qlanth

1 Notation

$$\overbrace{\Lambda}^{\text{Shorthand for all other quantum numbers}} \tag{1}$$

$$\overbrace{\underline{\ell}}^{\text{orbital angular momentum}} \tag{2}$$

$$\overbrace{\langle \Lambda LS \| \hat{O} \| \Lambda' L' S' \rangle}^{\text{LS-reduced matrix element of operator } \hat{O} \text{ between } \Lambda LS \text{ and } \Lambda' L' S'} \tag{3}$$

$$\overbrace{\langle \Lambda LSJ \| \hat{O} \| \Lambda' L' S' J' \rangle}^{\text{LSJ-reduced matrix element of operator } \hat{O} \text{ between } \Lambda LSJ \text{ and } \Lambda' L' S' J'} \tag{4}$$

$$\overbrace{\hat{X}^{(k)}}^{\text{spherical tensor operator of rank k}} \tag{5}$$

$$\overbrace{{}^{2S+1}\alpha L \equiv |\alpha LS\rangle}^{\text{Spectroscopic term } \alpha LS \text{ in Russel-Saunders notation}} \tag{6}$$

$$\overbrace{\hat{X}_q^{(k)}}^{\text{q-component of the spherical tensor operator } \hat{X}^{(k)}} \tag{7}$$

$$\overbrace{\left(\underline{\ell}^{n-1} \alpha' L' S' \right\} \underline{\ell}^n \alpha LS)}^{\text{The coefficient of fractional parentage from the parent term } |\underline{\ell}^{n-1} \alpha' L' S'\rangle \text{ for the daughter term } |\underline{\ell}^n \alpha LS\rangle} \tag{8}$$

2 Definitions

$$\overbrace{\hat{u}^{(k)}}^{\text{irreducible unit tensor operator of rank k}} \tag{9}$$

$$\overbrace{\hat{U}^{(k)} \coloneqq \sum_{i=1}^{n} \hat{u}^k}^{\text{symmetric unit tensor operator for n equivalent electrons}} \tag{10}$$

$$\overbrace{\left(\underline{\ell}^{n-1} \alpha' L' S' \right\} \underline{\ell}^n \alpha LS)}^{\text{The coefficient of fractional parentage from the parent term } |\underline{\ell}^{n-1} \alpha' L' S'\rangle \text{ for the daughter term } |\underline{\ell}^n \alpha LS\rangle} \tag{11}$$

3 The Effective Hamiltonian

Electrons in a multi-electron ion are subject to a number of interactions. They are subject to the attraction towards the nucleus around which they orbit. They are subject to the repulsion that they experience from other electrons. They have spin, also, so they are also subject to a number of magnetic interactions. The spin of each electron interacts with the magnetic field generated by either its own orbital angular momentum, or the orbital angular momentum of another electron. Finally, between pair so felectrons, the spin of one of them will also have an influence the other through the interaction of their respective magnetic dipoles.

This is already a good number of terms to consider in the description of a free ion. However, if we want to take into account the possibility that this description may also hold good for an ion inside a crystal, then we need to add elements to this description that are due to the crystal. The simplest way in which this effect is often included is through the so called crystal-field, which more accurately is often understood as originating from the electric field that an ion feels from the surrounding charges in the crystal lattice.

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

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

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

These effects can be brought within the simplified description only through the help of perturbation theory. The task not the usual one of correcting for the energies/eigenvectors given an added perturbation, but rather to consider the effects of using a truncated Hilbert space due to a known interaction. What results from this is are operator that now act solely within the single configuration but with a convoluted coefficient that depends on overlaps between different configurations. This coefficient one could try to evaluate, and there are some that have trodden this road. Others simply label that complex expression with an unassuming symbol, and leave it as a parameter that one can fit hopes to fit against experimental data. It is from this that the parameters $\alpha, \beta, \gamma, P^0, P^2,$ and P^4 enter into the description that we shall use here.

Something that is also borne out of the configuration interaction analysis is that their influence also modifies previously present intra-configuration operators. For instance, part of the configuration interaction influence that results from the Coulomb repulsion between electrons brings about new operators that need to be included, but they also contribute to the intra-configuration Slater integrals. As such, every parameter in the Hamiltonian becomes a quantity to be fitted against spectroscopic data.

When finding the matrix elements of the Hamiltonian defined by these terms, one also requires the specification of the basis in which

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

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

$$\hat{\mathcal{H}} = \underbrace{\hat{\mathcal{H}}_{\mathrm{k}}}_{\text{kinetic}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{e:sn}}}_{\text{e:shielded nuc}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{e:e}}}_{\text{e:e}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{s:o}}}_{\text{spin-orbit}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{cf}}}_{\text{crystal field}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{s:s\text{-}s:oo}}}_{\substack{\text{spin:spin} \\ \text{and spin:other-orbit}}} \tag{12}$$

$$+ \underbrace{\hat{\mathcal{H}}_{\mathcal{SO}(3)}}_{\text{Trees effective op}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{G}_2}}_{\mathrm{G}_2\text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{\mathcal{SO}(7)}}_{\mathcal{SO}(7)\text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{f3}}}_{\substack{\text{effective} \\ \text{three-body}}} + \underbrace{\hat{\mathcal{H}}_{\mathrm{ec\text{-}s:o}}}_{\substack{\text{electrostatically} \\ \text{correlated spin:orbit}}} \tag{13}$$

$$\hat{\mathcal{H}}_{\mathrm{k}} = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{N}\nabla_i^2 \text{ (kinetic energy of N v-electrons)} \tag{14}$$

$$\hat{\mathcal{H}}_{\mathrm{e:sn}} = \sum_{i=1}^{N} V_{\mathrm{sn}}(\hat{r}_i) \text{ (interaction of v-electrons with shielded nuclear charge)} \tag{15}$$

$$\hat{\mathcal{H}}_{\mathrm{e:e}} = \sum_{i>j}^{N,N} \frac{e^2}{\|\hat{\vec{r}}_i - \hat{\vec{r}}_j\|} \text{ (v-electron:v-electron repulsion)} \tag{16}$$

$$\hat{\mathcal{H}}_{\mathrm{s:o}} = \begin{cases} \sum_{i=1}^{N} \xi(r_i)\left(\hat{\vec{s}}_i \cdot \hat{\vec{l}}_i\right) \text{ with } \xi(r_i) = \frac{\hbar^2}{2m^2c^2r_i}\frac{\mathrm{d}V_{\mathrm{sn}}(r_i)}{\mathrm{d}r_i} \\ \sum_{i=1}^{N} \zeta\left(\hat{\vec{s}}_i \cdot \hat{\vec{l}}_i\right) \text{ with } \zeta \text{ the radial average of } \xi(r_i) \text{ or used as phenomenological parameter} \end{cases} \tag{17}$$

$$\hat{\mathcal{H}}_{\mathrm{cf}} = \sum_{i=1}^{N} V_{\mathrm{CF}}(\hat{r}_i) \text{ (crystal field interaction of v-electrons with electrostatic field due to surroundings)} \tag{18}$$

$$\hat{\mathcal{H}}_{\mathrm{s:s\text{-}s:oo}} = \sum_{i=0,2,4} M^i \hat{m}_i \tag{19}$$

$$\mathcal{C}(\mathcal{G}) \coloneqq \text{The Casimir operator of group } \mathcal{G}. \tag{20}$$

$$\hat{\mathcal{H}}_{\mathcal{SO}(3)} = \alpha\,\mathcal{C}(\mathbb{R}^3) = \alpha\hat{L}^2 = \underbrace{\alpha L(L+1)}_{\text{in LS coupling}} \text{ (Trees effective operator[1])} \tag{21}$$

$$\hat{\mathcal{H}}_{\mathrm{G}_2} = \beta\,\mathcal{C}(\mathrm{G}_2) \tag{22}$$

$$\hat{\mathcal{H}}_{\mathcal{SO}(7)} = \gamma\,\mathcal{C}(\mathcal{SO}(7)) \tag{23}$$

$$\hat{\mathcal{H}}_{\mathrm{f3}} = T'^2 t_2' + \sum_{i=2,3,4,6,7,8}^{N} T^i \hat{t}_i \text{ (effective three-body operators } \hat{t}_i \text{ with strengths } T_i)^2 \tag{24}$$

$$\hat{\mathcal{H}}_{\mathrm{ec\text{-}s:o}} = \sum_{i=2,4,6} P^i \hat{p}_i \tag{25}$$

3.1 $\hat{\mathcal{H}}_{\mathrm{k}}$: kinetic energy

$$\hat{\mathcal{H}}_{\mathrm{k}} = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{N}\nabla_i^2 \text{ (kinetic energy of N v-electrons)} \tag{26}$$

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

3.2 $\hat{\mathcal{H}}_{\mathrm{e:sn}}$: e:shielded nuc

$$\hat{\mathcal{H}}_{\mathrm{k}} = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{N}\nabla_i^2 \text{ (kinetic energy of N v-electrons)} \tag{27}$$

Instead of using the shielded nuclear charge this could have been instead the bare nuclear charge, but then we would have needed to take into account the repulsion from the electrons in closed shells. Here we are already bringing some simplification in that we approximate the compound effect on the valence electrons due to the charge of the filled shells and the charge of the nucleus is that of a central field.

Then again, this term also contributes a common energy shift to all the energies that we can obtain within the single-configuration description, so this one will also be omittted. It might be useful to use this term and the previous one to estimate the energy differences between the states in different configurations, but we will not do that here.

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

$$\hat{\mathcal{H}}_{\mathrm{e:e}} = \sum_{i>j}^{N,N}\frac{e^2}{\|\hat{\vec{r}}_i - \hat{\vec{r}}_j\|} = \sum_{k=0,2,4,6} F^k \hat{f}_k = \sum_{k=0,1,2,3} E_k \hat{e}^k \tag{28}$$

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

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

$$\langle \underline{\mathrm{f}}^n \alpha^{2S+1} L \| \hat{\mathcal{H}}_{\mathrm{e:e}} \| \underline{\mathrm{f}}^n \alpha'^{2S'+1} L' \rangle = \sum_{k=0,2,4,6} f_k(n, \alpha LS, \alpha' L' S') F^k \tag{29}$$

where

$$f_k(n, \alpha LS, \alpha' L' S') = \frac{1}{2}\delta(S,S')\delta(L,L')\langle \underline{\mathrm{f}} \| \hat{C}^{(k)} \| \underline{\mathrm{f}} \rangle^2 \times$$
$$\left\{ \frac{1}{2L+1}\sum_{\alpha'' L''} \langle \underline{\mathrm{f}}^n \alpha'' L'' S \| \hat{U}^{(k)} \| \underline{\mathrm{f}}^n \alpha L S \rangle \langle \underline{\mathrm{f}}^n \alpha'' L'' S \| \hat{U}^{(k)} \| \underline{\mathrm{f}}^n \alpha' L S \rangle - \delta(\alpha, \alpha') \frac{n\,(4\underline{\mathrm{f}} + 2 - n)}{(2\underline{\mathrm{f}}+1)(4\underline{\mathrm{f}}+1)} \right\} \tag{30}$$

4 qlanth.m

```
(* ------------------------------------------------------------------
+--------------------------------------------------------------------+
|                                                                    |
|                      __                          __     __         |
|            ____ _   / /  ____ _   ____     / /_   / /_            |
|           / __ `/  / /  / __ `/  / __ \   / __/  / __ \           |
|          / /_/ /  / /  / /_/ /  / / / /  / /_   / / / /           |
|          \__, /  /_/   \__, _/  /_/ /_/   \__/  /_/ /_/           |
|            /_/                                                     |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
This code was initially authored by Christopher Dodson and then
rewritten by David Lizarazo in the years 2022-2024. It has also
benefited from the discussions with Tharnier Puel.

It uses an effective Hamiltonian to describe the electronic
structure of lanthanide ions in crystals. This effective Hamiltonian
includes terms representing the following interactions/relativistic
corrections: spin-orbit, electrostatic repulsion, spin-spin, crystal
field and spin-other- orbit.

The Hilbert space used in this effective Hamiltonian is limited to
single f^n configurations. The inaccuracy of this single
configuration description is partially compensated by the inclusion
of configuration interaction terms as parametrized by the Casimir
operators of SO(3), G(2), and SO(7), and by three-body effective
operators ti.

The parameters included in this model are listed in the string
paramAtlas.

The notebook "qlanth.nb" contains a gallery with all the functions
included in this module with some simple use cases.

The notebook "The Lanthanides in LaF3.nb" is an example in which the
results from this code are compared against the published results by
Carnall et. al for the energy levels of lanthinde ions in crystals
of lanthanum fluoride.

REFERENCES:

+ Condon, E U, and G H Shortley. The Theory of Atomic Spectra, 1935.

+ Racah, Giulio. "Theory of Complex Spectra. III." Physical Review
63, no. 9-10 (May 1, 1943): 367-82.
https://doi.org/10.1103/PhysRev.63.367.

+ Racah, Giulio. "Theory of Complex Spectra. II." Physical Review
62, no. 9-10 (November 1, 1942): 438-62.
https://doi.org/10.1103/PhysRev.62.438.

```

```
+ Rajnak, K, and BG Wybourne. "Configuration Interaction Effects in
l^N Configurations." Physical Review 132, no. 1 (1963): 280.
https://doi.org/10.1103/PhysRev.132.280.

+ Wybourne, Brian G. Spectroscopic Properties of Rare Earths, 1965.

+ Judd, BR. "Three-Particle Operators for Equivalent Electrons."
Physical Review 141, no. 1 (1966): 4.
https://doi.org/10.1103/PhysRev.141.4.

+ Nielson, C. W., and George F Koster. "Spectroscopic Coefficients
for the p^n, d^n, and f^n Configurations", 1963.

+ Judd, BR, HM Crosswhite, and Hannah Crosswhite. "Intra-Atomic
Magnetic Interactions for f Electrons." Physical Review 169, no. 1
(1968): 130. https://doi.org/10.1103/PhysRev.169.130.

+ (TASS) Cowan, Robert Duane. The Theory of Atomic Structure and
Spectra. Los Alamos Series in Basic and Applied Sciences 3.
Berkeley: University of California Press, 1981.

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

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

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

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

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


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

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

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

```
E2: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
E3: linear combination of F_k, see eqn. (2-80) in Wybourne 1965

ζ: spin-orbit strength parameter.

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

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

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

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

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

gs: electronic gyromagnetic ratio

α: Trees' parameter α describing configuration interaction via the
    Casimir operator of SO(3)
β: Trees' parameter β describing configuration interaction via the
    Casimir operator of G(2)
γ: Trees' parameter γ describing configuration interaction via the
    Casimir operator of SO(7)

B02: crystal field parameter B_0^2 (real)
B04: crystal field parameter B_0^4 (real)
```

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

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

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

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

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

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

\[Epsilon]: ground level baseline shift
t2Switch: controls the usage of the t2 operator beyond f7
wChErrA: If 1 then the type-A errors in Chen are used, if 0 then not.
wChErrB: If 1 then the type-B errors in Chen are used, if 0 then not.
";
paramSymbols = StringSplit[paramAtlas, "\n"];
paramSymbols = Select[paramSymbols, # != ""& ];
paramSymbols = ToExpression[StringSplit[#, ":"][[1]]] & /@
    paramSymbols;
Protect /@ paramSymbols;
paramLines = Select[StringSplit[paramAtlas, "\n"], # != "" &];
usageTemplate = StringTemplate["`paramSymbol`::usage=\"`paramSymbol` :
     `paramUsage`\";"];
Do[(
  {paramString, paramUsage} = StringSplit[paramLine, ":"];
  paramUsage = StringTrim[paramUsage];
  expressionString = usageTemplate[<|"paramSymbol" -> paramString, "
    paramUsage" -> paramUsage|>];
  ToExpression[usageTemplate[<|"paramSymbol" -> paramString,
     "paramUsage" -> paramUsage|>]]
),
{paramLine, paramLines}
];

```

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

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

AllowedJ;
AllowedMforJ;
AllowedNKSLJMforJMTerms;
AllowedNKSLJMforJTerms;

AllowedNKSLJTerms;
AllowedNKSLTerms;
AllowedNKSLforJTerms;
AllowedSLJMTerms;
AllowedSLJTerms;

AllowedSLTerms;
BasisLSJMJ;
Bqk;
CFP;
CFPAssoc;

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

Cqk;
CrystalField;
Dk;
ElectrostaticConfigInteraction;
Electrostatic;

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

FastIonSolverLaF3;
FindNKLSTerm;
FindSL;

FtoE;
GG2U;
GSO7W;
```

```
GenerateCFP;
GenerateCFPAssoc;

GenerateCFPTable;
GenerateCrystalFieldTable;
GenerateElectrostaticTable;
GenerateReducedUkTable;
GenerateReducedV1kTable;

GenerateSOOandECSOLSTable;
GenerateSOOandECSOTable;
GenerateSpinOrbitTable;
GenerateSpinSpinTable;
GenerateT22Table;

GenerateThreeBodyTables;
GenerateThreeBodyTables;
Generator;
HamMatrixAssembly;
HamiltonianForm;

HamiltonianMatrixPlot;
HoleElectronConjugation;
IonSolverLaF3;
ImportMZip;
JJBlockMatrix;
JJBlockMatrixFileName;

JJBlockMatrixTable;
LabeledGrid;
LoadAll;
LoadCFP;
LoadCarnall;

LoadChenDeltas;
LoadElectrostatic;
LoadGuillotParameters;
LoadParameters;
LoadSOOandECSO;

LoadSOOandECSOLS;
LoadSpinOrbit;
LoadSpinSpin;
LoadSymbolicHamiltonians;
LoadT11;

LoadT22;
LoadTermLabels;
LoadThreeBody;
LoadUk;
LoadV1k;

MagneticInteractions;
MaxJ;
MinJ;
```

```
315 NKCFPPhase;
316
317 ParamPad;
318 ParseStates;
319 ParseStatesByNumBasisVecs;
320 ParseStatesByProbabilitySum;
321 ParseTermLabels;
322
323 Phaser;
324 PrettySaunders;
325 PrettySaundersSLJ;
326 PrettySaundersSLJmJ;
327 PrintL;
328
329 PrintSLJ;
330 PrintSLJM;
331 ReducedSOOandECSOinf2;
332 ReducedSOOandECSOinfn;
333 ReducedT11inf2;
334
335 ReducedT22inf2;
336 ReducedUk;
337 ReducedUkTable;
338 ReducedV1kTable;
339 Reducedt11inf2;
340
341 ReplaceInSparseArray;
342 SimplerSymbolicHamMatrix;
343 SOOandECSO;
344 SOOandECSOTable;
345 Seniority;
346
347 ShiftedLevels;
348 SixJay;
349 SpinOrbit;
350 SpinSpin;
351 SpinSpinTable;
352
353 Sqk;
354 SquarePrimeToNormal;
355 ReducedT11infn;
356 ReducedT22infn;
357 TPO;
358
359 TabulateJJBlockMatrixTable;
360 TabulateManyJJBlockMatrixTables;
361 ScalarOperatorProduct;
362 ThreeBodyTable;
363
364 ThreeBodyTables;
365 ThreeJay;
366 TotalCFIters;
367 chenDeltas;
368 fK;
369
```

```
370 fnTermLabels;
371 moduleDir;
372 symbolicHamiltonians;
373
374 (* this selects the function that is applied
375 to calculated matrix elements *)
376 SimplifyFun = Expand;
377
378 Begin["`Private`"]
379
380   moduleDir = DirectoryName[$InputFileName];
381   frontEndAvailable = (Head[$FrontEnd] === FrontEndObject);
382
383   (*
      ############################################################################
       *)
384   (* ################################## MISC
      ################################## *)
385
386   TPO::usage="Two plus one.";
387   TPO[args__] := Times @@ ((2*# + 1) & /@ {args});
388
389   Phaser::usage = "Phaser[x] returns (-1)^x";
390   Phaser[exponent_] := ((-1)^exponent);
391
392   TriangleCondition::usage = "TriangleCondition[a, b, c] returns True
      if a, b, and c satisfy the triangle condition.";
393   TriangleCondition[a_, b_, c_] := (Abs[b - c] <= a <= (b + c));
394
395   TriangleAndSumCondition::usage = "TriangleAndSumCondition[a, b, c]
      returns True if a, b, and c satisfy the triangle and sum
      conditions.";
396   TriangleAndSumCondition[a_, b_, c_] := (And[Abs[b - c] <= a <= (b +
      c), IntegerQ[a + b + c]]);
397
398   SquarePrimeToNormal::usage = "Given a list with the parts
      corresponding to the squared prime representation of a number,
      this function parses the result into standard notation.";
399   SquarePrimeToNormal[squarePrime_] :=
400   (
401     radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {idx1
      , 2, Length[squarePrime]}];
402     radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" -> 13};
403     val = squarePrime[[1]] * Sqrt[radical];
404     Return[val];
405   );
406
407   ParamPad::usage = "ParamPad[params] takes an association params
      whose keys are a subset of paramSymbols. The function returns a
      new association where all the keys not present in paramSymbols,
      will now be included in the returned association with their values
       set to zero.
408   The function additionally takes an option \"Print\" that if set to
      True, will print the symbols that were not present in the given
      association.";
```

