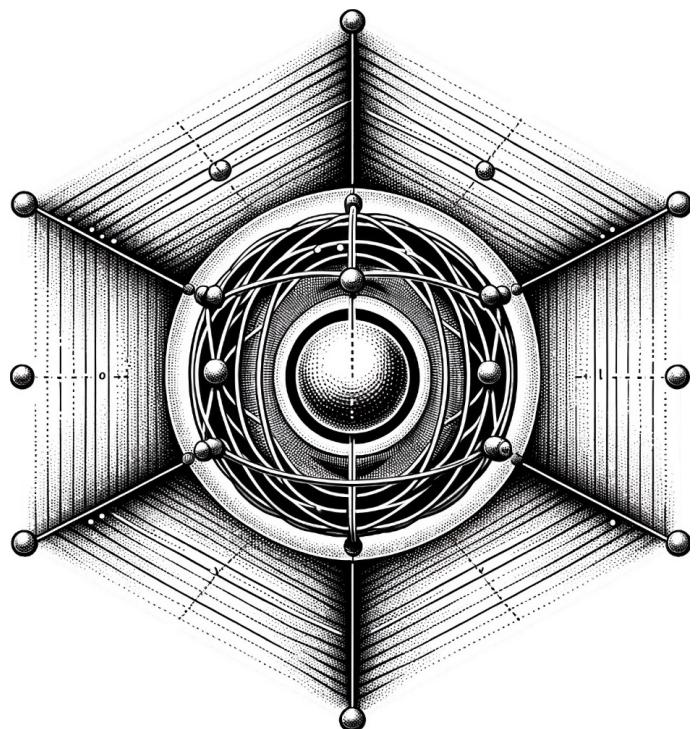


qlanth
doc version $|\alpha\rangle^{(17)}$



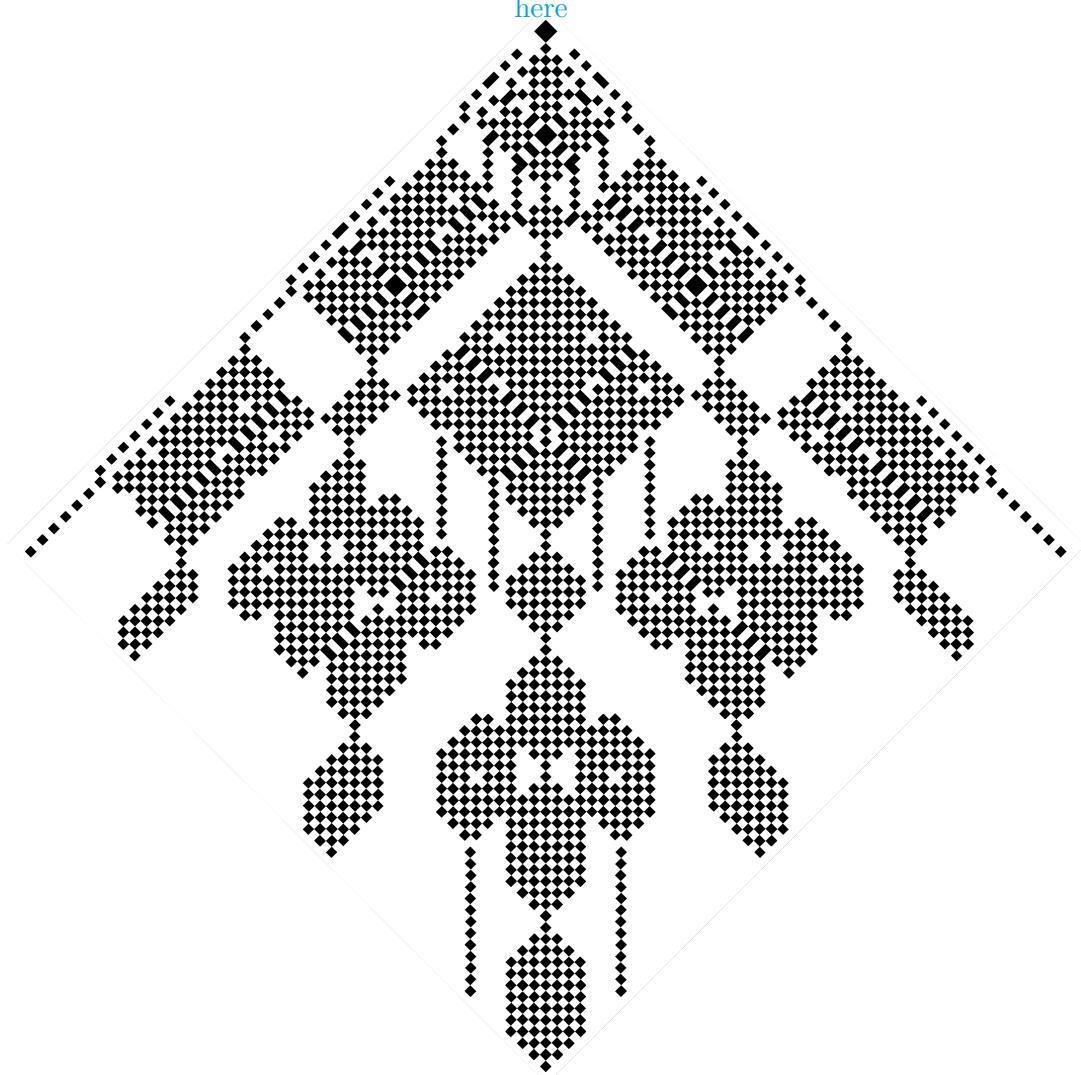
Juan David Lizarazo Ferro,
Christopher Dodson
& Rashid Zia

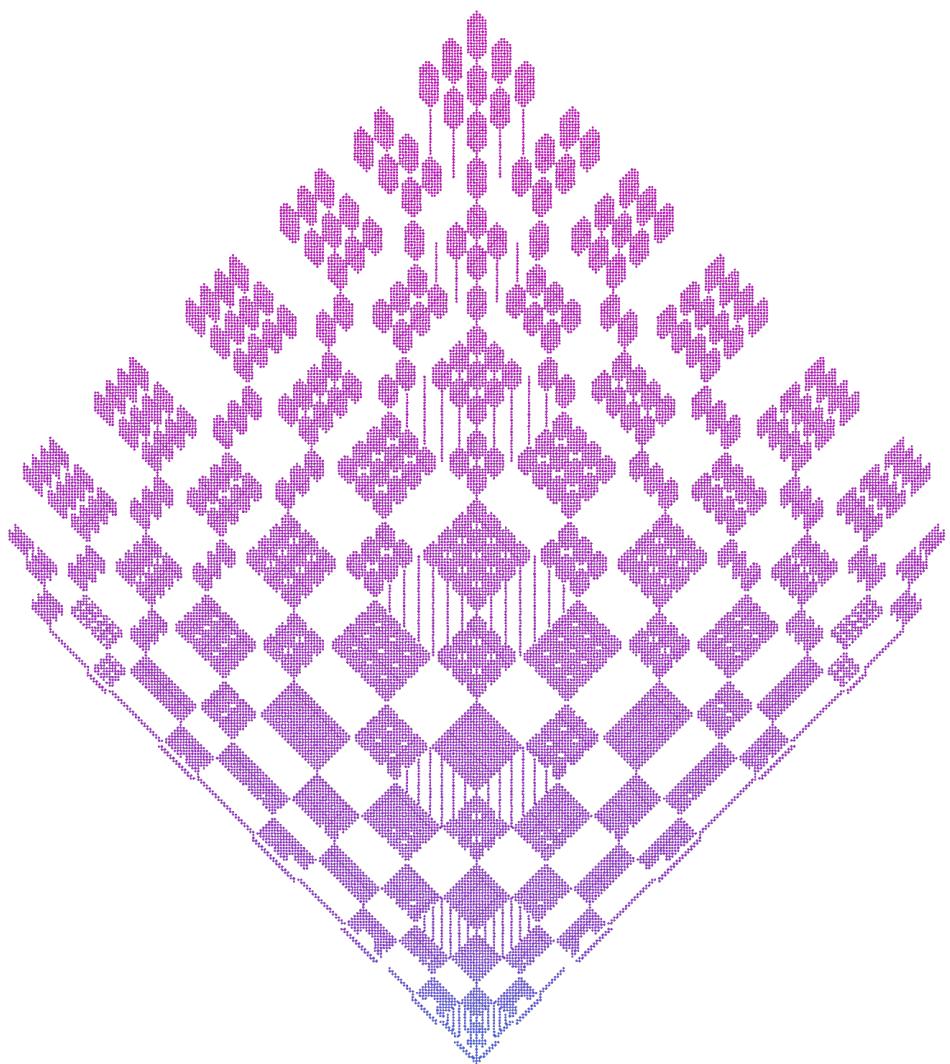
Brown University,
Department of Physics

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,¹ Giulio Racah,² Brian Judd,³ Gerhard Dieke,⁴ Hannah Crosswhite,⁵ Robert Cowan,⁶ Michael Reid,⁷ William Carnall,⁸ Clyde Morrison,⁹ Richard Leavitt,¹⁰ Brian Wybourne,¹¹ Richard Trees,¹² and Katherine Rajnak¹³ 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. LaF₃ is remarkable in that it was one of the systems in which a systematic study [Car+89] of all of the trivalent lanthanide ions were studied.

This code was originally authored by Christopher Dodson and Rashid Zia 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.

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

Contents

1	The semi-empirical Hamiltonian	1
2	LS coupling basis	3
2.1	$ LSJM\rangle$ states	4
2.2	More quantum numbers	8
2.2.1	Seniority ν	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
3	Interactions	22
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	40
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
4	Coordinate system	57
5	Spectroscopic measurements and uncertainty	57
6	Transitions	59
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
7	Parameter constraints	69
8	Fitting experimental data	69
9	Accompanying notebooks	85
10	Compiled data for $\text{LaF}_3:\text{Ln}^{3+}$ and $\text{LiYF}_4:\text{Ln}^{3+}$	85
11	sparsefn.py	87
12	Data sources	88
13	Other details	88
14	Units	88
15	Notation	89
16	Mathematica paclet	89
17	Definitions	90

18 code	92
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.

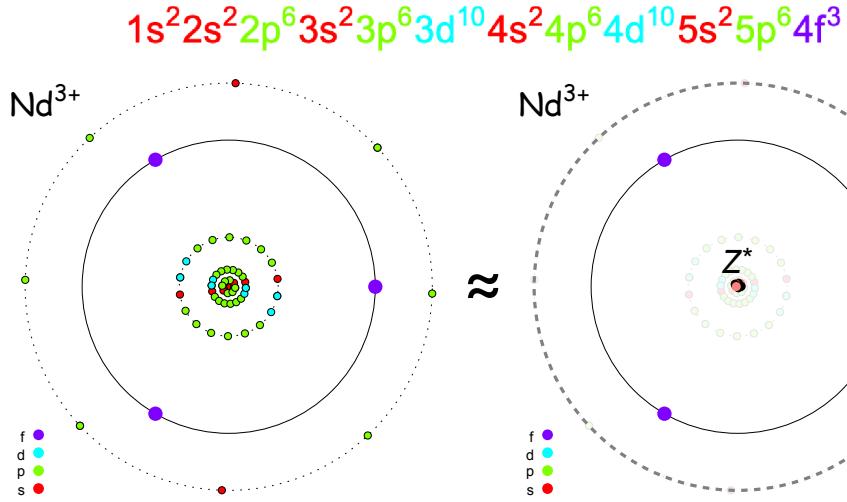


Figure 1: The trivalent neodymium ion shielded by $5s^2$ and $5p^6$ closed shells.

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

Ce³⁺ [Xe] f^1 Cerium	Pr³⁺ [Xe] f^2 Praseodymium	Nd³⁺ [Xe] f^3 Neodymium	Pm³⁺ [Xe] f^4 Promethium	Sm³⁺ [Xe] f^5 Samarium	Eu³⁺ [Xe] f^6 Europium	Gd³⁺ [Xe] f^7 Gadolinium	Tb³⁺ [Xe] f^8 Terbium	Dy³⁺ [Xe] f^9 Dysprosium	Ho³⁺ [Xe] f^{10} Holmium	Er³⁺ [Xe] f^{11} Erbium	Tm³⁺ [Xe] f^{12} Thulium	Yb³⁺ [Xe] f^{13} 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.

$$\hat{\mathcal{H}} = \underbrace{\hat{\mathcal{H}}_k}_{\text{kinetic}} + \underbrace{\hat{\mathcal{H}}_{e:\text{sn}}}_{\text{e:shielded nuc}} + \underbrace{\hat{\mathcal{H}}_{s:o}}_{\text{spin-orbit}} + \underbrace{\hat{\mathcal{H}}_{s:s}}_{\substack{\text{and spin:spin} \\ \text{and spin:other-orbit}}} + \underbrace{\hat{\mathcal{H}}_{s:oo \oplus \text{ecs:o}}}_{\substack{\text{spin:other-orbit} \\ \text{ec-correlated-spin:orbit}}} + \underbrace{\hat{\mathcal{H}}_Z}_{\text{Zeeman}} \\ + \underbrace{\hat{\mathcal{H}}_{e:e}}_{\text{e:e}} + \underbrace{\hat{\mathcal{H}}_{SO(3)}}_{\text{Trees effective op}} + \underbrace{\hat{\mathcal{H}}_{G_2}}_{\text{G}_2 \text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{SO(7)}}_{\text{SO}(7) \text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{\lambda}}_{\substack{\text{effective} \\ \text{three-body}}} + \underbrace{\hat{\mathcal{H}}_{cf}}_{\text{crystal field}}$$

(1)

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

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

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

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

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

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

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

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

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

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

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

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

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

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 Pr^{3+} in 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.

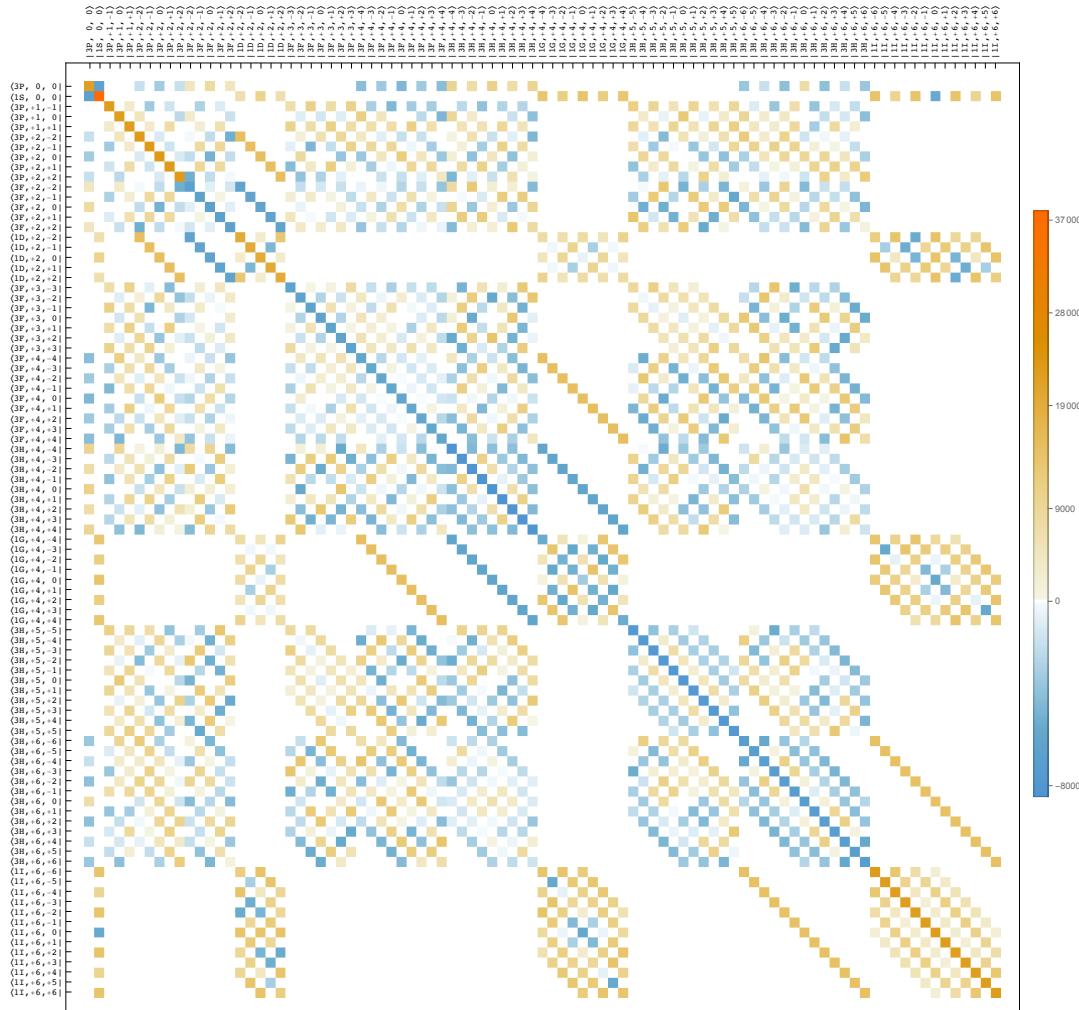


Figure 3: The matrix representation of $\hat{\mathcal{H}}$ for Pr^{3+} in LaF_3 in the $|LSJM_J\rangle$ basis.

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 number¹⁴ 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 $|LS\rangle$

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

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.

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 , $\hat{\mathbf{S}}^2$, $\hat{\mathbf{J}}^2$, and $\hat{\mathbf{J}}_z$. 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 **q1anth** 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 α (sadly, for historical reasons, we prepend it, rather than append it).

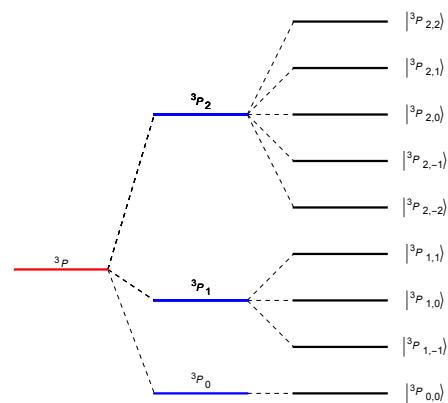


Figure 4: Levels and states associated with the 3P term in f^2 .

f^0 (1 LS term)
1S
f^1 (1 LS term)
2F

\underline{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)

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

$^1G_1, ^1G_2, ^1G_3, ^1G_4, ^1G_5, ^1G_6, ^1G_7, ^1G_8, ^1H_1, ^1H_2, ^1H_3, ^1H_4, ^1I_1, ^1I_2, ^1I_3, ^1I_4, ^1I_5, ^1I_6,$ $^1I_7, ^1K_1, ^1K_2, ^1K_3, ^1L_1, ^1L_2, ^1L_3, ^1L_4, ^1M_1, ^1M_2, ^1N_1, ^1N_2, ^1Q$
--

\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`.

<pre> 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"}; </pre>
--

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 **BasisLSJM** which provides a list with $\binom{14}{n}$ elements.

```

1 BasisLSJM::usage = "BasisLSJM[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[BasisLSJM] = {"AsAssociation" -> False};
5 BasisLSJM[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 $SO(7)$ (the Lie group of rotations in seven dimensions) and G_2 (the rank-2 exceptional simple Lie group).

2.2.1 Seniority ν

The seniority number connects different LS terms between configurations, so that a term below can be seen as the *senior* of a term above. To determine the seniority of a given term in configuration 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 term¹⁵. 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
2   term.";
3 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 G_2 . Without going into the group theoretical details, the irreducible representations of $SO(7)$ can be represented by triples of integer numbers, and those of G_2 as pairs of two integers.

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

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

```

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 FindNKLSTerm[SL_] := Module[
7   {NKterms, n},
8   (
9     n = 7;
10    NKterms = {};
11    Map[
12      If[! StringFreeQ[First[#], SL],
13        If[ToExpression[Part[#, 2]] <= n,
14          NKterms = Join[NKterms, {#}, 1]
15        ]
16      ] &,
17      fnTermLabels
18    ];
19    NKterms = DeleteCases[NKterms, {}];
20    NKterms
21  )
22];

```

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)

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

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

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

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

f⁶ (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 ,
 5D_1 , 5F_1 , 3P_1 , 3P_1 , 3P_1 , 3P_1 , 3P_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 , 3P_2 , 3D_2 , 3D_2 , 3D_2 ,
 3D_2 , 3D_2 , 3F_2 , 3F_2 , 3F_2 , 3F_2 , 3F_2 , 3F_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 , 3F_3 ,
 3F_3 , 3F_3 , 3F_3 , 3F_3 , 3F_3 , 3F_3 , 3G_3 , 1F_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 , 3F_4 , 3F_4 , 3G_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 , 3H_4 , 1G_4 , 7F_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 , 3H_5 , 3H_5 , 3H_5 ,
 3H_5 , 3H_5 , 3H_5 , 3H_5 , 3I_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 , 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 , 3K_8 , 3L_8 , 3L_8 , 3M_8 ,
 3M_8 , 3M_8 , 1L_8 , 1L_8 , 1L_8 , 5K_9 , 5L_9 , 3L_9 , 3L_9 , 3M_9 , 3M_9 , 3M_9 , 3N_9 , 1M_9 , 1M_9 ,
 $^5L_{10}$, $^3M_{10}$, $^3M_{10}$, $^3N_{10}$, $^3O_{10}$, $^1N_{10}$, $^1N_{10}$, $^3N_{11}$, $^3O_{11}$, $^3O_{12}$, $^1Q_{12}$

f⁷ (327 LSJ levels)

$^4K_{15/2}$, $^4L_{15/2}$, $^4L_{15/2}$, $^4M_{15/2}$, $^2K_{15/2}$, $^2K_{15/2}$, $^2K_{15/2}$, $^2K_{15/2}$, $^2K_{15/2}$, $^2K_{15/2}$,
 $^2K_{15/2}$, $^2L_{15/2}$, $^2L_{15/2}$, $^2L_{15/2}$, $^2L_{15/2}$, $^6I_{17/2}$, $^4K_{17/2}$, $^4K_{17/2}$, $^4L_{17/2}$,
 $^4L_{17/2}$, $^4L_{17/2}$, $^4M_{17/2}$, $^4N_{17/2}$, $^2L_{17/2}$, $^2L_{17/2}$, $^2L_{17/2}$, $^2L_{17/2}$, $^2M_{17/2}$, $^2M_{17/2}$,
 $^2M_{17/2}$, $^2M_{17/2}$, $^4L_{19/2}$, $^4L_{19/2}$, $^4L_{19/2}$, $^4M_{19/2}$, $^4N_{19/2}$, $^2M_{19/2}$, $^2M_{19/2}$, $^2M_{19/2}$,
 $^2N_{19/2}$, $^2N_{19/2}$, $^4M_{21/2}$, $^4N_{21/2}$, $^2N_{21/2}$, $^2O_{21/2}$, $^4N_{23/2}$, $^2O_{23/2}$, $^2Q_{23/2}$, $^2Q_{25/2}$

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

7F_0 , 5D_0 , 5D_0 , 3P_0 , 3P_0 , 3P_0 , 3P_0 , 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 , 1P_1 , 7F_2 , 5S_2 , 5P_2 ,
 5D_2 , 5D_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 , 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 , 5G_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 , 1F_3 , 1F_3 , 1F_3 ,
 1F_3 , 7F_4 , 5D_4 , 5D_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 , 5G_5 , 5G_5 ,
 5G_5 , 5H_5 , 5H_5 , 5I_5 , 5I_5 , 5K_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 , 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 , 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 , 5K_9 , 5L_9 , 3L_9 , 3L_9 , 3M_9 , 3M_9 , 3M_9 , 3N_9 , 1M_9 , 1M_9 ,
 $^5L_{10}$, $^3M_{10}$, $^3M_{10}$, $^3M_{10}$, $^3N_{10}$, $^1N_{10}$, $^3N_{11}$, $^3O_{11}$, $^3O_{12}$, $^1Q_{12}$

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

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

\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 , 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 , 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 , 1G_4 , 5F_5 , 5G_5 , 5I_5 , 3G_5 , 3G_5 , 3H_5 , 3H_5 ,
 3H_5 , 3H_5 , 3I_5 , 3I_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 ,
 5I_7 , 3I_7 , 3I_7 , 3K_7 , 3K_7 , 3L_7 , 1K_7 , 3K_8 , 3K_8 , 3L_8 , 3L_8 , 1L_8 , 3L_9 , 3M_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 `LevelSimplerSymbolicHamMatrix` is provided in `qlanth`.

```

1 LevelSimplerSymbolicHamMatrix::usage = "LevelSimplerSymbolicHamMatrix
   [numE] is a variation of HamMatrixAssembly 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[LevelSimplerSymbolicHamMatrix] = {
```

```

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 LevelSimplerSymbolicHamMatrix[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["Overwrite"]],
39       (
40         If[OptionValue["Return"],
41           (
42             Which[FileExistsQ[fnamemx],
43               (
44                 Print["File ", fnamemx, " already exists, and option ''Overwrite'' is set to False, loading file ..."];
45                 thisHamAssoc = Import[fnamemx];
46                 Return[thisHamAssoc];
47               ),
48               FileExistsQ[fname],
49               (
50                 Print["File ", fname, " already exists, and option ''Overwrite'' is set to False, loading file ..."];
51                 thisHamAssoc = Import[fname];
52                 Print["Exporting to file ", fnamemx, " for quicker loading."];
53               );
54               Export[fnamemx, thisHamAssoc];
55               Return[thisHamAssoc];
56             )
57           ],
58           (
59             Print["File ", fname, " already exists, skipping ..."];
60             Return[Null];
61           )
62         ];
63       Js = AllowedJ[numE];
64       thisHamAssoc = HamMatrixAssembly[numE,
65         "Set t2Switch" -> True,
66         "IncludeZeeman" -> False,
67         "ReturnInBlocks" -> True];
68       thisHamAssoc = Diagonal[thisHamAssoc];
69       thisHamAssoc = Map[SparseArray[ReplaceInSparseArray[#, simplifier] &, thisHamAssoc, {1}]];
70       thisHamAssoc = FreeHam[thisHamAssoc, numE];
71       thisHamAssoc = AssociationThread[Js -> thisHamAssoc];
72       If[OptionValue["Export"],
73         (

```

```

74     Print["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 mayor 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
30   for defining the parameters kept in the level description.
";
31 Options[LevelSolver] = {
32   "Overwrite Hamiltonian" -> False,
33   "Return as states" -> True,
34   "Simplifier" -> Join[
35     cfSymbols,
36     TSymbols,
37     casimirSymbols,
38     pseudoMagneticSymbols,
39     marvinSymbols,
40     DeleteCases[magneticSymbols, \[Zeta]]
41   ],
42   "PrintFun" -> PrintTemporary
43 };
44 LevelSolver[numE_Integer, params0_Association, OptionsPattern[]] :=
45   Module[
46   {ln, simplifier, simpleHam, basis,
47    numHam, eigensys, startTime, endTime,
48    diagonalTime, params=params0, globalBasis,
49    eigenVectors, eigenEnergies, eigenJs,
50    states, groundEnergy, allEnergies, PrintFun},
51   (
52     ln           = theLanthanides[[numE]];
53     basis        = BasisLSJ[numE, "AsAssociation" -> True];
54   ];

```

```

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

```

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[#, NumberString], ToExpression[#], #] &, list];
13    CFPTable      = ConstantArray[{}, 7];
14    CFPTable[[1]] = {{"2F", {"1S", 1}}};
15
16 (* Cleaning before processing is useful *)
17 rawText = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
18 rawLines = StringTrim/@StringSplit[rawText, "\n"];
19 rawLines = Select[rawLines, # != "" &];
20 rawLines = CleanWhitespace/@rawLines;
21 rawLines = AddSpaceBeforeMinus/@rawLines;
22
23 Do[(
24   (* the first character can be used to identify the start of a
25   block *)
26   leadChar=StringTake[line,{1}];
27   (* ...FN, N is at position 50 in that line *)
28   If[leadChar=="[",
29   (
30     configIndex=ToExpression[StringTake[line,{50}]];
31     Continue[];
32   )
33   ];
34   (* Identify which daughter term is being listed *)
35   If[StringContainsQ[line, "[DAUGHTER TERM]"],
36     daughter=StringSplit[line, "["][[1]];
37     CFPTable[[configIndex]]=Append[CFPTable[[configIndex]], {
38       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 `qanth` 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 `HamMatrixAssembly` 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 (2×2) , (2×3) , (2×15) , (2×7) , (2×27) , (2×11) , and (2×26) .

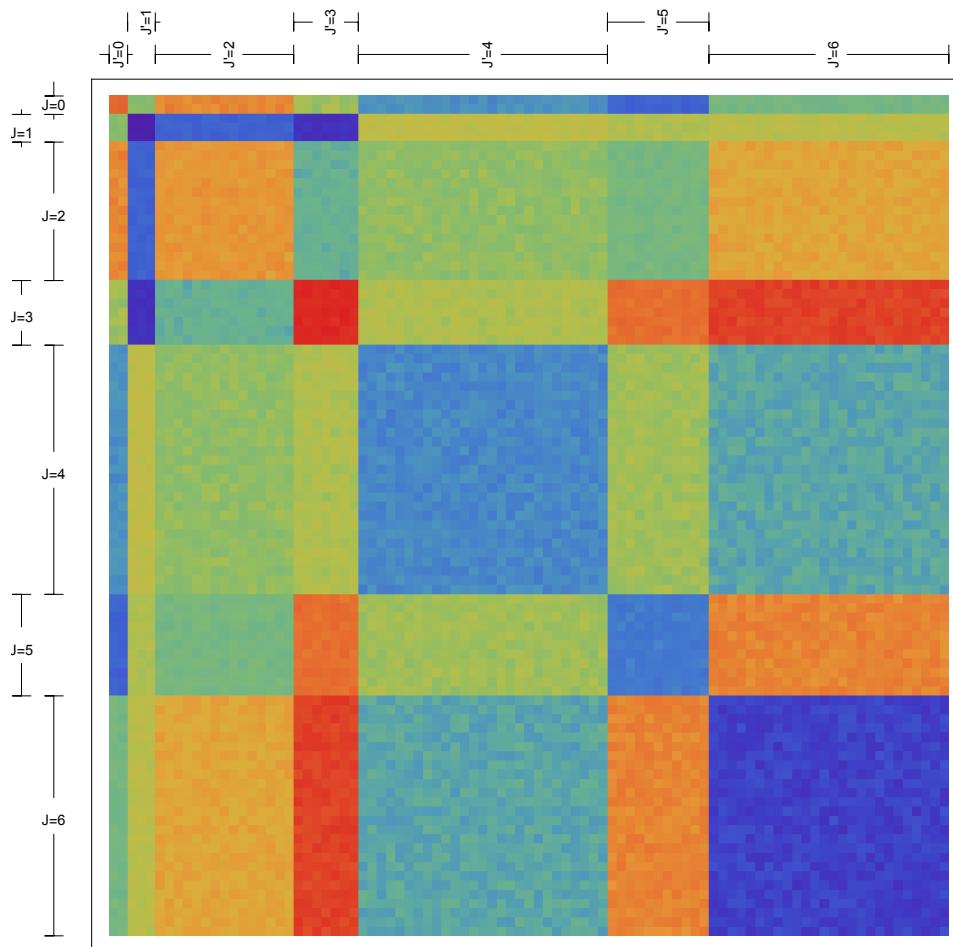


Figure 5: The J-J' block structure for f^2

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, NKSLJMps, NKSLJM, NKSLJMp,
5 SLterm, SpLpterm,
6 MJ, MJp,
7 subKron, matValue, eMatrix},
8 (
9   NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
10  NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
11  eMatrix =
12   Table[
13     (*Condition for a scalar matrix op*)
14     SLterm = NKSLJM[[1]];
15     SpLpterm = NKSLJM[[1]];
16     MJ = NKSLJM[[3]];
17     MJp = NKSLJM[[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[{SLterm, SpLpterm}] +
28         SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
29         MagneticInteractions[{numE, SLterm, SpLpterm, J},
30           "ChenDeltas" -> OptionValue["ChenDeltas"]] +
31         ThreeBodyTable[{numE, SLterm, SpLpterm}]
32       )
33     ];
34     matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp
35 }];
36     matValue,
37     {NKSLJMp, NKSLJMps},
38     {NKSLJM, NKSLJMs}
39   ];
40   If[OptionValue["Sparse"],
41     eMatrix = SparseArray[eMatrix]
42   ];
43   Return[eMatrix]
44 )
45 ];

```

Once these blocks have been calculated and saved to disk (in the folder `./hams/`) the function `HamMatrixAssembly` takes them, assembles the arrays in block form, and finally flattens 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 HamMatrixAssembly::usage = "HamMatrixAssembly[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[HamMatrixAssembly] = {
16   "FilenameAppendix" -> "",
17   "IncludeZeeman" -> False,
18   "Set t2Switch" -> True,
19   "ReturnInBlocks" -> False,
20   "OperatorBasis" -> "Legacy"};
21 HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
22   {numE, ii, jj, howManyJs, Js, blockHam, opBasis},
23 
```

```

14  (
15  (*#####
16  ImportFun = ImportMZip;
17  opBasis = OptionValue["OperatorBasis"];
18  If[Not[MemberQ>{"Legacy", "MostlyOrthogonal", "Orthogonal"}, opBasis]],
19    Print["Operator basis ", opBasis, " not recognized, using ",
20  Legacy'', basis."];
21  opBasis = "Legacy";
22 ];
23 If[opBasis == "Orthogonal",
24   Print["Operator basis ''Orthogonal'' not implemented yet,
25  aborting ..."];
26   Return[Null];
27 ];
28 (*#####
29 If[opBasis == "MostlyOrthogonal",
30   (
31   blockHam = HamMatrixAssembly[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         (
88           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]&,amp;, 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 = JJBBlockMatrixFileName[numE, "FilenameAppendix" ->
157 OptionValue["FilenameAppendix"]];
158 JJBBlockMatrixTable = ImportFun[emFname];
159 (*Patch together the entire matrix representation using J,J'
160 blocks.*)
161 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]&,amp;,blockHam
172       ,{2}]);
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 ]

```

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, [Eqn-??](#) 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 LaF₃, 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 $\approx 10^6 \mathcal{K}$ ¹⁶ 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 shell structure of the atom, the lanthanide ions “see” the nuclear charge as shielded by a

¹⁶ 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} $	$\begin{aligned} & 2\alpha + \beta + \frac{B_0^{(2)}}{5} + \gamma \\ & -\frac{\zeta}{2} + \frac{14F^{(0)}}{13} + \frac{43F^{(2)}}{195} + \frac{19F^{(4)}}{429} \\ & -\frac{875F^{(6)}}{5577} + 2m^{(0)}\sigma_{SS} + \frac{61m^{(0)}}{12} \\ & + 4m^{(2)}\sigma_{SS} + \frac{145m^{(2)}}{12} + \frac{50m^{(4)}\sigma_{SS}}{11} \\ & + \frac{1805m^{(4)}}{132} + \frac{43P^{(2)}}{1080} + \frac{19P^{(4)}}{2376} \\ & - \frac{875P^{(6)}}{30888} \end{aligned}$	$\begin{aligned} & \frac{\sqrt{3}B_1^{(2)}}{10} + \frac{1}{10}i\sqrt{3}S_2^{(2)} \\ & -\frac{1}{5}\sqrt{\frac{3}{2}}B_2^{(2)} + \frac{1}{5}i\sqrt{\frac{3}{2}}S_2^{(2)} \end{aligned}$	$\begin{aligned} & 2\alpha + \beta - \frac{B_0^{(2)}}{10} + \gamma \\ & -\frac{\zeta}{2} + \frac{14F^{(0)}}{13} + \frac{43F^{(2)}}{195} + \frac{19F^{(4)}}{429} \\ & -\frac{875F^{(6)}}{5577} + 2m^{(0)}\sigma_{SS} + \frac{61m^{(0)}}{12} \\ & + 4m^{(2)}\sigma_{SS} + \frac{145m^{(2)}}{12} + \frac{50m^{(4)}\sigma_{SS}}{11} \\ & + \frac{1805m^{(4)}}{132} + \frac{43P^{(2)}}{1080} + \frac{19P^{(4)}}{2376} \\ & - \frac{875P^{(6)}}{30888} \end{aligned}$

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

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

This term is the first we will not discard. Calculating this term for the \underline{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 \underline{f}^n configurations can be greatly simplified.

The functions used in `qlanth` to compute these LS-reduced matrix elements ¹⁷ are `Electrostatic` and `fsubk`. In addition to these, the LS-reduced matrix elements of the tensor operators $\hat{C}^{(k)}$ and $\hat{U}^{(k)}$ are also needed. These functions are based in equations 12.16 and 12.17 from [Cow81] as specialized for the case of electrons belonging to a single \underline{f}^n configuration. By default this term is computed in terms of $\mathbf{F}^{(k)}$ Slater integrals, but it can also be computed in terms of the \mathbf{E}_k Racah parameters, the functions `EtoF` and `FtoE` are useful for going from one representation to the other.

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

where

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

```

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         esub0 = fsup0;
32       )
33     ]
34   ]
35 
```

¹⁷ An LS-reduced matrix element is ...

```

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_, NKSLP_, 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[NKSLP];
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, NKSLP, k}]
25         ),
26         {SL, termsWithSameSpin}
27     ];
28     summand1 = 1 / TPO[L] * summand1;
29     summand2 = (
30         KroneckerDelta[NKSL, NKSLP] *
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
       superscripted Slater integrals, it is necessary to convert them
       using Dk.";
6 FtoE[F0_, F2_, F4_, F6_] := Module[
7     {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

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

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

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

From equations 2-106 to 2-109 in Wybourne [WYB63] the matrix elements we need are given by

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

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

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

in which the rank on the spin operator \hat{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]:

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

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

```

1 ReducedV1k::usage = "ReducedV1k[n, SL, SpLp, k] gives the reduced
  matrix element of the spherical tensor operator V^(1k). See
  equation 2-101 in Wybourne 1965.";
2 ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
3   {V1k, S, L, Sp, Lp,
4   Sb, Lb, spin, orbital,
5   cfpSL, cfpSpLp,
6   SLparents, SpLpparents,
7   commonParents, prefactor},
8   (
9     {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 )
30];

```

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

```

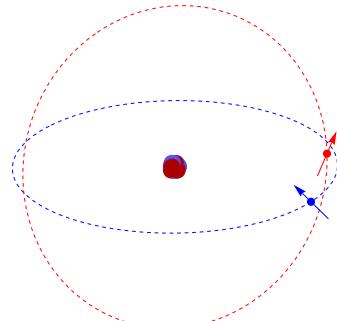
3.6 $\hat{\mathcal{H}}_{\text{s:s-oo}}$: spin-spin and spin-other-orbit

The calculation of the $\hat{\mathcal{H}}_{\text{s:s-oo}}$ 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]

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

and the one for *spin-other-orbit*

$$\begin{aligned} \hat{\mathcal{H}}_{\text{s:oo}} = & \sum_{i \neq j} \sum_k \sqrt{(k+1)(2k+1)(2k+3)} \times \\ & \left[\left\{ \hat{w}_i^{(0,k+1)} \hat{w}_j^{(1,k)} \right\}^{(11)0} \left\{ m^{(k-1)} \langle \underline{\ell} | \mathcal{C}^{(k+1)} | \underline{\ell} \rangle^2 + 2m^{(k)} \langle \underline{\ell} | \mathcal{C}^{(k)} | \underline{\ell} \rangle^2 \right\} + \right. \\ & \left. \left\{ \hat{w}_i^{(0,k)} \hat{w}_j^{(1,k+1)} \right\}^{(11)0} \left\{ m^{(k)} \langle \underline{\ell} | \mathcal{C}^{(k)} | \underline{\ell} \rangle^2 + 2m^{(k-1)} \langle \underline{\ell} | \mathcal{C}^{(k+1)} | \underline{\ell} \rangle^2 \right\} \right]. \end{aligned} \quad (36)$$

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

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

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

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

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

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 `q1anth` in the functions `SpinSpin` and `SOOandECSO`.

```

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 ];
27 
```

```

1 SOOandECSO::usage = "SOOandECSO[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   SOOandECSOLSTable and putting in the adequate phase and 6-j symbol
9   . The SOOandECSOLSTable 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 SOOandECSO[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       SOOandECSOLSTable[{numE, SL, SpLp}]
26     );
27     Return[val];
28   )
29 ]; 
```

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]

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

where the sum runs over the terms $\bar{\psi}$ and $\bar{\psi}'$ in \underline{f}^{n-1} which are parents common to ψ and ψ' . 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 $\hat{\mathcal{H}}_{s:oo}$. This is done for $\hat{\mathcal{H}}_{ss}$, but not for $\hat{\mathcal{H}}_{s:oo}$, 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"];
13      template3 = 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"]], 0.1] |>},
25              {template3[<|"speed" -> Round[QuantityMagnitude[Now -
26                startTime, "ms"]/(numiter), 0.01] |>}],
27              {ProgressIndicator[Dynamic[numiter], {1, totalIters}]}}},
28            Frame -> All],
29            Full,
30            Alignment -> Center]
31          ]
32        ];
33      ];
34      T22Table = <||>;
35      startTime = Now;
36      numiter = 1;
37      Do[
38        (
39          numiter += 1;
40          T22Table[{numE, SL, SpLp}] = Which[
41            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 ];

```

```

1 Reducedt11inf2::usage = "Reducedt11inf2[SL, SpLp] returns the reduced
   matrix element in f^2 of the double tensor operator t11 for the
   corresponding given terms {SL, SpLp}.
2 Values given here are those from Table VII of ''Judd, BR, HM
   Crosswhite, and Hannah Crosswhite. ''Intra-Atomic Magnetic
   Interactions for f Electrons.'' Physical Review 169, no. 1 (1968):
   130.''
3 ";
4 Reducedt11inf2[SL_, SpLp_] := Module[
5   {t11inf2},
6   (
7     t11inf2 = <|
8       {"1S", "3P"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
9       {"3P", "3P"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
10      {"3P", "1D"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
11      {"1D", "3F"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
12      {"3F", "3F"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
13      {"3F", "1G"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
14      {"1G", "3H"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
15      {"3H", "3H"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
16      {"3H", "1I"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
17     |>;
18   Which[
19     MemberQ[Keys[t11inf2], {SL, SpLp}],
20     Return[t11inf2[{SL, SpLp}]],
21     MemberQ[Keys[t11inf2], {SpLp, SL}],
22     Return[t11inf2[{SpLp, SL}]],
23     True,
24     Return[0]
25   ]
26 )
27 ];

```

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])

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

where $\xi(r_h)(\hat{\mathbf{r}}_h \cdot \hat{\mathbf{l}}_h)$ is the customary spin-orbit interaction, E_{χ} is the energy of state $|\chi\rangle$, i is a label for the valence electrons, $\hat{\mathcal{C}}$ 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

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

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 $m^{(k)}$ appear again, but a

second set of parameters, the *pseudo-magnetic* parameters $P^{(k)}$, is also necessary

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

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

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

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

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 $m^{(k)}$ and the set of $P^{(k)}$ ultimately rely on a single free parameter each. Such parsimony is desirable given the large number of parameters (about 20) that the Hamiltonian ends up having.

