..."];
599 linMat = (# - #[[1]] & /@ linMat);
599 PrintFun["Transposing derivative matrices into columns ..."];
599 linMat = Transpose[linMat];

```

```

600 PrintFun["Calculating the eigenvalue vector at solution ..."];
601 \[Lambda]0Vec = Transpose[{eigenEnergies[[presentDataIndices]]}];
602 PrintFun["Putting together the experimental vector ..."];
603 \[Lambda]exp = Transpose[{First /@ expData[[presentDataIndices]]}];
604 problemVarsVec = If[addShift,
605   Transpose[{constrainedProblemVarsList[[;; -2]]}],
606   Transpose[{constrainedProblemVarsList}]];
607 ];
608 indepSolVecVec = Transpose[{indepSolVec}];
609 PrintFun["Calculating the difference between eigenvalues at
610 solution ..."];
611 diff = If[linMat == {},
612   (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best,
613   (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best + linMat[[presentDataIndices]].(problemVarsVec - indepSolVecVec)];
614 ];
615 PrintFun["Calculating the sum of squares of differences around
616 solution ... "];
617 sqdiff = Expand[(Transpose[diff] . diff)[[1, 1]]];
618 PrintFun["Calculating the minimum (which should coincide with sol
619 ) ..."];
620 minpoly = sqdiff /. AssociationThread[problemVars -> solVec];
621 fmSolAssoc = Association[sol[[2]]];
622 If[\[Sigma]exp == Automatic,
623   \[Sigma]exp = Sqrt[minpoly / degressOfFreedom];
624 ];
625 \[CapitalDelta]\[Chi]2 = Sqrt[degressOfFreedom];
626 Amat = (1/\[Sigma]exp^2) * Transpose[linMat[[presentDataIndices]]].linMat[[presentDataIndices]];
627 paramIntervals = EllipsoidBoundingBox[Amat, \[CapitalDelta]\[Chi]2];
628 PrintFun["Calculating the uncertainty in the parameters ..."];
629 solWithUncertainty = Table[
630   (
631     aVar = constrainedProblemVarsList[[varIdx]];
632     paramBest = aVar /. fmSolAssoc;
633     (aVar -> {paramBest, paramIntervals[[varIdx, 2]]})
634   ),
635   {varIdx, 1, Length[constrainedProblemVarsList]-shiftToggle}
636 ];
637 bestRMS = Sqrt[minpoly / degressOfFreedom];
638 bestParams = sol[[2]];
639 bestWithConstraints = Association@Join[constraints, bestParams];
640 bestWithConstraints = bestWithConstraints /. bestWithConstraints;
641 bestWithConstraints = (# + 0.) & /@ bestWithConstraints;
642 solCompendium["degreesOfFreedom"] = degressOfFreedom;
643 solCompendium["solWithUncertainty"] = solWithUncertainty;
644 solCompendium["truncatedDim"] = truncationUmbral;
645 solCompendium["fittedLevels"] = Length[presentDataIndices];
646 solCompendium["actualSteps"] = steps;
647 solCompendium["bestRMS"] = bestRMS;
648 solCompendium["problemVars"] = problemVars;
649 solCompendium["paramSols"] = paramSols;
650 solCompendium["rmsHistory"] = rmsHistory;
651 solCompendium["Appendix"] = OptionValue["AppendToFile"];
652 solCompendium["timeTaken/s"] = timeTaken;
653 solCompendium["bestParams"] = bestParams;
654 solCompendium["bestParamsWithConstraints"] = bestWithConstraints;
655
656 If[OptionValue["SaveEigenvectors"],
657   solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]}
658   &/@ (Chop /@ ShiftedLevels[states]),
659   (
660     finalEnergies = Sort[First /@ states];
661     finalEnergies = finalEnergies - finalEnergies[[1]];
662     finalEnergies = finalEnergies + \[Epsilon]Best;
663     finalEnergies = Chop /@ finalEnergies;
664     solCompendium["energies"] = finalEnergies;
665   )
666 ];

```

```

665 ];
666 If[OptionValue["SaveToLog"],
667   PrintFun["Saving the solution to the log file ..."];
668   LogSol[solCompendium, logFilePrefix];
669 ];
670 PrintFun["Finished ..."];
671 Return[solCompendium];
672 )
673 ];

```

## 9 Accompanying notebooks

The code for this dissertation is accompanied by the following auxiliary *Mathematica* notebooks, which either document the functions included in the code, or serve as aids in the calculation of matrix elements. These notebooks assume that the **paclet** has been installed according to the instructions given in [Section 16](#).

- [/notebooks/qlanth - Table Generator.nb](#): generates the basic tables on which every calculation is based. This means that LS-reduced matrix elements are used to calculate matrix elements in the  $|LSJM_J\rangle$  basis.
- [/notebooks/qlanth - JJBlock Calculator.nb](#): can be used to generate the J-J' blocks for the different interactions. The data files produced here are necessary for **EffectiveHamiltonian** to work. These blocks are created by putting together matrix elements from different interactions.
- [/notebooks/The Lanthanides in LaF3.nb](#): runs **qlanth** over the lanthanide ions in LaF<sub>3</sub> and compares the results against the published values from Carnall *et al.* [[Car+89](#)]. It also calculates magnetic dipole transition rates and oscillator strengths.

## 10 Compiled data for LaF<sub>3</sub>:Ln<sup>3+</sup> and LiYF<sub>4</sub>:Ln<sup>3+</sup>

The study of Carnall *et al.* [[Car+89](#)] on lanthanum fluoride was a systematic review of trivalent lanthanide ions in LaF<sub>3</sub>. In this work they fitted the experimental data for all of the lanthanide ions using the single-configuration effective Hamiltonian. In their appendices one can find their calculated values, together with the experimental values that they used for their least squares fittings. In **qlanth** this data can be accessed by invoking the command **LoadCarnall** which brings into the session an association that has keys that have as values the tables and appendices from this article. [Fig. 6](#) shows the results of a calculation done with **qlanth** for the energy levels in LaF<sub>3</sub>. Additionally the function **LoadLaF3Parameters** can be used to query the data for the fitted parameters, which may serve as a useful starting point for the description of the lanthanides ions in hosts other than LaF<sub>3</sub>.

Similarly, Cheng *et al.* [[Che+16](#)] compiled data for LiYF<sub>4</sub>. In **qlanth** model parameters for LiYF<sub>4</sub> can be obtained by calling the function **LoadLiYF4Parameters**. [Fig. 7](#) shows the results of a calculation done with **qlanth** for the energy levels in LiYF<sub>4</sub>.

```

1 Carnall::usage = "Association of data from Carnall et al (1989) with
  the following keys: {data, annotations, paramSymbols, elementNames,
  rawData, rawAnnotations, annotatedData, appendix:Pr:Association
  , appendix:Pr:Calculated, appendix:Pr:RawTable, appendix:Headings}
  ";

```

```

1 LoadCarnall::usage = "LoadCarnall[] loads data for trivalent
  lanthanides in LaF3 using the data from Bill Carnall's 1989 paper.
  ";
2 LoadCarnall[] := (
3   If[ValueQ[Carnall], Return[]];
4   carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"]];
5   If[!FileExistsQ[carnallFname],
6     (PrintTemporary[">> Carnall.m not found, generating ..."];
7      Carnall = ParseCarnall[];
8    ),
9    Carnall = Import[carnallFname];
10  ];
11 );

```

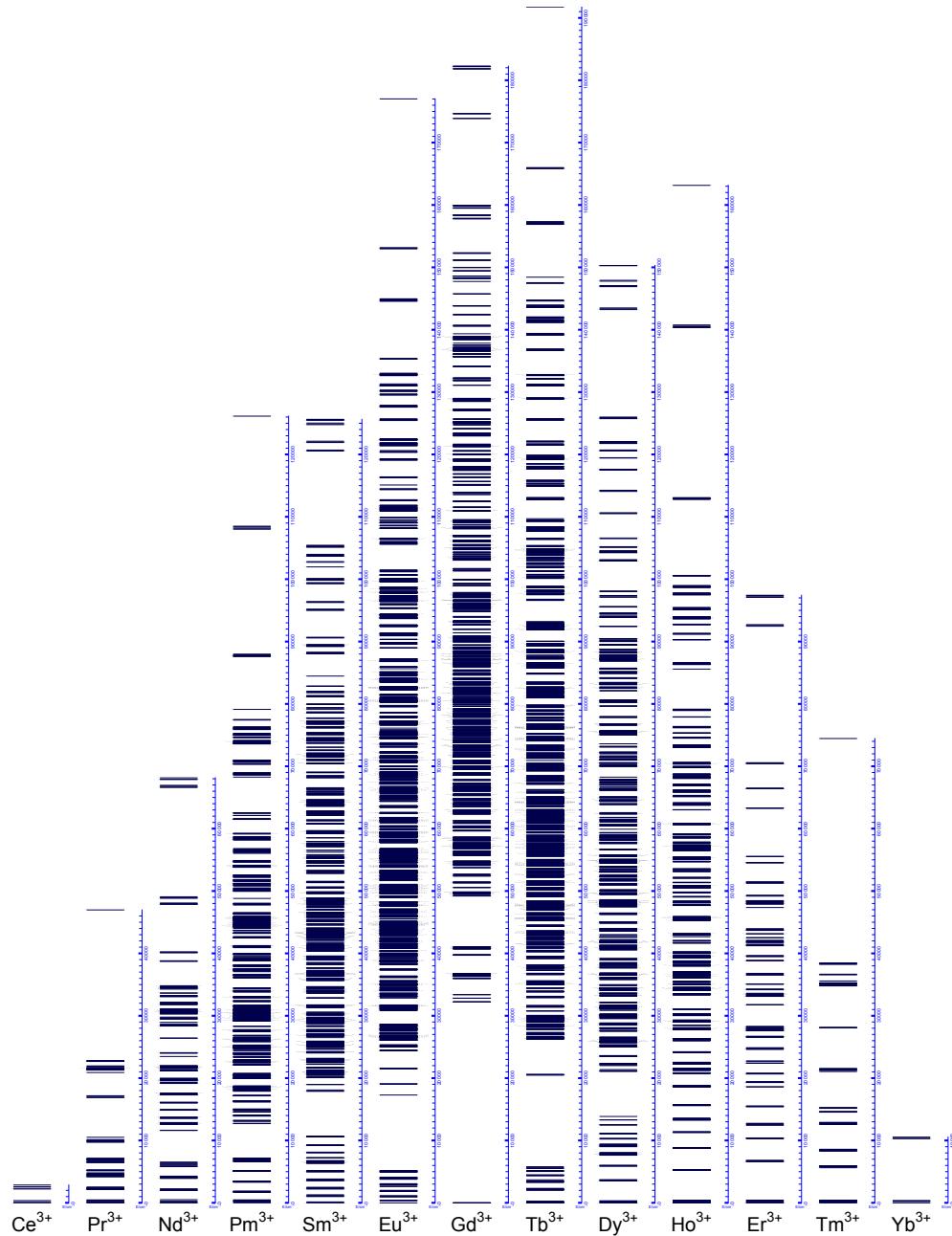


Figure 6: Energy levels in  $\text{LaF}_3$ .

```

1 LoadLaF3Parameters::usage="LoadLaF4Parameters[ln] gives the models
  for the semi-empirical Hamiltonian for the trivalent lanthanide
  ion with symbol ln.";
2 Options[LoadLaF3Parameters] = {
3   "Vintage" -> "Standard",
4   "With Uncertainties" -> False
5 };
6 LoadLaF3Parameters[Ln_String, OptionsPattern[]]:= Module[
7   {paramData, params},
8   (
9     paramData = Which[
10       OptionValue["Vintage"] == "Carnall",
11       Import[FileNameJoin[{moduleDir, "data", "carnallParams.m"}]],
12       OptionValue["Vintage"] == "Standard",
13       Import[FileNameJoin[{moduleDir, "data", "standardParams.m"}]]
14     ];
15     params = If[OptionValue["With Uncertainties"],
16       paramData[Ln],
17       If[Head[#] === Around, #[["Value"]], #] & /@ paramData[Ln]
18     ];
19     Return[params];
20   )
21 ];

```

```

1 LoadLiYF4Parameters::usage="LoadLiYF4Parameters[ln] takes a string
  with the symbol the element of a trivalent lanthanide ion and
  returns model parameters for it. It return the data for LiYF4 from

```

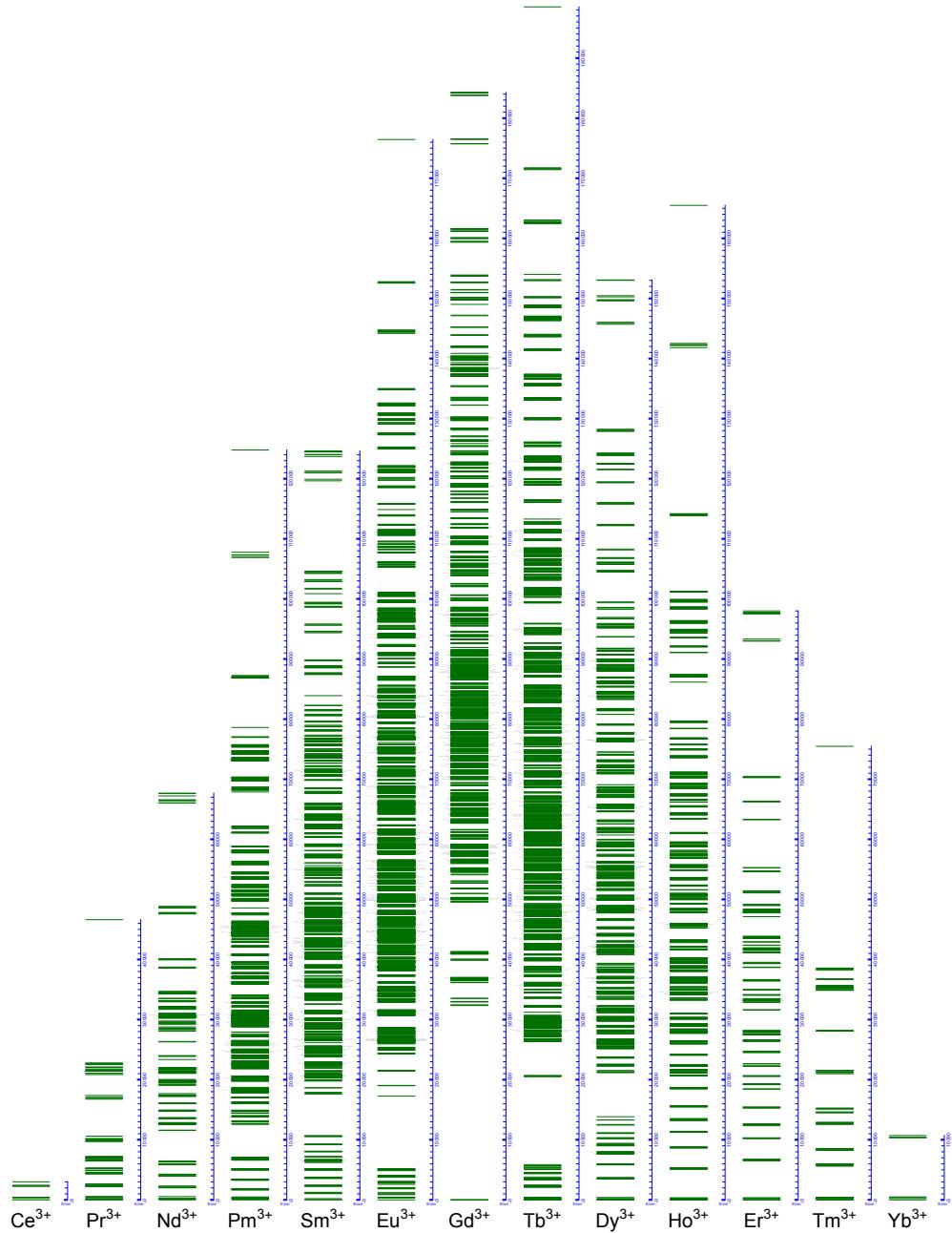


Figure 7: Energy levels in  $\text{LiYF}_4$ .

```

Cheng et al.";
2 LoadLiYF4Parameters[ln_, OptionsPattern[]]:=(
3   If[!ValueQ[paramsLiYF4],
4     paramsChengLiYF4 = Import[FileNameJoin[{moduleDir,"data",
5       "chengLiYF4.m"}]];
6   ];
7 )
```

## 11 sparsefn.py

`qlanth` is also accompanied by seven Python scripts `sparsefn[1-7].py`. Each of these contains a function `effective_hamiltonian_f[1-7]` which returns a sparse array (using this data structure as provided by `scipy`) 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  $f^7$ . These include changing the signs of all in `&qn=18` and setting `t2Switch` to 0. For configurations at or below  $f^7$  it is necessary to set `t2Switch` to 1.

## 12 Data sources

The data (and their provenance) upon which **qlanth** bases its calculations is the following:

- Coefficients of fractional parentage and seniority numbers from Velkov [Vel00].
- Terms labels from  $f^1$  to  $f^7$  from Nielson and Koster [NK63].
- 3j-symbol [Wol24b] and 6j-symbol [Wol24a] values from *Mathematica* (v 13.2),
- Reduced matrix elements for the three body operators from Judd [JS84].
- Reduced matrix elements for the magnetic interactions from Judd [JCC68].

## 13 Other details

- Fitting the experimental data for the entire row might take about 45 minutes, if run for the first time, but takes much less time once compiled functions from the truncated (or not truncated) Hamiltonian have been saved to disk.
- The code was run in *Mathematica* version 14.1 on MacOS Sequoia 15.2.

## 14 Units

Following the tradition of the spectroscopic community, all the matrix elements of the Hamiltonian are calculated using the Kayser ( $\mathcal{K} \equiv \text{cm}^{-1}$ ) 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`.

```
1 BeginPackage["qonstants`"];
2
3 (* Spectroscopic niceties*)
4 theLanthanides = {"Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy",
5   "Ho", "Er", "Tm", "Yb"};
6 theActinides = {"Ac", "Th", "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk",
7   "Cf", "Es", "Fm", "Md", "No", "Lr"};
8 theTrivalents = {"Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho",
9   "Er", "Tm"};
10 specAlphabet = "SPDFGHIKLMNOQRTUV";
11 complementaryNumE = {1, 13, 2, 12, 3, 11, 4, 10, 5, 9, 6, 8, 7};
12
13 (* SI *)
14 hPlanck = 6.62607015 * 10^-34; (* Planck's constant in J s
15   *)
16 hBar = hPlanck / (2 \[Pi]); (* reduced Planck's constant
17   in J s *)
18 \[Mu]B = 9.2740100783 * 10^-24; (* Bohr magneton in SI *)
19 me = 9.1093837015 * 10^-31; (* electron mass in kg *)
20 cLight = 2.99792458 * 10^8; (* speed of light in m/s *)
21 eCharge = 1.602176634 * 10^-19; (* elementary charge in SI *)
22 \[Epsilon]0 = 8.8541878128 * 10^-12; (* electric permittivity in
23   vacuum in SI *)
24 \[Mu]0 = 4 \[Pi] * 10^-7; (* magnetic permeability in
25   vacuum in SI *)
26 \[Alpha]Fine = 1/137.036; (* fine structure constant *)
27
28 bohrRadius = 5.29177*10^-11; (* Bohr radius in m *)
29 hartreeEnergy = hBar^2 / (me * bohrRadius^2); (* Hartree energy in J
30   *)
31 hartreeTime = hBar / hartreeEnergy; (* Hartree time in s *)
32
33 (* Hartree atomic units *)
34 hPlanckHartree = 2 \[Pi]; (* Planck's constant in Hartree *)
35 meHartree = 1; (* electron mass in Hartree *)
36 cLightHartree = 137.036; (* speed of light in Hartree *)
37 eChargeHartree = 1; (* elementary charge in Hartree *)
38 \[Mu]0Hartree = \[Alpha]Fine^2; (* magnetic permeability in vacuum in
39   Hartree *)
40
41 (* some conversion factors *)
```

```

33 eVToJoule      = eCharge;
34 jouleToeV       = 1 / eVToJoule;
35 jouleToHartree = 1 / hartreeEnergy;
36 eVToKayser     = eCharge / ( hPlanck * cLight * 100 ); (* 1 eV =
  8065.54429 cm^-1 *)
37 kayserToeV    = 1 / eVToKayser;
38 teslaToKayser = 2 * \[Mu]B / hPlanck / cLight / 100;
39 kayserToHartree = kayserToeV * eVToJoule * jouleToHartree;
40 hartreeToKayser = 1 / kayserToHartree;
41
42 EndPackage [];

```

## 15 Notation

orbital angular momentum operator of a single electron

$$\hat{l} \quad (121)$$

total orbital angular momentum operator

$$\hat{L} \quad (122)$$

spin angular momentum operator of a single electron

$$\hat{s} \quad (123)$$

total spin angular momentum operator

$$\hat{S} \quad (124)$$

Shorthand for all other quantum numbers

$$\Lambda \quad (125)$$

orbital angular momentum number

$$\underline{\ell} \quad (126)$$

spinning angular momentum number

$$\underline{s} \quad (127)$$

Coulomb non-central potential

$$\hat{c} \quad (128)$$

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

$$\langle \Lambda LS \| \hat{O} \| \Lambda' L' S' \rangle \quad (129)$$

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

$$\langle \Lambda LSJ \| \hat{O} \| \Lambda' L' S' J' \rangle \quad (130)$$

Spectroscopic term  $\alpha LS$  in Russel-Saunders notation

$$^{2S+1}\alpha L \equiv |\alpha LS\rangle \quad (131)$$

spherical tensor operator of rank k

$$\hat{X}^{(k)} \quad (132)$$

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

$$\hat{X}_q^{(k)} \quad (133)$$

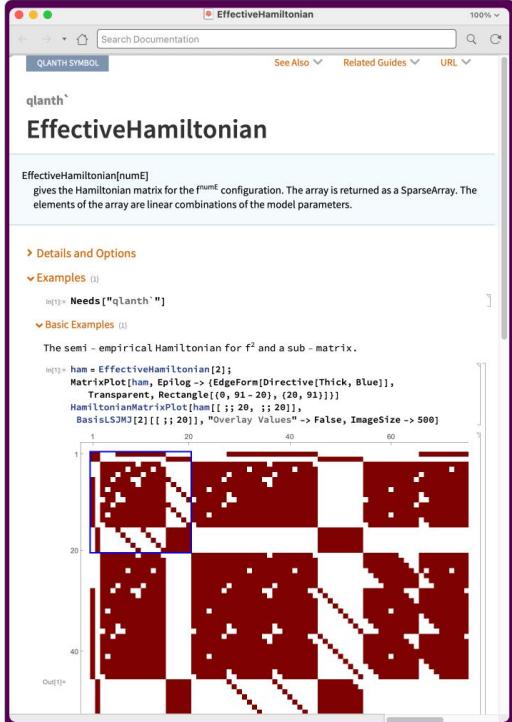
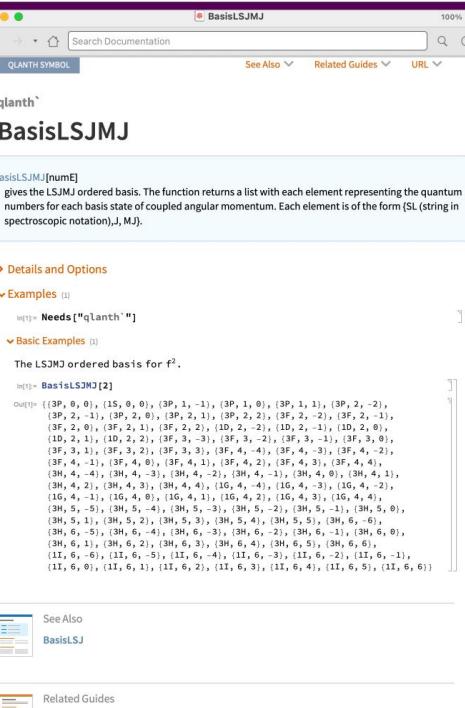
The coefficient of fractional parentage from the parent term  $|\underline{\ell}^{n-1} \alpha' L' S'\rangle$  for the daughter term  $|\underline{\ell}^n \alpha LS\rangle$

$$\left( \underline{\ell}^{n-1} \alpha' L' S' \right) \left\{ \underline{\ell}^n \alpha LS \right\} \quad (134)$$

## 16 Mathematica paclet

The simplest way of adding **qlanth** to *Mathematica* is by installing the corresponding paclet. To do this, first download the **paclet** file for the **latest release** of the code on GitHub. Install the **paclet** file using **PacletInstall** within a *Mathematica* session. After this **qlanth** may be loaded into a notebook by issuing the command **Needs["qlanth`"]**.

The paclet includes documentation in standard *Mathematica* format (see [Fig-8a](#)).

(a) `qlanth` guide in *Mathematica*(b) Documentation for `EffectiveHamiltonian`(c) Doc. for `BasisLSJMJ`(d) Doc. for `ToPythonSymPyExpression`

## 17 Definitions

$$\overline{[x]} := \begin{cases} \text{two plus one} & \text{if } x = 2 \\ 2x + 1 & \text{otherwise} \end{cases} \quad (135)$$

irreducible unit tensor operator of rank k

$$\overline{\hat{u}^{(k)}} \quad (136)$$

symmetric unit tensor operator for n equivalent electrons

$$\overline{\hat{U}^{(k)}} := \sum_{i=1}^n \overline{\hat{u}^{(k)}} \quad (137)$$

Renormalized spherical harmonics

$$\overline{\mathcal{C}_q^{(k)}} := \sqrt{\frac{4\pi}{2k+1}} \overline{Y_q^{(k)}} \quad (138)$$

Triangle “delta” between  $j_1, j_2, j_3$

$$\overline{\triangle(j_1, j_2, j_3)} := \begin{cases} 1 & \text{if } j_1 = (j_2 + j_3), (j_2 + j_3 - 1), \dots, |j_2 - j_3| \\ 0 & \text{otherwise} \end{cases} \quad (139)$$

## 18 code

### 18.1 qlanth.m

This file encapsulates the main functions in `qlanth` and contains all the physics related functions.

```
1 (* -----+
2 +-----+
3 |
4 |
5 |      / \    / \    / \    / \    / \    / \
6 |      / / \  / / \  / / \  / / \  / / \  / \
7 |      \_ , / \_ , / \_ , / \_ , / \_ , / \_ ,
8 |      / / / / / / / / / / / / / / / / / /
9 |      / / / / / / / / / / / / / / / / / /
10 |
11 |
12 +-----+
13 This code was initially authored by Christopher M. Dodson and Rashid
14 Zia, and then rewritten and expanded by Juan David Lizarazo Ferro in
15 the years 2022-2024 under the advisory of Dr. Rashid Zia. It has
16 also benefited from the discussions with Tharnier Puel at the
17 University of Iowa.
18
19 It grew out of a collaboration sponsored by the NSF (NSF
20 DMR-1922025) between the groups of Dr. Rashid Zia at Brown
21 University, the Quantum Engineering Laboratory at the University of
22 Pennsylvania led by Dr. Lee Bassett, and the group of Dr. Michael
23 Flatté at the University of Iowa.
24
25 It uses an effective Hamiltonian to describe the electronic
26 structure of lanthanide ions in crystals. This effective Hamiltonian
27 includes terms representing the following interactions/relativistic
28 corrections: spin-orbit, electrostatic repulsion, spin-spin, crystal
29 field, and spin-other-orbit.
30
31 The Hilbert space used in this effective Hamiltonian is limited to
32 single f^n configurations. The inaccuracy of this single
33 configuration description is partially compensated by the inclusion
34 of configuration interaction terms as parametrized by the Casimir
35 operators of SO(3), G(2), and SO(7), and by three-body effective
36 operators ti.
37
38 The parameters included in this model are listed in the string
39 paramAtlas.
40
41 The notebook "qlanth.nb" contains a gallery with many of the
42 functions included in this module with some simple use cases.
43
44 The notebook "The Lanthanides in LaF3.nb" is an example in which the
45 results from this code are compared against the published results by
46 Carnall et. al for the energy levels of lanthanide ions in crystals
47 of lanthanum trifluoride.
48
49 VERSION: SEPTEMBER 2024
50
51 REFERENCES:
52
53 + Condon, E U, and G Shortley. "The Theory of Atomic Spectra." 1935.
54
55 + Racah, Giulio. "Theory of Complex Spectra. II." Physical Review
56 62, no. 9-10 (November 1, 1942): 438-62.
57 https://doi.org/10.1103/PhysRev.62.438.
58
59 + Racah, Giulio. "Theory of Complex Spectra. III." Physical Review
60 63, no. 9-10 (May 1, 1943): 367-82.
61 https://doi.org/10.1103/PhysRev.63.367.
62
63 + Judd, B. R. "Optical Absorption Intensities of Rare-Earth Ions." Physical Review 127, no. 3 (August 1, 1962): 750-61.
64 https://doi.org/10.1103/PhysRev.127.750.
65
66 + Olfelt, GS. "Intensities of Crystal Spectra of Rare-Earth Ions." The Journal of Chemical Physics 37, no. 3 (1962): 511-20.
67
68
69
```

```

70 + Rajnak, K, and BG Wybourne. "Configuration Interaction Effects in
71 l^N Configurations." Physical Review 132, no. 1 (1963): 280.
72 https://doi.org/10.1103/PhysRev.132.280.
73
74 + Nielson, C. W., and George F Koster. "Spectroscopic Coefficients
75 for the p^n, d^n, and f^n Configurations", 1963.
76
77 + Wybourne, Brian. "Spectroscopic Properties of Rare Earths." 1965.
78
79 + Carnall, W To, PR Fields, and BG Wybourne. "Spectral Intensities
80 of the Trivalent Lanthanides and Actinides in Solution. I. Pr3+,
81 Nd3+, Er3+, Tm3+, and Yb3+." The Journal of Chemical Physics 42, no.
82 11 (1965): 3797-3806.
83
84 + Judd, BR. "Three-Particle Operators for Equivalent Electrons."
85 Physical Review 141, no. 1 (1966): 4.
86 https://doi.org/10.1103/PhysRev.141.4.
87
88 + Judd, BR, HM Crosswhite, and Hannah Crosswhite. "Intra-Atomic
89 Magnetic Interactions for f Electrons." Physical Review 169, no. 1
90 (1968): 130. https://doi.org/10.1103/PhysRev.169.130.
91
92 + (TASS) Cowan, Robert Duane. "The Theory of Atomic Structure and
93 Spectra." Los Alamos Series in Basic and Applied Sciences 3.
94 Berkeley: University of California Press, 1981.
95
96 + Judd, BR, and MA Suskin. "Complete Set of Orthogonal Scalar
97 Operators for the Configuration f^3." JOSA B 1, no. 2 (1984): 261-65.
98 https://doi.org/10.1364/JOSAB.1.000261.
99
100 + Carnall, W. T., G. L. Goodman, K. Rajnak, and R. S. Rana. "A
101 Systematic Analysis of the Spectra of the Lanthanides Doped into
102 Single Crystal LaF3." The Journal of Chemical Physics 90, no. 7
103 (1989): 3443-57. https://doi.org/10.1063/1.455853.
104
105 + Thorne, Anne, Ulf Litzen, and Sveneric Johansson. "Spectrophysics:
106 Principles and Applications." Springer Science & Business Media,
107 1999.
108
109 + Hansen, JE, BR Judd, and Hannah Crosswhite. "Matrix Elements of
110 Scalar Three-Electron Operators for the Atomic f-Shell." Atomic Data
111 and Nuclear Data Tables 62, no. 1 (1996): 1-49.
112 https://doi.org/10.1006/adnd.1996.0001.
113
114 + Velkov, Dobromir. "Multi-Electron Coefficients of Fractional
115 Parentage for the p, d, and f Shells." John Hopkins University,
116 2000. The B1F_ALL.TXT file is from this thesis.
117
118 + Dodson, Christopher M., and Rashid Zia. "Magnetic Dipole and
119 Electric Quadrupole Transitions in the Trivalent Lanthanide Series:
120 Calculated Emission Rates and Oscillator Strengths." Physical Review
121 B 86, no. 12 (September 5, 2012): 125102.
122 https://doi.org/10.1103/PhysRevB.86.125102.
123
124 + Hehlen, Markus P, Mikhail G Brik, and Karl W Kramer. "50th
125 Anniversary of the Judd-Ofelt Theory: An Experimentalist's View of
126 the Formalism and Its Application." Journal of Luminescence 136
127 (2013): 221-39.
128
129 + Rudzikas, Zenonas. Theoretical Atomic Spectroscopy, 2007.
130
131 + Benelli, Cristiano, and Dante Gatteschi. Introduction to Molecular
132 Magnetism: From Transition Metals to Lanthanides. John Wiley & Sons,
133 2015.
134 ----- *)
```

---

```

136 BeginPackage["qlanth`"];
137 Get[FileNameJoin[{DirectoryName[$InputFileName], "qconstants.m"}]]
138 Get[FileNameJoin[{DirectoryName[$InputFileName], "misc.m"}]]
139
140 paramAtlas = "
141 E0: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
142 E1: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
143 E2: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
144 E3: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
145
```

```

146  $\zeta$ : spin-orbit strength parameter.
147
148 F0: Direct Slater integral  $F^0$ , produces an overall shift of all
      energy levels.
149 F2: Direct Slater integral  $F^2$ 
150 F4: Direct Slater integral  $F^4$ , possibly constrained by ratio to  $F^2$ 
151 F6: Direct Slater integral  $F^6$ , possibly constrained by ratio to  $F^2$ 
152
153 M0: 0th Marvin integral
154 M2: 2nd Marvin integral
155 M4: 4th Marvin integral
156 \[Sigma]SS: spin-spin override, if 0 spin-spin is omitted, if 1 then
      spin-spin is included
157
158 T2: three-body effective operator parameter  $T^2$  (non-orthogonal)
159 T2p: three-body effective operator parameter  $T^{2'}$  (orthogonalized T2)
160 T3: three-body effective operator parameter  $T^3$ 
161 T4: three-body effective operator parameter  $T^4$ 
162 T6: three-body effective operator parameter  $T^6$ 
163 T7: three-body effective operator parameter  $T^7$ 
164 T8: three-body effective operator parameter  $T^8$ 
165
166 T11p: three-body effective operator parameter  $T^{11'}$  (orthogonalized
      T11)
167 T12: three-body effective operator parameter  $T^{12}$ 
168 T14: three-body effective operator parameter  $T^{14}$ 
169 T15: three-body effective operator parameter  $T^{15}$ 
170 T16: three-body effective operator parameter  $T^{16}$ 
171 T17: three-body effective operator parameter  $T^{17}$ 
172 T18: three-body effective operator parameter  $T^{18}$ 
173 T19: three-body effective operator parameter  $T^{19}$ 
174
175 P0: pseudo-magnetic parameter  $P^0$ 
176 P2: pseudo-magnetic parameter  $P^2$ 
177 P4: pseudo-magnetic parameter  $P^4$ 
178 P6: pseudo-magnetic parameter  $P^6$ 
179
180 gs: electronic gyromagnetic ratio
181
182  $\alpha$ : Trees' parameter  $\alpha$  describing configuration interaction via the
      Casimir operator of  $SO(3)$ 
183  $\beta$ : Trees' parameter  $\beta$  describing configuration interaction via the
      Casimir operator of  $G(2)$ 
184  $\gamma$ : Trees' parameter  $\gamma$  describing configuration interaction via the
      Casimir operator of  $SO(7)$ 
185
186 B02: crystal field parameter  $B_0^2$  (real)
187 B04: crystal field parameter  $B_0^4$  (real)
188 B06: crystal field parameter  $B_0^6$  (real)
189 B12: crystal field parameter  $B_1^2$  (real)
190 B14: crystal field parameter  $B_1^4$  (real)
191
192 B16: crystal field parameter  $B_1^6$  (real)
193 B22: crystal field parameter  $B_2^2$  (real)
194 B24: crystal field parameter  $B_2^4$  (real)
195 B26: crystal field parameter  $B_2^6$  (real)
196 B34: crystal field parameter  $B_3^4$  (real)
197
198 B36: crystal field parameter  $B_3^6$  (real)
199 B44: crystal field parameter  $B_4^4$  (real)
200 B46: crystal field parameter  $B_4^6$  (real)
201 B56: crystal field parameter  $B_5^6$  (real)
202 B66: crystal field parameter  $B_6^6$  (real)
203
204 S12: crystal field parameter  $S_1^2$  (real)
205 S14: crystal field parameter  $S_1^4$  (real)
206 S16: crystal field parameter  $S_1^6$  (real)
207 S22: crystal field parameter  $S_2^2$  (real)
208
209 S24: crystal field parameter  $S_2^4$  (real)
210 S26: crystal field parameter  $S_2^6$  (real)
211 S34: crystal field parameter  $S_3^4$  (real)
212 S36: crystal field parameter  $S_3^6$  (real)
213
214 S44: crystal field parameter  $S_4^4$  (real)
215 S46: crystal field parameter  $S_4^6$  (real)

```

```

216 S56: crystal field parameter S_5^6 (real)
217 S66: crystal field parameter S_6^6 (real)
218
219 \[Epsilon]: ground level baseline shift
220 t2Switch: controls the usage of the t2 operator beyond f7 (1 for f7
   or below, 0 for f8 or above)
221 wChErrA: If 1 then the type-A errors in Chen are used, if 0 then not.
222 wChErrB: If 1 then the type-B errors in Chen are used, if 0 then not.
223
224 Bx: x component of external magnetic field (in T)
225 By: y component of external magnetic field (in T)
226 Bz: z component of external magnetic field (in T)
227
228 \[CapitalOmega]2: Judd-Ofelt intensity parameter k=2 (in cm^2)
229 \[CapitalOmega]4: Judd-Ofelt intensity parameter k=4 (in cm^2)
230 \[CapitalOmega]6: Judd-Ofelt intensity parameter k=6 (in cm^2)
231
232 nE: number of electrons in a configuration
233
234 E0p: orthogonalized E0
235 E1p: orthogonalized E1
236 E2p: orthogonalized E2
237 E3p: orthogonalized E3
238 αp: orthogonalized α
239 βp: orthogonalized β
240 γp: orthogonalized γ
241 ";
242 paramSymbols = StringSplit[paramAtlas, "\n"];
243 paramSymbols = Select[paramSymbols, # != "" & ];
244 paramSymbols = ToExpression[StringSplit[#, ":"][[1]]] & /@ paramSymbols;
245 Protect /@ paramSymbols;
246
247 (* Parameter families *)
248 Unprotect[racahSymbols, chenSymbols, slaterSymbols, controlSymbols,
   cfSymbols, TSymbols, pseudoMagneticSymbols, marvinSymbols,
   casimirSymbols, magneticSymbols, juddOfeltIntensitySymbols,
   mostlyOrthogonalSymbols];
249 racahSymbols = {E0, E1, E2, E3};
250 chenSymbols = {wChErrA, wChErrB};
251 slaterSymbols = {F0, F2, F4, F6};
252 controlSymbols = {t2Switch, \[Sigma]SS};
253 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};
254 TSymbols = {T2, T2p, T3, T4, T6, T7, T8, T11p, T12, T14, T15,
   T16, T17, T18, T19};
255 pseudoMagneticSymbols = {P0, P2, P4, P6};
256 marvinSymbols = {M0, M2, M4};
257 magneticSymbols = {Bx, By, Bz, gs, ζ};
258 casimirSymbols = {α, β, γ};
259 juddOfeltIntensitySymbols = {\[CapitalOmega]2, \[CapitalOmega]4, \[CapitalOmega]6};
260 mostlyOrthogonalSymbols = Join[{E0p, E1p, E2p, E3p, αp, βp, γp},
   {T2p, T3, T4, T6, T7, T8, T11p, T12, T14, T15, T16, T17, T18, T19},
   cfSymbols];
261
262 paramFamilies = Hold[{racahSymbols, chenSymbols,
   slaterSymbols, controlSymbols, cfSymbols, TSymbols,
   pseudoMagneticSymbols, marvinSymbols, casimirSymbols,
   magneticSymbols, juddOfeltIntensitySymbols,
   mostlyOrthogonalSymbols}];
263 ReleaseHold[Protect /@ paramFamilies];
264 crystalGroups = {"C1", "Ci", "C2", "Cs", "C2h", "D2", "C2v", "D2h", "C4", "S4",
   "C4h", "D4", "C4v", "D3d", "D4h", "C3", "C3i", "D3", "C3v", "D3d", "C6", "C3h",
   "C6h", "D6", "C6v", "D3h", "D6h", "T", "Th", "O", "Td", "Oh"};
265
266 (* Parameter usage *)
267 paramLines = Select[StringSplit[paramAtlas, "\n"], # != "" &];
268 usageTemplate = StringTemplate["`paramSymbol`::usage=\``paramSymbol` `paramUsage`\`;"];
269 Do[(
270   {paramString, paramUsage} = StringSplit[paramLine, ":"];
271   paramUsage = StringTrim[paramUsage];

```

```

276   expressionString      = usageTemplate[<! "paramSymbol" ->
277     paramString, "paramUsage" -> paramUsage|>];
278   ToExpression[usageTemplate[<! "paramSymbol" -> paramString, "
279     paramUsage" -> paramUsage|>]]
280 ),
281 {paramLine, paramLines}
282 ];
283
284 AllowedJ;
285 AllowedMforJ;
286 AllowedNKS LJ Mfor JMTerms;
287 AllowedNKS LJ Mfor JTerms;
288 AllowedNKS LJ Terms;
289 AllowedNKS LT Terms;
290 AllowedSL JM Terms;
291 AllowedSL JT Terms;
292 AllowedSL Terms;
293
294 AngularMomentumMatrices;
295 BasisLSJ;
296 BasisLSJM J;
297 Bqk;
298 CFP;
299
300 CFPAssoc;
301 CFPTable;
302 CFPTerms;
303 Carnall;
304 CasimirG2;
305
306 CasimirS03;
307 CasimirS07;
308 Ck;
309 Cqk;
310 CrystalField;
311 CrystalFieldForm;
312
313 Dk;
314 EigenLever;
315 EffectiveHamiltonian;
316 Electrostatic;
317 ElectrostaticConfigInteraction;
318
319 ElectrostaticTable;
320 EnergyLevelDiagram;
321 EnergyStates;
322 EtoF;
323 ExportMZip;
324
325 FindNKLSTerm;
326 FindSL;
327 FreeHam;
328 FreeIonTable;
329 FromArrayToTable;
330 FtoE;
331
332 GG2U;
333 GS07W;
334 GenerateCFP;
335 GenerateCFPAssoc;
336 GenerateCFPTable;
337
338 GenerateCrystalFieldTable;
339 GenerateElectrostaticTable;
340 GenerateFreeIonTable;
341 GenerateReducedUkTable;
342 GenerateReducedV1kTable;
343 GenerateS00andECSOLSTable;
344
345 GenerateS00andECSOTable;
346 GenerateSpinOrbitTable;
347 GenerateSpinSpinTable;
348 GenerateT22Table;
349 GenerateThreeBodyTables;

```

```

350
351 GroundMagDipoleOscillatorStrength;
352 HamiltonianForm;
353
354 HamiltonianMatrixPlot;
355 HoleElectronConjugation;
356 ImportMZip;
357 IonSolver;
358 JJBlockMagDip;
359
360 JJBlockMatrix;
361 JJBlockMatrixFileName;
362 JJBlockMatrixTable;
363 JuddCFPPhase;
364 JuddOfeltUkSquared;
365 LabeledGrid;
366 LandeFactor;
367
368 LevelElecDipoleOscillatorStrength;
369 LevelJJBlockMagDipole;
370 LevelMagDipoleLineStrength;
371 LevelMagDipoleMatrixAssembly;
372 LevelMagDipoleOscillatorStrength;
373
374 LevelMagDipoleSpontaneousDecayRates;
375 LevelSimplerEffectiveHamiltonian;
376 LevelSolver;
377 LoadAll;
378
379 LoadCFP;
380 LoadCarnall;
381 LoadChenDeltas;
382 LoadElectrostatic;
383 LoadFreeIon;
384
385 LoadLaF3Parameters;
386 LoadLiYF4Parameters;
387 LoadSO0andECSO;
388 LoadSO0andECSOLS;
389 LoadSpinOrbit;
390 LoadSpinSpin;
391
392 LoadT22;
393 LoadTermLabels;
394 LoadThreeBody;
395
396 LoadUk;
397 LoadV1k;
398 MagDipLineStrength;
399 MagDipoleMatrixAssembly;
400 MagDipoleRates;
401
402 MagneticInteractions;
403 MapToSparseArray;
404 MaxJ;
405 MinJ;
406 NKCFPPhase;
407
408 ParamPad;
409 ParseBenelli2015;
410 ParseStates;
411 ParseStatesByNumBasisVecs;
412 ParseStatesByProbabilitySum;
413
414 ParseTermLabels;
415 Phaser;
416 PrettySaundersSL;
417 PrettySaundersSLJ;
418 PrettySaundersSLJmJ;
419
420 PrintL;
421 PrintSLJ;
422 PrintSLJM;
423 ReducedSO0andECS0inf2;
424 ReducedSO0andECS0infn;
425

```

```

426 ReducedT11inf2;
427 ReducedT22inf2;
428 ReducedT22infn;
429 ReducedUk;
430 ReducedUkTable;
431
432 ReducedV1k;
433 ReducedV1kTable;
434 Reducedt11inf2;
435 ReplaceInSparseArray;
436 ScalarLSJMFromLS;
437 SOOandECSO;
438 SOOandECSOLSTable;
439
440 SOOandECSOTable;
441 Seniority;
442 ShiftedLevels;
443 SimplerEffectiveHamiltonian;
444
445 SixJay;
446 SpinOrbit;
447 SpinOrbitTable;
448 SpinSpin;
449 SpinSpinTable;
450
451 Sqk;
452 SquarePrimeToNormal;
453 TPO;
454 T22Table;
455 TabulateJJBlockMagDipTable;
456 TabulateJJBlockMatrixTable;
457
458 TabulateManyJJBlockMagDipTables;
459 TabulateManyJJBlockMatrixTables;
460 ThreeBodyTable;
461 ThreeBodyTables;
462 ThreeJay;
463
464 chenDeltas;
465 fnTermLabels;
466 fsubk;
467
468 fsupk;
469 moduleDir;
470 symbolicHamiltonians;
471
472 (* this selects the function that is applied to calculated matrix
   elements which helps keep down the complexity of the resulting
   algebraic expressions *)
473 SimplifyFun = Expand;
474
475 Begin["`Private`"]
476
477 ListRepeater;
478 TotalCFIters;
479
480 moduleDir = ParentDirectory[DirectoryName[$InputFileName]];
481 frontEndAvailable = (Head[$FrontEnd] === FrontEndObject);
482
483 (* ##### MISC ####*)
484 (* ##### MISC ####*)
485
486 TPO::usage = "TPO[x, y, ...] gives the product of 2x+1, 2y+1, ...";
487 TPO[args__] := Times @@ ((2*# + 1) & /@ {args});
488
489 Phaser::usage = "Phaser[x] gives (-1)^x.";
490 Phaser[exponent_] := ((-1)^exponent);
491
492 TriangleCondition::usage = "TriangleCondition[a, b, c] evaluates
   the triangle condition on a, b, and c.";
493 TriangleCondition[a_, b_, c_] := (Abs[b - c] <= a <= (b + c));
494
495 TriangleAndSumCondition::usage = "TriangleAndSumCondition[a, b, c]
   evaluates the joint satisfaction of the triangle and sum
   conditions.";
496 TriangleAndSumCondition[a_, b_, c_] := (

```

```

497   And [
498     Abs[b - c] <= a <= (b + c),
499     IntegerQ[a + b + c]
500   ]
501 );
502
503 SquarePrimeToNormal::usage = "SquarePrimeToNormal[squarePrime]
evaluates the standard representation of a number from the squared
prime representation given in the list squarePrime. For
squarePrime of the form {c0, c1, c2, c3, ...} this function
returns the number c0 * Sqrt[p1^c1 * p2^c2 * p3^c3 * ...] where pi
is the ith prime number. Exceptionally some of the ci might be
letters in which case they have to be one of \"A\", \"B\", \"C\",
\"D\" with them corresponding to 10, 11, 12, and 13, respectively.
";
504 SquarePrimeToNormal[squarePrime_] :=
505 (
506   radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {
507     idx1, 2, Length[squarePrime]}];
508   radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" ->
509   13};
510   val = squarePrime[[1]] * Sqrt[radical];
511   Return[val];
512 );
513
514 ParamPad::usage = "ParamPad[params] takes an association params
whose keys are a subset of paramSymbols. The function returns a
new association where all the keys not present in paramSymbols,
will now be included in the returned association with their values
set to zero.
The function additionally takes an option \"Print\" that if set to
True, will print the symbols that were not present in the given
association. The default is True.";
515 Options[ParamPad] = {"PrintFun" -> PrintTemporary};
516 ParamPad[params_, OptionsPattern[]] :=
517   notPresentSymbols = Complement[paramSymbols, Keys[params]];
518   PrintFun = OptionValue["PrintFun"];
519   PrintFun["Following symbols were not given and are being set to
0: ",
520   notPresentSymbols];
521   newParams = Transpose[{paramSymbols, ConstantArray[0, Length[
522   paramSymbols]]}];
523   newParams = (#[[1]] -> #[[2]]) & /@ newParams;
524   newParams = Association[newParams];
525   newParams = Join[newParams, params];
526   Return[newParams];
527 )
528 (* ##### Angular Momentum #####
529
530 AngularMomentumMatrices::usage = "AngularMomentumMatrices[j] gives
the matrix representation for the angular momentum operators Jx,
Jy, and Jz for a given angular momentum j in the basis of
eigenvectors of jz. j may be a half-integer or an integer.
The options are \"Sparse\" which defaults to False and \"Order\""
which defaults to \"HighToLow\".
The option \"Order\" can be set to \"LowToHigh\" to get the
matrices in the order from -jay to jay otherwise they are returned
in the order jay to -jay.
The function returns a list {JxMatrix, JyMatrix, JzMatrix} with the
matrix representations for the cartesian components of the
angular momentum operator.";
531 Options[AngularMomentumMatrices] = {
532   "Sparse" -> False,
533   "Order" -> "HighToLow"};
534 AngularMomentumMatrices[jay_, OptionsPattern[]] := Module[
535   {
536     JxMatrix, JyMatrix, JzMatrix,
537     JPlusMatrix, JMinusMatrix,
538     m1, m2,
539     ArrayInverter
540   },
541   (
542     ArrayInverter = #[[-1 ;;, 1 ;;, -1, -1 ;;, 1 ;;, -1]] &;
543     JPlusMatrix = Table[
544

```

```

547     If[m2 == m1 + 1,
548       Sqrt[(jay - m1) (jay + m1 + 1)],
549       0
550     ],
551     {m1, jay, -jay, -1},
552     {m2, jay, -jay, -1}];
553 JMinusMatrix = Table[
554   If[m2 == m1 - 1,
555     Sqrt[(jay + m1) (jay - m1 + 1)],
556     0
557   ],
558   {m1, jay, -jay, -1},
559   {m2, jay, -jay, -1}];
560 JxMatrix = (JPlusMatrix + JMinusMatrix)/2;
561 JyMatrix = (JMinusMatrix - JPlusMatrix)/(2 I);
562 JzMatrix = DiagonalMatrix[Table[m, {m, jay, -jay, -1}]];
563 If[OptionValue["Sparse"],
564   {JxMatrix, JyMatrix, JzMatrix} = SparseArray /@ {JxMatrix,
565   JyMatrix, JzMatrix}
566 ];
567 If[OptionValue["Order"] == "LowToHigh",
568   {JxMatrix, JyMatrix, JzMatrix} = ArrayInverter /@ {JxMatrix,
569   JyMatrix, JzMatrix};
570 ];
571 Return[{JxMatrix, JyMatrix, JzMatrix}];
572 ]
573 LandeFactor::usage="LandeFactor[J, L, S] gives the Lande factor for
574   a given total angular momentum J, orbital angular momentum L, and
575   spin S.";
576 LandeFactor[J_, L_, S_] := (3/2) + (S*(S + 1) - L*(L + 1))/(2*J*(J
577 + 1));
578
579 (* ##### Angular Momentum #####
580 (* ##### Racah Algebra #####
581
582 ReducedUk::usage = "ReducedUk[n, l, SL, SpLp, k] gives the reduced
583   matrix element of the symmetric unit tensor operator U^(k). See
584   equation 11.53 in TASS.";
585 ReducedUk[numE_, l_, SL_, SpLp_, k_] := Module[
586   {spin, orbital, Uk, S, L,
587   Sp, Lp, Sb, Lb, parentSL,
588   cfpSL, cfpSpLp, Ukval,
589   SLparents, SLpparents,
590   commonParents, phase},
591   {spin, orbital} = {1/2, 3};
592   {S, L} = FindSL[SL];
593   {Sp, Lp} = FindSL[SpLp];
594   If[Not[S == Sp],
595     Return[0]
596   ];
597   cfpSL = CFP[{numE, SL}];
598   cfpSpLp = CFP[{numE, SpLp}];
599   SLparents = First /@ Rest[cfpSL];
600   SLpparents = First /@ Rest[cfpSpLp];
601   commonParents = Intersection[SLparents, SLpparents];
602   Uk = Sum[
603     {Sb, Lb} = FindSL[\[Psi]b];
604     Phaser[Lb] *
605       CFPAssoc[{numE, SL, \[Psi]b}] *
606       CFPAssoc[{numE, SpLp, \[Psi]b}] *
607       SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
608   ),
609   {\[Psi]b, commonParents}
610   ];
611   phase = Phaser[orbital + L + k];
612   prefactor = numE * phase * Sqrt[TPO[L, Lp]];
613   Ukval = prefactor*Uk;
614   Return[Ukval];
615 ]

```

```

616 Ck::usage = "Ck[orbital, k] gives the diagonal reduced matrix
617   element <l||C^(k)||l> where the Subscript[C, q]^^(k) are
618   renormalized spherical harmonics. See equation 11.23 in TASS with
619   l=l'.";
620 Ck[orbital_, k_] := (-1)^orbital * TPO[orbital] * ThreeJay[{orbital
621   , 0}, {k, 0}, {orbital, 0}];
622
623 SixJay::usage = "SixJay[{j1, j2, j3}, {j4, j5, j6}] provides the
624   value for SixJSymbol[{j1, j2, j3}, {j4, j5, j6}] with memorization
625   of computed values and short-circuiting values based on triangle
626   conditions.";
627 SixJay[{j1_, j2_, j3_}, {j4_, j5_, j6_}] := (
628   sixJayval = Which[
629     Not[TriangleAndSumCondition[j1, j2, j3]], 0,
630     Not[TriangleAndSumCondition[j1, j5, j6]], 0,
631     Not[TriangleAndSumCondition[j4, j2, j6]], 0,
632     Not[TriangleAndSumCondition[j4, j5, j3]], 0,
633     True, SixJSymbol[{j1, j2, j3}, {j4, j5, j6}]];
634   SixJay[{j1, j2, j3}, {j4, j5, j6}] = sixJayval);
635
636 ThreeJay::usage = "ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] gives the
637   value of the Wigner 3j-symbol and memorizes the computed value.";
638 ThreeJay[{j1_, m1_}, {j2_, m2_}, {j3_, m3_}] := (
639   threejval = Which[
640     Not[(m1 + m2 + m3) == 0], 0,
641     Not[TriangleCondition[j1, j2, j3]], 0,
642     True, ThreeJSymbol[{j1, m1}, {j2, m2}, {j3, m3}]];
643   ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] = threejval);
644
645 ReducedV1k::usage = "ReducedV1k[n, SL, SpLp, k] gives the reduced
646   matrix element of the spherical tensor operator V^(1k). See
647   equation 2-101 in Wybourne 1965.";
648 ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
649   {Vk1, S, L, Sp, Lp,
650   Sb, Lb, spin, orbital,
651   cfpSL, cfpSpLp,
652   SLparents, SpLpparents,
653   commonParents, prefactor},
654   (
655     {spin, orbital} = {1/2, 3};
656     {S, L} = FindSL[SL];
657     {Sp, Lp} = FindSL[SpLp];
658     cfpSL = CFP[{numE, SL}];
659     cfpSpLp = CFP[{numE, SpLp}];
660     SLparents = First /@ Rest[cfpSL];
661     SpLpparents = First /@ Rest[cfpSpLp];
662     commonParents = Intersection[SLparents, SpLpparents];
663     Vk1 = Sum[(
664       {Sb, Lb} = FindSL[\[Psi]b];
665       Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
666       CFPAssoc[{numE, SL, \[Psi]b}] *
667       CFPAssoc[{numE, SpLp, \[Psi]b}] *
668       SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
669       SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
670     ), {\[Psi]b, commonParents}];
671   ];
672   prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp,
673   Lp]];
674   Return[prefactor * Vk1];
675 )
676
677 GenerateReducedUkTable::usage = "GenerateReducedUkTable[numEmax]
678   can be used to generate the association of reduced matrix elements
679   for the unit tensor operators Uk from f^1 up to f^numEmax. If the
680   option \"Export\" is set to True then the resulting data is saved"

```

```

    to ./data/ReducedUkTable.m.";
678 Options[GenerateReducedUkTable] = {"Export" -> True, "Progress" ->
679   True};
680 GenerateReducedUkTable[numEmax_Integer:7, OptionsPattern[]] := (
681   numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
682     AllowedNKSLTerms[#]]&/@Range[1, numEmax]] * 4;
683   Echo["Calculating " <> ToString[numValues] <> " values for Uk k
684   =0,2,4,6."];
685   counter = 1;
686   If[And[OptionValue["Progress"], frontEndAvailable],
687     progBar = PrintTemporary[
688       Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
689         counter}]]]
690     ];
691   ReducedUkTable = Table[
692     (
693       counter = counter+1;
694       {numE, 3, SL, SpLp, k} -> SimplifyFun[ReducedUk[numE, 3, SL,
695         SpLp, k]]
696       ),
697       {numE, 1, numEmax},
698       {SL, AllowedNKSLTerms[numE]},
699       {SpLp, AllowedNKSLTerms[numE]},
700       {k, {0, 2, 4, 6}}
701     ];
702   ReducedUkTable = Association[Flatten[ReducedUkTable]];
703   ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "ReducedUkTable.m"}];
704   If[And[OptionValue["Progress"], frontEndAvailable],
705     NotebookDelete[progBar]
706   ];
707   If[OptionValue["Export"],
708     (
709       Echo["Exporting to file " <> ToString[ReducedUkTableFname]];
710       Export[ReducedUkTableFname, ReducedUkTable];
711     )
712   ];
713   Return[ReducedUkTable];
714 }
715
716 GenerateReducedV1kTable::usage = "GenerateReducedV1kTable[nmax]
717 calculates values for Vk and returns an association where the
718 keys are lists of the form {n, SL, SpLp, 1}. If the option \
719 \"Export\" is set to True then the resulting data is saved to ./data
720 /ReducedV1kTable.m.";
721 Options[GenerateReducedV1kTable] = {"Export" -> True, "Progress" ->
722   True};
723 GenerateReducedV1kTable[numEmax_Integer:7, OptionsPattern[]] := (
724   numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
725     AllowedNKSLTerms[#]]&/@Range[1, numEmax]];
726   Echo["Calculating " <> ToString[numValues] <> " values for Vk."
727 ];
728   counter = 1;
729   If[And[OptionValue["Progress"], frontEndAvailable],
730     progBar = PrintTemporary[
731       Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
732         counter}]]]
733     ];
734   ReducedV1kTable = Table[
735     (
736       counter = counter+1;
737       {n, SL, SpLp, 1} -> SimplifyFun[ReducedV1k[n, SL, SpLp, 1]]
738       ),
739       {n, 1, numEmax},
740       {SL, AllowedNKSLTerms[n]},
741       {SpLp, AllowedNKSLTerms[n]}
742     ];
743   ReducedV1kTable = Association[ReducedV1kTable];
744   If[And[OptionValue["Progress"], frontEndAvailable],
745     NotebookDelete[progBar]
746   ];
747   exportFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m
748 }];
749   If[OptionValue["Export"],
750     (
751       Echo["Exporting to file " <> ToString[exportFname]];
752     )
753   ];

```

```

740     Export[exportFname, ReducedV1kTable];
741   )
742 ];
743 Return[ReducedV1kTable];
744 )
745
746 (* ##### Racah Algebra ##### *)
747 (* ##### *)
748
749 (* ##### *)
750 (* ##### Electrostatic ##### *)
751
752 fsubk::usage = "fsubk[numE, orbital, SL, SLp, k] gives the Slater
    integral f_k for the given configuration and pair of SL terms. See
    equation 12.17 in TASS.";
753 fsubk[numE_, orbital_, NKSL_, NKSLp_, k_] := Module[
754   {terms, S, L, Sp, Lp,
755    termsWithSameSpin, SL,
756    fsubkVal, spinMultiplicity,
757    prefactor, summand1, summand2},
758   (
759     {S, L} = FindSL[NKSL];
760     {Sp, Lp} = FindSL[NKSLp];
761     terms = AllowedNKSLTerms[numE];
762     (* sum for summand1 is over terms with same spin *)
763     spinMultiplicity = 2*S + 1;
764     termsWithSameSpin = StringCases[terms, ToString[
765       spinMultiplicity] ~~ __];
766     termsWithSameSpin = Flatten[termsWithSameSpin];
767     If[Not[{S, L} == {Sp, Lp}],
768       Return[0]
769     ];
770     prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
771     summand1 = Sum[((
772       ReducedUkTable[{numE, orbital, SL, NKSL, k}] *
773       ReducedUkTable[{numE, orbital, SL, NKSLp, k}]
774     ),
775       {SL, termsWithSameSpin}
776     ];
777     summand1 = 1 / TPO[L] * summand1;
778     summand2 = (
779       KroneckerDelta[NKSL, NKSLp] *
780       (numE *(4*orbital + 2 - numE)) /
781       ((2*orbital + 1) * (4*orbital + 1))
782     );
783     fsubkVal = prefactor*(summand1 - summand2);
784     Return[fsubkVal];
785   )
786 ];
787
788 fsupk::usage = "fsupk[numE, orbital, SL, SLp, k] gives the
    superscripted Slater integral f^k = Subscript[f, k] * Subscript[D,
    k].";
789 fsupk[numE_, orbital_, NKSL_, NKSLp_, k_] := (
790   Dk[k] * fsubk[numE, orbital, NKSL, NKSLp, k]
791 )
792
793 Dk::usage = "D[k] gives the ratio between the super-script and sub-
    scripted Slater integrals (F^k / F_k). k must be even. See table
    6-3 in TASS, and also section 2-7 of Wybourne (1965). See also
    equation 6.41 in TASS.";
794 Dk[k_] := {1, 225, 1089, 184041/25}[[k/2+1]];
795
796 FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the Racah parameters
    {E0, E1, E2, E3} corresponding to the given Slater integrals.
    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.";
797 FtoE[F0_, F2_, F4_, F6_] := Module[
798   {E0, E1, E2, E3},
799   (
800     E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
801     E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
802     E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;

```

