

# qlanth

## 1 Notation

Shorthand for all other quantum numbers

$$\overline{\Lambda}$$

(1)

orbital angular momentum

$$\underline{\ell}$$

(2)

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

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

(3)

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

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

(4)

spherical tensor operator of rank k

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

(5)

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

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

(6)

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

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

(7)

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

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

(8)

## 2 Definitions

irreducible unit tensor operator of rank k

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

(9)

symmetric unit tensor operator for n equivalent electrons

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

(10)

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

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

(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 an 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 few 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  $f^n$  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 an 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  $\alpha$ , which in the notation of Nielson and Koster is simply an integer number that enumerates all the possible LS in a given  $f^n$  configuration.

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

$$\hat{\mathcal{H}} = \underbrace{\hat{\mathcal{H}}_k}_{\text{kinetic}} + \underbrace{\hat{\mathcal{H}}_{e:sn}}_{\text{e:shielded nuc}} + \underbrace{\hat{\mathcal{H}}_{e:e}}_{\text{e:e}} + \underbrace{\hat{\mathcal{H}}_{s:o}}_{\text{spin-orbit}} + \underbrace{\hat{\mathcal{H}}_{cf}}_{\text{crystal field}} + \underbrace{\hat{\mathcal{H}}_{s:s:o:o}}_{\substack{\text{spin:spin} \\ \text{and spin:other-orbit}}} \quad (12)$$

$$+ \underbrace{\hat{\mathcal{H}}_{\mathcal{SO}(3)}}_{\text{Trees effective op}} + \underbrace{\hat{\mathcal{H}}_{G_2}}_{\text{G}_2 \text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{\mathcal{SO}(7)}}_{\text{SO}(7) \text{ effective op}} + \underbrace{\hat{\mathcal{H}}_{f_3}}_{\text{effective three-body}} + \underbrace{\hat{\mathcal{H}}_{ec-s:o}}_{\text{electrostatically correlated spin:orbit}} \quad (13)$$

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

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

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

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

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

$$\hat{\mathcal{H}}_{s:s:o:o} = \sum_{i=0,2,4} M^i \hat{m}_i \quad (19)$$

$$\mathcal{C}(\mathcal{G}) := \text{The Casimir operator of group } \mathcal{G}. \quad (20)$$

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

$$\hat{\mathcal{H}}_{G_2} = \beta \mathcal{C}(G_2) \quad (22)$$

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

$$\hat{\mathcal{H}}_{f_3} = T'^2 t'_2 + \sum_{i=2,3,4,6,7,8}^N T^i \hat{t}_i \quad (\text{effective three-body operators } \hat{t}_i \text{ with strengths } T_i)^2 \quad (24)$$

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

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

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

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

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

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 omitted. 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 $\hat{\mathcal{H}}_{e:e}$ : e:e repulsion

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

This term is the first we will not discard. Calculating this term for the  $\mathbb{f}^n$  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  $\mathbb{f}^n$  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  $\mathbb{f}^n$  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.

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

where

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

## 4 qlanth.m

```

1  (* -----+-----+
2  |
3  |
4  |
5  |
6  |
7  |
8  |
9  |
10 |
11 |
12 |-----+-----+
13 This code was initially authored by Christopher Dodson and then
14 rewritten by David Lizarazo in the years 2022-2024. It has also
15 benefited from the discussions with Tharnier Puel.
16
17 It uses an effective Hamiltonian to describe the electronic
18 structure of lanthanide ions in crystals. This effective Hamiltonian
19 includes terms representing the following interactions/relativistic
20 corrections: spin-orbit, electrostatic repulsion, spin-spin, crystal
21 field and spin-other-orbit.
22
23 The Hilbert space used in this effective Hamiltonian is limited to
24 single f^n configurations. The inaccuracy of this single
25 configuration description is partially compensated by the inclusion
26 of configuration interaction terms as parametrized by the Casimir
27 operators of SO(3), G(2), and SO(7), and by three-body effective
28 operators ti.
29
30 The parameters included in this model are listed in the string
31 paramAtlas.
32
33 The notebook "qlanth.nb" contains a gallery with all the functions
34 included in this module with some simple use cases.
35
36 The notebook "The Lanthanides in LaF3.nb" is an example in which the
37 results from this code are compared against the published results by
38 Carnall et. al for the energy levels of lanthanide ions in crystals
39 of lanthanum fluoride.
40
41 REFERENCES:
42
43 + Condon, E U, and G H Shortley. The Theory of Atomic Spectra, 1935.
44
45 + Racah, Giulio. "Theory of Complex Spectra. III." Physical Review
46 63, no. 9-10 (May 1, 1943): 367-82.
47 https://doi.org/10.1103/PhysRev.63.367.
48
49 + Racah, Giulio. "Theory of Complex Spectra. II." Physical Review
50 62, no. 9-10 (November 1, 1942): 438-62.
51 https://doi.org/10.1103/PhysRev.62.438.
52

```

53 + Rajnak, K, and BG Wybourne. "Configuration Interaction Effects in  
54  $1^N$  Configurations." Physical Review 132, no. 1 (1963): 280.  
55 <https://doi.org/10.1103/PhysRev.132.280>.

56

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

58

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

62

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

65

66 + Judd, BR, HM Crosswhite, and Hannah Crosswhite. "Intra-Atomic  
67 Magnetic Interactions for f Electrons." Physical Review 169, no. 1  
68 (1968): 130. <https://doi.org/10.1103/PhysRev.169.130>.

69

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

73

74 + Judd, BR, and MA Suskin. "Complete Set of Orthogonal Scalar  
75 Operators for the Configuration  $f^3$ ." JOSA B 1, no. 2 (1984):  
76 261-65. <https://doi.org/10.1364/JOSAB.1.000261>.

77

78 + Carnall, W. T., G. L. Goodman, K. Rajnak, and R. S. Rana. "A  
79 Systematic Analysis of the Spectra of the Lanthanides Doped into  
80 Single Crystal  $\text{LaF}_3$ ." The Journal of Chemical Physics 90, no. 7  
81 (1989): 3443-57. <https://doi.org/10.1063/1.455853>.

82

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

87

88 + Velkov, Dobromir. "Multi-Electron Coefficients of Fractional  
89 Parentage for the p, d, and f Shells." John Hopkins University,  
90 2000.

91

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

97

98

99 ----- \*)

100

101 BeginPackage["qlanth"];  
102 Needs["qconstants"];  
103 Needs["qplotter"];

104

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

```

108 E2: linear combination of F_k
109 E3: linear combination of F_k,
110
111 ζ: spin-orbit strength parameter.
112
113 F0: Direct Slater integral F^0, produces an overall shift of all
    energy levels.
114 F2: Direct Slater integral F^2
115 F4: Direct Slater integral F^4, possibly constrained by ratio to F^2
116 F6: Direct Slater integral F^6, possibly constrained by ratio to F^2
117
118 M0: 0th Marvin integral
119 M2: 2nd Marvin integral
120 M4: 4th Marvin integral
121 \[Sigma]SS: spin-spin override, if 0 spin-spin omitted, and 1 if
    included
122
123 T2: three-body effective operator parameter T^2
124 T2p: three-body effective operator parameter T^2'
125 T3: three-body effective operator parameter T^3
126 T4: three-body effective operator parameter T^4
127 T6: three-body effective operator parameter T^6
128 T7: three-body effective operator parameter T^7
129 T8: three-body effective operator parameter T^8
130
131 T11: three-body effective operator parameter T^11
132 T11p: three-body effective operator parameter T^11'
133 T12: three-body effective operator parameter T^12
134 T14: three-body effective operator parameter T^14
135 T15: three-body effective operator parameter T^15
136 T16: three-body effective operator parameter T^16
137 T17: three-body effective operator parameter T^17
138 T18: three-body effective operator parameter T^18
139 T19: three-body effective operator parameter T^19
140
141 P0: 0th parameter for the two-body electrostatically correlated spin-
    orbit interaction
142 P2: 2nd parameter for the two-body electrostatically correlated spin-
    orbit interaction
143 P4: 4th parameter for the two-body electrostatically correlated spin-
    orbit interaction
144 P6: 6th parameter for the two-body electrostatically correlated spin-
    orbit interaction
145
146 gs: electronic gyromagnetic ratio
147
148 α: Trees' parameter α describing configuration interaction via the
    Casimir operator of S0(3)
149 β: Trees' parameter β describing configuration interaction via the
    Casimir operator of G(2)
150 γ: Trees' parameter γ describing configuration interaction via the
    Casimir operator of S0(7)
151
152 B02: crystal field parameter B_0^2 (real)
153 B04: crystal field parameter B_0^4 (real)

```

```

154 B06: crystal field parameter B_0^6 (real)
155 B12: crystal field parameter B_1^2 (real)
156 B14: crystal field parameter B_1^4 (real)
157
158 B16: crystal field parameter B_1^6 (real)
159 B22: crystal field parameter B_2^2 (real)
160 B24: crystal field parameter B_2^4 (real)
161 B26: crystal field parameter B_2^6 (real)
162 B34: crystal field parameter B_3^4 (real)
163
164 B36: crystal field parameter B_3^6 (real)
165 B44: crystal field parameter B_4^4 (real)
166 B46: crystal field parameter B_4^6 (real)
167 B56: crystal field parameter B_5^6 (real)
168 B66: crystal field parameter B_6^6 (real)
169
170 S12: crystal field parameter S_1^2 (real)
171 S14: crystal field parameter S_1^4 (real)
172 S16: crystal field parameter S_1^6 (real)
173 S22: crystal field parameter S_2^2 (real)
174
175 S24: crystal field parameter S_2^4 (real)
176 S26: crystal field parameter S_2^6 (real)
177 S34: crystal field parameter S_3^4 (real)
178 S36: crystal field parameter S_3^6 (real)
179
180 S44: crystal field parameter S_4^4 (real)
181 S46: crystal field parameter S_4^6 (real)
182 S56: crystal field parameter S_5^6 (real)
183 S66: crystal field parameter S_6^6 (real)
184
185 \[Epsilon]: ground level baseline shift
186 t2Switch: controls the usage of the t2 operator beyond f7
187 wChErrA: If 1 then the type-A errors in Chen are used, if 0 then not.
188 wChErrB: If 1 then the type-B errors in Chen are used, if 0 then not.
189 ";
190 paramSymbols = StringSplit[paramAtlas, "\n"];
191 paramSymbols = Select[paramSymbols, # != "" &];
192 paramSymbols = ToExpression[StringSplit[#, ":"][[1]]] & /@
    paramSymbols;
193 Protect /@ paramSymbols;
194 paramLines = Select[StringSplit[paramAtlas, "\n"], # != "" &];
195 usageTemplate = StringTemplate["'paramSymbol'::usage=\n'paramUsage'\n";
    ""];
196 Do[(
197     {paramString, paramUsage} = StringSplit[paramLine, ":"];
198     paramUsage = StringTrim[paramUsage];
199     expressionString = usageTemplate[<|"paramSymbol" -> paramString, "
    paramUsage" -> paramUsage|>];
200     ToExpression[usageTemplate[<|"paramSymbol" -> paramString,
    "paramUsage" -> paramUsage|>]]
201 ),
202 {paramLine, paramLines}
203 ];
204 ];
205

```



```

206 (* Parameter families*)
207 cfSymbols = {B02, B04, B06, B12, B14, B16, B22, B24, B26, B34, B36,
208     B44, B46, B56, B66, S12, S14, S16, S22, S24, S26, S34, S36, S44,
209     S46, S56, S66};
210
211 TSymbols = {T2, T2p, T3, T4, T6, T7, T8, T11, T11p, T12, T14, T15, T16
    , T17, T18, T19};
212
213 AllowedJ;
214 AllowedMforJ;
215 AllowedNKSLJMforJMTerms;
216 AllowedNKSLJMforJTerms;
217
218 AllowedNKSLJTerms;
219 AllowedNKSLTerms;
220 AllowedNKSLforJTerms;
221 AllowedSLJMTerms;
222 AllowedSLJTerms;
223
224 AllowedSLTerms;
225 BasisLSJMJ;
226 Bqk;
227 CFP;
228 CFPAssoc;
229
230 CFPTable;
231 CFPTerms;
232 Carnall;
233 CasimirG2;
234 CasimirS03;
235 CasimirS07;
236
237 Cqk;
238 CrystalField;
239 Dk;
240 ElectrostaticConfigInteraction;
241 Electrostatic;
242
243 ElectrostaticTable;
244 EnergyLevelDiagram;
245 EnergyStates;
246 ExportMZip;
247 BasisTableGenerator;
248 EtoF;
249 ExportmZip;
250 fsubk;
251 fsupk;
252
253 FastIonSolverLaF3;
254 FindNKLSLTerm;
255 FindSL;
256
257 FtoE;
258 GG2U;
259 GS07W;

```

```

260 GenerateCFP;
261 GenerateCFPAssoc;
262
263 GenerateCFPTable;
264 GenerateCrystalFieldTable;
265 GenerateElectrostaticTable;
266 GenerateReducedUkTable;
267 GenerateReducedV1kTable;
268
269 GenerateS00andECSOLSTable;
270 GenerateS00andECSOTable;
271 GenerateSpinOrbitTable;
272 GenerateSpinSpinTable;
273 GenerateT22Table;
274
275 GenerateThreeBodyTables;
276 GenerateThreeBodyTables;
277 Generator;
278 HamMatrixAssembly;
279 HamiltonianForm;
280
281 HamiltonianMatrixPlot;
282 HoleElectronConjugation;
283 IonSolverLaF3;
284 ImportMZip;
285 JJBBlockMatrix;
286 JJBBlockMatrixFileName;
287
288 JJBBlockMatrixTable;
289 LabeledGrid;
290 LoadAll;
291 LoadCFP;
292 LoadCarnall;
293
294 LoadChenDeltas;
295 LoadElectrostatic;
296 LoadGuillotParameters;
297 LoadParameters;
298 LoadS00andECSO;
299
300 LoadS00andECSOLS;
301 LoadSpinOrbit;
302 LoadSpinSpin;
303 LoadSymbolicHamiltonians;
304 LoadT11;
305
306 LoadT22;
307 LoadTermLabels;
308 LoadThreeBody;
309 LoadUk;
310 LoadV1k;
311
312 MagneticInteractions;
313 MaxJ;
314 MinJ;