Judd *et al.* further note that $P^{(0)}$ is proportional to the spin orbit operator, and as such its effect is absorbed by the standard spin-orbit parameter ζ . They also developed an alternative approach based on group theory arguments. They put together the *spin-other-orbit* and the *electrostatically-correlated-spin-orbit* as a sum of operators \hat{z}_i with useful transformation rules

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

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 $\hat{T}^{(11)} + \hat{t}^{(11)}$. This point was clarified by Chen *et al.* [Che+08]. Because of this, the final form of the operator contributing both to *spin-other-orbit* and the *electrostatically-correlated-spin-orbit* is

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

where

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

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 **S00andECSOTable**. In turn these two associations are generated by the functions **GenerateSpinOrbitTable** and **GenerateS00andECSOTable**. Note that both *spin-spin* and *spin-other-orbit* end up contributing through $m^{(k)}$, however there doesn't seem to be consensus about adding them together, as such **qlanth** allows including or excluding the *spin-spin* contribution, this is done with a control parameter σ_{SS} (1 for including, 0 for excluding).

```

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 SpinSpin[{numE_, SL_, SLP_, J_}].
3 The part corresponding to S00 and ECSO is provided by the function S00andECSO[{numE_, SL_, SLP_, J_}].
4 The option ''ChenDeltas'' can be used to include or exclude the Chen deltas from the calculation. The default is to exclude them. If this option is used, then the chenDeltas association needs to be loaded into the session with LoadChenDeltas[].";
5 Options[MagneticInteractions] = {"ChenDeltas" -> False};
6 MagneticInteractions[{numE_, SL_, SLP_, J_}, OptionsPattern[]] :=
  Module[
  7 {key, ss, sooandecso, total,

```

```

8 S, L, Sp, Lp, phase, sixjay,
9 M0v, M2v, M4v,
10 P2v, P4v, P6v},
11 (
12     key      = {numE, SL, SLP, J};
13     ss       = \[\Sigma]SS * SpinSpinTable[key];
14     sooandecso = SOOandECSOTable[key];
15     total = ss + sooandecso;
16     total = SimplifyFun[total];
17     If[
18         Not[OptionValue["ChenDeltas"]],
19         Return[total]
20     ];
21     (* In the type A errors the wrong values are different *)
22     If[MemberQ[Keys[chenDeltas["A"]], {numE, SL, SLP}],
23     (
24         {S, L} = FindSL[SL];
25         {Sp, Lp} = FindSL[SLP];
26         phase = Phaser[Sp + L + J];
27         sixjay = SixJay[{Sp, Lp, J}, {L, S, 1}];
28         {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SL,
29         SLP}]["wrong"];
30         total = (
31             phase * sixjay *
32             (
33                 M0v*M0 + M2v*M2 + M4v*M4 +
34                 P2v*P2 + P4v*P4 + P6v*P6
35             )
36         );
37         total = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
38     )
39     );
40     (* In the type B errors the wrong values are zeros all around *)
41     If[MemberQ[chenDeltas["B"], {numE, SL, SLP}],
42     (
43         total = (1 - wChErrB) * (ss + sooandecso)
44     )
45     ];
46     Return[total];
47 )
48 ];

```

```

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
21     exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
22     If[OptionValue["Export"],
23     (
24       Print["Exporting to file " <> ToString[exportFname]];
25       Export[exportFname, SpinOrbitTable];
26     )
27     ];
28     Return[SpinOrbitTable];
29   )
30 ];

```

```

1 GenerateSOOandECSOTable::usage = "GenerateSOOandECSOTable[nmax]
2   generates the reduced matrix elements in the |LSJ> basis for the (
3     spin-other-orbit + electrostatically-correlated-spin-orbit)
4   operator. It returns an association where the keys are of the form
5     {n, SL, SpLp, J}. If the option ''Export'' is set to True then
6   the resulting object is saved to the data folder. Since this is a
7   scalar operator, there is no MJ dependence. This dependence only
8   comes into play when the crystal field contribution is taken into
9   account.";
10 Options[GenerateSOOandECSOTable] = {"Export" -> False};
11 GenerateSOOandECSOTable[nmax_, OptionsPattern[]] :=
12   SOOandECSOTable = <||>;
13   Do[
14     SOOandECSOTable[{numE, SL, SpLp, J}] = (SOOandECSO[numE, SL, SpLp
15       , J] /. Prescaling),,
16     {numE, 1, nmax},
17     {J, MinJ[numE], MaxJ[numE]},,
18     {SL, First /@ AllowedNKSLforJTerms[numE, J]},,
19     {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
20   ];
21   If[OptionValue["Export"],
22     (
23       fname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
24       Export[fname, SOOandECSOTable];
25     )
26   ];
27   Return[SOOandECSOTable];
28 ];

```

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

```

1 GenerateSpinSpinTable::usage = "GenerateSpinSpinTable[nmax] generates
2   the reduced matrix elements in the |LSJ> basis for the spin-spin
3   operator. It returns an association where the keys are of the form
4     {numE, SL, SpLp, J}. If the option ''Export'' is set to True then
5   the resulting object is saved to the data folder. Since this is a
6   scalar operator, there is no MJ dependence. This dependence only
7   comes into play when the crystal field contribution is taken into
8   account.";
9 Options[GenerateSpinSpinTable] = {"Export" -> False};
10 GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
11   (
12     SpinSpinTable = <||>;
13     PrintTemporary[Dynamic[numE]];
14     Do[
15       SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp,
16         J]),,
17       {numE, 1, nmax},
18       {J, MinJ[numE], MaxJ[numE]},,
19       {SL, First /@ AllowedNKSLforJTerms[numE, J]},,
20       {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
21     ];
22     If[OptionValue["Export"],
23       (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
24        Export[fname, SpinSpinTable];
25      )
26    ];
27   );
28   Return[SpinSpinTable];
29 );

```

```

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[

```

```

6 {S, L, Sp, Lp, α, val},
7 (
8   α = 2;
9   {S, L} = FindSL[SL];
10  {Sp, Lp} = FindSL[SpLp];
11  val = (
12    Phaser[Sp + L + J] *
13    SixJay[{Sp, Lp, J}, {L, S, α}] *
14    T22Table[{numE, SL, SpLp}]
15  );
16  Return[val]
17 )
18 ];

```

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           ]
44         ];
45       ];
46     ];
47     T22Table = <||>;
48     startTime = Now;
49     numiter = 1;
50     Do[
51       (
52         numiter+= 1;
53         T22Table[{numE, SL, SpLp}] = Which[
54           numE==1,
55           0,
56           numE==2,
57           ]
58       );
59     ];
60   ];
61 
```

```

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.'
";
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.
";
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 `GenerateS00andECSOTable` calls the function `S00andECSO` over all possible combinations of the arguments $\{n, SL, S'L', J\}$ and uses their values to populate the association `S00andECSOTable`. In turn the function `S00andECSO` queries the precomputed values of [Eqn-48](#) as stored in the association `S00andECSOLSTable`.

```

1 GenerateS00andECSOTable::usage = "GenerateS00andECSOTable[nmax]
2   generates the reduced matrix elements in the |LSJ> basis for the (
3     spin-other-orbit + electrostatically-correlated-spin-orbit)
4   operator. It returns an association where the keys are of the form
5     {n, SL, SpLp, J}. If the option ''Export'' is set to True then
6     the resulting object is saved to the data folder. Since this is a
7     scalar operator, there is no MJ dependence. This dependence only
8     comes into play when the crystal field contribution is taken into
9     account.";
10 Options[GenerateS00andECSOTable] = {"Export" -> False};
11 GenerateS00andECSOTable[nmax_, OptionsPattern[]] := (
12   S00andECSOTable = <||>;
13   Do[
14     S00andECSOTable[{numE, SL, SpLp, J}] = (S00andECSO[numE, SL, SpLp
15       , J] /. Prescaling);
16     {numE, 1, nmax},
17     {J, MinJ[numE], MaxJ[numE]},
18     {SL, First /@ AllowedNKSLforJTerms[numE, J]},
19     {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
20   ];
21   If[OptionValue["Export"],
22   (
23     fname = FileNameJoin[{moduleDir, "data", "S00andECSOTable.m"}];
24     Export[fname, S00andECSOTable];
25   )
26 ];
27   Return[S00andECSOTable];
28 );
29 
```

```

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 
```

```
17 ];
```

```
1 S00andECSO::usage = "S00andECSO[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
  S00andECSOLSTable and putting in the adequate phase and 6-j symbol
  . The S00andECSOLSTable 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];
```

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]
  calculates the reduced matrix elements of the (spin-other-orbit +
  ECSO) operator for the  $f^{numE}$  configuration corresponding to the
  terms  $SL$  and  $SpLp$ . This is done recursively, starting from
  tabulated values for  $f^2$  from ''Judd, BR, HM Crosswhite, and
  Hannah Crosswhite. ''Intra-Atomic Magnetic Interactions for f
  Electrons.'' Physical Review 169, no. 1 (1968): 130.'', and by
  using equation (4) of that same paper.
2 ";
3 ReducedS00andECSOinfn[numE_, SL_, SpLp_] := Module[
4   {spin, orbital, t, S, L, Sp, Lp,
5   idx1, idx2, cfpSL, cfpSpLp, parentSL,
6   Sb, Lb, Sbp, Lbp, parentSpLp, funval},
7   (
8     {spin, orbital} = {1/2, 3};
9     {S, L} = FindSL[SL];
10    {Sp, Lp} = FindSL[SpLp];
11    t = 1;
12    cfpSL = CFP[{numE, SL}];
13    cfpSpLp = CFP[{numE, SpLp}];
14    funval = Sum[
15      (
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          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 ReducedSO0andECSOinf2::usage = "ReducedSO0andECSOinf2[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 ReducedSO0andECSOinf2[SL_, SpLp_] := Module[
23   {a13, z13, z13inf2, matElement, redSO0andECSOinf2},
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     redSO0andECSOinf2 = (
47       ReducedT11inf2[SL, SpLp] +
48       Reducedt11inf2[SL, SpLp] -
49       a13 / 6 * matElement
50     );
51     redSO0andECSOinf2 = SimplifyFun[redSO0andECSOinf2];
52     Return[redSO0andECSOinf2];
53   )
54 ];

```

3.8 $\hat{\mathcal{H}}_3$: 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)

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

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 ψ and ψ' 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" -> "
   }^{""}] <> "}") & /@ TSymbols;
5 TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
6 juddOperators = ParseJudd1984[];
7 (* 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. *)
8 op3MatrixElement[SL_, SpLp_, opSymbol_] := (
9   jOP = juddOperators[{3, opSymbol}];
10  key = {SL, SpLp};
11  val = If[MemberQ[Keys[jOP], key],
```

```

12     jOP[key],
13     0];
14   Return[val];
15 );
16 (* ti: This is the implementation of formula (2) in Judd & Suskin
17    1984. It computes the reduced matrix elements of ti in f^n by
18    using the reduced matrix elements in f^3 and the coefficients of
19    fractional parentage. If the option ''Fast'' is set to True then
20    the values for n>7 are simply computed as the negatives of the
21    values in the complementary configuration; this except for t2 and
22    t11 which are treated as special cases. *)
23 Options[ti] = {"Fast" -> True};
24 ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
25 Module[
26   {nn, S, L, Sp, Lp,
27   cfpSL, cfpSpLp,
28   parentSL, parentSpLp,
29   tnk, tnks},
30   (
31     {S, L} = FindSL[SL];
32     {Sp, Lp} = FindSL[SpLp];
33     fast = OptionValue["Fast"];
34     numH = 14 - nE;
35     If[fast && Not[MemberQ[{ "t_{2}" , "t_{11}" }, tiKey]] && nE > 7,
36       Return[-tktable[{numH, SL, SpLp, tiKey}]]
37     ];
38     If[(S == Sp && L == Lp),
39       (
40       cfpSL = CFP[{nE, SL}];
41       cfpSpLp = CFP[{nE, SpLp}];
42       tnks = Table[(
43         parentSL = cfpSL[[nn, 1]];
44         parentSpLp = cfpSpLp[[mm, 1]];
45         cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
46         tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
47       ),
48         {nn, 2, Length[cfpSL]},
49         {mm, 2, Length[cfpSpLp]}
50       ];
51       tnk = Total[Flatten[tnks]];
52     ),
53       tnk = 0;
54     ];
55     Return[nE / (nE - opOrder) * tnk];
56   )
57 ];
58 (* Calculate the reduced matrix elements of t^i for n up to 14 *)
59 tktable = <||>;
60 Do[(
61   Do[(
62     tkValue = Which[numE <= 2,
63       (*Initialize n=1,2 with zeros*)
64       0,
65       numE == 3,
66       (* Grab matrix elem in f^3 from Judd 1984 *)
67       SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
68       True,
69       SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2,
70       3]]];
71     ];
72     tktable[{numE, SL, SpLp, opKey}] = tkValue;
73   ),
74     {SL, AllowedNKSLTerms[numE]},
75     {SpLp, AllowedNKSLTerms[numE]},
76     {opKey, Append[tiKeys, "e_{3}"]}
77   ];
78   PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " "
79   configuration complete"]];
80   ),
81   {numE, 1, 14}
82 ];
83 (* Now use those reduced matrix elements to determine their sum as
84    weighted by their corresponding strengths Ti *)
85 ThreeBodyTable = <||>;
86 Do[(

```

```

78 Do [
79 (
80   ThreeBodyTable[{numE, SL, SpLp}] = (
81     Sum[(
82       If[tiKey == "t_{2}", t2Switch, 1] *
83         tktable[{numE, SL, SpLp, tiKey}] *
84         TSymbolsAssoc[tiKey] +
85       If[tiKey == "t_{2}", 1 - t2Switch, 0] *
86         (-tktable[{14 - numE, SL, SpLp, tiKey}]) *
87         TSymbolsAssoc[tiKey]
88       ),
89       {tiKey, tiKeys}
90     ]
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
101 ThreeBodyTables = Table[(
102   terms = AllowedNKSLTerms[numE];
103   singleThreeBodyTable =
104     Table[
105       {SL, SLP} -> ThreeBodyTable[{numE, SL, SLP}],
106       {SL, terms},
107       {SLP, terms}
108     ];
109   singleThreeBodyTable = Flatten[singleThreeBodyTable];
110   singleThreeBodyTables = Table[(
111     notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
112     reps = ConstantArray[0, Length[TSymbols]];
113     reps[[notNullPosition]] = 1;
114     rep = AssociationThread[TSymbols -> reps];
115     notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
116   ),
117   {notNullSymbol, TSymbols}
118 ];
119 singleThreeBodyTables = Association[singleThreeBodyTables];
120 numE -> singleThreeBodyTables),
121 {numE, 1, 7}
122 ];
123 ThreeBodyTables = Association[ThreeBodyTables];
124 If[OptionValue["Export"],
125 (
126   threeBodyTablefname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
127   Export[threeBodyTablefname, ThreeBodyTable];
128   threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
129   Export[threeBodyTablesfname, ThreeBodyTables];
130 )
131 ];
132 Return[{ThreeBodyTable, ThreeBodyTables}];
133 );

```

```

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};
6 ParseJudd1984[OptionsPattern[]} := (
7   ParseJuddTab1[str_] := (
8     strR = ToString[str];
9     strR = StringReplace[strR, ".5" -> "^(1/2)"];
10    num = ToExpression[strR];
11    sign = Sign[num];
12    num = sign*Simplify[Sqrt[num^2]];
13    If[Round[num] == num, num = Round[num]];
14    Return[num]);
15
16 (* 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
config*)
61 juddOperators = Association[];
62 Do[(
63   col      = normalTable[[colIndex]];
64   opLabel  = col[[1]];
65   opValues = col[[2 ;;]];
66   opMatrix = AssociationThread[matrixKeys -> opValues];
67   Do[(
68     opMatrix[Reverse[mKey]] = opMatrix[mKey]
69   ),
70   {mKey, matrixKeys}
71 ];
72   juddOperators[{3, opLabel}] = opMatrix,
73   {colIndex, 1, Length[normalTable]}
74 ];
75
76 (* special case of t2 in f3 *)
77 (* this is the same as getting the reduced matrix elements from
Judd 1966 *)
78 numE = 3;
79 e3Op    = juddOperators[{3, "e_{3}"}];
80 t2prime = juddOperators[{3, "t_{2}^{'}"}];
81 prefactor = 1/(70 Sqrt[2]);
82 t2Op = (# -> (t2prime[#] + prefactor*e3Op[#])) & /@ Keys[t2prime];
83 t2Op = Association[t2Op];
84 juddOperators[{3, "t_{2}"}] = t2Op;

```

```

84 (*Special case of t11 in f3*)
85 t11 = juddOperators[{3, "t_{11}"}];
86 eBetaPrimeOp = juddOperators[{3, "e_{\beta}^{\prime\prime}"}];
87 t11primeOp = (# -> (t11[#] + Sqrt[3/385] eBetaPrimeOp[#])) & /@ Keys[
88 t11];
89 t11primeOp = Association[t11primeOp];
90 juddOperators[{3, "t_{11}^{\prime\prime}"}] = t11primeOp;
91 If[OptionValue["Export"],
92 (
93 (*export them*)
94 PrintTemporary["Exporting ..."];
95 exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.m"}];
96 Export[exportFname, juddOperators];
97 )
98 ];
99 Return[juddOperators];
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 $\mathcal{C}_q^{(k)}$ under complex conjugation is considered, $\mathcal{C}_q^{(k)*} = (-1)^q \mathcal{C}_{-q}^{(k)}$, it is necessary that

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

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

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

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

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

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

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

From the $\langle \underline{\ell} \| \hat{C}^{(k)} \| \underline{\ell}' \rangle$, 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 $\langle \underline{\ell}^n \alpha SLJM_J | \hat{\mathcal{H}}_{\text{cf}} | \underline{\ell}'^n \alpha' SL'J'M_{J'} \rangle$ have the same parity, then the overall parity of the braket is determined by the parity of $C_q^{(k)}$, and since the parity of $C_q^{(k)}$ is $(-1)^k$ then for the braket to be non-zero we require that k should also be even. In view of this, in all the above equations for the crystal field the values for k should be limited to 2, 4, 6. The

value of $k = 0$ having been omitted from the start since this only contributes a common energy shift. Putting everything together:

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

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

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

```

```

7      Bqk[q, k] * (cqk + (-1)^q * cmqk) +
8      I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
9      ),
10     {k, {2, 4, 6}},
11     {q, 0, k}
12   ]
13 )

```

```

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).

\mathcal{S}_2 : Eqn-67	\mathcal{C}_s : Eqn-68	\mathcal{C}_{1h} : Eqn-69	\mathcal{C}_2 : Eqn-70	\mathcal{C}_{2h} : Eqn-71
\mathcal{C}_{2v} : Eqn-72	\mathcal{D}_2 : Eqn-73	\mathcal{D}_{2h} : Eqn-74	\mathcal{S}_4 : Eqn-75	\mathcal{C}_4 : Eqn-76
\mathcal{C}_{4h} : Eqn-77	\mathcal{D}_{2d} : Eqn-78	\mathcal{C}_{4v} : Eqn-79	\mathcal{D}_4 : Eqn-80	\mathcal{D}_{4h} : Eqn-81
\mathcal{C}_3 : Eqn-82	\mathcal{S}_6 : Eqn-83	\mathcal{C}_{3h} : Eqn-84	\mathcal{C}_{3v} : Eqn-85	\mathcal{D}_3 : Eqn-86
\mathcal{D}_{3d} : Eqn-87	\mathcal{D}_{3h} : Eqn-88	\mathcal{C}_6 : Eqn-89	\mathcal{C}_{6h} : Eqn-90	\mathcal{C}_{6v} : Eqn-91
\mathcal{D}_6 : Eqn-92	\mathcal{D}_{6h} : Eqn-93	\mathcal{T} : Eqn-94	\mathcal{T}_h : Eqn-95	\mathcal{T}_d : Eqn-96
\mathcal{O} : Eqn-97	\mathcal{O}_h : Eqn-98			

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

Crystal field expressions

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

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

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

$$\begin{aligned} \hat{\mathcal{H}}_{\text{cf}}(\mathcal{S}_6) = & B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_3^{(4)} \mathcal{C}_3^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} \\ & + \left(B_3^{(6)} + iS_3^{(6)} \right) \mathcal{C}_3^{(6)} + \left(B_6^{(6)} + iS_6^{(6)} \right) \mathcal{C}_6^{(6)} \end{aligned} \quad (83)$$

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

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

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_3) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_3^{(4)} \mathcal{C}_3^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_3^{(6)} \mathcal{C}_3^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (86)$$

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

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

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

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

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

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

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

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

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

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

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

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{O}_h) = B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} \quad (98)$$

(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 `HamMatrixAssembly`. Since the magnetic dipole operator is calculated in atomic units, and it seeming desirable that the input units of the magnetic field be Tesla, a conversion factor is included so that the final terms be congruent with the energy units assumed in the other terms in the Hamiltonian, namely the energy pseudo-unit Kayser (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.¹⁸

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 `HamMatrixAssembly`. In that option, the standard operator basis (i.e. the one use by Carnall *et al.* [Car+89]) is termed the *legacy* basis.

```
1 HamMatrixAssembly::usage = "HamMatrixAssembly[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."
```

¹⁸ 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[HamMatrixAssembly] = {
12   "FilenameAppendix" -> "",
13   "IncludeZeeman" -> False,
14   "Set t2Switch" -> True,
15   "ReturnInBlocks" -> False,
16   "OperatorBasis" -> "Legacy"};
17 HamMatrixAssembly[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       Print["Operator basis ", opBasis, " not recognized, using 'Legacy'", basis.];
25       opBasis = "Legacy";
26     ];
27     If[opBasis == "Orthogonal",
28       Print["Operator basis ''Orthogonal'', not implemented yet, aborting ..."];
29       Return[Null];
30     ];
31     (*#####
32     If[opBasis == "MostlyOrthogonal",
33       (
34       blockHam = HamMatrixAssembly[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     ];
81     (*#####
82     If[option == "ReturnInBlocks",
83       Return[Normal@blockHam];
84     ];
85     (*#####
86     If[option == "FlatForm",
87       Return[Flatten@blockHam];
88     ];
89     (*#####
90     If[option == "BlockForm",
91       Return@blockHam;
92     ];
93     (*#####
94     If[option == "Appendix",
95       Return@opBasis;
96     ];
97     (*#####
98     If[option == "IncludeZeeman",
99       Return@IncludeZeeman;
100    ];
101   ];
102 
```

```

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

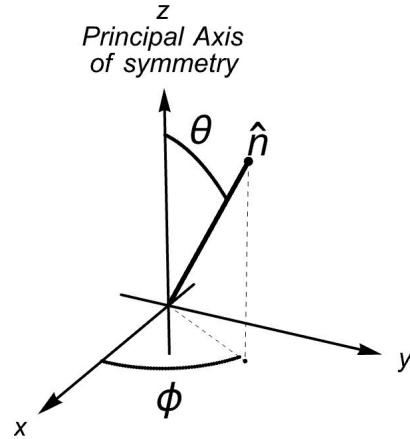
Before adding interactions lacking spherical symmetry, the orientation of the coordinate system is irrelevant. At the point when the crystal field is added, this orientation becomes relevant in the sense that only certain orientations of the coordinate system yield an expression for the crystal field potential in its simplest form¹⁹. To accomplish this the z-axis needs to be taken as one of the principal axis of symmetry²⁰. 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

¹⁹ 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 $B_q^{(k)}$ ²⁰ 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 χ^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 σ together with a chosen confidence interval p is then used to determine the contours of χ^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 1cm^{-1} . And when the data for magnetic dipole transitions was calculated, an uncertainty equal to the σ 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       Print["Invalid option for ''Units''. Options are ''SI'' and
51 ''Hartree''."];
52       Abort[];
53     )
54   ];
55 ]
56 )
57 ]

```

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 λ is the vacuum-equivalent wavelength of the transition between $|\nu\rangle$ and $|\nu'\rangle$, n the refractive index of the medium containing the ion, and g_i the degeneracy of the initial state $|\psi_i\rangle$. At the 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       Print["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 even²¹, 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-configuration²² 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

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

²² 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 ν is the frequency of the transition, χ 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 χ 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              Print["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 α 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 LaF₃ and LiYF₄.

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.²³

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).²⁴

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.

²³ 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). ²⁴ For example, in the [Table ??](#) for LaF₃, in the case of Yb, only two parameters are varied (ζ and ϵ) 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 LaF₃.
- `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.
- σ_{exp} : estimated uncertainty in the energy level differences between experimental and calculated values.
- `constraints`: a list of replacement rules defining constraints on the parameters. These 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 σ_{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^{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,
  problemVars, startValues, constraints] fits the given expData in
  an  $f^{\text{numE}}$  configuration, by using the symbols in problemVars. The
  symbols given in problemVars may be constrained or held constant,
  this being controlled by constraints list which is a list of
  replacement rules expressing desired constraints. The constraints
  list additional constraints imposed upon the model parameters that
  remain once other simplifications have been ''baked'' into the
  compiled Hamiltonians that are used to increase the speed of the
  calculation.
2
3 Important, note that in the case of odd number of electrons the given
  data must explicitly include the Kramers degeneracy;
  excludeDataIndices must be compatible with this.
4
5 The list expData needs to be a list of lists with the only
  restriction that the first element of them corresponds to energies
  of levels. In this list, an empty value can be used to indicate
  known gaps in the data. Even if the energy value for a level is
  known (and given in expData) certain values can be omitted from
  the fitting procedure through the list excludeDataIndices, which
  correspond to indices in expData that should be skipped over.
6

```

```

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 filider, 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 = HamMatrixAssembly[numE, "ReturnInBlocks" -> False];
330     theSimplifier = simplifier;
331     ham = Normal@ReplaceInSparseArray[ham, simplifier];
332     PrintFun["Compiling a function for the Hamiltonian with no
333 truncation ..."];
334     (* compile a function that will calculate the truncated
335     Hamiltonian given the parameters in allVars, this is the function
336     to be use in fitting *)
337     compileIntermediateTruncatedHam = Compile[Evaluate[allVars
338 ], Evaluate[ham]];
339     truncatedIntermediateBasis = SparseArray@IdentityMatrix[
340 Binomial[14, numE]];

```

```

326      (* save the compiled function *)
327      PrintFun["Saving the compiled function for the Hamiltonian
328      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
335          the block structure ..."];
336          ham           = HamMatrixAssembly[numE, "ReturnInBlocks" -> True];
337          (* apply the simplifier *)
338          PrintFun["Simplifying using the aggregate set of
339          simplification rules ..."];
340          ham           = Map[ReplaceInSparseArray[#, simplifier] &, ham,
341          {2}];
342          PrintFun["Zeroing out every symbol in the Hamiltonian that is
343          not a free-ion parameter ..."];
344          (* Get the free ion symbols *)
345          freeIonSimplifier = (# -> 0) & /@ Complement[
346          reducedModelSymbols, freeIonSymbols];
347          (* Take the diagonal blocks for the intermediate analysis *)
348          PrintFun["Grabbing the diagonal blocks of the Hamiltonian ...
349          "];
350          diagonalBlocks       = Diagonal[ham];
351          (* simplify them to only keep the free ion symbols *)
352          PrintFun["Simplifying the diagonal blocks to only keep the
353          free ion symbols ..."];
354          diagonalScalarBlocks = ReplaceInSparseArray[#, freeIonSimplifier] & /@ diagonalBlocks;
355          (* these include the MJ quantum numbers, remove that *)
356          PrintFun["Contracting the basis vectors by removing the MJ
357          quantum numbers from the diagonal blocks ..."];
358          diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
359
360          argsOfTheIntermediateEigensystems      = StringJoin[Riffle
361          [Prepend[(ToString[#] <> "v_") & /@ freeIonSymbols, "numE_"], ", ", "]];
362          argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle
363          [(ToString[#] <> "v") & /@ freeIonSymbols, ", "]];
364          PrintFun["argsOfTheIntermediateEigensystems = ",
365          argsOfTheIntermediateEigensystems];
366          PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
367          argsForEvalInsideOfTheIntermediateSystems];
368          PrintFun["(if the following fails, it might help to see if
369          the arguments of TheIntermediateEigensystems match the ones shown
370          above)"];
371
372          (* compile a function that will effectively calculate the
373          spectrum of all of the scalar blocks given the parameters of the
374          free-ion part of the Hamiltonian *)
375          (* compile one function for each of the blocks *)
376          PrintFun["Compiling functions for the diagonal blocks of the
377          Hamiltonian ..."];
378          compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate
379          [N[Normal[#]]] & /@ diagonalScalarBlocks;
380          (* use that to create a function that will calculate the free
381          -ion eigensystem *)
382          TheIntermediateEigensystems[numEv_, F0v_, F2v_, F4v_, F6v_,  $\zeta$ 
383          v_] := (
384              theNumericBlocks = (#[F0v, F2v, F4v, F6v,  $\zeta$ v] &) /@
385              compiledDiagonal;
386              theIntermediateEigensystems = Eigensystem /@
387              theNumericBlocks;
388              Js      = AllowedJ[numEv];
389              basisJ = BasisLSJM[numEv, "AsAssociation" -> True];
390              (* having calculated the eigensystems with the removed
391              degeneracies, put the degeneracies back in explicitly *)
392              elevatedIntermediateEigensystems = MapIndexed[EigenLever
393              [#1, 2Js[[#2[[1]]]] + 1] &, theIntermediateEigensystems];
394              (* Identify a single MJ to keep *)
395              pivot      = If[EvenQ[numEv], 0, -1/2];
396              LSJmultiplets = (#[[1]] <> ToString[InputForm[#[[2]]]]) & /
397              @Select[BasisLSJM[numEv], #[[{-1}]] == pivot &];
398              (* calculate the multiplet assignments that the
399              intermediate basis eigenvectors have *)
400

```

```

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];
447     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      ];
472      ];
473
474      truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
475      PrintFun["The truncated Hamiltonian has a dimension of ",
476      truncationUmbral, "x", truncationUmbral, "..."];
477      presentDataIndices = Select[presentDataIndices, # <=
478      truncationUmbral &];
479      solCompendium["presentDataIndices"] = presentDataIndices;
480
481      (* the problemVars are the symbols that will be fitted for *)
482
483      PrintFun["Starting up the fitting process using the Levenberg-
484      Marquardt method ..."];
485      (* using the problemVars I need to create the argument list
486      including _?NumericQ *)
487      problemVarsQ = (ToString[#] <> "_?NumericQ") & /@
488      problemVars;
489      problemVarsQString = StringJoin[Riffle[problemVarsQ, ", "]];
490      (* we also need to have the padded arguments with the variables
491      in the right position and the fixed values in the remaining ones
492      *)
493      problemVarsPositions = Position[allVars, #][[1, 1]] & /@
494      problemVars;
495      problemVarsString = StringJoin[Riffle[ToString /@ problemVars,
496      ", "]];
497      (* to feed parameters to the Hamiltonian, which includes all
498      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
478 (* this following function serves eigenvalues from the
479 Hamiltonian, has memoization so it might grow to use a lot of RAM
480 *)
481 Clear[HamSortedEigenvalues];
482 hamEigenvaluesTemplate = StringTemplate["
483 HamSortedEigenvalues['problemVarsQ']:=(
484     ham = compileIntermediateTruncatedHam@@
485 varsWithConstants';
486     eigenValues = Chop[Sort@Eigenvalues@ham];
487     eigenValues = eigenValues - Min[eigenValues];
488     HamSortedEigenvalues['problemVarsString'] = eigenValues;
489     Return[eigenValues]
490 )"];
491 hamString = hamEigenvaluesTemplate[<|
492     "problemVarsQ" -> problemVarsQString,
493     "varsWithConstants" -> varsWithConstantsString,
494     "problemVarsString" -> problemVarsString
495     |>];
496 ToExpression[hamString];
497
498 (* we also need a function that will pick the i-th eigenvalue,
499 this seems unnecessary but it's needed to form the right
500 functional form expected by the Levenberg-Marquardt method *)
501 eigenvalueDispenserTemplate = StringTemplate["
502 PartialHamEigenvalues['problemVarsQ'][i_]:=(
503     eigenVals = HamSortedEigenvalues['problemVarsString'];
504     eigenVals[[i]]
505 )"];
506 eigenValueDispenserString = eigenvalueDispenserTemplate[<|
507     "problemVarsQ" -> problemVarsQString,
508     "problemVarsString" -> problemVarsString
509     |>];
510 ToExpression[eigenValueDispenserString];
511
512 PrintFun["Determining the free variables after constraints ..."];
513 constrainedProblemVars = (problemVars /. constraints);
514 constrainedProblemVarsList = Variables[constrainedProblemVars];
515 If[addShift,
516     PrintFun["Adding a constant shift to the fitting parameters ...";
517     constrainedProblemVarsList = Append[constrainedProblemVarsList,
518     \[Epsilon]];
519 ];
520
521 indepVars = Complement[problemVars, #[[1]] & /@ constraints];
522 stringPartialVars = ToString/@constrainedProblemVarsList;
523
524 paramSols = {};
525 rmsHistory = {};
526 steps = 0;
527 problemVarsWithStartValues = KeyValueMap[{#1, #2} &, startValues];
528 If[addShift,
529     problemVarsWithStartValues = Append[problemVarsWithStartValues,
530     {\[Epsilon], 0}];
531 ];
532 openNotebooks = If[runningInteractive,
533     ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks[],
534     {}];
535
536 If[Not[MemberQ[openNotebooks, "Solver Progress"]] && OptionValue["ProgressView"],
537     ProgressNotebook["Basic"->False];
538 ];
539 degressOfFreedom = Length[presentDataIndices] - Length[
540 problemVars] - 1;
541 PrintFun["Fitting for ", Length[presentDataIndices], " data
542 points with ", Length[problemVars], " free parameters.", " The
543 effective degrees of freedom are ", degressOfFreedom, " ..."];
544
545

```

```

534 PrintFun["Fitting model to data ..."];
535 startTime = Now;
536 shiftToggle = If[addShift, 1, 0];
537 sol = FindMinimum[
538   Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
539     {j, presentDataIndices}],
540   problemVarsWithStartValues,
541   Method -> "LevenbergMarquardt",
542   MaxIterations -> OptionValue["MaxIterations"],
543   AccuracyGoal -> OptionValue["AccuracyGoal"],
544   StepMonitor :> (
545     steps += 1;
546     currentSqSum = Sum[(expData[[j]][[1]] - (
547       PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle *
548       \[Epsilon])^2, {j, presentDataIndices}];
549     currentRMS = Sqrt[currentSqSum / degreesOfFreedom];
550     paramSols = AddToList[paramSols, constrainedProblemVarsList,
551       maxHistory];
552     rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
553   )
554 ];
555 endTime = Now;
556 timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
557 PrintFun["Solution found in ", timeTaken, "s"];
558
559 solVec = constrainedProblemVars /. sol[[-1]];
560 indepSolVec = indepVars /. sol[[-1]];
561 If[addShift,
562   \[Epsilon]Best = \[Epsilon]/. sol[[-1]],
563   \[Epsilon]Best = 0
564 ];
565 fullSolVec = standardValues;
566 fullSolVec[[problemVarsPositions]] = solVec;
567 PrintFun["Calculating the truncated numerical Hamiltonian
corresponding to the solution ..."];
568 fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
569 PrintFun["Calculating energies and eigenvectors ..."];
570 {eigenEnergies, eigenVectors} = Eigensystem[fullHam];
571 states = Transpose[{eigenEnergies, eigenVectors}];
572 states = SortBy[states, First];
573 eigenEnergies = First /@ states;
574 PrintFun["Shifting energies to make ground state zero of energy
..."];
575 eigenEnergies = eigenEnergies - eigenEnergies[[1]];
576 PrintFun["Calculating the linear approximant to each eigenvalue
..."];
577 allVarsVec = Transpose[{allVars}];
578 p0 = Transpose[{fullSolVec}];
579 linMat = {};
580 If[addShift,
581   tail = -2,
582   tail = -1];
583 Do[
584   (
585     aVarPosition = Position[allVars, aVar][[1, 1]];
586     isolationValues = ConstantArray[0, Length[allVars]];
587     isolationValues[[aVarPosition]] = 1;
588     dependentVars = KeyValueMap[{#1, D[#2, aVar]} &, Association[
589       constraints]];
590     Do[
591       isolationValues[[Position[allVars, dVar[[1]]][[1, 1]]]] =
592       dVar[[2]],
593       {dVar, dependentVars}
594     ];
595     perHam = compileIntermediateTruncatedHam @@ isolationValues;
596     lin = FirstOrderPerturbation[Last /@ states, perHam];
597     linMat = Append[linMat, lin];
598   ),
599   {aVar, constrainedProblemVarsList[;;tail]}
600 ];
601 PrintFun["Removing the gradient of the ground state ..."];
602 linMat = (# - #[[1]] & /@ linMat);
603 PrintFun["Transposing derivative matrices into columns ..."];
604 linMat = Transpose[linMat];

```

```

601 PrintFun["Calculating the eigenvalue vector at solution ..."];
602 \[Lambda]0Vec = Transpose[{eigenEnergies[[presentDataIndices]]}];
603 PrintFun["Putting together the experimental vector ..."];
604 \[Lambda]exp = Transpose[{First /@ expData[[presentDataIndices
605 ]]}];
606 problemVarsVec = If[addShift,
607   Transpose[{constrainedProblemVarsList[[;; -2]]}],
608   Transpose[{constrainedProblemVarsList}]];
609 indepSolVecVec = Transpose[{indepSolVec}];
610 PrintFun["Calculating the difference between eigenvalues at
611 solution ..."];
612 diff = If[linMat == {},
613   (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best,
614   (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best + linMat[[presentDataIndices]].(problemVarsVec - indepSolVecVec)
615 ];
616 PrintFun["Calculating the sum of squares of differences around
617 solution ..."];
618 sqdiff = Expand[(Transpose[diff] . diff)[[1, 1]]];
619 PrintFun["Calculating the minimum (which should coincide with sol
620 ) ..."];
621 minpoly = sqdiff /. AssociationThread[problemVars -> solVec];
622 fmSolAssoc = Association[sol[[2]]];
623 If[\[Sigma]exp == Automatic,
624   \[Sigma]exp = Sqrt[minpoly / degressOfFreedom];
625 ];
626 \[CapitalDelta]\[Chi]2 = Sqrt[degressOfFreedom];
627 Amat = (1/\[Sigma]exp^2) * Transpose[linMat[[presentDataIndices
628 ]]].linMat[[presentDataIndices]];
629 paramIntervals = EllipsoidBoundingBox[Amat, \[CapitalDelta]\[Chi]2];
630 PrintFun["Calculating the uncertainty in the parameters ..."];
631 solWithUncertainty = Table[
632   (
633     aVar = constrainedProblemVarsList[[varIdx]];
634     paramBest = aVar /. fmSolAssoc;
635     (aVar -> {paramBest, paramIntervals[[varIdx, 2]]})
636   ),
637   {varIdx, 1, Length[constrainedProblemVarsList]-shiftToggle}
638 ];
639 bestRMS = Sqrt[minpoly / degressOfFreedom];
640 bestParams = sol[[2]];
641 bestWithConstraints = Association@Join[constraints, bestParams];
642 bestWithConstraints = bestWithConstraints /. bestWithConstraints;
643 bestWithConstraints = (# + 0.) & /@ bestWithConstraints;
644
645 solCompendium["degreesOfFreedom"] = degressOfFreedom;
646 solCompendium["solWithUncertainty"] = solWithUncertainty;
647 solCompendium["truncatedDim"] = truncationUmbral;
648 solCompendium["fittedLevels"] = Length[presentDataIndices];
649 solCompendium["actualSteps"] = steps;
650 solCompendium["bestRMS"] = bestRMS;
651 solCompendium["problemVars"] = problemVars;
652 solCompendium["paramSols"] = paramSols;
653 solCompendium["rmsHistory"] = rmsHistory;
654 solCompendium["Appendix"] = OptionValue["AppendToFile"];
655 solCompendium["timeTaken/s"] = timeTaken;
656 solCompendium["bestParams"] = bestParams;
657 solCompendium["bestParamsWithConstraints"] = bestWithConstraints;
658
659 If[OptionValue["SaveEigenvectors"],
660   solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]}
661 &/@ (Chop /@ ShiftedLevels[states]),
662   (
663     finalEnergies = Sort[First /@ states];
664     finalEnergies = finalEnergies - finalEnergies[[1]];
665     finalEnergies = finalEnergies + \[Epsilon]Best;
666     finalEnergies = Chop /@ finalEnergies;
667     solCompendium["energies"] = finalEnergies;
668   )
669 ];

```

```

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.nb`: gives an overview of functions included in `qlanth`.
- `/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 `HamMatrixAssembly` 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 LaF3 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 LaF3:Ln³⁺ and LiYF4:Ln³⁺

The study of Carnall *et al.* [[Car+89](#)] on lanthanum fluoride was a systematic review of trivalent lanthanide ions in LaF3. In this work they fitted the experimental data for all of the lanthanide ions using the single-configuration effective Hamiltonian. In their appendices one can find their calculated values, together with the experimental values that they used for their least squares fittings. In `qlanth` this data can be 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 LaF3. 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 LaF3.

Similarly, Cheng *et al.* [[Che+16](#)] compiled data for LiYF4. In `qlanth` model parameters for LiYF4 can be obtained by calling the function `LoadLiYF4Parameters`. [Fig-7](#) shows the results of a calculation done with `qlanth` for the energy levels in LiYF4.

```

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 );

```

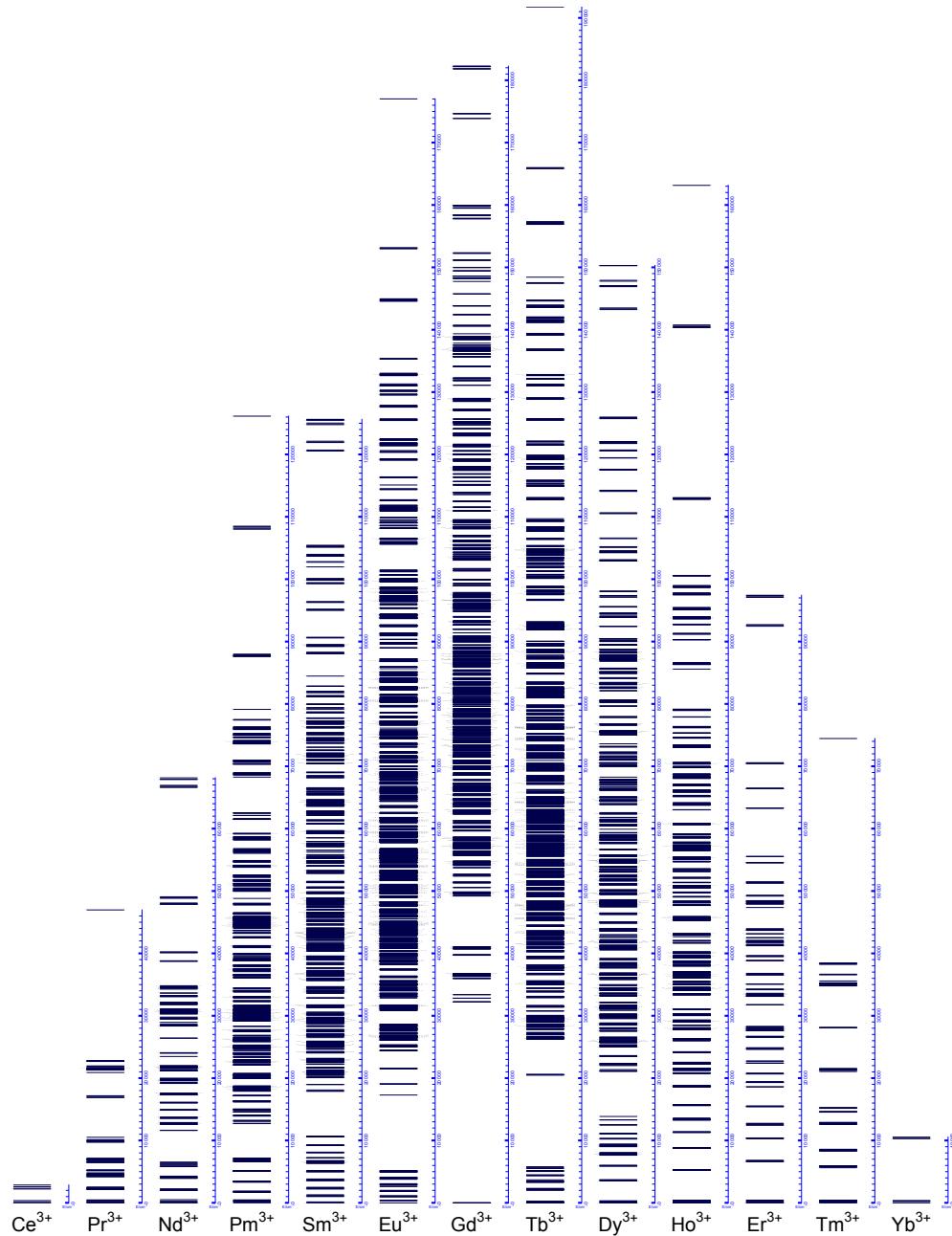


Figure 6: Energy levels in LaF_3 .

```

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

```

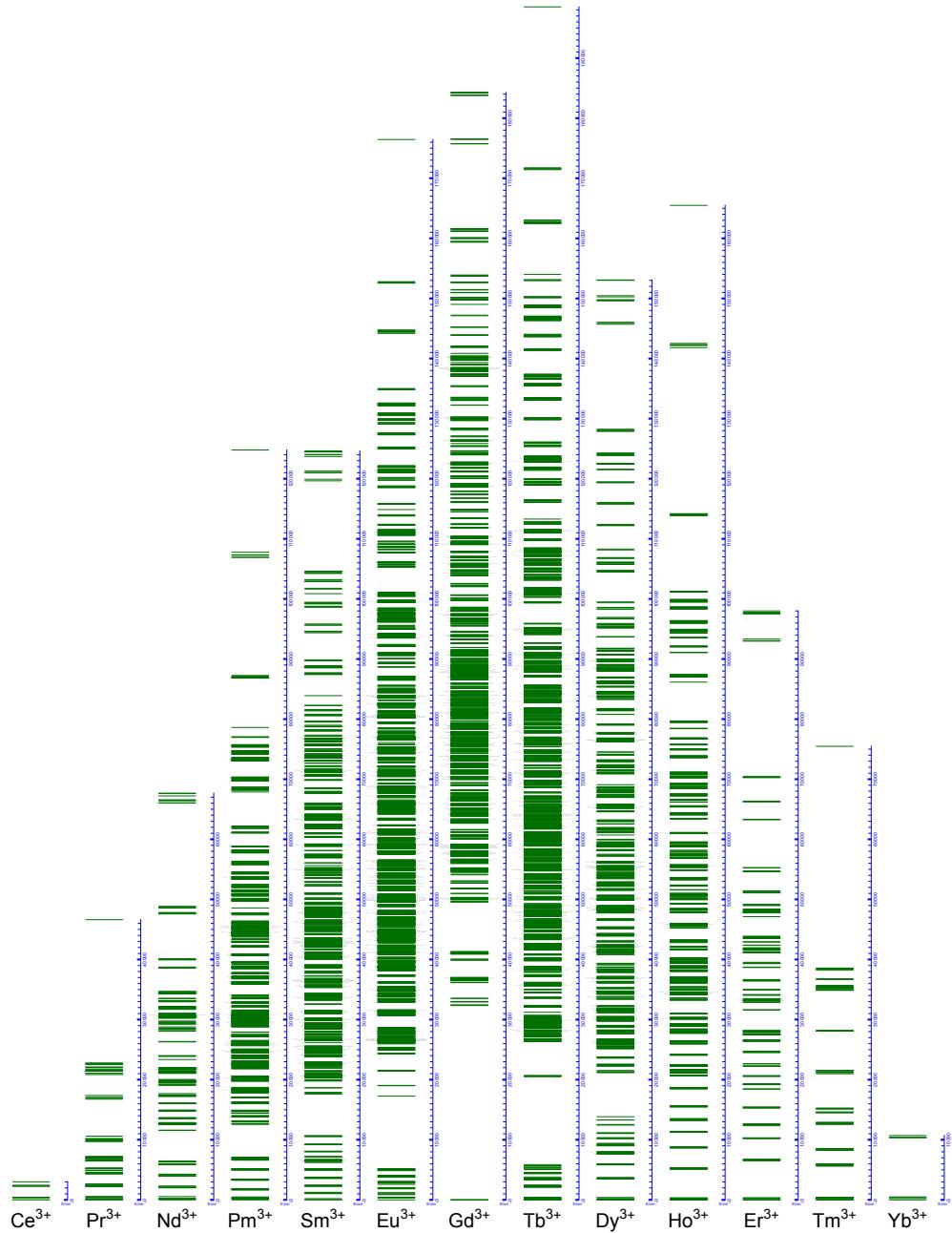


Figure 7: Energy levels in LiYF_4 .

```

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["DavidLizarazo`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 = "SPDFGHJKLMNOQRTUV";
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 hBar = hPlanck / (2 \[Pi]); (* reduced Planck's constant
16   in J s *)
17 \[Mu]B = 9.2740100783 * 10^-24; (* Bohr magneton in SI *)
18 me = 9.1093837015 * 10^-31; (* electron mass in kg *)
19 cLight = 2.99792458 * 10^8; (* speed of light in m/s *)
20 eCharge = 1.602176634 * 10^-19; (* elementary charge in SI *)
21 \[Epsilon]0 = 8.8541878128 * 10^-12; (* electric permittivity in
22   vacuum in SI *)
23 \[Mu]0 = 4 \[Pi] * 10^-7; (* magnetic permeability in
24   vacuum in SI *)
25 \[Alpha]Fine = 1/137.036; (* fine structure constant *)
26
27 bohrRadius = 5.29177*10^-11; (* Bohr radius in m *)
28 hartreeEnergy = hBar^2 / (me * bohrRadius^2); (* Hartree energy in J *)
29 hartreeTime = hBar / hartreeEnergy; (* Hartree time in s *)
30
31 (* Hartree atomic units *)
32 hPlanckHartree = 2 \[Pi]; (* Planck's constant in Hartree *)
33 meHartree = 1; (* electron mass in Hartree *)
34 cLightHartree = 137.036; (* speed of light in Hartree *)
35 eChargeHartree = 1; (* elementary charge in Hartree *)
36 \[Mu]0Hartree = \[Alpha]Fine^2; (* magnetic permeability in vacuum in
37   Hartree *)
38
39 (* 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

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

total orbital angular momentum operator

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

spin angular momentum operator of a single electron

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

total spin angular momentum operator

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

Shorthand for all other quantum numbers

$$\underline{\Lambda} \quad (125)$$

orbital angular momentum number

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

spinning angular momentum number

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

Coulomb non-central potential

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

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

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

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

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

Spectroscopic term αLS in Russel-Saunders notation

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

spherical tensor operator of rank k

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

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

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

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

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

16 Mathematica paclet

The simplest way of adding **qlanth** to *Mathematica* is by installing the corresponding paclet. After having cloned the repository to a local machine, execute the code below in a *Mathematica* session. After this, the downloaded files may be removed, and **qlanth** may be loaded into a notebook by issuing the command `Needs["DavidLizarazo`qlanth`"]`.

