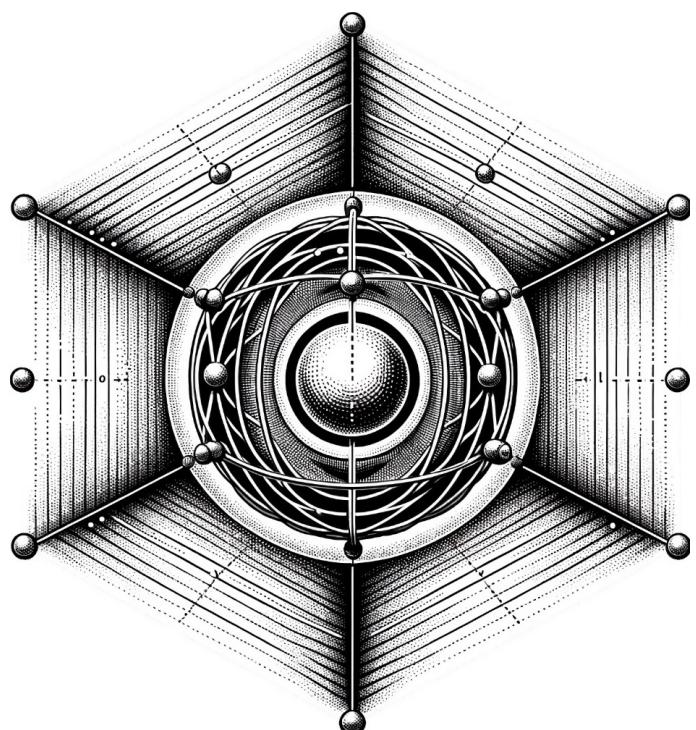


qlanth
doc version $|\alpha\rangle^{(16)}$



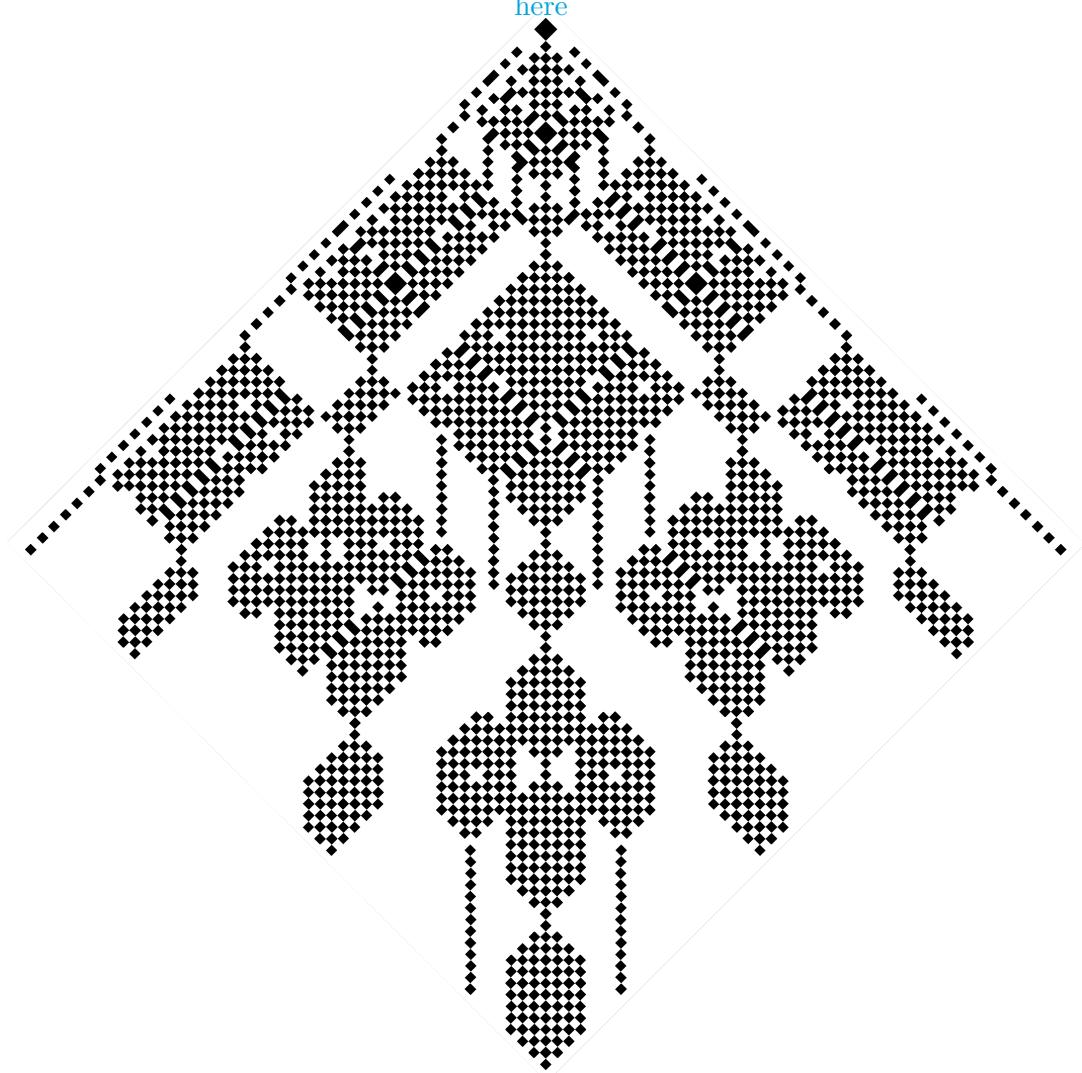
Juan David Lizarazo Ferro,
Christopher Dodson
& Rashid Zia

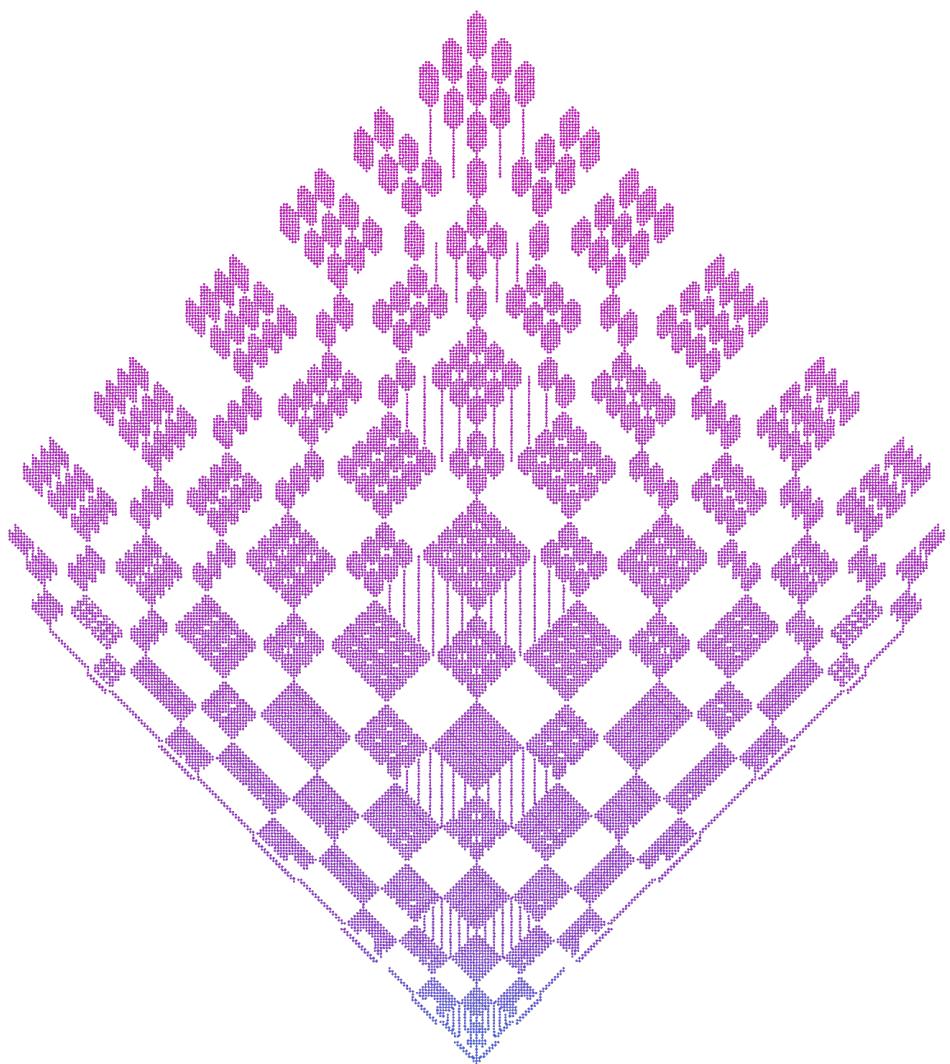
Brown University,
Department of Physics

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 17 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 16 include a summary of notation and definitions used throughout this document. Finally, Section 17 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_J\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 | 72 |
| 10 | Compiled data for $\text{LaF}_3:\text{Ln}^{3+}$ and $\text{LiYF}_4:\text{Ln}^{3+}$ | 73 |
| 11 | sparsefn.py | 76 |
| 12 | Data sources | 76 |
| 13 | Other details | 76 |
| 14 | Units | 77 |
| 15 | Notation | 78 |
| 16 | Definitions | 78 |

| | |
|---------------------------|-----------|
| 17 code | 79 |
| 17.1 qlanth.m | 79 |
| 17.2 fittings.m | 164 |
| 17.3 qplotter.m | 218 |
| 17.4 misc.m | 223 |

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.

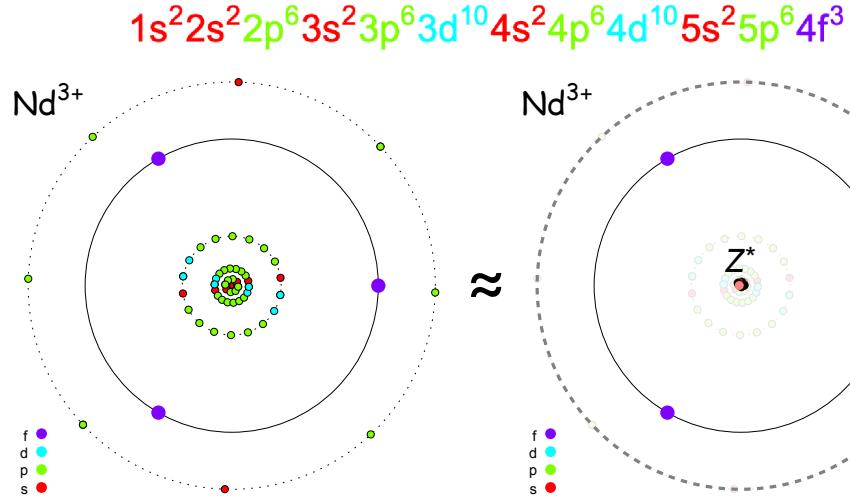


Figure 1: The trivalent neodymium ion shielded by $5s^2$ and $5p^6$ closed shells.

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⁺ | ⁵⁸ | Pr³⁺ | ⁵⁹ | Nd³⁺ | ⁶⁰ | Pm³⁺ | ⁶¹ | Sm³⁺ | ⁶² | Eu³⁺ | ⁶³ | Gd³⁺ | ⁶⁴ | Tb³⁺ | ⁶⁵ | Dy³⁺ | ⁶⁶ | Ho³⁺ | ⁶⁷ | Er³⁺ | ⁶⁸ | Tm³⁺ | ⁶⁹ | Yb³⁺ | ⁷⁰ |
| [Xe] f^1 | | [Xe] f^2 | | [Xe] f^3 | | [Xe] f^4 | | [Xe] f^5 | | [Xe] f^6 | | [Xe] f^7 | | [Xe] f^8 | | [Xe] f^9 | | [Xe] f^{10} | | [Xe] f^{11} | | [Xe] f^{12} | | [Xe] f^{13} | |
| Cerium | | Praseodymium | | Neodymium | | Promethium | | Samarium | | Europium | | Gadolinium | | Terbium | | Dysprosium | | Holmium | | Erbium | | Thulium | | 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} .

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

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

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

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

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

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.

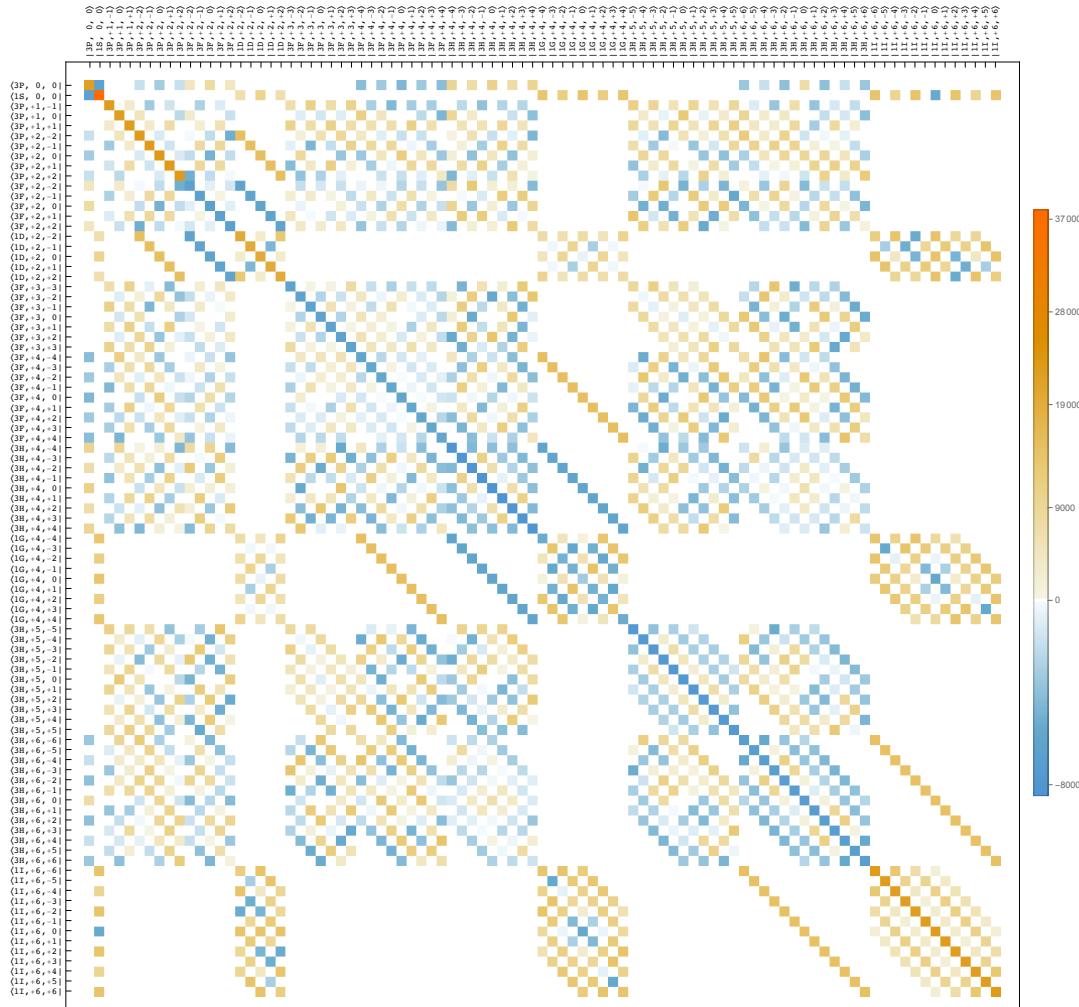


Figure 3: The matrix representation of $\hat{\mathcal{H}}$ for Pr^{3+} in LaF_3 in the $|LSJM_J\rangle$ basis.

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).

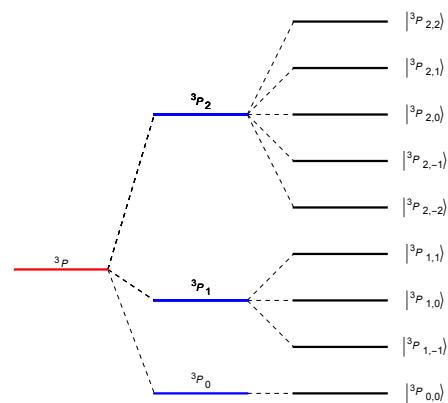


Figure 4: Levels and states associated with the 3P term in f^2 .

| |
|----------------------|
| 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)

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

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

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

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

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

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

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

\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)

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

\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 , 3K_7 , 1K_7 , 5I_8 , 3K_8 , 3K_8 , 3L_8 , 1L_8 , 3L_9 , 3M_9 , 3M_9 , 3M_9 , $^1N_{10}$

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

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

$$^2G_{9/2}, ^2H_{9/2}, ^2H_{9/2}, ^4G_{11/2}, ^4I_{11/2}, ^2H_{11/2}, ^2H_{11/2}, ^2I_{11/2}, ^4I_{13/2}, ^2I_{13/2}, ^2K_{13/2}, ^4I_{15/2}, \\ ^2K_{15/2}, ^2L_{15/2}, ^2L_{17/2}$$

f^{12} (13 LSJ levels)

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

f^{13} (2 LSJ levels)

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

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

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

```

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

```

To obtain the blocks (indexed by J) representing the Hamiltonian in the level description, the function `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[#, 
13      NumberString], ToExpression[#, #] &, list];
14    CFPTable      = ConstantArray[{}, 7];
15    CFPTable[[1]] = {{"2F", {"1S", 1}}};
16
17 (* Cleaning before processing is useful *)
18 rawText = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
19 rawLines = StringTrim/@StringSplit[rawText, "\n"];
20 rawLines = Select[rawLines, # != "" &];
21 rawLines = CleanWhitespace/@rawLines;
22 rawLines = AddSpaceBeforeMinus/@rawLines;
23
24 Do[(
25   (* the first character can be used to identify the start of a
26   block *)
27   leadChar=StringTake[line,{1}];
28   (* ...FN, N is at position 50 in that line *)
29   If[leadChar=="[",
30   (
31     configIndex=ToExpression[StringTake[line,{50}]];
32     Continue[];
33   )
34   ];
35   (* Identify which daughter term is being listed *)
36   If[StringContainsQ[line, "[DAUGHTER TERM]"],
37     daughter=StringSplit[line, "["][[1]];
38     CFPTable[[configIndex]]=Append[CFPTable[[configIndex]], {
daughter}];
39     Continue[];
40   ];
41   (* Once we get here we are already parsing a row with
42   coefficient data *)
43   lineParts = StringSplit[line, " "];

```

```

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

```

The coefficients of fractional parentage are traditionally only provided up to \underline{f}^7 (such is the case in [B1f_all.txt](#)), tabulating these beyond \underline{f}^7 would be redundant since the coefficients of fractional parentage beyond \underline{f}^7 satisfy relationships with those below \underline{f}^7 . According to [NK63]

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

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

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

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

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

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

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

2.5 Going beyond \underline{f}^7

In most cases all matrix elements in `q1anth` 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 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 $\langle \alpha LSJM_J | \hat{h} | \alpha' L'S'J'M'_J \rangle$ so that for each interaction an association with keys of the form $\{J, J'\}$ is created. The values being rectangular 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) .

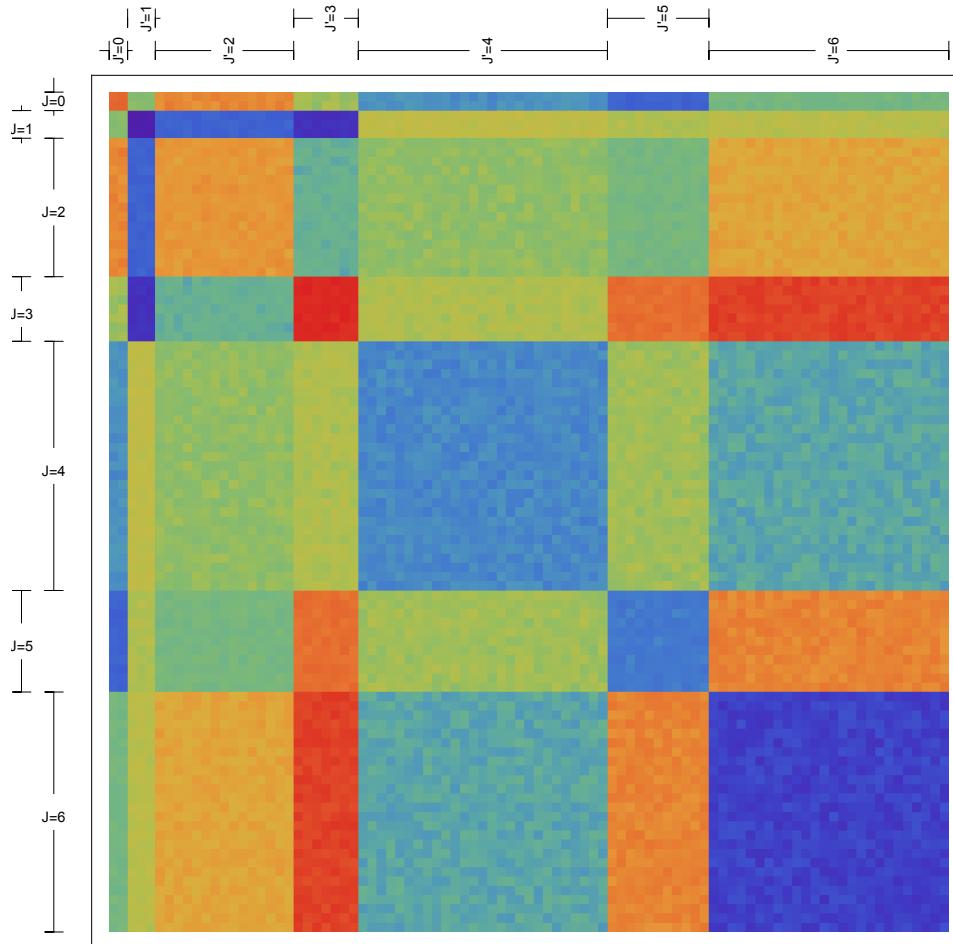


Figure 5: The J-J' block structure for f^2

In **qlanth** these blocks are put together by the function **JJBBlockMatrix** 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 = NKSLJMp[[1]];
16     MJ = NKSLJM[[3]];
17     MJp = NKSLJMp[[3]];
18     subKron = (
19       KroneckerDelta[J, Jp] *
20       KroneckerDelta[MJ, MJp]
21     );
22     matValue =
23     If[subKron == 0,
24       0,
25       (
26         ElectrostaticTable[{numE, SLterm, SpLpterm}] +
27         ElectrostaticConfigInteraction[{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^n_i configuration. The matrix is
3   returned as a SparseArray.
4 The function admits an optional parameter ''FilenameAppendix'', which
5   can be used to modify the filename to which the resulting array is
6   exported to.
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
167 (* Once the block form is created flatten it *)
168 If[Not[OptionValue["ReturnInBlocks"]],
169   (blockHam = ArrayFlatten[blockHam];
170   blockHam = ReplaceInSparseArray[blockHam, params];
171   ),
172   (blockHam = Map[ReplaceInSparseArray[#, params]&, blockHam
173 ,{2}]);
174 ];
175
176 If[OptionValue["IncludeZeeman"],
177   (
178   PrintTemporary["Including Zeeman terms ..."];
179   {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "
180 ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
181   blockHam += - teslaToKayser * (Bx * magx + By * magy + Bz *
182   magz);
183   )
184 ];
185
186 Return[blockHam];
187 ]
188 ];

```

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 2.6](#) 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

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

shell structure of the atom, the lanthanide ions “see” the nuclear charge as shielded by a xenon core. Since every closed shell is a singlet, having spherical symmetry, these shields are like spherical shells surrounding the nucleus.

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

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

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

This term is the first we will not discard. Calculating this term for the f^n configurations was one of the contributions from Slater, as such the parameters we use to write it up are called *Slater integrals*. After the analysis from Slater, Giulio Racah contributed further to the analysis of this term [Rac49]. The insight that Racah had was that if in a given operator one identifies the parts in it that transform accordingly to the different symmetry groups present in the problem, then calculating the necessary matrix element in all f^n configurations can be greatly simplified.

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

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

where

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

```

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

¹⁷ An LS-reduced matrix element is ...

```

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

```

```

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

```

```

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

```

```

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

```

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

```

3.4 $\hat{\mathcal{H}}_{\text{s:o}}$: spin-orbit

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

```

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

```

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

```

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

```

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

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

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

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

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

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

In `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  ];

```

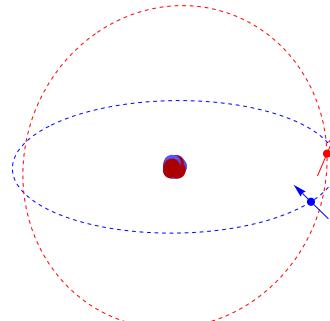
3.6 $\hat{\mathcal{H}}_{\text{s:s-oo}}$: spin-spin and spin-other-orbit

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*

$$\begin{aligned} \hat{\mathcal{H}}_{\text{s:oo}} = & \sum_{i \neq j} \sum_k \sqrt{(k+1)(2k+1)(2k+3)(2k+5)} \times \\ & \left[\left\{ \hat{w}_i^{(0,k+1)} \hat{w}_j^{(1,k)} \right\}^{(11)0} \left\{ m^{(k-1)} \langle \underline{\ell} | \mathcal{C}^{(k+1)} | \underline{\ell} \rangle^2 + 2m^{(k)} \langle \underline{\ell} | \mathcal{C}^{(k)} | \underline{\ell} \rangle^2 \right\} + \right. \\ & \left. \left\{ \hat{w}_i^{(0,k)} \hat{w}_j^{(1,k+1)} \right\}^{(11)0} \left\{ m^{(k)} \langle \underline{\ell} | \mathcal{C}^{(k)} | \underline{\ell} \rangle^2 + 2m^{(k-1)} \langle \underline{\ell} | \mathcal{C}^{(k+1)} | \underline{\ell} \rangle^2 \right\} \right]. \end{aligned} \quad (36)$$

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"]
25                ], 0.1]|>}],
26              {template3[<|"speed"->Round[QuantityMagnitude[Now-
27                startTime, "ms"]/(numiter), 0.01]|>}],
28              {ProgressIndicator[Dynamic[numiter], {1, totalIters
29                }]}},
30              Frame -> All,
31              Full,
32              Alignment -> Center]
33            ]];
34      ];
35      T22Table = <||>;
36      startTime = Now;
37      numiter = 1;
38      Do[
39        (
40          numiter+= 1;
41          T22Table[{numE, SL, SpLp}] = Which[
42            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. This is the
  operator corresponding to the inter-electron between spin.
2 It does this by using equation (4) of 'Judd, BR, HM Crosswhite, and
  Hannah Crosswhite. 'Intra-Atomic Magnetic Interactions for f
  Electrons.' Physical Review 169, no. 1 (1968): 130.'
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])

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

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

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

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

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

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 LoadChen[].";
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 MOv, 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         {MOv, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SL,
29         SLP}]["wrong"];
30         total = (
31             phase * sixjay *
32             (
33                 MOv*MO + 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 )

```

```

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   subfolder for the folder in which this package is in. It requires
7   ReducedV1kTable to be defined.";
8 Options[GenerateSpinOrbitTable] = {"Export" -> True};
9 GenerateSpinOrbitTable[nmax_Integer:7, OptionsPattern[]] := Module[
10   {numE, J, SL, SpLp, exportFname},
11   (
12     SpinOrbitTable =
13       Table[
14         {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
15         {numE, 1, nmax},
16         {J, MinJ[numE], MaxJ[numE]},
17         {SL, Map[First, AllowedNKSLforJTerms[numE, J]]},
18         {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
19       ];
20     SpinOrbitTable = Association[SpinOrbitTable];
21
22     exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
23     If[OptionValue["Export"],
24     (
25       Print["Exporting to file " <> ToString[exportFname]];
26       Export[exportFname, SpinOrbitTable];
27     )
28   ];
29   Return[SpinOrbitTable];
30 )

```

```

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 );

```

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 $\hat{\mathcal{T}}^{(22)}$ 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-
3     other-orbit + electrostatically-correlated-spin-orbit) operator.
4     It returns an association where the keys are of the form {numE, SL
5     , SpLp, J}. If the option ''Export'' is set to True then the
6     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[GenerateSpinSpinTable] = {"Export" -> False};
11 GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
12   (
13     SpinSpinTable = <||>;
14     PrintTemporary[Dynamic[numE]];
15     Do[
16       SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp,
17         , J]);
18       {numE, 1, nmax},
19       {J, MinJ[numE], MaxJ[numE]},
20       {SL, First /@ AllowedNKSLforJTerms[numE, J]},
21       {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
22     ];
23     If[OptionValue["Export"],
24       (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
25       Export[fname, SpinSpinTable];
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 ''

```

```

4 ";
5 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
  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    ];
13      template2 = StringTemplate["`remtime` min remaining"];template3
14      = StringTemplate["Iteration speed = `speed` ms/it"];
15      template4 = StringTemplate["Time elapsed = `runtime` min"];
16      progBar = PrintTemporary[
17        Dynamic[
18          Pane[
19            Grid[{{Superscript["f", numE]}, {
20              template1[<|"numiter" -> numiter, "totaliter" ->
21                totalIters |>]},
22              {template4[<|"runtime" -> Round[QuantityMagnitude[
23                UnitConvert[(Now - startTime), "min"]], 0.1] |>}],
24              {template2[<|"remtime" -> Round[QuantityMagnitude[
25                UnitConvert[(Now - startTime)/(numiter)*(totalIters - numiter), "min"]
26                ], 0.1] |>]},
27              {template3[<|"speed" -> Round[QuantityMagnitude[Now -
28                startTime, "ms"]/(numiter), 0.01] |>]},
29              {ProgressIndicator[Dynamic[numiter], {1, totalIters
30                }]}},
31              Frame -> All],
32              Full,
33              Alignment -> Center]
34            ]
35          ];
36      ];
37      T22Table = <||>;
38      startTime = Now;
39      numiter = 1;
40      Do[
41        (
42          numiter += 1;
43          T22Table[{numE, SL, SpLp}] = Which[
44            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. This is the
  operator corresponding to the inter-electron between spin.
2 It does this by using equation (4) of 'Judd, BR, HM Crosswhite, and
  Hannah Crosswhite. 'Intra-Atomic Magnetic Interactions for f
  Electrons.' Physical Review 169, no. 1 (1968): 130.'
3 ";
4 ReducedT22infn[numE_, SL_, SpLp_] := Module[
5   {spin, orbital, t, idx1, idx2, S, L,
6   Sp, Lp, cfpSL, cfpSpLp, parentSL,
7   parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
8   (
9     {spin, orbital} = {1/2, 3};
10    {S, L} = FindSL[SL];
11    {Sp, Lp} = FindSL[SpLp];
12    t = 2;
13    cfpSL = CFP[{numE, SL}];
14    cfpSpLp = CFP[{numE, SpLp}];
15    Tnkk = Sum[(
16      parentSL = cfpSL[[idx2, 1]];
17      parentSpLp = cfpSpLp[[idx1, 1]];
18      {Sb, Lb} = FindSL[parentSL];
19      {Sbp, Lbp} = FindSL[parentSpLp];
20      phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
21      (
22        phase *
23        cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
24        SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
25        SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
26        T22Table[{numE - 1, parentSL, parentSpLp}]
27      )
28    ),
29    {idx1, 2, Length[cfpSpLp]},
30    {idx2, 2, Length[cfpSL]}
31  ];
32  Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
33  Return[Tnkk];
34 )
35 ];

```

```

1 ReducedT22inf2::usage = "ReducedT22inf2[SL, SpLp] returns the reduced
  matrix element of the scalar component of the double tensor T22
  for the terms SL, SpLp in f^2.
2 Data used here for m0, m2, m4 is from Table I of Judd, BR, HM
  Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
  Interactions for f Electrons. Physical Review 169, no. 1 (1968):
  130.
3 ";
4 ReducedT22inf2[SL_, SpLp_] := Module[
5   {statePosition, PsiPsipStates, m0, m2, m4, Tk2m},
6   (
7     T22inf2 = <|

```

```

8  {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
9  {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
10 {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
11 {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
12 {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
13 |>;
14 Which [
15   MemberQ[Keys[T22inf2], {SL, SpLp}],
16   Return[T22inf2[{SL, SpLp}]],
17   MemberQ[Keys[T22inf2], {SpLp, SL}],
18   Return[T22inf2[{SpLp, SL}]],
19   True,
20   Return[0]
21 ]
22 )
23 ];

```

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

```

```

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 ];

```

```

1 S00andECSO::usage = "S00andECSO[n, SL, SpLp, J] returns the matrix
  element <|SL,J|spin-other-orbit> 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 Rigurously 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 $(4f)^{n\pm 1}(n'l')^{\mp 1}$ 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 *)

```

```

76 ThreeBodyTable = <||>;
77 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
124 ThreeBodyTables = Association[ThreeBodyTables];
125 If[OptionValue["Export"],
126 (
127   threeBodyTablefname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
128   Export[threeBodyTablefname, ThreeBodyTable];
129   threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
130   Export[threeBodyTablesfname, ThreeBodyTables];
131 )
132 ];
133 Return[{ThreeBodyTable, ThreeBodyTables}];
134

```

```

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

```

```

12 (* Parse table 1 from Judd 1984 *)
13 judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"}];
14
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];

```

```

82 juddOperators[{3, "t_{2}"}] = t20p;
83
84 (*Special case of t11 in f3*)
85 t11 = juddOperators[{3, "t_{11}"}];
86 eβprimeOp = juddOperators[{3, "e_{\beta}^{\prime}"}];
87 t11primeOp = (# -> (t11[#] + Sqrt[3/385] eβprimeOp[#])) & /@ Keys[
88 t11];
89 t11primeOp = Association[t11primeOp];
90 juddOperators[{3, "t_{11}^{\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];

```

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

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

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 $\hat{\mathbf{u}}^{(k)}$ 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 $\hat{\mathbf{u}}^{(k)}$ is defined as the tensor operator having the following reduced matrix elements (written in terms of the triangular delta, see section on notation):

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

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

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

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

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

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

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

$$(62)$$

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

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

$$(63)$$

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

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

$$(64)$$

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

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

$$(65)$$

From the $\langle \underline{\ell} \| \hat{\mathcal{C}}^{(k)} \| \underline{\ell} \rangle$, 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 bracket is determined by the parity of $\mathcal{C}_q^{(k)}$, and since the parity of $\mathcal{C}_q^{(k)}$ is $(-1)^k$ then for the bracket 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:

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

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



```

1 CrystalField::usage = "CrystalField[n, NKSL, J, M, NKSLp, Jp, Mp]
2      gives the general expression for the matrix element of the crystal
3      field Hamiltonian parametrized with Bqk and Sqk coefficients as a
4      sum over spherical harmonics Cqk.
5 Sometimes this expression only includes Bqk coefficients, see for
6      example eqn 6-2 in Wybourne (1965), but one may also split the
7      coefficient into real and imaginary parts as is done here, in an
8      expression that is patently Hermitian.";
9 CrystalField[numE_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := (
10   Sum[
11     (
12       cqk = Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp];
13       cmqk = Cqk[numE, -q, k, NKSL, J, M, NKSLp, Jp, Mp];
14       Bqk[q, k] * (cqk + (-1)^q * cmqk) +
15       I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
16     ),
17     {k, {2, 4, 6}}],
```

```

12 {q, 0, k}
13 ]
14 )

```

```

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

```

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 | \mathcal{C}_s | \mathcal{C}_{1h} | \mathcal{C}_2 | \mathcal{C}_{2h} |
| \mathcal{C}_{2v} | \mathcal{D}_2 | \mathcal{D}_{2h} | \mathcal{S}_4 | \mathcal{C}_4 |
| \mathcal{C}_{4h} | \mathcal{D}_{2d} | \mathcal{C}_{4v} | \mathcal{D}_4 | \mathcal{D}_{4h} |
| \mathcal{C}_3 | \mathcal{S}_6 | \mathcal{C}_{3h} | \mathcal{C}_{3v} | \mathcal{D}_3 |
| \mathcal{D}_{3d} | \mathcal{D}_{3h} | \mathcal{C}_6 | \mathcal{C}_{6h} | \mathcal{C}_{6v} |
| \mathcal{D}_6 | \mathcal{D}_{6h} | \mathcal{T} | \mathcal{T}_h | \mathcal{T}_d |
| \mathcal{O} | \mathcal{O}_h | | | |

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

Crystal field expressions

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

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

$$\begin{aligned} \hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_{1h}) = & B_0^{(2)} \mathcal{C}_0^{(2)} + B_2^{(2)} \mathcal{C}_2^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + \left(B_2^{(4)} + iS_2^{(4)} \right) \mathcal{C}_2^{(4)} \\ & + \left(B_4^{(4)} + iS_4^{(4)} \right) \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left(B_2^{(6)} + iS_2^{(6)} \right) \mathcal{C}_2^{(6)} + \left(B_4^{(6)} + iS_4^{(6)} \right) \mathcal{C}_4^{(6)} \\ & + \left(B_6^{(6)} + iS_6^{(6)} \right) \mathcal{C}_6^{(6)} \quad (69) \end{aligned}$$

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

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

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

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_2) = B_0^{(2)}C_0^{(2)} + B_2^{(2)}C_2^{(2)} + B_0^{(4)}C_0^{(4)} + B_2^{(4)}C_2^{(4)} + B_4^{(4)}C_4^{(4)} + B_0^{(6)}C_0^{(6)} + B_2^{(6)}C_2^{(6)} + B_4^{(6)}C_4^{(6)} + B_6^{(6)}C_6^{(6)} \quad (73)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{D}_{2h}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_2^{(2)} \mathcal{C}_2^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_2^{(4)} \mathcal{C}_2^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + B_2^{(6)} \mathcal{C}_2^{(6)} + B_4^{(6)} \mathcal{C}_4^{(6)} + B_6^{(6)} \mathcal{C}_6^{(6)} \quad (74)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{S}_4) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left(B_4^{(6)} + i \mathcal{S}_4^{(6)} \right) \mathcal{C}_4^{(6)} \quad (75)$$

$$\hat{\mathcal{H}}_{\text{cf}}(\mathcal{C}_4) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left(B_4^{(6)} + i S_4^{(6)} \right) \mathcal{C}_4^{(6)} \quad (76)$$

$$\mathcal{H}_{\text{cf}}(\mathcal{C}_{4h}) = B_0^{(2)} \mathcal{C}_0^{(2)} + B_0^{(4)} \mathcal{C}_0^{(4)} + B_4^{(4)} \mathcal{C}_4^{(4)} + B_0^{(6)} \mathcal{C}_0^{(6)} + \left(B_4^{(6)} + i S_4^{(6)} \right) \mathcal{C}_4^{(6)} \quad (77)$$

$$\mathcal{H}_{\text{cf}}(\mathcal{D}_{2d}) = B_0^{(2)}C_0^{(2)} + B_0^{(4)}C_0^{(4)} + B_4^{(4)}C_4^{(4)} + B_0^{(6)}C_0^{(6)} + B_4^{(6)}C_4^{(6)} \quad (78)$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(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

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

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

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

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

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

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

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
6   the Wigner-Eckart theorem. See TASS eqn 11.15.
7 ";
8 Options[JJBlockMagDip]={"Sparse"->True};
9 JJBlockMagDip[numE_, braJ_, ketJ_, OptionsPattern[]]:=Module[
10 {braSLJs, ketSLJs,
11 braSLJ, ketSLJ,
12 braSL, ketSL,
13 braS, braL,
14 ketS, ketL,
15 braMJ, ketMJ,
16 matValue, magMatrix,
17 summand1, summand2,
18 threejays},
19 (
20   braSLJs = AllowedNKSLJMforJTerms[numE, braJ];
21   ketSLJs = AllowedNKSLJMforJTerms[numE, ketJ];
22   magMatrix = Table[
23     braSL = braSLJ[[1]];
24     ketSL = ketSLJ[[1]];
25     {braS, braL} = FindSL[braSL];
26     {ketS, ketL} = FindSL[ketSL];
27     braMJ = braSLJ[[3]];
28     ketMJ = ketSLJ[[3]];
29     summand1 = If[Or[braJ != ketJ,
30                      braSL != ketSL],
31                   0,
32                   Sqrt[braJ*(braJ+1)*TP0[braJ]]
33 ];
34     (* looking at the string includes checking L=L', S=S', and \
35 alpha=alpha'*)
36     summand2 = If[braSL!=ketSL,
37                   0,
38                   (gs-1) *
39                     Phaser[braS+braL+ketJ+1] *
40                     Sqrt[TP0[braJ]*TP0[ketJ]] *
41                     SixJay[{braJ,1,ketJ},{braS,braL,bras}] *
42                     Sqrt[bras(braS+1)TP0[braS]]
43 ];
44     matValue = summand1 + summand2;
45     (* We are using the Racah convention for red matrix elements in
46 Wigner-Eckart *)
47     threejays = (ThreeJay[{braJ, -braMJ}, {1, #}, {ketJ, ketMJ}] &
48 /@ {-1, 0, 1};
49     threejays *= Phaser[braJ-braMJ];
50     matValue = - 1/2 * threejays * matValue;
51     matValue,
52     {braSLJ, braSLJs},
53     {ketSLJ, ketSLJs}
54   ];
55   If[OptionValue["Sparse"],
56     magMatrix = SparseArray[magMatrix]
57   ];
58   Return[magMatrix];
59 )
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]
2   returns the matrix representation of the operator - 1/2 (L + gs S)
3   in the f^numE configuration. The function returns a list with
4   three elements corresponding to the x,y,z components of this
5   operator. The option ''FilenameAppendix'' can be used to append a
6   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  If[OptionValue["ReturnInBlocks"],
30    (
31      opMinus = Map[#[[1]] &, blockOp, {4}];
32      opZero = Map[#[[2]] &, blockOp, {4}];
33      opPlus = Map[#[[3]] &, blockOp, {4}];
34      opX = (opMinus - opPlus)/Sqrt[2];
35      opY = I (opPlus + opMinus)/Sqrt[2];
36      opZ = opZero;
37    ),
38    blockOp = ArrayFlatten[blockOp];
39    opMinus = blockOp[[;; , ; , 1]];
40    opZero = blockOp[[;; , ; , 2]];
41    opPlus = blockOp[[;; , ; , 3]];
42    opX = (opMinus - opPlus)/Sqrt[2];
43    opY = I (opPlus + opMinus)/Sqrt[2];
44    opZ = opZero;
45  ];
46  Return[{opX, opY, opZ}];
47 )
];

```

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 `constants.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^n_i configuration. The matrix is
   returned as a SparseArray.
2 The function admits an optional parameter ''FilenameAppendix'', which
   can be used to modify the filename to which the resulting array is
   exported to.

```

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 
```

```

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
167 (* Once the block form is created flatten it *)
168 If [Not[OptionValue["ReturnInBlocks"]],
169 (blockHam = ArrayFlatten[blockHam];
170 blockHam = ReplaceInSparseArray[blockHam, params];
171 ),
172 (blockHam = Map[ReplaceInSparseArray[#, params]&, blockHam
,{2}]);
173 ];
174
175 If [OptionValue["IncludeZeeman"],
176 (
177     PrintTemporary["Including Zeeman terms ..."];
178     {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "
ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
179     blockHam += - teslaToKayser * (Bx * magx + By * magy + Bz *
magz);
180 )
181 ];
182 Return [blockHam];
183 )
184 ];

```

4 Coordinate system

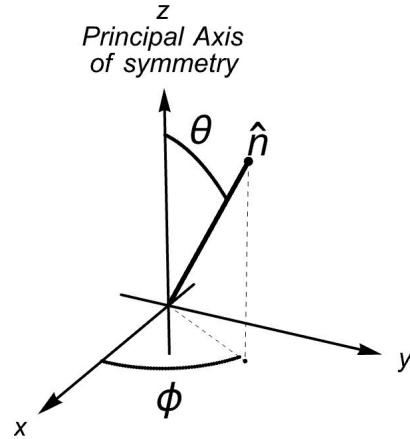
Before adding interactions lacking spherical symmetry, the orientation of the coordinate system is irrelevant. At the point when the crystal field is added, this orientation becomes relevant in the sense that only certain orientations of the coordinate system yield an expression for the crystal field 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

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.



illuminated with a broad spectrum light source, and the wavelength dependent absorption 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:

1. **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.

2. **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:

1. **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.

2. **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 this dissertation the uncertainty in the experimental data was assumed to be constant and equal to 1 cm^{-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 $\hat{\mathbf{p}} = \{\hat{p}_x, \hat{p}_y, \hat{p}_z\}$ the magnetic dipole operator, the line strength between two eigenstates $|\psi\rangle$ and $|\psi'\rangle$ is defined as (see for example equation 14.31 in [Cow81])

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

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 $\hat{\mathcal{S}}(\psi, \psi')$.

```

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
  line strength is calculated. The default, All, calculates the line
  strength for all states. A second option for this is to provide
  an index labelling a specific state, in which case only the line
  strengths between that state and all the others are computed.

```

```

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

```

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.
11 The returned array should be interpreted in the eigenbasis of the
12   Hamiltonian. As such the element AMD[[i,i]] corresponds to the
13   transition rate (or the radiative lifetime, depending on options)
14   between eigenstates  $|i\rangle$  and  $|j\rangle$ .

```

```

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

```

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.
7 eigenSys is a list of lists with two elements, in each list the first
8   element is the energy and the second one the corresponding
9   eigenvector.
10 The energy unit assumed in eigenSys is Kayser.

```

```

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

```

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 $|\alpha' S'L'J'\rangle$, the oscillator strength between them is approximated as [Jud62]

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

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.

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-Ofelt theory is implemented with help of the functions **JuddOfeltUkSquared** 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
8   f^n configuration.
9 - levelLabels : A list with the labels of the major components of
10  the level eigenstates.
11 - LevelUkSquared : An association with the squared matrix elements
12  of the Uk operators in the level eigenbasis. The keys being {2, 4,
13  6} corresponding to the rank of the Uk operator. The basis in
14  which the matrix elements are given is the one corresponding to
15  the level eigenstates given in eigenSys and whose major SLJ
16  components are given in levelLabels. The matrix is symmetric and
17  given as a SymmetrizedArray.
18 The function admits the following options:
19 ''PrintFun'' : A function that will be used to print the progress
20  of the calculations. The default is PrintTemporary.";
21 Options[JuddOfeltUkSquared] = {"PrintFun" -> PrintTemporary};
22 JuddOfeltUkSquared[numE_, params_, OptionsPattern[]] := Module[
23   {eigenChanger, numEH, basis, eigenSys,
24   Js, Ukm, LevelUkSquared, kRank,
25   S, L, Sp, Lp, J, Jp, phase,
26   braTerm, ketTerm, levelLabels,
27   eigenVecs, majorComponentIndices},
28   (
29     If[Not[ValueQ[ReducedUkTable]],
30       LoadUk[]
31     ];
32     numEH = Min[numE, 14-numE];
33     PrintFun = OptionValue["PrintFun"];
34     PrintFun["> Calculating the levels for the given parameters ..."];
35     {basis, majorComponents, eigenSys} = LevelSolver[numE, params];
36     (* The change of basis matrix to the eigenstate basis *)
37     eigenChanger = Transpose[Last /@ eigenSys];
38     PrintFun["Calculating the matrix elements of Uk in the physical
39     coupling basis ..."];
40     LevelUkSquared = <||>;
41     Do[(
42       Ukm = Table[(
43         {S, L} = FindSL[braTerm[[1]]];
44         J = braTerm[[2]];
45         Jp = ketTerm[[2]];
46         {Sp, Lp} = FindSL[ketTerm[[1]]];
47         phase = Phaser[S + Lp + J + kRank];
48         Simplify @ (
49           phase *
50           Sqrt[TPO[J]*TPO[Jp]] *
51           SixJay[{J, Jp, kRank}, {Lp, L, S}] *
52           ReducedUkTable[{numEH, 3, braTerm[[1]], ketTerm[[1]], kRank}]
53         )
54       ),
55       {braTerm, basis},
56       {ketTerm, basis}
57     ];
58     Ukm = (Transpose[eigenChanger] . Ukm . eigenChanger)^2;
59   ]
60 
```

```

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
7     equal to the three Judd-Ofelt intensity parameters {\[CapitalOmega]2,
8     \[CapitalOmega]4, \[CapitalOmega]6} and corresponding values
9     in cm^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   (
56     PrintFun = OptionValue["PrintFun"];
57     nRef = OptionValue["RefractiveIndex"];
58     PrintFun["Calculating the Uk^2 matrix elements for the given
59     parameters ..."];
60     {basis, eigenSys, levelLabels, LevelUkSquared} =

```

```

32 JuddOfeltUkSquared[numE, levelParams, "PrintFun" -> PrintFun];
33 eigenEnergies = First/@eigenSys;
34 (* converted to cm^-2 *)
35 const = (8\[Pi]^2)/3 me/hPlanck * 10^(-4);
36 energyDiffs = Transpose@Outer[Subtract,eigenEnergies,
eigenEnergies];
37 (* since energies are assumed in Kayser, speed of light needs to
be in cm/s, so that the frequencies are in 1/s *)
38 transitionFrequencies = energyDiffs*cLight*100;
39 (* grab the J for each level *)
40 levelJs = #[[2]] & /@ eigenSys;
41 oStrengthArray =
  juddOfeltParams[\[CapitalOmega]2]*LevelUkSquared[2]+
  juddOfeltParams[\[CapitalOmega]4]*LevelUkSquared[4]+
  juddOfeltParams[\[CapitalOmega]6]*LevelUkSquared[6]
);
42 oStrengthArray = Abs@(
  const * transitionFrequencies *
  oStrengthArray);
43 (* it is necessary to divide each oscillator strength by the
degeneracy of the initial level *)
44 oStrengthArray = MapIndexed[1/(2 levelJs[[#2[[1]]]]+1) #1 &,
oStrengthArray,{2}];
45 (* including the effects of the refractive index *)
46 fieldCorrectionType = OptionValue["LocalFieldCorrection"];
47 Which[
48   nRef === 1,
49   True,
50   NumberQ[nRef],
51   (
52     \[Chi] = Which[
53       fieldCorrectionType == "VirtualCavity",
54       (
55         (nRef^2 + 2) / 3 )^2
56       ),
57       fieldCorrectionType == "EmptyCavity",
58       (
59         (3 * nRef^2 / (2 * nRef^2 + 1) )^2
60       )
61     ];
62   \[Chi]OverN = \[Chi] / nRef;
63   oStrengthArray = \[Chi]OverN * oStrengthArray;
64   (* the refractive index participates differently in
absorption and in emission *)
65   aFunction = If[#2[[1]] > #2[[2]], #1 * nRef^2, #1]&;
66   oStrengthArray = MapIndexed[aFunction, oStrengthArray, {2}];
67   ),
68   True,
69   (
70     wavelengthsInNM = Abs[1 / energyDiffs] * 10^7;
71     nRefs = Map[nRef, wavelengthsInNM];
72     Print["Calculating the oscillator strengths for the given
refractive index ..."];
73     \[Chi] = Which[
74       fieldCorrectionType == "VirtualCavity",
75       (
76         (nRefs^2 + 2) / 3 )^2
77       ),
78       fieldCorrectionType == "EmptyCavity",
79       (
80         (3 * nRefs^2 / (2*nRefs^2 + 1) )^2
81       )
82     ];
83     \[Chi]OverN = \[Chi] / nRefs;
84     oStrengthArray = \[Chi]OverN * oStrengthArray
85   )
86   ];
87   Return[{basis, eigenSys, levelLabels, oStrengthArray}];
88 )
89 ]
90 );
91 ]
92 ];

```

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 level
4   description is determined by levelParams. The refractive index of
5   the medium is relevant, but here it is assumed to be 1, this can
6   be changed through the option 'RefractiveIndex'. eigenSys must
7   consist of a lists of lists with three elements: the first element
8   being the energy of the level, the second element being the J of
9   the level, and the third element being the eigenvector of the
10  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      - levelLabels : A list with the labels of the major components of
8      the calculated levels.
9      - magDipoleOstrength : A square array whose elements represent
10     the magnetic dipole oscillator strengths between the levels given
11     in eigenSys such that the element magDipoleOstrength[[i,j]] is the
12     oscillator strength between the levels |Subscript[\[Psi], i]> and
13     |Subscript[\[Psi], j]>. In this array the elements below the
14     diagonal represent emission oscillator strengths, and elements
15     above the diagonal represent absorption oscillator strengths. The
16     emission oscillator strengths are negative. The oscillator
17     strength is a dimensionless quantity.
18
19 The function admits the following option:
20   ''RefractiveIndex'' : The refractive index of the medium where
21   the transitions are taking place. This may be a number or a
22   function. If a number then the oscillator strengths are calculated
23   assuming a wavelength-independent refractive index as given. If a
24   function then the refractive indices are calculated accordingly
25   to the vaccum wavelength of each transition (the function must
26   admit a single argument equal to the wavelength in nm). The
27   default is 1.
28
29 For reference see equation (27.8) in Rudzikas (2007). The
30 expression for the line strength is the simplest when using atomic
31 units, (27.8) is missing a factor of  $\alpha^2$ .;
32 Options[LevelMagDipoleOscillatorStrength]={
33   "RefractiveIndex" -> 1
34 };
35 LevelMagDipoleOscillatorStrength[eigenSys_, numE_, OptionsPattern[]]
36   := Module[
37     {eigenEnergies, eigenVecs, levelJs,
38      energyDiffs, magDipoleOstrength, nRef,
39      wavelengthsInNM, nRefs, degenDivisor},
40     (
41       basis           = BasisLSJ[numE];
42       eigenEnergies = First/@eigenSys;
43       nRef          = OptionValue["RefractiveIndex"];
44       eigenVecs     = Last/@eigenSys;
45       levelJs       = #[[2]]&/@eigenSys;
46       energyDiffs   = -Outer[Subtract,eigenEnergies,eigenEnergies];
47       energyDiffs *= kayserToHartree;
48       magDipoleOstrength = LevelMagDipoleLineStrength[eigenSys, numE, "Units"->"Hartree"];
49       magDipoleOstrength = 2/3 * alphaFine^2 * energyDiffs *
50       magDipoleOstrength;
51       degenDivisor    = #1 / ( 2 * levelJs[[#2[[1]]]] + 1 ) &;
52       magDipoleOstrength = MapIndexed[degenDivisor, magDipoleOstrength,
53       {2}];
54       Which[nRef==1,
55         True,
56         NumberQ[nRef],
57         (
58           magDipoleOstrength = nRef * magDipoleOstrength;
59         ),
60         True,
61         (
62           wavelengthsInNM = Abs[kayserToHartree / energyDiffs] *
63           10^7;
64           nRefs            = Map[nRef, wavelengthsInNM];
65           magDipoleOstrength = nRefs * magDipoleOstrength;
66         )
67       ];
68       Return[{basis, eigenSys, magDipoleOstrength}];
69     )
70   ];
71 ];

```

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

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

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

```

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  vaccum 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   If[units == "SI",
69   (
70     spontaneousRatesInSI = 1/hartreeTime *
71     spontaneousRatesInHartree;
72     Return[SparseArray@spontaneousRatesInSI];
73   ),
74   Return[SparseArray@spontaneousRatesInHartree];
75 ];
76 ];

```

```
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 17.2](#)). 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 17.2](#).

- `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`: a 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.

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.

- `/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 LaF_3 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 $\text{LaF}_3:\text{Ln}^{3+}$ and $\text{LiYF}_4:\text{Ln}^{3+}$

The study of Carnall *et al.* [Car+89] on lanthanum fluoride was a systematic review of trivalent lanthanide ions in LaF_3 . In this work they fitted the experimental data for all of the lanthanide ions using the single-configuration effective Hamiltonian. In their appendices one can find their calculated values, together with the experimental values that they used for their least squares fittings. In `qlanth` this data can be accessed by invoking the command `LoadCarnall` which brings into the session an association that has keys that have as values the tables and appendices from this article. Fig-6 shows the results of a calculation done with `qlanth` for the energy levels in LaF_3 . Additionally the function `LoadLaF3Parameters` can be used to query the data for the fitted parameters, which may serve as a useful starting point for the description of the lanthanides ions in hosts other than LaF_3 .

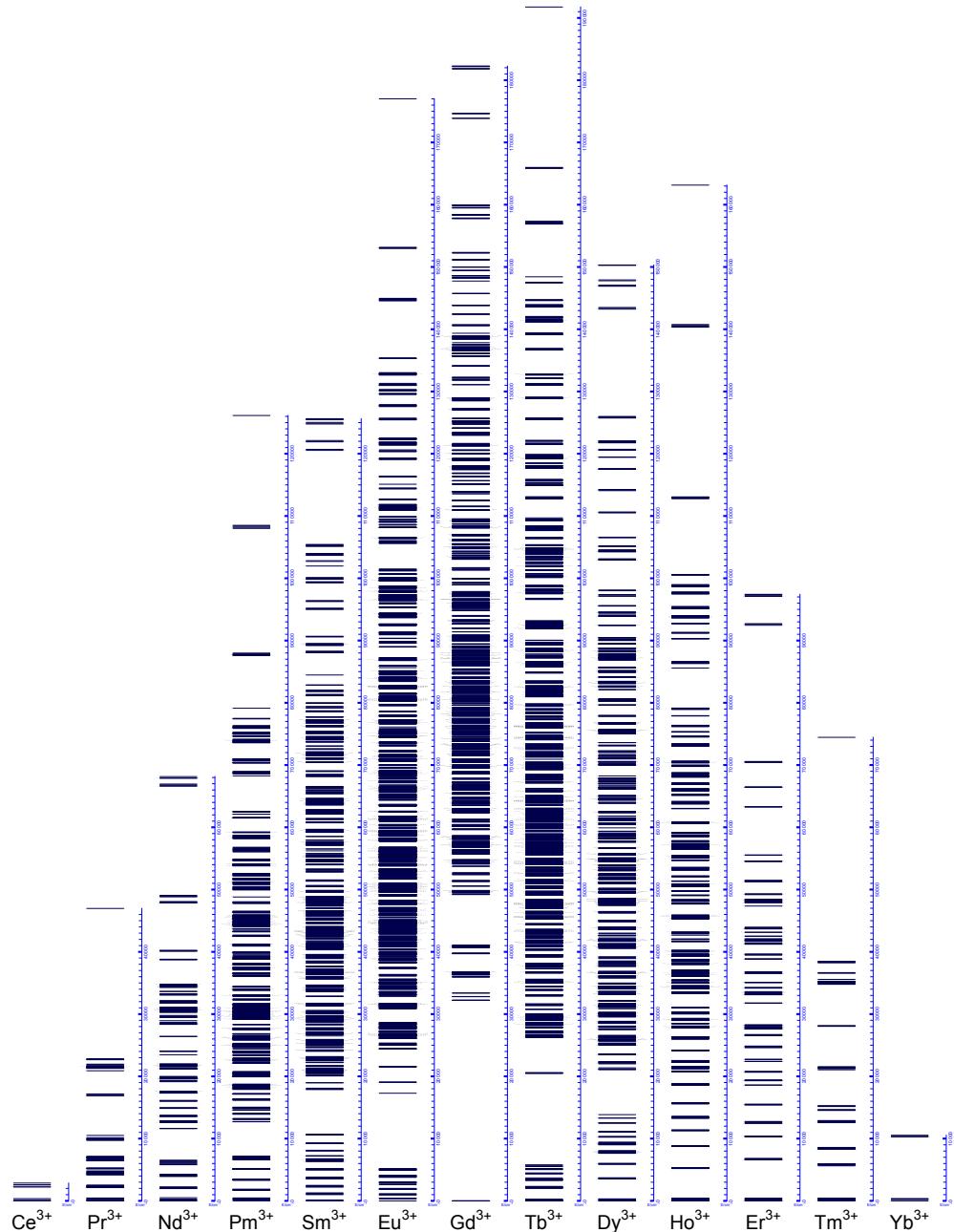


Figure 6: Energy levels in LaF_3 .

Similarly, Cheng *et al.* [Che+16] compiled data for LiYF_4 . In `qlanth` model parameters for LiYF_4 can be obtained by calling the function `LoadLiYF4Parameters`. Fig-7 shows the results of a calculation done with `qlanth` for the energy levels in LiYF_4 .

```
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}
  ";
```

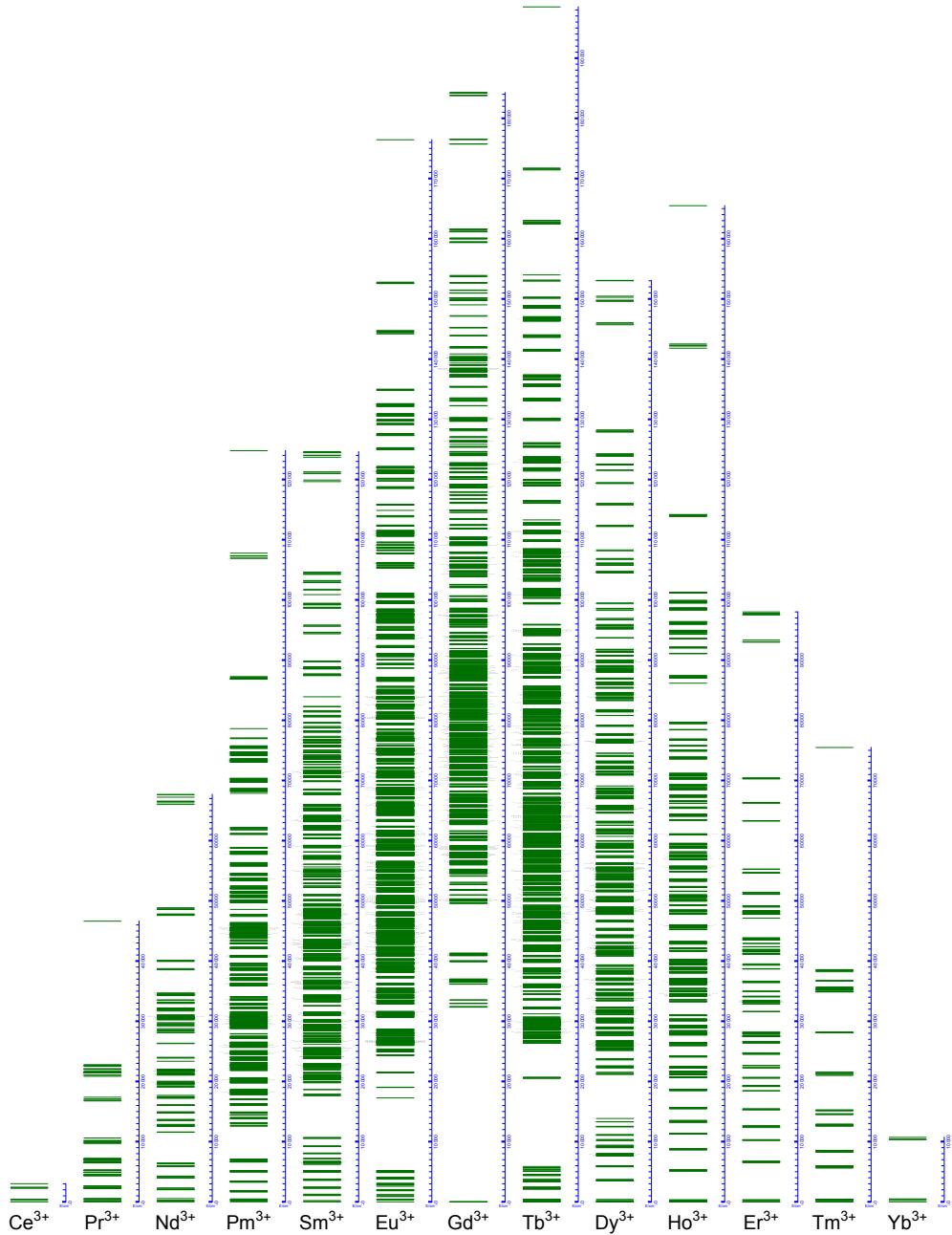


Figure 7: Energy levels in LiYF_4 .

```

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 );

```

```

1 LoadLaF3Parameters::usage = "LoadLaF3Parameters[ln] takes a string
  with the symbol the element of a trivalent lanthanide ion and
  returns model parameters for it. It is based on the data for LaF3.
  If the option ''Free Ion'' is set to True then the function sets
  all crystal field parameters to zero. Through the option ''gs'' it
  allows modifying the electronic gyromagnetic ratio. For
  completeness this function also computes the E parameters using
  the F parameters quoted on Carnall.";
2 Options[LoadLaF3Parameters] = {
3   "Free Ion" -> False,
4   "gs" -> 2.002319304386,
5   "With Uncertainties" -> False
6 };

```

```

7 LoadLaF3Parameters[Ln_String, OptionsPattern[]] := Module[
8   {params, uncertain,
9    uncertainKeys, uncertainRules},
10   (
11     If[Not[ValueQ[Carnall]],
12        LoadCarnall[];
13      ];
14     params = Association[Carnall["data"][[Ln]]];
15     (*If a free ion then all the parameters from the crystal field
16      are set to zero*)
17     If[OptionValue["Free Ion"],
18       Do[params[cfSymbol] = 0, {cfSymbol, cfSymbols}]
19     ];
20     params[F0] = 0;
21     params[M2] = 0.56 * params[M0]; (*See Carnall 1989,Table I,
22     caption,probably fixed based on HF values*)
23     params[M4] = 0.31 * params[M0]; (*See Carnall 1989,Table I,
24     caption,probably fixed based on HF values*)
25     params[P0] = 0;
26     params[P4] = 0.5 * params[P2]; (*See Carnall 1989,Table I,
27     caption,probably fixed based on HF values*)
28     params[P6] = 0.1 * params[P2]; (*See Carnall 1989,Table I,
29     caption,probably fixed based on HF values*)
30     params[gs] = OptionValue["gs"];
31     {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[F0],
32     params[F2], params[F4], params[F6]];
33     params[E0] = 0;
34     If[
35       Not[OptionValue["With Uncertainties"]],
36       Return[params],
37       (
38         uncertain = Association[Carnall["annotations"][[Ln]]];
39         uncertainKeys = Keys[uncertain];
40         uncertain = If[# == "Not allowed to vary in fitting." ||
41 # == "Interpolated",
42           0., #] & /@ uncertain;
43         paramKeys = Keys[params];
44         uncertainVals = Sort[Intersection[paramKeys, uncertainKeys]]
45 /. Association[uncertain];
46         uncertainRules = MapThread[Rule, {Sort[uncertainKeys],
47         uncertainVals}];
48         Which[
49           MemberQ[{Ce, "Yb"}, Ln],
50           (
51             subsetL = {F0};
52             subsetR = {0};
53           ),
54           True,
55           (
56             subsetL = {F0, M2, M4, P0, P4, P6, E0, E1, E2, E3};
57             subsetR = {0, M0*0.65, M0*0.31, 0, P2*0.5, P2*0.1,
58             0,
59             Sqrt[(196 F2^2)/164025 + (49 F4^2)/88209 + (122500 F6^2)
60             /134165889],
61             Sqrt[F2^2/4100625 + F4^2/10673289 + (30625 F6^2)
62             /2743558264161],
63             Sqrt[F2^2/18225 + (4 F4^2)/1185921 + (30625 F6^2)
64             /1803785841]};
65           )
66         ];
67         uncertainRules = Join[uncertainRules, MapThread[Rule, {
68           subsetL, subsetR /. uncertainRules}]];
69         uncertainRules = Association[uncertainRules];
70         Which[
71           Ln == "Eu",
72           (
73             uncertainRules[F4] = 12.121;
74             uncertainRules[F6] = 15.872;
75           ),
76           Ln == "Gd",
77           (
78             uncertainRules[F4] = 12.07;
79           ),
80           Ln == "Tb",
81           (
82             uncertainRules[F4] = 41.006;
83           )
84         ]
85       ]
86     ]
87   ]
88 ]

```

```

70      )
71  ];
72  If[MemberQ[{"Eu", "Gd", "Tb"}, Ln],
73  (
74      uncertainRules[E1] = Sqrt[(196 F2^2)/164025 + (49 F4^2)
75 /88209 + (122500 F6^2)/134165889] /. uncertainRules;
76      uncertainRules[E2] = Sqrt[F2^2/4100625 + F4^2/10673289 +
77 (30625 F6^2)/2743558264161] /. uncertainRules;
78      uncertainRules[E3] = Sqrt[F2^2/18225 + (4 F4^2)/1185921 +
79 (30625 F6^2)/1803785841] /. uncertainRules;
80  )
81  ];
82  uncertainKeys = First /@ Normal[uncertainRules];
83  fullParams = Association[MapThread[Rule, {uncertainKeys,
84 MapThread[Around, {uncertainKeys /. params, uncertainKeys /.
85 uncertainRules}]}]];
86  Return[Join[params, fullParams]]
87 )
88 ];
89 )
90 ];
91 
```

```

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
   Cheng et al.";
2 LoadLiYF4Parameters[ln_, OptionsPattern[]]:=(
3   If![ValueQ[paramsLiYF4],
4     paramsChengLiYF4 = Import[FileNameJoin[{moduleDir, "data",
5       "chengLiYF4.m"}]];
6   );
7   Return[paramsChengLiYF4[ln]];
8 )
```

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 **Eqn-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 13.2 on Mac OS Sonoma 14.5.