```
Options[ParamPad] = {"Print" -> True}
ParamPad[params_, OptionsPattern[]] := (
  notPresentSymbols = Complement[paramSymbols, Keys[params]];
  If[OptionValue["Print"],
    Print["Symbols not in given params: ",
    notPresentSymbols]
  ];
  newParams = Transpose[{paramSymbols, ConstantArray[0, Length[
  paramSymbols]]}];
  newParams = (#[[1]] -> #[[2]]) & /@ newParams;
  newParams = Association[newParams];
  newParams = Join[newParams, params];
  Return[newParams];
  )

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

ReducedUk::usage = "ReducedUk[n, l, SL, SpLp, k] gives the reduced
  matrix element of the symmetric unit tensor operator U^(k). See
  equation 11.53 in TASS.";
ReducedUk[numE_, l_, SL_, SpLp_, k_] :=
  Module[{spin, orbital, Uk,
    S, L, Sp, Lp, Sb, Lb,
    parentSL, cfpSL, cfpSpLp, Ukval, SLparents, SLpparents,
  commonParents, phase},
    {spin, orbital} = {1/2, 3};
    {S, L}          = FindSL[SL];
    {Sp, Lp}        = FindSL[SpLp];
    If[Not[S == Sp],
      Return[0]
    ];
    cfpSL       = CFP[{numE, SL}];
    cfpSpLp     = CFP[{numE, SpLp}];
    SLparents   = First /@ Rest[cfpSL];
    SLpparents = First /@ Rest[cfpSpLp];
    commonParents = Intersection[SLparents, SLpparents];
    Uk = Sum[(
      {Sb, Lb} = FindSL[\[Psi]b];
      Phaser[Lb] *
        CFPAssoc[{numE, SL, \[Psi]b}] *
        CFPAssoc[{numE, SpLp, \[Psi]b}] *
        SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
    ),
    {\[Psi]b, commonParents}
    ];
    phase     = Phaser[orbital + L + k];
    prefactor = numE * phase * Sqrt[TPO[L,Lp]];
    Ukval     = prefactor*Uk;
    Return[Ukval];
]

```

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

SixJay::usage = "SixJay[{j1, j2, j3}, {j4, j5, j6}] provides the
  value for SixJSymbol[{j1, j2, j3}, {j4, j5, j6}] with memorization
   of computed values.";
SixJay[{j1_, j2_, j3_}, {j4_, j5_, j6_}] := (
  sixJayval =
    Which[
    Not[TriangleAndSumCondition[j1, j2, j3]],
    0,
    Not[TriangleAndSumCondition[j1, j5, j6]],
    0,
    Not[TriangleAndSumCondition[j4, j2, j6]],
    0,
    Not[TriangleAndSumCondition[j4, j5, j3]],
    0,
    True,
    SixJSymbol[{j1, j2, j3}, {j4, j5, j6}]];
  SixJay[{j1, j2, j3}, {j4, j5, j6}] = sixJayval);

ThreeJay::usage = "ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] gives the
  value of the Wigner 3j-symbol and memorizes the computed value.";
ThreeJay[{j1_, m1_}, {j2_, m2_}, {j3_, m3_}] := (
 threejval = Which[
   Not[(m1 + m2 + m3) == 0],
   0,
   Not[TriangleCondition[j1,j2,j3]],
   0,
   True,
   ThreeJSymbol[{j1, m1}, {j2, m2}, {j3, m3}]
   ];
 ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] = threejval);

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

```
        CFPAssoc[{numE, SpLp, \[Psi]b}] *
        SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
        SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
      ),
    {\[Psi]b, commonParents}
    ];
    prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp, Lp
    ] ];
    Return[prefactor * Vk1];
    ]

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

```
549  GenerateReducedV1kTable::usage = "GenerateReducedV1kTable[nmax,
       export calculates values for Vk1 and returns an association where
       the keys are lists of the form {n, SL, SpLp, 1}. If the option \"
       Export\" is set to True then the resulting data is saved to ./data
       /ReducedV1kTable.m."
550  Options[GenerateReducedV1kTable] = {"Export" -> True, "Progress" ->
       True};
551  GenerateReducedV1kTable[numEmax_Integer:7, OptionsPattern[]]:= (
552    numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
       AllowedNKSLTerms[#]]&/@Range[1, numEmax]];
553    Print["Calculating " <> ToString[numValues] <> " values for Vk1."
       ];
554    counter = 1;
555    If[And[OptionValue["Progress"], frontEndAvailable],
556    progBar = PrintTemporary[
557        Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
558          counter}]]]
559      ];
560    ReducedV1kTable = Table[
561      (
562        counter = counter+1;
563        {n, SL, SpLp, 1} -> SimplifyFun[ReducedV1k[n, SL, SpLp, 1]]
564      ),
565      {n, 1, numEmax},
566      {SL, AllowedNKSLTerms[n]},
567      {SpLp, AllowedNKSLTerms[n]}
568    ];
569    ReducedV1kTable = Association[ReducedV1kTable];
570    If[And[OptionValue["Progress"], frontEndAvailable],
571      NotebookDelete[progBar]
572    ];
573    exportFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m"
       }];
574    If[OptionValue["Export"],
575      (
576        Print["Exporting to file "<>ToString[exportFname]];
577        Export[exportFname, ReducedV1kTable];
578      )
579    ];
580    Return[ReducedV1kTable];
581  )
582
583  (* ############################## Racah Algebra
       ############################## *)
584  (*
       ********************************************************************************
        *)
585
586  (*
       ********************************************************************************
        *)
587  (* ############################## Electrostatic
       ############################## *)
588
589  fsubk::usage = "Slater integral f_k. See equation 12.17 in TASS.";
```

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

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

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

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

```
    Return[{E0, E1, E2, E3}];
  ]
  );

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

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

```
    Return[eMatrixVal];
  ]

  GenerateElectrostaticTable::usage = "GenerateElectrostaticTable[
    numEmax] can be used to generate the table for the electrostatic
    interaction from f^1 to f^numEmax. If the option \"Export\" is set
     to True then the resulting data is saved to ./data/
    ElectrostaticTable.m.";
  Options[GenerateElectrostaticTable] = {"Export" -> True, "
    Coefficients" -> "Slater"};
  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
    ################################## *)

  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]
```

```
717     )
718     ];
719
720   BasisLSJMJ::usage = "BasisLSJMJ[numE] returns the ordered basis in L
        -S-J-MJ with the total orbital angular momentum L and total spin
        angular momentum S coupled together to form J. The function
        returns a list with each element representing the quantum numbers
        for each basis vector. Each element is of the form {SL (string in
        spectroscopic notation),J,MJ}.";
721   BasisLSJMJ[numE_] := Module[{energyStatesTable, basis, idx1},
722     (
723       energyStatesTable = BasisTableGenerator[numE];
724       basis = Table[
725         energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
726         {idx1, 1, Length[AllowedJ[numE]]}];
727       basis = Flatten[basis, 1];
728       Return[basis]
729     )
730     ];
731
732   (* ################################## Bases
        ################################## *)
733   (*
        ##############################################################################
         *)
734
735   (*
        ##############################################################################
         *)
736   (* ################### Coefficients of Fracional Parentage
        ################### *)
737
738   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.";
739   Options[GenerateCFP] = {"Export" -> True, "PhaseFunction"-> "NK"};
740   GenerateCFP[OptionsPattern[]]:= (
741     CFP = Table[
742       {numE, NKSL} -> First[CFPTerms[numE, NKSL]],
743       {numE, 1, 7},
744       {NKSL, AllowedNKSLTerms[numE]}];
745     CFP = Association[CFP];
746     (* Go all the way to f14 *)
747     CFP = CFPExpander["Export" -> False, "PhaseFunction"-> OptionValue
        ["PhaseFunction"]];
748     If[OptionValue["Export"],
749       Export[FileNameJoin[{moduleDir, "data", "CFPs.m"}], CFP];
750     ];
751     Return[CFP];
752   )
```

```
753
754  JuddCFPPhase::usage="Phase between conjugate coefficients of
       fractional parentage according to Velkov's thesis, page 40.";
755  JuddCFPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
       parentSeniority_, daughterSeniority_] := Module[
756    {spin, orbital, expo, phase},
757  (
758      {spin, orbital} = {1/2, 3};
759      expo = (
760          (parentS + parentL + daughterS + daughterL) -
761          (orbital + spin) +
762          1/2 * (parentSeniority + daughterSeniority - 1)
763      );
764      phase = Phaser[-expo];
765      Return[phase];
766  )
767  ]
768
769  NKCFPPhase::usage="Phase between conjugate coefficients of
       fractional parentage according to Nielson and Koster page viii.
       Note that there is a typo on there the expression for zeta should
       be (-1)^((v-1)/2) instead of (-1)^(v - 1/2).";
770  NKCFPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
       parentSeniority_, daughterSeniority_] := Module[{spin, orbital,
       expo, phase},
771    (
772      {spin, orbital} = {1/2, 3};
773      expo = (
774          (parentS + parentL + daughterS + daughterL) -
775          (orbital + spin)
776      );
777      phase = Phaser[-expo];
778      If[parent == 2*orbital,
779          phase = phase * Phaser[(daughterSeniority-1)/2]];
780      Return[phase];
781    )
782  ]
783
784  Options[CFPExpander] = {"Export" -> True, "PhaseFunction" -> "NK"};
785  CFPExpander::usage="Using the coefficients of fractional parentage
       up to f7 this function calculates them up to f14.
786
787  The coefficients of fractional parentage are taken beyond the half-
       filled shell using the phase convention determined by the option \
       "PhaseFunction\". The default is \"NK\" which corresponds to the
       phase convention of Nielson and Koster. The other option is \"Judd
       \" which corresponds to the phase convention of Judd. The result
       is exported to the file ./data/CFPs_extended.m.";
788  CFPExpander[OptionsPattern[]]:=Module[
789    {orbital, halfFilled, fullShell, parentMax, PhaseFun,
790    complementaryCFPs, daughter, conjugateDaughter,
791    conjugateParent, parentTerms, daughterTerms,
792    parentCFPs, daughterSeniority, daughterS, daughterL,
793    parentCFP, parentTerm, parentCFPval,
794    parentS, parentL, parentSeniority, phase, prefactor,
```

```
    newCFPval, key, extendedCFPs, exportFname},
  (
    orbital    = 3;
    halfFilled = 2 * orbital + 1;
    fullShell  = 2 * halfFilled;
    parentMax  = 2 * orbital;

    PhaseFun  = <|
        "Judd" -> JuddCFPPhase,
        "NK" -> NKCFPPhase|>[OptionValue["PhaseFunction"]];
    PrintTemporary["Calculating CFPs using the phase system from ",
  PhaseFun];
    (* Initialize everything with lists to be filled in the next Do
  *)
    complementaryCFPs =
        Table[
        ({numE, term} -> {term}),
        {numE, halfFilled + 1, fullShell - 1, 1},
        {term, AllowedNKSLTerms[numE]
        }];
    complementaryCFPs = Association[Flatten[complementaryCFPs]];
    Do[(
        daughter          = parent + 1;
        conjugateDaughter = fullShell - parent;
        conjugateParent   = conjugateDaughter - 1;
        parentTerms       = AllowedNKSLTerms[parent];
        daughterTerms     = AllowedNKSLTerms[daughter];
        Do[
        (
            parentCFPs             = Rest[CFP[{daughter,
  daughterTerm}]];
            daughterSeniority      = Seniority[daughterTerm];
            {daughterS, daughterL} = FindSL[daughterTerm];
            Do[
            (
                {parentTerm, parentCFPval} = parentCFP;
                {parentS, parentL}         = FindSL[parentTerm];
                parentSeniority            = Seniority[parentTerm];
                phase = PhaseFun[parent, parentS, parentL,
                                 daughterS, daughterL,
                                 parentSeniority, daughterSeniority];
                prefactor = (daughter * TPO[daughterS, daughterL]) /
                            (conjugateDaughter * TPO[parentS,
  parentL]);
                prefactor = Sqrt[prefactor];
                newCFPval = phase * prefactor * parentCFPval;
                key = {conjugateDaughter, parentTerm};
                complementaryCFPs[key] = Append[complementaryCFPs[
  key], {daughterTerm, newCFPval}]
            ),
            {parentCFP, parentCFPs}
            ]
        ),
        {daughterTerm, daughterTerms}
        ]
```

```
845             ),
846         {parent, 1, parentMax}
847         ];
848 
849         complementaryCFPs[{14, "1S"}] = {"1S", {"2F",1}};
850         extendedCFPs          = Join[CFP, complementaryCFPs];
851         If[OptionValue["Export"];,
852         (
853             exportFname = FileNameJoin[{moduleDir, "data", "
    CFPs_extended.m"}];
854             Print["Exporting to ", exportFname];
855             Export[exportFname, extendedCFPs];
856         )
857         ];
858         Return[extendedCFPs];
859     )
860     ]
861 
862     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.
863 
864     The data being parsed here is the file attachment B1F_ALL.TXT which
      comes from Velkov's thesis.";
865     Options[GenerateCFPTable] = {"Export" -> True};
866     GenerateCFPTable[OptionsPattern[]]:=Module[
867       {rawText, rawLines, leadChar, configIndex,
868       line, daughter, lineParts, numberCode, parsedNumber, toAppend,
      CFPTablefname},
869     (
870       CleanWhitespace[string_]      := StringReplace[string,
      RegularExpression["\\s+"]->" "];
871       AddSpaceBeforeMinus[string_] := StringReplace[string,
      RegularExpression["(?<!\\s)-"]->" -"];
872       ToIntegerOrString[list_]      := Map[If[StringMatchQ[#,
      NumberString], ToExpression[#], #] &, list];
873       CFPTable       = ConstantArray[{},7];
874       CFPTable[[1]] = {{"2F",{"1S",1}}};
875 
876       (* Cleaning before processing is useful *)
877       rawText   = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"
      }]];
878       rawLines = StringTrim/@StringSplit[rawText,"\n"];
879       rawLines = Select[rawLines,#!=""&];
880       rawLines = CleanWhitespace/@rawLines;
881       rawLines = AddSpaceBeforeMinus/@rawLines;
882 
883       Do[(
884         (* the first character can be used to identify the start of a
      block *)
885         leadChar=StringTake[line,{1}];
886         (* ..FN, N is at position 50 in that line *)
887         If[leadChar=="[",
888         (
```

```
889           configIndex=ToExpression[StringTake[line,{50}]];
890           Continue[];
891         )
892         ];
893         (* Identify which daughter term is being listed *)
894         If[StringContainsQ[line,"[DAUGHTER TERM]"],
895           daughter=StringSplit[line,"["][[1]];
896           CFPTable[[configIndex]]=Append[CFPTable[[configIndex]],{
    daughter}];
897           Continue[];
898         ];
899         (* Once we get here we are already parsing a row with
    coefficient data *)
900         lineParts    = StringSplit[line," "];
901         parent       = lineParts[[1]];
902         numberCode   = ToIntegerOrString[lineParts[[3;;]]];
903         parsedNumber = SquarePrimeToNormal[numberCode];
904         toAppend     = {parent,parsedNumber};
905         CFPTable[[configIndex]][[-1]] = Append[CFPTable[[configIndex
    ]][[-1]], toAppend]
906       ),
907       {line,rawLines}];
908       If[OptionValue["Export"],
909         (
910         CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
911         Export[CFPTablefname, CFPTable];
912         )
913       ];
914       Return[CFPTable];
915     )
916     ]
917
918     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.";
919   Options[GenerateCFPAssoc] = {"Export" -> True};
920   GenerateCFPAssoc[OptionsPattern[]]:= (
921     CFPAssoc = Association[];
922     Do[
923       (daughterTerms = AllowedNKSLTerms[numE];
924       parentTerms    = AllowedNKSLTerms[numE - 1];
925       Do[
926         (
927         cfps = CFP[{numE, daughter}];
928         cfps = cfps[[2 ;;]];
929         parents = First /@ cfps;
930         Do[
931           (
932           key = {numE, daughter, parent};
933           cfp = If[
934             MemberQ[parents, parent],
935             (
936               idx = Position[parents, parent][[1, 1]];
```

```
              cfps[[idx]][[2]]
            ),
            0
            ];
          CFPAssoc[key] = cfp;
          ),
          {parent, parentTerms}
          ]
        ),
        {daughter, daughterTerms}
        ]
      ),
      {numE, 1, 14}
      ];
    If[OptionValue["Export"],
      (
      CFPAssocfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
      Export[CFPAssocfname, CFPAssoc];
      )
    ];
    Return[CFPAssoc];
  )

  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.

  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.

  CFPTerms[numE, L, S] gives all the daughter and parent terms,
    together with the corresponding coefficients of fractional
    parentage, that correspond to the given total orbital angular
    momentum L and total spin S n the f^n configuration. L being an
    integer, and S being integer or half-integer.

  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}, ...}.
  Only the one-body coefficients for f-electrons are provided.
  In all cases it must be that 1 <= n <= 7.
  ";
  CFPTerms[numE_] := Part[CFPTable, numE]
  CFPTerms[numE_, SL_] :=
    Module[
      {NKterms, CFPconfig},
      NKterms   = {{}};
      CFPconfig = CFPTable[[numE]];
      Map[
        If[StringFreeQ[First[#], SL],
          Null,
          NKterms = Join[NKterms, {#}, 1]
        ] &,
      CFPconfig
```

```
      ];
      NKterms = DeleteCases[NKterms, {}]
    ]
CFPTerms[numE_, L_, S_] :=
Module[
  {NKterms, SL, CFPconfig},
  SL = StringJoin[ToString[2 S + 1], PrintL[L]];
  NKterms = {{}};
  CFPconfig = Part[CFPTable, numE];
  Map[
    If[StringFreeQ[First[#], SL],
      Null,
      NKterms = Join[NKterms, {#}, 1]
    ]&,
  CFPconfig
  ];
  NKterms = DeleteCases[NKterms, {}]
]

(* ################## Coefficients of Fracional Parentage
  ################## *)
(*
  ##############################################################################
   *)

(*
  ##############################################################################
   *)
(* ############################## Spin Orbit
  ################################## *)

SpinOrbit::usage = "SpinOrbit[numE, SL, SpLp, J] returns the LSJ
  reduced matrix element ζ <SL, J|L.S|SpLp, J>. These are given as a
   function of ζ. This function requires that the association
  ReducedV1kTable be defined.";
SpinOrbit[numE_, SL_, SpLp_, J_]:= Module[
  {S, L, Sp, Lp, orbital, sign, prefactor},
  orbital    = 3;
  {S, L}     = FindSL[SL];
  {Sp, Lp}   = FindSL[SpLp];
  prefactor = Sqrt[orbital*(orbital+1)*(2*orbital+1)] * SixJay[{L,
  Lp, 1}, {Sp, S, J}];
  sign       = Phaser[J + L + Sp];
  Return[sign * prefactor * ζ * ReducedV1kTable[{numE, SL, SpLp,
  1}]];
]

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

```
GenerateSpinOrbitTable[nmax_Integer:7, OptionsPattern[]]:= Module[
  {numE, J, SL, SpLp, exportFname},
(
  SpinOrbitTable =
    Table[
      {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
    {numE, 1, nmax},
    {J, MinJ[numE], MaxJ[numE]},
    {SL,   Map[First, AllowedNKSLforJTerms[numE, J]]},
    {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
    ];
  SpinOrbitTable = Association[SpinOrbitTable];

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

(* ############################## Spin Orbit
  ################################## *)
(*
  ##########################################################################
   *)

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

Options[ParseJudd1984] = {"Export" -> False};
ParseJudd1984::usage="This function parses the data from tables 1
  and 2 of Judd from Judd, BR, and MA Suskin. \"Complete Set of
  Orthogonal Scalar Operators for the Configuration f^3\". JOSA B 1,
   no. 2 (1984): 261-65.\"";
ParseJudd1984[OptionsPattern[]]:=(
  ParseJuddTab1[str_] := (
    strR = ToString[str];
    strR = StringReplace[strR, ".5" -> "^(1/2)"];
    num = ToExpression[strR];
    sign = Sign[num];
    num = sign*Simplify[Sqrt[num^2]];
    If[Round[num] == num, num = Round[num]];
    Return[num]);

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

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

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

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

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

```
      Do[(
        opMatrix[Reverse[mKey]] = opMatrix[mKey]
        ),
      {mKey, matrixKeys}
      ];
      juddOperators[{3, opLabel}] = opMatrix),
      {colIndex, 1, Length[normalTable]}
    ];

    (* special case of t2 in f3 *)
    (* this is the same as getting the matrix elements from Judd 1966
    *)
    numE = 3;
    e3Op      = juddOperators[{3, "e_{3}"}];
    t2prime   = juddOperators[{3, "t_{2}^{'}"}];
    prefactor = 1/(70 Sqrt[2]);
    t2Op = (# -> (t2prime[#] + prefactor*e3Op[#])) & /@ Keys[t2prime];
    t2Op = Association[t2Op];
    juddOperators[{3, "t_{2}"}] = t2Op;