```

804     E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
805     Return[{E0, E1, E2, E3}];
806   )
807 ];
808
809 EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the Slater integral
810   parameters {F0, F2, F4, F6} corresponding to the given Racah
811   parameters {E0, E1, E2, E3}. This is the inverse of the FtoE
812   function.";
813 EtoF[E0_, E1_, E2_, E3_] := Module[
814   {F0, F2, F4, F6},
815   (
816     F0 = 1/7      (7 E0 + 9 E1);
817     F2 = 75/14    (E1 + 143 E2 + 11 E3);
818     F4 = 99/7     (E1 - 130 E2 + 4 E3);
819     F6 = 5577/350 (E1 + 35 E2 - 7 E3);
820     Return[{F0, F2, F4, F6}];
821   )
822 ];
823
824 Electrostatic::usage = "Electrostatic[{numE, NKSL, NKSLp}] returns
825   the LS reduced matrix element for repulsion matrix element for
826   equivalent electrons. See equation 2-79 in Wybourne (1965). The
827   option \"Coefficients\" can be set to \"Slater\" or \"Racah\". If
828   set to \"Racah\" then E_k parameters and e^k operators are assumed
829   , otherwise the Slater integrals F^k and operators f_k. The
830   default is \"Slater\".";
831 Options[Electrostatic] = {"Coefficients" -> "Slater"};
832 Electrostatic[{numE_, NKSL_, NKSLp_}, OptionsPattern[]] := Module[
833   {fsub0, fsub2, fsub4, fsub6,
834   esub0, esub1, esub2, esub3,
835   fsup0, fsup2, fsup4, fsup6,
836   eMatrixVal, orbital},
837   (
838     orbital = 3;
839     Which[
840       OptionValue["Coefficients"] == "Slater",
841       (
842         fsub0 = fsubk[numE, orbital, NKSL, NKSLp, 0];
843         fsub2 = fsubk[numE, orbital, NKSL, NKSLp, 2];
844         fsub4 = fsubk[numE, orbital, NKSL, NKSLp, 4];
845         fsub6 = fsubk[numE, orbital, NKSL, NKSLp, 6];
846         eMatrixVal = fsub0*F0 + fsub2*F2 + fsub4*F4 + fsub6*F6;
847       ),
848       OptionValue["Coefficients"] == "Racah",
849       (
850         fsup0 = fsupk[numE, orbital, NKSL, NKSLp, 0];
851         fsup2 = fsupk[numE, orbital, NKSL, NKSLp, 2];
852         fsup4 = fsupk[numE, orbital, NKSL, NKSLp, 4];
853         fsup6 = fsupk[numE, orbital, NKSL, NKSLp, 6];
854         esub0 = fsup0;
855         esub1 = 9/7*fsup0 + 1/42*fsup2 + 1/77*fsup4 + 1/462*
856         fsup6;
857         esub2 = 143/42*fsup2 - 130/77*fsup4 + 35/462*
858         fsup6;
859         esub3 = 11/42*fsup2 + 4/77*fsup4 - 7/462*
860         fsup6;
861         eMatrixVal = esub0*E0 + esub1*E1 + esub2*E2 + esub3*E3;
862       )
863     ];
864     Return[eMatrixVal];
865   )
866 ];
867
868 GenerateElectrostaticTable::usage = "GenerateElectrostaticTable[
869   numEmax] can be used to generate the table for the electrostatic
870   interaction from f^1 to f^numEmax. If the option \"Export\" is set
871   to True then the resulting data is saved to ./data/
872   ElectrostaticTable.m.";
873 Options[GenerateElectrostaticTable] = {"Export" -> True, "
874   Coefficients" -> "Slater"};
875 GenerateElectrostaticTable[numEmax_Integer:7, OptionsPattern[]] :=
876   (
877     ElectrostaticTable = Table[
878       {numE, SL, SpLp} -> SimplifyFun[Electrostatic[{numE, SL, SpLp}],
879       "Coefficients" -> OptionValue["Coefficients"]]];

```

```

861     {numE, 1, numEmax},
862     {SL, AllowedNKSLTerms[numE]},
863     {SpLp, AllowedNKSLTerms[numE]}
864   ];
865   ElectrostaticTable = Association[Flatten[ElectrostaticTable]];
866   If[OptionValue["Export"],
867     Export[FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}],
868     ElectrostaticTable];
869   ];
870   Return[ElectrostaticTable];
871 );
872
873 (* ##### Electrostatic #####
874 (* ##### Bases #####
875 (* ##### BasisLSJM #####
876 (* ##### BasisLSJ #####
877
878 BasisTableGenerator::usage = "BasisTableGenerator[numE] gives an
879   association whose keys are lists of the form {numE, J} and whose
880   values are lists with elements of the form {LS, J, MJ}
881   representing the elements of the LSJM coupled basis.";
882 BasisTableGenerator[numE_] := Module[
883   {energyStatesTable, allowedJ, J, Jp},
884   (
885     energyStatesTable = <||>;
886     allowedJ = AllowedJ[numE];
887     Do[
888       (
889         energyStatesTable[{numE, J}] = EnergyStates[numE, J];
890       ),
891       {Jp, allowedJ},
892       {J, allowedJ}];
893     Return[energyStatesTable]
894   )
895 ];
896
897 BasisLSJM::usage = "BasisLSJM[numE] returns the ordered basis in
898   L-S-J-MJ with the total orbital angular momentum L and total spin
899   angular momentum S coupled together to form J. The function
900   returns a list with each element representing the quantum numbers
901   for each basis vector. Each element is of the form {SL (string in
902   spectroscopic notation),J, MJ}.
903 The option \"AsAssociation\" can be set to True to return the basis
904   as an association with the keys corresponding to values of J and
905   the values lists with the corresponding {L, S, J, MJ} list. The
906   default of this option is False.
907 ";
908 Options[BasisLSJM] = {"AsAssociation" -> False};
909 BasisLSJM[numE_, OptionsPattern[]] := Module[
910   {energyStatesTable, basis, idx1},
911   (
912     energyStatesTable = BasisTableGenerator[numE];
913     basis = Table[
914       energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
915       {idx1, 1, Length[AllowedJ[numE]]}];
916     basis = Flatten[basis, 1];
917     If[OptionValue["AsAssociation"],
918       (
919         Js = AllowedJ[numE];
920         basis = Table[(J -> Select[basis, #[[2]] == J &]), {J, Js}];
921       ];
922       basis = Association[basis];
923     ];
924   ];
925   Return[basis];
926 );
927 ];
928
929 BasisLSJ::usage = "BasisLSJ[numE] returns the level basis LSJ. The
930   function returns a list with each element representing the quantum
931   numbers for each basis vector. Each element is of the form {SL (
932   string in spectroscopic notation), J}.
933 The option \"AsAssociation\" can be set to True to return the basis
934   as an association with the keys being the allowed J values. The

```

```

920     default is False.
921     ";
922 Options[BasisLSJ] = {"AsAssociation" -> False};
923 BasisLSJ[numE_, OptionsPattern[]] := Module[
924   {Js, basis},
925   (
926     Js = AllowedJ[numE];
927     basis = BasisLSJMJ[numE, "AsAssociation" -> False];
928     basis = DeleteDuplicates[{{#[[1]], #[[2]]} & /@ basis];
929     If[OptionValue["AsAssociation"],
930       (
931         basis = Association @ Table[(J -> Select[basis, #[[2]] == J &]), {
932           J, Js}]
933       )
934     ];
935     Return[basis];
936   );
937 (* ##### Bases #####
938 (* ##### Coefficients of Fractional Parentage #####
939 (* ##### Coefficients of Fractional Parentage #####
940 (* ##### Coefficients of Fractional Parentage #####
941 (* ##### Coefficients of Fractional Parentage #####
942
943 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.";
944 Options[GenerateCFP] = {"Export" -> True, "PhaseFunction" -> "NK"};
945 GenerateCFP[OptionsPattern[]] := (
946   CFP = Table[
947     {numE, NKSL} -> First[CFPTerms[numE, NKSL]],
948     {numE, 1, 7},
949     {NKSL, AllowedNKSLTerms[numE]}];
950   CFP = Association[CFP];
951   (* Go all the way to f14 *)
952   CFP = CFPExander["Export" -> False, "PhaseFunction" ->
953   OptionValue["PhaseFunction"]];
954   If[OptionValue["Export"],
955     Export[FileNameJoin[{moduleDir, "data", "CFPs.m"}], CFP];
956   ];
957   Return[CFP];
958 );
959
960 JuddCFPPPhase::usage = "Phase between conjugate coefficients of
fractional parentage according to Velkov's thesis, page 40.";
961 JuddCFPPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
962 parentSeniority_, daughterSeniority_] := Module[
963   {spin, orbital, expo, phase},
964   (
965     {spin, orbital} = {1/2, 3};
966     expo = (
967       (parentS + parentL + daughterS + daughterL) -
968       (orbital + spin) +
969       1/2 * (parentSeniority + daughterSeniority - 1)
970     );
971     phase = Phaser[-expo];
972     Return[phase];
973   );
974
975 NKCFPPPhase::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  $(-1)^{v - 1/2}$ .";;
976 NKCFPPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
977 parentSeniority_, daughterSeniority_] := Module[
978   {spin, orbital, expo, phase},
979   (
980     {spin, orbital} = {1/2, 3};
981     expo = (

```

```

980         (parentS + parentL + daughterS + daughterL) -
981         (orbital + spin)
982     );
983     phase = Phaser[-expo];
984     If[parent == 2*orbital,
985         phase = phase * Phaser[(daughterSeniority - 1)/2]];
986     Return[phase];
987   )
988 ];
989
990 Options[CFPExpander] = {"Export" -> True, "PhaseFunction" -> "NK"};
991 CFPExpander::usage = "Using the coefficients of fractional
992   parentage up to f7 this function calculates them up to f14.
993 The coefficients of fractional parentage are taken beyond the half-
994   filled shell using the phase convention determined by the option \
995   \"PhaseFunction\". The default is \"NK\" which corresponds to the
996   phase convention of Nielson and Koster. The other option is \"Judd
997   \" which corresponds to the phase convention of Judd. The result
998   is exported to the file ./data/CFPs_extended.m.";
999 CFPExpander[OptionsPattern[]] := Module[
1000   {orbital, halfFilled, fullShell, parentMax, PhaseFun,
1001   complementaryCFPs, daughter, conjugateDaughter,
1002   conjugateParent, parentTerms, daughterTerms,
1003   parentCFPs, daughterSeniority, daughterS, daughterL,
1004   parentCFP, parentTerm, parentCFPval,
1005   parentS, parentL, parentSeniority, phase, prefactor,
1006   newCFPval, key, extendedCFPs, exportFname},
1007   (
1008     orbital      = 3;
1009     halfFilled   = 2 * orbital + 1;
1010     fullShell    = 2 * halfFilled;
1011     parentMax    = 2 * orbital;
1012
1013     PhaseFun = <|
1014       "Judd" -> JuddCFPPhase,
1015       "NK" -> NKCFPPhase|>[OptionValue["PhaseFunction"]];
1016     PrintTemporary["Calculating CFPs using the phase system from ",
1017     PhaseFun];
1018     (* Initialize everything with lists to be filled in the next Do
1019    *)
1020     complementaryCFPs =
1021       Table[
1022         ({numE, term} -> {term}),
1023         {numE, halfFilled + 1, fullShell - 1, 1},
1024         {term, AllowedNKSLTerms[numE]
1025           }];
1026     complementaryCFPs = Association[Flatten[complementaryCFPs]];
1027     Do[(
1028       daughter          = parent + 1;
1029       conjugateDaughter = fullShell - parent;
1030       conjugateParent   = conjugateDaughter - 1;
1031       parentTerms       = AllowedNKSLTerms[parent];
1032       daughterTerms     = AllowedNKSLTerms[daughter];
1033       Do[
1034         (
1035           parentCFPs          = Rest[CFP[{daughter,
1036             daughterTerm}]];
1037           daughterSeniority    = Seniority[daughterTerm];
1038           {daughterS, daughterL} = FindSL[daughterTerm];
1039           Do[
1040             (
1041               {parentTerm, parentCFPval} = parentCFP;
1042               {parentS, parentL}        = FindSL[parentTerm];
1043               parentSeniority         = Seniority[parentTerm];
1044               phase = PhaseFun[parent, parentS, parentL,
1045                                 daughterS, daughterL,
1046                                 parentSeniority, daughterSeniority
1047                 ];
1048               prefactor = (daughter * TPO[daughterS, daughterL])
1049             /
1050               (conjugateDaughter * TPO[parentS,
1051                 parentL]);
1052               prefactor = Sqrt[prefactor];
1053               newCFPval = phase * prefactor * parentCFPval;
1054               key = {conjugateDaughter, parentTerm};
1055               complementaryCFPs[key] = Append[complementaryCFPs[
1056

```

```

1044     key], {daughterTerm, newCFPval}]
1045         ),
1046         {parentCFP, parentCFPs}
1047     ]
1048     ),
1049     {daughterTerm, daughterTerms}
1050   ]
1051   ),
1052   {parent, 1, parentMax}
1053 ];
1054
1055 complementaryCFPs[{14, "1S"}] = {"1S", {"2F", 1}};
1056 extendedCFPs = Join[CFP, complementaryCFPs];
1057 If[OptionValue["Export"]];
1058 (
1059   exportFname = FileNameJoin[{moduleDir, "data", "CFPs_extended.m"}];
1060   Echo["Exporting to " <> exportFname];
1061   Export[exportFname, extendedCFPs];
1062 )
1063 ];
1064 Return[extendedCFPs];
1065 )
1066 ];
1067 GenerateCFPTable::usage = "GenerateCFPTable[] generates the table
1068   for the coefficients of fractional parentage. If the optional
1069   parameter \"Export\" is set to True then the resulting data is
1070   saved to ./data/CFPTable.m.
1071 The data being parsed here is the file attachment B1F_ALL.TXT which
1072   comes from Velkov's thesis.";
1073 Options[GenerateCFPTable] = {"Export" -> True};
1074 GenerateCFPTable[OptionsPattern[]] := Module[
1075   {rawText, rawLines, leadChar, configIndex, line, daughter,
1076   lineParts, numberCode, parsedNumber, toAppend, CFPTablefname},
1077   (
1078     CleanWhitespace[string_] := StringReplace[string,
1079       RegularExpression["\\s+"]->" "];
1080     AddSpaceBeforeMinus[string_] := StringReplace[string,
1081       RegularExpression["(?<!\\s)-"]->" -"];
1082     ToIntegerOrString[list_] := Map[If[StringMatchQ[#, NumberString], ToExpression[#], #] &, list];
1083     CFPTable = ConstantArray[{}, 7];
1084     CFPTable[[1]] = {{"2F", {"1S", 1}}};
1085
1086     (* Cleaning before processing is useful *)
1087     rawText = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
1088     rawLines = StringTrim/@StringSplit[rawText, "\n"];
1089     rawLines = Select[rawLines, #!="&"];
1090     rawLines = CleanWhitespace/@rawLines;
1091     rawLines = AddSpaceBeforeMinus/@rawLines;
1092
1093     Do[(
1094       (* the first character can be used to identify the start of a
1095       block *)
1096       leadChar=StringTake[line,{1}];
1097       (* ..FN, N is at position 50 in that line *)
1098       If[leadChar=="[",
1099       (
1100         configIndex=ToExpression[StringTake[line,{50}]];
1101         Continue[];
1102       )
1103     ];
1104     (* Identify which daughter term is being listed *)
1105     If[StringContainsQ[line, "[DAUGHTER TERM]"],
1106       daughter=StringSplit[line, "["[[1]];
1107       CFPTable[[configIndex]]=Append[CFPTable[[configIndex]],{daughter}];
1108       Continue[];
1109     ];
1110     (* Once we get here we are already parsing a row with
1111     coefficient data *)
1112     lineParts = StringSplit[line, " "];
1113     parent = lineParts[[1]];
1114     numberCode = ToIntegerOrString[lineParts[[3;;]]];

```

```

1107     parsedNumber = SquarePrimeToNormal[numberCode];
1108     toAppend = {parent, parsedNumber};
1109     CFPTable[[configIndex]][[-1]] = Append[CFPTable[[configIndex
1110     ]][[-1]], toAppend]
1111     ),
1112     {line, rawLines}];
1113     If[OptionValue["Export"],
1114     (
1115       CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"
1116     }];
1117       Export[CFPTablefname, CFPTable];
1118     )
1119   ];
1120 ];
1121
1122 GenerateCFPAssoc::usage = "GenerateCFPAssoc[] converts the
1123   coefficients of fractional parentage into an association in which
1124   zero values are explicit. If the option \"Export\" is set to True,
1125   the association is exported to the file /data/CFPAssoc.m. This
1126   function requires that the association CFP be defined.";
1127 Options[GenerateCFPAssoc] = {"Export" -> True};
1128 GenerateCFPAssoc[OptionsPattern[]] := (
1129   CFPAssoc = Association[];
1130   Do[
1131     (daughterTerms = AllowedNKSLTerms[numE];
1132      parentTerms = AllowedNKSLTerms[numE - 1];
1133      Do[
1134        (
1135          cfps = CFP[{numE, daughter}];
1136          cfps = cfps[[2 ;;]];
1137          parents = First /@ cfps;
1138          Do[
1139            (
1140              key = {numE, daughter, parent};
1141              cfp = If[
1142                MemberQ[parents, parent],
1143                (
1144                  idx = Position[parents, parent][[1, 1]];
1145                  cfps[[idx]][[2]]
1146                ),
1147                0
1148              ];
1149              CFPAssoc[key] = cfp;
1150            ),
1151            {parent, parentTerms}
1152          ]
1153        ),
1154        {daughter, daughterTerms}
1155      ]
1156    ),
1157    {numE, 1, 14}
1158  ];
1159  If[OptionValue["Export"],
1160  (
1161    CFPAssocfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"
1162  }];
1163    Export[CFPAssocfname, CFPAssoc];
1164  )
1165  ];
1166  Return[CFPAssoc];
1167 );
1168
1169 CFPTerms::usage = "CFPTerms[numE] gives all the daughter and parent
1170   terms, together with the corresponding coefficients of fractional
1171   parentage, that correspond to the the f^n configuration.
1172 CFPTerms[numE, SL] gives all the daughter and parent terms,
1173   together with the corresponding coefficients of fractional
1174   parentage, that are compatible with the given string SL in the f^n
1175   configuration.
1176 CFPTerms[numE, L, S] gives all the daughter and parent terms,
1177   together with the corresponding coefficients of fractional
1178   parentage, that correspond to the given total orbital angular
1179   momentum L and total spin S in the f^n configuration. L being an
1180   integer, and S being integer or half-integer.

```

```

1167 In all cases the output is in the shape of a list with enclosed
1168 lists having the format {daughter_term, {parent_term_1, CFP_1}, {
1169 parent_term_2, CFP_2}, ...}.
1170 Only the one-body coefficients for f-electrons are provided.
1171 In all cases it must be that 1 <= n <= 7.
1172 These are according to the tables from Nielson & Koster.
1173 ";
1174 CFPTerms[numE_] := Part[CFPTable, numE]
1175 CFPTerms[numE_, SL_] := Module[
1176   {NKterms, CFPconfig},
1177   (
1178     NKterms = {};
1179     CFPconfig = CFPTable[[numE]];
1180     Map[
1181       If[StringFreeQ[First[#], SL],
1182         Null,
1183         NKterms = Join[NKterms, {#}, 1]
1184       ] &,
1185       CFPconfig
1186     ];
1187     NKterms = DeleteCases[NKterms, {}]
1188   )
1189 ];
1190 CFPTerms[numE_, L_, S_] := Module[
1191   {NKterms, SL, CFPconfig},
1192   (
1193     SL = StringJoin[ToString[2 S + 1], PrintL[L]];
1194     NKterms = {};
1195     CFPconfig = Part[CFPTable, numE];
1196     Map[
1197       If[StringFreeQ[First[#], SL],
1198         Null,
1199         NKterms = Join[NKterms, {#}, 1]
1200       ] &,
1201       CFPconfig
1202     ];
1203     NKterms = DeleteCases[NKterms, {}]
1204   )
1205 ];
1206 (* ##### Coefficients of Fracional Parentage ##### *)
1207 (* ##### ##### ##### ##### ##### ##### ##### ##### *)
1208 (* ##### ##### ##### ##### ##### ##### ##### ##### *)
1209 (* ##### ##### ##### ##### ##### Spin Orbit ##### *)
1210
1211 SpinOrbit::usage = "SpinOrbit[numE, SL, SpLp, J] returns the LSJ
1212 reduced matrix element  $\zeta \langle SL, J | L.S| SpLp, J \rangle$ . These are given as a
1213 function of  $\zeta$ . This function requires that the association
1214 ReducedV1kTable be defined.
1215 See equations 2-106 and 2-109 in Wybourne (1965). Equivalently see
1216 eqn. 12.43 in TASS.";
1217 SpinOrbit[numE_, SL_, SpLp_, J_] := Module[
1218   {S, L, Sp, Lp, orbital, sign, prefactor, val},
1219   (
1220     orbital = 3;
1221     {S, L} = FindSL[SL];
1222     {Sp, Lp} = FindSL[SpLp];
1223     prefactor = Sqrt[orbital * (orbital+1) * (2*orbital+1)] *
1224           SixJay[{L, Lp, 1}, {Sp, S, J}];
1225     sign = Phaser[J + L + Sp];
1226     val = sign * prefactor *  $\zeta$  * ReducedV1kTable[{numE, SL,
1227     SpLp, 1}];
1228     Return[val];
1229   )
1230 ];
1231
1232 SpinOrbitTable::usage="An association containing the matrix
1233 elements for the spin-orbit interaction for f^n configurations.
1234 The keys are lists of the form {n, SL, SpLp, J}.";
1235
1236 GenerateSpinOrbitTable::usage = "GenerateSpinOrbitTable[nmax]
1237 computes the matrix elements for the spin-orbit interaction for f^
1238 n configurations up to n = nmax. The function returns an
1239 association whose keys are lists of the form {n, SL, SpLp, J}. If
1240 \"Export\" is set to True, then the result is exported to the data

```

```

1230   folder. It requires ReducedV1kTable to be defined.";
1231 Options[GenerateSpinOrbitTable] = {"Export" -> True};
1232 GenerateSpinOrbitTable[nmax_Integer:7, OptionsPattern[]] := Module[
1233 {numE, J, SL, SpLp, exportFname},
1234 (
1235   SpinOrbitTable =
1236   Table[
1237     {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
1238     {numE, 1, nmax},
1239     {J, MinJ[numE], MaxJ[numE]},
1240     {SL, Map[First, AllowedNKSLforJTerms[numE, J]]},
1241     {SpLp, Map[First, AllowedNKSforJTerms[numE, J]]}
1242   ];
1243   SpinOrbitTable = Association[SpinOrbitTable];
1244 
1245   exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
1246   If[OptionValue["Export"],
1247     (
1248       Echo["Exporting to file " <> ToString[exportFname]];
1249       Export[exportFname, SpinOrbitTable];
1250     )
1251   ];
1252   Return[SpinOrbitTable];
1253 ]
1254 
1255 (* ##### Spin Orbit #####
1256 (* ##### Three Body Operators #####
1257 
1258 (* ##### Three Body Operators #####
1259 
1260 ParseJudd1984::usage = "This function parses the data from tables 1
1261 and 2 of Judd from Judd, BR, and MA Suskin. \"Complete Set of
1262 Orthogonal Scalar Operators for the Configuration f^3\". JOSA B 1,
1263 no. 2 (1984): 261-65.";
1264 Options[ParseJudd1984] = {"Export" -> False};
1265 ParseJudd1984[OptionsPattern[]] :=
1266 ParseJuddTab1[str_] :=
1267   strR = ToString[str];
1268   strR = StringReplace[strR, ".5" -> "^(1/2)"];
1269   num = ToExpression[strR];
1270   sign = Sign[num];
1271   num = sign*Simplify[Sqrt[num^2]];
1272   If[Round[num] == num, num = Round[num]];
1273   Return[num];
1274 
1275 (* Parse table 1 from Judd 1984 *)
1276 judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"}];
1277 data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
1278 headers = data[[1]];
1279 data = data[[2 ;;]];
1280 data = Transpose[data];
1281 \[Psi] = Select[data[[1]], # != "" &];
1282 \[Psi]p = Select[data[[2]], # != "" &];
1283 matrixKeys = Transpose[{\[Psi], \[Psi]p}];
1284 data = data[[3 ;;]];
1285 cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data}];
1286 cols = Select[cols, Length[#] == 21 &];
1287 tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
1288 tab1 = Transpose[Prepend[Transpose[tab1], headers]];
1289 
1290 (* Parse table 2 from Judd 1984 *)
1291 judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"}];
1292 data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
1293 headers = data[[1]];
1294 data = data[[2 ;;]];
1295 data = Transpose[data];
1296 {operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} =
1297 data[[;; 4]];
1298 multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
1299 multiFactorValues = AssociationThread[multiFactorSymbols ->

```

```

1297 multiFactorValues];
1298 (*scale values of table 1 given the values in table 2*)
1299 oppyS = {};
1300 normalTable =
1301 Table[header = col[[1]];
1302 If[StringContainsQ[header, " "],
1303 (
1304 multiplierSymbol = StringSplit[header, " "][[1]];
1305 multiplierValue = multiFactorValues[multiplierSymbol];
1306 operatorSymbol = StringSplit[header, " "][[2]];
1307 oppyS = Append[oppyS, operatorSymbol];
1308 ),
1309 (
1310 multiplierValue = 1;
1311 operatorSymbol = header;
1312 )
1313 ];
1314 normalValues = 1/multiplierValue*col[[2 ;]];
1315 Join[{operatorSymbol}, normalValues], {col, tab1[[3 ;]]}
1316 ];
1317 (*Create an association for the reduced matrix elements in the f^3 config*)
1318 juddOperators = Association[];
1319 Do[
1320 col = normalTable[[colIndex]];
1321 opLabel = col[[1]];
1322 opValues = col[[2 ;]];
1323 opMatrix = AssociationThread[matrixKeys -> opValues];
1324 Do[
1325 opMatrix[Reverse[mKey]] = opMatrix[mKey]
1326 ],
1327 {mKey, matrixKeys}
1328 ];
1329 juddOperators[{3, opLabel}] = opMatrix,
1330 {colIndex, 1, Length[normalTable]}
1331 ];
1332 ;
1333 (* special case of t2 in f3 *)
1334 (* this is the same as getting the reduced matrix elements from Judd 1966 *)
1335 numE = 3;
1336 e30p = juddOperators[{3, "e_{3}"}];
1337 t2prime = juddOperators[{3, "t_{2}^{'}"}];
1338 prefactor = 1/(70 Sqrt[2]);
1339 t20p = (# -> (t2prime[#] + prefactor*e30p[#])) & /@ Keys[t2prime];
1340 t20p = Association[t20p];
1341 juddOperators[{3, "t_{2}"}] = t20p;
1342 ;
1343 (*Special case of t11 in f3*)
1344 t11 = juddOperators[{3, "t_{11}"}];
1345 eβprimeOp = juddOperators[{3, "e_{\beta}^{'}"}];
1346 t11primeOp = (# -> (t11[#] + Sqrt[3/385] eβprimeOp[#])) & /@ Keys[t11];
1347 t11primeOp = Association[t11primeOp];
1348 juddOperators[{3, "t_{11}^{'}"}] = t11primeOp;
1349 If[OptionValue["Export"],
1350 (
1351 (*export them*)
1352 PrintTemporary["Exporting ..."];
1353 exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.m"}];
1354 Export[exportFname, juddOperators];
1355 )
1356 ];
1357 Return[juddOperators];
1358 );
1359 ;
1360 ThreeBodyTable::usage="ThreeBodyTable is an association containing the LS-reduced matrix elements for the three-body operators for f^n configurations. The keys are lists of the form {n, SL, SpLp}.";
1361 ThreeBodyTables::usage="ThreeBodyTables is an association whose keys are integers n from 1 to 7 and whose values are associations"

```

```

whose keys are symbols for the different three-body operators, and
whose keys are of the form {LS, LpSp} where LS and LpSp are
strings for LS-terms in f^n.";

1364 GenerateThreeBodyTables::usage = "This function generates the
1365 reduced matrix elements for the three body operators using the
1366 coefficients of fractional parentage, including those beyond f^7."
1367 ;
1368 Options[GenerateThreeBodyTables] = {"Export" -> False};
1369 GenerateThreeBodyTables[OptionsPattern[]] := (
1370   tiKeys = (StringReplace[ToString[#], {"T" -> "t_{", "p" ->
1371     "}"^{"}"] <> "}") & /@ TSymbols;
1372   TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
1373   juddOperators = ParseJudd1984[];
1374   (* op3MatrixElement[SL, SpLp, opSymbol] returns the value for the
1375      reduced matrix element of the operator opSymbol for the terms {SL,
1376      SpLp} in the f^3 configuration. *)
1377   op3MatrixElement[SL_, SpLp_, opSymbol_] := (
1378     jOP = juddOperators[{3, opSymbol}];
1379     key = {SL, SpLp};
1380     val = If[MemberQ[Keys[jOP], key],
1381       jOP[key],
1382       0];
1383     Return[val];
1384   );
1385   (* ti: This is the implementation of formula (2) in Judd & Suskin
1386      1984. It computes the reduced matrix elements of ti in f^n by
1387      using the reduced matrix elements in f^3 and the coefficients of
1388      fractional parentage. If the option \Fast\ is set to True then
1389      the values for n>7 are simply computed as the negatives of the
1390      values in the complementary configuration; this except for t2 and
1391      t11 which are treated as special cases. *)
1392   Options[ti] = {"Fast" -> True};
1393   ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
1394     Module[
1395       {nn, S, L, Sp, Lp,
1396        cfpSL, cfpSpLp,
1397        parentSL, parentSpLp,
1398        tnk, tnks},
1399       (
1400         {S, L} = FindSL[SL];
1401         {Sp, Lp} = FindSL[SpLp];
1402         fast = OptionValue["Fast"];
1403         numH = 14 - nE;
1404         If[fast && Not[MemberQ[{t_{2}, t_{11}}, tiKey]] && nE > 7,
1405           Return[-tktable[{numH, SL, SpLp, tiKey}]];
1406         ];
1407         If[(S == Sp && L == Lp),
1408           (
1409             cfpSL = CFP[{nE, SL}];
1410             cfpSpLp = CFP[{nE, SpLp}];
1411             tnks = Table[(
1412               parentSL = cfpSL[[nn, 1]];
1413               parentSpLp = cfpSpLp[[mm, 1]];
1414               cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
1415               tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
1416             ),
1417               {nn, 2, Length[cfpSL]},
1418               {mm, 2, Length[cfpSpLp]}
1419             ];
1420             tnk = Total[Flatten[tnks]];
1421           ),
1422             tnk = 0;
1423           ];
1424           Return[nE / (nE - opOrder) * tnk];
1425         )
1426       ];
1427       (* Calculate the reduced matrix elements of t^i for n up to 14 *)
1428       tktable = <||>;
1429       Do[(
1430         Do[(
1431           tkValue = Which[numE <= 2,
1432             (*Initialize n=1,2 with zeros*)
1433             0,
1434             numE == 3,
1435             (* Grab matrix elem in f^3 from Judd 1984 *)
1436           ];
1437           tktable = Append[tktable, tkValue];
1438         ];
1439         numE++;
1440       ];
1441     ];

```

```

1424     SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],  

1425     True,  

1426     SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}",  

1427     2, 3]]]  

1427     ];  

1428     tktable[{numE, SL, SpLp, opKey}] = tkValue;  

1429   ),  

1430   {SL, AllowedNKSLTerms[numE]},  

1431   {SpLp, AllowedNKSLTerms[numE]},  

1432   {opKey, Append[tiKeys, "e_{3}"]}  

1433 ];
1434 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE],  

1435 configuration complete"]];
1435 ),
1436 {numE, 1, 14}
1437 ];
1438
1439 (* Now use those reduced matrix elements to determine their sum  

as weighted by their corresponding strengths Ti *)
1440 ThreeBodyTable = <||>;
1441 Do[
1442 Do[
1443 (
1444   ThreeBodyTable[{numE, SL, SpLp}] = (
1445     Sum[(  

1446       If[tiKey == "t_{2}", t2Switch, 1] *  

1447         tktable[{numE, SL, SpLp, tiKey}] *  

1448           TSymbolsAssoc[tiKey] +  

1449           If[tiKey == "t_{2}", 1 - t2Switch, 0] *  

1450             (-tktable[{14 - numE, SL, SpLp, tiKey}]) *  

1451               TSymbolsAssoc[tiKey]  

1452             ),  

1453           {tiKey, tiKeys}  

1454         ]
1455       );
1456     ),
1457     {SL, AllowedNKSLTerms[numE]},  

1458     {SpLp, AllowedNKSLTerms[numE]}
1459   ];
1460 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix  

1461 complete"]];
1461 {numE, 1, 7}
1462 ];
1463
1464 ThreeBodyTables = Table[(
1465   terms = AllowedNKSLTerms[numE];
1466   singleThreeBodyTable =
1467   Table[
1468     {SL, SLP} -> ThreeBodyTable[{numE, SL, SLP}],
1469     {SL, terms},
1470     {SLP, terms}
1471   ];
1472   singleThreeBodyTable = Flatten[singleThreeBodyTable];
1473   singleThreeBodyTables = Table[(
1474     notNullPosition = Position[TSymbols, notNullSymbol][[1,
1]];
1475     reps = ConstantArray[0, Length[TSymbols]];
1476     reps[[notNullPosition]] = 1;
1477     rep = AssociationThread[TSymbols -> reps];
1478     notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
1479     ),
1480     {notNullSymbol, TSymbols}
1481   ];
1482   singleThreeBodyTables = Association[singleThreeBodyTables];
1483   numE -> singleThreeBodyTables),
1484   {numE, 1, 7}
1485 ];
1486
1487 ThreeBodyTables = Association[ThreeBodyTables];
1488 If[OptionValue["Export"],
1489 (
1490   threeBodyTablefname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
1491   Export[threeBodyTablefname, ThreeBodyTable];
1492   threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];

```

```

1493     Export[threeBodyTablesfname, ThreeBodyTables];
1494   )
1495 ];
1496 Return[{ThreeBodyTable, ThreeBodyTables}];
1497 );
1498
1499 (* ##### Three Body Operators ##### *)
1500 (* ##### ##### ##### ##### ##### *)
1501 (* ##### ##### ##### ##### ##### *)
1502 (* ##### ##### ##### ##### ##### *)
1503 (* ##### ##### ##### ##### Reduced SOO and ECSO ##### *)
1504
1505 ReducedT11inf2::usage = "ReducedT11inf2[SL, SpLp] returns the
1506   reduced matrix element of the scalar component of the double
1507   tensor T11 for the given SL terms SL, SpLp.
1508 Data used here for m0, m2, m4 is from Table II of Judd, BR, HM
1509   Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
1510   Interactions for f Electrons. Physical Review 169, no. 1 (1968):
1511   130.
1512 ";
1513 ReducedT11inf2[SL_, SpLp_] := Module[
1514   {T11inf2},
1515   (
1516     T11inf2 = <|
1517       {"1S", "3P"} -> 6 M0 + 2 M2 + 10/11 M4,
1518       {"3P", "3P"} -> -36 M0 - 72 M2 - 900/11 M4,
1519       {"3P", "1D"} -> -Sqrt[(2/15)] (27 M0 + 14 M2 + 115/11 M4),
1520       {"1D", "3F"} -> Sqrt[2/5] (23 M0 + 6 M2 - 195/11 M4),
1521       {"3F", "3F"} -> 2 Sqrt[14] (-15 M0 - M2 + 10/11 M4),
1522       {"3F", "1G"} -> Sqrt[11] (-6 M0 + 64/33 M2 - 1240/363 M4),
1523       {"1G", "3H"} -> Sqrt[2/5] (39 M0 - 728/33 M2 - 3175/363 M4),
1524       {"3H", "3H"} -> 8/Sqrt[55] (-132 M0 + 23 M2 + 130/11 M4),
1525       {"3H", "1I"} -> Sqrt[26] (-5 M0 - 30/11 M2 - 375/1573 M4)
1526     |>;
1527     Which[
1528       MemberQ[Keys[T11inf2], {SL, SpLp}],
1529         Return[T11inf2[{SL, SpLp}]],
1530       MemberQ[Keys[T11inf2], {SpLp, SL}],
1531         Return[T11inf2[{SpLp, SL}]],
1532         True,
1533           Return[0]
1534     ]
1535   )
1536 ];
1537 Reducedt11inf2::usage = "Reducedt11inf2[SL, SpLp] returns the
1538   reduced matrix element in f^2 of the double tensor operator t11
1539   for the corresponding given terms {SL, SpLp}.
1540 Values given here are those from Table VII of \"Judd, BR, HM
1541   Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
1542   Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
1543   130.\"
1544 ";
1545 Reducedt11inf2[SL_, SpLp_] := Module[
1546   {t11inf2},
1547   (
1548     t11inf2 = <|
1549       {"1S", "3P"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
1550       {"3P", "3P"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
1551       {"3P", "1D"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
1552       {"1D", "3F"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
1553       {"3F", "3F"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
1554       {"3F", "1G"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
1555       {"1G", "3H"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
1556       {"3H", "3H"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
1557       {"3H", "1I"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
1558     |>;
1559     Which[
1560       MemberQ[Keys[t11inf2], {SL, SpLp}],
1561         Return[t11inf2[{SL, SpLp}]],
1562       MemberQ[Keys[t11inf2], {SpLp, SL}],
1563         Return[t11inf2[{SpLp, SL}]],
1564         True,
1565           Return[0]
1566     ]
1567   )

```

```

1559     )
1560   ];
1561 
1562 ReducedSOOandECSOinf2::usage = "ReducedSOOandECSOinf2[SL, SpLp]
1563   returns the reduced matrix element corresponding to the operator (
1564   T11 + t11 - a13 * z13 / 6) for the terms {SL, SpLp}. This
1565   combination of operators corresponds to the spin-other-orbit plus
1566   ECSO interaction.
1567 The T11 operator corresponds to the spin-other-orbit interaction,
1568   and the t11 operator (associated with electrostatically-correlated
1569   spin-orbit) originates from configuration interaction analysis.
1570 To their sum a factor proportional to the operator z13 is
1571   subtracted since its effect is redundant to the spin-orbit
1572   interaction. The factor of 1/6 is not on Judd's 1968 paper, but it
1573   is on \"Chen, Xueyuan, Guokui Liu, Jean Margerie, and Michael F
1574   Reid. \"A Few Mistakes in Widely Used Data Files for Fn
1575   Configurations Calculations.\\" Journal of Luminescence 128, no. 3
1576   (2008): 421-27\".
1577 The values for the reduced matrix elements of z13 are obtained from
1578   Table IX of the same paper. The value for a13 is from table VIII.
1579 Rigorously speaking the Pk parameters here are subscripted. The
1580   conversion to superscripted parameters is performed elsewhere with
1581   the Prescaling replacement rules.
1582 ";
1583 ReducedSOOandECSOinf2[SL_, SpLp_] := Module[
1584   {a13, z13, z13inf2, matElement, redSOOandECSOinf2},
1585   (
1586     a13 = (-33 M0 + 3 M2 + 15/11 M4 -
1587       6 P0 + 3/2 (35 P2 + 77 P4 + 143 P6));
1588     z13inf2 = <|
1589       {"1S", "3P"} -> 2,
1590       {"3P", "3P"} -> 1,
1591       {"3P", "1D"} -> -Sqrt[(15/2)],
1592       {"1D", "3F"} -> Sqrt[10],
1593       {"3F", "3F"} -> Sqrt[14],
1594       {"3F", "1G"} -> -Sqrt[11],
1595       {"1G", "3H"} -> Sqrt[10],
1596       {"3H", "3H"} -> Sqrt[55],
1597       {"3H", "1I"} -> -Sqrt[(13/2)]
1598     |>;
1599     matElement = Which[
1600       MemberQ[Keys[z13inf2], {SL, SpLp}],
1601       z13inf2[{SL, SpLp}],
1602       MemberQ[Keys[z13inf2], {SpLp, SL}],
1603       z13inf2[{SpLp, SL}],
1604       True,
1605       0
1606     ];
1607     redSOOandECSOinf2 = (
1608       ReducedT11inf2[SL, SpLp] +
1609       Reducedt11inf2[SL, SpLp] -
1610       a13 / 6 * matElement
1611     );
1612     redSOOandECSOinf2 = SimplifyFun[redSOOandECSOinf2];
1613     Return[redSOOandECSOinf2];
1614   )
1615 ];
1616 
1617 ReducedSOOandECSOinfn::usage = "ReducedSOOandECSOinfn[numE, SL,
1618   SpLp] calculates the reduced matrix elements of the (spin-other-
1619   orbit + ECSO) operator for the f^numE configuration corresponding
1620   to the terms SL and SpLp. This is done recursively, starting from
1621   tabulated values for f^2 from \"Judd, BR, HM Crosswhite, and
1622   Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
1623   Electrons.\" Physical Review 169, no. 1 (1968): 130.\", and by
1624   using equation (4) of that same paper.
1625 ";
1626 ReducedSOOandECSOinfn[numE_, SL_, SpLp_] := Module[
1627   {spin, orbital, t, S, L, Sp, Lp,
1628    idx1, idx2, cfpSL, cfpSpLp, parentSL,
1629    Sb, Lb, Sbp, Lbp, parentSpLp, funval},
1630   (
1631     {spin, orbital} = {1/2, 3};
1632     {S, L} = FindSL[SL];
1633     {Sp, Lp} = FindSL[SpLp];
1634     t = 1;

```

```

1612     cfpSL      = CFP[{numE, SL}];
1613     cfpSpLp   = CFP[{numE, SpLp}];
1614     funval    = Sum[
1615       (
1616         parentSL = cfpSL[[idx2, 1]];
1617         parentSpLp = cfpSpLp[[idx1, 1]];
1618         {Sb, Lb} = FindSL[parentSL];
1619         {Sbp, Lbp} = FindSL[parentSpLp];
1620         phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1621         (
1622           phase *
1623             cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1624             SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1625             SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1626             S00andECSOLSTable[{numE - 1, parentSL, parentSpLp}]
1627           )
1628         ),
1629         {idx1, 2, Length[cfpSpLp]},
1630         {idx2, 2, Length[cfpSL]}
1631       ];
1632     funval *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1633     Return[funval];
1634   )
1635 ];
1636
1637 GenerateS00andECSOLSTable::usage = "GenerateS00andECSOLSTable[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 \"ReducedS00andECSOLSTable.m\" in the data folder of this module.
The values are also returned as an association.";
1638 Options[GenerateS00andECSOLSTable] = {"Progress" -> True, "Export"
-> True};
1639 GenerateS00andECSOLSTable[nmax_Integer, OptionsPattern[]] := (
1640   If[And[OptionValue["Progress"], frontEndAvailable],
1641     (
1642       numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
1643         numE]]^2, {numE, 1, nmax}]];
1644       counters = Association[Table[numE->0, {numE, 1, nmax}]];
1645       totalIters = Total[Values[numItersai[[1;;nmax]]]];
1646       template1 = StringTemplate["Iteration `numiter` of `totaliter`"]
1647       template2 = StringTemplate["`remtime` min remaining"];
1648       template3 = StringTemplate["Iteration speed = `speed` ms/it"];
1649       template4 = StringTemplate["Time elapsed = `runtime` min"];
1650       progBar = PrintTemporary[
1651         Dynamic[
1652           Pane[
1653             Grid[{{
1654               Superscript["f", numE]}, {
1655                 template1 <|"numiter" -> numiter, "totaliter" ->
1656                   totalIters |>}},
1657                   {template4 <|"runtime" -> Round[QuantityMagnitude[
1658                     UnitConvert[(Now - startTime), "min"]], 0.1] |>},
1659                   {template2 <|"remtime" -> Round[QuantityMagnitude[
1660                     UnitConvert[(Now - startTime)/(numiter)*(totalIters - numiter), "min"]
1661                   ], 0.1] |>},
1662                   {template3 <|"speed" -> Round[QuantityMagnitude[Now
1663                     - startTime, "ms"]/(numiter), 0.01] |>}, {ProgressIndicator[Dynamic
1664                     [numiter], {1, totalIters}]}
1665                   },
1666                   ],
1667                   Frame -> All
1668                 ],
1669                 Full,
1670                 Alignment -> Center
1671               ]
1672             ]
1673           ];
1674         ];
1675       S00andECSOLSTable = <||>;
1676       numiter = 1;

```

```

1669 startTime = Now;
1670 Do[
1671 (
1672     numiter+= 1;
1673     S00andECSOLSTable[{numE, SL, SpLp}] = Which[
1674         numE==1,
1675         0,
1676         numE==2,
1677         SimplifyFun[ReducedS00andECSOinf2[SL, SpLp]],
1678         True,
1679         SimplifyFun[ReducedS00andECSOinfn[numE, SL, SpLp]]
1680     ];
1681 ),
1682 {numE, 1, nmax},
1683 {SL, AllowedNKSLTerms[numE]},
1684 {SpLp, AllowedNKSLTerms[numE]}
1685 ];
1686 If[And[OptionValue["Progress"], frontEndAvailable],
1687     NotebookDelete[progBar];
1688 If[OptionValue["Export"],
1689     (fname = FileNameJoin[{moduleDir, "data", "ReducedS00andECSOLSTable.m"}];
1690     Export[fname, S00andECSOLSTable];
1691     )
1692 ];
1693 Return[S00andECSOLSTable];
1694 );
1695 (* ##### Reduced S00 and ECSO ##### *)
1696 (* ##### ##### ##### ##### ##### *)
1697 (* ##### ##### ##### ##### ##### *)
1698 (* ##### ##### ##### ##### ##### *)
1699 (* ##### ##### ##### ##### ##### *)
1700 (* ##### ##### ##### ##### Spin-Spin ##### *)
1701
1702 ReducedT22inf2::usage = "ReducedT22inf2[SL, SpLp] returns the
1703     reduced matrix element of the scalar component of the double
1704     tensor T22 for the terms SL, SpLp in f^2.
1705 Data used here for m0, m2, m4 is from Table I of Judd, BR, HM
1706     Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
1707     Interactions for f Electrons. Physical Review 169, no. 1 (1968):
1708     130.
1709 ";
1710 ReducedT22inf2[SL_, SpLp_] := Module[
1711     {statePosition, PsiPsipStates, m0, m2, m4, Tk2m},
1712     (
1713         T22inf2 = <|
1714             {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
1715             {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
1716             {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
1717             {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
1718             {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
1719         |>;
1720         Which[
1721             MemberQ[Keys[T22inf2], {SL, SpLp}],
1722                 Return[T22inf2[{SL, SpLp}]],
1723             MemberQ[Keys[T22inf2], {SpLp, SL}],
1724                 Return[T22inf2[{SpLp, SL}]],
1725             True,
1726                 Return[0]
1727             ]
1728         )
1729     ];
1730
1731 ReducedT22infn::usage = "ReducedT22infn[n, SL, SpLp] calculates the
1732     reduced matrix element of the T22 operator for the f^n
1733     configuration corresponding to the terms SL and SpLp.
1734 This is done by using equation (4) of \"Judd, BR, HM Crosswhite,
1735     and Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
1736     Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
1737 ";
1738 ReducedT22infn[numE_, SL_, SpLp_] := Module[
1739     {spin, orbital, t, idx1, idx2, S, L,
1740     Sp, Lp, cfpSL, cfpSpLp, parentSL,
1741     parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
1742     (
1743         {spin, orbital} = {1/2, 3};

```

```

1735 {S, L} = FindSL[SL];
1736 {Sp, Lp} = FindSL[SpLp];
1737 t = 2;
1738 cfpSL = CFP[{numE, SL}];
1739 cfpSpLp = CFP[{numE, SpLp}];
1740 Tnkk = Sum[(
1741   parentSL = cfpSL[[idx2, 1]];
1742   parentSpLp = cfpSpLp[[idx1, 1]];
1743   {Sb, Lb} = FindSL[parentSL];
1744   {Sbp, Lbp} = FindSL[parentSpLp];
1745   phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1746   (
1747     phase *
1748     cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1749     SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1750     SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1751     T22Table[{numE - 1, parentSL, parentSpLp}]
1752   )
1753 ),
1754 {idx1, 2, Length[cfpSpLp]},
1755 {idx2, 2, Length[cfpSL]}
1756 ];
1757 Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1758 Return[Tnkk];
1759 )
1760 ];
1761
1762 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.
1763 This is an intermediate step to the calculation of the reduced
matrix elements of the spin-spin operator.";
1764 Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
1765 GenerateT22Table[nmax_Integer, OptionsPattern[]] := (
1766   If[And[OptionValue["Progress"], frontEndAvailable],
1767     (
1768       numItersai = Association[Table[numE -> Length[AllowedNKSLTerms[
1769         numE]]^2, {numE, 1, nmax}]];
1770       counters = Association[Table[numE -> 0, {numE, 1, nmax}]];
1771       totalIters = Total[Values[numItersai[[1;;nmax]]]];
1772       template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
1773       template2 = StringTemplate["`remtime` min remaining"];
1774       template3 = StringTemplate["Iteration speed = `speed` ms/it"];
1775       template4 = StringTemplate["Time elapsed = `runtime` min"];
1776       progBar = PrintTemporary[
1777         Dynamic[
1778           Pane[
1779             Grid[{{Superscript["f", numE]}, {
1780               template1 <|"numiter" -> numiter, "totaliter" ->
1781               totalIters |>}},
1782               {template4 <|"runtime" -> Round[QuantityMagnitude[
1783                 UnitConvert[(Now - startTime), "min"]], 0.1] |>},
1784               {template2 <|"remtime" -> Round[QuantityMagnitude[
1785                 UnitConvert[(Now - startTime)/(numiter)*(totalIters - numiter), "min"]], 0.1] |>},
1786               {template3 <|"speed" -> Round[QuantityMagnitude[Now -
1787                 startTime, "ms"]/(numiter), 0.01] |>},
1788               {ProgressIndicator[Dynamic[numiter], {1,
1789                 totalIters}]}],
1790               Frame -> All],
1791               Full,
1792               Alignment -> Center]
1793             ]
1794           ];
1795       ];
1796       T22Table = <||>;
1797       startTime = Now;
1798       numiter = 1;

```

```

1793 Do [
1794   (
1795     numiter+= 1;
1796     T22Table[{numE, SL, SpLp}] = Which[
1797       numE==1,
1798       0,
1799       numE==2,
1800       SimplifyFun[ReducedT22inf2[SL, SpLp]],
1801       True,
1802       SimplifyFun[ReducedT22infn[numE, SL, SpLp]]
1803     ];
1804   ),
1805   {numE, 1, nmax},
1806   {SL, AllowedNKSLTerms[numE]},
1807   {SpLp, AllowedNKSLTerms[numE]}
1808 ];
1809 If[And[OptionValue["Progress"], frontEndAvailable],
1810  NotebookDelete[progBar]
1811 ];
1812 If[OptionValue["Export"],
1813  (
1814    fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
1815    Export[fname, T22Table];
1816  )
1817 ];
1818 Return[T22Table];
1819 );
1820
1821 SpinSpin::usage = "SpinSpin[n, SL, SpLp, J] returns the matrix
element <|SL,J|spin-spin|SpLp,J|> for the spin-spin operator
within the configuration f^n. This matrix element is independent
of MJ. This is obtained by querying the relevant reduced matrix
element from the association T22Table, putting in the adequate
phase, and 6-j symbol.
1822 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.\""
1823 .
1824 ";
1825 SpinSpin[numE_, SL_, SpLp_, J_] := Module[
1826   {S, L, Sp, Lp, α, val},
1827   (
1828     α = 2;
1829     {S, L} = FindSL[SL];
1830     {Sp, Lp} = FindSL[SpLp];
1831     val = (
1832       Phaser[Sp + L + J] *
1833       SixJay[{Sp, Lp, J}, {L, S, α}] *
1834       T22Table[{numE, SL, SpLp}]
1835     );
1836     Return[val]
1837   )
1838 ];
1839
1840 GenerateSpinSpinTable::usage = "GenerateSpinSpinTable[nmax]
generates the reduced matrix elements in the |LSJ> basis for the
spin-spin 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.";
1841 Options[GenerateSpinSpinTable] = {"Export" -> False};
1842 GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
1843  (
1844   SpinSpinTable = <||>;
1845   PrintTemporary[Dynamic[numE]];
1846   Do[
1847     SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp
1848     , J]),
1849     {numE, 1, nmax},
1850     {J, MinJ[numE], MaxJ[numE]},
1851     {SL, First /@ AllowedNKSLforJTerms[numE, J]},
1852     {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
1853   ]
1854 );

```

```

1852 ];
1853 If[OptionValue["Export"],
1854 (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
1855 Export[fname, SpinSpinTable];
1856 )
1857 ];
1858 Return[SpinSpinTable];
1859 );
1860
1861 (* ##### Spin-Spin ##### *)
1862 (* ##### *)
1863
1864 (*
1865 ##### *)
1866 (* ## Spin-Other-Orbit and Electrostatically-Correlated-Spin-Orbit
1867 ## *)
1868
1869 S00andECSO::usage = "S00andECSO[n, SL, SpLp, J] returns the matrix
1870 element <|SL,J|spin-spin|SpLp,J|> for the combined effects of the
1871 spin-other-orbit interaction and the electrostatically-correlated-
1872 spin-orbit (which originates from configuration interaction
1873 effects) within the configuration f~n. This matrix element is
1874 independent of MJ. This is obtained by querying the relevant
1875 reduced matrix element by querying the association
1876 S00andECSOLSTable and putting in the adequate phase and 6-j symbol
1877 . The S00andECSOLSTable puts together the reduced matrix elements
1878 from three operators.
1879
1880 This is calculated according to equation (3) in \"Judd, BR, HM
1881 Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
1882 Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
1883 130.\".
1884 ";
1885 S00andECSO[numE_, SL_, SpLp_, J_] := Module[
1886 {S, Sp, L, Lp, α, val},
1887 (
1888 α = 1;
1889 {S, L} = FindSL[SL];
1890 {Sp, Lp} = FindSL[SpLp];
1891 val = (
1892 Phaser[Sp + L + J] *
1893 SixJay[{Sp, Lp, J}, {L, S, α}] *
1894 S00andECSOLSTable[{numE, SL, SpLp}]
1895 );
1896 Return[val];
1897 )
1898 ];
1899 Prescaling = {P2 -> P2/225, P4 -> P4/1089, P6 -> 25 * P6 / 184041};
1900
1901 GenerateS00andECSOTable::usage = "GenerateS00andECSOTable[nmax]
1902 generates the reduced matrix elements in the |LSJ> basis for the (
1903 spin-other-orbit + electrostatically-correlated-spin-orbit)
1904 operator. It returns an association where the keys are of the form
1905 {n, SL, SpLp, J}. If the option \"Export\" is set to True then
1906 the resulting object is saved to the data folder. Since this is a
1907 scalar operator, there is no MJ dependence. This dependence only
1908 comes into play when the crystal field contribution is taken into
1909 account.";
1910 Options[GenerateS00andECSOTable] = {"Export" -> False};
1911 GenerateS00andECSOTable[nmax_, OptionsPattern[]] := (
1912 S00andECSOTable = <||>;
1913 Do[
1914 S00andECSOTable[{numE, SL, SpLp, J}] = (S00andECSO[numE, SL,
1915 SpLp, J] /. Prescaling);,
1916 {numE, 1, nmax},
1917 {J, MinJ[numE], MaxJ[numE]},
1918 {SL, First /@ AllowedNKSLforJTerms[numE, J]},
1919 {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
1920 ];
1921 If[OptionValue["Export"],
1922 (
1923 fname = FileNameJoin[{moduleDir, "data", "S00andECSOTable.m"}];
1924 Export[fname, S00andECSOTable];
1925 )
1926 ];
1927 ]

```

```

1904     Return[S00andECSOTable];
1905   );
1906
1907 (* ## Spin-Other-Orbit and Electrostatically-Correlated-Spin-Orbit
1908   ## *)
1909 (*
1910   ##### Magnetic Interactions #####
1911   *)
1912
1913 MagneticInteractions::usage = "MagneticInteractions[{numE, SL, SLP,
1914   J}] returns the matrix element of the magnetic interaction
1915   between the terms SL and SLP in the f^numE configuration for the
1916   given value of J. The interaction is given by the sum of the spin-
1917   spin, the spin-other-orbit, and the electrostatically-correlated-
1918   spin-orbit interactions.
1919 The part corresponding to the spin-spin interaction is provided by
1920   SpinSpin[{numE, SL, SLP, J}].
1921 The part corresponding to S00 and ECSO is provided by the function
1922   S00andECSO[{numE, SL, SLP, J}].
1923 The option \"ChenDeltas\" can be used to include or exclude the
1924   Chen deltas from the calculation. The default is to exclude them.
1925   If this option is used, then the chenDeltas association needs to
1926   be loaded into the session with LoadChenDeltas[].";
1927 Options[MagneticInteractions] = {"ChenDeltas" -> False};
1928 MagneticInteractions[{numE_, SL_, SLP_, J_}, OptionsPattern[]] :=
1929 Module[
1930   {key, ss, sooandecso, total,
1931   S, L, Sp, Lp, phase, sixjay,
1932   M0v, M2v, M4v,
1933   P2v, P4v, P6v},
1934   (
1935     key      = {numE, SL, SLP, J};
1936     ss       = \[Sigma]SS * SpinSpinTable[key];
1937     sooandecso = S00andECSOTable[key];
1938     total = ss + sooandecso;
1939     total = SimplifyFun[total];
1940     If[
1941       Not[OptionValue["ChenDeltas"]],
1942       Return[total]
1943     ];
1944     (* In the type A errors the wrong values are different *)
1945     If[MemberQ[Keys[chenDeltas["A"]], {numE, SL, SLP}],
1946       (
1947         {S, L}    = FindSL[SL];
1948         {Sp, Lp} = FindSL[SLP];
1949         phase    = Phaser[Sp + L + J];
1950         sixjay   = SixJay[{Sp, Lp, J}, {L, S, 1}];
1951         {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SL,
1952           SLP}]["wrong"];
1953         total    = (
1954           phase * sixjay *
1955             (
1956               M0v*M0 + M2v*M2 + M4v*M4 +
1957               P2v*P2 + P4v*P4 + P6v*P6
1958             )
1959           );
1960         total   = wChErrA * total + (1 - wChErrA) * (ss +
1961           sooandecso)
1962         );
1963       ];
1964     (* In the type B errors the wrong values are zeros all around
1965     *)
1966     If[MemberQ[chenDeltas["B"], {numE, SL, SLP}],
1967       (
1968         total  = (1 - wChErrB) * (ss + sooandecso)
1969       )
1970     ];
1971     Return[total];
1972   )
1973 ];
1974
1975 (* ##### Magnetic Interactions #####
1976   *)
1977 (* ##### Magnetic Interactions #####
1978   *)

```

```

1963 (* ##### Free-Ion Energies ##### *)
1964 (* ##### Free-Ion Energies ##### *)
1965
1966
1967 GenerateFreeIonTable::usage="GenerateFreeIonTable[] generates an
   association for free-ion energies in terms of Slater integrals Fk
   and spin-orbit parameter  $\zeta$ . It returns an association where the
   keys are of the form {nE, SL, SpLp}. If the option \"Export\" is
   set to True then the resulting object is saved to the data folder.
   The free-ion Hamiltonian is the sum of the electrostatic and spin-
   orbit interactions. The electrostatic interaction is given by the
   function Electrostatic[{numE, SL, SpLp}] and the spin-orbit
   interaction is given by the function SpinOrbitTable[{numE, SL,
   SpLp}]. The values for the electrostatic interaction are taken
   from the data file ElectrostaticTable.m and the values for the
   spin-orbit interaction are taken from the data file SpinOrbitTable
   .m. The values for the free-ion Hamiltonian are then exported to a
   file \"FreeIonTable.m\" in the data folder of this module. The
   values are also returned as an association.";
1968 Options[GenerateFreeIonTable] = {"Export" -> False};
1969 GenerateFreeIonTable[OptionsPattern[]] := Module[
1970   {terms, numEH, zetaSign, fname, FreeIonTable},
1971   (
1972     If[Not[ValueQ[ElectrostaticTable]],
1973       LoadElectrostatic[]
1974     ];
1975     If[Not[ValueQ[SpinOrbitTable]],
1976       LoadSpinOrbit[]
1977     ];
1978     If[Not[ValueQ[ReducedUkTable]],
1979       LoadUk[]
1980     ];
1981     FreeIonTable = <||>;
1982     Do[
1983       (
1984         terms = AllowedNKSLJTerms[nE];
1985         numEH = Min[nE, 14 - nE];
1986         zetaSign = If[nE > 7, -1, 1];
1987         Do[
1988           FreeIonTable[{nE, term[[1]], term[[2]]}] = (
1989             Electrostatic[{numEH, term[[1]], term[[1]]}] +
1990               zetaSign * SpinOrbitTable[{numEH, term[[1]], term
1991               [[1]], term[[2]]}]
1992             ),
1993             {term, terms}];
1994           ), {nE, 1, 14}
1995         ];
1996         If[OptionValue["Export"],
1997           (
1998             fname = FileNameJoin[{moduleDir, "data", "FreeIonTable.m"
1999           }];
2000             Export[fname, FreeIonTable];
2001           )
2002         ];
2003         Return[FreeIonTable];
2004       );
2005     ];
2006     LoadFreeIon::usage = "LoadFreeIon[] loads the free-ion energies
2007       from the data folder. The values are stored in the association
2008       FreeIonTable.\";";
2009     LoadFreeIon[] := (
2010       If[ValueQ[FreeIonTable],
2011         Return[]
2012       ];
2013       PrintTemporary["Loading the association of free-ion energies ..."];
2014       FreeIonTableFname = FileNameJoin[{moduleDir, "data", "FreeIonTable.m"}];
2015       FreeIonTable = If[!FileExistsQ[FreeIonTableFname],
2016         (
2017           PrintTemporary[">> FreeIonTable.m not found, generating ..."];
2018           GenerateFreeIonTable["Export" -> True]
2019         ),
2020       ];
2021     );
2022   );
2023 
```