14 Units

Following the tradition of the spectroscopic community, all the matrix elements of the Hamiltonian are calculated using the Kayser ($\mathcal{K} \equiv \text{cm}^{-1}$) as the (pseudo) energy unit. All the parameters (except the magnetic field which is in Tesla) in the effective Hamiltonian are assumed to be in this unit. As is customary, the angular momentum operators assume atomic units in which $\hbar = 1$.

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

```

1 BeginPackage["qonstants`"];
2
3 (* Physical Constants*)
4 bohrRadius = 5.29177210903 * 10^-9;
5 ee          = 1.602176634 * 10^-19;
6
7 (* Spectroscopic niceties*)
8 theLanthanides = {"Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy",
9   "Ho", "Er", "Tm", "Yb"};
10 theActinides  = {"Ac", "Th", "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk",
11   "Cf", "Es", "Fm", "Md", "No", "Lr"};
12 theTrivalents = {"Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho",
13   "Er", "Tm"};
14 specAlphabet = "SPDFGHIKLMNOQRTUV";
15 complementaryNumE = {1,13,2,12,3,11,4,10,5,9,6,8,7};
16
17 (* SI *)
18 hPlanck = 6.62607015 * 10^-34; (* Planck's constant in J s
19   *)
20 hBar    = hPlanck / (2 \[Pi]); (* reduced Planck's constant
21   in J s *)
22 \[Mu]B  = 9.2740100783 * 10^-24; (* Bohr magneton in SI *)
23 me     = 9.1093837015 * 10^-31; (* electron mass in kg *)
24 cLight = 2.99792458 * 10^8; (* speed of light in m/s *)
25 eCharge = 1.602176634 * 10^-19; (* elementary charge in SI *)
26 \[Epsilon]0 = 8.8541878128 * 10^-12; (* electric permittivity in
27   vacuum in SI *)
28 \[Mu]0  = 4 \[Pi] * 10^-7; (* magnetic permeability in
29   vacuum in SI *)
30 \[Alpha]Fine = 1/137.036; (* fine structure constant *)
31
32 bohrRadius = 5.29177*10^-11; (* Bohr radius in m *)
33 hartreeEnergy = hBar^2 / (me * bohrRadius^2); (* Hartree energy in J
34   *)
35 hartreeTime = hBar / hartreeEnergy; (* Hartree time in s *)
36
37 (* Hartree atomic units *)
38 hPlanckHartree = 2 \[Pi]; (* Planck's constant in Hartree *)
39 meHartree      = 1; (* electron mass in Hartree *)
40 cLightHartree  = 137.036; (* speed of light in Hartree *)
41 eChargeHartree = 1; (* elementary charge in Hartree *)
42 \[Mu]0Hartree  = \[Alpha]Fine^2; (* magnetic permeability in vacuum in
43   Hartree *)
44
45 (* some conversion factors *)
46 eVToJoule      = eCharge;
47 jouleToeV       = 1 / eVToJoule;
48 jouleToHartree = 1 / hartreeEnergy;
49 eVToKayser     = eCharge / ( hPlanck * cLight * 100 ); (* 1 eV =
50   8065.54429 cm^-1 *)
51 kayserToeV     = 1 / eVToKayser;
52 teslaToKayser  = 2 * \[Mu]B / hPlanck / cLight / 100;
53 kayserToHartree = kayserToeV * eVToJoule * jouleToHartree;
54 hartreeToKayser = 1 / kayserToHartree;
55
56 EndPackage[];
```

15 Notation

orbital angular momentum operator of a single electron

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

total orbital angular momentum operator

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

spin angular momentum operator of a single electron

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

total spin angular momentum operator

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

Shorthand for all other quantum numbers

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

orbital angular momentum number

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

spinning angular momentum number

$$\overline{\underline{d}} \quad (127)$$

Coulomb non-central potential

$$\overline{\hat{e}} \quad (128)$$

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

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

spherical tensor operator of rank k

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

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

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

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

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

16 Definitions

$$\overline{[x]} := \begin{cases} 2x & \text{if } x \text{ is even} \\ 2x+1 & \text{if } x \text{ is odd} \end{cases} \quad (135)$$

irreducible unit tensor operator of rank k

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

symmetric unit tensor operator for n equivalent electrons

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

Renormalized spherical harmonics

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

Triangle “delta” between j_1, j_2, j_3

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

17 code

17.1 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
64 Review 127, no. 3 (August 1, 1962): 750-61.
65 https://doi.org/10.1103/PhysRev.127.750.
66
67 + Olfelt, GS. "Intensities of Crystal Spectra of Rare-Earth Ions." The Journal
68 of Chemical Physics 37, no. 3 (1962): 511-20.
69
```

```

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

```

136 BeginPackage["qlanth`"];
137 Needs["qconstants`"];
138 Needs["qplotter`"];
139 Needs["misc`"];
140
141 paramAtlas = "
E0: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
E1: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
E2: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
E3: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
```

```

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

```

```

216 S46: crystal field parameter S_4^6 (real)
217 S56: crystal field parameter S_5^6 (real)
218 S66: crystal field parameter S_6^6 (real)
219
220 \[Epsilon]: ground level baseline shift
221 t2Switch: controls the usage of the t2 operator beyond f7 (1 for f7
222     or below, 0 for f8 or above)
223 wChErrA: If 1 then the type-A errors in Chen are used, if 0 then not.
224 wChErrB: If 1 then the type-B errors in Chen are used, if 0 then not.
225
226 Bx: x component of external magnetic field (in T)
227 By: y component of external magnetic field (in T)
228 Bz: z component of external magnetic field (in T)
229
230 \[CapitalOmega]2: Judd-Ofelt intensity parameter k=2 (in cm^2)
231 \[CapitalOmega]4: Judd-Ofelt intensity parameter k=4 (in cm^2)
232 \[CapitalOmega]6: Judd-Ofelt intensity parameter k=6 (in cm^2)
233
234 nE: number of electrons in a configuration
235
236 E0p: orthogonalized E0
237 E1p: orthogonalized E1
238 E2p: orthogonalized E2
239 E3p: orthogonalized E3
240 αp: orthogonalized α
241 βp: orthogonalized β
242 γp: orthogonalized γ
243 ";
244 paramSymbols = StringSplit[paramAtlas, "\n"];
245 paramSymbols = Select[paramSymbols, # != "" & ];
246 paramSymbols = ToExpression[StringSplit[#, ":" ][[1]]] & /@
247     paramSymbols;
248 Protect /@ paramSymbols;
249
250 (* Parameter families *)
251 Unprotect[racahSymbols, chenSymbols, slaterSymbols, controlSymbols,
252     cfSymbols, TSymbols, pseudoMagneticSymbols, marvinSymbols,
253     casimirSymbols, magneticSymbols, juddOfeltIntensitySymbols,
254     mostlyOrthogonalSymbols];
255 racahSymbols = {E0, E1, E2, E3};
256 chenSymbols = {wChErrA, wChErrB};
257 slaterSymbols = {F0, F2, F4, F6};
258 controlSymbols = {t2Switch, \[Sigma]SS};
259 cfSymbols = {B02, B04, B06, B12, B14, B16, B22, B24, B26, B34,
260     B36,
261             B44, B46, B56, B66,
262             S12, S14, S16, S22, S24, S26, S34, S36, S44, S46,
263             S56, S66};
264 TSymbols = {T2, T2p, T3, T4, T6, T7, T8, T11p, T12, T14, T15,
265     T16, T17, T18, T19};
266 pseudoMagneticSymbols = {P0, P2, P4, P6};
267 marvinSymbols = {M0, M2, M4};
268 magneticSymbols = {Bx, By, Bz, gs, ζ};
269 casimirSymbols = {α, β, γ};
270 juddOfeltIntensitySymbols = {\[CapitalOmega]2, \[CapitalOmega]4, \
271     CapitalOmega]6};
272 mostlyOrthogonalSymbols = Join[{E0p, E1p, E2p, E3p, αp, βp, γp},
273     {T2p, T3, T4, T6, T7, T8, T11p, T12, T14, T15, T16, T17, T18, T19},
274     cfSymbols];
275
276 paramFamilies = Hold[{racahSymbols, chenSymbols,
277     slaterSymbols, controlSymbols, cfSymbols, TSymbols,
278     pseudoMagneticSymbols, marvinSymbols, casimirSymbols,
279     magneticSymbols, juddOfeltIntensitySymbols,
280     mostlyOrthogonalSymbols}];
281 ReleaseHold[Protect /@ paramFamilies];
282 crystalGroups = {"C1", "Ci", "C2", "Cs", "C2h", "D2", "C2v", "D2h", "C4", "S4",
283     "C4h", "D4", "C4v", "D3d", "D4h", "C3", "C3i", "D3", "C3v", "D3d", "C6", "C3h",
284     "C6h", "D6", "C6v", "D3h", "D6h", "T", "Th", "O", "Td", "Oh"};
285
286 (* Parameter usage *)
287 paramLines = Select[StringSplit[paramAtlas, "\n"], # != "" &];
288 usageTemplate = StringTemplate["`paramSymbol`::usage=``paramSymbol`\
289     : `paramUsage`\";`"];
290 Do[(
291     {paramString, paramUsage} = StringSplit[paramLine, ":"];

```

```

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

```

```

350 GenerateThreeBodyTables;
351
352 Generator;
353 GroundMagDipoleOscillatorStrength;
354 HamMatrixAssembly;
355 HamiltonianForm;
356
357 HamiltonianMatrixPlot;
358 HoleElectronConjugation;
359 ImportMZip;
360 IonSolver;
361 JJBlockMagDip;
362
363 JJBlockMatrix;
364 JJBlockMatrixFileName;
365 JJBlockMatrixTable;
366 JuddOfeltUkSquared;
367 LabeledGrid;
368 LandeFactor;
369
370 LevelElecDipoleOscillatorStrength;
371 LevelJJBlockMagDipole;
372 LevelMagDipoleLineStrength;
373 LevelMagDipoleMatrixAssembly;
374 LevelMagDipoleOscillatorStrength;
375
376 LevelMagDipoleSpontaneousDecayRates;
377 LevelSimplerSymbolicHamMatrix;
378 LevelSolver;
379 ListRepeater;
380 LoadAll;
381
382 LoadCFP;
383 LoadCarnall;
384 LoadChenDeltas;
385 LoadElectrostatic;
386 LoadFreeIon;
387 LoadGuillotParameters;
388
389 LoadLaF3Parameters;
390 LoadLiYF4Parameters;
391 LoadSO0andECS0;
392 LoadSO0andECSOLS;
393 LoadSpinOrbit;
394 LoadSpinSpin;
395
396 LoadSymbolicHamiltonians;
397 LoadT11;
398 LoadT22;
399 LoadTermLabels;
400 LoadThreeBody;
401
402 LoadUk;
403 LoadV1k;
404 MagDipLineStrength;
405 MagDipoleMatrixAssembly;
406 MagDipoleRates;
407
408 MagneticInteractions;
409 MapToSparseArray;
410 MaxJ;
411 MinJ;
412 NKCFPPhase;
413
414 ParamPad;
415 ParseBenelli2015;
416 ParseStates;
417 ParseStatesByNumBasisVecs;
418 ParseStatesByProbabilitySum;
419
420 ParseTermLabels;
421 Phaser;
422 PrettySaundersSL;
423 PrettySaundersSLJ;
424 PrettySaundersSLmJ;
425

```

```

426 PrintL;
427 PrintSLJ;
428 PrintSLJM;
429 ReducedS0OandECS0inf2;
430 ReducedS0OandECS0infn;
431
432 ReducedT11inf2;
433 ReducedT22inf2;
434 ReducedT22infn;
435 ReducedUk;
436 ReducedUkTable;
437
438 ReducedV1kTable;
439 Reducedt11inf2;
440 ReplaceInSparseArray;
441 ScalarLSJMFromLS;
442 S0OandECS0;
443 S0OandECS0LSTable;
444
445 S0OandECS0Table;
446 ScalarOperatorProduct;
447 Seniority;
448 ShiftedLevels;
449 SimplerSymbolicHamMatrix;
450
451 SixJay;
452 SpinOrbit;
453 SpinOrbitTable;
454 SpinSpin;
455 SpinSpinTable;
456
457 Sqk;
458 SquarePrimeToNormal;
459 TPO;
460 TabulateJJBlockMagDipTable;
461 TabulateJJBlockMatrixTable;
462
463 TabulateManyJJBlockMagDipTables;
464 TabulateManyJJBlockMatrixTables;
465 ThreeBodyTable;
466 ThreeBodyTables;
467 ThreeJay;
468
469 TotalCFITers;
470 chenDeltas;
471 fK;
472 fnTermLabels;
473 fsubk;
474
475 fsupk;
476 moduleDir;
477 symbolicHamiltonians;
478
479 (* this selects the function that is applied to calculated matrix
   elements which helps keep down the complexity of the resulting
   algebraic expressions *)
480 SimplifyFun = Expand;
481
482 Begin["`Private`"]
483
484 moduleDir =DirectoryName[$InputFileName];
485 frontEndAvailable = (Head[$FrontEnd] === FrontEndObject);
486
487 (* ##### MISC #####
488 (* ##### MISC #####
489
490 TPO::usage = "TPO[x, y, ...] gives the product of 2x+1, 2y+1, ...";
491 TPO[args_] := Times @@ ((2*# + 1) & /@ {args});
492
493 Phaser::usage = "Phaser[x] gives (-1)^x.";
494 Phaser[exponent_] := ((-1)^exponent);
495
496 TriangleCondition::usage = "TriangleCondition[a, b, c] evaluates
   the triangle condition on a, b, and c.";
497 TriangleCondition[a_, b_, c_] := (Abs[b - c] <= a <= (b + c));
498

```

```

499 TriangleAndSumCondition::usage = "TriangleAndSumCondition[a, b, c]
500   evaluates the joint satisfaction of the triangle and sum
501   conditions.";
502 TriangleAndSumCondition[a_, b_, c_] := (
503   And[
504     Abs[b - c] <= a <= (b + c),
505     IntegerQ[a + b + c]
506   ];
507
508 SquarePrimeToNormal::usage = "SquarePrimeToNormal[squarePrime]
509   evaluates the standard representation of a number from the squared
510   prime representation given in the list squarePrime. For
511   squarePrime of the form {c0, c1, c2, c3, ...} this function
512   returns the number c0 * Sqrt[p1^c1 * p2^c2 * p3^c3 * ...] where pi
513   is the ith prime number. Exceptionally some of the ci might be
514   letters in which case they have to be one of \"A\", \"B\",
515   \"C\", \"D\" with them corresponding to 10, 11, 12, and 13, respectively.
516   ";
517 SquarePrimeToNormal[squarePrime_] :=
518 (
519   radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {
520     idx1, 2, Length[squarePrime]}];
521   radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" ->
522   13};
523   val = squarePrime[[1]] * Sqrt[radical];
524   Return[val];
525 );
526
527 ParamPad::usage = "ParamPad[params] takes an association params
528   whose keys are a subset of paramSymbols. The function returns a
529   new association where all the keys not present in paramSymbols,
530   will now be included in the returned association with their values
531   set to zero.
532   The function additionally takes an option \"Print\" that if set to
533   True, will print the symbols that were not present in the given
534   association. The default is True.";
535 Options[ParamPad] = {"PrintFun" -> PrintTemporary};
536 ParamPad[params_, OptionsPattern[]] :=
537   notPresentSymbols = Complement[paramSymbols, Keys[params]];
538   PrintFun = OptionValue["PrintFun"];
539   PrintFun["Following symbols were not given and are being set to
540   0: ",
541     notPresentSymbols];
542   newParams = Transpose[{paramSymbols, ConstantArray[0, Length[
543     paramSymbols]]}];
544   newParams = (#[[1]] -> #[[2]]) & /@ newParams;
545   newParams = Association[newParams];
546   newParams = Join[newParams, params];
547   Return[newParams];
548 )
549
550 (* ##### Angular Momentum #####
551 (* ##### Angular Momentum #####
552
553 AngularMomentumMatrices::usage = "AngularMomentumMatrices[j] gives
554   the matrix representation for the angular momentum operators Jx,
555   Jy, and Jz for a given angular momentum j in the basis of
556   eigenvectors of jz. j may be a half-integer or an integer.
557   The options are \"Sparse\" which defaults to False and \"Order\""
558   which defaults to \"HighToLow\".
559   The option \"Order\" can be set to \"LowToHigh\" to get the
560   matrices in the order from -jay to jay otherwise they are returned
561   in the order jay to -jay.
562   The function returns a list {JxMatrix, JyMatrix, JzMatrix} with the
563   matrix representations for the cartesian components of the
564   angular momentum operator.";
565 Options[AngularMomentumMatrices] = {
566   "Sparse" -> False,
567   "Order" -> "HighToLow"};
568 AngularMomentumMatrices[jay_, OptionsPattern[]] := Module[
569   {
570     JxMatrix, JyMatrix, JzMatrix,
571     JPlusMatrix, JMinusMatrix,
572     m1, m2,
573     ArrayInverter
574   }
575
576 
```

```

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

```

```

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

677       {Sb, Lb} = FindSL[\[Psi]b];
678       Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
679       CFPAssoc[{numE, SL, \[Psi]b}] *
680       CFPAssoc[{numE, SpLp, \[Psi]b}] *
681       SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
682       SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
683     ),  

684     {\[Psi]b, commonParents}
685   ];
686   prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp,
687   Lp]];
688   Return[prefactor * Vk1];
689 )
690 ];
691

```

```

681 GenerateReducedUkTable::usage = "GenerateReducedUkTable[numEmax]
682   can be used to generate the association of reduced matrix elements
683   for the unit tensor operators Uk from f^1 up to f^numEmax. If the
684   option \"Export\" is set to True then the resulting data is saved
685   to ./data/ReducedUkTable.m.";
686 Options[GenerateReducedUkTable] = {"Export" -> True, "Progress" ->
687   True};
688 GenerateReducedUkTable[numEmax_Integer:7, OptionsPattern[]] := (
689   numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
690     AllowedNKSLTerms[#]]&/@Range[1, numEmax]] * 4;
691   Print["Calculating " <> ToString[numValues] <> " values for Uk k "
692     =0,2,4,6."];
693   counter = 1;
694   If[And[OptionValue["Progress"], frontEndAvailable],
695     progBar = PrintTemporary[
696       Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
697         counter}]]];
698   ];
699   ReducedUkTable = Table[
700     (
701       counter = counter+1;
702       {numE, 3, SL, SpLp, k} -> SimplifyFun[ReducedUk[numE, 3, SL,
703         SpLp, k]]
704     ),
705     {numE, 1, numEmax},
706     {SL, AllowedNKSLTerms[numE]},
707     {SpLp, AllowedNKSLTerms[numE]},
708     {k, {0, 2, 4, 6}}
709   ];
710   ReducedUkTable = Association[Flatten[ReducedUkTable]];
711   ReducedUkTableFname = FileNameJoin[{moduleDir, "data", " "
712     ReducedUkTable.m}];
713   If[And[OptionValue["Progress"], frontEndAvailable],
714     NotebookDelete[progBar]
715   ];
716   If[OptionValue["Export"],
717     (
718       Print["Exporting to file " <> ToString[ReducedUkTableFname]];
719       Export[ReducedUkTableFname, ReducedUkTable];
720     )
721   ];
722   Return[ReducedUkTable];
723 )
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
GenerateReducedV1kTable::usage = "GenerateReducedV1kTable[nmax]
calculates values for Vk1 and returns an association where the
keys are lists of the form {n, SL, SpLp, 1}. If the option \"
Export\" is set to True then the resulting data is saved to ./data
/ReducedV1kTable.m.";
Options[GenerateReducedV1kTable] = {"Export" -> True, "Progress" ->
  True};
GenerateReducedV1kTable[numEmax_Integer:7, OptionsPattern[]] := (
  numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
    AllowedNKSLTerms[#]]&/@Range[1, numEmax]];
  Print["Calculating " <> ToString[numValues] <> " values for Vk1."];
  counter = 1;
  If[And[OptionValue["Progress"], frontEndAvailable],
    progBar = PrintTemporary[
      Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
        counter}]]];
  ];
  ReducedV1kTable = Table[
    (
      counter = counter+1;
      {n, SL, SpLp, 1} -> SimplifyFun[ReducedV1k[n, SL, SpLp, 1]]
    ),
    {n, 1, numEmax},
    {SL, AllowedNKSLTerms[n]},
    {SpLp, AllowedNKSLTerms[n]}
  ];
  ReducedV1kTable = Association[ReducedV1kTable];
  If[And[OptionValue["Progress"], frontEndAvailable],
    NotebookDelete[progBar]
  ];
  exportFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m"}]
```

```

    "}] ;
741  If[OptionValue["Export"] ,
742   (
743     Print["Exporting to file "<>ToString[exportFname]];
744     Export[exportFname, ReducedV1kTable];
745   )
746 ];
747  Return[ReducedV1kTable];
748 )
749
750 (* ##### Racah Algebra ##### *)
751 (* ##### ##### ##### ##### ##### ##### ##### ##### ##### *)
752
753 (* ##### ##### ##### ##### ##### ##### ##### ##### ##### ##### *)
754 (* ##### ##### ##### ##### Electrostatic ##### ##### ##### *)
755
756 fsubk::usage = "fsubk[numE, orbital, SL, SLP, k] gives the Slater
757 integral f_k for the given configuration and pair of SL terms. See
758 equation 12.17 in TASS." ;
759 fsubk[numE_, orbital_, NKSL_, NKSLP_, k_] := Module[
760 {terms, S, L, Sp, Lp,
761 termsWithSameSpin, SL,
762 fsubkVal, spinMultiplicity,
763 prefactor, summand1, summand2},
764 (
765 {S, L} = FindSL[NKSL];
766 {Sp, Lp} = FindSL[NKSLP];
767 terms = AllowedNKSLTerms[numE];
768 (* sum for summand1 is over terms with same spin *)
769 spinMultiplicity = 2*S + 1;
770 termsWithSameSpin = StringCases[terms, ToString[
771 spinMultiplicity] ~~ __];
772 termsWithSameSpin = Flatten[termsWithSameSpin];
773 If[Not[{S, L} == {Sp, Lp}],
774   Return[0]
775 ];
776 prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
777 summand1 = Sum[(
778   ReducedUkTable[{numE, orbital, SL, NKSL, k}] *
779   ReducedUkTable[{numE, orbital, SL, NKSLP, k}]
780   ),
781   {SL, termsWithSameSpin}
782 ];
783 summand1 = 1 / TPO[L] * summand1;
784 summand2 = (
785   KroneckerDelta[NKSL, NKSLP] *
786   (numE *(4*orbital + 2 - numE)) /
787   ((2*orbital + 1) * (4*orbital + 1))
788 );
789 fsubkVal = prefactor*(summand1 - summand2);
790 Return[fsubkVal];
791 )
792 ];
793
794 fsupk::usage = "fsupk[numE, orbital, SL, SLP, k] gives the
795 superscripted Slater integral f^k = Subscript[f, k] * Subscript[D,
796 k].";
797 fsupk[numE_, orbital_, NKSL_, NKSLP_, k_] := (
798   Dk[k] * fsubk[numE, orbital, NKSL, NKSLP, k]
799 )
800
801 Dk::usage = "D[k] gives the ratio between the super-script and sub-
802 scripted Slater integrals (F^k / F_k). k must be even. See table
803 6-3 in TASS, and also section 2-7 of Wybourne (1965). See also
804 equation 6.41 in TASS." ;
805 Dk[k_] := {1, 225, 1089, 184041/25}[[k/2+1]];
806
807 FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the Racah parameters
808 {E0, E1, E2, E3} corresponding to the given Slater integrals.
809 See eqn. 2-80 in Wybourne.
810 Note that in that equation the subscripted Slater integrals are
811 used but since this function assumes the the input values are
812 superscripted Slater integrals, it is necessary to convert them
813 using Dk." ;
814 FtoE[F0_, F2_, F4_, F6_] := Module[
815 {E0, E1, E2, E3},

```

```

804 (
805   E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
806   E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
807   E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;
808   E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
809   Return[{E0, E1, E2, E3}];
810 )
811 ];
812
813 EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the Slater integral
814 parameters {F0, F2, F4, F6} corresponding to the given Racah
815 parameters {E0, E1, E2, E3}. This is the inverse of the FtoE
816 function.";
817 EtoF[E0_, E1_, E2_, E3_] := Module[
818   {F0, F2, F4, F6},
819   (
820     F0 = 1/7      (7 E0 + 9 E1);
821     F2 = 75/14    (E1 + 143 E2 + 11 E3);
822     F4 = 99/7     (E1 - 130 E2 + 4 E3);
823     F6 = 5577/350 (E1 + 35 E2 - 7 E3);
824     Return[{F0, F2, F4, F6}];
825   )
826 ];
827
828 Electrostatic::usage = "Electrostatic[{numE, NKSL, NKSLp}] returns
829 the LS reduced matrix element for repulsion matrix element for
830 equivalent electrons. See equation 2-79 in Wybourne (1965). The
831 option \"Coefficients\" can be set to \"Slater\" or \"Racah\". If
832 set to \"Racah\" then E_k parameters and e^k operators are assumed
833 , otherwise the Slater integrals F^k and operators f_k. The
834 default is \"Slater\".";
835 Options[Electrostatic] = {"Coefficients" -> "Slater"};
836 Electrostatic[{numE_, NKSL_, NKSLp_}, OptionsPattern[]] := Module[
837   {fsub0, fsub2, fsub4, fsub6,
838    esub0, esub1, esub2, esub3,
839    fsup0, fsup2, fsup4, fsup6,
840    eMatrixVal, orbital},
841   (
842     orbital = 3;
843     Which[
844       OptionValue["Coefficients"] == "Slater",
845       (
846         fsub0 = fsubk[numE, orbital, NKSL, NKSLp, 0];
847         fsub2 = fsubk[numE, orbital, NKSL, NKSLp, 2];
848         fsub4 = fsubk[numE, orbital, NKSL, NKSLp, 4];
849         fsub6 = fsubk[numE, orbital, NKSL, NKSLp, 6];
850         eMatrixVal = fsub0*F0 + fsub2*F2 + fsub4*F4 + fsub6*F6;
851       ),
852       OptionValue["Coefficients"] == "Racah",
853       (
854         fsup0 = fsupk[numE, orbital, NKSL, NKSLp, 0];
855         fsup2 = fsupk[numE, orbital, NKSL, NKSLp, 2];
856         fsup4 = fsupk[numE, orbital, NKSL, NKSLp, 4];
857         fsup6 = fsupk[numE, orbital, NKSL, NKSLp, 6];
858         esub0 = fsup0;
859         esub1 = 9/7*fsup0 + 1/42*fsup2 + 1/77*fsup4 + 1/462*
860         fsup6;
861         esub2 = 143/42*fsup2 - 130/77*fsup4 + 35/462*
862         fsup6;
863         esub3 = 11/42*fsup2 + 4/77*fsup4 - 7/462*
864         fsup6;
865         eMatrixVal = esub0*E0 + esub1*E1 + esub2*E2 + esub3*E3;
866       )
867     ];
868     Return[eMatrixVal];
869   )
870 ];
871
872 GenerateElectrostaticTable::usage = "GenerateElectrostaticTable[
873 numEmax] can be used to generate the table for the electrostatic
874 interaction from f^1 to f^numEmax. If the option \"Export\" is set
875 to True then the resulting data is saved to ./data/
876 ElectrostaticTable.m.";
877 Options[GenerateElectrostaticTable] = {"Export" -> True, "
878 Coefficients" -> "Slater"};
879 GenerateElectrostaticTable[numEmax_Integer:7, OptionsPattern[]] :=

```

```

(
ElectrostaticTable = Table[
  {numE, SL, SpLp} -> SimplifyFun[Electrostatic[{numE, SL, SpLp}],
 "Coefficients" -> OptionValue["Coefficients"]]],
  {numE, 1, numEmax},
  {SL, AllowedNKSLTerms[numE]},
  {SpLp, AllowedNKSLTerms[numE]}
];
ElectrostaticTable = Association[Flatten[ElectrostaticTable]];
If[OptionValue["Export"],
  Export[FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}],
];
  ElectrostaticTable];
];
Return[ElectrostaticTable];
);

(* ##### Electrostatic #####
(* ##### Bases #####
(* ##### BasisGenerator #####
(* ##### BasisLSJM #####
(* ##### BasisLSJ #####
BasisTableGenerator::usage = "BasisTableGenerator[numE] returns an
  association whose keys are triples of the form {numE, J} and whose
  values are lists having the basis elements that correspond to {
  numE, J}.";
BasisTableGenerator[numE_] := Module[
  {energyStatesTable, allowedJ, J, Jp},
  (
    energyStatesTable = <||>;
    allowedJ = AllowedJ[numE];
    Do[
    (
      energyStatesTable[{numE, J}] = EnergyStates[numE, J];
    ),
    {Jp, allowedJ},
    {J, allowedJ}];
    Return[energyStatesTable]
  )
];
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}.
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.
";
Options[BasisLSJM] = {"AsAssociation" -> False};
BasisLSJM[numE_, OptionsPattern[]] := Module[
  {energyStatesTable, basis, idx1},
  (
    energyStatesTable = BasisTableGenerator[numE];
    basis = Table[
      energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
      {idx1, 1, Length[AllowedJ[numE]]}];
    basis = Flatten[basis, 1];
    If[OptionValue["AsAssociation"],
      (
        Js = AllowedJ[numE];
        basis = Table[(J -> Select[basis, #[[2]] == J &]), {J, Js}];
      ];
      basis = Association[basis];
    );
  ];
  Return[basis]
];
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}.
923 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.
924 ";
925 Options[BasisLSJ]={"AsAssociation"→False};
926 BasisLSJ[numE_,OptionsPattern[]]:=Module[
927 {Js,basis},
928 (
929   Js=AllowedJ[numE];
930   basis=BasisLSJMJ[numE,"AsAssociation"→False];
931   basis=DeleteDuplicates[{#[[1]],#[[2]]}~@basis];
932   If[OptionValue["AsAssociation"],
933     (
934       basis=Association~Table[(J~>Select[basis,#[[2]]==J&]),{
935         J,Js}]
936     )
937   ];
938   Return[basis];
939 ]
940 ];
941 (* ##### Bases #####
942 (* ##### #####
943 (* ##### #####
944 (* ##### #####
945 (* ##### Coefficients of Fracional Parentage #####
946
947 GenerateCFP::usage="GenerateCFP[] generates the association for
    the coefficients of fractional parentage. Result is exported to
    the file ./data/CFP.m. The coefficients of fractional parentage
    are taken beyond the half-filled shell using the phase convention
    determined by the option \"PhaseFunction\". The default is \"NK\""
    which corresponds to the phase convention of Nielson and Koster.
    The other option is \"Judd\" which corresponds to the phase
    convention of Judd.";
948 Options[GenerateCFP]={ "Export"→True, "PhaseFunction"→"NK"};
949 GenerateCFP[OptionsPattern[]]:=((
950   CFP=Table[
951     {numE,NKSL}→First[CFPTerms[numE,NKSL]],
952     {numE,1,7},
953     {NKSL,AllowedNKSLTerms[numE]}];
954   CFP=Association[CFP];
955   (* Go all the way to f14 *)
956   CFP=CFPExpander["Export"→False, "PhaseFunction"→
957     OptionValue["PhaseFunction"]];
958   If[OptionValue["Export"],
959     Export[FileNameJoin[{moduleDir,"data","CFPs.m"}],CFP];
960   ];
961   Return[CFP];
962 );
963 JuddCFPPPhase::usage="Phase between conjugate coefficients of
    fractional parentage according to Velkov's thesis, page 40.";
964 JuddCFPPPhase[parent_,parentS_,parentL_,daughterS_,daughterL_,
965 parentSeniority_,daughterSeniority_]:=Module[
966 {spin,orbital,expo,phase},
967 (
968   {spin,orbital}={1/2,3};
969   expo=(
970     (parentS+parentL+daughterS+daughterL)-
971     (orbital+spin)+
972     1/2*(parentSeniority+daughterSeniority-1)
973   );
974   phase=Phaser[-expo];
975   Return[phase];
976 )
977 ];
978 NKCFPPPhase::usage="Phase between conjugate coefficients of
    fractional parentage according to Nielson and Koster page viii.
    Note that there is a typo on there the expression for zeta should
    be  $(-1)^{(v-1)/2}$  instead of  $(-1)^{v-1/2}$ .";
979 NKCFPPPhase[parent_,parentS_,parentL_,daughterS_,daughterL_,
980 parentSeniority_,daughterSeniority_]:=Module[

```

```

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

```

```

1044         prefactor = Sqrt[prefactor];
1045         newCFPval = phase * prefactor * parentCFPval;
1046         key = {conjugateDaughter, parentTerm};
1047         complementaryCFPs[key] = Append[complementaryCFPs[
1048             key], {daughterTerm, newCFPval}]
1049             ),
1050             {parentCFP, parentCFPs}
1051             ]
1052             ),
1053             {daughterTerm, daughterTerms}
1054             ]
1055             ),
1056             {parent, 1, parentMax}
1057             ];
1058
1059         complementaryCFPs[{14, "1S"}] = {"1S", {"2F", 1}};
1060         extendedCFPs = Join[CFP, complementaryCFPs];
1061         If[OptionValue["Export"]];
1062         (
1063             exportFname = FileNameJoin[{moduleDir, "data", "CFPs_extended.m"}];
1064             Print["Exporting to ", exportFname];
1065             Export[exportFname, extendedCFPs];
1066         )
1067     ];
1068
1069 ];
1070
1071 GenerateCFPTable::usage = "GenerateCFPTable[] generates the table
1072 for the coefficients of fractional parentage. If the optional
1073 parameter \"Export\" is set to True then the resulting data is
1074 saved to ./data/CFPTable.m.
1075 The data being parsed here is the file attachment B1F_ALL.TXT which
1076 comes from Velkov's thesis.";
1077 Options[GenerateCFPTable] = {"Export" -> True};
1078 GenerateCFPTable[OptionsPattern[]] := Module[
1079     {rawText, rawLines, leadChar, configIndex, line, daughter,
1080      lineParts, numberCode, parsedNumber, toAppend, CFPTablefname},
1081     (
1082         CleanWhitespace[string_] := StringReplace[string,
1083         RegularExpression["\s+"]->" "];
1084         AddSpaceBeforeMinus[string_] := StringReplace[string,
1085         RegularExpression["(?!\s)-"]->" -"];
1086         ToIntegerOrString[list_] := Map[If[StringMatchQ[#, NumberString], ToExpression[#], #] &, list];
1087         CFPTable = ConstantArray[{}, 7];
1088         CFPTable[[1]] = {{"2F", {"1S", 1}}};
1089
1090
1091         (* Cleaning before processing is useful *)
1092         rawText = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
1093         rawLines = StringTrim/@StringSplit[rawText, "\n"];
1094         rawLines = Select[rawLines, #!=""];
1095         rawLines = CleanWhitespace/@rawLines;
1096         rawLines = AddSpaceBeforeMinus/@rawLines;
1097
1098         Do[((* the first character can be used to identify the start of a
1099            block *)
1100            leadChar=StringTake[line,{1}];
1101            (* ..FN, N is at position 50 in that line *)
1102            If[leadChar=="[",
1103                (
1104                    configIndex=ToExpression[StringTake[line,{50}]];
1105                    Continue[];
1106                )
1107                ];
1108                (* Identify which daughter term is being listed *)
1109                If[StringContainsQ[line, "[DAUGHTER TERM]"],
1110                    daughter=StringSplit[line, "[]"][[1]];
1111                    CFPTable[[configIndex]]=Append[CFPTable[[configIndex]], {daughter}];
1112                    Continue[];
1113                ];
1114                (* 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;;]]];
1112 parsedNumber = SquarePrimeToNormal[numberCode];
1113 toAppend     = {parent, parsedNumber};
1114 CFPTable[[configIndex]][[-1]] = Append[CFPTable[[configIndex
1115 ]][[-1]], toAppend]
1116 ),
1117 {line, rawLines}];
1118 If[OptionValue["Export"],
1119 (
1120   CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"
1121 }];
1122   Export[CFPTablefname, CFPTable];
1123 )
1124 ];
1125
1126 GenerateCFPAssoc::usage = "GenerateCFPAssoc[export] converts the
coefficients of fractional parentage into an association in which
zero values are explicit. If \"Export\" is set to True, the
association is exported to the file /data/CFPAssoc.m. This
function requires that the association CFP be defined.";
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 );
1167
1168 CFPTerms::usage = "CFPTerms[numE] gives all the daughter and parent
terms, together with the corresponding coefficients of fractional
parentage, that correspond to the the f^n configuration.
1169 CFPTerms[numE, SL] gives all the daughter and parent terms,
together with the corresponding coefficients of fractional
parentage, that are compatible with the given string SL in the f^n
configuration.
1170 CFPTerms[numE, L, S] gives all the daughter and parent terms,

```

```

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

```

```

1232 subfolder for the folder in which this package is in. It requires
1233 ReducedV1kTable to be defined."];
1234 Options[GenerateSpinOrbitTable] = {"Export" -> True};
1235 GenerateSpinOrbitTable[nmax_Integer:7, OptionsPattern[]] := Module[
1236 {numE, J, SL, SpLp, exportFname},
1237 (
1238 SpinOrbitTable =
1239 Table[
1240 {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
1241 {numE, 1, nmax},
1242 {J, MinJ[numE], MaxJ[numE]},
1243 {SL, Map[First, AllowedNKSLforJTerms[numE, J]]},
1244 {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
1245 ];
1246 SpinOrbitTable = Association[SpinOrbitTable];
1247
1248 exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable."}]
1249 ];
1250 If[OptionValue["Export"],
1251 (
1252 Print["Exporting to file "<>ToString[exportFname]];
1253 Export[exportFname, SpinOrbitTable];
1254 )
1255 ];
1256 Return[SpinOrbitTable];
1257 ];
1258
1259 (* ##### Spin Orbit #####
1260 (* ##### #####
1261 (* ##### Three Body Operators #####
1262
1263 ParseJudd1984::usage = "This function parses the data from tables 1
1264 and 2 of Judd from Judd, BR, and MA Suskin. \"Complete Set of
1265 Orthogonal Scalar Operators for the Configuration f^3\". JOSA B 1,
1266 no. 2 (1984): 261-65.";
1267 Options[ParseJudd1984] = {"Export" -> False};
1268 ParseJudd1984[OptionsPattern[]] := (
1269 ParseJuddTab1[str_] := (
1270 strR = ToString[str];
1271 strR = StringReplace[strR, ".5" -> "^(1/2)"];
1272 num = ToExpression[strR];
1273 sign = Sign[num];
1274 num = sign*Simplify[Sqrt[num^2]];
1275 If[Round[num] == num, num = Round[num]];
1276 Return[num]);
1277
1278 (* Parse table 1 from Judd 1984 *)
1279 judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"}];
1280 data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
1281 headers = data[[1]];
1282 data = data[[2 ;;]];
1283 data = Transpose[data];
1284 \[Psi] = Select[data[[1]], # != "" &];
1285 \[Psi]p = Select[data[[2]], # != "" &];
1286 matrixKeys = Transpose[{\[Psi], \[Psi]p}];
1287 data = data[[3 ;;]];
1288 cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data}];
1289 cols = Select[cols, Length[#] == 21 &];
1290 tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
1291 tab1 = Transpose[Prepend[Transpose[tab1], headers]];
1292
1293 (* Parse table 2 from Judd 1984 *)
1294 judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"}];
1295 data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
1296 headers = data[[1]];
1297 data = data[[2 ;;]];
1298 data = Transpose[data];
1299 {operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} =
1300 data[[;; 4]];
1301 multiFactorValues = ParseJuddTab1 /@ multiFactorValues;

```

```

1298 multiFactorValues = AssociationThread[multiFactorSymbols ->
1299 multiFactorValues];
1300 (*scale values of table 1 given the values in table 2*)
1301 oppyS = {};
1302 normalTable =
1303 Table[header = col[[1]];
1304 If[StringContainsQ[header, " "],
1305 (
1306 multiplierSymbol = StringSplit[header, " "][[1]];
1307 multiplierValue = multiFactorValues[multiplierSymbol];
1308 operatorSymbol = StringSplit[header, " "][[2]];
1309 oppyS = Append[oppyS, operatorSymbol];
1310 ),
1311 (
1312 multiplierValue = 1;
1313 operatorSymbol = header;
1314 )
1315 ];
1316 normalValues = 1/multiplierValue*col[[2 ;]];
1317 Join[{operatorSymbol}, normalValues], {col, tab1[[3 ;]]}
1318 ];
1319
1320 (*Create an association for the reduced matrix elements in the f
^3 config*)
1321 juddOperators = Association[];
1322 Do[
1323 col = normalTable[[colIndex]];
1324 opLabel = col[[1]];
1325 opValues = col[[2 ;]];
1326 opMatrix = AssociationThread[matrixKeys -> opValues];
1327 Do[
1328 opMatrix[Reverse[mKey]] = opMatrix[mKey]
1329 ),
1330 {mKey, matrixKeys}
1331 ];
1332 juddOperators[{3, opLabel}] = opMatrix,
1333 {colIndex, 1, Length[normalTable]}
1334 ];
1335
1336 (* special case of t2 in f3 *)
1337 (* this is the same as getting the reduced matrix elements from
Judd 1966 *)
1338 numE = 3;
1339 e30p = juddOperators[{3, "e_{3}"}];
1340 t2prime = juddOperators[{3, "t_{2}^{'}"}];
1341 prefactor = 1/(70 Sqrt[2]);
1342 t20p = (# -> (t2prime[#] + prefactor*e30p[#])) & /@ Keys[t2prime];
1343 t20p = Association[t20p];
1344 juddOperators[{3, "t_{2}"}] = t20p;
1345
1346 (*Special case of t11 in f3*)
1347 t11 = juddOperators[{3, "t_{11}"}];
1348 eBetaPrimeOp = juddOperators[{3, "e_{\beta}^{'}"}];
1349 t11primeOp = (# -> (t11[#] + Sqrt[3/385] eBetaPrimeOp[#])) & /@ Keys[t11];
1350 t11primeOp = Association[t11primeOp];
1351 juddOperators[{3, "t_{11}^{'}"}] = t11primeOp;
1352 If[OptionValue["Export"],
1353 (
1354 (*export them*)
PrintTemporary["Exporting ..."];
1356 exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.m"}];
1357 Export[exportFname, juddOperators];
1358 )
1359 ];
1360 Return[juddOperators];
1361 );
1362
1363 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."
;
1364 Options[GenerateThreeBodyTables] = {"Export" -> False};

```

```

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

```

```

1430 {opKey, Append[tiKeys, "e_{3}"]}]
1431 ];
1432 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " configuration complete"]];
1433 ),
1434 {numE, 1, 14}
1435 ];
1436
1437 (* Now use those reduced matrix elements to determine their sum as weighted by their corresponding strengths Ti *)
1438 ThreeBodyTable = <||>;
1439 Do[
1440 Do[
1441 (
1442 ThreeBodyTable[{numE, SL, SpLp}] = (
1443 Sum[((
1444 If[tiKey == "t_{2}", t2Switch, 1] *
1445 tktable[{numE, SL, SpLp, tiKey}] *
1446 TSymbolsAssoc[tiKey] +
1447 If[tiKey == "t_{2}", 1 - t2Switch, 0] *
1448 (-tktable[{14 - numE, SL, SpLp, tiKey}]) *
1449 TSymbolsAssoc[tiKey]
1450 ),
1451 {tiKey, tiKeys}
1452 ]
1453 );
1454 ),
1455 {SL, AllowedNKSLTerms[numE]},
1456 {SpLp, AllowedNKSLTerms[numE]}
1457 ];
1458 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix complete"]];
1459 {numE, 1, 7}
1460 ];
1461
1462 ThreeBodyTables = Table[(
1463 terms = AllowedNKSLTerms[numE];
1464 singleThreeBodyTable =
1465 Table[
1466 {SL, SLP} -> ThreeBodyTable[{numE, SL, SLP}],
1467 {SL, terms},
1468 {SLP, terms}
1469 ];
1470 singleThreeBodyTable = Flatten[singleThreeBodyTable];
1471 singleThreeBodyTables = Table[(
1472 notNullPosition = Position[TSymbols, notNullSymbol][[1,
1]];
1473 reps = ConstantArray[0, Length[TSymbols]];
1474 reps[[notNullPosition]] = 1;
1475 rep = AssociationThread[TSymbols -> reps];
1476 notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
1477 ),
1478 {notNullSymbol, TSymbols}
1479 ];
1480 singleThreeBodyTables = Association[singleThreeBodyTables];
1481 numE -> singleThreeBodyTables),
1482 {numE, 1, 7}
1483 ];
1484
1485 ThreeBodyTables = Association[ThreeBodyTables];
1486 If[OptionValue["Export"],
1487 (
1488 threeBodyTablefname = FileNameJoin[{moduleDir, "data", " ThreeBodyTable.m"}];
1489 Export[threeBodyTablefname, ThreeBodyTable];
1490 threeBodyTablesfname = FileNameJoin[{moduleDir, "data", " ThreeBodyTables.m"}];
1491 Export[threeBodyTablesfname, ThreeBodyTables];
1492 )
1493 ];
1494 Return[{ThreeBodyTable, ThreeBodyTables}];
1495 );
1496
1497 ScalarOperatorProduct::usage = "ScalarOperatorProduct[op1, op2, numE] calculated the innerproduct between the two scalar operators op1 and op2.";
```

```

1498 ScalarOperatorProduct[op1_, op2_, numE_] := Module[
1499   {terms, S, L, factor, term1, term2},
1500   (
1501     terms = AllowedNKSLTerms[numE];
1502     Simplify[
1503       Sum[(
1504         {S, L} = FindSL[term1];
1505         factor = TPO[S, L];
1506         factor * op1[{term1, term2}] * op2[{term2, term1}]
1507         ),
1508         {term1, terms},
1509         {term2, terms}
1510       ]
1511     ]
1512   );
1513 ];
1514
1515 (* ##### Three Body Operators ##### *)
1516 (* ##### *)
1517
1518 (* ##### Reduced SOO and ECSO ##### *)
1519
1520 ReducedT11inf2::usage = "ReducedT11inf2[SL, SpLp] returns the
1521   reduced matrix element of the scalar component of the double
1522   tensor T11 for the given SL terms SL, SpLp.
1523 Data used here for m0, m2, m4 is from Table II of Judd, BR, HM
1524   Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
1525   Interactions for f Electrons. Physical Review 169, no. 1 (1968):
1526   130.
1527 ";
1528 ReducedT11inf2[SL_, SpLp_] := Module[
1529   {T11inf2},
1530   (
1531     T11inf2 = <|
1532       {"1S", "3P"} -> 6 M0 + 2 M2 + 10/11 M4,
1533       {"3P", "3P"} -> -36 M0 - 72 M2 - 900/11 M4,
1534       {"3P", "1D"} -> -Sqrt[(2/15)] (27 M0 + 14 M2 + 115/11 M4),
1535       {"1D", "3F"} -> Sqrt[2/5] (23 M0 + 6 M2 - 195/11 M4),
1536       {"3F", "3F"} -> 2 Sqrt[14] (-15 M0 - M2 + 10/11 M4),
1537       {"3F", "1G"} -> Sqrt[11] (-6 M0 + 64/33 M2 - 1240/363 M4),
1538       {"1G", "3H"} -> Sqrt[2/5] (39 M0 - 728/33 M2 - 3175/363 M4),
1539       {"3H", "3H"} -> 8/Sqrt[55] (-132 M0 + 23 M2 + 130/11 M4),
1540       {"3H", "1I"} -> Sqrt[26] (-5 M0 - 30/11 M2 - 375/1573 M4)
1541     |>;
1542     Which[
1543       MemberQ[Keys[T11inf2], {SL, SpLp}],
1544         Return[T11inf2[{SL, SpLp}]],
1545       MemberQ[Keys[T11inf2], {SpLp, SL}],
1546         Return[T11inf2[{SpLp, SL}]],
1547       True,
1548         Return[0]
1549     ]
1550   )
1551 ];
1552
1553 Reducedt11inf2::usage = "Reducedt11inf2[SL, SpLp] returns the
1554   reduced matrix element in f^2 of the double tensor operator t11
1555   for the corresponding given terms {SL, SpLp}.
1556 Values given here are those from Table VII of \"Judd, BR, HM
1557   Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
1558   Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
1559   130.\"
1560 ";
1561 Reducedt11inf2[SL_, SpLp_] := Module[
1562   {t11inf2},
1563   (
1564     t11inf2 = <|
1565       {"1S", "3P"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
1566       {"3P", "3P"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
1567       {"3P", "1D"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
1568       {"1D", "3F"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
1569       {"3F", "3F"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
1570       {"3F", "1G"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
1571       {"1G", "3H"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
1572       {"3H", "3H"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
1573

```

```

1564      {"3H", "1I"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
1565      |>;
1566      Which[
1567        MemberQ[Keys[t11inf2], {SL, SpLp}], ,
1568        Return[t11inf2[{SL, SpLp}]], ,
1569        MemberQ[Keys[t11inf2], {SpLp, SL}], ,
1570        Return[t11inf2[{SpLp, SL}]], ,
1571        True, ,
1572        Return[0]
1573      ]
1574    )
1575  ];
1576
1577 ReducedSO0andECSOinf2::usage = "ReducedSO0andECSOinf2[SL, SpLp]
1578   returns the reduced matrix element corresponding to the operator (
1579     T11 + t11 - a13 * z13 / 6) for the terms {SL, SpLp}. This
1580   combination of operators corresponds to the spin-other-orbit plus
1581   ECSO interaction.
1582   The T11 operator corresponds to the spin-other-orbit interaction,
1583   and the t11 operator (associated with electrostatically-correlated
1584   spin-orbit) originates from configuration interaction analysis.
1585   To their sum a factor proportional to the operator z13 is
1586   subtracted since its effect is redundant to the spin-orbit
1587   interaction. The factor of 1/6 is not on Judd's 1968 paper, but it
1588   is on \"Chen, Xueyuan, Guokui Liu, Jean Margerie, and Michael F
1589   Reid. \"A Few Mistakes in Widely Used Data Files for Fn
1590   Configurations Calculations.\" Journal of Luminescence 128, no. 3
1591   (2008): 421-27\".
1592   The values for the reduced matrix elements of z13 are obtained from
1593   Table IX of the same paper. The value for a13 is from table VIII.
1594   Rigorously speaking the Pk parameters here are subscripted. The
1595   conversion to superscripted parameters is performed elsewhere with
1596   the Prescaling replacement rules.
1597 ";
1598 ReducedSO0andECSOinf2[SL_, SpLp_] := Module[
1599   {a13, z13, z13inf2, matElement, redSO0andECSOinf2},
1600   (
1601     a13 = (-33 M0 + 3 M2 + 15/11 M4 -
1602       6 P0 + 3/2 (35 P2 + 77 P4 + 143 P6));
1603     z13inf2 = <|
1604       {"1S", "3P"} -> 2,
1605       {"3P", "3P"} -> 1,
1606       {"3P", "1D"} -> -Sqrt[(15/2)],
1607       {"1D", "3F"} -> Sqrt[10],
1608       {"3F", "3F"} -> Sqrt[14],
1609       {"3F", "1G"} -> -Sqrt[11],
1610       {"1G", "3H"} -> Sqrt[10],
1611       {"3H", "3H"} -> Sqrt[55],
1612       {"3H", "1I"} -> -Sqrt[(13/2)]
1613     |>;
1614     matElement = Which[
1615       MemberQ[Keys[z13inf2], {SL, SpLp}],
1616       z13inf2[{SL, SpLp}],
1617       MemberQ[Keys[z13inf2], {SpLp, SL}],
1618       z13inf2[{SpLp, SL}],
1619       True,
1620       0
1621     ];
1622     redSO0andECSOinf2 = (
1623       ReducedT11inf2[SL, SpLp] +
1624       Reducedt11inf2[SL, SpLp] -
1625       a13 / 6 * matElement
1626     );
1627     redSO0andECSOinf2 = SimplifyFun[redSO0andECSOinf2];
1628     Return[redSO0andECSOinf2];
1629   )
1630 ];
1631
1632 ReducedSO0andECSOinf2::usage = "ReducedSO0andECSOinf2[numE, SL,
1633   SpLp] calculates the reduced matrix elements of the (spin-other-
1634   orbit + ECSO) operator for the f^numE configuration corresponding
1635   to the terms SL and SpLp. This is done recursively, starting from
1636   tabulated values for f^2 from \"Judd, BR, HM Crosswhite, and
1637   Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
1638   Electrons.\" Physical Review 169, no. 1 (1968): 130.\", and by
1639   using equation (4) of that same paper.

```

```

1617 ";
1618 ReducedSOOandECSOinfn[numE_, SL_, SpLp_] := Module[
1619   {spin, orbital, t, S, L, Sp, Lp,
1620   idx1, idx2, cfpSL, cfpSpLp, parentSL,
1621   Sb, Lb, Sbp, Lbp, parentSpLp, funval},
1622   (
1623     {spin, orbital} = {1/2, 3};
1624     {S, L} = FindSL[SL];
1625     {Sp, Lp} = FindSL[SpLp];
1626     t = 1;
1627     cfpSL = CFP[{numE, SL}];
1628     cfpSpLp = CFP[{numE, SpLp}];
1629     funval = Sum[
1630       (
1631         parentSL = cfpSL[[idx2, 1]];
1632         parentSpLp = cfpSpLp[[idx1, 1]];
1633         {Sb, Lb} = FindSL[parentSL];
1634         {Sbp, Lbp} = FindSL[parentSpLp];
1635         phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1636         (
1637           phase *
1638             cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1639             SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1640             SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1641             SOOandECSOLSTable[{numE - 1, parentSL, parentSpLp}]
1642         )
1643       ),
1644       {idx1, 2, Length[cfpSpLp]},
1645       {idx2, 2, Length[cfpSL]}
1646     ];
1647     funval *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1648     Return[funval];
1649   )
1650 ];
1651
1652 GenerateSOOandECSOLSTable::usage = "GenerateSOOandECSOLSTable[nmax]
generates the LS reduced matrix elements of the spin-other-orbit
+ ECSO for the f^n configurations up to n=nmax. The values for n=1
and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
Physical Review 169, no. 1 (1968): 130.\", and the values for n
>2 are calculated recursively using equation (4) of that same
paper. The values are then exported to a file \
ReducedSOOandECSOLSTable.m\" in the data folder of this module.
The values are also returned as an association.";
1653 Options[GenerateSOOandECSOLSTable] = {"Progress" -> True, "Export"
-> True};
1654 GenerateSOOandECSOLSTable[nmax_Integer, OptionsPattern[]] := (
1655   If[And[OptionValue["Progress"], frontEndAvailable],
1656   (
1657     numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
1658       numE]]^2, {numE, 1, nmax}]];
1659     counters = Association[Table[numE->0, {numE, 1, nmax}]];
1660     totalIters = Total[Values[numItersai[[1;;nmax]]]];
1661     template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
1662     template2 = StringTemplate["`remtime` min remaining"];
1663     template3 = StringTemplate["Iteration speed = `speed` ms/it"];
1664     template4 = StringTemplate["Time elapsed = `runtime` min"];
1665     progBar = PrintTemporary[
1666       Dynamic[
1667         Pane[
1668           Grid[{
1669             {Superscript["f", numE]},
1670             {template1[<|"numiter" -> numiter, "totaliter" ->
1671               totalIters|>]},
1672             {template4[<|"runtime" -> Round[QuantityMagnitude[
1673               UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
1674             {template2[<|"remtime" -> Round[QuantityMagnitude[
1675               UnitConvert[(Now - startTime)/(numiter)*(totalIters - numiter), "min"]
1676             ], 0.1]|>]},
1677             {template3[<|"speed" -> Round[QuantityMagnitude[Now -
1678               startTime, "ms"]/(numiter), 0.01]|>]},
1679             {ProgressIndicator[Dynamic[
1680               numiter], {1, totalIters}]}
1681           }],
1682           Frame -> All
1683         ]
1684       ]
1685     ]
1686   )
1687 
```

```

1674     ],
1675     Full,
1676     Alignment -> Center
1677   ]
1678   ]
1679 ];
1680 )
1681 ];
S0OandECSOLSTable = <|||>;
1683 numiter = 1;
1684 startTime = Now;
1685 Do[
1686 (
1687   numiter+= 1;
1688   S0OandECSOLSTable[{numE, SL, SpLp}] = Which[
1689     numE==1,
1690     0,
1691     numE==2,
1692     SimplifyFun[ReducedS0OandECS0inf2[SL, SpLp]],
1693     True,
1694     SimplifyFun[ReducedS0OandECS0inf[n, SL, SpLp]]
1695   ];
1696   ),
1697 {numE, 1, nmax},
1698 {SL, AllowedNKSLTerms[numE]},
1699 {SpLp, AllowedNKSLTerms[numE]}
1700 ];
1701 If[And[OptionValue["Progress"], frontEndAvailable],
1702   NotebookDelete[progBar]];
1703 If[OptionValue["Export"],
1704   (fname = FileNameJoin[{moduleDir, "data", "
1705 ReducedS0OandECSOLSTable.m"}];
1706   Export[fname, S0OandECSOLSTable];
1707   )
1708 ];
1709 Return[S0OandECSOLSTable];
1710 );
1711 (* ##### Reduced S0O and ECS0 ##### *)
1712 (* ##### ##### ##### ##### ##### *)
1713 (* ##### ##### ##### ##### ##### *)
1714 (* ##### ##### ##### ##### ##### *)
1715 (* ##### ##### ##### ##### *)
1716
1717 ReducedT22inf2::usage = "ReducedT22inf2[SL, SpLp] returns the
1718   reduced matrix element of the scalar component of the double
1719   tensor T22 for the terms SL, SpLp in f^2.
Data used here for m0, m2, m4 is from Table I of Judd, BR, HM
1720   Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
1721   Interactions for f Electrons. Physical Review 169, no. 1 (1968):
1722   130.
1723 ";
1724 ReducedT22inf2[SL_, SpLp_] := Module[
1725   {statePosition, PsiPsipStates, m0, m2, m4, Tk2m},
1726   (
1727     T22inf2 = <|
1728       {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
1729       {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
1730       {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
1731       {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
1732       {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
1733     |>;
1734     Which[
1735       MemberQ[Keys[T22inf2], {SL, SpLp}],
1736         Return[T22inf2[{SL, SpLp}]],
1737       MemberQ[Keys[T22inf2], {SpLp, SL}],
1738         Return[T22inf2[{SpLp, SL}]],
1739       True,
1740         Return[0]
1741     ]
1742   )
1743 ];
1744
1745 ReducedT22inf[n, SL, SpLp]::usage = "ReducedT22inf[n, SL, SpLp] calculates the
1746   reduced matrix element of the T22 operator for the f^n
1747   configuration corresponding to the terms SL and SpLp. This is the

```

```

1742   operator corresponding to the inter-electron between spin.
1743   It does this by using equation (4) of \"Judd, BR, HM Crosswhite,
1744   and Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
1745   Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
1746
1747 ";
1748 ReducedT22inf[n[E_, SL_, SpLp_]] := Module[
1749   {spin, orbital, t, idx1, idx2, S, L,
1750   Sp, Lp, cfpSL, cfpSpLp, parentSL,
1751   parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
1752   (
1753     {spin, orbital} = {1/2, 3};
1754     {S, L} = FindSL[SL];
1755     {Sp, Lp} = FindSL[SpLp];
1756     t = 2;
1757     cfpSL = CFP[{numE, SL}];
1758     cfpSpLp = CFP[{numE, SpLp}];
1759     Tnkk = Sum[(
1760       parentSL = cfpSL[[idx2, 1]];
1761       parentSpLp = cfpSpLp[[idx1, 1]];
1762       {Sb, Lb} = FindSL[parentSL];
1763       {Sbp, Lbp} = FindSL[parentSpLp];
1764       phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1765       (
1766         phase *
1767         cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1768         SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1769         SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1770         T22Table[{numE - 1, parentSL, parentSpLp}]
1771       )
1772     ),
1773     {idx1, 2, Length[cfpSpLp]},
1774     {idx2, 2, Length[cfpSL]}
1775   ];
1776   Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1777   Return[Tnkk];
1778 ]
1779
1780 GenerateT22Table::usage = "GenerateT22Table[nmax] generates the LS
1781 reduced matrix elements for the double tensor operator T22 in f^n
1782 up to n=nmax. If the option \"Export\" is set to true then the
1783 resulting association is saved to the data folder. The values for
1784 n=1 and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
1785 Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
1786 \" Physical Review 169, no. 1 (1968): 130.\", and the values for n
1787 >2 are calculated recursively using equation (4) of that same
1788 paper.
1789 This is an intermediate step to the calculation of the reduced
1790 matrix elements of the spin-spin operator.";
1791 Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
1792 GenerateT22Table[nmax_Integer, OptionsPattern[]] := (
1793   If[And[OptionValue["Progress"], frontEndAvailable],
1794     (
1795       numItersai = Association[Table[numE -> Length[AllowedNKSLTerms[
1796       numE]]^2, {numE, 1, nmax}]];
1797       counters = Association[Table[numE -> 0, {numE, 1, nmax}]];
1798       totalIters = Total[Values[numItersai[[1;;nmax]]]];
1799       template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
2000       template2 = StringTemplate["`remtime` min remaining"];
2001       template3 = StringTemplate["Iteration speed = `speed` ms/it"];
2002       template4 = StringTemplate["Time elapsed = `runtime` min"];
2003       progBar = PrintTemporary[
2004         Dynamic[
2005           Pane[
2006             Grid[{{Superscript["f", numE]}, {
2007               template1 <| "numiter" -> numiter, "totaliter" ->
2008               totalIters |>}},
2009               {template4 <| "runtime" -> Round[QuantityMagnitude[
2010                 UnitConvert[(Now - startTime), "min"]], 0.1] |>},
2011               {template2 <| "remtime" -> Round[QuantityMagnitude[
2012                 UnitConvert[(Now - startTime)/(numiter)*(totalIters - numiter), "min"]], 0.1] |>},
2013               {template3 <| "speed" -> Round[QuantityMagnitude[Now -
2014                 startTime, "ms"]/(numiter), 0.01] |>}],
2015             {ProgressIndicator[Dynamic[numiter], {1,
2016               numItersai["totaliter"]}], "Control"}]
2017           ]];
2018         ];
2019       ];
2020     ];
2021   ];
2022 ];
2023 
```

```

    totalIters}]]},
1798           Frame -> All],
1799           Full,
1800           Alignment -> Center]
1801       ]
1802     ];
1803   );
1804 ];
1805 T22Table = <||>;
1806 startTime = Now;
1807 numiter = 1;
1808 Do[
1809 (
1810   numiter+= 1;
1811   T22Table[{numE, SL, SpLp}] = Which[
1812     numE==1,
1813     0,
1814     numE==2,
1815     SimplifyFun[ReducedT22inf2[SL, SpLp]],
1816     True,
1817     SimplifyFun[ReducedT22infn[numE, SL, SpLp]]
1818   ];
1819   ),
1820 {numE, 1, nmax},
1821 {SL, AllowedNKSLTerms[numE]},
1822 {SpLp, AllowedNKSLTerms[numE]}
1823 ];
1824 If[And[OptionValue["Progress"], frontEndAvailable],
1825   NotebookDelete[progBar]
1826 ];
1827 If[OptionValue["Export"],
1828 (
1829   fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
1830   Export[fname, T22Table];
1831 )
1832 ];
1833 Return[T22Table];
1834 );
1835
1836 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.
1837 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.\""
1838 ".
1839 ";
1840 SpinSpin[numE_, SL_, SpLp_, J_] := Module[
1841   {S, L, Sp, Lp, α, val},
1842   (
1843     α = 2;
1844     {S, L} = FindSL[SL];
1845     {Sp, Lp} = FindSL[SpLp];
1846     val = (
1847       Phaser[Sp + L + J] *
1848       SixJay[{Sp, Lp, J}, {L, S, α}] *
1849       T22Table[{numE, SL, SpLp}]
1850     );
1851     Return[val]
1852   )
1853 ];
1854
1855 GenerateSpinSpinTable::usage = "GenerateSpinSpinTable[nmax]
generates the reduced matrix elements in the |LSJ> basis for the (
spin-other-orbit + electrostatically-correlated-spin-orbit)
operator. It returns an association where the keys are of the form
{numE, SL, SpLp, J}. If the option \"Export\" is set to True then
the resulting object is saved to the data folder. Since this is a
scalar operator, there is no MJ dependence. This dependence only
comes into play when the crystal field contribution is taken into
account.";
```

```

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