    (*Special case of t11 in f3*)
    t11 = juddOperators[{3, "t_{11}"}];
    eβprimeOp = juddOperators[{3, "e_{\\beta}^{'}"}];
    t11primeOp = (# -> (t11[#] + Sqrt[3/385] eβprimeOp[#])) & /@ Keys[
    t11];
    t11primeOp = Association[t11primeOp];
    juddOperators[{3, "t_{11}^{'}"}] = t11primeOp;
    If[OptionValue["Export"],
      (
        (*export them*)
        PrintTemporary["Exporting ..."];
        exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.
    m"}];
        Export[exportFname, juddOperators];
      )
    ];
    Return[juddOperators];
  )

Options[GenerateThreeBodyTables] = {"Export" -> False};
GenerateThreeBodyTables::usage="This function generates the matrix
    elements for the three body operators using the coefficients of
    fractional parentage, including those beyond f^7.";
GenerateThreeBodyTables[nmax_Integer : 14, OptionsPattern[]] := (
  tiKeys = {"t_{2}", "t_{2}^{'}", "t_{3}", "t_{4}", "t_{6}", "t_{7}",
    "t_{8}", "t_{11}", "t_{11}^{'}", "t_{12}", "t_{14}", "t_{15}",
    "t_{16}", "t_{17}", "t_{18}", "t_{19}"};
  TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
  juddOperators = ParseJudd1984[];
  op3MatrixElement::usage = "op3MatrixElement[SL, SpLp, opSymbol]
    returns the value for the reduced matrix element of the operator
    opSymbol for the terms {SL, SpLp} in the f^3 configuration.";
  op3MatrixElement[SL_, SpLp_, opSymbol_] := (
    jOP = juddOperators[{3, opSymbol}];
    key = {SL, SpLp};
```

```
    val = If[MemberQ[Keys[jOP], key],
      jOP[key],
      0];
    Return[val];
    );
  ti::usage = "This is the implementation of formula (2) in Judd &
    Suskin 1984. It computes the matrix elements of ti in f^n by using
     the matrix elements in f3 and the coefficients of fractional
    parentage. If the option \"Fast\" is set to True then the values
    for n>7 are simply computed as the negatives of the values in the
    complementary configuration; this except for t2 and t11 which are
    treated as special cases.";
  Options[ti] = {"Fast" -> True};
  ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
   Module[{nn, S, L, Sp, Lp,
      cfpSL, cfpSpLp,
      parentSL, parentSpLp, tnk, tnks},
    {S, L}   = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    fast     = OptionValue["Fast"];
    numH = 14 - nE;
    If[fast && Not[MemberQ[{"t_{2}","t_{11}"},tiKey]] && nE > 7,
      Return[-tktable[{numH, SL, SpLp, tiKey}]]
    ];
    If[(S == Sp && L == Lp),
     (
      cfpSL   = CFP[{nE, SL}];
      cfpSpLp = CFP[{nE, SpLp}];
      tnks = Table[(
          parentSL   = cfpSL[[nn, 1]];
          parentSpLp = cfpSpLp[[mm, 1]];
          cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
          tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
          ),
          {nn, 2, Length[cfpSL]},
          {mm, 2, Length[cfpSpLp]}
          ];
      tnk = Total[Flatten[tnks]];
      ),
     tnk = 0;
     ];
    Return[ nE / (nE - opOrder) * tnk];];

  (*Calculate the matrix elements of t^i for n up to nmax*)
  tktable = <||>;
  Do[(
    Do[(
      tkValue = Which[numE <= 2,
        (*Initialize n=1,2 with zeros*)
        0,
        numE == 3,
        (*Grab matrix elem in f^3 from Judd 1984*)
        SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
        True,
```

```
        SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2,
  3]]]
        ];
      tktable[{numE, SL, SpLp, opKey}] = tkValue;
      ),
    {SL, AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]},
    {opKey, Append[tiKeys, "e_{3}"]}
    ];
    PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], "
    configuration complete"]];
    ),
  {numE, 1, nmax}
  ];

  (* Now use those matrix elements to determine their sum as weighted
    by their corresponding strengths Ti *)
  ThreeBodyTable = <||>;
  Do[
    Do[
    (
      ThreeBodyTable[{numE, SL, SpLp}] = (
        Sum[(
          If[tiKey == "t_{2}", t2Switch, 1] *
          tktable[{numE, SL, SpLp, tiKey}] *
          TSymbolsAssoc[tiKey] +
          If[tiKey == "t_{2}", 1 - t2Switch, 0] *
          (-tktable[{14 - numE, SL, SpLp, tiKey}]) *
          TSymbolsAssoc[tiKey]
          ),
        {tiKey, tiKeys}
        ]
      );
    ),
    {SL, AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]}
    ];
  PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix
    complete"]];,
  {numE, 1, 7}
  ];

  ThreeBodyTables = Table[(
    terms = AllowedNKSLTerms[numE];
    singleThreeBodyTable =
      Table[
        {SL, SLp} -> ThreeBodyTable[{numE, SL, SLp}],
        {SL, terms},
        {SLp, terms}
      ];
    singleThreeBodyTable  = Flatten[singleThreeBodyTable];
    singleThreeBodyTables = Table[(
        notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
        reps = ConstantArray[0, Length[TSymbols]];
        reps[[notNullPosition]] = 1;
```

```
1261          rep = AssociationThread[TSymbols -> reps];
1262          notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
1263          ),
1264        {notNullSymbol, TSymbols}
1265        ];
1266      singleThreeBodyTables = Association[singleThreeBodyTables];
1267      numE -> singleThreeBodyTables),
1268      {numE, 1, 7}];
1269
1270    ThreeBodyTables = Association[ThreeBodyTables];
1271    If[OptionValue["Export"], (
1272      threeBodyTablefname = FileNameJoin[{moduleDir, "data", "
          ThreeBodyTable.m"}];
1273      Export[threeBodyTablefname, ThreeBodyTable];
1274      threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "
          ThreeBodyTables.m"}];
1275      Export[threeBodyTablesfname, ThreeBodyTables];
1276      )
1277     ];
1278    Return[{ThreeBodyTable, ThreeBodyTables}];)
1279
1280    ScalarOperatorProduct::usage="ScalarOperatorProduct[op1, op2, numE]
        calculated the innerproduct between the two scalar operators op1
        and op2.";
1281    ScalarOperatorProduct[op1_, op2_, numE_] := Module[
1282      {terms, S, L, factor, term1, term2},
1283      (
1284      terms = AllowedNKSLTerms[numE];
1285      Simplify[
1286        Sum[(
1287          {S, L} = FindSL[term1];
1288          factor = TPO[S, L];
1289          factor * op1[{term1, term2}] * op2[{term2, term1}]
1290          ),
1291        {term1, terms},
1292        {term2, terms}
1293        ]
1294      ]
1295      )
1296    ];
1297
1298    (* ########################## Three Body Operators
        ############################## *)
1299    (*
        ################################################################################
         *)
1300
1301    (*
        ################################################################################
         *)
1302    (* ########################## Reduced SOO and ECSO
        ############################## *)
1303
1304    ReducedT11inf2::usage="ReducedT11inf2[SL, SpLp] returns the reduced
        matrix element of the scalar component of the double tensor T11
```

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

ReducedT11infn::usage="ReducedT11infn[n, SL, SpLp] calculate the
  reduced matrix element of the T11 operator for the f^n
  configuration corresponding to the terms SL and SpLp. This
  operator corresponds to the inter-electron interaction between the
   spin of one electron and the orbital angular momentum of another.

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.\"
";
ReducedT11infn[numE_, SL_, SpLp_]:= Module[
  {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp,
  parentSL, parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
  {spin, orbital} = {1/2, 3};
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  t = 1;
  cfpSL    = CFP[{numE, SL}];
  cfpSpLp  = CFP[{numE, SpLp}];
  Tnkk =
    Sum[(
      parentSL = cfpSL[[idx2, 1]];
      parentSpLp = cfpSpLp[[idx1, 1]];
      {Sb, Lb} = FindSL[parentSL];
      {Sbp, Lbp} = FindSL[parentSpLp];
      phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
```

```
        (
          phase *
          cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
          SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
          SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
          T11Table[{numE - 1, parentSL, parentSpLp}]
        )
      ),
      {idx1, 2, Length[cfpSpLp]},
      {idx2, 2, Length[cfpSL]}
      ];
    Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
    Return[Tnkk];
    ];

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

ReducedSOOandECSOinf2::usage="ReducedSOOandECSOinf2[SL, SpLp]
  returns the reduced matrix element corresponding to the operator (
  T11 + t11 - a13 * z13 / 6) for the terms {SL, SpLp}. This
  combination of operators corresponds to the spin-other-orbit plus
  ECSO interaction.

The T11 operator corresponds to the spin-other-orbit interaction,
  and the t11 operator (associated with electrostatically-correlated
   spin-orbit) originates from configuration interaction analysis.
```

```
  To their sum the a facor proportional to operator z13 is
  subtracted since its effect is seen as redundant to the spin-orbit
   interaction. The factor of 1/6 is not on Judd's 1966 paper, but
  it is on \"Chen, Xueyuan, Guokui Liu, Jean Margerie, and Michael F
   Reid. \"A Few Mistakes in Widely Used Data Files for Fn
  Configurations Calculations.\" Journal of Luminescence 128, no. 3
  (2008): 421-27\".

The values for the reduced matrix elements of z13 are obtained from
  Table IX of the same paper. The value for a13 is also from that
  paper.";
ReducedSOOandECSOinf2[SL_, SpLp_] :=
Module[{pairPosition, f2TermPairs, a13, z13, redSOOandECSOinf2},
  f2TermPairs = {
    {"1S", "3P"}, {"3P", "1S"},
    {"3P", "3P"}, {"3P", "1D"},
    {"1D", "3P"}, {"1D", "3F"},
    {"3F", "1D"}, {"3F", "3F"},
    {"3F", "1G"}, {"1G", "3F"},
    {"1G", "3H"}, {"3H", "1G"},
    {"3H", "3H"}, {"3H", "1I"},
    {"1I", "3H"}};
  a13 = (-33 M0 + 3 M2 + 15/11 M4 -
        6 P0 + 3/2 (35 P2 + 77 P4 + 143 P6));
  z13 = {2, 2,
    1,
    1/Sqrt[1080] (-90),
    1/Sqrt[1080] (-90),
    Sqrt[2/405] 45,
    Sqrt[2/405] 45,
    Sqrt[14],
    1/Sqrt[891] (-99),
    1/Sqrt[891] (-99),
    990/Sqrt[98010],
    990/Sqrt[98010],
    55/Sqrt[55],
    -2574/Sqrt[1019304],
    -2574/Sqrt[1019304]};
  pairPosition = Position[f2TermPairs, {SL, SpLp}];
  If[Length[pairPosition] == 0,
    Return[0],
    pairPosition = pairPosition[[1, 1]]
  ];

  redSOOandECSOinf2 = (
      ReducedT11inf2[SL, SpLp] +
      Reducedt11inf2[SL, SpLp] -
      a13 / 6 * z13[[pairPosition]]
  );
  redSOOandECSOinf2 = SimplifyFun[redSOOandECSOinf2];
  Return[redSOOandECSOinf2];
  ];

ReducedSOOandECSOinfn::usage="ReducedSOOandECSOinfn[numE, SL, SpLp]
  calculates the reduced matrix elements of the (spin-other-orbit +
```

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

  GenerateSOOandECSOLSTable::usage="GenerateSOOandECSOLSTable[nmax]
    generates the LS reduced matrix elements of the spin-other-orbit +
     ECSO for the f^n configurations up to n=nmax. The values for n=1
    and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\
    " Physical Review 169, no. 1 (1968): 130.\", and the values for n
    >2 are calculated recursively using equation (4) of that same
    paper. The values are then exported to a file \"
    ReducedSOOandECSOLSTable.m\" in the data folder of this module.
    The values are also returned as an association.";
  Options[GenerateSOOandECSOLSTable] = {"Progress" -> True, "Export"
    -> True};
  GenerateSOOandECSOLSTable[nmax_Integer, OptionsPattern[]]:= (
    If[And[OptionValue["Progress"], frontEndAvailable],
      (
```

```
    numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
numE]]^2, {numE, 1, nmax}]];
    counters   = Association[Table[numE->0, {numE, 1, nmax}]];
    totalIters = Total[Values[numItersai[[1;;nmax]]]];
    template1  = StringTemplate["Iteration `numiter` of `totaliter
`"];
    template2  = StringTemplate["`remtime` min remaining"];
template3 = StringTemplate["Iteration speed = `speed` ms/it"];
    template4  = StringTemplate["Time elapsed = `runtime` min"];
    progBar = PrintTemporary[
      Dynamic[
        Pane[
          Grid[{
                {Superscript["f", numE]},
                {template1[<|"numiter"->numiter, "totaliter"->
totalIters|>]},
                {template4[<|"runtime"->Round[QuantityMagnitude[
UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                {template2[<|"remtime"->Round[QuantityMagnitude[
UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
]], 0.1]|>]},
                {template3[<|"speed"->Round[QuantityMagnitude[Now-
startTime, "ms"]/(numiter), 0.01]|>]}, {ProgressIndicator[Dynamic[
numiter], {1, totalIters}]}
              },
              Frame->All
          ],
          Full,
          Alignment->Center
        ]
      ]
    ];
  )
];
SOOandECSOLSTable = <||>;
numiter   = 1;
startTime = Now;
Do[
  (
    numiter+= 1;
    SOOandECSOLSTable[{numE, SL, SpLp}] = Which[
      numE==1,
      0,
      numE==2,
      SimplifyFun[ReducedSOOandECSOinf2[SL, SpLp]],
      True,
      SimplifyFun[ReducedSOOandECSOinfn[numE,  SL, SpLp]]
    ];
  ),
{numE, 1, nmax},
{SL, AllowedNKSLTerms[numE]},
{SpLp, AllowedNKSLTerms[numE]}
];
If[And[OptionValue["Progress"], frontEndAvailable],
  NotebookDelete[progBar]];
```

```
    If[OptionValue["Export"],
      (fname = FileNameJoin[{moduleDir, "data", "
    ReducedSOOandECSOLSTable.m"}];
      Export[fname, SOOandECSOLSTable];
      )
    ];
    Return[SOOandECSOLSTable];
  );

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

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

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

  ReducedT22infn::usage="ReducedT22infn[n, SL, SpLp] calculates the
    reduced matrix element of the T22 operator for the f^n
    configuration corresponding to the terms SL and SpLp. This is the
    operator corresponding to the inter-electron between spin.
  It does this by using equation (4) of \"Judd, BR, HM Crosswhite, and
     Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
    Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
```

```
";
ReducedT22infn[numE_, SL_, SpLp_]:= Module[
  {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp,
  parentSL, parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
  {spin, orbital} = {1/2, 3};
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  t = 2;
  cfpSL    = CFP[{numE, SL}];
  cfpSpLp  = CFP[{numE, SpLp}];
  Tnkk =
    Sum[(
      parentSL = cfpSL[[idx2, 1]];
      parentSpLp = cfpSpLp[[idx1, 1]];
      {Sb, Lb} = FindSL[parentSL];
      {Sbp, Lbp} = FindSL[parentSpLp];
      phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
      (
        phase *
        cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
        SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
        SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
        T22Table[{numE - 1, parentSL, parentSpLp}]
      )
    ),
    {idx1, 2, Length[cfpSpLp]},
    {idx2, 2, Length[cfpSL]}
    ];
  Tnkk *= numE / (numE - 2) * Sqrt[TPO[S,Sp,L,Lp]];
  Return[Tnkk];
  ];

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

```
        template4 = StringTemplate["Time elapsed = `runtime` min"];
        progBar = PrintTemporary[
          Dynamic[
            Pane[
              Grid[{{Superscript["f", numE]},
                    {template1[<|"numiter"->numiter, "totaliter"->totalIters|>]},
                    {template4[<|"runtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
                    {template2[<|"remtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"]], 0.1]|>]},
                    {template3[<|"speed"->Round[QuantityMagnitude[Now-startTime, "ms"]/(numiter), 0.01]|>]},
                    {ProgressIndicator[Dynamic[numiter], {1, totalIters}]}},
                    Frame->All],
                    Full,
                    Alignment->Center]
                ]
              ];
      )
    ];
    T22Table = <||>;
    startTime = Now;
    numiter = 1;
    Do[
      (
        numiter+= 1;
        T22Table[{numE, SL, SpLp}] = Which[
          numE==1,
          0,
          numE==2,
          SimplifyFun[ReducedT22inf2[SL, SpLp]],
          True,
          SimplifyFun[ReducedT22infn[numE,  SL, SpLp]]
        ];
      ),
    {numE, 1, nmax},
    {SL,   AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]}
    ];
    If[And[OptionValue["Progress"],frontEndAvailable],
      NotebookDelete[progBar]
    ];
    If[OptionValue["Export"],
      (
        fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
        Export[fname, T22Table];
      )
    ];
    Return[T22Table];
  );

```

```
SpinSpin::usage="SpinSpin[n, SL, SpLp, J] returns the matrix element
   <|SL,J|spin-spin|SpLp,J|> for the spin-spin operator within the
  configuration f^n. This matrix element is independent of MJ. This
  is obtained by querying the relevant reduced matrix element by
  querying the association T22Table and putting in the adequate
  phase and 6-j symbol.

This is calculated according to equation (3) in \"Judd, BR, HM
  Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
  Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
   130.\"
\".
";
SpinSpin[numE_, SL_, SpLp_, J_] := Module[
  {S, L, Sp, Lp, α, val},
  α = 2;
  {S, L} = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  val = (
          Phaser[Sp + L + J] *
          SixJay[{Sp, Lp, J}, {L, S, α}] *
          T22Table[{numE, SL, SpLp}]
        );
  Return[val]
  ];

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

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

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

SOOandECSO::usage="SOOandECSO[n, SL, SpLp, J] returns the matrix
  element <|SL,J|spin-spin|SpLp,J|> for the combined effects of the
  spin-other-orbit interaction and the electrostatically-correlated-
  spin-orbit (which originates from configuration interaction
  effects) within the configuration f^n. This matrix element is
  independent of MJ. This is obtained by querying the relevant
  reduced matrix element by querying the association
  SOOandECSOLSTable and putting in the adequate phase and 6-j symbol
  . The SOOandECSOLSTable puts together the reduced matrix elements
  from three operators.

This is calculated according to equation (3) in \"Judd, BR, HM
  Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
  Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
   130.\".
";
SOOandECSO[numE_, SL_, SpLp_, J_]:= Module[
  {S, Sp, L, Lp, α, val},
  α = 1;
  {S, L}   = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  val = (
          Phaser[Sp + L + J] *
          SixJay[{Sp, Lp, J}, {L, S, α}] *
          SOOandECSOLSTable[{numE, SL, SpLp}]
        );
  Return[val];
]

Prescaling = {P2 -> P2/225, P4 -> P4/1089, P6 -> 25 * P6 / 184041};
GenerateSOOandECSOTable::usage="GenerateSOOandECSOTable[nmax]
  generates the matrix elements in the |LSJ> basis for the (spin-
  other-orbit + electrostatically-correlated-spin-orbit) operator.
  It returns an association where the keys are of the form {n, SL,
  SpLp, J}. If the option \"Export\" is set to True then the
  resulting object is saved to the data folder. Since this is a
  scalar operator, there is no MJ dependence. This dependence only
  comes into play when the crystal field contribution is taken into
  account.";
Options[GenerateSOOandECSOTable] = {"Export"->False}
GenerateSOOandECSOTable[nmax_, OptionsPattern[]]:= (
  SOOandECSOTable = <||>;
  Do[
```

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

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

  (*
    ################################################################################
     *)
  (* ########################## Magnetic Interactions
    ########################## *)

  MagneticInteractions::usage="MagneticInteractions[{numE, SLJ, SLJp,
    J}] returns the matrix element of the magnetic interaction between
     the terms SLJ and SLJp in the f^n configuration. The interaction
    is given by the sum of the spin-spin interaction and the SOO and
    ECSO interactions. The spin-spin interaction is given by the
    function SpinSpin[{numE, SLJ, SLJp, J}]. The SOO and ECSO
    interactions are given by the function SOOandECSO[{numE, SLJ, SLJp
    , J}]. The function requires chenDeltas to be loaded into the
    session. The option \"ChenDeltas\" can be use to include or
    exclude the Chen deltas from the calculation. The default is to
    exclude them.";
  Options[MagneticInteractions] = {"ChenDeltas" -> False};
  MagneticInteractions[{numE_, SLJ_, SLJp_, J_}, OptionsPattern[]] :=
    (
      key = {numE, SLJ, SLJp, J};
      ss = \[Sigma]SS * SpinSpinTable[key];
      sooandecso = SOOandECSOTable[key];
      total = ss + sooandecso;
      total = SimplifyFun[total];
      If[
        Not[OptionValue["ChenDeltas"]],
        Return[total]
      ];
      (* In the type A errors the wrong values are different *)
      If[MemberQ[Keys[chenDeltas["A"]], {numE, SLJ, SLJp}],
        (
          {S, L} = FindSL[SLJ];
```

```
1755            {Sp, Lp} = FindSL[SLJp];
1756            phase    = Phaser[Sp + L + J];
1757            Msixjay  = SixJay[{Sp, Lp, J},{L, S, 2}];
1758            Psixjay  = SixJay[{Sp, Lp, J},{L, S, 1}];
1759            {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ,
        SLJp}]["wrong"];
1760            total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1761            total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1762            total  = total /. Prescaling;
1763            total  = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
1764          )
1765        ];
1766        (* In the type B errors the wrong values are zeros all around *)
1767        If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
1768          (
1769            {S, L} = FindSL[SLJ];
1770            {Sp, Lp} = FindSL[SLJp];
1771            phase = Phaser[Sp + L + J];
1772            Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
1773            Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
1774            {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
1775            total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1776            total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1777            total  = total /. Prescaling;
1778            total  = wChErrB * total + (1 - wChErrB) * (ss + sooandecso)
1779          )
1780        ];
1781        Return[total];
1782      )
1783
1784    (* ########################## Magnetic Interactions
        ########################## *)
1785    (*
        ############################################################################
         *)
1786
1787    (*
        ############################################################################
         *)
1788    (* ############################## Crystal Field
        ############################### *)
1789
1790    Cqk::usage = "Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp].";
1791    Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
1792      {S, Sp, L, Lp, orbital, val},
1793      orbital = 3;
1794      {S, L}   = FindSL[NKSL];
1795      {Sp, Lp} = FindSL[NKSLp];
1796      f1  = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
1797      val =
1798        If[f1==0,
1799          0,
1800          (
1801            f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
1802            If[f2==0,
```

```
            0,
            (
              f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
              If[f3==0,
                0,
                (
                  (
                    Phaser[J - M + S + Lp + J + k] *
                    Sqrt[TPO[J, Jp]] *
                    f1 *
                    f2 *
                    f3 *
                    Ck[orbital, k]
                  )
                )
              ]
            )
          ]
        )
      ];
    val
    ]

Bqk[q_, 2] := {B02/2, B12, B22}[[q + 1]];
Bqk[q_, 4] := {B04/2, B14, B24, B34, B44}[[q + 1]];
Bqk[q_, 6] := {B06/2, B16, B26, B36, B46, B56, B66}[[q + 1]];

Sqk[q_, 2] := {Sm22, Sm12, S02,  S12,  S22}[[q + 3]];
Sqk[q_, 4] := {Sm44, Sm34, Sm24, Sm14, S04,  S14,  S24, S34, S44}[[q
   + 5]];
Sqk[q_, 6] := {Sm66, Sm56, Sm46, Sm36, Sm26, Sm16, S06, S16, S26,
  S36, S46, S56, S66}[[q + 7]];

CrystalField::usage = "CrystalField[n, NKSL, J, M, NKSLp, Jp, Mp]
  gives the general expression for the matrix element of the crystal
   field Hamiltonian parametrized with Bqk and Sqk coefficients as a
   sum over spherical harmonics Cqk.