```

2018     Import[FreeIonTableFname]
2019   ];
2020 );
2021
2022 (* ##### Free-Ion Energies ##### *)
2023 (* ##### Crystal Field ##### *)
2024
2025 (* ##### Crystal Field ##### *)
2026 (* ##### Crystal Field ##### *)
2027
2028 Cqk::usage = "Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_]. In
2029   Wybourne (1965) see equations 6-3, 6-4, and 6-5. Also in TASS see
2030   equation 11.53.";
2031 Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
2032   {S, Sp, L, Lp, orbital, val},
2033   (
2034     orbital = 3;
2035     {S, L} = FindSL[NKSL];
2036     {Sp, Lp} = FindSL[NKSLp];
2037     f1 = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
2038     val =
2039       If[f1==0,
2040         0,
2041         (
2042           f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
2043           If[f2==0,
2044             0,
2045             (
2046               f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
2047               If[f3==0,
2048                 0,
2049                 (
2050                   Phaser[J - M + S + Lp + J + k] *
2051                     Sqrt[TPO[J, Jp]] *
2052                       f1 *
2053                         f2 *
2054                           f3 *
2055                             Ck[orbital, k]
2056                           )
2057                         )
2058                       ]
2059                     ]
2060                   ]
2061                 ];
2062               Return[val];
2063             )
2064           ];
2065
2066 Bqk::usage = "Real part of the Bqk coefficients.";
2067 Bqk[q_, 2] := {B02/2, B12, B22}[[q + 1]];
2068 Bqk[q_, 4] := {B04/2, B14, B24, B34, B44}[[q + 1]];
2069 Bqk[q_, 6] := {B06/2, B16, B26, B36, B46, B56, B66}[[q + 1]];
2070
2071 Sqk::usage = "Imaginary part of the Bqk coefficients.";
2072 Sqk[q_, 2] := {0, S12, S22}[[q + 1]];
2073 Sqk[q_, 4] := {0, S14, S24, S34, S44}[[q + 1]];
2074 Sqk[q_, 6] := {0, S16, S26, S36, S46, S56, S66}[[q + 1]];
2075
2076 CrystalField::usage = "CrystalField[n, NKSL, J, M, NKSLp, Jp, Mp]
2077   calculates the matrix element of the crystal field in terms of Bqk
2078   and Sqk parameters for configuration f^numE. It is calculated as
2079   an association with keys of the form {n, NKSL, J, M, NKSLp, Jp, Mp
2080   }.";
2081 CrystalField[numE_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := (
2082   Sum[
2083     (
2084       cqk = Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp];
2085       cmqk = Cqk[numE, -q, k, NKSL, J, M, NKSLp, Jp, Mp];
2086       Bqk[q, k] * (cqk + (-1)^q * cmqk) +
2087         I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
2088       ),
2089     {k, {2, 4, 6}},
2090     {q, 0, k}
2091   ]

```

```

2088 )
2089
2090 TotalCFIter::usage = "TotalIter[i_, j_] returns total number of
2091   function evaluations for calculating all the matrix elements for
2092   the  $\forall (\forall \text{SuperscriptBox}[(f), (i)])$  to the  $\forall (\forall \text{SuperscriptBox}[(f), (j)])$  configurations.";
2093 TotalCFIter[i_, j_] := (
2094   numIter = {196, 8281, 132496, 1002001, 4008004, 9018009,
2095   11778624};
2096   Return[Total[numIter[[i ;; j]]]];
2097 )
2098
2099 GenerateCrystalFieldTable::usage = "GenerateCrystalFieldTable[{"
2100   numEs}] computes the matrix values for the crystal field
2101   interaction for  $f^n$  configurations the given list of numE in
2102   numEs. The function calculates the association CrystalFieldTable
2103   with keys of the form {numE, NKSL, J, M, NKSLP, Jp, Mp}. If the
2104   option "Export" is set to True, then the result is exported to
2105   the data subfolder for the folder in which this package is in. If
2106   the option "Progress" is set to True then an interactive
2107   progress indicator is shown. If "Compress" is set to true the
2108   exported values are compressed when exporting.";
2109 Options[GenerateCrystalFieldTable] = {"Export" -> False, "Progress" -
2110   -> True, "Compress" -> True, "Overwrite" -> False};
2111 GenerateCrystalFieldTable[numEs_List:{1,2,3,4,5,6,7},
2112   OptionsPattern[]] := (
2113   ExportFun =
2114   If[OptionValue["Compress"],
2115     ExportMZip,
2116     Export
2117   ];
2118   numIter = 1;
2119   template1 = StringTemplate["Iteration 'numIter' of 'totalIter'"];
2120   template2 = StringTemplate["'remTime' min remaining"];
2121   template3 = StringTemplate["Iteration speed = 'speed' ms/it"];
2122   template4 = StringTemplate["Time elapsed = 'runtime' min"];
2123   totalIter = Total[TotalCFIter[#, #] & /@ numEs];
2124   freebies = 0;
2125   startTime = Now;
2126   If[And[OptionValue["Progress"], frontEndAvailable],
2127     progBar = PrintTemporary[
2128       Dynamic[
2129         Pane[
2130           Grid[
2131             {
2132               {Superscript["f", numE]},
2133               {template1[<|"numIter" -> numIter, "totalIter" ->
2134             totalIter|>]},
2135               {template4[<|"runtime" -> Round[QuantityMagnitude[
2136             UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
2137               {template2[<|"remTime" -> Round[QuantityMagnitude[
2138             UnitConvert[(Now - startTime)/(numIter - freebies) * (totalIter -
2139             numIter), "min"]], 0.1]|>]},
2140               {template3[<|"speed" -> Round[QuantityMagnitude[Now -
2141             startTime, "ms"]/(numIter - freebies), 0.01]|>]},
2142               {ProgressIndicator[Dynamic[numIter], {1, totalIter}]}
2143             },
2144             Frame -> All
2145           ],
2146           Full,
2147           Alignment -> Center
2148         ]
2149       ]
2150     ];
2151   ];
2152   Do[
2153     (
2154       exportFname = FileNameJoin[{moduleDir, "data",
2155         CrystalFieldTable_f}<>ToString[numE]<>".m"];
2156       If[And[FileExistsQ[exportFname], Not[OptionValue["Overwrite"]]],
2157         Echo["File exists, skipping ..."];
2158         numIter+= TotalCFIter[numE, numE];
2159         freebies+= TotalCFIter[numE, numE];
2160         Continue[];
2161       ];
2162     ];
2163   ];
2164 
```

```

2142     CrystalFieldTable = <||>;
2143     Do[
2144       (
2145         numIter+=1;
2146         {S,L} = FindSL[NKSL];
2147         {Sp,Lp} = FindSL[NKSLp];
2148         CrystalFieldTable[{numE,NKSL,J,M,NKSLp,Jp,Mp}] =
2149           Which[
2150             Abs[M-Mp]>6,
2151             0,
2152             Abs[S-Sp]!=0,
2153             0,
2154             True,
2155             CrystalField[numE,NKSL,J,M,NKSLp,Jp,Mp]
2156           ];
2157       ),
2158       {J, MinJ[numE], MaxJ[numE]},
2159       {Jp, MinJ[numE], MaxJ[numE]},
2160       {M, AllowedMforJ[J]},
2161       {Mp, AllowedMforJ[Jp]},
2162       {NKSL, First/@AllowedNKSLforJTerms[numE,J]},
2163       {NKSLp, First/@AllowedNKSLforJTerms[numE,Jp]}
2164     ];
2165     If[And[OptionValue["Progress"],frontEndAvailable],
2166       NotebookDelete[progBar]
2167     ];
2168     If[OptionValue["Export"],
2169     (
2170       Echo["Exporting to file " <> ToString[exportFname]];
2171       ExportFun[exportFname, CrystalFieldTable];
2172     )
2173   ];
2174   ),
2175   {numE, numEs}
2176 ]
2177 )
2178
2179 ParseBenelli2015::usage="ParseBenelli2015[] parses the data from
file /data/benelli_and_gatteschi_table3p3.csv which corresponds to
Table 3.3 of Benelli and Gatteschi, Introduction to Molecular.
This data provides the form that the crystal field has under
different point group symmetries. This function parses that data
into an association with keys equal to strings representing any of
the 32 crystallographic point groups.";
Options[ParseBenelli2015] = {"Export" -> False};
ParseBenelli2015[OptionsPattern[]] := Module[
2182   {fname, crystalSym,
2183   crystalSymmetries, parseFun,
2184   chars, qk, groupName, family,
2185   groupNum, params},
2186   (
2187     fname      = FileNameJoin[{moduleDir,"data",
2188     benelli_and_gatteschi_table3p3.csv"}];
2189     crystalSym = Import[fname][[2;;33]];
2190     crystalSymmetries = <||>;
2191     parseFun[txt_] := (
2192       chars = Characters[txt];
2193       qk    = chars[[-2;;]];
2194       If[chars[[1]]=="R",
2195         (
2196           Return[{ToExpression@StringJoin[{"B",qk[[1]],qk[[2]]}]}]
2197         ),
2198         (
2199           If[qk[[1]]=="O",
2200             Return[{ToExpression@StringJoin[{"B",qk[[1]],qk[[2]]}]}]
2201           ];
2202           Return[{
2203             ToExpression@StringJoin[{"B",qk[[1]],qk[[2]]}],
2204             ToExpression@StringJoin[{"S",qk[[1]],qk[[2]]}]
2205           }]
2206         );
2207       Do[
2208         (
2209           groupNum = Round@ToExpression@row[[1]];
2210           groupName = row[[2]];

```

```

2211 family      = row[[3]];
2212 params      = Select[row[[4;;]], And[FreeQ[#, Missing], #!=!""]&];
2213 params      = parseFun/@params;
2214 params      = <|"BqkSqk" -> Sort@Flatten[params],
2215 "aliases" -> {groupNum},
2216 "constraints" -> {}|>;
2217 If[MemberQ[{"T", "Th", "O", "Td", "Oh"}, groupName],
2218 params["constraints"] = {
2219     {B44 -> Sqrt[5/14] B04, B46 -> -Sqrt[7/2] B06},
2220     {B44 -> -Sqrt[5/14] B04, B46 -> Sqrt[7/2] B06}
2221 }
2222 ];
2223 If[StringContainsQ[groupName, ", "],
2224 (
2225     {alias1, alias2} = StringSplit[groupName, ", "];
2226     crystalSymmetries[alias1] = params;
2227     crystalSymmetries[alias1]["aliases"] = {groupNum, alias2};
2228     crystalSymmetries[alias2] = params;
2229     crystalSymmetries[alias2]["aliases"] = {groupNum, alias1};
2230 ),
2231 (
2232     crystalSymmetries[groupName] = params;
2233 )
2234 ]
2235 ),
2236 {row, crystalSym}];
2237 crystalSymmetries["source"] = "Benelli and Gatteschi, 2015,
Introduction to Molecular Magnetism, table 3.3.";
2238 If[OptionValue["Export"],
2239     Export[FileNameJoin[{moduleDir, "data",
2240     crystalFieldFunctionalForms.m"}], crystalSymmetries];
2241 ];
2242 Return[crystalSymmetries];
2243 ];
2244
2245 CrystalFieldForm::usage = "CrystalFieldForm[symmetryGroup] returns
an association that describes the crystal field parameters that
are necessary to describe a crystal field for the given symmetry
group.
2246
2247 The symmetry group must be given as a string in Schoenflies
notation and must be one of C1, Ci, S2, Cs, C1h, C2, C2h, C2v, D2,
D2h, S4, C4, C4h, D2d, C4v, D4, D4h, C3, S6, C3h, C3v, D3, D3d,
D3h, C6, C6h, C6v, D6, D6h, T, Th, Td, O, Oh.
2248
2249 The returned association has three keys:
2250     \"BqkSqk\" whose values is a list with the nonzero Bqk and Sqk
parameters;
2251     \"constraints\" whose value is either an empty list, or a lists
of replacements rules that are constraints on the Bqk and Sqk
parameters;
2252     \"simplifier\" whose value is an association that can be used to
set to zero the crystal field parameters that are zero for the
given symmetry group;
2253     \"aliases\" whose value is a list with the integer by which the
point group is also known for and an alternate Schoenflies symbol
if it exists.
2254
2255 This uses data from table 3.3 in Benelli and Gatteschi, 2015.";
2256 CrystalFieldForm[symmetryGroupString_] := (
2257     If[Not@ValueQ[crystalFieldFunctionalForms],
2258         crystalFieldFunctionalForms = Import[FileNameJoin[{moduleDir, "data",
2259         "crystalFieldFunctionalForms.m"}]];
2260         ];
2261         cfForm = crystalFieldFunctionalForms[symmetryGroupString];
2262         simplifier = Association[(# -> 0) &/@ Complement[cfSymbols,
2263         cfForm["BqkSqk"]]];
2264         Return[Join[cfForm, <|"simplifier" -> simplifier|>]];
2265     )
2266
2267 (* ##### Crystal Field ##### *)
2268 (* ##### Configuration-Interaction via Casimir Operators ##### *)
2269 (* ###### Configuration-Interaction via Casimir Operators ##### *)

```

```

2270
2271 CasimirS03::usage = "CasimirS03[{SL, SpLp}] returns LS reduced
2272     matrix element of the configuration interaction term corresponding
2273     to the Casimir operator of R3.";
2274 CasimirS03[{SL_, SpLp_}] := (
2275     {S, L} = FindSL[SL];
2276     If[SL == SpLp,
2277         α * L * (L + 1),
2278         0
2279     ]
2280 )
2281
2282 GG2U::usage = "GG2U is an association whose keys are labels for the
2283     irreducible representations of group G2 and whose values are the
2284     eigenvalues of the corresponding Casimir operator.
2285 Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\",
2286     table 2-6.";
2287 GG2U = Association[
2288     "00" -> 0,
2289     "10" -> 6/12 ,
2290     "11" -> 12/12 ,
2291     "20" -> 14/12 ,
2292     "21" -> 21/12 ,
2293     "22" -> 30/12 ,
2294     "30" -> 24/12 ,
2295     "31" -> 32/12 ,
2296     "40" -> 36/12}
2297 ];
2298
2299 CasimirG2::usage = "CasimirG2[{SL, SpLp}] returns LS reduced matrix
2300     element of the configuration interaction term corresponding to
2301     the Casimir operator of G2.";
2302 CasimirG2[{SL_, SpLp_}] := (
2303     Ulabel = FindNKLSTerm[SL][[1]][[4]];
2304     If[SL==SpLp,
2305         β * GG2U[Ulabel],
2306         0
2307     ]
2308 )
2309
2310 GS07W::usage = "GS07W is an association whose keys are labels for
2311     the irreducible representations of group R7 and whose values are
2312     the eigenvalues of the corresponding Casimir operator.
2313 Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\",
2314     table 2-7.";
2315 GS07W := Association[
2316     {
2317         "000" -> 0,
2318         "100" -> 3/5,
2319         "110" -> 5/5,
2320         "111" -> 6/5,
2321         "200" -> 7/5,
2322         "210" -> 9/5,
2323         "211" -> 10/5,
2324         "220" -> 12/5,
2325         "221" -> 13/5,
2326         "222" -> 15/5
2327     }
2328 ];
2329
2330 CasimirS07::usage = "CasimirS07[{SL, SpLp}] returns the LS reduced
2331     matrix element of the configuration interaction term corresponding
2332     to the Casimir operator of R7.";
2333 CasimirS07[{SL_, SpLp_}] := (
2334     Wlabel = FindNKLSTerm[SL][[1]][[3]];
2335     If[SL==SpLp,
2336         γ * GS07W[Wlabel],
2337         0
2338     ]
2339 )
2340
2341 ElectrostaticConfigInteraction::usage =
2342     ElectrostaticConfigInteraction[numE, {SL, SpLp}] returns the
2343     matrix element for configuration interaction as approximated by
2344     the Casimir operators of the groups R3, G2, and R7. SL and SpLp
2345     are strings that represent terms under LS coupling.";
```

```

2330 ElectrostaticConfigInteraction[numE_, {SL_, SpLp_}] := Module[
2331   {S, L, val},
2332   (
2333     If [
2334       Or[numE == 1, numE==13],
2335       Return[0];
2336     ];
2337     {S, L} = FindSL[SL];
2338     val = (
2339       If[SL == SpLp,
2340         CasimirS03[{SL, SL}] +
2341         CasimirS07[{SL, SL}] +
2342         CasimirG2[{SL, SL}],
2343         0
2344       ]
2345     );
2346     ElectrostaticConfigInteraction[numE, {S, L}] = val;
2347     Return[val];
2348   )
2349 ];
2350
(* ##### Configuration-Interaction via Casimir Operators ##### *)
(* ##### Block assembly ##### *)
(* ##### Block assembly ##### *)
2354 (* ##### Block assembly ##### *)
2355 (* ##### Block assembly ##### *)
2356
2357 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.";
2358 Options[JJBlockMatrix] = {"Sparse" -> True, "ChenDeltas" -> False};
2359 JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]] := Module[
2360   [
2361   {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
2362   SLterm, SpLpterm,
2363   MJ, MJp,
2364   subKron, matValue, eMatrix},
2365   (
2366     NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
2367     NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
2368     eMatrix =
2369     Table[
2370       (*Condition for a scalar matrix op*)
2371       SLterm = NKSLJM[[1]];
2372       SpLpterm = NKSLJMp[[1]];
2373       MJ = NKSLJM[[3]];
2374       MJp = NKSLJMp[[3]];
2375       subKron = (
2376         KroneckerDelta[J, Jp] *
2377         KroneckerDelta[MJ, MJp]
2378       );
2379       matValue =
2380       If[subKron==0,
2381         0,
2382         (
2383           ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2384           ElectrostaticConfigInteraction[numE, {SLterm,
2385             SpLpterm}] +
2386             SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2387             MagneticInteractions[{numE, SLterm, SpLpterm, J}],
2388             "ChenDeltas" -> OptionValue["ChenDeltas"])] +
2389             ThreeBodyTable[{numE, SLterm, SpLpterm}]
2390       )
2391     ];
2392     matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp}
2393   ];
2394     matValue,
2395     {NKSLJMp, NKSLJMps},
2396     {NKSLJM, NKSLJMs}
2397   ];
2398   If[OptionValue["Sparse"],
2399     eMatrix = SparseArray[eMatrix]

```

```

2397 ];
2398 Return[eMatrix]
2399 )
2400 ];
2401
2402 EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some
2403 point may be used to redefine states used in basis.";
2404 EnergyStates[numE_, J_]:= AllowedNKSLJMforJTerms[numE, J];
2405
2406 JJBlockMatrixFileName::usage = "JJBlockMatrixFileName[numE] gives
2407 the filename for the energy matrix table for an atom with numE f-
2408 electrons. The function admits an optional parameter \
2409 FilenameAppendix" which can be used to modify the filename.";
2410 Options[JJBlockMatrixFileName] = {"FilenameAppendix" -> ""};
2411 JJBlockMatrixFileName[numE_Integer, OptionsPattern[]]:= (
2412 fileApp = OptionValue["FilenameAppendix"];
2413 fname = FileNameJoin[{moduleDir,
2414 "hams",
2415 StringJoin[{"f", ToString[numE], "_JJBlockMatrixTable",
2416 fileApp , ".m"}]}];
2417 Return[fname];
2418 );
2419
2420 TabulateJJBlockMatrixTable::usage = "TabulateJJBlockMatrixTable[
2421 numE, CFTable] returns an association JJBlockMatrixTable with keys
2422 equal to lists of the form {numE, J, Jp} and values equal to JJ
2423 blocks of the semi-empirical Hamiltonian. The function admits an
2424 optional parameter "Sparse" which can be used to control whether
2425 the matrix is returned as a SparseArray or not. The default is
2426 True. Another admitted option is "ChenDeltas" which can be used
2427 to include or exclude the Chen deltas from the calculation. The
2428 default is to exclude them. If this option is used, then the
2429 chenDeltas association needs to be loaded into the session with
2430 LoadChenDeltas[].";
2431 Options[TabulateJJBlockMatrixTable] = {"Sparse" -> True, "ChenDeltas" -
2432 > False};
2433 TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]]:= (
2434 JJBlockMatrixTable = <||>;
2435 totalIterations = Length[AllowedJ[numE]]^2;
2436 template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
2437 template2 = StringTemplate["`remtime` min remaining"];
2438 template4 = StringTemplate["Time elapsed = `runtime` min"];
2439 numiter = 0;
2440 startTime = Now;
2441 If[$FrontEnd != Null,
2442 (
2443 temp = PrintTemporary[
2444 Dynamic[
2445 Grid[
2446 {
2447 {template1[<|"numiter" -> numiter, "totaliter" ->
2448 totalIterations|>]},
2449 {template2[<|"remtime" -> Round[QuantityMagnitude[
2450 UnitConvert[(Now - startTime)/(Max[1, numiter])*(totalIterations -
2451 numiter), "min"]], 0.1]|>}},
2452 {template4[<|"runtime" -> Round[QuantityMagnitude[
2453 UnitConvert[(Now - startTime), "min"]], 0.1]|>}],
2454 {ProgressIndicator[numiter, {1, totalIterations}]}
2455 }
2456 ]
2457 ]
2458 ];
2459 )
2460 ];
2461 Do[
2462 (
2463 JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp
2464 , CFTable, "Sparse" -> OptionValue["Sparse"], "ChenDeltas" ->
2465 OptionValue["ChenDeltas"]];
2466 numiter += 1;
2467 ),
2468 {Jp, AllowedJ[numE]},
2469 {J, AllowedJ[numE]}
2470 ];
2471 If[$FrontEnd != Null,
2472 NotebookDelete[temp]
2473 ]
2474 ]
2475 ]
2476 ]
2477 ]
2478 ]
2479 ]
2480 ]
2481 ]
2482 ]
2483 ]
2484 ]
2485 ]
2486 ]
2487 ]
2488 ]
2489 ]
2490 ]
2491 ]
2492 ]
2493 ]
2494 ]
2495 ]
2496 ]
2497 ]
2498 ]
2499 ]
2500 ]
2501 ]
2502 ]
2503 ]
2504 ]
2505 ]
2506 ]
2507 ]
2508 ]
2509 ]
2510 ]
2511 ]
2512 ]
2513 ]
2514 ]
2515 ]
2516 ]
2517 ]
2518 ]
2519 ]
2520 ]
2521 ]
2522 ]
2523 ]
2524 ]
2525 ]
2526 ]
2527 ]
2528 ]
2529 ]
2530 ]
2531 ]
2532 ]
2533 ]
2534 ]
2535 ]
2536 ]
2537 ]
2538 ]
2539 ]
2540 ]
2541 ]
2542 ]
2543 ]
2544 ]
2545 ]
2546 ]
2547 ]
2548 ]
2549 ]
2550 ]
2551 ]
2552 ]
2553 ]
2554 ]
2555 ]
2556 ]
2557 ]
2558 ]
2559 ]
2560 ]
2561 ]
2562 ]
2563 ]
2564 ]
2565 ]
2566 ]
2567 ]
2568 ]
2569 ]
2570 ]
2571 ]
2572 ]
2573 ]
2574 ]
2575 ]
2576 ]
2577 ]
2578 ]
2579 ]
2580 ]
2581 ]
2582 ]
2583 ]
2584 ]
2585 ]
2586 ]
2587 ]
2588 ]
2589 ]
2590 ]
2591 ]
2592 ]
2593 ]
2594 ]
2595 ]
2596 ]
2597 ]
2598 ]
2599 ]
2600 ]
2601 ]
2602 ]
2603 ]
2604 ]
2605 ]
2606 ]
2607 ]
2608 ]
2609 ]
2610 ]
2611 ]
2612 ]
2613 ]
2614 ]
2615 ]
2616 ]
2617 ]
2618 ]
2619 ]
2620 ]
2621 ]
2622 ]
2623 ]
2624 ]
2625 ]
2626 ]
2627 ]
2628 ]
2629 ]
2630 ]
2631 ]
2632 ]
2633 ]
2634 ]
2635 ]
2636 ]
2637 ]
2638 ]
2639 ]
2640 ]
2641 ]
2642 ]
2643 ]
2644 ]
2645 ]
2646 ]
2647 ]
2648 ]
2649 ]
2650 ]
2651 ]
2652 ]
2653 ]
2654 ]
2655 ]
2656 ]
2657 ]
2658 ]
2659 ]
2660 ]
2661 ]
2662 ]
2663 ]
2664 ]
2665 ]
2666 ]
2667 ]
2668 ]
2669 ]
2670 ]
2671 ]
2672 ]
2673 ]
2674 ]
2675 ]
2676 ]
2677 ]
2678 ]
2679 ]
2680 ]
2681 ]
2682 ]
2683 ]
2684 ]
2685 ]
2686 ]
2687 ]
2688 ]
2689 ]
2690 ]
2691 ]
2692 ]
2693 ]
2694 ]
2695 ]
2696 ]
2697 ]
2698 ]
2699 ]
2700 ]
2701 ]
2702 ]
2703 ]
2704 ]
2705 ]
2706 ]
2707 ]
2708 ]
2709 ]
2710 ]
2711 ]
2712 ]
2713 ]
2714 ]
2715 ]
2716 ]
2717 ]
2718 ]
2719 ]
2720 ]
2721 ]
2722 ]
2723 ]
2724 ]
2725 ]
2726 ]
2727 ]
2728 ]
2729 ]
2730 ]
2731 ]
2732 ]
2733 ]
2734 ]
2735 ]
2736 ]
2737 ]
2738 ]
2739 ]
2740 ]
2741 ]
2742 ]
2743 ]
2744 ]
2745 ]
2746 ]
2747 ]
2748 ]
2749 ]
2750 ]
2751 ]
2752 ]
2753 ]
2754 ]
2755 ]
2756 ]
2757 ]
2758 ]
2759 ]
2760 ]
2761 ]
2762 ]
2763 ]
2764 ]
2765 ]
2766 ]
2767 ]
2768 ]
2769 ]
2770 ]
2771 ]
2772 ]
2773 ]
2774 ]
2775 ]
2776 ]
2777 ]
2778 ]
2779 ]
2780 ]
2781 ]
2782 ]
2783 ]
2784 ]
2785 ]
2786 ]
2787 ]
2788 ]
2789 ]
2790 ]
2791 ]
2792 ]
2793 ]
2794 ]
2795 ]
2796 ]
2797 ]
2798 ]
2799 ]
2800 ]
2801 ]
2802 ]
2803 ]
2804 ]
2805 ]
2806 ]
2807 ]
2808 ]
2809 ]
2810 ]
2811 ]
2812 ]
2813 ]
2814 ]
2815 ]
2816 ]
2817 ]
2818 ]
2819 ]
2820 ]
2821 ]
2822 ]
2823 ]
2824 ]
2825 ]
2826 ]
2827 ]
2828 ]
2829 ]
2830 ]
2831 ]
2832 ]
2833 ]
2834 ]
2835 ]
2836 ]
2837 ]
2838 ]
2839 ]
2840 ]
2841 ]
2842 ]
2843 ]
2844 ]
2845 ]
2846 ]
2847 ]
2848 ]
2849 ]
2850 ]
2851 ]
2852 ]
2853 ]
2854 ]
2855 ]
2856 ]
2857 ]
2858 ]
2859 ]
2860 ]
2861 ]
2862 ]
2863 ]
2864 ]
2865 ]
2866 ]
2867 ]
2868 ]
2869 ]
2870 ]
2871 ]
2872 ]
2873 ]
2874 ]
2875 ]
2876 ]
2877 ]
2878 ]
2879 ]
2880 ]
2881 ]
2882 ]
2883 ]
2884 ]
2885 ]
2886 ]
2887 ]
2888 ]
2889 ]
2890 ]
2891 ]
2892 ]
2893 ]
2894 ]
2895 ]
2896 ]
2897 ]
2898 ]
2899 ]
2900 ]
2901 ]
2902 ]
2903 ]
2904 ]
2905 ]
2906 ]
2907 ]
2908 ]
2909 ]
2910 ]
2911 ]
2912 ]
2913 ]
2914 ]
2915 ]
2916 ]
2917 ]
2918 ]
2919 ]
2920 ]
2921 ]
2922 ]
2923 ]
2924 ]
2925 ]
2926 ]
2927 ]
2928 ]
2929 ]
2930 ]
2931 ]
2932 ]
2933 ]
2934 ]
2935 ]
2936 ]
2937 ]
2938 ]
2939 ]
2940 ]
2941 ]
2942 ]
2943 ]
2944 ]
2945 ]
2946 ]
2947 ]
2948 ]
2949 ]
2950 ]
2951 ]
2952 ]
2953 ]
2954 ]
2955 ]
2956 ]
2957 ]
2958 ]
2959 ]
2960 ]
2961 ]
2962 ]
2963 ]
2964 ]
2965 ]
2966 ]
2967 ]
2968 ]
2969 ]
2970 ]
2971 ]
2972 ]
2973 ]
2974 ]
2975 ]
2976 ]
2977 ]
2978 ]
2979 ]
2980 ]
2981 ]
2982 ]
2983 ]
2984 ]
2985 ]
2986 ]
2987 ]
2988 ]
2989 ]
2990 ]
2991 ]
2992 ]
2993 ]
2994 ]
2995 ]
2996 ]
2997 ]
2998 ]
2999 ]
2999 ]
3000 ]
3001 ]
3002 ]
3003 ]
3004 ]
3005 ]
3006 ]
3007 ]
3008 ]
3009 ]
3010 ]
3011 ]
3012 ]
3013 ]
3014 ]
3015 ]
3016 ]
3017 ]
3018 ]
3019 ]
3020 ]
3021 ]
3022 ]
3023 ]
3024 ]
3025 ]
3026 ]
3027 ]
3028 ]
3029 ]
3030 ]
3031 ]
3032 ]
3033 ]
3034 ]
3035 ]
3036 ]
3037 ]
3038 ]
3039 ]
3040 ]
3041 ]
3042 ]
3043 ]
3044 ]
3045 ]
3046 ]
3047 ]
3048 ]
3049 ]
3050 ]
3051 ]
3052 ]
3053 ]
3054 ]
3055 ]
3056 ]
3057 ]
3058 ]
3059 ]
3060 ]
3061 ]
3062 ]
3063 ]
3064 ]
3065 ]
3066 ]
3067 ]
3068 ]
3069 ]
3070 ]
3071 ]
3072 ]
3073 ]
3074 ]
3075 ]
3076 ]
3077 ]
3078 ]
3079 ]
3080 ]
3081 ]
3082 ]
3083 ]
3084 ]
3085 ]
3086 ]
3087 ]
3088 ]
3089 ]
3090 ]
3091 ]
3092 ]
3093 ]
3094 ]
3095 ]
3096 ]
3097 ]
3098 ]
3099 ]
3099 ]
3100 ]
3101 ]
3102 ]
3103 ]
3104 ]
3105 ]
3106 ]
3107 ]
3108 ]
3109 ]
3110 ]
3111 ]
3112 ]
3113 ]
3114 ]
3115 ]
3116 ]
3117 ]
3118 ]
3119 ]
3120 ]
3121 ]
3122 ]
3123 ]
3124 ]
3125 ]
3126 ]
3127 ]
3128 ]
3129 ]
3130 ]
3131 ]
3132 ]
3133 ]
3134 ]
3135 ]
3136 ]
3137 ]
3138 ]
3139 ]
3140 ]
3141 ]
3142 ]
3143 ]
3144 ]
3145 ]
3146 ]
3147 ]
3148 ]
3149 ]
3150 ]
3151 ]
3152 ]
3153 ]
3154 ]
3155 ]
3156 ]
3157 ]
3158 ]
3159 ]
3160 ]
3161 ]
3162 ]
3163 ]
3164 ]
3165 ]
3166 ]
3167 ]
3168 ]
3169 ]
3170 ]
3171 ]
3172 ]
3173 ]
3174 ]
3175 ]
3176 ]
3177 ]
3178 ]
3179 ]
3180 ]
3181 ]
3182 ]
3183 ]
3184 ]
3185 ]
3186 ]
3187 ]
3188 ]
3189 ]
3190 ]
3191 ]
3192 ]
3193 ]
3194 ]
3195 ]
3196 ]
3197 ]
3198 ]
3199 ]
3199 ]
3200 ]
3201 ]
3202 ]
3203 ]
3204 ]
3205 ]
3206 ]
3207 ]
3208 ]
3209 ]
3210 ]
3211 ]
3212 ]
3213 ]
3214 ]
3215 ]
3216 ]
3217 ]
3218 ]
3219 ]
3220 ]
3221 ]
3222 ]
3223 ]
3224 ]
3225 ]
3226 ]
3227 ]
3228 ]
3229 ]
3230 ]
3231 ]
3232 ]
3233 ]
3234 ]
3235 ]
3236 ]
3237 ]
3238 ]
3239 ]
32310 ]
32311 ]
32312 ]
32313 ]
32314 ]
32315 ]
32316 ]
32317 ]
32318 ]
32319 ]
32320 ]
32321 ]
32322 ]
32323 ]
32324 ]
32325 ]
32326 ]
32327 ]
32328 ]
32329 ]
32330 ]
32331 ]
32332 ]
32333 ]
32334 ]
32335 ]
32336 ]
32337 ]
32338 ]
32339 ]
323310 ]
323311 ]
323312 ]
323313 ]
323314 ]
323315 ]
323316 ]
323317 ]
323318 ]
323319 ]
323320 ]
323321 ]
323322 ]
323323 ]
323324 ]
323325 ]
323326 ]
323327 ]
323328 ]
323329 ]
323330 ]
323331 ]
323332 ]
323333 ]
323334 ]
323335 ]
323336 ]
323337 ]
323338 ]
323339 ]
3233310 ]
3233311 ]
3233312 ]
3233313 ]
3233314 ]
3233315 ]
3233316 ]
3233317 ]
3233318 ]
3233319 ]
3233320 ]
3233321 ]
3233322 ]
3233323 ]
3233324 ]
3233325 ]
3233326 ]
3233327 ]
3233328 ]
3233329 ]
3233330 ]
3233331 ]
3233332 ]
3233333 ]
3233334 ]
3233335 ]
3233336 ]
3233337 ]
3233338 ]
3233339 ]
32333310 ]
32333311 ]
32333312 ]
32333313 ]
32333314 ]
32333315 ]
32333316 ]
32333317 ]
32333318 ]
32333319 ]
32333320 ]
32333321 ]
32333322 ]
32333323 ]
32333324 ]
32333325 ]
32333326 ]
32333327 ]
32333328 ]
32333329 ]
32333330 ]
32333331 ]
32333332 ]
32333333 ]
32333334 ]
32333335 ]
32333336 ]
32333337 ]
32333338 ]
32333339 ]
323333310 ]
323333311 ]
323333312 ]
323333313 ]
323333314 ]
323333315 ]
323333316 ]
323333317 ]
323333318 ]
323333319 ]
323333320 ]
323333321 ]
323333322 ]
323333323 ]
323333324 ]
323333325 ]
323333326 ]
323333327 ]
323333328 ]
323333329 ]
323333330 ]
323333331 ]
323333332 ]
323333333 ]
323333334 ]
323333335 ]
323333336 ]
323333337 ]
323333338 ]
323333339 ]
3233333310 ]
3233333311 ]
3233333312 ]
3233333313 ]
3233333314 ]
3233333315 ]
3233333316 ]
3233333317 ]
3233333318 ]
3233333319 ]
3233333320 ]
3233333321 ]
3233333322 ]
3233333323 ]
3233333324 ]
3233333325 ]
3233333326 ]
3233333327 ]
3233333328 ]
3233333329 ]
3233333330 ]
3233333331 ]
3233333332 ]
3233333333 ]
3233333334 ]
3233333335 ]
3233333336 ]
3233333337 ]
3233333338 ]
3233333339 ]
32333333310 ]
32333333311 ]
32333333312 ]
32333333313 ]
32333333314 ]
32333333315 ]
32333333316 ]
32333333317 ]
32333333318 ]
32333333319 ]
32333333320 ]
32333333321 ]
32333333322 ]
32333333323 ]
32333333324 ]
32333333325 ]
32333333326 ]
32333333327 ]
32333333328 ]
32333333329 ]
32333333330 ]
32333333331 ]
32333333332 ]
32333333333 ]
32333333334 ]
32333333335 ]
32333333336 ]
32333333337 ]
32333333338 ]
32333333339 ]
323333333310 ]
323333333311 ]
323333333312 ]
323333333313 ]
323333333314 ]
323333333315 ]
323333333316 ]
323333333317 ]
323333333318 ]
323333333319 ]
323333333320 ]
323333333321 ]
323333333322 ]
323333333323 ]
323333333324 ]
323333333325 ]
323333333326 ]
323333333327 ]
323333333328 ]
323333333329 ]
323333333330 ]
323333333331 ]
323333333332 ]
323333333333 ]
323333333334 ]
323333333335 ]
323333333336 ]
323333333337 ]
323333333338 ]
323333333339 ]
3233333333310 ]
3233333333311 ]
3233333333312 ]
3233333333313 ]
3233333333314 ]
3233333333315 ]
3233333333316 ]
3233333333317 ]
3233333333318 ]
3233333333319 ]
3233333333320 ]
3233333333321 ]
3233333333322 ]
3233333333323 ]
3233333333324 ]
3233333333325 ]
3233333333326 ]
3233333333327 ]
3233333333328 ]
3233333333329 ]
3233333333330 ]
3233333333331 ]
3233333333332 ]
3233333333333 ]
3233333333334 ]
3233333333335 ]
3233333333336 ]
3233333333337 ]
3233333333338 ]
3233333333339 ]
32333333333310 ]
32333333333311 ]
32333333333312 ]
32333333333313 ]
32333333333314 ]
32333333333315 ]
32333333333316 ]
32333333333317 ]
32333333333318 ]
32333333333319 ]
32333333333320 ]
32333333333321 ]
32333333333322 ]
32333333333323 ]
32333333333324 ]
32333333333325 ]
32333333333326 ]
32333333333327 ]
32333333333328 ]
32333333333329 ]
32333333333330 ]
32333333333331 ]
32333333333332 ]
32333333333333 ]
32333333333334 ]
32333333333335 ]
32333333333336 ]
32333333333337 ]
32333333333338 ]
32333333333339 ]
323333333333310 ]
323333333333311 ]
323333333333312 ]
323333333333313 ]
323333333333314 ]
323333333333315 ]
323333333333316 ]
323333333333317 ]
323333333333318 ]
323333333333319 ]
323333333333320 ]
323333333333321 ]
323333333333322 ]
323333333333323 ]
323333333333324 ]
323333333333325 ]
323333333333326 ]
323333333333327 ]
323333333333328 ]
323333333333329 ]
323333333333330 ]
323333333333331 ]
323333333333332 ]
323333333333333 ]
323333333333334 ]
323333333333335 ]
323333333333336 ]
323333333333337 ]
323333333333338 ]
323333333333339 ]
3233333333333310 ]
3233333333333311 ]
3233333333333312 ]
3233333333333313 ]
3233333333333314 ]
3233333333333315 ]
3233333333333316 ]
3233333333333317 ]
3233333333333318 ]
3233333333333319 ]
3233333333333320 ]
3233333333333321 ]
3233333333333322 ]
3233333333333323 ]
3233333333333324 ]
3233333333333325 ]
3233333333333326 ]
3233333333333327 ]
3233333333333328 ]
3233333333333329 ]
3233333333333330 ]
3233333333333331 ]
3233333333333332 ]
3233333333333333 ]
3233333333333334 ]
3233333333333335 ]
3233333333333336 ]
3233333333333337 ]
3233333333333338 ]
3233333333333339 ]
32333333333333310 ]
32333333333333311 ]
32333333333333312 ]
32333333333333313 ]
32333333333333314 ]
32333333333333315 ]
32333333333333316 ]
32333333333333317 ]
32333333333333318 ]
32333333333333319 ]
32333333333333320 ]
32333333333333321 ]
32333333333333322 ]
32333333333333323 ]
32333333333333324 ]
32333333333333325 ]
32333333333333326 ]
32333333333333327 ]
32333333333333328 ]
32333333333333329 ]
32333333333333330 ]
32333333333333331 ]
32333333333333332 ]
32333333333333333 ]
32333333333333334 ]
32333333333333335 ]
32333333333333336 ]
32333333333333337 ]
32333333333333338 ]
32333333333333339 ]
323333333333333310 ]
323333333333333311 ]
323333333333333312 ]
323333333333333313 ]
323333333333333314 ]
323333333333333315 ]
323333333333333316 ]
323333333333333317 ]
323333333333333318 ]
323333333333333319 ]
323333333333333320 ]
323333333333333321 ]
323333333333333322 ]
323333333333333323 ]
323333333333333324 ]
323333333333333325 ]
323333333333333326 ]
323333333333333327 ]
323333333333333328 ]
323333333333333329 ]
323333333333333330 ]
323333333333333331 ]
323
```

```

2451 ];
2452 Return[JJBlockMatrixTable];
2453 );
2454
2455 TabulateManyJJBlockMatrixTables::usage =
2456 TabulateManyJJBlockMatrixTables[{n1, n2, ...}] calculates the
2457 tables of matrix elements for the requested f^n_i configurations.
2458 The function does not return the matrices themselves. It instead
2459 returns an association whose keys are numE and whose values are
2460 the filenames where the output of TabulateJJBlockMatrixTables was
2461 saved to. The output consists of an association whose keys are of
2462 the form {n, J, Jp} and whose values are rectangular arrays given
2463 the values of <|LSJMJa|H|L'S'J'MJ'a'|>.";
2464 Options[TabulateManyJJBlockMatrixTables] = {"Overwrite" -> False,
2465 "Sparse" -> True, "ChenDeltas" -> False, "FilenameAppendix" -> "", "Compressed" -> False};
2466 TabulateManyJJBlockMatrixTables[ns_, OptionsPattern[]] := (
2467   overwrite = OptionValue["Overwrite"];
2468   fNames = <||>;
2469   fileApp = OptionValue["FilenameAppendix"];
2470   ExportFun = If[OptionValue["Compressed"], ExportMZip, Export];
2471   Do[
2472     (
2473       CFdataFilename = FileNameJoin[{moduleDir, "data", "CrystalFieldTable_f"} <> ToString[numE] <> ".zip"];
2474       PrintTemporary["Importing CrystalFieldTable from ", CFdataFilename, "..."];
2475       CrystalFieldTable = ImportMZip[CFdataFilename];
2476
2477       PrintTemporary["----- numE = ", numE, " -----#"];
2478       exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix" -> fileApp];
2479       fNames[numE] = exportFname;
2480       If[FileExistsQ[exportFname] && Not[overwrite],
2481         Continue[]
2482       ];
2483       JJBlockMatrixTable = TabulateJJBlockMatrixTable[numE, CrystalFieldTable, "Sparse" -> OptionValue["Sparse"], "ChenDeltas" -> OptionValue["ChenDeltas"]];
2484       If[FileExistsQ[exportFname] && overwrite,
2485         DeleteFile[exportFname]
2486       ];
2487       ExportFun[exportFname, JJBlockMatrixTable];
2488
2489       ClearAll[CrystalFieldTable];
2490     ),
2491     {numE, ns}
2492   ];
2493   Return[fNames];
2494 );
2495
2496 EffectiveHamiltonian::usage = "EffectiveHamiltonian[numE] returns
2497 the Hamiltonian matrix for the f^numE configuration. The matrix is
2498 returned as a SparseArray.
2499 The function admits an optional parameter \"FilenameAppendix\" which can be used to control which variant of the JJBlocks is used to assemble the matrix.
2500 It also admits an optional parameter \"IncludeZeeman\" which can be used to include the Zeeman interaction. The default is False.
2501 The option \"Set t2Switch\" can be used to toggle on or off setting the t2 selector automatically or not, the default is True, which replaces the parameter according to numE.
2502 The option \"ReturnInBlocks\" can be used to return the matrix in block or flattened form. The default is to return it in flattened form.";
2503 Options[EffectiveHamiltonian] = {
2504   "FilenameAppendix" -> "",
2505   "IncludeZeeman" -> False,
2506   "Set t2Switch" -> True,
2507   "ReturnInBlocks" -> False,
2508   "OperatorBasis" -> "Legacy"};
2509 EffectiveHamiltonian[nf_, OptionsPattern[]] := Module[
2510   {numE, ii, jj, howManyJs, Js, blockHam, opBasis},
2511   (
2512     (*#####
2513     ImportFun = ImportMZip;

```

```

2503 opBasis = OptionValue["OperatorBasis"];
2504 If[Not[MemberQ[{"Legacy", "MostlyOrthogonal", "Orthogonal"}, 
2505 opBasis]], 
2506     Echo["Operator basis " <> opBasis <> " not recognized, using 
2507 \\"Legacy\\" basis."];
2508     opBasis = "Legacy";
2509 ];
2510 If[opBasis == "Orthogonal",
2511     Echo["Operator basis \"Orthogonal\" not implemented yet,
2512 aborting ..."];
2513     Return[Null];
2514 ];
2515 (*#####
2516 If[opBasis == "MostlyOrthogonal",
2517 (
2518     blockHam = EffectiveHamiltonian[nf,
2519         "OperatorBasis" -> "Legacy",
2520         "FilenameAppendix" -> OptionValue["FilenameAppendix"],
2521         "IncludeZeeman" -> OptionValue["IncludeZeeman"],
2522         "Set t2Switch" -> OptionValue["Set t2Switch"],
2523         "ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
2524     paramChanger = Which[
2525         nf < 7,
2526             <|
2527                 F0 -> 1/91 (54 E1p+91 E0p+78 γp),
2528                 F2 -> (15/392 *
2529                     (
2530                         140 E1p +
2531                         20020 E2p +
2532                         1540 E3p +
2533                         770 αp -
2534                         70 γp +
2535                         22 Sqrt[2] T2p -
2536                         11 Sqrt[2] nf T2p t2Switch -
2537                         11 Sqrt[2] (14-nf) T2p (1 - t2Switch)
2538                     )
2539             ),
2540                 F4 -> (99/490 *
2541                     (
2542                         70 E1p -
2543                         9100 E2p +
2544                         280 E3p +
2545                         140 αp -
2546                         35 γp +
2547                         4 Sqrt[2] T2p -
2548                         2 Sqrt[2] nf T2p t2Switch -
2549                         2 Sqrt[2] (14-nf) T2p (1-t2Switch)
2550                     )
2551             ),
2552                 F6 -> (5577/7000 *
2553                     (
2554                         20 E1p +
2555                         700 E2p -
2556                         140 E3p -
2557                         70 αp -
2558                         10 γp -
2559                         2 Sqrt[2] T2p +
2560                         Sqrt[2] nf T2p t2Switch +
2561                         Sqrt[2] (14-nf) T2p (1-t2Switch)
2562                     )
2563             ),
2564                 ζ -> ζ,
2565                 α -> (5 αp)/4,
2566                 β -> -6 (5 αp + βp),
2567                 γ -> 5/2 (2 βp + 5 γp),
2568                 T2 -> 0
2569             |>,
2570             nf >= 7,
2571             <|
2572                 F0 -> 1/91 (54 E1p+91 E0p+78 γp),
2573                 F2 -> (15/392 *
2574                     (
2575                         140 E1p +
2576                         20020 E2p +
2577                         1540 E3p +
2578                         770 αp -

```

```

2576      70  $\gamma_p$  +
2577      22  $\text{Sqrt}[2]$   $T_{2p}$  -
2578      11  $\text{Sqrt}[2]$   $nf$   $T_{2p}$ 
2579    )
2580  ),
2581  F4  $\rightarrow$  (99/490 *
2582  (
2583    70  $E_{1p}$  -
2584    9100  $E_{2p}$  +
2585    280  $E_{3p}$  +
2586    140  $\alpha_p$  -
2587    35  $\gamma_p$  +
2588    4  $\text{Sqrt}[2]$   $T_{2p}$  -
2589    2  $\text{Sqrt}[2]$   $nf$   $T_{2p}$ 
2590  )
2591  ),
2592  F6  $\rightarrow$  (5577/7000 *
2593  (
2594    20  $E_{1p}$  +
2595    700  $E_{2p}$  -
2596    140  $E_{3p}$  -
2597    70  $\alpha_p$  -
2598    10  $\gamma_p$  -
2599    2  $\text{Sqrt}[2]$   $T_{2p}$  +
2600     $\text{Sqrt}[2]$   $nf$   $T_{2p}$ 
2601  )
2602  ),
2603   $\zeta \rightarrow \zeta$ ,
2604   $\alpha \rightarrow (5 \alpha_p)/4$ ,
2605   $\beta \rightarrow -6 (5 \alpha_p + \beta_p)$ ,
2606   $\gamma \rightarrow 5/2 (2 \beta_p + 5 \gamma_p)$ ,
2607  T2  $\rightarrow 0$ 
2608  |>
2609 ];
2610 blockHamMO = Which[
2611   OptionValue["ReturnInBlocks"] == False,
2612   ReplaceInSparseArray[blockHam, paramChanger],
2613   OptionValue["ReturnInBlocks"] == True,
2614   Map[ReplaceInSparseArray[#, paramChanger]&, blockHam,
2615 {2}]
2616 ];
2617 Return[blockHamMO];
2618 ]
2619 (*#####
2620 (*hole-particle equivalence enforcement*)
2621 numE = nf;
2622 allVars = {E0, E1, E2, E3,  $\zeta$ , F0, F2, F4, F6, M0, M2, M4, T2,
2623 T2p,
2624   T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
2625    $\alpha$ ,  $\beta$ ,  $\gamma$ , B02, B04, B06, B12, B14, B16,
2626   B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16,
2627 S22,
2628   S24, S26, S34, S36, S44, S46, S56, S66, T11p, T12, T14, T15,
2629 T16,
2630   T17, T18, T19, Bx, By, Bz};
2631 params0 = AssociationThread[allVars, allVars];
2632 If[nf > 7,
2633   (
2634     numE = 14 - nf;
2635     params = HoleElectronConjugation[params0];
2636     If[OptionValue["Set t2Switch"], params[t2Switch] = 0];
2637   ),
2638   params = params0;
2639   If[OptionValue["Set t2Switch"], params[t2Switch] = 1];
2640 ];
2641 (* Load symbolic expressions for LS,J,J' energy sub-matrices.
2642 *)
2643 emFname = JJBlockMatrixFileName[numE, "FilenameAppendix"  $\rightarrow$ 
2644 OptionValue["FilenameAppendix"]];
2645 JJBlockMatrixTable = ImportFun[emFname];
2646 (*Patch together the entire matrix representation using J,J'
2647 blocks.*)
2648 PrintTemporary["Patching JJ blocks ..."];
2649 Js = AllowedJ[numE];

```

```

2645     howManyJs = Length[Js];
2646     blockHam = ConstantArray[0, {howManyJs, howManyJs}];
2647     Do[
2648       blockHam[[jj, ii]] = JJBlockMatrixTable[{numE, Js[[ii]], Js[[jj]]}];,
2649       {ii, 1, howManyJs},
2650       {jj, 1, howManyJs}
2651     ];
2652     (* Once the block form is created flatten it *)
2653     If[Not[OptionValue["ReturnInBlocks"]],
2654       (blockHam = ArrayFlatten[blockHam];
2655        blockHam = ReplaceInSparseArray[blockHam, params];
2656       ),
2657       (blockHam = Map[ReplaceInSparseArray[#, params]&, blockHam
2658 ,{2}]);
2659     ];
2660
2661     If[OptionValue["IncludeZeeman"],
2662       (
2663         PrintTemporary["Including Zeeman terms ..."];
2664         {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
2665         blockHam += - teslaToKayser * (Bx * magx + By * magy + Bz * magz);
2666       )
2667       ];
2668     Return[blockHam];
2669   ]
2670 ];
2671 SimplerEffectiveHamiltonian::usage = "SimplerEffectiveHamiltonian[
2672   numE, simplifier] is a simple addition to EffectiveHamiltonian
2673   that applies a given simplification to the full Hamiltonian.
2674   simplifier is a list of replacement rules.
2675   If the option \"Export\" is set to True, then the function also
2676   exports the resulting sparse array to the ./hams/ folder.
2677   The option \"PrependToFilename\" can be used to prepend a string to
2678   the filename to which the function may export to.
2679   The option \"Return\" can be used to choose whether the function
2680   returns the matrix or not.
2681   The option \"Overwrite\" can be used to overwrite the file if it
2682   already exists, if this options is set to False then this function
2683   simply reloads a file that it assumed to be present already in
2684   the ./hams folder.
2685   The option \"IncludeZeeman\" can be used to toggle the inclusion of
2686   the Zeeman interaction with an external magnetic field.
2687   The option \"OperatorBasis\" can be used to choose the basis in
2688   which the operator is expressed. The default is the \"Legacy\" basis.
2689   Order alternatives being: \"MostlyOrthogonal\" and \"Orthogonal\".
2690   In the \"Legacy\" alternative the operators used are
2691   the same as in Carnall's work. In the \"MostlyOrthogonal\" all
2692   operators are orthogonal except those corresponding to the Mk and
2693   Pk parameters. In the \"Orthogonal\" basis all operators are
2694   orthogonal, with the operators corresponding to the Mk and Pk
2695   parameters replaced by zi operators and accompanying ai
2696   coefficients. The \"Orthogonal\" option has not been implemented
2697   yet.";
2698 Options[SimplerEffectiveHamiltonian] = {
2699   "Export" -> True,
2700   "PrependToFilename" -> "",
2701   "EorF" -> "F",
2702   "Overwrite" -> False,
2703   "Return" -> True,
2704   "Set t2Switch" -> False,
2705   "IncludeZeeman" -> False,
2706   "OperatorBasis" -> "Legacy"
2707 };
2708 SimplerEffectiveHamiltonian[numE_, simplifier_, OptionsPattern[]]
2709 := Module[
2710   {thisHam, fname, fnamemx},
2711   (
2712     If[Not[ValueQ[ElectrostaticTable]],
2713       LoadElectrostatic[]
2714     ];
2715     If[Not[ValueQ[S00andECSOTable]],
2716       LoadS00andECSO[]
2717     ];
2718   );
2719 ];

```

```

2696 ];
2697 If[Not[ValueQ[SpinOrbitTable]],
2698   LoadSpinOrbit[]
2699 ];
2700 If[Not[ValueQ[SpinSpinTable]],
2701   LoadSpinSpin[]
2702 ];
2703 If[Not[ValueQ[ThreeBodyTable]],
2704   LoadThreeBody[]
2705 ];
2706
2707 opBasis = OptionValue["OperatorBasis"];
2708 If[Not[MemberQ[{"Legacy", "MostlyOrthogonal", "Orthogonal"}, opBasis]],
2709   Echo["Operator basis " <> opBasis <> " not recognized, using \
2710   \"Legacy\" basis."];
2711   opBasis = "Legacy";
2712 ];
2713 If[opBasis == "Orthogonal",
2714   Echo["Operator basis \"Orthogonal\" not implemented yet,
2715   aborting ..."];
2716   Return[Null];
2717 ];
2718 fnamePrefix = Which[
2719   opBasis == "Legacy",
2720   "SymbolicMatrix-f",
2721   opBasis == "MostlyOrthogonal",
2722   "SymbolicMatrix-mostly-orthogonal-f",
2723   opBasis == "Orthogonal",
2724   "SymbolicMatrix-orthogonal-f"
2725 ];
2726 fname = FileNameJoin[{moduleDir, "hams",
2727   OptionValue["PrependToFilename"] <> fnamePrefix <> ToString[
2728     numE] <> ".m"}];
2729 fnamemx = FileNameJoin[{moduleDir, "hams",
2730   OptionValue["PrependToFilename"] <> fnamePrefix <> ToString[
2731     numE] <> ".mx"}];
2732 fnamezip = FileNameJoin[{moduleDir, "hams",
2733   OptionValue["PrependToFilename"] <> fnamePrefix <> ToString[
2734     numE] <> ".zip"}];
2735 If[Or[FileExistsQ[fname],
2736   FileExistsQ[fnamemx],
2737   FileExistsQ[fnamezip]]
2738   && Not[OptionValue["Overwrite"]],
2739   (
2740     If[OptionValue["Return"],
2741       (
2742         Which[
2743           FileExistsQ[fnamezip],
2744           (
2745             Echo["File " <> fnamezip <> " already exists, and
2746             option \"Overwrite\" is set to False, loading file ..."];
2747             thisHam = ImportMZip[fnamezip];
2748             Return[thisHam];
2749           ),
2750           FileExistsQ[fnamemx],
2751           (
2752             Echo["File " <> fnamemx <> " already exists, and
2753             option \"Overwrite\" is set to False, loading file ..."];
2754             thisHam = Import[fnamemx];
2755             Return[thisHam];
2756           ),
2757           FileExistsQ[fname],
2758           (
2759             Echo["File " <> fname <> " already exists, and
2760             option \"Overwrite\" is set to False, loading file ..."];
2761             thisHam = Import[fname];
2762             Echo["Exporting to file " <> fnamemx <> " for
2763             quicker loading."];
2764             Export[fnamemx, thisHam];
2765             Return[thisHam];
2766           )
2767         ]
2768       ),
2769     (
2770       (

```

```

2762           Echo["File " <> fname <> " already exists, skipping ..."];
2763       ];
2764       Return[Null];
2765   );
2766   ]
2767 ];
2768
2769 thisHam = EffectiveHamiltonian[numE, "Set t2Switch" ->
2770 OptionValue["Set t2Switch"], "IncludeZeeman" -> OptionValue["IncludeZeeman"], "OperatorBasis" -> opBasis];
2770 If[Length[simplifier] > 0,
2771     thisHam = ReplaceInSparseArray[thisHam, simplifier];
2772 ];
2773 (* This removes zero entries from being included in the sparse array *)
2774 thisHam = SparseArray[thisHam];
2775 If[OptionValue["Export"],
2776 (
2777     If[Not@FileExistsQ[fname],
2778     (
2779         Echo["Exporting to file " <> fname];
2780         Export[fname, thisHam];
2781     )
2782 ];
2783 If[Not@FileExistsQ[fnamemx],
2784 (
2785     Echo["Exporting to file " <> fnamemx];
2786     Export[fnamemx, thisHam];
2787 )
2788 ];
2789 )
2790 ];
2791 If[OptionValue["Return"],
2792     Return[thisHam],
2793     Return[Null]
2794 ];
2795 );
2796 ];
2797
2798 ScalarLSJMFfromLS::usage = "ScalarLSJMFfromLS[numE, LSReducedMatrixElements]. Given the LS-reduced matrix elements LSReducedMatrixElements of a scalar operator, this function returns the corresponding LSJM representation. This is returned as a SparseArray.";
2799 ScalarLSJMFfromLS[numE_, LSReducedMatrixElements_] := Module[
2800 {jjBlocktable, NKSLJMs, NKSLJMp, J, Jp, eMatrix, SLterm,
2801 SpLpterm,
2802 MJ, MJp, subKron, matValue, Js, howManyJs, blockHam, ii, jj},
2803 (
2804 SparseDiagonalArray[diagonalElements_] := SparseArray[
2805     Table[{i, i} -> diagonalElements[[i]],
2806     {i, 1, Length[diagonalElements]}]
2807 ];
2808 SparseZeroArray[width_, height_] := (
2809     SparseArray[
2810         Join[
2811             Table[{1, i} -> 0, {i, 1, width}],
2812             Table[{i, 1} -> 0, {i, 1, height}]
2813         ]
2814     );
2815 jjBlockTable = <||>;
2816 Do[
2817     NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
2818     NKSLJMp = AllowedNKSLJMforJTerms[numE, Jp];
2819     If[J != Jp,
2820         jjBlockTable[{numE, J, Jp}] = SparseZeroArray[Length[NKSLJMs],
2821             Length[NKSLJMp]];
2822         Continue[];
2823     ];
2824     eMatrix = Table[
2825         (* Condition for a scalar matrix op *)
2826         SLterm = NKSLJM[[1]];
2827         SpLpterm = NKSLJMp[[1]];
2828         MJ = NKSLJM[[3]];

```