```

```

1908 SpLp, J] /. Prescaling),,
1909 {numE, 1, nmax},
1910 {J, MinJ[numE], MaxJ[numE]},
1911 {SL, First /@ AllowedNKSLforJTerms[numE, J]},,
1912 {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
];
1913 If[OptionValue["Export"],
(
1915 fname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
1916 Export[fname, SOOandECSOTable];
)
];
1918 ];
1919 Return[SOOandECSOTable];
);
1921
1922 (* ## Spin-Other-Orbit and Electrostatically-Correlated-Spin-Orbit
## *)
1923 (*
#####
1924 (* #####
1925 (* #####
1926 (* #####
1927 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.
1929 The part corresponding to the spin-spin interaction is provided by
SpinSpin[{numE, SL, SLP, J}].
1930 The part corresponding to SOO and ECSO is provided by the function
SOOandECSO[{numE, SL, SLP, J}].
1931 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 LoadChen[].";
1932 Options[MagneticInteractions] = {"ChenDeltas" -> False};
1933 MagneticInteractions[{numE_, SL_, SLP_, J_}, OptionsPattern[]] :=
Module[
1934 {key, ss, sooandecso, total,
1935 S, L, Sp, Lp, phase, sixjay,
1936 MOv, M2v, M4v,
1937 P2v, P4v, P6v},
(
1939 key      = {numE, SL, SLP, J};
1940 ss       = \[\[Sigma]\]SS * SpinSpinTable[key];
1941 sooandecso = SOOandECSOTable[key];
1942 total = ss + sooandecso;
1943 total = SimplifyFun[total];
1944 If[
1945   Not[OptionValue["ChenDeltas"]],
1946   Return[total]
];
1947 (* In the type A errors the wrong values are different *)
1948 If[MemberQ[Keys[chenDeltas["A"]], {numE, SL, SLP}],
(
1951 {S, L}    = FindSL[SL];
1952 {Sp, Lp}  = FindSL[SLP];
1953 phase    = Phaser[Sp + L + J];
1954 sixjay   = SixJay[{Sp, Lp, J}, {L, S, 1}];
1955 {MOv, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SL,
SLP}]["wrong"];
1956 total    = (
1957   phase * sixjay *
1958   (
1959     MOv*MO + M2v*M2 + M4v*M4 +
1960     P2v*P2 + P4v*P4 + P6v*P6
1961   )
1962 );
1963 total    = wChErrA * total + (1 - wChErrA) * (ss +
1964 sooandecso
1965 )
];
1966 (* In the type B errors the wrong values are zeros all around

```

```

*)
1967   If [MemberQ [chenDeltas ["B"], {numE, SL, SpLp}],
1968     (
1969       total = (1 - wChErrB) * (ss + sooandecso)
1970     )
1971   ];
1972   Return [total];
1973 )
1974 ];
1975
1976 (* ##### Magnetic Interactions #### *)
1977 (* ##### ##### ##### ##### ##### ##### ##### ##### ##### ##### *)
1978
1979 (* ##### ##### ##### ##### ##### ##### ##### ##### ##### ##### *)
1980 (* ##### ##### ##### ##### Free-Ion Energies ##### ##### ##### *)
1981
1982 GenerateFreeIonTable::usage="GenerateFreeIonTable[] generates an
1983   association for free-ion energies in terms of Slater integrals Fk
1984   and spin-orbit parameter  $\zeta$ . It returns an association where the
1985   keys are of the form {nE, SL, SpLp}. If the option \"Export\" is
1986   set to True then the resulting object is saved to the data folder.
1987   The free-ion Hamiltonian is the sum of the electrostatic and spin-
1988   -orbit interactions. The electrostatic interaction is given by the
1989   function Electrostatic[{numE, SL, SpLp}] and the spin-orbit
1990   interaction is given by the function SpinOrbitTable[{numE, SL,
1991   SpLp}]. The values for the electrostatic interaction are taken
1992   from the data file ElectrostaticTable.m and the values for the
1993   spin-orbit interaction are taken from the data file SpinOrbitTable.
1994   m. The values for the free-ion Hamiltonian are then exported to a
1995   file \"FreeIonTable.m\" in the data folder of this module. The
1996   values are also returned as an association.";
1997 Options[GenerateFreeIonTable] = {"Export" -> False};
1998 GenerateFreeIonTable[OptionsPattern[]} := Module[
1999   {terms, numEH, zetaSign, fname, FreeIonTable},
2000   (
2001     If [Not [ValueQ [ElectrostaticTable]],
2002       LoadElectrostatic []
2003     ];
2004     If [Not [ValueQ [SpinOrbitTable]],
2005       LoadSpinOrbit []
2006     ];
2007     If [Not [ValueQ [ReducedUkTable]],
2008       LoadUk []
2009     ];
2010     FreeIonTable = <||>;
2011     Do [
2012       (
2013         terms = AllowedNKSLJTerms[nE];
2014         numEH = Min[nE, 14 - nE];
2015         zetaSign = If[nE > 7, -1, 1];
2016         Do [
2017           FreeIonTable[{nE, term[[1]], term[[2]]}] = (
2018             Electrostatic[{numEH, term[[1]], term[[1]]}] +
2019               zetaSign * SpinOrbitTable[{numEH, term[[1]], term
2020                 [[1]], term[[2]]}]
2021               ),
2022               {term, terms}];
2023             ),
2024             {nE, 1, 14}
2025           ];
2026           If [OptionValue["Export"],
2027             (
2028               fname = FileNameJoin[{moduleDir, "data", "FreeIonTable.m"
2029             }];
2030               Export [fname, FreeIonTable];
2031             )
2032           ];
2033           Return [FreeIonTable];
2034         )
2035       ];
2036     ];
2037   ];
2038
2039 LoadFreeIon::usage = "LoadFreeIon[] loads the free-ion energies
2040   from the data folder. The values are stored in the association
2041   FreeIonTable.";
2042 LoadFreeIon[] := (
2043   If [ValueQ [FreeIonTable],

```

```

2024     Return []
2025   ];
2026   PrintTemporary["Loading the association of free-ion energies ..."
2027 ];
2027   FreeIonTableFname = FileNameJoin[{moduleDir, "data", "FreeIonTable.m"}];
2028   FreeIonTable = If[!FileExistsQ[FreeIonTableFname],
2029   (
2030     PrintTemporary[">> FreeIonTable.m not found, generating ..."]
2031   ];
2031   GenerateFreeIonTable["Export" -> True]
2032   ),
2033   Import[FreeIonTableFname]
2034 ];
2035 );
2036
2037 (* ##### Free-Ion Energies ##### *)
2038 (* ##### *)
2039 (* ##### *)
2040 (* ##### *)
2041 (* ##### Crystal Field ##### *)
2042
2043 Cqk::usage = "Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_]. In
2044 Wybourne (1965) see equations 6-3, 6-4, and 6-5. Also in TASS see
2045 equation 11.53.";
2046 Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
2047 {S, Sp, L, Lp, orbital, val},
2048 (
2049   orbital = 3;
2050   {S, L} = FindSL[NKSL];
2051   {Sp, Lp} = FindSL[NKSLp];
2052   f1 = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
2053   val =
2054   If[f1==0,
2055     0,
2056     (
2057       f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
2058       If[f2==0,
2059         0,
2060         (
2061           f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
2062           If[f3==0,
2063             0,
2064             (
2065               Phaser[J - M + S + Lp + J + k] *
2066               Sqrt[TPO[J, Jp]] *
2067               f1 *
2068               f2 *
2069               f3 *
2070               Ck[orbital, k]
2071             )
2072           )
2073         )
2074       ]
2075     );
2076   ];
2077   Return[val];
2078 )
2079 ];
2080
2081 Bqk::usage = "Real part of the Bqk coefficients.";
2082 Bqk[q_, 2] := {B02/2, B12, B22}[[q + 1]];
2083 Bqk[q_, 4] := {B04/2, B14, B24, B34, B44}[[q + 1]];
2084 Bqk[q_, 6] := {B06/2, B16, B26, B36, B46, B56, B66}[[q + 1]];
2085
2086 Sqk::usage = "Imaginary part of the Bqk coefficients.";
2087 Sqk[q_, 2] := {0, S12, S22}[[q + 1]];
2088 Sqk[q_, 4] := {0, S14, S24, S34, S44}[[q + 1]];
2089 Sqk[q_, 6] := {0, S16, S26, S36, S46, S56, S66}[[q + 1]];
2090
2091 CrystalField::usage = "CrystalField[n, NKSL, J, M, NKSLp, Jp, Mp]
2092 gives the general expression for the matrix element of the crystal
2093 field Hamiltonian parametrized with Bqk and Sqk coefficients as a
2094 sum over spherical harmonics Cqk.

```

```

2092 Sometimes this expression only includes Bqk coefficients, see for
2093 example eqn 6-2 in Wybourne (1965), but one may also split the
2094 coefficient into real and imaginary parts as is done here, in an
2095 expression that is patently Hermitian.";
2096 CrystalField[numE_, NKS L_, J_, M_, NKS Lp_, Jp_, Mp_] := (
2097   Sum[
2098     (
2099       c qk = C qk[numE, q, k, NKS L, J, M, NKS Lp, Jp, Mp];
2100       cm qk = C qk[numE, -q, k, NKS L, J, M, NKS Lp, Jp, Mp];
2101       B qk[q, k] * (c qk + (-1)^q * cm qk) +
2102       I * S qk[q, k] * (c qk - (-1)^q * cm qk)
2103     ),
2104     {k, {2, 4, 6}},
2105     {q, 0, k}
2106   )
2107 )
2108
2109 TotalCFIters::usage = "TotalIters[i, j] returns total number of
2110 function evaluations for calculating all the matrix elements for
2111 the  $\forall (\ast \text{SuperscriptBox}[(f), (i)])$  to the  $\forall (\ast \text{SuperscriptBox}[(f), (j)])$  configurations.";
2112 TotalCFIters[i_, j_] := (
2113   numIters = {196, 8281, 132496, 1002001, 4008004, 9018009,
2114   11778624};
2115   Return[Total[numIters[[i ;; j]]]];
2116 )
2117
2118 GenerateCrystalFieldTable::usage = "GenerateCrystalFieldTable[{numEs}] computes the matrix values for the crystal field
2119 interaction for  $f^n$  configurations the given list of numE in
2120 numEs. The function calculates the association CrystalFieldTable
2121 with keys of the form {numE, NKS L, J, M, NKS Lp, Jp, Mp}. If the
2122 option \"Export\" is set to True, then the result is exported to
2123 the data subfolder for the folder in which this package is in. If
2124 the option \"Progress\" is set to True then an interactive
2125 progress indicator is shown. If \"Compress\" is set to true the
2126 exported values are compressed when exporting.";
2127 Options[GenerateCrystalFieldTable] = {"Export" -> False, "Progress"
2128   -> True, "Compress" -> True};
2129 GenerateCrystalFieldTable[numEs_List:{1,2,3,4,5,6,7},
2130   OptionsPattern[]] :=
2131   ExportFun =
2132   If[OptionValue["Compress"],
2133     ExportMZip,
2134     Export
2135   ];
2136   numiter = 1;
2137   template1 = StringTemplate["Iteration `numiter` of `totaliter`"]
2138   template2 = StringTemplate["`remtime` min remaining"];
2139   template3 = StringTemplate["Iteration speed = `speed` ms/it"];
2140   template4 = StringTemplate["Time elapsed = `runtime` min"];
2141   totalIter = Total[TotalCFIters[#, #] & /@ numEs];
2142   freebies = 0;
2143   startTime = Now;
2144   If[And[OptionValue["Progress"], frontEndAvailable],
2145     progBar = PrintTemporary[
2146       Dynamic[
2147         Pane[
2148           Grid[
2149             {
2150               {Superscript["f", numE]},
2151               {template1[<|"numiter" -> numiter, "totaliter" ->
2152             totalIter|>]},
2153               {template4[<|"runtime" -> Round[QuantityMagnitude[
2154             UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
2155               {template2[<|"remtime" -> Round[QuantityMagnitude[
2156             UnitConvert[(Now - startTime)/(numiter - freebies) * (totalIter -
2157             numiter), "min"]], 0.1]|>]},
2158               {template3[<|"speed" -> Round[QuantityMagnitude[Now -
2159             startTime, "ms"]/(numiter - freebies), 0.01]|>]},
2160               {ProgressIndicator[Dynamic[numiter], {1, totalIter}]}
2161             },
2162             Frame -> All
2163           ],
2164           Full,
2165           Alignment -> Center
2166         ]
2167       ]
2168     ]
2169   ]
2170 
```

```

2145      ]
2146      ]
2147    ];
2148  ];
2149 Do[
2150 (
2151   exportFname = FileNameJoin[{moduleDir, "data", "
2152 CrystalFieldTable_f" <> ToString[numE] <> ".m"}];
2153   If[FileExistsQ[exportFname],
2154     Print["File exists, skipping ..."];
2155     numiter += TotalCFITers[numE, numE];
2156     freebies += TotalCFITers[numE, numE];
2157     Continue[];
2158   ];
2159   CrystalFieldTable = <||>;
2160 Do[
2161 (
2162   numiter += 1;
2163   CrystalFieldTable[{numE, NKSL, J, M, NKSLP, Jp, Mp}] =
2164   CrystalField[numE, NKSL, J, M, NKSLP, Jp, Mp];
2165   ),
2166   {J, MinJ[numE], MaxJ[numE]},
2167   {Jp, MinJ[numE], MaxJ[numE]},
2168   {M, AllowedMforJ[J]},
2169   {Mp, AllowedMforJ[Jp]},
2170   {NKSL, First /@ AllowedNKSLforJTerms[numE, J]},
2171   {NKSLP, First /@ AllowedNKSLforJTerms[numE, Jp]}
2172 ];
2173 If[And[OptionValue["Progress"], frontEndAvailable],
2174   NotebookDelete[progBar]
2175 ];
2176 If[OptionValue["Export"],
2177 (
2178   Print["Exporting to file "<>ToString[exportFname]];
2179   ExportFun[exportFname, CrystalFieldTable];
2180 )
2181 ],
2182 {numE, numEs}
2183 ]
2184 )
2185 Options[ParseBenelli2015] = {"Export" -> False};
2186 ParseBenelli2015[OptionsPattern[]] := Module[
2187   {fname, crystalSym,
2188   crystalSymmetries, parseFun,
2189   chars, qk, groupName, family,
2190   groupNum, params},
2191 (
2192   fname = FileNameJoin[{moduleDir, "data",
2193   benelli_and_gatteschi_table3p3.csv"}];
2194   crystalSym = Import[fname][[2;;33]];
2195   crystalSymmetries = <||>;
2196   parseFun[txt_] := (
2197     chars = Characters[txt];
2198     qk = chars[[-2;;]];
2199     If[chars[[1]] == "R",
2200       (
2201         Return[{ToExpression@StringJoin[{"B", qk[[1]], qk[[2]]}]}]
2202       ),
2203       (
2204         If[qk[[1]] == "O",
2205           Return[{ToExpression@StringJoin[{"B", qk[[1]], qk[[2]]}]}]
2206         ];
2207         Return[{
2208           ToExpression@StringJoin[{"B", qk[[1]], qk[[2]]}],
2209           ToExpression@StringJoin[{"S", qk[[1]], qk[[2]]}]
2210         }]
2211       );
2212     Do[
2213 (
2214       groupNum = Round@ToExpression@row[[1]];
2215       groupName = row[[2]];
2216       family = row[[3]];
2217       params = Select[row[[4;;]], # != "&"];

```

```

2218     params      = parseFun/@params;
2219     params      = <|"BqkSqk" -> Sort@Flatten[params],
2220     "aliases" -> {groupNum},
2221     "constraints" -> {} |>;
2222 If [MemberQ [{"T", "Th", "O", "Td", "Oh"}, groupName],
2223   params["constraints"] = {
2224     {B44 -> Sqrt[5/14] B04, B46 -> -Sqrt[7/2] B06},
2225     {B44 -> -Sqrt[5/14] B04, B46 -> Sqrt[7/2] B06}
2226     }
2227 ];
2228 If [StringContainsQ[groupName, ","],
2229 (
2230   {alias1, alias2}           = StringSplit[groupName, ","];
2231   crystalSymmetries[alias1] = params;
2232   crystalSymmetries[alias1]["aliases"] = {groupNum, alias2};
2233   crystalSymmetries[alias2] = params;
2234   crystalSymmetries[alias2]["aliases"] = {groupNum, alias1};
2235 ),
2236 (
2237   crystalSymmetries[groupName] = params;
2238 )
2239 ]
2240 ),
2241 {row, crystalSymm}];
2242 crystalSymmetries["source"] = "Benelli and Gatteschi, 2015,
Introduction to Molecular Magnetism, table 3.3.";
2243 If[OptionValue["Export"],
2244   Export[FileNameJoin[{moduleDir, "data",
2245   crystalFieldFunctionalForms.m"}], crystalSymmetries];
2246 ];
2247 Return[crystalSymmetries];
2248 )
2249 ]
2250 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.
2251
2252 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.
2253
2254 The returned association has three keys:
2255   \"BqkSqk\" whose value is a list with the nonzero Bqk and Sqk
parameters;
2256   \"constraints\" whose value is either an empty list, or a lists
of replacements rules that are constraints on the Bqk and Sqk
parameters;
2257   \"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;
2258   \"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.
2259
2260 This uses data from table 3.3 in Benelli and Gatteschi, 2015.";
2261 CrystalFieldForm[symmetryGroupString_] := (
2262   If[Not@ValueQ[crystalFieldFunctionalForms],
2263     crystalFieldFunctionalForms = Import[FileNameJoin[{moduleDir, "data",
2264       "crystalFieldFunctionalForms.m"}]];
2265   ];
2266   cfForm = crystalFieldFunctionalForms[symmetryGroupString];
2267   simplifier = Association[(# -> 0) & /@ Complement[cfSymbols,
2268     cfForm["BqkSqk"]]];
2269   Return[Join[cfForm, <|"simplifier" -> simplifier|>]];
2270 )
2271 (* ##### Crystal Field ##### *)
2272 (* ##### Configuration-Interaction via Casimir Operators ##### *)
2273 (* ##### Configuration-Interaction via Casimir Operators ##### *)
2274
2275 CasimirS03::usage = "CasimirS03[SL, SpLp] returns LS reduced matrix

```

```

2277     element of the configuration interaction term corresponding to
2278     the Casimir operator of R3.";
2279     CasimirS03[{SL_, SpLp_}] := (
2280       {S, L} = FindSL[SL];
2281       If[SL == SpLp,
2282         α * L * (L + 1),
2283         0
2284       ]
2285     )
2286
2285 GG2U::usage = "GG2U is an association whose keys are labels for the
2286   irreducible representations of group G2 and whose values are the
2286   eigenvalues of the corresponding Casimir operator.
2286 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
2299   element of the configuration interaction term corresponding to the
2299   Casimir operator of G2.";
2300     CasimirG2[{SL_, SpLp_}] := (
2301       Ulabel = FindNKLSTerm[SL][[1]][[4]];
2302       If[SL==SpLp,
2303         β * GG2U[Ulabel],
2304         0
2305       ]
2306     )
2307
2308 GS07W::usage = "GS07W is an association whose keys are labels for
2308   the irreducible representations of group R7 and whose values are
2308   the eigenvalues of the corresponding Casimir operator.
2309 Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\",
2309   table 2-7.";
2310     GS07W := Association[
2311       {
2312         "000" -> 0,
2313         "100" -> 3/5,
2314         "110" -> 5/5,
2315         "111" -> 6/5,
2316         "200" -> 7/5,
2317         "210" -> 9/5,
2318         "211" -> 10/5,
2319         "220" -> 12/5,
2320         "221" -> 13/5,
2321         "222" -> 15/5
2322       }
2323     ];
2324
2325 CasimirS07::usage = "CasimirS07[SL, SpLp] returns the LS reduced
2325   matrix element of the configuration interaction term corresponding
2325   to the Casimir operator of R7.";
2326     CasimirS07[{SL_, SpLp_}] := (
2327       Wlabel = FindNKLSTerm[SL][[1]][[3]];
2328       If[SL==SpLp,
2329         γ * GS07W[Wlabel],
2330         0
2331       ]
2332     )
2333
2334 ElectrostaticConfigInteraction::usage =
2334   ElectrostaticConfigInteraction[{SL, SpLp}] returns the matrix
2334   element for configuration interaction as approximated by the
2334   Casimir operators of the groups R3, G2, and R7. SL and SpLp are
2334   strings that represent terms under LS coupling.";
2335     ElectrostaticConfigInteraction[{SL_, SpLp_}] := Module[
2336       {S, L, val},

```

```

2337 (
2338     {S, L} = FindSL[SL];
2339     val = (
2340         If[SL == SpLp,
2341             CasimirS03[{SL, SL}] +
2342             CasimirS07[{SL, SL}] +
2343             CasimirG2[{SL, SL}],
2344             0
2345         ]
2346     );
2347     ElectrostaticConfigInteraction[{S, L}] = val;
2348     Return[val];
2349 )
2350 ];
2351
2352 (* ##### Configuration-Interaction via Casimir Operators ##### *)
2353 (* ##### ##### ##### ##### ##### ##### ##### ##### ##### ##### *)
2354
2355 (* ##### ##### ##### ##### ##### ##### ##### ##### ##### ##### *)
2356 (* ##### ##### ##### ##### Block assembly ##### ##### ##### *)
2357
2358 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.";
2359 Options[JJBlockMatrix] = {"Sparse" -> True, "ChenDeltas" -> False};
2360 JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]] := Module[
2361     {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
2362      SLterm, SpLpterm,
2363      MJ, MJp,
2364      subKron, matValue, eMatrix},
2365     (
2366         NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
2367         NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
2368         eMatrix =
2369             Table[
2370                 (*Condition for a scalar matrix op*)
2371                 SLterm = NKSLJM[[1]];
2372                 SpLpterm = NKSLJMp[[1]];
2373                 MJ = NKSLJM[[3]];
2374                 MJp = NKSLJMp[[3]];
2375                 subKron = (
2376                     KroneckerDelta[J, Jp] *
2377                     KroneckerDelta[MJ, MJp]
2378                 );
2379                 matValue =
2380                     If[subKron == 0,
2381                         0,
2382                         (
2383                             ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2384                             ElectrostaticConfigInteraction[{SLterm, SpLpterm}]
2385                         +
2386                             SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2387                             MagneticInteractions[{numE, SLterm, SpLpterm, J}],
2388                             "ChenDeltas" -> OptionValue["ChenDeltas"] +
2389                             ThreeBodyTable[{numE, SLterm, SpLpterm}]
2390                         )
2391                     ];
2392                     matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp}];
2393                 ];
2394                 matValue,
2395                 {NKSLJMp, NKSLJMps},
2396                 {NKSLJM, NKSLJMs}
2397             ];
2398             If[OptionValue["Sparse"],
2399                 eMatrix = SparseArray[eMatrix]
2400             ];
2401             Return[eMatrix]
2402         ];
2403     EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some

```

```

2404     point may be used to redefine states used in basis.";
2405 EnergyStates[numE_, J_]:= AllowedNKSLJMforJTerms[numE, J];
2406
2407 JJBlockMatrixFileName::usage = "JJBlockMatrixFileName[numE] gives
2408 the filename for the energy matrix table for an atom with numE f-
2409 electrons. The function admits an optional parameter \
2410 FilenameAppendix" which can be used to modify the filename.";
2411 Options[JJBlockMatrixFileName] = {"FilenameAppendix" -> ""};
2412 JJBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
2413   fileApp = OptionValue["FilenameAppendix"];
2414   fname = FileNameJoin[{moduleDir,
2415     "hams",
2416     StringJoin[{"f", ToString[numE], "_JJBlockMatrixTable",
2417     fileApp, ".m"}]}];
2418   Return[fname];
2419 );
2420
2421 TabulateJJBlockMatrixTable::usage = "TabulateJJBlockMatrixTable[
2422 numE, I] returns a list with three elements {JJBlockMatrixTable,
2423 EnergyStatesTable, AllowedM}. JJBlockMatrixTable is an association
2424 with keys equal to lists of the form {numE, J, Jp}.
2425 EnergyStatesTable is an association with keys equal to lists of
2426 the form {numE, J}. AllowedM is another association with keys
2427 equal to lists of the form {numE, J} and values equal to lists
2428 equal to the corresponding values of MJ. It's unnecessary (and it
2429 won't work in this implementation) to give numE > 7 given the
2430 equivalency between electron and hole configurations.";
2431 Options[TabulateJJBlockMatrixTable] = {"Sparse" -> True, "ChenDeltas" -> False};
2432 TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]] := (
2433   JJBlockMatrixTable = <||>;
2434   totalIterations = Length[AllowedJ[numE]]^2;
2435   template1 = StringTemplate["Iteration `numiter` of `totaliter`"]
2436   template2 = StringTemplate["`remtime` min remaining"];
2437   template4 = StringTemplate["Time elapsed = `runtime` min"];
2438   numiter = 0;
2439   startTime = Now;
2440   If[$FrontEnd != Null,
2441     (
2442       temp = PrintTemporary[
2443         Dynamic[
2444           Grid[
2445             {
2446               {template1[<|"numiter" -> numiter, "totaliter" ->
2447                 totalIterations|>]},
2448               {template2[<|"remtime" -> Round[QuantityMagnitude[
2449                 UnitConvert[(Now - startTime)/(Max[1, numiter])*(totalIterations -
2450                 numiter), "min"]], 0.1]|>]},
2451               {template4[<|"runtime" -> Round[QuantityMagnitude[
2452                 UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
2453               {ProgressIndicator[numiter, {1, totalIterations}]}
2454             }
2455           ]
2456         ];
2457     );
2458   ];
2459   Do[
2460     (
2461       JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp,
2462       CFTable, "Sparse" -> OptionValue["Sparse"], "ChenDeltas" ->
2463       OptionValue["ChenDeltas"]];
2464       numiter += 1;
2465     ),
2466     {Jp, AllowedJ[numE]},
2467     {J, AllowedJ[numE]}
2468   ];
2469   If[$FrontEnd != Null,
2470     NotebookDelete[temp]
2471   ];
2472   Return[JJBlockMatrixTable];
2473 );
2474
2475 TabulateManyJJBlockMatrixTables::usage =
2476 TabulateManyJJBlockMatrixTables[{n1, n2, ...}] calculates the
2477 tables of matrix elements for the requested f^n_i configurations.

```

The function does not return the matrices themselves. It instead returns an association whose keys are numE and whose values are the filenames where the output of TabulateJJBlockMatrixTables was saved to. The output consists of an association whose keys are of the form {n, J, Jp} and whose values are rectangular arrays given the values of <|LSJMJa|H|L'S'J'MJ'a'|>.";

```

2457 Options[TabulateManyJJBlockMatrixTables] = {"Overwrite" -> False, "Sparse" -> True, "ChenDeltas" -> False, "FilenameAppendix" -> "", "Compressed" -> False};
2458 TabulateManyJJBlockMatrixTables[ns_, OptionsPattern[]] := (
2459   overwrite = OptionValue["Overwrite"];
2460   fNames = <||>;
2461   fileApp = OptionValue["FilenameAppendix"];
2462   ExportFun = If[OptionValue["Compressed"], ExportMZip, Export];
2463   Do[
2464     (
2465       CFdataFilename = FileNameJoin[{moduleDir, "data", "CrystalFieldTable_f"} <> ToString[numE] <> ".zip"];
2466       PrintTemporary["Importing CrystalFieldTable from ", CFdataFilename, "..."];
2467       CrystalFieldTable = ImportMZip[CFdataFilename];
2468
2469       PrintTemporary["#----- numE = ", numE, " -----#"];
2470       exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix" -> fileApp];
2471       fNames[numE] = exportFname;
2472       If[FileExistsQ[exportFname] && Not[overwrite],
2473         Continue[]
2474       ];
2475       JJBlockMatrixTable = TabulateJJBlockMatrixTable[numE, CrystalFieldTable, "Sparse" -> OptionValue["Sparse"], "ChenDeltas" -> OptionValue["ChenDeltas"]];
2476       If[FileExistsQ[exportFname] && overwrite,
2477         DeleteFile[exportFname]
2478       ];
2479       ExportFun[exportFname, JJBlockMatrixTable];
2480
2481       ClearAll[CrystalFieldTable];
2482     ),
2483     {numE, ns}
2484   ];
2485   Return[fNames];
2486 );
2487
2488 HamMatrixAssembly::usage = "HamMatrixAssembly[numE] returns the Hamiltonian matrix for the f^n_i configuration. The matrix is returned as a SparseArray.
2489 The function admits an optional parameter \"FilenameAppendix\" which can be used to modify the filename to which the resulting array is exported to.
2490 It also admits an optional parameter \"IncludeZeeman\" which can be used to include the Zeeman interaction. The default is False.
2491 The option \"Set t2Switch\" can be used to toggle on or off setting the t2 selector automatically or not, the default is True, which replaces the parameter according to numE.
2492 The option \"ReturnInBlocks\" can be used to return the matrix in block or flattened form. The default is to return it in flattened form.";
2493 Options[HamMatrixAssembly] = {
2494   "FilenameAppendix" -> "",
2495   "IncludeZeeman" -> False,
2496   "Set t2Switch" -> True,
2497   "ReturnInBlocks" -> False,
2498   "OperatorBasis" -> "Legacy"};
2499 HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
2500   {numE, ii, jj, howManyJs, Js, blockHam, opBasis},
2501   (
2502     (*#####
2503     ImportFun = ImportMZip;
2504     opBasis = OptionValue["OperatorBasis"];
2505     If[Not[MemberQ>{"Legacy", "MostlyOrthogonal", "Orthogonal"}, opBasis],
2506       opBasis]];
2507     Print["Operator basis ", opBasis, " not recognized, using \"Legacy\" basis."];
2508     opBasis = "Legacy";
2509   ],

```

```

2509 If[opBasis == "Orthogonal",
2510   Print["Operator basis \"Orthogonal\" not implemented yet,
2511   aborting ..."];
2512   Return[Null];
2513 ];
2514 (*#####
2515 If[opBasis == "MostlyOrthogonal",
2516   (
2517     blockHam = HamMatrixAssembly[nf,
2518       "OperatorBasis" -> "Legacy",
2519       "FilenameAppendix" -> OptionValue["FilenameAppendix"],
2520       "IncludeZeeman" -> OptionValue["IncludeZeeman"],
2521       "Set t2Switch" -> OptionValue["Set t2Switch"],
2522       "ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
2523   paramChanger = Which[
2524     nf < 7,
2525     <|
2526       F0 -> 1/91 (54 E1p+91 E0p+78 γp),
2527       F2 -> (15/392 *
2528         (
2529           140 E1p +
2530           20020 E2p +
2531           1540 E3p +
2532           770 αp -
2533           70 γp +
2534           22 Sqrt[2] T2p -
2535           11 Sqrt[2] nf T2p t2Switch -
2536           11 Sqrt[2] (14-nf) T2p (1 - t2Switch)
2537         )
2538       ),
2539       F4 -> (99/490 *
2540         (
2541           70 E1p -
2542           9100 E2p +
2543           280 E3p +
2544           140 αp -
2545           35 γp +
2546           4 Sqrt[2] T2p -
2547           2 Sqrt[2] nf T2p t2Switch -
2548           2 Sqrt[2] (14-nf) T2p (1-t2Switch)
2549         )
2550       ),
2551       F6 -> (5577/7000 *
2552         (
2553           20 E1p +
2554           700 E2p -
2555           140 E3p -
2556           70 αp -
2557           10 γp -
2558           2 Sqrt[2] T2p +
2559           Sqrt[2] nf T2p t2Switch +
2560           Sqrt[2] (14-nf) T2p (1-t2Switch)
2561         )
2562       ),
2563       ζ -> ζ,
2564       α -> (5 αp)/4,
2565       β -> -6 (5 αp + βp),
2566       γ -> 5/2 (2 βp + 5 γp),
2567       T2 -> 0
2568     |>,
2569     nf >= 7,
2570     <|
2571       F0 -> 1/91 (54 E1p+91 E0p+78 γp),
2572       F2 -> (15/392 *
2573         (
2574           140 E1p +
2575           20020 E2p +
2576           1540 E3p +
2577           770 αp -
2578           70 γp +
2579           22 Sqrt[2] T2p -
2580           11 Sqrt[2] nf T2p
2581         )
2582       ),
2583       F4 -> (99/490 *

```

```

2584      70   E1p -
2585      9100 E2p +
2586      280   E3p +
2587      140   αp -
2588      35   γp +
2589      4   Sqrt[2] T2p -
2590      2   Sqrt[2] nf T2p
2591      )
2592      ),
2593      F6 -> (5577/7000 *
2594      (
2595      20   E1p +
2596      700  E2p -
2597      140   E3p -
2598      70   αp -
2599      10   γp -
2600      2   Sqrt[2] T2p +
2601      Sqrt[2] nf T2p
2602      )
2603      ),
2604      ζ -> ζ,
2605      α -> (5 αp)/4,
2606      β -> -6 (5 αp + βp),
2607      γ -> 5/2 (2 βp + 5 γp),
2608      T2 -> 0
2609      |>
2610      ];
2611      blockHamMO = Which[
2612      OptionValue["ReturnInBlocks"] == False,
2613      ReplaceInSparseArray[blockHam, paramChanger],
2614      OptionValue["ReturnInBlocks"] == True,
2615      Map[ReplaceInSparseArray[#, paramChanger]&,amp;, blockHam,
2616      {2}]
2617      ];
2618      Return[blockHamMO];
2619      )
2620      (*#####
2621      (*hole-particle equivalence enforcement*)
2622      numE = nf;
2623      allVars = {E0, E1, E2, E3, ζ, F0, F2, F4, F6, M0, M2, M4, T2,
2624      T2p,
2625      T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
2626      α, β, γ, B02, B04, B06, B12, B14, B16,
2627      B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16,
2628      S22,
2629      S24, S26, S34, S36, S44, S46, S56, S66, T11p, T12, T14, T15,
2630      T16,
2631      T17, T18, T19, Bx, By, Bz};
2632      params0 = AssociationThread[allVars, allVars];
2633      If[nf > 7,
2634      (
2635      numE = 14 - nf;
2636      params = HoleElectronConjugation[params0];
2637      If[OptionValue["Set t2Switch"], params[t2Switch] = 0];
2638      ),
2639      params = params0;
2640      If[OptionValue["Set t2Switch"], params[t2Switch] = 1];
2641      ];
2642      (* Load symbolic expressions for LS,J,J' energy sub-matrices.*)
2643      emFname = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
2644      OptionValue["FilenameAppendix"]];
2645      JJBlockMatrixTable = ImportFun[emFname];
2646      (*Patch together the entire matrix representation using J,J'
2647      blocks.*)
2648      PrintTemporary["Patching JJ blocks ..."];
2649      Js = AllowedJ[numE];
2650      howManyJs = Length[Js];
2651      blockHam = ConstantArray[0, {howManyJs, howManyJs}];
2652      Do[
2653      blockHam[[jj, ii]] = JJBlockMatrixTable[{numE, Js[[ii]], Js[[jj]]}];,
2654      {ii, 1, howManyJs},
2655      {jj, 1, howManyJs}

```

```

2652 ];
2653
2654 (* Once the block form is created flatten it *)
2655 If[Not[OptionValue["ReturnInBlocks"]],
2656   (blockHam = ArrayFlatten[blockHam];
2657   blockHam = ReplaceInSparseArray[blockHam, params];
2658   ),
2659   (blockHam = Map[ReplaceInSparseArray[#, params]&, blockHam
2660 ,{2}]);
2661 ];
2662
2663 If[OptionValue["IncludeZeeman"],
2664   (
2665     PrintTemporary["Including Zeeman terms ..."];
2666     {magx, magy, magz} = MagDipoleMatrixAssembly[numE, "ReturnInBlocks" -> OptionValue["ReturnInBlocks"]];
2667     blockHam += - teslaToKayser * (Bx * magx + By * magy + Bz * magz);
2668   )
2669 ];
2670 Return[blockHam];
2671 ]
2672
2673 SimplerSymbolicHamMatrix::usage = "SimplerSymbolicHamMatrix[numE,
2674   simplifier] is a simple addition to HamMatrixAssembly that applies
2675   a given simplification to the full Hamiltonian. simplifier is a
2676   list of replacement rules.
2677 If the option \"Export\" is set to True, then the function also
2678   exports the resulting sparse array to the ./hams/ folder.
2679 The option \"PrependToFilename\" can be used to append a string to
2680   the filename to which the function may export to.
2681 The option \"Return\" can be used to choose whether the function
2682   returns the matrix or not.
2683 The option \"Overwrite\" can be used to overwrite the file if it
2684   already exists, if this options is set to False then this function
2685   simply reloads a file that it assumed to be present already in
2686   the ./hams folder.
2687 The option \"IncludeZeeman\" can be used to toggle the inclusion of
2688   the Zeeman interaction with an external magnetic field.
2689 The option \"OperatorBasis\" can be used to choose the basis in
2690   which the operator is expressed. The default is the \"Legacy\" basis.
2691   Order alternatives being: \"MostlyOrthogonal\" and \"Orthogonal\".
2692   In the \"Legacy\" alternative the operators used are
2693   the same as in Carnall's work. In the \"MostlyOrthogonal\" all
2694   operators are orthogonal except those corresponding to the Mk and
2695   Pk parameters. In the \"Orthogonal\" basis all operators are
2696   orthogonal, with the operators corresponding to the Mk and Pk
2697   parameters replaced by zi operators and accompanying ai
2698   coefficients. The \"Orthogonal\" option has not been implemented
2699   yet.";
2700 Options[SimplerSymbolicHamMatrix] = {
2701   "Export" -> True,
2702   "PrependToFilename" -> "",
2703   "EorF" -> "F",
2704   "Overwrite" -> False,
2705   "Return" -> True,
2706   "Set t2Switch" -> False,
2707   "IncludeZeeman" -> False,
2708   "OperatorBasis" -> "Legacy"
2709 };
2710 SimplerSymbolicHamMatrix[numE_Integer, simplifier_List,
2711 OptionsPattern[]] := Module[
2712 {thisHam, fname, fnamemx},
2713 (
2714   If[Not[ValueQ[ElectrostaticTable]],
2715     LoadElectrostatic[]
2716   ];
2717   If[Not[ValueQ[S0OandECSOTable]],
2718     LoadS0OandECSO[]
2719   ];
2720   If[Not[ValueQ[SpinOrbitTable]],
2721     LoadSpinOrbit[]
2722   ];
2723   If[Not[ValueQ[SpinSpinTable]],
2724     LoadSpinSpin[]
2725   ];
2726 ];
2727 
```

```

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

```

```

2769     Print["Exporting to file ", fname, " and to ", fnamemx];
2770     Export[fname, thisHam];
2771     Export[fnamemx, thisHam];
2772   )
2773 ];
2774 If[OptionValue["Return"],
2775   Return[thisHam],
2776   Return[Null]
2777 ];
2778 )
2779 ];
2780
2781 ScalarLSJMFromLS::usage = "ScalarLSJMFromLS[numE,
2782   LSReducedMatrixElements]. Given the LS-reduced matrix elements
2783   LSReducedMatrixElements of a scalar operator, this function
2784   returns the corresponding LSJM representation. This is returned as
2785   a SparseArray.";
2786 ScalarLSJMFromLS[numE_, LSReducedMatrixElements_] := Module[
2787   {jjBlocktable, NKSLJMs, NKSLJMps, J, Jp, eMatrix, SLterm,
2788   SpLpterm,
2789   MJ, MJp, subKron, matValue, Js, howManyJs, blockHam, ii, jj},
2790   (
2791   SparseDiagonalArray[diagonalElements_] := SparseArray[
2792     Table[{i, i} -> diagonalElements[[i]],
2793       {i, 1, Length[diagonalElements]}]
2794     ];
2795   SparseZeroArray[width_, height_] := (
2796     SparseArray[
2797       Join[
2798         Table[{i, i} -> 0, {i, 1, width}],
2799         Table[{i, 1} -> 0, {i, 1, height}]
2800       ]
2801     ];
2802   );
2803   jjBlockTable = <||>;
2804   Do[
2805     NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
2806     NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
2807     If[J != Jp,
2808       jjBlockTable[{numE, J, Jp}] = SparseZeroArray[Length[NKSLJMs],
2809       Length[NKSLJMps]];
2810     Continue[];
2811   ];
2812   eMatrix = Table[
2813     (* Condition for a scalar matrix op *)
2814     SLterm = NKSLJM[[1]];
2815     SpLpterm = NKSLJMp[[1]];
2816     MJ = NKSLJM[[3]];
2817     MJp = NKSLJMp[[3]];
2818     subKron = (KroneckerDelta[MJ, MJp]);
2819     matValue = If[subKron == 0,
2820       0,
2821       (
2822         Which[MemberQ[Keys[LSReducedMatrixElements], {numE,
2823           SLterm, SpLpterm}],
2824           LSReducedMatrixElements[{numE, SLterm, SpLpterm}],
2825           MemberQ[Keys[LSReducedMatrixElements], {numE, SpLpterm,
2826             SLterm}],
2827             LSReducedMatrixElements[{numE, SpLpterm, SLterm}],
2828             True,
2829             0
2830           ]
2831         )
2832       ];
2833     matValue,
2834     {NKSLJM, NKSLJMps},
2835     {NKSLJM, NKSLJMs}
2836   ];
2837   jjBlockTable[{numE, J, Jp}] = SparseArray[eMatrix],
2838   {J, AllowedJ[numE]},
2839   {Jp, AllowedJ[numE]}
2840 ];
2841
2842 Js = AllowedJ[numE];
2843 howManyJs = Length[Js];
2844 blockHam = ConstantArray[0, {howManyJs, howManyJs}];

```

```

2837 Do[blockHam[[jj, ii]] = jjBlockTable[{numE, Js[[ii]], Js[[jj]]}];,
2838 {ii, 1, howManyJs},
2839 {jj, 1, howManyJs}];
2840 blockHam = ArrayFlatten[blockHam];
2841 blockHam = SparseArray[blockHam];
2842 Return[blockHam];
2843 )
2844 ];
2845 (* ##### Block assembly ##### *)
2846 (* ##### *)
2847 (* ##### *)
2848 (* ##### *)
2849 (* ##### Level Description ##### *)
2850 (* ##### *)
2851
2852 FreeHam::usage = "FreeHam[JJBlocks, numE] given the JJ blocks of
the Hamiltonian for f^n, this function returns a list with all the
scalar-simplified versions of the blocks.";
2853 FreeHam[JJBlocks_List, numE_Integer] := Module[
2854 {Js, basisJ, pivot, freeHam, idx, J,
2855 thisJbasis, shrunkBasisPositions, theBlock},
2856 (
2857 Js = AllowedJ[numE];
2858 basisJ = BasisLSJMJ[numE, "AsAssociation" -> True];
2859 pivot = If[OddQ[numE], 1/2, 0];
2860 freeHam = Table[((
2861 J = Js[[idx]];
2862 theBlock = JJBlocks[[idx]];
2863 thisJbasis = basisJ[J];
2864 (* find the basis vectors that end with pivot *)
2865 shrunkBasisPositions = Flatten[Position[thisJbasis, {_ ..., pivot}]];
2866 (* take only those rows and columns *)
2867 theBlock[[shrunkBasisPositions, shrunkBasisPositions]]
2868 ),
2869 {idx, 1, Length[Js]}
2870 ];
2871 Return[freeHam];
2872 )
2873 ];
2874
2875 ListRepeater::usage = "ListRepeater[list, reps] repeats each
element of list reps times.";
2876 ListRepeater[list_List, repeats_Integer] := (
2877 Flatten[ConstantArray[#, repeats] & /@ list]
2878 );
2879
2880 ListLever::usage = "ListLever[vecs, multiplicity] takes a list of
vectors and returns all interleaved shifted versions of them.";
2881 ListLever[vecs_, multiplicity_] := Module[
2882 {uppyVecs, uppyVec},
2883 (
2884 uppyVecs = Table[((
2885 uppyVec = PadRight[{#}, multiplicity] & /@ vec;
2886 uppyVec = Permutations /@ uppyVec;
2887 uppyVec = Transpose[uppyVec];
2888 uppyVec = Flatten /@ uppyVec
2889 ),
2890 {vec, vecs}
2891 ];
2892 Return[Flatten[uppyVecs, 1]];
2893 )
2894 ];
2895
2896 EigenLever::usage = "EigenLever[eigenSys, multiplicity] takes a
list eigenSys of the form {eigenvalues, eigenvectors} and returns
the eigenvalues repeated multiplicity times and the eigenvectors
interleaved and shifted accordingly.";
2897 EigenLever[eigenSys_, multiplicity_] := Module[
2898 {eigenVals, eigenVecs,
2899 leveledEigenVecs, leveledEigenVals},
2900 (
2901 {eigenVals, eigenVecs} = eigenSys;
2902 leveledEigenVals = ListRepeater[eigenVals, multiplicity];
2903 leveledEigenVecs = ListLever[eigenVecs, multiplicity];

```

```

2904     Return[{Flatten[leveledEigenVals], leveledEigenVecs}]
2905   )
2906 ];
2907
2908
2909 LevelSimplerSymbolicHamMatrix::usage =
2910   LevelSimplerSymbolicHamMatrix[numE] is a variation of
2911   HamMatrixAssembly that returns the diagonal JJ Hamiltonian blocks
2912   applying a simplifier and with simplifications adequate for the
2913   level description. The keys of the given association correspond to
2914   the different values of J that are possible for f^numE, the
2915   values are sparse array that are meant to be interpreted in the
2916   basis provided by BasisLSJ.
2917 The option "Simplifier" is a list of symbols that are set to zero
2918 . At a minimum this has to include the crystal field parameters.
2919 By default this includes everything except the Slater parameters
2920 Fk and the spin orbit coupling  $\zeta$ .
2921 The option "Export" controls whether the resulting association is
2922 saved to disk, the default is True and the resulting file is
2923 saved to the ./hams/ folder. A hash is appended to the filename
2924 that corresponds to the simplifier used in the resulting
2925 expression. If the option "Overwrite" is set to False then these
2926 files may be used to quickly retrieve a previously computed case.
2927 The file is saved both in .m and .mx format.
2928 The option "PrependToFilename" can be used to append a string to
2929 the filename to which the function may export to.
2930 The option "Return" can be used to choose whether the function
2931 returns the matrix or not.
2932 The option "Overwrite" can be used to overwrite the file if it
2933 already exists.";
2934 Options[LevelSimplerSymbolicHamMatrix] = {
2935   "Export" -> True,
2936   "PrependToFilename" -> "",
2937   "Overwrite" -> False,
2938   "Return" -> True,
2939   "Simplifier" -> Join[
2940     {FO, \[\[Sigma\]]SS},
2941     cfSymbols,
2942     TSymbols,
2943     casimirSymbols,
2944     pseudoMagneticSymbols,
2945     marvinSymbols,
2946     DeleteCases[magneticSymbols, \zeta]
2947   ]
2948 };
2949
2950 LevelSimplerSymbolicHamMatrix[numE_Integer, OptionsPattern[]] :=
2951 Module[
2952   {thisHamAssoc, Js, fname,
2953   fnamemx, hash, simplifier},
2954 (
2955   simplifier = (#->0)&/@Sort[OptionValue["Simplifier"]];
2956   hash = Hash[simplifier];
2957   If[Not[ValueQ[ElectrostaticTable]], LoadElectrostatic[]];
2958   If[Not[ValueQ[S00andECSOTable]], LoadS00andECSO[]];
2959   If[Not[ValueQ[SpinOrbitTable]], LoadSpinOrbit[]];
2960   If[Not[ValueQ[SpinSpinTable]], LoadSpinSpin[]];
2961   If[Not[ValueQ[ThreeBodyTable]], LoadThreeBody[]];
2962   fname = FileNameJoin[{moduleDir, "hams", OptionValue[
2963     PrependToFilename]} <> "Level-SymbolicMatrix-f" <> ToString[numE] <> "-"
2964   <> ToString[hash] <> ".m"];
2965   fnamemx = FileNameJoin[{moduleDir, "hams", OptionValue[
2966     PrependToFilename]} <> "Level-SymbolicMatrix-f" <> ToString[numE] <> "-"
2967   <> ToString[hash] <> ".mx"];
2968   If[Or[FileExistsQ[fname], FileExistsQ[fnamemx]] && Not[OptionValue
2969     ["Overwrite"]],
2970     (
2971       If[OptionValue["Return"],
2972       (
2973         Which[FileExistsQ[fnamemx],
2974           (
2975             Print["File ", fnamemx, " already exists, and option \""
2976             Overwrite" is set to False, loading file ..."];
2977             thisHamAssoc=Import[fnamemx];
2978             Return[thisHamAssoc];
2979           ),
2980           FileExistsQ[fname],
2981           (
2982             Print["File ", fname, " does not exist, creating new file ..."];
2983             thisHamAssoc=CreateDocument[fnamemx];
2984             Return[thisHamAssoc];
2985           )
2986         ]
2987       )
2988     ]
2989   ]
2990 
```

```

2954      (
2955          Print["File ", fname, " already exists, and option \"Overwrite\" is set to False, loading file ..."];
2956          thisHamAssoc=Import[fname];
2957          Print["Exporting to file ", fnamemx, " for quicker loading."];
2958          Export[fnamemx,thisHamAssoc];
2959          Return[thisHamAssoc];
2960      )
2961  ]
2962  )
2963  (
2964      Print["File ", fname, " already exists, skipping ..."];
2965      Return[Null];
2966  )
2967  ]
2968  )
2969  ];
2970 Js = AllowedJ[numE];
2971 thisHamAssoc = HamMatrixAssembly[numE,
2972 "Set t2Switch" -> True,
2973 "IncludeZeeman" -> False,
2974 "ReturnInBlocks" -> True
2975 ];
2976 thisHamAssoc = Diagonal[thisHamAssoc];
2977 thisHamAssoc = Map[SparseArray[ReplaceInSparseArray[#, simplifier]] &, thisHamAssoc, {1}];
2978 thisHamAssoc = FreeHam[thisHamAssoc, numE];
2979 thisHamAssoc = AssociationThread[Js -> thisHamAssoc];
2980 If[OptionValue["Export"],
2981 (
2982     Print["Exporting to file ", fname, " and to ", fnamemx];
2983     Export[fname, thisHamAssoc];
2984     Export[fnamemx, thisHamAssoc];
2985 )
2986 ];
2987 If[OptionValue["Return"],
2988     Return[thisHamAssoc],
2989     Return[Null]
2990 ];
2991 ]
2992 ];
2993
2994 LevelSolver::usage = "LevelSolver[numE, params] puts together (or retrieves from disk) the symbolic level Hamiltonian for the f^numE configuration and solves it for the given params returning the resultant energies and eigenstates.
2995 If the option \"Return as states\" is set to False, then the function returns an association whose keys are values for J in f^numE, and whose values are lists with two elements. The first element being equal to the ordered basis for the corresponding subspace, given as a list of lists of the form {LS string, J}. The 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.
2996 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.
2997 The following options are admitted:
2998 - \"Overwrite Hamiltonian\", if set to True the function will overwrite the symbolic Hamiltonian. Default is False.
2999 - \"Return as states\", see description above. Default is True.
3000 - \"Simplifier\", this is a list with symbols that are set to zero for defining the parameters kept in the level description.
3001 ";
3002 Options[LevelSolver] = {
3003     "Overwrite Hamiltonian" -> False,

```

```

3004 "Return as states" -> True,
3005 "Simplifier" -> Join[
3006   cfSymbols,
3007   TSymbols,
3008   casimirSymbols,
3009   pseudoMagneticSymbols,
3010   marvinSymbols,
3011   DeleteCases[magneticSymbols, \[Zeta]]
3012 ],
3013 "PrintFun" -> PrintTemporary
3014 };
3015 LevelSolver[numE_Integer, params0_Association, OptionsPattern[]] :=
3016 Module[
3017 {ln, simplifier, simpleHam, basis,
3018 numHam, eigensys, startTime, endTime,
3019 diagonalTime, params=params0, globalBasis,
3020 eigenVectors, eigenEnergies, eigenJs,
3021 states, groundEnergy, allEnergies, PrintFun},
3022 (
3023   ln = theLanthanides[[numE]];
3024   basis = BasisLSJ[numE, "AsAssociation" -> True];
3025   simplifier = OptionValue["Simplifier"];
3026   PrintFun = OptionValue["PrintFun"];
3027   PrintFun["> LevelSolver for ", ln, " with ", numE, " f-electrons."]
3028 ];
3029 PrintFun["> Loading the symbolic level Hamiltonian ..."];
3030 simpleHam = LevelSimplerSymbolicHamMatrix[numE,
3031   "Simplifier" -> simplifier,
3032   "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
3033 ];
3034 (* Everything that is not given is set to zero *)
3035 PrintFun["> Setting to zero every parameter not given ..."];
3036 params = ParamPad[params, "PrintFun" -> PrintFun];
3037 PrintFun[params];
3038 (* Create the numeric hamiltonian *)
3039 PrintFun["> Replacing parameters in the J-blocks of the
3040 Hamiltonian to produce numeric arrays ..."];
3041 numHam = N /@ Map[ReplaceInSparseArray[#, params] &,
3042 simpleHam];
3043 Clear[simpleHam];
3044 (* Eigensolver *)
3045 PrintFun["> Diagonalizing the numerical Hamiltonian within each
3046 separate J-subspace ..."];
3047 startTime = Now;
3048 eigensys = Eigensystem /@ numHam;
3049 endTime = Now;
3050 diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
3051 allEnergies = Flatten[First /@ Values[eigensys]];
3052 groundEnergy = Min[allEnergies];
3053 eigenVectors = Map[Chop[{#[[1]] - groundEnergy, #[[2]]}] &, eigensys
3054 ];
3055 eigenSys = Association @ KeyValueMap[#1 -> {basis[#1], #2} &,
3056 eigensys];
3057 PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
3058 If[OptionValue["Return as states"],
3059 (
3060   PrintFun["> Padding the eigenvectors to correspond to the
3061 level basis ..."];
3062   eigenVectors = SparseArray @ BlockDiagonalMatrix[Values
3063   #[[2, 2]] & /@ eigenSys];
3064   globalBasis = Flatten[Values[basis], 1];
3065   eigenEnergies = Flatten[Values[#[[2, 1]] & /@ eigenSys]];
3066   eigenJs = Flatten[KeyValueMap[ConstantArray[#1,
3067 Length[#[[2, 2]]]] &, eigenSys]];
3068   states = Transpose[{eigenEnergies, eigenJs,
3069 eigenVectors}];
3070   states = SortBy[states, First];
3071   eigenVectors = Last /@ states;
3072   LSJmultiplets = (RemoveTrailingDigits[#[[1]]] <> ToString[
3073 InputForm[#[[2]]]]) & /@ globalBasis;
3074   majorComponentIndices = Ordering[Abs[#[[-1]]] & /@
3075 eigenVectors];
3076   levelLabels = LSJmultiplets[[
3077 majorComponentIndices]];
3078   Return[{globalBasis, levelLabels, states}];
3079 ),
3080 ]

```

```

3066     Return[{basis, eigensys}]
3067   ];
3068 ]
3069 ];
3070
3071 (* ##### Level Description ##### *)
3072 (* ##### *)
3073
3074 (* ##### Optical Operators ##### *)
3075
3076 magOp = <||>;
3077
3078 JJBlockMagDip::usage = "JJBlockMagDip[numE, J, Jp] returns an array
3079   for the LSJM matrix elements of the magnetic dipole operator
3080   between states with given J and Jp. The option \"Sparse\" can be
3081   used to return a sparse matrix. The default is to return a sparse
3082   matrix.
3083 See eqn 15.7 in TASS.
3084 Here it is provided in atomic units in which the Bohr magneton is
3085   1/2.
3086 \[Mu] = -(1/2) (L + gs S)
3087 We are using the Racah convention for the reduced matrix elements
3088   in the Wigner-Eckart theorem. See TASS eqn 11.15.
3089 ";
3090 Options[JJBlockMagDip]={ "Sparse" -> True};
3091 JJBlockMagDip[numE_, braJ_, ketJ_, OptionsPattern[]] := Module[
3092   {braSLJs, ketSLJs,
3093    braSLJ, ketSLJ,
3094    braSL, ketSL,
3095    braS, braL,
3096    ketS, ketL,
3097    braMJ, ketMJ,
3098    matValue, magMatrix,
3099    summand1, summand2,
3100    threejays},
3101 (
3102   braSLJs = AllowedNKSLJMforJTerms[numE, braJ];
3103   ketSLJs = AllowedNKSLJMforJTerms[numE, ketJ];
3104   magMatrix = Table[
3105     braSL = braSLJ[[1]];
3106     ketSL = ketSLJ[[1]];
3107     {braS, braL} = FindSL[braSL];
3108     {ketS, ketL} = FindSL[ketSL];
3109     braMJ = braSLJ[[3]];
3110     ketMJ = ketSLJ[[3]];
3111     summand1 = If[Or[braJ != ketJ,
3112                   braSL != ketSL],
3113                   0,
3114                   Sqrt[braJ*(braJ+1)*TPO[braJ]]
3115                 ];
3116     (* looking at the string includes checking L=L', S=S', and \
3117 alpha=alpha *)
3118     summand2 = If[braSL != ketSL,
3119                   0,
3120                   (gs-1) *
3121                     Phaser[braS+braL+ketJ+1] *
3122                     Sqrt[TPO[braJ]*TPO[ketJ]] *
3123                     SixJay[{braJ, 1, ketJ}, {braS, braL, braS}] *
3124                     Sqrt[braS(braS+1)TPO[braS]]
3125                 ];
3126     matValue = summand1 + summand2;
3127     (* We are using the Racah convention for red matrix elements
3128     in Wigner-Eckart *)
3129     threejays = (ThreeJay[{braJ, -braMJ}, {1, #}, {ketJ, ketMJ}]*
3130     ) /@ {-1, 0, 1};
3131     threejays *= Phaser[braJ-braMJ];
3132     matValue = - 1/2 * threejays * matValue;
3133     matValue,
3134     {braSLJ, braSLJs},
3135     {ketSLJ, ketSLJs}
3136   ];
3137   If[OptionValue["Sparse"],
3138     magMatrix = SparseArray[magMatrix]
3139   ];

```

```

3133     Return[magMatrix];
3134   )
3135 ];
3136
3137 Options[TabulateJJBlockMagDipTable]={"Sparse"->True};
3138 TabulateJJBlockMagDipTable[numE_,OptionsPattern[]]:=(
3139   JJBlockMagDipTable=<||>;
3140   Js=AllowedJ[numE];
3141   Do[
3142   (
3143     JJBlockMagDipTable[{numE,braJ,ketJ}]=
3144       JJBlockMagDip[numE,braJ,ketJ,"Sparse"->OptionValue["Sparse"]
3145     ]
3146     ,{braJ, Js},
3147     {ketJ, Js}
3148   ];
3149   Return[JJBlockMagDipTable]
3150 );
3151
3152 TabulateManyJJBlockMagDipTables::usage =
3153   TabulateManyJJBlockMagDipTables[{n1, n2, ...}] calculates the
3154   tables of matrix elements for the requested f^n_i configurations.
3155   The function does not return the matrices themselves. It instead
3156   returns an association whose keys are numE and whose values are
3157   the filenames where the output of TabulateManyJJBlockMagDipTables
3158   was saved to. The output consists of an association whose keys are
3159   of the form {n, J, Jp} and whose values are rectangular arrays
3160   given the values of <|LSJMJa|H_dip|L'S'J'MJ'a'|>.";
3161 Options[TabulateManyJJBlockMagDipTables]={"FilenameAppendix"->"", "Overwrite"->False, "Compressed"->True};
3162 TabulateManyJJBlockMagDipTables[ns_,OptionsPattern[]]:=(
3163   fnames=<||>;
3164   Do[
3165   (
3166     ExportFun=If[OptionValue["Compressed"],ExportMZip,Export];
3167     PrintTemporary["----- numE = ",numE," -----#"];
3168     appendTo=(OptionValue["FilenameAppendix"]<>"-magDip");
3169     exportFname=JJBlockMatrixFileName[numE,"FilenameAppendix"->appendTo];
3170     fnames[numE]=exportFname;
3171     If[FileExistsQ[exportFname]&&Not[OptionValue["Overwrite"]],
3172       Continue[]
3173     ];
3174     JJBlockMatrixTable=TabulateJJBlockMagDipTable[numE];
3175     If[FileExistsQ[exportFname]&&OptionValue["Overwrite"],
3176       DeleteFile[exportFname]
3177     ];
3178     ExportFun[exportFname,JJBlockMatrixTable];
3179   ),
3180   {numE,ns}
3181 ];
3182   Return[fnames];
3183 );
3184
3185 MagDipoleMatrixAssembly::usage = "MagDipoleMatrixAssembly[numE]
3186   returns the matrix representation of the operator - 1/2 (L + gs S)
3187   in the f^numE configuration. The function returns a list with
3188   three elements corresponding to the x,y,z components of this
3189   operator. The option \"FilenameAppendix\" can be used to append a
3190   string to the filename from which the function imports from in
3191   order to patch together the array. For numE beyond 7 the function
3192   returns the same as for the complementary configuration. The
3193   option \"ReturnInBlocks\" can be used to return the matrices in
3194   blocks. The default is to return the matrices in flattened form
3195   and as sparse array.";
3196 Options[MagDipoleMatrixAssembly]={
3197   "FilenameAppendix"->"",
3198   "ReturnInBlocks"->False};
3199 MagDipoleMatrixAssembly[nf_Integer, OptionsPattern[]]:=Module[
3200   {ImportFun, numE, appendTo,
3201   emFname, JJBlockMagDipTable,
3202   Js, howManyJs, blockOp,
3203   rowIdx, colIdx},
3204   (
3205     ImportFun=ImportMZip;

```

```

3188     numE      = nf;
3189     numH      = 14 - numE;
3190     numE      = Min[numE, numH];
3191
3192     appendTo  = (OptionValue["FilenameAppendix"] <> "-magDip");
3193     emFname   = JJBlockMatrixFileName[numE, "FilenameAppendix" ->
3194     appendTo];
3195     JJBlockMagDipTable = ImportFun[emFname];
3196
3197     Js        = AllowedJ[numE];
3198     howManyJs = Length[Js];
3199     blockOp   = ConstantArray[0, {howManyJs, howManyJs}];
3200     Do[
3201       blockOp[[rowIdx, colIdx]] = JJBlockMagDipTable[{numE, Js[[rowIdx]], Js[[colIdx]]}],
3202         {rowIdx, 1, howManyJs},
3203         {colIdx, 1, howManyJs}
3204     ];
3205     If[OptionValue["ReturnInBlocks"],
3206       (
3207         opMinus = Map[#[[1]] &, blockOp, {4}];
3208         opZero = Map[#[[2]] &, blockOp, {4}];
3209         opPlus = Map[#[[3]] &, blockOp, {4}];
3210         opX = (opMinus - opPlus)/Sqrt[2];
3211         opY = I (opPlus + opMinus)/Sqrt[2];
3212         opZ = opZero;
3213       ),
3214       blockOp = ArrayFlatten[blockOp];
3215       opMinus = blockOp[;;, ;;, 1];
3216       opZero = blockOp[;;, ;;, 2];
3217       opPlus = blockOp[;;, ;;, 3];
3218       opX = (opMinus - opPlus)/Sqrt[2];
3219       opY = I (opPlus + opMinus)/Sqrt[2];
3220       opZ = opZero;
3221     ];
3222     Return[{opX, opY, opZ}];
3223   )
3224 ];
3225 MagDipLineStrength::usage = "MagDipLineStrength[theEigensys, numE]
3226   takes the eigensystem of an ion and the number numE of f-electrons
3227   that correspond to it and calculates the line strength array Stot
3228 .
3229 The option \"Units\" can be set to either \"SI\" (so that the units
3230   of the returned array are (A m^2)^2) or to \"Hartree\".
3231 The option \"States\" can be used to limit the states for which the
3232   line strength is calculated. The default, All, calculates the
3233   line strength for all states. A second option for this is to
3234   provide an index labelling a specific state, in which case only
3235   the line strengths between that state and all the others are
3236   computed.
3237 The returned array should be interpreted in the eigenbasis of the
3238   Hamiltonian. As such the element Stot[[i,i]] corresponds to the
3239   line strength states between states |i> and |j>.";
3240 Options[MagDipLineStrength] = {"Reload MagOp" -> False, "Units" -> "SI"
3241   , "States" -> All};
3242 MagDipLineStrength[theEigensys_List, numE0_Integer, OptionsPattern[]
3243   ] := 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   ];
3256   allEigenvecs = Transpose[Last /@ theEigensys];
3257   Which[OptionValue["States"] === All,
3258     (
3259       {Sx, Sy, Sz} = (ConjugateTranspose[allEigenvecs].#
3260       allEigenvecs) & /@ magOp[numE];
3261       Stot           = Abs[Sx]^2 + Abs[Sy]^2 + Abs[Sz]^2;
3262     )
3263   ];
3264 ];