Sometimes this expression only includes Bqk coefficients, see for
  example eqn 6-2 in Wybourne (1965), but one may also split the
  coefficient into real and imaginary parts as is done here, in an
  expression that is patently Hermitian.";
CrystalField[numE_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := (
  Sum[
    (
      cqk  = Cqk[numE,  q, k, NKSL, J, M, NKSLp, Jp, Mp];
      cmqk = Cqk[numE, -q, k, NKSL, J, M, NKSLp, Jp, Mp];
      Bqk[q, k]   * (cqk + (-1)^q * cmqk) +
      I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
      ),
  {k, {2, 4, 6}},
  {q, 0, k}
  ]
)

```

```
TotalCFIters::usage = "TotalIters[i, j] returns total number of function evaluations for calculating all the matrix elements for the \!\(\*SuperscriptBox[\(f\), \(i\)]\) to the \!\(\*SuperscriptBox[\(f\), \(j\)]\) configurations.";
TotalCFIters[i_, j_] := (
  numIters = {196, 8281, 132496, 1002001, 4008004, 9018009, 11778624};
  Return[Total[numIters[[i ;; j]]]];
  )

GenerateCrystalFieldTable::usage = "GenerateCrystalFieldTable[{numEs}] computes the matrix values for the crystal field interaction for f^n configurations the given list of numE in  numEs. The function calculates the association CrystalFieldTable with keys of the form {numE, NKSL, J, M, NKSLp, Jp, Mp}. If the option \"Export\" is set to True, then the result is exported to the data subfolder for the folder in which this package is in. If the option \"Progress\" is set to True then an interactive progress indicator is shown. If \"Compress\" is set to true the exported values are compressed when exporting.";
Options[GenerateCrystalFieldTable] = {"Export" -> False, "Progress" -> True, "Compress" -> True}
GenerateCrystalFieldTable[numEs_List:{1,2,3,4,5,6,7}, OptionsPattern[]]:= (
  ExportFun =
  If[OptionValue["Compress"],
    ExportMZip,
    Export
  ];
  numiter = 1;
  template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
  template2 = StringTemplate["`remtime` min remaining"];
  template3 = StringTemplate["Iteration speed = `speed` ms/it"];
  template4 = StringTemplate["Time elapsed = `runtime` min"];
  totalIter = Total[TotalCFIters[#, #] & /@ numEs];
  freebies = 0;
  startTime = Now;
  If[And[OptionValue["Progress"], frontEndAvailable],
    progBar = PrintTemporary[
      Dynamic[
        Pane[
          Grid[
            {
              {Superscript["f", numE]},
              {template1[<|"numiter" -> numiter, "totaliter" -> totalIter|>]},
              {template4[<|"runtime" -> Round[QuantityMagnitude[UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
              {template2[<|"remtime" -> Round[QuantityMagnitude[UnitConvert[(Now - startTime)/(numiter - freebies) * (totalIter - numiter), "min"]], 0.1]|>]},
              {template3[<|"speed" -> Round[QuantityMagnitude[Now - startTime, "ms"]/(numiter-freebies), 0.01]|>]},
              {ProgressIndicator[Dynamic[numiter], {1, totalIter}]}
            },
```

```
1885              Frame -> All
1886              ],
1887            Full,
1888            Alignment -> Center
1889            ]
1890          ]
1891        ];
1892      ];
1893      Do[
1894        (
1895          exportFname = FileNameJoin[{moduleDir, "data", "
      CrystalFieldTable_f"<>ToString[numE]<>".m"}];
1896          If[FileExistsQ[exportFname],
1897            Print["File exists, skipping ..."];
1898            numiter+=  TotalCFIters[numE, numE];
1899            freebies+= TotalCFIters[numE, numE];
1900            Continue[];
1901          ];
1902          CrystalFieldTable = <||>;
1903          Do[
1904            (
1905              numiter+= 1;
1906              CrystalFieldTable[{numE, NKSL, J, M, NKSLp, Jp, Mp}] =
      CrystalField[numE, NKSL, J, M, NKSLp, Jp, Mp];
1907            ),
1908          {J, MinJ[numE], MaxJ[numE]},
1909          {Jp, MinJ[numE], MaxJ[numE]},
1910          {M, AllowedMforJ[J]},
1911          {Mp, AllowedMforJ[Jp]},
1912          {NKSL , First /@ AllowedNKSLforJTerms[numE, J]},
1913          {NKSLp, First /@ AllowedNKSLforJTerms[numE, Jp]}
1914          ];
1915          If[And[OptionValue["Progress"],frontEndAvailable],
1916            NotebookDelete[progBar]
1917          ];
1918          If[OptionValue["Export"],
1919            (
1920              Print["Exporting to file "<>ToString[exportFname]];
1921              ExportFun[exportFname, CrystalFieldTable];
1922            )
1923          ];
1924        ),
1925      {numE, numEs}
1926      ]
1927    )
1928
1929    (* ############################ Crystal Field
        ############################### *)
1930    (*
        ########################################################################
         *)
1931
1932    (*
        ########################################################################
         *)
```

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

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

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

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

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

```
      "211" -> 10/5,
      "220" -> 12/5,
      "221" -> 13/5,
      "222" -> 15/5
    }
  ];

  CasimirSO7::usage = "CasimirSO7[SL, SpLp] returns the LS reduced
    matrix element of the configuration interaction term corresponding
     to the Casimir operator of R7.";
  CasimirSO7[{SL_, SpLp_}] := (
    Wlabel = FindNKLSTerm[SL][[1]][[3]];
    If[SL==SpLp,
      γ * GSO7W[Wlabel],
      0
    ]
  )

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

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

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

  Options[JJBlockMatrix] = {"Sparse"->True, "ChenDeltas"->False};
  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.";
2017  JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]]:= Module[
2018    {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
2019    SLterm, SpLpterm,
2020    MJ, MJp,
2021    subKron, matValue, eMatrix},
2022    (
2023      NKSLJMs  = AllowedNKSLJMforJTerms[numE, J];
2024      NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
2025      eMatrix =
2026        Table[
2027          (*Condition for a scalar matrix op*)
2028          SLterm   =  NKSLJM[[1]];
2029          SpLpterm = NKSLJMp[[1]];
2030          MJ       =  NKSLJM[[3]];
2031          MJp      = NKSLJMp[[3]];
2032          subKron  =
2033            (
2034              KroneckerDelta[J, Jp] *
2035              KroneckerDelta[MJ, MJp]
2036            );
2037          matValue =
2038            If[subKron==0,
2039              0,
2040                (
2041                  ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2042                  ElectrostaticConfigInteraction[{SLterm, SpLpterm}] +
2043                  SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2044                  MagneticInteractions[{numE, SLterm, SpLpterm, J}, "
    ChenDeltas" -> OptionValue["ChenDeltas"]] +
2045                  ThreeBodyTable[{numE, SLterm, SpLpterm}]
2046                )
2047            ];
2048          matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp
    }];
2049          matValue,
2050        {NKSLJMp, NKSLJMps},
2051        {NKSLJM , NKSLJMs}
2052        ];
2053     If[OptionValue["Sparse"],
2054       eMatrix = SparseArray[eMatrix]
2055     ];
2056     Return[eMatrix]
2057  )
2058  ];
2059
2060  EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some
    point may be used to redefine states used in basis.";
2061  EnergyStates[numE_, J_]:= AllowedNKSLJMforJTerms[numE, J];
2062
2063  JJBlockMatrixFileName::usage = "JJBlockMatrixFileName[numE] gives
    the filename for the energy matrix table for an atom with numE f-
    electrons. The function admits an optional parameter \"
```

```
    FilenameAppendix\" which can be used to modify the filename.";
Options[JJBlockMatrixFileName] = {"FilenameAppendix" -> ""}
JJBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
  fileApp = OptionValue["FilenameAppendix"];
  fname = FileNameJoin[{moduleDir,
      "hams",
      StringJoin[{"f", ToString[numE], "_JJBlockMatrixTable",
  fileApp ,".m"}]}];
  Return[fname];
  );

Options[TabulateJJBlockMatrixTable] = {"Sparse"->True, "ChenDeltas"
  ->False};
TabulateJJBlockMatrixTable::usage = "TabulateJJBlockMatrixTable[numE
  , I] returns a list with three elements {JJBlockMatrixTable,
  EnergyStatesTable, AllowedM}. JJBlockMatrixTable is an association
   with keys equal to lists of the form {numE, J, Jp}.
  EnergyStatesTable is an association with keys equal to lists of
  the form {numE, J}. AllowedM is another association with keys
  equal to lists of the form {numE, J} and values equal to lists
  equal to the corresponding values of MJ. It's unnecessary (and it
  won't work in this implementation) to give numE > 7 given the
  equivalency between electron and hole configurations.";
TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]]:= (
  JJBlockMatrixTable = <||>;
  totalIterations = Length[AllowedJ[numE]]^2;
  template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
  template2 = StringTemplate["`remtime` min remaining"];
  template4 = StringTemplate["Time elapsed = `runtime` min"];
  numiter   = 0;
  startTime = Now;
  If[$FrontEnd =!= Null,
    (
      temp = PrintTemporary[
        Dynamic[
          Grid[
            {
              {template1[<|"numiter"->numiter, "totaliter"->
  totalIterations|>]},
              {template2[<|"remtime"->Round[QuantityMagnitude[
  UnitConvert[(Now-startTime)/(Max[1,numiter])*(totalIterations-
  numiter), "min"]], 0.1]|>]},
              {template4[<|"runtime"->Round[QuantityMagnitude[
  UnitConvert[(Now-startTime), "min"]], 0.1]|>]},
              {ProgressIndicator[numiter, {1, totalIterations}]}
            }
          ]
      ]
      ];
    )
  ];
  Do[
    (
      JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp,
   CFTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas" ->
```

```
    OptionValue["ChenDeltas"]];
2102        numiter += 1;
2103      ),
2104    {Jp, AllowedJ[numE]},
2105    {J, AllowedJ[numE]}
2106    ];
2107    If[$FrontEnd =!= Null,
2108      NotebookDelete[temp]
2109    ];
2110    Return[JJBlockMatrixTable];
2111  )
2112
2113  Options[TabulateManyJJBlockMatrixTables] = {"Overwrite"->False, "
      Sparse"->True, "ChenDeltas"->False, "FilenameAppendix"-> "", "
      Compressed" -> False};
2114  TabulateManyJJBlockMatrixTables::usage = "
      TabulateManyJJBlockMatrixTables[{n1, n2, ...}] calculates the
      tables of matrix elements for the requested f^n_i configurations.
      The function does not return the matrices themselves. It instead
      returns an association whose keys are numE and whose values are
      the filenames where the output of TabulateJJBlockMatrixTables was
      saved to. When these files are loaded with Get, the following
      three symbols are thus defined: JJBlockMatrixTable,
      EnergyStatesTable, and AllowedM.
2115  JJBlockMatrixTable is an association whose keys are of the form {n,
      J, Jp} and whose values are matrix elements.";
2116  TabulateManyJJBlockMatrixTables[ns_, OptionsPattern[]]:= (
2117    overwrite = OptionValue["Overwrite"];
2118    fNames = <||>;
2119    fileApp = OptionValue["FilenameAppendix"];
2120    ExportFun = If[OptionValue["Compressed"], ExportMZip, Export];
2121    Do[
2122      (
2123        CFdataFilename = FileNameJoin[{moduleDir, "data", "
      CrystalFieldTable_f"<>ToString[numE]<>".zip"}];
2124        PrintTemporary["Importing CrystalFieldTable from ",
      CFdataFilename, " ..."];
2125        CrystalFieldTable = ImportMZip[CFdataFilename];
2126
2127        PrintTemporary["#------- numE = ", numE, " -------#"];
2128        exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix"
      -> fileApp];
2129        fNames[numE] = exportFname;
2130        If[FileExistsQ[exportFname] && Not[overwrite],
2131          Continue[]
2132        ];
2133        JJBlockMatrixTable = TabulateJJBlockMatrixTable[numE,
      CrystalFieldTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas"
      -> OptionValue["ChenDeltas"]];
2134        If[FileExistsQ[exportFname]&&overwrite,
2135          DeleteFile[exportFname]
2136        ];
2137        ExportFun[exportFname, JJBlockMatrixTable];
2138
2139        ClearAll[CrystalFieldTable];
```

```
      ),
    {numE, ns}
    ];
  Return[fNames];
  )

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

Options[SimplerSymbolicHamMatrix]={
```

```
  "Export"->True,
  "PrependToFilename"->"",
  "EorF"->"F",
  "Overwrite" -> False,
  "Return" -> True};
SimplerSymbolicHamMatrix::usage="SimplerSymbolicHamMatrix[numE,
    simplifier] is a simple addition to HamMatrixAssembly that applies
     a given simplification to the full hamiltonian. Simplifier is a
    list of replacement rules. If the option \"Export\" is set to True
    , then the function also exports the resulting sparse array to the
     ./hams/ folder. The option \"PrependToFilename\" can be used to
    append a string to the filename to which the function may exports
    to. The option \"Return\" can be used to choose whether the
    function returns the matrix or not.";
SimplerSymbolicHamMatrix[numE_Integer, simplifier_List, OptionsPattern
    []]:=Module[
  {thisHam,eTofs,fname},
  (
    fname=FileNameJoin[{moduleDir,"hams",OptionValue["
    PrependToFilename"]<>"SymbolicMatrix-f"<>ToString[numE]<>".m"}];
    If[FileExistsQ[fname] && Not[OptionValue["Overwrite"]],
      (
        If[OptionValue["Return"],
          (
            Print["File ",fname," already exists, and option \"
    Overwrite\" is set to False, loading file ..."];
            thisHam = Import[fname];
            Return[thisHam];
          ),
          (
            Print["File ",fname," already exists, skipping ..."];
          Return[Null];
          )
        ]
      )
    ];
    thisHam=HamMatrixAssembly[numE];
    thisHam=ReplaceInSparseArray[thisHam,simplifier];
    If[OptionValue["Export"],
    (
      Print["Exporting to file ",fname];
      Export[fname,thisHam]
    )
    ];
    If[OptionValue["Return"],
      Return[thisHam],
      Return[Null]
    ];
  )
]

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

```
    *)

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

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

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

PrintSLJ::usage = "Given a list with three elements {S, L, J} this
  function returns a symbol where the spin multiplicity is presented
   as a superscript, the orbital angular momentum as its
  corresponding spectroscopic letter, and J as a subscript. Function
   does not check to see if the given J is compatible with the given
   S and L.";
PrintSLJ[SLJ_] :=
  RowBox[{SuperscriptBox[" ", 2 SLJ[[1]] + 1],
    SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]}] // DisplayForm;

PrintSLJM::usage = "Given a list with four elements {S, L, J, MJ}
  this function returns a symbol where the spin multiplicity is
  presented as a superscript, the orbital angular momentum as its
  corresponding spectroscopic letter, and {J, MJ} as a subscript. No
   attempt is made to guarantee that the given input is consistent."
  ;
PrintSLJM[SLJM_] :=
  RowBox[{SuperscriptBox[" ", 2 SLJM[[1]] + 1],
    SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]}] //
   DisplayForm;

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

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

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

AllowedSLTerms::usage = "AllowedSLTerms[numE] returns a list with
  the allowed terms in the f^numE configuration, the terms are given
   as lists in the format {S, L}. This list may have redundancies
  which are compatible with the degeneracies that might correspond
  to the given case.";
AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE,
  14-numE]]]

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

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

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

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.";
AllowedSLJTerms[numE_] :=
  Module[{idx1, allowedSL, allowedSLJ},
    allowedSL = AllowedSLTerms[numE];
    allowedSLJ = {};
    For[
      idx1 = 1,
      idx1 <= Length[allowedSL],
      termSL = allowedSL[[idx1]];
      termsSLJ =
        Table[
          {termSL[[1]], termSL[[2]], J},
        {J,Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
        ];
      allowedSLJ = Join[allowedSLJ, termsSLJ];
      idx1++
    ];
    SortBy[allowedSLJ, Last]
  ]

AllowedNKSLJTerms::usage = "AllowedNKSLJTerms[numE] returns a list
  with the allowed {SL, J} terms in the f^n configuration, the terms
```

```
    are given as lists in the format {SL, J} where SL is a string in
   spectroscopic notation.";
2302 AllowedNKSLJTerms[numE_] :=
2303   Module[{allowedSL, allowedNKSL, allowedSLJ, nn},
2304     allowedNKSL = AllowedNKSLTerms[numE];
2305     allowedSL = AllowedSLTerms[numE];
2306     allowedSLJ = {};
2307     For[
2308       nn = 1,
2309       nn <= Length[allowedSL],
2310       (
2311         termSL = allowedSL[[nn]];
2312         termNKSL = allowedNKSL[[nn]];
2313         termsSLJ =
2314           Table[{termNKSL, J},
2315           {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2316           ];
2317         allowedSLJ = Join[allowedSLJ, termsSLJ];
2318         nn++
2319       )
2320     ];
2321     SortBy[allowedSLJ, Last]
2322   ]
2323
2324 AllowedNKSLforJTerms::usage = "AllowedNKSLforJTerms[numE, J] gives
   the terms that correspond to the given total angular momentum J in
    the f^n configuration. The result is a list whose elements are
   lists of length 2, the first element being the SL term in
   spectroscopic notation, and the second element being J.";
2325 AllowedNKSLforJTerms[numE_, J_] := Module[
2326     {allowedSL, allowedNKSL, allowedSLJ, nn, termSL, termNKSL,
   termsSLJ},
2327     allowedNKSL = AllowedNKSLTerms[numE];
2328     allowedSL = AllowedSLTerms[numE];
2329     allowedSLJ = {};
2330     For[
2331       nn = 1,
2332       nn <= Length[allowedSL],
2333       (
2334         termSL = allowedSL[[nn]];
2335         termNKSL = allowedNKSL[[nn]];
2336         termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[
   termSL],
2337           {{termNKSL, J}},
2338           {}
2339           ];
2340         allowedSLJ = Join[allowedSLJ, termsSLJ];
2341         nn++
2342       )
2343     ];
2344     Return[allowedSLJ]
2345   ];
2346
2347 AllowedSLJMTerms::usage = "AllowedSLJMTerms[numE] returns a list
   with all the states that correspond to the configuration f^n. A
```

```
    list is returned whose elements are lists of the form {S, L, J, MJ
    }.";
2348 AllowedSLJMTerms[numE_] := Module[
2349     {allowedSLJ, allowedSLJM, termSLJ, termsSLJM, nn},
2350     allowedSLJ = AllowedSLJTerms[numE];
2351     allowedSLJM = {};
2352     For[
2353       nn = 1,
2354       nn <= Length[allowedSLJ],
2355       nn++,
2356       (
2357         termSLJ = allowedSLJ[[nn]];
2358         termsSLJM =
2359           Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
2360           {M, - termSLJ[[3]], termSLJ[[3]]}
2361           ];
2362         allowedSLJM = Join[allowedSLJM, termsSLJM];
2363       )
2364     ];
2365     Return[SortBy[allowedSLJM, Last]];
2366     ]
2367
2368 AllowedNKSLJMforJMTerms::usage = "AllowedNKSLJMforJMTerms[numE, J,
    MJ] returns a list with all the terms that contain states of the f
    ^n configuration that have a total angular momentum J, and a
    projection along the z-axis MJ. The returned list has elements of
    the form {SL (string in spectroscopic notation), J, MJ}.";
2369 AllowedNKSLJMforJMTerms[numE_, J_, MJ_] :=
2370   Module[{allowedSL, allowedNKSL, allowedSLJM, nn},
2371     allowedNKSL = AllowedNKSLTerms[numE];
2372     allowedSL   = AllowedSLTerms[numE];
2373     allowedSLJM = {};
2374     For[
2375       nn = 1,
2376       nn <= Length[allowedSL],
2377       termSL = allowedSL[[nn]];
2378       termNKSL = allowedNKSL[[nn]];
2379       termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
2380                     <= J
2381                     <= Total[termSL]
2382                     && (Abs[MJ] <= J)
2383                     ),
2384                     {{termNKSL, J, MJ}},
2385                     {}];
2386       allowedSLJM = Join[allowedSLJM, termsSLJ];
2387       nn++
2388     ];
2389     Return[allowedSLJM];
2390   ]
2391
2392 AllowedNKSLJMforJTerms::usage = "AllowedNKSLJMforJTerms[numE, J]
    returns a list with all the states that have a total angular
    momentum J. The returned list has elements of the form {{SL (
    string in spectroscopic notation), J}, MJ}, and if the option \"
    Flat\" is set to True then the returned list has element of the
```