```

```

315 NKCFPPPhase;
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 ReducedS00andECS0inf2;
332 ReducedS00andECS0infN;
333 ReducedT11inf2;
334
335 ReducedT22inf2;
336 ReducedUk;
337 ReducedUkTable;
338 ReducedV1kTable;
339 Reducedt11inf2;
340
341 ReplaceInSparseArray;
342 RobustMissingQ;
343 SimplerSymbolicHamMatrix;
344 S00andECS0;
345 S00andECS0Table;
346 Seniority;
347
348 ShiftedLevels;
349 SixJay;
350 SpinOrbit;
351 SpinSpin;
352 SpinSpinTable;
353
354 Sqr;
355 SquarePrimeToNormal;
356 T11n;
357 T22n;
358 TP0;
359
360 TabulateJJBlockMatrixTable;
361 TabulateManyJJBlockMatrixTables;
362 TextBasedProgressBar;
363 ScalarOperatorProduct;
364 ThreeBodyTable;
365
366 ThreeBodyTables;
367 ThreeJay;
368 TotalCFilters;
369 chenDeltas;

```

```

370 fK;
371
372 fnTermLabels;
373 moduleDir;
374 symbolicHamiltonians;
375
376 (* this selects the function that is applied
377 to calculated matrix elements *)
378 SimplifyFun = Expand;
379
380 Begin["Private"]
381
382 moduleDir = DirectoryName[$InputFileName];
383 frontEndAvailable = (Head[$FrontEnd] === FrontEndObject);
384
385 (*
386 #####
387 *)
388 (* ##### MISC
389 ##### *)
390
391 RobustMissingQ[expr_] := (FreeQ[expr, _Missing] === False);
392
393 TPO::usage="Two plus one.";
394 TPO[args_] := Times @@ ((2*# + 1) & /@ {args});
395
396 Phaser::usage = "Phaser[x] returns (-1)^x";
397 Phaser[exponent_] := ((-1)^exponent);
398
399 TriangleCondition[a_, b_, c_] := (Abs[b - c] <= a <= (b + c));
400
401 TriangleAndSumCondition[a_, b_, c_] := (And[Abs[b - c] <= a <= (b +
402 c), IntegerQ[a + b + c]]);
403
404 TextBasedProgressBar[progress_, totalIterations_, prefix_:""] :=
405 Module[
406 {progMessage},
407 progMessage = ToString[progress] <> "/" <> ToString[
408 totalIterations];
409 If[progress < totalIterations,
410 WriteString["stdout", StringJoin[prefix, progMessage, "\r"
411 ]],
412 WriteString["stdout", StringJoin[prefix, progMessage, "\n"]]]
413 ];
414
415 SquarePrimeToNormal::usage = "Given a list with the parts
416 corresponding to the squared prime representation of a number,
417 this function parses the result into standard notation.";
418 SquarePrimeToNormal[squarePrime_] :=
419 (
420 radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {idx1
421 , 2, Length[squarePrime]}];
422 radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" -> 13};
423 val = squarePrime[[1]] * Sqrt[radical];

```

```

415     Return[val];
416 );
417
418 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.
419 The function additionally takes an option \"Print\" that if set to
    True, will print the symbols that were not present in the given
    association.";
420 Options[ParamPad] = {"Print" -> True}
421 ParamPad[params_, OptionsPattern[]] := (
422     notPresentSymbols = Complement[paramSymbols, Keys[params]];
423     If[OptionValue["Print"],
424         Print["Symbols not in given params: ",
425             notPresentSymbols]
426     ];
427     newParams = Transpose[{paramSymbols, ConstantArray[0, Length[
    paramSymbols]]}];
428     newParams = (#[[1]] -> #[[2]]) & /@ newParams;
429     newParams = Association[newParams];
430     newParams = Join[newParams, params];
431     Return[newParams];
432 )
433
434 (*
    #####
    *)
435 (* ##### Racah Algebra
    ##### *)
436
437 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.";
438 ReducedUk[numE_, l_, SL_, SpLp_, k_] :=
439     Module[{orbital, Uk, S, L, Sp, Lp, Sb, Lb, parentSL, cfpSL,
    cfpSpLp, Ukval, SLparents, SLpparents, commonParents, phase},
440         {spin, orbital} = {1/2, 3};
441         {S, L} = FindSL[SL];
442         {Sp, Lp} = FindSL[SpLp];
443         If[Not[S == Sp],
444             Return[0]
445         ];
446         cfpSL = CFP[{numE, SL}];
447         cfpSpLp = CFP[{numE, SpLp}];
448         SLparents = First /@ Rest[cfpSL];
449         SLpparents = First /@ Rest[cfpSpLp];
450         commonParents = Intersection[SLparents, SLpparents];
451         Uk = Sum[(
452             {Sb, Lb} = FindSL[\"[Psi]b\"];
453             Phaser[Lb] *
454             CFPAssoc[{numE, SL, \"[Psi]b\"]; *
455             CFPAssoc[{numE, SpLp, \"[Psi]b\"]; *
456             SixJay[{orbital, k, orbital}, {L, Lb, Lp}]

```

```

457     ),
458     {\[Psi]b, commonParents}
459 ];
460 phase = Phaser[orbital + L + k];
461 prefactor = numE * phase * Sqrt[TPO[L,Lp]];
462 Ukval = prefactor*Uk;
463 Return[Ukval];
464 ]
465
466 Ck::usage = "Diagonal reduced matrix element <1||C^(k)||1> where the
Subscript[C, q]^k are reduced spherical harmonics. See equation
11.23 in TASS with l=1'.";
467 Ck[orbital_, k_] := (-1)^orbital * TPO[orbital] * ThreeJay[{orbital,
0}, {k, 0}, {orbital, 0}]
468
469 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.";
470 SixJay[{j1_, j2_, j3_}, {j4_, j5_, j6_}] := (
471     sixJayval =
472     Which[
473     Not[TriangleAndSumCondition[j1, j2, j3]],
474     0,
475     Not[TriangleAndSumCondition[j1, j5, j6]],
476     0,
477     Not[TriangleAndSumCondition[j4, j2, j6]],
478     0,
479     Not[TriangleAndSumCondition[j4, j5, j3]],
480     0,
481     True,
482     SixJSymbol[{j1, j2, j3}, {j4, j5, j6}]];
483 SixJay[{j1, j2, j3}, {j4, j5, j6}] = sixJayval);
484
485 ThreeJay::usage = "ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] gives the
value of the Wigner 3j-symbol and memorizes the computed value.";
486 ThreeJay[{j1_, m1_}, {j2_, m2_}, {j3_, m3_}] := (
487     threejval = Which[
488     Not[(m1 + m2 + m3) == 0],
489     0,
490     Not[TriangleCondition[j1,j2,j3]],
491     0,
492     True,
493     ThreeJSymbol[{j1, m1}, {j2, m2}, {j3, m3}]
494     ];
495     ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] = threejval);
496
497 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.";
498 ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
499     {Vk1, S, L, Sp, Lp, Sb, Lb, spin, orbital, cfpSL, cfpSpLp,
500     SLparents, SpLpparents, commonParents, prefactor},
501     {spin, orbital} = {1/2, 3};
502     {S, L} = FindSL[SL];
503     {Sp, Lp} = FindSL[SpLp];

```

```

504 cfpSL      = CFP[{numE, SL}];
505 cfpSpLp    = CFP[{numE, SpLp}];
506 SLparents  = First /@ Rest[cfpSL];
507 SpLpparents = First /@ Rest[cfpSpLp];
508 commonParents = Intersection[SLparents, SpLpparents];
509 Vk1 = Sum[(
510     {Sb, Lb} = FindSL[\[Psi]b];
511     Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
512     CFPAssoc[{numE, SL, \[Psi]b}] *
513     CFPAssoc[{numE, SpLp, \[Psi]b}] *
514     SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
515     SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
516 ),
517 {\[Psi]b, commonParents}
518 ];
519 prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp, Lp]
520 ];
521 Return[prefactor * Vk1];
522 ]
523
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.";
524 Options[GenerateReducedUkTable] = {"Export" -> True, "Progress" ->
    True};
525 GenerateReducedUkTable[numEmax_Integer:7, OptionsPattern[]]:= (
526     numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
527     AllowedNKSLTerms[#]]&/@Range[1, numEmax]] * 4;
528     Print["Calculating " <> ToString[numValues] <> " values for Uk k
    =0,2,4,6."];
529     counter = 1;
530     If[And[OptionValue["Progress"], frontEndAvailable],
531     progBar = PrintTemporary[
532     Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
533     counter}]]]
534     ];
535     ReducedUkTable = Table[
536     (
537     counter = counter+1;
538     {numE, 3, SL, SpLp, k} -> SimplifyFun[ReducedUk[numE, 3, SL,
539     SpLp, k]]
540     ),
541     {numE, 1, numEmax},
542     {SL, AllowedNKSLTerms[numE]},
543     {SpLp, AllowedNKSLTerms[numE]},
544     {k, {0, 2, 4, 6}}
545     ];
546     ReducedUkTable = Association[Flatten[ReducedUkTable]];
547     ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "
    ReducedUkTable.m"}];
548     If[And[OptionValue["Progress"], frontEndAvailable],
549     NotebookDelete[progBar]
550     ];

```

```

549     If[OptionValue["Export"],
550     (
551         Print["Exporting to file " <> ToString[ReducedUkTableFname]];
552         Export[ReducedUkTableFname, ReducedUkTable];
553     )
554 ];
555 Return[ReducedUkTable];
556 )
557
558 GenerateReducedVikTable::usage = "GenerateReducedVikTable[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
    /ReducedVikTable.m."
559 Options[GenerateReducedVikTable] = {"Export" -> True, "Progress" ->
    True};
560 GenerateReducedVikTable[numEmax_Integer:7, OptionsPattern[]]:= (
561     numValues = Total[Length[AllowedNKSLTerms[#]]*Length[
    AllowedNKSLTerms[#]]&/@Range[1, numEmax]];
562     Print["Calculating " <> ToString[numValues] <> " values for Vk1."
    ];
563     counter = 1;
564     If[And[OptionValue["Progress"], frontEndAvailable],
565     progBar = PrintTemporary[
566         Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
    counter}]]]
567     ];
568 ];
569 ReducedVikTable = Table[
570     (
571         counter = counter+1;
572         {n, SL, SpLp, 1} -> SimplifyFun[ReducedVik[n, SL, SpLp, 1]]
573     ),
574     {n, 1, numEmax},
575     {SL, AllowedNKSLTerms[n]},
576     {SpLp, AllowedNKSLTerms[n]}
577 ];
578 ReducedVikTable = Association[ReducedVikTable];
579 If[And[OptionValue["Progress"], frontEndAvailable],
580     NotebookDelete[progBar]
581 ];
582 exportFname = FileNameJoin[{moduleDir, "data", "ReducedVikTable.m"
    }];
583 If[OptionValue["Export"],
584     (
585         Print["Exporting to file " <> ToString[exportFname]];
586         Export[exportFname, ReducedVikTable];
587     )
588 ];
589 Return[ReducedVikTable];
590 )
591
592 (* ##### Racah Algebra
    ##### *)
593 (*
    #####

```



```

*)
594
595 (*
#####
*)
596 (* ##### Electrostatic
##### *)
597
598 fsubk::usage = "Slater integral f_k. See equation 12.17 in TASS.";
599 fsubk[numE_, orbital_, NKSL_, NKSLp_, k_] := Module[
600   {terms, S, L, Sp, Lp, termsWithSameSpin, SL, fsubkVal,
    spinMultiplicity,
601   prefactor, summand1, summand2},
602   {S, L} = FindSL[NKSL];
603   {Sp, Lp} = FindSL[NKSLp];
604   terms = AllowedNKSLTerms[numE];
605   (* sum for summand1 is over terms with same spin *)
606   spinMultiplicity = 2*S + 1;
607   termsWithSameSpin = StringCases[terms, ToString[spinMultiplicity]
~~ _];
608   termsWithSameSpin = Flatten[termsWithSameSpin];
609   If[Not[{S, L} == {Sp, Lp}],
610   Return[0]
611   ];
612   prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
613   summand1 = Sum[(
614     ReducedUkTable[{numE, orbital, SL, NKSL, k}] *
615     ReducedUkTable[{numE, orbital, SL, NKSLp, k}]
616     ),
617     {SL, termsWithSameSpin}
618   ];
619   summand1 = 1 / TPO[L] * summand1;
620   summand2 = (
621     KroneckerDelta[NKSL, NKSLp] *
622     (numE *(4*orbital + 2 - numE)) /
623     ((2*orbital + 1) * (4*orbital + 1))
624   );
625   fsubkVal = prefactor*(summand1 - summand2);
626   Return[fsubkVal];
627 ]
628
629 fsupk::usage = "Super-script Slater integral f^k = Subscript[f, k] *
Subscript[D, k]";
630 fsupk[numE_, orbital_, NKSL_, NKSLp_, k_] := (Dk[k] * fsubk[numE,
orbital, NKSL, NKSLp, k])
631
632 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.";
633 Dk[k_] := {1, 225, 1089, 184041/25}[[k/2+1]]
634
635 FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the corresponding {E0
, E1, E2, E3} values."