```

```

3247 ),
3248 IntegerQ[OptionValue["States"]],
3249 (
3250     singleState = theEigensys[[OptionValue["States"],2]];
3251     {Sx, Sy, Sz} = (ConjugateTranspose[allEigenvecs].#.
3252     singleState) & /@ magOp[numE];
3253     Stot = Abs[Sx]^2+Abs[Sy]^2+Abs[Sz]^2;
3254 )
3255 ];
3256 Which[
3257     OptionValue["Units"] == "SI",
3258         Return[4 \[Mu]B^2 * Stot],
3259     OptionValue["Units"] == "Hartree",
3260         Return[Stot],
3261     True,
3262     (
3263         Print["Invalid option for \"Units\". Options are \"SI\" and
3264 \\"Hartree\"."];
3265         Abort[];
3266     )
3267 ];
3268
3269 MagDipoleRates::usage = "MagDipoleRates[eigenSys, numE] calculates
the magnetic dipole transition rate array for the provided
eigensystem. The option \"Units\" can be set to \"SI\" or to \"
Hartree\". If the option \"Natural Radiative Lifetimes\" is set to
true then the reciprocal of the rate is returned instead.
eigenSys is a list of lists with two elements, in each list the
first element is the energy and the second one the corresponding
eigenvector.
3270 Based on table 7.3 of Thorne 1999, using g2=1.
3271 The energy unit assumed in eigenSys is kayser.
3272 The returned array should be interpreted in the eigenbasis of the
Hamiltonian. As such the element AMD[[i,i]] corresponds to the
transition rate (or the radiative lifetime, depending on options)
between eigenstates |i> and |j>.
3273 By default this assumes that the refractive index is unity, this
may be changed by setting the option \"RefractiveIndex\" to the
desired value.
3274 The option \"Lifetime\" can be used to return the reciprocal of the
transition rates. The default is to return the transition rates."
3275 ;
3276 Options[MagDipoleRates]={\"Units\"->"SI", "Lifetime"->False, "
RefractiveIndex"->1};
3277 MagDipoleRates[eigenSys_List, numE0_Integer, OptionsPattern[]] :=
Module[
3278 {AMD, Stot, eigenEnergies,
3279 transitionWaveLengthsInMeters, nRefractive},
3280 (
3281     nRefractive = OptionValue["RefractiveIndex"];
3282     numE = Min[14-numE0, numE0];
3283     Stot = MagDipLineStrength[eigenSys, numE, "Units" ->
3284 OptionValue["Units"]];
3285     eigenEnergies = Chop[First/@eigenSys];
3286     energyDiffs = Outer[Subtract, eigenEnergies, eigenEnergies];
3287     energyDiffs = ReplaceDiagonal[energyDiffs, Indeterminate];
3288     (* Energies assumed in kayser.*)
3289     transitionWaveLengthsInMeters = 0.01/energyDiffs;
3290
3291     unitFactor = Which[
3292     OptionValue["Units"]==="Hartree",
3293     (
3294         (* The bohrRadius factor in SI needed to convert the
wavelengths which are assumed in m*)
3295         16 \[Pi]^3 (\[Mu]0Hartree /(3 hPlanckFine)) * bohrRadius^3
3296     ),
3297     OptionValue["Units"]==="SI",
3298     (
3299         16 \[Pi]^3 \[Mu]0/(3 hPlanck)
3300     ),
3301     True,
3302     (
3303         Print["Invalid option for \"Units\". Options are \"SI\" and \
3304 \"Hartree\"."];

```

```

3302     Abort[];
3303   )
3304 ];
3305   AMD = unitFactor / transitionWaveLengthsInMeters^3 * Stot *
nRefractive^3;
3306   Which[OptionValue["Lifetime"],
3307     Return[1/AMD],
3308     True,
3309     Return[AMD]
3310   ]
3311 ]
3312 ];
3313
3314 GroundMagDipoleOscillatorStrength::usage =
  GroundMagDipoleOscillatorStrength[eigenSys, numE] calculates the
  magnetic dipole oscillator strengths between the ground state and
  the excited states as given by eigenSys.
3315 Based on equation 8 of Carnall 1965, removing the  $2J+1$  factor since
  this degeneracy has been removed by the crystal field.
3316 eigenSys is a list of lists with two elements, in each list the
  first element is the energy and the second one the corresponding
  eigenvector.
3317 The energy unit assumed in eigenSys is Kayser.
3318 The oscillator strengths are dimensionless.
3319 The returned array should be interpreted in the eigenbasis of the
  Hamiltonian. As such the element fMDGS[[i]] corresponds to the
  oscillator strength between ground state and eigenstate  $|i\rangle$ .
3320 By default this assumes that the refractive index is unity, this
  may be changed by setting the option \\"RefractiveIndex\\" to the
  desired value.";
3321 Options[GroundMagDipoleOscillatorStrength]=>{"RefractiveIndex"->1};
3322 GroundMagDipoleOscillatorStrength[eigenSys_List, numE_Integer,
  OptionsPattern[]] := Module[
3323 {eigenEnergies, SMDGS, GSEnergy, energyDiffs,
3324 transitionWaveLengthsInMeters, unitFactor, nRefractive},
3325 (
3326   eigenEnergies = First/@eigenSys;
3327   nRefractive = OptionValue["RefractiveIndex"];
3328   SMDGS = MagDipLineStrength[eigenSys, numE, "Units"->
SI", "States"->1];
3329   GSEnergy = eigenSys[[1,1]];
3330   energyDiffs = eigenEnergies-GSEnergy;
3331   energyDiffs[[1]] = Indeterminate;
3332   transitionWaveLengthsInMeters = 0.01/energyDiffs;
3333   unitFactor = (8\[Pi]^2 me)/(3 hPlanck eCharge^2 cLight);
3334   fMDGS = unitFactor / transitionWaveLengthsInMeters *
SMDGS * nRefractive;
3335   Return[fMDGS];
3336 )
3337 ];
3338
3339 (* ##### Optical Operators ##### *)
3340 (* ##### Printers and Labels ##### *)
3341
3342 (* ##### Printers and Labels ##### *)
3343 (* ##### Printers and Labels ##### *)
3344
3345 PrintL::usage = "PrintL[L] give the string representation of a
  given angular momentum.";
3346 PrintL[L_] := If[StringQ[L], L, StringTake[specAlphabet, {L + 1}]]
3347
3348 FindSL::usage = "FindSL[LS] gives the spin and orbital angular
  momentum that corresponds to the provided string LS.";
3349 FindSL[SL_] :=
3350   FindSL[SL] =
3351   If[StringQ[SL],
3352     {
3353       (ToExpression[StringTake[SL, 1]]-1)/2,
3354       StringPosition[specAlphabet, StringTake[SL, {2}]][[1, 1]]-1
3355     },
3356     SL
3357   ];
3358
3359 PrintSLJ::usage = "Given a list with three elements {S, L, J} this
  function returns a symbol where the spin multiplicity is presented"

```

```

    as a superscript, the orbital angular momentum as its
    corresponding spectroscopic letter, and J as a subscript. Function
    does not check to see if the given J is compatible with the given
    S and L.";
3361 PrintSLJ[SLJ_] := (
3362   RowBox[{  

3363     SuperscriptBox[" ", 2 SLJ[[1]] + 1],  

3364     SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]  

3365   }  

3366 ] // DisplayForm  

3367 );
3368
3369 PrintSLJM::usage = "Given a list with four elements {S, L, J, MJ}
    this function returns a symbol where the spin multiplicity is
    presented as a superscript, the orbital angular momentum as its
    corresponding spectroscopic letter, and {J, MJ} as a subscript. No
    attempt is made to guarantee that the given input is consistent."
;
3370 PrintSLJM[SLJM_] := (
3371   RowBox[{  

3372     SuperscriptBox[" ", 2 SLJM[[1]] + 1],  

3373     SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]  

3374   }  

3375 ] // DisplayForm  

3376 );
3377
3378 (* ##### Printers and Labels ##### *)
3379 (* ##### ##### ##### ##### *)
3380
3381 (* ##### ##### ##### ##### *)
3382 (* ##### ##### ##### Term management ##### *)
3383
3384 AllowedSLTerms::usage = "AllowedSLTerms[numE] returns a list with
    the allowed terms in the f^numE configuration, the terms are given
    as lists in the format {S, L}. This list may have redundancies
    which are compatible with the degeneracies that might correspond
    to the given case.";
3385 AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE,
    14-numE]]];
3386
3387 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.";
3388 AllowedNKSLTerms[numE_] := Map[First, CFPTerms[Min[numE, 14-numE
    ]]];
3389 AllowedNKSLTerms[0] = {"1S"};
3390 AllowedNKSLTerms[14] = {"1S"};
3391
3392 MaxJ::usage = "MaxJ[numE] gives the maximum J = S+L that
    corresponds to the configuration f^numE.";
3393 MaxJ[numE_] := Max[Map[Total, AllowedSLTerms[Min[numE, 14-numE]]]];
3394
3395 MinJ::usage = "MinJ[numE] gives the minimum J = S+L that
    corresponds to the configuration f^numE.";
3396 MinJ[numE_] := Min[Map[Abs[Part[#, 1] - Part[#, 2]] &,
    AllowedSLTerms[Min[numE, 14-numE]]]]
3397
3398 AllowedSLJTerms::usage = "AllowedSLJTerms[numE] returns a list with
    the allowed {S, L, J} terms in the f^n configuration, the terms
    are given as lists in the format {S, L, J}. This list may have
    repeated elements which account for possible degeneracies of the
    related term.";
3399 AllowedSLJTerms[numE_] := Module[
3400   {idx1, allowedSL, allowedSLJ},
3401   (
3402     allowedSL = AllowedSLTerms[numE];
3403     allowedSLJ = {};
3404     For[
3405       idx1 = 1,
3406       idx1 <= Length[allowedSL],
3407       termSL = allowedSL[[idx1]];
3408       termsSLJ =
3409         Table[
3410           {termSL[[1]], termSL[[2]], J},
3411           {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}]

```

```

3412 ];
3413     allowedSLJ = Join[allowedSLJ, termsSLJ];
3414     idx1++;
3415 ];
3416     SortBy[allowedSLJ, Last]
3417 )
3418 ];
3419
3420 AllowedNKSLJTerms::usage = "AllowedNKSLJTerms[numE] returns a list
3421   with the allowed {SL, J} terms in the f^n configuration, the terms
3422   are given as lists in the format {SL, J} where SL is a string in
3423   spectroscopic notation.";
3424 AllowedNKSLJTerms[numE_] := Module[
3425   {allowedSL, allowedNKSL, allowedSLJ, nn},
3426   (
3427     allowedNKSL = AllowedNKSLTerms[numE];
3428     allowedSL = AllowedSLTerms[numE];
3429     allowedSLJ = {};
3430     For[
3431       nn = 1,
3432       nn <= Length[allowedSL],
3433       (
3434         termSL = allowedSL[[nn]];
3435         termNKSL = allowedNKSL[[nn]];
3436         termsSLJ =
3437           Table[{termNKSL, J},
3438             {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
3439           ];
3440         allowedSLJ = Join[allowedSLJ, termsSLJ];
3441         nn++
3442       )
3443     ];
3444     SortBy[allowedSLJ, Last]
3445   )
3446 ];
3447
3448 AllowedNKSLforJTerms::usage = "AllowedNKSLforJTerms[numE, J] gives
3449   the terms that correspond to the given total angular momentum J in
3450   the f^n configuration. The result is a list whose elements are
3451   lists of length 2, the first element being the SL term in
3452   spectroscopic notation, and the second element being J.";
3453 AllowedNKSLforJTerms[numE_, J_] := Module[
3454   {allowedSL, allowedNKSL, allowedSLJ,
3455   nn, termSL, termNKSL, termsSLJ},
3456   (
3457     allowedNKSL = AllowedNKSLTerms[numE];
3458     allowedSL = AllowedSLTerms[numE];
3459     allowedSLJ = {};
3460     For[
3461       nn = 1,
3462       nn <= Length[allowedSL],
3463       (
3464         termSL = allowedSL[[nn]];
3465         termNKSL = allowedNKSL[[nn]];
3466         termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[
3467           termSL],
3468             {{termNKSL, J}},
3469             {}
3470           ];
3471         allowedSLJ = Join[allowedSLJ, termsSLJ];
3472         nn++
3473       )
3474     ];
3475     Return[allowedSLJ]
3476   )
3477 ];
3478
3479 AllowedSLJMTerms::usage = "AllowedSLJMTerms[numE] returns a list
3480   with all the states that correspond to the configuration f^n. A
3481   list is returned whose elements are lists of the form {S, L, J, MJ
3482   }.";
3483 AllowedSLJMTerms[numE_] := Module[
3484   {allowedSLJ, allowedSLJM,
3485   termSLJ, termsSLJM, nn},
3486   (
3487     allowedSLJ = AllowedSLJTerms[numE];

```

```

3477 allowedSLJM = {};
3478 For[
3479   nn = 1,
3480   nn <= Length[allowedSLJ],
3481   nn++,
3482   (
3483     termSLJ = allowedSLJ[[nn]];
3484     termsSLJM =
3485       Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
3486         {M, - termSLJ[[3]], termSLJ[[3]]}]
3487       ];
3488     allowedSLJM = Join[allowedSLJM, termsSLJM];
3489   )
3490 ];
3491 Return[SortBy[allowedSLJM, Last]];
3492 )
3493 ];
3494
3495 AllowedNKSLJMforJMTerms::usage = "AllowedNKSLJMforJMTerms[numE_, J,
3496 MJ] returns a list with all the terms that contain states of the f
3497 ^n configuration that have a total angular momentum J, and a
3498 projection along the z-axis MJ. The returned list has elements of
3499 the form {SL (string in spectroscopic notation), J, MJ}.";
3500 AllowedNKSLJMforJMTerms[numE_, J_, MJ_] := Module[
3501   {allowedSL, allowedNKSL,
3502   allowedSLJM, nn},
3503   (
3504     allowedNKSL = AllowedNKSLTerms[numE];
3505     allowedSL = AllowedSLTerms[numE];
3506     allowedSLJM = {};
3507     For[
3508       nn = 1,
3509       nn <= Length[allowedSL],
3510       termSL = allowedSL[[nn]];
3511       termNKSL = allowedNKSL[[nn]];
3512       termssSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
3513                     <= J
3514                     && (Abs[MJ] <= J)
3515                     ),
3516                     {{termNKSL, J, MJ}},
3517                     {}];
3518       allowedSLJM = Join[allowedSLJM, termssSLJ];
3519       nn++
3520     ];
3521     Return[allowedSLJM];
3522   )
3523 ];
3524
3525 AllowedNKSLJMforJTerms::usage = "AllowedNKSLJMforJTerms[numE_, J]
3526 returns a list with all the states that have a total angular
3527 momentum J. The returned list has elements of the form {{SL (
3528 string in spectroscopic notation), J}, MJ}, and if the option \"Flat\"
3529 is set to True then the returned list has element of the
3530 form {SL (string in spectroscopic notation), J, MJ}.";
3531 AllowedNKSLJMforJTerms[numE_, J_] := Module[
3532   {MJs, labelsAndMomenta, termsWithJ},
3533   (
3534     MJs = AllowedMforJ[J];
3535     (* Pair LS labels and their {S,L} momenta *)
3536     labelsAndMomenta = (#, FindSL[#]) & /@ AllowedNKSLTerms[numE];
3537     (* A given term will contain J if |L-S|<=J<=L+S *)
3538     ContainsJ[{SL_String, {S_, L_}}] := (Abs[S - L] <= J <= (S + L));
3539     (* Keep just the terms that satisfy this condition *)
3540     termsWithJ = Select[labelsAndMomenta, ContainsJ];
3541     (* We don't want to keep the {S,L} *)
3542     termsWithJ = #[[1]], J] & /@ termsWithJ;
3543     (* This is just a quick way of including up all the MJ values
3544    *)
3545     Return[Flatten /@ Tuples[{termsWithJ, MJs}]]
3546   )
3547 ];
3548
3549 AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J

```

```

    , 1].";
3541 AllowedMforJ[J_] := Range[-J, J, 1];
3542
3543 AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta
3544      J that appear in the f^numE configuration.";
3545 AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}];
3546
3547 Seniority::usage = "Seniority[LS] returns the seniority of the
3548      given term.";
3549 Seniority[LS_] := FindNKLSTerm[LS][[1, 2]];
3550
3551 FindNKLSTerm::usage = "Given the string LS FindNKLSTerm[SL] returns
3552      all the terms that are compatible with it. This is only for f^n
3553      configurations. The provided terms might belong to more than one
3554      configuration. The function returns a list with elements of the
3555      form {LS, seniority, W, U}.";
3556 FindNKLSTerm[SL_] := Module[
3557   {NKterms, n},
3558   (
3559     n = 7;
3560     NKterms = {};
3561     Map[
3562       If[! StringFreeQ[First[#], SL],
3563         If[ToExpression[Part[#, 2]] <= n,
3564           NKterms = Join[NKterms, {#}, 1]
3565         ]
3566       ] &,
3567       fnTermLabels
3568     ];
3569     NKterms = DeleteCases[NKterms, {}];
3570     NKterms
3571   )
3572 ];
3573
3574 ParseTermLabels::usage = "ParseTermLabels[] parses the labels for
3575      the terms in the f^n configurations based on the labels for the f6
3576      and f7 configurations. The function returns a list whose elements
3577      are of the form {LS, seniority, W, U}.";
3578 Options[ParseTermLabels] = {"Export" -> True};
3579 ParseTermLabels[OptionsPattern[]] := Module[
3580   {labelsTextData, fNtextLabels, nielsonKosterLabels,
3581   seniorities, RacahW, RacahU},
3582   (
3583     labelsTextData = FileNameJoin[{moduleDir, "data", "NielsonKosterLabels_f6_f7.txt"}];
3584     fNtextLabels = Import[labelsTextData];
3585     nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
3586     termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
3587     seniorities = Map[ToExpression[Part[#, {2}]] &,
3588     nielsonKosterLabels];
3589     racahW =
3590       Map[
3591         StringTake[
3592           Flatten[StringCases[Part[#, {3}],
3593             "(" ~~ DigitCharacter ~~ DigitCharacter ~~
3594             DigitCharacter ~~ ")"]],
3595             {2, 4}
3596           ] &,
3597           nielsonKosterLabels];
3598     racahU =
3599       Map[
3600         StringTake[
3601           Flatten[StringCases[Part[#, {3}],
3602             "(" ~~ DigitCharacter ~~ DigitCharacter ~~ ")"],
3603             {2, 3}
3604           ] &,
3605           nielsonKosterLabels];
3606     fNtermLabels = Join[termLabels, seniorities, racahW, racahU,
3607   2];
3608     fNtermLabels = Sort[fNtermLabels];
3609     If[OptionValue["Export"],
3610       (
3611         broadFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
3612         Export[broadFname, fNtermLabels];
3613       )
3614     ];
3615   ];

```

```

3603     Return[fnTermLabels];
3604   )
3605 ];
3606
3607 (* ##### Term management #### *)
3608 (* ##### *)
3609
3610 LoadLaF3Parameters::usage = "LoadLaF3Parameters[ln] takes a string
3611 with the symbol the element of a trivalent lanthanide ion and
3612 returns model parameters for it. It is based on the data for LaF3.
3613 If the option \"Free Ion\" is set to True then the function sets
3614 all crystal field parameters to zero. Through the option \"gs\" it
3615 allows modifying the electronic gyromagnetic ratio. For
3616 completeness this function also computes the E parameters using
3617 the F parameters quoted on Carnall.";
3618 Options[LoadLaF3Parameters] = {
3619   "Free Ion" -> False,
3620   "gs" -> 2.002319304386,
3621   "With Uncertainties" -> False
3622 };
3623 LoadLaF3Parameters[Ln_String, OptionsPattern[]] := Module[
3624   {params, uncertain,
3625    uncertainKeys, uncertainRules},
3626   (
3627     If[Not[ValueQ[Carnall]],
3628      LoadCarnall[];
3629    ];
3630    params = Association[Carnall["data"][[Ln]]];
3631    (*If a free ion then all the parameters from the crystal field
3632     are set to zero*)
3633    If[OptionValue["Free Ion"],
3634      Do[params[cfSymbol] = 0, {cfSymbol, cfSymbols}]
3635    ];
3636    params[F0] = 0;
3637    params[M2] = 0.56 * params[M0]; (*See Carnall 1989,Table I,
3638    caption,probably fixed based on HF values*)
3639    params[M4] = 0.31 * params[M0]; (*See Carnall 1989,Table I,
3640    caption,probably fixed based on HF values*)
3641    params[P0] = 0;
3642    params[P4] = 0.5 * params[P2]; (*See Carnall 1989,Table I,
3643    caption,probably fixed based on HF values*)
3644    params[P6] = 0.1 * params[P2]; (*See Carnall 1989,Table I,
3645    caption,probably fixed based on HF values*)
3646    params[gs] = OptionValue["gs"];
3647    {params[E0], params[E1], params[E2], params[E3]} = FtoE[{params[
3648      F0], params[F2], params[F4], params[F6]};
3649    params[E0] = 0;
3650    If[
3651      Not[OptionValue["With Uncertainties"]],
3652      Return[params],
3653      (
3654        uncertain = Association[Carnall["annotations"][[Ln]]];
3655        uncertainKeys = Keys[uncertain];
3656        uncertain = If[#, == "Not allowed to vary in fitting."
3657          || # == "Interpolated",
3658            0., #] & /@ uncertain;
3659        paramKeys = Keys[params];
3660        uncertainVals = Sort[Intersection[paramKeys, uncertainKeys
3661        ] /. Association[uncertain]];
3662        uncertainRules = MapThread[Rule, {Sort[uncertainKeys],
3663          uncertainVals}];
3664        Which[
3665          MemberQ[{Ce, "Yb"}, Ln],
3666          (
3667            subsetL = {F0};
3668            subsetR = {0};
3669          ),
3670          True,
3671          (
3672            subsetL = {F0, M2, M4, P0, P4, P6, E0, E1, E2, E3};
3673            subsetR = {0, M0*0.65, M0*0.31, 0, P2*0.5, P2*0.1,
3674              0,
3675              Sqrt[(196 F2^2)/164025 + (49 F4^2)/88209 + (122500 F6
3676              ^2)/134165889],
3677              Sqrt[F2^2/4100625 + F4^2/10673289 + (30625 F6^2)
3678              /2743558264161],
3679            ]
3680          )
3681        ]
3682      ]
3683    ]
3684  ]
3685 ]
3686 
```

```

3661           Sqrt[F2^2/18225 + (4 F4^2)/1185921 + (30625 F6^2)
3662 /1803785841]];
3663       )
3664     ];
3665   uncertainRules = Join[uncertainRules, MapThread[Rule, {
3666 subsetL, subsetR /. uncertainRules}]];
3667   uncertainRules = Association[uncertainRules];
3668   Which[
3669     Ln == "Eu",
3670     (
3671       uncertainRules[F4] = 12.121;
3672       uncertainRules[F6] = 15.872;
3673     ),
3674     Ln == "Gd",
3675     (
3676       uncertainRules[F4] = 12.07;
3677     ),
3678     Ln == "Tb",
3679     (
3680       uncertainRules[F4] = 41.006;
3681     )
3682   ];
3683   If[MemberQ[{"Eu", "Gd", "Tb"}, Ln],
3684     (
3685       uncertainRules[E1] = Sqrt[(196 F2^2)/164025 + (49 F4^2)
3686 /88209 + (122500 F6^2)/134165889] /. uncertainRules;
3687       uncertainRules[E2] = Sqrt[F2^2/4100625 + F4^2/10673289
3688 + (30625 F6^2)/2743558264161] /. uncertainRules;
3689       uncertainRules[E3] = Sqrt[F2^2/18225 + (4 F4^2)/1185921
3690 + (30625 F6^2)/1803785841] /. uncertainRules;
3691     )
3692   ];
3693   uncertainKeys = First /@ Normal[uncertainRules];
3694   fullParams = Association[MapThread[Rule, {uncertainKeys,
3695 MapThread[Around, {uncertainKeys /. params, uncertainKeys /.
3696 uncertainRules}]}]];
3697   Return[Join[params, fullParams]]
3698 )
3699 ];
3700 )
3701 ];
3702 );
3703 );
3704 LoadLiYF4Parameters::usage="LoadLiYF4Parameters[ln] takes a string
3705 with the symbol the element of a trivalent lanthanide ion and
3706 returns model parameters for it. It return the data for LiYF4 from
3707 Cheng et al.";
3708 LoadLiYF4Parameters[ln_, OptionsPattern[]]:=(
3709   If[!ValueQ[paramsLiYF4],
3710     paramsChengLiYF4 = Import[FileNameJoin[{moduleDir,"data",
3711 "chengLiYF4.m"}]];
3712   ];
3713   Return[paramsChengLiYF4[ln]];
3714 )
3715 HoleElectronConjugation::usage = "HoleElectronConjugation[params]
3716 takes the parameters (as an association) that define a
3717 configuration and converts them so that they may be interpreted as
3718 corresponding to a complementary hole configuration. Some of this
3719 can be simply done by changing the sign of the model parameters.
3720 In the case of the effective three body interaction the
3721 relationship is more complex and is controlled by the value of the
3722 t2Switch variable.";
3723 HoleElectronConjugation[params_] := Module[
3724   {newparams = params},
3725   (
3726     flipSignsOf = Join[{\zeta}, cfSymbols, TSymbols];
3727     flipped = Table[
3728       (
3729         flipper -> - newparams[flipper]
3730       ),
3731       {flipper, flipSignsOf}
3732     ];
3733     nonflipped = Table[
3734       (
3735         flipper -> newparams[flipper]
3736       ),
3737     ];
3738   );

```

```

3719     {flipper, Complement[Keys[newparams], flipSignsOf]}
3720   ];
3721   flippedParams = Association[Join[nonflipped, flipped]];
3722   flippedParams = Select[flippedParams, FreeQ[#, Missing]&];
3723   Return[flippedParams];
3724 )
3725 ];
3726
3727 IonSolver::usage = "IonSolver[numE, params, host] puts together (or
3728   retrieves from disk) the symbolic Hamiltonian for the f^numE
3729   configuration and solves it for the given params.
3730   params is an Association with keys equal to parameter symbols and
3731   values their numerical values. The function will replace the
3732   symbols in the symbolic Hamiltonian with their numerical values
3733   and then diagonalize the resulting matrix. Any parameter that is
3734   not defined in the params Association is assumed to be zero.
3735   host is an optional string that may be used to prepend the filename
3736   of the symbolic Hamiltonian that is saved to disk. The default is
3737   \"Ln\".
3738   The function returns the eigensystem as a list of lists where in
3739   each list the first element is the energy and the second element
3740   the corresponding eigenvector.
3741   The ordered basis in which this eigenvector is to be interpreted is
3742   the one corresponding to BasisLSJMJ[numE].
3743   The function admits the following options:
3744   \\"Include Spin-Spin\\" (bool) : If True then the spin-spin
3745   interaction is included as a contribution to the m_k operators.
3746   The default is True.
3747   \\"Overwrite Hamiltonian\\" (bool) : If True then the function will
3748   overwrite the symbolic Hamiltonian that is saved to disk to
3749   expedite calculations. The default is False. The symbolic
3750   Hamiltonian is saved to disk to the ./hams/ folder preceded by the
3751   string host.
3752   \\"Zeroes\\" (list) : A list with symbols assumed to be zero.
3753 ";
3754 Options[IonSolver] = {
3755   "Include Spin-Spin" -> True,
3756   "Overwrite Hamiltonian" -> False,
3757   "Zeroes" -> {}
3758 };
3759 IonSolver[numE_Integer, params0_Association, host_String:"Ln",
3760 OptionsPattern[]] := Module[
3761   {ln, simplifier, simpleHam, numHam, eigensys,
3762   startTime, endTime, diagonalTime,
3763   params=params0, zeroSymbols},
3764   (
3765     ln = theLanthanides[[numE]];

3766     (* This could be done when replacing values, but this produces
3767    smaller saved arrays. *)
3768     simplifier = (#-> 0) & /@ OptionValue["Zeroes"];
3769     simpleHam = SimplerSymbolicHamMatrix[numE,
3770       simplifier,
3771       "PrependToFilename" -> host,
3772       "Overwrite" -> OptionValue["Overwrite Hamiltonian"]
3773     ];
3774
3775     (* Note that we don't have to flip signs of parameters for fn
3776    beyond f7 since the matrix produced
3777    by SimplerSymbolicHamMatrix has already accounted for this. *)

3778     (* Everything that is not given is set to zero *)
3779     params = ParamPad[params];
3780     PrintFun[params];

3781     (* Enforce the override to the spin-spin contribution to the
3782    magnetic interactions *)
3783     params[\[Sigma]SS] = If[OptionValue["Include Spin-Spin"], 1,
3784     0];

3785     (* Create the numeric hamiltonian *)
3786     numHam = ReplaceInSparseArray[simpleHam, params];
3787     Clear[simpleHam];

3788     (* Eigensolver *)
3789     PrintFun["> Diagonalizing the numerical Hamiltonian ..."];

```

```

3773     startTime = Now;
3774     eigensys = Eigensystem[numHam];
3775     endTime = Now;
3776     diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"]
3777   ];
3778   PrintFun[">> Diagonalization took ", diagonalTime, " seconds."]
3779 ];
3780   eigensys = Chop[eigensys];
3781   eigensys = Transpose[eigensys];
3782 
3783   (* Shift the baseline energy *)
3784   eigensys = ShiftedLevels[eigensys];
3785   (* Sort according to energy *)
3786   eigensys = SortBy[eigensys, First];
3787   Return[eigensys];
3788 )
3789 ];
3790 
3791 ShiftedLevels::usage = "ShiftedLevels[eigenSys] takes a list of
3792 levels of the form
3793 {{energy_1, coeff_vector_1}, {energy_2, coeff_vector_2}, ...} and
3794 returns the same input except that now to every energy the minimum
3795 of all of them has been subtracted.";
3796 ShiftedLevels[originalLevels_] := Module[
3797   {groundEnergy, shifted},
3798   (
3799     groundEnergy = Sort[originalLevels][[1,1]];
3800     shifted = Map[{#[[1]] - groundEnergy, #[[2]]} &,
3801     originalLevels];
3802     Return[shifted];
3803   )
3804 ];
3805 
3806 (* ##### Optical Transitions for Levels ##### *)
3807 
3808 JuddOfeltUkSquared::usage = "JuddOfeltUkSquared[numE, params]
3809 calculates the matrix elements of the Uk operator in the level
3810 basis. These are calculated according to equation (7) in Carnall
3811 1965.
3812 The function returns a list with the following elements:
3813 - basis : A list with the allowed {SL, J} terms in the f^numE
3814 configuration. Equal to BasisLSJ[numE].
3815 - eigenSys : A list with the eigensystem of the Hamiltonian for
3816 the f^n configuration.
3817 - levelLabels : A list with the labels of the major components of
3818 the level eigenstates.
3819 - LevelUkSquared : An association with the squared matrix
3820 elements of the Uk operators in the level eigenbasis. The keys
3821 being {2, 4, 6} corresponding to the rank of the Uk operator. The
3822 basis in which the matrix elements are given is the one
3823 corresponding to the level eigenstates given in eigenSys and whose
3824 major SLJ components are given in levelLabels. The matrix is
3825 symmetric and given as a SymmetrizedArray.
3826 The function admits the following options:
3827 \\"PrintFun\\" : A function that will be used to print the progress
3828 of the calculations. The default is PrintTemporary.";
3829 Options[JuddOfeltUkSquared] = {"PrintFun" -> PrintTemporary};
3830 JuddOfeltUkSquared[numE_, params_, OptionsPattern[]] := Module[
3831   {eigenChanger, numEH, basis, eigenSys,
3832   Js, Ukm, LevelUkSquared, kRank,
3833   S, L, Sp, Lp, J, Jp, phase,
3834   braTerm, ketTerm, levelLabels,
3835   eigenVecs, majorComponentIndices},
3836   (
3837     If[Not[ValueQ[ReducedUkTable]],
3838       LoadUk[]
3839     ];
3840     numEH = Min[numE, 14-numE];
3841     PrintFun = OptionValue["PrintFun"];
3842     PrintFun["> Calculating the levels for the given parameters ...
3843 "];
3844     {basis, majorComponents, eigenSys} = LevelSolver[numE, params];
3845     (* The change of basis matrix to the eigenstate basis *)
3846     eigenChanger = Transpose[Last /@ eigenSys];

```

```

3829     PrintFun["Calculating the matrix elements of Uk in the physical
3830 coupling basis ..."];
3831     LevelUkSquared = <||>;
3832     Do[(
3833       Ukmat = Table[(  

3834         {S, L} = FindSL[braTerm[[1]]];  

3835         J = braTerm[[2]];  

3836         Jp = ketTerm[[2]];  

3837         {Sp, Lp} = FindSL[ketTerm[[1]]];  

3838         phase = Phaser[S + Lp + J + kRank];  

3839         Simplify @  

3840           phase *  

3841             Sqrt[TPO[J]*TPO[Jp]] *  

3842               SixJay[{J, Jp, kRank}, {Lp, L, S}] *  

3843                 ReducedUkTable[{numEH, 3, braTerm[[1]], ketTerm[[1]],  

3844                   kRank}])  

3845     ),  

3846     {braTerm, basis},  

3847     {ketTerm, basis}
3848   ];
3849   Ukmat = (Transpose[eigenChanger] . Ukmat . eigenChanger)^2;
3850   Chop@Ukmat;
3851   LevelUkSquared[kRank] = SymmetrizedArray[Ukmat, Dimensions[eigenChanger], Symmetric[{1, 2}]];
3852   ),
3853   {kRank, {2, 4, 6}}
3854 ];
3855 LSJmultiplets = (RemoveTrailingDigits[#[[1]]] <> ToString[
3856 InputForm[#[[2]]]]) & /@ basis;
3857 eigenVecs = Last /@ eigenSys;
3858 majorComponentIndices = Ordering[Abs[#][[-1]] & /@ eigenVecs];
3859 levelLabels = LSJmultiplets[[majorComponentIndices]];
3860 Return[{basis, eigenSys, levelLabels, LevelUkSquared}];
3861 )
3862 ];
3863 LevelElecDipoleOscillatorStrength::usage =
3864   LevelElecDipoleOscillatorStrength[numE, levelParams,
3865   juddOfeltParams] uses Judd-Ofelt theory to estimate the forced
3866   electric dipole oscillator strengths ions whose level description
3867   is determined by levelParams.
3868 The third parameter juddOfeltParams is an association with keys
3869   equal to the three Judd-Ofelt intensity parameters {\[CapitalOmega]2,
3870   \[CapitalOmega]4, \[CapitalOmega]6} and corresponding values
3871   in cm^2.
3872 The local field correction implemented here corresponds to the one
3873   given by the virtual cavity model of Lorentz.
3874 The function returns a list with the following elements:
3875 - basis : A list with the allowed {SL, J} terms in the f^n configuration. Equal to BasisLSJ[numE].
3876 - eigenSys : A list with the eigensystem of the Hamiltonian for
3877   the f^n configuration in the level description.
3878 - levelLabels : A list with the labels of the major components of
3879   the calculated levels.
3880 - oStrengthArray : A square array whose elements represent the
3881   oscillator strengths between levels such that the element
3882   oStrengthArray[[i,j]] is the oscillator strength between the
3883   levels |Subscript[\[Psi], i]> and |Subscript[\[Psi], j]>. In this
3884   array, the elements below the diagonal represent emission
3885   oscillator strengths, and elements above the diagonal represent
3886   absorption oscillator strengths.
3887 The function admits the following three options:
3888 - "PrintFun" : A function that will be used to print the progress
3889   of the calculations. The default is PrintTemporary.
3890 - "RefractiveIndex" : The refractive index of the medium where
3891   the transitions are taking place. This may be a number or a
3892   function. If a number then the oscillator strengths are calculated
3893   for assuming a wavelength-independent refractive index. If a
3894   function then the refractive indices are calculated accordingly to
3895   the wavelength of each transition (the function must admit a
3896   single argument equal to the wavelength in nm). The default is 1.
3897 - "LocalFieldCorrection" : The local field correction to be used.
3898   The default is "VirtualCavity". The options are: "
3899   VirtualCavity" and "EmptyCavity".
3900 The equation implemented here is the one given in eqn. 29 from the

```

```

review article of Hehlen (2013). See that same article for a
discussion on the local field correction.

3875 ";
3876 Options[LevelElecDipoleOscillatorStrength]={
3877 "PrintFun"      -> PrintTemporary,
3878 "RefractiveIndex" -> 1,
3879 "LocalFieldCorrection" -> "VirtualCavity"
3880 };
3881 LevelElecDipoleOscillatorStrength[numE_, levelParams_Association,
3882 juddOfeltParams_Association, OptionsPattern[]] := Module[
3883 {PrintFun, basis, eigenSys, levelLabels,
3884 LevelUkSquared, eigenEnergies, energyDiffs,
3885 oStrengthArray, nRef, \[Chi], nRefs,
3886 \[Chi]OverN, groundLevel, const,
3887 transitionFrequencies, wavelengthsInNM,
3888 fieldCorrectionType},
3889 (
3890   PrintFun = OptionValue["PrintFun"];
3891   nRef = OptionValue["RefractiveIndex"];
3892   PrintFun["Calculating the Uk^2 matrix elements for the given
parameters ..."];
3893   {basis, eigenSys, levelLabels, LevelUkSquared} =
3894     JuddOfeltUkSquared[numE, levelParams, "PrintFun" -> PrintFun];
3895   eigenEnergies = First/@eigenSys;
3896   (* converted to cm^-2 *)
3897   const = (8\[Pi]^2)/3 me/hPlanck * 10^(-4);
3898   energyDiffs = Transpose@Outer[Subtract, eigenEnergies,
3899   eigenEnergies];
3900   (* since energies are assumed in Kayser, speed of light needs
to be in cm/s, so that the frequencies are in 1/s *)
3901   transitionFrequencies = energyDiffs*cLight*100;
3902   (* grab the J for each level *)
3903   levelJs = #[[2]] & /@ eigenSys;
3904   oStrengthArray = (
3905     juddOfeltParams[[CapitalOmega2]*LevelUkSquared[2]+
3906     juddOfeltParams[[CapitalOmega4]*LevelUkSquared[4]+
3907     juddOfeltParams[[CapitalOmega6]*LevelUkSquared[6]
3908   );
3909   oStrengthArray = Abs@(const * transitionFrequencies *
3910   oStrengthArray);
3911   (* it is necessary to divide each oscillator strength by the
degeneracy of the initial level *)
3912   oStrengthArray = MapIndexed[1/(2 levelJs[[#2[[1]]]]+1) #1 &,
3913   oStrengthArray,{2}];
3914   (* including the effects of the refractive index *)
3915   fieldCorrectionType = OptionValue["LocalFieldCorrection"];
3916   Which[
3917     nRef === 1,
3918     True,
3919     NumberQ[nRef],
3920     (
3921       \[Chi] = Which[
3922         fieldCorrectionType == "VirtualCavity",
3923         (
3924           (nRef^2 + 2) / 3 )^2
3925         ),
3926         fieldCorrectionType == "EmptyCavity",
3927         (
3928           (3 * nRef^2 / (2 * nRef^2 + 1)) ^2
3929         )
3930       ];
3931       \[Chi]OverN = \[Chi] / nRef;
3932       oStrengthArray = \[Chi]OverN * oStrengthArray;
3933       (* the refractive index participates differently in
absorption and in emission *)
3934       aFunction = If[#2[[1]] > #2[[2]], #1 * nRef^2, #1] &;
3935       oStrengthArray = MapIndexed[aFunction, oStrengthArray,
3936       {2}];
3937       ),
3938       True,
3939       (
3940         wavelengthsInNM = Abs[1 / energyDiffs] * 10^7;
3941         nRefs = Map[nRef, wavelengthsInNM];
3942         Print["Calculating the oscillator strengths for the given
refractive index ..."];
3943         \[Chi] = Which[

```

```

3938         fieldCorrectionType == "VirtualCavity",
3939         (
3940             ( (nRefs^2 + 2) / 3 )^2
3941         ),
3942         fieldCorrectionType == "EmptyCavity",
3943         (
3944             ( 3 * nRefs^2 / ( 2*nRefs^2 + 1 ) )^2
3945         )
3946     ];
3947     \[Chi]OverN = \[Chi] / nRefs;
3948     oStrengthArray = \[Chi]OverN * oStrengthArray
3949   )
3950 ];
3951 Return[{basis, eigenSys, levelLabels, oStrengthArray}];
3952 )
3953 ];
3954
3955 LevelJJBlockMagDipole::usage = "LevelJJBlockMagDipole[numE, J, Jp]
3956 returns an array of the LSJ reduced matrix elements of the
3957 magnetic dipole operator between states with given J and Jp. The
3958 option \"Sparse\" can be used to return a sparse matrix. The
3959 default is to return a sparse matrix.";
3960 Options[LevelJJBlockMagDipole] = {"Sparse" -> True};
3961 LevelJJBlockMagDipole[numE_, braJ_, ketJ_, OptionsPattern[]] :=
3962 Module[
3963 {
3964   braSLJs, ketSLJs,
3965   braSLJ, ketSLJ,
3966   braSL, ketSL,
3967   braS, braL,
3968   ketS, ketL,
3969   matValue, magMatrix,
3970   summand1, summand2
3971 },
3972 (
3973   braSLJs = AllowedNKSLforJTerms[numE, braJ];
3974   ketSLJs = AllowedNKSLforJTerms[numE, ketJ];
3975   magMatrix = Table[
3976     (
3977       braSL = braSLJ[[1]];
3978       ketSL = ketSLJ[[1]];
3979       {braS, braL} = FindSL[braSL];
3980       {ketS, ketL} = FindSL[ketSL];
3981       summand1 = If[Or[braJ != ketJ, braSL != ketSL],
3982                     0,
3983                     Sqrt[braJ*(braJ+1)*TPO[braJ]
3984                     ]
3985                   ];
3986       (*looking at the string includes checking L=L', S=S', and \alpha
3987      = \alpha'*)
3988       summand2 = If[braSL != ketSL,
3989                     0,
3990                     (gs-1)*
3991                     Phaser[braS+braL+ketJ+1]*
3992                     Sqrt[TPO[braJ]*TPO[ketJ]]*
3993                     SixJay[{braJ, 1, ketJ}, {braS, braL, braS}]*
3994                     Sqrt[braS(braS+1)TPO[braS]]
3995                   ];
3996       matValue = summand1 + summand2;
3997       matValue = -1/2 * matValue;
3998       matValue
3999     ),
4000     {braSLJ, braSLJs},
4001     {ketSLJ, ketSLJs}
4002   ];
4003   If[OptionValue["Sparse"],
4004     magMatrix = SparseArray[magMatrix]];
4005   Return[magMatrix];
4006 )
4007 ];
4008
4009 LevelMagDipoleMatrixAssembly::usage = "LevelMagDipoleMatrixAssembly
4010 [numE] puts together an array with the reduced matrix elements of
4011 the magnetic dipole operator in the level basis for the f^numE
4012 configuration. The function admits the two following options:
4013 \"Flattened\": If True then the returned matrix is flattened. The

```

```

    default is True.
4005  \\"gs\\": The electronic gyromagnetic ratio. The default is 2.";
4006  Options[LevelMagDipoleMatrixAssembly] = {
4007    "Flattened" -> True,
4008    gs -> 2
4009  };
4010  LevelMagDipoleMatrixAssembly[numE_, OptionsPattern[]] := Module[
4011    {Js, magDip, braJ, ketJ},
4012    (
4013      Js       = AllowedJ[numE];
4014      magDip   = Table[
4015        ReplaceInSparseArray[LevelJJBlockMagDipole[numE, braJ, ketJ], {
4016          {gs -> OptionValue[gs]}, {
4017            {braJ, Js}, {ketJ, Js}
4018          ]];
4019        If[OptionValue["Flattened"],
4020          magDip = ArrayFlatten[magDip];
4021        ];
4022        Return[magDip];
4023      ]
4024    ];
4025
4026  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 |Subscript[[\Psi], i]> and |Subscript[[\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.
4027 The function admits the following options:
4028   \"Units\" : The units in which the line strengths are given. The
   default is \"SI\". The options are \"SI\" and \"Hartree\". If \"SI\"
   then the unit of the line strength is (A m^2)^2 = (J/T)^2. If \
   \"Hartree\" then the line strength is given in units of 2 \[Mu]B.";
4029  Options[LevelMagDipoleLineStrength] = {
4030    "Units" -> "SI"
4031  };
4032  LevelMagDipoleLineStrength[theEigensys_List, numE0_Integer,
4033    OptionsPattern[]] := Module[
4034    {numE, levelMagOp, allEigenvecs, magDipoleLineStrength, units},
4035    (
4036      numE       = Min[14 - numE0, numE0];
4037      levelMagOp = LevelMagDipoleMatrixAssembly[numE];
4038      allEigenvecs = Transpose[Last /@ theEigensys];
4039      units       = OptionValue["Units"];
4040      magDipoleLineStrength           = Transpose[allEigenvecs].
4041      levelMagOp.allEigenvecs;
4042      magDipoleLineStrength           = Abs[magDipoleLineStrength]^2;
4043      Which[
4044        units == "SI",
4045          Return[4 \[Mu]B^2 * magDipoleLineStrength],
4046        units == "Hartree",
4047          Return[magDipoleLineStrength]
4048      ];
4049    ];
4050
4051  LevelMagDipoleOscillatorStrength::usage =
4052    LevelMagDipoleOscillatorStrength[eigenSys, numE] calculates the
   magnetic dipole oscillator strengths for an ion whose level
   description is determined by levelParams. The refractive index of
   the medium is relevant, but here it is assumed to be 1, this can
   be changed through the option \"RefractiveIndex\". eigenSys must
   consist of a lists of lists with three elements: the first element
   being the energy of the level, the second element being the J of
   the level, and the third element being the eigenvector of the
   level.
4053  The function returns a list with the following elements:
   - basis : A list with the allowed {SL, J} terms in the f^numE
   configuration. Equal to BasisLSJ[numE].
   - eigenSys : A list with the eigensystem of the Hamiltonian for

```

```

4054     the f^n configuration in the level description.
4055     - levelLabels : A list with the labels of the major components
4056     of the calculated levels.
4057     - magDipole0strength : A square array whose elements represent
4058     the magnetic dipole oscillator strengths between the levels given
4059     in eigenSys such that the element magDipole0strength[[i,j]] is the
4060     oscillator strength between the levels |Subscript[\[Psi], i]> and
4061     |Subscript[\[Psi], j]>. In this array the elements below the
4062     diagonal represent emission oscillator strengths, and elements
4063     above the diagonal represent absorption oscillator strengths. The
4064     emission oscillator strengths are negative. The oscillator
4065     strength is a dimensionless quantity.
4066     The function admits the following option:
4067     \"RefractiveIndex\" : The refractive index of the medium where
4068     the transitions are taking place. This may be a number or a
4069     function. If a number then the oscillator strengths are calculated
4070     assuming a wavelength-independent refractive index as given. If a
4071     function then the refractive indices are calculated accordingly
4072     to the vacuum wavelength of each transition (the function must
4073     admit a single argument equal to the wavelength in nm). The
4074     default is 1.
4075     For reference see equation (27.8) in Rudzikas (2007). The
4076     expression for the line strength is the simplest when using atomic
4077     units, (27.8) is missing a factor of  $\alpha^2$ .;
4078 Options[LevelMagDipoleOscillatorStrength]={
4079   "RefractiveIndex" -> 1
4080 };
4081 LevelMagDipoleOscillatorStrength[eigenSys_, numE_, OptionsPattern[
4082   {}]:=Module[
4083   {eigenEnergies, eigenVecs, levelJs,
4084   energyDiffs, magDipole0strength, nRef,
4085   wavelengthsInNM, nRefs, degenDivisor},
4086   (
4087     basis      = BasisLSJ[numE];
4088     eigenEnergies = First/@eigenSys;
4089     nRef       = OptionValue["RefractiveIndex"];
4090     eigenVecs    = Last/@eigenSys;
4091     levelJs     = #[[2]]&/@eigenSys;
4092     energyDiffs  = -Outer[Subtract,eigenEnergies,eigenEnergies];
4093     energyDiffs *= kayserToHartree;
4094     magDipole0strength = LevelMagDipoleLineStrength[eigenSys, numE,
4095     "Units"->"Hartree"];
4096     magDipole0strength = 2/3 *  $\alpha$ Fine^2 * energyDiffs *
4097     magDipole0strength;
4098     degenDivisor   = #1 / ( 2 * levelJs[[#2[[1]]]] + 1 ) &;
4099     magDipole0strength = MapIndexed[degenDivisor,
4100     magDipole0strength, {2}];
4101     Which[nRef==1,
4102       True,
4103       NumberQ[nRef],
4104       (
4105         magDipole0strength = nRef * magDipole0strength;
4106       ),
4107       True,
4108       (
4109         wavelengthsInNM = Abs[kayserToHartree / energyDiffs] *
4110         10^7;
4111         nRefs           = Map[nRef, wavelengthsInNM];
4112         magDipole0strength = nRefs * magDipole0strength;
4113       )
4114     ];
4115     Return[{basis, eigenSys, magDipole0strength}];
4116   )
4117 ];
4118
4119 LevelMagDipoleSpontaneousDecayRates::usage =
4120 LevelMagDipoleSpontaneousDecayRates[eigenSys, numE] calculates the
4121 spontaneous emission rates for the magnetic dipole transitions
4122 between the levels given in eigenSys. The function returns a
4123 square array whose elements represent the spontaneous emission
4124 rates between the levels given in eigenSys such that the element
4125 [[i,j]] of the returned array is the rate of spontaneous emission
4126 from the level |Subscript[\[Psi], i]> to the level |Subscript[\[Psi],
4127 j]>. In this array the elements below the diagonal represent
4128 emission rates, and elements above the diagonal are identically
4129 zero.
```

```

4096 The function admits two optional arguments:
4097 + \"Units\" : The units in which the rates are given. The default
4098 is \"SI\". The options are \"SI\" and \"Hartree\". If \"SI\" then
4099 the rates are given in s^-1. If \"Hartree\" then the rates are
4100 given in the atomic unit of frequency.
4101 + \"RefractiveIndex\" : The refractive index of the medium where
4102 the transitions are taking place. This may be a number or a
4103 function. If a number then the rates are calculated assuming a
4104 wavelength-independent refractive index as given. If a function
4105 then the refractive indices are calculated accordingly to the
4106 vacuum wavelength of each transition (the function must admit a
4107 single argument equal to the wavelength in nm). The default is 1."
4108 ;
4109 Options[LevelMagDipoleSpontaneousDecayRates] = {
4110 "Units" -> "SI",
4111 "RefractiveIndex" -> 1};
4112 LevelMagDipoleSpontaneousDecayRates[eigenSys_List, numE_Integer,
4113 OptionsPattern[]] := Module[
4114 {levMDlineStrength, eigenEnergies, energyDiffs,
4115 levelJs, spontaneousRatesInHartree, spontaneousRatesInSI,
4116 degenDivisor, units, nRef, nRefs, wavelengthsInNM},
4117 (
4118 nRef = OptionValue["RefractiveIndex"];
4119 units = OptionValue["Units"];
4120 levMDlineStrength =
4121 LowerTriangularize@LevelMagDipoleLineStrength[eigenSys, numE, "Units"
4122 "->" "Hartree"];
4123 levMDlineStrength = SparseArray[levMDlineStrength];
4124 eigenEnergies = First /@ eigenSys;
4125 energyDiffs = Outer[Subtract, eigenEnergies,
4126 eigenEnergies];
4127 energyDiffs = kayserToHartree * energyDiffs;
4128 energyDiffs = SparseArray[LowerTriangularize[energyDiffs
4129 ]];
4130 levelJs = #[[2]] & /@ eigenSys;
4131 spontaneousRatesInHartree = 4/3 αFine^5 * energyDiffs^3 *
4132 levMDlineStrength;
4133 degenDivisor = #1 / (2*levelJs[[#2[[1]]]] + 1) &;
4134 spontaneousRatesInHartree = MapIndexed[degenDivisor,
4135 spontaneousRatesInHartree, {2}];
4136 Which[nRef === 1,
4137 True,
4138 NumberQ[nRef],
4139 (
4140 spontaneousRatesInHartree = nRef^3 *
4141 spontaneousRatesInHartree;
4142 ),
4143 True,
4144 (
4145 wavelengthsInNM = Abs[kayserToHartree / energyDiffs] *
4146 10^7;
4147 nRefs = Map[nRef, wavelengthsInNM];
4148 spontaneousRatesInHartree = nRefs^3 *
4149 spontaneousRatesInHartree;
4150 )
4151 ];
4152 If[units == "SI",
4153 (
4154 spontaneousRatesInSI = 1/hartreeTime *
4155 spontaneousRatesInHartree;
4156 Return[SparseArray@spontaneousRatesInSI];
4157 ),
4158 Return[SparseArray@spontaneousRatesInHartree];
4159 ];
4160 ]
4161
4162 (* ##### Optical Transitions for Levels ##### *)
4163 (* ##### ###### ###### ###### ###### ###### ###### *)
4164
4165 (* ##### ###### ###### ###### ###### ###### ###### *)
4166 (* ##### ###### ###### ###### Eigensystem analysis ###### *)
4167
4168 PrettySaundersSL::usage = "PrettySaundersSL[SL] produces a human-
4169 readable symbol for the spectroscopic term SL. SL can be either a
4170 string (in RS notation for the term) or a list of two numbers {s,

```

```

L}. The option \"Representation\" can be used to specify whether
the output is given as a symbol or as a ket. The default is \"Ket\".
";
4149 Options[PrettySaundersSL] = {"Representation" -> "Ket"};
4150 PrettySaundersSL[SL_, OptionsPattern[]] := (
4151   If[StringQ[SL],
4152     (
4153       {S,L} = FindSL[SL];
4154       L      = StringTake[SL,{2,-1}];
4155     ),
4156     {S,L}=SL
4157   ];
4158   pretty = RowBox[{(
4159     AdjustmentBox[Style[2*S+1,Smaller], BoxBaselineShift->-1,
4160     BoxMargins->0],
4161     AdjustmentBox[PrintL[L]]
4162   }];
4163   pretty = DisplayForm[pretty];
4164   pretty = Which[
4165     OptionValue["Representation"]=="Ket",
4166     Ket[pretty],
4167     OptionValue["Representation"]=="Symbol",
4168     pretty
4169   ];
4170   Return[pretty];
4171 );
4172
4173 PrettySaundersSLJmJ::usage = "PrettySaundersSLJmJ[{SL, J, mJ}]
produces a human-redeable symbol for the given basis vector {SL, J
, mJ}.";
4174 Options[PrettySaundersSLJmJ] = {"Representation" -> "Ket"};
4175 PrettySaundersSLJmJ[{SL_, J_, mJ_}, OptionsPattern[]] := (If[
4176   StringQ[SL],
4177   ({S, L} = FindSL[SL];
4178     L = StringTake[SL, {2, -1}];
4179   ),
4180   {S, L} = SL];
4181   pretty = RowBox[{AdjustmentBox[Style[2*S + 1, Smaller],
4182     BoxBaselineShift -> -1, BoxMargins -> 0],
4183     AdjustmentBox[PrintL[L], BoxMargins -> -0.2],
4184     AdjustmentBox[
4185       Style[Row[{InputForm[J], ", ", mJ}], Small],
4186       BoxBaselineShift -> 1,
4187       BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]}];
4188   pretty = DisplayForm[pretty];
4189   If[OptionValue["Representation"] == "Ket",
4190     pretty = Ket[pretty]
4191   ];
4192   Return[pretty];
4193 );
4194
4195 PrettySaundersSLJ::usage = "PrettySaundersSLJ[{SL, J}] produces a
human-redeable symbol for the given basis vector {SL, J}. SL can
be either a list of two numbers representing S and L or a string
representing the spin multiplicity and the total orbital angular
momentum J in spectroscopic notation. The option \"Representation\"
can be used to specify whether the output is given as a symbol
or as a ket. The default is \"Ket\".";
4196 Options[PrettySaundersSLJ] = {"Representation"->"Ket"};
4197 PrettySaundersSLJ[{SL_,J_},OptionsPattern[]]:= (
4198   If[StringQ[SL],
4199     (
4200       {S,L}=FindSL[SL];
4201       L=StringTake[SL,{2,-1}];
4202     ),
4203     {S,L}=SL
4204   ];
4205   pretty = RowBox[{(
4206     AdjustmentBox[Style[2*S+1,Smaller],BoxBaselineShift->-1,
4207     BoxMargins->0],
4208     AdjustmentBox[PrintL[L],BoxMargins->-0.2],
4209     AdjustmentBox[Style[InputForm[J],Small,FontTracking->"Narrow"
4210 ],BoxBaselineShift->1,BoxMargins->{{0.7,0},{0.4,0.4}}]
4211   }]
4212 );

```

```

4211 pretty = DisplayForm[pretty];
4212 pretty = Which[
4213   OptionValue["Representation"] == "Ket",
4214   Ket[pretty],
4215   OptionValue["Representation"] == "Symbol",
4216   pretty
4217 ];
4218 Return[pretty];
4219 );
4220
4221 BasisVecInRusselSaunders::usage = "BasisVecInRusselSaunders[
4222 basisVec] takes a basis vector in the format {LSstring, Jval,
4223 mJval} and returns a human-readable symbol for the corresponding
4224 Russel-Saunders term.";
4225 BasisVecInRusselSaunders[basisVec_] := (
4226 {LSstring, Jval, mJval} = basisVec;
4227 Ket[PrettySaundersSLJMJ[basisVec]]
4228 );
4229
4230 LSJMTemplate =
4231 StringTemplate[
4232 "#!\\(*TemplateBox[{\\nRowBox[{\"`LS`\", \"\", \"`\", \\nRowBox[{\"`J``,
4233 \"=\", \"`J`\"}], \"\", \"`\", \\nRowBox[{\"`mJ`\", \"=\", \"`mJ`\"}]}]},\\n`Ket`])"];
4234
4235 BasisVecInLSJM::usage = "BasisVecInLSJM[basisVec] takes a basis
4236 vector in the format {{LSstring, Jval}, mJval}, nucSpin} and
4237 returns a human-readable symbol for the corresponding LSJM term
4238 in the form |LS, J=..., mJ=...>.";
4239 BasisVecInLSJM[basisVec_] := (
4240 {LSstring, Jval, mJval} = basisVec;
4241 LSJMTemplate[<
4242 "LS" -> LSstring,
4243 "J" -> ToString[Jval, InputForm],
4244 "mJ" -> ToString[mJval, InputForm]|>]
4245 );
4246
4247 ParseStates::usage = "ParseStates[eigenSys, basis] takes a list of
4248 eigenstates in terms of their coefficients in the given basis and
4249 returns a list of the same states in terms of their energy, LSJM
4250 symbol, J, mJ, S, L, LSJ symbol, and LS symbol. eigenSys is a list
4251 of lists with two elements, in each list the first element is the
4252 energy and the second one the corresponding eigenvector. The LS
4253 symbol returned corresponds to the term with the largest
4254 coefficient in the given basis.";
4255 ParseStates[states_, basis_, OptionsPattern[]}]:=Module[
4256 {parsedStates},
4257 (
4258 parsedStates = Table[(  

4259 {energy, eigenVec} = state;  

4260 maxTermIndex = Ordering[Abs[eigenVec]][[-1]];
4261 {LSstring, Jval, mJval} = basis[[maxTermIndex]];
4262 LSJsymbol = Subscript[LSstring, {Jval, mJval}];
4263 LSJMsymbol = LSstring <> ToString[Jval,  

4264 InputForm];
4265 {S, L} = FindSL[LSstring];
4266 {energy, LSstring, Jval, mJval, S, L, LSJsymbol, LSJMsymbol}
4267 ),
4268 {state, states}
4269 ];
4270 Return[parsedStates];
4271 )
4272 ];
4273
4274 ParseStatesByNumBasisVecs::usage = "ParseStatesByNumBasisVecs[
4275 eigenSys, basis, numBasisVecs, roundTo] takes a list of
4276 eigenstates (given in eigenSys) in terms of their coefficients in
4277 the given basis and returns a list of the same states in terms of
4278 their energy and the coefficients at most numBasisVecs basis
4279 vectors. By default roundTo is 0.01 and this is the value used to
4280 round the amplitude coefficients. eigenSys is a list of lists with
4281 two elements, in each list the first element is the energy and
4282 the second one the corresponding eigenvector.
4283 The option \"Coefficients\" can be used to specify whether the
4284 coefficients are given as \"Amplitudes\" or \"Probabilities\". The
4285 default is \"Amplitudes\".