```

2828     MJp = NKSLJMp[[3]];
2829     subKron = (KroneckerDelta[MJ, MJp]);
2830     matValue = If[subKron == 0,
2831     0,
2832     (
2833       Which[MemberQ[Keys[LSReducedMatrixElements], {numE,
2834       SLterm, SpLpterm}], LSReducedMatrixElements[{numE, SLterm, SpLpterm}],
2835       MemberQ[Keys[LSReducedMatrixElements], {numE, SpLpterm,
2836       SLterm}], LSReducedMatrixElements[{numE, SpLpterm, SLterm}],
2837       True,
2838       0
2839     ]
2840   );
2841   matValue,
2842   {NKSLJMp, NKSLJMp},
2843   {NKSLJM, NKSLJM}
2844 ];
2845 ];
2846 jjBlockTable[{numE, J, Jp}] = SparseArray[eMatrix],
2847 {J, AllowedJ[numE]},
2848 {Jp, AllowedJ[numE]}
2849 ];
2850
2851 Js = AllowedJ[numE];
2852 howManyJs = Length[Js];
2853 blockHam = ConstantArray[0, {howManyJs, howManyJs}];
2854 Do[blockHam[[jj, ii]] = jjBlockTable[{numE, Js[[ii]], Js[[jj]]}];,
2855   {ii, 1, howManyJs},
2856   {jj, 1, howManyJs}];
2857 blockHam = ArrayFlatten[blockHam];
2858 blockHam = SparseArray[blockHam];
2859 Return[blockHam];
2860 )
2861 ];
2862
2863 (* ##### Block assembly ##### *)
2864 (* ##### *)
2865
2866 (* ##### Level Description ##### *)
2867
2868 FreeHam::usage = "FreeHam[JBlocks, numE] given the JJ blocks of
2869   the Hamiltonian for f^n, this function returns a list with all the
2870   scalar-simplified versions of the blocks.";
2871 FreeHam[JBlocks_List, numE_Integer] := Module[
2872   {Js, basisJ, pivot, freeHam, idx, J,
2873   thisJbasis, shrunkBasisPositions, theBlock},
2874   (
2875     Js = AllowedJ[numE];
2876     basisJ = BasisLSJMJ[numE, "AsAssociation" -> True];
2877     pivot = If[OddQ[numE], 1/2, 0];
2878     freeHam = Table[(
2879       J = Js[[idx]];
2880       theBlock = JBlocks[[idx]];
2881       thisJbasis = basisJ[J];
2882       (* find the basis vectors that end with pivot *)
2883       shrunkBasisPositions = Flatten[Position[thisJbasis, {_ ..., pivot}]];
2884       (* take only those rows and columns *)
2885       theBlock[[shrunkBasisPositions, shrunkBasisPositions]]
2886     ),
2887     {idx, 1, Length[Js]}
2888   ];
2889   Return[freeHam];
2890 )
2891 ];
2892
2893 ListRepeater::usage = "ListRepeater[list, reps] repeats each
2894   element of list reps times.";
2895 ListRepeater[list_List, repeats_Integer] := (
2896   Flatten[ConstantArray[#, repeats] & /@ list]
2897 );
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
3999

```

```

2897 ListLever::usage = "ListLever[vecs, multiplicity] takes a list of
2898   vectors and returns all interleaved shifted versions of them.";
2899 ListLever[vecs_, multiplicity_] := Module[
2900 {uppyVecs, uppyVec},
2901 (
2902   uppyVecs = Table[(
2903     uppyVec = PadRight[{#}, multiplicity] & /@ vec;
2904     uppyVec = Permutations /@ uppyVec;
2905     uppyVec = Transpose[uppyVec];
2906     uppyVec = Flatten /@ uppyVec
2907   ),
2908   {vec, vecs}
2909 ];
2910   Return[Flatten[uppyVecs, 1]];
2911 );
2912
2913 EigenLever::usage = "EigenLever[eigenSys, multiplicity] takes a
2914   list eigenSys of the form {eigenvalues, eigenvectors} and returns
2915   the eigenvalues repeated multiplicity times and the eigenvectors
2916   interleaved and shifted accordingly.";
2917 EigenLever[eigenSys_, multiplicity_] := Module[
2918 {eigenVals, eigenVecs,
2919 leveledEigenVecs, leveledEigenVals},
2920 (
2921   {eigenVals, eigenVecs} = eigenSys;
2922   leveledEigenVals = ListRepeater[eigenVals, multiplicity];
2923   leveledEigenVecs = ListLever[eigenVecs, multiplicity];
2924   Return[{Flatten[leveledEigenVals], leveledEigenVecs}]
2925 )
2926 ];
2927
2928
2929 LevelSimplerEffectiveHamiltonian::usage =
2930   LevelSimplerEffectiveHamiltonian[numE] is a variation of
2931   EffectiveHamiltonian that returns the diagonal JJ Hamiltonian
2932   blocks applying a simplifier and with simplifications adequate for
2933   the level description. The keys of the given association
2934   correspond to the different values of J that are possible for f^
2935   numE, the values are sparse array that are meant to be interpreted
2936   in the basis provided by BasisLSJ.
2937 The option \"Simplifier\" is a list of symbols that are set to zero
2938 . At a minimum this has to include the crystal field parameters.
2939 By default this includes everything except the Slater parameters
2940 Fk and the spin orbit coupling  $\zeta$ .
2941 The option \"Export\" controls whether the resulting association is
2942 saved to disk, the default is True and the resulting file is
2943 saved to the ./hams/ folder. A hash is appended to the filename
2944 that corresponds to the simplifier used in the resulting
2945 expression. If the option \"Overwrite\" is set to False then these
2946 files may be used to quickly retrieve a previously computed case.
2947 The file is saved both in .m and .mx format.
2948 The option \"PrependToFilename\" can be used to append a string to
2949 the filename to which the function may export to.
2950 The option \"Return\" can be used to choose whether the function
2951 returns the matrix or not.
2952 The option \"Overwrite\" can be used to overwrite the file if it
2953 already exists.";
2954 Options[LevelSimplerEffectiveHamiltonian] = {
2955   "Export" -> True,
2956   "PrependToFilename" -> "",
2957   "Overwrite" -> False,
2958   "Return" -> True,
2959   "Simplifier" -> Join[
2960     {F0, \[Sigma]SS},
2961     cfSymbols,
2962     TSymbols,
2963     casimirSymbols,
2964     pseudoMagneticSymbols,
2965     marvinSymbols,
2966     DeleteCases[magneticSymbols, \zeta]
2967   ]
2968 };
2969 LevelSimplerEffectiveHamiltonian[numE_Integer, OptionsPattern[]] :=
2970   Module[
2971     {thisHamAssoc, Js, fname,
2972

```

```

2949 fnamemx, hash, simplifier},
2950 (
2951     simplifier = (#->0)&/@Sort[OptionValue["Simplifier"]];
2952     hash = Hash[simplifier];
2953     If[Not[ValueQ[ElectrostaticTable]], LoadElectrostatic[]];
2954     If[Not[ValueQ[SOOandECSOTable]], LoadSOOandECSO[]];
2955     If[Not[ValueQ[SpinOrbitTable]], LoadSpinOrbit[]];
2956     If[Not[ValueQ[SpinSpinTable]], LoadSpinSpin[]];
2957     If[Not[ValueQ[ThreeBodyTable]], LoadThreeBody[]];
2958     fname = FileNameJoin[{moduleDir, "hams", OptionValue[
2959 PrependToFilename"]<>"Level-SymbolicMatrix-f"<>ToString[numE]<>"-
2960 <>ToString[hash]<>".m"}];
2961     fnamemx = FileNameJoin[{moduleDir, "hams", OptionValue[
2962 PrependToFilename"]<>"Level-SymbolicMatrix-f"<>ToString[numE]<>"-
2963 <>ToString[hash]<>".mx"}];
2964     If[Or[FileExistsQ[fname], FileExistsQ[fnamemx]]&&Not[OptionValue
2965 ["Overwrite"]],
2966     (
2967         If[OptionValue["Return"],
2968         (
2969             Which[FileExistsQ[fnamemx],
2970             (
2971                 Echo["File " <> fnamemx <> " already exists, and option \
2972 \"Overwrite\" is set to False, loading file ..."];
2973                 thisHamAssoc=Import[fnamemx];
2974                 Return[thisHamAssoc];
2975             ),
2976             FileExistsQ[fname],
2977             (
2978                 Echo["File " <> fname <> " already exists, and option \
2979 \"Overwrite\" is set to False, loading file ..."];
2980                 thisHamAssoc=Import[fname];
2981                 Echo["Exporting to file " <> fnamemx <> " for quicker \
2982 loading."];
2983                 Export[fnamemx,thisHamAssoc];
2984                 Return[thisHamAssoc];
2985             )
2986         ],
2987         (
2988             Echo["File " <> fname <> " already exists, skipping ..."];
2989             Return[Null];
2990         )
2991     ],
2992     );
2993     Js = AllowedJ[numE];
2994     thisHamAssoc = EffectiveHamiltonian[numE,
2995     "Set t2Switch"->True,
2996     "IncludeZeeman"->False,
2997     "ReturnInBlocks"->True
2998 ];
2999     thisHamAssoc = Diagonal[thisHamAssoc];
3000     thisHamAssoc = Map[SparseArray[ReplaceInSparseArray[#, simplifier]]&,thisHamAssoc,{1}];
3001     thisHamAssoc = FreeHam[thisHamAssoc,numE];
3002     thisHamAssoc = AssociationThread[Js->thisHamAssoc];
3003     If[OptionValue["Export"],
3004     (
3005         Echo["Exporting to file " <> fname <> " and to " <> fnamemx];
3006         Export[fname,thisHamAssoc];
3007         Export[fnamemx,thisHamAssoc];
3008     );
3009     ];
3010     LevelSolver::usage = "LevelSolver[numE, params] puts together (or \
3011 retrieves from disk) the symbolic level Hamiltonian for the f^numE \
3012 configuration and solves it for the given params returning the \
3013 resultant energies and eigenstates.

```

```

3012 If the option \"Return as states\" is set to False, then the
3013   function returns an association whose keys are values for J in f^
3014   numE, and whose values are lists with two elements. The first
3015   element being equal to the ordered basis for the corresponding
3016   subspace, given as a list of lists of the form {LS string, J}. The
3017   second element being another list of two elements, the first
3018   element being equal to the energies and the second being equal to
3019   the corresponding normalized eigenvectors. The energies given have
3020   been subtracted the energy of the ground state.
3021 If the option \"Return as states\" is set to True, then the
3022   function returns a list with three elements. The first element is
3023   the global level basis for the f^numE configuration, given as a
3024   list of lists of the form {LS string, J}. The second element are
3025   the mayor LSJ components in the returned eigenstates. The third
3026   element is a list of lists with three elements, in each list the
3027   first element being equal to the energy, the second being equal to
3028   the value of J, and the third being equal to the corresponding
3029   normalized eigenvector (given as a row). The energies given have
3030   been subtracted the energy of the ground state, and the states
3031   have been sorted in order of increasing energy.
3032 The following options are admitted:
3033 - \"Overwrite Hamiltonian\", if set to True the function will
3034   overwrite the symbolic Hamiltonian. Default is False.
3035 - \"Return as states\", see description above. Default is True.
3036 - \"Simplifier\", this is a list with symbols that are set to
3037   zero for defining the parameters kept in the level description.
3038 ";
3039 Options[LevelSolver] = {
3040   "Overwrite Hamiltonian" -> False,
3041   "Return as states" -> True,
3042   "Simplifier" -> Join[
3043     cfSymbols,
3044     TSymbols,
3045     casimirSymbols,
3046     pseudoMagneticSymbols,
3047     marvinSymbols,
3048     DeleteCases[magneticSymbols, \[Zeta]]
3049   ],
3050   "PrintFun" -> PrintTemporary
3051 };
3052 LevelSolver[numE_Integer, params0_Association, OptionsPattern[]] :=
3053   Module[
3054     {ln, simplifier, simpleHam, basis,
3055      numHam, eigensys, startTime, endTime,
3056      diagonalTime, params=params0, globalBasis,
3057      eigenVectors, eigenEnergies, eigenJs,
3058      states, groundEnergy, allEnergies, PrintFun},
3059     (
3060       ln           = theLanthanides[[numE]];
3061       basis        = BasisLSJ[numE, "AsAssociation" -> True];
3062       simplifier   = OptionValue["Simplifier"];
3063       PrintFun     = OptionValue["PrintFun"];
3064       PrintFun["> LevelSolver for ", ln, " with ", numE, " f-electrons."]
3065     ];
3066     PrintFun["> Loading the symbolic level Hamiltonian ..."];
3067     simpleHam = LevelSimplerEffectiveHamiltonian[numE,
3068       "Simplifier" -> simplifier,
3069       "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
3070     ];
3071     (* Everything that is not given is set to zero *)
3072     PrintFun["> Setting to zero every parameter not given ..."];
3073     params = ParamPad[params, "PrintFun" -> PrintFun];
3074     PrintFun[params];
3075     (* Create the numeric hamiltonian *)
3076     PrintFun["> Replacing parameters in the J-blocks of the
3077     Hamiltonian to produce numeric arrays ..."];
3078     numHam = N /@ Map[ReplaceInSparseArray[#, params]&,
3079     simpleHam];
3080     Clear[simpleHam];
3081     (* Eigensolver *)
3082     PrintFun["> Diagonalizing the numerical Hamiltonian within each
3083     separate J-subspace ..."];
3084     startTime = Now;
3085     eigensys = Eigensystem /@ numHam;
3086     endTime = Now;
3087     diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
3088   }
3089 
```

```

3063     allEnergies = Flatten[First/@Values[eigensys]];
3064     groundEnergy = Min[allEnergies];
3065     eigensys = Map[Chop[{#[[1]]-groundEnergy ,#[[2]]}]&, eigensys];
3066     eigensys = Association@KeyValueMap[#1->{basis[#1],#2} &,
3067     eigensys];
3068     PrintFun[">> Diagonalization took ",diagonalTime," seconds."];
3069     If[OptionValue["Return as states"],
3070      (
3071        PrintFun["> Padding the eigenvectors to correspond to the
3072        level basis ..."];
3073        eigenVectors = SparseArray @ BlockDiagonalMatrix[Values
3074        #[[2, 2]] & /@ eigensys];
3075        globalBasis = Flatten[Values[basis], 1];
3076        eigenEnergies = Flatten[Values[#[[2, 1]] & /@ eigensys]];
3077        eigenJs = Flatten[KeyValueMap[ConstantArray[#1,
3078        Length[#[[2, 2]]]] &, eigensys]];
3079        states = Transpose[{eigenEnergies, eigenJs,
3080        eigenVectors}];
3081        states = SortBy[states, First];
3082        eigenVectors = Last /@ states;
3083        LSJmultiplets = (RemoveTrailingDigits[#[[1]]] <> ToString[
3084        InputForm[#[[2]]]]) & /@ globalBasis;
3085        majorComponentIndices = Ordering[Abs[#][[-1]]] & /@
3086        eigenVectors;
3087        levelLabels = LSJmultiplets[[
3088        majorComponentIndices]];
3089        Return[{globalBasis, levelLabels, states}];
3090      ),
3091      Return[{basis, eigensys}]
3092    ];
3093  );
3094 ]
3095 ];
3096
3097 (* ##### Level Description ##### *)
3098 (* ##### *)
3099 (* ##### *)
3100 (* ##### Optical Operators ##### *)
3101
3102 magOp = <||>;
3103
3104 JJBlockMagDip::usage = "JJBlockMagDip[numE, J, Jp] returns an array
3105   for the LSJM matrix elements of the magnetic dipole operator
3106   between states with given J and Jp. The option \"Sparse\" can be
3107   used to return a sparse matrix. The default is to return a sparse
3108   matrix.
3109 See eqn 15.7 in TASS.
3110 Here it is provided in atomic units in which the Bohr magneton is
3111   1/2.
3112 \[Mu] = -(1/2) (L + gs S)
3113 We are using the Racah convention for the reduced matrix elements
3114   in the Wigner-Eckart theorem. See TASS eqn 11.15.
3115 ";
3116 Options[JJBlockMagDip]={ "Sparse" -> True};
3117 JJBlockMagDip[numE_, braJ_, ketJ_, OptionsPattern[]]:=Module[
3118 {braSLJs, ketSLJs,
3119 braSLJ, ketSLJ,
3120 braSL, ketSL,
3121 braS, braL,
3122 ketS, ketL,
3123 braMJ, ketMJ,
3124 matValue, magMatrix,
3125 summand1, summand2,
3126 threejays},
3127 (
3128   braSLJs = AllowedNKSLJMforJTerms[numE, braJ];
3129   ketSLJs = AllowedNKSLJMforJTerms[numE, ketJ];
3130   magMatrix = Table[
3131     braSL = braSLJ[[1]];
3132     ketSL = ketSLJ[[1]];
3133     {braS, braL} = FindSL[braSL];
3134     {ketS, ketL} = FindSL[ketSL];
3135     braMJ = braSLJ[[3]];
3136     ketMJ = ketSLJ[[3]];

```

```

3124      summand1 = If[Or[braJ != ketJ,
3125                      braSL != ketSL],
3126                      0,
3127                      Sqrt[braJ*(braJ+1)*TPO[braJ]]
3128                  ];
3129      (* looking at the string includes checking L=L', S=S', and \
alpha=\alpha *)
3130      summand2 = If[braSL!= ketSL,
3131                      0,
3132                      (gs-1) *
3133                      Phaser[braS+braL+ketJ+1] *
3134                      Sqrt[TPO[braJ]*TPO[ketJ]] *
3135                      SixJay[{braJ,1,ketJ},{braS,braL,braS}] *
3136                      Sqrt[braS(braS+1)TPO[braS]]
3137                  ];
3138      matValue = summand1 + summand2;
3139      (* We are using the Racah convention for red matrix elements
in Wigner-Eckart *)
3140      threejays = (ThreeJay[{braJ, -braMJ}, {1, #}, {ketJ, ketMJ}]
3141 &) /@ {-1,0,1};
3142      threejays *= Phaser[braJ-braMJ];
3143      matValue = - 1/2 * threejays * matValue;
3144      matValue,
3145      {braSLJ, braSLJs},
3146      {ketSLJ, ketSLJs}
3147 ];
3148      If[OptionValue["Sparse"],
3149          magMatrix = SparseArray[magMatrix]
3150      ];
3151      Return[magMatrix];
3152  ];
3153
3154 Options[TabulateJJBlockMagDipTable]= {"Sparse" -> True};
3155 TabulateJJBlockMagDipTable[numE_, OptionsPattern[]] := (
3156     JJBlockMagDipTable=<||>;
3157     Js=AllowedJ[numE];
3158     Do[
3159     (
3160         JJBlockMagDipTable[{numE, braJ, ketJ}] =
3161             JJBlockMagDip[numE, braJ, ketJ, "Sparse" -> OptionValue["Sparse"]]
3162     ],
3163     {braJ, Js},
3164     {ketJ, Js}
3165   ];
3166   Return[JJBlockMagDipTable];
3167 );
3168
3169 TabulateManyJJBlockMagDipTables::usage =
3170 TabulateManyJJBlockMagDipTables[{n1, n2, ...}] calculates the
3171 tables of matrix elements for the requested f^n_i configurations.
3172 The function does not return the matrices themselves. It instead
3173 returns an association whose keys are numE and whose values are
3174 the filenames where the output of TabulateManyJJBlockMagDipTables
3175 was saved to. The output consists of an association whose keys are
3176 of the form {n, J, Jp} and whose values are rectangular arrays
3177 given the values of <|LSJMJa|H_dip|L'S'J'MJ'a'|>.
3178 Options[TabulateManyJJBlockMagDipTables]= {"FilenameAppendix" -> "", "Overwrite" -> False, "Compressed" -> True};
3179 TabulateManyJJBlockMagDipTables[ns_, OptionsPattern[]] := (
3180     fnames=<||>;
3181     Do[
3182     (
3183         ExportFun=If[OptionValue["Compressed"], ExportMZip, Export];
3184         PrintTemporary["----- numE = ", numE, " -----#"];
3185         appendTo = (OptionValue["FilenameAppendix"] <> "-magDip");
3186         exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix" -> appendTo];
3187         fnames[numE] = exportFname;
3188         If[FileExistsQ[exportFname] && Not[OptionValue["Overwrite"]],
3189             Continue[]
3190         ];
3191         JJBlockMatrixTable = TabulateJJBlockMagDipTable[numE];
3192         If[FileExistsQ[exportFname] && OptionValue["Overwrite"],
3193             DeleteFile[exportFname]
3194         ]
3195     );

```

```

3186 ];
3187 ExportFun[exportFname, JJBlockMatrixTable];
3188 ),
3189 {numE,ns}
3190 ];
3191 Return[fnames];
3192 );
3193
3194 MagDipoleMatrixAssembly::usage = "MagDipoleMatrixAssembly[numE]
3195 returns the matrix representation of the operator - 1/2 (L + gs S)
3196 in the f^numE configuration. The function returns a list with
3197 three elements corresponding to the x,y,z components of this
3198 operator. The option \"FilenameAppendix\" can be used to append a
3199 string to the filename from which the function imports from in
3200 order to patch together the array. For numE beyond 7 the function
3201 returns the same as for the complementary configuration. The
3202 option \"ReturnInBlocks\" can be used to return the matrices in
3203 blocks. The default is to return the matrices in flattened form
3204 and as sparse array.";
3205 Options[MagDipoleMatrixAssembly]={
3206 "FilenameAppendix"->"",
3207 "ReturnInBlocks"->False};
3208 MagDipoleMatrixAssembly[nf_Integer, OptionsPattern[]]:=Module[
3209 {ImportFun, numE, appendTo,
3210 emFname, JJBlockMagDipTable,
3211 Js, howManyJs, blockOp,
3212 rowIdx, colIdx},
3213 (
3214 ImportFun=ImportMZip;
3215 numE=nf;
3216 numH=14-numE;
3217 numE=Min[numE,numH];
3218
3219 appendTo=(OptionValue["FilenameAppendix"]<>"-magDip");
3220 emFname=JJBlockMatrixFileName[numE,"FilenameAppendix"]->
3221 appendTo];
3222 JJBlockMagDipTable=ImportFun[emFname];
3223
3224 Js=AllowedJ[numE];
3225 howManyJs=Length[Js];
3226 blockOp=ConstantArray[0,{howManyJs,howManyJs}];
3227 Do[
3228 blockOp[[rowIdx,colIdx]]=JJBlockMagDipTable[{numE,Js[[rowIdx]],Js[[colIdx]]}],
3229 {rowIdx,1,howManyJs},
3230 {colIdx,1,howManyJs}
3231 ];
3232 If[OptionValue["ReturnInBlocks"],
3233 (
3234 opMinus=Map[#[[1]]&,blockOp,{4}];
3235 opZero=Map[#[[2]]&,blockOp,{4}];
3236 opPlus=Map[#[[3]]&,blockOp,{4}];
3237 opX=(opMinus-opPlus)/Sqrt[2];
3238 opY=I (opPlus+opMinus)/Sqrt[2];
3239 opZ=opZero;
3240 ),
3241 blockOp=ArrayFlatten[blockOp];
3242 opMinus=blockOp[[;,;,1]];
3243 opZero=blockOp[[;,;,2]];
3244 opPlus=blockOp[[;,;,3]];
3245 opX=(opMinus-opPlus)/Sqrt[2];
3246 opY=I (opPlus+opMinus)/Sqrt[2];
3247 opZ=opZero;
3248 ];
3249 Return[{opX,opY,opZ}];
3250 )
3251 ];
3252
3253 MagDipLineStrength::usage = "MagDipLineStrength[theEigensys, numE]
3254 takes the eigensystem of an ion and the number numE of f-electrons
3255 that correspond to it and calculates the line strength array Stot
3256 .
3257 The option \"Units\" can be set to either \"SI\" (so that the units
3258 of the returned array are (A m^2)^2) or to \"Hartree\".
3259 The option \"States\" can be used to limit the states for which the
3260 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.
3245 The returned array should be interpreted in the eigenbasis of the
3246 Hamiltonian. As such the element Stot[[i,i]] corresponds to the
3247 line strength states between states  $|i\rangle$  and  $|j\rangle$ .";
3248 Options[MagDipLineStrength]={ "Reload MagOp" -> False, "Units"->"SI"
3249 , "States" -> All};
3250 MagDipLineStrength[theEigensys_List, numE0_Integer, OptionsPattern[]] := Module[
3251 {numE, allEigenvecs, Sx, Sy, Sz, Stot, factor},
3252 (
3253 numE = Min[14-numE0, numE0];
3254 (*If not loaded then load it, *)
3255 If[Or[
3256 Not[MemberQ[Keys[magOp], numE]], OptionValue["Reload MagOp"]],
3257 (
3258 magOp[numE] = ReplaceInSparseArray[#, {gs->2}]& /@ MagDipoleMatrixAssembly[numE];
3259 )
3260 ];
3261 allEigenvecs = Transpose[Last /@ theEigensys];
3262 Which[OptionValue["States"] === All,
3263 (
3264 {Sx, Sy, Sz} = (ConjugateTranspose[allEigenvecs].#.
3265 allEigenvecs) & /@ magOp[numE];
3266 Stot = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
3267 ),
3268 IntegerQ[OptionValue["States"]],
3269 (
3270 singleState = theEigensys[[OptionValue["States"],2]];
3271 {Sx, Sy, Sz} = (ConjugateTranspose[allEigenvecs].#.
3272 singleState) & /@ magOp[numE];
3273 Stot = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
3274 )
3275 ];
3276 Which[
3277 OptionValue["Units"] == "SI",
3278 Return[4 \[Mu]B^2 * Stot],
3279 OptionValue["Units"] == "Hartree",
3280 Return[Stot],
3281 True,
3282 (
3283 Echo["Invalid option for \"Units\". Options are \"SI\" and
3284 " "Hartree\"."];
3285 Abort[];
3286 )
3287 ];
3288 ]
3289 ];
3290 ];
3291 ];
3292 MagDipoleRates::usage = "MagDipoleRates[eigenSys, numE] calculates
3293 the magnetic dipole transition rate array for the provided
3294 eigensystem. The option \"Units\" can be set to \"SI\" or to \"
3295 Hartree\". If the option \"Natural Radiative Lifetimes\" is set to
3296 true then the reciprocal of the rate is returned instead.
3297 eigenSys is a list of lists with two elements, in each list the
3298 first element is the energy and the second one the corresponding
3299 eigenvector.
3300 Based on table 7.3 of Thorne 1999, using g2=1.
3301 The energy unit assumed in eigenSys is kayser.
3302 The returned array should be interpreted in the eigenbasis of the
3303 Hamiltonian. As such the element AMD[[i,i]] corresponds to the
3304 transition rate (or the radiative lifetime, depending on options)
3305 between eigenstates  $|i\rangle$  and  $|j\rangle$ .
3306 By default this assumes that the refractive index is unity, this
3307 may be changed by setting the option \"RefractiveIndex\" to the
3308 desired value.
3309 The option \"Lifetime\" can be used to return the reciprocal of the
3310 transition rates. The default is to return the transition rates."
3311 ;
3312 Options[MagDipoleRates]={ "Units"->"SI", "Lifetime"->False, "
3313 RefractiveIndex"->1};
3314 MagDipoleRates[eigenSys_List, numE0_Integer, OptionsPattern[]] :=

```

```

Module[
{AMD, Stot, eigenEnergies,
transitionWaveLengthsInMeters, nRefractive},
(
  nRefractive = OptionValue["RefractiveIndex"];
  numE = Min[14-numE0, numE0];
  Stot = MagDipLineStrength[eigenSys, numE, "Units" ->
OptionValue["Units"]];
  eigenEnergies = Chop[First/@eigenSys];
  energyDiffs = Outer[Subtract, eigenEnergies, eigenEnergies];
  energyDiffs = ReplaceDiagonal[energyDiffs, Indeterminate];
(* Energies assumed in kayser.*)
  transitionWaveLengthsInMeters = 0.01/energyDiffs;

  unitFactor = Which[
  OptionValue["Units"]=="Hartree",
  (
    (* The bohrRadius factor in SI needed to convert the
wavelengths which are assumed in m*)
    16 \[Pi]^3 (\[Mu]0Hartree /(3 hPlanckHartree)) * bohrRadius^3
  ),
  OptionValue["Units"]=="SI",
  (
    16 \[Pi]^3 \[Mu]0/(3 hPlanck)
  ),
  True,
  (
    Echo["Invalid option for \"Units\". Options are \"SI\" and \""
Hartree\".."];
    Abort[];
  )
];
  AMD = unitFactor / transitionWaveLengthsInMeters^3 * Stot *
nRefractive^3;
  Which[OptionValue["Lifetime"],
  Return[1/AMD],
  True,
  Return[AMD]
]
]
)
];
GroundMagDipoleOscillatorStrength::usage =
GroundMagDipoleOscillatorStrength[eigenSys, numE] calculates the
magnetic dipole oscillator strengths between the ground state and
the excited states as given by eigenSys.
Based on equation 8 of Carnall 1965, removing the  $2J+1$  factor since
this degeneracy has been removed by the crystal field.
eigenSys is a list of lists with two elements, in each list the
first element is the energy and the second one the corresponding
eigenvector.
The energy unit assumed in eigenSys is Kayser.
The oscillator strengths are dimensionless.
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  $|i\rangle$ .
By default this assumes that the refractive index is unity, this
may be changed by setting the option \RefractiveIndex to the
desired value."];
Options[GroundMagDipoleOscillatorStrength]={"RefractiveIndex"->1};
GroundMagDipoleOscillatorStrength[eigenSys_List, numE_Integer,
OptionsPattern[]] := Module[
{eigenEnergies, SMDGS, GSEnergy, energyDiffs,
transitionWaveLengthsInMeters, unitFactor, nRefractive},
(
  eigenEnergies = First/@eigenSys;
  nRefractive = OptionValue["RefractiveIndex"];
  SMDGS = MagDipLineStrength[eigenSys, numE, "Units"->"SI", "States"->1];
  GSEnergy = eigenSys[[1,1]];
  energyDiffs = eigenEnergies-GSEnergy;
  energyDiffs[[1]] = Indeterminate;
  transitionWaveLengthsInMeters = 0.01/energyDiffs;
  unitFactor = (8\[Pi]^2 me)/(3 hPlanck eCharge^2 cLight);
  fMDGS = unitFactor / transitionWaveLengthsInMeters *
SMDGS * nRefractive;

```

```

3352     Return[fMDGS];
3353   )
3354 ];
3355
3356 (* ##### Optical Operators #### *)
3357 (* ##### Printers and Labels #### *)
3358
3359 (* ##### Printers and Labels #### *)
3360 (* ##### Printers and Labels #### *)
3361
3362 PrintL::usage = "PrintL[L] give the string representation of a
3363   given angular momentum.";
3364 PrintL[L_] := If[StringQ[L], L, StringTake[specAlphabet, {L + 1}]]
3365
3366 FindSL::usage = "FindSL[LS] gives the spin and orbital angular
3367   momentum that corresponds to the provided string LS.";
3368 FindSL[SL_] := (
3369   FindSL[SL] =
3370   If[StringQ[SL],
3371     {
3372       (ToExpression[StringTake[SL, 1]] - 1)/2,
3373       StringPosition[specAlphabet, StringTake[SL, {2}]][[1, 1]] - 1
3374     },
3375     SL
3376   ];
3377 )
3378
3379 PrintSLJ::usage = "Given a list with three elements {S, L, J} this
3380   function returns a symbol where the spin multiplicity is presented
3381   as a superscript, the orbital angular momentum as its
3382   corresponding spectroscopic letter, and J as a subscript. Function
3383   does not check to see if the given J is compatible with the given
3384   S and L.";
3385 PrintSLJ[SLJ_] := (
3386   RowBox[{(
3387     SuperscriptBox[" ", 2 SLJ[[1]] + 1],
3388     SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]
3389   })
3390   ] // DisplayForm
3391 );
3392
3393 PrintSLJM::usage = "Given a list with four elements {S, L, J, MJ}
3394   this function returns a symbol where the spin multiplicity is
3395   presented as a superscript, the orbital angular momentum as its
3396   corresponding spectroscopic letter, and {J, MJ} as a subscript. No
3397   attempt is made to guarantee that the given input is consistent."
3398 ;
3399 PrintSLJM[SLJM_] := (
3400   RowBox[{(
3401     SuperscriptBox[" ", 2 SLJM[[1]] + 1],
3402     SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]
3403   })
3404   ] // DisplayForm
3405 );
3406
3407 (* ##### Printers and Labels #### *)
3408 (* ##### Term management #### *)
3409
3410 AllowedSLTerms::usage = "AllowedSLTerms[numE] returns a list with
3411   the allowed terms in the f^numE configuration, the terms are given
3412   as lists in the format {S, L}. This list may have redundancies
3413   which are compatible with the degeneracies that might correspond
3414   to the given case.";
3415 AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE,
3416   14 - numE]]];
3417
3418 AllowedNKSLTerms::usage = "AllowedNKSLTerms[numE] returns a list
3419   with the allowed terms in the f^numE configuration, the terms are
3420   given as strings in spectroscopic notation. The integers in the
3421   last positions are used to distinguish cases with degeneracy.";
3422 AllowedNKSLTerms[numE_] := Map[First, CFPTerms[Min[numE, 14 - numE
3423   ]]];
3424 AllowedNKSLTerms[0] = {"1S"};

```

```

3407 AllowedNKSLTerms[14] = {"1S"};
3408
3409 MaxJ::usage = "MaxJ[numE] gives the maximum J = S+L that
3410     corresponds to the configuration f^numE.";
3411 MaxJ[numE_] := Max[Map[Total, AllowedSLTerms[Min[numE, 14-numE]]]];
3412
3413 MinJ::usage = "MinJ[numE] gives the minimum J = S+L that
3414     corresponds to the configuration f^numE.";
3415 MinJ[numE_] := Min[Map[Abs[Part[#, 1] - Part[#, 2]] &,
3416     AllowedSLTerms[Min[numE, 14-numE]]];
3417
3418 AllowedSLJTerms::usage = "AllowedSLJTerms[numE] returns a list with
3419     the allowed {S, L, J} terms in the f^n configuration, the terms
3420     are given as lists in the format {S, L, J}. This list may have
3421     repeated elements which account for possible degeneracies of the
3422     related term.";
3423 AllowedSLJTerms[numE_] := Module[
3424     {idx1, allowedSL, allowedSLJ},
3425     (
3426         allowedSL = AllowedSLTerms[numE];
3427         allowedSLJ = {};
3428         For[
3429             idx1 = 1,
3430             idx1 <= Length[allowedSL],
3431             termSL = allowedSL[[idx1]];
3432             termsSLJ =
3433                 Table[
3434                     {termSL[[1]], termSL[[2]], J},
3435                     {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
3436                 ];
3437             allowedSLJ = Join[allowedSLJ, termsSLJ];
3438             idx1++
3439         ];
3440         SortBy[allowedSLJ, Last]
3441     )
3442 ];
3443
3444 AllowedNKSLJTerms::usage = "AllowedNKSLJTerms[numE] returns a list
3445     with the allowed {SL, J} terms in the f^n configuration, the terms
3446     are given as lists in the format {SL, J} where SL is a string in
3447     spectroscopic notation.";
3448 AllowedNKSLJTerms[numE_] := Module[
3449     {allowedSL, allowedNKSL, allowedSLJ, nn},
3450     (
3451         allowedNKSL = AllowedNKSLTerms[numE];
3452         allowedSL = AllowedSLTerms[numE];
3453         allowedSLJ = {};
3454         For[
3455             nn = 1,
3456             nn <= Length[allowedSL],
3457             (
3458                 termSL = allowedSL[[nn]];
3459                 termNKSL = allowedNKSL[[nn]];
3460                 termsSLJ =
3461                     Table[{termNKSL, J},
3462                         {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
3463                     ];
3464                 allowedSLJ = Join[allowedSLJ, termsSLJ];
3465                 nn++
3466             )
3467         ];
3468         SortBy[allowedSLJ, Last]
3469     )
3470 ];
3471
3472 AllowedNKSLforJTerms::usage = "AllowedNKSLforJTerms[numE, J] gives
3473     the terms that correspond to the given total angular momentum J in
3474     the f^n configuration. The result is a list whose elements are
3475     lists of length 2, the first element being the SL term in
3476     spectroscopic notation, and the second element being J.";
3477 AllowedNKSLforJTerms[numE_, J_] := Module[
3478     {allowedSL, allowedNKSL, allowedSLJ,
3479     nn, termSL, termNKSL, termsSLJ},
3480     (
3481         allowedNKSL = AllowedNKSLTerms[numE];
3482         allowedSL = AllowedSLTerms[numE];

```

```

3469     allowedSLJ = {};
3470     For[
3471       nn = 1,
3472       nn <= Length[allowedSL],
3473       (
3474         termSL = allowedSL[[nn]];
3475         termNDSL = allowedNDSL[[nn]];
3476         termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[
3477           termSL],
3478             {{termNDSL, J}},
3479             {}
3480           ];
3481         allowedSLJ = Join[allowedSLJ, termsSLJ];
3482         nn++
3483       )
3484     ];
3485   Return[allowedSLJ]
3486 ]
3487
3488 AllowedSLJMTerms::usage = "AllowedSLJMTerms[numE] returns a list
3489   with all the states that correspond to the configuration f^n. A
3490   list is returned whose elements are lists of the form {S, L, J, MJ}
3491   .";
3492 AllowedSLJMTerms[numE_] := Module[
3493   {allowedSLJ, allowedSLJM,
3494   termSLJ, termsSLJM, nn},
3495   (
3496     allowedSLJ = AllowedSLJTerms[numE];
3497     allowedSLJM = {};
3498     For[
3499       nn = 1,
3500       nn <= Length[allowedSLJ],
3501       nn++,
3502       (
3503         termSLJ = allowedSLJ[[nn]];
3504         termsSLJM =
3505           Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
3506             {M, - termSLJ[[3]], termSLJ[[3]]}]
3507           ];
3508         allowedSLJM = Join[allowedSLJM, termsSLJM];
3509       )
3510     ];
3511   Return[SortBy[allowedSLJM, Last]];
3512 )
3513
3514 AllowedNDSLJMforJMTerms::usage = "AllowedNDSLJMforJMTerms[numE, J,
3515   MJ] returns a list with all the terms that contain states of the f
3516   ^n configuration that have a total angular momentum J, and a
3517   projection along the z-axis MJ. The returned list has elements of
3518   the form {SL (string in spectroscopic notation), J, MJ}.";
3519 AllowedNDSLJMforJMTerms[numE_, J_, MJ_] := Module[
3520   {allowedSL, allowedNDSL,
3521   allowedSLJM, nn},
3522   (
3523     allowedNDSL = AllowedNDSLTerms[numE];
3524     allowedSL = AllowedSLTerms[numE];
3525     allowedSLJM = {};
3526     For[
3527       nn = 1,
3528       nn <= Length[allowedSL],
3529       termSL = allowedSL[[nn]];
3530       termNDSL = allowedNDSL[[nn]];
3531       termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
3532                     <= J
3533                     <= Total[termSL]
3534                     && (Abs[MJ] <= J)
3535                     ),
3536                     {{termNDSL, J, MJ}},
3537                     {}];
3538       allowedSLJM = Join[allowedSLJM, termsSLJ];
3539       nn++
3540     ];
3541   Return[allowedSLJM];
3542 )

```

```

3537 ];
3538
3539 AllowedNKSLJMforJTerms::usage = "AllowedNKSLJMforJTerms[numE_, J]
3540   returns a list with all the states that have a total angular
3541   momentum J. The returned list has elements of the form {SL (string
3542   in spectroscopic notation), J, MJ}.";
3543 AllowedNKSLJMforJTerms[numE_, J_] := Module[
3544   {MJs, labelsAndMomenta, termsWithJ},
3545   (
3546     MJs = AllowedMforJ[J];
3547     (* Pair LS labels and their {S,L} momenta *)
3548     labelsAndMomenta = (#, FindSL[#]) & /@ AllowedNKSLTerms[numE]
3549   ];
3550     (* A given term will contain J if |L-S|<=J<=L+S *)
3551     ContainsJ[{SL_String, {S_, L_}}] := (Abs[S - L] <= J <= (S + L)
3552   );
3553     (* Keep just the terms that satisfy this condition *)
3554     termsWithJ = Select[labelsAndMomenta, ContainsJ];
3555     (* We don't want to keep the {S,L} *)
3556     termsWithJ = #[[1]], J] & /@ termsWithJ;
3557     (* This is just a quick way of including up all the MJ values
3558   *)
3559     Return[Flatten /@ Tuples[{termsWithJ, MJs}]]
3560   )
3561 ];
3562
3563 AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J
3564   , 1].";
3565 AllowedMforJ[J_] := Range[-J, J, 1];
3566
3567 AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta
3568   J that appear in the f^numE configuration.";
3569 AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}];
3570
3571 Seniority::usage = "Seniority[LS] returns the seniority of the
3572   given term.";
3573 Seniority[LS_] := FindNKLSTerm[LS][[1, 2]];
3574
3575 FindNKLSTerm::usage = "Given the string LS FindNKLSTerm[SL] returns
3576   all the terms that are compatible with it. This is only for f^n
3577   configurations. The provided terms might belong to more than one
3578   configuration. The function returns a list with elements of the
3579   form {LS, seniority, W, U}.";
3580 FindNKLSTerm[SL_] := Module[
3581   {NKterms, n},
3582   (
3583     n = 7;
3584     NKterms = {};
3585     Map[
3586       If[! StringFreeQ[First[#], SL],
3587         If[ToExpression[Part[#, 2]] <= n,
3588           NKterms = Join[NKterms, {#}, 1]
3589         ]
3590       ] &,
3591       fnTermLabels
3592     ];
3593     NKterms = DeleteCases[NKterms, {}];
3594     NKterms
3595   )
3596 ];
3597
3598 ParseTermLabels::usage = "ParseTermLabels[] parses the labels for
3599   the terms in the f^n configurations based on the labels for the f6
3600   and f7 configurations. The function returns a list whose elements
3601   are of the form {LS, seniority, W, U}.";
3602 Options[ParseTermLabels] = {"Export" -> True};
3603 ParseTermLabels[OptionsPattern[]}]:= Module[
3604   {labelsTextData, fNtextLabels, nielsonKosterLabels,
3605   seniorities, RacahW, RacahU},
3606   (
3607     labelsTextData = FileNameJoin[{moduleDir, "data", "NielsonKosterLabels_f6_f7.txt"}];
3608     fNtextLabels = Import[labelsTextData];
3609     nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
3610     termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
3611     seniorities = Map[ToExpression[Part[#, {2}]] &,

```

```

3596     nielsonKosterLabels];
3597     racahW =
3598     Map[
3599       StringTake[
3600         Flatten[StringCases[Part[#, {3}], "(" ~~ DigitCharacter ~~ DigitCharacter ~~
3601         DigitCharacter ~~ ")"]], {2, 4}
3602       ] &,
3603     nielsonKosterLabels];
3604     racahU =
3605     Map[
3606       StringTake[
3607         Flatten[StringCases[Part[#, {3}], "(" ~~ DigitCharacter ~~ DigitCharacter ~~ ")"]], {2, 3}
3608       ] &,
3609     nielsonKosterLabels];
3610   fnTermLabels = Join[termLabels, seniorities, racahW, racahU,
3611   2];
3612   fnTermLabels = Sort[fnTermLabels];
3613   If[OptionValue["Export"],
3614     (
3615       broadFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
3616       Export[broadFname, fnTermLabels];
3617     )
3618   ];
3619   Return[fnTermLabels];
3620 )
3621 ];
3622 ];
3623
3624 (* ##### Term management ##### *)
3625 (* ##### ##### ##### ##### ##### *)
3626
3627 LoadLaF3Parameters::usage="LoadLaF4Parameters[ln] gives the models
3628   for the semi-empirical Hamiltonian for the trivalent lanthanide
3629   ion with symbol ln.";
3630 Options[LoadLaF3Parameters] = {
3631   "Vintage" -> "Standard",
3632   "With Uncertainties"->False
3633 };
3634 LoadLaF3Parameters[Ln_String, OptionsPattern[]]:= Module[
3635   {paramData, params},
3636   (
3637     paramData = Which[
3638       OptionValue["Vintage"] == "Carnall",
3639       Import[FileNameJoin[{moduleDir, "data", "carnallParams.m"}]],
3640       OptionValue["Vintage"] == "Standard",
3641       Import[FileNameJoin[{moduleDir, "data", "standardParams.m"}]]
3642     ];
3643     params = If[OptionValue["With Uncertainties"],
3644       paramData[Ln],
3645       If[Head[#] === Around, #[["Value"]], #] & /@ paramData[Ln]
3646     ];
3647     Return[params];
3648   )
3649 ];
3650
3651 HoleElectronConjugation::usage = "HoleElectronConjugation[params]
3652   takes the parameters (as an association) that define a
3653   configuration and converts them so that they may be interpreted as
3654   corresponding to a complementary hole configuration. Some of this
3655   can be simply done by changing the sign of the model parameters.
3656   In the case of the effective three body interaction of T2 the
3657   relationship is more complex and is controlled by the value of the
3658   t2Switch variable.";
3659 HoleElectronConjugation[params_] := Module[
3660   {newparams = params},
3661   (
3662     flipSignsOf = Join[{\zeta}, cfSymbols, TSymbols];
3663     flipped = Table[
3664       (
3665         flipper -> - newparams[flipper]
3666       ),
3667       {flipper, flipSignsOf}
3668     ];

```

```

3660     nonflipped = Table[
3661       (
3662         flipper -> newparams[flipper]
3663       ),
3664       {flipper, Complement[Keys[newparams], flipSignsOf]}
3665     ];
3666     flippedParams = Association[Join[nonflipped, flipped]];
3667     flippedParams = Select[flippedParams, FreeQ[#, Missing]&];
3668     Return[flippedParams];
3669   )
3670 ];
3671
3672 IonSolver::usage = "IonSolver[numE, params, host] puts together (or
3673   retrieves from disk) the symbolic Hamiltonian for the f^numE
3674   configuration and solves it for the given params.
3675   params is an Association with keys equal to parameter symbols and
3676   values their numerical values. The function will replace the
3677   symbols in the symbolic Hamiltonian with their numerical values
3678   and then diagonalize the resulting matrix. Any parameter that is
3679   not defined in the params Association is assumed to be zero.
3680   host is an optional string that may be used to prepend the filename
3681   of the symbolic Hamiltonian that is saved to disk. The default is
3682   \"Ln\".
3683   The function returns the eigensystem as a list of lists where in
3684   each list the first element is the energy and the second element
3685   the corresponding eigenvector.
3686   The ordered basis in which these eigenvectors are to be interpreted
3687   is the one corresponding to BasisLSJMJ[numE].
3688   The function admits the following options:
3689   \\"Include Spin-Spin\\" (bool) : If True then the spin-spin
3690   interaction is included as a contribution to the m_k operators.
3691   The default is True.
3692   \\"Overwrite Hamiltonian\\" (bool) : If True then the function will
3693   overwrite the symbolic Hamiltonian that is saved to disk to
3694   expedite calculations. The default is False. The symbolic
3695   Hamiltonian is saved to disk to the ./hams/ folder preceded by the
3696   string host.
3697   \\"Zeroes\\" (list) : A list with symbols assumed to be zero.
3698   ";
3699 Options[IonSolver] = {
3700   "Include Spin-Spin" -> True,
3701   "Overwrite Hamiltonian" -> False,
3702   "Zeroes" -> {}
3703 };
3704 IonSolver[numE_Integer, params0_Association, host_String:"Ln",
3705 OptionsPattern[]] := Module[
3706   {ln, simplifier, simpleHam, numHam, eigensys,
3707   startTime, endTime, diagonalTime,
3708   params=params0, zeroSymbols},
3709   (
3710     ln = theLanthanides[[numE]];

3711     (* This could be done when replacing values, but this produces
3712     smaller saved arrays. *)
3713     simplifier = (#-> 0) & /@ OptionValue["Zeroes"];
3714     simpleHam = SimplerEffectiveHamiltonian[numE,
3715       simplifier,
3716       "PrependToFilename" -> host,
3717       "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
3718     ];
3719
3720     (* Note that we don't have to flip signs of parameters for fn
3721     beyond f7 since the matrix produced
3722     by SimplerEffectiveHamiltonian has already accounted for this.
3723     *)
3724
3725     (* Everything that is not given is set to zero *)
3726     params = ParamPad[params];
3727     PrintFun[params];

3728     (* Enforce the override to the spin-spin contribution to the
3729     magnetic interactions *)
3730     params[\[Sigma]SS] = If[OptionValue["Include Spin-Spin"], 1,
3731     0];
3732
3733     (* Create the numeric hamiltonian *)

```

```

3713 numHam = ReplaceInSparseArray[simpleHam, params];
3714 Clear[simpleHam];
3715
3716 (* Eigensolver *)
3717 PrintFun["> Diagonalizing the numerical Hamiltonian ..."];
3718 startTime = Now;
3719 eigensys = Eigensystem[numHam];
3720 endTime = Now;
3721 diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
3722 ];
3723 PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
3724 ];
3725 eigensys = Chop[eigensys];
3726 eigensys = Transpose[eigensys];
3727
3728 (* Shift the baseline energy *)
3729 eigensys = ShiftedLevels[eigensys];
3730 (* Sort according to energy *)
3731 eigensys = SortBy[eigensys, First];
3732 Return[eigensys];
3733 )
3734 ];
3735 ShiftedLevels::usage = "ShiftedLevels[eigenSys] takes a list of
3736 levels of the form
3737 {{energy_1, coeff_vector_1}, {energy_2, coeff_vector_2}, ...} and
3738 returns the same input except that now to every energy the minimum
3739 of all of them has been subtracted.";
3740 ShiftedLevels[originalLevels_] := Module[
3741 {groundEnergy, shifted},
3742 (
3743 groundEnergy = Sort[originalLevels][[1, 1]];
3744 shifted = Map[{#[[1]] - groundEnergy, #[[2]]} &,
3745 originalLevels];
3746 Return[shifted];
3747 )
3748 ];
3749
3750 (* ##### Optical Transitions for Levels ##### *)
3751 JuddOfeltUkSquared::usage = "JuddOfeltUkSquared[numE, params]
3752 calculates the matrix elements of the Uk operator in the level
3753 basis. These are calculated according to equation (7) in Carnall
3754 1965.
3755 The function returns a list with the following elements:
3756 - basis : A list with the allowed {SL, J} terms in the f^numE
3757 configuration. Equal to BasisLSJ[numE].
3758 - eigenSys : A list with the eigensystem of the Hamiltonian for
3759 the f^n configuration.
3760 - levelLabels : A list with the labels of the major components of
3761 the level eigenstates.
3762 - LevelUkSquared : An association with the squared matrix
3763 elements of the Uk operators in the level eigenbasis. The keys
3764 being {2, 4, 6} corresponding to the rank of the Uk operator. The
3765 basis in which the matrix elements are given is the one
3766 corresponding to the level eigenstates given in eigenSys and whose
3767 major SLJ components are given in levelLabels. The matrix is
3768 symmetric and given as a SymmetrizedArray.
3769 The function admits the following options:
3770 \\"PrintFun\\" : A function that will be used to print the progress
3771 of the calculations. The default is PrintTemporary.";
3772 Options[JuddOfeltUkSquared] = {"PrintFun" -> PrintTemporary};
3773 JuddOfeltUkSquared[numE_, params_, OptionsPattern[]] := Module[
3774 {eigenChanger, numEH, basis, eigenSys,
3775 Js, Ukm, LevelUkSquared, kRank,
3776 S, L, Sp, Lp, J, Jp, phase,
3777 braTerm, ketTerm, levelLabels,
3778 eigenVecs, majorComponentIndices},
3779 (
3780 If[Not[ValueQ[ReducedUkTable]],
3781 LoadUk[]
3782 ];
3783 numEH = Min[numE, 14 - numE];
3784 PrintFun = OptionValue["PrintFun"];

```

```

3770     PrintFun["> Calculating the levels for the given parameters ...";
3771 ];
3772 {basis, majorComponents, eigenSys} = LevelSolver[numE, params];
3773 (* The change of basis matrix to the eigenstate basis *)
3774 eigenChanger = Transpose[Last /@ eigenSys];
3775 PrintFun["Calculating the matrix elements of Uk in the physical
3776 coupling basis ..."];
3777 LevelUkSquared = <|||>;
3778 Do[(  

3779   Ukmatrix = Table[(  

3780     {S, L} = FindSL[braTerm[[1]]];  

3781     J = braTerm[[2]];  

3782     Jp = ketTerm[[2]];  

3783     {Sp, Lp} = FindSL[ketTerm[[1]]];  

3784     phase = Phaser[S + Lp + J + kRank];  

3785     Simplify @  

3786       phase *  

3787       Sqrt[TPO[J]*TPO[Jp]] *  

3788       SixJay[{J, Jp, kRank}, {Lp, L, S}] *  

3789       ReducedUkTable[{numEH, 3, braTerm[[1]], ketTerm[[1]],  

3790 kRank}];  

3791     )  

3792   ),  

3793   {braTerm, basis},  

3794   {ketTerm, basis}
3795 ];
3796 Ukmatrix = (Transpose[eigenChanger] . Ukmatrix . eigenChanger)^2;
3797 Ukmatrix = Chop@Ukmatrix;
3798 LevelUkSquared[kRank] = SymmetrizedArray[Ukmatrix, Dimensions[
3799 eigenChanger], Symmetric[{1, 2}]];
3800 ),
3801 {kRank, {2, 4, 6}}
3802 ];
3803 LSJmultiplets = (RemoveTrailingDigits[#[[1]]] <> ToString[
3804 InputForm[#[[2]]]]) & /@ basis;
3805 eigenVecs = Last /@ eigenSys;
3806 majorComponentIndices = Ordering[Abs[#][[-1]] & /@ eigenVecs];
3807 levelLabels = LSJmultiplets[[majorComponentIndices]];
3808 Return[{basis, eigenSys, levelLabels, LevelUkSquared}];
3809 )
3810 ];
3811 LevelElecDipoleOscillatorStrength::usage =
3812 "LevelElecDipoleOscillatorStrength[numE, levelParams,
3813 juddOfeltParams] uses Judd-Ofelt theory to estimate the forced
3814 electric dipole oscillator strengths ions whose level description
3815 is determined by levelParams.
3816 The third parameter juddOfeltParams is an association with keys
3817 equal to the three Judd-Ofelt intensity parameters {\[CapitalOmega]2,
3818 \[CapitalOmega]4, \[CapitalOmega]6} and corresponding values
3819 in cm^2.
3820 The local field correction implemented here corresponds to the one
3821 given by the virtual cavity model of Lorentz.
3822 The function returns a list with the following elements:
3823 - basis : A list with the allowed {SL, J} terms in the f^n configuration. Equal to BasisLSJ[numE].
3824 - eigenSys : A list with the eigensystem of the Hamiltonian for
3825 the f^n configuration in the level description.
3826 - levelLabels : A list with the labels of the major components of
3827 the calculated levels.
3828 - oStrengthArray : A square array whose elements represent the
3829 oscillator strengths between levels such that the element
3830 oStrengthArray[[i,j]] is the oscillator strength between the
3831 levels |Subscript[\[Psi], i]> and |Subscript[\[Psi], j]>. In this
3832 array, the elements below the diagonal represent emission
3833 oscillator strengths, and elements above the diagonal represent
3834 absorption oscillator strengths.
3835 The function admits the following three options:
3836 \"PrintFun\" : A function that will be used to print the progress
3837 of the calculations. The default is PrintTemporary.
3838 \"RefractiveIndex\" : The refractive index of the medium where
3839 the transitions are taking place. This may be a number or a
3840 function. If a number then the oscillator strengths are calculated
3841 for assuming a wavelength-independent refractive index. If a
3842 function then the refractive indices are calculated accordingly to
3843 the wavelength of each transition (the function must admit a

```

```

3818     single argument equal to the wavelength in nm). The default is 1.
3819     \\"LocalFieldCorrection\" : The local field correction to be used.
3820     The default is \\"VirtualCavity\". The options are: \
3821     \\"VirtualCavity\" and \\"EmptyCavity\".
3822 The equation implemented here is the one given in eqn. 29 from the
3823 review article of Hehlen (2013). See that same article for a
3824 discussion on the local field correction.
3825 ";
3826 Options[LevelElecDipoleOscillatorStrength]={
3827   "PrintFun"      -> PrintTemporary,
3828   "RefractiveIndex" -> 1,
3829   "LocalFieldCorrection" -> "VirtualCavity"
3830 };
3831 LevelElecDipoleOscillatorStrength[numE_, levelParams_Association,
3832 juddOfeltParams_Association, OptionsPattern[]] := Module[
3833 {PrintFun, basis, eigenSys, levelLabels,
3834 LevelUkSquared, eigenEnergies, energyDiffs,
3835 oStrengthArray, nRef, \[Chi], nRefs,
3836 \[Chi]OverN, groundLevel, const,
3837 transitionFrequencies, wavelengthsInNM,
3838 fieldCorrectionType},
3839 (
3840   PrintFun = OptionValue["PrintFun"];
3841   nRef = OptionValue["RefractiveIndex"];
3842   PrintFun["Calculating the Uk^2 matrix elements for the given
3843 parameters ..."];
3844   {basis, eigenSys, levelLabels, LevelUkSquared} =
3845   JuddOfeltUkSquared[numE, levelParams, "PrintFun" -> PrintFun];
3846   eigenEnergies = First/@eigenSys;
3847   (* converted to cm^-2 *)
3848   const = (8\[Pi]^2)/3 me/hPlanck * 10^(-4);
3849   energyDiffs = Transpose@Outer[Subtract, eigenEnergies,
3850 eigenEnergies];
3851   (* since energies are assumed in Kayser, speed of light needs
3852 to be in cm/s, so that the frequencies are in 1/s *)
3853   transitionFrequencies = energyDiffs*cLight*100;
3854   (* grab the J for each level *)
3855   levelJs = #[[2]] & /@ eigenSys;
3856   oStrengthArray = (
3857     juddOfeltParams[\[CapitalOmega]2]*LevelUkSquared[2]+
3858     juddOfeltParams[\[CapitalOmega]4]*LevelUkSquared[4]+
3859     juddOfeltParams[\[CapitalOmega]6]*LevelUkSquared[6]
3860   );
3861   oStrengthArray = Abs@(const * transitionFrequencies *
3862 oStrengthArray);
3863   (* it is necessary to divide each oscillator strength by the
3864 degeneracy of the initial level *)
3865   oStrengthArray = MapIndexed[1/(2 levelJs[[#2[[1]]]]+1) #1 &,
3866 oStrengthArray,{2}];
3867   (* including the effects of the refractive index *)
3868   fieldCorrectionType = OptionValue["LocalFieldCorrection"];
3869   Which[
3870     nRef === 1,
3871     True,
3872     NumberQ[nRef],
3873     (
3874       \[Chi] = Which[
3875         fieldCorrectionType == "VirtualCavity",
3876         (
3877           ( (nRef^2 + 2) / 3 )^2
3878         ),
3879         fieldCorrectionType == "EmptyCavity",
3880         (
3881           ( 3 * nRef^2 / ( 2 * nRef^2 + 1 ) )^2
3882         )
3883       ];
3884       \[Chi]OverN = \[Chi] / nRef;
3885       oStrengthArray = \[Chi]OverN * oStrengthArray;
3886       (* the refractive index participates differently in
3887 absorption and in emission *)
3888       aFunction = If[#2[[1]] > #2[[2]], #1 * nRef^2, #1] &;
3889       oStrengthArray = MapIndexed[aFunction, oStrengthArray,
3890 {2}];
3891       ),
3892       True,
3893       (

```

```

3879     wavelengthsInNM = Abs[1 / energyDiffs] * 10^7;
3880     nRefs          = Map[nRef, wavelengthsInNM];
3881     Echo["Calculating the oscillator strengths for the given
3882       refractive index ..."];
3883     \[Chi] = Which[
3884       fieldCorrectionType == "VirtualCavity",
3885       (
3886         (nRefs^2 + 2) / 3)^2
3887       ),
3888       fieldCorrectionType == "EmptyCavity",
3889       (
3890         (3 * nRefs^2 / (2*nRefs^2 + 1))^2
3891       )
3892     ];
3893     \[Chi]OverN      = \[Chi] / nRefs;
3894     oStrengthArray  = \[Chi]OverN * oStrengthArray
3895   )
3896   Return[{basis, eigenSys, levelLabels, oStrengthArray}];
3897 )
3898 ];
3899
3900 LevelJJBlockMagDipole::usage = "LevelJJBlockMagDipole[numE, J, Jp]
3901   returns an array of the LSJ reduced matrix elements of the
3902   magnetic dipole operator between states with given J and Jp. The
3903   option \"Sparse\" can be used to return a sparse matrix. The
3904   default is to return a sparse matrix.";
3905 Options[LevelJJBlockMagDipole] = {"Sparse" -> True};
3906 LevelJJBlockMagDipole[numE_, braJ_, ketJ_, OptionsPattern[]] :=
3907   Module[
3908   {
3909     braSLJs, ketSLJs,
3910     braSLJ, ketSLJ,
3911     braSL, ketSL,
3912     braS, braL,
3913     ketS, ketL,
3914     matValue, magMatrix,
3915     summand1, summand2
3916   },
3917   (
3918     braSLJs      = AllowedNKSLforJTerms[numE, braJ];
3919     ketSLJs      = AllowedNKSLforJTerms[numE, ketJ];
3920     magMatrix    = Table[
3921       (
3922         braSL        = braSLJ[[1]];
3923         ketSL        = ketSLJ[[1]];
3924         {braS, braL} = FindSL[braSL];
3925         {ketS, ketL} = FindSL[ketSL];
3926         summand1     = If[Or[braJ != ketJ, braSL != ketSL],
3927           0,
3928           Sqrt[braJ*(braJ+1)*TPO[braJ]
3929           ]
3930         ];
3931         (*looking at the string includes checking L=L', S=S', and \alpha
3932 =\alpha'*)
3933         summand2     = If[braSL != ketSL,
3934           0,
3935           (gs-1)*
3936             Phaser[braS+braL+ketJ+1]*
3937               Sqrt[TPO[braJ]*TPO[ketJ]]*
3938                 SixJay[{braJ, 1, ketJ}, {braS, braL, braS}]*
3939                   Sqrt[braS(braS+1)TPO[braS]]
3940                 ];
3941         matValue = summand1 + summand2;
3942         matValue = -1/2 * matValue;
3943         matValue
3944       ),
3945       {braSLJ, braSLJs},
3946       {ketSLJ, ketSLJs}
3947     ];
3948     If[OptionValue["Sparse"],
3949       magMatrix = SparseArray[magMatrix]];
3950     Return[magMatrix];
3951   )
3952 ];

```

```

3948 LevelMagDipoleMatrixAssembly::usage = "LevelMagDipoleMatrixAssembly
3949 [numE] puts together an array with the reduced matrix elements of
3950 the magnetic dipole operator in the level basis for the f^numE
3951 configuration. The function admits the two following options:
3952 \\"Flattened\\": If True then the returned matrix is flattened. The
3953 default is True.
3954 \\"gs\\": The electronic gyromagnetic ratio. The default is 2.";
3955 Options[LevelMagDipoleMatrixAssembly] = {
3956 "Flattened" -> True,
3957 gs -> 2
3958 };
3959 LevelMagDipoleMatrixAssembly[numE_, OptionsPattern[]] := Module[
3960 {Js, magDip, braJ, ketJ},
3961 (
3962 Js = AllowedJ[numE];
3963 magDip = Table[
3964 ReplaceInSparseArray[LevelJJBlockMagDipole[numE, braJ, ketJ],
3965 {gs -> OptionValue[gs]}, {
3966 {braJ, Js},
3967 {ketJ, Js}
3968 ];
3969 If[OptionValue["Flattened"],
3970 magDip = ArrayFlatten[magDip];
3971 ];
3972 Return[magDip];
3973 )
3974 ];
3975
3976 LevelMagDipoleLineStrength::usage = "LevelMagDipoleLineStrength[
3977 eigenSys, numE] calculates the magnetic dipole line strengths for
3978 an ion whose level description is determined by levelParams. The
3979 function returns a square array whose elements represent the
3980 magnetic dipole line strengths between the levels given in
3981 eigenSys such that the element magDipoleLineStrength[[i,j]] is the
3982 line strength between the levels |Subscript[\[Psi], i]> and |Subscript[\[Psi], j]>. Eigensys must be such that it consists of a
3983 lists of lists where in each list the last element corresponds to
3984 the eigenvector of a level (given as a row) in the standard basis
3985 for levels of the f^numE configuration.
3986 The function admits the following options:
3987 \\"Units\\": The units in which the line strengths are given. The
3988 default is \\"SI\\". The options are \\"SI\\" and \\"Hartree\\". If \\"SI\"
3989 \\" then the unit of the line strength is (A m^2)^2 = (J/T)^2. If \\"Hartree\"
3990 \\" then the line strength is given in units of 2 \[Mu]B.";
3991 Options[LevelMagDipoleLineStrength] = {
3992 "Units" -> "SI"
3993 };
3994 LevelMagDipoleLineStrength[theEigensys_List, numE0_Integer,
3995 OptionsPattern[]] := Module[
3996 {numE, levelMagOp, allEigenvecs, magDipoleLineStrength, units},
3997 (
3998 numE = Min[14 - numE0, numE0];
3999 levelMagOp = LevelMagDipoleMatrixAssembly[numE];
4000 allEigenvecs = Transpose[Last/@theEigensys];
4001 units = OptionValue["Units"];
4002 magDipoleLineStrength = Transpose[allEigenvecs].levelMagOp.allEigenvecs;
4003 magDipoleLineStrength = Abs[magDipoleLineStrength]^2;
4004 Which[
4005 units == "SI",
4006 Return[4 \[Mu]B^2 * magDipoleLineStrength],
4007 units == "Hartree",
4008 Return[magDipoleLineStrength]
4009 ];
4010 ]
4011 ];
4012 ];
4013
4014 LevelMagDipoleOscillatorStrength::usage =
4015 LevelMagDipoleOscillatorStrength[eigenSys, numE] calculates the
4016 magnetic dipole oscillator strengths for an ion whose levels are
4017 described by eigenSys in configuration f^numE. The refractive
4018 index of the medium is relevant, but here it is assumed to be 1,
4019 this can be changed through the option \\"RefractiveIndex\\".
4020 eigenSys must consist of a lists of lists with three elements: the
4021 first element being the energy of the level, the second element
4022 being the J of the level, and the third element being the

```

```

eigenvector of the level.
The function returns a list with the following elements:
- basis : A list with the allowed {SL, J} terms in the f^numE
configuration. Equal to BasisLSJ[numE].
- eigenSys : A list with the eigensystem of the Hamiltonian for
the f^n configuration in the level description.
- magDipole0strength : A square array whose elements represent
the magnetic dipole oscillator strengths between the levels given
in eigenSys such that the element magDipole0strength[[i,j]] is the
oscillator strength between the levels |Subscript[\[Psi], i]> and
|Subscript[\[Psi], j]>. In this array the elements below the
diagonal represent emission oscillator strengths, and elements
above the diagonal represent absorption oscillator strengths. The
emission oscillator strengths are negative. The oscillator
strength is a dimensionless quantity.
The function admits the following option:
  \\"RefractiveIndex\" : The refractive index of the medium where
the transitions are taking place. This may be a number or a
function. If a number then the oscillator strengths are calculated
assuming a wavelength-independent refractive index as given. If a
function then the refractive indices are calculated accordingly
to the vaccum wavelength of each transition (the function must
admit a single argument equal to the wavelength in nm). The
default is 1.
For reference see equation (27.8) in Rudzikas (2007). The
expression for the line strength is the simplest when using atomic
units, (27.8) is missing a factor of  $\alpha^2$ .";
Options[LevelMagDipoleOscillatorStrength]={
  "RefractiveIndex" -> 1
};
LevelMagDipoleOscillatorStrength[eigenSys_, numE_, OptionsPattern $[ ]$ ] := Module[
{eigenEnergies, eigenVecs, levelJs,
energyDiffs, magDipole0strength, nRef,
wavelengthsInNM, nRefs, degenDivisor},
(
  basis = BasisLSJ[numE];
  eigenEnergies = First/@eigenSys;
  nRef = OptionValue["RefractiveIndex"];
  eigenVecs = Last/@eigenSys;
  levelJs = #[[2]]&/@eigenSys;
  energyDiffs = -Outer[Subtract,eigenEnergies,eigenEnergies];
  energyDiffs *= kayserToHartree;
  magDipole0strength = LevelMagDipoleLineStrength[eigenSys, numE,
"Units" -> "Hartree"];
  magDipole0strength = 2/3 *  $\alpha$ Fine^2 * energyDiffs *
magDipole0strength;
  degenDivisor = #1 / ( 2 * levelJs[[#2[[1]]]] + 1 ) &;
  magDipole0strength = MapIndexed[degenDivisor,
magDipole0strength, {2}];
  Which[nRef==1,
    True,
    NumberQ[nRef],
    (
      magDipole0strength = nRef * magDipole0strength;
    ),
    True,
    (
      wavelengthsInNM = Abs[kayserToHartree / energyDiffs] *
10^7;
      nRefs = Map[nRef, wavelengthsInNM];
      magDipole0strength = nRefs * magDipole0strength;
    )
  ];
  Return[{basis, eigenSys, magDipole0strength}];
)
];
LevelMagDipoleSpontaneousDecayRates::usage =
LevelMagDipoleSpontaneousDecayRates[eigenSys, numE] calculates the
spontaneous emission rates for the magnetic dipole transitions
between the levels given in eigenSys. The function returns a
square array whose elements represent the spontaneous emission
rates between the levels given in eigenSys such that the element
[[i,j]] of the returned array is the rate of spontaneous emission
from the level |Subscript[\[Psi], i]> to the level |Subscript[\[Psi], j]>.

```

```

        Psi], j]>. In this array the elements below the diagonal represent
        emission rates, and elements above the diagonal are identically
        zero.