```

```

636 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.";
637 FtoE[F0_, F2_, F4_, F6_] := (Module[ (*Necessary here since Ei are
      protected.*)
638   {E0, E1, E2, E3},
639   E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
640   E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
641   E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;
642   E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
643   Return[{E0, E1, E2, E3}];
644 ]
645 );
646
647 EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the corresponding {F0
      , F2, F4, F6} values. The inverse of FtoE.";
648 EtoF[E0_, E1_, E2_, E3_] := (Module[ (*Necessary here since Fi are
      protected.*)
649   {F0, F2, F4, F6},
650   F0 = 1/7 (7 E0 + 9 E1);
651   F2 = 75/14 (E1 + 143 E2 + 11 E3);
652   F4 = 99/7 (E1 - 130 E2 + 4 E3);
653   F6 = 5577/350 (E1 + 35 E2 - 7 E3);
654   Return[{F0, F2, F4, F6}];
655 ]
656 );
657
658 Options[Electrostatic] = {"Coefficients" -> "Slater"};
659 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\".";
660 Electrostatic[{numE_, NKSL_, NKSLp_}, OptionsPattern[]]:= Module[
661   {fsub0, fsub2, fsub4, fsub6, esub0, esub1, esub2, esub3,
662     fsup0, fsup2, fsup4, fsup6,
663     eMatrixVal, orbital},
664   orbital = 3;
665   Which[
666     OptionValue["Coefficients"] == "Slater",
667     (
668       fsub0 = fsubk[numE, orbital, NKSL, NKSLp, 0];
669       fsub2 = fsubk[numE, orbital, NKSL, NKSLp, 2];
670       fsub4 = fsubk[numE, orbital, NKSL, NKSLp, 4];
671       fsub6 = fsubk[numE, orbital, NKSL, NKSLp, 6];
672       eMatrixVal = fsub0*F0 + fsub2*F2 + fsub4*F4 + fsub6*F6;
673     ),
674     OptionValue["Coefficients"] == "Racah",
675     (
676       fsup0 = fsupk[numE, orbital, NKSL, NKSLp, 0];
677       fsup2 = fsupk[numE, orbital, NKSL, NKSLp, 2];
678       fsup4 = fsupk[numE, orbital, NKSL, NKSLp, 4];

```

```

679         fsup6 = fsupk[numE, orbital, NKSL, NKSLp, 6];
680         esub0 = fsup0;
681         esub1 = 9/7*fsup0 + 1/42*fsup2 + 1/77*fsup4 + 1/462*fsup6
;
682         esub2 = 143/42*fsup2 - 130/77*fsup4 + 35/462*fsup6
;
683         esub3 = 11/42*fsup2 + 4/77*fsup4 - 7/462*fsup6
;
684         eMatrixVal = esub0*E0 + esub1*E1 + esub2*E2 + esub3*E3;
685     )
686 ];
687 Return[eMatrixVal];
688 ]
689
690 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.";
691 Options[GenerateElectrostaticTable] = {"Export" -> True, "
Coefficients" -> "Slater"};
692 GenerateElectrostaticTable[numEmax_Integer:7, OptionsPattern[]]:= (
693     ElectrostaticTable = Table[
694         {numE, SL, SpLp} -> SimplifyFun[Electrostatic[{numE, SL, SpLp},
"Coefficients" -> OptionValue["Coefficients"]]],
695         {numE, 1, numEmax},
696         {SL, AllowedNKSLTerms[numE]},
697         {SpLp, AllowedNKSLTerms[numE]}
698     ];
699     ElectrostaticTable = Association[Flatten[ElectrostaticTable]];
700     If[OptionValue["Export"],
701         Export[FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}
]],
702         ElectrostaticTable];
703 ];
704 Return[ElectrostaticTable];
705 )
706
707 (* ##### Electrostatic
##### *)
708 (*
#####
*)
709
710 (*
#####
*)
711 (* ##### Bases
##### *)
712
713 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}.";
714 BasisTableGenerator[numE_] := Module[{energyStatesTable}, (

```

```

715     energyStatesTable = <||>;
716     allowedJ = AllowedJ[numE];
717     Do[
718     (
719         energyStatesTable[{numE, J}] = EnergyStates[numE, J];
720     ),
721     {Jp, allowedJ},
722     {J, allowedJ}];
723     Return[energyStatesTable]
724 )
725 ];
726
727 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}.";
728 BasisLSJMJ[numE_] := Module[{energyStatesTable, basis, idx1},
729 (
730     energyStatesTable = BasisTableGenerator[numE];
731     basis = Table[
732         energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
733         {idx1, 1, Length[AllowedJ[numE]]}];
734     basis = Flatten[basis, 1];
735     Return[basis]
736 )
737 ];
738
739 (* ##### Bases
##### *)
740 (*
#####
*)
741
742 (*
#####
*)
743 (* ##### Coefficients of Fractional Parentage
##### *)
744
745 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.";
746 Options[GenerateCFP] = {"Export" -> True, "PhaseFunction"-> "NK"};
747 GenerateCFP[OptionsPattern[]]:= (
748     CFP = Table[
749         {numE, NKSL} -> First[CFPTerms[numE, NKSL]],
750         {numE, 1, 7},
751         {NKSL, AllowedNKSLTerms[numE]}}];

```

```

752 CFP = Association[CFP];
753 (* Go all the way to f14 *)
754 CFP = CFPExpander["Export" -> False, "PhaseFunction"-> OptionValue
["PhaseFunction"]];
755 If[OptionValue["Export"],
756   Export[FileNameJoin[{moduleDir, "data", "CFPs.m"}], CFP];
757 ];
758 Return[CFP];
759 )
760
761 JuddCFPPPhase::usage="Phase between conjugate coefficients of
fractional parentage according to Velkov's thesis, page 40.";
762 JuddCFPPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
parentSeniority_, daughterSeniority_] := (
763   {spin, orbital} = {1/2, 3};
764   expo = (
765     (parentS + parentL + daughterS + daughterL) -
766     (orbital + spin) +
767     1/2 * (parentSeniority + daughterSeniority - 1)
768   );
769   phase = Phaser[-expo];
770   Return[phase];
771 )
772
773 NKCFPPPhase::usage="Phase between conjugate coefficients of
fractional parentage according to Nielson and Koster page viii.
Note that there is a typo on there the expression for zeta should
be (-1)^((v-1)/2) instead of (-1)^(v - 1/2).";
774 NKCFPPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_,
parentSeniority_, daughterSeniority_] := (
775   {spin, orbital} = {1/2, 3};
776   expo = (
777     (parentS + parentL + daughterS + daughterL) -
778     (orbital + spin)
779   );
780   phase = Phaser[-expo];
781   If[parent == 2*orbital,
782     phase = phase * Phaser[(daughterSeniority-1)/2]];
783   Return[phase];
784 )
785
786 Options[CFPExpander] = {"Export" -> True, "PhaseFunction" -> "NK"};
787 CFPExpander::usage="Using the coefficients of fractional parentage
up to f7 this function calculates them up to f14.
788
789 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.";
790 CFPExpander[OptionsPattern[]]:= (
791   orbital = 3;
792   halfFilled = 2 * orbital + 1;
793   fullShell = 2 * halfFilled;

```

```

794     parentMax = 2 * orbital;
795
796     PhaseFun = <|
797         "Judd" -> JuddCFPPPhase,
798         "NK" -> NKCFPPPhase|>[OptionValue["PhaseFunction"]];
799     PrintTemporary["Calculating CFPs using the phase system from ",
800     PhaseFun];
801     (* Initialize everything with lists to be filled in the next Do
802     *)
803     complementaryCFPs =
804         Table[
805             ({numE, term} -> {term}),
806             {numE, halfFilled + 1, fullShell - 1, 1},
807             {term, AllowedNKSLTerms[numE]}
808         ];
809     complementaryCFPs = Association[Flatten[complementaryCFPs]];
810     Do[(
811         daughter = parent + 1;
812         conjugateDaughter = fullShell - parent;
813         conjugateParent = conjugateDaughter - 1;
814         parentTerms = AllowedNKSLTerms[parent];
815         daughterTerms = AllowedNKSLTerms[daughter];
816         Do[
817             (
818                 parentCFPs = Rest[CFP[{daughter,
819                 daughterTerm}]];
820                 daughterSeniority = Seniority[daughterTerm];
821                 {daughterS, daughterL} = FindSL[daughterTerm];
822                 Do[
823                     (
824                         {parentTerm, parentCFPval} = parentCFP;
825                         {parentS, parentL} = FindSL[parentTerm];
826                         parentSeniority = Seniority[parentTerm];
827                         phase = PhaseFun[parent, parentS, parentL,
828                         daughterS, daughterL,
829                         parentSeniority, daughterSeniority];
830                         prefactor = (daughter * TPO[daughterS, daughterL]) /
831                         (conjugateDaughter * TPO[parentS,
832                         parentL]);
833                         prefactor = Sqrt[prefactor];
834                         newCFPval = phase * prefactor * parentCFPval;
835                         key = {conjugateDaughter, parentTerm};
836                         complementaryCFPs[key] = Append[complementaryCFPs[
837                         key], {daughterTerm, newCFPval}]
838                     ),
839                     {parentCFP, parentCFPs}
840                 ]
841             ),
842             {daughterTerm, daughterTerms}
843         ],
844         {parent, 1, parentMax}
845     ];
846     complementaryCFPs[{14, "1S"}] = {"1S", {"2F", 1}};

```

```

844     extendedCFPs      = Join[CFP, complementaryCFPs];
845     If[OptionValue["Export"],,
846     (
847         exportFname = FileNameJoin[{moduleDir, "data", "
CFPs_extended.m"}]];
848         Print["Exporting to ", exportFname];
849         Export[exportFname, extendedCFPs];
850     )
851 ];
852     Return[extendedCFPs];
853 )
854
855 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";
856 Options[GenerateCFPTable] = {"Export" -> True};
857 GenerateCFPTable[OptionsPattern[]]:= (
858     CFPtextData = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT
"}]];
859     fConfigs = StringSplit[CFPtextData, "[ONE PARTICLE FRACTIONAL
PARENTAGE COEFFICIENTS "];
860     CFPTable = {};
861
862     (* This table parses the text file with the one-body coefficients
of fractional parentage *)
863     CFPTable = Table[
864     (
865         fNx = StringReplace[Part[fConfigs, idx1], "-" -> " -"];
866         daughterLabelSpots = StringPosition[fNx,
Shortest[StartOfLine ~~ DigitCharacter ~~ LetterCharacter ~~
--- ~~ "["],
867         Overlaps -> False];
868         daughterLabels = Map[StringDrop[#, -1] &, StringTake[fNx,
daughterLabelSpots]];
869         daughterLabelLines = StringPosition[fNx,
Shortest[StartOfLine ~~ DigitCharacter ~~ LetterCharacter ~~
-- ~~
870         EndOfLine], Overlaps -> False];
871         startDaughters = Map[Last, daughterLabelLines + 2];
872         stopDaughters = Delete[Append[Map[First, daughterLabelLines -
2], StringLength[fNx]], 1];
873         daughterLines = Join[Partition[startDaughters, 1], Partition[
stopDaughters, 1], 2];
874         testing = Map[StringSplit,
StringSplit[StringTake[fNx, daughterLines], EndOfLine]];
875         testing2 = Map[DeleteCases[#, {}] &, testing];
876         ToIntegerOrString[list_] := Map[If[StringMatchQ[#, NumberString
], ToExpression[#, #] &, list];
877         CFPs = Table[(
878             tt = Part[testing2, mm];
879             pLabels = Map[Extract[#, 1] &, tt];
880             pValues = Map[SquarePrimeToNormal, Map[ToIntegerOrString[Drop
[#, 2]] &, tt]];
881             Join[Partition[pLabels, 1], Partition[pValues, 1], 2]

```

```

885     ),
886     {mm, 1, Length[testing2]}
887 ];
888 CFPconfig = Join[Partition[daughterLabels, 1], CFPs, 2];
889 CFPconfig
890 ),
891 {idx1, 2, 7}
892 ];
893 CFPTable = Join[{{{ "2F", {"1S", 1}}}}, CFPTable];
894 If[OptionValue["Export"],
895   (
896     CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
897     Export[CFPTablefname, CFPTable];
898   )
899 ];
900 Return[CFPTable];
901 )
902
903 GenerateCFPAAssoc::usage = "GenerateCFPAAssoc[export] converts the
904   coefficients of fractional parentage into an association in which
905   zero values are explicit. If \"Export\" is set to True, the
906   association is exported to the file /data/CFPAAssoc.m.";
907 Options[GenerateCFPAAssoc] = {"Export" -> True};
908 GenerateCFPAAssoc[OptionsPattern[]]:= (
909   CFPAAssoc = Association[];
910   Do[
911     (daughterTerms = AllowedNKSLTerms[numE];
912      parentTerms   = AllowedNKSLTerms[numE - 1];
913      Do[
914        (
915          cfps = CFP[{numE, daughter}];
916          cfps = cfps[[2 ;;]];
917          parents = First /@ cfps;
918          Do[
919            (
920              key = {numE, daughter, parent};
921              cfp = If[
922                MemberQ[parents, parent],
923                (
924                  idx = Position[parents, parent][[1, 1]];
925                  cfps[[idx]][[2]]
926                ),
927                0
928              ];
929              CFPAAssoc[key] = cfp;
930            ),
931            {parent, parentTerms}
932          ]
933        ),
934        {daughter, daughterTerms}
935      ]
936    ],
937    {numE, 1, 14}
938  ];
939 If[OptionValue["Export"],

```



```

937     (
938     CFPAssocfname = FileNameJoin[{moduleDir, "data", "CFPAassoc.m"}];
939     Export[CFPAassocfname, CFPAssoc];
940     )
941 ];
942 Return[CFPAassoc];
943 )
944
945 CFPTerms::usage = "CFPTerms[numE] gives all the daughter and parent
946 terms, together with the corresponding coefficients of fractional
947 parentage, that correspond to the the f^n configuration.
948
949 CFPTerms[numE, SL] gives all the daughter and parent terms, together
950 with the corresponding coefficients of fractional parentage, that
951 are compatible with the given string SL in the f^n configuration.
952
953 CFPTerms[numE, L, S] gives all the daughter and parent terms,
954 together with the corresponding coefficients of fractional
955 parentage, that correspond to the given total orbital angular
956 momentum L and total spin S n the f^n configuration. L being an
957 integer, and S being integer or half-integer.
958
959 In all cases the output is in the shape of a list with enclosed
960 lists having the format {daughter_term, {parent_term_1, CFP_1}, {
961 parent_term_2, CFP_2}, ...}.
962
963 Only the one-body coefficients for f-electrons are provided.
964 In all cases it must be that 1 <= n <= 7.
965 ";
966 CFPTerms[numE_] := Part[CFPTable, numE]
967 CFPTerms[numE_, SL_] :=
968   Module[
969     {NKterms, CFPconfig},
970     NKterms = {};
971     CFPconfig = Part[CFPTable, numE];
972     Map[
973       If[StringFreeQ[First[#], SL],
974         Null,
975         NKterms = Join[NKterms, {#}, 1]
976       ] &,
977     CFPconfig
978   ];
979   NKterms = DeleteCases[NKterms, {}]
980 ]
981 CFPTerms[numE_, L_, S_] :=
982   Module[
983     {NKterms, SL, CFPconfig},
984     SL = StringJoin[ToString[2 S + 1], PrintL[L]];
985     NKterms = {};
986     CFPconfig = Part[CFPTable, numE];
987     Map[
988       If[StringFreeQ[First[#], SL],
989         Null,
990         NKterms = Join[NKterms, {#}, 1]
991       ] &,
992     CFPconfig