```

Needs["PacletTools`"];
pathToRepo = "~/Downloads/qlanth/"; (* change accordingly *)
pathToBuild = FileNameJoin[{pathToRepo, "build", "DavidLizarazo__qlanth"}];
CreatePacletArchive[pathToBuild];
pathToPaclet = FileNames[

```

```
FileNameJoin[{pathToRepo, "build", "/*.paclet"}]][[1]];
PacletInstall[pathToPaclet, ForceVersionInstall -> True];
```

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

17 Definitions

$$\overbrace{[x]}^{\text{two plus one}} := 2x + 1 \quad (135)$$

$$\text{irreducible unit tensor operator of rank } k \quad \overbrace{\hat{u}^{(k)}}^{\square} \quad (136)$$

$$\text{symmetric unit tensor operator for } n \text{ equivalent electrons} \quad \overbrace{\hat{U}^{(k)}}^{\square} := \sum_{i=1}^n \hat{u}^{(k)} \quad (137)$$

$$\text{Renormalized spherical harmonics} \quad \overbrace{\mathcal{C}_q^{(k)}}^{\square} := \sqrt{\frac{4\pi}{2k+1}} Y_q^{(k)} \quad (138)$$

$$\text{Triangle "delta" between } j_1, j_2, j_3 \quad \overbrace{\triangle(j_1, j_2, j_3) := \begin{cases} 1 & \text{if } j_1 = (j_2 + j_3), (j_2 + j_3 - 1), \dots, |j_2 - j_3| \\ 0 & \text{otherwise} \end{cases}}^{\square} \quad (139)$$

(a) qlanth guide in *Mathematica*

(b) Documentation for HamMatrixAssembly

(c) Doc. for BasisLSJMJ

(d) Doc. for ToPythonSymPyExpression

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["DavidLizarazo`qlanth`"];
137 Get[FileNameJoin[{DirectoryName[$InputFileName], "qonstants.m"}]]
138 Get[FileNameJoin[{DirectoryName[$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 ap: orthogonalized  $\alpha$ 
239 bp: orthogonalized  $\beta$ 
240 gp: orthogonalized  $\gamma$ 
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,  $\zeta$ };
258 casimirSymbols = { $\alpha$ ,  $\beta$ ,  $\gamma$ };
259 juddOfeltIntensitySymbols = {\[CapitalOmega]2, \[CapitalOmega]4, \[CapitalOmega]6};
260 mostlyOrthogonalSymbols = Join[{E0p, E1p, E2p, E3p, ap, bp, gp},
   {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 AllowedNKSLSJMforJMTerms;
287 AllowedNKSLSJMforJTerms;
288 AllowedNKSLSJTerms;
289
290 AllowedNKSLSTerms;
291 AllowedNKSLSforJTerms;
292 AllowedSLJMTerms;
293 AllowedSLJTerms;
294 AllowedSLTerms;
295
296 AngularMomentumMatrices;
297 BasisLSJ;
298 BasisLSJMJ;
299 Bqk;
300 CFP;
301
302 CFPAssoc;
303 CFPTable;
304 CFPTerms;
305 Carnall;
306 CasimirG2;
307
308 CasimirS03;
309 CasimirS07;
310 Ck;
311 Cqk;
312 CrystalField;
313 CrystalFieldForm;
314
315 Dk;
316 EigenLever;
317 Electrostatic;
318 ElectrostaticConfigInteraction;
319 ElectrostaticTable;
320
321 EnergyLevelDiagram;
322 EnergyStates;
323 EtoF;
324 ExportMZip;
325
326 FindNKLSTerm;
327 FindSL;
328 FreeHam;
329 FreeIonTable;
330 FromArrayToTable;
331 FtoE;
332
333 GG2U;
334 GS07W;
335 GenerateCFP;
336 GenerateCFPAssoc;
337 GenerateCFPTable;
338 GenerateCrystalFieldTable;
339 GenerateElectrostaticTable;
340 GenerateFreeIonTable;
341 GenerateReducedUkTable;
342 GenerateReducedV1kTable;
343 GenerateSOOandECSOLSTable;
344
345 GenerateSOOandECSOTable;
346 GenerateSpinOrbitTable;
347 GenerateSpinSpinTable;
348 GenerateT22Table;
349 GenerateThreeBodyTables;

```

```

350 GroundMagDipoleOscillatorStrength;
351 HamMatrixAssembly;
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 LevelSimplerSymbolicHamMatrix;
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 ScalarLSJMFfromLS;
437 SOOandECSO;
438 SOOandECSOLSTable;
439
440 SOOandECSOTable;
441 Seniority;
442 ShiftedLevels;
443 SimplerSymbolicHamMatrix;
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=1'.";
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   Print["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       Print["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   Print["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       Print["Exporting to file " <> ToString[exportFname]];
752     )
753   ];
754 }

```

```

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 (* ##### #####
939 (* ##### #####
940 (* ##### Coefficients of Fracional Parentage #####
941
942 GenerateCFP::usage = "GenerateCFP[] generates the association for
943   the coefficients of fractional parentage. Result is exported to
944   the file ./data/CFP.m. The coefficients of fractional parentage
945   are taken beyond the half-filled shell using the phase convention
946   determined by the option \"PhaseFunction\". The default is \"NK\""
947   which corresponds to the phase convention of Nielson and Koster.
948   The other option is \"Judd\" which corresponds to the phase
949   convention of Judd.";
950 Options[GenerateCFP] = {"Export" -> True, "PhaseFunction" -> "NK"};
951 GenerateCFP[OptionsPattern[]] := (
952   CFP = Table[
953     {numE, NKSL} -> First[CFPTerms[numE, NKSL]],
954     {numE, 1, 7},
955     {NKSL, AllowedNKSLTerms[numE]}];
956   CFP = Association[CFP];
957   (* Go all the way to f14 *)
958   CFP = CFPExander["Export" -> False, "PhaseFunction" ->
959   OptionValue["PhaseFunction"]];
960   If[OptionValue["Export"],
961     Export[FileNameJoin[{moduleDir, "data", "CFPs.m"}], CFP];
962   ];
963   Return[CFP];
964 );
965
966 JuddCFPPPhase::usage = "Phase between conjugate coefficients of
967   fractional parentage according to Velkov's thesis, page 40.";
968 JuddCFPPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
969   parentSeniority_, daughterSeniority_] := Module[
970   {spin, orbital, expo, phase},
971   (
972     {spin, orbital} = {1/2, 3};
973     expo = (
974       (parentS + parentL + daughterS + daughterL) -
975       (orbital + spin) +
976       1/2 * (parentSeniority + daughterSeniority - 1)
977     );
978     phase = Phaser[-expo];
979     Return[phase];
980   );
981 ];
982
983 NKCFPPPhase::usage = "Phase between conjugate coefficients of
984   fractional parentage according to Nielson and Koster page viii.
985   Note that there is a typo on there the expression for zeta should
986   be  $(-1)^{(v-1)/2}$  instead of  $(-1)^{v - 1/2}$ .";
987 NKCFPPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
988   parentSeniority_, daughterSeniority_] := Module[
989   {spin, orbital, expo, phase},
990   (
991     {spin, orbital} = {1/2, 3};
992     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   Print["Exporting to ", exportFname];
1061   Export[exportFname, extendedCFPs];
1062 )
1063 ];
1064 Return[extendedCFPs];
1065 )
1066 ];
1067 GenerateCFPTable::usage = "GenerateCFPTable[] generates the table
for the coefficients of fractional parentage. If the optional
parameter \"Export\" is set to True then the resulting data is
saved to ./data/CFPTable.m.
The data being parsed here is the file attachment B1F_ALL.TXT which
comes from Velkov's thesis.";
1069 Options[GenerateCFPTable] = {"Export" -> True};
1070 GenerateCFPTable[OptionsPattern[]}]:=Module[
1071 {rawText, rawLines, leadChar, configIndex, line, daughter,
1072 lineParts, numberCode, parsedNumber, toAppend, CFPTablefname},
1073 (
1074   CleanWhitespace[string_] := StringReplace[string,
1075 RegularExpression["\\s+"]->" "];
1076   AddSpaceBeforeMinus[string_] := StringReplace[string,
1077 RegularExpression["(?!\s)-"]->" -"];
1078   ToIntegerOrString[list_] := Map[If[StringMatchQ[#, NumberString], ToExpression[#], #] &, list];
1079   CFPTable = ConstantArray[{}, 7];
1080   CFPTable[[1]] = {{"2F", {"1S", 1}}};
1081
1082   (* Cleaning before processing is useful *)
1083   rawText = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
1084
1085   rawLines = StringTrim/@StringSplit[rawText, "\n"];
1086   rawLines = Select[rawLines, #!="&"];
1087   rawLines = CleanWhitespace/@rawLines;
1088   rawLines = AddSpaceBeforeMinus/@rawLines;
1089
1090   Do[(
1091     (* the first character can be used to identify the start of a
1092     block *)
1093     leadChar=StringTake[line,{1}];
1094     (* ..FN, N is at position 50 in that line *)
1095     If[leadChar=="[",
1096     (
1097       configIndex=ToExpression[StringTake[line,{50}]];
1098       Continue[];
1099     )
1100   ];
1101   (* Identify which daughter term is being listed *)
1102   If[StringContainsQ[line, "[DAUGHTER TERM]"],
1103     daughter=StringSplit[line, "["[[1]];
1104     CFPTable[[configIndex]]=Append[CFPTable[[configIndex]],{daughter}];
1105     Continue[];
1106   ];
1107   (* Once we get here we are already parsing a row with
1108   coefficient data *)
1109   lineParts = StringSplit[line, " "];
1110   parent = lineParts[[1]];
1111   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  $\zeta$  <SL, J|L.S|SpLp, J>. 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."}]];
1246   If[OptionValue["Export"],
1247     (
1248       Print["Exporting to file "<>ToString[exportFname]];
1249       Export[exportFname, SpinOrbitTable];
1250     )
1251   ];
1252   Return[SpinOrbitTable];
1253 ]
1254 
1255 (* ##### Spin Orbit #####
1256 (* ##### #####
1257 (* ##### #####
1258 (* ##### #####
1259 (* ##### Three Body Operators #####
1260 
1261 ParseJudd1984::usage = "This function parses the data from tables 1
1262 and 2 of Judd from Judd, BR, and MA Suskin. \"Complete Set of
1263 Orthogonal Scalar Operators for the Configuration f^3\". JOSA B 1,
1264 no. 2 (1984): 261-65.";
1265 Options[ParseJudd1984] = {"Export" -> False};
1266 ParseJudd1984[OptionsPattern[]] :=
1267 ParseJuddTab1[str_] :=
1268   ParseJuddTab1[str_] := (
1269     strR = ToString[str];
1270     strR = StringReplace[strR, ".5" -> "^(1/2)"];
1271     num = ToExpression[strR];
1272     sign = Sign[num];
1273     num = sign*Simplify[Sqrt[num^2]];
1274     If[Round[num] == num, num = Round[num]];
1275     Return[num]);
1276 
1277 (* Parse table 1 from Judd 1984 *)
1278 judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"}];
1279 data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
1280 headers = data[[1]];
1281 data = data[[2 ;;]];
1282 data = Transpose[data];
1283 \[Psi] = Select[data[[1]], # != "" &];
1284 \[Psi]p = Select[data[[2]], # != "" &];
1285 matrixKeys = Transpose[{\[Psi], \[Psi]p}];
1286 data = data[[3 ;;]];
1287 cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data}];
1288 cols = Select[cols, Length[#] == 21 &];
1289 tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
1290 tab1 = Transpose[Prepend[Transpose[tab1], headers]];
1291 
1292 (* Parse table 2 from Judd 1984 *)
1293 judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"}];
1294 data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
1295 headers = data[[1]];
1296 data = data[[2 ;;]];
1297 data = Transpose[data];
1298 {operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} =
1299 data[[;; 4]];
1300 multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
1301 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 eBetaPrimeOp = juddOperators[{3, "e_{\beta}^{'}"}];
1346 t11primeOp = (# -> (t11[#] + Sqrt[3/385] eBetaPrimeOp[#])) & /@ 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     ];
1442 
```

```

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         Print["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       Print["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[
{fname, crystalSym,
crystalSymmetries, parseFun,
chars, qk, groupName, family,
groupNum, params},
(
2186 fname      = FileNameJoin[{moduleDir,"data",
benelli_and_gatteschi_table3p3.csv}];
2187 crystalSym = Import[fname][[2;;33]];
2188 crystalSymmetries = <||>;
2189 parseFun[txt_] := (
2190   chars = Characters[txt];
2191   qk    = chars[[-2;;]];
2192   If[chars[[1]]=="R",
2193   (
2194     Return[{ToExpression@StringJoin[{"B",qk[[1]],qk[[2]]}]}]
2195   ),
2196   (
2197     If[qk[[1]]=="O",
2198       Return[{ToExpression@StringJoin[{"B",qk[[1]],qk[[2]]}]}]
2199     ];
2200     Return[{
2201       ToExpression@StringJoin[{"B",qk[[1]],qk[[2]]}],
2202       ToExpression@StringJoin[{"S",qk[[1]],qk[[2]]}]
2203     }]
2204   ])
2205 );
2206 Do[
2207 (
2208   groupNum  = Round@ToExpression@row[[1]];
2209   groupName = row[[2]];

```

```

2211 family      = row[[3]];
2212 params      = Select[row[[4;;]], #!=!"&"];
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[{SL, SpLp}] returns the matrix
2343     element for configuration interaction as approximated by the
2344     Casimir operators of the groups R3, G2, and R7. SL and SpLp are
2345     strings that represent terms under LS coupling.";
```

```

2330 ElectrostaticConfigInteraction[{SL_, SpLp_}] := Module[
2331 {S, L, val},
2332 (
2333 {S, L} = FindSL[SL];
2334 val = (
2335 If[SL == SpLp,
2336 CasimirS03[{SL, SL}] +
2337 CasimirS07[{SL, SL}] +
2338 CasimirG2[{SL, SL}],
2339 0
2340 ]
2341 );
2342 ElectrostaticConfigInteraction[{S, L}] = val;
2343 Return[val];
2344 )
2345 ];
2346
2347 (* ##### Configuration-Interaction via Casimir Operators ##### *)
2348 (* ##### Block assembly ##### *)
2349
2350 (* ##### Block assembly ##### *)
2351 (* ##### Block assembly ##### *)
2352
2353 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.";
2354 Options[JJBlockMatrix] = {"Sparse" -> True, "ChenDeltas" -> False};
2355 JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]] := Module[
2356 {
2357 {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
2358 SLterm, SpLpterm,
2359 MJ, MJp,
2360 subKron, matValue, eMatrix},
2361 (
2362 NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
2363 NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
2364 eMatrix =
2365 Table[
2366 (*Condition for a scalar matrix op*)
2367 SLterm = NKSLJM[[1]];
2368 SpLpterm = NKSLJMp[[1]];
2369 MJ = NKSLJM[[3]];
2370 MJp = NKSLJMp[[3]];
2371 subKron = (
2372 KroneckerDelta[J, Jp] *
2373 KroneckerDelta[MJ, MJp]
2374 );
2375 matValue =
2376 If[subKron == 0,
2377 0,
2378 (
2379 ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2380 ElectrostaticConfigInteraction[{SLterm, SpLpterm}]
2381 +
2382 SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2383 MagneticInteractions[{numE, SLterm, SpLpterm, J},
2384 "ChenDeltas" -> OptionValue["ChenDeltas"]] +
2385 ThreeBodyTable[{numE, SLterm, SpLpterm}]
2386 )
2387 ];
2388 matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp}
2389 ];
2390 matValue,
2391 {NKSLJMp, NKSLJMps},
2392 {NKSLJM, NKSLJMs}
2393 ];
2394 If[OptionValue["Sparse"],
2395 eMatrix = SparseArray[eMatrix]
2396 ];
2397 Return[eMatrix]
2398 )
2399 ];

```

```

2397 EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some
2398   point may be used to redefine states used in basis.";
2399 EnergyStates[numE_, J_]:= AllowedNKSLJMforJTerms[numE, J];
2400
2401 JJBlockMatrixFileName::usage = "JJBlockMatrixFileName[numE] gives
2402   the filename for the energy matrix table for an atom with numE f-
2403   electrons. The function admits an optional parameter \
2404   FilenameAppendix\\" which can be used to modify the filename.";
2405 Options[JJBlockMatrixFileName] = {"FilenameAppendix" -> ""};
2406 JJBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
2407   fileApp = OptionValue["FilenameAppendix"];
2408   fname = FileNameJoin[{moduleDir,
2409     "hams",
2410     StringJoin[{"f", ToString[numE], "_JJBlockMatrixTable",
2411     fileApp, ".m"}]}];
2412   Return[fname];
2413 );
2414
2415 TabulateJJBlockMatrixTable::usage = "TabulateJJBlockMatrixTable[
2416   numE, CFTable] returns an association JJBlockMatrixTable with keys
2417   equal to lists of the form {numE, J, Jp} and values equal to JJ
2418   blocks of the semi-empirical Hamiltonian. The function admits an
2419   optional parameter \"Sparse\" which can be used to control whether
2420   the matrix is returned as a SparseArray or not. The default is
2421   True. Another admitted option is \"ChenDeltas\" which can be used
2422   to include or exclude the Chen deltas from the calculation. The
2423   default is to exclude them. If this option is used, then the
2424   chenDeltas association needs to be loaded into the session with
2425   LoadChenDeltas[].";
2426 Options[TabulateJJBlockMatrixTable] = {"Sparse" -> True, "ChenDeltas" -> False};
2427 TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]] := (
2428   JJBlockMatrixTable = <||>;
2429   totalIterations = Length[AllowedJ[numE]]^2;
2430   template1 = StringTemplate["Iteration `numiter` of `totaliter`"]
2431   template2 = StringTemplate["`remtime` min remaining"];
2432   template4 = StringTemplate["Time elapsed = `runtime` min"];
2433   numiter = 0;
2434   startTime = Now;
2435   If[$FrontEnd != Null,
2436     (
2437       temp = PrintTemporary[
2438         Dynamic[
2439           Grid[
2440             {
2441               {template1[<|"numiter" -> numiter, "totaliter" ->
2442                 totalIterations|>]},
2443               {template2[<|"remtime" -> Round[QuantityMagnitude[
2444                 UnitConvert[(Now - startTime)/(Max[1, numiter])*(totalIterations -
2445                 numiter), "min"]], 0.1]|>}},
2446               {template4[<|"runtime" -> Round[QuantityMagnitude[
2447                 UnitConvert[(Now - startTime), "min"], 0.1]|>]},
2448               {ProgressIndicator[numiter, {1, totalIterations}]}
2449             ]
2450           ]
2451         ];
2452       ];
2453     );
2454   ];
2455   Do[
2456     (
2457       JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp,
2458       CFTable, "Sparse" -> OptionValue["Sparse"], "ChenDeltas" ->
2459       OptionValue["ChenDeltas"]];
2460       numiter += 1;
2461     ),
2462     {Jp, AllowedJ[numE]},
2463     {J, AllowedJ[numE]}
2464   ];
2465   If[$FrontEnd != Null,
2466     NotebookDelete[temp]
2467   ];
2468   Return[JJBlockMatrixTable];
2469 );
2470

```

```

2451 TabulateManyJJBlockMatrixTables::usage = "
2452   TabulateManyJJBlockMatrixTables[{n1, n2, ...}] calculates the
2453   tables of matrix elements for the requested f^n_i configurations.
2454   The function does not return the matrices themselves. It instead
2455   returns an association whose keys are numE and whose values are
2456   the filenames where the output of TabulateJJBlockMatrixTables was
2457   saved to. The output consists of an association whose keys are of
2458   the form {n, J, Jp} and whose values are rectangular arrays given
2459   the values of <|LSJMJa|H|L'S'J'MJ'a'|>.";
2460 Options[TabulateManyJJBlockMatrixTables] = {"Overwrite" -> False, "
2461   Sparse" -> True, "ChenDeltas" -> False, "FilenameAppendix" -> "", "
2462   Compressed" -> False};
2463 TabulateManyJJBlockMatrixTables[ns_, OptionsPattern[]] := (
2464   overwrite = OptionValue["Overwrite"];
2465   fNames = <||>;
2466   fileApp = OptionValue["FilenameAppendix"];
2467   ExportFun = If[OptionValue["Compressed"], ExportMZip, Export];
2468   Do[
2469     (
2470      CFdataFilename = FileNameJoin[{moduleDir, "data", "CrystalFieldTable_f"} <> ToString[numE] <> ".zip"];
2471      PrintTemporary["Importing CrystalFieldTable from ", CFdataFilename, " ..."];
2472      CrystalFieldTable = ImportMZip[CFdataFilename];
2473
2474      PrintTemporary["#----- numE = ", numE, " -----#"];
2475      exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix"
2476      -> fileApp];
2477      fNames[numE] = exportFname;
2478      If[FileExistsQ[exportFname] && Not[overwrite],
2479        Continue[]
2480      ];
2481      JJBlockMatrixTable = TabulateJJBlockMatrixTable[numE,
2482      CrystalFieldTable, "Sparse" -> OptionValue["Sparse"], "ChenDeltas"
2483      -> 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 HamMatrixAssembly::usage = "HamMatrixAssembly[numE] returns the
2497   Hamiltonian matrix for the f^numE configuration. The matrix is
2498   returned as a SparseArray.
2499 The function admits an optional parameter \"FilenameAppendix\"
2500   which can be used to control which variant of the JJBlocks is used
2501   to assemble the matrix.
2502 It also admits an optional parameter \"IncludeZeeman\"
2503   which can be used to include the Zeeman interaction. The default is False.
2504 The option \"Set t2Switch\" can be used to toggle on or off setting
2505   the t2 selector automatically or not, the default is True, which
2506   replaces the parameter according to numE.
2507 The option \"ReturnInBlocks\" can be used to return the matrix in
2508   block or flattened form. The default is to return it in flattened
2509   form.";
2510 Options[HamMatrixAssembly] = {
2511   "FilenameAppendix" -> "",
2512   "IncludeZeeman" -> False,
2513   "Set t2Switch" -> True,
2514   "ReturnInBlocks" -> False,
2515   "OperatorBasis" -> "Legacy"};
2516 HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
2517   {numE, ii, jj, howManyJs, Js, blockHam, opBasis},
2518   (
2519     (*#####
2520     ImportFun = ImportMZip;
2521     opBasis = OptionValue["OperatorBasis"];
2522     If[Not[MemberQ[{ "Legacy", "MostlyOrthogonal", "Orthogonal"}, opBasis]],
2523       opBasis],
2524     Print["Operator basis ", opBasis, " not recognized, using \"
2525

```

```

2502     Legacy\" basis."];
2503     opBasis = "Legacy";
2504   ];
2505   If[opBasis == "Orthogonal",
2506     Print["Operator basis \"Orthogonal\" not implemented yet,
2507     aborting ..."];
2508   ];
2509   (*#####
2510   If[opBasis == "MostlyOrthogonal",
2511     (
2512       blockHam = HamMatrixAssembly[nf,
2513       "OperatorBasis" -> "Legacy",
2514       "FilenameAppendix" -> OptionValue["FilenameAppendix"],
2515       "IncludeZeeman" -> OptionValue["IncludeZeeman"],
2516       "Set t2Switch" -> OptionValue["Set t2Switch"],
2517       "ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
2518       paramChanger = Which[
2519         nf < 7,
2520         <|
2521           F0 -> 1/91 (54 E1p+91 E0p+78 γp),
2522           F2 -> (15/392 *
2523             (
2524               140 E1p +
2525               20020 E2p +
2526               1540 E3p +
2527               770 αp -
2528               70 γp +
2529               22 Sqrt[2] T2p -
2530               11 Sqrt[2] nf T2p t2Switch -
2531               11 Sqrt[2] (14-nf) T2p (1 - t2Switch)
2532             )
2533           ),
2534           F4 -> (99/490 *
2535             (
2536               70 E1p -
2537               9100 E2p +
2538               280 E3p +
2539               140 αp -
2540               35 γp +
2541               4 Sqrt[2] T2p -
2542               2 Sqrt[2] nf T2p t2Switch -
2543               2 Sqrt[2] (14-nf) T2p (1-t2Switch)
2544             )
2545           ),
2546           F6 -> (5577/7000 *
2547             (
2548               20 E1p +
2549               700 E2p -
2550               140 E3p -
2551               70 αp -
2552               10 γp -
2553               2 Sqrt[2] T2p +
2554               Sqrt[2] nf T2p t2Switch +
2555               Sqrt[2] (14-nf) T2p (1-t2Switch)
2556             )
2557           ),
2558           ζ -> ζ,
2559           α -> (5 αp)/4,
2560           β -> -6 (5 αp + βp),
2561           γ -> 5/2 (2 βp + 5 γp),
2562           T2 -> 0
2563         |>,
2564         nf >= 7,
2565         <|
2566           F0 -> 1/91 (54 E1p+91 E0p+78 γp),
2567           F2 -> (15/392 *
2568             (
2569               140 E1p +
2570               20020 E2p +
2571               1540 E3p +
2572               770 αp -
2573               70 γp +
2574               22 Sqrt[2] T2p -
2575               11 Sqrt[2] nf T2p
2576             )
2577           )

```

```

2576 ),
2577 F4 -> (99/490 *
2578 (
2579   70 E1p -
2580   9100 E2p +
2581   280 E3p +
2582   140 αp -
2583   35 γp +
2584   4 Sqrt[2] T2p -
2585   2 Sqrt[2] nf T2p
2586 )
2587 ),
2588 F6 -> (5577/7000 *
2589 (
2590   20 E1p +
2591   700 E2p -
2592   140 E3p -
2593   70 αp -
2594   10 γp -
2595   2 Sqrt[2] T2p +
2596   Sqrt[2] nf T2p
2597 )
2598 ),
2599 ζ -> ζ,
2600 α -> (5 αp)/4,
2601 β -> -6 (5 αp + βp),
2602 γ -> 5/2 (2 βp + 5 γp),
2603 T2 -> 0
2604 |>
2605 ];
2606 blockHamMO = Which[
2607   OptionValue["ReturnInBlocks"] == False,
2608   ReplaceInSparseArray[blockHam, paramChanger],
2609   OptionValue["ReturnInBlocks"] == True,
2610   Map[ReplaceInSparseArray[#, paramChanger]&,amp;, blockHam,
2611 {2}]
2612 ];
2613 Return[blockHamMO];
2614 ];
2615 (*#####
2616 (*hole-particle equivalence enforcement*)
2617 numE = nf;
2618 allVars = {E0, E1, E2, E3, ζ, F0, F2, F4, F6, M0, M2, M4, T2,
2619 T2p,
2620   T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
2621   α, β, γ, B02, B04, B06, B12, B14, B16,
2622   B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16,
2623   S22,
2624   S24, S26, S34, S36, S44, S46, S56, S66, T11p, T12, T14, T15,
2625   T16,
2626   T17, T18, T19, Bx, By, Bz};
2627 params0 = AssociationThread[allVars, allVars];
2628 If[nf > 7,
2629   (
2630     numE = 14 - nf;
2631     params = HoleElectronConjugation[params0];
2632     If[OptionValue["Set t2Switch"], params[t2Switch] = 0];
2633   ),
2634   params = params0;
2635   If[OptionValue["Set t2Switch"], params[t2Switch] = 1];
2636 ];
2637 (* Load symbolic expressions for LS,J,J' energy sub-matrices.*)
2638 emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
2639 OptionValue["FilenameAppendix"]];
2640 JJBlockMatrixTable = ImportFun[emFname];
2641 (*Patch together the entire matrix representation using J,J'
2642 blocks.*)
2643 PrintTemporary["Patching JJ blocks ..."];
2644 Js = AllowedJ[numE];
2645 howManyJs = Length[Js];
2646 blockHam = ConstantArray[0, {howManyJs, howManyJs}];
2647 Do[
2648   blockHam[[jj, ii]] = JJBlockMatrixTable[{numE, Js[[ii]]}, Js[[
2649

```

```

        jj]]}}]; ,
2645      {ii, 1, howManyJs},
2646      {jj, 1, howManyJs}
2647    ];
2648    (* Once the block form is created flatten it *)
2649    If[Not[OptionValue["ReturnInBlocks"]],
2650      (blockHam = ArrayFlatten[blockHam];
2651       blockHam = ReplaceInSparseArray[blockHam, params];
2652     ),
2653      (blockHam = Map[ReplaceInSparseArray[#, params]&, blockHam
2654 ,{2}]);
2655    ];
2656
2657   If[OptionValue["IncludeZeeman"],
2658     (
2659       PrintTemporary["Including Zeeman terms ..."];
2660       {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
2661       blockHam += - teslaToKayser * (Bx * magx + By * magy + Bz *
2662       magz);
2663     );
2664   ];
2665 ];
2666
2667 SimplerSymbolicHamMatrix::usage = "SimplerSymbolicHamMatrix[numE,
2668   simplifier] is a simple addition to HamMatrixAssembly that applies
2669   a given simplification to the full Hamiltonian. simplifier is a
2670   list of replacement rules.
2671 If the option \"Export\" is set to True, then the function also
2672 exports the resulting sparse array to the ./hams/ folder.
2673 The option \"PrependToFilename\" can be used to prepend a string to
2674 the filename to which the function may export to.
2675 The option \"Return\" can be used to choose whether the function
2676 returns the matrix or not.
2677 The option \"Overwrite\" can be used to overwrite the file if it
2678 already exists, if this options is set to False then this function
2679 simply reloads a file that it assumed to be present already in
2680 the ./hams folder.
2681 The option \"IncludeZeeman\" can be used to toggle the inclusion of
2682 the Zeeman interaction with an external magnetic field.
2683 The option \"OperatorBasis\" can be used to choose the basis in
2684 which the operator is expressed. The default is the \"Legacy\" basis.
2685 Order alternatives being: \"MostlyOrthogonal\" and \"Orthogonal\".
2686 In the \"Legacy\" alternative the operators used are
2687 the same as in Carnall's work. In the \"MostlyOrthogonal\" all
2688 operators are orthogonal except those corresponding to the Mk and
2689 Pk parameters. In the \"Orthogonal\" basis all operators are
2690 orthogonal, with the operators corresponding to the Mk and Pk
2691 parameters replaced by zi operators and accompanying ai
2692 coefficients. The \"Orthogonal\" option has not been implemented
2693 yet.";
2694 Options[SimplerSymbolicHamMatrix] = {
2695   "Export" -> True,
2696   "PrependToFilename" -> "",
2697   "EorF" -> "F",
2698   "Overwrite" -> False,
2699   "Return" -> True,
2700   "Set t2Switch" -> False,
2701   "IncludeZeeman" -> False,
2702   "OperatorBasis" -> "Legacy"
2703 };
2704 SimplerSymbolicHamMatrix[numE_, simplifier_, OptionsPattern[]] :=
2705 Module[
2706   {thisHam, fname, fnamemx},
2707   (
2708     If[Not[ValueQ[ElectrostaticTable]],
2709       LoadElectrostatic[]
2710     ];
2711     If[Not[ValueQ[S0OandECSOTable]],
2712       LoadS0OandECSO[]
2713     ];
2714     If[Not[ValueQ[SpinOrbitTable]],
2715       LoadSpinOrbit[]
2716     ];
2717   ];

```

```

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

```

```

2762         )
2763     ];
2764
2765     thisHam = HamMatrixAssembly[numE, "Set t2Switch" -> OptionValue
2766 ["Set t2Switch"], "IncludeZeeman" -> OptionValue["IncludeZeeman"], 
2767 "OperatorBasis" -> opBasis];
2768     If[Length[simplifier] > 0,
2769      thisHam = ReplaceInSparseArray[thisHam, simplifier];
2770    ];
2771    (* This removes zero entries from being included in the sparse
2772   array *)
2773    thisHam = SparseArray[thisHam];
2774    If[OptionValue["Export"],
2775    (
2776      If[Not@FileExistsQ[fname],
2777      (
2778        Print["Exporting to file ", fname];
2779        Export[fname, thisHam];
2780      )
2781      ];
2782      If[Not@FileExistsQ[fnamemx],
2783      (
2784        Print["Exporting to file ", fnamemx];
2785        Export[fnamemx, thisHam];
2786      )
2787    ];
2788    If[OptionValue["Return"],
2789      Return[thisHam],
2790      Return[Null]
2791    ];
2792  ];
2793
2794 ScalarLSJMFromLS::usage = "ScalarLSJMFromLS[numE,
2795 LSReducedMatrixElements]. Given the LS-reduced matrix elements
2796 LSReducedMatrixElements of a scalar operator, this function
2797 returns the corresponding LSJM representation. This is returned as
2798 a SparseArray.";
2799 ScalarLSJMFromLS[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[{i, 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 = NKSLJM[[1]];
2828   MJ = NKSLJM[[3]];
2829   MJp = NKSLJM[[3]];
2830   subKron = (KroneckerDelta[MJ, MJp]);
2831   matValue = If[subKron == 0,
2832   0,
2833   (
2834

```

```

2829      Which[MemberQ[Keys[LSReducedMatrixElements], {numE,
2830      SLterm, SpLpterm}], ,
2831          LSReducedMatrixElements[{numE, SLterm, SpLpterm}],
2832          MemberQ[Keys[LSReducedMatrixElements], {numE, SpLpterm,
2833          SLterm}], ,
2834              LSReducedMatrixElements[{numE, SpLpterm, SLterm}],
2835              True,
2836              0
2837          ]
2838      ];
2839      matValue,
2840      {NKSLJMp, NKSLJMs},
2841      {NKSLJM, NKSLJMs}
2842      ];
2843      jjBlockTable[{numE, J, Jp}] = SparseArray[eMatrix],
2844      {J, AllowedJ[numE]},
2845      {Jp, AllowedJ[numE]}
2846      ];
2847
2848      Js = AllowedJ[numE];
2849      howManyJs = Length[Js];
2850      blockHam = ConstantArray[0, {howManyJs, howManyJs}];
2851      Do[blockHam[[jj, ii]] = jjBlockTable[{numE, Js[[ii]], Js[[jj
2852      ]]}], ,
2853      {ii, 1, howManyJs},
2854      {jj, 1, howManyJs}];
2855      blockHam = ArrayFlatten[blockHam];
2856      blockHam = SparseArray[blockHam];
2857      Return[blockHam];
2858  ]
2859 (* ##### Block assembly ##### *)
2860 (* ##### ##### ##### ##### *)
2861 (* ##### ##### ##### ##### *)
2862 (* ##### ##### ##### ##### *)
2863 (* ##### ##### ##### ##### Level Description ##### *)
2864
2865 FreeHam::usage = "FreeHam[JBlocks, numE] given the JJ blocks of
2866   the Hamiltonian for f^n, this function returns a list with all the
2867   scalar-simplified versions of the blocks.";
2868 FreeHam[JBlocks_List, numE_Integer] := Module[
2869   {Js, basisJ, pivot, freeHam, idx, J,
2870   thisJbasis, shrunkBasisPositions, theBlock},
2871   (
2872     Js = AllowedJ[numE];
2873     basisJ = BasisLSJMJ[numE, "AsAssociation" -> True];
2874     pivot = If[OddQ[numE], 1/2, 0];
2875     freeHam = Table[(  

2876       J = Js[[idx]];
2877       theBlock = JBlocks[[idx]];
2878       thisJbasis = basisJ[J];
2879       (* find the basis vectors that end with pivot *)
2880       shrunkBasisPositions = Flatten[Position[thisJbasis, {_ ...,  

2881       pivot}]];
2882       (* take only those rows and columns *)
2883       theBlock[[shrunkBasisPositions, shrunkBasisPositions]]
2884     ),  

2885     {idx, 1, Length[Js]}
2886   ];
2887   Return[freeHam];
2888 )
2889 ];
2890
2891 ListRepeater::usage = "ListRepeater[list, reps] repeats each
2892   element of list reps times.";
2893 ListRepeater[list_List, repeats_Integer] := (
2894   Flatten[ConstantArray[#, repeats] & /@ list]
2895 );
2896
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 (

```

```

2897 uppityVecs = Table[(
2898   uppityVec = PadRight[{#}, multiplicity] & /@ vec;
2899   uppityVec = Permutations /@ uppityVec;
2900   uppityVec = Transpose[uppityVec];
2901   uppityVec = Flatten /@ uppityVec
2902   ),
2903 {vec, vecs}
2904 ];
2905 Return[Flatten[uppityVecs, 1]];
2906 )
2907 ];
2908
2909 EigenLever::usage = "EigenLever[eigenSys, multiplicity] takes a
2910   list eigenSys of the form {eigenvalues, eigenvectors} and returns
2911   the eigenvalues repeated multiplicity times and the eigenvectors
2912   interleaved and shifted accordingly.";
2913 EigenLever[eigenSys_, multiplicity_] := Module[
2914   {eigenVals, eigenVecs,
2915   leveledEigenVecs, leveledEigenVals},
2916   (
2917     {eigenVals, eigenVecs} = eigenSys;
2918     leveledEigenVals      = ListRepeater[eigenVals, multiplicity];
2919     leveledEigenVecs      = ListLever[eigenVecs, multiplicity];
2920     Return[{Flatten[leveledEigenVals], leveledEigenVecs}]
2921   )
2922 ];
2923
2924
2925 LevelSimplerSymbolicHamMatrix::usage =
2926   LevelSimplerSymbolicHamMatrix[numE] is a variation of
2927   HamMatrixAssembly that returns the diagonal JJ Hamiltonian blocks
2928   applying a simplifier and with simplifications adequate for the
2929   level description. The keys of the given association correspond to
2930   the different values of J that are possible for f^numE, the
2931   values are sparse array that are meant to be interpreted in the
2932   basis provided by BasisLSJ.
2933 The option \\"Simplifier\\" is a list of symbols that are set to zero
2934 . At a minimum this has to include the crystal field parameters.
2935 By default this includes everything except the Slater parameters
2936 Fk and the spin orbit coupling  $\zeta$ .
2937 The option \\"Export\\" controls whether the resulting association is
2938 saved to disk, the default is True and the resulting file is
2939 saved to the ./hams/ folder. A hash is appended to the filename
2940 that corresponds to the simplifier used in the resulting
2941 expression. If the option \\"Overwrite\\" is set to False then these
2942 files may be used to quickly retrieve a previously computed case.
2943 The file is saved both in .m and .mx format.
2944 The option \\"PrependToFilename\\" can be used to append a string to
2945 the filename to which the function may export to.
2946 The option \\"Return\\" can be used to choose whether the function
2947 returns the matrix or not.
2948 The option \\"Overwrite\\" can be used to overwrite the file if it
2949 already exists.";
2950 Options[LevelSimplerSymbolicHamMatrix] = {
2951   "Export" -> True,
2952   "PrependToFilename" -> "",
2953   "Overwrite" -> False,
2954   "Return" -> True,
2955   "Simplifier" -> Join[
2956     {FO, \[\[Sigma\]]SS},
2957     cfSymbols,
2958     TSymbols,
2959     casimirSymbols,
2960     pseudoMagneticSymbols,
2961     marvinSymbols,
2962     DeleteCases[magneticSymbols, \zeta]
2963   ]
2964 };
2965 LevelSimplerSymbolicHamMatrix[numE_Integer, OptionsPattern[]] :=
2966 Module[
2967   {thisHamAssoc, Js, fname,
2968   fnamemx, hash, simplifier},
2969   (
2970     simplifier = (#->0)&/@Sort[OptionValue["Simplifier"]];
2971     hash       = Hash[simplifier];
2972     If[Not[ValueQ[ElectrostaticTable]], LoadElectrostatic[]];
2973   );
2974 ];

```

```

2950 If [Not [ValueQ[S00andECSOTable]], LoadS00andECSO[]];
2951 If [Not [ValueQ[SpinOrbitTable]], LoadSpinOrbit[]];
2952 If [Not [ValueQ[SpinSpinTable]], LoadSpinSpin[]];
2953 If [Not [ValueQ[ThreeBodyTable]], LoadThreeBody[]];
2954 fname = FileNameJoin[{moduleDir, "hams", OptionValue["PrependToFilename"]}<>"Level-SymbolicMatrix-f"<>ToString[numE]<>"-"
2955 <>ToString[hash]<>".m"];
2956 fnamemx = FileNameJoin[{moduleDir, "hams", OptionValue["PrependToFilename"]}<>"Level-SymbolicMatrix-f"<>ToString[numE]<>"-"
2957 <>ToString[hash]<>".mx"};
2958 If [Or[FileExistsQ[fname], FileExistsQ[fnamemx]]&&Not[OptionValue["Overwrite"]],
2959 (
2960   If[OptionValue["Return"],
2961   (
2962     Which[FileExistsQ[fnamemx],
2963     (
2964       Print["File ", fnamemx, " already exists, and option \"Overwrite\" is set to False, loading file ..."];
2965       thisHamAssoc=Import[fnamemx];
2966       Return[thisHamAssoc];
2967     ),
2968     FileExistsQ[fname],
2969     (
2970       Print["File ", fname, " already exists, and option \"Overwrite\" is set to False, loading file ..."];
2971       thisHamAssoc=Import[fname];
2972       Print["Exporting to file ", fnamemx, " for quicker loading."];
2973       Export[fnamemx,thisHamAssoc];
2974       Return[thisHamAssoc];
2975     )
2976   ],
2977   (
2978     Print["File ", fname, " already exists, skipping ..."];
2979     Return[Null];
2980   ]
2981 );
2982 ];
2983 Js = AllowedJ[numE];
2984 thisHamAssoc = HamMatrixAssembly[numE,
2985 "Set t2Switch"->True,
2986 "IncludeZeeman"->False,
2987 "ReturnInBlocks"->True
2988 ];
2989 thisHamAssoc = Diagonal[thisHamAssoc];
2990 thisHamAssoc = Map[SparseArray[ReplaceInSparseArray[#, simplifier]]&,thisHamAssoc,{1}];
2991 thisHamAssoc = FreeHam[thisHamAssoc,numE];
2992 thisHamAssoc = AssociationThread[Js->thisHamAssoc];
2993 If[OptionValue["Export"],
2994 (
2995   Print["Exporting to file ", fname, " and to ", fnamemx];
2996   Export[fname,thisHamAssoc];
2997   Export[fnamemx,thisHamAssoc];
2998   )
2999 ];
3000 If[OptionValue["Return"],
3001   Return[thisHamAssoc],
3002   Return[Null]
3003 ];
3004 ]
3005 ];
3006
3007 LevelSolver::usage = "LevelSolver[numE, params] puts together (or
3008 retrieves from disk) the symbolic level Hamiltonian for the f^numE
3009 configuration and solves it for the given params returning the
3010 resultant energies and eigenstates.
3011 If the option \"Return as states\" is set to False, then the
3012 function returns an association whose keys are values for J in f^
3013 numE, and whose values are lists with two elements. The first
3014 element being equal to the ordered basis for the corresponding
3015 subspace, given as a list of lists of the form {LS string, J}. The
3016 second element being another list of two elements, the first

```

element being equal to the energies and the second being equal to the corresponding normalized eigenvectors. The energies given have been subtracted the energy of the ground state.
 3009 **If** the option `\"Return as states\"` is set to `True`, then the
 function returns a list with three elements. The first element is the global level basis for the `f^numE` configuration, given as a list of lists of the form `{LS string, J}`. The second element are the mayor LSJ components in the returned eigenstates. The third element is a list of lists with three elements, in each list the first element being equal to the energy, the second being equal to the value of `J`, and the third being equal to the corresponding normalized eigenvector (given as a row). The energies given have been subtracted the energy of the ground state, and the states have been sorted in order of increasing energy.
 3010 The following options are admitted:
 3011 - `\"Overwrite Hamiltonian\"`, if set to `True` the function will
 overwrite the symbolic Hamiltonian. `Default` is `False`.
 3012 - `\"Return as states\"`, see description above. `Default` is `True`.
 3013 - `\"Simplifier\"`, this is a list with symbols that are set to zero for defining the parameters kept in the level description.
 3014 ";
 3015 Options[LevelSolver] = {
 3016 "Overwrite Hamiltonian" -> False,
 3017 "Return as states" -> True,
 3018 "Simplifier" -> Join[
 3019 cfSymbols,
 3020 TSymbols,
 3021 casimirSymbols,
 3022 pseudoMagneticSymbols,
 3023 marvinSymbols,
 3024 DeleteCases[magneticSymbols, ζ]
 3025],
 3026 "PrintFun" -> PrintTemporary
 3027 };
 3028 LevelSolver[numE_Integer, params0_Association, OptionsPattern[]] :=
 3029 Module[
 3030 {ln, simplifier, simpleHam, basis,
 3031 numHam, eigensys, startTime, endTime,
 3032 diagonalTime, params=params0, globalBasis,
 3033 eigenVectors, eigenEnergies, eigenJs,
 3034 states, groundEnergy, allEnergies, PrintFun},
 3035 (
 3036 ln = theLanthanides[[numE]];
 3037 basis = BasisLSJ[numE, "AsAssociation" -> True];
 3038 simplifier = OptionValue["Simplifier"];
 3039 PrintFun = OptionValue["PrintFun"];
 3040 PrintFun["> LevelSolver for ", ln, " with ", numE, " f-electrons."]
 3041];
 3042 PrintFun["> Loading the symbolic level Hamiltonian ..."];
 3043 simpleHam = LevelSimplerSymbolicHamMatrix[numE,
 3044 "Simplifier" -> simplifier,
 3045 "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
 3046];
 3047 (* Everything that is not given is set to zero *)
 3048 PrintFun["> Setting to zero every parameter not given ..."];
 3049 params = ParamPad[params, "PrintFun" -> PrintFun];
 3050 PrintFun[params];
 3051 (* Create the numeric hamiltonian *)
 3052 PrintFun["> Replacing parameters in the J-blocks of the
 3053 Hamiltonian to produce numeric arrays ..."];
 3054 numHam = N /@ Map[ReplaceInSparseArray[#, params] &,
 3055 simpleHam];
 3056 Clear[simpleHam];
 3057 (* Eigensolver *)
 3058 PrintFun["> Diagonalizing the numerical Hamiltonian within each
 3059 separate J-subspace ..."];
 3060 startTime = Now;
 3061 eigensys = Eigensystem /@ numHam;
 3062 endTime = Now;
 3063 diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
 3064 allEnergies = Flatten[First /@ Values[eigensys]];
 3065 groundEnergy = Min[allEnergies];
 3066 eigensys = Map[Chop[{#[[1]] - groundEnergy, #[[2]]}] &, eigensys]
 3067];
 3068 eigensys = Association @ KeyValueMap[#1 -> {basis[#1], #2} &,
 3069 eigensys];

```

3063 PrintFun[">> Diagonalization took ",diagonalTime," seconds."];
3064 If[OptionValue["Return as states"],
3065 (
3066     PrintFun["> Padding the eigenvectors to correspond to the
3067 level basis ..."];
3068     eigenVectors = SparseArray @ BlockDiagonalMatrix[Values
3069     #[[2, 2]] & /@ eigensys];
3070     globalBasis = Flatten[Values[basis], 1];
3071     eigenEnergies = Flatten[Values[#[[2, 1]] & /@ eigensys]];
3072     eigenJs = Flatten[KeyValueMap[ConstantArray[#1,
3073 Length[#[[2, 2]]]] &, eigensys]];
3074     states = Transpose[{eigenEnergies, eigenJs,
3075 eigenVectors}];
3076     states = SortBy[states, First];
3077     eigenVectors = Last /@ states;
3078     LSJmultiplets = (RemoveTrailingDigits[#[[1]]] <> ToString[
3079 InputForm[#[[2]]]]) & /@ globalBasis;
3080     majorComponentIndices = Ordering[Abs[#[[-1]]] & /@
3081 eigenVectors;
3082     levelLabels = LSJmultiplets[[majorComponentIndices]];
3083     Return[{globalBasis, levelLabels, states}];
3084   ),
3085   Return[{basis, eigensys}]
3086 ];
3087 )
3088 ];
3089
3090 (* ##### Level Description ##### *)
3091 (* ##### Optical Operators ##### *)
3092
3093 magOp = <||>;
3094
3095 JJBlockMagDip::usage = "JJBlockMagDip[numE, J, Jp] returns an array
3096 for the LSJM matrix elements of the magnetic dipole operator
3097 between states with given J and Jp. The option \"Sparse\" can be
3098 used to return a sparse matrix. The default is to return a sparse
3099 matrix.
3100 See eqn 15.7 in TASS.
3101 Here it is provided in atomic units in which the Bohr magneton is
3102 1/2.
3103 \[Mu] = -(1/2) (L + gs S)
3104 We are using the Racah convention for the reduced matrix elements
3105 in the Wigner-Eckart theorem. See TASS eqn 11.15.
3106 ";
3107 Options[JJBlockMagDip]={ "Sparse" -> True};
3108 JJBlockMagDip[numE_, braJ_, ketJ_, OptionsPattern[]]:=Module[
3109 {braSLJs, ketSLJs,
3110 braSLJ, ketSLJ,
3111 braSL, ketSL,
3112 braS, braL,
3113 ketS, ketL,
3114 braMJ, ketMJ,
3115 matValue, magMatrix,
3116 summand1, summand2,
3117 threejays},
3118 (
3119   braSLJs = AllowedNKSLJMforJTerms[numE, braJ];
3120   ketSLJs = AllowedNKSLJMforJTerms[numE, ketJ];
3121   magMatrix = Table[
3122     braSL = braSLJ[[1]];
3123     ketSL = ketSLJ[[1]];
3124     {braS, braL} = FindSL[braSL];
3125     {ketS, ketL} = FindSL[ketSL];
3126     braMJ = braSLJ[[3]];
3127     ketMJ = ketSLJ[[3]];
3128     summand1 = If[Or[braJ != ketJ,
3129                   braSL != ketSL],
3130                 0,
3131                 Sqrt[braJ*(braJ+1)*TPO[braJ]]
3132               ];
3133   (* looking at the string includes checking L=L', S=S', and \