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

AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J,
  1].";
AllowedMforJ[J_] := Range[-J, J, 1];

AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta
  J that appear in the f^numE configuration.";
AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}];

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

FindNKLSTerm::usage = "Given the string LS FindNKLSTerm[SL] returns
  all the terms that are compatible with it. This is only for f^n
  configurations. The provided terms might belong to more than one
  configuration. The function returns a list with elements of the
  form {LS, seniority, W, U}.";
FindNKLSTerm[SL_] := Module[
  {NKterms, n},
  n = 7;
  NKterms = {{}};
  Map[
    If[! StringFreeQ[First[#], SL],
      If[ToExpression[Part[#, 2]] <= n,
        NKterms = Join[NKterms, {#}, 1]
      ]
    ] &,
  fnTermLabels
  ];
  NKterms = DeleteCases[NKterms, {}];
  NKterms]

Options[ParseTermLabels] = {"Export" -> True};
ParseTermLabels::usage="ParseTermLabels[] parses the labels for the
  terms in the f^n configurations based on the labels for the f6 and
   f7 configurations. The function returns a list whose elements are
   of the form {LS, seniority, W, U}.";
```

```
2437  ParseTermLabels[OptionsPattern[]] := Module[
2438    {labelsTextData, fNtextLabels, nielsonKosterLabels, seniorities,
      RacahW, RacahU},
2439  (
2440    labelsTextData = FileNameJoin[{moduleDir, "data", "
      NielsonKosterLabels_f6_f7.txt"}];
2441    fNtextLabels    = Import[labelsTextData];
2442    nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
2443    termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
2444    seniorities = Map[ToExpression[Part[# , {2}]] &,
      nielsonKosterLabels];
2445    racahW =
2446      Map[
2447        StringTake[
2448          Flatten[StringCases[Part[# , {3}],
2449            "(" ~~ DigitCharacter ~~ DigitCharacter ~~ DigitCharacter
      ~~ ")"]],
2450          {2, 4}
2451        ] &,
2452      nielsonKosterLabels];
2453    racahU =
2454      Map[
2455        StringTake[
2456          Flatten[StringCases[Part[# , {3}],
2457            "(" ~~ DigitCharacter ~~ DigitCharacter ~~ ")"]],
2458          {2, 3}
2459        ] &,
2460      nielsonKosterLabels];
2461    fnTermLabels = Join[termLabels, seniorities, racahW, racahU, 2];
2462    fnTermLabels = Sort[fnTermLabels];
2463    If[OptionValue["Export"],
2464      (
2465        broadFname = FileNameJoin[{moduleDir,"data","fnTerms.m"}];
2466        Export[broadFname, fnTermLabels];
2467      )
2468    ];
2469    Return[fnTermLabels];
2470  )
2471  ]
2472
2473  (* ############################ Term management
      ############################## *)
2474  (*
      ################################################################################
       *)
2475
2476  Options[LoadParameters] = {
2477      "Source"->"Carnall",
2478      "Free Ion"->False,
2479      "gs"->2.002319304386
2480      };
2481  LoadParameters::usage="LoadParameters[ln] takes a string with the
      symbol the element of a trivalent lanthanide ion and returns model
       parameters for it. It is based on the data for LaF3. If the
      option \"Free Ion\" is set to True then the function sets all
```

```
    crystal field parameters to zero. Through the option \"gs\" it
    allows modyfing the electronic gyromagnetic ratio. For
    completeness this function also computes the E parameters using
    the F parameters quoted on Carnall.";
2482  LoadParameters[Ln_String, OptionsPattern[]]:=
2483    Module[{source, params},
2484    (
2485      source = OptionValue["Source"];
2486      params = Which[source=="Carnall",
2487               (Association[Carnall["data"][Ln]])
2488               ];
2489      (*If a free ion then all the parameters from the crystal field
    are set to zero*)
2490      If[OptionValue["Free Ion"],
2491        Do[params[cfSymbol] = 0,
2492        {cfSymbol, cfSymbols}
2493        ]
2494      ];
2495      params[F0] = 0;
2496      params[M2] = 0.56 * params[M0]; (* See Carnall 1989, Table I,
    caption, probably fixed based on HF values*)
2497      params[M4] = 0.31 * params[M0]; (* See Carnall 1989, Table I,
    caption, probably fixed based on HF values*)
2498      params[P0] = 0;
2499      params[P4] = 0.5 * params[P2];  (* See Carnall 1989, Table I,
    caption, probably fixed based on HF values*)
2500      params[P6] = 0.1 * params[P2];  (* See Carnall 1989, Table I,
    caption, probably fixed based on HF values*)
2501      params[gs] = OptionValue["gs"];
2502      {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[
    F0], params[F2], params[F4], params[F6]];
2503      params[E0] = 0;
2504      Return[params];
2505    )
2506  ];
2507
2508  HoleElectronConjugation::usage = "HoleElectronConjugation[params]
    takes the parameters (as an association) that define a
    configuration and converts them so that they may be interpreted as
     corresponding to a complentary hole configuration. Some of this
    can be simply done by changing the sign of the model parameters.
    In the case of the effective three body interaction the
    relationship is more complex and is controlled by the value of the
     isE variable.";
2509
2510  HoleElectronConjugation[params_] :=
2511    Module[{newparams = params},
2512      (
2513        flipSignsOf = {ζ, T2, T3, T4, T6, T7, T8};
2514        flipSignsOf = Join[flipSignsOf, cfSymbols];
2515        flipped =
2516          Table[(flipper -> - newparams[flipper]),
2517          {flipper, flipSignsOf}
2518          ];
2519        nonflipped =
```

```
          Table[(flipper -> newparams[flipper]),
          {flipper, Complement[Keys[newparams], flipSignsOf]}
          ];
        flippedParams = Association[Join[nonflipped, flipped]];
        Return[flippedParams];
      )
    ]

IonSolverLaF3::usage="IonSolverLaF3[numE] solves the energy levels
  of a lanthanide ion with numE f-electrons in lanthanum fluoride.
  It does this by querying the fit parameters from Carnall's tables.
   This function is used to compare the calculated values as
  calculated with qlanth with the calculated values quoted by
  Carnall.

Parameters
----------
numE (int) : Number of f-electrons.

Options
-------
\"Include Spin-Spin\" (bool) : If True then the spin-spin
  interaction is included as a contribution to the m_k operators.
  The default is True.

Returns
-------
{rmsDifference, gtEnergies, cfenergies, ln, carnallAssignments, {
  fstates, basis, symbolicMatrix}} (list): with
   rmsDifference (float) : The root-mean-square difference between
  the calculated values from Carnall and the ones computed here.
   gtEnergies (list) : The calculated values for the energy levels as
    quoted by Carnall.
   cfenergies (list) : The calculated values for the energy levels as
    calculated here.
   ln (string) : The symbol of the lanthanide ion.
   carnallAssignments (list) : The assignments of the energy levels
  as quoted by Carnall.
   {fstates, basis, symbolicMatrix} (list) : The eigenstates, basis
  and symbolic matrix as calculated here.
";
Options[IonSolverLaF3] = {"Include Spin-Spin" -> True};
IonSolverLaF3[numE_, OptionsPattern[]] := (
  spinspin = OptionValue["Include Spin-Spin"];
  host = "LaF3";
  ln = StringSplit["Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb"][[numE
  ]];
  terms = AllowedNKSLJTerms[Min[numE, 14 - numE]];
  expData = Flatten[#["Exp (1/cm)"] & /@ Values[Carnall["appendix:"
  <> ln <> ":Association"]]];

  (*In Carnall's approach the crystal field is assumed to have C_{2v
  } symmetry, which is a simplification from the actual point
  symmetry of C_2*)
  simplifier = {
```

```
    B12 -> 0, B14 -> 0, B16 -> 0, B34 -> 0, B36 -> 0, B56 -> 0,
    S12 -> 0, S14 -> 0, S16 -> 0, S22 -> 0, S24 -> 0, S26 -> 0,
    S34 -> 0, S36 -> 0, S44 -> 0, S46 -> 0, S56 -> 0, S66 -> 0,
    T11 -> 0, T12 -> 0, T14 -> 0, T15 -> 0, T16 -> 0, T18 -> 0, T11p -> 0,
    T17 -> 0, T19 -> 0
    };
eTofs = (#[[1]] -> #[[2]]) & /@ Transpose[{{E0, E1, E2, E3}, FtoE[F0, F2, F4, F6]}];
ham = Normal[HamMatrixAssembly[numE, 0]];
simpleHam = ham /. simplifier;
simpleHam = simpleHam /. eTofs;
hamParams = DeleteDuplicates[Flatten[Variables /@ simpleHam]];
ham = Normal[HamMatrixAssembly[numE, 0]];
termNames = First /@ terms;
termSimplifier =
  Table[
    termN -> If[StringLength[termN] == 3,
      StringTake[termN, {1, 2}],
      termN
      ],
  {termN, termNames}
  ];

(*Load the parameters from Carnall*)
params = LoadParameters[ln, "Free Ion" -> False];
(*Enforce the override to the spin-spin contribution to the magnetic interactions*)
params[\[Sigma]SS] = If[spinspin, 1, 0];
(*Everything that is not given is set to zero*)
params = ParamPad[params, "Print" -> True];

{fstates, basis, symbolicMatrix} =
SolveStates[params[nf], 0, params, "Return Symbolic Matrix" -> True];
symbolicMatrix =
  If[spinspin,
    ReplaceInSparseArray[symbolicMatrix, {\[Sigma]SS -> 1}],
    ReplaceInSparseArray[symbolicMatrix, {\[Sigma]SS -> 0}]
  ];
fstates = ShiftedLevels[fstates];
fstates = SortBy[fstates, First];
cfenergies = First /@ fstates;
cfenergies = Chop[cfenergies];
If[OddQ[numE],
(
  cfenergies = cfenergies[[;; ;; 2]];
)
];

mainKey = StringTemplate["appendix:‘Ln‘:Association"][<|"Ln" -> ln|>];
lnData  = Carnall[mainKey];
carnalKeys = lnData // Keys;
repetitions = Length[lnData[#]["Calc (1/cm)"]] & /@ carnalKeys;
```

```
2608      carnallAssignments =
2609      First /@ Carnall["appendix:" <> ln <> ":RawTable"];
2610
2611      carnalKey  = StringTemplate["appendix:`Ln`:Calculated"][<|"Ln" ->
          ln|>];
2612      gtEnergies = Sort[Carnall[carnalKey]];
2613      diffs = Sort[cfenergies][[;; Length[gtEnergies]]] - gtEnergies;
2614      rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
2615
2616      Return[{rmsDifference, gtEnergies, cfenergies, ln,
          carnallAssignments, {fstates, basis, symbolicMatrix}}]
2617      )
2618
2619    FastIonSolverLaF3::usage =
2620      "This function solves the energy levels of the given trivalent
          lanthanide in LaF3. The values for the Hamiltonian are simply
          taken from the values quoted by Carnall. It uses precomputed
          symbolic matrices for the Hamiltonian so it's faster than the
          previous alternatives.
2621
2622      The function returns a list with seven elements {rmsDifference,
          carnallEnergies, eigenEnergies, ln, carnallAssignments, eigensys,
          basis}. Where:
2623
2624      rmsDifference is the root mean squared difference between the
          calculated values and those quoted by Carnall
2625
2626      carnallEnergies are the quoted calculated values from Carnall;
2627
2628      eigenEnergies are the calculated energies (in the case of an odd
          number of electrons the kramers degeneracy has been elided from
          this list);
2629
2630      ln is simply a string labelling the corresponding lanthanide;
2631
2632      carnallAssignments is a list of strings providing the term
          assignments that Carnall assumed,
2633
2634      eigensys is a list of tuples where the first element is the energy
           corresponding to the eigenvector given as the second element;
2635
2636      basis a list that specifies the basis in which the Hamiltonian was
           constructed and diagonalized.
2637    ";
2638    Options[FastIonSolverLaF3] = {
2639      "MakeNotebook" -> True,
2640      "NotebookSave" -> True,
2641      "HTMLSave" -> False,
2642      "eigenstateTruncationProbability" -> 0.9,
2643      "Include spin-spin" -> True,
2644      "Max Eigenstates in Table" -> 100,
2645      "Sparse" -> True,
2646      "PrintFun" -> Print,
2647      "SaveData" -> True,
2648      "paramFiddle" -> {},
```

```
    "Append to Filename" -> ""
    };
FastIonSolverLaF3[numE_, OptionsPattern[]] := Module[{
    makeNotebook, eigenstateTruncationProbability, spinspin, host,
    ln, terms, termNames, carnallEnergies, eigenEnergies,
    simplerStateLabels,
    eigensys, basis, assignmentMatches, stateLabels,
    carnallAssignments},
(
    PrintFun = OptionValue["PrintFun"];
    makeNotebook = OptionValue["MakeNotebook"];
    eigenstateTruncationProbability = OptionValue["
    eigenstateTruncationProbability"];
    maxStatesInTable = OptionValue["Max Eigenstates in Table"];
    spinspin = OptionValue["Include spin-spin"];
    host = "LaF3";
    paramFiddle = OptionValue["paramFiddle"];
    ln = theLanthanides[[numE]];
    terms = AllowedNKSLJTerms[Min[numE, 14 - numE]];
    termNames = First /@ terms;
    (* For labeling the states, the degeneracy in some of the terms is
     elided *)
    PrintFun["> Calculating simpler term labels ..."];
    termSimplifier =
      Table[termN -> If[StringLength[termN] == 3,
        StringTake[termN, {1, 2}],
        termN
        ],
      {termN, termNames}
      ];

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

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

    (* Import the symbolic Hamiltonian *)
    PrintFun["> Loading the symbolic Hamiltonian for this
    configuration ..."];
    startTime = Now;
    numH = 14 - numE;
    numEH = Min[numE, numH];
    C2vsimplifier = {B12 -> 0, B14 -> 0, B16 -> 0, B34 -> 0, B36 -> 0,
```

```
    B56 -> 0,
    S12 -> 0, S14 -> 0, S16 -> 0, S22 -> 0, S24 -> 0, S26 -> 0,
    S34 -> 0, S36 -> 0,
    S44 -> 0, S46 -> 0, S56 -> 0, S66 -> 0, T11p -> 0, T11 -> 0,
    T12 -> 0, T14 -> 0, T15 -> 0,
    T16 -> 0, T18 -> 0, T17 -> 0, T19 -> 0};
  simpleHam = If[
    ValueQ[symbolicHamiltonians[numEH]],
    symbolicHamiltonians[numEH],
    SimplerSymbolicHamMatrix[numE, C2vsimplifier, "PrependToFilename" -> "C2v-", "Overwrite" -> False]
  ];
  endTime  = Now;
  loadTime = QuantityMagnitude[endTime - startTime, "Seconds"];
  PrintFun[">> Loading the symbolic Hamiltonian took ", loadTime, " seconds."];