4040 The function admits two optional arguments:
4041 + \"Units\" : The units in which the rates are given. The default
4042 is \"SI\". The options are \"SI\" and \"Hartree\". If \"SI\" then
4043 the rates are given in s^-1. If \"Hartree\" then the rates are
4044 given in the atomic unit of frequency.
4045 + \"RefractiveIndex\" : The refractive index of the medium where
4046 the transitions are taking place. This may be a number or a
4047 function. If a number then the rates are calculated assuming a
4048 wavelength-independent refractive index as given. If a function
4049 then the refractive indices are calculated accordingly to the
4050 vacuum wavelength of each transition (the function must admit a
4051 single argument equal to the wavelength in nm). The default is 1."
4052 ;
4053 Options[LevelMagDipoleSpontaneousDecayRates] = {
4054 "Units" -> "SI",
4055 "RefractiveIndex" -> 1};
4056 LevelMagDipoleSpontaneousDecayRates[eigenSys_List, numE_Integer,
4057 OptionsPattern[]] := Module[
4058 {levMDlineStrength, eigenEnergies, energyDiffs,
4059 levelJs, spontaneousRatesInHartree, spontaneousRatesInSI,
4060 degenDivisor, units, nRef, nRefs, wavelengthsInNM},
4061 (
4062     nRef                  = OptionValue["RefractiveIndex"];
4063     units                 = OptionValue["Units"];
4064     levMDlineStrength     =
4065 LowerTriangularize@LevelMagDipoleLineStrength[eigenSys, numE, "Units"
4066 "->" "Hartree"];
4067     levMDlineStrength    = SparseArray[levMDlineStrength];
4068     eigenEnergies         = First /@ eigenSys;
4069     energyDiffs          = Outer[Subtract, eigenEnergies,
4070 eigenEnergies];
4071     energyDiffs          = kayserToHartree * energyDiffs;
4072     energyDiffs          = SparseArray[LowerTriangularize[energyDiffs
4073 ]];
4074     levelJs               = #[[2]] & /@ eigenSys;
4075     spontaneousRatesInHartree = 4/3 αFine^5 * energyDiffs^3 *
4076 levMDlineStrength;
4077     degenDivisor          = #1 / (2*levelJs[[#2[[1]]]] + 1) &;
4078     spontaneousRatesInHartree = MapIndexed[degenDivisor,
4079 spontaneousRatesInHartree, {2}];
4080     Which[nRef === 1,
4081     True,
4082     NumberQ[nRef],
4083     (
4084         spontaneousRatesInHartree = nRef^3 *
4085 spontaneousRatesInHartree;
4086     ),
4087     True,
4088     (
4089         wavelengthsInNM      = Abs[kayserToHartree / energyDiffs] *
4090 10^7;
4091         nRefs                = Map[nRef, wavelengthsInNM];
4092         spontaneousRatesInHartree = nRefs^3 *
4093 spontaneousRatesInHartree;
4094     )
4095 ];
4096     If[units == "SI",
4097     (
4098         spontaneousRatesInSI = 1/hartreeTime *
4099 spontaneousRatesInHartree;
4100         Return[SparseArray@spontaneousRatesInSI];
4101     ),
4102     Return[SparseArray@spontaneousRatesInHartree];
4103 ];
4104 );
4105 ];
4106 (* ##### Optical Transitions for Levels ##### *)
4107 (* ##### ###### ##### ###### ##### ###### ##### *)
4108 (* ##### ###### ##### ###### ##### ###### ##### *)
4109 (* ##### ###### ##### ###### ##### ###### ##### *)
4110 (* ##### ###### ##### ###### ##### ###### ##### *)

```

```

4092 PrettySaundersSL::usage = "PrettySaundersSL[SL] produces a human-
4093   readable symbol for the spectroscopic term SL. SL can be either a
4094   string (in RS notation for the term) or a list of two numbers {S,
4095   L}. The option \"Representation\" can be used to specify whether
4096   the output is given as a symbol or as a ket. The default is \"Ket\".
4097 ";
4098 Options[PrettySaundersSL] = {"Representation" -> "Ket"};
4099 PrettySaundersSL[SL_, OptionsPattern[]] := (
4100   If[StringQ[SL],
4101   (
4102     {S,L} = FindSL[SL];
4103     L = StringTake[SL,{2,-1}];
4104   ),
4105   {S,L}=SL
4106 ];
4107 pretty = RowBox[{(
4108   AdjustmentBox[Style[2*S+1,Smaller], BoxBaselineShift->-1,
4109   BoxMargins->0],
4110   AdjustmentBox[PrintL[L]]
4111   )
4112 };
4113 pretty = DisplayForm[pretty];
4114 pretty = Which[
4115   OptionValue["Representation"]=="Ket",
4116   Ket[pretty],
4117   OptionValue["Representation"]=="Symbol",
4118   pretty
4119 ];
4120 Return[pretty];
4121 );
4122
4123 PrettySaundersSLJmJ::usage = "PrettySaundersSLJmJ[{SL, J, mJ}]
4124   produces a formatted symbol for the given basis vector {SL, J, mJ}.
4125 ";
4126 Options[PrettySaundersSLJmJ] = {"Representation" -> "Ket"};
4127 PrettySaundersSLJmJ[{SL_, J_, mJ_}, OptionsPattern[]] := (If[
4128   StringQ[SL],
4129   ({S, L} = FindSL[SL];
4130   L = StringTake[SL, {2, -1}];
4131   ),
4132   {S, L} = SL;
4133   pretty = RowBox[{(AdjustmentBox[Style[2*S + 1, Smaller],
4134   BoxBaselineShift -> -1, BoxMargins -> 0],
4135   AdjustmentBox[PrintL[L], BoxMargins -> -0.2],
4136   AdjustmentBox[
4137     Style[Row[{InputForm[J], ", ", mJ}], Small],
4138     BoxBaselineShift -> 1,
4139     BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]}];
4140   pretty = DisplayForm[pretty];
4141   If[OptionValue["Representation"] == "Ket",
4142     pretty = Ket[pretty]
4143   ];
4144   Return[pretty];
4145 );
4146
4147 PrettySaundersSLJ::usage = "PrettySaundersSLJ[{SL, J}] produces a
4148   formatted symbol for the given level {SL, J}. SL can be either a
4149   list of two numbers representing S and L or a string representing
4150   the spin multiplicity in spectroscopic notation. J is the total
4151   angular momentum to which S and L have coupled to. The option \"
4152   Representation\" can be used to specify whether the output is
4153   given as a symbol or as a ket. The default is \"Ket\".";
4154 Options[PrettySaundersSLJ] = {"Representation"->"Ket"};
4155 PrettySaundersSLJ[{SL_,J_},OptionsPattern[]]:= (
4156   If[StringQ[SL],
4157   (
4158     {S,L}=FindSL[SL];
4159     L=StringTake[SL,{2,-1}];
4160   ),
4161   {S,L}=SL
4162 ];
4163 pretty = RowBox[{(
4164   AdjustmentBox[Style[2*S+1,Smaller],BoxBaselineShift->-1,
4165   BoxMargins->0],
4166   AdjustmentBox[PrintL[L],BoxMargins->-0.2],
4167   AdjustmentBox[Style[InputForm[J],Small,FontTracking->"Narrow"]
4168   ]
4169   )
4170 };

```

```

4153   ], BoxBaselineShift -> 1, BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]
4154   }
4155 ];
4156 pretty = DisplayForm[pretty];
4157 pretty = Which[
4158   OptionValue["Representation"] == "Ket",
4159   Ket[pretty],
4160   OptionValue["Representation"] == "Symbol",
4161   pretty
4162 ];
4163 Return[pretty];
4164 );
4165
4166 BasisVecInRusselSaunders::usage = "BasisVecInRusselSaunders[
4167 basisVec] takes a basis vector in the format {LSstring, Jval,
4168 mJval} and returns a formatted symbol for the corresponding Russel-
4169 -Saunders term.";
4170 BasisVecInRusselSaunders[basisVec_] := (
4171   {LSstring, Jval, mJval} = basisVec;
4172   Ket[PrettySaundersSLJmJ[basisVec]]
4173 );
4174
4175 LSJMTemplate =
4176 StringTemplate[
4177   "#!\\(*TemplateBox[{\\nRowBox[{\\\"LS\\\", \\", \", \\nRowBox[{\\\"J\\",
4178   "\\\"=\\\", \\\"J\\\"}], \\", \", \\nRowBox[{\\\"mJ\\\", \\\"=\\\", \\\"mJ\\\"}]}], \\n\\
4179   \\\"Ket\\\"]\\)"];
4180
4181 BasisVecInLSJM::usage = "BasisVecInLSJM[basisVec] takes a basis
4182 vector in the format {{LSstring, Jval}, mJval}, nucSpin} and
4183 returns a formatted symbol for the corresponding LSJM term in the
4184 form |LS, J=..., mJ=...>.";
4185 BasisVecInLSJM[basisVec_] := (
4186   {LSstring, Jval, mJval} = basisVec;
4187   LSJMTemplate[<|
4188     "LS" -> LSstring,
4189     "J" -> ToString[Jval, InputForm],
4190     "mJ" -> ToString[mJval, InputForm]|>]
4191 );
4192
4193 ParseStates::usage = "ParseStates[eigenSys, basis] takes a list of
4194 eigenstates in terms of their coefficients in the given basis and
4195 returns a list of the same states in terms of their energy, LSJM
4196 symbol, J, mJ, S, L, LSJ symbol, and LS symbol. eigenSys is a list
4197 of lists with two elements, in each list the first element is the
4198 energy and the second one the corresponding eigenvector. The LS
4199 symbol returned corresponds to the term with the largest
4200 coefficient in the given basis.";
4201 ParseStates[states_, basis_, OptionsPattern[]] := Module[
4202   {parsedStates},
4203   (
4204     parsedStates = Table[((
4205       {energy, eigenVec} = state;
4206       maxTermIndex = Ordering[Abs[eigenVec]][[-1]];
4207       {LSstring, Jval, mJval} = basis[[maxTermIndex]];
4208       LSJsymbol = Subscript[LSstring, {Jval, mJval}];
4209       LSJMsymbol = LSstring <> ToString[Jval,
4210         InputForm];
4211       {S, L} = FindSL[LSstring];
4212       {energy, LSstring, Jval, mJval, S, L, LSJsymbol, LSJMsymbol}
4213     ),
4214     {state, states}
4215   ];
4216   Return[parsedStates];
4217 )
4218 ];
4219
4220 ParseStatesByNumBasisVecs::usage = "ParseStatesByNumBasisVecs[
4221 eigenSys, basis, numBasisVecs, roundTo] takes a list of
4222 eigenstates (given in eigenSys) in terms of their coefficients in
4223 the given basis and returns a list of the same states in terms of
4224 their energy and the coefficients at most numBasisVecs basis
4225 vectors. By default roundTo is 0.01 and this is the value used to
4226 round the amplitude coefficients. eigenSys is a list of lists with
4227 two elements, in each list the first element is the energy and
4228 the second one the corresponding eigenvector.

```

```

4206 The option \"Coefficients\" can be used to specify whether the
4207   coefficients are given as \"Amplitudes\" or \"Probabilities\". The
4208   default is \"Amplitudes\".
4209 ";
4210 Options[ParseStatesByNumBasisVecs] = {
4211   "Coefficients" -> "Amplitudes",
4212   "Representation" -> "Ket",
4213   "ReturnAs" -> "Dot"
4214 };
4215 ParseStatesByNumBasisVecs[eigensys_List, basis_List,
4216   numBasisVecs_Integer, roundTo_Real : 0.01, OptionsPattern[]] :=
4217 Module[
4218   {parsedStates, energy, eigenVec,
4219   probs, amplitudes, ordering,
4220   returnAs,
4221   chosenIndices, majorComponents,
4222   majorAmplitudes, majorRep},
4223 (
4224   returnAs = OptionValue["ReturnAs"];
4225   parsedStates = Table[(  

4226     {energy, eigenVec} = state;
4227     energy = Chop[energy];
4228     probs = Round[Abs[eigenVec^2], roundTo];
4229     amplitudes = Round[eigenVec, roundTo];
4230     ordering = Ordering[probs];
4231     chosenIndices = ordering[[-numBasisVecs ;;]];
4232     majorComponents = basis[[chosenIndices]];
4233     majorThings = If[OptionValue["Coefficients"] == "  

4234     Probabilities",
4235       (
4236         probs[[chosenIndices]]
4237       ),
4238       (
4239         amplitudes[[chosenIndices]]
4240       )
4241     ];
4242     majorComponents = PrettySaundersSLJmJ[#, "Representation"  

4243     -> OptionValue["Representation"]] & /@ majorComponents;
4244     nonZ = (# != 0.) & /@ majorThings;
4245     majorThings = Pick[majorThings, nonZ];
4246     majorComponents = Pick[majorComponents, nonZ];
4247     If[OptionValue["Coefficients"] == "Probabilities",
4248       (
4249         majorThings = majorThings * 100 * "%";
4250       )
4251     ];
4252     majorRep = Which[
4253       returnAs == "Dot",
4254         majorThings . majorComponents,
4255       returnAs == "List",
4256         Transpose[{Reverse@majorThings,
4257           Reverse@majorComponents}]
4258         ];
4259       {energy, majorRep}
4260     ),
4261     {state, eigensys}];
4262     Return[parsedStates]
4263   )
4264 ];
4265 FindThresholdPosition::usage = "FindThresholdPosition[list,  

4266   threshold] returns the position of the first element in list that  

4267   is greater than or equal to threshold. If no such element exists,  

4268   it returns the length of list. The elements of the given list must  

4269   be in ascending order.";
4270 FindThresholdPosition[list_, threshold_] := Module[
4271   {position},
4272   (
4273     position = Position[list, _?(# >= threshold &), 1, 1];
4274     thrPos = If[Length[position] > 0,
4275       position[[1, 1]],
4276       Length[list]];
4277     If[thrPos == 0,
4278       Return[1],
4279       Return[thrPos]
4280     ]

```

```

4271 ];
4272
4273 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.";
4274 ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ :
  0.01, maxParts_: 20] := Module[
4275 {parsedByProb, numStates, state, energy,
4276 eigenVec, amplitudes, probs, ordering,
4277 orderedProbs, truncationIndex, accProb,
4278 thresholdIndex, chosenIndices, majorComponents,
4279 majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep
}, ,
4280 (
4281   numStates = Length[eigensys];
4282   parsedByProb = Table[(
4283     state = eigensys[[idx]];
4284     {energy, eigenVec} = state;
4285     (*Round them up*)
4286     amplitudes = Round[eigenVec, roundTo];
4287     probs = Round[Abs[eigenVec^2], roundTo];
4288     ordering = Reverse[Ordering[probs]];
4289     (*Order the probabilities from high to low*)
4290     orderedProbs = probs[[ordering]];
4291     (*To speed up Accumulate, assume that only as much as
maxParts will be needed*)
4292     truncationIndex = Min[maxParts, Length[orderedProbs]];
4293     orderedProbs = orderedProbs[[;;truncationIndex]];
4294     (*Accumulate the probabilities*)
4295     accProb = Accumulate[orderedProbs];
4296     (*Find the index of the first element in accProb that is
greater than probSum*)
4297     thresholdIndex = Min[Length[accProb],
FindThresholdPosition[accProb, probSum]];
4298     (*Grab all the indicees up till that one*)
4299     chosenIndices = ordering[[;; thresholdIndex]];
4300     (*Select the corresponding elements from the basis*)
4301     majorComponents = basis[[chosenIndices]];
4302     (*Select the corresponding amplitudes*)
4303     majorAmplitudes = amplitudes[[chosenIndices]];
4304     (*Take their absolute value*)
4305     absMajorAmplitudes = Abs[majorAmplitudes];
4306     (*Make sure that there are no effectively zero contributions
*)
4307     notnullAmplitudes = Flatten[Position[absMajorAmplitudes, x_/
; x != 0]];
4308     (* majorComponents = PrettySaundersSLJmJ
[{{1}, {2}, {3}}] & /@ majorComponents; *)
4309     majorComponents = PrettySaundersSLJmJ /@ majorComponents;
4310     majorAmplitudes = majorAmplitudes[[notnullAmplitudes]];
4311     majorComponents = majorComponents[[notnullAmplitudes]];
4312     (*Multiply and add to build the final Ket*)
4313     majorRep = majorAmplitudes . majorComponents;
4314     {energy, majorRep}
4315     ), {idx, numStates}];
4316   Return[parsedByProb];
4317 )
4318 ];
4319
4320 (* ##### Eigensystem analysis ##### *)
4321 (* ##### *)
4322
4323 (* ##### Misc ##### *)
4324 (* ##### *)
4325
4326 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.";
4327 SymbToNum[expr_, replacementAssociation_] := (
4328   includedKeys = Keys[replacementAssociation];
4329   (*If a key is not defined, make its value zero.*)
4330   fullAssociation = Table[(
```

```

4331 If [MemberQ[includedKeys, key],
4332   ToExpression[key] -> replacementAssociation[key],
4333   ToExpression[key] -> 0
4334 ]
4335 ),
4336 {key, paramSymbols}];
4337 Return[expr/.fullAssociation];
4338 );
4339
4340 SimpleConjugate::usage = "SimpleConjugate[expr] takes an expression
4341 and applies a simplified version of the conjugate in that all it
4342 does is that it replaces the imaginary unit  $I$  with  $-I$ . It assumes
4343 that every other symbol is real so that it remains the same under
4344 complex conjugation. Among other expressions it is valid for any
4345 rational or polynomial expression with complex coefficients and
4346 real variables.";
4347 SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;
4348
4349 ExportMZip::usage = "ExportMZip[\"dest.[zip,m]\", expr] saves a
4350 compressed version of expr to the given destination.";
4351 ExportMZip[filename_, expr_] := Module[
4352   {baseName, exportName, mImportName, zipImportName},
4353   (
4354     baseName = FileBaseName[filename];
4355     exportName = StringReplace[filename, ".m" -> ".zip"];
4356     mImportName = StringReplace[exportName, ".zip" -> ".m"];
4357     If[FileExistsQ[mImportName],
4358       (
4359         PrintTemporary[mImportName <> " exists already, deleting"];
4360         DeleteFile[mImportName];
4361         Pause[2];
4362       )
4363     ];
4364     Export[exportName, (baseName <> ".m") -> expr];
4365   )
4366 ];
4367
4368 ImportMZip::usage = "ImportMZip[filename] imports a .m file inside
4369 a .zip file with corresponding filename. If the Option \"Leave
4370 Uncompressed\" is set to True (the default) then this function
4371 also leaves an uncompressed version of the object in the same
4372 folder of filename";
4373 Options[ImportMZip] = {"Leave Uncompressed" -> True};
4374 ImportMZip[filename_String, OptionsPattern[]} := Module[
4375   {baseName, importKey, zipImportName, mImportName, mxImportName,
4376    imported},
4377   (
4378     baseName = FileBaseName[filename];
4379     (*Function allows for the filename to be .m or .zip*)
4380     importKey = baseName <> ".m";
4381     zipImportName = StringReplace[filename, ".m" -> ".zip"];
4382     mImportName = StringReplace[zipImportName, ".zip" -> ".m"];
4383     mxImportName = StringReplace[zipImportName, ".zip" -> ".mx"];
4384     Which[
4385       FileExistsQ[mxImportName],
4386       (
4387         PrintTemporary[".mx version exists already, importing that
4388 instead ..."];
4389         Return[Import[mxImportName]];
4390       ),
4391       FileExistsQ[mImportName],
4392       (
4393         PrintTemporary[".m version exists already, importing that
4394 instead ..."];
4395         imported = Import[mImportName];
4396         If[OptionValue["Leave Uncompressed"],
4397           (
4398             Export[mxImportName, imported];
4399           )
4400         ];
4401         Return[Import[mImportName]];
4402       ),
4403       ];
4404       imported = Import[zipImportName, importKey];
4405       If[OptionValue["Leave Uncompressed"],
4406         (

```

```

4393     Export[mImportName, imported];
4394     Export[mxImportName, imported];
4395   )
4396 ]
4397   Return[imported];
4398 )
4399 ;
4400
4401 ReplaceInSparseArray::usage = "ReplaceInSparseArray[sparseArray,
4402   rules] takes a sparse array that may contain symbolic quantities
4403   and returns a sparse array in which the given rules have been used
4404   on every element.";
4405 ReplaceInSparseArray[sparseA_SparseArray, rules_] := (
4406   SparseArray[Automatic,
4407     Dimensions[sparseA],
4408     sparseA["Background"] /. rules,
4409     {
4410       1,
4411       {sparseA["RowPointers"], sparseA["ColumnIndices"]},
4412       sparseA["NonzeroValues"] /. rules
4413     }
4414   ]
4415 );
4416
4417
4418 MapToSparseArray::usage = "MapToSparseArray[sparseArray, function]
4419   takes a sparse array and returns a sparse array after the function
4420   has been applied to it.";
4421 MapToSparseArray[sparseA_SparseArray, func_] := Module[
4422   {nonZ, backg, mapped},
4423   (
4424     nonZ = func /@ sparseA["NonzeroValues"];
4425     backg = func[sparseA["Background"]];
4426     mapped = SparseArray[Automatic,
4427       Dimensions[sparseA],
4428       backg,
4429       {
4430         1,
4431         {sparseA["RowPointers"], sparseA["ColumnIndices"]},
4432         nonZ
4433       }
4434     ];
4435     Return[mapped];
4436   )
4437 ];
4438
4439 ParseTeXLikeSymbol::usage = "ParseTeXLikeSymbol[string] parses a
4440   string for a symbol given in LaTeX notation and returns a
4441   corresponding mathematica symbol. The string may have expressions
4442   for several symbols, they need to be separated by single spaces.
4443   In addition the _ and ^ symbols used in LaTeX notation need to
4444   have arguments that are enclosed in parenthesis, for example \"x_2
4445   \" is invalid, instead \"x_{2}\\" should have been given.";
4446 Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
4447 ParseTeXLikeSymbol[bigString_, OptionsPattern[]} := Module[
4448   {form, mainSymbol, symbols},
4449   (
4450     form = OptionValue["Form"];
4451     (* parse greek *)
4452     symbols = Table[(
4453       str = StringReplace[string, {"\\alpha" -> "\[Alpha]",
4454         "\\beta" -> "\[Beta]",
4455         "\\gamma" -> "\[Gamma]",
4456         "\\psi" -> "\[Psi]"}];
4457       symbol = Which[
4458         StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"]],
4459       ],
4460       (
4461         (*yes sub no sup*)
4462         mainSymbol = StringSplit[str, "_"][[1]];
4463         mainSymbol = ToExpression[mainSymbol];
4464
4465         subPart =
4466           StringCases[str,
4467             RegularExpression@"\\{(.*)}\\}" -> "$1"][[1]];
4468         Subscript[mainSymbol, subPart]
4469       ),
4470     ],
4471   )
4472 ];

```

```

4457      Not[StringContainsQ[str, "_"]] && StringContainsQ[str, "^"]
4458    ],
4459    (
4460      (*no sub yes sup*)
4461      mainSymbol = StringSplit[str, "^"][[1]];
4462      mainSymbol = ToExpression[mainSymbol];
4463
4464      supPart =
4465        StringCases[str,
4466          RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
4467      Superscript[mainSymbol, supPart]
4468    ),
4469    StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
4470    (
4471      (*yes sub yes sup*)
4472      mainSymbol = StringSplit[str, "_" ][[1]];
4473      mainSymbol = ToExpression[mainSymbol];
4474      {subPart, supPart} =
4475        StringCases[str, RegularExpression@"\\{(.*?)\\}" -> "
4476        $1"];
4477        Subsuperscript[mainSymbol, subPart, supPart]
4478    ),
4479    True,
4480    (
4481      (*no sup or sub*)
4482      str
4483    );
4484    symbol
4485    ),
4486    {string, StringSplit[bigString, " "]}
4487  ];
4488  Which[
4489    form == "Row",
4490    Return[Row[symbols]],
4491    form == "List",
4492    Return[symbols]
4493  ]
4494];
4495
4496 FromArrayToTable::usage = "FromArrayToTable[array, labels, energies]
4497 takes a square array of values and returns a table with the
4498 labels of the rows and columns, the energies of the initial and
4499 final levels, the level energies, the vacuum wavelength of the
4500 transition, and the value of the array. The array must be square
4501 and the labels and energies must be compatible with the order
4502 implied by the array. The array must be a square array of values.
4503 The function returns a list of lists with the following elements:
4504
4505 - Initial level index
4506 - Final level index
4507 - Initial level label
4508 - Final level label
4509 - Initial level energy
4510 - Final level energy
4511 - Vacuum wavelength
4512 - Value of the array element.
4513 - The reciprocal of the value of the array element.
4514 Elements in which the array is zero are not included in the return
4515 of this function.";
4516 FromArrayToTable[array_, labels_, energies_] := Module[
4517   {tableFun, atl},
4518   (
4519     tableFun = {
4520       #2[[1]],
4521       #2[[2]],
4522       labels[[#2[[1]]]],
4523       labels[[#2[[2]]]],
4524       energies[[#2[[1]]]],
4525       energies[[#2[[2]]]],
4526       If[#2[[1]]==#2[[2]], "--", 10^7/(energies[[#2[[1]]]]-energies
4527 [[#2[[2]]]])],
4528       #1
4529     }&);
4530     atl = Select[Flatten[MapIndexed[tableFun, array
4531     ,{2}],1],#[[{-1}]]==0.&];

```

```

4521      atl = Append[#, 1/#[[[-1]]]] & /@ atl;
4522      Return[atl]
4523    )
4524  ];
4525 (* ##### Misc #####
4526 (* ##### Some Plotting Routines #####
4527
4528 (* ##### Some Plotting Routines #####
4529 (* ##### Some Plotting Routines #####
4530
4531 EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes
4532   states and produces a visualization of its energy spectrum.
4533 The resultant visualization can be navigated by clicking and
4534   dragging to zoom in on a region, or by clicking and dragging
4535   horizontally while pressing Ctrl. Double-click to reset the view."
4536 ;
4537 Options[EnergyLevelDiagram] = {
4538   "Title" -> "",
4539   "ImageSize" -> 1000,
4540   "AspectRatio" -> 1/8,
4541   "Background" -> "Automatic",
4542   "Epilog" -> {},
4543   "Explorer" -> True,
4544   "Energy Unit" -> "cm^-1"
4545 };
4546 EnergyLevelDiagram[states_, OptionsPattern[]] := Module[
4547   {energies, epi, explora},
4548   (
4549     energies = If[Length[Dimensions[states]] == 1,
4550       states,
4551       First /@ states
4552     ];
4553     epi = OptionValue["Epilog"];
4554     explora = If[OptionValue["Explorer"],
4555       ExploreGraphics,
4556       Identity
4557     ];
4558     frameLabel = "E (" <> OptionValue["Energy Unit"] <> ")";
4559     plotTips = Which[
4560       OptionValue["Energy Unit"] == "cm^-1",
4561       Tooltip[{#, 0}, {#, 1}], {Quantity[#/8065.54429, "eV"],
4562         Quantity[#, 1/"Centimeters"]}] & /@ energies,
4563       OptionValue["Energy Unit"] == "eV",
4564       Tooltip[{#, 0}, {#, 1}], {Quantity[# * 8065.54429, 1/
4565         Centimeters], Quantity[#, "eV"]}] & /@ energies,
4566       True,
4567       Tooltip[{#, 0}, {#, 1}], Quantity[#, OptionValue["Energy
4568       Unit"]]] & /@ energies
4569     ];
4570     explora @ ListPlot[plotTips,
4571       Joined -> True,
4572       PlotStyle -> Black,
4573       AspectRatio -> OptionValue["AspectRatio"],
4574       ImageSize -> OptionValue["ImageSize"],
4575       Frame -> True,
4576       PlotRange -> {All, {0, 1}},
4577       FrameTicks -> {{None, None}, {Automatic, Automatic}},
4578       FrameStyle -> Directive[15, Dashed, Thin],
4579       PlotLabel -> Style[OptionValue["Title"], 15, Bold],
4580       Background -> OptionValue["Background"],
4581       FrameLabel -> {frameLabel},
4582       Epilog -> epi
4583     ]
4584   )
4585 ];
4586
4587 ExploreGraphics::usage = "Pass a Graphics object to explore it.
4588   Zoom by clicking and dragging a rectangle. Pan by clicking and
4589   dragging while pressing Ctrl. Click twice to reset view.
4590 Based on ZeitPolizei @ https://mathematica.stackexchange.com/questions/7142/how-to-manipulate-2d-plots.
4591 The option \"OptAxesRedraw\" can be used to specify whether the
4592   axes should be redrawn. The default is False.";
4593 Options[ExploreGraphics] = {OptAxesRedraw -> False};
4594 ExploreGraphics[graph_Graphics, opts : OptionsPattern[]] := With[
4595   {
4596     gr = First[graph],

```

```

4586 opt = DeleteCases[Options[graph],
4587   PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _, ],
4588 plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
4589 ar = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
4590 ao = AbsoluteOptions[AxesOrigin],
4591 rectangle = {Dashing[Small],
4592   Line[{#1,
4593     {First[#2], Last[#1]},
4594     #2,
4595     {First[#1], Last[#2]},
4596     #1}]} &,
4597 optAxesRedraw = OptionValue[OptAxesRedraw]
4598 },
4599 DynamicModule[
4600   {dragging=False, first, second, rx1, rx2, ry1, ry2,
4601   range = plr},
4602   {{rx1, rx2}, {ry1, ry2}} = plr;
4603 Panel@
4604 EventHandler[
4605   Dynamic@Graphics[
4606     If[dragging, {gr, rectangle[first, second]}, gr],
4607     PlotRange -> Dynamic@range,
4608     AspectRatio -> ar,
4609     AxesOrigin -> If[optAxesRedraw,
4610       Dynamic@Mean[range\[Transpose]], ao],
4611     Sequence @@ opt],
4612     {"MouseDown", 1} :> (
4613       first = MousePosition["Graphics"]
4614     ),
4615     {"MouseDragged", 1} :> (
4616       dragging = True;
4617       second = MousePosition["Graphics"]
4618     ),
4619     "MouseClicked" :> (
4620       If[CurrentValue@"MouseClicked"==2,
4621         range = plr];
4622     ),
4623     {"MouseUp", 1} :> If[dragging,
4624       dragging = False;

4625       range = {{rx1, rx2}, {ry1, ry2}} =
4626         Transpose@{first, second};
4627       range[[2]] = {0, 1}],
4628     {"MouseDown", 2} :> (
4629       first = {sx1, sy1} = MousePosition["Graphics"]
4630     ),
4631     {"MouseDragged", 2} :> (
4632       second = {sx2, sy2} = MousePosition["Graphics"];
4633       rx1 = rx1 - (sx2 - sx1);
4634       rx2 = rx2 - (sx2 - sx1);
4635       ry1 = ry1 - (sy2 - sy1);
4636       ry2 = ry2 - (sy2 - sy1);
4637       range = {{rx1, rx2}, {ry1, ry2}};
4638       range[[2]] = {0, 1};
4639     )}]];
4640
4641 LabeledGrid::usage = "LabeledGrid[data, rowHeaders, columnHeaders]
4642 provides a grid of given data interpreted as a matrix of values
4643 whose rows are labeled by rowHeaders and whose columns are labeled
4644 by columnHeaders. When hovering with the mouse over the grid
4645 elements, the row and column labels are displayed with the given
4646 separator between them.";
4647 Options[LabeledGrid]={
4648   ItemSize->Automatic,
4649   Alignment->Center,
4650   Frame->All,
4651   "Separator"->",",
4652   "Pivot"->""
4653 };
4654 LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]]:=Module[
4655   {gridList=data, rowHeads=rowHeaders, colHeads=columnHeaders},
4656   (
4657     separator=OptionValue["Separator"];
4658     pivot=OptionValue["Pivot"];
4659     gridList=Table[

```

```

4656     Tooltip[
4657       data[[rowIdx, colIdx]],
4658       DisplayForm[
4659         RowBox[{rowHeads[[rowIdx]],
4660               separator,
4661               colHeads[[colIdx]]}]
4662         ]
4663       ]
4664     ],
4665   {rowIdx, Dimensions[data][[1]]},
4666   {colIdx, Dimensions[data][[2]]}];
4667 gridList=Transpose[Prepend[gridList, colHeads]];
4668 rowHeads=Prepend[rowHeads, pivot];
4669 gridList=Prepend[gridList, rowHeads]//Transpose;
4670 Grid[gridList,
4671   Frame->OptionValue[Frame],
4672   Alignment->OptionValue[Alignment],
4673   Frame->OptionValue[Frame],
4674   ItemSize->OptionValue[ItemSize]
4675   ]
4676 ]
4677 ];
4678
4679 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.";
4680 Options[HamiltonianForm]={"Separator"->"", "Pivot"->};
4681 HamiltonianForm[hamMatrix_, basisLabels_List, OptionsPattern[]] :=
4682 (
4683   braLabels=DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]]& /@ basisLabels;
4684   ketLabels=DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"}]]& /@ basisLabels;
4685   LabeledGrid[hamMatrix, braLabels, ketLabels, "Separator"->
4686   OptionValue["Separator"], "Pivot"->OptionValue["Pivot"]]
4687 )
4688
4689 HamiltonianMatrixPlot::usage = "HamiltonianMatrixPlot[hamMatrix,
basisLabels] creates a matrix plot of the given hamiltonian matrix
with the given basis labels. The matrix elements can be hovered
over to display the corresponding row and column labels together
with the value of the matrix element. The option \\"OverlayValues\\"
can be used to specify whether the matrix elements should be
displayed on top of the matrix plot.";
4690 Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover"-
4691   -> True, "OverlayValues" -> True}];
4692 HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts : OptionsPattern[]] := (
4693   braLabels = DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]] & /@ basisLabels;
4694   ketLabels = DisplayForm[Rotate[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"}], \[Pi]/2]] & /@ basisLabels;
4695   ketLabelsUpright = DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"}]] & /@ basisLabels;
4696   numRows = Length[hamMatrix];
4697   numCols = Length[hamMatrix[[1]]];
4698   epiThings = Which[
4699     And[OptionValue["Hover"], Not[OptionValue["OverlayValues"]]], ,
4700     Flatten[
4701       Table[
4702         Tooltip[
4703           {
4704             Transparent,
4705             Rectangle[
4706               {j - 1, numRows - i},
4707               {j - 1, numRows - i} + {1, 1}
4708             ]
4709           },
4710           Row[{braLabels[[i]], ketLabelsUpright[[j]], "=" ,hamMatrix[[i,
4711           j]]}]]
4712         ],
4713       ],
4714       {i, 1, numRows},

```

```

4710     {j, 1, numCols}
4711     ]
4712   ],
4713   And[OptionValue["Hover"], OptionValue["OverlayValues"]],
4714   Flatten[
4715     Table[
4716       Tooltip[
4717         {
4718           Transparent,
4719           Rectangle[
4720             {j - 1, numRows - i},
4721             {j - 1, numRows - i} + {1, 1}
4722           ]
4723         },
4724         DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i
4725 ]], "\[LeftBracketingBar]", basisLabels[[j]], "\[RightAngleBracket
4726 ]}]]]
4727       ],
4728       {i, numRows},
4729       {j, numCols}
4730     ]
4731   ],
4732   True,
4733   {}
4734 ];
4735 textOverlay = If[OptionValue["OverlayValues"],
4736 (
4737   Flatten[
4738     Table[
4739       Text[hamMatrix[[i, j]],
4740         {j - 1/2, numRows - i + 1/2}]
4741       ],
4742       {i, 1, numRows},
4743       {j, 1, numCols}
4744     ]
4745   ]
4746 );
4747 epiThings = Join[epiThings, textOverlay];
4748 MatrixPlot[hamMatrix,
4749   FrameTicks -> {
4750     {Transpose[{Range[Length[braLabels]], braLabels}], None},
4751     {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}}
4752   ],
4753   Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
4754   Epilog -> epiThings
4755 ]
4756 );
4757 (* ##### Some Plotting Routines ##### *)
4758 (* ##### Load Functions ##### *)
4759
4760
4761 (* ##### Load Functions ##### *)
4762 (* ##### Load Functions ##### *)
4763
4764 LoadAll::usage = "LoadAll[] executes most Load* functions.";
4765 LoadAll[] := (
4766   LoadTermLabels[];
4767   LoadCFP[];
4768   LoadUk[];
4769   LoadV1k[];
4770   LoadT22[];
4771   LoadSOOandECSOLs[];
4772
4773   LoadElectrostatic[];
4774   LoadSpinOrbit[];
4775   LoadSOOandECSO[];
4776   LoadSpinSpin[];
4777   LoadThreeBody[];
4778   LoadChenDeltas[];
4779   LoadCarnall[];
4780 );
4781
4782 fnTermLabels::usage = "This list contains the labels of f^n
4783 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.";

4783 LoadTermLabels::usage = "LoadTermLabels[] loads into the session
4784   the labels for the terms in the f^n configurations.";
4785 LoadTermLabels[] := (
4786   If[ValueQ[fnTermLabels], Return[]];
4787   PrintTemporary["Loading data for state labels in the f^n
4788   configurations..."];
4789   fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
4790
4791   If[!FileExistsQ[fnTermsFname],
4792     (PrintTemporary[">> fnTerms.m not found, generating ..."];
4793      fnTermLabels = ParseTermLabels["Export" -> True];
4794    ),
4795    fnTermLabels = Import[fnTermsFname];
4796  ];
4797 )
4798
4799 Carnall::usage = "Association of data from Carnall et al (1989)
500   with the following keys: {data, annotations, paramSymbols,
501   elementNames, rawData, rawAnnotations, annotatedData, appendix:Pr
502   :Association, appendix:Pr:Calculated, appendix:Pr:RawTable,
503   appendix:Headings}";

504 LoadCarnall::usage = "LoadCarnall[] loads data for trivalent
505   lanthanides in LaF3 using the data from Bill Carnall's 1989 paper.
506 ";
507 LoadCarnall[] := (
508   If[ValueQ[Carnall], Return[]];
509   carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
510   If[!FileExistsQ[carnallFname],
511     (PrintTemporary[">> Carnall.m not found, generating ..."];
512      Carnall = ParseCarnall[];
513    ),
514    Carnall = Import[carnallFname];
515  ];
516 )
517
518 LoadChenDeltas::usage = "LoadChenDeltas[] loads the differences
519   noted by Chen.";
520 LoadChenDeltas[] := (
521   If[ValueQ[chenDeltas], Return[]];
522   PrintTemporary["Loading the association of discrepancies found by
523   Chen ..."];
524   chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"}];
525
526   If[!FileExistsQ[chenDeltasFname],
527     (PrintTemporary[">> chenDeltas.m not found, generating ..."];
528       chenDeltas = ParseChenDeltas[];
529     ),
530       chenDeltas = Import[chenDeltasFname];
531   ];
532 )
533
534 ParseChenDeltas::usage = "ParseChenDeltas[] parses the data found
535   in ./data/the-chen-deltas-A.csv and ./data/the-chen-deltas-B.csv.
536   If the option \"Export\" is set to True (True is the default),
537   then the parsed data is saved to ./data/chenDeltas.m";
538 Options[ParseChenDeltas] = {"Export" -> True};
539 ParseChenDeltas[OptionsPattern[]] := (
540   chenDeltasRaw = Import[FileNameJoin[{moduleDir, "data", "the-chen
541   -deltas-A.csv"}]];
542   chenDeltasRaw = chenDeltasRaw[[2 ;;]];
543   chenDeltas = <||>;
544   chenDeltasA = <||>;
545   Off[Power::infy];
546   Do[
547     {right, wrong} = {chenDeltasRaw[[row]][[4 ;;]],

548      chenDeltasRaw[[row + 1]][[4 ;;]]];
549     key = chenDeltasRaw[[row]][[1 ;; 3]];
550     repRule = (#[[1]] -> #[[2]]*#[[1]]) & /@
551       Transpose[{{M0, M2, M4, P2, P4, P6}, right/wrong}];
552     chenDeltasA[key] = <|"right" -> right, "wrong" -> wrong,
553     "repRule" -> repRule|>;
554   ];
555 )

```

```

4841     chenDeltasA[{key[[1]], key[[3]], key[[2]]}] = <|"right" ->
4842     right,
4843     "wrong" -> wrong, "repRule" -> repRule|>;
4844   ),
4845   {row, 1, Length[chenDeltasRaw], 2}];
4846   chenDeltas["A"] = chenDeltasA;
4847
4848   chenDeltasRawB = Import[FileNameJoin[{moduleDir, "data", "the-
4849   chen-deltas-B.csv"}], "Text"];
4850   chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
4851   chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
4852   chenDeltasB = {ToExpression[StringTake[#[[1]], {2}]], #[[2]],
4853   #[[3]]} & /@ chenDeltasB;
4854   chenDeltas["B"] = chenDeltasB;
4855   On[Power::infy];
4856   If[OptionValue["Export"],
4857     (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.
4858     m"}];
4859     Export[chenDeltasFname, chenDeltas];
4860   )
4861 ];
4862
4863 Return[chenDeltas];
4864
4865 ParseCarnall::usage = "ParseCarnall[] parses the data found in ./
4866   data/Carnall.xls. If the option \"Export\" is set to True (True is
4867   the default), then the parsed data is saved to ./data/Carnall.
4868   This data is from the tables and appendices of Carnall et al
4869   (1989).";
4870 Options[ParseCarnall] = {"Export" -> True};
4871 ParseCarnall[OptionsPattern[]] := (
4872   ions = {"Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho",
4873   "Er", "Tm", "Yb"};
4874   templates = StringTemplate/@StringSplit["appendix`ion`:
4875   Association appendix`ion`:Calculated appendix`ion`:RawTable
4876   appendix`ion`:Headings", " "];
4877
4878 (* How many unique eigenvalues, after removing Kramer's
4879 degeneracy *)
4880 fullSizes = AssociationThread[ions, {7, 91, 182, 1001, 1001,
4881 3003, 1716, 3003, 1001, 1001, 182, 91, 7}];
4882 carnall = Import[FileNameJoin[{moduleDir, "data", "Carnall.xls
4883 "}]][[2]];
4884 carnallErr = Import[FileNameJoin[{moduleDir, "data", "Carnall.xls
4885 "}]][[3]];
4886
4887 elementNames = carnall[[1]][[2;;]];
4888 carnall = carnall[[2;;]];
4889 carnallErr = carnallErr[[2;;]];
4890 carnall = Transpose[carnall];
4891 carnallErr = Transpose[carnallErr];
4892 paramNames = ToExpression/@carnall[[1]][[1;;]];
4893 carnall = carnall[[2;;]];
4894 carnallErr = carnallErr[[2;;]];
4895 carnallData = Table[(
4896   data = carnall[[i]];
4897   data = (#[[1]] -> #[[2]]) & /@ Select[
4898     Transpose[{paramNames, data}], #[[2]] != "" &];
4899     elementNames[[i]] -> data
      ),
      {i, 1, 13}
    ];
4900 carnallData = Association[carnallData];
4901 carnallNotes = Table[(
4902   data = carnallErr[[i]];
4903   elementName = elementNames[[i]];
4904   dataFun = (
4905     #[[1]] -> If[#[[2]] == {}, ,
4906     "Not allowed to vary in fitting.", ,
4907     If[#[[2]] == "[R]", ,
4908       "Ratio constrained by: " <> <|"Eu" -> "F4/
4909       F2=0.713; F6/F2=0.512",
4910         "Gd" -> "F4/F2=0.710] ",
4911         "Tb" -> "F4/F2=0.707" |> [elementName],
4912       If[#[[2]] == "i",
4913         "Interpolated",
4914         " "
4915       ]
4916     ]
4917   )
4918 );
4919
4920
4921
4922
4923
4924
4925
4926
4927
4928
4929
4930
4931
4932
4933
4934
4935
4936
4937
4938
4939
4940
4941
4942
4943
4944
4945
4946
4947
4948
4949
4950
4951
4952
4953
4954
4955
4956
4957
4958
4959
4960
4961
4962
4963
4964
4965
4966
4967
4968
4969
4970
4971
4972
4973
4974
4975
4976
4977
4978
4979
4980
4981
4982
4983
4984
4985
4986
4987
4988
4989
4990
4991
4992
4993
4994
4995
4996
4997
4998
4999
5000
5001
5002
5003
5004
5005
5006
5007
5008
5009
5010
5011
5012
5013
5014
5015
5016
5017
5018
5019
5020
5021
5022
5023
5024
5025
5026
5027
5028
5029
5030
5031
5032
5033
5034
5035
5036
5037
5038
5039
5040
5041
5042
5043
5044
5045
5046
5047
5048
5049
5050
5051
5052
5053
5054
5055
5056
5057
5058
5059
5060
5061
5062
5063
5064
5065
5066
5067
5068
5069
5070
5071
5072
5073
5074
5075
5076
5077
5078
5079
5080
5081
5082
5083
5084
5085
5086
5087
5088
5089
5090
5091
5092
5093
5094
5095
5096
5097
5098
5099
5100
5101
5102
5103
5104
5105
5106
5107
5108
5109
5110
5111
5112
5113
5114
5115
5116
5117
5118
5119
5120
5121
5122
5123
5124
5125
5126
5127
5128
5129
5130
5131
5132
5133
5134
5135
5136
5137
5138
5139
5140
5141
5142
5143
5144
5145
5146
5147
5148
5149
5150
5151
5152
5153
5154
5155
5156
5157
5158
5159
5160
5161
5162
5163
5164
5165
5166
5167
5168
5169
5170
5171
5172
5173
5174
5175
5176
5177
5178
5179
5180
5181
5182
5183
5184
5185
5186
5187
5188
5189
5190
5191
5192
5193
5194
5195
5196
5197
5198
5199
5200
5201
5202
5203
5204
5205
5206
5207
5208
5209
5210
5211
5212
5213
5214
5215
5216
5217
5218
5219
5220
5221
5222
5223
5224
5225
5226
5227
5228
5229
5230
5231
5232
5233
5234
5235
5236
5237
5238
5239
5240
5241
5242
5243
5244
5245
5246
5247
5248
5249
5250
5251
5252
5253
5254
5255
5256
5257
5258
5259
5260
5261
5262
5263
5264
5265
5266
5267
5268
5269
5270
5271
5272
5273
5274
5275
5276
5277
5278
5279
5280
5281
5282
5283
5284
5285
5286
5287
5288
5289
5290
5291
5292
5293
5294
5295
5296
5297
5298
5299
5300
5301
5302
5303
5304
5305
5306
5307
5308
5309
5310
5311
5312
5313
5314
5315
5316
5317
5318
5319
5320
5321
5322
5323
5324
5325
5326
5327
5328
5329
5330
5331
5332
5333
5334
5335
5336
5337
5338
5339
5340
5341
5342
5343
5344
5345
5346
5347
5348
5349
5350
5351
5352
5353
5354
5355
5356
5357
5358
5359
5360
5361
5362
5363
5364
5365
5366
5367
5368
5369
5370
5371
5372
5373
5374
5375
5376
5377
5378
5379
5380
5381
5382
5383
5384
5385
5386
5387
5388
5389
5390
5391
5392
5393
5394
5395
5396
5397
5398
5399
5400
5401
5402
5403
5404
5405
5406
5407
5408
5409
5410
5411
5412
5413
5414
5415
5416
5417
5418
5419
5420
5421
5422
5423
5424
5425
5426
5427
5428
5429
5430
5431
5432
5433
5434
5435
5436
5437
5438
5439
5440
5441
5442
5443
5444
5445
5446
5447
5448
5449
5450
5451
5452
5453
5454
5455
5456
5457
5458
5459
5460
5461
5462
5463
5464
5465
5466
5467
5468
5469
5470
5471
5472
5473
5474
5475
5476
5477
5478
5479
5480
5481
5482
5483
5484
5485
5486
5487
5488
5489
5490
5491
5492
5493
5494
5495
5496
5497
5498
5499
5500
5501
5502
5503
5504
5505
5506
5507
5508
5509
5510
5511
5512
5513
5514
5515
5516
5517
5518
5519
5520
5521
5522
5523
5524
5525
5526
5527
5528
5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549
5550
5551
5552
5553
5554
5555
5556
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585
5586
5587
5588
5589
5590
5591
5592
5593
5594
5595
5596
5597
5598
5599
5600
5601
5602
5603
5604
5605
5606
5607
5608
5609
5610
5611
5612
5613
5614
5615
5616
5617
5618
5619
5620
5621
5622
5623
5624
5625
5626
5627
5628
5629
5630
5631
5632
5633
5634
5635
5636
5637
5638
5639
5640
5641
5642
5643
5644
5645
5646
5647
5648
5649
5650
5651
5652
5653
5654
5655
5656
5657
5658
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668
5669
5670
5671
5672
5673
5674
5675
5676
5677
5678
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5699
5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5720
5721
5722
5723
5724
5725
5726
5727
5728
5729
5730
5731
5732
5733
5734
5735
5736
5737
5738
5739
5740
5741
5742
5743
5744
5745
5746
5747
5748
5749
5750
5751
5752
5753
5754
5755
5756
5757
5758
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5850
5851
5852
5853
5854
5855
5856
5857
5858
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5970
5971
5972
5973
5974
5975
5976
5977
5978
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
5999

```

```

4900                                #[[2]]
4901                                ]
4902                                ]
4903                                ]) &;
4904                                data = dataFun /@ Select[Transpose[{paramNames,
4905                                data}]] ,#[[2]]!="&];
4906                                elementName->data
4907                                ),
4908                                {i,1,13}
4909                                ];
4910                                carnallNotes = Association[carnallNotes];
4911
4912                                annotatedData = Table[
4913                                    If[NumberQ[#[[1]]], Tooltip[#[[1]], #[[2]]], ""]
4914                                    & /
4915                                    @ Transpose[{paramNames/.carnallData[element],
4916                                    paramNames/.carnallNotes[element]
4917                                    }], {element, elementNames}
4918                                ];
4919                                annotatedData = Transpose[annotatedData];
4920
4921                                Carnall = <|"data"      -> carnallData,
4922                                "annotations"   -> carnallNotes,
4923                                "paramSymbols" -> paramNames,
4924                                "elementNames"  -> elementNames,
4925                                "rawData"       -> carnall,
4926                                "rawAnnotations" -> carnallErr,
4927                                "includedTableIons" -> ions,
4928                                "annnotatedData"  -> annotatedData
4929                                |>;
4930
4931                                Do[(
4932                                    carnallData = Import[FileNameJoin[{moduleDir, "data",
4933                                    "Carnall.xls"}]] [[sheetIdx]];
4934                                    headers = carnallData[[1]];
4935                                    calcIndex = Position[headers, "Calc (1/cm)"][[1,1]];
4936                                    headers = headers[[2;;]];
4937                                    carnallLabels = carnallData[[1]];
4938                                    carnallData = carnallData[[2;;]];
4939                                    carnallTerms = DeleteDuplicates[First/@carnallData];
4940                                    parsedData = Table[(
4941                                        rows = Select[carnallData, #[[1]]==term&];
4942                                        rows = #[[2;;]]&@rows;
4943                                        rows = Transpose[rows];
4944                                        rows = Transpose[{headers, rows}];
4945                                        rows = Association[({#[[1]] -> #[[2]]})&@rows
4946                                    ];
4947                                    term->rows
4948                                    ),
4949                                    {term, carnallTerms}
4950                                ];
4951                                    carnallAssoc = Association[parsedData];
4952                                    carnallCalcEnergies = #[[calcIndex]]&@carnallData;
4953                                    carnallCalcEnergies = If[NumberQ[#], #, Missing[]]&/
4954                                    @carnallCalcEnergies;
4955                                    ion = ions[[sheetIdx-3]];
4956                                    carnallCalcEnergies = PadRight[carnallCalcEnergies, fullSizes
4957                                    [ion], Missing[]];
4958                                    keys = #[<|"ion"->ion|]&@templates;
4959                                    Carnall[keys[[1]]] = carnallAssoc;
4960                                    Carnall[keys[[2]]] = carnallCalcEnergies;
4961                                    Carnall[keys[[3]]] = carnallData;
4962                                    Carnall[keys[[4]]] = headers;
4963                                ),
4964                                {sheetIdx, 4, 16}
4965                            ];
4966
4967                                goodions = Select[ions, #!="Pm"&];
4968                                expData = Select[Transpose[Carnall["appendix:<>#<>":RawTable"
4969                                ]][[1+Position[Carnall["appendix:<>#<>":Headings],"Exp (1/cm)""
4970                                ][[1,1]]]], NumberQ]&@goodions;
4971                                Carnall["All Experimental Data"] = AssociationThread[goodions,
4972                                expData];
4973                                If[OptionValue["Export"],
4974                                (
4975                                    carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}]
```

```

    }];
4967    Echo["Exporting to " <> carnallFname];
4968    Export[carnallFname, Carnall];
4969  )
4970  ];
4971  Return[Carnall];
4972 );
4973
4974 CFP::usage = "CFP[{n, NKSL}] provides a list whose first element
4975   echoes NKSL and whose other elements are lists with two elements
4976   the first one being the symbol of a parent term and the second
4977   being the corresponding coefficient of fractional parentage. n
4978   must satisfy 1 <= n <= 7.
4979 These are according to the tables from Nielson & Koster.";
4980
4981 CFPAssoc::usage = "CFPAssoc is an association where keys are of
4982   lists of the form {num_electrons, daughterTerm, parentTerm} and
4983   values are the corresponding coefficients of fractional parentage.
4984   The terms given in string-spectroscopic notation. If a certain
4985   daughter term does not have a parent term, the value is 0. Loaded
4986   using LoadCFP[].
4987 These are according to the tables from Nielson & Koster.";
4988
4989 LoadCFP::usage = "LoadCFP[] loads CFP, CFPAssoc, and CFPTable into
4990   the session.";
4991 LoadCFP[] := (
4992   If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]], Return
4993   []];
4994
4995   PrintTemporary["Loading CFPTable ..."];
4996   CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
4997   If[!FileExistsQ[CFPTablefname],
4998     (PrintTemporary[">> CFPTable.m not found, generating ..."];
4999       CFPTable = GenerateCFPTable["Export" -> True];
5000     ),
5001     CFPTable = Import[CFPTablefname];
5002   ];
5003
5004   PrintTemporary["Loading CFPs.m ..."];
5005   CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
5006   If[!FileExistsQ[CFPfname],
5007     (PrintTemporary[">> CFPs.m not found, generating ..."];
5008       CFP = GenerateCFP["Export" -> True];
5009     ),
5010     CFP = Import[CFPfname];
5011   ];
5012
5013   PrintTemporary["Loading CFPAssoc.m ..."];
5014   CFPAfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
5015   If[!FileExistsQ[CFPAfname],
5016     (PrintTemporary[">> CFPAssoc.m not found, generating ..."];
5017       CFPAssoc = GenerateCFPAssoc["Export" -> True];
5018     ),
5019     CFPAssoc = Import[CFPAfname];
5020   ];
5021
5022 ReducedUkTable::usage = "ReducedUkTable[{n, l = 3, SL, SpLp, k}]
5023   provides reduced matrix elements of the unit spherical tensor
5024   operator Uk. See TASS section 11-9 \"Unit Tensor Operators\".
5025   Loaded using LoadUk[].";
```

```

5024     ];
5025 );
5026
5027 ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}]"
5028      provides the calculated result of Electrostatic[{n, SL, SpLp}].  