```

```

4263 ";
4264 Options[ParseStatesByNumBasisVecs] = {
4265   "Coefficients" -> "Amplitudes",
4266   "Representation" -> "Ket",
4267   "ReturnAs" -> "Dot"
4268 };
4269 ParseStatesByNumBasisVecs[eigensys_List, basis_List,
4270   numBasisVecs_Integer, roundTo_Real : 0.01, OptionsPattern[]] :=
4271 Module[
4272 {parsedStates, energy, eigenVec,
4273  probs, amplitudes, ordering,
4274  returnAs,
4275  chosenIndices, majorComponents,
4276  majorAmplitudes, majorRep},
4277 (
4278   returnAs = OptionValue["ReturnAs"];
4279   parsedStates = Table[(
4280     {energy, eigenVec} = state;
4281     energy = Chop[energy];
4282     probs = Round[Abs[eigenVec^2], roundTo];
4283     amplitudes = Round[eigenVec, roundTo];
4284     ordering = Ordering[probs];
4285     chosenIndices = ordering[[-numBasisVecs ;;]];
4286     majorComponents = basis[[chosenIndices]];
4287     majorThings = If[OptionValue["Coefficients"] == "
4288 Probabilities",
4289     (
4290       probs[[chosenIndices]]
4291     ),
4292     (
4293       amplitudes[[chosenIndices]]
4294     )
4295   ];
4296   majorComponents = PrettySaundersSLJmJ[#, "Representation"
4297 -> OptionValue["Representation"]] & /@ majorComponents;
4298   nonZ = (# != 0.) & /@ majorThings;
4299   majorThings = Pick[majorThings, nonZ];
4300   majorComponents = Pick[majorComponents, nonZ];
4301   If[OptionValue["Coefficients"] == "Probabilities",
4302     (
4303       majorThings = majorThings * 100 * "%";
4304     )
4305   ];
4306   majorRep = Which[
4307     returnAs == "Dot",
4308       majorThings . majorComponents,
4309     returnAs == "List",
4310       Transpose[{Reverse@majorThings,
4311 Reverse@majorComponents}]
4312   ];
4313   {energy, majorRep}
4314 },
4315 {state, eigensys}];
4316 Return[parsedStates]
4317 )
4318 ];
4319 FindThresholdPosition::usage = "FindThresholdPosition[list,
4320 threshold] returns the position of the first element in list that
4321 is greater than or equal to threshold. If no such element exists,
4322 it returns the length of list. The elements of the given list must
4323 be in ascending order.";
4324 FindThresholdPosition[list_, threshold_] := Module[
4325 {position},
4326 (
4327   position = Position[list, _?(# >= threshold &), 1, 1];
4328   thrPos = If[Length[position] > 0,
4329     position[[1, 1]],
4330     Length[list]];
4331   If[thrPos == 0,
4332     Return[1],
4333     Return[thrPos]
4334   ]
4335 ];
4336 ];
4337 ParseStateByProbabilitySum[{energy_, eigenVec_}, probSum_, roundTo_

```

```

4330 :0.01, maxParts_:20] := Compile[
4331 {{energy, _Real, 0}, {eigenVec, _Complex, 1},
4332 {probSum, _Real, 0}, {roundTo, _Real, 0},
4333 {maxParts, _Integer, 0}},
4334 Module[
4335 {numStates, state, amplitudes, probs, ordering,
4336 orderedProbs, truncationIndex, accProb, thresholdIndex,
4337 chosenIndices, majorComponents,
4338 majorAmplitudes, absMajorAmplitudes, notnullAmplitudes,
4339 majorRep},
4340 (
4341 numStates = Length[eigenVec];
4342 (*Round them up*)
4343 amplitudes = Round[eigenVec, roundTo];
4344 probs = Round[Abs[eigenVec^2], roundTo];
4345 ordering = Reverse[Ordering[probs]];
4346 (*Order the probabilities from high to low*)
4347 orderedProbs = probs[[ordering]];
4348 (*To speed up Accumulate, assume that only as much as
4349 maxParts will be needed*)
4350 truncationIndex = Min[maxParts, Length[orderedProbs]];
4351 orderedProbs = orderedProbs[[;;truncationIndex]];
4352 (*Accumulate the probabilities*)
4353 accProb = Accumulate[orderedProbs];
4354 (*Find the index of the first element in accProb that is
4355 greater than probSum*)
4356 thresholdIndex = Min[Length[accProb],
4357 FindThresholdPosition[accProb, probSum]];
4358 (*Grab all the indicees up till that one*)
4359 chosenIndices = ordering[[;; thresholdIndex]];
4360 (*Select the corresponding elements from the basis*)
4361 majorComponents = basis[[chosenIndices]];
4362 (*Select the corresponding amplitudes*)
4363 majorAmplitudes = amplitudes[[chosenIndices]];
4364 (*Take their absolute value*)
4365 absMajorAmplitudes = Abs[majorAmplitudes];
4366 (*Make sure that there are no effectively zero
4367 contributions*)
4368 notnullAmplitudes = Flatten[Position[absMajorAmplitudes,
4369 x_ /; x != 0]];
4370 (* majorComponents = PrettySaundersSLJmJ
4371 [#[[1]], #[[2]], #[[3]]] & /@ majorComponents; *)
4372 majorComponents = PrettySaundersSLJmJ /@ majorComponents
4373 ;
4374 majorAmplitudes = majorAmplitudes[[notnullAmplitudes]];
4375 (*Make them into Kets*)
4376 majorComponents = Ket /@ majorComponents[[[
4377 notnullAmplitudes]];
4378 (*Multiply and add to build the final Ket*)
4379 majorRep = majorAmplitudes . majorComponents;
4380 Return[{energy, majorRep}];
4381 )
4382 ],
4383 CompilationTarget -> "C",
4384 RuntimeAttributes -> {Listable},
4385 Parallelization -> True,
4386 RuntimeOptions -> "Speed"
4387 ];
4388 ParseStatesByProbabilitySum::usage = "ParseStatesByProbabilitySum[
4389 eigensys, basis, probSum] takes a list of eigenstates in terms of
4390 their coefficients in the given basis and returns a list of the
4391 same states in terms of their energy and the coefficients of the
4392 basis vectors that sum to at least probSum.";
4393 ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ :
4394 0.01, maxParts_: 20] := Module[
4395 {parsedByProb, numStates, state, energy,
4396 eigenVec, amplitudes, probs, ordering,
4397 orderedProbs, truncationIndex, accProb,
4398 thresholdIndex, chosenIndices, majorComponents,
4399 majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep
500 },
501 (
502 numStates = Length[eigensys];
503 parsedByProb = Table[(  

504 state = eigensys[[idx]];

```

```

4389 {energy, eigenVec} = state;
4390 (*Round them up*)
4391 amplitudes = Round[eigenVec, roundTo];
4392 probs = Round[Abs[eigenVec^2], roundTo];
4393 ordering = Reverse[Ordering[probs]];
4394 (*Order the probabilities from high to low*)
4395 orderedProbs = probs[[ordering]];
4396 (*To speed up Accumulate, assume that only as much as
4397 maxParts will be needed*)
4398 truncationIndex = Min[maxParts, Length[orderedProbs]];
4399 orderedProbs = orderedProbs[[;; truncationIndex]];
4400 (*Accumulate the probabilities*)
4401 accProb = Accumulate[orderedProbs];
4402 (*Find the index of the first element in accProb that is
4403 greater than probSum*)
4404 thresholdIndex = Min[Length[accProb],
4405 FindThresholdPosition[accProb, probSum]];
4406 (*Grab all the indicees up till that one*)
4407 chosenIndices = ordering[[;; thresholdIndex]];
4408 (*Select the corresponding elements from the basis*)
4409 majorComponents = basis[[chosenIndices]];
4410 (*Select the corresponding amplitudes*)
4411 majorAmplitudes = amplitudes[[chosenIndices]];
4412 (*Take their absolute value*)
4413 absMajorAmplitudes = Abs[majorAmplitudes];
4414 (*Make sure that there are no effectively zero contributions
4415 *)
4416 notnullAmplitudes = Flatten[Position[absMajorAmplitudes, x_/
4417 x != 0]];
4418 (* majorComponents = PrettySaundersSLJmJ
4419 {[#[[1]], #[[2]], #[[3]]]} & /@ majorComponents; *)
4420 majorComponents = PrettySaundersSLJmJ /@ majorComponents;
4421 majorAmplitudes = majorAmplitudes[[notnullAmplitudes]];
4422 majorComponents = majorComponents[[notnullAmplitudes]];
4423 (*Multiply and add to build the final Ket*)
4424 majorRep = majorAmplitudes . majorComponents;
4425 {energy, majorRep}
4426 ), {idx, numStates}];
4427 Return[parsedByProb];
4428 )
4429 ];
4430
4431 (* ##### Eigensystem analysis ##### *)
4432 (* ##### Misc ##### *)
4433 (* ##### *)
4434 (* ##### *)
4435 SymbToNum::usage = "SymbToNum[expr, numAssociation] takes an
4436 expression expr and returns what results after making the
4437 replacements defined in the given replacementAssociation. If
4438 replacementAssociation doesn't define values for expected keys,
4439 they are taken to be zero.";
4440 SymbToNum[expr_, replacementAssociation_] := (
4441 includedKeys = Keys[replacementAssociation];
4442 (*If a key is not defined, make its value zero.*)
4443 fullAssociation = Table[(
4444 If[MemberQ[includedKeys, key],
4445 ToExpression[key] -> replacementAssociation[key],
4446 ToExpression[key] -> 0
4447 ]
4448 ),
4449 {key, paramSymbols}];
4450 Return[expr/.fullAssociation];
4451 );
4452
4453 SimpleConjugate::usage = "SimpleConjugate[expr] takes an expression
4454 and applies a simplified version of the conjugate in that all it
4455 does is that it replaces the imaginary unit I with -I. It assumes
4456 that every other symbol is real so that it remains the same under
4457 complex conjugation. Among other expressions it is valid for any
4458 rational or polynomial expression with complex coefficients and
4459 real variables.";
4460 SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;
4461
4462 ExportMZip::usage = "ExportMZip[\"dest.[zip,m]\"] saves a

```

```

        compressed version of expr to the given destination.";
4449 ExportMZip[filename_, expr_] := Module[
4450   {baseName, exportName, mImportName, zipImportName},
4451   (
4452     baseName = FileBaseName[filename];
4453     exportName = StringReplace[filename, ".m" -> ".zip"];
4454     mImportName = StringReplace[exportName, ".zip" -> ".m"];
4455     If[FileExistsQ[mImportName],
4456     (
4457       PrintTemporary[mImportName <> " exists already, deleting"];
4458       DeleteFile[mImportName];
4459       Pause[2];
4460     )
4461   ];
4462   Export[exportName, (baseName <> ".m") -> expr];
4463   )
4464 ];
4465
4466 ImportMZip::usage = "ImportMZip[filename] imports a .m file inside
a .zip file with corresponding filename. If the Option \"Leave
Uncompressed\" is set to True (the default) then this function
also leaves an umcompressed version of the object in the same
folder of filename";
4467 Options[ImportMZip] = {"Leave Uncompressed" -> True};
4468 ImportMZip[filename_String, OptionsPattern[]] := Module[
4469   {baseName, importKey, zipImportName, mImportName, imported},
4470   (
4471     baseName = FileBaseName[filename];
4472     (*Function allows for the filename to be .m or .zip*)
4473     importKey = baseName <> ".m";
4474     zipImportName = StringReplace[filename, ".m" -> ".zip"];
4475     mImportName = StringReplace[zipImportName, ".zip" -> ".m"];
4476     mxImportName = StringReplace[zipImportName, ".zip" -> ".mx"];
4477     Which[
4478       FileExistsQ[mxImportName],
4479       (
4480         PrintTemporary[".mx version exists already, importing that
instead ..."];
4481         Return[Import[mxImportName]];
4482       ),
4483       FileExistsQ[mImportName],
4484       (
4485         PrintTemporary[".m version exists already, importing that
instead ..."];
4486         Return[Import[mImportName]];
4487       )
4488     ];
4489     imported = Import[zipImportName, importKey];
4490     If[OptionValue["Leave Uncompressed"],
4491     (
4492       Export[mImportName, imported];
4493       Export[mxImportName, imported];
4494     )
4495   ];
4496   Return[imported];
4497   )
4498 ];
4499
4500 ReplaceInSparseArray::usage = "ReplaceInSparseArray[sparseArray,
rules] takes a sparse array that may contain symbolic quantities
and returns a sparse array in which the given rules have been used
on every element.";
4501 ReplaceInSparseArray[sparseA_SparseArray, rules_] := (
4502   SparseArray[Automatic,
4503   sparseA["Dimensions"],
4504   sparseA["Background"] /. rules,
4505   {
4506     1,
4507     {sparseA["RowPointers"], sparseA["ColumnIndices"]},
4508     sparseA["NonzeroValues"] /. rules
4509   }
4510   ]
4511 );
4512
4513 MapToSparseArray::usage = "MapToSparseArray[sparseArray, function]
takes a sparse array and returns a sparse array after the function

```

```

    has been applied to it.";
4514 MapToSparseArray[sparseA_SparseArray, func_] := Module[
4515   {nonZ, backg, mapped},
4516   (
4517     nonZ = func /@ sparseA["NonzeroValues"];
4518     backg = func[sparseA["Background"]];
4519     mapped = SparseArray[Automatic,
4520       sparseA["Dimensions"],
4521       backg,
4522       {
4523         1,
4524         {sparseA["RowPointers"], sparseA["ColumnIndices"]},
4525         nonZ
4526       }
4527     ];
4528     Return[mapped];
4529   )
4530 ];
4531
4532 ParseTeXLikeSymbol::usage = "ParseTeXLikeSymbol[string] parses a
4533   string for a symbol given in LaTeX notation and returns a
4534   corresponding mathematica symbol. The string may have expressions
4535   for several symbols, they need to be separated by single spaces.
4536   In addition the _ and ^ symbols used in LaTeX notation need to
4537   have arguments that are enclosed in parenthesis, for example \"x_2
4538   \" is invalid, instead \"x_{2}\\" should have been given.";
4539 Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
4540 ParseTeXLikeSymbol[bigString_, OptionsPattern[]} := Module[
4541   {form, mainSymbol, symbols},
4542   (
4543     form = OptionValue["Form"];
4544     (* parse greek *)
4545     symbols = Table[((
4546       str = StringReplace[string, {"\\alpha" -> "\[Alpha]",
4547         "\\beta" -> "\[Beta]",
4548         "\\gamma" -> "\[Gamma]",
4549         "\\psi" -> "\[Psi]"}];
4550       symbol = Which[
4551         StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"]
4552       ],
4553         (
4554           (*yes sub no sup*)
4555           mainSymbol = StringSplit[str, "_"][[1]];
4556           mainSymbol = ToExpression[mainSymbol];
4557
4558           subPart =
4559             StringCases[str,
4560               RegularExpression@"\\{(.*)\\}" -> "$1"][[1]];
4561             Subscript[mainSymbol, subPart]
4562           ),
4563           Not[StringContainsQ[str, "_"]] && StringContainsQ[str, "^"]
4564         ],
4565         (
4566           (*no sub yes sup*)
4567           mainSymbol = StringSplit[str, "^"][[1]];
4568           mainSymbol = ToExpression[mainSymbol];
4569
4570           supPart =
4571             StringCases[str,
4572               RegularExpression@"\\{(.*)\\}" -> "$1"][[1]];
4573             Superscript[mainSymbol, supPart]
4574           ),
4575           StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
4576           (
4577             (*yes sub yes sup*)
4578             mainSymbol = StringSplit[str, "_"][[1]];
4579             mainSymbol = ToExpression[mainSymbol];
4580             {subPart, supPart} =
4581               StringCases[str, RegularExpression@"\\{(.*)\\}" -> "
4582             $1"];
4583             Subsuperscript[mainSymbol, subPart, supPart]
4584           ),
4585           True,
4586           (
4587             (*no sup or sub*)
4588             str

```

```

4580      )
4581      ];
4582      symbol
4583      ),
4584      {string, StringSplit[bigString, " "]}
4585    ];
4586    Which[
4587      form == "Row",
4588      Return[Row[symbols]],
4589      form == "List",
4590      Return[symbols]
4591    ]
4592  )
4593];
4594
4595 FromArrayToTable::usage = "FromArrayToTable[array, labels, energies
4596 ] takes a square array of values and returns a table with the
4597 labels of the rows and columns, the energies of the initial and
4598 final levels, the level energies, the vacuum wavelength of the
4599 transition, and the value of the array. The array must be square
4600 and the labels and energies must be compatible with the order
4601 implied by the array. The array must be a square array of values.
4602 The function returns a list of lists with the following elements:
4603 - Initial level index
4604 - Final level index
4605 - Initial level label
4606 - Final level label
4607 - Initial level energy
4608 - Final level energy
4609 - Vacuum wavelength
4610 - Value of the array element.
4611 Elements in which the array is zero are not included in the return
4612 of this function.";
4613 FromArrayToTable[array_, labels_, energies_] := Module[
4614   {tableFun, atl},
4615   (
4616     tableFun = {
4617       #2[[1]],
4618       #2[[2]],
4619       labels[[#2[[1]]]],
4620       labels[[#2[[2]]]],
4621       energies[[#2[[1]]]],
4622       energies[[#2[[2]]]],
4623       If[#2[[1]] == #2[[2]], "--", 10^7/(energies[[#2[[1]]]] - energies
4624       [[#2[[2]]]])],
4625       #
4626       }&);
4627     atl = Select[Flatten[MapIndexed[tableFun, array
4628       ,{2}],1],##[[-1]]!=0.&];
4629     atl = Append[#,1/##[[-1]]]&/@atl;
4630     Return[atl]
4631   )
4632 ]
4633 (* ##### Misc #### *)
4634 (* ##### Plotting Routines #### *)
4635 (* ##### Some Plotting Routines #### *)
4636
4637 EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes
4638 states and produces a visualization of its energy spectrum.
4639 The resultant visualization can be navigated by clicking and
4640 dragging to zoom in on a region, or by clicking and dragging
4641 horizontally while pressing Ctrl. Double-click to reset the view."
4642 ;
4643 Options[EnergyLevelDiagram] = {
4644   "Title" -> "",
4645   "ImageSize" -> 1000,
4646   "AspectRatio" -> 1/8,
4647   "Background" -> "Automatic",
4648   "Epilog" -> {},
4649   "Explorer" -> True
4650 };
4651 EnergyLevelDiagram[states_, OptionsPattern[]] := Module[
4652   {energies, epi, explora},
4653   (

```

```

4642 energies = First/@states;
4643 epi = OptionValue["Epilog"];
4644 explora = If[OptionValue["Explorer"],
4645   ExploreGraphics,
4646   Identity
4647 ];
4648 explora@ListPlot[Tooltip[{#, 0}, {#, 1}], {Quantity
4649 [#/8065.54429, "eV"], Quantity[#, 1/"Centimeters"]}] &/@ energies ,
4650   Joined      -> True,
4651   PlotStyle    -> Black,
4652   AspectRatio  -> OptionValue["AspectRatio"],
4653   ImageSize    -> OptionValue["ImageSize"],
4654   Frame        -> True,
4655   PlotRange    -> {All, {0, 1}},
4656   FrameTicks   -> {{None, None}, {Automatic, Automatic}},
4657   FrameStyle   -> Directive[15, Dashed, Thin],
4658   PlotLabel    -> Style[OptionValue["Title"], 15, Bold],
4659   Background   -> OptionValue["Background"],
4660   FrameLabel   -> {"\!\(*FractionBox[\!(E\!), SuperscriptBox[\!(cm\!), \!(-1\!)]]\)"},
4661   Epilog       -> epi]
4662 )
4663 ;
4664
4665 ExploreGraphics::usage = "Pass a Graphics object to explore it.
4666   Zoom by clicking and dragging a rectangle. Pan by clicking and
4667   dragging while pressing Ctrl. Click twice to reset view.
4668 Based on ZeitPolizei @ https://mathematica.stackexchange.com/questions/7142/how-to-manipulate-2d-plots.
4669 The option \"OptAxesRedraw\" can be used to specify whether the
4670   axes should be redrawn. The default is False.";
4671 Options[ExploreGraphics] = {OptAxesRedraw -> False};
4672 ExploreGraphics[graph_Graphics, opts : OptionsPattern[]] := With[
4673   {
4674     gr = First[graph],
4675     opt = DeleteCases[Options[graph],
4676       PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _],
4677     plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
4678     ar = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
4679     ao = AbsoluteOptions[AxesOrigin],
4680     rectangle = {Dashing[Small],
4681       Line[{#1,
4682         {First[#2], Last[#1]},
4683         #2,
4684         {First[#1], Last[#2]},
4685         #1}]} &,
4686     optAxesRedraw = OptionValue[OptAxesRedraw]
4687   },
4688   DynamicModule[
4689     {dragging=False, first, second, rx1, rx2, ry1, ry2,
4690      range = plr},
4691     {{rx1, rx2}, {ry1, ry2}} = plr;
4692     Panel@
4693     EventHandler[
4694       Dynamic@Graphics[
4695         If[dragging, {gr, rectangle[first, second]}, gr],
4696         PlotRange -> Dynamic@range,
4697         AspectRatio -> ar,
4698         AxesOrigin -> If[optAxesRedraw,
4699           Dynamic@Mean[range\[Transpose]], ao],
4700           Sequence @@ opt],
4701         {"MouseDown", 1} :> (
4702           first = MousePosition["Graphics"]
4703         ),
4704         {"MouseDragged", 1} :> (
4705           dragging = True;
4706           second = MousePosition["Graphics"]
4707         ),
4708         "MouseClicked" :> (
4709           If[CurrentValue@"MouseClicked"==2,
4710             range = plr];
4711         ),
4712         {"MouseUp", 1} :> If[dragging,
4713           dragging = False;
4714
4715           range = {{rx1, rx2}, {ry1, ry2}} =
4716         ]
4717       ]
4718     ]
4719   ]
4720 ]

```

```

4712     Transpose@{first, second};
4713     range[[2]] = {0, 1}],
4714     {"MouseDown", 2} :> (
4715       first = {sx1, sy1} = MousePosition["Graphics"]
4716     ),
4717     {"MouseDragged", 2} :> (
4718       second = {sx2, sy2} = MousePosition["Graphics"];
4719       rx1 = rx1 - (sx2 - sx1);
4720       rx2 = rx2 - (sx2 - sx1);
4721       ry1 = ry1 - (sy2 - sy1);
4722       ry2 = ry2 - (sy2 - sy1);
4723       range = {{rx1, rx2}, {ry1, ry2}};
4724       range[[2]] = {0, 1};
4725     )}]];
4726 
4727 LabeledGrid::usage = "LabeledGrid[data, rowHeaders, columnHeaders]
4728 provides a grid of given data interpreted as a matrix of values
4729 whose rows are labeled by rowHeaders and whose columns are labeled
4730 by columnHeaders. When hovering with the mouse over the grid
4731 elements, the row and column labels are displayed with the given
4732 separator between them.";
4733 Options[LabeledGrid]={
4734   ItemSize->Automatic,
4735   Alignment->Center,
4736   Frame->All,
4737   "Separator"->",",
4738   "Pivot"->"
4739 };
4740 LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]] :=
4741 Module[
4742 {gridList=data, rowHeads=rowHeaders, colHeads=columnHeaders},
4743 (
4744   separator=OptionValue["Separator"];
4745   pivot=OptionValue["Pivot"];
4746   gridList=Table[
4747     Tooltip[
4748       data[[rowIdx,colIdx]],
4749       DisplayForm[
4750         RowBox[{rowHeads[[rowIdx]],
4751           separator,
4752           colHeads[[colIdx]]}
4753         ]
4754       ]
4755     ],
4756     {rowIdx,Dimensions[data][[1]]},
4757     {colIdx,Dimensions[data][[2]]}];
4758   gridList=Transpose[Prepend[gridList,colHeads]];
4759   rowHeads=Prepend[rowHeads,pivot];
4760   gridList=Prepend[gridList,rowHeads]//Transpose;
4761   Grid[gridList,
4762     Frame->OptionValue[Frame],
4763     Alignment->OptionValue[Alignment],
4764     Frame->OptionValue[Frame],
4765     ItemSize->OptionValue[ItemSize]
4766   ]
4767 )
4768 ];
4769 
4770 HamiltonianForm::usage = "HamiltonianForm[hamMatrix, basisLabels]
4771 takes the matrix representation of a hamiltonian together with a
4772 set of symbols representing the ordered basis in which the
4773 operator is represented. With this it creates a displayed form
4774 that has adequately labeled row and columns together with
4775 informative values when hovering over the matrix elements using
4776 the mouse cursor.";
4777 Options[HamiltonianForm]={"Separator"->",","Pivot"->"};
4778 HamiltonianForm[hamMatrix_, basisLabels_List, OptionsPattern[]] :=
4779 (
4780   braLabels=DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]]& /@ basisLabels;
4781   ketLabels=DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"}]]& /@ basisLabels;
4782   LabeledGrid[hamMatrix,braLabels,ketLabels,"Separator"->
4783   OptionValue["Separator"],"Pivot"->OptionValue["Pivot"]]
4784 )
4785 
```

```

4772 HamiltonianMatrixPlot::usage = "HamiltonianMatrixPlot[hamMatrix,
4773   basisLabels] creates a matrix plot of the given hamiltonian matrix
4774   with the given basis labels. The matrix elements can be hovered
4775   over to display the corresponding row and column labels together
4776   with the value of the matrix element. The option \"Overlay Values\
4777   \" can be used to specify whether the matrix elements should be
4778   displayed on top of the matrix plot.";
4779 Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover"-
4780   > True, "Overlay Values" -> True}];
4781 HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts : OptionsPattern[]] :=
4782   braLabels = DisplayForm[RowBox[{"\\"[LeftAngleBracket]", #, "\\"[
4783     RightBracketingBar]"}]] & /@ basisLabels;
4784   ketLabels = DisplayForm[Rotate[RowBox[{"\\"[LeftBracketingBar]", #,
4785     "\\"[RightAngleBracket]"}], \[Pi]/2]] & /@ basisLabels;
4786   ketLabelsUpright = DisplayForm[RowBox[{"\\"[LeftBracketingBar]", #,
4787     "\\"[RightAngleBracket]"}]] & /@ basisLabels;
4788   numRows = Length[hamMatrix];
4789   numCols = Length[hamMatrix[[1]]];
4790   epiThings = Which[
4791     And[OptionValue["Hover"], Not[OptionValue["Overlay Values"]]], ,
4792     Flatten[
4793       Table[
4794         Tooltip[
4795           {
4796             Transparent,
4797             Rectangle[
4798               {j - 1, numRows - i},
4799               {j - 1, numRows - i} + {1, 1}
4800             ]
4801           },
4802             Row[{braLabels[[i]], ketLabelsUpright[[j]], "=" , hamMatrix[[i,
4803               j]]}]
4804             ],
4805             {i, 1, numRows},
4806             {j, 1, numCols}
4807           ]
4808         ],
4809         And[OptionValue["Hover"], OptionValue["Overlay Values"]],
4810         Flatten[
4811           Table[
4812             Tooltip[
4813               {
4814                 Transparent,
4815                 Rectangle[
4816                   {j - 1, numRows - i},
4817                   {j - 1, numRows - i} + {1, 1}
4818                 ]
4819                 },
4820                 DisplayForm[RowBox[{"\\"[LeftAngleBracket]", basisLabels[[i
4821                   ]], "\\"[LeftBracketingBar]", basisLabels[[j]], "\\"[RightAngleBracket
4822                   ]"}]]
4823                 ],
4824                 {i, numRows},
4825                 {j, numCols}
4826               ]
4827             ],
4828             True,
4829             {}
4830           ];
4831           textOverlay = If[OptionValue["Overlay Values"],
4832             (
4833               Flatten[
4834                 Table[
4835                   Text[hamMatrix[[i, j]],
4836                     {j - 1/2, numRows - i + 1/2}
4837                   ],
4838                   {i, 1, numRows},
4839                   {j, 1, numCols}
4840                 ]
4841               ],
4842               {}
4843             ];
4844             epiThings = Join[epiThings, textOverlay];
4845             MatrixPlot[hamMatrix,

```

```

4834   FrameTicks -> {
4835     {Transpose[{Range[Length[braLabels]], braLabels}], None},
4836     {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}}
4837   },
4838   Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
4839   Epilog -> epiThings
4840 ]
4841 );
4842
4843 (* ##### Some Plotting Routines ##### *)
4844 (* ##### ##### ##### ##### ##### ##### *)
4845 (* ##### ##### ##### ##### ##### ##### *)
4846 (* ##### ##### ##### ##### ##### ##### *)
4847 (* ##### ##### ##### Load Functions ##### *)
4848
4849 LoadAll::usage = "LoadAll[] executes most Load* functions.";
4850 LoadAll[] := (
4851   LoadTermLabels[];
4852   LoadCFP[];
4853   LoadUk[];
4854   LoadV1k[];
4855   LoadT22[];
4856   LoadSOOandECSOLS[];
4857
4858   LoadElectrostatic[];
4859   LoadSpinOrbit[];
4860   LoadSOOandECSO[];
4861   LoadSpinSpin[];
4862   LoadThreeBody[];
4863   LoadChenDeltas[];
4864   LoadCarnall[];
4865 );
4866
4867 fnTermLabels::usage = "This list contains the labels of f^n
4868 configurations. Each element of the list has four elements {LS,
4869 seniority, W, U}. At first sight this seems to only include the
4870 labels for the f^6 and f^7 configuration, however, all is included
4871 in these two.";
4872
4873 LoadTermLabels::usage = "LoadTermLabels[] loads into the session
4874 the labels for the terms in the f^n configurations.";
4875 LoadTermLabels[] := (
4876   If[ValueQ[fnTermLabels], Return[]];
4877   PrintTemporary["Loading data for state labels in the f^n
4878 configurations..."];
4879   fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
4880
4881   If[!FileExistsQ[fnTermsFname],
4882     (PrintTemporary[">> fnTerms.m not found, generating ..."];
4883      fnTermLabels = ParseTermLabels["Export" -> True];
4884    ),
4885     fnTermLabels = Import[fnTermsFname];
4886   ];
4887 )
4888
4889 Carnall::usage = "Association of data from Carnall et al (1989)
4890 with the following keys: {data, annotations, paramSymbols,
4891 elementNames, rawData, rawAnnotations, annotatedData, appendix:Pr
4892 :Association, appendix:Pr:Calculated, appendix:Pr:RawTable,
4893 appendix:Headings}";
4894
4895 LoadCarnall::usage = "LoadCarnall[] loads data for trivalent
4896 lanthanides in LaF3 using the data from Bill Carnall's 1989 paper.
4897 ";
4898 LoadCarnall[] := (
4899   If[ValueQ[Carnall], Return[]];
4900   carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
4901   If[!FileExistsQ[carnallFname],
4902     (PrintTemporary[">> Carnall.m not found, generating ..."];
4903      Carnall = ParseCarnall[];
4904    ),
4905     Carnall = Import[carnallFname];
4906   ];
4907 )
4908
4909 LoadChenDeltas::usage = "LoadChenDeltas[] loads the differences

```

```

        noted by Chen.";
4898 LoadChenDeltas[] := (
4899   If[ValueQ[chenDeltas], Return[]];
4900   PrintTemporary["Loading the association of discrepancies found by
4901   Chen ..."];
4902   chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"}];
4903   If[!FileExistsQ[chenDeltasFname],
4904     (PrintTemporary[">> chenDeltas.m not found, generating ..."];
4905      chenDeltas = ParseChenDeltas[];
4906      chenDeltas = Import[chenDeltasFname];
4907    ];
4908  );
4909
4910 ParseChenDeltas::usage = "ParseChenDeltas[] parses the data found
4911   in ./data/the-chen-deltas-A.csv and ./data/the-chen-deltas-B.csv.
4912   If the option \"Export\" is set to True (True is the default),
4913   then the parsed data is saved to ./data/chenDeltas.m";
4914 Options[ParseChenDeltas] = {"Export" -> True};
4915 ParseChenDeltas[OptionsPattern[]] :=
4916   chenDeltasRaw = Import[FileNameJoin[{moduleDir, "data", "the-chen-
4917   -deltas-A.csv"}]];
4918   chenDeltasRaw = chenDeltasRaw[[2 ;;]];
4919   chenDeltas = <||>;
4920   chenDeltasA = <||>;
4921   Off[Power::infy];
4922   Do[
4923     ({right, wrong} = {chenDeltasRaw[[row]][[4 ;;]], 
4924       chenDeltasRaw[[row + 1]][[4 ;;]]};
4925     key = chenDeltasRaw[[row]][[1 ;; 3]];
4926     repRule = (#[[1]] -> #[[2]]*#[[1]]) & /@
4927       Transpose[{{M0, M2, M4, P2, P4, P6}, right/wrong}];
4928     chenDeltasA[key] = <|"right" -> right, "wrong" -> wrong,
4929     "repRule" -> repRule|>;
4930     chenDeltasA[{key[[1]], key[[3]], key[[2]]}] = <|"right" ->
4931     right,
4932     "wrong" -> wrong, "repRule" -> repRule|>;
4933   ),
4934   {row, 1, Length[chenDeltasRaw], 2}];
4935   chenDeltas["A"] = chenDeltasA;
4936
4937   chenDeltasRawB = Import[FileNameJoin[{moduleDir, "data", "the-
4938   -chen-deltas-B.csv"}], "Text"];
4939   chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
4940   chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
4941   chenDeltasB = {ToExpression[StringTake[#[[1]], {2}], #[[2]],
4942     #[[3]]] & /@ chenDeltasB;
4943   chenDeltas["B"] = chenDeltasB;
4944   On[Power::infy];
4945   If[OptionValue["Export"],
4946     (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.
4947     m"}];
4948     Export[chenDeltasFname, chenDeltas];
4949   )
4950 ];
4951   Return[chenDeltas];
4952 );
4953
4954 ParseCarnall::usage = "ParseCarnall[] parses the data found in ./data/Carnall.xls. If the option \"Export\" is set to True (True is the default), then the parsed data is saved to ./data/Carnall. This data is from the tables and appendices of Carnall et al (1989).";
4955 Options[ParseCarnall] = {"Export" -> True};
4956 ParseCarnall[OptionsPattern[]] :=
4957   ions = {"Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho",
4958   "Er", "Tm", "Yb"};
4959   templates = StringTemplate/@StringSplit["appendix:`ion`:
4960   Association appendix:`ion`:Calculated appendix:`ion`:RawTable
4961   appendix:`ion`:Headings", " "];
4962
4963   (* How many unique eigenvalues, after removing Kramer's
4964   degeneracy *)
4965   fullSizes = AssociationThread[ions, {7, 91, 182, 1001, 1001,
4966   3003, 1716, 3003, 1001, 1001, 182, 91, 7}];

```

```

4954      carnall      = Import[FileNameJoin[{moduleDir,"data","Carnall.xls
4955      "}]][[2]];
4956      carnallErr   = Import[FileNameJoin[{moduleDir,"data","Carnall.xls
4957      "}]][[3]];
4958
4959      elementNames = carnall[[1]][[2;;]];
4960      carnall      = carnall[[2;;]];
4961      carnallErr   = carnallErr[[2;;]];
4962      carnall      = Transpose[carnall];
4963      carnallErr   = Transpose[carnallErr];
4964      paramNames   = ToExpression@carnall[[1]][[1;;]];
4965      carnall      = carnall[[2;;]];
4966      carnallErr   = carnallErr[[2;;]];
4967      carnallData  = Table[((
4968          data           = carnall[[i]];
4969          data           = (#[[1]]->#[[2]])&/@Select[
4970          Transpose[{paramNames,data}],#[[2]]!=="&"];
4971          elementNames[[i]]->data
4972          ),
4973          {i,1,13}
4974          ];
4975      carnallData  = Association[carnallData];
4976      carnallNotes = Table[((
4977          data           = carnallErr[[i]];
4978          elementName  = elementNames[[i]];
4979          dataFun      = (
4980              #[[1]] -> If[#[[2]]==[],
4981                  "Not allowed to vary in fitting.",
4982                  If[#[[2]]=="[R]",
4983                      "Ratio constrained by: " <> <|"Eu"->"F4/
4984                      F2=0.713; F6/F2=0.512",
4985                      "Gd"->"F4/F2=0.710",
4986                      "Tb"->"F4/F2=0.707"|>[elementName],
4987                      If[#[[2]]=="i",
4988                          "Interpolated",
4989                          #[[2]]
4990                      ]
4991                      ]
4992                      ]
4993                      );
4994      carnallNotes = Association[carnallNotes];
4995
4996      annotatedData = Table[
4997          If[NumberQ[#[[1]]], Tooltip[#[[1]], #[[2]], ""]
4998          & /
4999          @ Transpose[{paramNames/.carnallData[element],
5000              paramNames/.carnallNotes[element]
5001              }],
5002          {element,elementNames}
5003          ];
5004      annotatedData = Transpose[annotatedData];
5005
5006      Carnall = <|"data"      -> carnallData,
5007          "annotations"    -> carnallNotes,
5008          "paramSymbols"   -> paramNames,
5009          "elementNames"   -> elementNames,
5010          "rawData"        -> carnall,
5011          "rawAnnotations" -> carnallErr,
5012          "includedTableIons" -> ions,
5013          "annnotatedData"  -> annotatedData
5014      |>;
5015
5016      Do[((
5017          carnallData   = Import[FileNameJoin[{moduleDir,"data",
5018          "Carnall.xls"}]][[sheetIdx]];
5019          headers       = carnallData[[1]];
5020          calcIndex     = Position[headers,"Calc (1/cm)"][[1,1]];
5021          headers       = headers[[2;;]];
5022          carnallLabels = carnallData[[1]];
5023          carnallData   = carnallData[[2;;]];
5024          carnallTerms  = DeleteDuplicates[First/@carnallData];
5025          parsedData    = Table[((
5026

```

```

5023         rows = Select[carnallData ,#[[1]]==term&];
5024         rows = #[[2;;]]&/@rows;
5025         rows = Transpose[rows];
5026         rows = Transpose[{headers,rows}];
5027         rows = Association[(#[[1]]->#[[2]])&/@rows
5028 ];
5029         term->rows
5030     ),
5031     {term,carnallTerms}
5032 ];
5033 carnallAssoc = Association[parsedData];
5034 carnallCalcEnergies = #[[calcIndex]]&/@carnallData;
5035 carnallCalcEnergies = If[NumberQ[#, #, Missing[]]&/
5036 @carnallCalcEnergies;
5037     ion = ions[[sheetIdx-3]];
5038     carnallCalcEnergies = PadRight[carnallCalcEnergies, fullSizes
5039 [ion], Missing[]];
5040     keys = #[<"ion"->ion|]&/@templates;
5041     Carnall[keys[[1]]] = carnallAssoc;
5042     Carnall[keys[[2]]] = carnallCalcEnergies;
5043     Carnall[keys[[3]]] = carnallData;
5044     Carnall[keys[[4]]] = headers;
5045   ),
5046 {sheetIdx,4,16}
5047 ];
5048
5049 goodions = Select[ions,#!=="Pm"&];
5050 expData = Select[Transpose[Carnall["appendix:<>#<>":RawTable"]
5051 ][[1+Position[Carnall["appendix:<>#<>":Headings],"Exp (1/cm)""
5052 ][[1,1]]]],NumberQ]&/@goodions;
5053 Carnall["All Experimental Data"] = AssociationThread[goodions,
5054 expData];
5055 If[OptionValue["Export"],
5056 (
5057   carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"
5058 }];
5059   Print["Exporting to "<>carnallFname];
5060   Export[carnallFname, Carnall];
5061 )
5062 ];
5063 Return[Carnall];
5064
5065 CFP::usage = "CFP[{n, NKSL}] provides a list whose first element
5066 echoes NKSL and whose other elements are lists with two elements
5067 the first one being the symbol of a parent term and the second
5068 being the corresponding coefficient of fractional parentage. n
5069 must satisfy 1 <= n <= 7.
5070 These are according to the tables from Nielson & Koster.";
5071
5072 CFPAssoc::usage = "CFPAssoc is an association where keys are of
5073 lists of the form {num_electrons, daughterTerm, parentTerm} and
5074 values are the corresponding coefficients of fractional parentage.
5075 The terms given in string-spectroscopic notation. If a certain
5076 daughter term does not have a parent term, the value is 0. Loaded
5077 using LoadCFP[].
5078 These are according to the tables from Nielson & Koster.";
5079
5080 LoadCFP::usage = "LoadCFP[] loads CFP, CFPAssoc, and CFPTable into
5081 the session.";
5082 LoadCFP[] := (
5083   If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]],Return
5084   []];
5085
5086   PrintTemporary["Loading CFPTable ..."];
5087   CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
5088   If[!FileExistsQ[CFPTablefname],
5089     (PrintTemporary[">> CFPTable.m not found, generating ..."];
5090      CFPTable = GenerateCFPTable["Export"->True];
5091    ),
5092     CFPTable = Import[CFPTablefname];
5093   ];
5094
5095   PrintTemporary["Loading CFPs.m ..."];
5096   CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
5097   If[!FileExistsQ[CFPfname],
5098

```

```

5081     (PrintTemporary[">> CFPs.m not found, generating ..."];
5082      CFP = GenerateCFP["Export" -> True];
5083    ),
5084    CFP = Import[CFPfname];
5085  ];
5086
5087 PrintTemporary["Loading CFPAssoc.m ..."];
5088 CFPAfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
5089 If[!FileExistsQ[CFPAfname],
5090   (PrintTemporary[">> CFPAssoc.m not found, generating ..."];
5091    CFPAssoc = GenerateCFPAssoc["Export" -> True];
5092   ),
5093   CFPAssoc = Import[CFPAfname];
5094 ];
5095 );
5096
5097 ReducedUkTable::usage = "ReducedUkTable[{n, l = 3, SL, SpLp, k}]
5098 provides reduced matrix elements of the unit spherical tensor
5099 operator Uk. See TASS section 11-9 \"Unit Tensor Operators\".
5099 Loaded using LoadUk[].";
5100
5101 LoadUk::usage = "LoadUk[] loads into session the reduced matrix
5102 elements for unit tensor operators.";
5103 LoadUk[] := (
5104   If[ValueQ[ReducedUkTable], Return[]];
5105   PrintTemporary["Loading the association of reduced matrix
5106 elements for unit tensor operators ..."];
5107   ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "ReducedUkTable.m"}];
5108   If[!FileExistsQ[ReducedUkTableFname],
5109     (PrintTemporary[">> ReducedUkTable.m not found, generating ..."]);
5110     ReducedUkTable = GenerateReducedUkTable[7];
5111   ],
5112   ReducedUkTable = Import[ReducedUkTableFname];
5113 );
5114 );
5115
5116 ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}]
5117 provides the calculated result of Electrostatic[{n, SL, SpLp}]."
5118 Load using LoadElectrostatic[].";
5119
5120 LoadElectrostatic::usage = "LoadElectrostatic[] loads the reduced
5121 matrix elements for the electrostatic interaction.";
5122 LoadElectrostatic[] := (
5123   If[ValueQ[ElectrostaticTable], Return[]];
5124   PrintTemporary["Loading the association of matrix elements for
5125 the electrostatic interaction ..."];
5126   ElectrostaticTableFname = FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}];
5127   If[!FileExistsQ[ElectrostaticTableFname],
5128     (PrintTemporary[">> ElectrostaticTable.m not found, generating
5129     ..."]);
5130     ElectrostaticTable = GenerateElectrostaticTable[7];
5131   ],
5132   ElectrostaticTable = Import[ElectrostaticTableFname];
5133 );
5134 );
5135
5136 LoadV1k::usage = "LoadV1k[] loads into session the matrix elements
5137 of V1k.";
5138 LoadV1k[] := (
5139   If[ValueQ[ReducedV1kTable], Return[]];
5140   PrintTemporary["Loading the association of matrix elements for
5141 V1k ..."];
5142   ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m"}];
5143   If[!FileExistsQ[ReducedV1kTableFname],
5144     (PrintTemporary[">> ReducedV1kTable.m not found, generating ...
5145     "]);
5146     ReducedV1kTable = GenerateReducedV1kTable[7];
5147   ],
5148   ReducedV1kTable = Import[ReducedV1kTableFname];
5149 );
5150 );

```

```

5140 LoadSpinOrbit::usage = "LoadSpinOrbit[] loads into session the
5141   matrix elements of the spin-orbit interaction.";
5142 LoadSpinOrbit[] := (
5143   If[ValueQ[SpinOrbitTable], Return[]];
5144   PrintTemporary["Loading the association of matrix elements for
5145   spin-orbit ..."];
5146   SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "}
5147   SpinOrbitTable.m"}];
5148   If[!FileExistsQ[SpinOrbitTableFname],
5149     (
5150       PrintTemporary[">> SpinOrbitTable.m not found, generating ..."
5151     ];
5152       SpinOrbitTable = GenerateSpinOrbitTable[7, "Export" -> True];
5153     ),
5154     SpinOrbitTable = Import[SpinOrbitTableFname];
5155   ]
5156 );
5157
5158 LoadSOOandECSOLS::usage = "LoadSOOandECSOLS[] loads into session
5159   the LS reduced matrix elements of the SOO-ECSO interaction.";
5160 LoadSOOandECSOLS[] := (
5161   If[ValueQ[SOOandECSOLSTable], Return[]];
5162   PrintTemporary["Loading the association of LS reduced matrix
5163   elements for SOO-ECSO ..."];
5164   SOOandECSOLSTableFname = FileNameJoin[{moduleDir, "data", "}
5165   ReducedSOOandECSOLSTable.m"}];
5166   If[!FileExistsQ[SOOandECSOLSTableFname],
5167     (PrintTemporary[">> ReducedSOOandECSOLSTable.m not found,
5168   generating ..."]);
5169     SOOandECSOLSTable = GenerateSOOandECSOLSTable[7];
5170   ),
5171   SOOandECSOLSTable = Import[SOOandECSOLSTableFname];
5172 ]
5173 );
5174
5175 LoadSOOandECSO::usage = "LoadSOOandECSO[] loads into session the
5176   LSJ reduced matrix elements of spin-other-orbit and
5177   electrostatically-correlated-spin-orbit.";
5178 LoadSOOandECSO[] := (
5179   If[ValueQ[SOOandECSOTableFname], Return[]];
5180   PrintTemporary["Loading the association of matrix elements for
5181   spin-other-orbit and electrostatically-correlated-spin-orbit ..."
5182   ];
5183   SOOandECSOTableFname = FileNameJoin[{moduleDir, "data", "}
5184   SOOandECSOTable.m"}];
5185   If[!FileExistsQ[SOOandECSOTableFname],
5186     (PrintTemporary[">> SOOandECSOTable.m not found, generating ..."
5187   ];
5188     SOOandECSOTable = GenerateSOOandECSOTable[7, "Export" -> True];
5189   ),
5190   SOOandECSOTable = Import[SOOandECSOTableFname];
5191 ]
5192 );
5193
5194 LoadT22::usage = "LoadT22[] loads into session the matrix elements
5195   of T22.";
5196 LoadT22[] := (
5197   If[ValueQ[T22Table], Return[]];
5198   PrintTemporary["Loading the association of reduced T22 matrix
5199   elements ..."];
5200   T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.
5201   m"}];
5202   If[!FileExistsQ[T22TableFname],
5203     (PrintTemporary[">> ReducedT22Table.m not found, generating ..."
5204   ];
5205     T22Table = GenerateT22Table[7];
5206   ),
5207   T22Table = Import[T22TableFname];
5208 ]
5209 );
5210
5211 LoadSpinSpin::usage = "LoadSpinSpin[] loads into session the matrix
5212   elements of spin-spin.";
5213 LoadSpinSpin[] := (
5214   If[ValueQ[SpinSpinTable], Return[]];
5215   PrintTemporary["Loading the association of matrix elements for

```

```

5197   spin-spin ..."];
5198   SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
5199   If[!FileExistsQ[SpinSpinTableFname],
5200     (PrintTemporary[">> SpinSpinTable.m not found, generating ..."]);
5201     SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
5202   ],
5203   SpinSpinTable = Import[SpinSpinTableFname];
5204 ];
5205
5206 LoadThreeBody::usage = "LoadThreeBody[] loads into session the
5207   matrix elements of three-body configuration-interaction effects.";
5208 LoadThreeBody[] := (
5209   If[ValueQ[ThreeBodyTable], Return[]];
5210   PrintTemporary["Loading the association of matrix elements for
5211   three-body configuration-interaction effects ..."];
5212   ThreeBodyFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
5213   ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
5214   If[!FileExistsQ[ThreeBodyFname],
5215     (PrintTemporary[">> ThreeBodyTable.m not found, generating ..."]);
5216     {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTables
5217       [14, "Export" -> True];
5218   ],
5219   ThreeBodyTable = Import[ThreeBodyFname];
5220   ThreeBodyTables = Import[ThreeBodiesFname];
5221 );
5222
5223 (* ##### Load Functions ##### *)
5224 End[];
5225
5226 LoadTermLabels[];
5227 LoadCFP[];
5228
5229 EndPackage[];

```

17.2 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 |~~~~~| This script puts together some code useful for fitting |
19 |~~~~~|           the model Hamiltonian to data. |
20 |~~~~~|
21 |~~~~~|
22 |~~~~~+-----+
23 |~~~~~+-----+
24 |~~~~~+-----+
25 +-----+*)
26
27 Get["qlanth.m"]
28 Get["qonstants.m"];
29 Get["misc.m"];

```

```

30 LoadCarnall[];
31 LoadFreeIon[];
32
33 Jiggle::usage = "Jiggle[num, wiggleRoom] takes a number and
34     randomizes it a little by adding or subtracting a random fraction
35     of itself. The fraction is controlled by wiggleRoom.";
36 Jiggle[num_, wiggleRoom_ : 0.1] := RandomReal[{1 - wiggleRoom, 1 +
37     wiggleRoom}] * num;
38
39 AddToList::usage = "AddToList[list, element, maxSize, addOnlyNew]
40     prepends the element to list and returns the list. If maxSize is
41     reached, the last element is dropped. If addOnlyNew is True (the
42     default), the element is only added if it is different from the
43     last element.";
44 AddToList[list_, element_, maxSize_, addOnlyNew_ : True] := Module[{list,
45     tempList = If[
46         addOnlyNew,
47         If[
48             Length[list] == 0,
49             {element},
50             If[
51                 element != list[[-1]],
52                 Append[list, element],
53                 list
54             ]
55         ],
56         Append[list, element]
57     ],
58     If[Length[tempList] > maxSize,
59         Drop[tempList, Length[tempList] - maxSize],
60         tempList]
61 ];
62
63 ProgressNotebook::usage="ProgressNotebook[] creates a progress
64     notebook for the solver. This notebook includes a plot of the RMS
65     history and the current parameter values. The notebook is returned
66     . The RMS history and the parameter values are updated by setting
67     the variables rmsHistory and paramSols. The variables
68     stringPartialVars and paramSols are used to display the parameter
69     values in the notebook.";
70 Options[ProgressNotebook] = {
71     "Basic" -> True,
72     "UpdateInterval" -> 0.5};
73 ProgressNotebook[OptionsPattern[]] := (
74     nb = Which[
75         OptionValue["Basic"],
76         CreateDocument[(
77             {
78                 Dynamic[
79                     TextCell[
80                         If[
81                             Length[paramSols] > 0,
82                             TableForm[
83                                 Prepend[
84                                     Transpose[{stringPartialVars,
85                                         paramSols[[-1]]}],
86                                     {"RMS", rmsHistory[[-1]]}]
87                                 ],
88                                 " "
89                             ],
90                             "Output"
91                         ],
92                         TrackedSymbols :> {paramSols, stringPartialVars},
93                         UpdateInterval -> OptionValue["UpdateInterval"]
94                     ]
95                 }
96             ),
97             WindowSize -> {600, 1000},
98             WindowSelected -> True,
99             TextAlignment -> Center,
100             WindowTitle -> "Solver Progress"
101         ],
102         True,
103         CreateDocument[(
104             {
105                 " "
106             }
107         )
108     ]
109 );
110
111 
```

```

93     Dynamic[Framed[progressMessage],
94       UpdateInterval -> OptionValue["UpdateInterval"]], 
95     Dynamic[
96       GraphicsColumn[
97         {ListPlot[rmsHistory,
98           PlotMarkers -> "OpenMarkers",
99             Frame -> True,
100            FrameLabel -> {"Iteration", "RMS"}, 
101            ImageSize -> 800,
102            AspectRatio -> 1/3,
103            FrameStyle -> Directive[Thick, 15],
104            PlotLabel -> If[Length[rmsHistory] != 0, rmsHistory[[-1]], 
105              ""]
106            ],
107            ListPlot[(#/#[[1]]) & /@ Transpose[paramSols],
108              Joined -> True,
109              PlotRange -> {All, {-5, 5}},
110              Frame -> True,
111              ImageSize -> 800,
112              AspectRatio -> 1,
113              FrameStyle -> Directive[Thick, 15],
114              FrameLabel -> {"Iteration", "Params"}]
115            ]
116          ],
117          TrackedSymbols :> {rmsHistory, paramSols},
118          UpdateInterval -> OptionValue["UpdateInterval"]
119        ],
120        Dynamic[
121          TextCell[
122            If[
123              Length[paramSols] > 0,
124              TableForm[Transpose[{stringPartialVars, paramSols[[-1]]}]],
125              ""
126            ],
127            "Output"
128          ],
129          TrackedSymbols :> {paramSols, stringPartialVars}
130        ]
131      }
132    ),
133    WindowSize -> {600, 1000},
134    WindowSelected -> True,
135    TextAlignment -> Center,
136    WindowTitle -> "Solver Progress"
137  ]
138];
139 Return[nb];
140 );
141
142 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'";
143 energyCostFunTemplate = StringTemplate[
144 TheRightEnergyPath['varPatterns']:= (
145   {eigenEnergies, eigenVecs} = Eigensystem[compHam['vars']];
146   ordering = Ordering[eigenEnergies];
147   eigenEnergies = eigenEnergies - Min[eigenEnergies];
148   states = Transpose[{eigenEnergies, eigenVecs}];
149   states = states[[ordering]];
150   coarseStates = ParseStates[states, basis];
151   coarseStates = {#[[1]],#[[-1]]}& /@ coarseStates;
152   (* The eigenvectors need to be simplified in order to compare
      labels to labels *)
153   missingLevels = Length[coarseStates]-Length[expData];
154   (* The energies are in the first element of the tuples. *)
155   energyDiffFun = (Abs[#1[[1]]-#2[[1]]])&;
156   (* match disregarding labels *)
157   energyFlow = FlowMatching[coarseStates,
158     expData,
159     \"notMatched\" -> missingLevels,
160     \"CostFun\" -> energyDiffFun

```

```

161 ];
162 energyPairs = {#[[1]][[1]], #[[2]][[1]]} & /@ energyFlow[[1]];
163 energyRms = Sqrt[Total[(Abs[#[[2]] - #[[1]])^2 & /@ energyPairs]
164 / Length[energyPairs]]];
165 Return[energyRms];
166 )];
167
168 AppendToLog[message_, file_String] := Module[
169 {timestamp = DateString["ISODateTime"], msgString},
170 (
171 msgString = ToString[message, InputForm]; (* Convert any
172 expression to a string *)
173 OpenAppend[file];
174 WriteString[file, timestamp, " - ", msgString, "\n"];
175 Close[file];
176 )
177 ];
178
179 energyAndLabelCostFunTemplate::usage="energyAndLabelCostFunTemplate
180 is a template used to define the cost function that includes both
181 the differences between energies and the differences between
182 labels. The template is used to define a function
183 TheRightSignedPath that takes a list of variables and returns the
184 RMS of the energy differences between the computed and the
185 experimental energies together with a term that depends on the
186 differences between the labels. The template requires the values
187 to the following keys to be provided: 'vars' and 'varPatterns';
188 energyAndLabelCostFunTemplate = StringTemplate["
189 TheRightSignedPath['varPatterns'] := Module[
190 {energyRms, eigenEnergies, eigenVecs, ordering, states,
191 coarseStates, missingLevels, energyDiffFun, energyFlow,
192 energyPairs, energyAndLabelFun, energyAndLabelFlow, totalAvgCost},
193 (
194 {eigenEnergies, eigenVecs} = Eigensystem[compHam['vars']];
195 ordering = Ordering[eigenEnergies];
196 eigenEnergies = eigenEnergies - Min[eigenEnergies];
197 states = Transpose[{eigenEnergies, eigenVecs}];
198 states = states[[ordering]];
199 coarseStates = ParseStates[states, basis];
200
201 (* The eigenvectors need to be simplified in order to compare
202 labels to labels *)
203 coarseStates = {#[[1]], #[[-1]]} & /@ coarseStates;
204 missingLevels = Length[coarseStates] - Length[expData];
205
206 (* The energies are in the first element of the tuples. *)
207 energyDiffFun = ( Abs[#[1][[1]] - #2[[1]]] ) &;
208
209 (* matching disregarding labels to get overall scale for scaling
210 differences in labels *)
211 energyFlow = FlowMatching[coarseStates,
212 expData,
213 \ "notMatched\" -> missingLevels,
214 \ "CostFun\" -> energyDiffFun
215 ];
216 energyPairs = {#[[1]][[1]], #[[2]][[1]]} &/@energyFlow[[1]];
217 energyRms = Sqrt[Total[(Abs[#[[2]] - #[[1]])^2 & /@
218 energyPairs]/Length[energyPairs]];
219
220 (* matching using both labels and energies *)
221 energyAndLabelFun = With[{del=energyRms},
222 (Abs[#[1][[1]] - #2[[1]]] +
223 If[#[1][[2]] == #2[[2]],
224 0.,
225 del]) &];
226
227 (* energyAndLabelFun = With[{del=energyRms},
228 (Abs[#[1][[1]] - #2[[1]]] +
229 del*EditDistance[#[1][[2]], #2[[2]]]) &]; *)
230 energyAndLabelFun = ( Abs[#[1][[1]] - #2[[1]]] + EditDistance
231 #[1][[2]], #2[[2]] ) &;
232 energyAndLabelFlow = FlowMatching[coarseStates,
233 expData,
234 \ "notMatched\" -> missingLevels,
235 \ "CostFun\" -> energyAndLabelFun
236 ];

```

```

221 totalAvgCost      = Total[energyAndLabelFun@@# & /@  

222 energyAndLabelFlow[[1]]]/Length[energyAndLabelFlow[[1]]];  

223 Return[totalAvgCost];  

224 )  

225 ]";  

226 truncatedEnergyCostTemplate = StringTemplate["  

227 TheTruncatedAndSignedPath['varsWithNumericQ'] :=  

228 (  

229 (* Calculate the truncated Hamiltonian *)  

230 numericalFreeIonHam = compileIntermediateTruncatedHam['  

231 varsMixedWithFixedVals'];  

232 (* Diagonalize it *)  

233 {truncatedEigenvalues, truncatedEigenVectors} = Eigensystem[  

234 numericalFreeIonHam];  

235 (* Using the truncated eigenvectors push them up to the full state  

space *)  

236 pulledTruncatedEigenVectors = truncatedEigenVectors.Transpose[  

237 truncatedIntermediateBasis];  

238 states = Transpose[{truncatedEigenvalues,  

239 pulledTruncatedEigenVectors}];  

240 states = SortBy[states, First];  

241 states = ShiftedLevels[states];  

242 (* Coarsen the resulting eigenstates *)  

243 coarseStates = ParseStates[states, basis];  

244 (* Grab the parts that are needed for fitting *)  

245 coarseStates = {#[[1]], #[[-1]]} & /@ coarseStates;  

246 (* This cost function takes into account both labels and energies a  

random factor is added for the sake of stability of the solver*)  

247 energyAndLabelFun = (  

248 (
249     Abs[#1[[1]] - #2[[1]]] +  

250     EditDistance[#1[[2]], #2[[2]]]  

251 ) *  

252 (1 + RandomReal[{-10^-6, 10^-6}]) &;  

253 (* This one only takes into account the energies *)  

254 energyFun = (Abs[#1[[1]] - #2[[1]]]*(1 + RandomReal[{0, 10^-6}])) &;  

255 ;  

256 (* Choose which cost function to use *)  

257 costFun = energyAndLabelFun;  

258 (* Not all states are to be matched to the experimental data *)  

259 missingLevels = Length[coarseStates] - Length[expData];  

260 (* If there are more experimental data than calculated ones, don't  

leave any state unmatched to those*)  

261 missingLevels = If[missingLevels < 0, 0, missingLevels];  

262 (* Apply the Hungarian algorithm to match the two sets of data *)  

263 energyAndLabelFlow = FlowMatching[coarseStates,  

264 expData,  

265 \\"notMatched\\" -> missingLevels,  

266 \\"CostFun\\" -> costFun];  

267 totalCosts = (costFun @@ #)& /@ energyAndLabelFlow[[1]];  

268 totalAvgCost = Total[totalCosts] / Length[energyAndLabelFlow[[1]]];  

269 Return[totalAvgCost]  

270 )"  

271 ];
272
273 Constrained::usage = "Constrained[problemVars, ln] returns a list of  

274 constraints for the variables in problemVars for trivalent  

275 lanthanide ion ln. problemVars are standard model symbols (F2, F4,  

276 ...). The ranges returned are based in the fitted parameters for  

277 LaF3 as found in Carnall et al. They could probably be more fine  

278 grained, but these ranges are seen to describe all the ions in  

279 that case.";  

280 Constrained[problemVars_, ln_] := (  

281 slater = Which[  

282 MemberQ[{\"Ce\", \"Yb\"}, ln],  

283

```

```

282  },
283  True,
284  {#, (20000. < # < 120000.)} & /@ {F2, F4, F6}
285  ];
286 alpha = Which[
287   MemberQ[{"Ce", "Yb"}, ln],
288   {} ,
289   True,
290   {{ $\alpha$ , 14. <  $\alpha$  < 22.}}
291   ];
292 zeta = {{ $\zeta$ , 500. <  $\zeta$  < 3200.}};
293 beta = Which[
294   MemberQ[{"Ce", "Yb"}, ln],
295   {} ,
296   True,
297   {{ $\beta$ , -1000. <  $\beta$  < -400.}}
298   ];
299 gamma = Which[
300   MemberQ[{"Ce", "Yb"}, ln],
301   {} ,
302   True,
303   {{ $\gamma$ , 1000. <  $\gamma$  < 2000.}}
304   ];
305 tees = Which[
306   ln == "Tm",
307   {100. < T2 < 500.},
308   MemberQ[{"Ce", "Pr", "Yb"}, ln],
309   {} ,
310   True,
311   {#, -500. < # < 500.} & /@ {T2, T3, T4, T6, T7, T8}];
312 marvins = Which[
313   MemberQ[{"Ce", "Yb"}, ln],
314   {} ,
315   True,
316   {{M0, 1.0 < M0 < 5.0}}
317   ];
318 peas = Which[
319   MemberQ[{"Ce", "Yb"}, ln],
320   {} ,
321   True,
322   {{P2, -200. < P2 < 1200.}}
323   ];
324 crystalRanges = {#, (-2000. < # < 2000.)} & /@ (Intersection[
325   cfSymbols, problemVars]);
326 allCons =
327   Join[slater, zeta, alpha, beta, gamma, tees, marvins, peas,
328   crystalRanges];
329 allCons = Select[allCons, MemberQ[problemVars, #[[1]]] &];
330 Return[Flatten[Rest /@ allCons]]
331 )
332
333 Options[LogSol] = {"PrintFun" -> PrintTemporary};
334 LogSol::usage = "LogSol[expr, solHistory, 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.";
335 LogSol[theSolution_, prefix_, OptionsPattern[]] := (
336   PrintFun = OptionValue["PrintFun"];
337   fname = prefix <> "-sols-" <> CreateUUID[] <> ".m";
338   fname = FileNameJoin[{".", "log", fname}];
339   PrintFun["Saving solution to: ", fname];
340   Export[fname, theSolution];
341   Return[fname];
342 );
343
344
345 FitToHam::usage = "FitToHam[numE, expData, fitToSymbols, simplifier,
   OptionsPattern[]} fits the model Hamiltonian to the experimental
   data for the trivalent lanthanide ion with number numE. The
   experimental data is given in the form of a list of tuples. The
   first element of the tuple is the energy and the second element is
   the label. The function saves the results to a file, with the
   string filePrefix prepended to it, by default this is an empty
   string, in which case the filePrefix is modified to be the name of
   the lanthanide.

```

```

346 The fitToSymbols is a list of the symbols to be fit. The simplifier
347     is a list of rules that simplify the Hamiltonian.
348 The options and their defaults are:
349 \\"PrintFun\\\"->PrintTemporary,
350 \\"FilePrefix\\\"->\"\",
351 \\"SlackChannel\\\"->None,
352 \\"MaxHistory\\\"->100,
353 \\"MaxIters\\\"->100,
354 \\"NumCycles\\\"->10,
355 \\"ProgressWindow\\\"->True
356 The PrintFun option is the function used to print progress messages.
357 The FilePrefix option is the prefix to use for the file name, by
358     default this is the symbol for the lanthanide.
359 The SlackChannel option is the channel to post progress messages to.
360 The MaxHistory option is the maximum number of iterations to keep in
361     the history.
362 The MaxIters option is the maximum number of iterations for the
363     solver.
364 The NumCycles option is the number of cycles to run the solver for.
365 The function returns a list of solutions. The solutions are the
366     results of the NMinimize function. The solutions are a list of
367     tuples. The first element of the tuple is the RMS error and the
368     second element is the parameter values
369 The function also saves the solutions to a file. The file is named
370     with a prefix and a UUID. The file is saved in the current
371     directory. The file is saved in the format of a .m file.";
372 Options[FitToHam] = {
373     "PrintFun" -> PrintTemporary,
374     "FilePrefix" -> "",
375     "SlackChannel" -> None,
376     "MaxHistory" -> 100,
377     "ProgressWindow" -> True,
378     "MaxIters" -> 100,
379     "NumCycles" -> 10};
380 FitToHam[numE_Integer, expData_List, fitToSymbols_List,
381     simplifier_List, OptionsPattern[]] :=
382 (
383     PrintFun = OptionValue["PrintFun"];
384     fitToVars = ToExpression[ToString[#] <> "v"] & /@ fitToSymbols;
385     stringfitToVars = ToString /@ fitToVars;
386     slackChan = OptionValue["SlackChannel"];
387     maxHistory = OptionValue["MaxHistory"];
388     maxIters = OptionValue["MaxIters"];
389     numCycles = OptionValue["NumCycles"];
390     ln = theLanthanides[[numE]];
391     logFilePrefix = If[OptionValue["FilePrefix"] == "", ToString[theLanthanides[[numE]]], OptionValue["FilePrefix"]];
392     PrintFun["Assembling the Hamiltonian for f^", numE, "..."];
393     ham = HamMatrixAssembly[numE];
394     PrintFun["Simplifying the symbolic expression for the Hamiltonian
395         in terms of the given simplifier..."];
396     ham = ReplaceInSparseArray[ham, simplifier];
397     PrintFun["Determining the variables to be fit for ..."];
398     (* as they remain after simplifying *)
399     fitVars = Variables[Normal[ham]];
400     (* append v to symbols *)
401     varVars = ToExpression[ToString[#] <> "v"] & /@ fitVars;
402
403     PrintFun[
404         "Compiling a function for efficient evaluation of the Hamiltonian
405             matrix ..."];
406     compHam = Compile[Evaluate[fitVars], Evaluate[N[Normal[ham]]]];
407
408     PrintFun[
409         "Defining the cost function according to given energies and state
410             labels ..."];
411
412     varPatterns = StringJoin[{ToString[#], "_?NumericQ"}] & /@ fitVars;
413     varPatterns = Riffle[varPatterns, ", "];
414     varPatterns = StringJoin[varPatterns];
415     vars = ToString[#] & /@ fitVars;
416     vars = Riffle[vars, ", "];
417     vars = StringJoin[vars];

```

```

407 basis = BasisLSJMJ[numE];
408
409 (* define the cost functions given the problem variables *)
410 energyCostFunString =
411 energyCostFunTemplate[<|
412 "varPatterns" -> varPatterns,
413 "vars" -> vars|>];
414 ToExpression[energyCostFunString];
415 energyAndLabelCostFunString = energyAndLabelCostFunTemplate[<|
416 "varPatterns" -> varPatterns, "vars" -> vars|>];
417 ToExpression[energyAndLabelCostFunString];
418
419 PrintFun["getting starting values from LaF3..."];
420 lnParams = LoadLaF3Parameters[ln];
421 bills = Table[lnParams[ToExpression[StringTake[ToString[varvar], {1, -2}]]], {varvar, varVars}];
422
423 (* define the function arguments with the frozen args in place *)
424 activeArgs = Table[
425 If[MemberQ[fitToVars, varvar],
426 varvar,
427 lnParams[ToExpression[StringTake[ToString[varvar], {1, -2}]]]],
428 {varvar, varVars}
429 ];
430 activeArgs = StringJoin[Riffle[ToString /@ activeArgs, ", "]];
431 (* the constraints, very important *)
432 constraints = N[Constrainer[fitToVars, ln]];
433 complementaryArgs = Table[
434 If[MemberQ[fitToVars, varvar],
435 varvar,
436 lnParams[ToExpression[StringTake[ToString[varvar], {1, -2}]]]],
437 {varvar, varVars}
438 ];
439
440 fromBill = {B02v -> B02, B04v -> B04, B06v -> B06, B22v -> B22,
441 B24v -> B24, B26v -> B26, B44v -> B44, B46v -> B46, B66v -> B66,
442 M0v -> M0, P2v -> P2} /. lnParams;
443
444 If[Not[ValueQ[noteboo]] && OptionValue["ProgressWindow"],
445 noteboo = ProgressNotebook["Basic" -> False];
446 ];
447
448 threadHeaderTemplate = StringTemplate[
449 "(`idx`/`reps`) Fitting data for `ln` using `freeVars`."
450 ];
451 solutions = {};
452 Do[
453 (
454 (* Remove the downvalues of the cost function *)
455 (* DownValues[TheRightSignedPath] = {DownValues[
456 TheRightSignedPath][[-1]]}; *)
457 rmsHistory = {};
458 paramSols = {};
459 startTime = Now;
460 threadMessage = threadHeaderTemplate[
461 <|"reps" -> numCycles,
462 "idx" -> rep,
463 "ln" -> ln,
464 "freeVars" -> ToString[fitToVars]|>];
465 If[slackChan != None,
466 threadTS = PostMessageToSlack[threadMessage, slackChan]["ts"]
467 ];
468 solverTemplateNMini = StringTemplate[""
469 numIter = 0;
470 sol = NMinimize[
471 Evaluate[
472 Join[{TheRightSignedPath['activeArgs']},
473 constraints
474 ]
475 ],
476 fitToVars,
477 MaxIterations -> 'maxIterations',
478 Method -> 'Method',
479 'Monitor' :>(

```

```

480         currentErr = TheRightSignedPath[‘activeArgs’];
481         numIter += 1;
482         rmsHistory = AddToList[rmsHistory, currentErr, maxHistory
483 , False];
484         paramSols = AddToList[paramSols, fitToVars, maxHistory,
485 False];
486     )
487   ];
488   solverCode = solverTemplateNMini[<|
489     "maxIterations" -> maxIters,
490     "Method" -> "{\"DifferentialEvolution\",
491       \"PostProcess\" -> False,
492       \"ScalingFactor\" -> 0.9,
493       \"RandomSeed\" -> RandomInteger[{0,1000000}],
494       \"SearchPoints\" -> 10},
495     "Monitor" -> "StepMonitor",
496     "activeArgs" -> activeArgs|>];
497   ToExpression[solverCode];
498   timeTaken = QuantityMagnitude[Now - startTime, "Seconds"];
499   Print["Took " <> ToString[timeTaken] <> "s"];
500   Print[sol];
501   {bestError, bestParams} = sol;
502   resultMessage = "sigma=" <> ToString[Round[bestError, 0.1]];
503   logFname = LogSol[sol, logFilePrefix];
504   If[slackChan != None,
505   (
506     PostMessageToSlack[resultMessage, slackChan, "threadTS" ->
507 threadTS];
508     PostFileToSlack[logFname, logFname, slackChan, "threadTS" ->
509 threadTS];
510   )
511 ];
512
513 vsBill = TableForm[
514   Transpose[{{
515     First /@ fromBill,
516     Last /@ fromBill,
517     Round[Last /@ bestParams, 1.]}},
518   TableHeadings -> {None, {"Param", "Bill Bkq", "ql Bkq"}}
519 ];
520
521 (* analysis code *)
522
523 finalHam = compHam @@ (complementaryArgs /. bestParams);
524 {eigenEnergies, eigenVecs} = Eigensystem[finalHam];
525 ordering = Ordering[eigenEnergies];
526 eigenEnergies = eigenEnergies - Min[eigenEnergies];
527 states = Transpose[{eigenEnergies, eigenVecs}];
528 states = states[[ordering]];
529 coarseStates = ParseStates[states, basis];
530
531 (* The eigenvectors need to be simplified in order to compare
532 labels to labels *)
533 coarseStates = {#[[1]], #[[-1]]} & /@ coarseStates;
534 missingLevels = Length[coarseStates] - Length[expData];
535 (* The energies are in the first element of the tuples. *)
536 energyDiffFun = (Abs[#1[[1]] - #2[[1]]]) &;
537 (* matching disregarding labels to get overall scale for
538 scaling differences in labels *)
539 energyFlow = FlowMatching[coarseStates,
540   expData,
541   "notMatched" -> missingLevels,
542   "CostFun" -> energyDiffFun];
543 energyPairs = {#[[1]][[1]], #[[2]][[1]]} & /@ energyFlow[[1]];
544 energyRms = Sqrt[Total[(Abs[#1[[2]] - #1[[1]]])^2 & /@
545 energyPairs] / Length[energyPairs]];
546 (* matching using both labels and energies *)
547 energyAndLabelFun = (Abs[#1[[1]] - #2[[1]]] + EditDistance
548 #[[1][[2]], #2[[2]]]) &;
549 energyAndLabelFlow = FlowMatching[coarseStates,
550   expData,

```

```

547         "notMatched" -> (Length[coarseStates] - Length[expData]),
548         "CostFun" -> energyAndLabelFun];
549     totalAvgCost = Total[energyAndLabelFun @@ # & /@ 
energyAndLabelFlow[[1]]] / Length[energyAndLabelFlow[[1]]];
550
551     compa = (Flatten /@ energyAndLabelFlow[[1]]);
552     compa = Join[
553       #,
554       {
555         #[[2]] == #[[4]],
556         If[NumberQ #[[1]],
557           Round[#[[1]] - #[[3]], 1],
558           ""
559         ],
560         #[[5]] - #[[3]],
561         Which[
562           Round[Abs[#[[1]] - #[[3]]]] < Round[Abs[#[[5]] - 
#[[3]]]],
563             "Better",
564             Round[Abs[#[[1]] - #[[3]]]] == Round[Abs[#[[5]] - 
#[[3]]]],
565             "Equal",
566             True,
567             "Worse"
568         ]
569       }
570     ] & /@ compa;
571     atable = TableForm[compa,
572       TableHeadings -> {None,
573         {"ql", "ql", "Bill (exp)", "Bill (exp)",
574          "Bill (calc)", "labels=", "ql - exp", "bill - exp"}}
575     ];
576     atable = Framed[atable, FrameMargins -> 20];
577     upsAndDowns = {
578       {"Better", Length[Select[compa, #[[-1]] == "Better" &}}},
579       {"Equal", Length[Select[compa, #[[-1]] == "Equal" &}}},
580       {"Worse", Length[Select[compa, #[[-1]] == "Worse" &]}}
581     };
582     upsAndDowns = TableForm[upsAndDowns];
583     If[slackChan != None,
584       PostPdfToSlack["table", atable, slackChan, "threadTS" ->
threadTS];
585     ];
586     solutions = Append[solutions, sol];
587   ),
588   {rep, 1, numCycles}
589 ];
590 )
591
592 TruncationFit::usage="TruncationFit[numE, expData, numReps,
activeVars, startingValues, Options] fits the given expData in an
f^numE configuration, generating numReps different solutions, and
varying the symbols in activeVars. The list startingValues is a
list with all of the parameters needed to define the Hamiltonian (
including values for activeVars, which will be disregarded but are
required as position placeholders). The function returns a list
of solutions. The solutions are the results of the NMinimize
function using the Differential Evolution method. The solutions
are a list of tuples. The first element of the tuple is the RMS
error and the second element is a list of replacement rules for
the fitted parameters. Once each NMinimize is done, the function
saves the solutions to a file. The file is named with a prefix and
a UUID. The file is saved in the log sub-directory as a .m file.
The solver is always constrained by the relevant subsets of
constraints for the parameters as provided by the Constrainer
function. By default the Differential Evolution method starts with
a generation of points within the given constraints, however it
is also possible here to have a different region from which the
initial points are chosen with the option \"StartingForVars\".
593
594 The following options can be used:
595  \"SignatureCheck\" : if True then then the function ends
prematurely, printing a list with the symbols that would have
defined the Hamiltonian after all simplifications have been
applied. Useful to check the entire parameter set that the
Hamiltonian has, which has to match one-to-one what is provided by

```

```

      startingValues.