```

```

alpha=\alpha'*)
summand2 = If[braSL!= ketSL,
0,
(gs-1) *
Phaser[braS+braL+ketJ+1] *
Sqrt[TPO[braJ]*TPO[ketJ]] *
SixJay[{braJ,1,ketJ},{braS,braL,braS}] *
Sqrt[braS(braS+1)TPO[braS]]
];
matValue = summand1 + summand2;
(* We are using the Racah convention for red matrix elements
in Wigner-Eckart *)
threejays = (ThreeJay[{braJ, -braMJ}, {1, #}, {ketJ, ketMJ}]
&) /@ {-1,0,1};
threejays *= Phaser[braJ-braMJ];
matValue = - 1/2 * threejays * matValue;
matValue,
{braSLJ, braSLJs},
{ketSLJ, ketSLJs}
];
If[OptionValue["Sparse"],
magMatrix = SparseArray[magMatrix]
];
Return[magMatrix];
)
];
Options[TabulateJJBlockMagDipTable]= {"Sparse" -> True};
TabulateJJBlockMagDipTable[numE_, OptionsPattern[]] := (
JJBlockMagDipTable=<||>;
Js=AllowedJ[numE];
Do[
(
JJBlockMagDipTable[{numE, braJ, ketJ}] =
JJBlockMagDip[numE, braJ, ketJ, "Sparse" -> OptionValue["Sparse"]]
),
{braJ, Js},
{ketJ, Js}
];
Return[JJBlockMagDipTable];
);
TabulateManyJJBlockMagDipTables::usage =
TabulateManyJJBlockMagDipTables[{n1, n2, ...}] calculates the
tables of matrix elements for the requested f^n_i configurations.
The function does not return the matrices themselves. It instead
returns an association whose keys are numE and whose values are
the filenames where the output of TabulateManyJJBlockMagDipTables
was saved to. The output consists of an association whose keys are
of the form {n, J, Jp} and whose values are rectangular arrays
given the values of <|LSJMJa|H_dip|L'S'J'MJ'a'|>.";
Options[TabulateManyJJBlockMagDipTables]= {"FilenameAppendix" -> "", "Overwrite" -> False, "Compressed" -> True};
TabulateManyJJBlockMagDipTables[ns_, OptionsPattern[]] := (
fnames=<||>;
Do[
(
ExportFun=If[OptionValue["Compressed"], ExportMZip, Export];
PrintTemporary["----- numE = ", numE, " -----#"];
appendTo = (OptionValue["FilenameAppendix"] <> "-magDip");
exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix" -> appendTo];
fnames[numE] = exportFname;
If[FileExistsQ[exportFname] && Not[OptionValue["Overwrite"]],
Continue[]
];
JJBlockMatrixTable = TabulateJJBlockMagDipTable[numE];
If[FileExistsQ[exportFname] && OptionValue["Overwrite"],
DeleteFile[exportFname]
];
ExportFun[exportFname, JJBlockMatrixTable];
),
{numE, ns}
];
Return[fnames];
)

```

```

3188 );
3189
3190 MagDipoleMatrixAssembly::usage = "MagDipoleMatrixAssembly[numE]
3191   returns the matrix representation of the operator - 1/2 (L + gs S)
3192   in the f^numE configuration. The function returns a list with
3193   three elements corresponding to the x,y,z components of this
3194   operator. The option \"FilenameAppendix\" can be used to append a
3195   string to the filename from which the function imports from in
3196   order to patch together the array. For numE beyond 7 the function
3197   returns the same as for the complementary configuration. The
3198   option \"ReturnInBlocks\" can be used to return the matrices in
3199   blocks. The default is to return the matrices in flattened form
3200   and as sparse array.";
3201 Options[MagDipoleMatrixAssembly]={
3202   "FilenameAppendix" -> "", 
3203   "ReturnInBlocks" -> False};
3204 MagDipoleMatrixAssembly[nf_Integer, OptionsPattern[]] := Module[
3205   {ImportFun, numE, appendTo,
3206   emFname, JJBlockMagDipTable,
3207   Js, howManyJs, blockOp,
3208   rowIdx, colIdx},
3209   (
3210     ImportFun = ImportMZip;
3211     numE = nf;
3212     numH = 14 - numE;
3213     numE = Min[numE, numH];
3214
3215     appendTo = (OptionValue["FilenameAppendix"] <> "-magDip");
3216     emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
3217     appendTo];
3218     JJBlockMagDipTable = ImportFun[emFname];
3219
3220     Js = AllowedJ[numE];
3221     howManyJs = Length[Js];
3222     blockOp = ConstantArray[0, {howManyJs, howManyJs}];
3223     Do[
3224       blockOp[[rowIdx, colIdx]] = JJBlockMagDipTable[{numE, Js[[rowIdx]], Js[[colIdx]]}],
3225       {rowIdx, 1, howManyJs},
3226       {colIdx, 1, howManyJs}
3227     ];
3228     If[OptionValue["ReturnInBlocks"],
3229       (
3230         opMinus = Map[#[[1]] &, blockOp, {4}];
3231         opZero = Map[#[[2]] &, blockOp, {4}];
3232         opPlus = Map[#[[3]] &, blockOp, {4}];
3233         opX = (opMinus - opPlus)/Sqrt[2];
3234         opY = I (opPlus + opMinus)/Sqrt[2];
3235         opZ = opZero;
3236       ),
3237         blockOp = ArrayFlatten[blockOp];
3238         opMinus = blockOp[[;, , 1]];
3239         opZero = blockOp[[;, , 2]];
3240         opPlus = blockOp[[;, , 3]];
3241         opX = (opMinus - opPlus)/Sqrt[2];
3242         opY = I (opPlus + opMinus)/Sqrt[2];
3243         opZ = opZero;
3244       ];
3245       Return[{opX, opY, opZ}];
3246     )
3247   ];
3248
3249 MagDipLineStrength::usage = "MagDipLineStrength[theEigensys, numE]
3250   takes the eigensystem of an ion and the number numE of f-electrons
3251   that correspond to it and calculates the line strength array Stot
3252 .
3253 The option \"Units\" can be set to either \"SI\" (so that the units
3254   of the returned array are (A m^2)^2) or to \"Hartree\".
3255 The option \"States\" can be used to limit the states for which the
3256   line strength is calculated. The default, All, calculates the
3257   line strength for all states. A second option for this is to
3258   provide an index labelling a specific state, in which case only
3259   the line strengths between that state and all the others are
3260   computed.
3261 The returned array should be interpreted in the eigenbasis of the
3262   Hamiltonian. As such the element Stot[[i,i]] corresponds to the

```

```

    line strength states between states  $|i\rangle$  and  $|j\rangle.$ .";

3242 Options[MagDipLineStrength] = {"Reload MagOp" -> False, "Units" -> "SI"
3243   , "States" -> All};

3243 MagDipLineStrength[theEigensys_List, numE0_Integer, OptionsPattern[]] := Module[
3244   {numE, allEigenvecs, Sx, Sy, Sz, Stot, factor},
3245   (
3246     numE = Min[14 - numE0, numE0];
3247     (*If not loaded then load it, *)
3248     If[Or[
3249       Not[MemberQ[Keys[magOp], numE]],
3250       OptionValue["Reload MagOp"]],
3251       (
3252         magOp[numE] = ReplaceInSparseArray[#, {gs -> 2}] & /@
3253         MagDipoleMatrixAssembly[numE];
3254       )
3255     ];
3255     allEigenvecs = Transpose[Last /@ theEigensys];
3256     Which[OptionValue["States"] === All,
3257       (
3258         {Sx, Sy, Sz} = (ConjugateTranspose[allEigenvecs].#.
3259         allEigenvecs) & /@ magOp[numE];
3260         Stot = Abs[Sx]^2 + Abs[Sy]^2 + Abs[Sz]^2;
3261       ),
3261       IntegerQ[OptionValue["States"]],
3262       (
3263         singleState = theEigensys[[OptionValue["States"], 2]];
3264         {Sx, Sy, Sz} = (ConjugateTranspose[allEigenvecs].#.
3265         singleState) & /@ magOp[numE];
3266         Stot = Abs[Sx]^2 + Abs[Sy]^2 + Abs[Sz]^2;
3267       )
3267     ];
3268     Which[
3269       OptionValue["Units"] == "SI",
3270         Return[4 \[Mu]B^2 * Stot],
3271       OptionValue["Units"] == "Hartree",
3272         Return[Stot],
3273       True,
3274       (
3275         Print["Invalid option for \"Units\". Options are \"SI\" and
3276           \"Hartree\"."];
3276         Abort[];
3277       )
3278     ];
3279   );
3280 ];

3281 MagDipoleRates::usage = "MagDipoleRates[eigenSys, numE] calculates
3282   the magnetic dipole transition rate array for the provided
3283   eigensystem. The option \"Units\" can be set to \"SI\" or to \"
3284   Hartree\". If the option \"Natural Radiative Lifetimes\" is set to
3285   true then the reciprocal of the rate is returned instead.
3286   eigenSys is a list of lists with two elements, in each list the
3287   first element is the energy and the second one the corresponding
3288   eigenvector.

3288 Based on table 7.3 of Thorne 1999, using g2=1.
3289 The energy unit assumed in eigenSys is kayser.
3290 The returned array should be interpreted in the eigenbasis of the
3291   Hamiltonian. As such the element AMD[[i,i]] corresponds to the
3292   transition rate (or the radiative lifetime, depending on options)
3293   between eigenstates  $|i\rangle$  and  $|j\rangle$ .
3294 By default this assumes that the refractive index is unity, this
3295   may be changed by setting the option \"RefractiveIndex\" to the
3296   desired value.
3297 The option \"Lifetime\" can be used to return the reciprocal of the
3298   transition rates. The default is to return the transition rates."
3298 ;
3299 Options[MagDipoleRates] = {"Units" -> "SI", "Lifetime" -> False, "
3300   RefractiveIndex" -> 1};
3300 MagDipoleRates[eigenSys_List, numE0_Integer, OptionsPattern[]] :=
3301   Module[
3302   {AMD, Stot, eigenEnergies,
3303   transitionWaveLengthsInMeters, nRefractive},
3304   (
3305     nRefractive = OptionValue["RefractiveIndex"];
3306     numE = Min[14 - numE0, numE0];
3307   ];
3308   AMD = Table[0, {numE}, {numE}];
3309   For[i = 1, i <= numE, i++,
3310     For[j = 1, j <= numE, j++,
3311       If[i != j,
3312         Stot = 0;
3313         For[k = 1, k <= Length[eigenEnergies], k++,
3314           Stot += eigenEnergies[[k, i]] eigenEnergies[[k, j]]*
3315             transitionWaveLengthsInMeters[[k]]/nRefractive];
3316         AMD[[i, j]] = Stot];
3317       ];
3318     ];
3319   ];
3320   AMD;
3321 ];
3322 
```

```

3295     Stot          = MagDipLineStrength[eigenSys, numE, "Units" ->
3296 OptionValue["Units"]];
3297     eigenEnergies = Chop[First/@eigenSys];
3298     energyDiffs  = Outer[Subtract, eigenEnergies, eigenEnergies];
3299     energyDiffs  = ReplaceDiagonal[energyDiffs, Indeterminate];
3300     (* Energies assumed in kayser.*)
3301     transitionWaveLengthsInMeters = 0.01/energyDiffs;
3302
3303     unitFactor    = Which[
3304 OptionValue["Units"]== "Hartree",
3305 (
3306     (* The bohrRadius factor in SI needed to convert the
3307 wavelengths which are assumed in m*)
3308     16 \[Pi]^3 (\[Mu]0Hartree /(3 hPlanckHartree)) * bohrRadius^3
3309 ),
3310 OptionValue["Units"]== "SI",
3311 (
3312     16 \[Pi]^3 \[Mu]0/(3 hPlanck)
3313 ),
3314 True,
3315 (
3316     Print["Invalid option for \"Units\". Options are \"SI\" and \
3317 \"Hartree\"."];
3318     Abort[];
3319 )
3320 ];
3321 ];
3322 AMD = unitFactor / transitionWaveLengthsInMeters^3 * Stot *
nRefractive^3;
3323 Which[OptionValue["Lifetime"],
3324 Return[1/AMD],
3325 True,
3326 Return[AMD]
3327 ]
3328 ]
3329 ];
3330
3331 GroundMagDipoleOscillatorStrength::usage =
3332 "GroundMagDipoleOscillatorStrength[eigenSys, numE] calculates the
3333 magnetic dipole oscillator strengths between the ground state and
3334 the excited states as given by eigenSys.
3335 Based on equation 8 of Carnall 1965, removing the 2J+1 factor since
3336 this degeneracy has been removed by the crystal field.
3337 eigenSys is a list of lists with two elements, in each list the
3338 first element is the energy and the second one the corresponding
3339 eigenvector.
3340 The energy unit assumed in eigenSys is Kayser.
3341 The oscillator strengths are dimensionless.
3342 The returned array should be interpreted in the eigenbasis of the
3343 Hamiltonian. As such the element fMDGS[[i]] corresponds to the
3344 oscillator strength between ground state and eigenstate |i>.
3345 By default this assumes that the refractive index is unity, this
3346 may be changed by setting the option \"RefractiveIndex\" to the
3347 desired value.";
3348 Options[GroundMagDipoleOscillatorStrength]= {"RefractiveIndex" -> 1};
3349 GroundMagDipoleOscillatorStrength[eigenSys_List, numE_Integer,
3350 OptionsPattern[]] := Module[
3351 {eigenEnergies, SMDGS, GSEnergy, energyDiffs,
3352 transitionWaveLengthsInMeters, unitFactor, nRefractive},
3353 (
3354     eigenEnergies = First/@eigenSys;
3355     nRefractive = OptionValue["RefractiveIndex"];
3356     SMDGS = MagDipLineStrength[eigenSys, numE, "Units" ->
3357 SI", "States" -> 1];
3358     GSEnergy = eigenSys[[1,1]];
3359     energyDiffs = eigenEnergies - GSEnergy;
3360     energyDiffs[[1]] = Indeterminate;
3361     transitionWaveLengthsInMeters = 0.01/energyDiffs;
3362     unitFactor = (8\[Pi]^2 me)/(3 hPlanck eCharge^2 cLight);
3363     fMDGS = unitFactor / transitionWaveLengthsInMeters *
3364 SMDGS * nRefractive;
3365     Return[fMDGS];
3366 )
3367 ];
3368 ];
3369
3370 (* ##### Optical Operators ##### *)
3371 (* ##### *)

```

```

3354
3355 (* ##### Printers and Labels #### *)
3356
3357
3358 PrintL::usage = "PrintL[L] give the string representation of a
3359   given angular momentum.";
3360 PrintL[L_] := If[StringQ[L], L, StringTake[specAlphabet, {L + 1}]]
3361
3362 FindSL::usage = "FindSL[LS] gives the spin and orbital angular
3363   momentum that corresponds to the provided string LS.";
3364 FindSL[SL_] := (
3365   FindSL[SL] =
3366   If[StringQ[SL],
3367     {
3368       (ToExpression[StringTake[SL, 1]] - 1)/2,
3369       StringPosition[specAlphabet, StringTake[SL, {2}]][[1, 1]] - 1
3370     },
3371     SL
3372   ];
3373
3374 PrintSLJ::usage = "Given a list with three elements {S, L, J} this
3375   function returns a symbol where the spin multiplicity is presented
3376   as a superscript, the orbital angular momentum as its
3377   corresponding spectroscopic letter, and J as a subscript. Function
3378   does not check to see if the given J is compatible with the given
3379   S and L.";
3380 PrintSLJ[SLJ_] := (
3381   RowBox[{(
3382     SuperscriptBox[" ", 2 SLJ[[1]] + 1],
3383     SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]
3384   }] // DisplayForm
3385 );
3386
3387 PrintSLJM::usage = "Given a list with four elements {S, L, J, MJ}
3388   this function returns a symbol where the spin multiplicity is
3389   presented as a superscript, the orbital angular momentum as its
3390   corresponding spectroscopic letter, and {J, MJ} as a subscript. No
3391   attempt is made to guarantee that the given input is consistent."
3392 ;
3393 PrintSLJM[SLJM_] := (
3394   RowBox[{(
3395     SuperscriptBox[" ", 2 SLJM[[1]] + 1],
3396     SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]
3397   }] // DisplayForm
3398 );
3399
3400 (* ##### Printers and Labels #### *)
3401 (* ##### Term management #### *)
3402
3403
3404 AllowedSLTerms::usage = "AllowedSLTerms[numE] returns a list with
3405   the allowed terms in the f^numE configuration, the terms are given
3406   as lists in the format {S, L}. This list may have redundancies
3407   which are compatible with the degeneracies that might correspond
3408   to the given case.";
3409 AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE,
3410   14 - numE]]];
3411
3412 AllowedNKSLTerms::usage = "AllowedNKSLTerms[numE] returns a list
3413   with the allowed terms in the f^numE configuration, the terms are
3414   given as strings in spectroscopic notation. The integers in the
3415   last positions are used to distinguish cases with degeneracy.";
3416 AllowedNKSLTerms[numE_] := Map[First, CFPTerms[Min[numE, 14 - numE
3417   ]]];
3418 AllowedNKSLTerms[0] = {"1S"};
3419 AllowedNKSLTerms[14] = {"1S"};
3420
3421 MaxJ::usage = "MaxJ[numE] gives the maximum J = S+L that
3422   corresponds to the configuration f^numE.";
3423 MaxJ[numE_] := Max[Map[Total, AllowedSLTerms[Min[numE, 14 - numE]]]];
3424
3425
3426
3427

```

```

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

```

```

3471         termNDSL = allowedNDSL[[nn]];
3472         termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[
3473             termSL],
3474                 {{termNDSL, J}},
3475                 {}
3476             ];
3477             allowedSLJ = Join[allowedSLJ, termsSLJ];
3478             nn++;
3479         )
3480     ];
3481   );
3482 ];
3483
3484 AllowedSLJMTerms::usage = "AllowedSLJMTerms[numE] returns a list
3485   with all the states that correspond to the configuration f^n. A
3486   list is returned whose elements are lists of the form {S, L, J, MJ}
3487   .";
3488 AllowedSLJMTerms[numE_] := Module[
3489   {allowedSLJ, allowedSLJM,
3490   termSLJ, termsSLJM, nn},
3491   (
3492     allowedSLJ = AllowedSLJTerms[numE];
3493     allowedSLJM = {};
3494     For[
3495       nn = 1,
3496       nn <= Length[allowedSLJ],
3497       nn++,
3498       (
3499         termSLJ = allowedSLJ[[nn]];
3500         termsSLJM =
3501           Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
3502             {M, - termSLJ[[3]], termSLJ[[3]]}]
3503         ];
3504         allowedSLJM = Join[allowedSLJM, termsSLJM];
3505       )
3506     ];
3507   ];
3508   Return[SortBy[allowedSLJM, Last]];
3509 ]
3510
3511 AllowedNDSLJMforJMTerms::usage = "AllowedNDSLJMforJMTerms[numE, J,
3512   MJ] returns a list with all the terms that contain states of the f
3513   ^n configuration that have a total angular momentum J, and a
3514   projection along the z-axis MJ. The returned list has elements of
3515   the form {SL (string in spectroscopic notation), J, MJ}.";
3516 AllowedNDSLJMforJMTerms[numE_, J_, MJ_] := Module[
3517   {allowedSL, allowedNDSL,
3518   allowedSLJM, nn},
3519   (
3520     allowedNDSL = AllowedNDSLTerms[numE];
3521     allowedSL = AllowedSLTerms[numE];
3522     allowedSLJM = {};
3523     For[
3524       nn = 1,
3525       nn <= Length[allowedSL],
3526       termSL = allowedSL[[nn]];
3527       termNDSL = allowedNDSL[[nn]];
3528       termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
3529                     <= J
3530                     <= Total[termSL]
3531                     && (Abs[MJ] <= J)
3532                     ),
3533                     {{termNDSL, J, MJ}},
3534                     {}];
3535       allowedSLJM = Join[allowedSLJM, termsSLJ];
3536       nn++
3537     ];
3538   ];
3539   Return[allowedSLJM];
3540 ]
3541
3542 AllowedNDSLJMforJTerms::usage = "AllowedNDSLJMforJTerms[numE, J]
3543   returns a list with all the states that have a total angular
3544   momentum J. The returned list has elements of the form {SL (string
3545   in spectroscopic notation), J, MJ}.";

```

```

3536 AllowedNKSLJMforJTerms[numE_, J_] := Module[
3537   {MJs, labelsAndMomenta, termsWithJ},
3538   (
3539     MJs = AllowedMforJ[J];
3540     (* Pair LS labels and their {S,L} momenta *)
3541     labelsAndMomenta = ({#, FindSL[#]}) & /@ AllowedNKSLTerms[numE]
3542   ];
3543   (* A given term will contain J if |L-S|<=J<=L+S *)
3544   ContainsJ[{SL_String, {S_, L_}}] := (Abs[S - L] <= J <= (S + L)
3545 );
3546   (* Keep just the terms that satisfy this condition *)
3547   termsWithJ = Select[labelsAndMomenta, ContainsJ];
3548   (* We don't want to keep the {S,L} *)
3549   termsWithJ = {#[[1]], J} & /@ termsWithJ;
3550   (* This is just a quick way of including up all the MJ values
3551   *)
3552   Return[Flatten /@ Tuples[{termsWithJ, MJs}]]
3553 )
3554 ];
3555
3556 AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J
3557 , 1].";
3558 AllowedMforJ[J_] := Range[-J, J, 1];
3559
3560 AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta
3561 J that appear in the f^numE configuration.";
3562 AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}];
3563
3564 Seniority::usage = "Seniority[LS] returns the seniority of the
3565 given term.";
3566 Seniority[LS_] := FindNKLSTerm[LS][[1, 2]];
3567
3568 FindNKLSTerm::usage = "Given the string LS FindNKLSTerm[SL] returns
3569 all the terms that are compatible with it. This is only for f^n
3570 configurations. The provided terms might belong to more than one
3571 configuration. The function returns a list with elements of the
3572 form {LS, seniority, W, U}.";
3573 FindNKLSTerm[SL_] := Module[
3574   {NKterms, n},
3575   (
3576     n = 7;
3577     NKterms = {};
3578     Map[
3579       If[! StringFreeQ[First[#], SL],
3580         If[ToExpression[Part[#, 2]] <= n,
3581           NKterms = Join[NKterms, {#}, 1]
3582         ]
3583       ] &,
3584       fnTermLabels
3585     ];
3586     NKterms = DeleteCases[NKterms, {}];
3587     NKterms
3588   )
3589 ];
3590
3591 ParseTermLabels::usage = "ParseTermLabels[] parses the labels for
3592 the terms in the f^n configurations based on the labels for the f6
3593 and f7 configurations. The function returns a list whose elements
3594 are of the form {LS, seniority, W, U}.";
3595 Options[ParseTermLabels] = {"Export" -> True};
3596 ParseTermLabels[OptionsPattern[]] := Module[
3597   {labelsTextData, fNtextLabels, nielsonKosterLabels,
3598   seniorities, RacahW, RacahU},
3599   (
3600     labelsTextData = FileNameJoin[{moduleDir, "data", "
3601 NielsonKosterLabels_f6_f7.txt"}];
3602     fNtextLabels = Import[labelsTextData];
3603     nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
3604     termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
3605     seniorities = Map[ToExpression[Part[#, {2}]] &,
3606     nielsonKosterLabels];
3607     racahW =
3608     Map[
3609       StringTake[
3610         Flatten[StringCases[Part[#, {3}],
3611           "(" ~~ DigitCharacter ~~ DigitCharacter ~~
3612           DigitCharacter ~~ ")"
3613         ]
3614       ]
3615     ];
3616   ];

```

```

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

```

```

3662     flippedParams = Association[Join[nonflipped, flipped]];
3663     flippedParams = Select[flippedParams, FreeQ[#, Missing]&];
3664     Return[flippedParams];
3665   )
3666 ];
3667
3668 IonSolver::usage = "IonSolver[numE, params, host] puts together (or
3669 retrieves from disk) the symbolic Hamiltonian for the f^numE
3670 configuration and solves it for the given params.
3671 params is an Association with keys equal to parameter symbols and
3672 values their numerical values. The function will replace the
3673 symbols in the symbolic Hamiltonian with their numerical values
3674 and then diagonalize the resulting matrix. Any parameter that is
3675 not defined in the params Association is assumed to be zero.
3676 host is an optional string that may be used to prepend the filename
3677 of the symbolic Hamiltonian that is saved to disk. The default is
3678 \\"Ln\\".
3679 The function returns the eigensystem as a list of lists where in
3680 each list the first element is the energy and the second element
3681 the corresponding eigenvector.
3682 The ordered basis in which these eigenvectors are to be interpreted
3683 is the one corresponding to BasisLSJMJ[numE].
3684 The function admits the following options:
3685 \\"Include Spin-Spin\\" (bool) : If True then the spin-spin
3686 interaction is included as a contribution to the m_k operators.
3687 The default is True.
3688 \\"Overwrite Hamiltonian\\" (bool) : If True then the function will
3689 overwrite the symbolic Hamiltonian that is saved to disk to
3690 expedite calculations. The default is False. The symbolic
3691 Hamiltonian is saved to disk to the ./hams/ folder preceded by the
3692 string host.
3693 \\"Zeroes\\" (list) : A list with symbols assumed to be zero.
3694 ";
3695 Options[IonSolver] = {
3696   "Include Spin-Spin" -> True,
3697   "Overwrite Hamiltonian" -> False,
3698   "Zeroes" -> {}
3699 };
3700 IonSolver[numE_Integer, params0_Association, host_String:"Ln",
3701 OptionsPattern[]] := Module[
3702 {ln, simplifier, simpleHam, numHam, eigensys,
3703 startTime, endTime, diagonalTime,
3704 params=params0, zeroSymbols},
3705 (
3706   ln = theLanthanides[[numE]];
3707
3708   (* This could be done when replacing values, but this produces
3709 smaller saved arrays. *)
3710   simplifier = (#-> 0) & /@ OptionValue["Zeroes"];
3711   simpleHam = SimplerSymbolicHamMatrix[numE,
3712     simplifier,
3713     "PrependToFilename" -> host,
3714     "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
3715   ];
3716
3717   (* Note that we don't have to flip signs of parameters for fn
3718 beyond f7 since the matrix produced
3719 by SimplerSymbolicHamMatrix has already accounted for this. *)
3720
3721   (* Everything that is not given is set to zero *)
3722   params = ParamPad[params];
3723   PrintFun[params];
3724
3725   (* Enforce the override to the spin-spin contribution to the
3726 magnetic interactions *)
3727   params[\[\Sigma]SS] = If[OptionValue["Include Spin-Spin"], 1,
3728 0];
3729
3730   (* Create the numeric hamiltonian *)
3731   numHam = ReplaceInSparseArray[simpleHam, params];
3732   Clear[simpleHam];
3733
3734   (* Eigensolver *)
3735   PrintFun["> Diagonalizing the numerical Hamiltonian ..."];
3736   startTime = Now;
3737   eigensys = Eigensystem[numHam];
3738
3739

```

```

3716     endTime = Now;
3717     diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
3718 ];
3719 PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
3720 ];
3721 eigensys = Chop[eigensys];
3722 eigensys = Transpose[eigensys];
3723
3724 (* Shift the baseline energy *)
3725 eigensys = ShiftedLevels[eigensys];
3726 (* Sort according to energy *)
3727 eigensys = SortBy[eigensys, First];
3728 Return[eigensys];
3729 )
3730 ];
3731 ShiftedLevels::usage = "ShiftedLevels[eigenSys] takes a list of
3732 levels of the form
3733 {{energy_1, coeff_vector_1}, {energy_2, coeff_vector_2}, ...} and
3734 returns the same input except that now to every energy the minimum
3735 of all of them has been subtracted.";
3736 ShiftedLevels[originalLevels_] := Module[
3737 {groundEnergy, shifted},
3738 (
3739 groundEnergy = Sort[originalLevels][[1,1]];
3740 shifted = Map[{#[[1]] - groundEnergy, #[[2]]} &,
3741 originalLevels];
3742 Return[shifted];
3743 )
3744 ];
3745
3746 (* ##### Optical Transitions for Levels ##### *)
3747
3748 JuddOfeltUkSquared::usage = "JuddOfeltUkSquared[numE, params]
3749 calculates the matrix elements of the Uk operator in the level
3750 basis. These are calculated according to equation (7) in Carnall
3751 1965.
3752 The function returns a list with the following elements:
3753 - basis : A list with the allowed {SL, J} terms in the f^numE
3754 configuration. Equal to BasisLSJ[numE].
3755 - eigenSys : A list with the eigensystem of the Hamiltonian for
3756 the f^n configuration.
3757 - levelLabels : A list with the labels of the major components of
3758 the level eigenstates.
3759 - LevelUkSquared : An association with the squared matrix
3760 elements of the Uk operators in the level eigenbasis. The keys
3761 being {2, 4, 6} corresponding to the rank of the Uk operator. The
3762 basis in which the matrix elements are given is the one
3763 corresponding to the level eigenstates given in eigenSys and whose
3764 major SLJ components are given in levelLabels. The matrix is
3765 symmetric and given as a SymmetrizedArray.
3766 The function admits the following options:
3767 \\"PrintFun\\" : A function that will be used to print the progress
3768 of the calculations. The default is PrintTemporary.";
3769 Options[JuddOfeltUkSquared] = {"PrintFun" -> PrintTemporary};
3770 JuddOfeltUkSquared[numE_, params_, OptionsPattern[]] := Module[
3771 {eigenChanger, numEH, basis, eigenSys,
3772 Js, Ukm, LevelUkSquared, kRank,
3773 S, L, Sp, Lp, J, Jp, phase,
3774 braTerm, ketTerm, levelLabels,
3775 eigenVecs, majorComponentIndices},
3776 (
3777 If[Not[ValueQ[ReducedUkTable]],
3778 LoadUk[];
3779 ];
3780 numEH = Min[numE, 14-numE];
3781 PrintFun = OptionValue["PrintFun"];
3782 PrintFun["> Calculating the levels for the given parameters ..."];
3783 ];
3784 {basis, majorComponents, eigenSys} = LevelSolver[numE, params];
3785 (* The change of basis matrix to the eigenstate basis *)
3786 eigenChanger = Transpose[Last /@ eigenSys];
3787 PrintFun["Calculating the matrix elements of Uk in the physical
3788 coupling basis ..."];

```

```

3771 LevelUkSquared = <||>;
3772 Do[(
3773     Uksmat = Table[(  

3774         {S, L} = FindSL[braTerm[[1]]];  

3775         J = braTerm[[2]];  

3776         Jp = ketTerm[[2]];  

3777         {Sp, Lp} = FindSL[ketTerm[[1]]];  

3778         phase = Phaser[S + Lp + J + kRank];  

3779         Simplify @ (
3780             phase *
3781             Sqrt[TPO[J]*TPO[Jp]] *
3782             SixJay[{J, Jp, kRank}, {Lp, L, S}] *
3783             ReducedUkTable[{numEH, 3, braTerm[[1]], ketTerm[[1]],  

3784             kRank}]
3785             )
3786             ),
3787             {braTerm, basis},
3788             {ketTerm, basis}
3789             ];
3790             Uksmat = (Transpose[eigenChanger] . Uksmat . eigenChanger)^2;
3791             Uksmat = Chop@Uksmat;
3792             LevelUkSquared[kRank] = SymmetrizedArray[Uksmat, Dimensions[
3793                 eigenChanger], Symmetric[{1, 2}]];
3794             ),
3795             {kRank, {2, 4, 6}}
3796             ];
3797             LSJmultiplets = (RemoveTrailingDigits[#[[1]]] <> ToString[
3798                 InputForm[#[[2]]]]) & /@ basis;
3799                 eigenVecs = Last /@ eigenSys;
3800                 majorComponentIndices = Ordering[Abs[#][[-1]] & /@ eigenVecs;
3801                 levelLabels = LSJmultiplets[[majorComponentIndices]];
3802                 Return[{basis, eigenSys, levelLabels, LevelUkSquared}];
3803             )
3804             ];
3805             LevelElecDipoleOscillatorStrength::usage =
3806             LevelElecDipoleOscillatorStrength[numE, levelParams,
3807             juddOfeltParams] uses Judd-Ofelt theory to estimate the forced
3808             electric dipole oscillator strengths ions whose level description
3809             is determined by levelParams.
3810             The third parameter juddOfeltParams is an association with keys
3811             equal to the three Judd-Ofelt intensity parameters {\[CapitalOmega]2,
3812             \[CapitalOmega]4, \[CapitalOmega]6} and corresponding values
3813             in cm^2.
3814             The local field correction implemented here corresponds to the one
3815             given by the virtual cavity model of Lorentz.
3816             The function returns a list with the following elements:
3817             - basis : A list with the allowed {SL, J} terms in the f^numE
3818             configuration. Equal to BasisLSJ[numE].
3819             - eigenSys : A list with the eigensystem of the Hamiltonian for
3820             the f^n configuration in the level description.
3821             - levelLabels : A list with the labels of the major components of
3822             the calculated levels.
3823             - oStrengthArray : A square array whose elements represent the
3824             oscillator strengths between levels such that the element
3825             oStrengthArray[[i,j]] is the oscillator strength between the
3826             levels |Subscript[\[Psi], i]> and |Subscript[\[Psi], j]>. In this
3827             array, the elements below the diagonal represent emission
3828             oscillator strengths, and elements above the diagonal represent
3829             absorption oscillator strengths.
3830             The function admits the following three options:
3831             \"PrintFun\" : A function that will be used to print the progress
3832             of the calculations. The default is PrintTemporary.
3833             \"RefractiveIndex\" : The refractive index of the medium where
3834             the transitions are taking place. This may be a number or a
3835             function. If a number then the oscillator strengths are calculated
3836             for assuming a wavelength-independent refractive index. If a
3837             function then the refractive indices are calculated accordingly to
3838             the wavelength of each transition (the function must admit a
3839             single argument equal to the wavelength in nm). The default is 1.
3840             \"LocalFieldCorrection\" : The local field correction to be used.
3841             The default is \"VirtualCavity\". The options are: \
3842             \"VirtualCavity\" and \"EmptyCavity\".
3843             The equation implemented here is the one given in eqn. 29 from the
3844             review article of Hehlen (2013). See that same article for a
3845             discussion on the local field correction.

```

```

3816 ";
3817 Options[LevelElecDipoleOscillatorStrength]={
3818 "PrintFun"      -> PrintTemporary,
3819 "RefractiveIndex" -> 1,
3820 "LocalFieldCorrection" -> "VirtualCavity"
3821 };
3822 LevelElecDipoleOscillatorStrength[numE_, levelParams_Association,
3823 juddOfeltParams_Association, OptionsPattern[]] := Module[
3824 {PrintFun, basis, eigenSys, levelLabels,
3825 LevelUkSquared, eigenEnergies, energyDiffs,
3826 oStrengthArray, nRef, \[Chi], nRefs,
3827 \[Chi]OverN, groundLevel, const,
3828 transitionFrequencies, wavelengthsInNM,
3829 fieldCorrectionType},
3830 (
3831   PrintFun = OptionValue["PrintFun"];
3832   nRef = OptionValue["RefractiveIndex"];
3833   PrintFun["Calculating the Uk^2 matrix elements for the given
parameters ..."];
3834   {basis, eigenSys, levelLabels, LevelUkSquared} =
3835 JuddOfeltUkSquared[numE, levelParams, "PrintFun" -> PrintFun];
3836   eigenEnergies = First@eigenSys;
3837   (* converted to cm^-2 *)
3838   const = (8\[Pi]^2)/3 me/hPlanck * 10^(-4);
3839   energyDiffs = Transpose@Outer[Subtract, eigenEnergies,
3840 eigenEnergies];
3841   (* since energies are assumed in Kayser, speed of light needs
to be in cm/s, so that the frequencies are in 1/s *)
3842   transitionFrequencies = energyDiffs*cLight*100;
3843   (* grab the J for each level *)
3844   levelJs = #[[2]] & /@ eigenSys;
3845   oStrengthArray = (
3846     juddOfeltParams[\[CapitalOmega]2]*LevelUkSquared[2]+
3847     juddOfeltParams[\[CapitalOmega]4]*LevelUkSquared[4]+
3848     juddOfeltParams[\[CapitalOmega]6]*LevelUkSquared[6]
3849   );
3850   oStrengthArray = Abs@(const * transitionFrequencies *
3851 oStrengthArray);
3852   (* it is necessary to divide each oscillator strength by the
degeneracy of the initial level *)
3853   oStrengthArray = MapIndexed[1/(2 levelJs[[#2[[1]]]]+1) #1 &,
3854 oStrengthArray, {2}];
3855   (* including the effects of the refractive index *)
3856   fieldCorrectionType = OptionValue["LocalFieldCorrection"];
3857   Which[
3858     nRef === 1,
3859     True,
3860     NumberQ[nRef],
3861     (
3862       \[Chi] = Which[
3863         fieldCorrectionType == "VirtualCavity",
3864         (
3865           (nRef^2 + 2) / 3 )^2
3866         ),
3867         fieldCorrectionType == "EmptyCavity",
3868         (
3869           (3 * nRef^2 / (2 * nRef^2 + 1)) ^2
3870         )
3871       ];
3872       \[Chi]OverN = \[Chi] / nRef;
3873       oStrengthArray = \[Chi]OverN * oStrengthArray;
3874       (* the refractive index participates differently in
absorption and in emission *)
3875       aFunction = If[#2[[1]] > #2[[2]], #1 * nRef^2, #1]&;
3876       oStrengthArray = MapIndexed[aFunction, oStrengthArray,
3877 {2}];
3878     ),
3879     True,
3880     (
3881       wavelengthsInNM = Abs[1 / energyDiffs] * 10^7;
3882       nRefs = Map[nRef, wavelengthsInNM];
3883       Print["Calculating the oscillator strengths for the given
refractive index ..."];
3884       \[Chi] = Which[
3885         fieldCorrectionType == "VirtualCavity",
3886         (

```

```

3881          ( (nRefs^2 + 2) / 3 )^2
3882      ),
3883      fieldCorrectionType == "EmptyCavity",
3884      (
3885          ( 3 * nRefs^2 / ( 2*nRefs^2 + 1 ) )^2
3886      )
3887  ];
3888  \[Chi]OverN = \[Chi] / nRefs;
3889  oStrengthArray = \[Chi]OverN * oStrengthArray
3890  )
3891 ];
3892 Return[{basis, eigenSys, levelLabels, oStrengthArray}];
3893 )
3894 ];
3895
3896 LevelJJBlockMagDipole::usage = "LevelJJBlockMagDipole[numE, J, Jp]
3897 returns an array of the LSJ reduced matrix elements of the
3898 magnetic dipole operator between states with given J and Jp. The
3899 option \"Sparse\" can be used to return a sparse matrix. The
3900 default is to return a sparse matrix.";
3901 Options[LevelJJBlockMagDipole] = {"Sparse" -> True};
3902 LevelJJBlockMagDipole[numE_, braJ_, ketJ_, OptionsPattern[]] :=
3903 Module[
3904 {
3905   braSLJs, ketSLJs,
3906   braSLJ, ketSLJ,
3907   braSL, ketSL,
3908   braS, braL,
3909   ketS, ketL,
3910   matValue, magMatrix,
3911   summand1, summand2
3912 },
3913 (
3914   braSLJs = AllowedNKSLforJTerms[numE, braJ];
3915   ketSLJs = AllowedNKSLforJTerms[numE, ketJ];
3916   magMatrix = Table[
3917     (
3918       braSL = braSLJ[[1]];
3919       ketSL = ketSLJ[[1]];
3920       {braS, braL} = FindSL[braSL];
3921       {ketS, ketL} = FindSL[ketSL];
3922       summand1 = If[Or[braJ != ketJ, braSL != ketSL],
3923                     0,
3924                     Sqrt[braJ*(braJ+1)*TPO[braJ]]
3925                   ];
3926       (*looking at the string includes checking L=L', S=S', and \alpha
3927 = \alpha'*)
3928       summand2 = If[braSL != ketSL,
3929                     0,
3930                     (gs-1)*
3931                     Phaser[braS+braL+ketJ+1]*
3932                     Sqrt[TPO[braJ]*TPO[ketJ]]*
3933                     SixJay[{braJ, 1, ketJ}, {braS, braL, braS}]*
3934                     Sqrt[braS(braS+1) TPO[braS]]
3935                   ];
3936       matValue = summand1 + summand2;
3937       matValue = -1/2 * matValue;
3938       matValue
3939     ),
3940     {braSLJ, braSLJs},
3941     {ketSLJ, ketSLJs}
3942   ];
3943   If[OptionValue["Sparse"],
3944     magMatrix = SparseArray[magMatrix]];
3945   Return[magMatrix];
3946 )
3947 ];
3948
3949 LevelMagDipoleMatrixAssembly::usage = "LevelMagDipoleMatrixAssembly
3950 [numE] puts together an array with the reduced matrix elements of
3951 the magnetic dipole operator in the level basis for the f^numE
3952 configuration. The function admits the two following options:
3953 \"Flattened\": If True then the returned matrix is flattened. The
3954 default is True.
3955 \"gs\": The electronic gyromagnetic ratio. The default is 2.";
```

```

3947 Options[LevelMagDipoleMatrixAssembly] = {
3948   "Flattened" -> True,
3949   gs -> 2
3950 };
3951 LevelMagDipoleMatrixAssembly[numE_, OptionsPattern[]] := Module[
3952   {Js, magDip, braJ, ketJ},
3953   (
3954     Js      = AllowedJ[numE];
3955     magDip = Table[
3956       ReplaceInSparseArray[LevelJJBlockMagDipole[numE, braJ, ketJ], 
3957       {gs -> OptionValue[gs]}, 
3958       {braJ, Js}, 
3959       {ketJ, Js}
3960     ];
3961     If[OptionValue["Flattened"],
3962       magDip = ArrayFlatten[magDip];
3963     ];
3964     Return[magDip];
3965   )
3966 ];
3967
3968 LevelMagDipoleLineStrength::usage = "LevelMagDipoleLineStrength[
eigenSys, numE] calculates the magnetic dipole line strengths for
an ion whose level description is determined by levelParams. The
function returns a square array whose elements represent the
magnetic dipole line strengths between the levels given in
eigenSys such that the element magDipoleLineStrength[[i,j]] is the
line strength between the levels | $\Psi_i$ > and | $\Psi_j$ >. Eigensys must be such that it consists of a
lists of lists where in each list the last element corresponds to
the eigenvector of a level (given as a row) in the standard basis
for levels of the f^numE configuration.
3969 The function admits the following options:
3970   \\"Units\\" : The units in which the line strengths are given. The
3971   default is \\"SI\\". The options are \\"SI\\" and \\"Hartree\\". If \\"SI\"
3972   \\" then the unit of the line strength is  $(A \ m^2)^2 = (J/T)^2$ . If \
3973   \\"Hartree\\" then the line strength is given in units of  $2 \ [\mu B]$ ;";
3974 Options[LevelMagDipoleLineStrength] = {
3975   "Units" -> "SI"
3976 };
3977 LevelMagDipoleLineStrength[theEigensys_List, numE0_Integer,
3978 OptionsPattern[]] := Module[
3979   {numE, levelMagOp, allEigenvecs, magDipoleLineStrength, units},
3980   (
3981     numE      = Min[14 - numE0, numE];
3982     levelMagOp = LevelMagDipoleMatrixAssembly[numE];
3983     allEigenvecs = Transpose[Last /@ theEigensys];
3984     units      = OptionValue["Units"];
3985     magDipoleLineStrength = Transpose[allEigenvecs]. 
3986     levelMagOp.allEigenvecs;
3987     magDipoleLineStrength = Abs[magDipoleLineStrength]^2;
3988     Which[
3989       units == "SI",
3990         Return[4 \ [\mu B]^2 * magDipoleLineStrength],
3991       units == "Hartree",
3992         Return[magDipoleLineStrength]
3993     ];
3994   )
3995 ];
3996
3997 LevelMagDipoleOscillatorStrength::usage =
3998 LevelMagDipoleOscillatorStrength[eigenSys, numE] calculates the
3999 magnetic dipole oscillator strengths for an ion whose levels are
4000 described by eigenSys in configuration f^numE. The refractive
4001 index of the medium is relevant, but here it is assumed to be 1,
4002 this can be changed through the option \\"RefractiveIndex\\".
4003 eigenSys must consist of a lists of lists with three elements: the
4004 first element being the energy of the level, the second element
4005 being the J of the level, and the third element being the
4006 eigenvector of the level.
4007 The function returns a list with the following elements:
4008   - basis : A list with the allowed {SL, J} terms in the f^numE
4009   configuration. Equal to BasisLSJ[numE].
4010   - eigenSys : A list with the eigensystem of the Hamiltonian for
4011   the f^n configuration in the level description.
4012   - magDipoleOstrength : A square array whose elements represent

```

the magnetic dipole oscillator strengths between the levels given in eigenSys such that the element magDipoleOstrength[[i,j]] is the oscillator strength between the levels $|\Psi_i\rangle$ and $|\Psi_j\rangle$. 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.