```

```

982 ];
983 NKterms = DeleteCases[NKterms, {}]
984 ]
985
986 (* ##### Coefficients of Fractional Parentage
987 ##### *)
988 (*
989 #####
990 *)
991 (* ##### Spin Orbit
992 ##### *)
993
994 SpinOrbit::usage = "SpinOrbit[numE, SL, SpLp, J] returns the LSJ
995 reduced matrix element  $\zeta$  <SL, J|L.S|SpLp, J>. These are given as a
996 function of  $\zeta$ . This function requires that the association
997 ReducedVikTable be defined.";
998
999 SpinOrbit[numE_, SL_, SpLp_, J_] := Module[
1000 {S, L, Sp, Lp, orbital, sign, prefact},
1001 orbital = 3;
1002 {S, L} = FindSL[SL];
1003 {Sp, Lp} = FindSL[SpLp];
1004 prefact = Sqrt[orbital*(orbital+1)*(2*orbital+1)] * SixJay[{L, Lp,
1005 1}, {Sp, S, J}];
1006 sign = Phaser[J + L + Sp];
1007 Return[sign * prefact *  $\zeta$  * ReducedVikTable[{numE, SL, SpLp, 1}]];
1008 ]
1009
1010 GenerateSpinOrbitTable::usage = "GenerateSpinOrbitTable[nmax, export
1011 ] computes the matrix values for the spin-orbit interaction for f^
1012 n configurations up to n = nmax. The function returns an
1013 association whose keys are lists of the form {n, SL, SpLp, J}. If
1014 export is set to True, then the result is exported to the data
1015 subfolder for the folder in which this package is in. It requires
1016 ReducedVikTable to be defined.";
1017
1018 GenerateSpinOrbitTable[nmax_:7, export_:False] := (
1019 SpinOrbitTable =
1020 Table[
1021 {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
1022 {numE, 1, nmax},
1023 {J, MinJ[numE], MaxJ[numE]},
1024 {SL, Map[First, AllowedNKSLforJTerms[numE, J]]},
1025 {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
1026 ];
1027 SpinOrbitTable = Association[SpinOrbitTable];
1028
1029 exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
1030 If[export,
1031 (
1032 Print["Exporting to file "<>ToString[exportFname]];
1033 Export[exportFname, SpinOrbitTable];
1034 )
1035 ]

```

```

1020     )
1021 ];
1022 Return[SpinOrbitTable];
1023 )
1024
1025 (* ##### Spin Orbit
1026 ##### *)
1027 (*
1028 #####
1029 *)
1030
1031 (* ##### Three Body Operators
1032 ##### *)
1033
1034 Options[ParseJudd1984] = {"Export" -> False};
1035 ParseJudd1984::usage="This function parses the data from tables 1
1036 and 2 of Judd from Judd, BR, and MA Suskin. \"Complete Set of
1037 Orthogonal Scalar Operators for the Configuration f^3\". JOSA B 1,
1038 no. 2 (1984): 261-65.\"";
1039 ParseJudd1984[OptionsPattern[]]:= (
1040   export = OptionValue["Export"];
1041   ParseJuddTab1[str_] := (
1042     strR = ToString[str];
1043     strR = StringReplace[strR, ".5" -> "^(1/2)"];
1044     num = ToExpression[strR];
1045     sign = Sign[num];
1046     num = sign*Simplify[Sqrt[num^2]];
1047     If[Round[num] == num, num = Round[num]];
1048     Return[num]);
1049
1050 (* Parse table 1 from Judd 1984 *)
1051 judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"
1052 }];
1053 data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
1054 headers = data[[1]];
1055 data = data[[2 ;;]];
1056 data = Transpose[data];
1057 \[Psi] = Select[data[[1]], # != "" &];
1058 \[Psi]p = Select[data[[2]], # != "" &];
1059 matrixKeys = Transpose[{\[Psi], \[Psi]p}];
1060 data = data[[3 ;;]];
1061 cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data
1062 }];
1063 cols = Select[cols, Length[#] == 21 &];
1064 tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
1065 tab1 = Transpose[Prepend[Transpose[tab1], headers]];
1066
1067 (* Parse table 2 from Judd 1984 *)
1068 judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"
1069 }];
1070 data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
1071 headers = data[[1]];

```

```

1063 data = data[[2 ;;]];
1064 data = Transpose[data];
1065 {operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues}
= data[;; 4];
1066 multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
1067 multiFactorValues = AssociationThread[multiFactorSymbols ->
multiFactorValues];
1068
1069 (*scale values of table 1 given the values in table 2*)
1070 oppyS = {};
1071 normalTable =
1072 Table[header = col[[1]];
1073 If[StringContainsQ[header, " "],
1074 (
1075 multiplierSymbol = StringSplit[header, " "][[1]];
1076 multiplierValue = multiFactorValues[multiplierSymbol];
1077 operatorSymbol = StringSplit[header, " "][[2]];
1078 oppyS = Append[oppyS, operatorSymbol];
1079 ),
1080 (
1081 multiplierValue = 1;
1082 operatorSymbol = header;
1083 )
1084 ];
1085 normalValues = 1/multiplierValue*col[[2 ;;]];
1086 Join[{operatorSymbol}, normalValues], {col, tab1[[3 ;;]]}
1087 ];
1088
1089 (*Create an association for the matrix elements in the f^3 config
*)
1090 juddOperators = Association[];
1091 Do[(
1092 col = normalTable[[colIndex]];
1093 opLabel = col[[1]];
1094 opValues = col[[2 ;;]];
1095 opMatrix = AssociationThread[matrixKeys -> opValues];
1096 Do[(
1097 opMatrix[Reverse[mKey]] = opMatrix[mKey]
1098 ),
1099 {mKey, matrixKeys}
1100 ];
1101 juddOperators[{3, opLabel}] = opMatrix),
1102 {colIndex, 1, Length[normalTable]}
1103 ];
1104
1105 (* special case of t2 in f3 *)
1106 (* this is the same as getting the matrix elements from Judd 1966
*)
1107 numE = 3;
1108 e30p = juddOperators[{3, "e_{3}"}];
1109 t2prime = juddOperators[{3, "t_{2}^{'}"}];
1110 prefactor = 1/(70 Sqrt[2]);
1111 t20p = (# -> (t2prime[#] + prefactor*e30p[#])) & /@ Keys[t2prime];
1112 t20p = Association[t20p];
1113 juddOperators[{3, "t_{2}"}] = t20p;

```

```

1114
1115 (*Special case of t11 in f3*)
1116 t11 = juddOperators[{3, "t_{11}"}];
1117 e $\beta$ primeOp = juddOperators[{3, "e_{\\beta}^{'}"}];
1118 t11primeOp = (# -> (t11[#] + Sqrt[3/385] e $\beta$ primeOp[#])) & /@ Keys[
t11];
1119 t11primeOp = Association[t11primeOp];
1120 juddOperators[{3, "t_{11}^{'}"}] = t11primeOp;
1121 If[export,
1122 (
1123 (*export them*)
1124 PrintTemporary["Exporting ..."];
1125 exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.
m"}];
1126 Export[exportFname, juddOperators];
1127 )
1128 ];
1129 Return[juddOperators];
1130 )
1131
1132 Options[GenerateThreeBodyTables] = {"Export" -> False};
1133 GenerateThreeBodyTables::usage="This function generates the matrix
elements for the three body operators using the coefficients of
fractional parentage, including those beyond f^7.";
1134 GenerateThreeBodyTables[nmax_Integer : 14, OptionsPattern[]] := (
1135 tiKeys = {"t_{2}", "t_{2}^{'}", "t_{3}", "t_{4}", "t_{6}", "t_{7}",
1136 "t_{8}", "t_{11}", "t_{11}^{'}", "t_{12}", "t_{14}", "t_{15}",
1137 "t_{16}", "t_{17}", "t_{18}", "t_{19}"};
1138 TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
1139 juddOperators = ParseJudd1984[];
1140 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.";
1141 op3MatrixElement[SL_, SpLp_, opSymbol_] := (
1142 jOP = juddOperators[{3, opSymbol}];
1143 key = {SL, SpLp};
1144 val = If[MemberQ[Keys[jOP], key],
1145 jOP[key],
1146 0];
1147 Return[val];
1148 );
1149 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.";
1150 Options[ti] = {"Fast" -> True};
1151 ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
Module[{nn, S, L, Sp, Lp,
1152 cfpSL, cfpSpLp,
1153 parentSL, parentSpLp, tnk, tnks},
1154 {S, L} = FindSL[SL];
1155 {Sp, Lp} = FindSL[SpLp];

```

```

1157 fast      = OptionValue["Fast"];
1158 numH = 14 - nE;
1159 If[fast && Not[MemberQ[{"t_{2}", "t_{11}"}, tiKey]] && nE > 7,
1160   Return[-tktable[{numH, SL, SpLp, tiKey}]]
1161 ];
1162 If[(S == Sp && L == Lp),
1163   (
1164     cfpSL = CFP[{nE, SL}];
1165     cfpSpLp = CFP[{nE, SpLp}];
1166     tnks = Table[(
1167       parentSL = cfpSL[[nn, 1]];
1168       parentSpLp = cfpSpLp[[mm, 1]];
1169       cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
1170       tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
1171     ),
1172     {nn, 2, Length[cfpSL]},
1173     {mm, 2, Length[cfpSpLp]}
1174   ];
1175     tnk = Total[Flatten[tnks]];
1176   ),
1177   tnk = 0;
1178 ];
1179 Return[ nE / (nE - opOrder) * tnk];];
1180
1181 (*Calculate the matrix elements of  $t^i$  for n up to nmax*)
1182 tktable = <||>;
1183 Do[(
1184   Do[(
1185     tkValue = Which[numE <= 2,
1186       (*Initialize n=1,2 with zeros*)
1187       0,
1188       numE == 3,
1189       (*Grab matrix elem in  $f^3$  from Judd 1984*)
1190       SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
1191       True,
1192       SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2,
1193         3]]]
1194     ];
1195     tktable[{numE, SL, SpLp, opKey}] = tkValue;
1196   ),
1197   {SL, AllowedNKSLTerms[numE]},
1198   {SpLp, AllowedNKSLTerms[numE]},
1199   {opKey, Append[tiKeys, "e_{3}"]}
1200 ];
1201 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], "
configuration complete"]];
1202 {numE, 1, nmax}
1203 ];
1204
1205 (* Now use those matrix elements to determine their sum as weighted
by their corresponding strengths  $T_i$  *)
1206 ThreeBodyTable = <||>;
1207 Do[
1208   Do[

```

```

1209 (
1210   ThreeBodyTable[{numE, SL, SpLp}] = (
1211     Sum[(
1212       If[tiKey == "t_{2}", t2Switch, 1] *
1213       tktable[{numE, SL, SpLp, tiKey}] *
1214       TSymbolsAssoc[tiKey] +
1215       If[tiKey == "t_{2}", 1 - t2Switch, 0] *
1216       (-tktable[{14 - numE, SL, SpLp, tiKey}]) *
1217       TSymbolsAssoc[tiKey]
1218     ),
1219     {tiKey, tiKeys}
1220   ]
1221 );
1222 ),
1223 {SL, AllowedNKSLTerms[numE]},
1224 {SpLp, AllowedNKSLTerms[numE]}
1225 ];
1226 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix
complete"]],,
1227 {numE, 1, 7}
1228 ];
1229
1230 ThreeBodyTables = Table[(
1231   terms = AllowedNKSLTerms[numE];
1232   singleThreeBodyTable =
1233     Table[
1234       {SL, SLp} -> ThreeBodyTable[{numE, SL, SLp}],
1235       {SL, terms},
1236       {SLp, terms}
1237     ];
1238   singleThreeBodyTable = Flatten[singleThreeBodyTable];
1239   singleThreeBodyTables = Table[(
1240     notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
1241     reps = ConstantArray[0, Length[TSymbols]];
1242     reps[[notNullPosition]] = 1;
1243     rep = AssociationThread[TSymbols -> reps];
1244     notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
1245   ),
1246   {notNullSymbol, TSymbols}
1247 ];
1248 singleThreeBodyTables = Association[singleThreeBodyTables];
1249 numE -> singleThreeBodyTables),
1250 {numE, 1, 7}];
1251
1252 ThreeBodyTables = Association[ThreeBodyTables];
1253 If[OptionValue["Export"], (
1254   threeBodyTablefname = FileNameJoin[{moduleDir, "data", "
ThreeBodyTable.m"}];
1255   Export[threeBodyTablefname, ThreeBodyTable];
1256   threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "
ThreeBodyTables.m"}];
1257   Export[threeBodyTablesfname, ThreeBodyTables];
1258 )
1259 ];
1260 Return[{ThreeBodyTable, ThreeBodyTables}];)

```

```

1261
1262 ScalarOperatorProduct::usage="ScalarOperatorProduct[op1, op2, numE]
    calculated the innerproduct between the two scalar operators op1
    and op2.";
1263 ScalarOperatorProduct[op1_, op2_, numE_] := Module[
1264     {terms, S, L, factor, term1, term2},
1265     (
1266     terms = AllowedNKSLTerms[numE];
1267     Simplify[
1268     Sum[(
1269     {S, L} = FindSL[term1];
1270     factor = TPO[S, L];
1271     factor * op1[{term1, term2}] * op2[{term2, term1}]
1272     ),
1273     {term1, terms},
1274     {term2, terms}
1275     ]
1276     ]
1277 )
1278 ];
1279
1280 (* ##### Three Body Operators
    ##### *)
1281 (*
    #####
    *)
1282
1283 (*
    #####
    *)
1284 (* ##### Reduced S00 and ECS0
    ##### *)
1285
1286 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.
1287 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.
1288 ";
1289 ReducedT11inf2[SL_, SpLp_] :=
1290 Module[{T11inf2},
1291 T11inf2 = <|
1292 {"1S", "3P"} -> 6 M0 + 2 M2 + 10/11 M4,
1293 {"3P", "3P"} -> -36 M0 - 72 M2 - 900/11 M4,
1294 {"3P", "1D"} -> -Sqrt[(2/15)] (27 M0 + 14 M2 + 115/11 M4),
1295 {"1D", "3F"} -> Sqrt[2/5] (23 M0 + 6 M2 - 195/11 M4),
1296 {"3F", "3F"} -> 2 Sqrt[14] (-15 M0 - M2 + 10/11 M4),
1297 {"3F", "1G"} -> Sqrt[11] (-6 M0 + 64/33 M2 - 1240/363 M4),
1298 {"1G", "3H"} -> Sqrt[2/5] (39 M0 - 728/33 M2 - 3175/363 M4),
1299 {"3H", "3H"} -> 8/Sqrt[55] (-132 M0 + 23 M2 + 130/11 M4),
1300 {"3H", "1I"} -> Sqrt[26] (-5 M0 - 30/11 M2 - 375/1573 M4)
1301 |>;
1302 Which[

```



```

1303     MemberQ[Keys[T11inf2],{SL,SpLp}],
1304     Return[T11inf2[{SL,SpLp}]],
1305     MemberQ[Keys[T11inf2],{SpLp,SL}],
1306     Return[T11inf2[{SpLp,SL}]],
1307     True,
1308     Return[0]
1309 ]
1310 ];
1311
1312 T11n::usage="T11n[n, SL, SpLp] calculate the reduced matrix element
of the T11 operator for the f^n configuration corresponding to the
terms SL and SpLp. It is essentially the same as T22n with a
different value of t. This operator corresponds to the inter-
electron interaction between the spin of one electron and the
orbital angular momentum of another.
1313
1314 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.\"
1315 ";
1316 T11n[numE_, SL_, SpLp_] := Module[
1317 {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp,
parentSL, parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
1318 {spin, orbital} = {1/2, 3};
1319 {S, L} = FindSL[SL];
1320 {Sp, Lp} = FindSL[SpLp];
1321 t = 1;
1322 cfpSL = CFP[{numE, SL}];
1323 cfpSpLp = CFP[{numE, SpLp}];
1324 Tnkk =
1325 Sum[(
1326     parentSL = cfpSL[[idx2, 1]];
1327     parentSpLp = cfpSpLp[[idx1, 1]];
1328     {Sb, Lb} = FindSL[parentSL];
1329     {Sbp, Lbp} = FindSL[parentSpLp];
1330     phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1331     (
1332         phase *
1333         cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1334         SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1335         SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1336         T11Table[{numE - 1, parentSL, parentSpLp}]
1337     )
1338 ),
{idx1, 2, Length[cfpSpLp]},
{idx2, 2, Length[cfpSL]}
];
1339 Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1340 Return[Tnkk];
1341 ];
1342
1343 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}."