596  \\"FilePrefix\\": the prefix to use for the file name, by default
      this is the symbol for the lanthanide.
597  \\"AccuracyGoal\\": sets the accuracy goal for NMinimize, the default
      is 3.
598  \\"MaxHistory\\": determines how long the logs for the solver can be
      .
599  \\"MaxIterations\\": determines the maximum number of iterations used
      by NMinimize.
600
601  \\"AccuracyGoal\\": the accuracy goal used by NMinimize, default of
      3.
602  \\"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 truncated to this value.
603  \\"PrintFun\\": the function used to print progress messages, the
      default is PrintTemporary.
604  \\"SlackChannel\\": name of the Slack channel to which to dump
      progress messaages, the default is None which disables this option
      entirely.
605  \\"ProgressView\\": whether or not a progress window will be opened
      to show the progress of the solver, the default is True.
606
607  \\"ReturnHashFileNameAndExit\\": if True then the function returns
      the name of the file with the solutions and exits, the default is
      False.
608  \\"StartingForVars\\": if different from {} then it has to be a list
      with two elements. The first element being a number that
      determines the fraction half-width of the interval used for
      choosing the initial generation of points. The second element
      being a list with as many elements as activeVars corresponding to
      the midpoints from which the intial generation points are chosen.
      The default is {}.
609  \\"DE:CrossProbability\\": the cross probability used by the
      Differential Evolution method, the default is 0.5.
610  \\"DE:ScalingFactor\\": the scaling factor used by the Differential
      Evolution method, the default is 0.6.
611  \\"DE:SearchPoints\\": the number of search points used by the
      Differential Evolution method, the default is Automatic.
612
613  \\"MagneticSimplifier\\": a list of replacement rules to simplify the
      Marvin and pesudo-magnetic paramters.
614  \\"MagFieldSimplifier\\": a list of replacement rules to specify a
      magnetic field (in T), if set to {}, then {Bx, By, Bz} can also
      then be used as variables to be fitted for.
615  \\"SymmetrySimplifier\\": a list of replacements rules to simplify
      the crystal field.
616  \\"OtherSimplifier\\": an additiona list of replacement rules that
      are applied to the Hamiltonian before computing with it.
617  \\"ThreeBodySimplifier\\": the default is an Association that simply
      states which three body parameters Tk are zero in different
      configurations, if a list of replacement rules is used then that
      is used instead for the given problem.
618
619  \\"FreeIonSymbols\\": a list with the symbols to be included in the
      intermediate coupling basis.
620  \\"AppendToLogFile\\": an association appended to the log file under
      the key \\"Appendix\\".
621  ";
622 Options[TruncationFit]={
623   "MaxHistory"        -> 200,
624   "MaxIterations"     -> 100,
625   "FilePrefix"        -> "",
626   "AccuracyGoal"      -> 3,
627   "TruncationEnergy" -> Automatic ,
628   "PrintFun"          -> PrintTemporary ,
629   "SlackChannel"      -> None ,
630   "ProgressView"      -> True ,
631   "SignatureCheck"    -> False ,
632   "AppendToLogFile"   -> <||>,
633   "StartingForVars"   -> {},
634   "ReturnHashFileNameAndExit" -> False ,
635   "DE:CrossProbability" -> 0.5,
636   "DE:ScalingFactor"    -> 0.6,
637   "DE:SearchPoints"     -> Automatic ,
638   "MagneticSimplifier" -> {

```

```

639      M2 -> 56/100 M0,
640      M4 -> 31/100 M0,
641      P4 -> 1/2 P2,
642      P6 -> 1/10 P2},
643 "MagFieldSimplifier" -> {
644     Bx->0,By->0,Bz->0
645 },
646 "SymmetrySimplifier" -> {
647     B12->0,B14->0,B16->0,B34->0,B36->0,B56->0,
648     S12->0,S14->0,S16->0,S22->0,S24->0,S26->0,S34->0,S36->0,
649     S44->0,S46->0,S56->0,S66->0
650 },
651 "OtherSimplifier" -> {
652     F0->0,
653     P0->0,
654     \[\Sigma\] SS->0,
655     T11p->0,T12->0,T14->0,T15->0,
656     T16->0,T18->0,T17->0,T19->0,T2p->0
657 },
658 "ThreeBodySimplifier" -> <|
659     1 -> {
660         T2->0,T3->0,T4->0,T6->0,T7->0,T8->0,T11p->0,T12->0,T14->0,T15
661         ->0,T16->0,T18->0,T17->0,T19->0,T2p->0},2->\{T2->0,T3->0,T4->0,T6
662         ->0,T7->0,T8->0,T11p->0,T12->0,T14->0,T15->0,T16->0,T18->0,T17->0,
663         T19->0,T2p->0
664     },
665     3 -> {},
666     4 -> {},
667     5 -> {},
668     6 -> {},
669     7 -> {},
670     8 -> {},
671     9 -> {},
672     10 -> {},
673     11 -> {},
674     12 -> {
675         T3->0,T4->0,T6->0,T7->0,T8->0,T11p->0,T12->0,T14->0,T15->0,T16
676         ->0,T18->0,T17->0,T19->0,T2p->0
677     },
678     13->{
679         T2->0,T3->0,T4->0,T6->0,T7->0,T8->0,T11p->0,T12->0,T14->0,T15
680         ->0,T16->0,T18->0,T17->0,T19->0,T2p->0
681     }
682     |>,
683 "FreeIonSymbols" -> {F0, F2, F4, F6, M0, P2,  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\zeta$ , T2, T3, T4,
684     T6, T7, T8}
685 };
686 TruncationFit[numE_Integer, expData0_List, numReps_Integer,
687     activeVars_List, startingValues_List, OptionsPattern[]]:=(
688     ln = theLanthanides[[numE]];
689     expData = expData0;
690     PrintFun = OptionValue["PrintFun"];
691     truncationEnergy = If[OptionValue["TruncationEnergy"]==Automatic,
692         Max[First/@expData],
693         OptionValue["TruncationEnergy"]
694     ];
695     oddsAndEnds = <||>;
696     expData = Select[expData, #[[1]] <= truncationEnergy &];
697     maxIterations = OptionValue["MaxIterations"];
698     maxHistory = OptionValue["MaxHistory"];
699     slackChan = OptionValue["SlackChannel"];
700     accuracyGoal = OptionValue["AccuracyGoal"];
701     logFilePrefix = If[OptionValue["FilePrefix"] == "",
702         ToString[theLanthanides[[numE]]],
703         OptionValue["FilePrefix"]];
704
705     usingInitialRange = Not[OptionValue["StartingForVars"] === {}];
706     If[usingInitialRange,
707     (
708         PrintFun["Using the solver for initial values in range ..."];
709         {fractionalWidth, startVarValues} = OptionValue[
710             "StartingForVars"];
711     )
712 ];
713
714 magneticSimplifier = OptionValue["MagneticSimplifier"];

```

```

707 magFieldSimplifier = OptionValue["MagFieldSimplifier"];
708 symmetrySimplifier = OptionValue["SymmetrySimplifier"];
709 otherSimplifier = OptionValue["OtherSimplifier"];
710 threeBodySimplifier = If[Head[OptionValue["ThreeBodySimplifier"]]]
711     == Association,
712     OptionValue["ThreeBodySimplifier"][numE],
713     OptionValue["ThreeBodySimplifier"]
714 ];
714 simplifier = Join[magneticSimplifier,
715                     magFieldSimplifier,
716                     symmetrySimplifier,
717                     threeBodySimplifier,
718                     otherSimplifier];
719 freeIonSymbols = OptionValue["FreeIonSymbols"];
720 runningInteractive = (Head[$ParentLink] === LinkObject);
721
722 oddsAndEnds["simplifier"] = simplifier;
723 oddsAndEnds["freeIonSymbols"] = freeIonSymbols;
724 oddsAndEnds["truncationEnergy"] = truncationEnergy;
725 oddsAndEnds["numE"] = numE;
726 oddsAndEnds["expData"] = expData;
727 oddsAndEnds["numReps"] = numReps;
728 oddsAndEnds["activeVars"] = activeVars;
729 oddsAndEnds["startingValues"] = startingValues;
730 oddsAndEnds["maxIterations"] = maxIterations;
731 oddsAndEnds["PrintFun"] = PrintFun;
732 oddsAndEnds["ln"] = ln;
733 oddsAndEnds["numE"] = numE;
734 oddsAndEnds["accuracyGoal"] = accuracyGoal;
735 oddsAndEnds["Appendix"] = OptionValue["AppendToFile"];
736
737 hamDim = Binomial[14, numE];
738 modelSymbols = Sort[Select[paramSymbols, Not[MemberQ[Join[
739     racahSymbols, chenSymbols, {t2Switch, \[Epsilon], gs}], #]]&]];
739 (* Remove the symbols that will be removed by the simplifier, no
739    symbol should remain here that is not in the symbolic hamiltonian
739 *)
740 reducedModelSymbols = Select[modelSymbols, Not[MemberQ[Keys[
741     simplifier], #]]&];
741 If[OptionValue["SignatureCheck"],
742     (
743         PrintFun["Given the model parameters and the simplifying
743 assumptions, the resultant model parameters are:"];
744         PrintFun[{reducedModelSymbols}];
745         PrintFun["The ordering in these needs to be respected in the
745 startValues parameter ..."];
746         PrintFun["Exiting ..."];
747         Return[];
748     )
749 ];
750
751 (*calculate the basis*)
752 basis = BasisLSJMJ[numE];
753 (* grab the Hamiltonian preserving its block structure *)
754 PrintFun["Assembling the Hamiltonian for f^", numE, " keeping the
754    block structure ..."];
755 ham = HamMatrixAssembly[numE, "ReturnInBlocks" -> True];
756 (* apply the simplifier *)
757 PrintFun["Simplifying using the given aggregate set of
757    simplification rules ..."];
758 ham = Map[ReplaceInSparseArray[#, simplifier]&, ham, {2}];
759
760 (* Get the reference parameters from LaF3 *)
761 PrintFun["Getting reference parameters for ", ln, " using LaF3 ..."];
762 lnParams = LoadLaF3Parameters[ln];
763 freeBies = Prepend[Values[(# -> (#/.lnParams))&/@freeIonSymbols], numE
763 ];
764 (* a more explicit alias *)
765 allVars = reducedModelSymbols;
766
767 oddsAndEnds["allVars"] = allVars;
768 oddsAndEnds["freeBies"] = freeBies;
769
770 (* reload compiled version if found *)
771 varHash = Hash[{numE, allVars, freeBies,
771     truncationEnergy}];
```

```

772 compileIntermediateFname = "compileIntermediateTruncatedHam-<>
773   ToString[varHash]<>".mx";
774 truncatedFname           = "TheTruncatedAndSignedPath-<>ToString[
775   varHash]<>".mx";
776 If[OptionValue["ReturnHashFileNameAndExit"],
777   (
778     Print[varHash];
779     Return[truncatedFname];
780   )
781 ];
782 If[FileExistsQ[compileIntermediateFname],
783   PrintFun["This ion and free-ion params have been compiled before
784   (as determined by {numE, allVars, freeBies, truncationEnergy}).
785   Loading the previously saved function and intermediate coupling
786   basis ..."];
787   {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =
788   Import[compileIntermediateFname];
789   (
790     PrintFun["Zeroing out every symbol in the Hamiltonian that is not
791     a free-ion parameter ..."];
792     (* Get the free ion symbols *)
793     freeIonSimplifier = (#->0) & /@ Complement[reducedModelSymbols,
794     freeIonSymbols];
795     (* Take the diagonal blocks for the intermediate analysis *)
796     PrintFun["Grabbing the diagonal blocks of the Hamiltonian ..."];
797     diagonalBlocks      = Diagonal[ham];
798     (* simplify them to only keep the free ion symbols *)
799     PrintFun["Simplifying the diagonal blocks to only keep the free
800     ion symbols ..."];
801     diagonalScalarBlocks = ReplaceInSparseArray[#,freeIonSimplifier
802     ]&/@diagonalBlocks;
803     (* these include the MJ quantum numbers, remove that *)
804     PrintFun["Contracting the basis vectors by removing the MJ
805     quantum numbers from the diagonal blocks ..."];
806     diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
807
808     argsOfTheIntermediateEigensystems          = StringJoin[Riffle[
809       Prepend[(ToString[#]<>"v_") & /@ freeIonSymbols,"numE_"],", ", "]];
810     argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle[((
811       ToString[#]<>"v") & /@ freeIonSymbols,", ", "]];
812     PrintFun["argsOfTheIntermediateEigensystems = ",
813     argsOfTheIntermediateEigensystems];
814     PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
815     argsForEvalInsideOfTheIntermediateSystems];
816     PrintFun["If the following fails, make sure to modify the
817     arguments of TheIntermediateEigensystems to match the ones above
818     ..."];
819
820     (* Compile a function that will effectively calculate the
821     spectrum of all of the scalar blocks given the parameters of the
822     free-ion part of the Hamiltonian *)
823     (* Compile one function for each of the blocks *)
824     PrintFun["Compiling functions for the diagonal blocks of the
825     Hamiltonian ..."];
826     compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate[N[
827       Normal[#]]]&/@diagonalScalarBlocks;
828     (* Use that to create a function that will calculate the free-ion
829     eigensystem *)
830     TheIntermediateEigensystems[numEv_, F0v_, F2v_, F4v_, F6v_, M0v_,
831     P2v_, αv_, βv_, γv_, T2v_, T3v_, T4v_, T6v_, T7v_, T8v_] :=
832     (
833       theNumericBlocks = (#[F0v, F2v, F4v, F6v, M0v, P2v, αv, βv,
834       γv, T2v, T3v, T4v, T6v, T7v, T8v])&/@compiledDiagonal;
835       theIntermediateEigensystems = Eigensystem/@theNumericBlocks;
836       Js      = AllowedJ[numEv];
837       basisJ = BasisLSJMJ[numEv,"AsAssociation"->True];
838       (* Having calculated the eigensystems with the removed
839       degeneracies, put the degeneracies back in explicitly *)
840       elevatedIntermediateEigensystems = MapIndexed[EigenLever[#1,2Js
841       [[#2[[1]]]]+1]&, theIntermediateEigensystems];
842       pivot = If[EvenQ[numEv],0,-1/2];
843       LSJmultiplets = (#[[1]]<>ToString[InputForm[#[[2]]]])&/@Select[
844       BasisLSJMJ[numEv],#[[{-1}]]== pivot &];
845       (* Calculate the multiplet assignments that the intermediate
846       basis eigenvectors have *)
847       multipletAssignments = Table[
848

```

```

819      (
820          J           = Js[[idx]];
821          eigenVecs = theIntermediateEigensystems[[idx]][[2]];
822          majorComponentIndices = Ordering[Abs[#]][[-1]] &/
823          @eigenVecs;
824          majorComponentAssignments = LSJmultiplets[[#]] &/
825          @majorComponentIndices;
826          (* All of the degenerate eigenvectors belong to the same
827          multiplet*)
828          elevatedMultipletAssignments = ListRepeater[
829          majorComponentAssignments, 2J+1];
830          elevatedMultipletAssignments
831          ),
832          {idx, 1, Length[Js]}
833          ];
834          (* Put together the multiplet assignments and the energies *)
835          freeEnergiesAndMultiplets = Transpose /@ Transpose[{First/
836          @elevatedIntermediateEigensystems, multipletAssignments}];
837          freeEnergiesAndMultiplets = Flatten[freeEnergiesAndMultiplets
838          , 1];
839          (* Calculate the change of basis matrix using the intermediate
840          coupling eigenvectors *)
841          basisChanger = BlockDiagonalMatrix[Transpose /@ Last /
842          @elevatedIntermediateEigensystems];
843          basisChanger = SparseArray[basisChanger];
844          Return[{theIntermediateEigensystems, multipletAssignments,
845          elevatedIntermediateEigensystems, freeEnergiesAndMultiplets,
846          basisChanger}]
847          );
848
849          PrintFun["Calculating the intermediate eigensystems for ", ln,
850          " using free-ion params from LaF3 ..."];
851          (* Calculate intermediate coupling basis using the free-ion
852          params for LaF3 *)
853          {theIntermediateEigensystems, multipletAssignments,
854          elevatedIntermediateEigensystems, freeEnergiesAndMultiplets,
855          basisChanger} = TheIntermediateEigensystems @@ freeBies;
856
857          (* Use that intermediate coupling basis to compile a function for
858          the full Hamiltonian *)
859          allFreeEnergies = Flatten[First /@ elevatedIntermediateEigensystems
860          ];
861          (* Important that the intermediate coupling basis have attached
862          energies, which make possible the truncation *)
863          ordering = Ordering[allFreeEnergies];
864          (* Sort the free ion energies and determine which indices should
865          be included in the truncation *)
866          allFreeEnergiesSorted = Sort[allFreeEnergies];
867          {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];
868          (* Determine the index at which the energy is equal or larger
869          than the truncation energy *)
870          sortedTruncationIndex = Which[
871              truncationEnergy > (maxFreeEnergy - minFreeEnergy),
872              hamDim,
873              True,
874              FirstPosition[allFreeEnergiesSorted - Min[allFreeEnergiesSorted], x_ /; x > truncationEnergy, {0}, 1][[1]]
875              ];
876          (* The actual energy at which the truncation is made *)
877          roundedTruncationEnergy = allFreeEnergiesSorted[[sortedTruncationIndex]];
878
879          (* The indices that enact the truncation *)
880          truncationIndices = ordering[[;; sortedTruncationIndex]];
881
882          (* Using the ham (with all the symbols) change the basis to the
883          computed one *)
884          PrintFun["Changing the basis of the Hamiltonian to the
885          intermediate coupling basis ..."];
886          intermediateHam = Transpose[basisChanger].ArrayFlatten
887          [ham].basisChanger;
888          (* Using the truncation indices truncate that one *)
889          PrintFun["Truncating the Hamiltonian ..."];
890          truncatedIntermediateHam = intermediateHam[[truncationIndices,
891          truncationIndices]];
892          (* These are the basis vectors for the truncated hamiltonian *)

```

```

870 PrintFun["Saving the truncated intermediate basis ..."];
871 truncatedIntermediateBasis = basisChanger[[All, truncationIndices
872 ]];
873 PrintFun["Compiling a function for the truncated Hamiltonian ..."
874 ];
875 (* Compile a function that will calculate the truncated
876 Hamiltonian given the parameters in allVars, this is the function
877 to be use in fitting *)
878 compileIntermediateTruncatedHam = Compile[Evaluate[allVars],
879 Evaluate[N[Normal[
880 truncatedIntermediateHam]]]];
881 (* Save the compiled function *)
882 PrintFun["Saving the compiled function for the truncated
883 Hamiltonian and the truncatedIntermediateBasis..."];
884 Export[compileIntermediateFname, {compileIntermediateTruncatedHam,
885 truncatedIntermediateBasis}];
886 )
887 ];
888
889 TheTruncatedAndSignedPathGenerator::usage = "This function puts
890 together the necessary expression for defining a function which
891 has as arguments all the symbolic values in varsMixedWithVals and
892 which feeds to compileIntermediateTruncatedHam the arguments as
893 given in varsMixedWithVals. varsMixedWithVals needs to respect the
894 order of aruments expected by compileIntermediateTruncatedHam.
895 Once the necessary template has been used this function then
896 results in the definition of the function
897 TheTruncatedAndSignedPath.";
898 TheTruncatedAndSignedPathGenerator[varsMixedWithVals_List]:=(
899 variableVars = Select[varsMixedWithVals, Not[NumericQ[#]]&];
900 numQSignature = StringJoin[Riffle[(ToString[#]<>"_?NumericQ")&/
901 @variableVars, ", "]];
902 varWithValsSignature = StringJoin[Riffle[(ToString[#]<> "")&/
903 @varsMixedWithVals, ", "]];
904 funcString = truncatedEnergyCostTemplate[<|"varsWithNumericQ"-
905 >numQSignature,"varsMixedWithFixedVals" -> varWithValsSignature
906 |>];
907 ClearAll[TheTruncatedAndSignedPath];
908 ToExpression[funcString]
909 );
910
911 (* We need to create a function call that has all the frozen
912 parameters in place and all the active symbols unevaluated *)
913 (* find the indices of the activeVars to create the function
914 signature *)
915 activeVarIndices = Flatten[Position[allVars, #]&/@activeVars];
916 (* we start from the numerical values in the current best*)
917 jobVars = startingValues;
918 (* we then put back the symbols that should be unevaluated *)
919 jobVars[[activeVarIndices]] = activeVars;
920
921 oddsAndEnds["jobVars"] = jobVars;
922 (* calculate the constraints *)
923 constraints = N[Constrainer[activeVars, ln]];
924 oddsAndEnds["constraints"] = constraints;
925 (* This is useful for the progress window *)
926 activeVarsString = StringJoin[Riffle[ToString/@activeVars, ", "]];
927 TheTruncatedAndSignedPathGenerator[jobVars];
928 stringPartialVars = ToString/@activeVars;
929
930 activeVarsWithRange = If[usingInitialRange,
931 MapIndexed[Flatten[{#1,
932 (1-Sign[startVarValues[[#2]]]*fractionalWidth) *
933 startVarValues[[#2]],
934 (1+Sign[startVarValues[[#2]]]*fractionalWidth) *
935 startVarValues[[#2]]
936 }]&, activeVars],
937 activeVars
938 ];
939
940 (* this is the template for the minimizer *)
941 solverTemplateNMini = StringTemplate["
942 numIter = 0;
943 sol = NMinimize[
944 Evaluate[

```

```

923 Join[{TheTruncatedAndSignedPath[{"activeVarsString"}],
924   constraints
925 ]
926 ],
927 activeVarsWithRange,
928 AccuracyGoal -> "accuracyGoal",
929 MaxIterations -> "maxIterations",
930 Method -> "Method",
931 "Monitor":>(
932   currentErr = TheTruncatedAndSignedPath[{"activeVarsString"}];
933   currentParams = activeVars;
934   numIter += 1;
935   rmsHistory = AddToList[rmsHistory, currentErr, maxHistory, False];
936   paramSols = AddToList[paramSols, activeVars, maxHistory, False];
937 ];
938 If[Not[runningInteractive], (
939   Print[numIter, "/\\", "maxIterations"];
940   Print["err = \\", ToString[NumberForm[Round[currentErr, 0.001], {Infinity, 3}]]];
941   Print["params = \\", ToString[NumberForm[Round[#, 0.0001], {Infinity, 4}]] &/@ currentParams];
942   )
943 );
944 ]
945 ];
methodStringTemplate = StringTemplate[
946   {"DifferentialEvolution",
947     "PostProcess" -> False,
948     "ScalingFactor" -> "DE:ScalingFactor",
949     "CrossProbability" -> "DE:CrossProbability",
950     "RandomSeed" -> RandomInteger[{0, 1000000}],
951     "SearchPoints" -> "DE:SearchPoints"};
952 methodString = methodStringTemplate[<|
953   "DE:ScalingFactor" -> OptionValue["DE:ScalingFactor"],
954   "DE:CrossProbability" -> OptionValue["DE:CrossProbability"],
955   "DE:SearchPoints" -> OptionValue["DE:SearchPoints"]|>];
(* Evaluate the template *)
solverCode = solverTemplateNMini[<|
959   "accuracyGoal" -> accuracyGoal,
960   "maxIterations" -> maxIterations,
961   "Method" -> {"DifferentialEvolution",
962     "PostProcess" -> False,
963     "ScalingFactor" -> 0.6,
964     "CrossProbability" -> 0.25,
965     "RandomSeed" -> RandomInteger[{0, 1000000}],
966     "SearchPoints" -> Automatic},
967   "Monitor" -> "StepMonitor",
968   "activeVarsString" -> activeVarsString|>];
969 ];
threadHeaderTemplate = StringTemplate[ "({idx}/{reps}) Fitting data
  for `ln` using `freeVars`."];
(* Find as many solutions as numReps *)
sols = Table[(
973   rmsHistory = {};
974   paramSols = {};
975   openNotebooks = If[runningInteractive,
976     ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks
977     []|,
978     {}];
979   If[Not[MemberQ[openNotebooks, "Solver Progress"]] && OptionValue["ProgressView"],
980     ProgressNotebook["Basic" -> False]
981   ];
982   If[Not[slackChan === None],
983     (
984       threadMessage = threadHeaderTemplate[<|"reps" -> numReps, "idx"
985       -> rep, "ln" -> ln,
986       "freeVars" -> ToString[activeVars]|>];
987       threadTS = PostMessageToSlack[threadMessage, slackChan]["ts"]
988     )
989   ];
990   startTime = Now;
991   ToExpression[solverCode];
992 ];

```

```

991 timeTaken = QuantityMagnitude[Now - startTime, "Seconds"];
992 Print["Took " <> ToString[timeTaken] <> "s"];
993 Print[sol];
994 bestError = sol[[1]];
995 bestParams = sol[[2]];
996 resultMessage = "sigma=" <> ToString[Round[bestError, 0.1]];
997 solAssoc = <|
998   "bestRMS" -> bestError,
999   "solHistory" -> rmsHistory,
1000  "bestParams" -> bestParams,
1001  "paramHistory" -> paramSols,
1002  "timeTaken/s" -> timeTaken
1003 |>;
1004 solAssoc = Join[solAssoc, oddsAndEnds];
1005 logFname = LogSol[solAssoc, logFilePrefix];
1006
1007 If[Not[slackChan === None], (
1008   PostMessageToSlack[resultMessage, slackChan, "threadTS" ->
1009   threadTS];
1010   PostFileToSlack[StringSplit[logFname, "/"][[ -1]], logFname,
1011   slackChan, "threadTS" -> threadTS]
1012 )
1013 ];
1014 solAssoc
1015 ),
1016 {rep, 1, numReps}
1017 ];
1018 Return[sols];
1019 );
1020
1021 ClassicalFit::usage = "ClassicalFit[numE, expData, excludeDataIndices,
1022 problemVars, startValues, \[Sigma]exp, constraints_List, Options]
1023 fits the given expData in an f^numE configuration, by using the
1024 symbols in problemVars. The symbols given in problemVars may be
1025 constrained or held constant, this being controlled by constraints
1026 list which is a list of replacement rules expressing desired
1027 constraints. The constraints list additional constraints imposed
1028 upon the model parameters that remain once other simplifications
1029 have been \"baked\" into the compiled Hamiltonians that are used
1030 to increase the speed of the calculation.
1031
1032 Important, note that in the case of odd number of electrons the given
1033 data must explicitly include the Kramers degeneracy;
1034 excludeDataIndices must be compatible with this.
1035
1036 The list expData needs to be a list of lists with the only
1037 restriction that the first element of them corresponds to energies
1038 of levels. In this list, an empty value can be used to indicate
1039 known gaps in the data. Even if the energy value for a level is
1040 known (and given in expData) certain values can be omitted from
1041 the fitting procedure through the list excludeDataIndices, which
1042 correspond to indices in expData that should be skipped over.
1043
1044 The Hamiltonian used for fitting is version that has been truncated
1045 either by using the maximum energy given in expData or by manually
1046 setting a truncation energy using the option \"TruncationEnergy\".
1047
1048 The argument \[Sigma]exp is the estimated uncertainty in the
1049 differences between the calculated and the experimental energy
1050 levels. This is used to estimate the uncertainty in the fitted
1051 parameters. Admittedly this will be a rough estimate (at least on
1052 the contribution of the calculated uncertainty), but it is better
1053 than nothing and may at least provide a lower bound to the
1054 uncertainty in the fitted parameters. It is assumed that the
1055 uncertainty in the differences between the calculated and the
1056 experimental energy levels is the same for all of them.
1057
1058 The list startValues is a list with all of the parameters needed to
1059 define the Hamiltonian (including the initial values for
1060 problemVars).
1061
1062 The function saves the solution to a file. The file is named with a
1063 prefix (controlled by the option \"FilePrefix\") and a UUID. The
1064 file is saved in the log sub-directory as a .m file.
1065
1066

```

```

1033 Here's a description of the different parts of this function: first
1034   the Hamiltonian is assembled and simplified using the given
1035   simplifications. Then the intermediate coupling basis is
1036   calculated using the free-ion parameters for the given lanthanide.
1037   The Hamiltonian is then changed to the intermediate coupling
1038   basis and truncated. The truncated Hamiltonian is then compiled
1039   into a function that can be used to calculate the energy levels of
1040   the truncated Hamiltonian. The function that calculates the
1041   energy levels is then used to fit the experimental data. The
1042   fitting is done using FindMinimum with the Levenberg-Marquardt
1043   method.
1044
1045 The function returns an association with the following keys:
1046
1047 - \\"bestRMS\\" which is the best \[Sigma] value found.
1048 - \\"bestParams\\" which is the best set of parameters found for the
1049   variables that were not constrained.
1050 - \\"bestParamsWithConstraints\\" which has the best set of parameters
1051   (from - \\"bestParams\") together with the used constraints. These
1052   include all the parameters in the model, even those that were not
1053   fitted for.
1054 - \\"paramSols\\" which is a list of the parameters trajectories during
1055   the stepping of the fitting algorithm.
1056 - \\"timeTaken/s\\" which is the time taken to find the best fit.
1057 - \\"simplifier\\" which is the simplifier used to simplify the
1058   Hamiltonian.
1059 - \\"excludeDataIndices\\" as given in the input.
1060 - \\"startValues\\" as given in the input.
1061
1062 - \\"freeIonSymbols\\" which are the symbols used in the intermediate
1063   coupling basis.
1064 - \\"truncationEnergy\\" which is the energy used to truncate the
1065   Hamiltonian, if it was set to Automatic, the value here is the
1066   actual energy used.
1067 - \\"numE\\" which is the number of electrons in the f^numE
1068   configuration.
1069 - \\"expData\\" which is the experimental data used for fitting.
1070 - \\"problemVars\\" which are the symbols considered for fitting
1071
1072 - \\"maxIterations\\" which is the maximum number of iterations used by
1073   NMinimize.
1074 - \\"hamDim\\" which is the dimension of the full Hamiltonian.
1075 - \\"allVars\\" which are all the symbols defining the Hamiltonian
1076   under the aggregate simplifications.
1077 - \\"freeBies\\" which are the free-ion parameters used to define the
1078   intermediate coupling basis.
1079 - \\"truncatedDim\\" which is the dimension of the truncated
1080   Hamiltonian.
1081 - \\"compiledIntermediateFname\\" the file name of the compiled
1082   function used for the truncated Hamiltonian.
1083
1084 - \\"fittedLevels\\" which is the number of levels fitted for.
1085 - \\"actualSteps\\" the number of steps that FindMinimun actually
1086   took.
1087 - \\"solWithUncertainty\\" which is a list of replacement rules of the
1088   form (paramSymbol -> {bestEstimate, uncertainty}).
1089 - \\"rmsHistory\\" which is a list of the \[Sigma] values found during
1090   the fitting.
1091 - \\"Appendix\\" which is an association appended to the log file under
1092   the key \\"Appendix\\".
1093 - \\"presentDataIndices\\" which is the list of indices in expData that
1094   were used for fitting, this takes into account both the empty
1095   indices in expData and also the indices in excludeDataIndices.
1096
1097 - \\"states\\" which contains a list of eigenvalues and eigenvectors
1098   for the fitted model, this is only available if the option \"
1099   SaveEigenvectors\" is set to True; if a general shift of energy
1100   was allowed for in the fitting, then the energies are shifted
1101   accordingly.
1102 - \\"energies\\" which is a list of the energies of the fitted levels,
1103   this is only available if the option \\"SaveEigenvectors\" is set
1104   to False. If a general shift of energy was allowed for in the
1105   fitting, then the energies are shifted accordingly.
1106 - \\"degreesOfFreedom\\" which is equal to the number of fitted state
1107   energies minus the number of parameters used in fitting.

```

```

1070 | The function admits the following options with corresponding default
1071 | values:
1072 | - \\"MaxHistory\\": determines how long the logs for the solver can be
1073 | .
1074 | - \\"MaxIterations\\": determines the maximum number of iterations used
1075 | by NMinimize.
1076 | - \\"FilePrefix\\": the prefix to use for the subfolder in the log
1077 | filder, in which the solution files are saved, by default this is
1078 | \\"calcs\\" so that the calculation files are saved under the
1079 | directory \\"log/calcs\\".
1080 | - \\"AddConstantShift\\": if True then a constant shift is allowed in
1081 | the fitting, default is False. If this is the case the variable \"
1082 | \\[Epsilon]\\" is added to the list of variables to be fitted for,
1083 | it must not be included in problemVars.
1084 |
1085 | - \\"AccuracyGoal\\": the accuracy goal used by NMinimize, default of
1086 | 5.
1087 | - \\"TruncationEnergy\\": if Automatic then the maximum energy in
1088 | expData is taken, else it takes the value set by this option. In
1089 | all cases the energies in expData are only considered up to this
1090 | value.
1091 | - \\"PrintFun\\": the function used to print progress messages, the
1092 | default is PrintTemporary.
1093 | - \\"RefParamsVintage\\": the vintage of the reference parameters to
1094 | use. The reference parameters are both used to determine the
1095 | truncated Hamiltonian, and also as starting values for the solver.
1096 | It may be \\"LaF3\\", in which case reference parameters from
1097 | Carnall are used. It may also be \\"LiYF4\\", in which case the
1098 | reference parameters from the LiYF4 paper are used. It may also be
1099 | Automatic, in which case the given experimental data is used to
1100 | determine starting values for  $F^k$  and  $\zeta$ . It may also be a list or
1101 | association that provides values for the Slater integrals and spin
1102 | -orbit coupling, the remaining necessary parameters complemented
1103 | by using \\"LaF3\\".
1104 |
1105 | - \\"SlackChannel\\": name of the Slack channel to which to dump
1106 | progress messaages, the default is None which disables this option
1107 | .
1108 | - \\"ProgressView\\": whether or not a progress window will be opened
1109 | to show the progress of the solver, the default is True.
1110 | - \\"SignatureCheck\\": if True then then the function returns
1111 | prematurely, returning a list with the symbols that would have
1112 | defined the Hamiltonian after all simplifications have been
1113 | applied. Useful to check the entire parameter set that the
1114 | Hamiltonian has, which has to match one-to-one what is provided by
1115 | startingValues.
1116 | - \\"SaveEigenvectors\\": if True then the both the eigenvectors and
1117 | eigenvalues are saved under the \\"states\\" key of the returned
1118 | association. If False then only the energies are saved, the
1119 | default is False.
1120 |
1121 | - \\"AppendToFile\\": an association appended to the log file under
1122 | the key \\"Appendix\\".
1123 | - \\"MagneticSimplifier\\": a list of replacement rules to simplify the
1124 | Marvin and pesudo-magnetic paramters. Here the ratios of the
1125 | Marvin parameters and the pseudo-magnetic parameters are defined
1126 | to simplify the magnetic part of the Hamiltonian.
1127 | - \\"MagFieldSimplifier\\": a list of replacement rules to specify a
1128 | magnetic field (in T), if set to {}, then {Bx, By, Bz} can also be
1129 | used as variables to be fitted for.
1130 |
1131 | - \\"SymmetrySimplifier\\": a list of replacements rules to simplify
1132 | the crystal field.
1133 | - \\"OtherSimplifier\\": an additional list of replacement rules that
1134 | are applied to the Hamiltonian before computing with it. Here the
1135 | spin-spin contribution can be turned off by setting \[Sigma]SS->0,
1136 | which is the default.
1137 |
1138 | Options[ClassicalFit] = {
1139 |   "MaxHistory"      -> 200,
1140 |   "MaxIterations"   -> 100,
1141 |   "FilePrefix"      -> "calcs",
1142 |   "ProgressView"    -> True,
1143 |   "TruncationEnergy" -> Automatic,
1144 |   "AccuracyGoal"    -> 5,
1145 |   "PrintFun"        -> PrintTemporary,

```

```

1101 "SlackChannel"      -> None,
1102 "RefParamsVintage"  -> "LaF3",
1103 "SignatureCheck"    -> False,
1104 "AddConstantShift" -> False,
1105 "SaveEigenvectors" -> False,
1106 "AppendToLogFile"   -> <||>,
1107 "SaveToLog"         -> False,
1108 "Energy Uncertainty in K" -> Automatic,
1109 "MagneticSimplifier" -> {
1110     M2 -> 56/100 MO,
1111     M4 -> 31/100 MO,
1112     P4 -> 1/2 P2,
1113     P6 -> 1/10 P2
1114 },
1115 "MagFieldSimplifier" -> {
1116     Bx -> 0,
1117     By -> 0,
1118     Bz -> 0
1119 },
1120 "SymmetrySimplifier" -> {
1121     B12->0, B14->0, B16->0, B34->0, B36->0, B56->0,
1122     S12->0, S14->0, S16->0, S22->0, S24->0, S26->0,
1123     S34->0, S36->0, S44->0, S46->0, S56->0, S66->0
1124 },
1125 "OtherSimplifier" -> {
1126     F0->0,
1127     P0->0,
1128     \[\Sigma]SS->0,
1129     T11p->0, T12->0, T14->0, T15->0,
1130     T16->0, T18->0, T17->0, T19->0, T2p->0,
1131     wChErrA ->0, wChErrB->0
1132 },
1133 "ThreeBodySimplifier" -> <|
1134     1 -> {
1135         T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
1136         T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1137     ->0,
1138         T2p->0},
1139     2 -> {
1140         T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
1141         T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1142     ->0,
1143         T2p->0
1144     },
1145     3 -> {},
1146     4 -> {},
1147     5 -> {},
1148     6 -> {},
1149     7 -> {},
1150     8 -> {},
1151     9 -> {},
1152     10 -> {},
1153     11 -> {},
1154     12 -> {
1155         T3->0, T4->0, T6->0, T7->0, T8->0,
1156         T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1157     ->0,
1158         T2p->0
1159     },
1160     13->{
1161         T2->0, T3->0, T4->0, T6->0, T7->0, T8->0,
1162         T11p->0, T12->0, T14->0, T15->0, T16->0, T18->0, T17->0, T19
1163     ->0,
1164         T2p->0
1165     }
1166     |>,
1167     "FreeIonSymbols" -> {F0, F2, F4, F6,  $\zeta$ }
1168 };
1169 ClassicalFit[numE_Integer, expData_List, excludeDataIndices_List,
1170 problemVars_List, startValues_Association, constraints_List,
1171 OptionsPattern[]]:=Module[
1172 {accuracyGoal, activeVarIndices, activeVars, activeVarsString,
1173 activeVarsWithRange, allFreeEnergies, allFreeEnergiesSorted,
1174 allVars, allVarsVec, argsForEvalInsideOfTheIntermediateSystems,
1175 argsOfTheIntermediateEigensystems, aVar, aVarPosition, basis,
1176 basisChanger, basisChangerBlocks, bestError, bestParams, bestRMS,

```

```

blockShifts, blockSizes, colIdx, compiledDiagonal,
compiledIntermediateFname, constrainedProblemVars,
constrainedProblemVarsList, currentRMS, degreesOfFreedom,
dependentVars, diagonalBlocks, diagonalScalarBlocks, diff,
eigenEnergies, eigenvalueDispenserTemplate, eigenVectors,
elevatedIntermediateEigensystems, endTime, fmSol, fmSolAssoc,
fractionalWidth, freeBies, freeEigenvaluesAndMultiplets, fullHam,
fullSolVec, funcString, ham, hamDim, hamEigenvaluesTemplate,
hamString, indepSolVecVec, indepVars, intermediateHam,
isolationValues, jobVars, lin, linMat, ln, lnParams, logFilePrefix
, logFname, magneticSimplifier, maxFreeEnergy, maxHistory,
maxIterations, methodString, methodStringTemplate, minFreeEnergy,
minpoly, modelSymbols, multipletAssignments, needlePosition,
numBlocks, numQSignature, numReps, solCompendium, openNotebooks,
ordering, otherSimplifier, p0, paramBest, paramSigma, perHam,
polySols, presentDataIndices, PrintFun, problemVarsPositions,
problemVarsQ, problemVarsQString, problemVarsVec,
problemVarsWithStartValues, reducedModelSymbols, resultMessage,
roundedTruncationEnergy, rowIdx, runningInteractive, shiftToggle,
simplifier, slackChan, sol, solAssoc, sols, solWithUncertainty,
sortedTruncationIndex, sqdiff, standardValues, starTime,
startingValues, startTime, startVarValues, states, steps,
symmetrySimplifier, theIntermediateEigensystems,
TheIntermediateEigensystems, TheTruncatedAndSignedPathGenerator,
thisPoly, threadHeaderTemplate, threadMessage, threadTS, timeTaken
, totalVariance, truncadedFname, truncatedIntermediateBasis,
truncatedIntermediateHam, truncationEnergy, truncationIndices,
RefParams, truncationUmbra, usingInitialRange, varHash, varIdx,
varsWithConstants, varWithValsSignature, \[Lambda]OVec, \[Lambda]
exp},
1167 (
1168 \[Sigma]exp = OptionValue["Energy Uncertainty in K"];
1169 solCompendium = <||>;
1170 refParamsVintage = OptionValue["RefParamsVintage"];
1171 RefParams = Which[
1172   refParamsVintage === "LaF3",
1173   LoadLaF3Parameters,
1174   refParamsVintage === "LiYF4",
1175   LoadLiYF4Parameters,
1176   True,
1177   refParamsVintage
1178 ];
1179 hamDim = Binomial[14, numE];
1180 addShift = OptionValue["AddConstantShift"];
1181 ln = theLanthanides[[numE]];
1182 maxHistory = OptionValue["MaxHistory"];
1183 maxIterations = OptionValue["MaxIterations"];
1184 logFilePrefix = If[OptionValue["FilePrefix"] == "",
1185   ToString[theLanthanides[[numE]]],
1186   OptionValue["FilePrefix"]
1187 ];
1188 accuracyGoal = OptionValue["AccuracyGoal"];
1189 slackChan = OptionValue["SlackChannel"];
1190 PrintFun = OptionValue["PrintFun"];
1191 freeIonSymbols = OptionValue["FreeIonSymbols"];
1192 runningInteractive = (Head[$ParentLink] === LinkObject);
1193 magneticSimplifier = OptionValue["MagneticSimplifier"];
1194 magFieldSimplifier = OptionValue["MagFieldSimplifier"];
1195 symmetrySimplifier = OptionValue["SymmetrySimplifier"];
1196 otherSimplifier = OptionValue["OtherSimplifier"];
1197 threeBodySimplifier = If[Head[OptionValue["ThreeBodySimplifier"]]
1198 == Association,
1199   OptionValue["ThreeBodySimplifier"][numE],
1200   OptionValue["ThreeBodySimplifier"]
1201 ];
1202 truncationEnergy = If[OptionValue["TruncationEnergy"] ===
1203 Automatic,
1204 (
1205   PrintFun["Truncation energy set to Automatic, using the
1206 maximum energy (+20%) in the data ..."];
1207   Round[1.2 * Max[Select[First /@ expData, NumericQ[#] &]]
1208 ],
1209 OptionValue["TruncationEnergy"]
1210 ];
1211 truncationEnergy = Max[50000, truncationEnergy];

```

```

1210 PrintFun["Using a truncation energy of ", truncationEnergy, " K"]
];
1211
1212 simplifier = Join[magneticSimplifier,
1213                      magFieldSimplifier,
1214                      symmetrySimplifier,
1215                      threeBodySimplifier,
1216                      otherSimplifier];
1217
1218 PrintFun["Determining gaps in the data ..."];
1219 (* whatever is non-numeric is assumed as a known gap *)
1220 presentDataIndices = Flatten[Position[expData, {_?(NumericQ[#] &
1221 , ___]}];
1222 (* some indices omitted here based on the excludeDataIndices
argument *)
1223 presentDataIndices = Complement[presentDataIndices,
excludeDataIndices];
1224
1225 solCompendium["simplifier"] = simplifier;
1226 solCompendium["excludeDataIndices"] = excludeDataIndices;
1227 solCompendium["startValues"] = startValues;
1228 solCompendium["freeIonSymbols"] = freeIonSymbols;
1229 solCompendium["truncationEnergy"] = truncationEnergy;
1230 solCompendium["numE"] = numE;
1231 solCompendium["expData"] = expData;
1232 solCompendium["problemVars"] = problemVars;
1233 solCompendium["maxIterations"] = maxIterations;
1234 solCompendium["hamDim"] = hamDim;
1235 solCompendium["constraints"] = constraints;
1236
1237 modelSymbols = Sort[Select[paramSymbols, Not[MemberQ[Join[
1238 racahSymbols, juddOfeltIntensitySymbols, chenSymbols, {t2Switch, \
Epsilon}, gs, nE}], #]]&];
(* remove the symbols that will be removed by the simplifier, no
symbol should remain here that is not in the symbolic Hamiltonian *)
1239 reducedModelSymbols = Select[modelSymbols, Not[MemberQ[Keys[
1240 simplifier], #]]&];
1241
1242 (* this is useful to understand what are the arguments of the
truncated compiled Hamiltonian *)
1243 If[OptionValue["SignatureCheck"],
1244 (
1245 PrintFun["Given the model parameters and the simplifying
assumptions, the resultant model parameters are:"];
1246 PrintFun[{reducedModelSymbols}];
1247 PrintFun["Exiting ..."];
1248 Return[""];
1249 )
];
1250
1251 (* calculate the basis *)
1252 PrintFun["Retrieving the LSJM basis for f^", numE, " ..."];
1253 basis = BasisLSJM[numE];
1254
1255 Which[refParamsVintage === Automatic,
1256 (
1257 PrintFun["Using the automatic vintage with freshly fitted
free-ion parameters and others as in LaF3 ..."];
1258 lnParams = LoadLaF3Parameters[ln];
1259 freeIonSol = FreeIonSolver[expData, numE];
1260 freeIonParams = freeIonSol["bestParams"];
1261 lnParams = Join[lnParams, freeIonParams];
1262 ),
1263 MemberQ[{List, Association}, Head[RefParams]],
1264 (
1265 RefParams = Association[RefParams];
1266 PrintFun["Using the given parameters as a starting point ..."
];
1267 lnParams = RefParams;
1268 extraParams = LoadLaF3Parameters[ln];
1269 lnParams = Join[extraParams, lnParams];
1270 ),
1271 True,
1272 (
(* get the reference parameters from the given vintage *)

```

```

1273     PrintFun["Getting reference free-ion parameters for ", ln, " "
1274     using , refParamsVintage, " ..."];
1275     lnParams = ParamPad[RefParams[ln], "PrintFun" -> PrintFun];
1276   );
1277   freeBies = Prepend[Values[(#->(#/lnParams)) &/@ freeIonSymbols], 
1278   numE];
1279   (* a more explicit alias *)
1280   allVars = reducedModelSymbols;
1281   numericConstraints = Association@Select[constraints, NumericQ
1282   #[[2]]] &;
1283   standardValues = allVars /. Join[lnParams, numericConstraints];
1284   solCompendium["allVars"] = allVars;
1285   solCompendium["freeBies"] = freeBies;
1286 
1287   (* reload compiled version if found *)
1288   varHash = Hash[{numE, allVars, freeBies,
1289   truncationEnergy, simplifier}];
1290   compiledIntermediateFname = ln <> "-compiled-intermediate-
1291   truncated-ham-" <> ToString[varHash] <> ".mx";
1292   compiledIntermediateFname = FileNameJoin[{moduleDir, "compiled",
1293   compiledIntermediateFname}];
1294   solCompendium["compiledIntermediateFname"] =
1295   compiledIntermediateFname;
1296 
1297   If[FileExistsQ[compiledIntermediateFname],
1298     PrintFun["This ion, free-ion params, and full set of variables
1299     have been used before (as determined by {numE, allVars, freeBies,
1300     truncationEnergy, simplifier}). Loading the previously saved
1301     compiled function and intermediate coupling basis ..."];
1302     PrintFun["Using : ", compiledIntermediateFname];
1303     {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =
1304     Import[compiledIntermediateFname];
1305     (
1306       If[truncationEnergy == Infinity,
1307       (
1308         ham = HamMatrixAssembly[numE, "ReturnInBlocks" -> False];
1309         theSimplifier = simplifier;
1310         ham = Normal@ReplaceInSparseArray[ham, simplifier];
1311         PrintFun["Compiling a function for the Hamiltonian with no
1312         truncation ..."];
1313         (* compile a function that will calculate the truncated
1314         Hamiltonian given the parameters in allVars, this is the function
1315         to be use in fitting *)
1316         compileIntermediateTruncatedHam = Compile[Evaluate[allVars
1317         ], Evaluate[ham]];
1318         truncatedIntermediateBasis = SparseArray@IdentityMatrix[
1319           Binomial[14, numE]];
1320         (* save the compiled function *)
1321         PrintFun["Saving the compiled function for the Hamiltonian
1322         with no truncation and a placeholder intermediate basis ..."];
1323         Export[compiledIntermediateFname, {
1324           compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
1325       ),
1326       (
1327         (* grab the Hamiltonian preserving the block structure *)
1328         PrintFun["Assembling the Hamiltonian for f^", numE, " keeping
1329         the block structure ..."];
1330         ham = HamMatrixAssembly[numE, "ReturnInBlocks" -> True
1331         ];
1332         (* apply the simplifier *)
1333         PrintFun["Simplifying using the aggregate set of
1334         simplification rules ..."];
1335         ham = Map[ReplaceInSparseArray[#, simplifier]&, ham,
1336         {2}];
1337         PrintFun["Zeroing out every symbol in the Hamiltonian that is
1338         not a free-ion parameter ..."];
1339         (* Get the free ion symbols *)
1340         freeIonSimplifier = (#->0) & /@ Complement[
1341           reducedModelSymbols, freeIonSymbols];
1342         (* Take the diagonal blocks for the intermediate analysis *)
1343         PrintFun["Grabbing the diagonal blocks of the Hamiltonian ...
1344         "];
1345         diagonalBlocks = Diagonal[ham];
1346         (* simplify them to only keep the free ion symbols *)
1347       )
1348     )
1349   )
1350 
```

```

1323     PrintFun["Simplifying the diagonal blocks to only keep the
1324 free ion symbols ..."];
1325     diagonalScalarBlocks = ReplaceInSparseArray[#, 
1326 freeIonSimplifier]&/@diagonalBlocks;
1327     (* these include the MJ quantum numbers, remove that *)
1328     PrintFun["Contracting the basis vectors by removing the MJ
1329 quantum numbers from the diagonal blocks ..."];
1330     diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
1331
1332     argsOfTheIntermediateEigensystems      = StringJoin[Riffle
1333 [Prepend[(ToString[#]<>"v_") & /@ freeIonSymbols,"numE_"],", ", "]];
1334     argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle
1335 [((ToString[#]<>"v") & /@ freeIonSymbols),", "]];
1336     PrintFun["argsOfTheIntermediateEigensystems = ",
1337 argsOfTheIntermediateEigensystems];
1338     PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
1339 argsForEvalInsideOfTheIntermediateSystems];
1340     PrintFun["(if the following fails, it might help to see if
1341 the arguments of TheIntermediateEigensystems match the ones shown
1342 above)"];
1343
1344     (* compile a function that will effectively calculate the
1345 spectrum of all of the scalar blocks given the parameters of the
1346 free-ion part of the Hamiltonian *)
1347     (* compile one function for each of the blocks *)
1348     PrintFun["Compiling functions for the diagonal blocks of the
1349 Hamiltonian ..."];
1350     compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate
1351 [N[Normal[#]]]]&/@diagonalScalarBlocks;
1352     (* use that to create a function that will calculate the free
1353 -ion eigensystem *)
1354     TheIntermediateEigensystems[numEv_, F0v_, F2v_, F4v_, F6v_,  $\zeta$ 
1355 v_] := (
1356     theNumericBlocks = (#[F0v, F2v, F4v, F6v,  $\zeta$ v]&) /@
1357 compiledDiagonal;
1358     theIntermediateEigensystems = Eigensystem /@
1359 theNumericBlocks;
1360     Js      = AllowedJ[numEv];
1361     basisJ = BasisLSJMJ[numEv, "AsAssociation" -> True];
1362     (* having calculated the eigensystems with the removed
1363 degeneracies, put the degeneracies back in explicitly *)
1364     elevatedIntermediateEigensystems = MapIndexed[EigenLever
1365 #[#1, 2Js[[#2[[1]]]]+1 ]&, theIntermediateEigensystems];
1366     (* Identify a single MJ to keep *)
1367     pivot      = If[EvenQ[numEv], 0, -1/2];
1368     LSJmultiplets = (#[[1]]<>ToString[InputForm[#[[2]]]])&/
1369 @Select[BasisLSJMJ[numEv], ##[[-1]] == pivot &];
1370     (* calculate the multiplet assignments that the
1371 intermediate basis eigenvectors have *)
1372     needlePosition = 0;
1373     multipletAssignments = Table[
1374       (
1375         J          = Js[[idx]];
1376         eigenVecs = theIntermediateEigensystems[[idx]][[2]];
1377         majorComponentIndices = Ordering[Abs[#][[-1]]]&/
1378 @eigenVecs;
1379         majorComponentIndices += needlePosition;
1380         needlePosition += Length[
1381 majorComponentIndices];
1382         majorComponentAssignments = LSJmultiplets[[#]]&/
1383 @majorComponentIndices;
1384         (* All of the degenerate eigenvectors belong to the
1385 same multiplet*)
1386         elevatedMultipletAssignments = ListRepeater[
1387 majorComponentAssignments, 2J+1];
1388         elevatedMultipletAssignments
1389       ),
1390       {idx, 1, Length[Js]}
1391     ];
1392     (* put together the multiplet assignments and the energies
1393 *)
1394     freeIenergiesAndMultiplets = Transpose /@ Transpose[{First/
1395 @elevatedIntermediateEigensystems, multipletAssignments}];
1396     freeIenergiesAndMultiplets = Flatten[
1397 freeIenergiesAndMultiplets, 1];
1398     (* calculate the change of basis matrix using the
1399 */

```

```

1370     intermediate coupling eigenvectors *)
1371     basisChanger = BlockDiagonalMatrix[Transpose/@Last/
1372 @elevatedIntermediateEigensystems];
1373     basisChanger = SparseArray[basisChanger];
1374     Return[{theIntermediateEigensystems, multipletAssignments,
1375 elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
1376 basisChanger}]
1377   );
1378
1379   PrintFun["Calculating the intermediate eigensystems for ",ln,
1380 " using free-ion params from LaF3 ..."];
1381   (* calculate intermediate coupling basis using the free-ion
1382 params for LaF3 *)
1383   {theIntermediateEigensystems, multipletAssignments,
1384 elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
1385 basisChanger} = TheIntermediateEigensystems@@freeBies;
1386
1387   (* use that intermediate coupling basis to compile a function
1388 for the full Hamiltonian *)
1389   allFreeEnergies = Flatten[First/
1390 @elevatedIntermediateEigensystems];
1391   (* important that the intermediate coupling basis have
1392 attached energies, which make possible the truncation *)
1393   ordering = Ordering[allFreeEnergies];
1394   (* sort the free ion energies and determine which indices
1395 should be included in the truncation *)
1396   allFreeEnergiesSorted = Sort[allFreeEnergies];
1397   {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];
1398   (* determine the index at which the energy is equal or larger
1399 than the truncation energy *)
1400   sortedTruncationIndex = Which[
1401     truncationEnergy > (maxFreeEnergy-minFreeEnergy),
1402     hamDim,
1403     True,
1404     FirstPosition[allFreeEnergiesSorted - Min[
1405 allFreeEnergiesSorted],x_/_;x>truncationEnergy,{0},1][[1]]
1406   ];
1407   (* the actual energy at which the truncation is made *)
1408   roundedTruncationEnergy = allFreeEnergiesSorted[[sortedTruncationIndex]];
1409
1410   (* the indices that participate in the truncation *)
1411   truncationIndices = ordering[[;;sortedTruncationIndex]];
1412   PrintFun["Computing the block structure of the change of
1413 basis array ..."];
1414   blockSizes = BlockArrayDimensionsArray[ham];
1415   basisChangerBlocks = ArrayBlocker[basisChanger, blockSizes];
1416   blockShifts = First /@ Diagonal[blockSizes];
1417   numBlocks = Length[blockSizes];
1418   (* using the ham (with all the symbols) change the basis to
1419 the computed one *)
1420   PrintFun["Changing the basis of the Hamiltonian to the
1421 intermediate coupling basis ..."];
1422   intermediateHam = BlockMatrixMultiply[ham, basisChangerBlocks];
1423
1424   PrintFun["Distributing products inside of symbolic matrix
1425 elements to keep complexity in check ..."];
1426   Do[
1427     intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
1428 intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
1429     {rowIdx, 1, numBlocks},
1430     {colIdx, 1, numBlocks}
1431   ];
1432   intermediateHam = BlockMatrixMultiply[BlockTranspose[
1433 basisChangerBlocks], intermediateHam];
1434   PrintFun["Distributing products inside of symbolic matrix
1435 elements to keep complexity in check ..."];
1436   Do[
1437     intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
1438 intermediateHam[[rowIdx, colIdx]], Distribute /@ # &],
1439     {rowIdx, 1, numBlocks},
1440     {colIdx, 1, numBlocks}
1441   ];
1442   (* using the truncation indices truncate that one *)
1443   PrintFun["Truncating the Hamiltonian ..."];
1444   truncatedIntermediateHam = TruncateBlockArray[intermediateHam]

```

```

1422 , truncationIndices, blockShifts];
(* these are the basis vectors for the truncated hamiltonian *)
1423 PrintFun["Saving the truncated intermediate basis ..."];
1424 truncatedIntermediateBasis = basisChanger[[All,
truncationIndices]];
1425
1426 PrintFun["Compiling a function for the truncated Hamiltonian
..."];
(* compile a function that will calculate the truncated
Hamiltonian given the parameters in allVars, this is the function
to be use in fitting *)
1428 compileIntermediateTruncatedHam = Compile[Evaluate[allVars],
Evaluate[truncatedIntermediateHam]];
(* save the compiled function *)
1430 PrintFun["Saving the compiled function for the truncated
Hamiltonian and the truncated intermediate basis ..."];
1431 Export[compiledIntermediateFname, {
compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
1432 )
1433 ];
1434 )
1435 ];
1436
1437 truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
PrintFun["The truncated Hamiltonian has a dimension of ",
truncationUmbral, "x", truncationUmbral, "..."];
1439 presentDataIndices = Select[presentDataIndices, # <=
truncationUmbral &];
1440 solCompendium["presentDataIndices"] = presentDataIndices;
1441
(* the problemVars are the symbols that will be fitted for *)
1443
1444 PrintFun["Starting up the fitting process using the Levenberg-
Marquardt method ..."];
1445 (* using the problemVars I need to create the argument list
including _?NumericQ *)
1446 problemVarsQ = (ToString[#] <> "_?NumericQ") & /@ problemVars;
1447 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
*)
1449 problemVarsPositions = Position[allVars, #][[1, 1]] & /@ problemVars;
1450 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 *)
1452 varsWithConstants = standardValues;
1453 varsWithConstants[[problemVarsPositions]] = problemVars;
1454 varsWithConstantsString = ToString[
varsWithConstants];
1455
1456 (* this following function serves eigenvalues from the
Hamiltonian, has memoization so it might grow to use a lot of RAM
*)
1457 Clear[HamSortedEigenvalues];
1458 hamEigenvaluesTemplate = StringTemplate[
"HamSortedEigenvalues['problemVarsQ']:=(
  ham      = compileIntermediateTruncatedHam@@'
varsWithConstants';
  eigenValues = Chop[Sort@Eigenvalues@ham];
  eigenValues = eigenValues - Min[eigenValues];
  HamSortedEigenvalues['problemVarsString'] = eigenValues;
  Return[eigenValues]
)"];
1466 hamString = hamEigenvaluesTemplate[<|
  "problemVarsQ"    -> problemVarsQString,
  "varsWithConstants" -> varsWithConstantsString,
  "problemVarsString" -> problemVarsString
|>];
1471 ToExpression[hamString];
1472
(* we also need a function that will pick the i-th eigenvalue,
1473

```