5029      Load using LoadElectrostatic[] .";
5030
5031 LoadElectrostatic::usage = "LoadElectrostatic[] loads the reduced
5032      matrix elements for the electrostatic interaction.";
5033 LoadElectrostatic[] := (
5034   If[ValueQ[ElectrostaticTable], Return[]];
5035   PrintTemporary["Loading the association of matrix elements for
5036      the electrostatic interaction ..."];
5037   ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}];
5038   If[!FileExistsQ[ElectrostaticTablefname],
5039     (PrintTemporary[">> ElectrostaticTable.m not found, generating
5040      ..."]);
5041     ElectrostaticTable = GenerateElectrostaticTable[7];
5042   ),
5043   ElectrostaticTable = Import[ElectrostaticTablefname];
5044 ];
5045 )
5046
5047 LoadV1k::usage = "LoadV1k[] loads into session the matrix elements
5048      of V1k.";
5049 LoadV1k[] := (
5050   If[ValueQ[ReducedV1kTable], Return[]];
5051   PrintTemporary["Loading the association of matrix elements for
5052      V1k ..."];
5053   ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m"}];
5054   If[!FileExistsQ[ReducedV1kTableFname],
5055     (PrintTemporary[">> ReducedV1kTable.m not found, generating ...
5056      "]);
5057     ReducedV1kTable = GenerateReducedV1kTable[7];
5058   ),
5059   ReducedV1kTable = Import[ReducedV1kTableFname];
5060 ];
5061 )
5062
5063 LoadSpinOrbit::usage = "LoadSpinOrbit[] loads into session the
5064      matrix elements of the spin-orbit interaction.";
5065 LoadSpinOrbit[] := (
5066   If[ValueQ[SpinOrbitTable], Return[]];
5067   PrintTemporary["Loading the association of matrix elements for
5068      spin-orbit ..."];
5069   SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
5070   If[!FileExistsQ[SpinOrbitTableFname],
5071     (
5072       PrintTemporary[">> SpinOrbitTable.m not found, generating ...
5073      "];
5074       SpinOrbitTable = GenerateSpinOrbitTable[7, "Export" -> True];
5075     ),
5076     SpinOrbitTable = Import[SpinOrbitTableFname];
5077   ];
5078 );
5079
5080 LoadSOOandECSOLS::usage = "LoadSOOandECSOLS[] loads into session
5081      the LS reduced matrix elements of the SOO-ECSO interaction.";
5082 LoadSOOandECSOLS[] := (
5083   If[ValueQ[SOOandECSOLSTable], Return[]];
5084   PrintTemporary["Loading the association of LS reduced matrix
5085      elements for SOO-ECSO ..."];
5086   SOOandECSOLSTableFname = FileNameJoin[{moduleDir, "data", "ReducedSOOandECSOLSTable.m"}];
5087   If[!FileExistsQ[SOOandECSOLSTableFname],
5088     (PrintTemporary[">> ReducedSOOandECSOLSTable.m not found,
5089      generating ..."]);
5090     SOOandECSOLSTable = GenerateSOOandECSOLSTable[7];
5091   ),
5092   SOOandECSOLSTable = Import[SOOandECSOLSTableFname];
5093 ];
5094 )

```

```

5082 LoadSOOandECSO::usage = "LoadSOOandECSO[] loads into session the
5083   LSJ reduced matrix elements of spin-other-orbit and
5084   electrostatically-correlated-spin-orbit.";
5085 LoadSOOandECSO[] := (
5086   If[ValueQ[SOOandECSOTableFname], Return[]];
5087   PrintTemporary["Loading the association of matrix elements for
5088     spin-other-orbit and electrostatically-correlated-spin-orbit ..."];
5089   SOOandECSOTableFname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
5090   If[!FileExistsQ[SOOandECSOTableFname],
5091     (PrintTemporary[">> SOOandECSOTable.m not found, generating ..."]);
5092     SOOandECSOTable = GenerateSOOandECSOTable[7, "Export" -> True];
5093   ],
5094   SOOandECSOTable = Import[SOOandECSOTableFname];
5095 );
5096 );
5097
5098 LoadT22::usage = "LoadT22[] loads into session the matrix elements
5099   of the double tensor operator T22.";
5100 LoadT22[] := (
5101   If[ValueQ[T22Table], Return[]];
5102   PrintTemporary["Loading the association of reduced T22 matrix
5103     elements ..."];
5104   T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.
5105 .m"}];
5106   If[!FileExistsQ[T22TableFname],
5107     (PrintTemporary[">> ReducedT22Table.m not found, generating ..."]);
5108     T22Table = GenerateT22Table[7];
5109   ],
5110   T22Table = Import[T22TableFname];
5111 );
5112 );
5113
5114 LoadSpinSpin::usage = "LoadSpinSpin[] loads into session the matrix
5115   elements of spin-spin.";
5116 LoadSpinSpin[] := (
5117   If[ValueQ[SpinSpinTable], Return[]];
5118   PrintTemporary["Loading the association of matrix elements for
5119     spin-spin ..."];
5120   SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
5121   If[!FileExistsQ[SpinSpinTableFname],
5122     (PrintTemporary[">> SpinSpinTable.m not found, generating ..."]);
5123     SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
5124   ],
5125   SpinSpinTable = Import[SpinSpinTableFname];
5126 );
5127 );
5128
5129 LoadThreeBody::usage = "LoadThreeBody[] loads into session the
5130   matrix elements of three-body configuration-interaction effects.";
5131 LoadThreeBody[] := (
5132   If[ValueQ[ThreeBodyTable], Return[]];
5133   PrintTemporary["Loading the association of matrix elements for
5134     three-body configuration-interaction effects ..."];
5135   ThreeBodyFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
5136   ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
5137   If[!FileExistsQ[ThreeBodyFname],
5138     (PrintTemporary[">> ThreeBodyTable.m not found, generating ..."]);
5139     {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTables["
5140       Export" -> True];
5141   ],
5142   ThreeBodyTable = Import[ThreeBodyFname];
5143   ThreeBodyTables = Import[ThreeBodiesFname];
5144 );
5145
5146 (* ##### Load Functions ##### *)
5147 (* ##### ##### ##### ##### *)

```

```
5138 |  
5139 End [] ;  
5140  
5141 LoadTermLabels [] ;  
5142 LoadCFP [] ;  
5143  
5144 EndPackage [] ;
```

## 18.2 fittings.m

This file has code useful for fitting the Hamiltonian.

```

59      },
60 "Eu" -> {
61   F4 -> 0.713 F2,
62   F6 -> 0.512 F2,
63   B22 -> -50.,
64   B24 -> 597.,
65   B26 -> -706.,
66   B44 -> 408.,
67   B46 -> -508.,
68   B66 -> -692.,
69   M0 -> 2.1,
70   P2 -> 360.,
71   T2 -> 300.,
72   T3 -> 40.,
73   T4 -> 60.,
74   T6 -> -300.,
75   T7 -> 370.,
76   T8 -> 320.,
77    $\alpha$  -> 20.16,
78    $\beta$  -> -566.9,
79    $\gamma$  -> 1500.
80 },
81 "Pm" -> {
82   B02 -> -245.,
83   B04 -> 470.,
84   B06 -> 640.,
85   B22 -> -50.,
86   B24 -> 525.,
87   B26 -> -750.,
88   B44 -> 490.,
89   B46 -> -450.,
90   B66 -> -760.,
91   F2 -> 76400.,
92   F4 -> 54900.,
93   F6 -> 37700.,
94   M0 -> 2.4,
95   P2 -> 275.,
96   T2 -> 300.,
97   T3 -> 35.,
98   T4 -> 58.,
99   T6 -> -310.,
100  T7 -> 350.,
101  T8 -> 320.,
102   $\alpha$  -> 20.5,
103   $\beta$  -> -560.,
104   $\gamma$  -> 1475.,
105   $\zeta$  -> 1025.},
106 "Gd" -> {
107   F4 -> 0.710 F2,
108   B02 -> -231.,
109   B04 -> 604.,
110   B06 -> 280.,
111   B22 -> -99.,
112   B24 -> 340.,
113   B26 -> -721.,
114   B44 -> 452.,
115   B46 -> -204.,
116   B66 -> -509.,
117   T2 -> 300.,
118   T3 -> 42.,
119   T4 -> 62.,
120   T6 -> -295.,
121   T7 -> 350.,
122   T8 -> 310.,
123    $\beta$  -> -600.,
124    $\gamma$  -> 1575.
125 },
126 "Tb" -> {
127   F4 -> 0.707 F2,
128   T2 -> 320.,
129   T3 -> 40.,
130   T4 -> 50.,
131    $\gamma$  -> 1650.
132 },
133 "Dy" -> {},
134 "Ho" -> {

```

```

135     B02 -> -240.,
136     T2 -> 400.,
137     γ -> 1800.
138   },
139 "Er" -> {
140     T2 -> 400.,
141     γ -> 1800.
142   },
143 "Tm" -> {
144     T2 -> 400.,
145     γ -> 1820.
146   },
147 "Yb" -> {
148     B02 -> -249.,
149     B04 -> 457.,
150     B06 -> 282.,
151     B22 -> -105.,
152     B24 -> 320.,
153     B26 -> -482.,
154     B44 -> 428.,
155     B46 -> -234.,
156     B66 -> -492.
157 }
158 |>;
159
160 variedSymbols =<|
161   "Ce" -> {ζ},
162   "Pr" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
163     F2, F4, F6,
164     M0, P2,
165     α, β, γ,
166     ζ},
167   "Nd" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
168     F2, F4, F6,
169     M0, P2,
170     T2, T3, T4, T6, T7, T8,
171     α, β, γ,
172     ζ},
173   "Pm" -> {},
174   "Sm" -> {B02, B04, B06, B24, B26, B44, B46, B66,
175     F2, F4, F6, M0, P2,
176     T6, T7, T8,
177     α, β, ζ},
178   "Eu" -> {B02, B04, B06,
179     F2, F4, F6, ζ},
180   "Gd" -> {F2, F4, F6,
181     M0, P2,
182     α, ζ},
183   "Tb" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
184     F2, F4, F6,
185     M0, P2,
186     T6, T7, T8,
187     α, β, ζ},
188   "Dy" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
189     F2, F4, F6,
190     M0, P2,
191     T2, T3, T4, T6, T7, T8,
192     α, β, γ, ζ},
193   "Ho" -> {B04, B06, B22, B24, B26, B44, B46, B66,
194     F2, F4, F6,
195     M0, P2,
196     T3, T4, T6, T7, T8,
197     α, β, ζ},
198   "Er" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
199     F2, F4, F6,
200     M0, P2,
201     T3, T4, T6, T7, T8, α, β, ζ},
202   "Tm" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
203     F2, F4, F6,
204     M0, P2,
205     α, β, ζ},
206   "Yb" -> {ζ}
207 |>;
208
209 caseConstraintsM0::usage="This association contains the symbols that
210   are held constant in fitting different ions. It only contains

```

```

    symbols for which the constraint doesn't apply to all ions.
210 This association has keys equal to symbols of lanthanides and values
     equal to associations with the symbols for the parameters that are
     held fixed or made proportional to another. If the value is to be
     held constant a placeholder value of 0 is given, if a ratio
     constraint is given, the value is constraint.
211 The operator basis for which this is applicable is the mostly-
     orthogonal basis.
212 ";
213 caseConstraintsM0 = <|
214 "Ce" -> {
215     B02 -> 0,
216     B04 -> 0,
217     B06 -> 0,
218     B22 -> 0,
219     B24 -> 0,
220     B26 -> 0,
221     B44 -> 0,
222     B46 -> 0,
223     B66 -> 0
224 },
225 "Pr" -> {},
226 "Nd" -> {},
227 "Sm" -> {
228     B22 -> 0,
229     T2p -> 0,
230     T3 -> 0,
231     T4 -> 0,
232     γp -> 0
233 },
234 "Eu" -> {
235     E2p -> 0.0049 E1p,
236     E3p -> 0.098 E1p,
237     B22 -> 0,
238     B24 -> 0,
239     B26 -> 0,
240     B44 -> 0,
241     B46 -> 0,
242     B66 -> 0,
243     M0 -> 0,
244     P2 -> 0,
245     T2p -> 0,
246     T3 -> 0,
247     T4 -> 0,
248     T6 -> 0,
249     T7 -> 0,
250     T8 -> 0,
251     αp -> 0,
252     βp -> 0,
253     γp -> 0
254 },
255 "Pm" -> {
256     B02 -> 0,
257     B04 -> 0,
258     B06 -> 0,
259     B22 -> 0,
260     B24 -> 0,
261     B26 -> 0,
262     B44 -> 0,
263     B46 -> 0,
264     B66 -> 0,
265     E1p -> 0,
266     E2p -> 0,
267     E3p -> 0,
268     M0 -> 0,
269     P2 -> 0,
270     T2p -> 0,
271     T3 -> 0,
272     T4 -> 0,
273     T6 -> 0,
274     T7 -> 0,
275     T8 -> 0,
276     αp -> 0,
277     βp -> 0,
278     γp -> 0,
279     ζ -> 0

```

```

280     },
281 "Gd" -> {
282     E2p -> 0.0049 E1p,
283     B02 -> 0,
284     B04 -> 0,
285     B06 -> 0,
286     B22 -> 0,
287     B24 -> 0,
288     B26 -> 0,
289     B44 -> 0,
290     B46 -> 0,
291     B66 -> 0,
292     T2p -> 0,
293     T3 -> 0,
294     T4 -> 0,
295     T6 -> 0,
296     T7 -> 0,
297     T8 -> 0,
298     βp -> 0,
299     γp -> 0
300   },
301 "Tb" -> {
302     E2p -> 0.0049 E1p,
303     T2p -> 0,
304     T3 -> 0,
305     T4 -> 0,
306     γp -> 0
307   },
308 "Dy" -> {},
309 "Ho" -> {
310     B02 -> 0,
311     T2p -> 0,
312     γp -> 0
313   },
314 "Er" -> {
315     T2p -> 0,
316     γp -> 0
317   },
318 "Tm" -> {
319     γp -> 0
320   },
321 "Yb" -> {
322     B02 -> 0,
323     B04 -> 0,
324     B06 -> 0,
325     B22 -> 0,
326     B24 -> 0,
327     B26 -> 0,
328     B44 -> 0,
329     B46 -> 0,
330     B66 -> 0
331   }
332 |>;
333
334 variedSymbolsM0 = <|
335   "Ce" -> {ζ},
336   "Pr" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
337     E1p, E2p, E3p,
338     M0, P2,
339     αp, βp, γp,
340     ζ},
341   "Nd" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
342     E1p, E2p, E3p,
343     M0, P2,
344     T2p, T3, T4, T6, T7, T8,
345     αp, βp, γp,
346     ζ},
347   "Pm" -> {},
348   "Sm" -> {B02, B04, B06, B24, B26, B44, B46, B66,
349     E1p, E2p, E3p, M0, P2,
350     T6, T7, T8,
351     αp, βp, ζ},
352   "Eu" -> {B02, B04, B06,
353     E1p, E2p, E3p, ζ},
354   "Gd" -> {E1p, E2p, E3p,
355     M0, P2,

```

```

356       $\alpha_p, \zeta\}$ ,
357  "Tb" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
358      E1p, E2p, E3p,
359      M0, P2,
360      T6, T7, T8,
361       $\alpha_p, \beta_p, \zeta\}$ ,
362  "Dy" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
363      E1p, E2p, E3p,
364      M0, P2,
365      T2p, T3, T4, T6, T7, T8,
366       $\alpha_p, \beta_p, \gamma_p, \zeta\}$ ,
367  "Ho" -> {B04, B06, B22, B24, B26, B44, B46, B66,
368      E1p, E2p, E3p,
369      M0, P2,
370      T3, T4, T6, T7, T8,
371       $\alpha_p, \beta_p, \zeta\}$ ,
372  "Er" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
373      E1p, E2p, E3p,
374      M0, P2,
375      T3, T4, T6, T7, T8,  $\alpha_p, \beta_p, \zeta\}$ ,
376  "Tm" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
377      E1p, E2p, E3p,
378      M0, P2,
379       $\alpha_p, \beta_p, \zeta\}$ ,
380  "Yb" -> {\zeta}
381  | >;
382
383 caseConstraintsLiYF4 = <|
384  "Ce" -> {
385      B04 -> -1043.,
386      B44 -> -1249.,
387      B06 -> -65.,
388      B46 -> -1069.
389  },
390  "Pr" -> {
391       $\beta$  -> -644.,
392       $\gamma$  -> 1413.,
393      M0 -> 1.88,
394      P2 -> 244.
395  },
396  "Nd" -> {
397      M0 -> 1.85
398  },
399  "Sm" -> {
400       $\alpha$  -> 20.5,
401       $\beta$  -> -616.,
402       $\gamma$  -> 1565.,
403      T2 -> 282.,
404      T3 -> 26.,
405      T4 -> 71.,
406      T6 -> -257.,
407      T7 -> 314.,
408      T8 -> 328.,
409      M0 -> 2.38,
410      P2 -> 336.
411  },
412  "Eu" -> {
413      T2 -> 370.,
414      T3 -> 40.,
415      T4 -> 40.,
416      T6 -> -300.,
417      T7 -> 380.,
418      T8 -> 370.
419  },
420  "Tb" -> {
421      F4 -> 0.709 F2,
422      F6 -> 0.503 F2,
423       $\alpha$  -> 17.6,
424       $\beta$  -> -581.,
425       $\gamma$  -> 1792.,
426      T2 -> 330.,
427      T3 -> 40.,
428      T4 -> 45.,
429      T6 -> -365.,
430      T7 -> 320.,
431      T8 -> 349.,

```

```

432   M0 -> 2.7,
433   P2 -> 482.
434   },
435 "Dy" -> {
436   (* F4 -> 0.707 F2,
437   F6 -> 0.516 F2, *)
438   F2 -> 90421,
439   F4 -> 63928,
440   F6 -> 46657,
441    $\alpha$  -> 17.9,
442    $\beta$  -> -628.,
443    $\gamma$  -> 1790.,
444   T2 -> 326.,
445   T3 -> 23.,
446   T4 -> 83.,
447   T6 -> -294.,
448   T7 -> 403.,
449   T8 -> 340.,
450   M0 -> 4.46,
451   P2 -> 610.,
452   B46 -> -700.
453   },
454 "Ho" -> {
455    $\alpha$  -> 17.2,
456    $\beta$  -> -596.,
457    $\gamma$  -> 1839.,
458   T2 -> 365.,
459   T3 -> 37.,
460   T4 -> 95.,
461   T6 -> -274.,
462   T7 -> 331.,
463   T8 -> 343.,
464   P2 -> 582.
465   },
466 "Er" -> {},
467 "Tm" -> {
468    $\alpha$  -> 17.3,
469    $\beta$  -> -665.,
470    $\gamma$  -> 1936.,
471   M0 -> 4.93,
472   P2 -> 730.,
473   T2 -> 400.
474   },
475 "Yb" -> {
476   B06 -> -23.,
477   B46 -> -512.
478   }
479 |>;
480
481 variedSymbolsLiYF4 = <|
482 "Ce" -> {
483   B02,  $\zeta$ 
484   },
485 "Pr" -> {
486   B02, B04, B06, B44, B46,
487   F2, F4, F6,
488    $\alpha$ ,  $\zeta$ 
489   },
490 "Nd" -> {
491   B02, B04, B06, B44, B46,
492   F2, F4, F6,
493   P2,
494   T2, T3, T4, T6, T7, T8,
495    $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\zeta$ 
496   },
497 "Sm" -> {
498   B02, B04, B06, B44, B46,
499   F2, F4, F6,
500    $\zeta$ 
501   },
502 "Eu" -> {
503   B02, B04, B06, B44, B46,
504   F2, F4, F6,
505   M0, P2,
506    $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\zeta$ 
507   }

```

```

508 "Tb" -> {
509     B02, B04, B06, B44, B46,
510     F2, F4, F6,
511     ζ
512 },
513 "Dy" -> {
514     B02, B04, B06, B44,
515     F2, F4, F6,
516     ζ
517 },
518 "Ho" -> {
519     B02, B04, B06, B44, B46,
520     F2, F4, F6,
521     M0,
522     ζ
523 },
524 "Er" -> {
525     B02, B04, B06, B44, B46,
526     F2, F4, F6,
527     M0, P2,
528     T2, T3, T4, T6, T7, T8,
529     α, β, γ,
530     ζ
531 },
532 "Tm" -> {
533     B02, B04, B06, B44, B46,
534     F2, F4, F6,
535     ζ
536 },
537 "Yb" -> {
538     B02, B04, B44,
539     ζ
540 },
541 |>;
542
543 paramsChengLiYF4::usage="This association has the model parameters as
544 fitted by Cheng et. al \"Crystal-field analyses for trivalent
545 lanthanide ions in LiYF4\".";
546 paramsChengLiYF4 = <|
547     "Ce" -> {
548         ζ -> 630.,
549         B02 -> 354., B04 -> -1043.,
550         B44 -> -1249., B06 -> -65.,
551         B46 -> -1069.
552     },
553     "Pr" -> {
554         F2 -> 68955., F4 -> 50505., F6 -> 33098.,
555         ζ -> 748.,
556         α -> 23.3, β -> -644., γ -> 1413.,
557         M0 -> 1.88, P2 -> 244.,
558         B02 -> 512., B04 -> -1127.,
559         B44 -> -1239., B06 -> -85.,
560         B46 -> -1205.
561     },
562     "Nd" -> {
563         F2 -> 72952., F4 -> 52681., F6 -> 35476.,
564         ζ -> 877.,
565         α -> 21., β -> -579., γ -> 1446.,
566         T2 -> 210., T3 -> 41., T4 -> 74., T6 -> -293., T7 -> 321., T8 ->
567         205.,
568         M0 -> 1.85, P2 -> 304.,
569         B02 -> 391., B04 -> -1031.,
570         B44 -> -1271., B06 -> -28.,
571         B46 -> -1046.
572     },
573     "Sm" -> {
574         F2 -> 79515., F4 -> 56766., F6 -> 40078.,
575         ζ -> 1168.,
576         α -> 20.5, β -> -616., γ -> 1565.,
577         T2 -> 282., T3 -> 26., T4 -> 71., T6 -> -257., T7 -> 314., T8 ->
578         328.,
579         M0 -> 2.38, P2 -> 336.,
580         B02 -> 370., B04 -> -757.,
581         B44 -> -941., B06 -> -67.,
582         B46 -> -895.
583     },

```

```

580 "Eu" -> {
581   F2 -> 82573., F4 -> 59646., F6 -> 43203.,
582    $\zeta$  -> 1329.,
583    $\alpha$  -> 21.6,  $\beta$  -> -482.,  $\gamma$  -> 1140.,
584   T2 -> 370., T3 -> 40., T4 -> 40., T6 -> -300., T7 -> 380., T8 ->
585   370.,
586   M0 -> 2.41, P2 -> 332.,
587   B02 -> 339., B04 -> -733.,
588   B44 -> -1067., B06 -> -36.,
589   B46 -> -764.
590 },
591 "Tb" -> {
592   F2 -> 90972., F4 -> 64499., F6 -> 45759.,
593    $\zeta$  -> 1702.,
594    $\alpha$  -> 17.6,  $\beta$  -> -581.,  $\gamma$  -> 1792.,
595   T2 -> 330., T3 -> 40., T4 -> 45., T6 -> -365., T7 -> 320., T8 ->
596   349.,
597   M0 -> 2.7, P2 -> 482.,
598   B02 -> 413., B04 -> -867.,
599   B44 -> -1114., B06 -> -41.,
600   B46 -> -736.
601 },
602 "Dy" -> {
603   F0 -> 0,
604   F2 -> 90421., F4 -> 63928., F6 -> 46657.,
605    $\zeta$  -> 1895.,
606    $\alpha$  -> 17.9,  $\beta$  -> -628.,  $\gamma$  -> 1790.,
607   T2 -> 326., T3 -> 23., T4 -> 83., T6 -> -294., T7 -> 403., T8 ->
608   340.,
609   M0 -> 4.46, P2 -> 610.,
610   B02 -> 360., B04 -> -737.,
611   B44 -> -943., B06 -> -35.,
612   B46 -> -700.
613 },
614 "Ho" -> {
615   F2 -> 93512., F4 -> 66084., F6 -> 49765.,
616    $\zeta$  -> 2126.,
617    $\alpha$  -> 17.2,  $\beta$  -> -596.,  $\gamma$  -> 1839.,
618   T2 -> 365., T3 -> 37., T4 -> 95., T6 -> -274., T7 -> 331., T8 ->
619   343.,
620   M0 -> 3.92, P2 -> 582.,
621   B02 -> 386., B04 -> -629.,
622   B44 -> -841., B06 -> -33.,
623   B46 -> -687.
624 },
625 "Er" -> {
626   F2 -> 97326., F4 -> 67987., F6 -> 53651.,
627    $\zeta$  -> 2377.,
628    $\alpha$  -> 18.1,  $\beta$  -> -599.,  $\gamma$  -> 1870.,
629   T2 -> 380., T3 -> 41., T4 -> 69., T6 -> -356., T7 -> 239., T8 ->
630   390.,
631   M0 -> 4.41, P2 -> 795.,
632   B02 -> 325., B04 -> -749.,
633   B44 -> -1014., B06 -> -19.,
634   B46 -> -635.
635 },
636 "Tm" -> {
637   F0 -> 0.,
638   T2 -> 0.,
639   F2 -> 101938., F4 -> 71553., F6 -> 51359.,
640    $\zeta$  -> 2632.,
641    $\alpha$  -> 17.3,  $\beta$  -> -665.,  $\gamma$  -> 1936.,
642   M0 -> 4.93, P2 -> 730.,
643   B02 -> 339., B04 -> -627.,
644   B44 -> -913., B06 -> -39.,
645   B46 -> -584.
646 },
647 "Yb" -> {
648    $\zeta$  -> 2916.,
649   B02 -> 446., B04 -> -560.,
650   B44 -> -843., B06 -> -23.,
651   B46 -> -512.
652 }
653 |>;
654 Jiggle::usage = "Jiggle[num, wiggleRoom] takes a number and"

```

```

randomizes it a little by adding or subtracting a random fraction
of itself. The fraction is controlled by wiggleRoom.";
651 Jiggle[num_, wiggleRoom_ : 0.1] := RandomReal[{1 - wiggleRoom, 1 +
652   wiggleRoom}] * num;
653 AddToList::usage = "AddToList[list, element, maxSize, addOnlyNew]
654   prepends the element to list and returns the list. If maxSize is
655   reached, the last element is dropped. If addOnlyNew is True (the
656   default), the element is only added if it is different from the
657   last element.";
658 AddToList[list_, element_, maxSize_, addOnlyNew_ : True] := Module[{

659   tempList = If[
660     addOnlyNew,
661     If[
662       Length[list] == 0,
663       {element},
664       If[
665         element != list[[-1]],
666         Append[list, element],
667         list
668       ],
669       Append[list, element]
670     ],
671     If[Length[tempList] > maxSize,
672       Drop[tempList, Length[tempList] - maxSize],
673       tempList]
674   ];
675 ProgressNotebook::usage="ProgressNotebook[] creates a progress
676   notebook for the solver. This notebook includes a plot of the RMS
677   history and the current parameter values. The notebook is returned
678   . The RMS history and the parameter values are updated by setting
679   the variables rmsHistory and paramSols. The variables
680   stringPartialVars and paramSols are used to display the parameter
681   values in the notebook.";
682 Options[ProgressNotebook] = {
683   "Basic" -> True,
684   "UpdateInterval" -> 0.5};
685 ProgressNotebook[OptionsPattern[]] := (
686   nb = Which[
687     OptionValue["Basic"],
688     CreateDocument[(
689       {
690         Dynamic[
691           TextCell[
692             If[
693               Length[paramSols] > 0,
694               TableForm[
695                 Prepend[
696                   Transpose[{stringPartialVars,
697                     paramSols[[-1]]}],
698                   {"RMS", rmsHistory[[-1]]}]
699                 ],
700                 " "
701               ],
702               ],
703               "Output"
704             ],
705             TrackedSymbols :> {paramSols, stringPartialVars},
706             UpdateInterval -> OptionValue["UpdateInterval"]
707           ]
708         }
709       ),
710       WindowSize -> {600, 1000},
711       WindowSelected -> True,
712       TextAlignment -> Center,
713       WindowTitle -> "Solver Progress"
714     ],
715     True,
716     CreateDocument[(
717       {
718         "",
719         Dynamic[Framed[progressMessage],
720           UpdateInterval -> OptionValue["UpdateInterval"]],
721         Dynamic[
722           GraphicsColumn[

```

```

714     {ListPlot[rmsHistory,
715      PlotMarkers -> "OpenMarkers",
716      Frame -> True,
717      FrameLabel -> {"Iteration", "RMS"},
718      ImageSize -> 800,
719      AspectRatio -> 1/3,
720      FrameStyle -> Directive[Thick, 15],
721      PlotLabel -> If[Length[rmsHistory] != 0, rmsHistory[[-1]],
722      ""]
723      ],
724      ListPlot[(#/#[[1]]) & /@ Transpose[paramSols],
725      Joined -> True,
726      PlotRange -> {All, {-5, 5}},
727      Frame -> True,
728      ImageSize -> 800,
729      AspectRatio -> 1,
730      FrameStyle -> Directive[Thick, 15],
731      FrameLabel -> {"Iteration", "Params"}
732      ]
733      ],
734      TrackedSymbols :> {rmsHistory, paramSols},
735      UpdateInterval -> OptionValue["UpdateInterval"]
736      ],
737      Dynamic[
738      TextCell[
739      If[
740      Length[paramSols] > 0,
741      TableForm[Transpose[{stringPartialVars, paramSols[[-1]]}]],
742      ""]
743      ],
744      "Output"
745      ],
746      TrackedSymbols :> {paramSols, stringPartialVars}
747      ]
748      ]
749      ),
750      WindowSize -> {600, 1000},
751      WindowSelected -> True,
752      TextAlignment -> Center,
753      WindowTitle -> "Solver Progress"
754      ]
755      ];
756      Return[nb];
757      );
758
759 energyCostFunTemplate::usage="energyCostFunTemplate is template used
    to define the cost function for the energy matching. The template
    is used to define a function TheRightEnergyPath that takes a list
    of variables and returns the RMS of the energy differences between
    the computed and the experimental energies. The template requires
    the values to the following keys to be provided: 'vars' and 'varPatterns'";
760 energyCostFunTemplate = StringTemplate["
TheRightEnergyPath['varPatterns']:= (
761 {eigenEnergies, eigenVecs} = Eigensystem[compHam['vars']];
762 ordering = Ordering[eigenEnergies];
763 eigenEnergies = eigenEnergies - Min[eigenEnergies];
764 states = Transpose[{eigenEnergies, eigenVecs}];
765 states = states[[ordering]];
766 coarseStates = ParseStates[states, basis];
767 coarseStates = {#[[1]],#[[-1]]}& /@ coarseStates;
768 (* The eigenvectors need to be simplified in order to compare
    labels to labels *)
769 missingLevels = Length[coarseStates]-Length[expData];
770 (* The energies are in the first element of the tuples. *)
771 energyDiffFun = (Abs[#1[[1]]-#2[[1]]])&;
772 (* match disregarding labels *)
773 energyFlow = FlowMatching[coarseStates,
774 expData,
775     \"notMatched\" -> missingLevels,
776     \"CostFun\" -> energyDiffFun
777     ];
778 energyPairs = {#[[1]][[1]], #[[2]][[1]]} & /@ energyFlow[[1]];
779 energyRms = Sqrt[Total[(Abs[#[[2]]-#[[1]]])^2 & /@ energyPairs]
780 / Length[energyPairs]];

```

```

781     Return[energyRms];
782   )];
783 
784 AppendToLog[message_, file_String] := Module[
785   {timestamp = DateString["ISODateTime"], msgString},
786   (
787     msgString = ToString[message, InputForm]; (* Convert any
788       expression to a string *)
789     OpenAppend[file];
790     WriteString[file, timestamp, " - ", msgString, "\n"];
791     Close[file];
792   )
793 ];
794 
795 energyAndLabelCostFunTemplate::usage="energyAndLabelCostFunTemplate
796 is a template used to define the cost function that includes both
797 the differences between energies and the differences between
798 labels. The template is used to define a function
799 TheRightSignedPath that takes a list of variables and returns the
800 RMS of the energy differences between the computed and the
801 experimental energies together with a term that depends on the
802 differences between the labels. The template requires the values
803 to the following keys to be provided: 'vars' and 'varPatterns';
804 
805 energyAndLabelCostFunTemplate = StringTemplate["
806 TheRightSignedPath['varPatterns'] := Module[
807   {energyRms, eigenEnergies, eigenVecs, ordering, states,
808    coarseStates, missingLevels, energyDiffFun, energyFlow,
809    energyPairs, energyAndLabelFun, energyAndLabelFlow, totalAvgCost},
810   (
811     {eigenEnergies, eigenVecs} = Eigensystem[compHam['vars']];
812     ordering = Ordering[eigenEnergies];
813     eigenEnergies = eigenEnergies - Min[eigenEnergies];
814     states = Transpose[{eigenEnergies, eigenVecs}];
815     states = states[[ordering]];
816     coarseStates = ParseStates[states, basis];
817 
818     (* The eigenvectors need to be simplified in order to compare
819       labels to labels *)
820     coarseStates = {#[[1]], #[[-1]]} & /@ coarseStates;
821     missingLevels = Length[coarseStates] - Length[expData];
822 
823     (* The energies are in the first element of the tuples. *)
824     energyDiffFun = ( Abs[#1[[1]] - #2[[1]]] ) &;
825 
826     (* matching disregarding labels to get overall scale for scaling
827       differences in labels *)
828     energyFlow = FlowMatching[coarseStates,
829                   expData,
830                   \"notMatched\" -> missingLevels,
831                   \"CostFun\" -> energyDiffFun
832                   ];
833     energyPairs = {#[[1]][[1]], #[[2]][[1]]} &/@energyFlow[[1]];
834     energyRms = Sqrt[Total[(Abs[#[[2]] - #[[1]]])^2 & /@
835     energyPairs]/Length[energyPairs]];
836 
837     (* matching using both labels and energies *)
838     energyAndLabelFun = With[{del=energyRms},
839       (Abs[#1[[1]] - #2[[1]]] +
840        If[#1[[2]] == #2[[2]],
841            0.,
842            del]) &];
843 
844     (* energyAndLabelFun = With[{del=energyRms},
845       (Abs[#1[[1]] - #2[[1]]] +
846        del*EditDistance[#1[[2]], #2[[2]]]) &]; *)
847     energyAndLabelFun = ( Abs[#1[[1]] - #2[[1]]] + EditDistance
848       #[[2]], #2[[1]] ) &;
849     energyAndLabelFlow = FlowMatching[coarseStates,
850                   expData,
851                   \"notMatched\" -> missingLevels,
852                   \"CostFun\" -> energyAndLabelFun
853                   ];
854 
855     totalAvgCost = Total[energyAndLabelFun @@ # & /@
856     energyAndLabelFlow[[1]]]/Length[energyAndLabelFlow[[1]]];
857     Return[totalAvgCost];
858   )
859 ];

```

```

841 ] "];
842
843 Constrained::usage = "Constrained[problemVars_, ln] returns a list of
844 constraints for the variables in problemVars for trivalent
845 lanthanide ion ln. problemVars are standard model symbols (F2, F4,
846 ...). The ranges returned are based in the fitted parameters for
847 LaF3 as found in Carnall et al. They could probably be more fine
848 grained, but these ranges are seen to describe all the ions in
849 that case.";
850
851 Constrained[problemVars_, ln_] := (
852   slater = Which[
853     MemberQ[{"Ce", "Yb"}, ln],
854     {},
855     True,
856     {#, (20000. < # < 120000.)} & /@ {F2, F4, F6}
857   ];
858   alpha = Which[
859     MemberQ[{"Ce", "Yb"}, ln],
860     {},
861     True,
862     {{\alpha, 14. < \alpha < 22.}}
863   ];
864   zeta = {{\zeta, 500. < \zeta < 3200.}};
865   beta = Which[
866     MemberQ[{"Ce", "Yb"}, ln],
867     {},
868     True,
869     {{\beta, -1000. < \beta < -400.}}
870   ];
871   gamma = Which[
872     MemberQ[{"Ce", "Yb"}, ln],
873     {},
874     True,
875     {{\gamma, 1000. < \gamma < 2000.}}
876   ];
877   tees = Which[
878     ln == "Tm",
879     {100. < T2 < 500.},
880     MemberQ[{"Ce", "Pr", "Yb"}, ln],
881     {},
882     True,
883     {#, -500. < # < 500.} & /@ {T2, T3, T4, T6, T7, T8}];
884   marvins = Which[
885     MemberQ[{"Ce", "Yb"}, ln],
886     {},
887     True,
888     {{M0, 1.0 < M0 < 5.0}}
889   ];
890   crystalRanges = {#, (-2000. < # < 2000.)} & /@ (Intersection[
891     cfSymbols, problemVars]);
892   allCons =
893     Join[slater, zeta, alpha, beta, gamma, tees, marvins, peas,
894     crystalRanges];
895   allCons = Select[allCons, MemberQ[problemVars, #[[1]]] &];
896   Return[Flatten[Rest /@ allCons]]
897 );
898
899 Options[LogSol] = {"PrintFun" -> PrintTemporary};
900 LogSol::usage = "LogSol[expr, prefix] saves the given expression to a
901 file. The file is named with the given prefix and a created UUID.
902 The file is saved in the \"log\" directory under the current
903 directory. The file is saved in the format of a .m file. The
904 function returns the name of the file.";
905 LogSol[theSolution_, prefix_, OptionsPattern[]] :=
906   PrintFun = OptionValue["PrintFun"];
907   fname = prefix <> "-sols-" <> CreateUUID[] <> ".m";
908   fname = FileNameJoin[{".", "log", fname}];
909   PrintFun["Saving solution to: ", fname];
910   Export[fname, theSolution];
911   Return[fname];

```

```

907 );
908
909 ClassicalFit::usage="ClassicalFit[numE, expData, excludeDataIndices,
910   problemVars, startValues, constraints] fits the given expData in
911   an f^numE configuration, by using the symbols in problemVars. The
912   symbols given in problemVars may be constrained or held constant,
913   this being controlled by constraints list which is a list of
914   replacement rules expressing desired constraints. The constraints
915   list additional constraints imposed upon the model parameters that
916   remain once other simplifications have been \"baked\" into the
917   compiled Hamiltonians that are used to increase the speed of the
918   calculation.
919
920 Important, note that in the case of odd number of electrons the given
921   data must explicitly include the Kramers degeneracy;
922   excludeDataIndices must be compatible with this.
923
924 The list expData needs to be a list of lists with the only
925   restriction that the first element of them corresponds to energies
926   of levels. In this list, an empty value can be used to indicate
927   known gaps in the data. Even if the energy value for a level is
928   known (and given in expData) certain values can be omitted from
929   the fitting procedure through the list excludeDataIndices, which
930   correspond to indices in expData that should be skipped over.
931
932 The Hamiltonian used for fitting is a version that has been truncated
933   either by using the maximum energy given in expData or by
934   manually setting a truncation energy using the option \"TruncationEnergy\".
935
936 The option \"Experimental Uncertainty in K\" is the estimated
937   uncertainty in the differences between the calculated and the
938   experimental energy levels. This is used to estimate the
939   uncertainty in the fitted parameters. Admittedly this will be a
940   rough estimate (at least on the contribution of the calculated
941   uncertainty), but it is better than nothing and may at least
942   provide a lower bound to the uncertainty in the fitted parameters.
943   It is assumed that the uncertainty in the differences between the
944   calculated and the experimental energy levels is the same for all
945   of them.
946
947 The list startValues is a list with all of the parameters needed to
948   define the Hamiltonian (including the initial values for
949   problemVars).
950
951 The function saves the solution to a file. The file is named with a
952   prefix (controlled by the option \"FilePrefix\") and a UUID. The
953   file is saved in the log sub-directory as a .m file.
954
955 Here's a description of the different parts of this function: first
956   the Hamiltonian is assembled and simplified using the given
957   simplifications. Then the intermediate coupling basis is
958   calculated using the free-ion parameters for the given lanthanide.
959   The Hamiltonian is then changed to the intermediate coupling
960   basis and truncated. The truncated Hamiltonian is then compiled
961   into a function that can be used to calculate the energy levels of
962   the truncated Hamiltonian. The function that calculates the
963   energy levels is then used to fit the experimental data. The
964   fitting is done using FindMinimum with the Levenberg-Marquardt
965   method.
966
967 The function returns an association with the following keys:
968
969 - \"bestRMS\" which is the best \[Sigma] value found.
970 - \"bestParams\" which is the best set of parameters found for the
971   variables that were not constrained.
972 - \"bestParamsWithConstraints\" which has the best set of parameters
973   (from - \"bestParams\") together with the used constraints. These
974   include all the parameters in the model, even those that were not
975   fitted for.
976 - \"paramSols\" which is a list of the parameters trajectories during
977   the stepping of the fitting algorithm.
978 - \"timeTaken/s\" which is the time taken to find the best fit.
979 - \"simplifier\" which is the simplifier used to simplify the
980   Hamiltonian.
981 - \"excludeDataIndices\" as given in the input.

```

```

934 - \"startValues\" as given in the input.
935
936 - \"freeIonSymbols\" which are the symbols used in the intermediate
937   coupling basis.
938 - \"truncationEnergy\" which is the energy used to truncate the
939   Hamiltonian, if it was set to Automatic, the value here is the
940   actual energy used.
941 - \"numE\" which is the number of electrons in the f^numE
942   configuration.
943 - \"expData\" which is the experimental data used for fitting.
944 - \"problemVars\" which are the symbols considered for fitting
945
946 - \"maxIterations\" which is the maximum number of iterations used by
947   NMinimize.
948 - \"hamDim\" which is the dimension of the full Hamiltonian.
949 - \"allVars\" which are all the symbols defining the Hamiltonian
950   under the aggregate simplifications.
951 - \"freeBies\" which are the free-ion parameters used to define the
952   intermediate coupling basis.
953 - \"truncatedDim\" which is the dimension of the truncated
954   Hamiltonian.
955 - \"compiledIntermediateFname\" the file name of the compiled
956   function used for the truncated Hamiltonian.
957
958 - \"fittedLevels\" which is the number of levels fitted for.
959 - \"actualSteps\" the number of steps that FindMinimum actually
960   took.
961 - \"solWithUncertainty\" which is a list of replacement rules of the
962   form (paramSymbol -> {bestEstimate, uncertainty}).
963 - \"rmsHistory\" which is a list of the \[Sigma] values found during
964   the fitting.
965 - \"Appendix\" which is an association appended to the log file under
966   the key \"Appendix\".
967 - \"presentDataIndices\" which is the list of indices in expData that
968   were used for fitting, this takes into account both the empty
969   indices in expData and also the indices in excludeDataIndices.
970
971 - \"states\" which contains a list of eigenvalues and eigenvectors
972   for the fitted model, this is only available if the option \
973   \"SaveEigenvectors\" is set to True; if a general shift of energy
974   was allowed for in the fitting, then the energies are shifted
975   accordingly.
976 - \"energies\" which is a list of the energies of the fitted levels,
977   this is only available if the option \"SaveEigenvectors\" is set
978   to False. If a general shift of energy was allowed for in the
979   fitting, then the energies are shifted accordingly.
980 - \"degreesOfFreedom\" which is equal to the number of fitted state
981   energies minus the number of parameters used in fitting.
982
983 The function admits the following options with corresponding default
984   values:
985 - \"MaxHistory\" : determines how long the logs for the solver can be
986   .
987 - \"MaxIterations\" : determines the maximum number of iterations used
988   by NMinimize.
989 - \"FilePrefix\" : the prefix to use for the subfolder in the log
990   folder, in which the solution files are saved, by default this is
991   \"calcs\" so that the calculation files are saved under the
992   directory \"log/calcs\".
993 - \"AddConstantShift\" : if True then a constant shift is allowed in
994   the fitting, default is False. If this is the case the variable \
995   \[Epsilon] is added to the list of variables to be fitted for,
996   it must not be included in problemVars.
997
998 - \"AccuracyGoal\" : the accuracy goal used by NMinimize, default of
999   5.
1000 - \"TruncationEnergy\" : if Automatic then the maximum energy in
1001   expData is taken, else it takes the value set by this option. In
1002   all cases the energies in expData are only considered up to this
1003   value.
1004 - \"PrintFun\" : the function used to print progress messages, the
1005   default is PrintTemporary.
1006 - \"RefParamsVintage\" : the vintage of the reference parameters to
1007   use. The reference parameters are both used to determine the
1008   truncated Hamiltonian, and also as starting values for the solver.
1009   It may be \"LaF3\", in which case reference parameters from

```

Carnall are used. It may also be `\"LiYF4\"`, in which case the reference parameters from the LiYF4 paper are used. It may also be `Automatic`, in which case the given experimental data is used to determine starting values for  $F^k$  and  $\zeta$ . It may also be a list or association that provides values for the Slater integrals and spin-orbit coupling, the remaining necessary parameters complemented by using `\"LaF3\"`.

```

970
971 - \"ProgressView\": whether or not a progress window will be opened
972   to show the progress of the solver, the default is True.
973 - \"SignatureCheck\": if True then the function returns
974   prematurely, returning a list with the symbols that would have
975   defined the Hamiltonian after all simplifications have been
976   applied. Useful to check the entire parameter set that the
977   Hamiltonian has, which has to match one-to-one what is provided by
978   startValues.
979 - \"SaveEigenvectors\": if True then both the eigenvectors and
980   eigenvalues are saved under the \"states\" key of the returned
981   association. If False then only the energies are saved, the
982   default is False.
983
984 - \"AppendToFile\": an association appended to the log file under
985   the key \"Appendix\".
986 - \"MagneticSimplifier\": a list of replacement rules to simplify the
987   Marvin and pseudo-magnetic parameters. Here the ratios of the
988   Marvin parameters and the pseudo-magnetic parameters are defined
989   to simplify the magnetic part of the Hamiltonian.
990 - \"MagFieldSimplifier\": a list of replacement rules to specify a
991   magnetic field (in T), if set to {}, then {Bx, By, Bz} can also be
992   used as variables to be fitted for.
993
994 - \"SymmetrySimplifier\": a list of replacement rules to simplify
995   the crystal field.
996 - \"OtherSimplifier\": an additional list of replacement rules that
997   are applied to the Hamiltonian before computing with it. Here the
998   spin-spin contribution can be turned off by setting \[Sigma]SS->0,
999   which is the default.
1000 ";
1001 Options[ClassicalFit] = {
1002   "MaxHistory"      -> 200,
1003   "MaxIterations"   -> 100,
1004   "FilePrefix"      -> "calcs",
1005   "ProgressView"    -> True,
1006   "TruncationEnergy" -> Automatic,
1007   "AccuracyGoal"    -> 5,
1008   "PrintFun"        -> PrintTemporary,
1009   "RefParamsVintage" -> "LaF3",
1010   "SignatureCheck"  -> False,
1011   "AddConstantShift" -> False,
1012   "SaveEigenvectors" -> False,
1013   "AppendToFile"    -> <||>,
1014   "SaveToLog"        -> False,
1015   "Energy Uncertainty in K" -> Automatic,
1016   "MagneticSimplifier" -> {
1017     M2 -> 56/100 MO,
1018     M4 -> 31/100 MO,
1019     P4 -> 1/2 P2,
1020     P6 -> 1/10 P2
1021   },
1022   "MagFieldSimplifier" -> {
1023     Bx -> 0,
1024     By -> 0,
1025     Bz -> 0
1026   },
1027   "SymmetrySimplifier" -> {
1028     B12->0, B14->0, B16->0, B34->0, B36->0, B56->0,
1029     S12->0, S14->0, S16->0, S22->0, S24->0, S26->0,
1030     S34->0, S36->0, S44->0, S46->0, S56->0, S66->0
1031   },
1032   "OtherSimplifier" -> {
1033     F0->0,
1034     P0->0,
1035     \[Sigma]SS->0,
1036     T11p->0, T12->0, T14->0, T15->0,
1037     T16->0, T18->0, T17->0, T19->0, T2p->0,
1038     wChErrA ->0, wChErrB->0
1039 }
```

```

1020 },
1021 "ThreeBodySimplifier" -> <|
1022   1 -> {
1023     T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
1024     T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1025     ->0,
1026     T2p->0},
1027   2 -> {
1028     T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
1029     T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1030     ->0,
1031     T2p->0
1032   },
1033   3 -> {},
1034   4 -> {},
1035   5 -> {},
1036   6 -> {},
1037   7 -> {},
1038   8 -> {},
1039   9 -> {},
1040   10 -> {},
1041   11 -> {},
1042   12 -> {
1043     T3->0, T4->0, T6->0, T7->0, T8->0,
1044     T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1045     ->0,
1046     T2p->0
1047   },
1048   13->{
1049     T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
1050     T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1051     ->0,
1052     T2p->0
1053   }
1054 };
1055 ClassicalFit[numE_Integer, expData_List, excludeDataIndices_List,
1056   problemVars_List, startValues_Association, constraints_List,
1057   OptionsPattern[]]:=Module[
1058   {
1059     accuracyGoal,
1060     allFreeEnergies,
1061     allFreeEnergiesSorted, allVars, allVarsVec,
1062     argsForEvalInsideOfTheIntermediateSystems,
1063     argsOfTheIntermediateEigensystems, aVar, aVarPosition,
1064     basis, basisChanger, basisChangerBlocks,
1065     bestParams, bestRMS, blockShifts, blockSizes,
1066     compiledDiagonal, compiledIntermediateFname,
1067     constrainedProblemVars, constrainedProblemVarsList,
1068     currentRMS, degressOfFreedom, dependentVars,
1069     diagonalBlocks, diagonalScalarBlocks, diff,
1070     eigenEnergies, eigenvalueDispenserTemplate,
1071     eigenVectors, elevatedIntermediateEigensystems,
1072     endTime, fmSolAssoc, freeBies,
1073     freeIenergiesAndMultiplets, fullHam, fullSolve,
1074     ham, hamDim, hamEigenvaluesTemplate,
1075     hamString, indepSolveVec, indepVars, intermediateHam,
1076     isolationValues, lin, linMat, ln, lnParams,
1077     logFilePrefix, magneticSimplifier,
1078     maxFreeEnergy, maxHistory, maxIterations,
1079     minFreeEnergy, minpoly,
1080     modelSymbols, multipletAssignments, needlePosition,
1081     numBlocks, solCompendium,
1082     openNotebooks, ordering, otherSimplifier, p0,
1083     paramBest, perHam,
1084     presentDataIndices, PrintFun, problemVarsPositions,
1085     problemVarsQ, problemVarsQString, problemVarsVec,
1086     problemVarsWithStartValues, reducedModelSymbols,
1087     roundedTruncationEnergy,
1088     runningInteractive, shiftToggle, simplifier,
1089     sol, solWithUncertainty,
1090     sortedTruncationIndex, sqdiff, standardValues,
1091     startTime,
1092     states, steps, symmetrySimplifier,
1093     theIntermediateEigensystems, TheIntermediateEigensystems,
1094   }

```

```

1090 TheTruncatedAndSignedPathGenerator, timeTaken,
1091 truncatedIntermediateBasis, truncatedIntermediateHam,
1092 truncationEnergy, truncationIndices, RefParams,
1093 truncationUmbral, varHash,
1094 varsWithConstants, \[Lambda]0Vec,
1095 \[Lambda]exp
1096 },
1097 (
1098 \[Sigma]exp = OptionValue["Energy Uncertainty in K"];
1099 solCompendium = <||>;
1100 refParamsVintage = OptionValue["RefParamsVintage"];
1101 RefParams = Which[
1102   refParamsVintage === "LaF3",
1103   LoadLaF3Parameters,
1104   refParamsVintage === "LiYF4",
1105   LoadLiYF4Parameters,
1106   True,
1107   refParamsVintage
1108 ];
1109 hamDim = Binomial[14, numE];
1110 addShift = OptionValue["AddConstantShift"];
1111 ln = theLanthanides[[numE]];
1112 maxHistory = OptionValue["MaxHistory"];
1113 maxIterations = OptionValue["MaxIterations"];
1114 logFilePrefix = If[OptionValue["FilePrefix"] == "",
1115   ToString[theLanthanides[[numE]]],
1116   OptionValue["FilePrefix"]
1117 ];
1118 accuracyGoal = OptionValue["AccuracyGoal"];
1119 PrintFun = OptionValue["PrintFun"];
1120 freeIonSymbols = OptionValue["FreeIonSymbols"];
1121 runningInteractive = (Head[$ParentLink] === LinkObject);
1122 magneticSimplifier = OptionValue["MagneticSimplifier"];
1123 magFieldSimplifier = OptionValue["MagFieldSimplifier"];
1124 symmetrySimplifier = OptionValue["SymmetrySimplifier"];
1125 otherSimplifier = OptionValue["OtherSimplifier"];
1126 threeBodySimplifier = If[Head[OptionValue["ThreeBodySimplifier"]]
1127 == Association,
1128   OptionValue["ThreeBodySimplifier"][numE],
1129   OptionValue["ThreeBodySimplifier"]
1130 ];
1131 truncationEnergy = If[OptionValue["TruncationEnergy"] ===
1132 Automatic,
1133 (
1134   PrintFun["Truncation energy set to Automatic, using the
1135 maximum energy (+20%) in the data ..."];
1136   Round[1.2 * Max[Select[First /@ expData, NumericQ[#] &]]
1137 ],
1138   OptionValue["TruncationEnergy"]
1139 ];
1140 truncationEnergy = Max[50000, truncationEnergy];
1141 PrintFun["Using a truncation energy of ", truncationEnergy, " K"
1142 ];
1143 simplifier = Join[magneticSimplifier,
1144   magFieldSimplifier,
1145   symmetrySimplifier,
1146   threeBodySimplifier,
1147   otherSimplifier];
1148 PrintFun["Determining gaps in the data ..."];
1149 (* whatever is non-numeric is assumed as a known gap *)
1150 presentDataIndices = Flatten[Position[expData, {_?(NumericQ[#] &
1151 , ___}]];
1152 (* some indices omitted here based on the excludeDataIndices
1153 argument *)
1154 presentDataIndices = Complement[presentDataIndices,
1155 excludeDataIndices];
1156
1157 solCompendium["simplifier"] = simplifier;
1158 solCompendium["excludeDataIndices"] = excludeDataIndices;
1159 solCompendium["startValues"] = startValues;
1160 solCompendium["freeIonSymbols"] = freeIonSymbols;
1161 solCompendium["truncationEnergy"] = truncationEnergy;
1162 solCompendium["numE"] = numE;

```

```

1159 solCompendium["expData"] = expData;
1160 solCompendium["problemVars"] = problemVars;
1161 solCompendium["maxIterations"] = maxIterations;
1162 solCompendium["hamDim"] = hamDim;
1163 solCompendium["constraints"] = constraints;
1164
1165 modelSymbols = Sort[Select[paramSymbols, Not[MemberQ[Join[
racahSymbols, juddOfeltIntensitySymbols, chenSymbols, {t2Switch, \
Epsilon}], gs, nE}], #]]&]];
(* remove the symbols that will be removed by the simplifier, no
symbol should remain here that is not in the symbolic Hamiltonian *)
1166
1167 reducedModelSymbols = Select[modelSymbols, Not[MemberQ[Keys[
simplifier], #]]&];
1168
1169 (* this is useful to understand what are the arguments of the
truncated compiled Hamiltonian *)
1170 If[OptionValue["SignatureCheck"],
(
1171 PrintFun["Given the model parameters and the simplifying
assumptions, the resultant model parameters are:"];
PrintFun[{reducedModelSymbols}];
PrintFun["Exiting ..."];
Return[];
)
];
1178
1179 (* calculate the basis *)
1180 PrintFun["Retrieving the LSJMJ basis for f^", numE, " ..."];
1181 basis = BasisLSJMJ[numE];
1182
1183 Which[refParamsVintage === Automatic,
(
1184 PrintFun["Using the automatic vintage with freshly fitted
free-ion parameters and others as in LaF3 ..."];
1185 lnParams = LoadLaF3Parameters[ln];
1186 freeIonSol = FreeIonSolver[expData, numE];
1187 freeIonParams = freeIonSol["bestParams"];
1188 lnParams = Join[lnParams, freeIonParams];
),
MemberQ[{List, Association}, Head[RefParams]],
(
1193 RefParams = Association[RefParams];
PrintFun["Using the given parameters as a starting point ..."]
];
1195 lnParams = RefParams;
1196 extraParams = LoadLaF3Parameters[ln];
1197 lnParams = Join[extraParams, lnParams];
),
True,
(
1201 (* get the reference parameters from the given vintage *)
1202 PrintFun["Getting reference free-ion parameters for ", ln, "
using ", refParamsVintage, " ..."];
1203 lnParams = ParamPad[RefParams[ln], "PrintFun" -> PrintFun];
)
];
1206 freeBies = Prepend[Values[(#->(#/lnParams)) &/@ freeIonSymbols], 
numE];
(* a more explicit alias *)
1208 allVars = reducedModelSymbols;
1209 numericConstraints = Association@Select[constraints, NumericQ
[#[[2]]] &];
1210 standardValues = allVars /. Join[lnParams, numericConstraints];
1211 solCompendium["allVars"] = allVars;
solCompendium["freeBies"] = freeBies;
1213
1214 (* reload compiled version if found *)
1215 varHash = Hash[{numE, allVars, freeBies,
truncationEnergy, simplifier}];
1216 compiledIntermediateFname = ln <> "-compiled-intermediate-
truncated-ham-" <> ToString[varHash] <> ".mx";
1217 compiledIntermediateFname = FileNameJoin[{moduleDir, "compiled",
compiledIntermediateFname}];
solCompendium["compiledIntermediateFname"] =

```

```

1219 compiledIntermediateFname;
1220
1221 If[FileExistsQ[compiledIntermediateFname],
1222   PrintFun["This ion, free-ion params, and full set of variables
1223 have been used before (as determined by {numE, allVars, freeBies,
1224 truncationEnergy, simplifier}). Loading the previously saved
1225 compiled function and intermediate coupling basis ..."];
1226   PrintFun["Using : ", compiledIntermediateFname];
1227   {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =
1228   Import[compiledIntermediateFname];
1229 (
1230   If[truncationEnergy == Infinity,
1231   (
1232     ham = EffectiveHamiltonian[numE, "ReturnInBlocks" -> False
1233   ];
1234   theSimplifier = simplifier;
1235   ham = Normal@ReplaceInSparseArray[ham, simplifier];
1236   PrintFun["Compiling a function for the Hamiltonian with no
1237 truncation ..."];
1238   (* compile a function that will calculate the truncated
1239   Hamiltonian given the parameters in allVars, this is the function
1240   to be use in fitting *)
1241   compileIntermediateTruncatedHam = Compile[Evaluate[allVars
1242 ], Evaluate[ham]];
1243   truncatedIntermediateBasis = SparseArray@IdentityMatrix[
1244 Binomial[14, numE]];
1245   (* save the compiled function *)
1246   PrintFun["Saving the compiled function for the Hamiltonian
1247 with no truncation and a placeholder intermediate basis ..."];
1248   Export[compiledIntermediateFname, {
1249     compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
1250   ),
1251   (
1252     (* grab the Hamiltonian preserving the block structure *)
1253     PrintFun["Assembling the Hamiltonian for f^", numE, " keeping
1254 the block structure ..."];
1255     ham = EffectiveHamiltonian[numE, "ReturnInBlocks" ->
1256 True];
1257     (* apply the simplifier *)
1258     PrintFun["Simplifying using the aggregate set of
1259 simplification rules ..."];
1260     ham = Map[ReplaceInSparseArray[#, simplifier]&,amp;, ham,
1261 {2}];
1262     PrintFun["Zeroing out every symbol in the Hamiltonian that is
1263 not a free-ion parameter ..."];
1264     (* Get the free ion symbols *)
1265     freeIonSimplifier = (#->0) & /@ Complement[
1266 reducedModelSymbols, freeIonSymbols];
1267     (* Take the diagonal blocks for the intermediate analysis *)
1268     PrintFun["Grabbing the diagonal blocks of the Hamiltonian ...
1269 "];
1270     diagonalBlocks = Diagonal[ham];
1271     (* simplify them to only keep the free ion symbols *)
1272     PrintFun["Simplifying the diagonal blocks to only keep the
1273 free ion symbols ..."];
1274     diagonalScalarBlocks = ReplaceInSparseArray[#,amp;
1275 freeIonSimplifier]&/@diagonalBlocks;
1276     (* these include the MJ quantum numbers, remove that *)
1277     PrintFun["Contracting the basis vectors by removing the MJ
1278 quantum numbers from the diagonal blocks ..."];
1279     diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
1280
1281     argsOfTheIntermediateEigensystems = StringJoin[Riffle
1282 [Prepend[(ToString[#]<>"v_") & /@ freeIonSymbols, "numE_"], ", ", "]];
1283     argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle
1284 [(ToString[#]<>"v") & /@ freeIonSymbols, ", "]];
1285     PrintFun["argsOfTheIntermediateEigensystems = ",
1286 argsOfTheIntermediateEigensystems];
1287     PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
1288 argsForEvalInsideOfTheIntermediateSystems];
1289     PrintFun["(if the following fails, it might help to see if
1290 the arguments of TheIntermediateEigensystems match the ones shown
1291 above)"];
1292
1293     (* compile a function that will effectively calculate the
1294 spectrum of all of the scalar blocks given the parameters of the

```