```

```

1347 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.\"
1348 \"
1349 Reducedt11inf2[SL_, SpLp_] := Module[
1350   {t11inf2},
1351   t11inf2 = <|
1352     {\"1S\", \"3P\"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
1353     {\"3P\", \"3P\"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
1354     {\"3P\", \"1D\"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
1355     {\"1D\", \"3F\"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
1356     {\"3F\", \"3F\"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
1357     {\"3F\", \"1G\"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
1358     {\"1G\", \"3H\"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
1359     {\"3H\", \"3H\"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
1360     {\"3H\", \"1I\"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
1361   |>;
1362   Which[
1363     MemberQ[Keys[t11inf2], {SL, SpLp}],
1364     Return[t11inf2[{SL, SpLp}]],
1365     MemberQ[Keys[t11inf2], {SpLp, SL}],
1366     Return[t11inf2[{SpLp, SL}]],
1367     True,
1368     Return[0]
1369   ]
1370 ]
1371
1372 ReducedS00andECS0inf2::usage = \"ReducedS00andECS0inf2[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
      ECS0 interaction.
1373
1374 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\".
1375
1376 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.\";
1377 ReducedS00andECS0inf2[SL_, SpLp_] :=
1378 Module[{pairPosition, f2TermPairs, a13, z13, redS00andECS0inf2},
1379   f2TermPairs = {
1380     {\"1S\", \"3P\"}, {\"3P\", \"1S\"},
1381     {\"3P\", \"3P\"}, {\"3P\", \"1D\"},
1382     {\"1D\", \"3P\"}, {\"1D\", \"3F\"},
1383     {\"3F\", \"1D\"}, {\"3F\", \"3F\"},

```

```

1384      {"3F", "1G"}, {"1G", "3F"},
1385      {"1G", "3H"}, {"3H", "1G"},
1386      {"3H", "3H"}, {"3H", "1I"},
1387      {"1I", "3H"}};
1388      a13 = (-33 M0 + 3 M2 + 15/11 M4 -
1389            6 P0 + 3/2 (35 P2 + 77 P4 + 143 P6));
1390      z13 = {2, 2,
1391            1,
1392            1/Sqrt[1080] (-90),
1393            1/Sqrt[1080] (-90),
1394            Sqrt[2/405] 45,
1395            Sqrt[2/405] 45,
1396            Sqrt[14],
1397            1/Sqrt[891] (-99),
1398            1/Sqrt[891] (-99),
1399            990/Sqrt[98010],
1400            990/Sqrt[98010],
1401            55/Sqrt[55],
1402            -2574/Sqrt[1019304],
1403            -2574/Sqrt[1019304]};
1404      pairPosition = Position[f2TermPairs, {SL, SpLp}];
1405      If[Length[pairPosition] == 0,
1406        Return[0],
1407        pairPosition = pairPosition[[1, 1]]
1408      ];
1409
1410      redS00andECS0inf2 = (
1411        ReducedT11inf2[SL, SpLp] +
1412        Reducedt11inf2[SL, SpLp] -
1413        a13 / 6 * z13[[pairPosition]]
1414      );
1415      redS00andECS0inf2 = SimplifyFun[redS00andECS0inf2];
1416      Return[redS00andECS0inf2];
1417    ];
1418
1419    ReducedS00andECS0infn::usage="ReducedS00andECS0infn[numE, SL, SpLp]
calculates the reduced matrix elements of the (spin-other-orbit +
ECS0) 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.
";
1420
1421    ReducedS00andECS0infn[numE_, SL_, SpLp_]:= Module[
1422      {spin, orbital, t, S, L, Sp, Lp, idx1, idx2, cfpSL, cfpSpLp,
parentSL, Sb, Lb, Sbp, Lbp, parentSpLp, funval},
1423      {spin, orbital} = {1/2, 3};
1424      {S, L} = FindSL[SL];
1425      {Sp, Lp} = FindSL[SpLp];
1426      t = 1;
1427      cfpSL = CFP[{numE, SL}];
1428      cfpSpLp = CFP[{numE, SpLp}];
1429      funval =
1430      Sum[

```

```

1431     (
1432         parentSL = cfpSL[[idx2, 1]];
1433         parentSpLp = cfpSpLp[[idx1, 1]];
1434         {Sb, Lb} = FindSL[parentSL];
1435         {Sbp, Lbp} = FindSL[parentSpLp];
1436         phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1437         (
1438             phase *
1439             cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1440             SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1441             SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1442             S00andECSOLSTable[{numE - 1, parentSL, parentSpLp}]
1443         )
1444     ),
1445     {idx1, 2, Length[cfpSpLp]},
1446     {idx2, 2, Length[cfpSL]}
1447 ];
1448 funval *= numE / (numE - 2) * Sqrt[TP0[S, Sp, L, Lp]];
1449 Return[funval];
1450 ];
1451
1452 GenerateS00andECSOLSTable::usage="GenerateS00andECSOLSTable[nmax]
generates the LS reduced matrix elements of the spin-other-orbit +
ECSO for the f^n configurations up to n=nmax. The values for n=1
and n=2 are taken from \"Judd, BR, HM Crosswhite, and Hannah
Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\"
Physical Review 169, no. 1 (1968): 130.\", and the values for n
>2 are calculated recursively using equation (4) of that same
paper. The values are then exported to a file \"
ReducedS00andECSOLSTable.m\" in the data folder of this module.
The values are also returned as an association.";
1453 Options[GenerateS00andECSOLSTable] = {"Progress" -> True, "Export"
-> True};
1454 GenerateS00andECSOLSTable[nmax_Integer, OptionsPattern[]]:= (
1455     If[And[OptionValue["Progress"], frontEndAvailable],
1456         (
1457             numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
numE]]^2, {numE, 1, nmax}]];
1458             counters = Association[Table[numE->0, {numE, 1, nmax}]];
1459             totalIters = Total[Values[numItersai][[1;;nmax]]];
1460             template1 = StringTemplate["Iteration 'numiter' of 'totaliter
'"];
1461             template2 = StringTemplate["'remtime' min remaining"];
1462             template3 = StringTemplate["Iteration speed = 'speed' ms/it"];
1463             template4 = StringTemplate["Time elapsed = 'runtime' min"];
1464             progBar = PrintTemporary[
1465                 Dynamic[
1466                     Pane[
1467                         Grid[{
1468                             {Superscript["f", numE]},
1469                             {template1[<|"numiter"->numiter, "totaliter"->
totalIters|>]}],
1470                             {template4[<|"runtime"->Round[QuantityMagnitude[
UnitConvert[(Now-startTime), "min"]], 0.1]|>]}],

```

```

1470         {template2[<|"remtime"->Round[QuantityMagnitude[
UnitConvert[(Now-startTime)/(numiter)*(totalIters-numiter), "min"
]], 0.1]]>]],
1471         {template3[<|"speed"->Round[QuantityMagnitude[Now-
startTime, "ms"]/(numiter), 0.01]]>]], {ProgressIndicator[Dynamic[
numiter], {1, totalIters}]]}
1472     },
1473     Frame->All
1474 ],
1475 Full,
1476 Alignment->Center
1477 ]
1478 ]
1479 ];
1480 )
1481 ];
1482 S00andECSOLSTable = <||>;
1483 numiter = 1;
1484 startTime = Now;
1485 Do[
1486 (
1487     numiter+= 1;
1488     S00andECSOLSTable[{numE, SL, SpLp}] = Which[
1489         numE==1,
1490         0,
1491         numE==2,
1492         SimplifyFun[ReducedS00andECS0inf2[SL, SpLp]],
1493         True,
1494         SimplifyFun[ReducedS00andECS0infn[numE, SL, SpLp]]
1495     ];
1496 ),
1497 {numE, 1, nmax},
1498 {SL, AllowedNKSLTerms[numE]},
1499 {SpLp, AllowedNKSLTerms[numE]}
1500 ];
1501 If[And[OptionValue["Progress"], frontEndAvailable],
1502     NotebookDelete[progBar]];
1503 If[OptionValue["Export"],
1504     (fname = FileNameJoin[{moduleDir, "data", "
ReducedS00andECSOLSTable.m"}]];
1505     Export[fname, S00andECSOLSTable];
1506 )
1507 ];
1508 Return[S00andECSOLSTable];
1509 );
1510
1511 (* ##### Reduced S00 and ECS0
##### *)
1512 (*
#####
*)
1513
1514 (*
#####
*)

```

```

1515 (* ##### Spin-Spin
1516 ##### *)
1517 ReducedT22inf2::usage="ReducedT22inf2[SL, SpLp] returns the reduced
1518 matrix element of the scalar component of the double tensor T22
1519 for the terms SL, SpLp in f^2.
1520 Data used here for m0, m2, m4 is from Table I of Judd, BR, HM
1521 Crosswhite, and Hannah Crosswhite. Intra-Atomic Magnetic
1522 Interactions for f Electrons. Physical Review 169, no. 1 (1968):
1523 130.
1524 ";
1525 ReducedT22inf2[SL_, SpLp_] :=
1526 Module[{statePosition, PsiPspStates, m0, m2, m4, Tkk2m},
1527 T22inf2 = <|
1528 {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
1529 {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),
1530 {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
1531 {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
1532 {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
1533 |>;
1534 Which[
1535 MemberQ[Keys[T22inf2],{SL,SpLp}],
1536 Return[T22inf2[{SL,SpLp}]],
1537 MemberQ[Keys[T22inf2],{SpLp,SL}],
1538 Return[T22inf2[{SpLp,SL}]],
1539 True,
1540 Return[0]
1541 ];
1542 ];
1543
1544 T22n::usage="T22n[n, SL, SpLp] calculates the reduced matrix element
1545 of the T22 operator for the f^n configuration corresponding to
1546 the terms SL and SpLp. This is the operator corresponding to the
1547 inter-electron between spin.
1548 It does this by using equation (4) of \"Judd, BR, HM Crosswhite, and
1549 Hannah Crosswhite. \"Intra-Atomic Magnetic Interactions for f
1550 Electrons.\" Physical Review 169, no. 1 (1968): 130.\"
1551 ";
1552 T22n[numE_, SL_, SpLp_] := Module[
1553 {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp,
1554 parentSL, parentSpLp, Sb, Lb, Tnkk, phase, Sbp, Lbp},
1555 {spin, orbital} = {1/2, 3};
1556 {S, L} = FindSL[SL];
1557 {Sp, Lp} = FindSL[SpLp];
1558 t = 2;
1559 cfpSL = CFP[{numE, SL}];
1560 cfpSpLp = CFP[{numE, SpLp}];
1561 Tnkk =
1562 Sum[(
1563 parentSL = cfpSL[[idx2, 1]];
1564 parentSpLp = cfpSpLp[[idx1, 1]];
1565 {Sb, Lb} = FindSL[parentSL];
1566 {Sbp, Lbp} = FindSL[parentSpLp];
1567 phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1568 (

```

```

1558         phase *
1559         cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1560         SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1561         SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1562         T22Table[{numE - 1, parentSL, parentSpLp}]
1563     )
1564 },
1565 {idx1, 2, Length[cfpSpLp]},
1566 {idx2, 2, Length[cfpSL]}
1567 ];
1568 Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1569 Return[Tnkk];
1570 ];
1571
1572 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.
1573 This is an intermediate step to the calculation of the reduced
matrix elements of the spin-spin operator.";
1574 Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
1575 GenerateT22Table[nmax_Integer, OptionsPattern[]]:= (
1576     If[And[OptionValue["Progress"], frontEndAvailable],
1577     (
1578         numItersai = Association[Table[numE->Length[AllowedNKSLTerms[
numE]]^2, {numE, 1, nmax}]];
1579         counters = Association[Table[numE->0, {numE, 1, nmax}]];
1580         totalIters = Total[Values[numItersai[[1;;nmax]]]];
1581         template1 = StringTemplate["Iteration 'numiter' of 'totaliter'
"];
1582         template2 = StringTemplate["'remtime' min remaining"];
1583         template3 = StringTemplate["Iteration speed = 'speed' ms/it"];
1584         template4 = StringTemplate["Time elapsed = 'runtime' min"];
1585         progBar = PrintTemporary[
1586             Dynamic[
1587                 Pane[
1588                     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,