```

1474      this seems unnecessary but it's needed to form the right
1475      functional form expected by the Levenberg-Marquardt method *)
1476      eigenvalueDispenserTemplate = StringTemplate["
1477      PartialHamEigenvalues['problemVarsQ'][i_]:=(
1478          eigenVals = HamSortedEigenvalues['problemVarsString'];
1479          eigenVals[[i]]
1480      )
1481      "];
1482      eigenValueDispenserString = eigenvalueDispenserTemplate[<|
1483          "problemVarsQ"      -> problemVarsQString,
1484          "problemVarsString" -> problemVarsString
1485          |>];
1486      ToExpression[eigenValueDispenserString];
1487
1488      PrintFun["Determining the free variables after constraints ..."];
1489      constrainedProblemVars      = (problemVars /. constraints);
1490      constrainedProblemVarsList = Variables[constrainedProblemVars];
1491      If[addShift,
1492          PrintFun["Adding a constant shift to the fitting parameters ...
1493          "];
1494          constrainedProblemVarsList = Append[constrainedProblemVarsList,
1495          \[Epsilon]];
1496      ];
1497
1498      indepVars = Complement[problemVars, #[[1]] & /@ constraints];
1499      stringPartialVars = ToString/@constrainedProblemVarsList;
1500
1501      paramSols = {};
1502      rmsHistory = {};
1503      steps = 0;
1504      problemVarsWithStartValues = KeyValueMap[{#1,#2} &, startValues];
1505      If[addShift,
1506          problemVarsWithStartValues = Append[problemVarsWithStartValues,
1507          {\[Epsilon], 0}];
1508      ];
1509      openNotebooks = If[runningInteractive,
1510          ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks
1511          [],
1512          {}];
1513      If[Not[MemberQ[openNotebooks,"Solver Progress"]] && OptionValue["ProgressView"],
1514          ProgressNotebook["Basic"->False]
1515      ];
1516      degressOfFreedom = Length[presentDataIndices] - Length[
1517      problemVars] - 1;
1518      PrintFun["Fitting for ", Length[presentDataIndices], " data
1519      points with ", Length[problemVars], " free parameters.", " The
1520      effective degrees of freedom are ", degressOfFreedom, " ..."];
1521
1522      PrintFun["Fitting model to data ..."];
1523      startTime = Now;
1524      shiftToggle = If[addShift, 1, 0];
1525      sol = FindMinimum[
1526          Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@
1527          constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
1528          {j, presentDataIndices}],
1529          problemVarsWithStartValues,
1530          Method      -> "LevenbergMarquardt",
1531          MaxIterations -> OptionValue["MaxIterations"],
1532          AccuracyGoal -> OptionValue["AccuracyGoal"],
1533          StepMonitor :> (
1534              steps      += 1;
1535              currentSqSum = Sum[(expData[[j]][[1]] - (
1536                  PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle
1537                  * \[Epsilon])^2, {j, presentDataIndices}];
1538              currentRMS = Sqrt[currentSqSum / degressOfFreedom];
1539              paramSols = AddToList[paramSols, constrainedProblemVarsList,
1540                  maxHistory];
1541              rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
1542          )
1543      ];
1544      endTime = Now;
1545      timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
1546      PrintFun["Solution found in ", timeTaken, "s"];
1547
1548      solVec = constrainedProblemVars /. sol[[-1]];

```

```

1536   indepSolVec = indepVars /. sol[[-1]];
1537   If[addShift,
1538     \[Epsilon]Best = \[Epsilon]/. sol[[-1]],
1539     \[Epsilon]Best = 0
1540   ];
1541   fullSolVec = standardValues;
1542   fullSolVec[[problemVarsPositions]] = solve;
1543   PrintFun["Calculating the truncated numerical Hamiltonian
corresponding to the solution ..."];
1544   fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
1545   PrintFun["Calculating energies and eigenvectors ..."];
1546   {eigenEnergies, eigenVectors} = Eigensystem[fullHam];
1547   states = Transpose[{eigenEnergies, eigenVectors}];
1548   states = SortBy[states, First];
1549   eigenEnergies = First /@ states;
1550   PrintFun["Shifting energies to make ground state zero of energy
..."];
1551   eigenEnergies = eigenEnergies - eigenEnergies[[1]];
1552   PrintFun["Calculating the linear approximant to each eigenvalue
..."];
1553   allVarsVec = Transpose[{allVars}];
1554   p0 = Transpose[{fullSolVec}];
1555   linMat = {};
1556   If[addShift,
1557     tail = -2,
1558     tail = -1];
1559   Do[
1560     (
1561       aVarPosition = Position[allVars, aVar][[1, 1]];
1562       isolationValues = ConstantArray[0, Length[allVars]];
1563       isolationValues[[aVarPosition]] = 1;
1564       dependentVars = KeyValueMap[{#1, D[#2, aVar]} &, Association[
constraints]];
1565       Do[
1566         isolationValues[[Position[allVars, dVar[[1]]][[1, 1]]]] =
dVar[[2]],
1567         {dVar, dependentVars}
1568       ];
1569       perHam = compileIntermediateTruncatedHam @@ isolationValues;
1570       lin = FirstOrderPerturbation[Last /@ states, perHam];
1571       linMat = Append[linMat, lin];
1572     ),
1573     {aVar, constrainedProblemVarsList[[;;tail]]}
1574   ];
1575   PrintFun["Removing the gradient of the ground state ..."];
1576   linMat = (# - #[[1]] & /@ linMat);
1577   PrintFun["Transposing derivative matrices into columns ..."];
1578   linMat = Transpose[linMat];
1579
1580   PrintFun["Calculating the eigenvalue vector at solution ..."];
1581   \[Lambda]0Vec = Transpose[{eigenEnergies[[presentDataIndices]]}];
1582   PrintFun["Putting together the experimental vector ..."];
1583   \[Lambda]exp = Transpose[{First /@ expData[[presentDataIndices
]]}];
1584   problemVarsVec = If[addShift,
1585     Transpose[{constrainedProblemVarsList[[;;-2]]}],
1586     Transpose[{constrainedProblemVarsList}]
1587   ];
1588   indepSolVecVec = Transpose[{indepSolVec}];
1589   PrintFun["Calculating the difference between eigenvalues at
solution ..."];
1590   diff = If[linMat == {},
1591     (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best,
1592     (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best + linMat[[presentDataIndices]].(problemVarsVec - indepSolVecVec)
1593   ];
1594   PrintFun["Calculating the sum of squares of differences around
solution ..."];
1595   sqdiff = Expand[(Transpose[diff] . diff)[[1, 1]]];
1596   PrintFun["Calculating the minimum (which should coincide with sol
) ..."];
1597   minpoly = sqdiff /. AssociationThread[problemVars -> solve];
1598   fmSolAssoc = Association[sol[[2]]];
1599   If\[Sigma]exp == Automatic,
1600     \[Sigma]exp = Sqrt[minpoly / degreesOfFreedom];

```

```

1601 ];
1602 \[CapitalDelta]\[\Chi]2 = Sqrt[degressOfFreedom];
1603 Amat = (1/\[Sigma]exp^2) * Transpose[linMat[[presentDataIndices
1604 ]]].linMat[[presentDataIndices]];
1605 paramIntervals = EllipsoidBoundingBox[Amat, \[CapitalDelta]\[\Chi]
1606 ]2];
1607 PrintFun["Calculating the uncertainty in the parameters ..."];
1608 solWithUncertainty = Table[
1609 (
1610     aVar = constrainedProblemVarsList[[varIdx]];
1611     paramBest = aVar /. fmSolAssoc;
1612     (aVar -> {paramBest, paramIntervals[[varIdx, 2]]})
1613 ),
1614 {varIdx, 1, Length[constrainedProblemVarsList]-shiftToggle}
1615 ];
1616 bestRMS = Sqrt[minpoly / degressOfFreedom];
1617 bestParams = sol[[2]];
1618 bestWithConstraints = Association@Join[constraints, bestParams];
1619 bestWithConstraints = bestWithConstraints /. bestWithConstraints;
1620 bestWithConstraints = (# + 0.) & /@ bestWithConstraints;
1621
1622 solCompendium["degreesOfFreedom"] = degressOfFreedom;
1623 solCompendium["solWithUncertainty"] = solWithUncertainty;
1624 solCompendium["truncatedDim"] = truncationUmbral;
1625 solCompendium["fittedLevels"] = Length[presentDataIndices
1626 ];
1627 solCompendium["actualSteps"] = steps;
1628 solCompendium["bestRMS"] = bestRMS;
1629 solCompendium["problemVars"] = problemVars;
1630 solCompendium["paramSols"] = paramSols;
1631 solCompendium["rmsHistory"] = rmsHistory;
1632 solCompendium["Appendix"] = OptionValue["  

1633 AppendToFile"];
1634 solCompendium["timeTaken/s"] = timeTaken;
1635 solCompendium["bestParams"] = bestParams;
1636 solCompendium["bestParamsWithConstraints"] = bestWithConstraints;
1637
1638 If[OptionValue["SaveEigenvectors"],
1639     solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]}
1640     &/@ (Chop /@ ShiftedLevels[states]),
1641     (
1642         finalEnergies = Sort[First /@ states];
1643         finalEnergies = finalEnergies - finalEnergies[[1]];
1644         finalEnergies = finalEnergies + \[Epsilon]Best;
1645         finalEnergies = Chop /@ finalEnergies;
1646         solCompendium["energies"] = finalEnergies;
1647     )
1648 ];
1649 If[OptionValue["SaveToLog"],
1650     PrintFun["Saving the solution to the log file ..."];
1651     LogSol[solCompendium, logFilePrefix];
1652 ];
1653 PrintFun["Finished ..."];
1654 Return[solCompendium];
1655 ]
1656 ];
1657
1658 MostlyOrthogonalFit::usage="MostlyOrthogonalFit[numE, expData,
1659 excludeDataIndices, problemVars, startValues, \[Sigma]exp,
1660 constraints_List, Options] fits the given expData in an f^numE
1661 configuration, by using the symbols in problemVars. The symbols
1662 given in problemVars may be constrained or held constant, this
1663 being controlled by constraints list which is a list of
1664 replacement rules expressing desired constraints. The constraints
1665 list additional constraints imposed upon the model parameters that
1666 remain once other simplifications have been \"baked\" into the
1667 compiled Hamiltonians that are used to increase the speed of the
1668 calculation.
1669
1670 Important, note that in the case of odd number of electrons the given
1671 data must explicitly include the Kramers degeneracy;
1672 excludeDataIndices must be compatible with this.
1673
1674 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.

1660
 1661 The Hamiltonian used for fitting is version that has been truncated either by using the maximum energy given in `expData` or by manually setting a truncation energy using the option `\"TruncationEnergy\"`
 .

1662
 1663 The argument `\[Sigma]exp` is the estimated uncertainty in the differences between the calculated and the experimental energy levels. This is used to estimate the uncertainty in the fitted parameters. Admittedly this will be a rough estimate (at least on the contribution of the calculated uncertainty), but it is better than nothing and may at least provide a lower bound to the uncertainty in the fitted parameters. It is assumed that the uncertainty in the differences between the calculated and the experimental energy levels is the same for all of them.

1664
 1665 The list `startValues` is a list with all of the parameters needed to define the Hamiltonian (including the initial values for `problemVars`).

1666
 1667 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.

1668
 1669 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.

1670
 1671 The function returns an association with the following keys:
 1672
 1673 - `\"bestRMS\"` which is the best `\[Sigma]` value found.
 1674 - `\"bestParams\"` which is the best set of parameters found for the variables that were not constrained.
 1675 - `\"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.
 1676 - `\"paramSols\"` which is a list of the parameters trajectories during the stepping of the fitting algorithm.
 1677 - `\"timeTaken/s\"` which is the time taken to find the best fit.
 1678 - `\"simplifier\"` which is the simplifier used to simplify the Hamiltonian.
 1679 - `\"excludeDataIndices\"` as given in the input.
 1680 - `\"startValues\"` as given in the input.
 1681
 1682 - `\"freeIonSymbols\"` which are the symbols used in the intermediate coupling basis.
 1683 - `\"truncationEnergy\"` which is the energy used to truncate the Hamiltonian, if it was set to Automatic, the value here is the actual energy used.
 1684 - `\"numE\"` which is the number of electrons in the f^{numE} configuration.
 1685 - `\"expData\"` which is the experimental data used for fitting.
 1686 - `\"problemVars\"` which are the symbols considered for fitting
 1687
 1688 - `\"maxIterations\"` which is the maximum number of iterations used by `NMinimize`.
 1689 - `\"hamDim\"` which is the dimension of the full Hamiltonian.
 1690 - `\"allVars\"` which are all the symbols defining the Hamiltonian under the aggregate simplifications.
 1691 - `\"freeBies\"` which are the free-ion parameters used to define the intermediate coupling basis.

```

1692 - \\"truncatedDim\" which is the dimension of the truncated
      Hamiltonian.
1693 - \\"compiledIntermediateFname\" the file name of the compiled
      function used for the truncated Hamiltonian.
1694
1695 - \\"fittedLevels\" which is the number of levels fitted for.
1696 - \\"actualSteps\" the number of steps that FindMinimun actually
      took.
1697 - \\"solWithUncertainty\" which is a list of replacement rules of the
      form (paramSymbol -> {bestEstimate, uncertainty}).
1698 - \\"rmsHistory\" which is a list of the \[Sigma] values found during
      the fitting.
1699 - \\"Appendix\" which is an association appended to the log file under
      the key \\"Appendix\\".
1700 - \\"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.
1701
1702 - \\"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.
1703 - \\"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.
1704 - \\"degreesOfFreedom\" which is equal to the number of fitted state
      energies minus the number of parameters used in fitting.
1705
1706 The function admits the following options with corresponding default
      values:
1707 - \\"MaxHistory\" : determines how long the logs for the solver can be
      .
1708 - \\"MaxIterations\" : determines the maximum number of iterations used
      by NMinimize.
1709 - \\"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\\".
1710 - \\"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.
1711
1712 - \\"AccuracyGoal\" : the accuracy goal used by NMinimize, default of
      5.
1713 - \\"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.
1714 - \\"PrintFun\" : the function used to print progress messages, the
      default is PrintTemporary.
1715 - \\"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\".
1716
1717 - \\"ProgressView\" : whether or not a progress window will be opened
      to show the progress of the solver, the default is True.
1718 - \\"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.
1719 - \\"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.

```

```

1720
1721 - \\"AppendToLogFile\\": an association appended to the log file under
1722   the key \\\"Appendix\\".
1723 - \\"MagneticSimplifier\\": a list of replacement rules to simplify the
1724   Marvin and pesudo-magnetic paramters. Here the ratios of the
1725   Marvin parameters and the pseudo-magnetic parameters are defined
1726   to simplify the magnetic part of the Hamiltonian.
1727 - \\"MagFieldSimplifier\\": a list of replacement rules to specify a
1728   magnetic field (in T), if set to {}, then {Bx, By, Bz} can also be
1729   used as variables to be fitted for.
1730
1731 - \\"SymmetrySimplifier\\": a list of replacements rules to simplify
1732   the crystal field.
1733 - \\"OtherSimplifier\\": an additional list of replacement rules that
1734   are applied to the Hamiltonian before computing with it. Here the
1735   spin-spin contribution can be turned off by setting \\[Sigma]SS->0,
1736   which is the default.
1737
1738 ";
1739 Options[MostlyOrthogonalFit] = {
1740   "MaxHistory"      -> 200,
1741   "MaxIterations"   -> 100,
1742   "FilePrefix"       -> "calcs",
1743   "ProgressView"    -> True,
1744   "TruncationEnergy" -> Automatic,
1745   "AccuracyGoal"    -> 5,
1746   "PrintFun"         -> PrintTemporary,
1747   "RefParamsVintage" -> "LaF3",
1748   "SignatureCheck"   -> False,
1749   "AddConstantShift" -> False,
1750   "SaveEigenvectors" -> False,
1751   "AppendToLogFile"  -> <||>,
1752   "SaveToLog"        -> False,
1753   "Energy Uncertainty in K" -> Automatic,
1754   "MagneticSimplifier" -> {
1755     M2 -> 56/100 MO,
1756     M4 -> 31/100 MO,
1757     P4 -> 1/2 P2,
1758     P6 -> 1/10 P2
1759   },
1760   "MagFieldSimplifier" -> {
1761     Bx -> 0,
1762     By -> 0,
1763     Bz -> 0
1764   },
1765   "SymmetrySimplifier" -> {
1766     B12->0, B14->0, B16->0, B34->0, B36->0, B56->0,
1767     S12->0 ,S14->0, S16->0, S22->0, S24->0, S26->0,
1768     S34->0 ,S36->0, S44->0, S46->0, S56->0, S66->0
1769   },
1770   "OtherSimplifier" -> {
1771     EOp->0,
1772     P0->0,
1773     \\[Sigma]SS->0 ,
1774     T11p->0,
1775     T12->0,
1776     T14->0,
1777     T15->0,
1778     T16->0,
1779     T18->0,
1780     T17->0,
1781     T19->0,
1782     wChErrA ->0,
1783     wChErrB->0
1784   },
1785   "ThreeBodySimplifier" -> <|
1786     1 -> {
1787       T2->0,
1788       T3->0,
1789       T4->0,
1790       T6->0,
1791       T7->0,
1792       T8->0,
1793       T11p->0,
1794       T12->0,
1795       T14->0,
1796       T15->0,

```

```

1786      T16->0 ,
1787      T18->0 ,
1788      T17->0 ,
1789      T19->0 ,
1790      T2p->0} ,
1791 2 -> {
1792      T2->0 ,
1793      T3->0 ,
1794      T4->0 ,
1795      T6->0 ,
1796      T7->0 ,
1797      T8->0 ,
1798      T11p->0 ,
1799      T12->0 ,
1800      T14->0 ,
1801      T15->0 ,
1802      T16->0 ,
1803      T18->0 ,
1804      T17->0 ,
1805      T19->0 ,
1806      T2p->0
1807  },
1808 3 -> {t2Switch -> 1} ,
1809 4 -> {t2Switch -> 1} ,
1810 5 -> {t2Switch -> 1} ,
1811 6 -> {t2Switch -> 1} ,
1812 7 -> {t2Switch -> 1} ,
1813 8 -> {t2Switch -> 0} ,
1814 9 -> {t2Switch -> 0} ,
1815 10 -> {t2Switch -> 0} ,
1816 11 -> {t2Switch -> 0} ,
1817 12 -> {
1818      t2Switch -> 0 ,
1819      T2->0 ,
1820      T2p->0 ,
1821      T3->0 ,
1822      T4->0 ,
1823      T6->0 ,
1824      T7->0 ,
1825      T8->0 ,
1826      T11p->0 ,
1827      T12->0 ,
1828      T14->0 ,
1829      T15->0 ,
1830      T16->0 ,
1831      T18->0 ,
1832      T17->0 ,
1833      T19->0
1834  },
1835 13->{
1836      t2Switch -> 0 ,
1837      T2->0 ,
1838      T2p->0 ,
1839      T3->0 ,
1840      T4->0 ,
1841      T6->0 ,
1842      T7->0 ,
1843      T8->0 ,
1844      T11p->0 ,
1845      T12->0 ,
1846      T14->0 ,
1847      T15->0 ,
1848      T16->0 ,
1849      T18->0 ,
1850      T17->0 ,
1851      T19->0
1852  }
1853 |> ,
1854 "FreeIonSymbols" -> {E0p, E1p, E2p, E3p,  $\zeta$ }
1855 };
1856 MostlyOrthogonalFit[numE_Integer, expData_List,
1857   excludeDataIndices_List, problemVars_List, startValues_Association
1858   , constraints_List, OptionsPattern[]]:=Module[
1859   {accuracyGoal,
1860    allFreeEnergies,
1861    allFreeEnergiesSorted,

```

```

1860    allVars ,
1861    allVarsVec ,
1862    argsForEvalInsideOfTheIntermediateSystems ,
1863    argsOfTheIntermediateEigensystems ,
1864    aVar ,
1865    aVarPosition ,
1866    basis ,
1867
1868    basisChanger ,
1869    basisChangerBlocks ,
1870    bestParams ,
1871    bestRMS ,
1872    blockShifts ,
1873    blockSizes ,
1874    compiledDiagonal ,
1875    compiledIntermediateFname ,
1876    constrainedProblemVars ,
1877    constrainedProblemVarsList ,
1878
1879    currentRMS ,
1880    degreesOfFreedom ,
1881    dependentVars ,
1882    diagonalBlocks ,
1883    diagonalScalarBlocks ,
1884    diff ,
1885    eigenEnergies ,
1886    eigenvalueDispenserTemplate ,
1887    eigenVectors ,
1888    elevatedIntermediateEigensystems ,
1889
1890    endTime ,
1891    fmSolAssoc ,
1892    freeBies ,
1893    freeLenergiesAndMultiplets ,
1894    fullHam ,
1895    fullSolVec ,
1896    ham ,
1897    hamDim ,
1898    hamEigenvaluesTemplate ,
1899    hamString ,
1900
1901    indepSolVecVec ,
1902    indepVars ,
1903    intermediateHam ,
1904    isolationValues ,
1905    lin ,
1906    linMat ,
1907    ln ,
1908    lnParams ,
1909    logFilePrefix ,
1910    magneticSimplifier ,
1911
1912    maxFreeEnergy ,
1913    maxHistory ,
1914    maxIterations ,
1915    minFreeEnergy ,
1916    minpoly ,
1917    modelSymbols ,
1918    multipletAssignments ,
1919    needlePosition ,
1920    numBlocks ,
1921    openNotebooks ,
1922
1923    ordering ,
1924    otherSimplifier ,
1925    p0 ,
1926    paramBest ,
1927    perHam ,
1928    presentDataIndices ,
1929    PrintFun ,
1930    problemVarsPositions ,
1931    problemVarsQ ,
1932    problemVarsQString ,
1933
1934    problemVarsVec ,
1935    problemVarsWithStartValues ,

```

```

1936    reducedModelSymbols ,
1937    RefParams ,
1938    roundedTruncationEnergy ,
1939    runningInteractive ,
1940    shiftToggle ,
1941    simplifier ,
1942    sol ,
1943    solCompendium ,
1944
1945    solWithUncertainty ,
1946    sortedTruncationIndex ,
1947    sqdiff ,
1948    standardValues ,
1949    startTime ,
1950    states ,
1951    steps ,
1952    symmetrySimplifier ,
1953    theIntermediateEigensystems ,
1954    TheIntermediateEigensystems ,
1955
1956    timeTaken ,
1957    truncatedIntermediateBasis ,
1958    truncatedIntermediateHam ,
1959    truncationEnergy ,
1960    truncationIndices ,
1961    truncationUmbral ,
1962    varHash ,
1963    varsWithConstants ,
1964    \[Lambda]0Vec ,
1965    \[Lambda]exp
1966 },
1967 (
1968     \[Sigma]exp = OptionValue["Energy Uncertainty in K"];
1969     refParamsVintage = OptionValue["RefParamsVintage"];
1970     RefParams = Which[
1971       refParamsVintage === "LaF3",
1972       LoadLaF3Parameters ,
1973       refParamsVintage === "LiYF4",
1974       LoadLiYF4Parameters ,
1975       True ,
1976       refParamsVintage
1977     ];
1978     solCompendium = <||>;
1979     hamDim = Binomial[14, numE];
1980     addShift = OptionValue["AddConstantShift"];
1981     ln = theLanthanides[[numE]];
1982     maxHistory = OptionValue["MaxHistory"];
1983     maxIterations = OptionValue["MaxIterations"];
1984     logFilePrefix = If[OptionValue["FilePrefix"] == "",
1985       ToString[theLanthanides[[numE]]],
1986       OptionValue["FilePrefix"]
1987     ];
1988     accuracyGoal = OptionValue["AccuracyGoal"];
1989     PrintFun = OptionValue["PrintFun"];
1990     freeIonSymbols = OptionValue["FreeIonSymbols"];
1991     runningInteractive = (Head[$ParentLink] === LinkObject);
1992     magneticSimplifier = OptionValue["MagneticSimplifier"];
1993     magFieldSimplifier = OptionValue["MagFieldSimplifier"];
1994     symmetrySimplifier = OptionValue["SymmetrySimplifier"];
1995     otherSimplifier = OptionValue["OtherSimplifier"];
1996     threeBodySimplifier = If[Head[OptionValue["ThreeBodySimplifier"]]
1997       == Association,
1998         OptionValue["ThreeBodySimplifier"][[numE]],
1999         OptionValue["ThreeBodySimplifier"]
2000       ];
2001
2002     truncationEnergy = If[OptionValue["TruncationEnergy"] ===
2003       Automatic ,
2004       (
2005         PrintFun["Truncation energy set to Automatic, using the
2006           maximum energy (+20%) in the data ..."];
2007         Round[1.2 * Max[Select[First /@ expData, NumericQ[#] &]]
2008       ),
2009       OptionValue["TruncationEnergy"]
2010     ];
2011     truncationEnergy = Max[50000, truncationEnergy];

```

```

2009 PrintFun["Using a truncation energy of ", truncationEnergy, " K"]
];
2010
2011 simplifier = Join[magneticSimplifier,
2012                         magFieldSimplifier,
2013                         symmetrySimplifier,
2014                         threeBodySimplifier,
2015                         otherSimplifier];
2016
2017 PrintFun["Determining gaps in the data ..."];
2018 (* whatever is non-numeric is assumed as a known gap *)
2019 presentDataIndices = Flatten[Position[expData, {_?(NumericQ[#] &
2020 , ___]}];
2021 (* some indices omitted here based on the excludeDataIndices
2022 argument, note that presentDataIndices is somewhat a misnomer*)
2023 presentDataIndices = Complement[presentDataIndices,
2024 excludeDataIndices];
2025
2026 solCompendium["simplifier"] = simplifier;
2027 solCompendium["excludeDataIndices"] = excludeDataIndices;
2028 solCompendium["startValues"] = startValues;
2029 solCompendium["freeIonSymbols"] = freeIonSymbols;
2030 solCompendium["truncationEnergy"] = truncationEnergy;
2031 solCompendium["numE"] = numE;
2032 solCompendium["expData"] = expData;
2033 solCompendium["problemVars"] = problemVars;
2034 solCompendium["maxIterations"] = maxIterations;
2035 solCompendium["hamDim"] = hamDim;
2036 solCompendium["constraints"] = constraints;
2037
2038 modelSymbols = Select[paramSymbols,
2039     Not[MemberQ[Join[racahSymbols,
2040                     juddOfeltIntensitySymbols,
2041                     slaterSymbols,
2042                     {\alpha, \beta, \gamma},
2043                     {T2},
2044                     chenSymbols,
2045                     {t2Switch, \[Epsilon], gs, nE}], #]] &
2046 ];
2047 modelSymbols = Sort[modelSymbols];
2048 (* remove the symbols that will be removed by the simplifier, no
2049 symbol should remain here that is not in the symbolic Hamiltonian
2050 *)
2051 reducedModelSymbols = Select[modelSymbols, Not[MemberQ[Keys[
2052 simplifier], #]] &];
2053
2054 (* this is useful to understand what are the arguments of the
2055 truncated compiled Hamiltonian *)
2056 If[OptionValue["SignatureCheck"],
2057 (
2058     PrintFun["Given the model parameters and the simplifying
2059 assumptions, the resultant model parameters are:"];
2060     PrintFun[{reducedModelSymbols}];
2061     PrintFun["Exiting ..."];
2062     Return[];
2063 )
2064 ];
2065
2066 (* calculate the basis *)
2067 PrintFun["Retrieving the LSJM basis for f^", numE, " ..."];
2068 basis = BasisLSJM[numE];
2069
2070 Which[refParamsVintage === Automatic,
2071 (
2072     PrintFun["This is not currently supported, please provide a
2073 vintage for the reference parameters."];
2074     Return[$Failed];
2075 ),
2076 MemberQ[{List, Association}, Head[RefParams]],
2077 (
2078     RefParams = Association[RefParams];
2079     PrintFun["Using the given parameters as a starting point ..."
2080 ];
2081     lnParams = RefParams;
2082     extraParams = FromNonOrthogonalToMostlyOrthogonal[
2083 LoadLaF3Parameters[ln], numE];

```

```

2073     lnParams      = Join[extraParams, lnParams];
2074   ),
2075   True ,
2076   (
2077     (* get the reference parameters from the given vintage *)
2078     PrintFun["Getting reference free-ion parameters for ", ln, " "
2079       using ", refParamsVintage, "..."];
2080     lnParams = ParamPad[FromNonOrthogonalToMostlyOrthogonal[
2081       RefParams[ln], numE],
2082       "PrintFun" -> PrintFun];
2083   );
2084
2085   freeBies = Prepend[Values[(# -> (# /. lnParams)) &/@*
2086     freeIonSymbols], numE];
2087
2088   (* a more explicit alias *)
2089   allVars           = reducedModelSymbols;
2090   numericConstraints = Association@Select[constraints, NumericQ
2091     #[[2]]] & ;
2092   standardValues    = allVars /. Join[lnParams, numericConstraints];
2093
2094   solCompendium["allVars"] = allVars;
2095   solCompendium["freeBies"] = freeBies;
2096
2097   (* reload compiled version if found *)
2098   varHash           = Hash[{numE, allVars, freeBies,
2099     truncationEnergy, simplifier}];
2100   compiledIntermediateFname = ln <> "-compiled-intermediate-
2101     truncated-ham-" <> ToString[varHash] <> ".mx";
2102   compiledIntermediateFname = FileNameJoin[{moduleDir, "compiled",
2103     compiledIntermediateFname}];
2104   solCompendium["compiledIntermediateFname"] =
2105   compiledIntermediateFname;
2106
2107   If[FileExistsQ[compiledIntermediateFname],
2108   (
2109     PrintFun["This ion, free-ion params, and full set of variables
2110       have been used before (as determined by {numE, allVars, freeBies,
2111       truncationEnergy, simplifier}). Loading the previously saved
2112       compiled function and intermediate coupling basis ..."];
2113     PrintFun["Using : ", compiledIntermediateFname];
2114     {compileIntermediateTruncatedHam, truncatedIntermediateBasis} =
2115     Import[compiledIntermediateFname];
2116   ),
2117   (
2118     If[truncationEnergy == Infinity,
2119     (
2120       ham = HamMatrixAssembly[numE,
2121         "ReturnInBlocks" -> False,
2122         "OperatorBasis" -> "MostlyOrthogonal"];
2123       theSimplifier = simplifier;
2124       ham = Normal @ ReplaceInSparseArray[ham, simplifier];
2125       PrintFun["Compiling a function for the Hamiltonian with no
2126         truncation ..."];
2127       (* compile a function that will calculate the truncated
2128         Hamiltonian given the parameters in allVars, this is the function
2129         to be use in fitting *)
2130       compileIntermediateTruncatedHam = Compile[Evaluate[allVars
2131         ], Evaluate[ham]];
2132       truncatedIntermediateBasis =
2133       SparseArray@IdentityMatrix[Binomial[14, numE]];
2134       (* save the compiled function *)
2135       PrintFun["Saving the compiled function for the Hamiltonian
2136         with no truncation and a placeholder intermediate basis ..."];
2137       Export[compiledIntermediateFname, {
2138         compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
2139     ),
2140     (
2141       (* grab the Hamiltonian preserving the block structure *)
2142       PrintFun["Assembling the Hamiltonian for f^", numE, " keeping
2143         the block structure ..."];
2144       ham      = HamMatrixAssembly[numE,
2145         "ReturnInBlocks" -> True,
2146         "OperatorBasis" -> "MostlyOrthogonal"];
2147       (* apply the simplifier *)

```

```

2128     PrintFun["Simplifying using the aggregate set of
2129      simplification rules ..."];
2130      ham           = Map[ReplaceInSparseArray[#, simplifier]&,amp;, ham,
2131      {2}];
2132      PrintFun["Zeroing out every symbol in the Hamiltonian that is
2133      not a free-ion parameter ..."];
2134
2135      (* Get the free ion symbols *)
2136      freeIonSimplifier = (#->0) & /@ Complement[
2137      reducedModelSymbols, freeIonSymbols];
2138
2139      (* Take the diagonal blocks for the intermediate analysis *)
2140      PrintFun["Grabbing the diagonal blocks of the Hamiltonian ..."];
2141      diagonalBlocks = Diagonal[ham];
2142
2143      (* simplify them to only keep the free ion symbols *)
2144      PrintFun["Simplifying the diagonal blocks to only keep the
2145      free ion symbols ..."];
2146      diagonalScalarBlocks = ReplaceInSparseArray[#, freeIonSimplifier] &/@ diagonalBlocks;
2147
2148      (* these include the MJ quantum numbers, remove that *)
2149      PrintFun["Contracting the basis vectors by removing the MJ
2150      quantum numbers from the diagonal blocks ..."];
2151      diagonalScalarBlocks = FreeHam[diagonalScalarBlocks, numE];
2152
2153      argsOfTheIntermediateEigensystems = StringJoin[Riffle
2154      [Prepend[(ToString[#]<>"v_") & /@ freeIonSymbols, "numE_"], ", ", "]];
2155      argsForEvalInsideOfTheIntermediateSystems = StringJoin[Riffle
2156      [(ToString[#]<>"v") & /@ freeIonSymbols, ", ", "]];
2157      PrintFun["argsOfTheIntermediateEigensystems = ",
2158      argsOfTheIntermediateEigensystems];
2159      PrintFun["argsForEvalInsideOfTheIntermediateSystems = ",
2160      argsForEvalInsideOfTheIntermediateSystems];
2161      PrintFun["(if the following fails, it might help to see if
2162      the arguments of TheIntermediateEigensystems match the ones shown
2163      above)"];
2164
2165      (* compile a function that will effectively calculate the
2166      spectrum of all of the scalar blocks given the parameters of the
2167      free-ion part of the Hamiltonian *)
2168      (* compile one function for each of the blocks *)
2169      PrintFun["Compiling functions for the diagonal blocks of the
2170      Hamiltonian ..."];
2171      compiledDiagonal = Compile[Evaluate[freeIonSymbols], Evaluate
2172      [N[Normal[#]]]]& /@ diagonalScalarBlocks;
2173      (* use that to create a function that will calculate the free
2174      -ion eigensystem *)
2175      TheIntermediateEigensystems[numEv_, E0pv_, E1pv_, E2pv_,
2176      E3pv_,  $\zeta$ v_] := (
2177          theNumericBlocks = (#[E0pv, E1pv, E2pv, E3pv,  $\zeta$ v]&) /@
2178          compiledDiagonal;
2179          theIntermediateEigensystems = Eigensystem /@
2180          theNumericBlocks;
2181          Js = AllowedJ[numEv];
2182          basisJ = BasisLSJMJ[numEv, "AsAssociation"->True];
2183          (* having calculated the eigensystems with the removed
2184          degeneracies, put the degeneracies back in explicitly *)
2185          elevatedIntermediateEigensystems = MapIndexed[EigenLever
2186          [#1, 2Js[[#2[[1]]]]+1 ]&, theIntermediateEigensystems];
2187          (* Identify a single MJ to keep *)
2188          pivot = If[EvenQ[numEv], 0, -1/2];
2189          LSJmultiplets = (#[[1]]<>ToString[InputForm[#[[2]]]])&/
2190          @Select[BasisLSJMJ[numEv], #[[{-1}]]== pivot &];
2191          (* calculate the multiplet assignments that the
2192          intermediate basis eigenvectors have *)
2193          needlePosition = 0;
2194          multipletAssignments = Table[
2195          (
2196              J = Js[[idx]];
2197              eigenVecs = theIntermediateEigensystems[[idx]][[2]];
2198              majorComponentIndices = Ordering[Abs[#][[-1]]]&/
2199              eigenVecs;
2200              majorComponentIndices += needlePosition;
2201              needlePosition += Length[

```

```

2177     majorComponentIndices];
2178     majorComponentAssignments      = LSJmultiplets[[#]]&/
2179     @majorComponentIndices;
2180     (* All of the degenerate eigenvectors belong to the
2181     same multiplet*)
2182     elevatedMultipletAssignments = ListRepeater[
2183     majorComponentAssignments, 2J+1];
2184     elevatedMultipletAssignments
2185     ),
2186     {idx, 1, Length[Js]}
2187   ];
2188
2189   (* put together the multiplet assignments and the energies
2190   *)
2191   freeIenergiesAndMultiplets = Transpose/@Transpose[{First/
2192   @elevatedIntermediateEigensystems, multipletAssignments}];
2193   freeIenergiesAndMultiplets = Flatten[
2194   freeIenergiesAndMultiplets, 1];
2195   (* calculate the change of basis matrix using the
2196   intermediate coupling eigenvectors *)
2197   basisChanger = BlockDiagonalMatrix[Transpose/@Last/
2198   @elevatedIntermediateEigensystems];
2199   basisChanger = SparseArray[basisChanger];
2200   Return[{theIntermediateEigensystems,
2201   multipletAssignments,
2202   elevatedIntermediateEigensystems,
2203   freeIenergiesAndMultiplets,
2204   basisChanger}]
2205   );
2206
2207   PrintFun["Calculating the intermediate eigensystems for ",ln,
2208   " using free-ion params from LaF3 ..."];
2209   (* calculate intermediate coupling basis using the free-ion
2210   params for LaF3 *)
2211   {theIntermediateEigensystems, multipletAssignments,
2212   elevatedIntermediateEigensystems, freeIenergiesAndMultiplets,
2213   basisChanger} = TheIntermediateEigensystems@@freeBies;
2214
2215   (* use that intermediate coupling basis to compile a function
2216   for the full Hamiltonian *)
2217   allFreeEnergies = Flatten[First/
2218   @elevatedIntermediateEigensystems];
2219   (* important that the intermediate coupling basis have
2220   attached energies, which make possible the truncation *)
2221   ordering = Ordering[allFreeEnergies];
2222   (* sort the free ion energies and determine which indices
2223   should be included in the truncation *)
2224   allFreeEnergiesSorted          = Sort[allFreeEnergies];
2225   {minFreeEnergy, maxFreeEnergy} = MinMax[allFreeEnergies];
2226   (* determine the index at which the energy is equal or larger
2227   than the truncation energy *)
2228   sortedTruncationIndex = Which[
2229     truncationEnergy > (maxFreeEnergy-minFreeEnergy),
2230     hamDim,
2231     True,
2232     FirstPosition[allFreeEnergiesSorted - Min[
2233     allFreeEnergiesSorted],x_/_;x>truncationEnergy,{0},1][[1]]
2234   ];
2235   (* the actual energy at which the truncation is made *)
2236   roundedTruncationEnergy = allFreeEnergiesSorted[[[
2237   sortedTruncationIndex]]];
2238
2239   (* the indices that participate in the truncation *)
2240   truncationIndices = ordering[;;sortedTruncationIndex];
2241   PrintFun["Computing the block structure of the change of
2242   basis array ..."];
2243   blockSizes           = BlockArrayDimensionsArray[ham];
2244   basisChangerBlocks = ArrayBlocker[basisChanger, blockSizes];
2245   blockShifts         = First /@ Diagonal[blockSizes];
2246   numBlocks           = Length[blockSizes];
2247
2248   (* using the ham (with all the symbols) change the basis to
2249   the computed one *)
2250   PrintFun["Changing the basis of the Hamiltonian to the
2251   intermediate coupling basis ..."];
2252   intermediateHam = BlockMatrixMultiply[ham, basisChangerBlocks

```

```

];
2230   PrintFun["Distributing products inside of symbolic matrix
elements to keep complexity in check ..."];
2231   Do[
2232     intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
intermediateHam[[rowIdx, colIdx]], Distribute /@ # &], ,
2233     {rowIdx, 1, numBlocks},
2234     {colIdx, 1, numBlocks}
2235   ];
2236   intermediateHam = BlockMatrixMultiply[BlockTranspose[
basisChangerBlocks], intermediateHam];
2237   PrintFun["Distributing products inside of symbolic matrix
elements to keep complexity in check ..."];
2238   Do[
2239     intermediateHam[[rowIdx, colIdx]] = MapToSparseArray[
intermediateHam[[rowIdx, colIdx]], Distribute /@ # &], ,
2240     {rowIdx, 1, numBlocks},
2241     {colIdx, 1, numBlocks}
2242   ];
2243   (* using the truncation indices truncate that one *)
2244   PrintFun["Truncating the Hamiltonian ..."];
2245   truncatedIntermediateHam = TruncateBlockArray[intermediateHam
, truncationIndices, blockShifts];
2246   (* these are the basis vectors for the truncated hamiltonian
*)
2247   PrintFun["Saving the truncated intermediate basis ..."];
2248   truncatedIntermediateBasis = basisChanger[[All,
truncationIndices]];

2249
2250   PrintFun["Compiling a function for the truncated Hamiltonian
..."];
2251   (* compile a function that will calculate the truncated
Hamiltonian given the parameters in allVars, this is the function
to be use in fitting *)
2252   compileIntermediateTruncatedHam = Compile[Evaluate[allVars],
Evaluate[truncatedIntermediateHam]];
2253   (* save the compiled function *)
2254   PrintFun["Saving the compiled function for the truncated
Hamiltonian and the truncated intermediate basis ..."];
2255   Export[compiledIntermediateFname, {
compileIntermediateTruncatedHam, truncatedIntermediateBasis}];
2256   )
2257 ];
2258 )
2259 ];

2260
2261 truncationUmbral = Dimensions[truncatedIntermediateBasis][[2]];
2262 PrintFun["The truncated Hamiltonian has a dimension of ",
truncationUmbral, "x", truncationUmbral, "..."];
2263 presentDataIndices = Select[presentDataIndices, # <=
truncationUmbral &];
2264 solCompendium["presentDataIndices"] = presentDataIndices;
2265
2266 (* the problemVars are the symbols that will be fitted for *)

2267
2268 PrintFun["Starting up the fitting process using the Levenberg-
Marquardt method ..."];
2269 (* using the problemVars I need to create the argument list
including _?NumericQ *)
2270 problemVarsQ      = (ToString[#] <> "_?NumericQ") & /@
problemVars;
2271 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
*)
2272 problemVarsPositions = Position[allVars, #][[1, 1]] & /@
problemVars;
2273 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 *)
2274 varsWithConstants      = standardValues;
2275 varsWithConstants[[problemVarsPositions]] = problemVars;
2276 varsWithConstantsString = ToString[
varsWithConstants];

```

```

2279
2280 (* this following function serves eigenvalues from the
2281 Hamiltonian, has memoization so it might grow to use a lot of RAM
2282 *)
2283 Clear[HamSortedEigenvalues];
2284 hamEigenvaluesTemplate = StringTemplate["
2285 HamSortedEigenvalues['problemVarsQ']:=(
2286     ham
2287         = compileIntermediateTruncatedHam@@
2288 varsWithConstants';
2289     eigenValues = Chop[Sort@Eigenvalues@ham];
2290     eigenValues = eigenValues - Min[eigenValues];
2291     HamSortedEigenvalues['problemVarsString'] = eigenValues;
2292     Return[eigenValues]
2293 )"];
2294 hamString = hamEigenvaluesTemplate[<|
2295 "problemVarsQ"      -> problemVarsQString,
2296 "varsWithConstants" -> varsWithConstantsString,
2297 "problemVarsString" -> problemVarsString
2298 |>];
2299 ToExpression[hamString];
2300
2301 (* we also need a function that will pick the i-th eigenvalue,
2302 this seems unnecessary but it's needed to form the right
2303 functional form expected by the Levenberg-Marquardt method *)
2304 eigenvalueDispenserTemplate = StringTemplate["
2305 PartialHamEigenvalues['problemVarsQ'][i_]:=(
2306     eigenVals = HamSortedEigenvalues['problemVarsString'];
2307     eigenVals[[i]]
2308 )
2309 ];
2310 eigenValueDispenserString = eigenvalueDispenserTemplate[<|
2311 "problemVarsQ"      -> problemVarsQString,
2312 "problemVarsString" -> problemVarsString
2313 |>];
2314 ToExpression[eigenValueDispenserString];
2315
2316 PrintFun["Determining the free variables after constraints ..."];
2317 constrainedProblemVars = (problemVars /. constraints);
2318 constrainedProblemVarsList = Variables[constrainedProblemVars];
2319 If[addShift,
2320   PrintFun["Adding a constant shift to the fitting parameters ...";
2321   constrainedProblemVarsList = Append[constrainedProblemVarsList,
2322   \[Epsilon]];
2323 ];
2324
2325 indepVars = Complement[problemVars, #[[1]] & /@ constraints];
2326 stringPartialVars = ToString/@constrainedProblemVarsList;
2327
2328 paramSols = {};
2329 rmsHistory = {};
2330 steps = 0;
2331 problemVarsWithStartValues = KeyValueMap[{#1, #2} &, startValues];
2332 If[addShift,
2333   problemVarsWithStartValues = Append[problemVarsWithStartValues,
2334   {\[Epsilon], 0}];
2335 ];
2336 openNotebooks = If[runningInteractive,
2337   ("WindowTitle"/.NotebookInformation[#]) & /@ Notebooks
2338 [] ,
2339   {}];
2340 If[Not[MemberQ[openNotebooks, "Solver Progress"]] && OptionValue["ProgressView"],
2341   ProgressNotebook["Basic" -> False]
2342 ];
2343 degressOfFreedom = Length[presentDataIndices] - Length[
2344 problemVars] - 1;
2345 PrintFun["Fitting for ", Length[presentDataIndices], " data
2346 points with ", Length[problemVars], " free parameters.", " The
2347 effective degrees of freedom are ", degressOfFreedom, " ..."];
2348
2349 PrintFun["Fitting model to data ..."];
2350 startTime = Now;
2351 shiftToggle = If[addShift, 1, 0];
2352 sol = FindMinimum[
2353   Sum[(expData[[j]][[1]] - (PartialHamEigenvalues @@

```

```

2342 constrainedProblemVars)[j] - shiftToggle * \[Epsilon])^2,
2343 {j, presentDataIndices}],  

2344 problemVarsWithStartValues,  

2345 Method -> "LevenbergMarquardt",  

2346 MaxIterations -> OptionValue["MaxIterations"],  

2347 AccuracyGoal -> OptionValue["AccuracyGoal"],  

2348 StepMonitor :> (
2349 steps += 1;
2350 currentSqSum = Sum[(expData[[j]][[1]] - (
2351 PartialHamEigenvalues @@ constrainedProblemVars)[j] - shiftToggle
2352 * \[Epsilon])^2, {j, presentDataIndices}];
2353 currentRMS = Sqrt[currentSqSum / degressOfFreedom];
2354 paramSols = AddToList[paramSols, constrainedProblemVarsList,
2355 maxHistory];
2356 rmsHistory = AddToList[rmsHistory, currentRMS, maxHistory];
2357 )
2358 ];
2359 endTime = Now;
2360 timeTaken = QuantityMagnitude[endTime - startTime, "Seconds"];
2361 PrintFun["Solution found in ", timeTaken, "s"];
2362
2363 solVec = constrainedProblemVars /. sol[[-1]];
2364 indepSolVec = indepVars /. sol[[-1]];
2365 If[addShift,
2366 \[Epsilon]Best = \[Epsilon]/. sol[[-1]],
2367 \[Epsilon]Best = 0
2368 ];
2369 fullSolVec = standardValues;
2370 fullSolVec[[problemVarsPositions]] = solVec;
2371 PrintFun["Calculating the truncated numerical Hamiltonian
corresponding to the solution ..."];
2372 fullHam = compileIntermediateTruncatedHam @@ fullSolVec;
2373 PrintFun["Calculating energies and eigenvectors ..."];
2374 {eigenEnergies, eigenVectors} = Eigensystem[fullHam];
2375 states = Transpose[{eigenEnergies, eigenVectors}];
2376 states = SortBy[states, First];
2377 eigenEnergies = First /@ states;
2378 PrintFun["Shifting energies to make ground state zero of energy
..."];
2379 eigenEnergies = eigenEnergies - eigenEnergies[[1]];
2380 PrintFun["Calculating the linear approximant to each eigenvalue
..."];
2381 allVarsVec = Transpose[{allVars}];
2382 p0 = Transpose[{fullSolVec}];
2383 linMat = {};
2384 If[addShift,
2385 tail = -2,
2386 tail = -1];
2387 Do[
2388 (
2389 aVarPosition = Position[allVars, aVar][[1, 1]];
2390 isolationValues = ConstantArray[0, Length[allVars]];
2391 isolationValues[[aVarPosition]] = 1;
2392 dependentVars = KeyValueMap[{#1, D[#2, aVar]} &, Association[
2393 constraints]];
2394 Do[
2395 isolationValues[[Position[allVars, dVar[[1]]][[1, 1]]]] =
2396 dVar[[2]],
2397 {dVar, dependentVars}
2398 ];
2399 perHam = compileIntermediateTruncatedHam @@ isolationValues;
2400 lin = FirstOrderPerturbation[Last /@ states, perHam];
2401 linMat = Append[linMat, lin];
2402 ),
2403 {aVar, constrainedProblemVarsList[;;tail]}
2404 ];
2405 PrintFun["Removing the gradient of the ground state ..."];
2406 linMat = (# - #[[1]] & /@ linMat);
2407 PrintFun["Transposing derivative matrices into columns ..."];
2408 linMat = Transpose[linMat];
2409
2410 PrintFun["Calculating the eigenvalue vector at solution ..."];
2411 \[Lambda]0Vec = Transpose[{eigenEnergies[[presentDataIndices]]}];
2412 PrintFun["Putting together the experimental vector ..."];
2413 \[Lambda]exp = Transpose[{First /@ expData[[presentDataIndices
2414 ]]}];

```

```

2408 problemVarsVec = If[addShift,
2409   Transpose[{constrainedProblemVarsList[[;; -2]]}],
2410   Transpose[{constrainedProblemVarsList}]];
2411 ];
2412 indepSolVecVec = Transpose[{indepSolVec}];
2413 PrintFun["Calculating the difference between eigenvalues at
solution ..."];
2414 diff = If[linMat == {},
2415   (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best,
2416   (\[Lambda]0Vec - \[Lambda]exp) + \[Epsilon]Best + linMat[[presentDataIndices]].(problemVarsVec - indepSolVecVec)
];
2417 PrintFun["Calculating the sum of squares of differences around
solution ... "];
2418 sqdiff = Expand[(Transpose[diff] . diff)[[1, 1]]];
2419 PrintFun["Calculating the minimum (which should coincide with sol
) ..."];
2420 minpoly = sqdiff /. AssociationThread[problemVars -> solVec];
2421 fmSolAssoc = Association[sol[[2]]];
2422 If[\[Sigma]exp == Automatic,
2423   \[Sigma]exp = Sqrt[minpoly / degressOfFreedom];
2424 ];
2425 \[CapitalDelta]\[Chi]2 = Sqrt[degressOfFreedom];
2426 Amat = (1/\[Sigma]exp^2) * Transpose[linMat[[presentDataIndices
]]].linMat[[presentDataIndices]];
2427 paramIntervals = EllipsoidBoundingBox[Amat, \[CapitalDelta]\[Chi]
2];
2428 PrintFun["Calculating the uncertainty in the parameters ..."];
2429 solWithUncertainty = Table[
2430 (
2431   aVar = constrainedProblemVarsList[[varIdx]];
2432   paramBest = aVar /. fmSolAssoc;
2433   (aVar -> {paramBest, paramIntervals[[varIdx, 2]]})
);
2434 {varIdx, 1, Length[constrainedProblemVarsList]-shiftToggle}
];
2435
2436 bestRMS = Sqrt[minpoly / degressOfFreedom];
2437 bestParams = sol[[2]];
2438 bestWithConstraints = Association@Join[constraints, bestParams];
2439 bestWithConstraints = bestWithConstraints /. bestWithConstraints;
2440 bestWithConstraints = (# + 0.) & /@ bestWithConstraints;
2441
2442 solCompendium["degreesOfFreedom"] = degressOfFreedom;
2443 solCompendium["solWithUncertainty"] = solWithUncertainty;
2444 solCompendium["truncatedDim"] = truncationUmbral;
2445 solCompendium["fittedLevels"] = Length[presentDataIndices];
2446
2447 solCompendium["actualSteps"] = steps;
2448 solCompendium["bestRMS"] = bestRMS;
2449 solCompendium["problemVars"] = problemVars;
2450 solCompendium["paramSols"] = paramSols;
2451 solCompendium["rmsHistory"] = rmsHistory;
2452 solCompendium["Appendix"] = OptionValue["
AppendToFile"];
2453 solCompendium["timeTaken/s"] = timeTaken;
2454 solCompendium["bestParams"] = bestParams;
2455 solCompendium["bestParamsWithConstraints"] = bestWithConstraints;
2456
2457 If[OptionValue["SaveEigenvectors"],
2458   solCompendium["states"] = {#[[1]] + \[Epsilon]Best, #[[2]]}
2459 &/@ (Chop /@ ShiftedLevels[states]),
2460   (
2461     finalEnergies = Sort[First /@ states];
2462     finalEnergies = finalEnergies - finalEnergies[[1]];
2463     finalEnergies = finalEnergies + \[Epsilon]Best;
2464     finalEnergies = Chop /@ finalEnergies;
2465     solCompendium["energies"] = finalEnergies;
2466   )
];
2467
2468 If[OptionValue["SaveToLog"],
2469   PrintFun["Saving the solution to the log file ..."];
2470   LogSol[solCompendium, logFilePrefix];
2471 ];
2472 PrintFun["Finished ..."];
2473

```

```

2474     Return[solCompendium];
2475   )
2476 ];
2477
2478 caseConstraints::usage="This Association contains the constraints
2479   that are not the same across all of the lanthanides. For instance,
2480   since the ratio between M2 and M0 is assumed the same for all the
2481   trivalent lanthanides, that one is not included here.
2482 This association has keys equal to symbols of lanthanides and values
2483   equal to lists of rules that express either a parameter being held
2484   fixed or made proportional to another.
2485 In Table I of Carnall 1989 these correspond to cases where values are
2486   given in square brackets.";
2487 caseConstraints = <|
2488 "Ce" -> {
2489   B02 -> -218.,
2490   B04 -> 738.,
2491   B06 -> 679.,
2492   B22 -> -50.,
2493   B24 -> 431.,
2494   B26 -> -921.,
2495   B44 -> 616.,
2496   B46 -> -348.,
2497   B66 -> -788.
2498 },
2499 "Pr" -> {},
2500 "Nd" -> {},
2501 "Sm" -> {
2502   B22 -> -50.,
2503   T2 -> 300.,
2504   T3 -> 36.,
2505   T4 -> 56.,
2506   γ -> 1500.
2507 },
2508 "Eu" -> {
2509   F4 -> 0.713 F2,
2510   F6 -> 0.512 F2,
2511   B22 -> -50.,
2512   B24 -> 597.,
2513   B26 -> -706.,
2514   B44 -> 408.,
2515   B46 -> -508.,
2516   B66 -> -692.,
2517   M0 -> 2.1,
2518   P2 -> 360.,
2519   T2 -> 300.,
2520   T3 -> 40.,
2521   T4 -> 60.,
2522   T6 -> -300.,
2523   T7 -> 370.,
2524   T8 -> 320.,
2525   α -> 20.16,
2526   β -> -566.9,
2527   γ -> 1500.
2528 },
2529 "Pm" -> {
2530   B02 -> -245.,
2531   B04 -> 470.,
2532   B06 -> 640.,
2533   B22 -> -50.,
2534   B24 -> 525.,
2535   B26 -> -750.,
2536   B44 -> 490.,
2537   B46 -> -450.,
2538   B66 -> -760.,
2539   F2 -> 76400.,
2540   F4 -> 54900.,
2541   F6 -> 37700.,
2542   M0 -> 2.4,
2543   P2 -> 275.,
2544   T2 -> 300.,
2545   T3 -> 35.,
2546   T4 -> 58.,
2547   T6 -> -310.,
2548   T7 -> 350.,
2549   T8 -> 320.,
2550 }
```

```

2544       $\alpha \rightarrow 20.5$ ,
2545       $\beta \rightarrow -560.$ ,
2546       $\gamma \rightarrow 1475.$ ,
2547       $\zeta \rightarrow 1025.$ },
2548 "Gd" -> {
2549     F4 -> 0.710 F2,
2550     B02 -> -231.,
2551     B04 -> 604.,
2552     B06 -> 280.,
2553     B22 -> -99.,
2554     B24 -> 340.,
2555     B26 -> -721.,
2556     B44 -> 452.,
2557     B46 -> -204.,
2558     B66 -> -509.,
2559     T2 -> 300.,
2560     T3 -> 42.,
2561     T4 -> 62.,
2562     T6 -> -295.,
2563     T7 -> 350.,
2564     T8 -> 310.,
2565      $\beta \rightarrow -600.$ ,
2566      $\gamma \rightarrow 1575.$ .
2567   },
2568 "Tb" -> {
2569     F4 -> 0.707 F2,
2570     T2 -> 320.,
2571     T3 -> 40.,
2572     T4 -> 50.,
2573      $\gamma \rightarrow 1650.$ .
2574   },
2575 "Dy" -> {},
2576 "Ho" -> {
2577     B02 -> -240.,
2578     T2 -> 400.,
2579      $\gamma \rightarrow 1800.$ .
2580   },
2581 "Er" -> {
2582     T2 -> 400.,
2583      $\gamma \rightarrow 1800.$ .
2584   },
2585 "Tm" -> {
2586     T2 -> 400.,
2587      $\gamma \rightarrow 1820.$ .
2588   },
2589 "Yb" -> {
2590     B02 -> -249.,
2591     B04 -> 457.,
2592     B06 -> 282.,
2593     B22 -> -105.,
2594     B24 -> 320.,
2595     B26 -> -482.,
2596     B44 -> 428.,
2597     B46 -> -234.,
2598     B66 -> -492.
2599   }
2600 |>;
2601
2602 variedSymbols =<|
2603   "Ce" -> { $\zeta$ },
2604   "Pr" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2605     F2, F4, F6,
2606     M0, P2,
2607      $\alpha, \beta, \gamma,$ 
2608      $\zeta$ },
2609   "Nd" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2610     F2, F4, F6,
2611     M0, P2,
2612     T2, T3, T4, T6, T7, T8,
2613      $\alpha, \beta, \gamma,$ 
2614      $\zeta$ },
2615   "Pm" -> {},
2616   "Sm" -> {B02, B04, B06, B24, B26, B44, B46, B66,
2617     F2, F4, F6, M0, P2,
2618     T6, T7, T8,
2619      $\alpha, \beta, \zeta$ },

```

```

2620 "Eu" -> {B02, B04, B06,
2621     F2, F4, F6,  $\zeta$ },
2622 "Gd" -> {F2, F4, F6,
2623     M0, P2,
2624      $\alpha$ ,  $\zeta$ },
2625 "Tb" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2626     F2, F4, F6,
2627     M0, P2,
2628     T6, T7, T8,
2629      $\alpha$ ,  $\beta$ ,  $\zeta$ },
2630 "Dy" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2631     F2, F4, F6,
2632     M0, P2,
2633     T2, T3, T4, T6, T7, T8,
2634      $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\zeta$ },
2635 "Ho" -> {B04, B06, B22, B24, B26, B44, B46, B66,
2636     F2, F4, F6,
2637     M0, P2,
2638     T3, T4, T6, T7, T8,
2639      $\alpha$ ,  $\beta$ ,  $\zeta$ },
2640 "Er" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2641     F2, F4, F6,
2642     M0, P2,
2643     T3, T4, T6, T7, T8,  $\alpha$ ,  $\beta$ ,  $\zeta$ },
2644 "Tm" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2645     F2, F4, F6,
2646     M0, P2,
2647      $\alpha$ ,  $\beta$ ,  $\zeta$ },
2648 "Yb" -> { $\zeta$ }
2649 |>;
2650
2651 caseConstraintsM0::usage="This association contains the symbols that
2652     are held constant in fitting different ions. It only contains
2653     symbols for which the constraint doesn't apply to all ions.
2654 This association has keys equal to symbols of lanthanides and values
2655     equal to associations with the symbols for the parameters that are
2656     held fixed or made proportional to another. If the value is to be
2657     held constant a placeholder value of 0 is given, if a ratio
2658     constraint is given, the value is constraint.
2659 The operator basis for which this is applicable is the mostly-
2660     orthogonal basis.
2661 ";
2662 caseConstraintsM0 = <|
2663 "Ce" -> {
2664     B02 -> 0,
2665     B04 -> 0,
2666     B06 -> 0,
2667     B22 -> 0,
2668     B24 -> 0,
2669     B26 -> 0,
2670     B44 -> 0,
2671     B46 -> 0,
2672     B66 -> 0
2673 },
2674 "Pr" -> {},
2675 "Nd" -> {},
2676 "Sm" -> {
2677     B22 -> 0,
2678     T2p -> 0,
2679     T3 -> 0,
2680     T4 -> 0,
2681      $\gamma$ p -> 0
2682 },
2683 "Eu" -> {
2684     E2p -> 0.0049 E1p,
2685     E3p -> 0.098 E1p,
2686     B22 -> 0,
2687     B24 -> 0,
2688     B26 -> 0,
2689     B44 -> 0,
2690     B46 -> 0,
2691     B66 -> 0,
2692     M0 -> 0,
2693     P2 -> 0,
2694     T2p -> 0,
2695     T3 -> 0,
2696 }
```

```

2689      T4    -> 0 ,
2690      T6    -> 0 ,
2691      T7    -> 0 ,
2692      T8    -> 0 ,
2693       $\alpha p$  -> 0 ,
2694       $\beta p$  -> 0 ,
2695       $\gamma p$  -> 0
2696      } ,
2697  "Pm" -> {
2698      B02   -> 0 ,
2699      B04   -> 0 ,
2700      B06   -> 0 ,
2701      B22   -> 0 ,
2702      B24   -> 0 ,
2703      B26   -> 0 ,
2704      B44   -> 0 ,
2705      B46   -> 0 ,
2706      B66   -> 0 ,
2707      E1p   -> 0 ,
2708      E2p   -> 0 ,
2709      E3p   -> 0 ,
2710      M0    -> 0 ,
2711      P2    -> 0 ,
2712      T2p   -> 0 ,
2713      T3    -> 0 ,
2714      T4    -> 0 ,
2715      T6    -> 0 ,
2716      T7    -> 0 ,
2717      T8    -> 0 ,
2718       $\alpha p$  -> 0 ,
2719       $\beta p$  -> 0 ,
2720       $\gamma p$  -> 0 ,
2721       $\zeta$  -> 0
2722      } ,
2723  "Gd" -> {
2724      E2p   -> 0.0049 E1p ,
2725      B02   -> 0 ,
2726      B04   -> 0 ,
2727      B06   -> 0 ,
2728      B22   -> 0 ,
2729      B24   -> 0 ,
2730      B26   -> 0 ,
2731      B44   -> 0 ,
2732      B46   -> 0 ,
2733      B66   -> 0 ,
2734      T2p   -> 0 ,
2735      T3    -> 0 ,
2736      T4    -> 0 ,
2737      T6    -> 0 ,
2738      T7    -> 0 ,
2739      T8    -> 0 ,
2740       $\beta p$  -> 0 ,
2741       $\gamma p$  -> 0
2742      } ,
2743  "Tb" -> {
2744      E2p   -> 0.0049 E1p ,
2745      T2p   -> 0 ,
2746      T3    -> 0 ,
2747      T4    -> 0 ,
2748       $\gamma p$  -> 0
2749      } ,
2750  "Dy" -> {} ,
2751  "Ho" -> {
2752      B02   -> 0 ,
2753      T2p   -> 0 ,
2754       $\gamma p$  -> 0
2755      } ,
2756  "Er" -> {
2757      T2p   -> 0 ,
2758       $\gamma p$  -> 0
2759      } ,
2760  "Tm" -> {
2761       $\gamma p$  -> 0
2762      } ,
2763  "Yb" -> {
2764      B02   -> 0 ,

```

```

2765     B04 -> 0,
2766     B06 -> 0,
2767     B22 -> 0,
2768     B24 -> 0,
2769     B26 -> 0,
2770     B44 -> 0,
2771     B46 -> 0,
2772     B66 -> 0
2773 }
2774 |>;
2775
2776 variedSymbolsMO = <|
2777     "Ce" -> { $\zeta$ },
2778     "Pr" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2779         E1p, E2p, E3p,
2780         M0, P2,
2781          $\alpha_p$ ,  $\beta_p$ ,  $\gamma_p$ ,
2782          $\zeta$ },
2783     "Nd" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2784         E1p, E2p, E3p,
2785         M0, P2,
2786         T2p, T3, T4, T6, T7, T8,
2787          $\alpha_p$ ,  $\beta_p$ ,  $\gamma_p$ ,
2788          $\zeta$ },
2789     "Pm" -> {},
2790     "Sm" -> {B02, B04, B06, B24, B26, B44, B46, B66,
2791         E1p, E2p, E3p, M0, P2,
2792         T6, T7, T8,
2793          $\alpha_p$ ,  $\beta_p$ ,  $\zeta$ },
2794     "Eu" -> {B02, B04, B06,
2795         E1p, E2p, E3p,  $\zeta$ },
2796     "Gd" -> {E1p, E2p, E3p,
2797         M0, P2,
2798          $\alpha_p$ ,  $\zeta$ },
2799     "Tb" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2800         E1p, E2p, E3p,
2801         M0, P2,
2802         T6, T7, T8,
2803          $\alpha_p$ ,  $\beta_p$ ,  $\zeta$ },
2804     "Dy" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2805         E1p, E2p, E3p,
2806         M0, P2,
2807         T2p, T3, T4, T6, T7, T8,
2808          $\alpha_p$ ,  $\beta_p$ ,  $\gamma_p$ ,  $\zeta$ },
2809     "Ho" -> {B04, B06, B22, B24, B26, B44, B46, B66,
2810         E1p, E2p, E3p,
2811         M0, P2,
2812         T3, T4, T6, T7, T8,
2813          $\alpha_p$ ,  $\beta_p$ ,  $\zeta$ },
2814     "Er" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2815         E1p, E2p, E3p,
2816         M0, P2,
2817         T3, T4, T6, T7, T8,  $\alpha_p$ ,  $\beta_p$ ,  $\zeta$ },
2818     "Tm" -> {B02, B04, B06, B22, B24, B26, B44, B46, B66,
2819         E1p, E2p, E3p,
2820         M0, P2,
2821          $\alpha_p$ ,  $\beta_p$ ,  $\zeta$ },
2822     "Yb" -> { $\zeta$ }
2823 |>;
2824
2825 caseConstraintsLiYF4 = <|
2826     "Ce" -> {
2827         B04 -> -1043.,
2828         B44 -> -1249.,
2829         B06 -> -65.,
2830         B46 -> -1069.
2831     },
2832     "Pr" -> {
2833          $\beta$  -> -644.,
2834          $\gamma$  -> 1413.,
2835         M0 -> 1.88,
2836         P2 -> 244.
2837     },
2838     "Nd" -> {
2839         M0 -> 1.85
2840     },