3996 The function admits the following option:
 3997 `"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.
 3998 For reference see equation (27.8) in Rudzikas (2007). The
 expression for the line strenght is the simplest when using atomic
 units, (27.8) is missing a factor of $\alpha^2.$ ";
 3999 Options[LevelMagDipoleOscillatorStrength]={
 4000 "RefractiveIndex" -> 1
 4001 };
 4002 LevelMagDipoleOscillatorStrength[eigenSys_, numE_, OptionsPattern
 4003 []]:=Module[
 4004 {eigenEnergies, eigenVecs, levelJs,
 4005 energyDiffs, magDipoleOstrength, nRef,
 4006 wavelengthsInNM, nRefs, degenDivisor},
 4007 (
 4008 basis = BasisLSJ[numE];
 4009 eigenEnergies = First/@eigenSys;
 4010 nRef = OptionValue["RefractiveIndex"];
 4011 eigenVecs = Last/@eigenSys;
 4012 levelJs = #[[2]]&/@eigenSys;
 4013 energyDiffs = -Outer[Subtract,eigenEnergies,eigenEnergies];
 4014 energyDiffs *= kayserToHartree;
 4015 magDipoleOstrength = LevelMagDipoleLineStrength[eigenSys, numE,
 4016 "Units"->"Hartree"];
 4017 magDipoleOstrength = 2/3 * α Fine^2 * energyDiffs *
 4018 magDipoleOstrength;
 4019 degenDivisor = #1 / (2 * levelJs[[#2[[1]]]] + 1) &;
 4020 magDipoleOstrength = MapIndexed[degenDivisor,
 4021 magDipoleOstrength, {2}];
 4022 Which[nRef==1,
 4023 True,
 4024 NumberQ[nRef],
 4025 (
 4026 magDipoleOstrength = nRef * magDipoleOstrength;
 4027),
 4028 True,
 4029 (
 4030 wavelengthsInNM = Abs[kayserToHartree / energyDiffs] *
 4031 10^7;
 4032 nRefs = Map[nRef, wavelengthsInNM];
 4033 magDipoleOstrength = nRefs * magDipoleOstrength;
 4034);
 4035 Return[{basis, eigenSys, magDipoleOstrength}];
 4036)
 4037];
 4038 LevelMagDipoleSpontaneousDecayRates::usage = "
 4039 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 $|\Psi_i\rangle$ to the level $|\Psi_j\rangle$. 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
 is `"SI"`. The options are `"SI"` and `"Hartree"`. If `"SI"` then
 the rates are given in s^{-1} . If `"Hartree"` then the rates are

```

4038 given in the atomic unit of frequency.
4039 + \\"RefractiveIndex\\" : The refractive index of the medium where
4040 the transitions are taking place. This may be a number or a
4041 function. If a number then the rates are calculated assuming a
4042 wavelength-independent refractive index as given. If a function
4043 then the refractive indices are calculated accordingly to the
4044 vaccum wavelength of each transition (the function must admit a
4045 single argument equal to the wavelength in nm). The default is 1."
4046 ;
4039 Options[LevelMagDipoleSpontaneousDecayRates] = {
4040 "Units" -> "SI",
4041 "RefractiveIndex" -> 1};
4042 LevelMagDipoleSpontaneousDecayRates[eigenSys_List, numE_Integer,
4043 OptionsPattern[]] := Module[
4044 {levMDlineStrength, eigenEnergies, energyDiffs,
4045 levelJs, spontaneousRatesInHartree, spontaneousRatesInSI,
4046 degenDivisor, units, nRef, nRefs, wavelengthsInNM},
4047 (
4048 nRef = OptionValue["RefractiveIndex"];
4049 units = OptionValue["Units"];
4050 levMDlineStrength =
4051 LowerTriangularize@LevelMagDipoleLineStrength[eigenSys, numE, "Units"
4052 "->"Hartree"];
4053 levMDlineStrength = SparseArray[levMDlineStrength];
4054 eigenEnergies = First /@ eigenSys;
4055 energyDiffs = Outer[Subtract, eigenEnergies,
4056 eigenEnergies];
4057 energyDiffs = kayserToHartree * energyDiffs;
4058 energyDiffs = SparseArray[LowerTriangularize[energyDiffs
4059 ]];
4060 levelJs = #[[2]] & /@ eigenSys;
4061 spontaneousRatesInHartree = 4/3 αFine^5 * energyDiffs^3 *
4062 levMDlineStrength;
4063 degenDivisor = #1 / (2*levelJs[[2[[1]]]] + 1) &;
4064 spontaneousRatesInHartree = MapIndexed[degenDivisor,
4065 spontaneousRatesInHartree, {2}];
4066 Which[nRef === 1,
4067 True,
4068 NumberQ[nRef],
4069 (
4070 spontaneousRatesInHartree = nRef^3 *
4071 spontaneousRatesInHartree;
4072 ),
4073 True,
4074 (
4075 wavelengthsInNM = Abs[kayserToHartree / energyDiffs] *
4076 10^7;
4077 nRefs = Map[nRef, wavelengthsInNM];
4078 spontaneousRatesInHartree = nRefs^3 *
4079 spontaneousRatesInHartree;
4080 )
4081 ];
4082 If[units == "SI",
4083 (
4084 spontaneousRatesInSI = 1/hartreeTime *
4085 spontaneousRatesInHartree;
4086 Return[SparseArray@spontaneousRatesInSI];
4087 ),
4088 Return[SparseArray@spontaneousRatesInHartree];
4089 ];
4090 ];
4091 ]
4092 (* ##### Optical Transitions for Levels ##### *)
4093 (* ##### Eigensystem analysis ##### *)
4094
4095 PrettySaundersSL::usage = "PrettySaundersSL[SL] produces a human-
4096 readable symbol for the spectroscopic term SL. SL can be either a
4097 string (in RS notation for the term) or a list of two numbers {S,
4098 L}. The option \"Representation\" can be used to specify whether
4099 the output is given as a symbol or as a ket. The default is \"Ket\"
4100 \".\";
4091 Options[PrettySaundersSL] = {"Representation" -> "Ket"};

```

```

4090 PrettySaundersSL[SL_, OptionsPattern[]] := (
4091   If[StringQ[SL],
4092     (
4093       {S,L} = FindSL[SL];
4094       L      = StringTake[SL,{2,-1}];
4095     ),
4096     {S,L}=SL
4097   ];
4098   pretty = RowBox[{  

4099     AdjustmentBox[Style[2*S+1,Smaller], BoxBaselineShift->-1,  

4100     BoxMargins->0],
4101     AdjustmentBox[PrintL[L]]
4102   }];
4103   pretty = DisplayForm[pretty];
4104   pretty = Which[
4105     OptionValue["Representation"]=="Ket",
4106     Ket[pretty],
4107     OptionValue["Representation"]=="Symbol",
4108     pretty
4109   ];
4110   Return[pretty];
4111 );
4112
4113 PrettySaundersSLJmJ::usage = "PrettySaundersSLJmJ[{SL, J, mJ}]  

4114 produces a formatted symbol for the given basis vector {SL, J, mJ  

4115 }.";
4116 Options[PrettySaundersSLJmJ] = {"Representation" -> "Ket"};
4117 PrettySaundersSLJmJ[{SL_, J_, mJ_}, OptionsPattern[]] := (If[
4118   StringQ[SL],
4119   ({S, L} = FindSL[SL];
4120     L = StringTake[SL, {2, -1}];
4121   ),
4122   {S, L} = SL);
4123   pretty = RowBox[{AdjustmentBox[Style[2*S + 1, Smaller],
4124     BoxBaselineShift -> -1, BoxMargins -> 0],
4125     AdjustmentBox[PrintL[L], BoxMargins -> -0.2],
4126     AdjustmentBox[
4127       Style[Row[{InputForm[J], ", ", mJ}], Small],
4128       BoxBaselineShift -> 1,
4129       BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]];
4130   pretty = DisplayForm[pretty];
4131   If[OptionValue["Representation"] == "Ket",
4132     pretty = Ket[pretty]
4133   ];
4134   Return[pretty];
4135 );
4136
4137 PrettySaundersSLJ::usage = "PrettySaundersSLJ[{SL, J}] produces a  

4138 formatted symbol for the given level {SL, J}. SL can be either a  

4139 list of two numbers representing S and L or a string representing  

4140 the spin multiplicity in spectroscopic notation. J is the total  

4141 angular momentum to which S and L have coupled to. The option \"  

4142 Representation\" can be used to specify whether the output is  

4143 given as a symbol or as a ket. The default is \"Ket\".";
4144 Options[PrettySaundersSLJ] = {"Representation"->"Ket"};
4145 PrettySaundersSLJ[{SL_,J_},OptionsPattern[]] := (
4146   If[StringQ[SL],
4147     (
4148       {S,L}=FindSL[SL];
4149       L=StringTake[SL,{2,-1}];
4150     ),
4151     {S,L}=SL
4152   ];
4153   pretty = RowBox[{  

4154     AdjustmentBox[Style[2*S+1,Smaller],BoxBaselineShift->-1,  

4155     BoxMargins->0],
4156     AdjustmentBox[PrintL[L],BoxMargins->-0.2],
4157     AdjustmentBox[Style[InputForm[J],Small,FontTracking->"Narrow"],  

4158     BoxBaselineShift->1,BoxMargins->{{0.7,0},{0.4,0.4}}]
4159   }];
4160   pretty = DisplayForm[pretty];
4161   pretty = Which[
4162     OptionValue["Representation"]=="Ket",
4163     Ket[pretty],

```

```

4155     OptionValue["Representation"] == "Symbol",
4156     pretty
4157   ];
4158   Return[pretty];
4159 );
4160
4161 BasisVecInRusselSaunders::usage = "BasisVecInRusselSaunders[
4162   basisVec] takes a basis vector in the format {LSstring, Jval,
4163   mJval} and returns a formatted symbol for the corresponding Russel
4164 -Saunders term.";
4165 BasisVecInRusselSaunders[basisVec_] := (
4166   {LSstring, Jval, mJval} = basisVec;
4167   Ket[PrettySaundersSLJMJ[basisVec]]
4168 );
4169
4170 LSJMTemplate =
4171 StringTemplate[
4172   "#!\\(*TemplateBox[{\\nRowBox[{\\\"LS\\\", \\",\\\", \\nRowBox[{\\\"J\\\", \
4173   \"=\", \\\"J\\\"}], \\",\\\", \\nRowBox[{\\\"mJ\\\", \\\"=\\\", \\\"mJ\\\"}]}],\\n\\
4174   \"Ket\\\"]\\)"];
4175
4176 BasisVecInLSJM::usage = "BasisVecInLSJM[basisVec] takes a basis
4177   vector in the format {{LSstring, Jval}, mJval}, nucSpin} and
4178   returns a formatted symbol for the corresponding LSJM term in the
4179   form |LS, J=..., mJ=...>.";
4180 BasisVecInLSJM[basisVec_] := (
4181   {LSstring, Jval, mJval} = basisVec;
4182   LSJMTemplate[<|
4183     "LS" -> LSstring,
4184     "J" -> ToString[Jval, InputForm],
4185     "mJ" -> ToString[mJval, InputForm]|>]
4186 );
4187
4188 ParseStates::usage = "ParseStates[eigenSys, basis] takes a list of
4189   eigenstates in terms of their coefficients in the given basis and
4190   returns a list of the same states in terms of their energy, LSJM
4191   symbol, J, mJ, S, L, LSJ symbol, and LS symbol. eigenSys is a list
4192   of lists with two elements, in each list the first element is the
4193   energy and the second one the corresponding eigenvector. The LS
4194   symbol returned corresponds to the term with the largest
4195   coefficient in the given basis.";
4196 ParseStates[states_, basis_, OptionsPattern[]} := Module[
4197   {parsedStates},
4198   (
4199     parsedStates = Table[((
4200       {energy, eigenVec} = state;
4201       maxTermIndex = Ordering[Abs[eigenVec]][[-1]];
4202       {LSstring, Jval, mJval} = basis[[maxTermIndex]];
4203       LSsymbol = Subscript[LSstring, {Jval, mJval}];
4204       LSJMsymbol = LSstring <> ToString[Jval,
4205         InputForm];
4206       {S, L} = FindSL[LSstring];
4207       {energy, LSstring, Jval, mJval, S, L, LSsymbol, LSJMsymbol}
4208     ), {
4209       state, states
4210     }];
4211     Return[parsedStates];
4212   )
4213 ];
4214
4215 ParseStatesByNumBasisVecs::usage = "ParseStatesByNumBasisVecs[
4216   eigenSys, basis, numBasisVecs, roundTo] takes a list of
4217   eigenstates (given in eigenSys) in terms of their coefficients in
4218   the given basis and returns a list of the same states in terms of
4219   their energy and the coefficients at most numBasisVecs basis
4220   vectors. By default roundTo is 0.01 and this is the value used to
4221   round the amplitude coefficients. eigenSys is a list of lists with
4222   two elements, in each list the first element is the energy and
4223   the second one the corresponding eigenvector.
4224   The option \"Coefficients\" can be used to specify whether the
4225   coefficients are given as \"Amplitudes\" or \"Probabilities\". The
4226   default is \"Amplitudes\".
4227 ";
4228 Options[ParseStatesByNumBasisVecs] = {
4229   "Coefficients" -> "Amplitudes",
4230   "Representation" -> "Ket",

```

```

4207 "ReturnAs" -> "Dot"
4208 };
4209 ParseStatesByNumBasisVecs[eigensys_List, basis_List,
4210 numBasisVecs_Integer, roundTo_Real : 0.01, OptionsPattern[]] :=
4211 Module[
4212 {parsedStates, energy, eigenVec,
4213 probs, amplitudes, ordering,
4214 returnAs,
4215 chosenIndices, majorComponents,
4216 majorAmplitudes, majorRep},
4217 (
4218   returnAs = OptionValue["ReturnAs"];
4219   parsedStates = Table[(
4220     {energy, eigenVec} = state;
4221     energy = Chop[energy];
4222     probs = Round[Abs[eigenVec^2], roundTo];
4223     amplitudes = Round[eigenVec, roundTo];
4224     ordering = Ordering[probs];
4225     chosenIndices = ordering[[-numBasisVecs ;;]];
4226     majorComponents = basis[[chosenIndices]];
4227     majorThings = If[OptionValue["Coefficients"] == "Probabilities",
4228       (
4229         probs[[chosenIndices]]
4230       ),
4231       (
4232         amplitudes[[chosenIndices]]
4233       );
4234       majorComponents = PrettySaundersSLJmJ[#, "Representation" -> OptionValue["Representation"]] & /@ majorComponents;
4235       nonZ = (# != 0.) & /@ majorThings;
4236       majorThings = Pick[majorThings, nonZ];
4237       majorComponents = Pick[majorComponents, nonZ];
4238       If[OptionValue["Coefficients"] == "Probabilities",
4239         (
4240           majorThings = majorThings * 100 * "%";
4241         )
4242       ];
4243       majorRep = Which[
4244         returnAs == "Dot",
4245           majorThings . majorComponents,
4246           returnAs == "List",
4247             Transpose[{Reverse@majorThings,
4248             Reverse@majorComponents}]
4249           ];
4250           {energy, majorRep}
4251         ),
4252           {state, eigensys}];
4253       Return[parsedStates]
4254     )
4255   ];
4256 FindThresholdPosition::usage = "FindThresholdPosition[list,
4257 threshold] returns the position of the first element in list that
4258 is greater than or equal to threshold. If no such element exists,
4259 it returns the length of list. The elements of the given list must
4260 be in ascending order.";
4261 FindThresholdPosition[list_, threshold_] := Module[
4262 {position},
4263 (
4264 position = Position[list, _?(# >= threshold &), 1, 1];
4265 thrPos = If[Length[position] > 0,
4266   position[[1, 1]],
4267   Length[list]];
4268 If[thrPos == 0,
4269   Return[1],
4270   Return[thrPos]
4271 ]
4272 ];
4273 ParseStatesByProbabilitySum::usage = "ParseStatesByProbabilitySum[
4274 eigensys, basis, probSum] takes a list of eigenstates in terms of
4275 their coefficients in the given basis and returns a list of the
4276 same states in terms of their energy and the coefficients of the
4277 basis vectors that sum to at least probSum.";
```

```

4270 ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ : 0.01, maxParts_: 20] := Module[
4271   {parsedByProb, numStates, state, energy,
4272   eigenVec, amplitudes, probs, ordering,
4273   orderedProbs, truncationIndex, accProb,
4274   thresholdIndex, chosenIndices, majorComponents,
4275   majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
4276   (
4277     numStates = Length[eigensys];
4278     parsedByProb = Table[(  

4279       state = eigensys[[idx]];
4280       {energy, eigenVec} = state;
4281       (*Round them up*)
4282       amplitudes = Round[eigenVec, roundTo];
4283       probs = Round[Abs[eigenVec^2], roundTo];
4284       ordering = Reverse[Ordering[probs]];
4285       (*Order the probabilities from high to low*)
4286       orderedProbs = probs[[ordering]];
4287       (*To speed up Accumulate, assume that only as much as  

maxParts will be needed*)
4288       truncationIndex = Min[maxParts, Length[orderedProbs]];
4289       orderedProbs = orderedProbs[[;;truncationIndex]];
4290       (*Accumulate the probabilities*)
4291       accProb = Accumulate[orderedProbs];
4292       (*Find the index of the first element in accProb that is  

greater than probSum*)
4293       thresholdIndex = Min[Length[accProb],
4294       FindThresholdPosition[accProb, probSum]];
4295       (*Grab all the indicees up till that one*)
4296       chosenIndices = ordering[[;; thresholdIndex]];
4297       (*Select the corresponding elements from the basis*)
4298       majorComponents = basis[[chosenIndices]];
4299       (*Select the corresponding amplitudes*)
4300       majorAmplitudes = amplitudes[[chosenIndices]];
4301       (*Take their absolute value*)
4302       absMajorAmplitudes = Abs[majorAmplitudes];
4303       (*Make sure that there are no effectively zero contributions  

*)
4304       notnullAmplitudes = Flatten[Position[absMajorAmplitudes, x_ /; x != 0]];
4305       (* majorComponents = PrettySaundersSLJmJ  

{#[[1]], #[[2]], #[[3]]}] & /@ majorComponents; *)
4306       majorComponents = PrettySaundersSLJmJ /@ majorComponents;
4307       majorAmplitudes = majorAmplitudes[[notnullAmplitudes]];
4308       majorComponents = majorComponents[[notnullAmplitudes]];
4309       (*Multiply and add to build the final Ket*)
4310       majorRep = majorAmplitudes . majorComponents;
4311       {energy, majorRep}
4312     ), {idx, numStates}];
4313     Return[parsedByProb];
4314   )
4315 ];
4316 (* ##### Eigensystem analysis ##### *)
4317 (* ##### ##### ##### *)
4318 (* ##### ##### ##### *)
4319 (* ##### ##### ##### Misc ##### *)
4320 (* ##### ##### ##### *)
4321
4322 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.";  

4323 SymbToNum[expr_, replacementAssociation_] := (  

4324   includedKeys = Keys[replacementAssociation];
4325   (*If a key is not defined, make its value zero.*)
4326   fullAssociation = Table[(  

4327     If[MemberQ[includedKeys, key],  

4328       ToExpression[key] -> replacementAssociation[key],  

4329       ToExpression[key] -> 0
4330     ]
4331   ),
4332   {key, paramSymbols}];
4333   Return[expr/.fullAssociation];

```

```

4334 );
4335
4336 SimpleConjugate::usage = "SimpleConjugate[expr] takes an expression
4337     and applies a simplified version of the conjugate in that all it
4338     does is that it replaces the imaginary unit I with -I. It assumes
4339     that every other symbol is real so that it remains the same under
4340     complex conjugation. Among other expressions it is valid for any
4341     rational or polynomial expression with complex coefficients and
4342     real variables.";
4343 SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;
4344
4345 ExportMZip::usage = "ExportMZip[\"dest.[zip,m]\", expr] saves a
4346     compressed version of expr to the given destination.";
4347 ExportMZip[filename_, expr_] := Module[
4348     {baseName, exportName, mImportName, zipImportName},
4349     (
4350         baseName = FileBaseName[filename];
4351         exportName = StringReplace[filename, ".m" -> ".zip"];
4352         mImportName = StringReplace[exportName, ".zip" -> ".m"];
4353         If[FileExistsQ[mImportName],
4354             (
4355                 PrintTemporary[mImportName <> " exists already, deleting"];
4356                 DeleteFile[mImportName];
4357                 Pause[2];
4358             )
4359         ];
4360         Export[exportName, (baseName <> ".m") -> expr];
4361     )
4362 ];
4363
4364 ImportMZip::usage = "ImportMZip[filename] imports a .m file inside
4365     a .zip file with corresponding filename. If the Option \"Leave
4366     Uncompressed\" is set to True (the default) then this function
4367     also leaves an uncompressed version of the object in the same
4368     folder of filename";
4369 Options[ImportMZip] = {"Leave Uncompressed" -> True};
4370 ImportMZip[filename_String, OptionsPattern[]} := Module[
4371     {baseName, importKey, zipImportName, mImportName, mxImportName,
4372     imported},
4373     (
4374         baseName = FileBaseName[filename];
4375         (*Function allows for the filename to be .m or .zip*)
4376         importKey = baseName <> ".m";
4377         zipImportName = StringReplace[filename, ".m" -> ".zip"];
4378         mImportName = StringReplace[zipImportName, ".zip" -> ".m"];
4379         mxImportName = StringReplace[zipImportName, ".zip" -> ".mx"];
4380         Which[
4381             FileExistsQ[mxImportName],
4382             (
4383                 PrintTemporary[".mx version exists already, importing that
4384                 instead ..."];
4385                 Return[Import[mxImportName]];
4386             ),
4387             FileExistsQ[mImportName],
4388             (
4389                 PrintTemporary[".m version exists already, importing that
4390                 instead ..."];
4391                 imported = Import[mImportName];
4392                 If[OptionValue["Leave Uncompressed"],
4393                     (
4394                         Export[mImportName, imported];
4395                     )
4396                 ];
4397                 Return[Import[mImportName]];
4398             )
4399         ];
4400         imported = Import[zipImportName, importKey];
4401         If[OptionValue["Leave Uncompressed"],
4402             (
4403                 Export[mImportName, imported];
4404                 Export[mxImportName, imported];
4405             )
4406         ];
4407         Return[imported];
4408     )
4409 ];

```

```

4396
4397 ReplaceInSparseArray::usage = "ReplaceInSparseArray[sparseArray,
4398   rules] takes a sparse array that may contain symbolic quantities
4399   and returns a sparse array in which the given rules have been used
4400   on every element.";
4401 ReplaceInSparseArray[sparseA_SparseArray, rules_] := (
4402   SparseArray[Automatic,
4403     sparseA["Dimensions"],
4404     sparseA["Background"] /. rules,
4405     {
4406       1,
4407       {sparseA["RowPointers"], sparseA["ColumnIndices"]},
4408       sparseA["NonzeroValues"] /. rules
4409     }
4410   ]
4411 );
4412
4413 MapToSparseArray::usage = "MapToSparseArray[sparseArray, function]
4414   takes a sparse array and returns a sparse array after the function
4415   has been applied to it.";
4416 MapToSparseArray[sparseA_SparseArray, func_] := Module[
4417   {nonZ, backg, mapped},
4418   (
4419     nonZ = func /@ sparseA["NonzeroValues"];
4420     backg = func[sparseA["Background"]];
4421     mapped = SparseArray[Automatic,
4422       sparseA["Dimensions"],
4423       backg,
4424       {
4425         1,
4426         {sparseA["RowPointers"], sparseA["ColumnIndices"]},
4427         nonZ
4428       }
4429     ];
4430     Return[mapped];
4431   )
4432 ];
4433
4434 ParseTeXLikeSymbol::usage = "ParseTeXLikeSymbol[string] parses a
4435   string for a symbol given in LaTeX notation and returns a
4436   corresponding mathematica symbol. The string may have expressions
4437   for several symbols, they need to be separated by single spaces.
4438   In addition the _ and ^ symbols used in LaTeX notation need to
4439   have arguments that are enclosed in parenthesis, for example \"x_2
4440   \" is invalid, instead \"x_{2}\\" should have been given.";
4441 Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
4442 ParseTeXLikeSymbol[bigString_, OptionsPattern[]] := Module[
4443   {form, mainSymbol, symbols},
4444   (
4445     form = OptionValue["Form"];
4446     (* parse greek *)
4447     symbols = Table[(
4448       str = StringReplace[string, {"\\alpha" -> "\[Alpha]",
4449         "\\beta" -> "\[Beta]",
4450         "\\gamma" -> "\[Gamma]",
4451         "\\psi" -> "\[Psi]"}];
4452       symbol = Which[
4453         StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"]
4454       ],
4455         (
4456           (*yes sub no sup*)
4457           mainSymbol = StringSplit[str, "_"][[1]];
4458           mainSymbol = ToExpression[mainSymbol];
4459
4460           subPart =
4461             StringCases[str,
4462               RegularExpression@"\\{(.*)}\\}" -> "$1"][[1]];
4463             Subscript[mainSymbol, subPart]
4464           ),
4465           Not[StringContainsQ[str, "_"]] && StringContainsQ[str, "^"]
4466         ],
4467         (
4468           (*no sub yes sup*)
4469           mainSymbol = StringSplit[str, "^"][[1]];
4470           mainSymbol = ToExpression[mainSymbol];
4471
4472
4473
4474
4475
4476
4477
4478
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4520
4521
4522
4523
4524
4525
4526
4527
4528
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4560
4561
4562
4563
4564
4565
4566
4567
4568
4569
4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4580
4581
4582
4583
4584
4585
4586
4587
4588
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4599
4600
4601
4602
4603
4604
4605
4606
4607
4608
4609
4610
4611
4612
4613
4614
4615
4616
4617
4618
4619
4620
4621
4622
4623
4624
4625
4626
4627
4628
4629
4630
4631
4632
4633
4634
4635
4636
4637
4638
4639
4640
4641
4642
4643
4644
4645
4646
4647
4648
4649
4650
4651
4652
4653
4654
4655
4656
4657
4658
4659
4660
4661
4662
4663
4664
4665
4666
4667
4668
4669
4670
4671
4672
4673
4674
4675
4676
4677
4678
4679
4680
4681
4682
4683
4684
4685
4686
4687
4688
4689
4690
4691
4692
4693
4694
4695
4696
4697
4698
4699
4700
4701
4702
4703
4704
4705
4706
4707
4708
4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4730
4731
4732
4733
4734
4735
4736
4737
4738
4739
4740
4741
4742
4743
4744
4745
4746
4747
4748
4749
4750
4751
4752
4753
4754
4755
4756
4757
4758
4759
4760
4761
4762
4763
4764
4765
4766
4767
4768
4769
4770
4771
4772
4773
4774
4775
4776
4777
4778
4779
4780
4781
4782
4783
4784
4785
4786
4787
4788
4789
4790
4791
4792
4793
4794
4795
4796
4797
4798
4799
4800
4801
4802
4803
4804
4805
4806
4807
4808
4809
4810
4811
4812
4813
4814
4815
4816
4817
4818
4819
4820
4821
4822
4823
4824
4825
4826
4827
4828
4829
4830
4831
4832
4833
4834
4835
4836
4837
4838
4839
4840
4841
4842
4843
4844
4845
4846
4847
4848
4849
4850
4851
4852
4853
4854
4855
4856
4857
4858
4859
4860
4861
4862
4863
4864
4865
4866
4867
4868
4869
4870
4871
4872
4873
4874
4875
4876
4877
4878
4879
4880
4881
4882
4883
4884
4885
4886
4887
4888
4889
4890
4891
4892
4893
4894
4895
4896
4897
4898
4899
4900
4901
4902
4903
4904
4905
4906
4907
4908
4909
4910
4911
4912
4913
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
5099

```

```

4459         supPart =
4460             StringCases[str,
4461                 RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
4462             Superscript[mainSymbol, supPart]
4463         ),
4464         StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
4465         (
4466             (*yes sub yes sup*)
4467             mainSymbol = StringSplit[str, "_"][[1]];
4468             mainSymbol = ToExpression[mainSymbol];
4469             {subPart, supPart} =
4470                 StringCases[str, RegularExpression@"\\{(.*?)\\}" -> "
4471 $1"];
4472                 Subsuperscript[mainSymbol, subPart, supPart]
4473             ),
4474             True,
4475             (
4476                 (*no sup or sub*)
4477                 str
4478             );
4479             symbol
4480         ),
4481         {string, StringSplit[bigString, " "]}
4482     ];
4483     Which[
4484         form == "Row",
4485         Return[Row[symbols]],
4486         form == "List",
4487         Return[symbols]
4488     ]
4489   )
4490 ];
4491
4492 FromArrayToTable::usage = "FromArrayToTable[array, labels, energies
4493 ] takes a square array of values and returns a table with the
4494 labels of the rows and columns, the energies of the initial and
4495 final levels, the level energies, the vacuum wavelength of the
4496 transition, and the value of the array. The array must be square
4497 and the labels and energies must be compatible with the order
4498 implied by the array. The array must be a square array of values.
4499 The function returns a list of lists with the following elements:
4500 - Initial level index
4501 - Final level index
4502 - Initial level label
4503 - Final level label
4504 - Initial level energy
4505 - Final level energy
4506 - Vacuum wavelength
4507 - Value of the array element.
4508 - The reciprocal of the value of the array element.
4509 Elements in which the array is zero are not included in the return
4510 of this function.";
4511 FromArrayToTable[array_, labels_, energies_] := Module[
4512     {tableFun, atl},
4513     (
4514         tableFun = {
4515             #2[[1]],
4516             #2[[2]],
4517             labels[[#2[[1]]]],
4518             labels[[#2[[2]]]],
4519             energies[[#2[[1]]]],
4520             energies[[#2[[2]]]],
4521             If[#2[[1]] == #2[[2]], "--", 10^7/(energies[[#2[[1]]]] - energies
4522 [[#2[[2]]]]]),
4523             #1
4524         }&;
4525         atl = Select[Flatten[MapIndexed[tableFun, array
4526 , {2}], 1], ##[[-1]] != 0.&];
4527         atl = Append[#, 1/##[[-1]]]&/@atl;
4528         Return[atl]
4529     )
4530 ];
4531 (* ##### Misc ##### *)
4532 (* ##### *)

```

```

4524 (* ##### Some Plotting Routines ##### *)
4525 (* ##### Some Plotting Routines ##### *)
4526
4527 EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes
4528 states and produces a visualization of its energy spectrum.
4529 The resultant visualization can be navigated by clicking and
4530 dragging to zoom in on a region, or by clicking and dragging
4531 horizontally while pressing Ctrl. Double-click to reset the view."
4532 ;
4533 Options[EnergyLevelDiagram] = {
4534 "Title" -> "",
4535 "ImageSize" -> 1000,
4536 "AspectRatio" -> 1/8,
4537 "Background" -> "Automatic",
4538 "Epilog" -> {},
4539 "Explorer" -> True,
4540 "Energy Unit" -> "cm^-1"
4541 };
4542 EnergyLevelDiagram[states_, OptionsPattern[]}]:=Module[
4543 {energies, epi, explora},
4544 (
4545 energies = If[Length[Dimensions[states]]==1,
4546 states,
4547 First/@states
4548 ];
4549 epi = OptionValue["Epilog"];
4550 explora = If[OptionValue["Explorer"],
4551 ExploreGraphics,
4552 Identity
4553 ];
4554 frameLabel = "E (" <> OptionValue["Energy Unit"] <> ")";
4555 plotTips = Which[
4556 OptionValue["Energy Unit"] == "cm^-1",
4557 Tooltip[{#, 0}, {#, 1}], {Quantity[#/8065.54429, "eV"],
4558 Quantity[#, 1/"Centimeters"]}] &/@ energies,
4559 OptionValue["Energy Unit"] == "eV",
4560 Tooltip[{#, 0}, {#, 1}], {Quantity[#[# * 8065.54429, 1/
4561 Centimeters], Quantity[#, "eV"]}] &/@ energies,
4562 True,
4563 Tooltip[{#, 0}, {#, 1}], Quantity[#, OptionValue["Energy
4564 Unit"]]] &/@ energies
4565 ];
4566 explora@ListPlot[plotTips,
4567 Joined -> True,
4568 PlotStyle -> Black,
4569 AspectRatio -> OptionValue["AspectRatio"],
4570 ImageSize -> OptionValue["ImageSize"],
4571 Frame -> True,
4572 PlotRange -> {All, {0, 1}},
4573 FrameTicks -> {{None, None}, {Automatic, Automatic}},
4574 FrameStyle -> Directive[15, Dashed, Thin],
4575 PlotLabel -> Style[OptionValue["Title"], 15, Bold],
4576 Background -> OptionValue["Background"],
4577 FrameLabel -> {frameLabel},
4578 Epilog -> epi]
4579 )
4580 ]
4581 ];
4582
4583 ExploreGraphics::usage = "Pass a Graphics object to explore it.
4584 Zoom by clicking and dragging a rectangle. Pan by clicking and
4585 dragging while pressing Ctrl. Click twice to reset view.
4586 Based on ZeitPolizei @ https://mathematica.stackexchange.com/questions/7142/how-to-manipulate-2d-plots.
4587 The option \"OptAxesRedraw\" can be used to specify whether the
4588 axes should be redrawn. The default is False.";
4589 Options[ExploreGraphics] = {OptAxesRedraw -> False};
4590 ExploreGraphics[graph_Graphics, opts : OptionsPattern[]]:=With[
4591 {
4592 gr = First[graph],
4593 opt = DeleteCases[Options[graph],
4594 PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _],
4595 plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
4596 ar = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
4597 ao = AbsoluteOptions[AxesOrigin],
4598 rectangle = {Dashing[Small],
4599 Line[{#1,
4600
4601
4602
4603
4604
4605
4606
4607
4608
4609
4610
4611
4612
4613
4614
4615
4616
4617
4618
4619
4620
4621
4622
4623
4624
4625
4626
4627
4628
4629
4630
4631
4632
4633
4634
4635
4636
4637
4638
4639
4640
4641
4642
4643
4644
4645
4646
4647
4648
4649
4650
4651
4652
4653
4654
4655
4656
4657
4658
4659
4660
4661
4662
4663
4664
4665
4666
4667
4668
4669
4670
4671
4672
4673
4674
4675
4676
4677
4678
4679
4680
4681
4682
4683
4684
4685
4686
4687
4688
4689
4690
4691
4692
4693
4694
4695
4696
4697
4698
4699
4700
4701
4702
4703
4704
4705
4706
4707
4708
4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4730
4731
4732
4733
4734
4735
4736
4737
4738
4739
4740
4741
4742
4743
4744
4745
4746
4747
4748
4749
4750
4751
4752
4753
4754
4755
4756
4757
4758
4759
4760
4761
4762
4763
4764
4765
4766
4767
4768
4769
4770
4771
4772
4773
4774
4775
4776
4777
4778
4779
4780
4781
4782
4783
4784
4785
4786
4787
4788
4789
4790
4791
4792
4793
4794
4795
4796
4797
4798
4799
4800
4801
4802
4803
4804
4805
4806
4807
4808
4809
4810
4811
4812
4813
4814
4815
4816
4817
4818
4819
4820
4821
4822
4823
4824
4825
4826
4827
4828
4829
4830
4831
4832
4833
4834
4835
4836
4837
4838
4839
4840
4841
4842
4843
4844
4845
4846
4847
4848
4849
4850
4851
4852
4853
4854
4855
4856
4857
4858
4859
4860
4861
4862
4863
4864
4865
4866
4867
4868
4869
4870
4871
4872
4873
4874
4875
4876
4877
4878
4879
4880
4881
4882
4883
4884
4885
4886
4887
4888
4889
4890
4891
4892
4893
4894
4895
4896
4897
4898
4899
4900
4901
4902
4903
4904
4905
4906
4907
4908
4909
4910
4911
4912
4913
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
5599

```

```

4589          {First[#2], Last[#1]},
4590          #2,
4591          {First[#1], Last[#2]},
4592          #1}]} &,
4593 optAxesRedraw = OptionValue[OptAxesRedraw]
4594 },
4595 DynamicModule[
4596     {dragging=False, first, second, rx1, rx2, ry1, ry2,
4597      range = plr},
4598      {{rx1, rx2}, {ry1, ry2}} = plr;
4599 Panel@
4600 EventHandler[
4601     Dynamic@Graphics[
4602         If[dragging, {gr, rectangle[first, second]}, gr],
4603         PlotRange -> Dynamic@range,
4604         AspectRatio -> ar,
4605         AxesOrigin -> If[optAxesRedraw,
4606             Dynamic@Mean[range\[Transpose]], ao],
4607             Sequence @@ opt],
4608             {"MouseDown", 1} :> (
4609                 first = MousePosition["Graphics"]
4610             ),
4611             {"MouseDragged", 1} :> (
4612                 dragging = True;
4613                 second = MousePosition["Graphics"]
4614             ),
4615             "MouseClicked" :> (
4616                 If[CurrentValue@"MouseClickedCount"==2,
4617                     range = plr];
4618             ),
4619             {"MouseUp", 1} :> If[dragging,
4620                 dragging = False;
4621
4622                 range = {{rx1, rx2}, {ry1, ry2}} =
4623                     Transpose@{first, second};
4624                     range[[2]] = {0, 1}],
4625                     {"MouseDown", 2} :> (
4626                         first = {sx1, sy1} = MousePosition["Graphics"]
4627                     ),
4628                     {"MouseDragged", 2} :> (
4629                         second = {sx2, sy2} = MousePosition["Graphics"];
4630                         rx1 = rx1 - (sx2 - sx1);
4631                         rx2 = rx2 - (sx2 - sx1);
4632                         ry1 = ry1 - (sy2 - sy1);
4633                         ry2 = ry2 - (sy2 - sy1);
4634                         range = {{rx1, rx2}, {ry1, ry2}};
4635                         range[[2]] = {0, 1};
4636                     )}]];
4637
4638 LabeledGrid::usage = "LabeledGrid[data, rowHeaders, columnHeaders]
provides a grid of given data interpreted as a matrix of values
whose rows are labeled by rowHeaders and whose columns are labeled
by columnHeaders. When hovering with the mouse over the grid
elements, the row and column labels are displayed with the given
separator between them.";
4639 Options[LabeledGrid]={
4640     ItemSize->Automatic,
4641     Alignment->Center,
4642     Frame->All,
4643     "Separator"->",",
4644     "Pivot"->""
4645 };
4646 LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]]:=Module[
4647 {gridList=data, rowHeads=rowHeaders, colHeads=columnHeaders},
4648 (
4649     separator=OptionValue["Separator"];
4650     pivot=OptionValue["Pivot"];
4651     gridList=Table[
4652         Tooltip[
4653             data[[rowIdx,colIdx]],
4654             DisplayForm[
4655                 RowBox[{rowHeads[[rowIdx]],
4656                     separator,
4657                     colHeads[[colIdx]]}
4658                 ]

```

```

4659 ]
4660 ],
4661 {rowIdx,Dimensions[data][[1]]},
4662 {colIdx,Dimensions[data][[2]]}];
4663 gridList=Transpose[Prepend[gridList,colHeads]];
4664 rowHeads=Prepend[rowHeads,pivot];
4665 gridList=Prepend[gridList,rowHeads]//Transpose;
4666 Grid[gridList,
4667 Frame->OptionValue[Frame],
4668 Alignment->OptionValue[Alignment],
4669 Frame->OptionValue[Frame],
4670 ItemSize->OptionValue[ItemSize]
4671 ]
4672 )
4673 ];
4674
4675 HamiltonianForm::usage = "HamiltonianForm[hamMatrix, basisLabels]
4676 takes the matrix representation of a hamiltonian together with a
4677 set of symbols representing the ordered basis in which the
4678 operator is represented. With this it creates a displayed form
4679 that has adequately labeled row and columns together with
4680 informative values when hovering over the matrix elements using
4681 the mouse cursor.";
4682 Options[HamiltonianForm]={"Separator"->"", "Pivot"->};
4683 HamiltonianForm[hamMatrix_, basisLabels_List, OptionsPattern[]]:=(
4684 braLabels=DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]]& /@ basisLabels;
4685 ketLabels=DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"}]]& /@ basisLabels;
4686 LabeledGrid[hamMatrix,braLabels,ketLabels,"Separator"->
4687 OptionValue["Separator"], "Pivot"->OptionValue["Pivot"]]
4688 )
4689
4690 HamiltonianMatrixPlot::usage = "HamiltonianMatrixPlot[hamMatrix,
4691 basisLabels] creates a matrix plot of the given hamiltonian matrix
4692 with the given basis labels. The matrix elements can be hovered
4693 over to display the corresponding row and column labels together
4694 with the value of the matrix element. The option \"OverlayValues\""
4695 can be used to specify whether the matrix elements should be
4696 displayed on top of the matrix plot.";
4697 Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover"-
4698 -> True, "OverlayValues" -> True}];
4699 HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts : OptionsPattern[]]:=(
4700 braLabels = DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]] & /@ basisLabels;
4701 ketLabels = DisplayForm[Rotate[RowBox[{"\[LeftBracketingBar]", #,
4702 "\[RightAngleBracket]"}],\[Pi]/2]] & /@ basisLabels;
4703 ketLabelsUpright = DisplayForm[RowBox[{"\[LeftBracketingBar]", #,
4704 "\[RightAngleBracket]"}]] & /@ basisLabels;
4705 numRows = Length[hamMatrix];
4706 numCols = Length[hamMatrix[[1]]];
4707 epiThings = Which[
4708 And[OptionValue["Hover"], Not[OptionValue["OverlayValues"]]],
4709 Flatten[
4710 Table[
4711 Tooltip[
4712 {
4713 Transparent,
4714 Rectangle[
4715 {j - 1, numRows - i},
4716 {j - 1, numRows - i} + {1, 1}
4717 ]
4718 },
4719 Row[{braLabels[[i]],ketLabelsUpright[[j]],"=",hamMatrix[[i,
4720 j]]}]
4721 ],
4722 {{i, 1, numRows},
4723 {j, 1, numCols}
4724 }
4725 ],
4726 ],
4727 And[OptionValue["Hover"], OptionValue["OverlayValues"]],
4728 Flatten[
4729 Table[
4730 Tooltip[

```

```

4713 {
4714   Transparent ,
4715   Rectangle[
4716     {j - 1, numRows - i},
4717     {j - 1, numRows - i} + {1, 1}
4718   ]
4719 },
4720   DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i
4721 ]], "\[LeftBracketingBar]", basisLabels[[j]], "\[RightAngleBracket"
4722 ]}]]
4723 ],
4724 {i, numRows},
4725 {j, numCols}
4726 ]
4727 ],
4728 True,
4729 {};
4730 ];
4731 textOverlay = If[OptionValue["OverlayValues"],
4732 (
4733   Flatten[
4734     Table[
4735       Text[hamMatrix[[i, j]],
4736         {j - 1/2, numRows - i + 1/2}
4737       ],
4738       {i, 1, numRows},
4739       {j, 1, numCols}
4740     ]
4741   ]
4742 ];
4743 epiThings = Join[epiThings, textOverlay];
4744 MatrixPlot[hamMatrix,
4745   FrameTicks -> {
4746     {Transpose[{Range[Length[braLabels]], braLabels}], None},
4747     {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}}
4748   ],
4749   Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
4750   Epilog -> epiThings
4751 ]
4752 );
4753
4754 (* ##### Some Plotting Routines ##### *)
4755 (* ##### ##### ##### ##### ##### ##### *)
4756
4757 (* ##### ##### ##### ##### ##### ##### *)
4758 (* ##### ##### ##### Load Functions ##### *)
4759
4760 LoadAll::usage = "LoadAll[] executes most Load* functions.";
4761 LoadAll[] := (
4762   LoadTermLabels[];
4763   LoadCFP[];
4764   LoadUk[];
4765   LoadV1k[];
4766   LoadT22[];
4767   LoadS00andECSOLS[];
4768
4769   LoadElectrostatic[];
4770   LoadSpinOrbit[];
4771   LoadS00andECS0[];
4772   LoadSpinSpin[];
4773   LoadThreeBody[];
4774   LoadChenDeltas[];
4775   LoadCarnall[];
4776 );
4777
4778 fnTermLabels::usage = "This list contains the labels of f^n
4779 configurations. Each element of the list has four elements {LS,
4780 seniority, W, U}. At first sight this seems to only include the
4781 labels for the f^6 and f^7 configuration, however, all is included
4782 in these two.";
4783
4784 LoadTermLabels::usage = "LoadTermLabels[] loads into the session
4785 the labels for the terms in the f^n configurations.";
4786 LoadTermLabels[] := (

```

```

4782 If[ValueQ[fnTermLabels], Return[]];
4783 PrintTemporary["Loading data for state labels in the fn
4784 configurations..."];
4785 fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
4786
4787 If[!FileExistsQ[fnTermsFname],
4788     (PrintTemporary[">> fnTerms.m not found, generating ..."];
4789      fnTermLabels = ParseTermLabels["Export" -> True];
4790     ),
4791     fnTermLabels = Import[fnTermsFname];
4792   ];
4793
4794 Carnall::usage = "Association of data from Carnall et al (1989)
4795 with the following keys: {data, annotations, paramSymbols,
4796 elementNames, rawData, rawAnnotations, annotatedData, appendix:Pr
4797 :Association, appendix:Pr:Calculated, appendix:Pr:RawTable,
4798 appendix:Headings}";
4799
4800 LoadCarnall::usage = "LoadCarnall[] loads data for trivalent
4801 lanthanides in LaF3 using the data from Bill Carnall's 1989 paper.
4802 ";
4803 LoadCarnall[] := (
4804   If[ValueQ[Carnall], Return[]];
4805   carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
4806   If[!FileExistsQ[carnallFname],
4807       (PrintTemporary[">> Carnall.m not found, generating ..."];
4808        Carnall = ParseCarnall[];
4809       ),
4810       Carnall = Import[carnallFname];
4811     ];
4812   );
4813
4814 LoadChenDeltas::usage = "LoadChenDeltas[] loads the differences
4815 noted by Chen.";
4816 LoadChenDeltas[] := (
4817   If[ValueQ[chenDeltas], Return[]];
4818   PrintTemporary["Loading the association of discrepancies found by
4819 Chen ..."];
4820   chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"
4821 }];
4822   If[!FileExistsQ[chenDeltasFname],
4823       (PrintTemporary[">> chenDeltas.m not found, generating ..."];
4824        chenDeltas = ParseChenDeltas[];
4825       ),
4826       chenDeltas = Import[chenDeltasFname];
4827     ];
4828   );
4829
4830 ParseChenDeltas::usage = "ParseChenDeltas[] parses the data found
4831 in ./data/the-chen-deltas-A.csv and ./data/the-chen-deltas-B.csv.
4832 If the option \"Export\" is set to True (True is the default),
4833 then the parsed data is saved to ./data/chenDeltas.m";
4834 Options[ParseChenDeltas] = {"Export" -> True};
4835 ParseChenDeltas[OptionsPattern[]] :=
4836   chenDeltasRaw = Import[FileNameJoin[{moduleDir, "data", "the-chen
4837 -deltas-A.csv"}]];
4838   chenDeltasRaw = chenDeltasRaw[[2 ;;]];
4839   chenDeltas = <||>;
4840   chenDeltasA = <||>;
4841   Off[Power::infy];
4842   Do[
4843     {right, wrong} = {chenDeltasRaw[[row]][[4 ;;]], chenDeltasRaw[[row + 1]][[4 ;;]]];
4844     key = chenDeltasRaw[[row]][[1 ;; 3]];
4845     repRule = (#[[1]] -> #[[2]]*#[[1]]) & /@ Transpose[{{M0, M2, M4, P2, P4, P6}, right/wrong}];
4846     chenDeltasA[key] = <|"right" -> right, "wrong" -> wrong,
4847     "repRule" -> repRule|>;
4848     chenDeltasA[{key[[1]], key[[3]], key[[2]]}] = <|"right" ->
4849     right,
4850     "wrong" -> wrong, "repRule" -> repRule|>;
4851   },
4852   {row, 1, Length[chenDeltasRaw], 2}];
4853   chenDeltas["A"] = chenDeltasA;
4854