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

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

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

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

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

```
2748          eigenEnergies = eigenEnergies[[;; ;; 2]];
2749        )
2750      ];
2751
2752      (*Compare against the data quoted by Bill Carnall*)
2753      PrintFun["> Comparing against the data from Carnall ..."];
2754      mainKey            = StringTemplate["appendix:`Ln`:Association"
          ][<|"Ln" -> ln|>];
2755      lnData             = Carnall[mainKey];
2756      carnalKeys         = lnData // Keys;
2757      repetitions        = Length[lnData[#]["Calc (1/cm)"]] & /@
          carnalKeys;
2758      carnallAssignments = First /@ Carnall["appendix:" <> ln <> ":
          RawTable"];
2759      carnalKey          = StringTemplate["appendix:`Ln`:Calculated"][<|
          "Ln" -> ln|>];
2760      carnallEnergies    = Carnall[carnalKey];
2761
2762      (* For the difference take as many energies as quoted by Bill*)
2763      eigenEnergies = eigenEnergies + carnallEnergies[[1]];
2764      diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] -
          carnallEnergies;
2765      (* Remove the differences where the appendix tables have elided
          values*)
2766      rmsDifference = Sqrt[Mean[(Select[diffs, FreeQ[#, Missing[]] &])
          ^2]];
2767      titleTemplate = StringTemplate[
2768        "Energy Level Diagram of \!\(\*SuperscriptBox[\(`ion`\), \(\(3\)
          \(+\)\)]\)"];
2769      title = titleTemplate[<|"ion" -> ln|>];
2770      parsedStates = ParseStates[eigensys, basis];
2771      If[OddQ[numE],
2772        parsedStates = parsedStates[[;; ;; 2]]];
2773
2774      stateLabels = #[[-1]] & /@ parsedStates;
2775      simplerStateLabels = ((#[[2]] /. termSimplifier) <> ToString
          [#[[3]], InputForm]) & /@ parsedStates;
2776
2777      PrintFun[">> Truncating eigenvectors to given probability ..."];
2778      startTime = Now;
2779      truncatedStates = ParseStatesByProbabilitySum[eigensys, basis,
2780          eigenstateTruncationProbability,
2781          0.01];
2782      endTime = Now;
2783      truncationTime = QuantityMagnitude[endTime - startTime, "Seconds"
          ];
2784      PrintFun[">>> Truncation took ", truncationTime, " seconds."];
2785
2786      If[makeNotebook,
2787        (
2788          PrintFun["> Putting together results in a notebook ..."];
2789          energyDiagram = Framed[
2790            EnergyLevelDiagram[eigensys, "Title" -> title,
2791            "Background" -> White]
2792            , Background -> White, FrameMargins -> 50];
```

```
2793        appToFname = OptionValue["Append to Filename"];
2794        PrintFun[">> Comparing the term assignments between qlanth and
     Carnall ..."];
2795        assignmentMatches =
2796        If[StringContainsQ[#[[1]], #[[2]]], "\[Checkmark]", "X"] & /@
2797          Transpose[{carnallAssignments, simplerStateLabels[[;; Length
    [carnallAssignments]]]}];
2798        assignmentMatches = {{"\[Checkmark]",
2799          Count[assignmentMatches, "\[Checkmark]"]}, {"X",
2800          Count[assignmentMatches, "X"]}};
2801        labelComparison = (If[StringContainsQ[#[[1]], #[[2]]], "\[
    Checkmark]", "X"] & /@
2802          Transpose[{carnallAssignments,
2803            simplerStateLabels[[;; Length[carnallAssignments]]]}]);
2804        labelComparison =
2805        PadRight[labelComparison, Length[simplerStateLabels], "-"];
2806
2807        statesTable =
2808        Grid[Prepend[{Round[#[[1]]], #[[2]]} & /@
2809            truncatedStates[[;;Min[Length[eigensys],maxStatesInTable
    ]]], {"Energy/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)",
2810            "\[Psi]"}], Frame -> All, Spacings -> {2, 2},
2811          FrameStyle -> Blue,
2812          Dividers -> {{False, True, False}, {True, True}}];
2813        DefaultIfMissing[expr_]:= If[FreeQ[expr, Missing[]], expr,"NA"
    ];
2814        PrintFun[">> Rounding the energy differences for table
    presentation ..."];
2815        roundedDiffs = Round[diffs, 0.1];
2816        roundedDiffs = PadRight[roundedDiffs, Length[
    simplerStateLabels], "-"];
2817        roundedDiffs = DefaultIfMissing /@ roundedDiffs;
2818        diffs = PadRight[diffs, Length[simplerStateLabels], "-"];
2819        diffs = DefaultIfMissing /@ diffs;
2820        diffTableData = Transpose[{simplerStateLabels, eigenEnergies,
2821            labelComparison,
2822            PadRight[carnallAssignments, Length[simplerStateLabels], "
    -"],
2823            DefaultIfMissing/@PadRight[carnallEnergies, Length[
    simplerStateLabels], "-"],
2824            roundedDiffs}];
2825        diffTable =
2826        TableForm[diffTableData,
2827          TableHeadings -> {None, {"qlanth",
2828            "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)", "", "Carnall",
2829            "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)",
2830            "\[CapitalDelta]E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"
    }}];
2831
2832        diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] -
    carnallEnergies;
2833        notBad = FreeQ[#,Missing[]]&/@diffs;
2834        diffs = Pick[diffs,notBad];
2835        diffHistogram =
2836        Histogram[diffs, Frame -> True, ImageSize -> 800,
```

```
2837            AspectRatio -> 1/3, FrameStyle -> Directive[16],
2838            FrameLabel -> {"(qlanth-carnall)/Ky", "Freq"}];
2839          rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
2840          labelTempate =
2841          StringTemplate[
2842            "\!\(\*SuperscriptBox[\(`ln`\), \(\(3\)\(+\)\)]\)"];
2843          diffData = diffs;
2844          diffLabels = simplerStateLabels[[;;Length[notBad]]];
2845          diffLabels = Pick[diffLabels, notBad];
2846          diffPlot = Framed[
2847            ListLabelPlot[diffData,
2848            diffLabels,
2849            Frame -> True,
2850            PlotRange -> All,
2851            ImageSize -> 1200,
2852            AspectRatio -> 1/3,
2853            FrameLabel -> {"",
2854              "(qlanth-carnall) / \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)
    "},
2855            PlotMarkers -> "OpenMarkers",
2856            PlotLabel ->
2857              Style[labelTempate[<|"ln" -> ln|>] <> " | " <> "\[Sigma]="
     <>
2858                ToString[Round[rmsDifference, 0.01]] <>
2859                " \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)\n", 20],
2860            Background -> White
2861            ],
2862            Background -> White,
2863            FrameMargins -> 50
2864            ];
2865          nb = CreateDocument[{
2866            TextCell[Style[DisplayForm[SuperscriptBox[host <> ":" <> ln,
     "3+"]]], "Title", TextAlignment -> Center],
2867            TextCell["Energy Diagram", "Section", TextAlignment ->
    Center],
2868            TextCell[energyDiagram, TextAlignment -> Center],
2869            TextCell["Multiplet Assignments & Energy Levels", "Section",
     TextAlignment -> Center],
2870            TextCell[diffHistogram, TextAlignment -> Center],
2871            TextCell[diffPlot, "Output", TextAlignment -> Center],
2872            TextCell[assignmentMatches, "Output", TextAlignment ->
    Center],
2873            TextCell[diffTable, "Output", TextAlignment -> Center],
2874            TextCell["Truncated Eigenstates", "Section", TextAlignment
    -> Center],
2875            TextCell["These are some of the resultant eigenstates which
    add up to at least a total probability of " <> ToString[
    eigenstateTruncationProbability] <> ".", "Text", TextAlignment ->
    Center],
2876            TextCell[statesTable, "Output", TextAlignment -> Center]
2877            },
2878          WindowSelected -> True,
2879          WindowTitle -> ln <> " in " <> "LaF3" <> appToFname,
2880          WindowSize -> {1600, 800}];
2881          If[OptionValue["SaveData"],
```

```
2882            (
2883              exportFname = FileNameJoin[{moduleDir,"calcs", ln <> " in
    " <> "LaF3" <> appToFname <> ".m"}];
2884              SelectionMove[nb, After, Notebook];
2885              NotebookWrite[nb, Cell["Reload Data", "Section",
    TextAlignment -> Center]];
2886              NotebookWrite[nb, Cell[(
2887                "{rmsDifference, carnallEnergies, eigenEnergies, ln,
    carnallAssignments, simplerStateLabels, eigensys, basis,
    truncatedStates} = Import[FileNameJoin[{NotebookDirectory[],\"" <>
     StringSplit[exportFname,"/"][[-1]] <> "\"}]];"
2888                ),"Input"]];
2889              NotebookWrite[nb, Cell[(
2890                "Manipulate[First[MinimalBy[truncatedStates, Abs[First
    [#] - energy] &]], {energy,0}]"
2891                ),"Input"]];
2892              SelectionMove[nb, Before, Notebook];
2893              Export[exportFname, {rmsDifference, carnallEnergies,
    eigenEnergies, ln, carnallAssignments, simplerStateLabels,
    eigensys, basis, truncatedStates}];
2894              tinyexportFname = FileNameJoin[{moduleDir,"calcs", ln <> "
     in " <> "LaF3" <> appToFname <> "- tiny.m"}];
2895              tinyExport = <|"ln"->ln,
2896                             "carnallEnergies"->carnallEnergies,
2897                             "rmsDifference"-> rmsDifference,
2898                             "eigenEnergies"-> eigenEnergies,
2899                             "carnallAssignments"-> carnallAssignments,
2900                             "simplerStateLabels" -> simplerStateLabels
    |>;
2901              Export[tinyexportFname, tinyExport];
2902            )
2903          ];
2904          If[OptionValue["NotebookSave"],
2905            (
2906              nbFname = FileNameJoin[{moduleDir,"calcs", ln <> " in " <>
     "LaF3" <> appToFname <> ".nb"}];
2907              PrintFun[">> Saving notebook to ", nbFname, " ..."];
2908              NotebookSave[nb, nbFname];
2909            )
2910          ];
2911          If[OptionValue["HTMLSave"],
2912            (
2913              htmlFname = FileNameJoin[{moduleDir,"calcs", "html", ln <>
     " in " <> "LaF3" <> appToFname <> ".html"}];
2914              PrintFun[">> Saving html version to ", htmlFname, " ..."];
2915              Export[htmlFname, nb];
2916            )
2917          ];
2918        )
2919      ];
2920
2921      Return[{rmsDifference, carnallEnergies, eigenEnergies, ln,
    carnallAssignments, simplerStateLabels, eigensys, basis,
    truncatedStates}];
2922      )
```

```
];

ShiftedLevels::usage = "
ShiftedLevels[originalLevels] takes a list of levels of the form
{{energy_1, coeff_vector_1},
{energy_2, coeff_vector_2},
...}}
and returns the same input except that now to every energy the
  minimum of all of them has been subtracted.";
ShiftedLevels[originalLevels_] :=
  Module[{groundEnergy, shifted},
    groundEnergy = Sort[originalLevels][[1,1]];
    shifted      = Map[{#[[1]] - groundEnergy, #[[2]]} &,
  originalLevels];
    Return[shifted];
  ]

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

PrettySaundersSLJmJ::usage = "PrettySaundersSLJmJ[{SL, J, mJ}]
  produces a human-redeable symbol for the given basis vector {SL, J
  , mJ}."
PrettySaundersSLJmJ[{SL_, J_, mJ_}] := (If[
  StringQ[SL],
  ({S, L} = FindSL[SL];
    L = StringTake[SL, {2, -1}];
    ),
  {S, L} = SL];
  Return[
  RowBox[{AdjustmentBox[Style[2*S + 1, Smaller],
      BoxBaselineShift -> -1, BoxMargins -> 0],
      AdjustmentBox[PrintL[L], BoxMargins -> -0.2],
      AdjustmentBox[
      Style[{InputForm[J], mJ}, Small, FontTracking -> "Narrow"],
      BoxBaselineShift -> 1,
      BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]}] // DisplayForm])

BasisVecInRusselSaunders::usage = "BasisVecInRusselSaunders[basisVec
  ] takes a basis vector in the format {LSstring, Jval, mJval} and
  returns a human-readable symbol for the corresponding Russel-
  Saunders term."
BasisVecInRusselSaunders[basisVec_] := (
  {LSstring, Jval, mJval} = basisVec;
  Ket[PrettySaunders[LSstring, Jval], mJval]
  )

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

```
2968  BasisVecInLSJMJ::usage = "BasisVecInLSJMJ[basisVec] takes a basis
        vector in the format {{{LSstring, Jval}, mJval}, nucSpin} and
        returns a human-readable symbol for the corresponding LSJMJ term
        in the form |LS, J=..., mJ=...>."
2969  BasisVecInLSJMJ[basisVec_] := (
2970    {LSstring, Jval, mJval} = basisVec;
2971    LSJMJTemplate[<|
2972      "LS" -> LSstring,
2973      "J" -> ToString[Jval, InputForm],
2974      "mJ" -> ToString[mJval, InputForm]|>]
2975    );
2976
2977  ParseStates::usage = "ParseStates[states, basis] takes a list of
        eigenstates in terms of their coefficients in the given basis and
        returns a list of the same states in terms of their energy, LSJMJ
        symbol, J, mJ, S, L, LSJ symbol, and LS symbol. The LS symbol
        returned corresponds to the term with the largest coefficient in
        the given basis.";
2978  ParseStates[states_, basis_, OptionsPattern[]] := Module[{
        parsedStates},
2979  (
2980    parsedStates = Table[(
2981      {energy, eigenVec} = state;
2982      maxTermIndex = Ordering[Abs[eigenVec]][[-1]];
2983      {LSstring, Jval, mJval} = basis[[maxTermIndex]];
2984      LSJsymbol = Subscript[LSstring, {Jval, mJval}];
2985      LSJMJsymbol = LSstring <> ToString[Jval, InputForm];
2986      {S, L} = FindSL[LSstring];
2987      {energy, LSstring, Jval, mJval, S, L, LSJsymbol, LSJMJsymbol}
2988      ),
2989      {state, states}];
2990    Return[parsedStates]
2991    )
2992  ]
2993
2994  ParseStatesByNumBasisVecs::usage = "ParseStatesByNumBasisVecs[states
        , basis, numBasisVecs] takes a list of eigenstates in terms of
        their coefficients in the given basis and returns a list of the
        same states in terms of their energy and the coefficients of the
        numBasisVecs most significant basis vectors.";
2995  ParseStatesByNumBasisVecs[states_, basis_, numBasisVecs_, roundTo_ :
        0.01] := (
2996    parsedStates = Table[(
2997      {energy, eigenVec} = state;
2998      energy = Chop[energy];
2999      probs = Round[Abs[eigenVec^2], roundTo];
3000      amplitudes = Round[eigenVec, roundTo];
3001      ordering = Ordering[probs];
3002      chosenIndices = ordering[[-numBasisVecs ;;]];
3003      majorComponents = basis[[chosenIndices]];
3004      majorProbabilities = amplitudes[[chosenIndices]];
3005      majorComponents = BasisVecInLSJMJ /@ majorComponents;
3006      majorRep = majorProbabilities . majorComponents;
3007      {energy, majorRep}
3008      ),
```

```
3009        {state, fstates}];
3010      Return[parsedStates]
3011      )
3012
3013    FindThresholdPosition::usage = "FindThresholdPosition[list,
          threshold] returns the position of the first element in list that
          is greater than threshold. If no such element exists, it returns
          the length of list. The elements of the given list must be in
          ascending order.";
3014    FindThresholdPosition[list_, threshold_] :=
3015    Module[{position},
3016      position = Position[list, _?(# > threshold &), 1, 1];
3017      thrPos = If[Length[position] > 0,
3018        position[[1, 1]],
3019        Length[list]];
3020      If[thrPos == 0, Return[1], Return[thrPos+1]]
3021      ]
3022
3023    ParseStateByProbabilitySum[{energy_, eigenVec_}, probSum_, roundTo_
          :0.01, maxParts_:20] := Compile[
3024      {{energy, _Real, 0},{eigenVec, _Complex, 1}, {probSum, _Real, 0},
          {roundTo, _Real, 0}, {maxParts, _Integer, 0}},
3025      Module[
3026      {numStates, state, amplitudes, probs, ordering,
3027      orderedProbs, truncationIndex, accProb, thresholdIndex,
          chosenIndices, majorComponents,
3028      majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
3029    (
3030      numStates     = Length[eigenVec];
3031      (*Round them up*)
3032      amplitudes          = Round[eigenVec, roundTo];
3033      probs               = Round[Abs[eigenVec^2], roundTo];
3034      ordering            = Reverse[Ordering[probs]];
3035      (*Order the probabilities from high to low*)
3036      orderedProbs        = probs[[ordering]];
3037      (*To speed up Accumulate, assume that only as much as maxParts
          will be needed*)
3038      truncationIndex     = Min[maxParts, Length[orderedProbs]];
3039      orderedProbs        = orderedProbs[[;;truncationIndex]];
3040      (*Accumulate the probabilities*)
3041      accProb             = Accumulate[orderedProbs];
3042      (*Find the index of the first element in accProb that is greater
          than probSum*)
3043      thresholdIndex      = Min[Length[accProb], FindThresholdPosition[
          accProb, probSum]];
3044      (*Grab all the indicees up till that one*)
3045      chosenIndices       = ordering[[;; thresholdIndex]];
3046      (*Select the corresponding elements from the basis*)
3047      majorComponents     = basis[[chosenIndices]];
3048      (*Select the corresponding amplitudes*)
3049      majorAmplitudes     = amplitudes[[chosenIndices]];
3050      (*Take their absolute value*)
3051      absMajorAmplitudes = Abs[majorAmplitudes];
3052      (*Make sure that there are no effectively zero contributions*)
```

```
    notnullAmplitudes  = Flatten[Position[absMajorAmplitudes, x_ /; x
    != 0]];
    (* majorComponents    = PrettySaundersSLJmJ
    [{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents; *)
    majorComponents    = PrettySaundersSLJmJ /@ majorComponents;
    majorAmplitudes    = majorAmplitudes[[notnullAmplitudes]];
    (*Make them into Kets*)
    majorComponents    = Ket /@ majorComponents[[notnullAmplitudes]];
    (*Multiply and add to build the final Ket*)
    majorRep           = majorAmplitudes . majorComponents;
      );
  Return[{energy, majorRep}]
  ],
    CompilationTarget -> "C",
    RuntimeAttributes -> {Listable},
    Parallelization -> True,
    RuntimeOptions -> "Speed"
  ];

  ParseStatesByProbabilitySum::usage = "ParseStatesByProbabilitySum[
    eigensys, basis, probSum] takes a list of eigenstates in terms of
    their coefficients in the given basis and returns a list of the
    same states in terms of their energy and the coefficients of the
    basis vectors that sum to at least probSum.";
  ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ :
    0.01, maxParts_: 20] := Module[
    {parsedByProb, numStates, state, energy, eigenVec, amplitudes,
    probs, ordering,
    orderedProbs, truncationIndex, accProb, thresholdIndex,
    chosenIndices, majorComponents,
    majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
  (
    numStates    = Length[eigensys];
    parsedByProb = Table[(
      state              = eigensys[[idx]];
      {energy, eigenVec} = state;
      (*Round them up*)
      amplitudes         = Round[eigenVec, roundTo];
      probs              = Round[Abs[eigenVec^2], roundTo];
      ordering           = Reverse[Ordering[probs]];
      (*Order the probabilities from high to low*)
      orderedProbs       = probs[[ordering]];
      (*To speed up Accumulate, assume that only as much as maxParts
    will be needed*)
      truncationIndex    = Min[maxParts, Length[orderedProbs]];
      orderedProbs       = orderedProbs[[;;truncationIndex]];
      (*Accumulate the probabilities*)
      accProb            = Accumulate[orderedProbs];
      (*Find the index of the first element in accProb that is greater
     than probSum*)
      thresholdIndex     = Min[Length[accProb], FindThresholdPosition[
    accProb, probSum]];
      (*Grab all the indicees up till that one*)
      chosenIndices      = ordering[[;; thresholdIndex]];
      (*Select the corresponding elements from the basis*)
```

```
3096        majorComponents    = basis[[chosenIndices]];
3097        (*Select the corresponding amplitudes*)
3098        majorAmplitudes    = amplitudes[[chosenIndices]];
3099        (*Take their absolute value*)
3100        absMajorAmplitudes = Abs[majorAmplitudes];
3101        (*Make sure that there are no effectively zero contributions*)
3102        notnullAmplitudes  = Flatten[Position[absMajorAmplitudes, x_ /;
      x != 0]];
3103        (* majorComponents    = PrettySaundersSLJmJ
      [{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents; *)
3104        majorComponents    = PrettySaundersSLJmJ /@ majorComponents;
3105        majorAmplitudes    = majorAmplitudes[[notnullAmplitudes]];
3106        (*Make them into Kets*)
3107        majorComponents    = Ket /@ majorComponents[[notnullAmplitudes
      ]];
3108        (*Multiply and add to build the final Ket*)
3109        majorRep           = majorAmplitudes . majorComponents;
3110        {energy, majorRep}
3111        ), {idx, numStates}];
3112    Return[parsedByProb]
3113    )
3114    ];
3115
3116    (* ########################## Eigensystem analysis
        ########################## *)
3117    (*
        ############################################################################
         *)
3118
3119    (*
        ############################################################################
         *)
3120    (* ################################## Misc
        ################################## *)
3121
3122    SymbToNum::usage = "SymbToNum[expr, numAssociation] takes an
        expression expr and returns what results after making the
        replacements defined in the given replacementAssociation. If
        replacementAssociation doesn't define values for expected keys,
        they are taken to be zero.";
3123    SymbToNum[expr_, replacementAssociation_]:= (
3124      includedKeys = Keys[replacementAssociation];
3125      (*If a key is not defined, make its value zero.*)
3126      fullAssociation = Table[(
3127        If[MemberQ[includedKeys, key],
3128          ToExpression[key]->replacementAssociation[key],
3129          ToExpression[key]->0
3130        ]
3131      ),
3132      {key, paramSymbols}];
3133      Return[expr/.fullAssociation];
3134    )
3135
3136    SimpleConjugate::usage = "SimpleConjugate[expr] takes an expression
        and applies a simplified version of the conjugate in that all it
```

```
  does is that it replaces the imaginary unit I with -I. It assumes
  that every other symbol is real so that it remains the same under
  complex conjugation. Among other expressions it is valid for any
  rational or polynomial expression with complex coefficients and
  real variables.";
SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;

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

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

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

Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
ParseTeXLikeSymbol::usage = "ParseTeXLikeSymbol[string] parses a
  string for a symbol given in LaTeX notation and returns a
  corresponding mathematica symbol. The string may have expressions
  for several symbols, they need to be separated by single spaces.
  In addition the _ and ^ symbols used in LaTeX notation need to
  have arguments that are enclosed in parenthesis, for example \"x_2
  \" is invalid, instead \"x_{2}\" should have been given.";
ParseTeXLikeSymbol[bigString_, OptionsPattern[]] := (
  form = OptionValue["Form"];
  (*parse greek*)
  symbols = Table[(
      str = StringReplace[string, {"\\alpha" -> "α",
        "\\beta" -> "β",
        "\\gamma" -> "γ",
        "\\psi" -> "\[Psi]"}];
      symbol = Which[
        StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"]],
        (
        (*yes sub no sup*)
        mainSymbol = StringSplit[str, "_"][[1]];
        mainSymbol = ToExpression[mainSymbol];

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

        supPart =
          StringCases[str,
            RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
```

```
3226            Superscript[mainSymbol, supPart]),
3227            StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
3228            (
3229            (*yes sub yes sup*)
3230            mainSymbol = StringSplit[str, "_"][[1]];
3231            mainSymbol = ToExpression[mainSymbol];
3232            {subPart, supPart} =
3233              StringCases[str, RegularExpression@"\\{(.*?)\\}" -> "$1"];
3234            Subsuperscript[mainSymbol, subPart, supPart]
3235            ),
3236            True,
3237            ((*no sup or sub*)
3238            str)
3239            ];
3240          symbol
3241          ),
3242        {string, StringSplit[bigString, " "]}];
3243      Which[
3244        form == "Row",
3245        Return[Row[symbols]],
3246        form == "List",
3247        Return[symbols]
3248        ]
3249      );
3250
3251  (* ################################## Misc
       ################################## *)
3252  (*
       ############################################################################
        *)
3253
3254  (*
       ############################################################################
        *)
3255  (* ######################### Some Plotting Routines
       ######################### *)
3256
3257  EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes states
       and produces a visualization of its energy spectrum.
3258  The resultant visualization can be navigated by clicking and
       dragging to zoom in on a region, or by clicking and dragging
       horizontally while pressing Ctrl. Double-click to reset the view."
       ;
3259  Options[EnergyLevelDiagram] = {
3260    "Title"->"",
3261    "ImageSize"->1000,
3262    "AspectRatio" -> 1/8,
3263    "Background"->"Automatic",
3264    "Epilog"->{}
3265    };
3266  EnergyLevelDiagram[states_, OptionsPattern[]]:= (
3267    energies = First/@states;
3268    epi = OptionValue["Epilog"];
3269    ExploreGraphics@ListPlot[Tooltip[{{#, 0}, {#, 1}}, {Quantity
       [#/8065.54429, "eV"], Quantity[#, 1/"Centimeters"]}] &/@ energies,
```