```

```

1595             Alignment->Center]
1596         ]
1597     ];
1598 )
1599 ];
1600 T22Table = <||>;
1601 startTime = Now;
1602 numiter = 1;
1603 Do[
1604     (
1605         numiter+= 1;
1606         T22Table[{numE, SL, SpLp}] = Which[
1607             numE==1,
1608             0,
1609             numE==2,
1610             SimplifyFun[ReducedT22inf2[SL, SpLp]],
1611             True,
1612             SimplifyFun[T22n[numE, SL, SpLp]]
1613         ];
1614     ),
1615     {numE, 1, nmax},
1616     {SL, AllowedNKSLTerms[numE]},
1617     {SpLp, AllowedNKSLTerms[numE]}
1618 ];
1619 If[And[OptionValue["Progress"],frontEndAvailable],
1620     NotebookDelete[progBar]
1621 ];
1622 If[OptionValue["Export"],
1623     (
1624         fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
1625         Export[fname, T22Table];
1626     )
1627 ];
1628 Return[T22Table];
1629 );
1630
1631 SpinSpin::usage="SpinSpin[n, SL, SpLp, J] returns the matrix element
1632     <|SL,J|spin-spin|SpLp,J> for the spin-spin operator within the
1633     configuration f^n. This matrix element is independent of MJ. This
1634     is obtained by querying the relevant reduced matrix element by
1635     querying the association T22Table and putting in the adequate
1636     phase and 6-j symbol.
1637
1638 This is calculated according to equation (3) in \"Judd, BR, HM
1639 Crosswhite, and Hannah Crosswhite. \"Intra-Atomic Magnetic
1640 Interactions for f Electrons.\" Physical Review 169, no. 1 (1968):
1641 130.\"
1642
1643 \".
1644 ";
1645 SpinSpin[numE_, SL_, SpLp_, J_] := Module[
1646     {S, L, Sp, Lp,  $\alpha$ , val},
1647      $\alpha$  = 2;
1648     {S, L} = FindSL[SL];
1649     {Sp, Lp} = FindSL[SpLp];

```



```

1641     val = (
1642         Phaser[Sp + L + J] *
1643         SixJay[{Sp, Lp, J}, {L, S,  $\alpha$ }] *
1644         T22Table[{numE, SL, SpLp}]
1645     );
1646     Return[val]
1647 ];
1648
1649 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.";
1650 Options[GenerateSpinSpinTable] = {"Export"->False};
1651 GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
1652 (
1653     SpinSpinTable = <||>;
1654     PrintTemporary[Dynamic[numE]];
1655     Do[
1656         SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp, J
1657         ]),
1658         {numE, 1, nmax},
1659         {J, MinJ[numE], MaxJ[numE]},
1660         {SL, First /@ AllowedNKSLforJTerms[numE, J]},
1661         {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
1662     ];
1663     If[OptionValue["Export"],
1664         (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
1665          Export[fname, SpinSpinTable];
1666         )
1667     ];
1668     Return[SpinSpinTable];
1669 );
1670
1671 (* #####
1672 *)
1673 (* ##### Spin-Spin
1674 ##### *)
1675
1676 (* #####
1677 *)
1678 (* ##### Spin-Other-Orbit and Electrostatically-Correlated-Spin-
1679 Orbit ##### *)
1680
1681 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
S00andECSOLSTable and putting in the adequate phase and 6-j symbol
. The S00andECSOLSTable puts together the reduced matrix elements
from three operators.
1677
1678 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.\".
1679
1680 S00andECSO[numE_, SL_, SpLp_, J_] := Module[
1681 {S, Sp, L, Lp,  $\alpha$ , val},
1682  $\alpha$  = 1;
1683 {S, L} = FindSL[SL];
1684 {Sp, Lp} = FindSL[SpLp];
1685 val = (
1686 Phaser[Sp + L + J] *
1687 SixJay[{Sp, Lp, J}, {L, S,  $\alpha$ }] *
1688 S00andECSOLSTable[{numE, SL, SpLp}]
1689 );
1690 Return[val];
1691 ]
1692
1693 Prescaling = {P2 -> P2/225, P4 -> P4/1089, P6 -> 25 * P6 / 184041};
1694 GenerateS00andECSOTable::usage=\"GenerateS00andECSOTable[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.\";
1695 Options[GenerateS00andECSOTable] = {\"Export\"->False}
1696 GenerateS00andECSOTable[nmax_, OptionsPattern[]]:= (
1697 S00andECSOTable = <||>;
1698 Do[
1699 S00andECSOTable[{numE, SL, SpLp, J}] = (S00andECSO[numE, SL,
SpLp, J] /. Prescaling);,
1700 {numE, 1, nmax},
1701 {J, MinJ[numE], MaxJ[numE]},
1702 {SL, First /@ AllowedNKSLforJTerms[numE, J]},
1703 {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
1704 ];
1705 If[OptionValue[\"Export\"],
1706 (
1707 fname = FileNameJoin[{moduleDir, \"data\", \"S00andECSOTable.m\"}];
1708 Export[fname, S00andECSOTable];
1709 )
1710 ];
1711 Return[S00andECSOTable];
1712 );
1713
1714 (* ##### Spin-Other-Orbit and Electrostatically-Correlated-Spin-
Orbit ##### *)

```

```

1715 (* #####
1716 *)
1717 (* #####
1718 *)
1719 (* ##### Magnetic Interactions
1720 ##### *)
1721
1722 MagneticInteractions::usage="MagneticInteractions[{numE, SLJ, SLJp,
1723 J}] returns the matrix element of the magnetic interaction between
1724 the terms SLJ and SLJp in the f^n configuration. The interaction
1725 is given by the sum of the spin-spin interaction and the S00 and
1726 ECS0 interactions. The spin-spin interaction is given by the
1727 function SpinSpin[{numE, SLJ, SLJp, J}]. The S00 and ECS0
1728 interactions are given by the function S00andECS0[{numE, SLJ, SLJp
1729 , J}]. The function requires chenDeltas to be loaded into the
1730 session. The option \"ChenDeltas\" can be use to include or
1731 exclude the Chen deltas from the calculation. The default is to
1732 exclude them.";
1733
1734 Options[MagneticInteractions] = {"ChenDeltas" -> False};
1735 MagneticInteractions[{numE_, SLJ_, SLJp_, J_}, OptionsPattern[]] :=
1736 (
1737     key = {numE, SLJ, SLJp, J};
1738     ss = \[Sigma]SS * SpinSpinTable[key];
1739     sooandecso = S00andECS0Table[key];
1740     total = ss + sooandecso;
1741     total = SimplifyFun[total];
1742     If[
1743         Not[OptionValue["ChenDeltas"]],
1744         Return[total]
1745     ];
1746     (* In the type A errors the wrong values are different *)
1747     If[MemberQ[Keys[chenDeltas["A"]], {numE, SLJ, SLJp}],
1748         (
1749             {S, L} = FindSL[SLJ];
1750             {Sp, Lp} = FindSL[SLJp];
1751             phase = Phaser[Sp + L + J];
1752             Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
1753             Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
1754             {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ,
1755 SLJp}][\"wrong\"];
1756             total = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1757             total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1758             total = total /. Prescaling;
1759             total = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
1760         )
1761     ];
1762     (* In the type B errors the wrong values are zeros all around *)
1763     If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
1764         (
1765             {S, L} = FindSL[SLJ];
1766             {Sp, Lp} = FindSL[SLJp];
1767             phase = Phaser[Sp + L + J];

```

```

1754         Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
1755         Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
1756         {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
1757         total = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1758         total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1759         total = total /. Prescaling;
1760         total = wChErrB * total + (1 - wChErrB) * (ss + sooandecso)
1761     )
1762 ];
1763 Return[total];
1764 )
1765
1766 (* ##### Magnetic Interactions
##### *)
1767 (*
#####
*)
1768
1769 (*
#####
*)
1770 (* ##### Crystal Field
##### *)
1771
1772 Cqk::usage = "Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp].";
1773 Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
1774     {S, Sp, L, Lp, orbital, val},
1775     orbital = 3;
1776     {S, L} = FindSL[NKSL];
1777     {Sp, Lp} = FindSL[NKSLp];
1778     f1 = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
1779     val =
1780     If[f1==0,
1781         0,
1782         (
1783             f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
1784             If[f2==0,
1785                 0,
1786                 (
1787                     f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
1788                     If[f3==0,
1789                         0,
1790                         (
1791                             (
1792                                 Phaser[J - M + S + Lp + J + k] *
1793                                 Sqrt[TP0[J, Jp]] *
1794                                 f1 *
1795                                 f2 *
1796                                 f3 *
1797                                 Ck[orbital, k]
1798                             )
1799                         )
1800                     ]
1801                 )
1802             ]
1803         )
1804     ]

```

```

1803     )
1804   ];
1805   val
1806 ]
1807
1808 Bqk[q_, 2] := {B02/2, B12, B22}[[q + 1]];
1809 Bqk[q_, 4] := {B04/2, B14, B24, B34, B44}[[q + 1]];
1810 Bqk[q_, 6] := {B06/2, B16, B26, B36, B46, B56, B66}[[q + 1]];
1811
1812 Sqk[q_, 2] := {Sm22, Sm12, S02, S12, S22}[[q + 3]];
1813 Sqk[q_, 4] := {Sm44, Sm34, Sm24, Sm14, S04, S14, S24, S34, S44}[[q
+ 5]];
1814 Sqk[q_, 6] := {Sm66, Sm56, Sm46, Sm36, Sm26, Sm16, S06, S16, S26,
S36, S46, S56, S66}[[q + 7]];
1815
1816 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.
1817
1818 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.";
1819 CrystalField[numE_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := (
1820   Sum[
1821     (
1822       cqk = Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp];
1823       cmqk = Cqk[numE, -q, k, NKSL, J, M, NKSLp, Jp, Mp];
1824       Bqk[q, k] * (cqk + (-1)^q * cmqk) +
1825       I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
1826     ),
1827     {k, {2, 4, 6}},
1828     {q, 0, k}
1829   ]
1830 )
1831
1832 TotalCFIters::usage = "TotalIters[i, j] returns total number of
function evaluations for calculating all the matrix elements for
the  $f^n$  configurations to the  $f^n$  configurations.";
1833 TotalCFIters[i_, j_] := (
1834   numIters = {196, 8281, 132496, 1002001, 4008004, 9018009,
11778624};
1835   Return[Total[numIters[[i ;; j]]]];
1836 )
1837
1838 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.";
1839 Options[GenerateCrystalFieldTable] = {"Export" -> False, "Progress"
-> True, "Compress" -> True}
1840 GenerateCrystalFieldTable[numEs_List:{1,2,3,4,5,6,7}, OptionsPattern
[]]:= (
1841   ExportFun =
1842   If[OptionValue["Compress"],
1843     ExportMZip,
1844     Export
1845   ];
1846   numiter = 1;
1847   template1 = StringTemplate["Iteration 'numiter' of 'totaliter'"];
1848   template2 = StringTemplate["'remtime' min remaining"];
1849   template3 = StringTemplate["Iteration speed = 'speed' ms/it"];
1850   template4 = StringTemplate["Time elapsed = 'runtime' min"];
1851   totalIter = Total[TotalCFilters[#, #] & /@ numEs];
1852   freebies = 0;
1853   startTime = Now;
1854   If[And[OptionValue["Progress"], frontEndAvailable],
1855     progBar = PrintTemporary[
1856       Dynamic[
1857         Pane[
1858           Grid[
1859             {
1860               {Superscript["f", numE]},
1861               {template1[<|"numiter" -> numiter, "totaliter" ->
totalIter|>]},
1862               {template4[<|"runtime" -> Round[QuantityMagnitude[
UnitConvert[(Now - startTime), "min"]], 0.1]|>]},
1863               {template2[<|"remtime" -> Round[QuantityMagnitude[
UnitConvert[(Now - startTime)/(numiter - freebies) * (totalIter -
numiter), "min"]], 0.1]|>]},
1864               {template3[<|"speed" -> Round[QuantityMagnitude[Now -
startTime, "ms"]/(numiter-freebies), 0.01]|>]},
               {ProgressIndicator[Dynamic[numiter], {1, totalIter}]}
1865             },
1866             Frame -> All
1867           ],
1868           Full,
1869           Alignment -> Center
1870         ]
1871       ]
1872     ];
1873   ];
1874   Do[
1875     (
1876       exportFname = FileNameJoin[{moduleDir, "data", "
CrystalFieldTable_f"<>ToString[numE]<>".m"}];
1877       If[FileExistsQ[exportFname],
1878         Print["File exists, skipping ..."];
1879         numiter+= TotalCFilters[numE, numE];
1880         freebies+= TotalCFilters[numE, numE];
1881         Continue[];
1882       ]
1883     );

```

```

1884     CrystalFieldTable = <||>;
1885     Do[
1886     (
1887         numiter+= 1;
1888         CrystalFieldTable[{numE, NKSL, J, M, NKSLp, Jp, Mp}] =
CrystalField[numE, NKSL, J, M, NKSLp, Jp, Mp];
1889     ),
1890     {J, MinJ[numE], MaxJ[numE]},
1891     {Jp, MinJ[numE], MaxJ[numE]},
1892     {M, AllowedMforJ[J]},
1893     {Mp, AllowedMforJ[Jp]},
1894     {NKSL, First /@ AllowedNKSLforJTerms[numE, J]},
1895     {NKSLp, First /@ AllowedNKSLforJTerms[numE, Jp]}
1896     ];
1897     If[And[OptionValue["Progress"], frontEndAvailable],
1898         NotebookDelete[progBar]
1899     ];
1900     If[OptionValue["Export"],
1901     (
1902         Print["Exporting to file "<>ToString[exportFname]];
1903         ExportFun[exportFname, CrystalFieldTable];
1904     )
1905     ];
1906     ),
1907     {numE, numEs}
1908     ]
1909 )
1910
1911 (* ##### Crystal Field
##### *)
1912 (*
#####
*)
1913
1914 (*
#####
*)
1915 (* ##### Configuration-Interaction via Casimir Operators
##### *)
1916
1917 CasimirS03::usage = "CasimirS03[SL, SpLp] returns LS reduced matrix
element of the configuration interaction term corresponding to the
Casimir operator of R3.";
1918 CasimirS03[{SL_, SpLp_}] := (
1919     {S, L} = FindSL[SL];
1920     If[SL == SpLp,
1921          $\alpha$  * L * (L + 1),
1922         0
1923     ]
1924 )
1925
1926 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."