```

```

4843 chenDeltasRawB = Import[FileNameJoin[{moduleDir, "data", "the-
4844 chen-deltas-B.csv"}], "Text"];
4845 chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
4846 chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
4847 chenDeltasB = {ToExpression[StringTake[#[[1]], {2}]], #[[2]],
4848 #[[3]]} & /@ chenDeltasB;
4849 chenDeltas["B"] = chenDeltasB;
4850 On[Power::infy];
4851 If[OptionValue["Export"],
4852   (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.
4853 .m"}];
4854   Export[chenDeltasFname, chenDeltas];
4855 )
4856 ];
4857
4858 ParseCarnall::usage = "ParseCarnall[] parses the data found in ./
4859 data/Carnall.xls. If the option \"Export\" is set to True (True is
4860 the default), then the parsed data is saved to ./data/Carnall.
4861 This data is from the tables and appendices of Carnall et al
4862 (1989).";
4863 Options[ParseCarnall] = {"Export" -> True};
4864 ParseCarnall[OptionsPattern[]] :=
4865   ions = {"Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho",
4866   "Er", "Tm", "Yb"};
4867   templates = StringTemplate/@StringSplit["appendix:`ion`:
4868 Association appendix:`ion`:Calculated appendix:`ion`:RawTable
4869 appendix:`ion`:Headings", " "];
4870
4871 (* How many unique eigenvalues , after removing Kramer's
4872 degeneracy *)
4873 fullSizes = AssociationThread[ions, {7, 91, 182, 1001, 1001,
4874 3003, 1716, 3003, 1001, 1001, 182, 91, 7}];
4875 carnall = Import[FileNameJoin[{moduleDir, "data", "Carnall.xls
4876 "}]][[2]];
4877 carnallErr = Import[FileNameJoin[{moduleDir, "data", "Carnall.xls
4878 "}]][[3]];
4879
4880 elementNames = carnall[[1]][[2;;]];
4881 carnall = carnall[[2;;]];
4882 carnallErr = carnallErr[[2;;]];
4883 carnall = Transpose[carnall];
4884 carnallErr = Transpose[carnallErr];
4885 paramNames = ToExpression/@carnall[[1]][[1;;]];
4886 carnall = carnall[[2;;]];
4887 carnallErr = carnallErr[[2;;]];
4888 carnallData = Table[(
4889   data = carnall[[i]];
4890   data = (#[[1]] -> #[[2]]) & /@ Select[
4891     Transpose[{paramNames, data}], #[[2]] != "" &];
4892     elementNames[[i]] -> data
4893   ),
4894   {i, 1, 13}
4895 ];
4896 carnallData = Association[carnallData];
4897 carnallNotes = Table[((
4898   data = carnallErr[[i]];
4899   elementName = elementNames[[i]];
5000   dataFun = (
5001     #[[1]] -> If[#[[2]] == {},
5002       "Not allowed to vary in fitting.",
5003       If[#[[2]] == "[R]",
5004         "Ratio constrained by: " <> <|"Eu" ->"F4/
5005 F2=0.713; F6/F2=0.512",
5006         "Gd" -> "F4/F2=0.710",
5007         "Tb" -> "F4/F2=0.707" |> [elementName],
5008       If[#[[2]] == "i",
5009         "Interpolated",
5010         #[[2]]
5011       ]
5012     ]
5013   )
5014 ) &;
5015   data = dataFun /@ Select[Transpose[{paramNames,
5016   data}], #[[2]] != "" &];
5017   elementName -> data
5018 ];

```

```

4902         ),
4903 {i,1,13}
4904 ];
4905 carnallNotes = Association[carnallNotes];
4906
4907 annotatedData = Table[
4908     If[NumberQ[#[[1]]], Tooltip[#[[1]], #[[2]]], ""] & /
4909     @ Transpose[{paramNames/.carnallData[element],
4910                 paramNames/.carnallNotes[element]
4911             }],
4912             {element,elementNames}
4913         ];
4914 annotatedData = Transpose[annotatedData];
4915
4916 Carnall = <|"data"      -> carnallData,
4917 "annotations"    -> carnallNotes,
4918 "paramSymbols"   -> paramNames,
4919 "elementNames"   -> elementNames,
4920 "rawData"        -> carnall,
4921 "rawAnnotations" -> carnallErr,
4922 "includedTableIons" -> ions,
4923 "annnotatedData" -> annotatedData
4924 |>;
4925
4926 Do[(
4927     carnallData = Import[FileNameJoin[{moduleDir,"data",
4928     Carnall.xls"}]][[sheetIdx]];
4929     headers = carnallData[[1]];
4930     calcIndex = Position[headers,"Calc (1/cm)"][[1,1]];
4931     headers = headers[[2;;]];
4932     carnallLabels = carnallData[[1]];
4933     carnallData = carnallData[[2;;]];
4934     carnallTerms = DeleteDuplicates[First/@carnallData];
4935     parsedData = Table[(
4936         rows = Select[carnallData ,#[[1]]==term&];
4937         rows = #[[2;;]]&/@rows;
4938         rows = Transpose[rows];
4939         rows = Transpose[{headers,rows}];
4940         rows = Association[({#[[1]]->#[[2]]})&/@rows
4941     ];
4942         term->rows
4943     ),
4944     {term,carnallTerms}
4945 ];
4946     carnallAssoc = Association[parsedData];
4947     carnallCalcEnergies = #[[calcIndex]]&/@carnallData;
4948     carnallCalcEnergies = If[NumberQ[#],#,Missing[]]&/
4949     @carnallCalcEnergies;
4950     ion = ions[[sheetIdx-3]];
4951     carnallCalcEnergies = PadRight[carnallCalcEnergies, fullSizes
4952 [ion], Missing[]];
4953     keys = #[<"ion"-->ion|>]&/@templates;
4954     Carnall[keys[[1]]] = carnallAssoc;
4955     Carnall[keys[[2]]] = carnallCalcEnergies;
4956     Carnall[keys[[3]]] = carnallData;
4957     Carnall[keys[[4]]] = headers;
4958     ),
4959     {sheetIdx,4,16}
4960 ];
4961
4962     goodions = Select[ions,#!=="Pm"&];
4963     expData = Select[Transpose[Carnall["appendix:<>#<>":RawTable
4964 ]][[1+Position[Carnall["appendix:<>#<>":Headings],"Exp (1/cm)">[[1,1]]]],NumberQ]&/@goodions;
4965     Carnall["All Experimental Data"] = AssociationThread[goodions,
4966     expData];
4967     If[OptionValue["Export"],
4968     (
4969         carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"
4970     }];
4971         Print["Exporting to "<>carnallFname];
4972         Export[carnallFname, Carnall];
4973     )
4974     ];
4975     Return[Carnall];
4976 );

```

```

4969
4970 CFP::usage = "CFP[{n, NKSL}] provides a list whose first element
4971 echoes NKSL and whose other elements are lists with two elements
4972 the first one being the symbol of a parent term and the second
4973 being the corresponding coefficient of fractional parentage. n
4974 must satisfy 1 <= n <= 7.
4975 These are according to the tables from Nielson & Koster.";
4976
4977 CFPAssoc::usage = "CFPAssoc is an association where keys are of
4978 lists of the form {num_electrons, daughterTerm, parentTerm} and
4979 values are the corresponding coefficients of fractional parentage.
4980 The terms given in string-spectroscopic notation. If a certain
4981 daughter term does not have a parent term, the value is 0. Loaded
4982 using LoadCFP [].
4983 These are according to the tables from Nielson & Koster.";
4984
4985 LoadCFP::usage = "LoadCFP[] loads CFP, CFPAssoc, and CFPTable into
4986 the session.";
4987 LoadCFP[] := (
4988   If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]], Return
4989   []];
4990
4991   PrintTemporary["Loading CFPTable ..."];
4992   CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
4993   If[!FileExistsQ[CFPTablefname],
4994     (PrintTemporary[">> CFPTable.m not found, generating ..."]);
4995     CFPTable = GenerateCFPTable["Export" -> True];
4996   ),
4997     CFPTable = Import[CFPTablefname];
4998   ];
4999
5000   PrintTemporary["Loading CFPs.m ..."];
5001   CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
5002   If[!FileExistsQ[CFPfname],
5003     (PrintTemporary[">> CFPs.m not found, generating ..."]);
5004     CFP = GenerateCFP["Export" -> True];
5005   ),
5006     CFP = Import[CFPfname];
5007   ];
5008
5009   PrintTemporary["Loading CFPAssoc.m ..."];
5010   CFPAfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
5011   If[!FileExistsQ[CFPAfname],
5012     (PrintTemporary[">> CFPAssoc.m not found, generating ..."]);
5013     CFPAssoc = GenerateCFPAssoc["Export" -> True];
5014   ),
5015     CFPAssoc = Import[CFPAfname];
5016   ];
5017
5018 ReducedUkTable::usage = "ReducedUkTable[{n, l = 3, SL, SpLp, k}]
5019 provides reduced matrix elements of the unit spherical tensor
5020 operator Uk. See TASS section 11-9 \"Unit Tensor Operators\".
5021 Loaded using LoadUk[].";
5022
5023 LoadUk::usage = "LoadUk[] loads into session the reduced matrix
5024 elements for unit tensor operators.";
5025 LoadUk[] := (
5026   If[ValueQ[ReducedUkTable], Return[]];
5027   PrintTemporary["Loading the association of reduced matrix
5028 elements for unit tensor operators ..."];
5029   ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "ReducedUkTable.m"}];
5030   If[!FileExistsQ[ReducedUkTableFname],
5031     (PrintTemporary[">> ReducedUkTable.m not found, generating ..."]);
5032     ReducedUkTable = GenerateReducedUkTable[7];
5033   ),
5034     ReducedUkTable = Import[ReducedUkTableFname];
5035   ];
5036
5037 ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}]
5038 provides the calculated result of Electrostatic[{n, SL, SpLp}]."
5039 Load using LoadElectrostatic[].";
```

```

5025 LoadElectrostatic::usage = "LoadElectrostatic[] loads the reduced
5026   matrix elements for the electrostatic interaction.";
5027 LoadElectrostatic[] := (
5028   If[ValueQ[ElectrostaticTable], Return[]];
5029   PrintTemporary["Loading the association of matrix elements for
5030   the electrostatic interaction ..."];
5031   ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}];
5032   If[!FileExistsQ[ElectrostaticTablefname],
5033     (PrintTemporary[">> ElectrostaticTable.m not found, generating
5034     ..."]);
5035     ElectrostaticTable = GenerateElectrostaticTable[7];
5036   ),
5037   ElectrostaticTable = Import[ElectrostaticTablefname];
5038 ];
5039 );
5040
5041 LoadV1k::usage = "LoadV1k[] loads into session the matrix elements
5042   of V1k.";
5043 LoadV1k[] := (
5044   If[ValueQ[ReducedV1kTable], Return[]];
5045   PrintTemporary["Loading the association of matrix elements for
5046   V1k ..."];
5047   ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m"}];
5048   If[!FileExistsQ[ReducedV1kTableFname],
5049     (PrintTemporary[">> ReducedV1kTable.m not found, generating ...
5050     "]);
5051     ReducedV1kTable = GenerateReducedV1kTable[7];
5052   ),
5053   ReducedV1kTable = Import[ReducedV1kTableFname];
5054 ];
5055 );
5056
5057 LoadSpinOrbit::usage = "LoadSpinOrbit[] loads into session the
5058   matrix elements of the spin-orbit interaction.";
5059 LoadSpinOrbit[] := (
5060   If[ValueQ[SpinOrbitTable], Return[]];
5061   PrintTemporary["Loading the association of matrix elements for
5062   spin-orbit ..."];
5063   SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
5064   If[!FileExistsQ[SpinOrbitTableFname],
5065     (
5066       PrintTemporary[">> SpinOrbitTable.m not found, generating ...
5067     "];
5068       SpinOrbitTable = GenerateSpinOrbitTable[7, "Export" -> True];
5069     ),
5070     SpinOrbitTable = Import[SpinOrbitTableFname];
5071   ];
5072 );
5073
5074 LoadS00andECSOLS::usage = "LoadS00andECSOLS[] loads into session
5075   the LS reduced matrix elements of the S00-ECSO interaction.";
5076 LoadS00andECSOLS[] := (
5077   If[ValueQ[S00andECSOLSTable], Return[]];
5078   PrintTemporary["Loading the association of LS reduced matrix
5079   elements for S00-ECSO ..."];
5080   S00andECSOLSTableFname = FileNameJoin[{moduleDir, "data", "ReducedS00andECSOLSTable.m"}];
5081   If[!FileExistsQ[S00andECSOLSTableFname],
5082     (PrintTemporary[">> ReducedS00andECSOLSTable.m not found,
5083     generating ..."]);
5084     S00andECSOLSTable = GenerateS00andECSOLSTable[7];
5085   ),
5086   S00andECSOLSTable = Import[S00andECSOLSTableFname];
5087 ];
5088 );
5089
5090 LoadS00andECSO::usage = "LoadS00andECSO[] loads into session the
5091   LSJ reduced matrix elements of spin-other-orbit and
5092   electrostatically-correlated-spin-orbit.";
5093 LoadS00andECSO[] := (
5094   If[ValueQ[S00andECSOTableFname], Return[]];
5095   PrintTemporary["Loading the association of matrix elements for
5096   spin-other-orbit and electrostatically-correlated-spin-orbit ..."]

```

```

];
5082 S0OandECSOTableFname = FileNameJoin[{moduleDir, "data", "S0OandECSOTable.m"}];
5083 If[!FileExistsQ[S0OandECSOTableFname],
5084   (PrintTemporary[">> S0OandECSOTable.m not found, generating ..."]);
5085   S0OandECSOTable = GenerateS0OandECSOTable[7, "Export" -> True];
5086   ),
5087   S0OandECSOTable = Import[S0OandECSOTableFname];
5088 ];
5089 );
5090
5091 LoadT22::usage = "LoadT22[] loads into session the matrix elements
5092   of the double tensor operator T22.";
5093 LoadT22[] := (
5094   If[ValueQ[T22Table], Return[]];
5095   PrintTemporary["Loading the association of reduced T22 matrix
5096   elements ..."];
5097   T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.
5098   m"}];
5099   If[!FileExistsQ[T22TableFname],
5100     (PrintTemporary[">> ReducedT22Table.m not found, generating ..."]);
5101     T22Table = GenerateT22Table[7];
5102     ),
5103     T22Table = Import[T22TableFname];
5104   ];
5105 );
5106
5107 LoadSpinSpin::usage = "LoadSpinSpin[] loads into session the matrix
5108   elements of spin-spin.";
5109 LoadSpinSpin[] := (
5110   If[ValueQ[SpinSpinTable], Return[]];
5111   PrintTemporary["Loading the association of matrix elements for
5112   spin-spin ..."];
5113   SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.
5114   m"}];
5115   If[!FileExistsQ[SpinSpinTableFname],
5116     (PrintTemporary[">> SpinSpinTable.m not found, generating ..."]);
5117     SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
5118     ),
5119     SpinSpinTable = Import[SpinSpinTableFname];
5120   ];
5121 );
5122
5123 LoadThreeBody::usage = "LoadThreeBody[] loads into session the
5124   matrix elements of three-body configuration-interaction effects.";
5125 LoadThreeBody[] := (
5126   If[ValueQ[ThreeBodyTable], Return[]];
5127   PrintTemporary["Loading the association of matrix elements for
5128   three-body configuration-interaction effects ..."];
5129   ThreeBodyFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
5130   ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
5131   If[!FileExistsQ[ThreeBodyFname],
5132     (PrintTemporary[">> ThreeBodyTable.m not found, generating ..."]);
5133     {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTables["
5134     Export" -> True];
5135     ),
5136     ThreeBodyTable = Import[ThreeBodyFname];
5137     ThreeBodyTables = Import[ThreeBodiesFname];
5138   ];
5139 );
5140
5141 (* ##### Load Functions ##### *)
5142 (* ##### *)
5143
5144 End[];
5145
5146 LoadTermLabels[];
5147 LoadCFP[];
5148
5149 EndPackage[];

```

18.2 fittings.m

This file has code useful for fitting the Hamiltonian.

```

1 (*-----+
2 |-----+-----+
3 |-----+-----+
4 |-----+-----+
5 |-----+-----+
6 |-----+-----+
7 |-----+-----+
8 |-----+-----+
9 |-----+-----+
10|-----+
11|-----+-----+
12|-----+
13|-----+
14|-----+
15|-----+-----+
16|-----+
17|-----+
18|-----+-----+
19|-----+-----+
20|-----+
21|-----+
22|-----+-----+
23|-----+
24|-----+
25+-----+-----+*)
26
27 BeginPackage["DavidLizarazo`fittings`"];
28
29 Get[FileNameJoin[{DirectoryName[$InputFileName], "qlanth.m"}]];
30 Get[FileNameJoin[{DirectoryName[$InputFileName], "qonstants.m"}]];
31 Get[FileNameJoin[{DirectoryName[$InputFileName], "misc.m"}]];
32
33 LoadCarnall[];
34 LoadFreeIon[];
35
36 caseConstraints::usage="This Association contains the constraints
   that are not the same across all of the lanthanides. For instance,
   since the ratio between M2 and M0 is assumed the same for all the
   trivalent lanthanides, that one is not included here.
37 This association has keys equal to symbols of lanthanides and values
   equal to lists of rules that express either a parameter being held
   fixed or made proportional to another.
38 In Table I of Carnall 1989 these correspond to cases were values are
   given in square brackets.";
39 caseConstraints = <|
40 "Ce" -> {
41   B02 -> -218.,
42   B04 -> 738.,
43   B06 -> 679.,
44   B22 -> -50.,
45   B24 -> 431.,
46   B26 -> -921.,
47   B44 -> 616.,
48   B46 -> -348.,
49   B66 -> -788.
50 },
51 "Pr" -> {},
52 "Nd" -> {},
53 "Sm" -> {
54   B22 -> -50.,
55   T2 -> 300.,
56   T3 -> 36.,
57   T4 -> 56.,
58   γ -> 1500.
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      $\gamma$  -> 1800.  

138     },  

139 "Er" -> {  

140     T2 -> 400.,  

141      $\gamma$  -> 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
211     symbols for which the constraint doesn't apply to all ions.
212 This association has keys equal to symbols of lanthanides and values
213     equal to associations with the symbols for the parameters that are
214     held fixed or made proportional to another. If the value is to be
215     held constant a placeholder value of 0 is given, if a ratio
216     constraint is given, the value is constraint.
217 The operator basis for which this is applicable is the mostly-
218     orthogonal basis.
219 ";

```

```

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      $\beta$ p -> 0,
299      $\gamma$ p -> 0
300     },
301 "Tb" -> {
302     E2p -> 0.0049 E1p,
303     T2p -> 0,
304     T3 -> 0,
305     T4 -> 0,
306      $\gamma$ p -> 0
307     },
308 "Dy" -> {},
309 "Ho" -> {
310     B02 -> 0,
311     T2p -> 0,
312      $\gamma$ p -> 0
313     },
314 "Er" -> {
315     T2p -> 0,
316      $\gamma$ p -> 0
317     },
318 "Tm" -> {
319      $\gamma$ p -> 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" -> { $\zeta$ },
336     "Pr" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
337         E1p, E2p, E3p,
338         M0, P2,
339          $\alpha$ p,  $\beta$ p,  $\gamma$ p,
340          $\zeta$ },
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          $\alpha$ p,  $\beta$ p,  $\gamma$ p,
346          $\zeta$ },
347     "Pm" -> {},
348     "Sm" -> {B02, B04, B06, B24, B26, B44, B46, B66,
349         E1p, E2p, E3p, M0, P2,
350         T6, T7, T8,
351          $\alpha$ p,  $\beta$ p,  $\zeta$ },
352     "Eu" -> {B02, B04, B06,
353         E1p, E2p, E3p,  $\zeta$ },
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       $\alpha \rightarrow 17.9$ ,
442       $\beta \rightarrow -628.$ ,
443       $\gamma \rightarrow 1790.$ ,
444       $T_2 \rightarrow 326.$ ,
445       $T_3 \rightarrow 23.$ ,
446       $T_4 \rightarrow 83.$ ,
447       $T_6 \rightarrow -294.$ ,
448       $T_7 \rightarrow 403.$ ,
449       $T_8 \rightarrow 340.$ ,
450       $M_0 \rightarrow 4.46$ ,
451       $P_2 \rightarrow 610.$ ,
452       $B_{46} \rightarrow -700.$ 
453    },
454  "Ho" -> {
455     $\alpha \rightarrow 17.2$ ,
456     $\beta \rightarrow -596.$ ,
457     $\gamma \rightarrow 1839.$ ,
458     $T_2 \rightarrow 365.$ ,
459     $T_3 \rightarrow 37.$ ,
460     $T_4 \rightarrow 95.$ ,
461     $T_6 \rightarrow -274.$ ,
462     $T_7 \rightarrow 331.$ ,
463     $T_8 \rightarrow 343.$ ,
464     $P_2 \rightarrow 582.$ 
465  },
466 "Er" -> {},
467 "Tm" -> {
468     $\alpha \rightarrow 17.3$ ,
469     $\beta \rightarrow -665.$ ,
470     $\gamma \rightarrow 1936.$ ,
471     $M_0 \rightarrow 4.93$ ,
472     $P_2 \rightarrow 730.$ ,
473     $T_2 \rightarrow 400.$ 
474  },
475 "Yb" -> {
476     $B_{06} \rightarrow -23.$ ,
477     $B_{46} \rightarrow -512.$ 
478  }
479 |>;
480
481 variedSymbolsLiYF4 = <|
482  "Ce" -> {
483     $B_{02}$ ,  $\zeta$ 
484  },
485  "Pr" -> {
486     $B_{02}$ ,  $B_{04}$ ,  $B_{06}$ ,  $B_{44}$ ,  $B_{46}$ ,
487     $F_2$ ,  $F_4$ ,  $F_6$ ,
488     $\alpha$ ,  $\zeta$ 
489  },
490  "Nd" -> {
491     $B_{02}$ ,  $B_{04}$ ,  $B_{06}$ ,  $B_{44}$ ,  $B_{46}$ ,
492     $F_2$ ,  $F_4$ ,  $F_6$ ,
493     $P_2$ ,
494     $T_2$ ,  $T_3$ ,  $T_4$ ,  $T_6$ ,  $T_7$ ,  $T_8$ ,
495     $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\zeta$ 
496  },
497  "Sm" -> {
498     $B_{02}$ ,  $B_{04}$ ,  $B_{06}$ ,  $B_{44}$ ,  $B_{46}$ ,
499     $F_2$ ,  $F_4$ ,  $F_6$ ,
500     $\zeta$ 
501  },
502  "Eu" -> {
503     $B_{02}$ ,  $B_{04}$ ,  $B_{06}$ ,  $B_{44}$ ,  $B_{46}$ ,
504     $F_2$ ,  $F_4$ ,  $F_6$ ,
505     $M_0$ ,  $P_2$ ,
506     $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\zeta$ 
507  },
508  "Tb" -> {
509     $B_{02}$ ,  $B_{04}$ ,  $B_{06}$ ,  $B_{44}$ ,  $B_{46}$ ,
510     $F_2$ ,  $F_4$ ,  $F_6$ ,
511     $\zeta$ 
512  },
513  "Dy" -> {
514     $B_{02}$ ,  $B_{04}$ ,  $B_{06}$ ,  $B_{44}$ ,
515     $F_2$ ,  $F_4$ ,  $F_6$ ,
516     $\zeta$ 

```

```

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   },
584   "Eu" -> {
585     F2 -> 82573., F4 -> 59646., F6 -> 43203.,
586     ζ -> 1329.,
587     α -> 21.6, β -> -482., γ -> 1140.,
588     T2 -> 370., T3 -> 40., T4 -> 40., T6 -> -300., T7 -> 380., T8 ->
589     370.,
590     M0 -> 2.41, P2 -> 332.,
591     B02 -> 339., B04 -> -733.,
592     B44 -> -1067., B06 -> -36.,
593   }
594 |>;

```

```

588      B46 -> -764.
589      },
590 "Tb" -> {
591     F2 -> 90972., F4 -> 64499., F6 -> 45759.,
592      $\zeta$  -> 1702.,
593      $\alpha$  -> 17.6,  $\beta$  -> -581.,  $\gamma$  -> 1792.,
594     T2 -> 330., T3 -> 40., T4 -> 45., T6 -> -365., T7 -> 320., T8 ->
595     349.,
596     M0 -> 2.7, P2 -> 482.,
597     B02 -> 413., B04 -> -867.,
598     B44 -> -1114., B06 -> -41.,
599     B46 -> -736.
600   },
601 "Dy" -> {
602     F0 -> 0,
603     F2 -> 90421., F4 -> 63928., F6 -> 46657.,
604      $\zeta$  -> 1895.,
605      $\alpha$  -> 17.9,  $\beta$  -> -628.,  $\gamma$  -> 1790.,
606     T2 -> 326., T3 -> 23., T4 -> 83., T6 -> -294., T7 -> 403., T8 ->
607     340.,
608     M0 -> 4.46, P2 -> 610.,
609     B02 -> 360., B04 -> -737.,
610     B44 -> -943., B06 -> -35.,
611     B46 -> -700.
612   },
613 "Ho" -> {
614     F2 -> 93512., F4 -> 66084., F6 -> 49765.,
615      $\zeta$  -> 2126.,
616      $\alpha$  -> 17.2,  $\beta$  -> -596.,  $\gamma$  -> 1839.,
617     T2 -> 365., T3 -> 37., T4 -> 95., T6 -> -274., T7 -> 331., T8 ->
618     343.,
619     M0 -> 3.92, P2 -> 582.,
620     B02 -> 386., B04 -> -629.,
621     B44 -> -841., B06 -> -33.,
622     B46 -> -687.
623   },
624 "Er" -> {
625     F2 -> 97326., F4 -> 67987., F6 -> 53651.,
626      $\zeta$  -> 2377.,
627      $\alpha$  -> 18.1,  $\beta$  -> -599.,  $\gamma$  -> 1870.,
628     T2 -> 380., T3 -> 41., T4 -> 69., T6 -> -356., T7 -> 239., T8 ->
629     390.,
630     M0 -> 4.41, P2 -> 795.,
631     B02 -> 325., B04 -> -749.,
632     B44 -> -1014., B06 -> -19.,
633     B46 -> -635.
634   },
635 "Tm" -> {
636     F0 -> 0.,
637     T2 -> 0.,
638     F2 -> 101938., F4 -> 71553., F6 -> 51359.,
639      $\zeta$  -> 2632.,
640      $\alpha$  -> 17.3,  $\beta$  -> -665.,  $\gamma$  -> 1936.,
641     M0 -> 4.93, P2 -> 730.,
642     B02 -> 339., B04 -> -627.,
643     B44 -> -913., B06 -> -39.,
644     B46 -> -584.
645   },
646 "Yb" -> {
647      $\zeta$  -> 2916.,
648     B02 -> 446., B04 -> -560.,
649     B44 -> -843., B06 -> -23.,
650     B46 -> -512.
651   }
652 |>;
653 Jiggle::usage = "Jiggle[num, wiggleRoom] takes a number and
654     randomizes it a little by adding or subtracting a random fraction
655     of itself. The fraction is controlled by wiggleRoom.";
656 Jiggle[num_, wiggleRoom_ : 0.1] := RandomReal[{1 - wiggleRoom, 1 +
657     wiggleRoom}] * num;
658 AddToList::usage = "AddToList[list, element, maxSize, addOnlyNew]
659     prepends the element to list and returns the list. If maxSize is
660     reached, the last element is dropped. If addOnlyNew is True (the
661     default), the element is only added if it is different from the

```

```

    last element.";
654 AddToList[list_, element_, maxSize_, addOnlyNew_ : True] := Module[{ 
655   tempList = If[ 
656     addOnlyNew, 
657     If[ 
658       Length[list] == 0, 
659       {element}, 
660       If[ 
661         element != list[[-1]], 
662         Append[list, element], 
663         list 
664       ] 
665     ], 
666     Append[list, element] 
667   ], 
668   If[Length[tempList] > maxSize, 
669     Drop[tempList, Length[tempList] - maxSize], 
670     tempList] 
671 ];
672
673 ProgressNotebook::usage="ProgressNotebook[] creates a progress
notebook for the solver. This notebook includes a plot of the RMS
history and the current parameter values. The notebook is returned
. The RMS history and the parameter values are updated by setting
the variables rmsHistory and paramSols. The variables
stringPartialVars and paramSols are used to display the parameter
values in the notebook.";
674 Options[ProgressNotebook] = {
675   "Basic" -> True,
676   "UpdateInterval" -> 0.5};
677 ProgressNotebook[OptionsPattern[]] := (
678   nb = Which[
679     OptionValue["Basic"],
680     CreateDocument[(
681       {
682         Dynamic[
683           TextCell[
684             If[ 
685               Length[paramSols] > 0,
686               TableForm[ 
687                 Prepend[ 
688                   Transpose[{stringPartialVars,
689                     paramSols[[-1]]}],
690                   {"RMS", rmsHistory[[-1]]}]
691                 ],
692               ""
693             ],
694             "Output"
695           ],
696             TrackedSymbols :> {paramSols, stringPartialVars},
697             UpdateInterval -> OptionValue["UpdateInterval"]
698           ]
699       }
700     ),
701     WindowSize -> {600, 1000},
702     WindowSelected -> True,
703     TextAlignment -> Center,
704     WindowTitle -> "Solver Progress"
705   ],
706   True,
707   CreateDocument[(
708     {
709       "",
710       Dynamic[Framed[progressMessage],
711         UpdateInterval -> OptionValue["UpdateInterval"]],
712       Dynamic[
713         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       ]
725     ]
726   ]
727 );
728
729 
```

```

722     ],
723     ListPlot[(#/#[[1]]) & /@ Transpose[paramSols],
724     Joined -> True,
725     PlotRange -> {All, {-5, 5}},
726     Frame -> True,
727     ImageSize -> 800,
728     AspectRatio -> 1,
729     FrameStyle -> Directive[Thick, 15],
730     FrameLabel -> {"Iteration", "Params"}]
731   ]
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']:= (
{eigenEnergies, eigenVecs} = Eigensystem[compHam['vars']];
ordering = Ordering[eigenEnergies];
eigenEnergies = eigenEnergies - Min[eigenEnergies];
states = Transpose[{eigenEnergies, eigenVecs}];
states = states[[ordering]];
coarseStates = ParseStates[states, basis];
coarseStates = {#[[1]], #[[-1]]}& /@ coarseStates;
(* The eigenvectors need to be simplified in order to compare
labels to labels *)
missingLevels = Length[coarseStates]-Length[expData];
(* The energies are in the first element of the tuples. *)
energyDiffFun = (Abs[#1[[1]]-#2[[1]]])&;
(* match disregarding labels *)
energyFlow = FlowMatching[coarseStates,
expData,
\"notMatched\" -> missingLevels,
\"CostFun\" -> energyDiffFun
];
energyPairs = {#[[1]][[1]], #[[2]][[1]]} & /@ energyFlow[[1]];
energyRms = Sqrt[Total[(Abs[#[[2]]-#[[1]]])^2 & /@ energyPairs]
/ Length[energyPairs]];
Return[energyRms];
)"];
783
784 AppendToLog[message_, file_String] := Module[
785   {timestamp = DateString["ISODateTime"], msgString},
786   (
787     msgString = ToString[message, InputForm]; (* Convert any
expression to a string *)
788     OpenAppend[file];

```

```

789     WriteString[file, timestamp, " - ", msgString, "\n"];
790     Close[file];
791   );
792 ];
793
794 energyAndLabelCostFunTemplate::usage="energyAndLabelCostFunTemplate
795 is a template used to define the cost function that includes both
796 the differences between energies and the differences between
797 labels. The template is used to define a function
798 TheRightSignedPath that takes a list of variables and returns the
799 RMS of the energy differences between the computed and the
800 experimental energies together with a term that depends on the
801 differences between the labels. The template requires the values
802 to the following keys to be provided: 'vars' and 'varPatterns'";
803
804 energyAndLabelCostFunTemplate = StringTemplate["
805 TheRightSignedPath['varPatterns'] := Module[
806   {energyRms, eigenEnergies, eigenVecs, ordering, states,
807    coarseStates, missingLevels, energyDiffFun, energyFlow,
808    energyPairs, energyAndLabelFun, energyAndLabelFlow, totalAvgCost},
809   (
810     {eigenEnergies, eigenVecs} = Eigensystem[compHam['vars']];
811     ordering      = Ordering[eigenEnergies];
812     eigenEnergies = eigenEnergies - Min[eigenEnergies];
813     states        = Transpose[{eigenEnergies, eigenVecs}];
814     states        = states[[ordering]];
815     coarseStates = ParseStates[states, basis];
816
817     (* The eigenvectors need to be simplified in order to compare
818     labels to labels *)
819     coarseStates = {#[[1]], #[[-1]]} & /@ coarseStates;
820     missingLevels = Length[coarseStates] - Length[expData];
821
822     (* The energies are in the first element of the tuples. *)
823     energyDiffFun = ( Abs[#1[[1]] - #2[[1]]] ) &;
824
825     (* matching disregarding labels to get overall scale for scaling
826     differences in labels *)
827     energyFlow      = FlowMatching[coarseStates,
828                                   expData,
829                                   \"notMatched\" -> missingLevels,
830                                   \"CostFun\"      -> energyDiffFun
831                                   ];
832     energyPairs     = {#[[1]][[1]], #[[2]][[1]]} & /@ energyFlow[[1]];
833     energyRms       = Sqrt[Total[(Abs[#[[2]] - #[[1]]])^2] & /@
834     energyPairs]/Length[energyPairs];
835
836     (* matching using both labels and energies *)
837     energyAndLabelFun = With[{del=energyRms},
838       (Abs[#1[[1]] - #2[[1]]] +
839        If[#1[[2]] == #2[[2]],
840            0.,
841            del]) &];
842
843     (* energyAndLabelFun = With[{del=energyRms},
844       (Abs[#1[[1]] - #2[[1]]] +
845        del*EditDistance[#1[[2]], #2[[2]]]) &]; *)
846     energyAndLabelFun = ( Abs[#1[[1]] - #2[[1]]] + EditDistance
847     [[#1[[2]], #2[[2]]]] ) &;
848     energyAndLabelFlow = FlowMatching[coarseStates,
849                               expData,
850                               \"notMatched\" -> missingLevels,
851                               \"CostFun\"      -> energyAndLabelFun
852                               ];
853     totalAvgCost      = Total[energyAndLabelFun @@ # & /@
854     energyAndLabelFlow[[1]]]/Length[energyAndLabelFlow[[1]]];
855     Return[totalAvgCost];
856   )
857 ];
858
859 Constrainer::usage = "Constrainer[problemVars, ln] returns a list of
860 constraints for the variables in problemVars for trivalent
861 lanthanide ion ln. problemVars are standard model symbols (F2, F4,
862 ...). The ranges returned are based in the fitted parameters for
863 LaF3 as found in Carnall et al. They could probably be more fine
864 grained, but these ranges are seen to describe all the ions in
865 that case.";
```

```

844 Constrainer[problemVars_, ln_] := (
845   slater = Which[
846     MemberQ[{"Ce", "Yb"}, ln],
847     {},
848     True,
849     {#, (20000. < # < 120000.)} & /@ {F2, F4, F6}
850   ];
851   alpha = Which[
852     MemberQ[{"Ce", "Yb"}, ln],
853     {},
854     True,
855     {{α, 14. < α < 22.}}
856   ];
857   zeta = {{ζ, 500. < ζ < 3200.}};
858   beta = Which[
859     MemberQ[{"Ce", "Yb"}, ln],
860     {},
861     True,
862     {{β, -1000. < β < -400.}}
863   ];
864   gamma = Which[
865     MemberQ[{"Ce", "Yb"}, ln],
866     {},
867     True,
868     {{γ, 1000. < γ < 2000.}}
869   ];
870   tees = Which[
871     ln == "Tm",
872     {100. < T2 < 500.},
873     MemberQ[{"Ce", "Pr", "Yb"}, ln],
874     {},
875     True,
876     {#, -500. < # < 500.} & /@ {T2, T3, T4, T6, T7, T8}];
877   marvins = Which[
878     MemberQ[{"Ce", "Yb"}, ln],
879     {},
880     True,
881     {{M0, 1.0 < M0 < 5.0}}
882   ];
883   peas = Which[
884     MemberQ[{"Ce", "Yb"}, ln],
885     {},
886     True,
887     {{P2, -200. < P2 < 1200.}}
888   ];
889   crystalRanges = {#, (-2000. < # < 2000.)} & /@ (Intersection[
890     cfSymbols, problemVars]);
891   allCons =
892     Join[slater, zeta, alpha, beta, gamma, tees, marvins, peas,
893       crystalRanges];
894   allCons = Select[allCons, MemberQ[problemVars, #[[1]]] &];
895   Return[Flatten[Rest /@ allCons]]
896   );
897
898 Options[LogSol] = {"PrintFun" -> PrintTemporary};
899 LogSol::usage = "LogSol[expr, prefix] saves the given expression to a
  file. The file is named with the given prefix and a created UUID.
  The file is saved in the \\"log\\" directory under the current
  directory. The file is saved in the format of a .m file. The
  function returns the name of the file.";
900 LogSol[theSolution_, prefix_, OptionsPattern[]] :=
901   PrintFun = OptionValue["PrintFun"];
902   fname = prefix <> "-sols-" <> CreateUUID[] <> ".m";
903   fname = FileNameJoin[{".", "log", fname}];
904   PrintFun["Saving solution to: ", fname];
905   Export[fname, theSolution];
906   Return[fname];
907   );
908
909 ClassicalFit::usage="ClassicalFit[numE, expData, excludeDataIndices,
  problemVars, startValues, constraints] fits the given expData in
  an f^numE configuration, by using the symbols in problemVars. The
  symbols given in problemVars may be constrained or held constant,
  this being controlled by constraints list which is a list of
  replacement rules expressing desired constraints. The constraints
  list additional constraints imposed upon the model parameters that

```

```

    remain once other simplifications have been \"baked\" into the
    compiled Hamiltonians that are used to increase the speed of the
    calculation.

910
911 Important, note that in the case of odd number of electrons the given
    data must explicitly include the Kramers degeneracy;
    excludeDataIndices must be compatible with this.

912
913 The list expData needs to be a list of lists with the only
    restriction that the first element of them corresponds to energies
    of levels. In this list, an empty value can be used to indicate
    known gaps in the data. Even if the energy value for a level is
    known (and given in expData) certain values can be omitted from
    the fitting procedure through the list excludeDataIndices, which
    correspond to indices in expData that should be skipped over.

914
915 The Hamiltonian used for fitting is a version that has been truncated
    either by using the maximum energy given in expData or by
    manually setting a truncation energy using the option \"
    TruncationEnergy\".

916
917 The option \"Experimental Uncertainty in K\" is the estimated
    uncertainty in the differences between the calculated and the
    experimental energy levels. This is used to estimate the
    uncertainty in the fitted parameters. Admittedly this will be a
    rough estimate (at least on the contribution of the calculated
    uncertainty), but it is better than nothing and may at least
    provide a lower bound to the uncertainty in the fitted parameters.
    It is assumed that the uncertainty in the differences between the
    calculated and the experimental energy levels is the same for all
    of them.

918
919 The list startValues is a list with all of the parameters needed to
    define the Hamiltonian (including the initial values for
    problemVars).

920
921 The function saves the solution to a file. The file is named with a
    prefix (controlled by the option \"FilePrefix\") and a UUID. The
    file is saved in the log sub-directory as a .m file.

922
923 Here's a description of the different parts of this function: first
    the Hamiltonian is assembled and simplified using the given
    simplifications. Then the intermediate coupling basis is
    calculated using the free-ion parameters for the given lanthanide.
    The Hamiltonian is then changed to the intermediate coupling
    basis and truncated. The truncated Hamiltonian is then compiled
    into a function that can be used to calculate the energy levels of
    the truncated Hamiltonian. The function that calculates the
    energy levels is then used to fit the experimental data. The
    fitting is done using FindMinimum with the Levenberg-Marquardt
    method.

924
925 The function returns an association with the following keys:
926
927 - \"bestRMS\" which is the best \[Sigma] value found.
928 - \"bestParams\" which is the best set of parameters found for the
    variables that were not constrained.
929 - \"bestParamsWithConstraints\" which has the best set of parameters
    (from - \"bestParams\") together with the used constraints. These
    include all the parameters in the model, even those that were not
    fitted for.
930 - \"paramSols\" which is a list of the parameters trajectories during
    the stepping of the fitting algorithm.
931 - \"timeTaken/s\" which is the time taken to find the best fit.
932 - \"simplifier\" which is the simplifier used to simplify the
    Hamiltonian.
933 - \"excludeDataIndices\" as given in the input.
934 - \"startValues\" as given in the input.

935
936 - \"freeIonSymbols\" which are the symbols used in the intermediate
    coupling basis.
937 - \"truncationEnergy\" which is the energy used to truncate the
    Hamiltonian, if it was set to Automatic, the value here is the
    actual energy used.
938 - \"numE\" which is the number of electrons in the f^numE
    configuration.