```

1265 free-ion part of the Hamiltonian *)
1266 (* compile one function for each of the blocks *)
1267 PrintFun["Compiling functions for the diagonal blocks of the
1268 Hamiltonian ..."];
1269 compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate
1270 [N[Normal[#]]]]&/@diagonalScalarBlocks;
1271 (* use that to create a function that will calculate the free
1272 -ion eigensystem *)
1273 TheIntermediateEigensystems[numEv_, F0v_, F2v_, F4v_, F6v_, ζ
1274 v_] := (
1275   theNumericBlocks = (#[F0v, F2v, F4v, F6v, ζv]&) /@ compiledDiagonal;
1276   theIntermediateEigensystems = Eigensystem /@
1277 theNumericBlocks;
1278 Js = AllowedJ[numEv];
1279 basisJ = BasisLSJMJ[numEv, "AsAssociation" -> True];
1280 (* having calculated the eigensystems with the removed
1281 degeneracies, put the degeneracies back in explicitly *)
1282 elevatedIntermediateEigensystems = MapIndexed[EigenLever
1283 [#1, 2Js[[#2[[1]]]]+1 ]&, theIntermediateEigensystems];
1284 (* Identify a single MJ to keep *)
1285 pivot = If[EvenQ[numEv], 0, -1/2];
1286 LSJmultiplets = (#[[1]] <> ToString[InputForm[#[[2]]]])&/
1287 @Select[BasisLSJMJ[numEv], #[[{-1}] == pivot &];
1288 (* calculate the multiplet assignments that the
1289 intermediate basis eigenvectors have *)
1290 needlePosition = 0;
1291 multipletAssignments = Table[
1292 (
1293   J = Js[[idx]];
1294   eigenVecs = theIntermediateEigensystems[[idx]][[2]];
1295   majorComponentIndices = Ordering[Abs[#][[-1]]]&/
1296 @eigenVecs;
1297   majorComponentIndices += needlePosition;
1298   needlePosition += Length[
1299     majorComponentIndices];
1300   majorComponentAssignments = LSJmultiplets[[#]]&/
1301 @majorComponentIndices;
1302   (* All of the degenerate eigenvectors belong to the
1303 same multiplet*)
1304   elevatedMultipletAssignments = ListRepeater[
1305     majorComponentAssignments, 2J+1];
1306   elevatedMultipletAssignments
1307   ),
1308   {idx, 1, Length[Js]}
1309 ];
1310 (* put together the multiplet assignments and the energies
1311 *)
1312 freeIenergiesAndMultiplets = Transpose /@ Transpose[{First/
1313 @elevatedIntermediateEigensystems, multipletAssignments}];
1314 freeIenergiesAndMultiplets = Flatten[
1315 freeIenergiesAndMultiplets, 1];
1316 (* calculate the change of basis matrix using the
1317 intermediate coupling eigenvectors *)
1318 basisChanger = BlockDiagonalMatrix[Transpose /@ Last /
1319 @elevatedIntermediateEigensystems];
1320 basisChanger = SparseArray[basisChanger];
1321 Return[{theIntermediateEigensystems, multipletAssignments,
1322 elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
1323 basisChanger}]
1324 );
1325
1326 PrintFun["Calculating the intermediate eigensystems for ",ln,
1327 " using free-ion params from LaF3 ..."];
1328 (* calculate intermediate coupling basis using the free-ion
1329 params for LaF3 *)
1330 {theIntermediateEigensystems, multipletAssignments,
1331 elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
1332 basisChanger} = TheIntermediateEigensystems@@freeBies;
1333
1334 (* use that intermediate coupling basis to compile a function
1335 for the full Hamiltonian *)
1336 allFreeEnergies = Flatten[First /
1337 @elevatedIntermediateEigensystems];
1338 (* important that the intermediate coupling basis have
1339 attached energies, which make possible the truncation *)

```

```

1311     ordering = Ordering[allFreeEnergies];
1312     (* sort the free ion energies and determine which indices
1313      should be included in the truncation *)
1314     allFreeEnergiesSorted          = Sort[allFreeEnergies];
1315     {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];
1316     (* determine the index at which the energy is equal or larger
1317      than the truncation energy *)
1318     sortedTruncationIndex = Which[
1319       truncationEnergy > (maxFreeEnergy - minFreeEnergy),
1320       hamDim,
1321       True,
1322       FirstPosition[allFreeEnergiesSorted - Min[
1323         allFreeEnergiesSorted], x_ /; x > truncationEnergy, {0}, 1][[1]]
1324     ];
1325     (* the actual energy at which the truncation is made *)
1326     roundedTruncationEnergy = allFreeEnergiesSorted[[sortedTruncationIndex]];
1327
1328     (* the indices that participate in the truncation *)
1329     truncationIndices = ordering[[;; sortedTruncationIndex]];
1330     PrintFun["Computing the block structure of the change of
1331 basis array ..."];
1332     blockSizes           = BlockArrayDimensionsArray[ham];
1333     basisChangerBlocks = ArrayBlocker[basisChanger, blockSizes];
1334     blockShifts         = First /@ Diagonal[blockSizes];
1335     numBlocks           = Length[blockSizes];
1336     (* using the ham (with all the symbols) change the basis to
1337     the computed one *)
1338     PrintFun["Changing the basis of the Hamiltonian to the
1339 intermediate coupling basis ..."];
1340     intermediateHam = BlockMatrixMultiply[ham, basisChangerBlocks];
1341
1342     PrintFun["Distributing products inside of symbolic matrix
1343 elements to keep complexity in check ..."];
1344     Do[
1345       intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
1346         intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
1347         {rowIdx, 1, numBlocks},
1348         {colIdx, 1, numBlocks}
1349     ];
1350     intermediateHam = BlockMatrixMultiply[BlockTranspose[
1351       basisChangerBlocks], intermediateHam];
1352     PrintFun["Distributing products inside of symbolic matrix
1353 elements to keep complexity in check ..."];
1354     Do[
1355       intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
1356         intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
1357         {rowIdx, 1, numBlocks},
1358         {colIdx, 1, numBlocks}
1359     ];
1360     (* using the truncation indices truncate that one *)
1361     PrintFun["Truncating the Hamiltonian ..."];
1362     truncatedIntermediateHam = TruncateBlockArray[intermediateHam,
1363     truncationIndices, blockShifts];
1364     (* these are the basis vectors for the truncated hamiltonian
1365    *)
1366     PrintFun["Saving the truncated intermediate basis ..."];
1367     truncatedIntermediateBasis = basisChanger[[All,
1368     truncationIndices]];
1369
1370     PrintFun["Compiling a function for the truncated Hamiltonian
1371    ..."];
1372     (* compile a function that will calculate the truncated
1373     Hamiltonian given the parameters in allVars, this is the function
1374     to be use in fitting *)
1375     compileIntermediateTruncatedHam = Compile[Evaluate[allVars],
1376     Evaluate[truncatedIntermediateHam]];
1377     (* save the compiled function *)
1378     PrintFun["Saving the compiled function for the truncated
1379     Hamiltonian and the truncated intermediate basis ..."];
1380     Export[compiledIntermediateFname, {
1381       compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
1382   )
1383 ]
1384 ]

```

```

1365 truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
1366 PrintFun["The truncated Hamiltonian has a dimension of ",
1367 truncationUmbral, "x", truncationUmbral, "..."];
1368 presentDataIndices = Select[presentDataIndices, # <=
1369 truncationUmbral &];
1370 solCompendium["presentDataIndices"] = presentDataIndices;
1371 (* the problemVars are the symbols that will be fitted for *)
1372
1373 PrintFun["Starting up the fitting process using the Levenberg-
1374 Marquardt method ..."];
1375 (* using the problemVars I need to create the argument list
1376 including _?NumericQ *)
1377 problemVarsQ = (ToString[#] <> "_?NumericQ") & /@
1378 problemVars;
1379 problemVarsQString = StringJoin[Riffle[problemVarsQ, ", "]];
1380 (* we also need to have the padded arguments with the variables
1381 in the right position and the fixed values in the remaining ones
1382 *)
1383 problemVarsPositions = Position[allVars, #][[1, 1]] & /@
1384 problemVars;
1385 problemVarsString = StringJoin[Riffle[ToString /@ problemVars,
1386 ", "]];
1387 (* to feed parameters to the Hamiltonian, which includes all
1388 parameters, we need to form the set of arguments, with fixed
1389 values where needed, and the variables in the right position *)
1390 varsWithConstants = standardValues;
1391 varsWithConstants[[problemVarsPositions]] = problemVars;
1392 varsWithConstantsString = ToString[
1393 varsWithConstants];
1394
1395 (* this following function serves eigenvalues from the
1396 Hamiltonian, has memoization so it might grow to use a lot of RAM
1397 *)
1398 Clear[HamSortedEigenvalues];
1399 hamEigenvaluesTemplate = StringTemplate["<|HamSortedEigenvalues['problemVarsQ']|:=(<|ham=compileIntermediateTruncatedHam@@'varsWithConstants'|>;eigenValues=Chop[Sort@Eigenvalues@ham];eigenValues=eigenValues-Min[eigenValues];HamSortedEigenvalues['problemVarsString']=eigenValues;Return[eigenValues]|>)"];
1400 hamString = hamEigenvaluesTemplate[<|
1401 "problemVarsQ" -> problemVarsQString,
1402 "varsWithConstants" -> varsWithConstantsString,
1403 "problemVarsString" -> problemVarsString
1404 |>];
1405 ToExpression[hamString];
1406
1407 (* we also need a function that will pick the i-th eigenvalue,
1408 this seems unnecessary but it's needed to form the right
1409 functional form expected by the Levenberg-Marquardt method *)
1410 eigenvalueDispenserTemplate = StringTemplate["<|PartialHamEigenvalues['problemVarsQ'][i_]|:=(<|eigenVals=HamSortedEigenvalues['problemVarsString'];eigenVals[[i]]|>)"];
1411 eigenValueDispenserString = eigenvalueDispenserTemplate[<|
1412 "problemVarsQ" -> problemVarsQString,
1413 "problemVarsString" -> problemVarsString
1414 |>];
1415 ToExpression[eigenValueDispenserString];
1416
1417 PrintFun["Determining the free variables after constraints ..."];
1418 constrainedProblemVars = (problemVars /. constraints);
1419 constrainedProblemVarsList = Variables[constrainedProblemVars];
1420 If[addShift,
1421 PrintFun["Adding a constant shift to the fitting parameters ..."];
1422 constrainedProblemVarsList = Append[constrainedProblemVarsList,
1423 \[Epsilon]];
1424 ];

```

```

1422
1423     indepVars = Complement[problemVars, #[[1]] & /@ constraints];
1424     stringPartialVars = ToString/@constrainedProblemVarsList;
1425
1426     paramSols = {};
1427     rmsHistory = {};
1428     steps = 0;
1429     problemVarsWithStartValues = KeyValueMap[{#1, #2} &, startValues];
1430     If[addShift,
1431         problemVarsWithStartValues = Append[problemVarsWithStartValues,
1432         {\[Epsilon], 0}];
1433     ];
1434     openNotebooks = If[runningInteractive,
1435         ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks[],
1436         {}];
1437     If[Not[MemberQ[openNotebooks, "Solver Progress"]] && OptionValue["ProgressView"],
1438         ProgressNotebook["Basic" -> False]
1439     ];
1440     degressOfFreedom = Length[presentDataIndices] - Length[
1441     problemVars] - 1;
1442     PrintFun["Fitting for ", Length[presentDataIndices], " data
1443     points with ", Length[problemVars], " free parameters.", " The
1444     effective degrees of freedom are ", degressOfFreedom, " ..."];
1445
1446     PrintFun["Fitting model to data ..."];
1447     startTime = Now;
1448     shiftToggle = If[addShift, 1, 0];
1449     sol = FindMinimum[
1450         Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
1451             {j, presentDataIndices}],
1452         problemVarsWithStartValues,
1453         Method -> "LevenbergMarquardt",
1454         MaxIterations -> OptionValue["MaxIterations"],
1455         AccuracyGoal -> OptionValue["AccuracyGoal"],
1456         StepMonitor :> (
1457             steps += 1;
1458             currentSqSum = Sum[(expData[[j]][[1]] - (
1459                 PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle
1460                 * \[Epsilon])^2, {j, presentDataIndices}];
1461             currentRMS = Sqrt[currentSqSum / degressOfFreedom];
1462             paramSols = AddToList[paramSols, constrainedProblemVarsList,
1463                 maxHistory];
1464             rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
1465         )
1466     ];
1467     endTime = Now;
1468     timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
1469     PrintFun["Solution found in ", timeTaken, "s"];
1470
1471     solVec = constrainedProblemVars /. sol[[-1]];
1472     indepSolVec = indepVars /. sol[[-1]];
1473     If[addShift,
1474         \[Epsilon]Best = \[Epsilon]/. sol[[-1]],
1475         \[Epsilon]Best = 0
1476     ];
1477     fullSolVec = standardValues;
1478     fullSolVec[[problemVarsPositions]] = solvec;
1479     PrintFun["Calculating the truncated numerical Hamiltonian
1480     corresponding to the solution ..."];
1481     fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
1482     PrintFun["Calculating energies and eigenvectors ..."];
1483     {eigenEnergies, eigenVectors} = Eigensystem[fullHam];
1484     states = Transpose[{eigenEnergies, eigenVectors}];
1485     states = SortBy[states, First];
1486     eigenEnergies = First /@ states;
1487     PrintFun["Shifting energies to make ground state zero of energy
1488     ..."];
1489     eigenEnergies = eigenEnergies - eigenEnergies[[1]];
1490     PrintFun["Calculating the linear approximant to each eigenvalue
1491     ..."];
1492     allVarsVec = Transpose[{allVars}];
1493     p0 = Transpose[{fullSolVec}];
1494     linMat = {};

```

```

1485 If [addShift,
1486   tail = -2,
1487   tail = -1];
1488 Do [
1489   (
1490     aVarPosition = Position [allVars, aVar][[1, 1]];
1491     isolationValues = ConstantArray [0, Length [allVars]];
1492     isolationValues [[aVarPosition]] = 1;
1493     dependentVars = KeyValueMap [{#1, D [#2, aVar]} &, Association [
1494       constraints]];
1495     Do [
1496       isolationValues [[Position [allVars, dVar[[1]]][[1, 1]]]] =
1497       dVar [[2]],
1498       {dVar, dependentVars}
1499     ];
1500     perHam = compileIntermediateTruncatedHam @@ isolationValues;
1501     lin = FirstOrderPerturbation [Last /@ states, perHam];
1502     linMat = Append [linMat, lin];
1503   ),
1504   {aVar, constrainedProblemVarsList [[;; tail]]}
1505 ];
1506 PrintFun ["Removing the gradient of the ground state ..."];
1507 linMat = (# - # [[1]] & /@ linMat);
1508 PrintFun ["Transposing derivative matrices into columns ..."];
1509 linMat = Transpose [linMat];
1510
1511 PrintFun ["Calculating the eigenvalue vector at solution ..."];
1512 \[Lambda]0Vec = Transpose [{eigenEnergies [[presentDataIndices]]}];
1513 PrintFun ["Putting together the experimental vector ..."];
1514 \[Lambda]exp = Transpose [{First /@ expData [[presentDataIndices]]}];
1515
1516 problemVarsVec = If [addShift,
1517   Transpose [{constrainedProblemVarsList [[;; -2]]}],
1518   Transpose [{constrainedProblemVarsList}]
1519 ];
1520 indepSolVecVec = Transpose [{indepSolVec}];
1521 PrintFun ["Calculating the difference between eigenvalues at
1522 solution ..."];
1523 diff = If [linMat == {},
1524   (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best,
1525   (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best + linMat [[
1526   presentDataIndices]].(problemVarsVec - indepSolVecVec)
1527 ];
1528 PrintFun ["Calculating the sum of squares of differences around
1529 solution ..."];
1530 sqdiff = Expand [(Transpose [diff] . diff)[[1, 1]]];
1531 PrintFun ["Calculating the minimum (which should coincide with sol
1532 ) ..."];
1533 minpoly = sqdiff /. AssociationThread [problemVars -> solVec
1534 ];
1535 fmSolAssoc = Association [sol [[2]]];
1536 If [\[\[Sigma\]exp == Automatic,
1537   \[\[Sigma\]exp = Sqrt [minpoly / degressOfFreedom];
1538 ];
1539 \[\[CapitalDelta]\]\[\[Chi\]2 = Sqrt [degressOfFreedom];
1540 Amat = (1/\[\[Sigma\]exp^2) * Transpose [linMat [[presentDataIndices
1541 ]]].linMat [[presentDataIndices]];
1542 paramIntervals = EllipsoidBoundingBox [Amat, \[\[CapitalDelta]\]\[\[Chi
1543 \]2];
1544 PrintFun ["Calculating the uncertainty in the parameters ..."];
1545 solWithUncertainty = Table [
1546   (
1547     aVar = constrainedProblemVarsList [[varIdx]];
1548     paramBest = aVar /. fmSolAssoc;
1549     (aVar -> {paramBest, paramIntervals [[varIdx, 2]]})
1550   ),
1551   {varIdx, 1, Length [constrainedProblemVarsList]-shiftToggle}
1552 ];
1553
1554 bestRMS = Sqrt [minpoly / degressOfFreedom];
1555 bestParams = sol [[2]];
1556 bestWithConstraints = Association @Join [constraints, bestParams];
1557 bestWithConstraints = bestWithConstraints /. bestWithConstraints;
1558 bestWithConstraints = (# + 0.) & /@ bestWithConstraints;
1559
1560 solCompendium ["degreesOfFreedom"] = degressOfFreedom;

```

```

1551 solCompendium["solWithUncertainty"] = solWithUncertainty;
1552 solCompendium["truncatedDim"] = truncationUmbral;
1553 solCompendium["fittedLevels"] = Length[presentDataIndices];
1554 solCompendium["actualSteps"] = steps;
1555 solCompendium["bestRMS"] = bestRMS;
1556 solCompendium["problemVars"] = problemVars;
1557 solCompendium["paramSols"] = paramSols;
1558 solCompendium["rmsHistory"] = rmsHistory;
1559 solCompendium["Appendix"] = OptionValue["AppendToFile"];
1560 solCompendium["timeTaken/s"] = timeTaken;
1561 solCompendium["bestParams"] = bestParams;
1562 solCompendium["bestParamsWithConstraints"] = bestWithConstraints;
1563
1564 If[OptionValue["SaveEigenvectors"],
1565   solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]} &/@ (Chop /@ ShiftedLevels[states]),
1566   (
1567     finalEnergies = Sort[First /@ states];
1568     finalEnergies = finalEnergies - finalEnergies[[1]];
1569     finalEnergies = finalEnergies + \[Epsilon]Best;
1570     finalEnergies = Chop /@ finalEnergies;
1571     solCompendium["energies"] = finalEnergies;
1572   )
1573 ];
1574 If[OptionValue["SaveToLog"],
1575   PrintFun["Saving the solution to the log file ..."];
1576   LogSol[solCompendium, logFilePrefix];
1577 ];
1578 PrintFun["Finished ..."];
1579 Return[solCompendium];
1580 )
1581 ];
1582
1583
1584 MostlyOrthogonalFit::usage="MostlyOrthogonalFit[numE, expData,
1585   excludeDataIndices, problemVars, startValues, \[Sigma]exp,
1586   constraints_List, Options] fits the given expData in an f^numE
1587   configuration, by using the symbols in problemVars. The symbols
1588   given in problemVars may be constrained or held constant, this
1589   being controlled by constraints list which is a list of
1590   replacement rules expressing desired constraints. The constraints
1591   list additional constraints imposed upon the model parameters that
1592   remain once other simplifications have been \"baked\" into the
1593   compiled Hamiltonians that are used to increase the speed of the
1594   calculation.
1595
1596 Important, note that in the case of odd number of electrons the given
1597   data must explicitly include the Kramers degeneracy;
1598   excludeDataIndices must be compatible with this.
1599
1600 The list expData needs to be a list of lists with the only
1601   restriction that the first element of them corresponds to energies
1602   of levels. In this list, an empty value can be used to indicate
1603   known gaps in the data. Even if the energy value for a level is
1604   known (and given in expData) certain values can be omitted from
1605   the fitting procedure through the list excludeDataIndices, which
1606   correspond to indices in expData that should be skipped over.
1607
1608 The Hamiltonian used for fitting is version that has been truncated
1609   either by using the maximum energy given in expData or by manually
1610   setting a truncation energy using the option \"TruncationEnergy\"".
1611
1612 The argument \[Sigma]exp is the estimated uncertainty in the
1613   differences between the calculated and the experimental energy
1614   levels. This is used to estimate the uncertainty in the fitted
1615   parameters. Admittedly this will be a rough estimate (at least on
1616   the contribution of the calculated uncertainty), but it is better
1617   than nothing and may at least provide a lower bound to the
1618   uncertainty in the fitted parameters. It is assumed that the
1619   uncertainty in the differences between the calculated and the
1620   experimental energy levels is the same for all of them.
1621
1622 The list startValues is a list with all of the parameters needed to

```

```

1595 define the Hamiltonian (including the initial values for
1596 problemVars).
1597
1598 The function saves the solution to a file. The file is named with a
1599 prefix (controlled by the option \"FilePrefix\") and a UUID. The
1600 file is saved in the log sub-directory as a .m file.
1601
1602 Here's a description of the different parts of this function: first
1603 the Hamiltonian is assembled and simplified using the given
1604 simplifications. Then the intermediate coupling basis is
1605 calculated using the free-ion parameters for the given lanthanide.
1606 The Hamiltonian is then changed to the intermediate coupling
1607 basis and truncated. The truncated Hamiltonian is then compiled
1608 into a function that can be used to calculate the energy levels of
1609 the truncated Hamiltonian. The function that calculates the
1610 energy levels is then used to fit the experimental data. The
1611 fitting is done using FindMinimum with the Levenberg-Marquardt
1612 method.
1613
1614 The function returns an association with the following keys:
1615
1616 - \"bestRMS\" which is the best  $\langle \Sigma \rangle$  value found.
1617 - \"bestParams\" which is the best set of parameters found for the
1618 variables that were not constrained.
1619 - \"bestParamsWithConstraints\" which has the best set of parameters
1620 (from - \\"bestParams\\") together with the used constraints. These
1621 include all the parameters in the model, even those that were not
1622 fitted for.
1623 - \\"paramSols\" which is a list of the parameters trajectories during
1624 the stepping of the fitting algorithm.
1625 - \\"timeTaken/s\" which is the time taken to find the best fit.
1626 - \\"simplifier\" which is the simplifier used to simplify the
1627 Hamiltonian.
1628 - \\"excludeDataIndices\" as given in the input.
1629 - \\"startValues\" as given in the input.
1630
1631 - \\"freeIonSymbols\" which are the symbols used in the intermediate
1632 coupling basis.
1633 - \\"truncationEnergy\" which is the energy used to truncate the
1634 Hamiltonian, if it was set to Automatic, the value here is the
1635 actual energy used.
1636 - \\"numE\" which is the number of electrons in the fnumE
1637 configuration.
1638 - \\"expData\" which is the experimental data used for fitting.
1639 - \\"problemVars\" which are the symbols considered for fitting
1640
1641 - \\"maxIterations\" which is the maximum number of iterations used by
1642 NMinimize.
1643 - \\"hamDim\" which is the dimension of the full Hamiltonian.
1644 - \\"allVars\" which are all the symbols defining the Hamiltonian
1645 under the aggregate simplifications.
1646 - \\"freeBies\" which are the free-ion parameters used to define the
1647 intermediate coupling basis.
1648 - \\"truncatedDim\" which is the dimension of the truncated
1649 Hamiltonian.
1650 - \\"compiledIntermediateFname\" the file name of the compiled
1651 function used for the truncated Hamiltonian.
1652
1653 - \\"fittedLevels\" which is the number of levels fitted for.
1654 - \\"actualSteps\" the number of steps that FindMininum actually
1655 took.
1656 - \\"solWithUncertainty\" which is a list of replacement rules of the
1657 form (paramSymbol -> {bestEstimate, uncertainty}).
1658 - \\"rmsHistory\" which is a list of the  $\langle \Sigma \rangle$  values found during
1659 the fitting.
1660 - \\"Appendix\" which is an association appended to the log file under
1661 the key \\"Appendix\\".
1662 - \\"presentDataIndices\" which is the list of indices in expData that
1663 were used for fitting, this takes into account both the empty
1664 indices in expData and also the indices in excludeDataIndices.
1665
1666 - \\"states\" which contains a list of eigenvalues and eigenvectors
1667 for the fitted model, this is only available if the option \"
1668 SaveEigenvectors\" is set to True; if a general shift of energy
1669 was allowed for in the fitting, then the energies are shifted
1670 accordingly.

```

```

1632 - \\"energies\\" which is a list of the energies of the fitted levels,
      this is only available if the option \\\"SaveEigenvectors\\" is set
      to False. If a general shift of energy was allowed for in the
      fitting, then the energies are shifted accordingly.
1633 - \\"degreesOfFreedom\\" which is equal to the number of fitted state
      energies minus the number of parameters used in fitting.
1634
1635 The function admits the following options with corresponding default
      values:
1636 - \\"MaxHistory\\" : determines how long the logs for the solver can be
      .
1637 - \\"MaxIterations\\": determines the maximum number of iterations used
      by NMinimize.
1638 - \\"FilePrefix\\" : the prefix to use for the subfolder in the log
      folder, in which the solution files are saved, by default this is
      \\\"calcs\\" so that the calculation files are saved under the
      directory \\\"log/calcs\\".
1639 - \\"AddConstantShift\\" : if True then a constant shift is allowed in
      the fitting, default is False. If this is the case the variable \\
      \\[Epsilon]\\" is added to the list of variables to be fitted for,
      it must not be included in problemVars.
1640
1641 - \\"AccuracyGoal\\": the accuracy goal used by NMinimize, default of
      5.
1642 - \\"TruncationEnergy\\": if Automatic then the maximum energy in
      expData is taken, else it takes the value set by this option. In
      all cases the energies in expData are only considered up to this
      value.
1643 - \\"PrintFun\\": the function used to print progress messages, the
      default is PrintTemporary.
1644 - \\"RefParamsVintage\\": the vintage of the reference parameters to
      use. The reference parameters are both used to determine the
      truncated Hamiltonian, and also as starting values for the solver.
      It may be \\\"LaF3\\", in which case reference parameters from
      Carnall are used. It may also be \\\"LiYF4\\", in which case the
      reference parameters from the LiYF4 paper are used. It may also be
      Automatic, in which case the given experimental data is used to
      determine starting values for F^k and  $\zeta$ . It may also be a list or
      association that provides values for the Slater integrals and spin
      -orbit coupling, the remaining necessary parameters complemented
      by using \\\"LaF3\\".
1645
1646 - \\"ProgressView\\": whether or not a progress window will be opened
      to show the progress of the solver, the default is True.
1647 - \\"SignatureCheck\\": if True then then the function returns
      prematurely, returning a list with the symbols that would have
      defined the Hamiltonian after all simplifications have been
      applied. Useful to check the entire parameter set that the
      Hamiltonian has.
1648 - \\"SaveEigenvectors\\": if True then the both the eigenvectors and
      eigenvalues are saved under the \\\"states\\" key of the returned
      association. If False then only the energies are saved, the
      default is False.
1649
1650 - \\"AppendToFile\\": an association appended to the log file under
      the key \\\"Appendix\\".
1651 - \\"MagneticSimplifier\\": a list of replacement rules to simplify the
      Marvin and pesudo-magnetic paramters. Here the ratios of the
      Marvin parameters and the pseudo-magnetic parameters are defined
      to simplify the magnetic part of the Hamiltonian.
1652 - \\"MagFieldSimplifier\\": a list of replacement rules to specify a
      magnetic field (in T), if set to {}, then {Bx, By, Bz} can also be
      used as variables to be fitted for.
1653
1654 - \\"SymmetrySimplifier\\": a list of replacements rules to simplify
      the crystal field.
1655 - \\"OtherSimplifier\\": an additional list of replacement rules that
      are applied to the Hamiltonian before computing with it. Here the
      spin-spin contribution can be turned off by setting \\[Sigma]SS->0,
      which is the default.
1656 ";
1657 Options[MostlyOrthogonalFit] = {
1658   "MaxHistory"      -> 200,
1659   "MaxIterations"   -> 100,
1660   "FilePrefix"      -> "calcs",
1661   "ProgressView"    -> True,

```

```

1662 "TruncationEnergy" -> Automatic ,
1663 "AccuracyGoal" -> 5,
1664 "PrintFun" -> PrintTemporary ,
1665 "RefParamsVintage" -> "LaF3",
1666 "SignatureCheck" -> False ,
1667 "AddConstantShift" -> False ,
1668 "SaveEigenvectors" -> False ,
1669 "AppendToLogFile" -> <||>,
1670 "SaveToLog" -> False ,
1671 "Energy Uncertainty in K" -> Automatic ,
1672 "MagneticSimplifier" -> {
1673   M2 -> 56/100 MO ,
1674   M4 -> 31/100 MO ,
1675   P4 -> 1/2 P2 ,
1676   P6 -> 1/10 P2
1677 },
1678 "MagFieldSimplifier" -> {
1679   Bx -> 0,
1680   By -> 0,
1681   Bz -> 0
1682 },
1683 "SymmetrySimplifier" -> {
1684   B12->0, B14->0, B16->0, B34->0, B36->0, B56->0,
1685   S12->0 ,S14->0, S16->0, S22->0, S24->0, S26->0,
1686   S34->0 ,S36->0, S44->0, S46->0, S56->0, S66->0
1687 },
1688 "OtherSimplifier" -> {
1689   EOp->0,
1690   P0->0,
1691   \[\Sigma\] SS->0 ,
1692   T11p->0 ,
1693   T12->0 ,
1694   T14->0 ,
1695   T15->0 ,
1696   T16->0 ,
1697   T18->0 ,
1698   T17->0 ,
1699   T19->0 ,
1700   wChErrA ->0 ,
1701   wChErrB ->0
1702 },
1703 "ThreeBodySimplifier" -> <|
1704   1 -> {
1705     T2->0 ,
1706     T3->0 ,
1707     T4->0 ,
1708     T6->0 ,
1709     T7->0 ,
1710     T8->0 ,
1711     T11p->0 ,
1712     T12->0 ,
1713     T14->0 ,
1714     T15->0 ,
1715     T16->0 ,
1716     T18->0 ,
1717     T17->0 ,
1718     T19->0 ,
1719     T2p ->0 } ,
1720   2 -> {
1721     T2->0 ,
1722     T3->0 ,
1723     T4->0 ,
1724     T6->0 ,
1725     T7->0 ,
1726     T8->0 ,
1727     T11p->0 ,
1728     T12->0 ,
1729     T14->0 ,
1730     T15->0 ,
1731     T16->0 ,
1732     T18->0 ,
1733     T17->0 ,
1734     T19->0 ,
1735     T2p ->0
1736 },
1737   3 -> {t2Switch -> 1},

```

```

1738 4 -> {t2Switch -> 1},
1739 5 -> {t2Switch -> 1},
1740 6 -> {t2Switch -> 1},
1741 7 -> {t2Switch -> 1},
1742 8 -> {t2Switch -> 0},
1743 9 -> {t2Switch -> 0},
1744 10 -> {t2Switch -> 0},
1745 11 -> {t2Switch -> 0},
1746 12 -> {
1747   t2Switch -> 0,
1748   T2->0,
1749   T2p->0,
1750   T3->0,
1751   T4->0,
1752   T6->0,
1753   T7->0,
1754   T8->0,
1755   T11p->0,
1756   T12->0,
1757   T14->0,
1758   T15->0,
1759   T16->0,
1760   T18->0,
1761   T17->0,
1762   T19->0
1763 },
1764 13->{
1765   t2Switch -> 0,
1766   T2->0,
1767   T2p->0,
1768   T3->0,
1769   T4->0,
1770   T6->0,
1771   T7->0,
1772   T8->0,
1773   T11p->0,
1774   T12->0,
1775   T14->0,
1776   T15->0,
1777   T16->0,
1778   T18->0,
1779   T17->0,
1780   T19->0
1781 }
1782 |>,
1783 "FreeIonSymbols" -> {E0p, E1p, E2p, E3p,  $\zeta$ }
1784 };
1785 MostlyOrthogonalFit[numE_Integer, expData_List,
1786   excludeDataIndices_List, problemVars_List, startValues_Association
1787   , constraints_List, OptionsPattern[]]:=Module[
1788   {accuracyGoal,
1789   allFreeEnergies,
1790   allFreeEnergiesSorted,
1791   allVars,
1792   allVarsVec,
1793   argsForEvalInsideOfTheIntermediateSystems,
1794   argsOfTheIntermediateEigensystems,
1795   aVar,
1796   aVarPosition,
1797   basis,
1798   basisChanger,
1799   basisChangerBlocks,
1800   bestParams,
1801   bestRMS,
1802   blockShifts,
1803   blockSizes,
1804   compiledDiagonal,
1805   compiledIntermediateFname,
1806   constrainedProblemVars,
1807   constrainedProblemVarsList,
1808   currentRMS,
1809   degreesOfFreedom,
1810   dependentVars,
1811   diagonalBlocks,

```

```

1812 diagonalScalarBlocks ,
1813 diff ,
1814 eigenEnergies ,
1815 eigenvalueDispenserTemplate ,
1816 eigenVectors ,
1817 elevatedIntermediateEigensystems ,
1818
1819 endTime ,
1820 fmSolAssoc ,
1821 freeBies ,
1822 freeLenergiesAndMultiplets ,
1823 fullHam ,
1824 fullSolvVec ,
1825 ham ,
1826 hamDim ,
1827 hamEigenvaluesTemplate ,
1828 hamString ,
1829
1830 indepSolvVecVec ,
1831 indepVars ,
1832 intermediateHam ,
1833 isolationValues ,
1834 lin ,
1835 linMat ,
1836 ln ,
1837 lnParams ,
1838 logFilePrefix ,
1839 magneticSimplifier ,
1840
1841 maxFreeEnergy ,
1842 maxHistory ,
1843 maxIterations ,
1844 minFreeEnergy ,
1845 minpoly ,
1846 modelSymbols ,
1847 multipletAssignments ,
1848 needlePosition ,
1849 numBlocks ,
1850 openNotebooks ,
1851
1852 ordering ,
1853 otherSimplifier ,
1854 p0 ,
1855 paramBest ,
1856 perHam ,
1857 presentDataIndices ,
1858 PrintFun ,
1859 problemVarsPositions ,
1860 problemVarsQ ,
1861 problemVarsQString ,
1862
1863 problemVarsVec ,
1864 problemVarsWithStartValues ,
1865 reducedModelSymbols ,
1866 RefParams ,
1867 roundedTruncationEnergy ,
1868 runningInteractive ,
1869 shiftToggle ,
1870 simplifier ,
1871 sol ,
1872 solCompendium ,
1873
1874 solWithUncertainty ,
1875 sortedTruncationIndex ,
1876 sqdiff ,
1877 standardValues ,
1878 startTime ,
1879 states ,
1880 steps ,
1881 symmetrySimplifier ,
1882 theIntermediateEigensystems ,
1883 TheIntermediateEigensystems ,
1884
1885 timeTaken ,
1886 truncatedIntermediateBasis ,
1887 truncatedIntermediateHam ,

```

```

1888 truncationEnergy ,
1889 truncationIndices ,
1890 truncationUmbral ,
1891 varHash ,
1892 varsWithConstants ,
1893 \[Lambda]0Vec ,
1894 \[Lambda]exp
1895 },
1896 (
1897 \[Sigma]exp = OptionValue["Energy Uncertainty in K"];
1898 refParamsVintage = OptionValue["RefParamsVintage"];
1899 RefParams = Which[
1900 refParamsVintage === "LaF3",
1901 LoadLaF3Parameters,
1902 refParamsVintage === "LiYF4",
1903 LoadLiYF4Parameters,
1904 True,
1905 refParamsVintage
1906 ];
1907 solCompendium = <||>;
1908 hamDim = Binomial[14, numE];
1909 addShift = OptionValue["AddConstantShift"];
1910 ln = theLanthanides[[numE]];
1911 maxHistory = OptionValue["MaxHistory"];
1912 maxIterations = OptionValue["MaxIterations"];
1913 logFilePrefix = If[OptionValue["FilePrefix"] == "",
1914 ToString[theLanthanides[[numE]]],
1915 OptionValue["FilePrefix"]
1916 ];
1917 accuracyGoal = OptionValue["AccuracyGoal"];
1918 PrintFun = OptionValue["PrintFun"];
1919 freeIonSymbols = OptionValue["FreeIonSymbols"];
1920 runningInteractive = (Head[$ParentLink] === LinkObject);
1921 magneticSimplifier = OptionValue["MagneticSimplifier"];
1922 magFieldSimplifier = OptionValue["MagFieldSimplifier"];
1923 symmetrySimplifier = OptionValue["SymmetrySimplifier"];
1924 otherSimplifier = OptionValue["OtherSimplifier"];
1925 threeBodySimplifier = If[Head[OptionValue["ThreeBodySimplifier"]]
1926 == Association,
1927 OptionValue["ThreeBodySimplifier"][[numE]],
1928 OptionValue["ThreeBodySimplifier"]
1929 ];
1930 truncationEnergy = If[OptionValue["TruncationEnergy"] ===
1931 Automatic,
1932 (
1933 PrintFun["Truncation energy set to Automatic, using the
1934 maximum energy (+20%) in the data ..."];
1935 Round[1.2 * Max[Select[First /@ expData, NumericQ[#] &]]
1936 ],
1937 OptionValue["TruncationEnergy"]
1938 ];
1939 truncationEnergy = Max[50000, truncationEnergy];
1940 PrintFun["Using a truncation energy of ", truncationEnergy, " K"
1941 ];
1942 simplifier = Join[magneticSimplifier,
1943 magFieldSimplifier,
1944 symmetrySimplifier,
1945 threeBodySimplifier,
1946 otherSimplifier];
1947 PrintFun["Determining gaps in the data ..."];
1948 (* whatever is non-numeric is assumed as a known gap *)
1949 presentDataIndices = Flatten[Position[expData, {_?(NumericQ[#] &
1950 , ___]}];
1951 (* some indices omitted here based on the excludeDataIndices
1952 argument, note that presentDataIndices is somewhat a misnomer*)
1953 presentDataIndices = Complement[presentDataIndices,
1954 excludeDataIndices];
1955
1956 solCompendium["simplifier"] = simplifier;
1957 solCompendium["excludeDataIndices"] = excludeDataIndices;
1958 solCompendium["startValues"] = startValues;
1959 solCompendium["freeIonSymbols"] = freeIonSymbols;
1960 solCompendium["truncationEnergy"] = truncationEnergy;

```

```

1957 solCompendium["numE"] = numE;
1958 solCompendium["expData"] = expData;
1959 solCompendium["problemVars"] = problemVars;
1960 solCompendium["maxIterations"] = maxIterations;
1961 solCompendium["hamDim"] = hamDim;
1962 solCompendium["constraints"] = constraints;
1963
1964 modelSymbols = Select[paramSymbols,
1965   Not[MemberQ[Join[racahSymbols,
1966     juddOfeltIntensitySymbols,
1967     slaterSymbols,
1968     {\alpha, \beta, \gamma},
1969     {T2},
1970     chenSymbols,
1971     {t2Switch, \[Epsilon], gs, nE}], #]] &
1972 ];
1973 modelSymbols = Sort[modelSymbols];
1974 (* remove the symbols that will be removed by the simplifier, no
symbol should remain here that is not in the symbolic Hamiltonian
*)
1975 reducedModelSymbols = Select[modelSymbols, Not[MemberQ[Keys[
simplifier], #]] &];
1976
1977 (* this is useful to understand what are the arguments of the
truncated compiled Hamiltonian *)
1978 If[OptionValue["SignatureCheck"],
1979 (
1980   PrintFun["Given the model parameters and the simplifying
assumptions, the resultant model parameters are:"];
1981   PrintFun[{reducedModelSymbols}];
1982   PrintFun["Exiting ..."];
1983   Return[];
1984 )
1985 ];
1986
1987 (* calculate the basis *)
1988 PrintFun["Retrieving the LSJM basis for f^", numE, " ..."];
1989 basis = BasisLSJM[numE];
1990
1991 Which[refParamsVintage === Automatic,
1992 (
1993   PrintFun["This is not currently supported, please provide a
vintage for the reference parameters."];
1994   Return[$Failed];
1995 ),
1996 MemberQ[{List, Association}, Head[RefParams]],
1997 (
1998   RefParams = Association[RefParams];
1999   PrintFun["Using the given parameters as a starting point ..."]
];
2000   lnParams = RefParams;
2001   extraParams = FromNonOrthogonalToMostlyOrthogonal[
LoadLaF3Parameters[ln], numE];
2002   lnParams = Join[extraParams, lnParams];
2003 ),
2004 True,
2005 (
2006   (* get the reference parameters from the given vintage *)
2007   PrintFun["Getting reference free-ion parameters for ", ln, "
using ", refParamsVintage, " ..."];
2008   lnParams = ParamPad[FromNonOrthogonalToMostlyOrthogonal[
RefParams[ln], numE],
2009   "PrintFun" -> PrintFun];
2010 )
2011 ];
2012
2013 freeBies = Prepend[Values[(# -> (# /. lnParams)) &/@ freeIonSymbols], numE];
2014
2015 (* a more explicit alias *)
2016 allVars = reducedModelSymbols;
2017 numericConstraints = Association@Select[constraints, NumericQ
[#[[2]]] &];
2018 standardValues = allVars /. Join[lnParams, numericConstraints];
2019 solCompendium["allVars"] = allVars;

```

```

2020     solCompendium["freeBies"] = freeBies;
2021
2022 (* reload compiled version if found *)
2023 varHash = Hash[{numE, allVars, freeBies,
2024 truncationEnergy, simplifier}];
2025 compiledIntermediateFname = ln <> "--compiled-intermediate-
truncated-ham-" <> ToString[varHash] <> ".mx";
2026 compiledIntermediateFname = FileNameJoin[{moduleDir, "compiled",
2027 compiledIntermediateFname}];
2028 solCompendium["compiledIntermediateFname"] =
2029 compiledIntermediateFname;
2030
2031 If[FileExistsQ[compiledIntermediateFname],
2032 (
2033   PrintFun["This ion, free-ion params, and full set of variables
have been used before (as determined by {numE, allVars, freeBies,
truncationEnergy, simplifier}). Loading the previously saved
compiled function and intermediate coupling basis ..."];
2034   PrintFun["Using : ", compiledIntermediateFname];
2035   {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =
2036   Import[compiledIntermediateFname];
2037 ),
2038 (
2039   If[truncationEnergy == Infinity,
2040   (
2041     ham = EffectiveHamiltonian[numE,
2042       "ReturnInBlocks" -> False,
2043       "OperatorBasis" -> "MostlyOrthogonal"];
2044     theSimplifier = simplifier;
2045     ham = Normal @ ReplaceInSparseArray[ham, simplifier];
2046     PrintFun["Compiling a function for the Hamiltonian with no
truncation ..."];
2047     (* compile a function that will calculate the truncated
Hamiltonian given the parameters in allVars, this is the function
to be use in fitting *)
2048     compileIntermediateTruncatedHam = Compile[Evaluate[allVars
], Evaluate[ham]];
2049     truncatedIntermediateBasis =
2050     SparseArray@IdentityMatrix[Binomial[14, numE]];
2051     (* save the compiled function *)
2052     PrintFun["Saving the compiled function for the Hamiltonian
with no truncation and a placeholder intermediate basis ..."];
2053     Export[compiledIntermediateFname, {
2054       compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
2055   ),
2056   (
2057     (* grab the Hamiltonian preserving the block structure *)
2058     PrintFun["Assembling the Hamiltonian for f^", numE, " keeping
the block structure ..."];
2059     ham = EffectiveHamiltonian[numE,
2060       "ReturnInBlocks" -> True,
2061       "OperatorBasis" -> "MostlyOrthogonal"];
2062     (* apply the simplifier *)
2063     PrintFun["Simplifying using the aggregate set of
simplification rules ..."];
2064     ham = Map[ReplaceInSparseArray[#, simplifier] &, ham,
2065     {2}];
2066     PrintFun["Zeroing out every symbol in the Hamiltonian that is
not a free-ion parameter ..."];
2067
2068     (* Get the free ion symbols *)
2069     freeIonSimplifier = (#->0) & /@ Complement[
2070       reducedModelSymbols, freeIonSymbols];
2071
2072     (* Take the diagonal blocks for the intermediate analysis *)
2073     PrintFun["Grabbing the diagonal blocks of the Hamiltonian ...
"];
2074     diagonalBlocks = Diagonal[ham];
2075
2076     (* simplify them to only keep the free ion symbols *)
2077     PrintFun["Simplifying the diagonal blocks to only keep the
free ion symbols ..."];
2078     diagonalScalarBlocks = ReplaceInSparseArray[#, freeIonSimplifier] &/@ diagonalBlocks;
2079
2080     (* these include the MJ quantum numbers, remove that *)

```

```

2073 PrintFun["Contracting the basis vectors by removing the MJ
2074 quantum numbers from the diagonal blocks ..."];
2075 diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
2076
2077 argsOfTheIntermediateEigensystems = StringJoin[Riffle
2078 [Prepend[ToString[#]<>"v_"] & /@ freeIonSymbols,"numE_"],", "]];
2079 argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle
2080 [({ToString[#]<>"v"}) & /@ freeIonSymbols,", "]];
2081 PrintFun["argsOfTheIntermediateEigensystems = ",
2082 argsOfTheIntermediateEigensystems];
2083 PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
2084 argsForEvalInsideOfTheIntermediateSystems];
2085 PrintFun["(if the following fails, it might help to see if
2086 the arguments of TheIntermediateEigensystems match the ones shown
2087 above)"];
2088
2089 (* compile a function that will effectively calculate the
2090 spectrum of all of the scalar blocks given the parameters of the
2091 free-ion part of the Hamiltonian *)
2092 (* compile one function for each of the blocks *)
2093 PrintFun["Compiling functions for the diagonal blocks of the
2094 Hamiltonian ..."];
2095 compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate
2096 [N[Normal[#]]]]& /@ diagonalScalarBlocks;
2097 (* use that to create a function that will calculate the free
2098 -ion eigensystem *)
2099 TheIntermediateEigensystems[numEv_, E0pv_, E1pv_, E2pv_,
2100 E3pv_,  $\zeta$ v_] := (
2101 theNumericBlocks = (#[E0pv, E1pv, E2pv, E3pv,  $\zeta$ v]&) /@
2102 compiledDiagonal;
2103 theIntermediateEigensystems = Eigensystem /@
2104 theNumericBlocks;
2105 Js = AllowedJ[numEv];
2106 basisJ = BasisLSJMJ[numEv, "AsAssociation" -> True];
2107 (* having calculated the eigensystems with the removed
2108 degeneracies, put the degeneracies back in explicitly *)
2109 elevatedIntermediateEigensystems = MapIndexed[EigenLever
2110 [#1, 2Js[[#2[[1]]]]+1 ]&, theIntermediateEigensystems];
2111 (* Identify a single MJ to keep *)
2112 pivot = If[EvenQ[numEv], 0, -1/2];
2113 LSJmultiplets = (#[[1]]<>ToString[InputForm[#[[2]]]])&/
2114 @Select[BasisLSJMJ[numEv], #[[{-1}]]== pivot &];
2115 (* calculate the multiplet assignments that the
2116 intermediate basis eigenvectors have *)
2117 needlePosition = 0;
2118 multipletAssignments = Table[
2119 [
2120
2121 J = Js[[idx]];
2122 eigenVecs = theIntermediateEigensystems[[idx]][[2]];
2123 majorComponentIndices = Ordering[Abs[#][[-1]]]&/
2124 @eigenVecs;
2125 majorComponentIndices += needlePosition;
2126 needlePosition += Length[
2127 majorComponentIndices];
2128 majorComponentAssignments = LSJmultiplets[[#]]&/
2129 @majorComponentIndices;
2130 (* All of the degenerate eigenvectors belong to the
2131 same multiplet*)
2132 elevatedMultipletAssignments = ListRepeater[
2133 majorComponentAssignments, 2J+1];
2134 elevatedMultipletAssignments
2135 ),
2136 {idx, 1, Length[Js]}
2137 ];
2138
2139 (* put together the multiplet assignments and the energies
2140 *)
2141 freeIenergiesAndMultiplets = Transpose/@Transpose[{First/
2142 @elevatedIntermediateEigensystems, multipletAssignments}];
2143 freeIenergiesAndMultiplets = Flatten[
2144 freeIenergiesAndMultiplets, 1];
2145 (* calculate the change of basis matrix using the
2146 intermediate coupling eigenvectors *)
2147 basisChanger = BlockDiagonalMatrix[Transpose/@Last/
2148 @elevatedIntermediateEigensystems];
2149 basisChanger = SparseArray[basisChanger];

```

```

2120     Return[{theIntermediateEigensystems,
2121      multipletAssignments,
2122      elevatedIntermediateEigensystems,
2123      freeIenergiesAndMultiplets,
2124      basisChanger}]
2125   );
2126
2127   PrintFun["Calculating the intermediate eigensystems for ",ln,
2128 " using free-ion params from LaF3 ..."];
2129   (* calculate intermediate coupling basis using the free-ion
2130  params for LaF3 *)
2131   {theIntermediateEigensystems, multipletAssignments,
2132   elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
2133   basisChanger} = TheIntermediateEigensystems@@freeBies;
2134
2135   (* use that intermediate coupling basis to compile a function
2136  for the full Hamiltonian *)
2137   allFreeEnergies = Flatten[First/
2138 @elevatedIntermediateEigensystems];
2139   (* important that the intermediate coupling basis have
2140  attached energies, which make possible the truncation *)
2141   ordering = Ordering[allFreeEnergies];
2142   (* sort the free ion energies and determine which indices
2143  should be included in the truncation *)
2144   allFreeEnergiesSorted = Sort[allFreeEnergies];
2145   {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];
2146   (* determine the index at which the energy is equal or larger
2147  than the truncation energy *)
2148   sortedTruncationIndex = Which[
2149     truncationEnergy > (maxFreeEnergy - minFreeEnergy),
2150     hamDim,
2151     True,
2152     FirstPosition[allFreeEnergiesSorted - Min[
2153       allFreeEnergiesSorted], x_ /; x > truncationEnergy, {0}, 1][[1]]
2154   ];
2155   (* the actual energy at which the truncation is made *)
2156   roundedTruncationEnergy = allFreeEnergiesSorted[[sortedTruncationIndex]];
2157
2158   (* the indices that participate in the truncation *)
2159   truncationIndices = ordering[;;sortedTruncationIndex];
2160   PrintFun["Computing the block structure of the change of
2161  basis array ..."];
2162   blockSizes = BlockArrayDimensionsArray[ham];
2163   basisChangerBlocks = ArrayBlocker[basisChanger, blockSizes];
2164   blockShifts = First /@ Diagonal[blockSizes];
2165   numBlocks = Length[blockSizes];
2166
2167   (* using the ham (with all the symbols) change the basis to
2168  the computed one *)
2169   PrintFun["Changing the basis of the Hamiltonian to the
2170  intermediate coupling basis ..."];
2171   intermediateHam = BlockMatrixMultiply[ham, basisChangerBlocks];
2172
2173   PrintFun["Distributing products inside of symbolic matrix
2174  elements to keep complexity in check ..."];
2175   Do[
2176     intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
2177     intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
2178     {rowIdx, 1, numBlocks},
2179     {colIdx, 1, numBlocks}
2180   ];
2181   intermediateHam = BlockMatrixMultiply[BlockTranspose[
2182     basisChangerBlocks], intermediateHam];
2183   PrintFun["Distributing products inside of symbolic matrix
2184  elements to keep complexity in check ..."];
2185   Do[
2186     intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
2187     intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
2188     {rowIdx, 1, numBlocks},
2189     {colIdx, 1, numBlocks}
2190   ];
2191   (* using the truncation indices truncate that one *)
2192   PrintFun["Truncating the Hamiltonian ..."];
2193   truncatedIntermediateHam = TruncateBlockArray[intermediateHam
2194 , truncationIndices, blockShifts];

```

```

2175      (* these are the basis vectors for the truncated hamiltonian
2176      *)
2177      PrintFun["Saving the truncated intermediate basis ..."];
2178      truncatedIntermediateBasis = basisChanger[[All,
2179      truncationIndices]];
2180
2181      PrintFun["Compiling a function for the truncated Hamiltonian
2182      ..."];
2183      (* compile a function that will calculate the truncated
2184      Hamiltonian given the parameters in allVars, this is the function
2185      to be use in fitting *)
2186      compileIntermediateTruncatedHam = Compile[Evaluate[allVars],
2187      Evaluate[truncatedIntermediateHam]];
2188      (* save the compiled function *)
2189      PrintFun["Saving the compiled function for the truncated
2190      Hamiltonian and the truncated intermediate basis ..."];
2191      Export[compiledIntermediateFname, {
2192      compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
2193
2194      ]
2195      )
2196      ];
2197
2198      truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
2199      PrintFun["The truncated Hamiltonian has a dimension of ",
2200      truncationUmbral, "x", truncationUmbral, "..."];
2201      presentDataIndices = Select[presentDataIndices, # <=
2202      truncationUmbral &];
2203      solCompendium["presentDataIndices"] = presentDataIndices;
2204
2205      (* the problemVars are the symbols that will be fitted for *)
2206
2207      PrintFun["Starting up the fitting process using the Levenberg-
2208      Marquardt method ..."];
2209      (* using the problemVars I need to create the argument list
2210      including _?NumericQ *)
2211      problemVarsQ = (ToString[#] <> "_?NumericQ") & /@
2212      problemVars;
2213      problemVarsQString = StringJoin[Riffle[problemVarsQ, ", "]];
2214      (* we also need to have the padded arguments with the variables
2215      in the right position and the fixed values in the remaining ones
2216      *)
2217      problemVarsPositions = Position[allVars, #][[1, 1]] & /@
2218      problemVars;
2219      problemVarsString = StringJoin[Riffle[ToString /@ problemVars,
2220      ", "]];
2221      (* to feed parameters to the Hamiltonian, which includes all
2222      parameters, we need to form the set of arguments, with fixed
2223      values where needed, and the variables in the right position *)
2224      varsWithConstants = standardValues;
2225      varsWithConstants[[problemVarsPositions]] = problemVars;
2226      varsWithConstantsString = ToString[
2227      varsWithConstants];
2228
2229      (* this following function serves eigenvalues from the
2230      Hamiltonian, has memoization so it might grow to use a lot of RAM
2231      *)
2232      Clear[HamSortedEigenvalues];
2233      hamEigenvaluesTemplate = StringTemplate["
2234      HamSortedEigenvalues['problemVarsQ']:=(
2235          ham          = compileIntermediateTruncatedHam@@'
2236          varsWithConstants';
2237          eigenValues = Chop[Sort@Eigenvalues@ham];
2238          eigenValues = eigenValues - Min[eigenValues];
2239          HamSortedEigenvalues['problemVarsString'] = eigenValues;
2240          Return[eigenValues]
2241      )"];
2242      hamString = hamEigenvaluesTemplate[<|
2243          "problemVarsQ"    -> problemVarsQString,
2244          "varsWithConstants" -> varsWithConstantsString,
2245          "problemVarsString" -> problemVarsString
2246          |>];
2247      ToExpression[hamString];
2248
2249      (* we also need a function that will pick the i-th eigenvalue,
2250      this seems unnecessary but it's needed to form the right

```

```

functional form expected by the Levenberg-Marquardt method *)
2227 eigenvalueDispenserTemplate = StringTemplate["
2228 PartialHamEigenvalues['problemVarsQ'][i_]:=(
2229   eigenVals = HamSortedEigenvalues['problemVarsString'];
2230   eigenVals[[i]]
2231 )
2232 ";
2233 eigenValueDispenserString = eigenvalueDispenserTemplate[<|
2234   "problemVarsQ"      -> problemVarsQString,
2235   "problemVarsString" -> problemVarsString
2236 |>];
2237 ToExpression[eigenValueDispenserString];
2238
2239 PrintFun["Determining the free variables after constraints ..."];
2240 constrainedProblemVars = (problemVars /. constraints);
2241 constrainedProblemVarsList = Variables[constrainedProblemVars];
2242 If[addShift,
2243   PrintFun["Adding a constant shift to the fitting parameters ...";
2244   constrainedProblemVarsList = Append[constrainedProblemVarsList,
2245   \[Epsilon]];
2246 ];
2247
2248 indepVars = Complement[problemVars, #[[1]] & /@ constraints];
2249 stringPartialVars = ToString/@constrainedProblemVarsList;
2250
2251 paramSols = {};
2252 rmsHistory = {};
2253 steps = 0;
2254 problemVarsWithStartValues = KeyValueMap[{#1,#2} &, startValues];
2255 If[addShift,
2256   problemVarsWithStartValues = Append[problemVarsWithStartValues,
2257   {\[Epsilon], 0}];
2258 ];
2259 openNotebooks = If[runningInteractive,
2260   ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks[],
2261   {}];
2262 If[Not[MemberQ[openNotebooks,"Solver Progress"]] && OptionValue["ProgressView"],
2263   ProgressNotebook["Basic" -> False];
2264 ];
2265 degressOfFreedom = Length[presentDataIndices] - Length[
2266 problemVars] - 1;
2267 PrintFun["Fitting for ", Length[presentDataIndices], " data
2268 points with ", Length[problemVars], " free parameters.", " The
2269 effective degrees of freedom are ", degressOfFreedom, " ..."];
2270
2271 PrintFun["Fitting model to data ..."];
2272 startTime = Now;
2273 shiftToggle = If[addShift, 1, 0];
2274 sol = FindMinimum[
2275   Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@
2276 constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
2277   {j, presentDataIndices}],
2278   problemVarsWithStartValues,
2279   Method -> "LevenbergMarquardt",
2280   MaxIterations -> OptionValue["MaxIterations"],
2281   AccuracyGoal -> OptionValue["AccuracyGoal"],
2282   StepMonitor :> (
2283     steps += 1;
2284     currentSqSum = Sum[(expData[[j]][[1]] - (
2285       PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle
2286 * \[Epsilon])^2, {j, presentDataIndices}];
2287     currentRMS = Sqrt[currentSqSum / degressOfFreedom];
2288     paramSols = AddToList[paramSols, constrainedProblemVarsList,
2289     maxHistory];
2290     rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
2291   )
2292 ];
2293 endTime = Now;
2294 timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
2295 PrintFun["Solution found in ", timeTaken, "s"];
2296
2297 solVec = constrainedProblemVars /. sol[[-1]];
2298 indepSolveVec = indepVars /. sol[[-1]];

```

```

2290  If[addShift,
2291    \[Epsilon]Best = \[Epsilon]/. sol[[-1]],
2292    \[Epsilon]Best = 0
2293  ];
2294  fullSolVec = standardValues;
2295  fullSolVec[[problemVarsPositions]] = solVec;
2296  PrintFun["Calculating the truncated numerical Hamiltonian
corresponding to the solution ..."];
2297  fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
2298  PrintFun["Calculating energies and eigenvectors ..."];
2299  {eigenEnergies, eigenVectors} = Eigensystem[fullHam];
2300  states = Transpose[{eigenEnergies, eigenVectors}];
2301  states = SortBy[states, First];
2302  eigenEnergies = First /@ states;
2303  PrintFun["Shifting energies to make ground state zero of energy
..."];
2304  eigenEnergies = eigenEnergies - eigenEnergies[[1]];
2305  PrintFun["Calculating the linear approximant to each eigenvalue
..."];
2306  allVarsVec = Transpose[{allVars}];
2307  p0 = Transpose[{fullSolVec}];
2308  linMat = {};
2309  If[addShift,
2310    tail = -2,
2311    tail = -1];
2312  Do[
2313  (
2314    aVarPosition = Position[allVars, aVar][[1, 1]];
2315    isolationValues = ConstantArray[0, Length[allVars]];
2316    isolationValues[[aVarPosition]] = 1;
2317    dependentVars = KeyValueMap[{#1, D[#2, aVar]} &, Association[
2318 constraints]];
2319    Do[
2320      isolationValues[[Position[allVars, dVar[[1]]][[1, 1]]]] =
2321      dVar[[2]],
2322      {dVar, dependentVars}
2323    ];
2324    perHam = compileIntermediateTruncatedHam @@ isolationValues;
2325    lin = FirstOrderPerturbation[Last /@ states, perHam];
2326    linMat = Append[linMat, lin];
2327  ),
2328  {aVar, constrainedProblemVarsList[;;tail]}];
2329  PrintFun["Removing the gradient of the ground state ..."];
2330  linMat = (# - #[[1]] & /@ linMat);
2331  PrintFun["Transposing derivative matrices into columns ..."];
2332  linMat = Transpose[linMat];
2333  PrintFun["Calculating the eigenvalue vector at solution ..."];
2334  \[Lambda]0Vec = Transpose[{eigenEnergies[[presentDataIndices]]}];
2335  PrintFun["Putting together the experimental vector ..."];
2336  \[Lambda]exp = Transpose[{First /@ expData[[presentDataIndices
2337 ]]}];
2338  problemVarsVec = If[addShift,
2339    Transpose[{constrainedProblemVarsList[;;-2]}],
2340    Transpose[{constrainedProblemVarsList}]];
2341  indepSolVecVec = Transpose[{indepSolVec}];
2342  PrintFun["Calculating the difference between eigenvalues at
solution ..."];
2343  diff = If[linMat == {},
2344    (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best,
2345    (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best + linMat[[presentDataIndices]].(problemVarsVec - indepSolVecVec)
2346  ];
2347  PrintFun["Calculating the sum of squares of differences around
solution ... "];
2348  sqdiff = Expand[(Transpose[diff] . diff)[[1, 1]]];
2349  PrintFun["Calculating the minimum (which should coincide with sol
) ..."];
2350  minpoly = sqdiff /. AssociationThread[problemVars -> solVec];
2351  fmSolAssoc = Association[sol[[2]]];
2352  If[\[Sigma]exp == Automatic,
2353    \[Sigma]exp = Sqrt[minpoly / degressOfFreedom];
2354  ];

```