```
      Joined        -> True,
      PlotStyle     -> Black,
      AspectRatio   -> OptionValue["AspectRatio"],
      ImageSize     -> OptionValue["ImageSize"],
      Frame         -> True,
      PlotRange     -> {All, {0, 1}},
      FrameTicks    -> {{None, None}, {Automatic, Automatic}},
      FrameStyle    -> Directive[15, Dashed, Thin],
      PlotLabel     -> Style[OptionValue["Title"], 15, Bold],
      Background    -> OptionValue["Background"],
      FrameLabel    -> {"\!\(\*FractionBox[\(E\), SuperscriptBox[\(cm\)
    , \(-1\)]]\)"},
      Epilog        -> epi]
  )

  ExploreGraphics::usage =
    "Pass a Graphics object to explore it. Zoom by clicking and
    dragging a rectangle. Pan by clicking and dragging while pressing
    Ctrl. Click twice to reset view.
     Based on ZeitPolizei @ https://mathematica.stackexchange.com/
    questions/7142/how-to-manipulate-2d-plots";

  OptAxesRedraw::usage =
    "Option for ExploreGraphics to specify redrawing of axes. Default
    False.";
  Options[ExploreGraphics] = {OptAxesRedraw -> False};

  ExploreGraphics[graph_Graphics, opts : OptionsPattern[]] := With[
    {gr  = First[graph],
      opt = DeleteCases[Options[graph],
            PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _],
      plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
      ar  = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
      ao  = AbsoluteOptions[AxesOrigin],
      rectangle = {Dashing[Small],
        Line[{#1,
              {First[#2], Last[#1]},
              #2,
              {First[#1], Last[#2]},
              #1}]} &,
      optAxesRedraw = OptionValue[OptAxesRedraw]},
    DynamicModule[
        {dragging=False, first, second, rx1, rx2, ry1, ry2,
        range = plr},
        {{rx1, rx2}, {ry1, ry2}} = plr;
      Panel@
      EventHandler[
        Dynamic@Graphics[
          If[dragging, {gr, rectangle[first, second]}, gr],
          PlotRange -> Dynamic@range,
          AspectRatio -> ar,
          AxesOrigin -> If[optAxesRedraw,
            Dynamic@Mean[range\[Transpose]], ao],
          Sequence @@ opt],
        {{"MouseDown", 1} :> (
```

```
3320            first = MousePosition["Graphics"]
3321            ),
3322          {"MouseDragged", 1} :> (
3323            dragging = True;
3324            second = MousePosition["Graphics"]
3325            ),
3326          "MouseClicked" :> (
3327            If[CurrentValue@"MouseClickCount"==2,
3328              range = plr];
3329            ),
3330          {"MouseUp", 1} :> If[dragging,
3331            dragging = False;
3332
3333            range = {{rx1, rx2}, {ry1, ry2}} =
3334              Transpose@{first, second};
3335            range[[2]] = {0, 1}],
3336          {"MouseDown", 2} :> (
3337            first = {sx1, sy1} = MousePosition["Graphics"]
3338            ),
3339          {"MouseDragged", 2} :> (
3340            second = {sx2, sy2} = MousePosition["Graphics"];
3341            rx1 = rx1 - (sx2 - sx1);
3342            rx2 = rx2 - (sx2 - sx1);
3343            ry1 = ry1 - (sy2 - sy1);
3344            ry2 = ry2 - (sy2 - sy1);
3345            range = {{rx1, rx2}, {ry1, ry2}};
3346            range[[2]] = {0, 1};
3347            )}]]];
3348
3349  Options[LabeledGrid]={
3350      ItemSize->Automatic,
3351      Alignment->Center,
3352      Frame->All,
3353      "Separator"->",",
3354      "Pivot"->""
3355  };
3356  LabeledGrid::usage="LabeledGrid[data, rowHeaders, columnHeaders]
       provides a grid of given data interpreted as a matrix of values
       whose rows are labeled by rowHeaders and whose columns are labeled
        by columnHeaders. When hovering with the mouse over the grid
       elements, the row and column labels are displayed with the given
       separator between them.";
3357  LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]]:=
       Module[
3358     {gridList=data,rowHeads=rowHeaders,colHeads=columnHeaders},
3359     (
3360     separator=OptionValue["Separator"];
3361     pivot=OptionValue["Pivot"];
3362     gridList=Table[
3363             Tooltip[
3364               data[[rowIdx,colIdx]],
3365               DisplayForm[
3366                 RowBox[{rowHeads[[rowIdx]],
3367                         separator,
3368                         colHeads[[colIdx]]}
```

```
                        ]
                        ]
                    ],
            {rowIdx,Dimensions[data][[1]]},
            {colIdx,Dimensions[data][[2]]}];
        gridList=Transpose[Prepend[gridList,colHeads]];
        rowHeads=Prepend[rowHeads,pivot];
        gridList=Prepend[gridList,rowHeads]//Transpose;
        Grid[gridList,
            Frame->OptionValue[Frame],
            Alignment->OptionValue[Alignment],
            Frame->OptionValue[Frame],
            ItemSize->OptionValue[ItemSize]
            ]
)
]

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

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

```
3410                {j - 1, numRows - i} + {1, 1}
3411                ]
3412              },
3413            Row[{braLabels[[i]],ketLabelsUpright[[j]],"=",hamMatrix[[i,
    j]]}]
3414            ],
3415          {i, 1, numRows},
3416          {j, 1, numCols}
3417          ]
3418        ],
3419        And[OptionValue["Hover"], OptionValue["Overlay Values"]],
3420        Flatten[
3421          Table[
3422            Tooltip[
3423              {
3424                Transparent,
3425                Rectangle[
3426                {j - 1, numRows - i},
3427                {j - 1, numRows - i} + {1, 1}
3428                ]
3429              },
3430            DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i]],
     "\[LeftBracketingBar]", basisLabels[[j]], "\[RightAngleBracket]"
    }]]
3431            ],
3432          {i, numRows},
3433          {j, numCols}
3434          ]
3435        ],
3436        True,
3437        {}
3438        ];
3439      textOverlay = If[OptionValue["Overlay Values"],
3440        (
3441          Flatten[
3442          Table[
3443            Text[hamMatrix[[i, j]],
3444              {j - 1/2, numRows - i + 1/2}
3445            ],
3446          {i, 1, numRows},
3447          {j, 1, numCols}
3448          ]
3449          ]
3450        ),
3451        {}
3452        ];
3453      epiThings = Join[epiThings, textOverlay];
3454      MatrixPlot[hamMatrix,
3455        FrameTicks -> {
3456          {Transpose[{Range[Length[braLabels]], braLabels}], None},
3457          {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}
3458          },
3459        Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
3460        Epilog -> epiThings
3461      ]
```

```
3462      );
3463
3464    (* ########################## Some Plotting Routines
            ########################## *)
3465    (*
            ############################################################################
            *)
3466
3467    (*
            ############################################################################
            *)
3468    (* ############################## Load Functions
            ############################## *)
3469
3470    LoadAll::usage="LoadAll[] executes all Load* functions.";
3471    LoadAll[]:=(
3472        LoadTermLabels[];
3473        LoadCFP[];
3474        LoadUk[];
3475        LoadV1k[];
3476        LoadT22[];
3477        LoadSOOandECSOLS[];
3478
3479        LoadElectrostatic[];
3480        LoadSpinOrbit[];
3481        LoadSOOandECSO[];
3482        LoadSpinSpin[];
3483        LoadThreeBody[];
3484        LoadChenDeltas[];
3485        LoadCarnall[];
3486      )
3487
3488    LoadTermLabels::usage="LoadTermLabels[] loads into the session the
          labels for the terms in the f^n configurations.";
3489    LoadTermLabels[]:= (
3490      If[ValueQ[fnTermLabels], Return[]];
3491      PrintTemporary["Loading data for state labels in the f^n
          configurations..."];
3492      fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
3493      fnTermLabels::usage = "This list contains the labels of f^n
          configurations. Each element of the list has four elements {LS,
          seniority, W, U}. At first sight this seems to only include the
          labels for the f^6 and f^7 configuration, however, all is included
           in these two.";
3494      If[!FileExistsQ[fnTermsFname],
3495        (PrintTemporary[">> fnTerms.m not found, generating ..."];
3496          fnTermLabels = ParseTermLabels["Export"->True];
3497        ),
3498        fnTermLabels = Import[fnTermsFname];
3499      ];
3500    )
3501
3502
3503    Carnall::usage = "Association of data from Carnall et al (1989) with
          the following keys: {data, annotations, paramSymbols,
```

```
  elementNames, rawData, rawAnnotations, annnotatedData, appendix:Pr
  :Association, appendix:Pr:Calculated, appendix:Pr:RawTable,
  appendix:Headings}";
LoadCarnall::usage="LoadCarnall[] loads data for trivalent
  lanthanides in LaF3 using the data from Bill Carnall's 1989 paper.
  ";
LoadCarnall[]:=(
  If[ValueQ[Carnall], Return[]];
  carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
  If[!FileExistsQ[carnallFname],
    (PrintTemporary[">> Carnall.m not found, generating ..."];
      Carnall = ParseCarnall[];
    ),
    Carnall = Import[carnallFname];
  ];
)

LoadChenDeltas::usage="LoadChenDeltas[] loads the differences noted
  by Chen.";
LoadChenDeltas[]:=(
  If[ValueQ[chenDeltas], Return[]];
  PrintTemporary["Loading the association of discrepancies found by
  Chen ..."];
  chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"
  }];
  If[!FileExistsQ[chenDeltasFname],
    (PrintTemporary[">> chenDeltas.m not found, generating ..."];
      chenDeltas = ParseChenDeltas[];
    ),
    chenDeltas = Import[chenDeltasFname];
  ];
);

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

```
3546          "wrong" -> wrong, "repRule" -> repRule|>;
3547        ),
3548        {row, 1, Length[chenDeltasRaw], 2}];
3549      chenDeltas["A"] = chenDeltasA;
3550
3551      chenDeltasRawB = Import[FileNameJoin[{moduleDir, "data", "the-chen
        -deltas-B.csv"}], "Text"];
3552      chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
3553      chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
3554      chenDeltasB = {ToExpression[StringTake[#[[1]], {2}]], #[[2]],
        #[[3]]} & /@ chenDeltasB;
3555      chenDeltas["B"] = chenDeltasB;
3556      On[Power::infy];
3557      If[OptionValue["Export"],
3558        (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.
        m"}];
3559        Export[chenDeltasFname, chenDeltas];
3560        )
3561        ];
3562      Return[chenDeltas];
3563    )
3564
3565    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).";
3566    Options[ParseCarnall] = {"Export" -> True};
3567    ParseCarnall[] := (
3568      ions          = {"Pr","Nd","Pm","Sm","Eu","Gd","Tb","Dy","Ho","Er",
        "Tm"};
3569      templates     = StringTemplate/@StringSplit["appendix:`ion`:
        Association appendix:`ion`:Calculated appendix:`ion`:RawTable
        appendix:`ion`:Headings"," "];
3570
3571      (* How many unique eigenvalues, after removing Kramer's degeneracy
         *)
3572      fullSizes     = AssociationThread[ions, {91, 182, 1001, 1001, 3003,
         1716, 3003, 1001, 1001, 182, 91}];
3573      carnall       = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"
        }]][[2]];
3574      carnallErr    = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"
        }]][[3]];
3575
3576      elementNames = carnall[[1]][[2;;]];
3577      carnall       = carnall[[2;;]];
3578      carnallErr    = carnallErr[[2;;]];
3579      carnall       = Transpose[carnall];
3580      carnallErr    = Transpose[carnallErr];
3581      paramNames    = ToExpression/@carnall[[1]][[1;;]];
3582      carnall       = carnall[[2;;]];
3583      carnallErr    = carnallErr[[2;;]];
3584      carnallData   = Table[(
3585                       data          = carnall[[i]];
3586                       data          = (#[[1]]->#[[2]])&/@Select[Transpose
        [{paramNames,data}],#[[2]]!=""&];
```

```
                    elementNames[[i]]->data
                    ),
                    {i,1,13}
                    ];
    carnallData  = Association[carnallData];
    carnallNotes = Table[(
                    data        = carnallErr[[i]];
                    elementName = elementNames[[i]];
                    dataFun     = (
                        #[[1]] -> If[#[[2]]=="[]",
                        "Not allowed to vary in fitting.",
                        If[#[[2]]=="[R]",
                            "Ratio constrained by: " <> <|"Eu"->"F4/F2=0.713; F6/F2=0.512",
                                "Gd"->"F4/F2=0.710]",
                                "Tb"->"F4/F2=0.707"|>[elementName],
                            If[#[[2]]=="i",
                                "Interpolated",
                                #[[2]]
                            ]
                        ]
                        ]) &;
                    data = dataFun /@ Select[Transpose[{paramNames,data}],#[[2]]!=""&];
                    elementName->data
                      ),
                  {i,1,13}
                  ];
    carnallNotes  = Association[carnallNotes];

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

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

    Do[(
        carnallData    = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[i]];
        headers        = carnallData[[1]];
        calcIndex      = Position[headers,"Calc (1/cm)"][[1,1]];
        headers        = headers[[2;;]];
```

```
3638        carnallLabels = carnallData[[1]];
3639        carnallData   = carnallData[[2;;]];
3640        carnallTerms  = DeleteDuplicates[First/@carnallData];
3641        parsedData    = Table[(
3642                            rows = Select[carnallData,#[[1]]==term&];
3643                            rows = #[[2;;]]&/@rows;
3644                            rows = Transpose[rows];
3645                            rows = Transpose[{headers,rows}];
3646                            rows = Association[(#[[1]]->#[[2]])&/@rows];
3647                            term->rows
3648                            ),
3649                        {term,carnallTerms}
3650                        ];
3651        carnallAssoc         = Association[parsedData];
3652        carnallCalcEnergies = #[[calcIndex]]&/@carnallData;
3653        carnallCalcEnergies = If[NumberQ[#],#,Missing[]]&/
  @carnallCalcEnergies;
3654        ion                  = ions[[i-3]];
3655        carnallCalcEnergies = PadRight[carnallCalcEnergies, fullSizes[
  ion], Missing[]];
3656        keys                 = #[<|"ion"->ion|>]&/@templates;
3657        Carnall[keys[[1]]]   = carnallAssoc;
3658        Carnall[keys[[2]]]   = carnallCalcEnergies;
3659        Carnall[keys[[3]]]   = carnallData;
3660        Carnall[keys[[4]]]   = headers;
3661        ),
3662    {i,4,14}
3663    ];
3664
3665    goodions = Select[ions,#!="Pm"&];
3666    expData  = Select[Transpose[Carnall["appendix:"<>#<>":RawTable"
  ]][[1+Position[Carnall["appendix:"<>#<>":Headings"],"Exp (1/cm)"
  ][[1,1]]]],NumberQ]&/@goodions;
3667    Carnall["All Experimental Data"]=AssociationThread[goodions,
  expData];
3668    If[OptionValue["Export"],
3669      (
3670        carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
3671        Print["Exporting to "<>exportFname];
3672        Export[carnallFname, Carnall];
3673      )
3674      ];
3675    Return[Carnall];
3676  )
3677
3678  CFP::usage = "CFP[{n, NKSL}] provides a list whose first element
    echoes NKSL and whose other elements are lists with two elements
    the first one being the symbol of a parent term and the second
    being the corresponding coefficient of fractional parentage. n
    must satisfy 1 <= n <= 7";
3679  CFPAssoc::usage = " CFPAssoc is an association where keys are of
    lists of the form {num_electrons, daugherTerm, parentTerm} and
    values are the corresponding coefficients of fractional parentage.
     The terms given in string-spectroscopic notation. If a certain
    daughter term does not have a parent term, the value is 0. Loaded
```

```
  using LoadCFP[].";
LoadCFP::usage="LoadCFP[] loads CFP, CFPAssoc, and CFPTable into the session.";
LoadCFP[]:=(
  If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]],Return[]];

  PrintTemporary["Loading CFPtable ..."];
  CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
  If[!FileExistsQ[CFPTablefname],
    (PrintTemporary[">> CFPTable.m not found, generating ..."];
      CFPTable = GenerateCFPTable["Export"->True];
    ),
    CFPTable = Import[CFPTablefname];
  ];

  PrintTemporary["Loading CFPs.m ..."];
  CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
  If[!FileExistsQ[CFPfname],
    (PrintTemporary[">> CFPs.m not found, generating ..."];
      CFP = GenerateCFP["Export"->True];
    ),
    CFP = Import[CFPfname];
  ];

  PrintTemporary["Loading CFPAssoc.m ..."];
  CFPAfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
  If[!FileExistsQ[CFPAfname],
    (PrintTemporary[">> CFPAssoc.m not found, generating ..."];
      CFPAssoc = GenerateCFPAssoc["Export"->True];
    ),
    CFPAssoc = Import[CFPAfname];
  ];
);

ReducedUkTable::usage = "ReducedUkTable[{n, l = 3, SL, SpLp, k}] provides reduced matrix elements of the spherical tensor operator Uk. See TASS section 11-9 \"Unit Tensor Operators\". Loaded using LoadUk[].";
LoadUk::usage="LoadUk[] loads into session the reduced matrix elements for unit tensor operators.";
LoadUk[]:=(
  If[ValueQ[ReducedUkTable], Return[]];
  PrintTemporary["Loading the association of reduced matrix elements for unit tensor operators ..."];
  ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "ReducedUkTable.m"}];
  If[!FileExistsQ[ReducedUkTableFname],
    (PrintTemporary[">> ReducedUkTable.m not found, generating ..."];
      ReducedUkTable = GenerateReducedUkTable[7];
    ),
    ReducedUkTable = Import[ReducedUkTableFname];
  ];
);

```

```
3726  ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}]
        provides the calculated result of Electrostatic[{n, SL, SpLp}].
        Load using LoadElectrostatic[].";
3727  LoadElectrostatic::usage="LoadElectrostatic[] loads the reduced
        matrix elements for the electrostatic interaction.";
3728  LoadElectrostatic[]:=(
3729    If[ValueQ[ElectrostaticTable], Return[]];
3730    PrintTemporary["Loading the association of matrix elements for the
         electrostatic interaction ..."];
3731    ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "
       ElectrostaticTable.m"}];
3732    If[!FileExistsQ[ElectrostaticTablefname],
3733      (PrintTemporary[">> ElectrostaticTable.m not found, generating
       ..."];
3734        ElectrostaticTable = GenerateElectrostaticTable[7];
3735      ),
3736      ElectrostaticTable = Import[ElectrostaticTablefname];
3737    ];
3738  );
3739
3740  LoadV1k::usage="LoadV1k[] loads into session the matrix elements of
        V1k.";
3741  LoadV1k[]:=(
3742    If[ValueQ[ReducedV1kTable], Return[]];
3743    PrintTemporary["Loading the association of matrix elements for V1k
         ..."];
3744    ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "
       ReducedV1kTable.m"}];
3745    If[!FileExistsQ[ReducedV1kTableFname],
3746      (PrintTemporary[">> ReducedV1kTable.m not found, generating ..."
       ];
3747        ReducedV1kTable = GenerateReducedV1kTable[7];
3748      ),
3749      ReducedV1kTable = Import[ReducedV1kTableFname];
3750    ]
3751  );
3752
3753  LoadSpinOrbit::usage="LoadSpinOrbit[] loads into session the matrix
        elements of the spin-orbit interaction.";
3754  LoadSpinOrbit[]:=(
3755    If[ValueQ[SpinOrbitTable], Return[]];
3756    PrintTemporary["Loading the association of matrix elements for
       spin-orbit ..."];
3757    SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "
       SpinOrbitTable.m"}];
3758    If[!FileExistsQ[SpinOrbitTableFname],
3759      (PrintTemporary[">> SpinOrbitTable.m not found, generating ..."
       ];
3760        SpinOrbitTable = GenerateSpinOrbitTable[7, "Export" -> True];
3761      ),
3762      SpinOrbitTable = Import[SpinOrbitTableFname];
3763    ]
3764  );
3765
```