```

```

1927 Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\",
1928         table 2-6.";
1929 GG2U = Association[{
1930     "00" -> 0,
1931     "10" -> 6/12,
1932     "11" -> 12/12,
1933     "20" -> 14/12,
1934     "21" -> 21/12,
1935     "22" -> 30/12,
1936     "30" -> 24/12,
1937     "31" -> 32/12,
1938     "40" -> 36/12}
1939 ];
1940
1941 CasimirG2::usage = "CasimirG2[SL, SpLp] returns LS reduced matrix
1942         element of the configuration interaction term corresponding to the
1943         Casimir operator of G2.";
1944 CasimirG2[{SL_, SpLp_}] := (
1945     Ulabel = FindNKLSTerm[SL][[1]][[4]];
1946     If[SL==SpLp,
1947          $\beta$  * GG2U[Ulabel],
1948         0
1949     ]
1950 )
1951
1952 GS07W::usage = "GS07W is an association whose keys are labels for
1953         the irreducible representations of group R7 and whose values are
1954         the eigenvalues of the corresponding Casimir operator.
1955 Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\",
1956         table 2-7.";
1957 GS07W := Association[
1958     {
1959         "000" -> 0,
1960         "100" -> 3/5,
1961         "110" -> 5/5,
1962         "111" -> 6/5,
1963         "200" -> 7/5,
1964         "210" -> 9/5,
1965         "211" -> 10/5,
1966         "220" -> 12/5,
1967         "221" -> 13/5,
1968         "222" -> 15/5
1969     }
1970 ];
1971
1972 CasimirS07::usage = "CasimirS07[SL, SpLp] returns the LS reduced
1973         matrix element of the configuration interaction term corresponding
1974         to the Casimir operator of R7.";
1975 CasimirS07[{SL_, SpLp_}] := (
1976     Wlabel = FindNKLSTerm[SL][[1]][[3]];
1977     If[SL==SpLp,
1978          $\gamma$  * GS07W[Wlabel],
1979         0
1980     ]
1981 )

```



```

1974 ElectrostaticConfigInteraction::usage = "
1975 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.";
1976 ElectrostaticConfigInteraction[{SL_, SpLp_}] := Module[
1977   {S, L, val},
1978   {S, L} = FindSL[SL];
1979   val = (
1980     If[SL == SpLp,
1981       CasimirS03[{SL, SL}] +
1982       CasimirS07[{SL, SL}] +
1983       CasimirG2[{SL, SL}],
1984       0
1985     ]
1986   );
1987   ElectrostaticConfigInteraction[{S, L}] = val;
1988   Return[val];
1989 ]
1990
1991 (* ##### Configuration-Interaction via Casimir Operators
      ##### *)
1992 (*
      #####
      *)
1993
1994 (*
      #####
      *)
1995 (* ##### Block assembly
      ##### *)
1996
1997 Options[JJBlockMatrix] = {"Sparse"->True, "ChenDeltas"->False};
1998 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.";
1999 JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]]:= Module[
2000   {NKSLJMs, NKSLJMps, NKSLJMJ, NKSLJMp,
2001     SLterm, SpLpterm,
2002     MJ, MJp,
2003     subKron, matValue, eMatrix},
2004   (
2005     NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
2006     NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
2007     eMatrix =
2008       Table[
2009         (*Condition for a scalar matrix op*)
2010         SLterm = NKSLJMJ[[1]];
2011         SpLpterm = NKSLJMps[[1]];
2012         MJ = NKSLJMJ[[3]];

```

```

2013     MJp      = NKSLJMp[[3]];
2014     subKron  =
2015     (
2016         KroneckerDelta[J, Jp] *
2017         KroneckerDelta[MJ, MJp]
2018     );
2019     matValue =
2020     If[subKron==0,
2021         0,
2022         (
2023             ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2024             ElectrostaticConfigInteraction[{SLterm, SpLpterm}] +
2025             SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2026             MagneticInteractions[{numE, SLterm, SpLpterm, J}, "
ChenDeltas" -> OptionValue["ChenDeltas"]] +
2027             ThreeBodyTable[{numE, SLterm, SpLpterm}]
2028         )
2029     ];
2030     matValue += CTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp
2031     }];
2032     matValue,
2033     {NKSLJMp, NKSLJMps},
2034     {NKSLJM , NKSLJMs}
2035 ];
2036     If[OptionValue["Sparse"],
2037         eMatrix = SparseArray[eMatrix]
2038     ];
2039     Return[eMatrix]
2040 );
2041 ];
2042 EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some
point may be used to redefine states used in basis.";
2043 EnergyStates[numE_, J_] := AllowedNKSLJMforJTerms[numE, J];
2044
2045 JJBBlockMatrixFileName::usage = "JJBBlockMatrixFileName[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.";
2046 Options[JJBBlockMatrixFileName] = {"FilenameAppendix" -> ""}
2047 JJBBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
2048     fileApp = OptionValue["FilenameAppendix"];
2049     fname = FileNameJoin[{moduleDir,
2050         "hams",
2051         StringJoin[{"f", ToString[numE], "_JJBBlockMatrixTable",
fileApp, ".m"}]}}];
2052     Return[fname];
2053 );
2054
2055 Options[TabulateJJBBlockMatrixTable] = {"Sparse" -> True, "ChenDeltas"
-> False};
2056 TabulateJJBBlockMatrixTable::usage = "TabulateJJBBlockMatrixTable[numE
, I] returns a list with three elements {JJBBlockMatrixTable,
EnergyStatesTable, AllowedM}. JJBBlockMatrixTable is an association
with keys equal to lists of the form {numE, J, Jp}."

```

```

2057 EnergyStatesTable is an association with keys equal to lists of
2058 the form {numE, J}. AllowedM is another association with keys
2059 equal to lists of the form {numE, J} and values equal to lists
2060 equal to the corresponding values of MJ. It's unnecessary (and it
2061 won't work in this implementation) to give numE > 7 given the
2062 equivalency between electron and hole configurations.";
2063
2064 TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]]:= (
2065     JJBlockMatrixTable = <||>;
2066     totalIterations = Length[AllowedJ[numE]]^2;
2067     template1 = StringTemplate["Iteration 'numiter' of 'totaliter'"];
2068     template2 = StringTemplate["'remtime' min remaining"];
2069     template4 = StringTemplate["Time elapsed = 'runtime' min"];
2070     numiter = 0;
2071     startTime = Now;
2072     If[$FrontEnd != Null,
2073     (
2074         temp = PrintTemporary[
2075             Dynamic[
2076                 Grid[
2077                     {
2078                         {template1[<|"numiter"->numiter, "totaliter"->
2079 totalIterations|>]},
2080                         {template2[<|"remtime"->Round[QuantityMagnitude[
2081 UnitConvert[(Now-startTime)/(Max[1,numiter])*(totalIterations-
2082 numiter), "min"]], 0.1|>]},
2083                         {template4[<|"runtime"->Round[QuantityMagnitude[
2084 UnitConvert[(Now-startTime), "min"]], 0.1|>]},
2085                         {ProgressIndicator[numiter, {1, totalIterations}]}
2086                     ]
2087                 ]
2088             ]
2089         ];
2090     );
2091     ];
2092     Do[
2093     (
2094         JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp,
2095 CFTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas" ->
2096 OptionValue["ChenDeltas"]];
2097         numiter += 1;
2098     ),
2099     {Jp, AllowedJ[numE]},
2100     {J, AllowedJ[numE]}
2101 ];
2102 If[$FrontEnd != Null,
2103     NotebookDelete[temp]
2104 ];
2105 Return[JJBlockMatrixTable];
2106 )
2107
2108 Options[TabulateManyJJBlockMatrixTables] = {"Overwrite"->False, "
2109 Sparse"->True, "ChenDeltas"->False, "FilenameAppendix"-> "", "
2110 Compressed" -> False};
2111
2112 TabulateManyJJBlockMatrixTables::usage = "
2113 TabulateManyJJBlockMatrixTables[{n1, n2, ...}] calculates the

```

```

tables of matrix elements for the requested fni 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.
2097 JJBlockMatrixTable is an association whose keys are of the form {n,
J, Jp} and whose values are matrix elements.";
2098 TabulateManyJJBlockMatrixTables[ns_, OptionsPattern[]]:= (
2099     overwrite = OptionValue["Overwrite"];
2100     fName = <||>;
2101     fileApp = OptionValue["FilenameAppendix"];
2102     ExportFun = If[OptionValue["Compressed"], ExportMZip, Export];
2103     Do[
2104         (
2105             CFdataFilename = FileNameJoin[{moduleDir, "data", "
CrystalFieldTable_f"<>ToString[numE]<>".zip"}];
2106             PrintTemporary["Importing CrystalFieldTable from ",
CFdataFilename, " ..."];
2107             CrystalFieldTable = ImportMZip[CFdataFilename];
2108
2109             PrintTemporary["#----- numE = ", numE, " -----#"];
2110             exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix"
-> fileApp];
2111             fName[numE] = exportFname;
2112             If[FileExistsQ[exportFname] && Not[overwrite],
2113                 Continue[]
2114             ];
2115             JJBlockMatrixTable = TabulateJJBlockMatrixTable[numE,
CrystalFieldTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas"
-> OptionValue["ChenDeltas"]];
2116             If[FileExistsQ[exportFname]&&overwrite,
2117                 DeleteFile[exportFname]
2118             ];
2119             ExportFun[exportFname, JJBlockMatrixTable];
2120
2121             ClearAll[CrystalFieldTable];
2122         ),
2123         {numE, ns}
2124     ];
2125     Return[fName];
2126 )
2127
2128 HamMatrixAssembly::usage="HamMatrixAssembly[numE] returns the
Hamiltonian matrix for the fni configuration. The matrix is
returned as a SparseArray."
2129 Options[HamMatrixAssembly] = {"FilenameAppendix"->""};
2130 HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
2131     {numE, ii, jj, howManyJs, Js, blockHam},
2132     (*******)
2133     ImportFun = ImportMZip;
2134     (*******)
2135     (*hole-particle equivalence enforcement*)
2136     numE = nf;

```

```

2137 allVars = {E0, E1, E2, E3, ζ, F0, F2, F4, F6, M0, M2, M4, T2, T2p,
2138   T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
2139   α, β, γ, B02, B04, B06, B12, B14, B16,
2140   B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16, S22,
2141   S24, S26, S34, S36, S44, S46, S56, S66, T11, T11p, T12, T14, T15
, T16,
2142   T17, T18, T19};
2143 params0 = AssociationThread[allVars, allVars];
2144 If[nf > 7,
2145   (
2146     numE = 14 - nf;
2147     params = HoleElectronConjugation[params0];
2148   ),
2149   params = params0;
2150 ];
2151 (* Load symbolic expressions for LS,J,J' energy sub-matrices. *)
2152 emFname = JJBBlockMatrixFileName[numE, "FilenameAppendix" ->
OptionValue["FilenameAppendix"]];
2153 JJBBlockMatrixTable = ImportFun[emFname];
2154 (*Patch together the entire matrix representation using J,J'
blocks.*)
2155 PrintTemporary["Patching JJ blocks ..."];
2156 Js = AllowedJ[numE];
2157 howManyJs = Length[Js];
2158 blockHam = ConstantArray[0, {howManyJs, howManyJs}];
2159 Do[
2160   blockHam[[jj, ii]] = JJBBlockMatrixTable[{numE, Js[[ii]], Js[[jj
]]}];,
2161   {ii, 1, howManyJs},
2162   {jj, 1, howManyJs}
2163 ];
2164 (* Once the block form is created flatten it *)
2165 blockHam = ArrayFlatten[blockHam];
2166 blockHam = ReplaceInSparseArray[blockHam, params];
2167 Return[blockHam];
2168 ]
2169
2170 Options[SimplerSymbolicHamMatrix]={
2171   "Export"->True,
2172   "PrependToFilename"->"",
2173   "EorF"->"F",
2174   "Overwrite" -> False,
2175   "Return" -> True};
2176 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.";
2177 SimplerSymbolicHamMatrix[numE_Integer, simplifier_List, OptionsPattern
[]]:=Module[
2178   {thisHam, eTofs, fname},

```

```

2179 (
2180   fname=FileNameJoin[{moduleDir,"hams",OptionValue["
PrependToFilename"]<>"SymbolicMatrix-f"<>ToString[numE]<>".m"}];
2181   If[FileExistsQ[fname] && Not[OptionValue["Overwrite"]],
2182     (
2183       If[OptionValue["Return"],
2184         (
2185           Print["File ",fname," already exists, and option \"
Overwrite\" is set to False, loading file ..."];
2186           thisHam = Import[fname];
2187           Return[thisHam];
2188         ),
2189         (
2190           Print["File ",fname," already exists, skipping ..."];
2191           Return[Null];
2192         )
2193       ]
2194     )
2195 ];
2196 thisHam=HamMatrixAssembly[numE];
2197 thisHam=ReplaceInSparseArray[thisHam,simplifier];
2198 If[OptionValue["Export"],
2199 (
2200   Print["Exporting to file ",fname];
2201   Export[fname,thisHam]
2202 )
2203 ];
2204 If[OptionValue["Return"],
2205   Return[thisHam],
2206   Return[Null]
2207 ];
2208 )
2209 ]
2210
2211 (* ##### Block assembly
##### *)
2212 (*
#####
*)
2213
2214 (*
#####
*)
2215 (* ##### Printers and Labels
##### *)
2216
2217 PrintL::usage = "PrintL[L] give the string representation of a given
angular momentum.";
2218 PrintL[L_] := If[StringQ[L], L, StringTake[specAlphabet, {L + 1}]]
2219
2220 FindSL::usage = "FindSL[LS] gives the spin and orbital angular
momentum that corresponds to the provided string LS.";
2221 FindSL[SL_] := (
2222   FindSL[SL] =
2223   If[StringQ[SL],

```

```

2224 {
2225   (ToExpression[StringTake[SL, 1]]-1)/2,
2226   StringPosition[specAlphabet, StringTake[SL, {2}]][[1, 1]]-1
2227 },
2228 SL
2229 ]
2230 )
2231
2232 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.";
2233 PrintSLJ[SLJ_] :=
2234   RowBox[{SuperscriptBox[" ", 2 SLJ[[1]] + 1],
2235     SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]}] // DisplayForm;
2236
2237 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."
  ;
2238 PrintSLJM[SLJM_] :=
2239   RowBox[{SuperscriptBox[" ", 2 SLJM[[1]] + 1],
2240     SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]}] //
2241   DisplayForm;
2242
2243 (* ##### Printers and Labels
  ##### *)
2244 (*
  #####
  *)
2245
2246 (*
  #####
  *)
2247 (* ##### Term management
  ##### *)
2248
2249 AllowedSLTerms::usage = "AllowedSLTerms[numE] returns a list with
  the allowed terms in the fnumE 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.";
2250 AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE,
  14-numE]]]
2251
2252 AllowedNKSLTerms::usage = "AllowedNKSLTerms[numE] returns a list
  with the allowed terms in the fnumE configuration, the terms are
  given as strings in spectroscopic notation. The integers in the
  last positions are used to distinguish cases with degeneracy.";
2253 AllowedNKSLTerms[numE_] := (Map[First, CFPTerms[Min[numE, 14-numE
  ]]])