```

\[CapitalDelta]\[Chi]2 = Sqrt[degressOfFreedom];
Amat = (1/\[Sigma]exp^2) * Transpose[linMat[[presentDataIndices
]]].linMat[[presentDataIndices]];
paramIntervals = EllipsoidBoundingBox[Amat, \[CapitalDelta]\[Chi]
];
PrintFun["Calculating the uncertainty in the parameters ..."];
solWithUncertainty = Table[
(
    aVar = constrainedProblemVarsList[[varIdx]];
    paramBest = aVar /. fmSolAssoc;
    (aVar -> {paramBest, paramIntervals[[varIdx, 2]]})
),
{varIdx, 1, Length[constrainedProblemVarsList]-shiftToggle}
];

bestRMS = Sqrt[minpoly / degressOfFreedom];
bestParams = sol[[2]];
bestWithConstraints = Association@Join[constraints, bestParams];
bestWithConstraints = bestWithConstraints /. bestWithConstraints;
bestWithConstraints = (# + 0.) & /@ bestWithConstraints;

solCompendium["degreesOfFreedom"] = degressOfFreedom;
solCompendium["solWithUncertainty"] = solWithUncertainty;
solCompendium["truncatedDim"] = truncationUmbral;
solCompendium["fittedLevels"] = Length[presentDataIndices];
];
solCompendium["actualSteps"] = steps;
solCompendium["bestRMS"] = bestRMS;
solCompendium["problemVars"] = problemVars;
solCompendium["paramSols"] = paramSols;
solCompendium["rmsHistory"] = rmsHistory;
solCompendium["Appendix"] = OptionValue["
AppendToFile"];
solCompendium["timeTaken/s"] = timeTaken;
solCompendium["bestParams"] = bestParams;
solCompendium["bestParamsWithConstraints"] = bestWithConstraints;

If[OptionValue["SaveEigenvectors"],
    solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]}
&/@ (Chop /@ ShiftedLevels[states]),
(
    finalEnergies = Sort[First /@ states];
    finalEnergies = finalEnergies - finalEnergies[[1]];
    finalEnergies = finalEnergies + \[Epsilon]Best;
    finalEnergies = Chop /@ finalEnergies;
    solCompendium["energies"] = finalEnergies;
)
];
If[OptionValue["SaveToLog"],
    PrintFun["Saving the solution to the log file ..."];
    LogSol[solCompendium, logFilePrefix];
];
PrintFun["Finished ..."];
Return[solCompendium];
)
];
StringToSLJ[string_] := Module[
{stringed = string, LS, J, LSindex},
(
If[StringContainsQ[stringed, "+"],
    Return["mixed"]
];
LS = StringTake[stringed, {1, 2}];
If[StringContainsQ[stringed, "("],
(
    LSindex =
    StringCases[stringed, RegularExpression["\\(([^)]*)\\)"] :> "$1"
];
    LS = LS <> LSindex;
    stringed = StringSplit[stringed, ")"][[{-1}]];
    J = ToExpression[stringed];
),
(
    J = ToExpression@StringTake[stringed, {3, -1}];
)
];

```

```

2425 ];
2426 {LS, J}
2427 )
2428 ];
2429
2430 FreeIonSolver::usage="This function takes a list of experimental data
2431     and the number of electrons in the lanthanide ion and returns the
2432     free-ion parameters that best fit the data. The options are:
2433 - F4F6_SlaterRatios: a list of two numbers that represent the ratio
2434     of F4 to F2 and F6 to F2, respectively.
2435 - PrintFun: a function that will be used to print the progress of
2436     the fitting process.
2437 - MaxIterations: the maximum number of iterations that the fitting
2438     process will run.
2439 - MaxMultiplets: the maximum number of multiplets that will be used
2440     in the fitting process.
2441 - MaxPercent: the maximum percentage of the data that can be off by
2442     the fitting.
2443 - SubSetBounds: a list of two numbers that represent the minimum
2444     and maximum number of multiplets that will be used in the fitting
2445     process.
2446
2447 The function returns an association with the following keys:
2448 - bestParams: the best parameters found in the fitting.
2449 - worstRelativeError: the worst relative error in the fitting.
2450 - SlaterRatios: the Slater ratios used in the fitting.
2451 - usedBaricenters: the baricenters used in the fitting.
2452 If no acceptable solution is found, the function will return all
2453     solutions that are not worse than 10*MaxPercent. A solution is
2454     acceptable if the worst relative error is less than the MaxPercent
2455     option.
2456 ";
2457 Options[FreeIonSolver] = {
2458   "F4F6_SlaterRatios" -> {0.707, 0.516},
2459   "PrintFun" -> PrintTemporary,
2460   "MaxIterations" -> 10000,
2461   "MaxMultiplets" -> 12,
2462   "MaxPercent" -> 3.,
2463   "SubSetBounds" -> {5, 12}
2464 };
2465 FreeIonSolver[expData_, numE_, OptionsPattern[]] := Module[
2466   {maxMultiplets, maxPercent, F4overF2, F6overF2, PrintFun,
2467   minSubsetSize, maxSubsetSize, multipletEnergies, numMultiplets,
2468   allEqns, subsetSizes, ln, solutions, subsets, subset, eqns, slope,
2469   intercept, meritFun, sol, goodThings, bestThings, bestOfAll,
2470   finalSol, usedMultiplets, usedBaricenters},
2471   (
2472     maxMultiplets = OptionValue["MaxMultiplets"];
2473     maxIterations = OptionValue["MaxIterations"];
2474     maxPercent = OptionValue["MaxPercent"];
2475     F4overF2 = OptionValue["F4F6_SlaterRatios"][[1]];
2476     F6overF2 = OptionValue["F4F6_SlaterRatios"][[2]];
2477     PrintFun = OptionValue["PrintFun"];
2478     minSubsetSize = OptionValue["SubSetBounds"][[1]];
2479     maxSubsetSize = OptionValue["SubSetBounds"][[2]];
2480     freeIonParams = {F0, F2, F4, F6, \[Zeta]};
2481     ln = theLanthanides[[numE]];
2482
2483     PrintFun["Parsing the barycenters of the different multiplets ..."];
2484     multipletEnergies = Map[First, #] & /@ GroupBy[expData, #[[2]]];
2485     multipletEnergies = Mean[Select[#, NumberQ]] & /@ multipletEnergies;
2486     multipletEnergies = Select[multipletEnergies, FreeQ[#, Mean] &];
2487     multipletEnergies = KeySelect[KeyMap[StringToSLJ,
2488       multipletEnergies], # != "mixed" &];
2489     multipletEnergies = KeyMap[Prepend[#, numE] &, multipletEnergies];
2490     numMultiplets = Length[multipletEnergies];
2491
2492     PrintFun["Composing the system of equations for the free-ion
2493     energies ..."];
2494     allEqns = KeyValueMap[FreeIonTable[#1 == #2 &,
2495       multipletEnergies];
2496     allEqns = Select[allEqns, FreeQ[#, Missing] &];
2497     allEqns = Append[Coefficient[#[[1]], {F0, F2, F4, F6, \[Zeta]}],
```

```

2478 #[[2]]] & /@ allEqns;
2479 allEqns = allEqns[[;; Min[Length[allEqns], maxMultiplets]]];
2500 subsetSizes = Range[minSubsetSize, Min[numMultiplets,
maxSubsetSize]];
2501 numSubsets = #, Binomial[numMultiplets, #] & /@ subsetSizes;
2502 numSubsets = Transpose@SortBy[numSubsets, Last];
2503 accSizes = Accumulate[numSubsets[[2]]];
2504 numSubsets = Transpose@Append[numSubsets, accSizes];
2505 lastSub = SelectFirst[numSubsets, #[[3]] > 1000 &, Last[
numSubsets]];
2506 lastPosition = Position[numSubsets, lastSub][[1, 1]];
2507 chosenSubsetSizes = #[[1]] & /@ numSubsets[[;; lastPosition]];
2508 solutions = <||>;
2509
2510 PrintFun["Selecting subsets of different lengths and fitting with
ratio-constraints ..."];
2511 Do[
2512   subsets = Subsets[Range[1, Length[allEqns]], {subsetSize}];
2513   PrintFun["Considering ", Length[subsets], " barycenter subsets
of size ", subsetSize, " ..."];
2514   Do[
2515     (
2516       subset = subsets[[subsetIndex]];
2517       eqns = allEqns[[subset]];
2518       slope = #[[;; 5]] & /@ eqns;
2519       intercept = #[[6]] & /@ eqns;
2520       meritFun = Max[100 * Expand[Abs[(slope . freeIonParams -
intercept)]/intercept]];
2521       sol = NMinimize[{meritFun,
2522         F0 > 0,
2523         F2 > 1000.,
2524         F4 == F4overF2 * F2,
2525         F6 == F6overF2 * F2,
2526         ζ > 0},
2527         freeIonParams,
2528         MaxIterations -> maxIterations,
2529         Method -> "Convex"];
2530       solutions[{subsetSize, subset}] = sol;
2531     )
2532     , {subsetIndex, 1, Length[subsets]}
2533   ],
2534   {subsetSize, chosenSubsetSizes}
2535 ];
2536
2537 PrintFun["Collecting solutions of different subset size ..."];
2538 goodThings = Table[
2539   (chunk =
2540    Normal[Sort[
2541      KeySelect #[[1]] & /@ solutions, #[[1]] == subSize &]];
2542    If[Length[chunk] >= 1,
2543      chunk[[1]],
2544      Null
2545    ]), {subSize, subsetSizes}];
2546 goodThings = Select[goodThings, Head[#] === Rule &];
2547
2548 PrintFun["Picking the solutions that are not worse than ",
2549 maxPercent, "% ..."];
2550 bestThings = Select[goodThings, #[[2]] <= maxPercent &];
2551 If[bestThings == {},
2552   Print["No acceptable solution found, consider increasing
maxPercent or inspecting the given data ..."];
2553   Return[goodThings];
2554 ];
2555
2556 PrintFun["Keeping the solution with the largest number of used
barycenters ..."];
2557 bestOfAll = bestThings[[-1]];
2558 sol = solutions[[bestOfAll[[1]]]];
2559 subset = bestOfAll[[1, 2]];
2560 eqns = allEqns[[subset]];
2561 slope = #[[;; 5]] & /@ eqns;
2562 intercept = #[[6]] & /@ eqns;
2563 usedMultiplets = Keys[multipletEnergies][[subset]];
2564 usedBaricenters = {#, multipletEnergies[#]} & /@ usedMultiplets;
2565 uniqueLS = DeleteDuplicates[#[[2]] &/@ Keys[multipletEnergies]];
2566 solAssoc = Association[sol[[2]]];

```

```

2545 usedLaF3 = False;
2546 If[Length[uniqueLS] == 1,
2547 (
2548     Print["There is too little data to find Slater parameters,
2549     using the ones for LaF3, and keeping the fitted spin-orbit zeta
2550     ..."];
2551     laf3params = LoadLaF3Parameters[ln];
2552     usedLaF3 = True;
2553     solAssoc[F0] = laf3params[F0];
2554     solAssoc[F2] = laf3params[F2];
2555     solAssoc[F4] = laf3params[F4];
2556     solAssoc[F6] = laf3params[F6];
2557 )
2558 ];
2559 finalSol = <| "bestParams" -> solAssoc,
2560   "usedLaF3" -> usedLaF3,
2561   "worstRelativeError" -> sol[[1]],
2562   "SlaterRatios" -> {F4overF2, F6overF2},
2563   "usedBaricenters" -> usedBaricenters|>;
2564 Return[finalSol];
2565 )
2566 ];
2567 EndPackage[];
```

### 18.3 qplotter.m

This module has a few useful plotting routines.

```

1
2 BeginPackage["qplotter`"];
3
4 GetColor;
5 IndexMappingPlot;
6 ListLabelPlot;
7 AutoGraphicsGrid;
8 SpectrumPlot;
9 WaveToRGB;
10
11 Begin["`Private`"];
12
13 AutoGraphicsGrid::usage="AutoGraphicsGrid[graphsList] takes a list
14   of graphics and creates a GraphicsGrid with them. The number of
15   columns and rows is chosen automatically so that the grid has a
16   squarish shape.";
17 Options[AutoGraphicsGrid] = Options[GraphicsGrid];
18 AutoGraphicsGrid[graphsList_, opts : OptionsPattern[]] :=
19 (
20   numGraphs = Length[graphsList];
21   width = Floor[Sqrt[numGraphs]];
22   height = Ceiling[numGraphs/width];
23   groupedGraphs = Partition[graphsList, width, width, 1, Null];
24   GraphicsGrid[groupedGraphs, opts]
25 )
26
27 Options[IndexMappingPlot] = Options[Graphics];
28 IndexMappingPlot::usage =
29   "IndexMappingPlot[pairs] take a list of pairs of integers and
30   creates a visual representation of how they are paired. The first
31   indices being depicted in the bottom and the second indices being
32   depicted on top.";
33 IndexMappingPlot[pairs_, opts : OptionsPattern[]] := Module[{width,
34   height},
35   width = Max[First /@ pairs];
36   height = width/3;
37   Return[
38     Graphics[{{Tooltip[Point[{#[[1]], 0}], #[[1]]}, Tooltip[Point
39       [#[[2]], height], #[[2]]], Line[{{#[[1]], 0}, {#[[2]], height}}]} & /@ pairs, opts,
40     ImageSize -> 800]]
41   ]
42 ]
43
44 TickCompressor[fTicks_] :=
45 Module[{avgTicks, prevTickLabel, groupCounter, groupTally, idx,
```



```

102             }, #1] &, compTicks];
103             frameTicks = {{Automatic, Automatic}, {bottomTicks,
104 topTicks}});
105         ];
106         ListPlot[scatterGroups,
107           opts,
108           Frame -> True,
109           AxesStyle -> {Directive[Black, Dotted], Automatic},
110           PlotStyle -> groupedColors,
111           FrameTicks -> frameTicks]
112       )
113     ]
114 
WaveToRGB::usage="WaveToRGB[wave, gamma] takes a wavelength in nm
and returns the corresponding RGB color. The gamma parameter is
optional and defaults to 0.8. The wavelength wave is assumed to be
in nm. If the wavelength is below 380 the color will be the same
as for 380 nm. If the wavelength is above 750 the color will be
the same as for 750 nm. The function returns an RGBColor object.
REF: https://www.noah.org/wiki/wave\_to\_rgb\_in\_Python. ";
115 WaveToRGB[wave_, gamma_ : 0.8] := (
116   wavelength = (wave);
117   Which[
118     wavelength < 380,
119     wavelength = 380,
120     wavelength > 750,
121     wavelength = 750
122   ];
123   Which[380 <= wavelength <= 440,
124 (
125     attenuation = 0.3 + 0.7*(wavelength - 380)/(440 - 380);
126     R = ((-(wavelength - 440)/(440 - 380))*attenuation)^gamma;
127     G = 0.0;
128     B = (1.0*attenuation)^gamma;
129   ),
130   440 <= wavelength <= 490,
131   (
132     R = 0.0;
133     G = ((wavelength - 440)/(490 - 440))^gamma;
134     B = 1.0;
135   ),
136   490 <= wavelength <= 510,
137   (
138     R = 0.0;
139     G = 1.0;
140     B = (-(wavelength - 510)/(510 - 490))^gamma;
141   ),
142   510 <= wavelength <= 580,
143   (
144     R = ((wavelength - 510)/(580 - 510))^gamma;
145     G = 1.0;
146     B = 0.0;
147   ),
148   580 <= wavelength <= 645,
149   (
150     R = 1.0;
151     G = (-(wavelength - 645)/(645 - 580))^gamma;
152     B = 0.0;
153   ),
154   645 <= wavelength <= 750,
155   (
156     attenuation = 0.3 + 0.7*(750 - wavelength)/(750 - 645);
157     R = (1.0*attenuation)^gamma;
158     G = 0.0;
159     B = 0.0;
160   ),
161   True,
162   (
163     R = 0;
164     G = 0;
165     B = 0;
166   )];
167   Return[RGBColor[R, G, B]]
168 )
169 
FuzzyRectangle::usage = "FuzzyRectangle[xCenter, width, ymin,
170

```

```

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."];
171 FuzzyRectangle[xCenter_, width_, ymin_, height_, color_, intensity_-
:1] := Module[
172 {intenseColor, nocolor, ymax, polys},
173 (
174 nocolor = Directive[Opacity[0], color];
175 ymax = ymin + height;
176 intenseColor = Directive[Opacity[intensity], color];
177 polys = {
178 Polygon[{{
179 {xCenter - width/2, ymin},
180 {xCenter, ymin},
181 {xCenter, ymax},
182 {xCenter - width/2, ymax}},
183 VertexColors -> {
184 nocolor,
185 intenseColor,
186 intenseColor,
187 nocolor,
188 nocolor}],
189 Polygon[{{
190 {xCenter, ymin},
191 {xCenter + width/2, ymin},
192 {xCenter + width/2, ymax},
193 {xCenter, ymax}},
194 VertexColors -> {
195 intenseColor,
196 nocolor,
197 nocolor,
198 intenseColor,
199 intenseColor}]
200 }};
201 Return[polys]
202 );
203 ]
204
205 Options[SpectrumPlot] = Options[Graphics];
206 Options[SpectrumPlot] = Join[Options[SpectrumPlot], {"Intensities"-
-> {},"Tooltips" -> True, "Comments" -> {}, "SpectrumFunction" ->
WaveToRGB}];
207 SpectrumPlot::usage="SpectrumPlot[lines, widthToHeightAspect,
lineWidth] takes a list of spectral lines and creates a visual
representation of them. The lines are represented as fuzzy
rectangles with a width of lineWidth and a height that is
determined by the overall condition that the width to height ratio
of the resulting graph is widthToHeightAspect. The color of the
lines is determined by the wavelength of the line. The function
assumes that the lines are given in nm.
208 If the lineWidth parameter is a single number, then every line
shares that width. If the lineWidth parameter is a list of numbers
, then each line has a different width. The function returns a
Graphics object. The function also accepts any options that
Graphics accepts. The background of the plot is black by default.
The plot range is set to the minimum and maximum wavelength of the
given lines.
209 Besides the options for Graphics the function also admits the
option Intensities. This option is a list of numbers that
determines the intensity of each line. If the Intensities option
is not given, then the lines are drawn with full intensity. If the
Intensities option is given, then the lines are drawn with the
given intensity. The intensity is a number between 0 and 1.
210 The function also admits the option \"Tooltips\". If this option is
set to True, then the lines will have a tooltip that shows the
wavelength of the line. If this option is set to False, then the
lines will not have a tooltip. The default value for this option
is True.
211 If \"Tooltips\" is set to True and the option \"Comments\" is a non
-empty list, then the tooltip will append the wavelength and the
values in the comments list for the tooltips.
212 The function also admits the option \"SpectrumFunction\". This
option is a function that takes a wavelength and returns a color.

```

```

    The default value for this option is WaveToRGB.
213   ";
214   SpectrumPlot[lines_, widthToHeightAspect_, lineWidth_, opts : OptionsPattern[]] := Module[
215     {minWave, maxWave, height, fuzzyLines},
216     (
217       colorFun = OptionValue["SpectrumFunction"];
218       {minWave, maxWave} = MinMax[lines];
219       height = (maxWave - minWave)/widthToHeightAspect;
220       fuzzyLines = Which[
221         NumberQ[lineWidth] && Length[OptionValue["Intensities"]] == 0,
222           FuzzyRectangle[#, lineWidth, 0, height, colorFun[#]] &/@ lines,
223           Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]] ==
224             0,
225               MapThread[FuzzyRectangle[#, #2, 0, height, colorFun[#1]] &,
226             {lines, lineWidth}],
227               NumberQ[lineWidth] && Length[OptionValue["Intensities"]] > 0,
228                 MapThread[FuzzyRectangle[#, lineWidth, 0, height, colorFun
229                   [#1], #2] &, {lines, OptionValue["Intensities"]}],
230                   Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]] >
231                     0,
232                       MapThread[FuzzyRectangle[#, #2, 0, height, colorFun[#1], #3]
233                         &, {lines, lineWidth, OptionValue["Intensities"]}]
234                     ];
235       comments = Which[
236         Length[OptionValue["Comments"]] > 0,
237           MapThread[StringJoin[ToString[#1]<>" nm", "\n", ToString[#2]]&,
238             {lines, OptionValue["Comments"]}],
239             Length[OptionValue["Comments"]] == 0,
240               ToString[#] <>" nm" & /@ lines,
241               True,
242               {}
243             ];
244       If[OptionValue["Tooltips"],
245         fuzzyLines = MapThread[Tooltip[#1, #2] &, {fuzzyLines, comments
246             }];
247       ];
248     ]
249   ];
250 End[];
251
252 EndPackage[];

```

## 18.4 misc.m

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

```

1 BeginPackage["misc`"];
2 (* Needs["MaTeX`"]; *)
3
4 ArrayBlocker;
5 BlockAndIndex;
6 BlockArrayDimensionsArray;
7 BlockMatrixMultiply;
8 BlockTranspose;
9
10 EllipsoidBoundingBox;
11 ExportToH5;
12 FirstOrderPerturbation;
13 FlattenBasis;
14
15 GetModificationDate;
16 HamTeX;
17 HelperNotebook;
18
19 RecoverBasis;
20 RemoveTrailingDigits;
21 ReplaceDiagonal;
22 RobustMissingQ;

```

```

23 RoundToSignificantFigures;
24 RoundValueWithUncertainty;
25 SecondOrderPerturbation;
26 SuperIdentity;
27
28 TextBasedProgressBar;
29 ToPythonSparseFunction;
30 ToPythonSymPyExpression;
31 TruncateBlockArray;
32
33 Begin["`Private`"];
34
35 ExtractSymbolNames;
36
37 RemoveTrailingDigits[s_String] := StringReplace[s,
38   RegularExpression["\d+"] -> ""];
39
40 BlockTranspose::usage="BlcockTranspose[array] takes a 2D array
41   with a congruent block structure and returns the transposed array
42   with the same block structure.";
43 BlockTranspose[array_]:=(
44   Map[Transpose, Transpose[array], {2}]
45 );
46
47 BlockMatrixMultiply::usage="BlockMatrixMultiply[A,B] gives the
48   matrix multiplication of A and B, with A and B having a compatible
49   block structure that allows for matrix multiplication into a
50   congruent block structure.";
51 BlockMatrixMultiply[amat_,bmat_]:=Module[{rowIdx,colIdx,sumIdx},
52 (
53   Table[
54     Sum[amat[[rowIdx,sumIdx]].bmat[[sumIdx,colIdx]],{sumIdx,1,
55       Dimensions[amat][[2]]}],
56     {rowIdx,1,Dimensions[amat][[1]]},
57     {colIdx,1,Dimensions[bmat][[2]]}
58   ]
59 )
59 ];
59
60 BlockAndIndex::usage="BlockAndIndex[{w1, w2, ...}, idx] takes a
61   list of block lengths wi and an index idx. The function returns in
62   which block idx would be in a a list defined by {Range[1,w1],
63   Range[1+w1,w1+w2], ...}. The function also returns the position
64   within the bin in which the given index would be found in. The
65   function returns these two numbers as a list of two elements {blockIndex,
66   blockSubIndex}.";
66 BlockAndIndex[bblockSizes_List, index_Integer]:=Module[{
67   accumulatedBlockSize,blockIndex, blockSubIndex},
68 (
69   accumulatedBlockSize = Accumulate[blockSizes];
70   If[accumulatedBlockSize[[-1]]-index<0,
71     Print["Index out of bounds"];
72     Abort[]
73   ];
74   blockIndex = Flatten[Position[accumulatedBlockSize-index,n_ /;
75     n>=0]][[1]];
76   blockSubIndex = Mod[index-accumulatedBlockSize[[blockIndex]],
77   blockSizes[[blockIndex]],1];
78   Return[{blockIndex,blockSubIndex}]
79 )
79 ];
79
80 TruncateBlockArray::usage="TruncateBlockArray[blockArray,
81   truncationIndices, blockWidths] takes an array of blocks and
82   selects the columns and rows corresponding to truncationIndices.
83   The indices being given in what would be the ArrayFlatten[
84   blockArray] version of the array. The blocks in the given array
85   may be SparseArray. This is equivalent to FlattenArray[blockArray
86   ][truncationIndices, truncationIndices] but may be more efficient
87   if blockArray is sparse.";
87 TruncateBlockArray[blockArray_,truncationIndices_,blockWidths_]:=(
88   Module[{
89     truncatedArray,blockCol,blockRow,blockSubCol,blockSubRow},(
90     truncatedArray = Table[
91       {blockCol,blockSubCol} = BlockAndIndex[blockWidths,fullColIndex];
92     ]
92   ]
92 ];
92

```

```

75 {blockRow,blockSubRow} = BlockAndIndex[blockWidths,fullRowIndex];
76 blockArray[[blockRow,blockCol]][[blockSubRow,blockSubCol]],
77 {fullRowIndex,truncationIndices},
78 {fullColIndex,truncationIndices}
79 ];
80 Return[truncatedArray]
81 )
82 ];
83
84 BlockArrayDimensionsArray::usage="BlockArrayDimensionsArray[
85   blockArray] returns the array of block sizes in a given blocked
86   array.";
87 BlockArrayDimensionsArray[blockArray_]:=(
88   Map[Dimensions,blockArray,{2}]
89 );
90
91 ArrayBlocker::usage="ArrayBlocker[anArray, blockSizes] takes a flat
92   2d array and a congruent 2D array of block sizes, and with them
93   it returns the original array with the block structure imposed by
94   blockSizes. The resulting array satisfies ArrayFlatten[
95   blockedArray]==anArray, and also Map[Dimensions, blockedArray
96   ,{2}]==blockSizes.";
97 ArrayBlocker[anArray_, blockSizes_] := Module[{rowStart,colStart,
98   colEnd,numBlocks,blockedArray,blockSize,rowEnd,aBlock,idxRow,
99   idxCol},(
100   rowStart = 1;
101   colStart = 1;
102   colEnd = 1;
103   case = Length[Dimensions[blockSizes]];
104   Which[
105     case == 3,
106     (
107       numBlocks = Length[blockSizes];
108       blockedArray = Table[(
109         blockSize = blockSizes[[idxRow, idxCol]];
110         rowEnd = rowStart+blockSize[[1]]-1;
111         colEnd = colStart+blockSize[[2]]-1;
112         aBlock = anArray[[rowStart;;rowEnd,colStart;;colEnd]];
113         colStart = colEnd+1;
114         If[idxCol==numBlocks,
115           rowStart=rowEnd+1;
116           colStart=1;
117           ];
118           aBlock
119         ),
120         {idxRow,1,numBlocks},
121         {idxCol,1,numBlocks}];
122       ),
123       case == 1,
124       (
125         expandedSizes = Table[
126           {blockSizes[[idxRow]], blockSizes[[idxCol]]},
127           {idxRow,1,Length[blockSizes]},
128           {idxCol,1,Length[blockSizes]}
129         ];
130         numBlocks = Length[expandedSizes];
131         blockedArray = Table[(
132           blockSize = expandedSizes[[idxRow, idxCol]];
133           rowEnd = rowStart+blockSize[[1]]-1;
134           colEnd = colStart+blockSize[[2]]-1;
135           aBlock = anArray[[rowStart;;rowEnd,colStart;;colEnd]];
136           colStart = colEnd+1;
137           If[idxCol==numBlocks,
138             rowStart=rowEnd+1;
139             colStart=1;
140             ];
141             aBlock
142           ),
143           {idxRow,1,numBlocks},
144           {idxCol,1,numBlocks}];
145         )
146       ];
147       Return[blockedArray]
148     )
149   ];
150 ];
151

```

```

142 ReplaceDiagonal::usage =
143   "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.";
144 ReplaceDiagonal[matrix_, repValue_] :=
145   ReplacePart[matrix,
146     Table[{i, i} -> repValue, {i, 1, Length[matrix]}]];
147
148 Options[RoundValueWithUncertainty] = {"SetPrecision" -> False};
149 RoundValueWithUncertainty::usage = "RoundValueWithUncertainty[x, dx]
given a number x together with an uncertainty dx this function
rounds x to the first significant figure of dx and also rounds dx
to have a single significant figure.
The returned value is a list with the form {roundedX, roundedDx}.
The option \"SetPrecision\" can be used to control whether the
Mathematica precision of x and dx is also set accordingly to these
rules, otherwise the rounded numbers still have the original
precision of the input values.
If the position of the first significant figure of x is after the
position of the first significant figure of dx, the function
returns {0,dx} with dx rounded to one significant figure.";
150 RoundValueWithUncertainty[x_, dx_, OptionsPattern[]] := Module[
151   {xExpo, dxExpo, sigFigs, roundedX, roundedDx, returning},
152   (
153     xExpo = RealDigits[x][[2]];
154     dxExpo = RealDigits[dx][[2]];
155     sigFigs = (xExpo - dxExpo) + 1;
156     {roundedX, roundedDx} = If[sigFigs <= 0,
157       {0., N@RoundToSignificantFigures[dx, 1]},
158       N[
159       {
160         RoundToSignificantFigures[x, xExpo - dxExpo + 1],
161         RoundToSignificantFigures[dx, 1]}
162       ];
163     ];
164     returning = If[
165       OptionValue["SetPrecision"],
166       {SetPrecision[roundedX, Max[1, sigFigs]],
167        SetPrecision[roundedDx, 1]},
168       {roundedX, roundedDx}
169     ];
170     Return[returning]
171   );
172 ];
173
174 RoundToSignificantFigures::usage =
175   "RoundToSignificantFigures[x, sigFigs] rounds x so that it only
has \
176   sigFigs significant figures.";
177 RoundToSignificantFigures[x_, sigFigs_] :=
178   Sign[x]*N[FromDigits[RealDigits[x, 10, sigFigs]]];
179
180 RobustMissingQ[expr_] := (FreeQ[expr, _Missing] === False);
181
182 TextBasedProgressBar[progress_, totalIterations_, prefix_:""] :=
183   Module[
184     {progMessage},
185     progMessage = ToString[progress] <> "/" <> ToString[
186       totalIterations];
187     If[progress < totalIterations,
188       WriteString["stdout", StringJoin[prefix, progMessage, "\r"]
189     ],
190       WriteString["stdout", StringJoin[prefix, progMessage, "\n"]
191     ];
192   ];
193
194 FirstOrderPerturbation::usage="Given the eigenVectors of a matrix A
(which doesn't need to be given) together with a corresponding
perturbation matrix perMatrix, this function calculates the first
derivative of the eigenvalues with respect to the scale factor of
the perturbation matrix. In the sense that the eigenvalues of the

```

```

matrix A +  $\beta$  perMatrix are to first order equal to  $\lambda_i + \Delta_i \beta$ , where the  $\Delta_i$  are the returned values. This assuming that the eigenvalues are non-degenerate.";
195 FirstOrderPerturbation[eigenVectors_ ,
196   perMatrix_] := (Chop@Diagonal[
197     Conjugate@eigenVectors . perMatrix . Transpose[eigenVectors]])
```

SecondOrderPerturbation::usage="Given the eigenValues and eigenVectors of a matrix A (which doesn't need to be given) together with a corresponding perturbation matrix perMatrix, this function calculates the second derivative of the eigenvalues with respect to the scale factor of the perturbation matrix. In the sense that the eigenvalues of the matrix A +  $\beta$  perMatrix are to second order equal to  $\lambda_i + \Delta_i \beta + \frac{1}{2} \Delta_i^2 \beta^2$ , 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.";

```

200 SecondOrderPerturbation[eigenValues_, eigenVectors_, perMatrix_] :=
201   (
202     dim = Length[perMatrix];
203     eigenBras = Conjugate[eigenVectors];
204     eigenKets = eigenVectors;
205     matV = Abs[eigenBras . perMatrix . Transpose[eigenKets]]^2;
206     OneOver[x_, y_] := If[x == y, 0, 1/(x - y)];
207     eigenDiff = Outer[OneOver, eigenValues, eigenValues, 1];
208     pProduct = Transpose[eigenDiff] * matV;
209     Return[2*(Total /@ Transpose[pProduct])];
210   )
211
212 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.";
```

```

213 SuperIdentity[args___] := {args};
214
215 ExtractSymbolNames[expr_Hold] := Module[
216   {strSymbols},
217   strSymbols = ToString[expr, InputForm];
218   StringCases[strSymbols, RegularExpression["\\w+"]][[2 ;;]]
219 ]
220
221 ExportToH5::usage =
222   "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 which mixes numbers and strings will fail, but an Association with mixed values exports ok. Do give it a try.
223   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.";
```

```

224 Options[ExportToH5] = {"Overwrite" -> True};
225 ExportToH5[fname_String, symbols_Hold, OptionsPattern[]] :=
226   (
227     If[And[FileExistsQ[fname], OptionValue["Overwrite"]],
228       (
229         Print["File already exists, overwriting ..."];
230         DeleteFile[fname];
231       )
232     ];
233     symbolNames = ExtractSymbolNames[symbols];
234     Do[(Print[symbolName];
235       Export[fname, ToExpression[symbolName], {"Datasets", symbolName}],
236       OverwriteTarget -> "Append"]
237     ), {symbolName, symbolNames}]
238   )
239
240 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.";
```

```

241 HelperNotebook[nbName_: "OUT"] :=
242   Module[{screenDims, screenWidth, screenHeight, nbWidth, leftMargin,
243     PrintToOutputNb}, (
244     screenDims =
```

```

243     SystemInformation["Devices", "ScreenInformation"][[1, 2, 2]];
244     screenWidth = screenDims[[1, 2]];
245     screenHeight = screenDims[[2, 2]];
246     nbWidth = Round[screenWidth/3];
247     leftMargin = screenWidth - nbWidth;
248     outputNb = CreateDocument[{}, WindowTitle -> nbName,
249                               WindowMargins -> {{leftMargin, Automatic}, {Automatic,
250                                         Automatic}},WindowSize -> {nbWidth, screenHeight}];
251     PrintToOutputNb[text_] :=
252     (
253       SelectionMove[outputNb, After, Notebook];
254       NotebookWrite[outputNb, Cell[BoxData[ToBoxes[text]], "Output"]];
255     );
256     Return[PrintToOutputNb]
257   )
258 ]
259
260 GetModificationDate::usage="GetModificationDate[fname] returns the
261   modification date of the given file.";
262 GetModificationDate[theFileName_] := FileDate[theFileName, "Modification"];
263
264 (*Helper function to convert Mathematica expressions to standard
265   form*)
266 StandardFormExpression[expr0_] := Module[{expr=expr0}, ToString[
267   expr, InputForm]];
268
269 (*Helper function to translate to Python/Sympy expressions*)
270 ToPythonSymPyExpression::usage="ToPythonSymPyExpression[expr]
271   converts a Mathematica expression to a SymPy expression. This is a
272   little iffy and might break if the expression includes
273   Mathematica functions that haven't been given a SymPy equivalent."
274 ;
275 ToPythonSymPyExpression[expr0_] := Module[{standardForm, expr=expr0},
276   standardForm = StandardFormExpression[expr];
277   StringReplace[standardForm, {
278     "Power[" -> "Pow(",
279     "Sqrt[" -> "sq(",
280     "[" -> "(",
281     "]" -> ")",
282     "\\\" -> """",
283     "I" -> "1j",
284     "^" -> "**",
285     (*Remove special Mathematica backslashes*)
286     "/" -> "/" (*Ensure division is represented with a slash*)}]];
287
288 ToPythonSparseFunction::usage="ToPythonSparseFunction[array,
289   funName] takes a 2D array (whose elements are linear combinations
290   of a set of variables) and returns a string which defines a Python
291   function that can be used to evaluate the given array in Python.
292   In Python the name of the function is equal to funName. The given
293   array may be a list of lists or a 2D SparseArray.";
294 ToPythonSparseFunction[arrayInput_, funName_] :=
295   Module[{{
296     sparseArray = arrayInput,
297     rowPointers,
298     dimensions,
299     pyCode,
300     vars,
301     varList,
302     dataPyList,
303     colIndicesPyList
304   },
305   (
306     If[Head[sparseArray] === List,
307       sparseArray = SparseArray[sparseArray]
308     ];
309     (* Extract unique symbolic variables from the SparseArray *)
310     vars      = Union[Cases[Normal[sparseArray], _Symbol, Infinity
311 ]];
312     varList   = StringRiffle[ToString /@ vars, ", "];
313     (* Convert data to SymPy compatible strings *)
314     dataPyList = StringRiffle[ToPythonSymPyExpression /@ Normal[
315       sparseArray["NonzeroValues"]], ",\n          "];

```

```

302     colIndicesPyList = StringRiffle[
303       ToPythonSymPyExpression /@ (Flatten[Normal[sparseArray["  

304       ColumnIndices"]]] - 1]), ", "];
305       (* Extract sparse array properties *)
306       rowPointers = Normal[sparseArray["RowPointers"]];
307       dimensions = Dimensions[sparseArray];
308       (*Create Python code string*)
309       pyCode = StringJoin[
310         "#!/usr/bin/env python3\n\n",
311         "from scipy.sparse import csr_matrix\n",
312         "import numpy as np\n",
313         "\n",
314         "sq = np.sqrt\n",
315         "\n",
316         "def ", funName, "(",
317         varList,
318         "):\n",
319         "    data = np.array([\n", dataPyList, "\n        ])\n",
320         "    indices = np.array([",
321         colIndicesPyList,
322         "])\n",
323         "    indptr = np.array([
324           StringRiffle[ToString /@ rowPointers, ", ", "], "])\n",
325         "    shape = (" , StringRiffle[ToString /@ dimensions, ", ", "],
326         ")\n",
327         "    return csr_matrix((data, indices, indptr), shape=shape)\n",
328       ];
329       Return[pyCode];
330     ];
331   ];
332 Options[HamTeX] = {"T2" -> False};
333 HamTeX::usage="HamTeX[numE] returns the LaTeX code for the semi-
334 empirical Hamiltonian for f^numE. The option \"T2\" can be used to
335 specify whether the T2 term should be included in the Hamiltonian
336 for the f^12 configuration. The default is False and the option
337 is ignored if the number of electrons is not 12.";
338 HamTeX[nE_, OptionsPattern[]] := (
339   tex = Which[
340     MemberQ[{1, 13}, nE],
341       "\hat{H}=\zeta \sum_{i=1}^n \left( \hat{s}_i \cdot \hat{l}_i + \sum_{i=1}^n \sum_{k=2,4,6} \sum_{q=-k}^k B_q^{(k)} \mathcal{C}(i)_q + \epsilon \right)",
342     nE == 2,
343       "\hat{H}=\sum_{k=2,4,6} F^{(k)} \hat{f}_k + \alpha \hat{L}^2 + \beta \mathcal{C} \left( \mathcal{G}(2) \right) + \gamma \mathcal{C} \left( \mathcal{S}(7) \right) + \quad + \zeta \sum_{i=1}^n \left( \hat{s}_i \cdot \hat{l}_i + \sum_{k=0,2,4} M^{(k)} \hat{m}_k + \sum_{k=2,4,6} P^{(k)} \hat{p}_k \right) + \quad + \sum_{q=-k}^k B_q^{(k)} \mathcal{C}(i)_q + \epsilon",
344     And[nE == 12, OptionValue["T2"]],
345       "\hat{H}=\sum_{k=2,4,6} F^{(k)} \hat{f}_k + T^{(2)} \hat{t}_2 + \alpha \hat{L}^2 + \beta \mathcal{C} \left( \mathcal{G}(2) \right) + \gamma \mathcal{C} \left( \mathcal{S}(7) \right) + \quad + \zeta \sum_{i=1}^n \left( \hat{s}_i \cdot \hat{l}_i + \sum_{k=0,2,4} M^{(k)} \hat{m}_k + \sum_{k=2,4,6} P^{(k)} \hat{p}_k \right) + \quad + \sum_{q=-k}^k B_q^{(k)} \mathcal{C}(i)_q + \epsilon",
346     And[nE == 12, Not@OptionValue["T2"]],
347       "\hat{H}=\sum_{k=2,4,6} F^{(k)} \hat{f}_k + \alpha \hat{L}^2 + \quad + \sum_{q=-k}^k B_q^{(k)} \mathcal{C}(i)_q + \epsilon"
348   ]

```

```

366      +\\beta \\\,,\\mathcal{C}\\left(\\mathcal{G}(2)\\right)
367      +\\gamma \\\,,\\mathcal{C}\\left(\\mathcal{S}_0(7)\\right)\\\\\\
368      &\\quad + \\zeta \\sum_{i=1}^n\\left(\\hat{s}_i \\cdot \\
369      \\hat{l}_i\\right)
370      +\\sum_{k=0,2,4}M^{(k)}\\hat{m}_k
371      +\\sum_{k=2,4,6}P^{(k)}\\hat{p}_k \\\\\\
372      &\\quad\\quad\\quad+\\sum_{i=1}^n\\sum_{k=2,4,6}\\sum_{q=-k}
373      }^k B_q^{(\\
374      k)}\\mathcal{C}(i)_q^{(k)} + \\epsilon,
375      True,
376      "\\hat{H}&=\\sum_{k=2,4,6}F^{(k)}\\hat{f}_k
377      +\\sum_{k=2,3,4,6,7,8}T^{(k)}\\hat{t}_k
378      +\\alpha \\hat{L}^2
379      +\\beta \\\,,\\mathcal{C}\\left(\\mathcal{G}(2)\\right)
380      +\\gamma \\\,,\\mathcal{C}\\left(\\mathcal{S}_0(7)\\right)\\\\\\
381      &\\quad\\quad\\quad+\\quad\\quad\\quad+\\zeta \\sum_{i=1}^n\\left(\\
382      \\hat{s}_i \\
383      \\cdot \\hat{l}_i\\right)
384      +\\sum_{k=0,2,4}M^{(k)}\\hat{m}_k
385      +\\sum_{k=2,4,6}P^{(k)}\\hat{p}_k \\\\\\
386      &\\quad\\quad\\quad+\\quad\\quad\\quad+\\quad\\quad\\quad+\\quad\\quad\\quad+\\sum_{i=1}^n\\sum_{k=2,4,6}\\sum_{q=-k}
387      }^k B_q^{(k)}\\mathcal{C}(i)_q^{(k)} + \\epsilon"
388      ];
389
390 EllipsoidBoundingBox::usage = "EllipsoidBoundingBox[A,\\[Kappa]]
391 gives the coordinate intervals that contain the ellipsoid
392 determined by r^T.A.r==\\[Kappa]^2. The matrix A must be square NxN
393 , symmetric, and positive definite. The function returns a list
394 with N pairs of numbers, each pair being of the form {-x_i, x_i}."
395 ;
396
397 EllipsoidBoundingBox[Amat_,\\[Kappa]]:=Module[
398   {invAmat, stretchFactors, boundingPlanes, quad},
399   (
400     invAmat = Inverse[Amat];
401     stretchFactors = Sqrt[1/Diagonal[invAmat]];
402     boundingPlanes = DiagonalMatrix[stretchFactors].invAmat;
403     (* The solution is proportional to \\[Kappa] *)
404     boundingPlanes = \\[Kappa] * boundingPlanes;
405     boundingPlanes = Max /@ Transpose[boundingPlanes];
406     Return[{-#, #}& /@ boundingPlanes]
407   )
408 ];
409
410 End[] ;
411 EndPackage[] ;

```

## References

- [BG15] Cristiano Benelli and Dante Gatteschi. *Introduction to molecular magnetism: from transition metals to lanthanides*. John Wiley & Sons, 2015.
- [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.
- [BS57] Hans Bethe and Edwin Salpeter. *Quantum Mechanics of One- and Two-Electron Atoms*. 1957.
- [Car+70] WT Carnall et al. “Absorption spectrum of Tm<sup>3+</sup>: LaF<sub>3</sub>”. In: *The Journal of Chemical Physics* 52.8 (1970). Publisher: American Institute of Physics, pp. 4054–4059.
- [Car+76] WT Carnall et al. “Energy level analysis of Pm<sup>3+</sup>: LaCl<sub>3</sub>”. In: *The Journal of Chemical Physics* 64.9 (1976). Publisher: American Institute of Physics, pp. 3582–3591.
- [Car+89] W. T. Carnall et al. “A systematic analysis of the spectra of the lanthanides doped into single crystal LaF<sub>3</sub>”. en. In: *The Journal of Chemical Physics* 90.7 (1989), pp. 3443–3457. ISSN: 0021-9606, 1089-7690.
- [Car92] William T Carnall. “A systematic analysis of the spectra of trivalent actinide chlorides in D3h site symmetry”. In: *The Journal of chemical physics* 96.12 (1992). Publisher: American Institute of Physics, pp. 8713–8726.
- [CCJ68] Hannah Crosswhite, HM Crosswhite, and BR Judd. “Magnetic Parameters for the Configuration f 3”. In: *Physical Review* 174.1 (1968). Publisher: APS, p. 89.
- [CFR68a] WT Carnall, PR Fields, and K Rajnak. “Electronic energy levels in the trivalent lanthanide aquo ions. I. Pr<sup>3+</sup>, Nd<sup>3+</sup>, Pm<sup>3+</sup>, Sm<sup>3+</sup>, Dy<sup>3+</sup>, Ho<sup>3+</sup>, Er<sup>3+</sup>, and Tm<sup>3+</sup>”. In: *The Journal of chemical physics* 49.10 (1968). Publisher: American Institute of Physics, pp. 4424–4442.
- [CFR68b] WT Carnall, PR Fields, and K Rajnak. “Electronic energy levels of the trivalent lanthanide aquo ions. IV. Eu<sup>3+</sup>”. In: *The Journal of Chemical Physics* 49.10 (1968). Publisher: American Institute of Physics, pp. 4450–4455.
- [CFR68c] WT Carnall, PR Fields, and K Rajnak. “Electronic energy levels of the trivalent lanthanide aquo ions. III. Tb<sup>3+</sup>”. In: *The Journal of chemical physics* 49.10 (1968). Publisher: American Institute of Physics, pp. 4447–4449.
- [CFR68d] WT Carnall, PR Fields, and K Rajnak. “Electronic energy levels of the trivalent lanthanide aquo ions. II. Gd<sup>3+</sup>”. In: *The Journal of chemical physics* 49.10 (1968). Publisher: American Institute of Physics, pp. 4443–4446.
- [CFR68e] WT Carnall, PR Fields, and K Rajnak. “Spectral intensities of the trivalent lanthanides and actinides in solution. II. Pm<sup>3+</sup>, Sm<sup>3+</sup>, Eu<sup>3+</sup>, Gd<sup>3+</sup>, Tb<sup>3+</sup>, Dy<sup>3+</sup>, and Ho<sup>3+</sup>”. In: *The journal of chemical physics* 49.10 (1968). Publisher: American Institute of Physics, pp. 4412–4423.
- [CFW65] W To Carnall, PR Fields, and BG Wybourne. “Spectral intensities of the trivalent lanthanides and actinides in solution. I. Pr<sup>3+</sup>, Nd<sup>3+</sup>, Er<sup>3+</sup>, Tm<sup>3+</sup>, and Yb<sup>3+</sup>”. In: *The Journal of Chemical Physics* 42.11 (1965). Publisher: American Institute of Physics, pp. 3797–3806.
- [Che+08] Xueyuan Chen et al. “A few mistakes in widely used data files for fn configurations calculations”. In: *Journal of luminescence* 128.3 (2008). Publisher: Elsevier, pp. 421–427.
- [Che+16] Jun Cheng et al. “Crystal-field analyses for trivalent lanthanide ions in LiYF<sub>4</sub>”. In: *Journal of Rare Earths* 34.10 (2016). Publisher: Elsevier, pp. 1048–1052.
- [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.
- [Cro+76] HM Crosswhite et al. “The spectrum of Nd<sup>3+</sup>: LaCl<sub>3</sub>”. In: *The Journal of Chemical Physics* 64.5 (1976). Publisher: American Institute of Physics, pp. 1981–1985.

- [Cro+77] HM Crosswhite et al. “Parametric energy level analysis of Ho<sup>3+</sup>: LaCl<sub>3</sub>”. In: *The Journal of Chemical Physics* 67.7 (1977). Publisher: American Institute of Physics, pp. 3002–3010.
- [Cro71] HM Crosswhite. “Effective electrostatic operators for two inequivalent electrons”. In: *Physical Review A* 4.2 (1971). Publisher: APS, p. 485.
- [CW63] JG Conway and BG Wybourne. “Low-lying energy levels of lanthanide atoms and intermediate coupling”. In: *Physical Review* 130.6 (1963). Publisher: APS, p. 2325.
- [DC63] G. H. Dieke and H. M. Crosswhite. “The Spectra of the Doubly and Triply Ionized Rare Earths”. en. In: *Applied Optics* 2.7 (July 1963), p. 675. ISSN: 0003-6935, 1539-4522.
- [Die68] G. H. Dieke. *Spectra and Energy Levels of Rare Earth Ions in Crystals*. Ed. by Hannah Crosswhite and H. M. Crosswhite. 1968.
- [DR06] Chang-Kui Duan and Michael F Reid. “Dependence of the spontaneous emission rates of emitters on the refractive index of the surrounding media”. In: *Journal of alloys and compounds* 418.1-2 (2006). Publisher: Elsevier, pp. 213–216.
- [DZ12] Christopher M. Dodson and Rashid Zia. “Magnetic dipole and electric quadrupole transitions in the trivalent lanthanide series: Calculated emission rates and oscillator strengths”. en. In: *Physical Review B* 86.12 (Sept. 2012), p. 125102. ISSN: 1098-0121, 1550-235X.
- [GW+91] C Görller-Walrand et al. “Magnetic dipole transitions as standards for Judd–Ofelt parametrization in lanthanide spectra”. In: *The Journal of chemical physics* 95.5 (1991). Publisher: American Institute of Physics, pp. 3099–3106.
- [JC84] BR Judd and Hannah Crosswhite. “Orthogonalized operators for the f shell”. In: *JOSA B* 1.2 (1984). Publisher: Optica Publishing Group, pp. 255–260.
- [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.
- [JL93] BR Judd and GMS Lister. “Symmetries of the f shell”. In: *Journal of alloys and compounds* 193.1-2 (1993). Publisher: Elsevier, pp. 155–159.
- [JS84] BR Judd and MA Suskin. “Complete set of orthogonal scalar operators for the configuration f<sup>7</sup>3”. In: *JOSA B* 1.2 (1984). Publisher: Optica Publishing Group, pp. 261–265.
- [Jud05] Brian R Judd. “Interaction with William Carnall”. In: *Journal of Solid State Chemistry* 178.2 (2005). Publisher: Elsevier, pp. 408–411.
- [Jud62] B. R. Judd. “Optical Absorption Intensities of Rare-Earth Ions”. en. In: *Physical Review* 127.3 (Aug. 1962), pp. 750–761. ISSN: 0031-899X.
- [Jud63a] B R Judd. “Configuration Interaction in Rare Earth Ions”. en. In: *Proceedings of the Physical Society* 82.6 (Dec. 1963), pp. 874–881. ISSN: 0370-1328.
- [Jud63b] Brian R. Judd. *Operator techniques in atomic spectroscopy*. en. Princeton landmarks in mathematics and physics. Princeton, N.J: Princeton University Press, 1963. ISBN: 978-0-691-05901-3.
- [Jud66] BR Judd. “Three-particle operators for equivalent electrons”. In: *Physical Review* 141.1 (1966). Publisher: APS, p. 4.
- [Jud67] Brian R Judd. *Second quantization and atomic spectroscopy*. 1967.
- [Jud82] BR Judd. “Parametric fits in the atomic d shell”. In: *Journal of Physics B: Atomic and Molecular Physics* 15.10 (1982). Publisher: IOP Publishing, p. 1457.
- [Jud83] BR Judd. “Operator averages and orthogonalities”. In: *Group Theoretical Methods in Physics: Proceedings of the XIIth International Colloquium Held at the International Centre for Theoretical Physics, Trieste, Italy, September 5–11, 1983*. Springer, 1983, pp. 340–342.
- [Jud85] BR Judd. “Complex atomic spectra”. In: *Reports on Progress in Physics* 48.7 (1985). Publisher: IOP Publishing, p. 907.

- [Jud86] BR Judd. “Classification of Operators in Atomic Spectroscopy by Lie Groups”. In: *Symmetries in Science II*. Springer, 1986, pp. 265–269.
- [Jud88] BR Judd. “Atomic theory and optical spectroscopy”. In: *Handbook on the physics and chemistry of rare earths* 11 (1988). Publisher: Elsevier, pp. 81–195.
- [Jud89] BR Judd. “Developments in the Theory of Complex Spectra”. In: *Physica Scripta* 1989.T26 (1989). Publisher: IOP Publishing, p. 29.
- [Jud96] Brian R Judd. “Group Theory for atomic shells”. In: *Springer Handbook of Atomic, Molecular, and Optical Physics*. Springer, 1996, pp. 71–80.
- [Lea82] Richard P. Leavitt. “On the role of certain rotational invariants in crystal-field theory”. en. In: *The Journal of Chemical Physics* 77.4 (Aug. 1982), pp. 1661–1663. ISSN: 0021-9606, 1089-7690.
- [Lea87] RC Leavitt. “A complete set of f-electron scalar operators”. In: *Journal of Physics A: Mathematical and General* 20.11 (1987). Publisher: IOP Publishing, p. 3171.
- [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.
- [LM80] RP Leavitt and CA Morrison. “Crystal-field analysis of triply ionized rare earth ions in lanthanum trifluoride. II. Intensity calculations”. In: *The Journal of Chemical Physics* 73.2 (1980). Publisher: American Institute of Physics, pp. 749–757.
- [MKW77a] Clyde A Morrison, Nick Karayianis, and Donald E Wortman. *Rare-Earth Ion-Host Lattice Interactions. 4. Predicting Spectra and Intensities of Lanthanides in Crystals*. Tech. rep. HARRY DIAMOND LABS ADELPHI MD, 1977.
- [MKW77b] Clyde A Morrison, Nick Karayianis, and Donald E Wortman. *Theoretical Free-Ion Energies, Derivatives and Reduced Matrix Elements I. Pr (3+), Tm (3+), Nd (3+), and Er (3+)*. Tech. rep. HARRY DIAMOND LABS ADELPHI MD, 1977.
- [ML79] CA Morrison and RP Leavitt. “Crystal-field analysis of triply ionized rare earth ions in lanthanum trifluoride”. In: *The Journal of Chemical Physics* 71.6 (1979). Publisher: American Institute of Physics, pp. 2366–2374.
- [ML82] Clyde A Morrison and Richard P Leavitt. “Spectroscopic properties of triply ionized”. In: *Handbook on the physics and chemistry of rare earths* 5 (1982). Publisher: Elsevier, pp. 461–692.
- [Mor+83] Clyde A Morrison et al. “Optical spectra, energy levels, and crystal-field analysis of tripositive rare-earth ions in Y<sub>2</sub>O<sub>3</sub>. III. Intensities and g values for C<sub>2</sub> sites”. In: *The Journal of chemical physics* 79.10 (1983). Publisher: American Institute of Physics, pp. 4758–4763.
- [Mor80] Clyde Morrison. “Host dependence of the rare-earth ion energy separation 4f N–4f N–1 nl”. In: *The Journal of Chemical Physics* (1980).
- [MT87] Clyde Morrison and Gregory Turner. *Analysis of the Optical Spectra of Triply Ionized Transition Metal Ions in Yttrium Aluminum Garnet*. Tech. rep. 1987.
- [MW94] CA Morrison and DE Wortman. *Energy Levels, Transition Probabilities, and Branching Ratios for Rare-Earth Ions in Transparent Solids*. SPIE Optical Engineering Press, 1994.
- [MWK76] CA Morrison, DE Wortman, and N Karayianis. “Crystal-field parameters for triply-ionized lanthanides in yttrium aluminium garnet”. In: *Journal of Physics C: Solid State Physics* 9.8 (1976). Publisher: IOP Publishing, p. L191.
- [New82] DJ Newman. “Operator orthogonality and parameter uncertainty”. In: *Physics Letters A* 92.4 (1982). Publisher: Elsevier, pp. 167–169.
- [NK63] C. W. Nielson and George F Koster. *Spectroscopic Coefficients for the pn, dn, and fn configurations*. 1963.

- [Ofe62] GS Ofelt. “Intensities of crystal spectra of rare-earth ions”. In: *The journal of chemical physics* 37.3 (1962). Publisher: American Institute of Physics, pp. 511–520.
- [PDC67] AH Piksis, GH Dieke, and HM Crosswhite. “Energy levels and crystal field of LaCl<sub>3</sub>: Gd<sup>3+</sup>”. In: *The Journal of Chemical Physics* 47.12 (1967). Publisher: American Institute of Physics, pp. 5083–5089.
- [Rac42a] Giulio Racah. “Theory of Complex Spectra. I”. en. In: *Physical Review* 61.3-4 (Feb. 1942), pp. 186–197. ISSN: 0031-899X.
- [Rac42b] Giulio Racah. “Theory of Complex Spectra. II”. en. In: *Physical Review* 62.9-10 (Nov. 1942), pp. 438–462. ISSN: 0031-899X.
- [Rac43] Giulio Racah. “Theory of Complex Spectra. III”. en. In: *Physical Review* 63.9-10 (May 1943), pp. 367–382. ISSN: 0031-899X.
- [Rac49] Giulio Racah. “Theory of Complex Spectra. IV”. en. In: *Physical Review* 76.9 (Nov. 1949), pp. 1352–1365. ISSN: 0031-899X.
- [Raj65] K Rajnak. “Configuration Interaction in the 4f 3 Configuration of Pr iii”. In: *JOSA* 55.2 (1965). Publisher: Optica Publishing Group, pp. 126–132.
- [Rei81] Michael F Reid. “Applications of Group Theory in Solid State Physics”. PhD thesis. University of Canterbury, 1981.
- [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.
- [RW64a] K Rajnak and BG Wybourne. “Configuration interaction in crystal field theory”. In: *The Journal of Chemical Physics* 41.2 (1964). Publisher: American Institute of Physics, pp. 565–569.
- [RW64b] K Rajnak and BG Wybourne. “Electrostatically correlated spin-orbit interactions in l n-type configurations”. In: *Physical Review* 134.3A (1964). Publisher: APS, A596.
- [Sla29] J. C. Slater. “The Theory of Complex Spectra”. en. In: *Physical Review* 34.10 (Nov. 1929), pp. 1293–1322. ISSN: 0031-899X.
- [TLJ99] Anne Thorne, Ulf Litzén, and Sveneric Johansson. *Spectrophysics: principles and applications*. Springer Science & Business Media, 1999.
- [Tre51] RE Trees. “Spin-spin interaction”. In: *Physical Review* 82.5 (1951). Publisher: APS, p. 683.
- [Tre52] R. E. Trees. “The L ( L + 1 ) Correction to the Slater Formulas for the Energy Levels”. en. In: *Physical Review* 85.2 (Jan. 1952), pp. 382–382. ISSN: 0031-899X.
- [Tre58] Richard E. Trees. “Comparison of First, Second, and Third Approximations in Bacher and Goudsmit’s Theory of Atomic Spectra”. In: *J. Opt. Soc. Am.* 48.5 (May 1958). Publisher: Optica Publishing Group, pp. 293–300.
- [Vel00] Dobromir Velkov. “Multi-electron coefficients of fractional parentage for the p, d, and f shells”. PhD thesis. John Hopkins University, 2000.
- [WS07] Brian Wybourne and Lidia Smentek. *Optical Spectroscopy of Lanthanides*. 2007.
- [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.
- [Wyb64a] BG Wybourne. “Low-Lying Energy Levels of Trivalent Curium”. In: *The Journal of Chemical Physics* 40.5 (1964). Publisher: American Institute of Physics, pp. 1456–1457.
- [Wyb64b] BG Wybourne. “Orbit—Orbit Interactions and the“Linear”Theory of Configuration Interaction”. In: *The Journal of Chemical Physics* 40.5 (1964). Publisher: American Institute of Physics, pp. 1457–1458.
- [Wyb65] Brian G Wybourne. *Spectroscopic Properties of Rare Earths*. 1965.
- [Wyb70] Brian G Wybourne. *Symmetry principles and atomic spectroscopy*. 1970.

- [Wol24a] Wolfram Research. *SixJSymbol*. 2024.
- [Wol24b] Wolfram Research. *ThreeJSymbol*. 2024.

## Index

configuration interaction, 2  
crystal field, 45  
electrostatically-correlated-spin-orbit, 33  
forced electric dipole transitions, 62  
Judd-Ofelt theory, 62  
Kayser, 53  
Laporte's rule, 62  
level, 3  
magnetic dipole operator, 51  
Marvin integrals, 28, 33  
Mk, 33  
mostly orthogonal basis, 54  
orthogonal operators, 54  
Pk, 33  
Pseudo-magnetic parameters, 33  
Racah convention, 45  
semi-empirical approach, 2  
spherical harmonics, 45  
spin-other-orbit, 29  
spin-spin, 28  
state, 3  
t2Switch, 41  
term, 3  
three-body effective operators, 41  
Tk, 41