```
3766  LoadSOOandECSOLS::usage="LoadSOOandECSOLS[] loads into session the
        LS reduced matrix elements of the SOO-ECSO interaction.";
3767  LoadSOOandECSOLS[]:=(
3768    If[ValueQ[SOOandECSOLSTable], Return[]];
3769    PrintTemporary["Loading the association of LS reduced matrix
      elements for SOO-ECSO ..."];
3770    SOOandECSOLSTableFname = FileNameJoin[{moduleDir, "data", "
      ReducedSOOandECSOLSTable.m"}];
3771    If[!FileExistsQ[SOOandECSOLSTableFname],
3772      (PrintTemporary[">> ReducedSOOandECSOLSTable.m not found,
      generating ..."];
3773        SOOandECSOLSTable = GenerateSOOandECSOLSTable[7];
3774      ),
3775      SOOandECSOLSTable = Import[SOOandECSOLSTableFname];
3776    ];
3777  );
3778
3779  LoadSOOandECSO::usage="LoadSOOandECSO[] loads into session the LSJ
      reduced matrix elements of spin-other-orbit and electrostatically-
      correlated-spin-orbit.";
3780  LoadSOOandECSO[]:=(
3781    If[ValueQ[SOOandECSOTableFname], Return[]];
3782    PrintTemporary["Loading the association of matrix elements for
      spin-other-orbit and electrostatically-correlated-spin-orbit ..."
      ];
3783    SOOandECSOTableFname = FileNameJoin[{moduleDir, "data", "
      SOOandECSOTable.m"}];
3784    If[!FileExistsQ[SOOandECSOTableFname],
3785      (PrintTemporary[">> SOOandECSOTable.m not found, generating ..."
      ];
3786        SOOandECSOTable = GenerateSOOandECSOTable[7, "Export"->True];
3787      ),
3788      SOOandECSOTable = Import[SOOandECSOTableFname];
3789    ];
3790  );
3791
3792  LoadT22::usage="LoadT22[] loads into session the matrix elements of
      T22.";
3793  LoadT22[]:=(
3794    If[ValueQ[T22Table], Return[]];
3795    PrintTemporary["Loading the association of reduced T22 matrix
      elements ..."];
3796    T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.
      m"}];
3797    If[!FileExistsQ[T22TableFname],
3798      (PrintTemporary[">> ReducedT22Table.m not found, generating ..."
      ];
3799        T22Table = GenerateT22Table[7];
3800      ),
3801      T22Table = Import[T22TableFname];
3802    ];
3803  );
3804
3805  LoadSpinSpin::usage="LoadSpinSpin[] loads into session the matrix
      elements of spin-spin.";
```

```
3806   LoadSpinSpin[]:=(
3807     If[ValueQ[SpinSpinTable], Return[]];
3808     PrintTemporary["Loading the association of matrix elements for
       spin-spin ..."];
3809     SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "
       SpinSpinTable.m"}];
3810     If[!FileExistsQ[SpinSpinTableFname],
3811       (PrintTemporary[">> SpinSpinTable.m not found, generating ..."];
3812         SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
3813       ),
3814       SpinSpinTable = Import[SpinSpinTableFname];
3815     ];
3816   );
3817
3818   LoadThreeBody::usage="LoadThreeBody[] loads into session the matrix
       elements of three-body configuration-interaction effects.";
3819   LoadThreeBody[]:=(
3820     If[ValueQ[ThreeBodyTable], Return[]];
3821     PrintTemporary["Loading the association of matrix elements for
       three-body configuration-interaction effects ..."];
3822     ThreeBodyFname   = FileNameJoin[{moduleDir, "data", "
       ThreeBodyTable.m"}];
3823     ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "
       ThreeBodyTables.m"}];
3824     If[!FileExistsQ[ThreeBodyFname],
3825       (PrintTemporary[">> ThreeBodyTable.m not found, generating ..."
       ];
3826         {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTables
       [14, "Export" -> True];
3827       ),
3828       ThreeBodyTable = Import[ThreeBodyFname];
3829       ThreeBodyTables = Import[ThreeBodiesFname];
3830     ];
3831   );
3832
3833   (* ############################ Load Functions
       ############################## *)
3834   (*
       ################################################################################
        *)
3835
3836 End[]
3837
3838 LoadTermLabels[];
3839 LoadCFP[];
3840
3841 EndPackage[]
```

5 qonstants.m

```
1 BeginPackage["qonstants`"];
2
3 (* Physical Constants*)
```

```
bohrRadius = 5.29177210903 * 10^-9;
ee = 1.602176634 * 10^-19;

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

EndPackage[];
```

6 qplotter.m

```

BeginPackage["qplotter`"];

GetColor;
IndexMappingPlot;
ListLabelPlot;
AutoGraphicsGrid;

Begin["`Private`"];

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

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

```
        Line[{{#[[1]], 0}, {#[[2]], height}}]} & /@ pairs, opts,
    ImageSize -> 800]]
   )
  ]

TickCompressor[fTicks_] :=
 Module[{avgTicks, prevTickLabel, groupCounter, groupTally, idx,
  tickPosition, tickLabel, avgPosition, groupLabel}, (avgTicks = {};
  prevTickLabel = fTicks[[1, 2]];
  groupCounter = 0;
  groupTally = 0;
  idx = 1;
  Do[({tickPosition, tickLabel} = tick;
    If[
     tickLabel === prevTickLabel,
     (groupCounter += 1;
      groupTally += tickPosition;
      groupLabel = tickLabel;),
     (
      avgPosition = groupTally/groupCounter;
      avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
      groupCounter = 1;
      groupTally = tickPosition;
      groupLabel = tickLabel;
      )
     ];
    If[idx != Length[fTicks],
     prevTickLabel = tickLabel;
     idx += 1;]
    ), {tick, fTicks}];
  If[Or[Not[prevTickLabel === tickLabel], groupCounter > 1],
   (
    avgPosition = groupTally/groupCounter;
    avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
    )
   ];
  Return[avgTicks];)]

GetColor[s_Style] := s /. Style[_, c_] :> c
GetColor[_] := Black

ListLabelPlot::usage="ListLabelPlot[data, labels] takes a list of
    numbers with corresponding labels. The data is grouped according
    to the labels and a ListPlot is created with them so that each
    group has a different color and their corresponding label is shown
     in the horizontal axis.";
Options[ListLabelPlot] = Append[Options[ListPlot], "TickCompression"->
    True];
ListLabelPlot[data_, labels_, opts : OptionsPattern[]] := Module[
  {uniqueLabels, pallete, groupedByTerm, groupedKeys, scatterGroups,
   groupedColors, frameTicks, compTicks, bottomTicks, topTicks},
  (
   uniqueLabels  = DeleteDuplicates[labels];
   pallete = Table[ColorData["Rainbow", i], {i, 0, 1,
      1/(Length[uniqueLabels] - 1)}];
```

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

End[];

EndPackage[];
```

7 misc.m

```
BeginPackage["misc`"];

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

GreedyMatching;
```

```
HelperNotebook;
StochasticMatching;
ExtractSymbolNames;
GetModificationDate;
TextBasedProgressBar;
ToPythonSparseFunction;

FirstOrderPerturbation;
SecondOrderPerturbation;

ToPythonSymPyExpression;
RobustMissingQ;

Begin["`Private`"];

RobustMissingQ[expr_] := (FreeQ[expr, _Missing] === False);

TextBasedProgressBar[progress_, totalIterations_, prefix_:""] :=
    Module[
     {progMessage},
     progMessage = ToString[progress] <> "/" <> ToString[
    totalIterations];
     If[progress < totalIterations,
         WriteString["stdout", StringJoin[prefix, progMessage, "\r"]],
         WriteString["stdout", StringJoin[prefix, progMessage, "\n"]]
     ];
];

FirstOrderPerturbation::usage="Given the eigenValues and eigenVectors
    of a matrix A (which doesn't need to be given) together with a
    corresponding perturbation matrix perMatrix, this function
    calculates the first derivative of the eigenvalues with respect to
     the scale factor of the perturbation matrix. In the sense that
    the eigenvalues of the matrix A + β perMatrix are to first order
    equal to \[Lambda] + \[Delta]_i β, where the \[Delta]_i are the
    returned values. The eigenvalues and eigenvectors are assumed to
    be given in the same order, i.e. the ith eigenvalue corresponds to
     the ith eigenvector. This assuming that the eigenvalues are non-
    degenerate.";
FirstOrderPerturbation[eigenValues_, eigenVectors_,
  perMatrix_] := (Diagonal[
   eigenVectors . perMatrix . Transpose[eigenVectors]])

SecondOrderPerturbation::usage="Given the eigenValues and eigenVectors
     of a matrix A (which doesn't need to be given) together with a
    corresponding perturbation matrix perMatrix, this function
    calculates the second derivative of the eigenvalues with respect
    to the scale factor of the perturbation matrix. In the sense that
    the eigenvalues of the matrix A + β perMatrix are to second order
    equal to \[Lambda] + \[Delta]_i β + \[Delta]_i^{(2)}/2 β^2, where
    the \[Delta]_i^{(2)} are the returned values. The eigenvalues and
    eigenvectors are assumed to be given in the same order, i.e. the
    ith eigenvalue corresponds to the ith eigenvector. This assuming
    that the eigenvalues are non-degenerate.";
SecondOrderPerturbation[eigenValues_, eigenVectors_, perMatrix_] := (
```

```
  dim = Length[perMatrix];
  eigenBras = Conjugate[eigenVectors];
  eigenKets = eigenVectors;
  matV = Abs[eigenBras . perMatrix . Transpose[eigenKets]]^2;
  OneOver[x_, y_] := If[x == y, 0, 1/(x - y)];
  eigenDiffs = Outer[OneOver, eigenValues, eigenValues, 1];
  pProduct = Transpose[eigenDiffs]*matV;
  Return[2*(Total /@ Transpose[pProduct])];
  )

SuperIdentity::usage="SuperIdentity[args] returns the arguments passed to it. This is useful for defining a function that does nothing, but that can be used in a composition.";
SuperIdentity[args___] := {args};

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

RecoverBasis::usage="RecoverBasis[{flatBasisLS, flatbasisNums}] takes the output of FlattenBasis and returns the original basis. The input is a list with two elements {flatbasisLS, flatbasisNums}.";
RecoverBasis[{flatbasisLS_, flatbasisNums_}] := Module[{recBasis},
  (
   recBasis = {{{#[[1]], #[[2]]}, #[[3]]}, #[[4]]} & /@ (Flatten /@
       Transpose[{flatbasisLS,
         Partition[Round[2*#]/2 & /@ flatbasisNums, 3]}]);
   Return[recBasis];
   )
  ]

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

ExportToH5::usage =
  "ExportToH5[fname, Hold[{symbol1, symbol2, ...}]] takes an .h5 filename and a held list of symbols and export to the .h5 file the values of the symbols with keys equal the symbol names. The values of the symbols cannot be arbitrary, for instance a list
```

```
    with mixes numbers and string will fail, but an Association with
    mixed values exports ok. Do give it a try.
  If the file is already present in disk, this function will overwrite
     it by default. If the value of a given symbol contains symbolic
    numbers, e.g. \[Pi], these will be converted to floats in the
    exported file.";
Options[ExportToH5] = {"Overwrite" -> True};
ExportToH5[fname_String, symbols_Hold, OptionsPattern[]] := (
  If[And[FileExistsQ[fname], OptionValue["Overwrite"]],
   (
    Print["File already exists, overwriting ..."];
    DeleteFile[fname];
    )
   ];
  symbolNames = ExtractSymbolNames[symbols];
  Do[(Print[symbolName];
    Export[fname, ToExpression[symbolName], {"Datasets", symbolName},
     OverwriteTarget -> "Append"]
    ), {symbolName, symbolNames}]
  )

GreedyMatching::usage="GreedyMatching[aList, bList] returns a list of
    pairs of elements from aList and bList that are closest to each
    other, this is returned in a list together with a mapping of
    indices from the aList to those in bList to which they were
    matched. The option \"alistLabels\" can be used to specify labels
    for the elements in aList. The option \"blistLabels\" can be used
    to specify labels for the elements in bList. If these options are
    used, the function returns a list with three elements the pairs of
     matched elements, the pairs of corresponding matched labels, and
    the mapping of indices.";
Options[GreedyMatching] = {
    "alistLabels" -> {},
    "blistLabels" -> {}};
GreedyMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{
   aValues = aValues0,
   bValues = bValues0,
   bValuesOriginal = bValues0,
   bestLabels, bestMatches,
   bestLabel, aElement, givenLabels,
   aLabels, aLabel,
   diffs, minDiff,
   bLabels,
   minDiffPosition, bestMatch},
  (
   aLabels     = OptionValue["alistLabels"];
   bLabels     = OptionValue["blistLabels"];
   bestMatches = {};
   bestLabels  = {};
   givenLabels = (Length[aLabels] > 0);
   Do[
    (
     aElement        = aValues[[idx]];
     diffs           = Abs[bValues - aElement];
     minDiff         = Min[diffs];
```

```
126       minDiffPosition = Position[diffs, minDiff][[1, 1]];
127       bestMatch       = bValues[[minDiffPosition]];
128       bestMatches     = Append[bestMatches, {aElement, bestMatch}];
129       If[givenLabels,
130        (
131         aLabel     = aLabels[[idx]];
132         bestLabel  = bLabels[[minDiffPosition]];
133         bestLabels = Append[bestLabels, {aLabel, bestLabel}];
134         bLabels    = Drop[bLabels, {minDiffPosition}];
135         )
136        ];
137       bValues = Drop[bValues, {minDiffPosition}];
138       If[Length[bValues] == 0, Break[]];
139       ),
140      {idx, 1, Length[aValues]}
141      ];
142     pairedIndices = MapIndexed[{#2[[1]], Position[bValuesOriginal,
      #1[[2]]][[1, 1]]} &, bestMatches];
143     If[givenLabels,
144      Return[{bestMatches, bestLabels, pairedIndices}],
145      Return[{bestMatches, pairedIndices}]
146      ]
147     )
148    ]
149
150 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.";
151 Options[StochasticMatching] = {"alistLabels" -> {},
152    "blistLabels" -> {}};
153 StochasticMatching[aValues0_, bValues0_, numShuffles_ : 200,
      OptionsPattern[]] := Module[{
154    aValues = aValues0,
155    bValues = bValues0,
156    matchingLabels, ranger, matches, noShuff, bestMatch, highestCost,
      lowestCost, dev, sorter, bestValues,
157    pairedIndices, bestLabels, matchedIndices, shuffler
158    },
159   (
160    matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
161    ranger = Range[1, Length[aValues]];
162    matches = If[Not[matchingLabels], (
163       Table[(
164         shuffler = If[i == 1, ranger, RandomSample[ranger]];
165         {bestValues, matchedIndices} =
166          GreedyMatching[aValues[[shuffler]], bValues];
```

```
        cost = Total[Abs[#[[1]] - #[[2]]] & /@ bestValues];
        {cost, {bestValues, matchedIndices}}
        ), {i, 1, numShuffles}]
      ),
     Table[(
       shuffler = If[i == 1, ranger, RandomSample[ranger]];
       {bestValues, bestLabels, matchedIndices} =
        GreedyMatching[aValues[[shuffler]], bValues,
         "alistLabels" -> OptionValue["alistLabels"][[shuffler]],
         "blistLabels" -> OptionValue["blistLabels"]];
       cost = Total[Abs[#[[1]] - #[[2]]] & /@ bestValues];
       {cost, {bestValues, bestLabels, matchedIndices}}
       ), {i, 1, numShuffles}]
     ];
   noShuff = matches[[1, 1]];
   matches = SortBy[matches, First];
   bestMatch = matches[[1, 2]];
   highestCost = matches[[-1, 1]];
   lowestCost = matches[[1, 1]];
   dev = StandardDeviation[First /@ matches];
   Print[lowestCost, " <-> ", highestCost, " | \[Sigma]=", dev,
    " | N=", numShuffles, " | null=", noShuff];
   If[matchingLabels,
    (
     {bestValues, bestLabels, matchedIndices} = bestMatch;
     sorter = Ordering[First /@ bestValues];
     bestValues = bestValues[[sorter]];
     bestLabels = bestLabels[[sorter]];
     pairedIndices =
      MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
       bestValues];
     Return[{bestValues, bestLabels, pairedIndices}]
     ),
    (
     {bestValues, matchedIndices} = bestMatch;
     sorter = Ordering[First /@ bestValues];
     bestValues = bestValues[[sorter]];
     pairedIndices =
      MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
       bestValues];
     Return[{bestValues, pairedIndices}]
     )
    ];
   )
  ]

FlowMatching::usage="FlowMatching[aList, bList] returns a list of
    pairs of elements from aList and bList that are closest to each
    other, this is returned in a list together with a mapping of
    indices from the aList to those in bList to which they were
    matched. The option \"alistLabels\" can be used to specify labels
    for the elements in aList. The option \"blistLabels\" can be used
    to specify labels for the elements in bList. If these options are
    used, the function returns a list with three elements the pairs of
     matched elements, the pairs of corresponding matched labels, and
```

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

   (*From all the elements of A to all the elements of B*)
   midLayer = Table[{{"A", i} \[DirectedEdge] ({"B", j}), 1, cFun[
    aValues[[i]], bValues[[j]]]}, {i, 1, nA}, {j, 1, nB}];
   midLayer = Flatten[midLayer, 1];
   {midLayerEdges, midCapacities, midCosts} = Transpose[midLayer];

   (*From the elements of B to the sink*)
   edgesBtoSink = ({"B", #} \[DirectedEdge] "sink") & /@ Range[1, nB];
   capacityBtoSink = ConstantArray[1, nB];
   costBtoSink = ConstantArray[0, nB];

   (*Put it all together*)
   allCapacities = Join[capacitySourceToA, midCapacities,
    capacityBtoSink];
   allCosts = Join[costSourceToA, midCosts, costBtoSink];
   allEdges = Join[edgesSourceToA, midLayerEdges, edgesBtoSink];
   graph = Graph[allEdges, EdgeCapacity -> allCapacities,
     EdgeCost -> allCosts];

   (*Solve it*)
   flow = FindMinimumCostFlow[graph, "source", "sink", nA -
    OptionValue["notMatched"], "OptimumFlowData"];
```

```
    (*Collect the pairs of matched indices*)
    pairedIndices = Select[flow["EdgeList"], And[Not[#[[1]] === "source
     "], Not[#[[2]] === "sink"]] &];
    pairedIndices = {#[[1, 2]], #[[2, 2]]} & /@ pairedIndices;
    (*Collect the pairs of matched values*)
    bestValues = {aValues[[#[[1]]]], bValues[[#[[2]]]]} & /@
     pairedIndices;
    (*Account for having been given labels*)
    If[matchingLabels,
     (
      bestLabels = {aLabels[[#[[1]]]], bLabels[[#[[2]]]]} & /@
     pairedIndices;
      Return[{bestValues, bestLabels, pairedIndices}]
      ),
     (
      Return[{bestValues, pairedIndices}]
      )
     ];
    )
   ]

HelperNotebook::usage="HelperNotebook[nbName] creates a separate
    notebook and returns a function that can be used to print to the
    bottom of it. The name of the notebook, nbName, is optional and
    defaults to OUT.";
HelperNotebook[nbName_:"OUT"] :=
 Module[{screenDims, screenWidth, screenHeight, nbWidth, leftMargin,
    PrintToOutputNb}, (
    screenDims =
     SystemInformation["Devices", "ScreenInformation"][[1, 2, 2]];
    screenWidth = screenDims[[1, 2]];
    screenHeight = screenDims[[2, 2]];
    nbWidth = Round[screenWidth/3];
    leftMargin = screenWidth - nbWidth;
    outputNb = CreateDocument[{}, WindowTitle -> nbName,
      WindowMargins -> {{leftMargin, Automatic}, {Automatic,
         Automatic}}, WindowSize -> {nbWidth, screenHeight}];
    PrintToOutputNb[text_] :=
     (
         SelectionMove[outputNb, After, Notebook];
         NotebookWrite[outputNb, Cell[BoxData[ToBoxes[text]], "Output"
    ]];
     );
    Return[PrintToOutputNb]
    )
   ]

GetModificationDate::usage="GetModificationDate[fname] returns the
    modification date of the given file.";
GetModificationDate[theFileName_] := FileDate[theFileName, "
    Modification"];

(*Helper function to convert Mathematica expressions to standard form
    *)
```

```
StandardFormExpression[expr0_] := Module[{expr=expr0}, ToString[expr,
    InputForm]];

(*Helper function to translate to Python/SymPy expressions*)
ToPythonSymPyExpression::usage="ToPythonSymPyExpression[expr] converts
     a Mathematica expression to a SymPy expression. This is a little
    iffy and might break if the expression includes Mathematica
    functions that haven't been given a SymPy equivalent.";
ToPythonSymPyExpression[expr0_] := Module[{standardForm, expr=expr0},
   standardForm = StandardFormExpression[expr];
   StringReplace[standardForm, {
     "Power[" -> "Pow(",
     "Sqrt[" -> "sqrt(",
     "[" -> "(",
     "]" -> ")",
     "\\" -> "",
     (*Remove special Mathematica backslashes*)
     "/" -> "/" (*Ensure division is represented with a slash*)}]];

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

```
      StringRiffle[ToString /@ rowPointers, ", "], "])\n",
      "    shape = (", StringRiffle[ToString /@ dimensions, ", "],
      ")\n",
      "    return csr_matrix((data, indices, indptr), shape=shape)"];
   pyCode
   ];

End[];
EndPackage[];
```