```

```

2841 "Sm" -> {
2842   α -> 20.5,
2843   β -> -616.,
2844   γ -> 1565.,
2845   T2 -> 282.,
2846   T3 -> 26.,
2847   T4 -> 71.,
2848   T6 -> -257.,
2849   T7 -> 314.,
2850   T8 -> 328.,
2851   M0 -> 2.38,
2852   P2 -> 336.
2853 },
2854 "Eu" -> {
2855   T2 -> 370.,
2856   T3 -> 40.,
2857   T4 -> 40.,
2858   T6 -> -300.,
2859   T7 -> 380.,
2860   T8 -> 370.
2861 },
2862 "Tb" -> {
2863   F4 -> 0.709 F2,
2864   F6 -> 0.503 F2,
2865   α -> 17.6,
2866   β -> -581.,
2867   γ -> 1792.,
2868   T2 -> 330.,
2869   T3 -> 40.,
2870   T4 -> 45.,
2871   T6 -> -365.,
2872   T7 -> 320.,
2873   T8 -> 349.,
2874   M0 -> 2.7,
2875   P2 -> 482.
2876 },
2877 "Dy" -> {
2878   (* F4 -> 0.707 F2,
2879   F6 -> 0.516 F2, *)
2880   F2 -> 90421,
2881   F4 -> 63928,
2882   F6 -> 46657,
2883   α -> 17.9,
2884   β -> -628.,
2885   γ -> 1790.,
2886   T2 -> 326.,
2887   T3 -> 23.,
2888   T4 -> 83.,
2889   T6 -> -294.,
2890   T7 -> 403.,
2891   T8 -> 340.,
2892   M0 -> 4.46,
2893   P2 -> 610.,
2894   B46 -> -700.
2895 },
2896 "Ho" -> {
2897   α -> 17.2,
2898   β -> -596.,
2899   γ -> 1839.,
2900   T2 -> 365.,
2901   T3 -> 37.,
2902   T4 -> 95.,
2903   T6 -> -274.,
2904   T7 -> 331.,
2905   T8 -> 343.,
2906   P2 -> 582.
2907 },
2908 "Er" -> {},
2909 "Tm" -> {
2910   α -> 17.3,
2911   β -> -665.,
2912   γ -> 1936.,
2913   M0 -> 4.93,
2914   P2 -> 730.,
2915   T2 -> 400.
2916 }

```

```

2917 "Yb" -> {
2918     B06 -> -23.,
2919     B46 -> -512.
2920 }
2921 |>;
2922
2923 variedSymbolsLiYF4 = <|
2924     "Ce" -> {
2925         B02,  $\zeta$ 
2926     },
2927     "Pr" -> {
2928         B02, B04, B06, B44, B46,
2929         F2, F4, F6,
2930          $\alpha$ ,  $\zeta$ 
2931     },
2932     "Nd" -> {
2933         B02, B04, B06, B44, B46,
2934         F2, F4, F6,
2935         P2,
2936         T2, T3, T4, T6, T7, T8,
2937          $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\zeta$ 
2938     },
2939     "Sm" -> {
2940         B02, B04, B06, B44, B46,
2941         F2, F4, F6,
2942          $\zeta$ 
2943     },
2944     "Eu" -> {
2945         B02, B04, B06, B44, B46,
2946         F2, F4, F6,
2947         M0, P2,
2948          $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\zeta$ 
2949     },
2950     "Tb" -> {
2951         B02, B04, B06, B44, B46,
2952         F2, F4, F6,
2953          $\zeta$ 
2954     },
2955     "Dy" -> {
2956         B02, B04, B06, B44,
2957         F2, F4, F6,
2958          $\zeta$ 
2959     },
2960     "Ho" -> {
2961         B02, B04, B06, B44, B46,
2962         F2, F4, F6,
2963         M0,
2964          $\zeta$ 
2965     },
2966     "Er" -> {
2967         B02, B04, B06, B44, B46,
2968         F2, F4, F6,
2969         M0, P2,
2970         T2, T3, T4, T6, T7, T8,
2971          $\alpha$ ,  $\beta$ ,  $\gamma$ ,
2972          $\zeta$ 
2973     },
2974     "Tm" -> {
2975         B02, B04, B06, B44, B46,
2976         F2, F4, F6,
2977          $\zeta$ 
2978     },
2979     "Yb" -> {
2980         B02, B04, B44,
2981          $\zeta$ 
2982     }
2983 |>
2984
2985 paramsChengLiYF4::usage="This association has the model parameters as
2986 fitted by Cheng et. al \"Crystal-field analyses for trivalent
2987 lanthanide ions in LiYF4\".";
2988 paramsChengLiYF4 = <|
2989     "Ce" -> {
2990          $\zeta$  -> 630.,
2991         B02 -> 354., B04 -> -1043.,
2992         B44 -> -1249., B06 -> -65.,
2993

```

```

2991     B46 -> -1069.
2992   },
2993 "Pr" -> {
2994   F2 -> 68955., F4 -> 50505., F6 -> 33098.,
2995    $\zeta$  -> 748.,
2996    $\alpha$  -> 23.3,  $\beta$  -> -644.,  $\gamma$  -> 1413.,
2997   M0 -> 1.88, P2 -> 244.,
2998   B02 -> 512., B04 -> -1127.,
2999   B44 -> -1239., B06 -> -85.,
3000   B46 -> -1205.
3001 },
3002 "Nd" -> {
3003   F2 -> 72952., F4 -> 52681., F6 -> 35476.,
3004    $\zeta$  -> 877.,
3005    $\alpha$  -> 21.,  $\beta$  -> -579.,  $\gamma$  -> 1446.,
3006   T2 -> 210., T3 -> 41., T4 -> 74., T6 -> -293., T7 -> 321., T8 ->
3007   205.,
3008   M0 -> 1.85, P2 -> 304.,
3009   B02 -> 391., B04 -> -1031.,
3010   B44 -> -1271., B06 -> -28.,
3011   B46 -> -1046.
3012 },
3013 "Sm" -> {
3014   F2 -> 79515., F4 -> 56766., F6 -> 40078.,
3015    $\zeta$  -> 1168.,
3016    $\alpha$  -> 20.5,  $\beta$  -> -616.,  $\gamma$  -> 1565.,
3017   T2 -> 282., T3 -> 26., T4 -> 71., T6 -> -257., T7 -> 314., T8 ->
3018   328.,
3019   M0 -> 2.38, P2 -> 336.,
3020   B02 -> 370., B04 -> -757.,
3021   B44 -> -941., B06 -> -67.,
3022   B46 -> -895.
3023 },
3024 "Eu" -> {
3025   F2 -> 82573., F4 -> 59646., F6 -> 43203.,
3026    $\zeta$  -> 1329.,
3027    $\alpha$  -> 21.6,  $\beta$  -> -482.,  $\gamma$  -> 1140.,
3028   T2 -> 370., T3 -> 40., T4 -> 40., T6 -> -300., T7 -> 380., T8 ->
3029   370.,
3030   M0 -> 2.41, P2 -> 332.,
3031   B02 -> 339., B04 -> -733.,
3032   B44 -> -1067., B06 -> -36.,
3033   B46 -> -764.
3034 },
3035 "Tb" -> {
3036   F2 -> 90972., F4 -> 64499., F6 -> 45759.,
3037    $\zeta$  -> 1702.,
3038    $\alpha$  -> 17.6,  $\beta$  -> -581.,  $\gamma$  -> 1792.,
3039   T2 -> 330., T3 -> 40., T4 -> 45., T6 -> -365., T7 -> 320., T8 ->
3040   349.,
3041   M0 -> 2.7, P2 -> 482.,
3042   B02 -> 413., B04 -> -867.,
3043   B44 -> -1114., B06 -> -41.,
3044   B46 -> -736.
3045 },
3046 "Dy" -> {
3047   F0 -> 0,
3048   F2 -> 90421., F4 -> 63928., F6 -> 46657.,
3049    $\zeta$  -> 1895.,
3050    $\alpha$  -> 17.9,  $\beta$  -> -628.,  $\gamma$  -> 1790.,
3051   T2 -> 326., T3 -> 23., T4 -> 83., T6 -> -294., T7 -> 403., T8 ->
3052   340.,
3053   M0 -> 4.46, P2 -> 610.,
3054   B02 -> 360., B04 -> -737.,
3055   B44 -> -943., B06 -> -35.,
3056   B46 -> -700.
3057 },
3058 "Ho" -> {
3059   F2 -> 93512., F4 -> 66084., F6 -> 49765.,
3060    $\zeta$  -> 2126.,
3061    $\alpha$  -> 17.2,  $\beta$  -> -596.,  $\gamma$  -> 1839.,
3062   T2 -> 365., T3 -> 37., T4 -> 95., T6 -> -274., T7 -> 331., T8 ->
3063   343.,
3064   M0 -> 3.92, P2 -> 582.,
3065   B02 -> 386., B04 -> -629.,
3066   B44 -> -841., B06 -> -33.,
3067 }

```

```

3061      B46 -> -687.
3062    },
3063 "Er" -> {
3064   F2 -> 97326., F4 -> 67987., F6 -> 53651.,
3065   ζ -> 2377.,
3066   α -> 18.1, β -> -599., γ -> 1870.,
3067   T2 -> 380., T3 -> 41., T4 -> 69., T6 -> -356., T7 -> 239., T8 ->
3068   390.,
3069   M0 -> 4.41, P2 -> 795.,
3070   B02 -> 325., B04 -> -749.,
3071   B44 -> -1014., B06 -> -19.,
3072   B46 -> -635.
3073 },
3074 "Tm" -> {
3075   F0 -> 0.,
3076   T2 -> 0.,
3077   F2 -> 101938., F4 -> 71553., F6 -> 51359.,
3078   ζ -> 2632.,
3079   α -> 17.3, β -> -665., γ -> 1936.,
3080   M0 -> 4.93, P2 -> 730.,
3081   B02 -> 339., B04 -> -627.,
3082   B44 -> -913., B06 -> -39.,
3083   B46 -> -584.
3084 },
3085 "Yb" -> {
3086   ζ -> 2916.,
3087   B02 -> 446., B04 -> -560.,
3088   B44 -> -843., B06 -> -23.,
3089   B46 -> -512.
3090 }
3091 |>
3092 StringToSLJ[string_] := Module[
3093   {stringed = string, LS, J, LSindex},
3094   (
3095     If[StringContainsQ[stringed, "+"],
3096       Return["mixed"]
3097     ];
3098     LS = StringTake[stringed, {1, 2}];
3099     If[StringContainsQ[stringed, "("],
3100       (
3101         LSindex =
3102           StringCases[stringed, RegularExpression["\\((([^])*\\))"] :> "$1"
3103       ];
3104       LS = LS <> LSindex;
3105       stringed = StringSplit[stringed, ")"][[{-1}]];
3106       J = ToExpression[stringed];
3107     ),
3108     J = ToExpression@StringTake[stringed, {3, -1}];
3109   )
3110 ];
3111 {LS, J}
3112 )
3113 ];
3114
3115 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:
3116 - F4F6_SlaterRatios: a list of two numbers that represent the ratio
of F4 to F2 and F6 to F2, respectively.
3117 - PrintFun: a function that will be used to print the progress of
the fitting process.
3118 - MaxIterations: the maximum number of iterations that the fitting
process will run.
3119 - MaxMultiplets: the maximum number of multiplets that will be used
in the fitting process.
3120 - MaxPercent: the maximum percentage of the data that can be off by
the fitting.
3121 - SubSetBounds: a list of two numbers that represent the minimum
and maximum number of multiplets that will be used in the fitting
process.
3122 The function returns an association with the following keys:
3123 - bestParams: the best parameters found in the fitting.
3124 - worstRelativeError: the worst relative error in the fitting.
3125 - SlaterRatios: the Slater ratios used in the fitting.

```

```

3126 - usedBaricenters: the baricenters used in the fitting.
3127 If no acceptable solution is found, the function will return all
3128   solutions that are not worse than 10*MaxPercent. A solution is
3129   acceptable if the worst relative error is less than the MaxPercent
3130   option.
3131 ";
3132 Options[FreeIonSolver] = {
3133 "F4F6_SlaterRatios" -> {0.707, 0.516},
3134 "PrintFun" -> PrintTemporary,
3135 "MaxIterations" -> 10000,
3136 "MaxMultiplets" -> 12,
3137 "MaxPercent" -> 3.,
3138 "SubSetBounds" -> {5, 12}
3139 };
3140 FreeIonSolver[expData_, numE_, OptionsPattern[]] := Module[
3141 (* {maxMultiplets, maxPercent, F4overF2, F6overF2, PrintFun,
3142   minSubSetSize, maxSubSetSize, multipletEnergies, numMultiplets,
3143   allEqns, subsetSizes, ln, solutions, subsets, subset, eqns, m, b,
3144   meritFun, sol, goodThings, bestThings, bestOfAll, finalSol,
3145   usedMultiplets, usedBaricenters}, *)
3146 {},
3147 (
3148   maxMultiplets = OptionValue["MaxMultiplets"];
3149   maxIterations = OptionValue["MaxIterations"];
3150   maxPercent = OptionValue["MaxPercent"];
3151   F4overF2 = OptionValue["F4F6_SlaterRatios"][[1]];
3152   F6overF2 = OptionValue["F4F6_SlaterRatios"][[2]];
3153   PrintFun = OptionValue["PrintFun"];
3154   minSubSetSize = OptionValue["SubSetBounds"][[1]];
3155   maxSubSetSize = OptionValue["SubSetBounds"][[2]];
3156   freeIonParams = {F0, F2, F4, F6, \[Zeta]};
3157   ln = theLanthanides[[numE]];
3158 
3159   PrintFun["Parsing the barycenters of the different multiplets ..."];
3160   multipletEnergies = Map[First, #] & /@ GroupBy[expData, #[[2]] &];
3161   multipletEnergies = Mean[Select[#, NumberQ]] & /@ multipletEnergies;
3162   multipletEnergies = Select[multipletEnergies, FreeQ[#, Mean] &];
3163   multipletEnergies = KeySelect[KeyMap[StringToSLJ, multipletEnergies], # != "mixed" &];
3164   multipletEnergies = KeyMap[Prepend[#, numE] &, multipletEnergies];
3165   numMultiplets = Length[multipletEnergies];
3166 
3167   PrintFun["Composing the system of equations for the free-ion energies ..."];
3168   allEqns = KeyValueMap[FreeIonTable[#1] == #2 &, multipletEnergies];
3169   allEqns = Append[Coefficient[#[[1]], {F0, F2, F4, F6, \[Zeta]}], #[[2]]] & /@ allEqns;
3170   allEqns = allEqns[;; Min[Length[allEqns], maxMultiplets]];
3171   subsetSizes = Range[1, numMultiplets];
3172   numSubsets = {#, Binomial[numMultiplets, #]} & /@ subsetSizes;
3173   numSubsets = Transpose@SortBy[numSubsets, Last];
3174   accSizes = Accumulate[numSubsets[[2]]];
3175   numSubsets = Transpose@Append[numSubsets, accSizes];
3176   lastSub = SelectFirst[numSubsets, #[[3]] > 1000 &, Last];
3177   numSubsets = numSubsets[[;; lastSub]];
3178   lastPosition = Position[numSubsets, lastSub][[1, 1]];
3179   chosenSubsetSizes = #[[1]] & /@ numSubsets[[;; lastPosition]];
3180   solutions = <||>;
3181 
3182   PrintFun["Selecting subsets of different lengths and fitting with ratio-constraints ..."];
3183   Do[
3184     subsets = Subsets[Range[1, Length[allEqns]], {subsetSize}];
3185     PrintFun["Considering ", Length[subsets], " barycenter subsets of size ", subsetSize, " ..."];
3186     Do[
3187       (
3188         subset = subsets[[subsetIndex]];
3189         eqns = allEqns[[subset]];
3190         m = #[[;; 5]] & /@ eqns;
3191         b = #[[6]] & /@ eqns;
3192       );
3193     ];
3194   ];
3195 
```

```

3184     meritFun = Max[100 * Expand[Abs[(m . freeIonParams - b)]/b]];
3185     sol = NMinimize[{meritFun,
3186       F0 > 0,
3187       F2 > 0,
3188       F4 == F4overF2 * F2,
3189       F6 == F6overF2 * F2,
3190       ζ > 0},
3191       freeIonParams,
3192       MaxIterations -> maxIterations,
3193       Method -> "Convex"];
3194     solutions[{subsetSize, subset}] = sol;
3195   )
3196   , {subsetIndex, 1, Length[subsets]}
3197 ],
3198 {subsetSize, chosenSubsetSizes}
3199 ];
3200
3201 PrintFun["Collecting solutions of different subset size ..."];
3202 goodThings = Table[Normal[Sort[KeySelect[#[[1]] & /@ solutions, #[[1]] == subSize &]]][[1]],
3203 {subSize, subsetSizes}];
3204
3205
3206 PrintFun["Picking the solutions that are not worse than ",
3207 maxPercent, "% ..."];
3208 bestThings = Select[goodThings, #[[2]] <= maxPercent &];
3209 If[bestThings == {},
3210   Print["No acceptable solution found, consider increasing
maxPercent or inspecting the given data ..."];
3211   Return[goodThings];
3212 ];
3213
3214 PrintFun["Keeping the solution with the largest number of used
barycenters ..."];
3215 bestOfAll = bestThings[[-1]];
3216 sol = solutions[bestOfAll[[1]]];
3217 subset = bestOfAll[[1, 2]];
3218 eqns = allEqns[[subset]];
3219 m = #[[;; 5]] & /@ eqns;
3220 b = #[[6]] & /@ eqns;
3221 usedMultiplets = Keys[multipletEnergies][[subset]];
3222 usedBaricenters = {#, multipletEnergies[#]} & /@ usedMultiplets;
3223 uniqueLS = DeleteDuplicates[#[[2]] &/@ Keys[multipletEnergies]];
3224 solAssoc = Association[sol[[2]]];
3225 usedLaF3 = False;
3226 If[Length[uniqueLS] == 1,
3227 (
3228   Print["There is too little data to find Slater parameters,
using the ones for LaF3, and keeping the fitted spin-orbit zeta
..."];
3229   laf3params = LoadLaF3Parameters[ln];
3230   usedLaF3 = True;
3231   solAssoc[F0] = laf3params[F0];
3232   solAssoc[F2] = laf3params[F2];
3233   solAssoc[F4] = laf3params[F4];
3234   solAssoc[F6] = laf3params[F6];
3235 )
3236 ];
3237 finalSol = <|
3238   "bestParams" -> solAssoc,
3239   "usedLaF3" -> usedLaF3,
3240   "worstRelativeError" -> sol[[1]],
3241   "SlaterRatios" -> {F4overF2, F6overF2},
3242   "usedBaricenters" -> usedBaricenters|>;
3243 Return[finalSol];
3244 ]
3245 ];

```

17.3 qplotter.m

This module has a few useful plotting routines.

```

1
2 BeginPackage["qplotter`"];
3
4 GetColor;
5 IndexMappingPlot;

```

```

6 ListLabelPlot;
7 AutoGraphicsGrid;
8 SpectrumPlot;
9 WaveToRGB;
10
11 Begin[{"`Private`"}];
12
13 AutoGraphicsGrid::usage="AutoGraphicsGrid[graphsList] takes a list
   of graphics and creates a GraphicsGrid with them. The number of
   columns and rows is chosen automatically so that the grid has a
   squarish shape.";
14 Options[AutoGraphicsGrid] = Options[GraphicsGrid];
15 AutoGraphicsGrid[graphsList_, opts : OptionsPattern[]] :=
16 (
17   numGraphs = Length[graphsList];
18   width = Floor[Sqrt[numGraphs]];
19   height = Ceiling[numGraphs/width];
20   groupedGraphs = Partition[graphsList, width, width, 1, Null];
21   GraphicsGrid[groupedGraphs, opts]
22 )
23
24 Options[IndexMappingPlot] = Options[Graphics];
25 IndexMappingPlot::usage =
26 "IndexMappingPlot[pairs] take a list of pairs of integers and
   creates a visual representation of how they are paired. The first
   indices being depicted in the bottom and the second indices being
   depicted on top.";
27 IndexMappingPlot[pairs_, opts : OptionsPattern[]] := Module[{width,
   height}, (
28   width = Max[First /@ pairs];
29   height = width/3;
30   Return[
31     Graphics[{{Tooltip[Point[{#[[1]], 0}], #[[1]]}, Tooltip[Point
32       {[#[[2]], height}], #[[2]]],
33         Line[{{#[[1]], 0}, {#[[2]], height}}]} & /@ pairs, opts,
34         ImageSize -> 800]]
35   )
36 ]
37
38 TickCompressor[fTicks_] :=
39 Module[{avgTicks, prevTickLabel, groupCounter, groupTally, idx,
40   tickPosition, tickLabel, avgPosition, groupLabel}, (avgTicks = {};
41   prevTickLabel = fTicks[[1, 2]];
42   groupCounter = 0;
43   groupTally = 0;
44   idx = 1;
45   Do[({tickPosition, tickLabel} = tick;
46     If[
47       tickLabel === prevTickLabel,
48       (groupCounter += 1;
49         groupTally += tickPosition;
50         groupLabel = tickLabel;),
51       (
52         avgPosition = groupTally/groupCounter;
53         avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
54         groupCounter = 1;
55         groupTally = tickPosition;
56         groupLabel = tickLabel;
57       )
58     ];
59     If[idx != Length[fTicks],
60       prevTickLabel = tickLabel;
61       idx += 1;]
62     ), {tick, fTicks}];
63   If[Or[Not[prevTickLabel === tickLabel], groupCounter > 1],
64   (
65     avgPosition = groupTally/groupCounter;
66     avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
67   )
68   ];
69   Return[avgTicks];)]
70
71 GetColor[s_Style] := s /. Style[_ , c_] :> c
72 GetColor[_] := Black

```

```

72  ListLabelPlot::usage="ListLabelPlot[data, labels] takes a list of
73    numbers with corresponding labels. The data is grouped according
74    to the labels and a ListPlot is created with them so that each
75    group has a different color and their corresponding label is shown
76    in the horizontal axis.";
77 Options[ListLabelPlot] = Join[Options[ListPlot], {"TickCompression" ->True,
78 "LabelLevels"->1}];
79 ListLabelPlot[data_, labels_, opts : OptionsPattern[]] := Module[
80   {uniqueLabels, pallete, groupedByTerm, groupedKeys, scatterGroups ,
81   groupedColors, frameTicks, compTicks, bottomTicks, topTicks},
82   (
83     uniqueLabels = DeleteDuplicates[labels];
84     pallete = Table[ColorData["Rainbow", i], {i, 0, 1,
85       1/(Length[uniqueLabels] - 1)}];
86     uniqueLabels = (#[[1]] -> #[[2]]) & /@ Transpose[{RandomSample[uniqueLabels], pallete}];
87     uniqueLabels = Association[uniqueLabels];
88     groupedByTerm = GroupBy[Transpose[{labels, Range[Length[data]], data}], First];
89     groupedKeys = Keys[groupedByTerm];
90     scatterGroups = Transpose[Transpose[#[[2 ;; 3]]] & /@ Values[groupedByTerm]];
91     groupedColors = uniqueLabels[#] & /@ groupedKeys;
92     frameTicks = {Transpose[{Range[Length[data]],
93       Style[Rotate[#, 90 Degree], uniqueLabels[#] & /@ labels]}, Automatic];
94     If[OptionValue["TickCompression"], (
95       compTicks = TickCompressor[frameTicks[[1]]];
96       bottomTicks =
97         MapIndexed[
98           If[EvenQ[First[#2]], {#1[[1]],
99             Tooltip[Style["\[SmallCircle]", GetColor
100             #[[2]]], #1[[2]]],
101             }, #1] &, compTicks];
102       topTicks =
103         MapIndexed[
104           If[OddQ[First[#2]], {#1[[1]],
105             Tooltip[Style["\[SmallCircle]", GetColor
106             #[[2]]], #1[[2]]],
107             }, #1] &, compTicks];
108       frameTicks = {{Automatic, Automatic}, {bottomTicks,
109       topTicks}});
110     ];
111     ListPlot[scatterGroups ,
112       opts,
113       Frame -> True,
114       AxesStyle -> {Directive[Black, Dotted], Automatic},
115       PlotStyle -> groupedColors ,
116       FrameTicks -> frameTicks]
117   )
118 ]
119
120 WaveToRGB::usage="WaveToRGB[wave, gamma] takes a wavelength in nm
121 and returns the corresponding RGB color. The gamma parameter is
122 optional and defaults to 0.8. The wavelength wave is assumed to be
123 in nm. If the wavelength is below 380 the color will be the same
124 as for 380 nm. If the wavelength is above 750 the color will be
125 the same as for 750 nm. The function returns an RGBColor object.
126 REF: https://www.noah.org/wiki/wave\_to\_rgb\_in\_Python. ";
127 WaveToRGB[wave_, gamma_ : 0.8] :=
128   wavelength = (wave);
129   Which[
130     wavelength < 380,
131     wavelength = 380,
132     wavelength > 750,
133     wavelength = 750
134   ];
135   Which[380 <= wavelength <= 440,
136   (
137     attenuation = 0.3 + 0.7*(wavelength - 380)/(440 - 380);
138     R = ((-(wavelength - 440)/(440 - 380))*attenuation)^gamma;
139     G = 0.0;
140     B = (1.0*attenuation)^gamma;
141   ),

```

```

130   440 <= wavelength <= 490,
131   (
132     R = 0.0;
133     G = ((wavelength - 440)/(490 - 440))^gamma;
134     B = 1.0;
135   ),
136   490 <= wavelength <= 510,
137   (
138     R = 0.0;
139     G = 1.0;
140     B = (-(wavelength - 510)/(510 - 490))^gamma;
141   ),
142   510 <= wavelength <= 580,
143   (
144     R = ((wavelength - 510)/(580 - 510))^gamma;
145     G = 1.0;
146     B = 0.0;
147   ),
148   580 <= wavelength <= 645,
149   (
150     R = 1.0;
151     G = (-(wavelength - 645)/(645 - 580))^gamma;
152     B = 0.0;
153   ),
154   645 <= wavelength <= 750,
155   (
156     attenuation = 0.3 + 0.7*(750 - wavelength)/(750 - 645);
157     R = (1.0*attenuation)^gamma;
158     G = 0.0;
159     B = 0.0;
160   ),
161   True,
162   (
163     R = 0;
164     G = 0;
165     B = 0;
166   ]];
167   Return[RGBColor[R, G, B]]
168 )
169
170 FuzzyRectangle::usage = "FuzzyRectangle[xCenter, width, ymin,
height, color] creates a polygon with a fuzzy edge. The polygon is
centered at xCenter and has a full horizontal width of width. The
bottom of the polygon is at ymin and the height is height. The
color of the polygon is color. The left edge and the right edge of
the resulting polygon will be transparent and the middle will be
colored. The polygon is returned as a list of polygons.";
171 FuzzyRectangle[xCenter_, width_, ymin_, height_, color_, intensity_:
1] := Module[
172   {intenseColor, nocolor, ymax, polys},
173   (
174     nocolor = Directive[Opacity[0], color];
175     ymax = ymin + height;
176     intenseColor = Directive[Opacity[intensity], color];
177     polys = {
178       Polygon[{
179         {xCenter - width/2, ymin},
180         {xCenter, ymin},
181         {xCenter, ymax},
182         {xCenter - width/2, ymax}],
183         VertexColors -> {
184           nocolor,
185           intenseColor,
186           intenseColor,
187           nocolor,
188           nocolor}],
189       Polygon[{
190         {xCenter, ymin},
191         {xCenter + width/2, ymin},
192         {xCenter + width/2, ymax},
193         {xCenter, ymax}],
194         VertexColors -> {
195           intenseColor,
196           nocolor,
197           nocolor,
198           intenseColor,

```

```

199         intenseColor}]  

200     };  

201     Return[polys]  

202   );  

203 ]  

204  

205 Options[SpectrumPlot] = Options[Graphics];  

206 Options[SpectrumPlot] = Join[Options[SpectrumPlot], {"Intensities" -> {}, "Tooltips" -> True, "Comments" -> {}, "SpectrumFunction" -> WaveToRGB}];  

207 SpectrumPlot::usage="SpectrumPlot[lines, widthToHeightAspect,  

208   lineWidth] takes a list of spectral lines and creates a visual  

209   representation of them. The lines are represented as fuzzy  

210   rectangles with a width of lineWidth and a height that is  

211   determined by the overall condition that the width to height ratio  

212   of the resulting graph is widthToHeightAspect. The color of the  

213   lines is determined by the wavelength of the line. The function  

214   assumes that the lines are given in nm.  

215 If the lineWidth parameter is a single number, then every line  

216   shares that width. If the lineWidth parameter is a list of numbers  

217   , then each line has a different width. The function returns a  

218   Graphics object. The function also accepts any options that  

219   Graphics accepts. The background of the plot is black by default.  

220   The plot range is set to the minimum and maximum wavelength of the  

221   given lines.  

222 Besides the options for Graphics the function also admits the  

223   option Intensities. This option is a list of numbers that  

224   determines the intensity of each line. If the Intensities option  

225   is not given, then the lines are drawn with full intensity. If the  

226   Intensities option is given, then the lines are drawn with the  

227   given intensity. The intensity is a number between 0 and 1.  

228 The function also admits the option \"Tooltips\". If this option is  

229   set to True, then the lines will have a tooltip that shows the  

230   wavelength of the line. If this option is set to False, then the  

231   lines will not have a tooltip. The default value for this option  

232   is True.  

233 If \"Tooltips\" is set to True and the option \"Comments\" is a non  

234   -empty list, then the tooltip will append the wavelength and the  

235   values in the comments list for the tooltips.  

236 The function also admits the option \"SpectrumFunction\". This  

237   option is a function that takes a wavelength and returns a color.  

238   The default value for this option is WaveToRGB.  

239 ";

```

SpectrumPlot[lines_, widthToHeightAspect_, lineWidth_, opts : OptionsPattern[]] := Module[
{minWave, maxWave, height, fuzzyLines},
(
 colorFun = OptionValue["SpectrumFunction"];
{minWave, maxWave} = MinMax[lines];
height = (maxWave - minWave)/widthToHeightAspect;
fuzzyLines = Which[
 NumberQ[lineWidth] && Length[OptionValue["Intensities"]] == 0,
 FuzzyRectangle[#, lineWidth, 0, height, colorFun[#]] &/@ lines,
 Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]]
== 0,
 MapThread[FuzzyRectangle[#, #2, 0, height, colorFun[#1]] &, {lines, lineWidth}],
 NumberQ[lineWidth] && Length[OptionValue["Intensities"]] > 0,
 MapThread[FuzzyRectangle[#, lineWidth, 0, height, colorFun[#1], #2] &, {lines, OptionValue["Intensities"]}],
 Not[NumberQ[lineWidth]] && Length[OptionValue["Intensities"]]
0,
 MapThread[FuzzyRectangle[#, #2, 0, height, colorFun[#1], #3] &, {lines, lineWidth, OptionValue["Intensities"]}]\n];
 comments = Which[
 Length[OptionValue["Comments"]] > 0,
 MapThread[StringJoin[ToString[#1]<>" nm", "\n", ToString[#2]]&, {lines, OptionValue["Comments"]}],
 Length[OptionValue["Comments"]] == 0,
 ToString[#]<>" nm" & /@ lines,
 True,
 {}];
];
If[OptionValue["Tooltips"],

```

240     fuzzyLines = MapThread[Tooltip[#1, #2] &, {fuzzyLines, comments
241   }];
242   ];
243   graphicsOpts = FilterRules[{opts}, Options[Graphics]];
244   Graphics[fuzzyLines,
245     graphicsOpts,
246     Background -> Black,
247     PlotRange -> {{minWave, maxWave}, {0, height}}]
248   )
249 ];
250 End[];
251
252 EndPackage[];

```

17.4 misc.m

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

```

1 BeginPackage["misc`"];
2 (* Needs["MaTeX`"]; *)
3
4 ArrayBlocker;
5 BlockAndIndex;
6 BlockArrayDimensionsArray;
7 BlockMatrixMultiply;
8 BlockTranspose;
9
10 EllipsoidBoundingBox;
11 EllipsoidBoundingBox2;
12 ExportToH5;
13 ExtractSymbolNames;
14 FirstOrderPerturbation;
15 FlattenBasis;
16
17 FlowMatching;
18 GetModificationDate;
19 GreedyMatching;
20 HamTeX;
21 HelperNotebook;
22
23 RecoverBasis;
24 RemoveTrailingDigits;
25 ReplaceDiagonal;
26 RobustMissingQ;
27 RobustMissingQ;
28
29 RoundToSignificantFigures;
30 RoundValueWithUncertainty;
31 SecondOrderPerturbation;
32 StochasticMatching;
33 SuperIdentity;
34
35 TextBasedProgressBar;
36 ToPythonSparseFunction;
37 ToPythonSymPyExpression;
38 TruncateBlockArray;
39
40 Begin["`Private`"];
41
42 RemoveTrailingDigits[s_String] := StringReplace[s,
43   RegularExpression["\\d+\$"] -> " "];
44
45 BlockTranspose[anArray_] := (
46   Map[Transpose, Transpose[anArray], {2}]
47 );
48
49 BlockMatrixMultiply::usage = "BlockMatrixMultiply[A,B] gives the
50   matrix multiplication of A and B, with A and B having a compatible
51   block structure that allows for matrix multiplication into a
52   congruent block structure.";
53 BlockMatrixMultiply[Amat_, Bmat_] := Module[{rowIdx, colIdx, sumIdx},
54   (
55     Table[
56       Sum[Amat[[rowIdx, sumIdx]].Bmat[[sumIdx, colIdx]], {sumIdx, 1,
57         Dimensions[Amat][[2]]}],
58     ]
59   )
60 ];

```

```

53 {rowIdx,1,Dimensions[Amat][[1]]},  

54 {colIdx,1,Dimensions[Bmat][[2]]}  

55 ]  

56 )  

57 ];  

58  

59 BlockAndIndex::usage="BlockAndIndex[blockSizes, index] takes a list  

   of bin widths and index. The function return in which block the  

   index would be, were the bins to be layed out from left to right.  

   The function also returns the position within the bin in which it  

   is accomodated. The function returns these two numbers as a list  

   of two elements {blockIndex, blockSubIndex}";  

60 BlockAndIndex[blockSizes_List, index_Integer]:=Module[{  

   accumulatedBlockSize,blockIndex, blockSubIndex},  

61 (  

62   accumulatedBlockSize = Accumulate[blockSizes];  

63   If[accumulatedBlockSize[[-1]]-index<0,  

64     Print["Index out of bounds"];  

65     Abort[]  

66   ];  

67   blockIndex = Flatten[Position[accumulatedBlockSize-index,n_ /;  

68   n>=0]][[1]];  

69   blockSubIndex = Mod[index-accumulatedBlockSize[[blockIndex]],  

70   blockSizes[[blockIndex]],1];  

71   Return[{blockIndex,blockSubIndex}]  

72 )  

73 ];  

74  

75 TruncateBlockArray::usage="TruncateBlockArray[blockArray,  

   truncationIndices, blockWidths] takes a an array of blocks and  

   selects the columns and rows corresponding to truncationIndices.  

   The indices being given in what would be the ArrayFlatten[  

   blockArray] version of the array. They blocks in the given array  

   may be SparseArray. This is equivalent to FlattenArray[blockArray]  

   ][truncationIndices, truncationIndices] but may be more efficient  

   blockArray is sparse.";  

76 TruncateBlockArray[blockArray_,truncationIndices_,blockWidths_]:=  

77 Module[  

78 {truncatedArray,blockCol,blockRow,blockSubCol,blockSubRow},(  

79 truncatedArray = Table[  

80   {blockCol,blockSubCol} = BlockAndIndex[blockWidths,fullColIndex];  

81   {blockRow,blockSubRow} = BlockAndIndex[blockWidths,fullRowIndex];  

82   blockArray[[blockRow,blockCol]][[blockSubRow,blockSubCol]],  

83   {fullColIndex,truncationIndices},  

84   {fullRowIndex,truncationIndices}  

85 ];  

86   Return[truncatedArray]  

87 )  

88 ];  

89  

90 BlockArrayDimensionsArray::usage="BlockArrayDimensionsArray[  

   blockArray] returns the array of block sizes in a given blocked  

   array.";  

91 BlockArrayDimensionsArray[blockArray_]:=({  

92   Map[Dimensions,blockArray,{2}]  

93 );  

94  

95 ArrayBlocker::usage="ArrayBlocker[anArray, blockSizes] takes a flat  

   2d array and a congruent 2D array of block sizes, and with them  

   it returns the original array with the block structure imposed by  

   blockSizes. The resulting array satisfies ArrayFlatten[  

   blockedArray] == anArray, and also Map[Dimensions, blockedArray  

   ,{2}] == blockSizes.";  

96 ArrayBlocker[anArray_,blockSizes_]:=Module[{rowStart,colStart,  

   colEnd,numBlocks,blockedArray,blockSize,rowEnd,aBlock,idxRow,  

   idxCol},(  

97   rowStart = 1;  

98   colStart = 1;  

99   colEnd = 1;  

100  numBlocks = Length[blockSizes];  

101  blockedArray = Table[()  

102    blockSize = blockSizes[[idxRow, idxCol]];  

103    rowEnd = rowStart+blockSize[[1]]-1;  

104    colEnd = colStart+blockSize[[2]]-1;  

105    aBlock = anArray[[rowStart;;rowEnd,colStart;;colEnd]];  

106    colStart = colEnd+1;

```

```

104     If[idxCol==numBlocks ,
105      rowStart=rowEnd+1;
106      colStart=1;
107    ];
108    aBlock
109  ),
110  {idxRow,1,numBlocks},
111  {idxCol,1,numBlocks}
112 ];
113 Return[blockedArray]
114 )
115 ];
116
117 ReplaceDiagonal::usage =
118 "ReplaceDiagonal[matrix, repValue] replaces all the diagonal of
the given array to the given value. The array is assumed to be
square and the replacement value is assumed to be a number. The
returned value is the array with the diagonal replaced. This
function is useful for setting the diagonal of an array to a given
value. The original array is not modified. The given array may be
sparse.";
119 ReplaceDiagonal[matrix_, repValue_] :=
120   ReplacePart[matrix,
121     Table[{i, i} -> repValue, {i, 1, Length[matrix]}]];
122
123 Options[RoundValueWithUncertainty] = {"SetPrecision" -> False};
124 RoundValueWithUncertainty::usage = "RoundValueWithUncertainty[x,dx]
given a number x together with an uncertainty dx this function
rounds x to the first significant figure of dx and also rounds dx
to have a single significant figure.
The returned value is a list with the form {roundedX, roundedDx}.
The option \"SetPrecision\" can be used to control whether the
Mathematica precision of x and dx is also set accordingly to these
rules, otherwise the rounded numbers still have the original
precision of the input values.
If the position of the first significant figure of x is after the
position of the first significant figure of dx, the function
returns {0,dx} with dx rounded to one significant figure.";
125 RoundValueWithUncertainty[x_, dx_, OptionsPattern[]] := Module[
126   {xExpo, dxExpo, sigFigs, roundedX, roundedDx, returning},
127   (
128     xExpo = RealDigits[x][[2]];
129     dxExpo = RealDigits[dx][[2]];
130     sigFigs = (xExpo - dxExpo) + 1;
131     {roundedX, roundedDx} = If[sigFigs <= 0,
132       {0., N@RoundToSignificantFigures[dx, 1]},
133       N[
134         {
135           RoundToSignificantFigures[x, xExpo - dxExpo + 1],
136           RoundToSignificantFigures[dx, 1]}
137         ]
138       ];
139     ];
140     returning = If[
141       OptionValue["SetPrecision"],
142       {SetPrecision[roundedX, Max[1, sigFigs]],
143        SetPrecision[roundedDx, 1]},
144       {roundedX, roundedDx}
145     ];
146     Return[returning]
147   )
148 ];
149 ];
150
151 RoundToSignificantFigures::usage =
152 "RoundToSignificantFigures[x, sigFigs] rounds x so that it only
has \
153 sigFigs significant figures.";
154 RoundToSignificantFigures[x_, sigFigs_] :=
155   Sign[x]*N[FromDigits[RealDigits[x, 10, sigFigs]]];
156
157 RobustMissingQ[expr_] := (FreeQ[expr, _Missing] === False);
158
159 TextBasedProgressBar[progress_, totalIterations_, prefix_:""] :=
160   Module[
161     {progMessage},
162     progMessage = ToString[progress] <> "/" <> ToString[
163       totalIterations];

```

```

163     If [progress < totalIterations,
164         WriteString["stdout", StringJoin[prefix, progMessage, "\r"]
165     ]],
166         WriteString["stdout", StringJoin[prefix, progMessage, "\n"]
167     ];
168 ];
169
FirstOrderPerturbation::usage="Given the eigenVectors of a matrix A
(which doesn't need to be given) together with a corresponding
perturbation matrix perMatrix, this function calculates the first
derivative of the eigenvalues with respect to the scale factor of
the perturbation matrix. In the sense that the eigenvalues of the
matrix A +  $\beta$  perMatrix are to first order equal to  $\lambda_i + \Delta_i \beta$ , where the  $\Delta_i$  are the returned values. This
assuming that the eigenvalues are non-degenerate.";
FirstOrderPerturbation[eigenVectors_,
perMatrix_] := (Chop@Diagonal[
Conjugate@eigenVectors . perMatrix . Transpose[eigenVectors]])
```

```

170
SecondOrderPerturbation::usage="Given the eigenValues and
eigenVectors of a matrix A (which doesn't need to be given)
together with a corresponding perturbation matrix perMatrix, this
function calculates the second derivative of the eigenvalues with
respect to the scale factor of the perturbation matrix. In the
sense that the eigenvalues of the matrix A +  $\beta$  perMatrix are to
second order equal to  $\lambda_i + \Delta_i \beta + \frac{1}{2} \Delta_i^2 \beta^2$ , where the  $\Delta_i^2$  are the returned values. The
eigenvalues and eigenvectors are assumed to be given in the same
order, i.e. the ith eigenvalue corresponds to the ith eigenvector.
This assuming that the eigenvalues are non-degenerate.";
SecondOrderPerturbation[eigenValues_, eigenVectors_, perMatrix_] :=
(
dim = Length[perMatrix];
eigenBras = Conjugate[eigenVectors];
eigenKets = eigenVectors;
matV = Abs[eigenBras . perMatrix . Transpose[eigenKets]]^2;
OneOver[x_, y_] := If[x == y, 0, 1/(x - y)];
eigenDiffs = Outer[OneOver, eigenValues, eigenValues, 1];
pProduct = Transpose[eigenDiffs]*matV;
Return[2*(Total /@ Transpose[pProduct])];
)
```

```

185
186 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.";
187 SuperIdentity[args___] := {args};
188
FlattenBasis::usage="FlattenBasis[basis] takes a basis in the
standard representation and separates out the strings that
describe the LS part of the labels and the additional numbers that
define the values of J MJ and MI. It returns a list with two
elements {flatbasisLS, flatbasisNums}. This is useful for saving
the basis to an h5 file where the strings and numbers need to be
separated.";
189 FlattenBasis[basis_] := Module[{flatbasis, flatbasisLS,
flatbasisNums},
(
flatbasis = Flatten[basis];
flatbasisLS = flatbasis[[1 ;; ; ; 4]];
flatbasisNums = Select[flatbasis, Not[StringQ[#]] &];
Return[{flatbasisLS, flatbasisNums}]
)
];
190
191 RecoverBasis::usage="RecoverBasis[{flatbasisLS, flatbasisNums}]
takes the output of FlattenBasis and returns the original basis.
The input is a list with two elements {flatbasisLS, flatbasisNums
}.";
```

```

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

```

207 ]
208
209 ExtractSymbolNames[expr_Hold] := Module[
210   {strSymbols},
211   strSymbols = ToString[expr, InputForm];
212   StringCases[strSymbols,RegularExpression["\\w+"]][[2 ;;]]
213 ]
214
215 ExportToH5::usage =
216   "ExportToH5[fname, Hold[{symbol1, symbol2, ...}]] takes an .h5
217   filename and a held list of symbols and export to the .h5 file the
218   values of the symbols with keys equal the symbol names. The
219   values of the symbols cannot be arbitrary, for instance a list
220   with mixes numbers and string will fail, but an Association with
221   mixed values exports ok. Do give it a try.
222   If the file is already present in disk, this function will
223   overwrite it by default. If the value of a given symbol contains
224   symbolic numbers, e.g. \[Pi], these will be converted to floats in
225   the exported file.";
226 Options[ExportToH5] = {"Overwrite" -> True};
227 ExportToH5[fname_String, symbols_Hold, OptionsPattern[]] :=
228   If[And[FileExistsQ[fname], OptionValue["Overwrite"]],
229     (
230       Print["File already exists, overwriting ..."];
231       DeleteFile[fname];
232     )
233   ];
234   symbolNames = ExtractSymbolNames[symbols];
235   Do[(Print[symbolName];
236     Export[fname, ToExpression[symbolName], {"Datasets", symbolName}
237   ],
238     OverwriteTarget -> "Append"
239   ), {symbolName, symbolNames}]
240 )
241
242 GreedyMatching::usage="GreedyMatching[aList, bList] returns a list
243   of pairs of elements from aList and bList that are closest to each
244   other, this is returned in a list together with a mapping of
245   indices from the aList to those in bList to which they were
246   matched. The option \"alistLabels\" can be used to specify labels
247   for the elements in aList. The option \"blistLabels\" can be used
248   to specify labels for the elements in bList. If these options are
249   used, the function returns a list with three elements the pairs of
250   matched elements, the pairs of corresponding matched labels, and
251   the mapping of indices.";
252 Options[GreedyMatching] = {
253   "alistLabels" -> {},
254   "blistLabels" -> {}};
255 GreedyMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{aValues = aValues0,
256   bValues = bValues0,
257   bValuesOriginal = bValues0,
258   bestLabels, bestMatches,
259   bestLabel, aElement, givenLabels,
260   aLabels, aLabel,
261   diffs, minDiff,
262   bLabels,
263   minDiffPosition, bestMatch},
264   (
265     aLabels = OptionValue["alistLabels"];
266     bLabels = OptionValue["blistLabels"];
267     bestMatches = {};
268     bestLabels = {};
269     givenLabels = (Length[aLabels] > 0);
270     Do[
271       (
272         aElement = aValues[[idx]];
273         diffs = Abs[bValues - aElement];
274         minDiff = Min[diffs];
275         minDiffPosition = Position[diffs, minDiff][[1, 1]];
276         bestMatch = bValues[[minDiffPosition]];
277         bestMatches = Append[bestMatches, {aElement, bestMatch}];
278         If[givenLabels,
279           (
280             aLabel = aLabels[[idx]];
281             bestLabel = bLabels[[minDiffPosition]];
282           )
283         ];
284       );
285     ];
286   ];
287   {bestMatches, bestLabels, givenLabels}
288 ]

```

```

265     bestLabels = Append[bestLabels, {aLabel, bestLabel}];
266     bLabels     = Drop[bLabels, {minDiffPosition}];
267   )
268 ];
269 bValues = Drop[bValues, {minDiffPosition}];
270 If[Length[bValues] == 0, Break[]];
271 ),
272 {idx, 1, Length[aValues]}
273 ];
274 pairedIndices = MapIndexed[{#2[[1]], Position[bValuesOriginal,
#1[[2]]][[1, 1]]} &, bestMatches];
275 If[givenLabels,
276   Return[{bestMatches, bestLabels, pairedIndices}],
277   Return[{bestMatches, pairedIndices}]
278 ]
279 )
280 ]
281
282 StochasticMatching::usage="StochasticMatching[aValues, bValues]
finds a better assignment by randomly shuffling the elements of
aValues and then applying the greedy assignment algorithm. The
function prints what is the range of total absolute differences
found during shuffling, the standard deviation of all of them, and
the number of shuffles that were attempted. The option \"alistLabels\" can be used to specify labels for the elements in
aValues. The option \"blistLabels\" can be used to specify labels
for the elements in bValues. If these options are used, the
function returns a list with three elements the pairs of matched
elements, the pairs of corresponding matched labels, and the
mapping of indices.";
283 Options[StochasticMatching] = {"alistLabels" -> {}, 
284 "blistLabels" -> {}};
285 StochasticMatching[aValues0_, bValues0_, numShuffles_ : 200,
OptionsPattern[]] := Module[{ 
286   aValues = aValues0,
287   bValues = bValues0,
288   matchingLabels, ranger, matches, noShuff, bestMatch, highestCost,
lowestCost, dev, sorter, bestValues,
289   pairedIndices, bestLabels, matchedIndices, shuffler
290 },
291 (
292   matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
293   ranger = Range[1, Length[aValues]];
294   matches = If[Not[matchingLabels], (
295     Table[( 
296       shuffler = If[i == 1, ranger, RandomSample[ranger]];
297       {bestValues, matchedIndices} =
298       GreedyMatching[aValues[[shuffler]], bValues];
299       cost = Total[Abs[#[[1]] - #[[2]]] & /@ bestValues];
300       {cost, {bestValues, matchedIndices}}
301     ), {i, 1, numShuffles}]
302   ),
303   Table[( 
304     shuffler = If[i == 1, ranger, RandomSample[ranger]];
305     {bestValues, bestLabels, matchedIndices} =
306       GreedyMatching[aValues[[shuffler]], bValues,
307       "alistLabels" -> OptionValue["alistLabels"][[shuffler]],
308       "blistLabels" -> OptionValue["blistLabels"]];
309     cost = Total[Abs[#[[1]] - #[[2]]] & /@ bestValues];
310     {cost, {bestValues, bestLabels, matchedIndices}}
311   ), {i, 1, numShuffles}]
312 ];
313   noShuff = matches[[1, 1]];
314   matches = SortBy[matches, First];
315   bestMatch = matches[[1, 2]];
316   highestCost = matches[[-1, 1]];
317   lowestCost = matches[[1, 1]];
318   dev = StandardDeviation[First /@ matches];
319   Print[lowestCost, " <-> ", highestCost, " | \[Sigma]=", dev,
320   " | N=", numShuffles, " | null=", noShuff];
321   If[matchingLabels,
322   (
323     {bestValues, bestLabels, matchedIndices} = bestMatch;
324     sorter = Ordering[First /@ bestValues];
325     bestValues = bestValues[[sorter]];
326     bestLabels = bestLabels[[sorter]];

```

```

327     pairedIndices =
328     MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
329     bestValues];
330     Return[{bestValues, bestLabels, pairedIndices}]
331   ),
332   (
333     {bestValues, matchedIndices} = bestMatch;
334     sorter = Ordering[First /@ bestValues];
335     bestValues = bestValues[[sorter]];
336     pairedIndices =
337       MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
338       bestValues];
339     Return[{bestValues, pairedIndices}]
340   )
341 ];
342 ]
343 ]
344
345 FlowMatching::usage="FlowMatching[aList, bList] returns a list of
pairs of elements from aList and bList that are closest to each
other, this is returned in a list together with a mapping of
indices from the aList to those in bList to which they were
matched. The option \"alistLabels\" can be used to specify labels
for the elements in aList. The option \"blistLabels\" can be used
to specify labels for the elements in bList. If these options are
used, the function returns a list with three elements the pairs of
matched elements, the pairs of corresponding matched labels, and
the mapping of indices. This is basically a wrapper around
Mathematica's FindMinimumCostFlow function. By default the option
\"notMatched\" is zero, and this means that all elements of aList
must be matched to elements of bList. If this is not the case, the
option \"notMatched\" can be used to specify how many elements of
aList can be left unmatched. By default the cost function is Abs
[#1-#2]&, but this can be changed with the option \"CostFun\",
this function needs to take two arguments.";
346 Options[FlowMatching] = {"alistLabels" -> {}, "blistLabels" -> {},
347 "notMatched" -> 0, "CostFun" -> (Abs[#1-#2] &)};
348 FlowMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{aValues = aValues0, bValues = bValues0, edgesSourceToA, capacitySourceToA, nA, nB, costSourceToA, midLayer, midLayerEdges, midCapacities, midCosts, edgesBtoSink, capacityBtoSink, costBtoSink, allCapacities, allCosts, allEdges, graph, flow, bestValues, bestLabels, cFun, aLabels, bLabels, pairedIndices, matchingLabels},
349   (
350     matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
351     aLabels = OptionValue["alistLabels"];
352     bLabels = OptionValue["blistLabels"];
353     cFun = OptionValue["CostFun"];
354     nA = Length[aValues];
355     nB = Length[bValues];
356     (*Build up the edges costs and capacities*)
357     (*From source to the nodes representing the values of the first \
358      list*)
359     edgesSourceToA = ("source" \[DirectedEdge] {"A", #}) & /@ Range[1, nA];
360     capacitySourceToA = ConstantArray[1, nA];
361     costSourceToA = ConstantArray[0, nA];
362
363     (*From all the elements of A to all the elements of B*)
364     midLayer = Table[{{"A", i} \[DirectedEdge] {"B", j}}, {i, 1, nA}, {j, 1, nB}];
365     midLayer = Flatten[midLayer, 1];
366     {midLayerEdges, midCapacities, midCosts} = Transpose[midLayer];
367
368     (*From the elements of B to the sink*)
369     edgesBtoSink = ({"B", #} \[DirectedEdge] "sink") & /@ Range[1, nB];
370     capacityBtoSink = ConstantArray[1, nB];
371     costBtoSink = ConstantArray[0, nB];
372
373     (*Put it all together*)
374     allCapacities = Join[capacitySourceToA, midCapacities, capacityBtoSink];
375     allCosts = Join[costSourceToA, midCosts, costBtoSink];

```

```

381      allEdges      = Join[edgesSourceToA, midLayerEdges, edgesBtoSink
382    ];
383      graph        = Graph[allEdges, EdgeCapacity -> allCapacities,
384                            EdgeCost -> allCosts];
385
386      (*Solve it*)
387      flow          = FindMinimumCostFlow[graph, "source", "sink", nA -
388        OptionValue["notMatched"], "OptimumFlowData"];
389      (*Collect the pairs of matched indices*)
390      pairedIndices = Select[flow["EdgeList"], And[Not[#[[1]] == "source"],
391                                Not[#[[2]] == "sink"]]];
392      pairedIndices = {#[[1, 2]], #[[2, 2]]} & /@ pairedIndices;
393      (*Collect the pairs of matched values*)
394      bestValues    = {aValues[[#[[1]]]], bValues[[#[[2]]]]} & /@
395      pairedIndices;
396      (*Account for having been given labels*)
397      If[matchingLabels,
398        (
399          bestLabels = {aLabels[[#[[1]]]], bLabels[[#[[2]]]]} & /@
400          pairedIndices;
401          Return[{bestValues, bestLabels, pairedIndices}]
402        ),
403        (
404          Return[{bestValues, pairedIndices}]
405        )
406      ];
407
408
409 HelperNotebook::usage="HelperNotebook[nbName] creates a separate
410   notebook and returns a function that can be used to print to the
411   bottom of it. The name of the notebook, nbName, is optional and
412   defaults to OUT.";
413 HelperNotebook[nbName_:>"OUT"] :=
414 Module[{screenDims, screenWidth, screenHeight, nbWidth, leftMargin,
415   PrintToOutputNb}, (
416   screenDims =
417     SystemInformation["Devices", "ScreenInformation"][[1, 2, 2]];
418   screenWidth = screenDims[[1, 2]];
419   screenHeight = screenDims[[2, 2]];
420   nbWidth = Round[screenWidth/3];
421   leftMargin = screenWidth - nbWidth;
422   outputNb = CreateDocument[], WindowTitle -> nbName,
423   WindowMargins -> {{leftMargin, Automatic}, {Automatic,
424     Automatic}},WindowSize -> {nbWidth, screenHeight}];
425   PrintToOutputNb[text_] :=
426     (
427       SelectionMove[outputNb, After, Notebook];
428       NotebookWrite[outputNb, Cell[BoxData[ToBoxes[text]], "Output"]];
429     );
430   Return[PrintToOutputNb]
431 )
432
433
434 GetModificationDate::usage="GetModificationDate[fname] returns the
435   modification date of the given file.";
436 GetModificationDate[theFileName_] := FileDate[theFileName, "Modification"];
437
438 (*Helper function to convert Mathematica expressions to standard
439   form*)
440 StandardFormExpression[expr0_] := Module[{expr=expr0}, ToString[
441   expr, InputForm]];
442
443 (*Helper function to translate to Python/Sympy expressions*)
444 ToPythonSymPyExpression::usage="ToPythonSymPyExpression[expr]
445   converts a Mathematica expression to a SymPy expression. This is a
446   little iffy and might break if the expression includes
447   Mathematica functions that haven't been given a SymPy equivalent."
448   ;
449 ToPythonSymPyExpression[expr0_] := Module[{standardForm, expr=expr0
450   },
451   standardForm = StandardFormExpression[expr];
452   StringReplace[standardForm, {
453     "Power[" -> "Pow(",
454

```

```

439 "Sqrt[" -> "sq(",
440 "[" -> "(",
441 "]" -> ")",
442 "\\\" -> "\\",
443 "I" -> "1j",
444 (*Remove special Mathematica backslashes*)
445 "/" -> "/" (*Ensure division is represented with a slash*)}]];
446
447 ToPythonSparseFunction[sparseArray_SparseArray, funName_] :=
448   Module[{  

449     rowPointers,  

450     dimensions,  

451     pyCode,  

452     vars,  

453     varList,  

454     dataPyList,  

455     colIndicesPyList  

456   },  

457   (* Extract unique symbolic variables from the SparseArray *)  

458   vars = Union[Cases[Normal[sparseArray], _Symbol, Infinity]];  

459   varList = StringRiffle[ToString /@ vars, ", "];  

460   (* Convert data to SymPy compatible strings *)  

461   dataPyList = StringRiffle[ToPythonSymPyExpression /@ Normal[  

462     sparseArray["NonzeroValues"]], ",\n          "];  

463   colIndicesPyList = StringRiffle[  

464     ToPythonSymPyExpression /@ (Flatten[Normal[sparseArray["  

465       ColumnIndices"]]] - 1)), ", "];  

466   (* Extract sparse array properties *)  

467   rowPointers = Normal[sparseArray["RowPointers"]];  

468   dimensions = Dimensions[sparseArray];  

469   (*Create Python code string*)  

470   pyCode = StringJoin[  

471     "#!/usr/bin/env python3\n\n",  

472     "from scipy.sparse import csr_matrix\n",  

473     "import numpy as np\n",  

474     "\n",  

475     "sq = np.sqrt\n",  

476     "\n",  

477     "def ", funName, "(  

478       varList,  

479       ):\\n",  

480       data = np.array([\n           , dataPyList, "\n           ])\\n"
481     ,  

482     "      indices = np.array([  

483       colIndicesPyList,  

484       ]))\\n",  

485     "      indptr = np.array([  

486       StringRiffle[ToString /@ rowPointers, ", ", "], "])\\n",  

487     "      shape = (" , StringRiffle[ToString /@ dimensions, ", ", "],  

488     ")\n",  

489     "      return csr_matrix((data, indices, indptr), shape=shape)\\n"
490   ];  

491   pyCode
492 ];
493
494 Options[HamTeX] = {"T2" -> False};
495 HamTeX::usage="HamTeX[nE] returns an image with parsed LaTeX code
496   for the Hamiltonian of the given number of electrons. The option
497   \"T2\" can be used to specify whether the T2 term should be
498   included in the Hamiltonian for the f^12 configuration. The
499   default is False and the option is ignored if the number of
500   electrons is not 12. The function requires the MaTeX package.";
501 HamTeX[nE_, OptionsPattern[]] := (
502   tex = Which[
503     MemberQ[{1, 13}, nE],
504       "\\zeta \\sum_{i=1}^n \\left(\\hat{s}_i \\cdot \\
505       \\hat{l}_i \\right) \\n" +
506       "\\sum_{i=1}^n \\sum_{k=2,4,6} \\sum_{q=-k}^{k} B_q^{(k)} \\mathcal{C}_{(i)} \\
507       q^{(k)} + \\epsilon",
508     nE == 2,
509       "\\hat{H} = \\sum_{k=2,4,6} F^{(k)} \\hat{f}_k \\
510       + \\alpha \\hat{L}^2 \\
511       + \\beta \\mathcal{C} \\left( \\mathcal{G}(2) \\right) \\
512       + \\gamma \\mathcal{C} \\left( \\mathcal{S}(7) \\right) \\n\\n"
513   ]

```

```

504 &\\quad + \zeta \sum_{i=1}^n \left( \hat{s}_i \cdot \right. \\
505 \left. \hat{l}_i \right) \\
506 + \sum_{k=0,2,4} M^{(k)} \hat{m}_k \\
507 + \sum_{k=2,4,6} P^{(k)} \hat{p}_k \\\\ \\
508 &\\quad \\quad \\quad \\quad + \sum_{i=1}^n \sum_{k=2,4,6} \\
509 \sum_{q=-} \\
510 k^{(k)} B_q^{(k)} \mathcal{C}(i)_q^{(k)} + \epsilon, \\
511 And[nE == 12, OptionValue["T2"], \\
512 " \hat{H}=& \sum_{k=2,4,6} F^{(k)} \hat{f}_k \\
513 + T^{(2)} \hat{t}_2 \\
514 + \alpha \hat{L}^2 \\
515 + \beta , \mathcal{C} \left( \mathcal{G}(2) \right) \\
516 + \gamma , \mathcal{C} \left( \mathcal{S}(7) \right) \\\\ \\
517 &\\quad \\quad + \zeta \sum_{i=1}^n \left( \hat{s}_i \cdot \right. \\
518 \left. \hat{l}_i \right) \\
519 + \sum_{k=0,2,4} M^{(k)} \hat{m}_k \\
520 + \sum_{k=2,4,6} P^{(k)} \hat{p}_k \\\\ \\
521 &\\quad \\quad + \sum_{q=-} \\
522 6 B_q^{(k)} \mathcal{C}(i)_q^{(k)} + \epsilon, \\
523 And[nE == 12, Not@OptionValue["T2"], \\
524 " \hat{H}=& \sum_{k=2,4,6} F^{(k)} \hat{f}_k \\
525 + \alpha \hat{L}^2 \\
526 + \beta , \mathcal{C} \left( \mathcal{G}(2) \right) \\
527 + \gamma , \mathcal{C} \left( \mathcal{S}(7) \right) \\\\ \\
528 &\\quad + \zeta \sum_{i=1}^n \left( \hat{s}_i \cdot \right. \\
529 \left. \hat{l}_i \right) \\
530 + \sum_{k=0,2,4} M^{(k)} \hat{m}_k \\
531 + \sum_{k=2,4,6} P^{(k)} \hat{p}_k \\\\ \\
532 &\\quad \\quad + \sum_{q=-} \\
533 k B_q^{(k)} \mathcal{C}(i)_q^{(k)} + \epsilon, \\
534 True, \\
535 " \hat{H}=& \sum_{k=2,4,6} F^{(k)} \hat{f}_k \\
536 + \sum_{k=2,3,4,6,7,8} T^{(k)} \hat{t}_k \\
537 + \alpha \hat{L}^2 \\
538 + \beta , \mathcal{C} \left( \mathcal{G}(2) \right) \\
539 + \gamma , \mathcal{C} \left( \mathcal{S}(7) \right) \\\\ \\
540 &\\quad \\quad + \zeta \sum_{i=1}^n \left( \hat{s}_i \cdot \right. \\
541 \left. \hat{l}_i \right) \\
542 + \sum_{k=0,2,4} M^{(k)} \hat{m}_k \\
543 + \sum_{k=2,4,6} P^{(k)} \hat{p}_k \\\\ \\
544 &\\quad \\quad + \sum_{i=1}^n \sum_{k=2,4,6} \\
545 \sum_{q=-} \\
546 k B_q^{(k)} \mathcal{C}(i)_q^{(k)} + \epsilon \\
547 ] \\
548 MaTeX[StringJoin[{"\\begin{aligned}\n", tex, "\n\\end{aligned}" \\
549 }]] \\
550 )
551 
552 
553 EllipsoidBoundingBox::usage = "EllipsoidBoundingBox[A,\[Kappa]] \\
554 gives the coordinate intervals that contain the ellipsoid \\
555 determined by r^T.A.r==\[Kappa]^2. The matrix A must be square NxN \\
556 , symmetric, and positive definite. The function returns a list \\
557 with N pairs of numbers, each pair being of the form {-x_i, x_i}." \\
558 ;
559 
560 EllipsoidBoundingBox[Amat_,\[Kappa]_]:=Module[ \\
561 {invAmat, stretchFactors, boundingPlanes, quad}, \\
562 (
563 invAmat = Inverse[Amat];
564 stretchFactors = Sqrt[1/Diagonal[invAmat]];
565 boundingPlanes = DiagonalMatrix[stretchFactors].invAmat;
566 (* The solution is proportional to \[Kappa] *)
567 boundingPlanes = \[Kappa] * boundingPlanes;
568 boundingPlanes = Max /@ Transpose[boundingPlanes];
569 Return[{-#, #}& /@ boundingPlanes]
570 )
571 ];
572 
573 End[]; \\
574 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