```

```

939 - \\"expData\\\" which is the experimental data used for fitting.
940 - \\"problemVars\\\" which are the symbols considered for fitting
941
942 - \\"maxIterations\\\" which is the maximum number of iterations used by
  NMinimize.
943 - \\"hamDim\\\" which is the dimension of the full Hamiltonian.
944 - \\"allVars\\\" which are all the symbols defining the Hamiltonian
  under the aggregate simplifications.
945 - \\"freeBies\\\" which are the free-ion parameters used to define the
  intermediate coupling basis.
946 - \\"truncatedDim\\\" which is the dimension of the truncated
  Hamiltonian.
947 - \\"compiledIntermediateFname\\\" the file name of the compiled
  function used for the truncated Hamiltonian.
948
949 - \\"fittedLevels\\\" which is the number of levels fitted for.
950 - \\"actualSteps\\\" the number of steps that FindMininum actually
  took.
951 - \\"solWithUncertainty\\\" which is a list of replacement rules of the
  form (paramSymbol -> {bestEstimate, uncertainty}).
952 - \\"rmsHistory\\\" which is a list of the \[Sigma] values found during
  the fitting.
953 - \\"Appendix\\\" which is an association appended to the log file under
  the key \\"Appendix\\\".
954 - \\"presentDataIndices\\\" which is the list of indices in expData that
  were used for fitting, this takes into account both the empty
  indices in expData and also the indices in excludeDataIndices.
955
956 - \\"states\\\" which contains a list of eigenvalues and eigenvectors
  for the fitted model, this is only available if the option \"
  SaveEigenvectors\\\" is set to True; if a general shift of energy
  was allowed for in the fitting, then the energies are shifted
  accordingly.
957 - \\"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.
958 - \\"degreesOfFreedom\\\" which is equal to the number of fitted state
  energies minus the number of parameters used in fitting.
959
960 The function admits the following options with corresponding default
961 values:
962 - \\"MaxHistory\\\" : determines how long the logs for the solver can be
  .
963 - \\"MaxIterations\\\" : determines the maximum number of iterations used
  by NMinimize.
964 - \\"FilePrefix\\\" : the prefix to use for the subfolder in the log
  filder, in which the solution files are saved, by default this is
  \\"calcs\\\" so that the calculation files are saved under the
  directory \\"log/calcs\\\".
965 - \\"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.
966 - \\"AccuracyGoal\\\" : the accuracy goal used by NMinimize, default of
  5.
967 - \\"TrucationEnergy\\\" : if Automatic then the maximum energy in
  expData is taken, else it takes the value set by this option. In
  all cases the energies in expData are only considered up to this
  value.
968 - \\"PrintFun\\\" : the function used to print progress messages, the
  default is PrintTemporary.
969 - \\"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 \u03b6. 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

```

```

    to show the progress of the solver, the default is True.
972 - \\"SignatureCheck\\": if True then then the function returns
      prematurely, returning a list with the symbols that would have
      defined the Hamiltonian after all simplifications have been
      applied. Useful to check the entire parameter set that the
      Hamiltonian has, which has to match one-to-one what is provided by
      startValues.
973 - \\"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.
974
975 - \\"AppendToFile\\": an association appended to the log file under
      the key \\\"Appendix\\\".
976 - \\"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.
977 - \\"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.
978
979 - \\"SymmetrySimplifier\\": a list of replacements rules to simplify
      the crystal field.
980 - \\"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.
981 ";
982 Options[ClassicalFit] = {
983   "MaxHistory"      -> 200,
984   "MaxIterations"   -> 100,
985   "FilePrefix"      -> "calcs",
986   "ProgressView"    -> True,
987   "TruncationEnergy" -> Automatic,
988   "AccuracyGoal"    -> 5,
989   "PrintFun"        -> PrintTemporary,
990   "RefParamsVintage" -> "LaF3",
991   "SignatureCheck"  -> False,
992   "AddConstantShift" -> False,
993   "SaveEigenvectors" -> False,
994   "AppendToFile"    -> <||>,
995   "SaveToLog"       -> False,
996   "Energy Uncertainty in K" -> Automatic,
997   "MagneticSimplifier" -> {
998     M2 -> 56/100 MO,
999     M4 -> 31/100 MO,
1000    P4 -> 1/2 P2,
1001    P6 -> 1/10 P2
1002  },
1003   "MagFieldSimplifier" -> {
1004     Bx -> 0,
1005     By -> 0,
1006     Bz -> 0
1007  },
1008   "SymmetrySimplifier" -> {
1009     B12->0, B14->0, B16->0, B34->0, B36->0, B56->0,
1010     S12->0, S14->0, S16->0, S22->0, S24->0, S26->0,
1011     S34->0, S36->0, S44->0, S46->0, S56->0, S66->0
1012  },
1013   "OtherSimplifier" -> {
1014     F0->0,
1015     P0->0,
1016     \\[Sigma]SS->0,
1017     T11p->0, T12->0, T14->0, T15->0,
1018     T16->0, T18->0, T17->0, T19->0, T2p->0,
1019     wChErrA ->0, wChErrB->0
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,

```

```

1028     T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1029     ->0,
1030     T2p->0
1031     },
1032     3 -> {},
1033     4 -> {},
1034     5 -> {},
1035     6 -> {},
1036     7 -> {},
1037     8 -> {},
1038     9 -> {},
1039     10 -> {},
1040     11 -> {},
1041     12 -> {
1042       T3->0, T4->0, T6->0, T7->0, T8->0,
1043       T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1044     ->0,
1045     T2p->0
1046     },
1047   13->{
1048     T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
1049     T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1050   ->0,
1051     T2p->0
1052   }
1053 |>,
1054 "FreeIonSymbols" -> {F0, F2, F4, F6,  $\zeta$ }
1055 };
1056 ClassicalFit[numE_Integer, expData_List, excludeDataIndices_List,
1057 problemVars_List, startValues_Association, constraints_List,
1058 OptionsPattern[]]:=Module[
1059 {
1060   accuracyGoal,
1061   allFreeEnergies,
1062   allFreeEnergiesSorted, allVars, allVarsVec,
1063   argsForEvalInsideOfTheIntermediateSystems,
1064   argsOfTheIntermediateEigensystems, aVar, aVarPosition,
1065   basis, basisChanger, basisChangerBlocks,
1066   bestParams, bestRMS, blockShifts, blockSizes,
1067   compiledDiagonal, compiledIntermediateFname,
1068   constrainedProblemVars, constrainedProblemVarsList,
1069   currentRMS, degressOffFreedom, dependentVars,
1070   diagonalBlocks, diagonalScalarBlocks, diff,
1071   eigenEnergies, eigenvalueDispenserTemplate,
1072   eigenVectors, elevatedIntermediateEigensystems,
1073   endTime, fmSolAssoc, freeBies,
1074   freeIenergiesAndMultiplets, fullHam, fullSolve,
1075   ham, hamDim, hamEigenvaluesTemplate,
1076   hamString, indepSolveVec, indepVars, intermediateHam,
1077   isolationValues, lin, linMat, ln, lnParams,
1078   logFilePrefix, magneticSimplifier,
1079   maxFreeEnergy, maxHistory, maxIterations,
1080   minFreeEnergy, minpoly,
1081   modelSymbols, multipletAssignments, needlePosition,
1082   numBlocks, solCompendium,
1083   openNotebooks, ordering, otherSimplifier, p0,
1084   paramBest, perHam,
1085   presentDataIndices, PrintFun, problemVarsPositions,
1086   problemVarsQ, problemVarsQString, problemVarsVec,
1087   problemVarsWithStartValues, reducedModelSymbols,
1088   roundedTruncationEnergy,
1089   runningInteractive, shiftToggle, simplifier,
1090   sol, solWithUncertainty,
1091   sortedTruncationIndex, sqdiff, standardValues,
1092   startTime,
1093   states, steps, symmetrySimplifier,
1094   theIntermediateEigensystems, TheIntermediateEigensystems,
1095   TheTruncatedAndSignedPathGenerator, timeTaken,
1096   truncatedIntermediateBasis, truncatedIntermediateHam,
1097   truncationEnergy, truncationIndices, RefParams,
1098   truncationUmbral, varHash,
1099   varsWithConstants, \[Lambda]0Vec,
1100   \[Lambda]exp
1101 },
1102 ],
1103 \[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;
1163 solCompendium["expData"]         = expData;
1164 solCompendium["problemVars"]     = problemVars;
1165 solCompendium["maxIterations"]   = maxIterations;
1166 solCompendium["hamDim"]          = hamDim;
1167 solCompendium["constraints"]     = constraints;
1168
1169 modelSymbols = Sort[Select[paramSymbols, Not[MemberQ[Join[
1170 racahSymbols, juddOfeltIntensitySymbols, chenSymbols, {t2Switch, \
1171 Epsilon}, gs, nE}], #]]];

```

```

1166 (* remove the symbols that will be removed by the simplifier, no
1167 symbol should remain here that is not in the symbolic Hamiltonian
1168 *)
1169 reducedModelSymbols = Select[modelSymbols, Not[MemberQ[Keys[
1170 simplifier], #]] &];
1171
1172 (* this is useful to understand what are the arguments of the
1173 truncated compiled Hamiltonian *)
1174 If[OptionValue["SignatureCheck"],
1175 (
1176 PrintFun["Given the model parameters and the simplifying
assumptions, the resultant model parameters are:"];
1177 PrintFun[{reducedModelSymbols}];
1178 PrintFun["Exiting ..."];
1179 Return[""];
1180 )
1181 ];
1182
1183 (* calculate the basis *)
1184 PrintFun["Retrieving the LSJMJ basis for f^", numE, " ..."];
1185 basis = BasisLSJMJ[numE];
1186
1187 Which[refParamsVintage === Automatic,
1188 (
1189 PrintFun["Using the automatic vintage with freshly fitted
free-ion parameters and others as in LaF3 ..."];
1190 lnParams = LoadLaF3Parameters[ln];
1191 freeIonSol = FreeIonSolver[expData, numE];
1192 freeIonParams = freeIonSol["bestParams"];
1193 lnParams = Join[lnParams, freeIonParams];
1194 ),
1195 MemberQ[{List, Association}, Head[RefParams]],
1196 (
1197 RefParams = Association[RefParams];
1198 PrintFun["Using the given parameters as a starting point ..."]
1199 ];
1200 lnParams = RefParams;
1201 extraParams = LoadLaF3Parameters[ln];
1202 lnParams = Join[extraParams, lnParams];
1203 ),
1204 True,
1205 (
1206 (* get the reference parameters from the given vintage *)
1207 PrintFun["Getting reference free-ion parameters for ", ln, "
using ", refParamsVintage, " ..."];
1208 lnParams = ParamPad[RefParams[ln], "PrintFun" -> PrintFun];
1209 )
1210 ];
1211 freeBies = Prepend[Values[(# -> (# /. lnParams)) &/@ freeIonSymbols], numE];
1212 (* a more explicit alias *)
1213 allVars = reducedModelSymbols;
1214 numericConstraints = Association@Select[constraints, NumericQ
1215 #[[2]]] &;
1216 standardValues = allVars /. Join[lnParams, numericConstraints];
1217 solCompendium["allVars"] = allVars;
1218 solCompendium["freeBies"] = freeBies;
1219
1220 (* reload compiled version if found *)
1221 varHash = Hash[{numE, allVars, freeBies,
truncationEnergy, simplifier}];
1222 compiledIntermediateFname = ln <> "-compiled-intermediate-
truncated-ham-" <> ToString[varHash] <> ".mx";
1223 compiledIntermediateFname = FileNameJoin[{moduleDir, "compiled",
compiledIntermediateFname}];
1224 solCompendium["compiledIntermediateFname"] =
1225 compiledIntermediateFname;
1226
1227 If[FileExistsQ[compiledIntermediateFname],
1228 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 ..."];
1229 PrintFun["Using : ", compiledIntermediateFname];
1230 {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =

```

```

1224 Import[compiledIntermediateFname], ,
1225 (
1226   If[truncationEnergy == Infinity,
1227     (
1228       ham = HamMatrixAssembly[numE, "ReturnInBlocks" -> False];
1229       theSimplifier = simplifier;
1230       ham = Normal@ReplaceInSparseArray[ham, simplifier];
1231       PrintFun["Compiling a function for the Hamiltonian with no
1232         truncation ..."];
1233         (* compile a function that will calculate the truncated
1234         Hamiltonian given the parameters in allVars, this is the function
1235         to be use in fitting *)
1236         compileIntermediateTruncatedHam = Compile[Evaluate[allVars
1237 ], Evaluate[ham]];
1238         truncatedIntermediateBasis = SparseArray@IdentityMatrix[
1239           Binomial[14, numE]];
1240           (* save the compiled function *)
1241           PrintFun["Saving the compiled function for the Hamiltonian
1242             with no truncation and a placeholder intermediate basis ..."];
1243             Export[compiledIntermediateFname, {
1244               compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
1245             ),
1246             (
1247               (* grab the Hamiltonian preserving the block structure *)
1248               PrintFun["Assembling the Hamiltonian for f^", numE, " keeping
1249                 the block structure ..."];
1250               ham = HamMatrixAssembly[numE, "ReturnInBlocks" -> True
1251 ];
1252               (* apply the simplifier *)
1253               PrintFun["Simplifying using the aggregate set of
1254                 simplification rules ..."];
1255               ham = Map[ReplaceInSparseArray[#, simplifier] &, ham,
1256 {2}];
1257               PrintFun["Zeroing out every symbol in the Hamiltonian that is
1258                 not a free-ion parameter ..."];
1259               (* Get the free ion symbols *)
1260               freeIonSimplifier = (# -> 0) & /@ Complement[
1261                 reducedModelSymbols, freeIonSymbols];
1262               (* Take the diagonal blocks for the intermediate analysis *)
1263               PrintFun["Grabbing the diagonal blocks of the Hamiltonian ...
1264 ];
1265               diagonalBlocks = Diagonal[ham];
1266               (* simplify them to only keep the free ion symbols *)
1267               PrintFun["Simplifying the diagonal blocks to only keep the
1268                 free ion symbols ..."];
1269               diagonalScalarBlocks = ReplaceInSparseArray[#, freeIonSimplifier] & /@ diagonalBlocks;
1270               (* these include the MJ quantum numbers, remove that *)
1271               PrintFun["Contracting the basis vectors by removing the MJ
1272                 quantum numbers from the diagonal blocks ..."];
1273               diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
1274
1275               argsOfTheIntermediateEigensystems = StringJoin[Riffle
1276 [Prepend[(ToString[#] <> "v_") & /@ freeIonSymbols, "numE_"], ", "]];
1277               argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle
1278 [(ToString[#] <> "v") & /@ freeIonSymbols, ", "]];
1279               PrintFun["argsOfTheIntermediateEigensystems = ",
1280                 argsOfTheIntermediateEigensystems];
1281               PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
1282                 argsForEvalInsideOfTheIntermediateSystems];
1283               PrintFun["(if the following fails, it might help to see if
1284                 the arguments of TheIntermediateEigensystems match the ones shown
1285                 above)"];
1286
1287               (* compile a function that will effectively calculate the
1288                 spectrum of all of the scalar blocks given the parameters of the
1289                 free-ion part of the Hamiltonian *)
1290               (* compile one function for each of the blocks *)
1291               PrintFun["Compiling functions for the diagonal blocks of the
1292                 Hamiltonian ..."];
1293               compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate
1294 [N[Normal[#]]] & /@ diagonalScalarBlocks;
1295               (* use that to create a function that will calculate the free
1296                 -ion eigensystem *)
1297               TheIntermediateEigensystems[numEv_, F0v_, F2v_, F4v_, F6v_, \[Zeta]
1298                 v_] := (

```

```

1270      theNumericBlocks = (#[F0v, F2v, F4v, F6v,  $\zeta$ v]&) /@  

1271      compiledDiagonal;  

1272      theIntermediateEigensystems = Eigensystem /@  

1273      theNumericBlocks;  

1274      Js = AllowedJ[numEv];  

1275      basisJ = BasisLSJMJ[numEv, "AsAssociation" -> True];  

1276      (* having calculated the eigensystems with the removed  

1277      degeneracies, put the degeneracies back in explicitly *)  

1278      elevatedIntermediateEigensystems = MapIndexed[EigenLever  

1279      [#1, 2Js[[#2[[1]]]]+1 ]&, theIntermediateEigensystems];  

1280      (* Identify a single MJ to keep *)  

1281      pivot = If[EvenQ[numEv], 0, -1/2];  

1282      LSJmultiplets = (#[[1]] <> ToString[InputForm[#[[2]]]]) &/  

1283      @Select[BasisLSJMJ[numEv], #[[{-1}] == pivot &];  

1284      (* calculate the multiplet assignments that the  

1285      intermediate basis eigenvectors have *)  

1286      needlePosition = 0;  

1287      multipletAssignments = Table[  

1288      (  

1289          J = Js[[idx]];  

1290          eigenVecs = theIntermediateEigensystems[[idx]][[2]];  

1291          majorComponentIndices = Ordering[Abs[#][[-1]]] &/  

1292          @eigenVecs;  

1293          majorComponentIndices += needlePosition;  

1294          needlePosition += Length[  

1295          majorComponentIndices];  

1296          majorComponentAssignments = LSJmultiplets[[#]] &/  

1297          @majorComponentIndices;  

1298          (* All of the degenerate eigenvectors belong to the  

1299          same multiplet*)  

1300          elevatedMultipletAssignments = ListRepeater[  

1301          majorComponentAssignments, 2J+1];  

1302          elevatedMultipletAssignments  

1303          ),  

1304          {idx, 1, Length[Js]}  

1305          ];  

1306          (* put together the multiplet assignments and the energies  

1307          *)  

1308          freeIenergiesAndMultiplets = Transpose /@ Transpose[{First /  

1309          @elevatedIntermediateEigensystems, multipletAssignments}];  

1310          freeIenergiesAndMultiplets = Flatten[  

1311          freeIenergiesAndMultiplets, 1];  

1312          (* calculate the change of basis matrix using the  

1313          intermediate coupling eigenvectors *)  

1314          basisChanger = BlockDiagonalMatrix[Transpose /@ Last /  

1315          @elevatedIntermediateEigensystems];  

1316          basisChanger = SparseArray[basisChanger];  

1317          Return[{theIntermediateEigensystems, multipletAssignments,  

1318          elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,  

1319          basisChanger}];  

1320      );  

1321  

1322      PrintFun["Calculating the intermediate eigensystems for ", ln,  

1323      " using free-ion params from LaF3 ..."];  

1324      (* calculate intermediate coupling basis using the free-ion  

1325      params for LaF3 *)  

1326      {theIntermediateEigensystems, multipletAssignments,  

1327      elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,  

1328      basisChanger} = TheIntermediateEigensystems @@ freeBies;  

1329  

1330      (* use that intermediate coupling basis to compile a function  

1331      for the full Hamiltonian *)  

1332      allFreeEnergies = Flatten[First /  

1333      @elevatedIntermediateEigensystems];  

1334      (* important that the intermediate coupling basis have  

1335      attached energies, which make possible the truncation *)  

1336      ordering = Ordering[allFreeEnergies];  

1337      (* sort the free ion energies and determine which indices  

1338      should be included in the truncation *)  

1339      allFreeEnergiesSorted = Sort[allFreeEnergies];  

1340      {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];  

1341      (* determine the index at which the energy is equal or larger  

1342      than the truncation energy *)  

1343      sortedTruncationIndex = Which[  

1344          truncationEnergy > (maxFreeEnergy - minFreeEnergy),  

1345          hamDim,

```

```

1319      True,
1320      FirstPosition[allFreeEnergiesSorted - Min[
1321      allFreeEnergiesSorted], x_ /; x > truncationEnergy, {0}, 1][[1]]
1321      ];
1322      (* the actual energy at which the truncation is made *)
1323      roundedTruncationEnergy = allFreeEnergiesSorted[[sortedTruncationIndex]];
1324
1325      (* the indices that participate in the truncation *)
1326      truncationIndices = ordering[;; sortedTruncationIndex];
1327      PrintFun["Computing the block structure of the change of
basis array ..."];
1328      blockSizes = BlockArrayDimensionsArray[ham];
1329      basisChangerBlocks = ArrayBlocker[basisChanger, blockSizes];
1330      blockShifts = First /@ Diagonal[blockSizes];
1331      numBlocks = Length[blockSizes];
1332      (* using the ham (with all the symbols) change the basis to
the computed one *)
1333      PrintFun["Changing the basis of the Hamiltonian to the
intermediate coupling basis ..."];
1334      intermediateHam = BlockMatrixMultiply[ham, basisChangerBlocks];
1335
1336      PrintFun["Distributing products inside of symbolic matrix
elements to keep complexity in check ..."];
1337      Do[
1338          intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
1339          {rowIdx, 1, numBlocks},
1340          {colIdx, 1, numBlocks}
1341      ];
1342      intermediateHam = BlockMatrixMultiply[BlockTranspose[
basisChangerBlocks], intermediateHam];
1343      PrintFun["Distributing products inside of symbolic matrix
elements to keep complexity in check ..."];
1344      Do[
1345          intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
1346          {rowIdx, 1, numBlocks},
1347          {colIdx, 1, numBlocks}
1348      ];
1349      (* using the truncation indices truncate that one *)
1350      PrintFun["Truncating the Hamiltonian ..."];
1351      truncatedIntermediateHam = TruncateBlockArray[intermediateHam,
truncationIndices, blockShifts];
1352      (* these are the basis vectors for the truncated hamiltonian
*)
1353      PrintFun["Saving the truncated intermediate basis ..."];
1354      truncatedIntermediateBasis = basisChanger[[All,
truncationIndices]];
1355
1356      PrintFun["Compiling a function for the truncated Hamiltonian
..."];
1357      (* compile a function that will calculate the truncated
Hamiltonian given the parameters in allVars, this is the function
to be use in fitting *)
1358      compileIntermediateTruncatedHam = Compile[Evaluate[allVars],
Evaluate[truncatedIntermediateHam]];
1359      (* save the compiled function *)
1360      PrintFun["Saving the compiled function for the truncated
Hamiltonian and the truncated intermediate basis ..."];
1361      Export[compiledIntermediateFname, {
1362          compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
1363
1364
1365
1366      truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
1367      PrintFun["The truncated Hamiltonian has a dimension of ",
truncationUmbral, "x", truncationUmbral, "..."];
1368      presentDataIndices = Select[presentDataIndices, # <=
truncationUmbral &];
1369      solCompendium["presentDataIndices"] = presentDataIndices;
1370
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") & /@ 
problemVars;
1378 problemVarsQString = StringJoin[Riffle[problemVarsQ, ", "]];
(* we also need to have the padded arguments with the variables
in the right position and the fixed values in the remaining ones
*)
1379 problemVarsPositions = Position[allVars, #][[1, 1]] & /@ 
problemVars;
1380 problemVarsString    = StringJoin[Riffle[ToString /@ problemVars,
", "]];
(* 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 *)
1381 varsWithConstants      = standardValues;
1382 varsWithConstants[[problemVarsPositions]] = problemVars;
1383 varsWithConstantsString = ToString[
varsWithConstants];
1384
1385 (* this following function serves eigenvalues from the
Hamiltonian, has memoization so it might grow to use a lot of RAM
*)
1386 Clear[HamSortedEigenvalues];
1387 hamEigenvaluesTemplate = StringTemplate["
HamSortedEigenvalues['problemVarsQ']:=(
1388   ham          = compileIntermediateTruncatedHam@@'
varsWithConstants';
1389   eigenValues = Chop[Sort@Eigenvalues@ham];
1390   eigenValues = eigenValues - Min[eigenValues];
1391   HamSortedEigenvalues['problemVarsString'] = eigenValues;
1392   Return[eigenValues]
1393 )"];
1394 hamString = hamEigenvaluesTemplate[<|
1395   "problemVarsQ"      -> problemVarsQString,
1396   "varsWithConstants" -> varsWithConstantsString,
1397   "problemVarsString" -> problemVarsString
1398 |>];
1399 ToExpression[hamString];
1400
1401 (* 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 *)
1402 eigenvalueDispenserTemplate = StringTemplate["
PartialHamEigenvalues['problemVarsQ'][i_]:=(
1403   eigenVals = HamSortedEigenvalues['problemVarsString'];
1404   eigenVals[[i]]
1405 )
1406 ];
1407 eigenValueDispenserString = eigenvalueDispenserTemplate[<|
1408   "problemVarsQ"      -> problemVarsQString,
1409   "problemVarsString" -> problemVarsString
1410 |>];
1411 ToExpression[eigenValueDispenserString];
1412
1413 PrintFun["Determining the free variables after constraints ..."];
1414 constrainedProblemVars      = (problemVars /. constraints);
1415 constrainedProblemVarsList = Variables[constrainedProblemVars];
1416 If[addShift,
1417   PrintFun["Adding a constant shift to the fitting parameters ...
1418 "];
1419   constrainedProblemVarsList = Append[constrainedProblemVarsList,
\[\Epsilon]];
1420 ];
1421
1422 indepVars = Complement[problemVars, #[[1]] & /@ constraints];
1423 stringPartialVars = ToString/@constrainedProblemVarsList;
1424
1425 paramSols  = {};
1426 rmsHistory = {};
1427 steps      = 0;
1428 problemVarsWithStartValues = KeyValueMap[{#1, #2} &, startValues];
1429 If[addShift,
1430   problemVarsWithStartValues = Append[problemVarsWithStartValues,

```

```

1432   {\[Epsilon] ,0}];
1433 ];
1434 openNotebooks = If[runningInteractive,
1435   ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks
1436 [] ,
1437 {}];
1438 If[Not[MemberQ[openNotebooks,"Solver Progress"]]&& OptionValue["ProgressView"],
1439   ProgressNotebook["Basic"]->False]
1440 ];
1441 degressOfFreedom = Length[presentDataIndices] - Length[
1442 problemVars] - 1;
1443 PrintFun["Fitting for ", Length[presentDataIndices], " data
1444 points with ", Length[problemVars], " free parameters.", " The
1445 effective degrees of freedom are ", degressOfFreedom, " ..."];
1446 PrintFun["Fitting model to data ..."];
1447 startTime = Now;
1448 shiftToggle = If[addShift, 1, 0];
1449 sol = FindMinimum[
1450   Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@
1451 constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
1452 {j, presentDataIndices}],
1453 problemVarsWithStartValues,
1454 Method -> "LevenbergMarquardt",
1455 MaxIterations -> OptionValue["MaxIterations"],
1456 AccuracyGoal -> OptionValue["AccuracyGoal"],
1457 StepMonitor :> (
1458   steps += 1;
1459   currentSqSum = Sum[(expData[[j]][[1]] - (
1460     PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle
1461 * \[Epsilon])^2, {j, presentDataIndices}];
1462   currentRMS = Sqrt[currentSqSum / degressOfFreedom];
1463   paramSols = AddToList[paramSols, constrainedProblemVarsList,
1464 maxHistory];
1465   rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
1466 )
1467 ];
1468 endTime = Now;
1469 timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
1470 PrintFun["Solution found in ", timeTaken, "s"];
1471
1472 solVec = constrainedProblemVars /. sol[[-1]];
1473 indepSolVec = indepVars /. sol[[-1]];
1474 If[addShift,
1475   \[Epsilon]Best = \[Epsilon]/. sol[[-1]],
1476   \[Epsilon]Best = 0
1477 ];
1478 fullSolVec = standardValues;
1479 fullSolVec[[problemVarsPositions]] = solVec;
1480 PrintFun["Calculating the truncated numerical Hamiltonian
1481 corresponding to the solution ..."];
1482 fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
1483 PrintFun["Calculating energies and eigenvectors ..."];
1484 {eigenEnergies, eigenVectors} = Eigensystem[fullHam];
1485 states = Transpose[{eigenEnergies, eigenVectors}];
1486 states = SortBy[states, First];
1487 eigenEnergies = First /@ states;
1488 PrintFun["Shifting energies to make ground state zero of energy
1489 ..."];
1490 eigenEnergies = eigenEnergies - eigenEnergies[[1]];
1491 PrintFun["Calculating the linear approximant to each eigenvalue
1492 ..."];
1493 allVarsVec = Transpose[{allVars}];
1494 p0 = Transpose[{fullSolVec}];
1495 linMat = {};
1496 If[addShift,
1497   tail = -2,
1498   tail = -1];
1499 Do[
1500   (
1501   aVarPosition = Position[allVars, aVar][[1, 1]];
1502   isolationValues = ConstantArray[0, Length[allVars]];
1503   isolationValues[[aVarPosition]] = 1;
1504   dependentVars = KeyValueMap[{#1, D[#2, aVar]} &, Association[
1505 constraints]];

```

```

1494      Do[
1495        isolationValues[[Position[allVars, dVar[[1]]][[1, 1]]]] = 
1496          dVar[[2]],
1497          {dVar, dependentVars}
1498        ];
1499        perHam = compileIntermediateTruncatedHam @@ isolationValues;
1500        lin = FirstOrderPerturbation[Last /@ states, perHam];
1501        linMat = Append[linMat, lin];
1502      },
1503      {aVar, constrainedProblemVarsList[[;;tail]]}
1504    ];
1505    PrintFun["Removing the gradient of the ground state ..."];
1506    linMat = (# - #[[1]] & /@ linMat);
1507    PrintFun["Transposing derivative matrices into columns ..."];
1508    linMat = Transpose[linMat];
1509
1510    PrintFun["Calculating the eigenvalue vector at solution ..."];
1511    \[Lambda]0Vec = Transpose[{eigenEnergies[[presentDataIndices]]}];
1512    PrintFun["Putting together the experimental vector ..."];
1513    \[Lambda]exp = Transpose[{First /@ expData[[presentDataIndices]]}];
1514
1515    problemVarsVec = If[addShift,
1516      Transpose[{constrainedProblemVarsList[[;;-2]]}],
1517      Transpose[{constrainedProblemVarsList}]];
1518
1519    indepSolVecVec = Transpose[{indepSolVec}];
1520    PrintFun["Calculating the difference between eigenvalues at
1521 solution ..."];
1522    diff = If[linMat == {},
1523      (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best,
1524      (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best + linMat[[presentDataIndices]].(problemVarsVec - indepSolVecVec)];
1525
1526    PrintFun["Calculating the sum of squares of differences around
1527 solution ... "];
1528    sqdiff = Expand[(Transpose[diff] . diff)[[1, 1]]];
1529    PrintFun["Calculating the minimum (which should coincide with sol
1530 ) ..."];
1531    minpoly = sqdiff /. AssociationThread[problemVars -> solVec];
1532
1533    fmSolAssoc = Association[sol[[2]]];
1534    If[\[Sigma]exp == Automatic,
1535      \[Sigma]exp = Sqrt[minpoly / degressOfFreedom];
1536    ];
1537    CapitalDelta\[Chi]2 = Sqrt[degressOfFreedom];
1538    Amat = (1/\[Sigma]exp^2) * Transpose[linMat[[presentDataIndices]]].linMat[[presentDataIndices]];
1539    paramIntervals = EllipsoidBoundingBox[Amat, CapitalDelta\[Chi]2];
1540
1541    PrintFun["Calculating the uncertainty in the parameters ..."];
1542    solWithUncertainty = Table[
1543      (
1544        aVar = constrainedProblemVarsList[[varIdx]];
1545        paramBest = aVar /. fmSolAssoc;
1546        (aVar -> {paramBest, paramIntervals[[varIdx, 2]]})
1547      ),
1548      {varIdx, 1, Length[constrainedProblemVarsList]-shiftToggle}
1549    ];
1550
1551    bestRMS = Sqrt[minpoly / degressOfFreedom];
1552    bestParams = sol[[2]];
1553    bestWithConstraints = Association@Join[constraints, bestParams];
1554    bestWithConstraints = bestWithConstraints /. bestWithConstraints;
1555    bestWithConstraints = (# + 0.) & /@ bestWithConstraints;
1556
1557    solCompendium["degreesOfFreedom"] = degressOfFreedom;
1558    solCompendium["solWithUncertainty"] = solWithUncertainty;
1559    solCompendium["truncatedDim"] = truncationUmbral;
1560    solCompendium["fittedLevels"] = Length[presentDataIndices];
1561
1562    solCompendium["actualSteps"] = steps;
1563    solCompendium["bestRMS"] = bestRMS;
1564    solCompendium["problemVars"] = problemVars;
1565    solCompendium["paramSols"] = paramSols;
1566    solCompendium["rmsHistory"] = rmsHistory;
1567    solCompendium["Appendix"] = OptionValue["
```

```

1560 AppendToLogFile"];
1561 solCompendium["timeTaken/s"] = timeTaken;
1562 solCompendium["bestParams"] = bestParams;
1563 solCompendium["bestParamsWithConstraints"] = bestWithConstraints;
1564
1565 If[OptionValue["SaveEigenvectors"],
1566     solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]}
1567     &/@ (Chop /@ ShiftedLevels[states]),
1568     (
1569         finalEnergies = Sort[First /@ states];
1570         finalEnergies = finalEnergies - finalEnergies[[1]];
1571         finalEnergies = finalEnergies + \[Epsilon]Best;
1572         finalEnergies = Chop /@ finalEnergies;
1573         solCompendium["energies"] = finalEnergies;
1574     )
1575 ];
1576 If[OptionValue["SaveToLog"],
1577     PrintFun["Saving the solution to the log file ..."];
1578     LogSol[solCompendium, logFilePrefix];
1579 ];
1580 PrintFun["Finished ..."];
1581 Return[solCompendium];
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
1623   define the Hamiltonian (including the initial values for
1624   problemVars).
1625
1626 The function saves the solution to a file. The file is named with a
1627   prefix (controlled by the option \"FilePrefix\") and a UUID. The
1628   file is saved in the log sub-directory as a .m file.
1629
1630 Here's a description of the different parts of this function: first
1631   the Hamiltonian is assembled and simplified using the given
1632   simplifications. Then the intermediate coupling basis is

```

calculated using the free-ion parameters for the given lanthanide. The Hamiltonian is then changed to the intermediate coupling basis and truncated. The truncated Hamiltonian is then compiled into a function that can be used to calculate the energy levels of the truncated Hamiltonian. The function that calculates the energy levels is then used to fit the experimental data. The fitting is done using `FindMinimum` with the Levenberg-Marquardt method.

```

1599
1600 The function returns an association with the following keys:
1601
1602 - "bestRMS" which is the best  $\langle \Sigma \rangle$  value found.
1603 - "bestParams" which is the best set of parameters found for the variables that were not constrained.
1604 - "bestParamsWithConstraints" which has the best set of parameters (from - "bestParams") together with the used constraints. These include all the parameters in the model, even those that were not fitted for.
1605 - "paramSols" which is a list of the parameters trajectories during the stepping of the fitting algorithm.
1606 - "timeTaken/s" which is the time taken to find the best fit.
1607 - "simplifier" which is the simplifier used to simplify the Hamiltonian.
1608 - "excludeDataIndices" as given in the input.
1609 - "startValues" as given in the input.
1610
1611 - "freeIonSymbols" which are the symbols used in the intermediate coupling basis.
1612 - "truncationEnergy" which is the energy used to truncate the Hamiltonian, if it was set to Automatic, the value here is the actual energy used.
1613 - "numE" which is the number of electrons in the fnumE configuration.
1614 - "expData" which is the experimental data used for fitting.
1615 - "problemVars" which are the symbols considered for fitting
1616
1617 - "maxIterations" which is the maximum number of iterations used by NMinimize.
1618 - "hamDim" which is the dimension of the full Hamiltonian.
1619 - "allVars" which are all the symbols defining the Hamiltonian under the aggregate simplifications.
1620 - "freeBies" which are the free-ion parameters used to define the intermediate coupling basis.
1621 - "truncatedDim" which is the dimension of the truncated Hamiltonian.
1622 - "compiledIntermediateFname" the file name of the compiled function used for the truncated Hamiltonian.
1623
1624 - "fittedLevels" which is the number of levels fitted for.
1625 - "actualSteps" the number of steps that FindMininum actually took.
1626 - "solWithUncertainty" which is a list of replacement rules of the form (paramSymbol -> {bestEstimate, uncertainty}).
1627 - "rmsHistory" which is a list of the  $\langle \Sigma \rangle$  values found during the fitting.
1628 - "Appendix" which is an association appended to the log file under the key "Appendix".
1629 - "presentDataIndices" which is the list of indices in expData that were used for fitting, this takes into account both the empty indices in expData and also the indices in excludeDataIndices.
1630
1631 - "states" which contains a list of eigenvalues and eigenvectors for the fitted model, this is only available if the option "SaveEigenvectors" is set to True; if a general shift of energy was allowed for in the fitting, then the energies are shifted accordingly.
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 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   "AppendToFile"     -> <||>,
1670   "SaveToLog"        -> False,
1671   "Energy Uncertainty in K" -> Automatic,

```

```

1672 "MagneticSimplifier" -> {
1673   M2 -> 56/100 M0 ,
1674   M4 -> 31/100 M0 ,
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  degressOfFreedom,
1810  dependentVars,
1811  diagonalBlocks,
1812  diagonalScalarBlocks,
1813  diff,
1814  eigenEnergies,
1815  eigenvalueDispenserTemplate,
1816  eigenVectors,
1817  elevatedIntermediateEigensystems,
1818
1819  endTime,
1820  fmSolAssoc,
1821  freeBies,

```

```

1822 freeIenergiesAndMultiplets ,
1823 fullHam ,
1824 fullSolVec ,
1825 ham ,
1826 hamDim ,
1827 hamEigenvaluesTemplate ,
1828 hamString ,
1829
1830 indepSolVecVec ,
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
1943 simplifier = Join[magneticSimplifier,
1944   magFieldSimplifier,
1945   symmetrySimplifier,
1946   threeBodySimplifier,
1947   otherSimplifier];
1948
1949 PrintFun["Determining gaps in the data ..."];
1950 (* whatever is non-numeric is assumed as a known gap *)
1951 presentDataIndices = Flatten[Position[expData, {_?(NumericQ[#] &
1952 , ___]}];
1953 (* some indices omitted here based on the excludeDataIndices
1954 argument, note that presentDataIndices is somewhat a misnomer*)
1955 presentDataIndices = Complement[presentDataIndices,
1956 excludeDataIndices];
1957
1958 solCompendium["simplifier"] = simplifier;
1959 solCompendium["excludeDataIndices"] = excludeDataIndices;
1960 solCompendium["startValues"] = startValues;
1961 solCompendium["freeIonSymbols"] = freeIonSymbols;
1962 solCompendium["truncationEnergy"] = truncationEnergy;
1963 solCompendium["numE"] = numE;
1964 solCompendium["expData"] = expData;
1965 solCompendium["problemVars"] = problemVars;
1966 solCompendium["maxIterations"] = maxIterations;
1967 solCompendium["hamDim"] = hamDim;
1968 solCompendium["constraints"] = constraints;
1969
1970 modelSymbols = Select[paramSymbols,
1971   Not[MemberQ[Join[racahSymbols,
1972     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[
LoadLa3Parameters[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,
truncationEnergy, simplifier}];
2024 compiledIntermediateFname = ln <> "-compiled-intermediate-
truncated-ham-" <> ToString[varHash] <> ".mx";
2025 compiledIntermediateFname = FileNameJoin[{moduleDir, "compiled",
compiledIntermediateFname}];
2026 solCompendium["compiledIntermediateFname"] =

```

```

2027 compiledIntermediateFname;
2028
2029 If [FileExistsQ[compiledIntermediateFname],
2030 (
2031     PrintFun["This ion, free-ion params, and full set of variables
2032 have been used before (as determined by {numE, allVars, freeBies,
2033 truncationEnergy, simplifier}). Loading the previously saved
2034 compiled function and intermediate coupling basis ..."];
2035     PrintFun["Using : ", compiledIntermediateFname];
2036     {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =
2037     Import[compiledIntermediateFname];
2038 ),
2039 (
2040     If [truncationEnergy == Infinity,
2041     (
2042         ham = HamMatrixAssembly[numE,
2043             "ReturnInBlocks" -> False,
2044             "OperatorBasis" -> "MostlyOrthogonal"];
2045         theSimplifier = simplifier;
2046         ham = Normal @ ReplaceInSparseArray[ham, simplifier];
2047         PrintFun["Compiling a function for the Hamiltonian with no
2048 truncation ..."];
2049         (* compile a function that will calculate the truncated
2050 Hamiltonian given the parameters in allVars, this is the function
2051 to be use in fitting *)
2052         compileIntermediateTruncatedHam = Compile[Evaluate[allVars
2053 ], Evaluate[ham]];
2054         truncatedIntermediateBasis =
2055 SparseArray@IdentityMatrix[Binomial[14, numE]];
2056         (* save the compiled function *)
2057         PrintFun["Saving the compiled function for the Hamiltonian
2058 with no truncation and a placeholder intermediate basis ..."];
2059         Export[compiledIntermediateFname, {
2060         compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
2061     ),
2062     (
2063         (* grab the Hamiltonian preserving the block structure *)
2064         PrintFun["Assembling the Hamiltonian for f^", numE, " keeping
2065 the block structure ..."];
2066         ham = HamMatrixAssembly[numE,
2067             "ReturnInBlocks" -> True,
2068             "OperatorBasis" -> "MostlyOrthogonal"];
2069         (* apply the simplifier *)
2070         PrintFun["Simplifying using the aggregate set of
2071 simplification rules ..."];
2072         ham = Map[ReplaceInSparseArray[#, simplifier] &, ham,
2073 {2}];
2074         PrintFun["Zeroing out every symbol in the Hamiltonian that is
2075 not a free-ion parameter ..."];
2076
2077         (* Get the free ion symbols *)
2078         freeIonSimplifier = (# -> 0) & /@ Complement[
2079             reducedModelSymbols, freeIonSymbols];
2080
2081         (* Take the diagonal blocks for the intermediate analysis *)
2082         PrintFun["Grabbing the diagonal blocks of the Hamiltonian ...
2083 "];
2084         diagonalBlocks = Diagonal[ham];
2085
2086         (* simplify them to only keep the free ion symbols *)
2087         PrintFun["Simplifying the diagonal blocks to only keep the
2088 free ion symbols ..."];
2089         diagonalScalarBlocks = ReplaceInSparseArray[#, freeIonSimplifier] & /@ diagonalBlocks;
2090
2091         (* these include the MJ quantum numbers, remove that *)
2092         PrintFun["Contracting the basis vectors by removing the MJ
2093 quantum numbers from the diagonal blocks ..."];
2094         diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
2095
2096         argsOfTheIntermediateEigensystems = StringJoin[Riffle
2097             [Prepend[(ToString[#] <> "v_") & /@ freeIonSymbols, "numE_"], ", ", ""]];
2098         argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle
2099             [(ToString[#] <> "v") & /@ freeIonSymbols, ", ", "]];
2100
2101         PrintFun["argsOfTheIntermediateEigensystems = ",
2102             argsOfTheIntermediateEigensystems];

```

```

2079     PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",  

2080     argsForEvalInsideOfTheIntermediateSystems];  

2081     PrintFun["(if the following fails, it might help to see if  

2082     the arguments of TheIntermediateEigensystems match the ones shown  

2083     above)"];  

2084     (* compile a function that will effectively calculate the  

2085     spectrum of all of the scalar blocks given the parameters of the  

2086     free-ion part of the Hamiltonian *)  

2087     (* compile one function for each of the blocks *)  

2088     PrintFun["Compiling functions for the diagonal blocks of the  

2089     Hamiltonian ..."];  

2090     compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate  

2091     [N[Normal[#]]]]& /@ diagonalScalarBlocks;  

2092     (* use that to create a function that will calculate the free  

2093     -ion eigensystem *)  

2094     TheIntermediateEigensystems[numEv_, E0pv_, E1pv_, E2pv_,  

2095     E3pv_,  $\zeta$ v_] :=  

2096     theNumericBlocks = (#[E0pv, E1pv, E2pv, E3pv,  $\zeta$ v]&) /@  

2097     compiledDiagonal;  

2098     theIntermediateEigensystems = Eigensystem /@  

2099     theNumericBlocks;  

2100     Js = AllowedJ[numEv];  

2101     basisJ = BasisLSJMJ[numEv, "AsAssociation" -> True];  

2102     (* having calculated the eigensystems with the removed  

2103     degeneracies, put the degeneracies back in explicitly *)  

2104     elevatedIntermediateEigensystems = MapIndexed[EigenLever  

2105     [#1, 2Js[[#2[[1]]]]+1 ]&, theIntermediateEigensystems];  

2106     (* Identify a single MJ to keep *)  

2107     pivot = If[EvenQ[numEv], 0, -1/2];  

2108     LSJmultiplets = (#[[1]] <> ToString[InputForm[#[[2]]]])&/  

2109     @Select[BasisLSJMJ[numEv], #[[{-1}] == pivot &];  

2110     (* calculate the multiplet assignments that the  

2111     intermediate basis eigenvectors have *)  

2112     needlePosition = 0;  

2113     multipletAssignments = Table[  

2114       (
2115         J = Js[[idx]];  

2116         eigenVecs = theIntermediateEigensystems[[idx]][[2]];  

2117         majorComponentIndices = Ordering[Abs[#][[-1]]]&/  

2118         @eigenVecs;  

2119         majorComponentIndices += needlePosition;  

2120         needlePosition += Length[  

2121           majorComponentIndices];  

2122         majorComponentAssignments = LSJmultiplets[[#]]&/  

2123         @majorComponentIndices;  

2124         (* All of the degenerate eigenvectors belong to the  

2125         same multiplet*)  

2126         elevatedMultipletAssignments = ListRepeater[  

2127           majorComponentAssignments, 2J+1];  

2128           elevatedMultipletAssignments  

2129           ),  

2130           {idx, 1, Length[Js]}  

2131         ];  

2132  

2133     (* put together the multiplet assignments and the energies  

2134     *)  

2135     freeIenergiesAndMultiplets = Transpose/@Transpose[{First/  

2136     @elevatedIntermediateEigensystems, multipletAssignments}];  

2137     freeIenergiesAndMultiplets = Flatten[  

2138     freeIenergiesAndMultiplets, 1];  

2139     (* calculate the change of basis matrix using the  

2140     intermediate coupling eigenvectors *)  

2141     basisChanger = BlockDiagonalMatrix[Transpose/@Last/  

2142     @elevatedIntermediateEigensystems];  

2143     basisChanger = SparseArray[basisChanger];  

2144     Return[{theIntermediateEigensystems,  

2145     multipletAssignments,  

2146     elevatedIntermediateEigensystems,  

2147     freeIenergiesAndMultiplets,  

2148     basisChanger}]\n2149   );\n2150\n2151   PrintFun["Calculating the intermediate eigensystems for ",ln,  

2152   " using free-ion params from LaF3 ..."];\n2153   (* calculate intermediate coupling basis using the free-ion

```

```

1567 params for LaF3 *)
1568 {theIntermediateEigensystems, multipletAssignments,
1569 elevatedIntermediateEigensystems, freeEnergiesAndMultiplets,
1570 basisChanger} = TheIntermediateEigensystems@@freeBies;
1571
1572 (* use that intermediate coupling basis to compile a function
1573 for the full Hamiltonian *)
1574 allFreeEnergies = Flatten[First/
1575 @elevatedIntermediateEigensystems];
1576 (* important that the intermediate coupling basis have
1577 attached energies, which make possible the truncation *)
1578 ordering = Ordering[allFreeEnergies];
1579 (* sort the free ion energies and determine which indices
1580 should be included in the truncation *)
1581 allFreeEnergiesSorted = Sort[allFreeEnergies];
1582 {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];
1583 (* determine the index at which the energy is equal or larger
1584 than the truncation energy *)
1585 sortedTruncationIndex = Which[
1586 truncationEnergy > (maxFreeEnergy - minFreeEnergy),
1587 hamDim,
1588 True,
1589 FirstPosition[allFreeEnergiesSorted - Min[
1590 allFreeEnergiesSorted], x_ /; x > truncationEnergy, {0}, 1][[1]]
1591 ];
1592 (* the actual energy at which the truncation is made *)
1593 roundedTruncationEnergy = allFreeEnergiesSorted[[sortedTruncationIndex]];
1594
1595 (* the indices that participate in the truncation *)
1596 truncationIndices = ordering[[;; sortedTruncationIndex]];
1597 PrintFun["Computing the block structure of the change of
1598 basis array ..."];
1599 blockSizes = BlockArrayDimensionsArray[ham];
1600 basisChangerBlocks = ArrayBlocker[basisChanger, blockSizes];
1601 blockShifts = First /@ Diagonal[blockSizes];
1602 numBlocks = Length[blockSizes];
1603
1604 (* using the ham (with all the symbols) change the basis to
1605 the computed one *)
1606 PrintFun["Changing the basis of the Hamiltonian to the
1607 intermediate coupling basis ..."];
1608 intermediateHam = BlockMatrixMultiply[ham, basisChangerBlocks];
1609 PrintFun["Distributing products inside of symbolic matrix
1610 elements to keep complexity in check ..."];
1611 Do[
1612 intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
1613 intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
1614 {rowIdx, 1, numBlocks},
1615 {colIdx, 1, numBlocks}
1616 ];
1617 intermediateHam = BlockMatrixMultiply[BlockTranspose[
1618 basisChangerBlocks], intermediateHam];
1619 PrintFun["Distributing products inside of symbolic matrix
1620 elements to keep complexity in check ..."];
1621 Do[
1622 intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
1623 intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
1624 {rowIdx, 1, numBlocks},
1625 {colIdx, 1, numBlocks}
1626 ];
1627 (* using the truncation indices truncate that one *)
1628 PrintFun["Truncating the Hamiltonian ..."];
1629 truncatedIntermediateHam = TruncateBlockArray[intermediateHam,
1630 truncationIndices, blockShifts];
1631 (* these are the basis vectors for the truncated hamiltonian
1632 *)
1633 PrintFun["Saving the truncated intermediate basis ..."];
1634 truncatedIntermediateBasis = basisChanger[[All,
1635 truncationIndices]];
1636
1637 PrintFun["Compiling a function for the truncated Hamiltonian
1638 ..."];
1639 (* compile a function that will calculate the truncated
1640 Hamiltonian given the parameters in allVars, this is the function

```

```

    to be use in fitting *)
2181 compileIntermediateTruncatedHam = Compile[Evaluate[allVars], 
2182 Evaluate[truncatedIntermediateHam]];
    (* save the compiled function *)
2183 PrintFun["Saving the compiled function for the truncated
Hamiltonian and the truncated intermediate basis ..."];
2184 Export[compiledIntermediateFname, {
compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
    )
2185 ];
)
2186 );
2187 ];
2188 ];

2189 truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
2190 PrintFun["The truncated Hamiltonian has a dimension of ",
truncationUmbral, "x", truncationUmbral, "..."];
2191 presentDataIndices = Select[presentDataIndices, # <=
truncationUmbral &];
2192 solCompendium["presentDataIndices"] = presentDataIndices;
2193
(* the problemVars are the symbols that will be fitted for *)

2194
2195 PrintFun["Starting up the fitting process using the Levenberg-
Marquardt method ..."];
2196 (* using the problemVars I need to create the argument list
including _?NumericQ *)
2197 problemVarsQ = (ToString[#] <> "_?NumericQ") & /@ 
problemVars;
2198 problemVarsQString = StringJoin[Riffle[problemVarsQ, ", "]];
(* we also need to have the padded arguments with the variables
in the right position and the fixed values in the remaining ones
*)
2199 problemVarsPositions = Position[allVars, #][[1, 1]] & /@ 
problemVars;
2200 problemVarsString = StringJoin[Riffle[ToString /@ problemVars, 
", "]];
(* 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 *)
2201 varsWithConstants = standardValues;
2202 varsWithConstants[[problemVarsPositions]] = problemVars;
2203 varsWithConstantsString = ToString[
varsWithConstants];
2204
(* this following function serves eigenvalues from the
Hamiltonian, has memoization so it might grow to use a lot of RAM
*)
2205 Clear[HamSortedEigenvalues];
2206 hamEigenvaluesTemplate = StringTemplate["
HamSortedEigenvalues['problemVarsQ']:=(
2207     ham = compileIntermediateTruncatedHam@@'
varsWithConstants';
2208     eigenValues = Chop[Sort@Eigenvalues@ham];
2209     eigenValues = eigenValues - Min[eigenValues];
2210     HamSortedEigenvalues['problemVarsString'] = eigenValues;
2211     Return[eigenValues]
2212 )"];
2213 hamString = hamEigenvaluesTemplate[<|
2214     "problemVarsQ"      -> problemVarsQString,
2215     "varsWithConstants" -> varsWithConstantsString,
2216     "problemVarsString" -> problemVarsString
2217     |>];
2218 ToExpression[hamString];
2219
(* 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 *)
2220 eigenvalueDispenserTemplate = StringTemplate["
PartialHamEigenvalues['problemVarsQ'][i_]:=(
2221     eigenVals = HamSortedEigenvalues['problemVarsString'];
2222     eigenVals[[i]]
2223 )
2224 ];
2225 eigenValueDispenserString = eigenvalueDispenserTemplate[<|
2226     "problemVarsQ"      -> problemVarsQString,
2227     "problemVarsString" -> problemVarsString
2228 ];

```

```

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
2263 If[Not[MemberQ[openNotebooks,"Solver Progress"]]&& OptionValue["ProgressView"],
2264   ProgressNotebook["Basic" -> False];
2265 ];
2266 degressOfFreedom = Length[presentDataIndices] - Length[
2267 problemVars] - 1;
2268 PrintFun["Fitting for ", Length[presentDataIndices], " data
2269 points with ", Length[problemVars], " free parameters.", " The
2270 effective degrees of freedom are ", degressOfFreedom, " ..."];
2271
2272 PrintFun["Fitting model to data ..."];
2273 startTime = Now;
2274 shiftToggle = If[addShift, 1, 0];
2275 sol = FindMinimum[
2276   Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@
2277 constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
2278   {j, presentDataIndices}],
2279   problemVarsWithStartValues,
2280   Method -> "LevenbergMarquardt",
2281   MaxIterations -> OptionValue["MaxIterations"],
2282   AccuracyGoal -> OptionValue["AccuracyGoal"],
2283   StepMonitor :> (
2284     steps += 1;
2285     currentSqSum = Sum[(expData[[j]][[1]] - (
2286       PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle
2287 * \[Epsilon])^2, {j, presentDataIndices}];
2288     currentRMS = Sqrt[currentSqSum / degressOfFreedom];
2289     paramSols = AddToList[paramSols, constrainedProblemVarsList,
2290     maxHistory];
2291     rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
2292   )
2293 ];
2294 endTime = Now;
2295 timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
2296 PrintFun["Solution found in ", timeTaken, "s"];
2297
2298 solVec = constrainedProblemVars /. sol[[-1]];
2299 indepSolVec = indepVars /. sol[[-1]];
2300 If[addShift,
2301   \[Epsilon]Best = \[Epsilon]/.sol[[-1]],
2302   \[Epsilon]Best = 0
2303 ];
2304 fullSolVec = standardValues;
2305 fullSolVec[[problemVarsPositions]] = solVec;
2306 PrintFun["Calculating the truncated numerical Hamiltonian
2307 corresponding to the solution ..."];
2308 fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
2309 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 ];
2330 PrintFun["Removing the gradient of the ground state ..."];
2331 linMat = (# - #[[1]] & /@ linMat);
2332 PrintFun["Transposing derivative matrices into columns ..."];
2333 linMat = Transpose[linMat];
2334
2335 PrintFun["Calculating the eigenvalue vector at solution ..."];
2336 \[Lambda]0Vec = Transpose[{eigenEnergies[[presentDataIndices]]}];
2337 PrintFun["Putting together the experimental vector ..."];
2338 \[Lambda]exp = Transpose[{First /@ expData[[presentDataIndices
]]}];
2339 problemVarsVec = If[addShift,
2340     Transpose[{constrainedProblemVarsList[[;; -2]]}],
2341     Transpose[{constrainedProblemVarsList}]
2342 ];
2343 indepSolVecVec = Transpose[{indepSolVec}];
2344 PrintFun["Calculating the difference between eigenvalues at
solution ..."];
2345 diff = If[linMat == {},
2346     (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best,
2347     (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best + linMat[[presentDataIndices]].(problemVarsVec - indepSolVecVec)
2348 ];
2349 PrintFun["Calculating the sum of squares of differences around
solution ..."];
2350 sqdiff = Expand[(Transpose[diff] . diff)[[1, 1]]];
2351 PrintFun["Calculating the minimum (which should coincide with sol
) ..."];
2352 minpoly = sqdiff /. AssociationThread[problemVars -> solVec
];
2353 fmSolAssoc = Association[sol[[2]]];
2354 If[\[Sigma]exp == Automatic,
2355     \[Sigma]exp = Sqrt[minpoly / degressOfFreedom];
2356 ];
2357 \[CapitalDelta]\[Chi]2 = Sqrt[degressOfFreedom];
2358 Amat = (1/\[Sigma]exp^2) * Transpose[linMat[[presentDataIndices
]]].linMat[[presentDataIndices]];
2359 paramIntervals = EllipsoidBoundingBox[Amat, \[CapitalDelta]\[Chi]
2];
2360 PrintFun["Calculating the uncertainty in the parameters ..."];
2361 solWithUncertainty = Table[
2362 (
2363     aVar = constrainedProblemVarsList[[varIdx]];
2364     paramBest = aVar /. fmSolAssoc;

```

```

2363     (aVar -> {paramBest, paramIntervals[[varIdx, 2]]})
2364   ),
2365 {varIdx, 1, Length[constrainedProblemVarsList]-shiftToggle}
2366 ];
2367
2368 bestRMS = Sqrt[minpoly / degressOfFreedom];
2369 bestParams = sol[[2]];
2370 bestWithConstraints = Association@Join[constraints, bestParams];
2371 bestWithConstraints = bestWithConstraints /. bestWithConstraints;
2372 bestWithConstraints = (# + 0.) & /@ bestWithConstraints;
2373
2374 solCompendium["degreesOfFreedom"] = degressOfFreedom;
2375 solCompendium["solWithUncertainty"] = solWithUncertainty;
2376 solCompendium["truncatedDim"] = truncationUmbral;
2377 solCompendium["fittedLevels"] = Length[presentDataIndices]
2378 ];
2379 solCompendium["actualSteps"] = steps;
2380 solCompendium["bestRMS"] = bestRMS;
2381 solCompendium["problemVars"] = problemVars;
2382 solCompendium["paramSols"] = paramSols;
2383 solCompendium["rmsHistory"] = rmsHistory;
2384 solCompendium["Appendix"] = OptionValue["AppendToFile"];
2385 solCompendium["timeTaken/s"] = timeTaken;
2386 solCompendium["bestParams"] = bestParams;
2387 solCompendium["bestParamsWithConstraints"] = bestWithConstraints;
2388
2389 If[OptionValue["SaveEigenvalues"],
2390   solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]}
2391 &/@ (Chop /@ ShiftedLevels[states]),
2392 (
2393   finalEnergies = Sort[First /@ states];
2394   finalEnergies = finalEnergies - finalEnergies[[1]];
2395   finalEnergies = finalEnergies + \[Epsilon]Best;
2396   finalEnergies = Chop /@ finalEnergies;
2397   solCompendium["energies"] = finalEnergies;
2398 )
2399 ];
2400 If[OptionValue["SaveToLog"],
2401   PrintFun["Saving the solution to the log file ..."];
2402   LogSol[solCompendium, logFilePrefix];
2403 ];
2404 PrintFun["Finished ..."];
2405 Return[solCompendium];
2406 )
2407 ];
2408
2409 StringToSLJ[string_] := Module[
2410 {stringed = string, LS, J, LSindex},
2411 (
2412 If[StringContainsQ[stringed, "+"],
2413   Return["mixed"]
2414 ];
2415 LS = StringTake[stringed, {1, 2}];
2416 If[StringContainsQ[stringed, "("],
2417 (
2418   LSindex =
2419   StringCases[stringed, RegularExpression["\\(((^)*\\))"] :> "$1"];
2420   LS = LS <> LSindex;
2421   stringed = StringSplit[stringed, ")"][[{-1}]];
2422   J = ToExpression[stringed];
2423   ),
2424   (
2425   J = ToExpression@StringTake[stringed, {3, -1}];
2426   )
2427 ];
2428 {LS, J}
2429 )
2430 ];
2431
2432 FreeIonSolver::usage="This function takes a list of experimental data
and the number of electrons in the lanthanide ion and returns the
free-ion parameters that best fit the data. The options are:
- F4F6_SlaterRatios: a list of two numbers that represent the ratio
of F4 to F2 and F6 to F2, respectively.

```

```

2432 - PrintFun: a function that will be used to print the progress of
2433   the fitting process.
2434 - MaxIterations: the maximum number of iterations that the fitting
2435   process will run.
2436 - MaxMultiplets: the maximum number of multiplets that will be used
2437   in the fitting process.
2438 - MaxPercent: the maximum percentage of the data that can be off by
2439   the fitting.
2440 - SubSetBounds: a list of two numbers that represent the minimum
2441   and maximum number of multiplets that will be used in the fitting
2442   process.
2443 The function returns an association with the following keys:
2444 - bestParams: the best parameters found in the fitting.
2445 - worstRelativeError: the worst relative error in the fitting.
2446 - SlaterRatios: the Slater ratios used in the fitting.
2447 - usedBaricenters: the baricenters used in the fitting.
2448 If no acceptable solution is found, the function will return all
2449   solutions that are not worse than 10*MaxPercent. A solution is
2450   acceptable if the worst relative error is less than the MaxPercent
2451   option.
2452 ";
2453 Options[FreeIonSolver] = {
2454 "F4F6_SlaterRatios" -> {0.707, 0.516},
2455 "PrintFun" -> PrintTemporary,
2456 "MaxIterations" -> 10000,
2457 "MaxMultiplets" -> 12,
2458 "MaxPercent" -> 3.,
2459 "SubSetBounds" -> {5, 12}
2460 };
2461 FreeIonSolver[expData_, numE_] := Module[
2462   {maxMultiplets, maxPercent, F4overF2, F6overF2, PrintFun,
2463    minSubsetSize, maxSubsetSize, multipletEnergies, numMultiplets,
2464    allEqns, subsetSizes, ln, solutions, subsets, subset, eqns, slope,
2465    intercept, meritFun, sol, goodThings, bestThings, bestOfAll,
2466    finalSol, usedMultiplets, usedBaricenters},
2467   (
2468     maxMultiplets = OptionValue["MaxMultiplets"];
2469     maxIterations = OptionValue["MaxIterations"];
2470     maxPercent = OptionValue["MaxPercent"];
2471     F4overF2 = OptionValue["F4F6_SlaterRatios"][[1]];
2472     F6overF2 = OptionValue["F4F6_SlaterRatios"][[2]];
2473     PrintFun = OptionValue["PrintFun"];
2474     minSubsetSize = OptionValue["SubSetBounds"][[1]];
2475     maxSubsetSize = OptionValue["SubSetBounds"][[2]];
2476     freeIonParams = {F0, F2, F4, F6, \[Zeta]};
2477     ln = theLanthanides[[numE]];

2478     PrintFun["Parsing the barycenters of the different multiplets ..."];
2479     multipletEnergies = Map[First, #] & /@ GroupBy[expData, #[[2]] &];
2480     multipletEnergies = Mean[Select[#, NumberQ]] & /@
2481     multipletEnergies;
2482     multipletEnergies = Select[multipletEnergies, FreeQ[#, Mean] &];
2483     multipletEnergies = KeySelect[KeyMap[StringToSLJ,
2484     multipletEnergies], # != "mixed" &];
2485     multipletEnergies = KeyMap[Prepend[#, numE] &, multipletEnergies];
2486     numMultiplets = Length[multipletEnergies];

2487     PrintFun["Composing the system of equations for the free-ion
2488 energies ..."];
2489     allEqns = KeyValueMap[FreeIonTable[#1] == #2 &,
2490     multipletEnergies];
2491     allEqns = Select[allEqns, FreeQ[#, Missing] &];
2492     allEqns = Append[Coefficient[#[[1]], {F0, F2, F4, F6, \[Zeta]}],
2493     #[[2]]] & /@ allEqns;
2494     allEqns = allEqns[;; Min[Length[allEqns], maxMultiplets]];
2495     subsetSizes = Range[minSubsetSize, Min[numMultiplets,
2496     maxSubsetSize]];
2497     numSubsets = #[, Binomial[numMultiplets, #]} & /@ subsetSizes;
2498     numSubsets = Transpose@SortBy[numSubsets, Last];
2499     accSizes = Accumulate[numSubsets[[2]]];
2500     numSubsets = Transpose@Append[numSubsets, accSizes];
2501     lastSub = SelectFirst[numSubsets, #[[3]] > 1000 &, Last[
2502     numSubsets]];

```

```

2485 lastPosition = Position[numSubsets, lastSub][[1, 1]];
2486 chosenSubsetSizes = #[[1]] & /@ numSubsets[[;; lastPosition]];
2487 solutions = <||>;
2488
2489 PrintFun["Selecting subsets of different lengths and fitting with
2490 ratio-constraints ..."];
2491 Do[
2492   subsets = Subsets[Range[1, Length[allEqns]], {subsetSize}];
2493   PrintFun["Considering ", Length[subsets], " barycenter subsets
2494 of size ", subsetSize, " ..."];
2495   Do[
2496     (
2497       subset = subsets[[subsetIndex]];
2498       eqns = allEqns[[subset]];
2499       slope = #[[;; 5]] & /@ eqns;
2500       intercept = #[[6]] & /@ eqns;
2501       meritFun = Max[100 * Expand[Abs[(slope . freeIonParams -
2502 intercept)]/intercept]];
2503       sol = NMinimize[{meritFun,
2504         F0 > 0,
2505         F2 > 1000.,
2506         F4 == F4overF2 * F2,
2507         F6 == F6overF2 * F2,
2508         ζ > 0},
2509       freeIonParams,
2510       MaxIterations -> maxIterations,
2511       Method -> "Convex"];
2512       solutions[{subsetSize, subset}] = sol;
2513     )
2514   , {subsetIndex, 1, Length[subsets]}
2515   ],
2516   {subsetSize, chosenSubsetSizes}
2517 ];
2518
2519 PrintFun["Collecting solutions of different subset size ..."];
2520 goodThings = Table[
2521   (chunk =
2522   Normal[Sort[
2523     KeySelect[#[[1]] & /@ solutions, #[[1]] == subSize &]];
2524   If[Length[chunk] >= 1,
2525     chunk[[1]],
2526     Null
2527   ]), {subSize, subsetSizes}];
2528 goodThings = Select[goodThings, Head[#] === Rule &];
2529
2530 PrintFun["Picking the solutions that are not worse than ",
2531 maxPercent, "% ..."];
2532 bestThings = Select[goodThings, #[[2]] <= maxPercent &];
2533 If[bestThings == {},
2534   Print["No acceptable solution found, consider increasing
2535 maxPercent or inspecting the given data ..."];
2536   Return[bestThings];
2537 ];
2538
2539 PrintFun["Keeping the solution with the largest number of used
2540 barycenters ..."];
2541 bestOfAll = bestThings[[-1]];
2542 sol = solutions[bestOfAll[[1]]];
2543 subset = bestOfAll[[1, 2]];
2544 eqns = allEqns[[subset]];
2545 slope = #[[;; 5]] & /@ eqns;
2546 intercept = #[[6]] & /@ eqns;
2547 usedMultiplets = Keys[multipletEnergies][[subset]];
2548 usedBaricenters = {#, multipletEnergies[#]} & /@ usedMultiplets;
2549 uniqueLS = DeleteDuplicates[#[[2]] &/@ Keys[multipletEnergies]];
2550 solAssoc = Association[sol[[2]]];
2551 usedLaF3 = False;
2552 If[Length[uniqueLS] == 1,
2553   (
2554     Print["There is too little data to find Slater parameters,
2555 using the ones for LaF3, and keeping the fitted spin-orbit zeta
2556 ..."];
2557     laf3params = LoadLaF3Parameters[ln];
2558     usedLaF3 = True;
2559     solAssoc[F0] = laf3params[F0];
2560     solAssoc[F2] = laf3params[F2];

```

```

2553     solAssoc[F4] = laf3params[F4];
2554     solAssoc[F6] = laf3params[F6];
2555   )
2556 ];
2557 finalSol = <| "bestParams" -> solAssoc,
2558           "usedLaF3" -> usedLaF3,
2559           "worstRelativeError" -> sol[[1]],
2560           "SlaterRatios" -> {F4overF2, F6overF2},
2561           "usedBaricenters" -> usedBaricenters|>;
2562 Return[finalSol];
2563 )
2564 ];
2565
2566 EndPackage [];

```

18.3 qplotter.m

This module has a few useful plotting routines.

```

1 BeginPackage["qplotter`"];
2
3 GetColor;
4 IndexMappingPlot;
5 ListLabelPlot;
6 AutoGraphicsGrid;
7 SpectrumPlot;
8 WaveToRGB;
9
10 Begin["`Private`"];
11
12 AutoGraphicsGrid::usage="AutoGraphicsGrid[graphsList] takes a list
13   of graphics and creates a GraphicsGrid with them. The number of
14   columns and rows is chosen automatically so that the grid has a
15   squarish shape.";
16 Options[AutoGraphicsGrid] = Options[GraphicsGrid];
17 AutoGraphicsGrid[graphsList_, opts : OptionsPattern[]] :=
18   (
19     numGraphs = Length[graphsList];
20     width = Floor[Sqrt[numGraphs]];
21     height = Ceiling[numGraphs/width];
22     groupedGraphs = Partition[graphsList, width, width, 1, Null];
23     GraphicsGrid[groupedGraphs, opts]
24   )
25
26 Options[IndexMappingPlot] = Options[Graphics];
27 IndexMappingPlot::usage =
28   "IndexMappingPlot[pairs] take a list of pairs of integers and
29   creates a visual representation of how they are paired. The first
30   indices being depicted in the bottom and the second indices being
31   depicted on top.";
32 IndexMappingPlot[pairs_, opts : OptionsPattern[]] := Module[{width,
33   height}, (
34   width = Max[First /@ pairs];
35   height = width/3;
36   Return[
37     Graphics[{{Tooltip[Point[{#[[1]], 0}], #[[1]]}, Tooltip[Point
38       [{#[[2]], height}], #[[2]]], Line[{{#[[1]], 0}, {#[[2]], height}}]} & /@ pairs, opts,
39     ImageSize -> 800]]
40   )
41 ]
42
43 TickCompressor[fTicks_] :=
44 Module[{avgTicks, prevTickLabel, groupCounter, groupTally, idx,
45   tickPosition, tickLabel, avgPosition, groupLabel}, (avgTicks =
46   {};;
47   prevTickLabel = fTicks[[1, 2]];
48   groupCounter = 0;
49   groupTally = 0;
50   idx = 1;
51   Do[({{tickPosition, tickLabel} = tick;
52     If[
53       tickLabel === prevTickLabel,
54       (groupCounter += 1;
55       groupTally += tickPosition;

```

```

48     groupLabel = tickLabel),
49 (
50     avgPosition = groupTally/groupCounter;
51     avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
52     groupCounter = 1;
53     groupTally = tickPosition;
54     groupLabel = tickLabel;
55   )
56 ];
57 If[idx != Length[fTicks],
58 prevTickLabel = tickLabel;
59 idx += 1;
60 ), {tick, fTicks}];
61 If[Or[Not[prevTickLabel === tickLabel], groupCounter > 1],
62 (
63   avgPosition = groupTally/groupCounter;
64   avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
65 )
66 ];
67 Return[avgTicks]);
68
69 GetColor[s_Style] := s /. Style[_ , c_] :> c
70 GetColor[_] := Black
71
72 ListLabelPlot::usage="ListLabelPlot[data, labels] takes a list of
    numbers with corresponding labels. The data is grouped according
    to the labels and a ListPlot is created with them so that each
    group has a different color and their corresponding label is shown
    in the horizontal axis.";
73 Options[ListLabelPlot] = Join[Options[ListPlot], {"TickCompression" -> True,
74 "LabelLevels" -> 1}];
75 ListLabelPlot[data_, labels_, opts : OptionsPattern[]] := Module[
76   {uniqueLabels, pallete, groupedByTerm, groupedKeys, scatterGroups,
77   groupedColors, frameTicks, compTicks, bottomTicks, topTicks},
78   (
79     uniqueLabels = DeleteDuplicates[labels];
80     pallete = Table[ColorData["Rainbow", i], {i, 0, 1,
81       1/(Length[uniqueLabels] - 1)}];
82     uniqueLabels = (#[[1]] -> #[[2]]) & /@ Transpose[{RandomSample[uniqueLabels], pallete}];
83     uniqueLabels = Association[uniqueLabels];
84     groupedByTerm = GroupBy[Transpose[{labels, Range[Length[data]], data}], First];
85     groupedKeys = Keys[groupedByTerm];
86     scatterGroups = Transpose[Transpose[#[[2 ;; 3]]] & /@ Values[groupedByTerm]];
87     groupedColors = uniqueLabels[#] & /@ groupedKeys;
88     frameTicks = {Transpose[{Range[Length[data]],
89       Style[Rotate[#, 90 Degree], uniqueLabels[#] & /@ labels]}],
90       Automatic};
91     If[OptionValue["TickCompression"], (
92       compTicks = TickCompressor[frameTicks[[1]]];
93       bottomTicks =
94         MapIndexed[
95           If[EvenQ[First[#2]], {#1[[1]],
96             Tooltip[Style["\[SmallCircle]", GetColor[#1[[2]]], #1[[2]]],
97             }, #1] &, compTicks];
98       topTicks =
99         MapIndexed[
100           If[OddQ[First[#2]], {#1[[1]],
101             Tooltip[Style["\[SmallCircle]", GetColor[#1[[2]]], #1[[2]]],
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]
111 )

```

```

112 ]
113
114 WaveToRGB::usage="WaveToRGB[wave, gamma] takes a wavelength in nm
115 and returns the corresponding RGB color. The gamma parameter is
116 optional and defaults to 0.8. The wavelength wave is assumed to be
117 in nm. If the wavelength is below 380 the color will be the same
118 as for 380 nm. If the wavelength is above 750 the color will be
119 the same as for 750 nm. The function returns an RGBColor object.
120 REF: https://www.noah.org/wiki/wave\_to\_rgb\_in\_Python. ";
121 WaveToRGB[wave_, gamma_ : 0.8] := (
122   wavelength = (wave);
123   Which[
124     wavelength < 380,
125     wavelength = 380,
126     wavelength > 750,
127     wavelength = 750
128   ];
129   Which[380 <= wavelength <= 440,
130     (
131       attenuation = 0.3 + 0.7*(wavelength - 380)/(440 - 380);
132       R = ((-(wavelength - 440)/(440 - 380))*attenuation)^gamma;
133       G = 0.0;
134       B = (1.0*attenuation)^gamma;
135     ),
136     440 <= wavelength <= 490,
137     (
138       R = 0.0;
139       G = ((wavelength - 440)/(490 - 440))^gamma;
140       B = 1.0;
141     ),
142     490 <= wavelength <= 510,
143     (
144       R = 0.0;
145       G = 1.0;
146       B = (-(wavelength - 510)/(510 - 490))^gamma;
147     ),
148     510 <= wavelength <= 580,
149     (
150       R = ((wavelength - 510)/(580 - 510))^gamma;
151       G = 1.0;
152       B = 0.0;
153     ),
154     580 <= wavelength <= 645,
155     (
156       R = 1.0;
157       G = (-(wavelength - 645)/(645 - 580))^gamma;
158       B = 0.0;
159     ),
160     645 <= wavelength <= 750,
161     (
162       attenuation = 0.3 + 0.7*(750 - wavelength)/(750 - 645);
163       R = (1.0*attenuation)^gamma;
164       G = 0.0;
165       B = 0.0;
166     ),
167     True,
168     (
169       R = 0;
170       G = 0;
171       B = 0;
172     )];
173   Return[RGBColor[R, G, B]]
174 )
175
176 FuzzyRectangle::usage = "FuzzyRectangle[xCenter, width, ymin,
177 height, color] creates a polygon with a fuzzy edge. The polygon is
178 centered at xCenter and has a full horizontal width of width. The
179 bottom of the polygon is at ymin and the height is height. The
180 color of the polygon is color. The left edge and the right edge of
181 the resulting polygon will be transparent and the middle will be
182 colored. The polygon is returned as a list of polygons.";
183 FuzzyRectangle[xCenter_, width_, ymin_, height_, color_, intensity_:
184 1] := Module[
185   {intenseColor, nocolor, ymax, polys},
186   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 graphicsOpts = FilterRules[{opts}, Options[Graphics]];
249 Graphics[fuzzyLines,
250   graphicsOpts,
251   Background -> Black,
252   PlotRange -> {{minWave, maxWave}, {0, height}}]
253 ];
254 ];
255 End[]];
256 EndPackage[];

```

18.4 misc.m

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

```

1 BeginPackage["DavidLizarazo`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
24 RoundToSignificantFigures;
25 RoundValueWithUncertainty;
26 SecondOrderPerturbation;
27 SuperIdentity;
28
29 TextBasedProgressBar;
30 ToPythonSparseFunction;
31 ToPythonSymPyExpression;
32 TruncateBlockArray;
33

```

```

34 Begin["`Private`"];
35
36 ExtractSymbolNames;
37
38 RemoveTrailingDigits[s_String] := StringReplace[s,
39   RegularExpression["\\d+$"] -> ""];
40
41 BlockTranspose::usage="BlcockTranspose[anArray] takes a 2D array
42   with a congruent block structure and returns the transposed array
43   with the same block structure.";
44 BlockTranspose[anArray_]:=(
45   Map[Transpose, Transpose[anArray], {2}]
46 );
47
48 BlockMatrixMultiply::usage="BlockMatrixMultiply[A,B] gives the
49   matrix multiplication of A and B, with A and B having a compatible
50   block structure that allows for matrix multiplication into a
51   congruent block structure.";
52 BlockMatrixMultiply[Amat_,Bmat_]:=Module[{rowIdx,colIdx,sumIdx},
53 (
54   Table[
55     Sum[Amat[[rowIdx,sumIdx]].Bmat[[sumIdx,colIdx]],{sumIdx,1,
56 Dimensions[Amat][[2]]}],
57     {rowIdx,1,Dimensions[Amat][[1]]},
58     {colIdx,1,Dimensions[Bmat][[2]]}
59   ]
60 )
61 ];
62
63 BlockAndIndex::usage="BlockAndIndex[{w1, w2, ...}, idx] takes a
64   list of block lengths wi and an index idx. The function returns in
65   which block idx would be in a a list defined by {Range[1,w1],
66   Range[1+w1,w1+w2], ...}. The function also returns the position
67   within the bin in which the given index would be found in. The
68   function returns these two numbers as a list of two elements {blockIndex,
69   blockSubIndex}.";
70 BlockAndIndex[blockSizes_List, index_Integer]:=Module[{blockIndex,
71   accumulatedBlockSize,blockSubIndex},
72 (
73   accumulatedBlockSize = Accumulate[blockSizes];
74   If[accumulatedBlockSize[[-1]]-index<0,
75     Print["Index out of bounds"];
76     Abort[]
77   ];
78   blockIndex = Flatten[Position[accumulatedBlockSize-index,n_ /;
79     n>=0][[1]];
80   blockSubIndex = Mod[index-accumulatedBlockSize[[blockIndex]],
81   blockSizes[[blockIndex]],1];
82   Return[{blockIndex,blockSubIndex}]
83 )
84 ];
85
86 TruncateBlockArray::usage="TruncateBlockArray[blockArray,
87   truncationIndices, blockWidths] takes an array of blocks and
88   selects the columns and rows corresponding to truncationIndices.
89   The indices being given in what would be the ArrayFlatten[
90   blockArray] version of the array. The blocks in the given array
91   may be SparseArray. This is equivalent to FlattenArray[blockArray
92   ][truncationIndices, truncationIndices] but may be more efficient
93   if blockArray is sparse.";
94 TruncateBlockArray[blockArray_,truncationIndices_,blockWidths_]:=Module[
95 {
96   truncatedArray,blockCol,blockRow,blockSubCol,blockSubRow},
97 {
98   truncatedArray = Table[
99     {blockCol,blockSubCol} = BlockAndIndex[blockWidths,fullColIndex];
100    {blockRow,blockSubRow} = BlockAndIndex[blockWidths,fullRowIndex];
101    blockArray[[blockRow,blockCol]][[blockSubRow,blockSubCol]],
102    {fullRowIndex,truncationIndices},
103    {fullColIndex,truncationIndices}
104  ];
105  Return[truncatedArray]
106 }
107 ];
108
109 BlockArrayDimensionsArray::usage="BlockArrayDimensionsArray[
110   blockArray] returns the array of block sizes in a given blocked

```

```

        array.";
85  BlockArrayDimensionsArray[blockArray_]:=(
86    Map[Dimensions,blockArray,{2}]
87  );
88
89  ArrayBlocker::usage="ArrayBlocker[{anArray, blockSizes}] takes a flat
90  2d array and a congruent 2D array of block sizes, and with them
91  it returns the original array with the block structure imposed by
92  blockSizes. The resulting array satisfies ArrayFlatten[
93  blockedArray]==anArray, and also Map[Dimensions, blockedArray
94  ,{2}]==blockSizes.";
95  ArrayBlocker[{anArray_, blockSizes_}]:=Module[{rowStart,colStart,
96  colEnd,numBlocks,blockedArray,blockSize,rowEnd,aBlock,idxRow,
97  idxCol},(
98    rowStart=1;
99    colStart=1;
100   colEnd=1;
101   case=Length[Dimensions[blockSizes]];
102   Which[
103     case==3,
104     (
105       numBlocks=Length[blockSizes];
106       blockedArray=Table[(
107         blockSize=blockSizes[[idxRow, idxCol]];
108         rowEnd=rowStart+blockSize[[1]]-1;
109         colEnd=colStart+blockSize[[2]]-1;
110         aBlock=anArray[[rowStart;;rowEnd, colStart;;colEnd]];
111         colStart=colEnd+1;
112         If[idxCol==numBlocks,
113             rowStart=rowEnd+1;
114             colStart=1;
115           ];
116           aBlock
117         ),
118         {idxRow,1,numBlocks},
119         {idxCol,1,numBlocks}]];
120       ),
121       case==1,
122       (
123         expandedSizes=Table[
124           {blockSizes[[idxRow]], blockSizes[[idxCol]]},
125           {idxRow,1,Length[blockSizes]},
126           {idxCol,1,Length[blockSizes]}];
127         ];
128         numBlocks=Length[expandedSizes];
129         blockedArray=Table[((
130           blockSize=expandedSizes[[idxRow, idxCol]];
131           rowEnd=rowStart+blockSize[[1]]-1;
132           colEnd=colStart+blockSize[[2]]-1;
133           aBlock=anArray[[rowStart;;rowEnd, colStart;;colEnd]];
134           colStart=colEnd+1;
135           If[idxCol==numBlocks,
136               rowStart=rowEnd+1;
137               colStart=1;
138             ];
139             aBlock
140           ),
141           {idxRow,1,numBlocks},
142           {idxCol,1,numBlocks}]];
143         );
144       ];
145     Return[blockedArray]
146   );
147
148 ReplaceDiagonal::usage =
149 "ReplaceDiagonal[matrix, repValue] replaces all the diagonal of
150 the given array to the given value. The array is assumed to be
151 square and the replacement value is assumed to be a number. The
152 returned value is the array with the diagonal replaced. This
153 function is useful for setting the diagonal of an array to a given
154 value. The original array is not modified. The given array may be
155 sparse.";
156 ReplaceDiagonal[{matrix_, repValue_}]:=(
157   ReplacePart[matrix,
158     Table[{i, i}>>repValue, {i, 1, Length[matrix]}]];

```

```

147 Options[RoundValueWithUncertainty] = {"SetPrecision" -> False};
148 RoundValueWithUncertainty::usage = "RoundValueWithUncertainty[x, dx]
149   given a number x together with an uncertainty dx this function
150   rounds x to the first significant figure of dx and also rounds dx
151   to have a single significant figure.
152 The returned value is a list with the form {roundedX, roundedDx}.
153 The option \"SetPrecision\" can be used to control whether the
154   Mathematica precision of x and dx is also set accordingly to these
155   rules, otherwise the rounded numbers still have the original
156   precision of the input values.
157 If the position of the first significant figure of x is after the
158   position of the first significant figure of dx, the function
159   returns {0,dx} with dx rounded to one significant figure.";
160 RoundValueWithUncertainty[x_, dx_, OptionsPattern[]] := Module[
161   {xExpo, dxExpo, sigFigs, roundedX, roundedDx, returning},
162   (
163     xExpo = RealDigits[x][[2]];
164     dxExpo = RealDigits[dx][[2]];
165     sigFigs = (xExpo - dxExpo) + 1;
166     {roundedX, roundedDx} = If[sigFigs <= 0,
167       {0., N@RoundToSignificantFigures[dx, 1]},
168       N[
169       {
170         RoundToSignificantFigures[x, xExpo - dxExpo + 1],
171         RoundToSignificantFigures[dx, 1]}
172       ]
173     ];
174     returning = If[
175       OptionValue["SetPrecision"],
176       {SetPrecision[roundedX, Max[1, sigFigs]],
177        SetPrecision[roundedDx, 1]},
178       {roundedX, roundedDx}
179     ];
180     Return[returning]
181   )
182 ];
183 RoundToSignificantFigures::usage =
184   "RoundToSignificantFigures[x, sigFigs] rounds x so that it only
185   has \
186   sigFigs significant figures.";
187 RoundToSignificantFigures[x_, sigFigs_] :=
188   Sign[x]*N[FromDigits[RealDigits[x, 10, sigFigs]]];
189 RobustMissingQ[expr_] := (FreeQ[expr, _Missing] === False);
190
191 TextBasedProgressBar[progress_, totalIterations_, prefix_:""] :=
192   Module[
193     {progMessage},
194     progMessage = ToString[progress] <> "/" <> ToString[
195       totalIterations];
196     If[progress < totalIterations,
197       WriteString["stdout", StringJoin[prefix, progMessage, "\r"]
198     ],
199       WriteString["stdout", StringJoin[prefix, progMessage, "\n"]
200     ];
201   ];
202 ];
203 FirstOrderPerturbation::usage="Given the eigenVectors of a matrix A
204   (which doesn't need to be given) together with a corresponding
205   perturbation matrix perMatrix, this function calculates the first
206   derivative of the eigenvalues with respect to the scale factor of
207   the perturbation matrix. In the sense that the eigenvalues of the
208   matrix A +  $\beta$  perMatrix are to first order equal to  $\lambda_i + \delta_i \beta$ , where the  $\delta_i$  are the returned values. This
209   assuming that the eigenvalues are non-degenerate.";
210 FirstOrderPerturbation[eigenVectors_,
211   perMatrix_] := (Chop@Diagonal[
212     Conjugate@eigenVectors . perMatrix . Transpose[eigenVectors]])
213
214 SecondOrderPerturbation::usage="Given the eigenValues and
215   eigenVectors of a matrix A (which doesn't need to be given)
216   together with a corresponding perturbation matrix perMatrix, this
217   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  $\lambda_i + \delta_i \beta + \delta_i^2 \beta^2 / 2$ , where the  $\delta_i$  are the returned values. The
eigenvalues and eigenvectors are assumed to be given in the same
order, i.e. the  $i$ th eigenvalue corresponds to the  $i$ th eigenvector.
This assuming that the eigenvalues are non-degenerate.";
200 SecondOrderPerturbation[eigenValues_, eigenVectors_, perMatrix_] :=
(
201   dim = Length[perMatrix];
202   eigenBras = Conjugate[eigenVectors];
203   eigenKets = eigenVectors;
204   matV = Abs[eigenBras . perMatrix . Transpose[eigenKets]]^2;
205   OneOver[x_, y_] := If[x == y, 0, 1/(x - y)];
206   eigenDiffs = Outer[OneOver, eigenValues, eigenValues, 1];
207   pProduct = Transpose[eigenDiffs]*matV;
208   Return[2*(Total /@ Transpose[pProduct])];
209 )
210
211 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.";
212 SuperIdentity[args___] := {args};
213
214 ExtractSymbolNames[expr_Hold] := Module[
215   {strSymbols},
216   strSymbols = ToString[expr, InputForm];
217   StringCases[strSymbols,RegularExpression["\\w+"]][[2 ;;]];
218 ]
219
220 ExportToH5::usage =
221   "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.
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.";
222 Options[ExportToH5] = {"Overwrite" -> True};
223 ExportToH5[fname_String, symbols_Hold, OptionsPattern[]] :=
224   If[And[FileExistsQ[fname], OptionValue["Overwrite"]],
225     (
226       Print["File already exists, overwriting ..."];
227       DeleteFile[fname];
228     )
229   ];
230   symbolNames = ExtractSymbolNames[symbols];
231   Do[(Print[symbolName];
232     Export[fname, ToExpression[symbolName], {"Datasets", symbolName}
233   ],
234     OverwriteTarget -> "Append"
235   ), {symbolName, symbolNames}]
236 )
237
238 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.";
239 HelperNotebook[nbName_: "OUT"] :=
240 Module[{screenDims, screenWidth, screenHeight, nbWidth, leftMargin,
241   PrintToOutputNb}, (
242   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          "];
316     colIndicesPyList = StringRiffle[
317       ToPythonSymPyExpression /@ (Flatten[Normal[sparseArray["ColumnIndices"]]] - 1)), ", "];
318     (* Extract sparse array properties *)
319     rowPointers = Normal[sparseArray["RowPointers"]];
320     dimensions = Dimensions[sparseArray];
321     (*Create Python code string*)
322     pyCode = StringJoin[
323       "#!/usr/bin/env python3\n\n",
324       "from scipy.sparse import csr_matrix\n",
325       "import numpy as np\n",

```

```

312     "\n",
313     "sq = np.sqrt\n",
314     "\n",
315     "def ", funName, "(", ,
316     varList,
317     "):\n",
318     "    data = np.array([\n          ", dataPyList, "\n          ])\\
n",
319     "    indices = np.array([",
320     colIndicesPyList,
321     "])\n",
322     "    indptr = np.array([",
323     StringRiffle[ToString /@ rowPointers, ", ", "], "])\n",
324     "    shape = (" , StringRiffle[ToString /@ dimensions, ", ", "],
325     ")\n",
326     "    return csr_matrix((data, indices, indptr), shape=shape)\\
n"] ;
327     Return[pyCode];
328   )
329 ];
330
331 Options[HamTeX] = {"T2" -> False};
332 HamTeX::usage="HamTeX[numE] returns the LaTeX code for the semi-
empirical Hamiltonian for f^numE. The option \"T2\" can be used to
specify whether the T2 term should be included in the Hamiltonian
for the f^12 configuration. The default is False and the option
is ignored if the number of electrons is not 12.";
333 HamTeX[nE_, OptionsPattern[]] := (
334   tex = Which[
335     MemberQ[{1, 13}, nE],
336       "\\" \hat{H} &= \\" \zeta \\" \sum_{i=1}^n \\" \left( \\" \hat{s}_i \\" \cdot \\" \hat{l}_i \\" \right) \\" +
337       \\" \sum_{i=1}^n \\" \sum_{k=2,4,6} \\" \sum_{q=-k}^k B_q^{(k)} \\" \mathcal{C}(i)_q + \\" \epsilon",
338       nE == 2,
339         "\\" \hat{H} &= \\" \sum_{k=2,4,6} F^{(k)} \\" \hat{f}_k +
340         \\" \alpha \\" \hat{L}^2 +
341         \\" \beta \\", \\" \mathcal{C} \\" \left( \\" \mathcal{G}(2) \\" \right) +
342         \\" \gamma \\", \\" \mathcal{C} \\" \left( \\" \mathcal{S}(7) \\" \right) \\
&\\" \quad + \\" \zeta \\" \sum_{i=1}^n \\" \left( \\" \hat{s}_i \\" \cdot \\" \hat{l}_i \\" \right) \\" +
343         \\" \sum_{k=0,2,4} M^{(k)} \\" \hat{m}_k +
344         \\" \sum_{k=2,4,6} P^{(k)} \\" \hat{p}_k \\
&\\" \quad \\" \quad \\" \quad + \\" \quad \\" \quad \\" \quad + \\" \sum_{i=1}^n \\" \sum_{k=2,4,6} \\
345         \\" \sum_{q=-k}^k B_q^{(k)} \\" \mathcal{C}(i)_q + \\" \epsilon",
346       And[nE == 12, OptionValue["T2"]],
347         "\\" \hat{H} &= \\" \sum_{k=2,4,6} F^{(k)} \\" \hat{f}_k +
348         \\" T^{(2)} \\" \hat{t}_2^2 +
349         \\" \alpha \\" \hat{L}^2 +
350         \\" \beta \\", \\" \mathcal{C} \\" \left( \\" \mathcal{G}(2) \\" \right) +
351         \\" \gamma \\", \\" \mathcal{C} \\" \left( \\" \mathcal{S}(7) \\" \right) \\
&\\" \quad + \\" \quad \\" \quad + \\" \quad \\" \quad + \\" \zeta \\" \sum_{i=1}^n \\" \left( \\" \hat{s}_i \\" \cdot \\" \hat{l}_i \\" \right) \\" +
352         \\" \sum_{k=0,2,4} M^{(k)} \\" \hat{m}_k +
353         \\" \sum_{k=2,4,6} P^{(k)} \\" \hat{p}_k \\
&\\" \quad \\" \quad \\" \quad + \\" \quad \\" \quad \\" \quad + \\" \sum_{i=1}^n \\" \sum_{k=2,4,6} \\
354         \\" \sum_{q=-k}^k B_q^{(k)} \\" \mathcal{C}(i)_q + \\" \epsilon",
355       And[nE == 12, Not@OptionValue["T2"]],
356         "\\" \hat{H} &= \\" \sum_{k=2,4,6} F^{(k)} \\" \hat{f}_k +
357         \\" \alpha \\" \hat{L}^2 +
358         \\" \beta \\", \\" \mathcal{C} \\" \left( \\" \mathcal{G}(2) \\" \right) +
359         \\" \gamma \\", \\" \mathcal{C} \\" \left( \\" \mathcal{S}(7) \\" \right) \\
&\\" \quad + \\" \quad \\" \quad + \\" \quad \\" \quad + \\" \zeta \\" \sum_{i=1}^n \\" \left( \\" \hat{s}_i \\" \cdot \\" \hat{l}_i \\" \right) \\" +
360         \\" \sum_{k=0,2,4} M^{(k)} \\" \hat{m}_k +
361         \\" \sum_{k=2,4,6} P^{(k)} \\" \hat{p}_k \\
&\\" \quad \\" \quad \\" \quad + \\" \quad \\" \quad \\" \quad + \\" \sum_{i=1}^n \\" \sum_{k=2,4,6} \\
362         \\" \sum_{q=-k}^k B_q^{(k)} \\" \mathcal{C}(i)_q + \\" \epsilon",
363       True,
364         "\\" \hat{H} &= \\" \sum_{k=2,4,6} F^{(k)} \\" \hat{f}_k

```

```

376      +\\sum_{k=2,3,4,6,7,8}T^{(k)}\\hat{t}_k
377      +\\alpha \\hat{L}^2
378      +\\beta \\",\\mathcal{C}\\left(\\mathcal{G}(2)\\right)
379      +\\gamma \\",\\mathcal{C}\\left(\\mathcal{S}(7)\\right)\\\\\\
380      &\\quad\\quad\\quad + \\zeta \\sum_{i=1}^n\\left(\\
381      \\hat{s}_i \\
382      \\cdot \\hat{l}_i\\right)
383      +\\sum_{k=0,2,4}M^{(k)}\\hat{m}_k
384      +\\sum_{k=2,4,6}P^{(k)}\\hat{p}_k \\\\\\
385      &\\quad\\quad\\quad \\quad\\quad\\quad + \\sum_{i=1}^n\\sum_{\{\\
386      k=2,4,6\\}}\\sum_{q=-k}^kB_q q^{(k)}\\mathcal{C}(i)_q + \\
387      \\epsilon"
388      ];
389      Return[StringJoin[{"\\begin{aligned}\\n", tex, "\\n\\end{aligned}"}
390      }];
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
EllipsoidBoundingBox::usage = "EllipsoidBoundingBox[A,\\[Kappa]]
gives the coordinate intervals that contain the ellipsoid
determined by r^T.A.r==\\[Kappa]^2. The matrix A must be square NxN
, symmetric, and positive definite. The function returns a list
with N pairs of numbers, each pair being of the form {-x_i, x_i}."
;
EllipsoidBoundingBox[Amat_,\\[Kappa]_]:=Module[
{invAmat, stretchFactors, boundingPlanes, quad},
(
invAmat = Inverse[Amat];
stretchFactors = Sqrt[1/Diagonal[invAmat]];
boundingPlanes = DiagonalMatrix[stretchFactors].invAmat;
(* The solution is proportional to \\[Kappa] *)
boundingPlanes = \\[Kappa] * boundingPlanes;
boundingPlanes = Max /@ Transpose[boundingPlanes];
Return[{-#, #}& /@ boundingPlanes]
)
];
End[] ;
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 Tm³⁺: LaF₃”. 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 Pm³⁺: LaCl₃”. 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 LaF₃”. 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. Pr³⁺, Nd³⁺, Pm³⁺, Sm³⁺, Dy³⁺, Ho³⁺, Er³⁺, and Tm³⁺”. 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. Eu³⁺”. 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. Tb³⁺”. 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. Gd³⁺”. 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. Pm³⁺, Sm³⁺, Eu³⁺, Gd³⁺, Tb³⁺, Dy³⁺, and Ho³⁺”. 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. Pr³⁺, Nd³⁺, Er³⁺, Tm³⁺, and Yb³⁺”. In: *The Journal of Chemical Physics* 42.11 (1965). Publisher: American Institute of Physics, pp. 3797–3806.
- [Che+08] Xueyuan Chen et al. “A few mistakes in widely used data files for fn configurations calculations”. In: *Journal of luminescence* 128.3 (2008). Publisher: Elsevier, pp. 421–427.
- [Che+16] Jun Cheng et al. “Crystal-field analyses for trivalent lanthanide ions in LiYF₄”. 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 Nd³⁺: LaCl₃”. 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 Ho³⁺: LaCl₃”. 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 f⁷3”. 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 Y₂O₃. III. Intensities and g values for C₂ 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 LaCl₃: Gd³⁺”. 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, 28
spin-spin, 28
state, 3
t2Switch, 41
term, 3
three-body effective operators, 40
Tk, 40