```

```

2254 AllowedNKSLTerms[0] = {"1S"};
2255 AllowedNKSLTerms[14] = {"1S"};
2256
2257 MaxJ::usage = "MaxJ[numE] gives the maximum J = S+L that corresponds
to the configuration f^numE.";
2258 MaxJ[numE_] := Max[Map[Total, AllowedSLTerms[Min[numE, 14-numE]]]]
2259
2260 MinJ::usage = "MinJ[numE] gives the minimum J = S+L that corresponds
to the configuration f^numE.";
2261 MinJ[numE_] := Min[Map[Abs[Part[#, 1] - Part[#, 2]] &,
AllowedSLTerms[Min[numE, 14-numE]]]]
2262
2263 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.";
2264 AllowedSLJTerms[numE_] :=
2265   Module[{idx1, allowedSL, allowedSLJ},
2266     allowedSL = AllowedSLTerms[numE];
2267     allowedSLJ = {};
2268     For[
2269       idx1 = 1,
2270       idx1 <= Length[allowedSL],
2271       termSL = allowedSL[[idx1]];
2272       termsSLJ =
2273         Table[
2274           {termSL[[1]], termSL[[2]], J},
2275           {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2276         ];
2277       allowedSLJ = Join[allowedSLJ, termsSLJ];
2278       idx1++
2279     ];
2280     SortBy[allowedSLJ, Last]
2281   ]
2282
2283 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.";
2284 AllowedNKSLJTerms[numE_] :=
2285   Module[{allowedSL, allowedNKSL, allowedSLJ, nn},
2286     allowedNKSL = AllowedNKSLTerms[numE];
2287     allowedSL = AllowedSLTerms[numE];
2288     allowedSLJ = {};
2289     For[
2290       nn = 1,
2291       nn <= Length[allowedSL],
2292       (
2293         termSL = allowedSL[[nn]];
2294         termNKSL = allowedNKSL[[nn]];
2295         termsSLJ =
2296           Table[{termNKSL, J},
2297             {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2298         ];

```



```

2299         allowedSLJ = Join[allowedSLJ, termsSLJ];
2300         nn++
2301     )
2302 ];
2303     SortBy[allowedSLJ, Last]
2304 ]
2305
2306 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.";
2307 AllowedNKSLforJTerms[numE_, J_] := Module[
2308     {allowedSL, allowedNKSL, allowedSLJ, nn, termSL, termNKSL,
termsSLJ},
2309     allowedNKSL = AllowedNKSLTerms[numE];
2310     allowedSL = AllowedSLTerms[numE];
2311     allowedSLJ = {};
2312     For[
2313         nn = 1,
2314         nn <= Length[allowedSL],
2315         (
2316             termSL = allowedSL[[nn]];
2317             termNKSL = allowedNKSL[[nn]];
2318             termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[
termSL],
2319                 {{termNKSL, J}},
2320                 {}
2321             ];
2322             allowedSLJ = Join[allowedSLJ, termsSLJ];
2323             nn++
2324         )
2325     ];
2326     Return[allowedSLJ]
2327 ];
2328
2329 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
}.";
2330 AllowedSLJMTerms[numE_] := Module[
2331     {allowedSLJ, allowedSLJM, termSLJ, termsSLJM, nn},
2332     allowedSLJ = AllowedSLJTerms[numE];
2333     allowedSLJM = {};
2334     For[
2335         nn = 1,
2336         nn <= Length[allowedSLJ],
2337         nn++,
2338         (
2339             termSLJ = allowedSLJ[[nn]];
2340             termsSLJM =
2341                 Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
2342                     {M, - termSLJ[[3]], termSLJ[[3]]}
2343             ];
2344             allowedSLJM = Join[allowedSLJM, termsSLJM];

```

```

2345     )
2346   ];
2347   Return[SortBy[allowedSLJM, Last]];
2348 ]
2349
2350 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}.";
2351 AllowedNKSLJMforJMTerms[numE_, J_, MJ_] :=
2352   Module[{allowedSL, allowedNKSL, allowedSLJM, nn},
2353     allowedNKSL = AllowedNKSLTerms[numE];
2354     allowedSL = AllowedSLTerms[numE];
2355     allowedSLJM = {};
2356     For[
2357       nn = 1,
2358       nn <= Length[allowedSL],
2359       termSL = allowedSL[[nn]];
2360       termNKSL = allowedNKSL[[nn]];
2361       termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
2362         <= J
2363         <= Total[termSL]
2364         && (Abs[MJ] <= J)
2365         ),
2366         {{termNKSL, J, MJ}},
2367         {}];
2368       allowedSLJM = Join[allowedSLJM, termsSLJ];
2369       nn++;
2370     ];
2371     Return[allowedSLJM];
2372 ]
2373
2374 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}.";
2375 AllowedNKSLJMforJTerms[numE_, J_] :=
2376   Module[{MJs, labelsAndMomenta, termsWithJ},
2377     (
2378       MJs = AllowedMforJ[J];
2379       (* Pair LS labels and their {S,L} momenta *)
2380       labelsAndMomenta = ({#, FindSL[#]}) & /@ AllowedNKSLTerms[numE];
2381       (* A given term will contain J if |L-S|<=J<=L+S *)
2382       ContainsJ[{SL_String, {S_, L_}}] := (Abs[S - L] <= J <= (S + L));
2383       (* Keep just the terms that satisfy this condition *)
2384       termsWithJ = Select[labelsAndMomenta, ContainsJ];
2385       (* We don't want to keep the {S,L} *)
2386       termsWithJ = {#[[1]], J} & /@ termsWithJ;
2387       (* This is just a quick way of including up all the MJ values *)
2388       Return[Flatten /@ Tuples[{termsWithJ, MJs}]]
2389     )
2390 ]

```

```

2391 AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J,
2392 1].";
2393 AllowedMforJ[J_] := Range[-J, J, 1];
2394
2395 AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta
2396 J that appear in the f^numE configuration.";
2397 AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}};
2398
2399 Seniority::usage="Seniority[LS] returns the seniority of the given
2400 term."
2401 Seniority[LS_] := FindNKLSTerm[LS][[1, 2]]
2402
2403 FindNKLSTerm::usage = "Given the string LS FindNKLSTerm[SL] returns
2404 all the terms that are compatible with it. This is only for f^n
2405 configurations. The provided terms might belong to more than one
2406 configuration. The function returns a list with elements of the
2407 form {LS, seniority, W, U}.";
2408 FindNKLSTerm[SL_] := Module[
2409   {NKterms, n},
2410   n = 7;
2411   NKterms = {};
2412   Map[
2413     If[! StringFreeQ[First[#], SL],
2414       If[ToExpression[Part[#, 2]] <= n,
2415         NKterms = Join[NKterms, {#}, 1]
2416       ]
2417     ] &,
2418     fnTermLabels
2419   ];
2420   NKterms = DeleteCases[NKterms, {}];
2421   NKterms]
2422
2423 Options[ParseTermLabels] = {"Export" -> True};
2424 ParseTermLabels::usage="ParseTermLabels[] parses the labels for the
2425 terms in the f^n configurations based on the labels for the f6 and
2426 f7 configurations. The function returns a list whose elements are
2427 of the form {LS, seniority, W, U}.";
2428 ParseTermLabels[OptionsPattern[]] := Module[
2429   {labelsTextData, fNtextLabels, nielsonKosterLabels, seniorities,
2430   RacahW, RacahU},
2431   (
2432     labelsTextData = FileNameJoin[{moduleDir, "data", "
2433     NielsonKosterLabels_f6_f7.txt"}];
2434     fNtextLabels = Import[labelsTextData];
2435     nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
2436     termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
2437     seniorities = Map[ToExpression[Part[#, {2}]] &,
2438     nielsonKosterLabels];
2439     racahW =
2440     Map[
2441       StringTake[
2442         Flatten[StringCases[Part[#, {3}],
2443         "(" ~~ DigitCharacter ~~ DigitCharacter ~~ DigitCharacter
2444         ~~ ")"]],

```

```

2432         {2, 4}
2433     ] &,
2434     nielsonKosterLabels];
2435   racahU =
2436     Map[
2437       StringTake[
2438         Flatten[StringCases[Part[# , {3}],
2439           "(" ~~ DigitCharacter ~~ DigitCharacter ~~ ")"[]],
2440         {2, 3}
2441       ] &,
2442       nielsonKosterLabels];
2443   fnTermLabels = Join[termLabels, seniorities, racahW, racahU, 2];
2444   fnTermLabels = Sort[fnTermLabels];
2445   If[OptionValue["Export"],
2446     (
2447       broadFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
2448       Export[broadFname, fnTermLabels];
2449     )
2450   ];
2451   Return[fnTermLabels];
2452 )
2453 ]
2454
2455 (* ##### Term management
2456 ##### *)
2457
2458 Options[LoadParameters] = {
2459   "Source" -> "Carnall",
2460   "Free Ion" -> False,
2461   "gs" -> 2.002319304386
2462 };
2463 LoadParameters::usage = "LoadParameters[ln] takes a string with the
2464   symbol the element of a trivalent lanthanide ion and returns model
2465   parameters for it. It is based on the data for LaF3. If the
2466   option \"Free Ion\" is set to True then the function sets all
2467   crystal field parameters to zero. Through the option \"gs\" it
2468   allows modifying the electronic gyromagnetic ratio. For
2469   completeness this function also computes the E parameters using
2470   the F parameters quoted on Carnall.";
2471 LoadParameters[Ln_String, OptionsPattern[]]:=
2472   Module[{source, params},
2473     (
2474       source = OptionValue["Source"];
2475       params = Which[source=="Carnall",
2476         (Association[Carnall["data"][Ln]])
2477       ];
2478       (*If a free ion then all the parameters from the crystal field
2479       are set to zero*)
2480       If[OptionValue["Free Ion"],
2481         Do[params[cfSymbol] = 0,
2482           {cfSymbol, cfSymbols}
2483       ]
2484     )

```

```

2476 ];
2477 params[F0] = 0;
2478 params[M2] = 0.56 * params[M0]; (* See Carnall 1989, Table I,
caption, probably fixed based on HF values*)
2479 params[M4] = 0.31 * params[M0]; (* See Carnall 1989, Table I,
caption, probably fixed based on HF values*)
2480 params[P0] = 0;
2481 params[P4] = 0.5 * params[P2]; (* See Carnall 1989, Table I,
caption, probably fixed based on HF values*)
2482 params[P6] = 0.1 * params[P2]; (* See Carnall 1989, Table I,
caption, probably fixed based on HF values*)
2483 params[gs] = OptionValue["gs"];
2484 {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[
F0], params[F2], params[F4], params[F6]];
2485 params[E0] = 0;
2486 Return[params];
2487 )
2488 ];
2489
2490 HoleElectronConjugation::usage = "HoleElectronConjugation[params]
takes the parameters (as an association) that define a
configuration and converts them so that they may be interpreted as
corresponding to a complementary hole configuration. Some of this
can be simply done by changing the sign of the model parameters.
In the case of the effective three body interaction the
relationship is more complex and is controlled by the value of the
isE variable.";
2491
2492 HoleElectronConjugation[params_] :=
2493 Module[{newparams = params},
2494 (
2495   flipSignsOf = {ζ, T2, T3, T4, T6, T7, T8};
2496   flipSignsOf = Join[flipSignsOf, cfSymbols];
2497   flipped =
2498     Table[(flipper -> - newparams[flipper]),
2499     {flipper, flipSignsOf}
2500   ];
2501   nonflipped =
2502     Table[(flipper -> newparams[flipper]),
2503     {flipper, Complement[Keys[newparams], flipSignsOf]}
2504   ];
2505   flippedParams = Association[Join[nonflipped, flipped]];
2506   Return[flippedParams];
2507 )
2508 ]
2509
2510 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.
2511
2512 Parameters
2513 -----

```

```

2514 numE (int) : Number of f-electrons.
2515
2516 Options
2517 -----
2518 \["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.
2519
2520 Returns
2521 -----
2522 {rmsDifference, gtEnergies, cfenergies, ln, carnallAssignments, {
      fstates, basis, symbolicMatrix}} (list): with
2523   rmsDifference (float) : The root-mean-square difference between
      the calculated values from Carnall and the ones computed here.
2524   gtEnergies (list) : The calculated values for the energy levels as
      quoted by Carnall.
2525   cfenergies (list) : The calculated values for the energy levels as
      calculated here.
2526   ln (string) : The symbol of the lanthanide ion.
2527   carnallAssignments (list) : The assignments of the energy levels
      as quoted by Carnall.
2528   {fstates, basis, symbolicMatrix} (list) : The eigenstates, basis
      and symbolic matrix as calculated here.
2529 ";
2530 Options[IonSolverLaF3] = {"Include Spin-Spin" -> True};
2531 IonSolverLaF3[numE_, OptionsPattern[]] := (
2532   spinspin = OptionValue["Include Spin-Spin"];
2533   host = "LaF3";
2534   ln = StringSplit["Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb"][[numE
      ]];
2535   terms = AllowedNKSJTerms[Min[numE, 14 - numE]];
2536   expData = Flatten[#["Exp (1/cm)"] & /@ Values[Carnall["appendix:"
      <> ln <> ":Association"]]]];
2537
2538   (*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*})
2539   simplifier = {
2540     B12 -> 0, B14 -> 0, B16 -> 0, B34 -> 0, B36 -> 0, B56 -> 0,
2541     S12 -> 0, S14 -> 0, S16 -> 0, S22 -> 0, S24 -> 0, S26 -> 0,
2542     S34 -> 0, S36 -> 0, S44 -> 0, S46 -> 0, S56 -> 0, S66 -> 0,
2543     T11 -> 0, T12 -> 0, T14 -> 0, T15 -> 0, T16 -> 0, T18 -> 0, T11p
      -> 0,
2544     T17 -> 0, T19 -> 0
2545   };
2546   eTofs = (#[[1]] -> #[[2]]) & /@ Transpose[{{E0, E1, E2, E3}, FtoE[
      F0, F2, F4, F6]}}];
2547   ham = Normal[HamMatrixAssembly[numE, 0]];
2548   simpleHam = ham /. simplifier;
2549   simpleHam = simpleHam /. eTofs;
2550   hamParams = DeleteDuplicates[Flatten[Variables /@ simpleHam]];
2551   ham = Normal[HamMatrixAssembly[numE, 0]];
2552   termNames = First /@ terms;
2553   termSimplifier =
2554     Table[

```

```

2555     termN -> If[StringLength[termN] == 3,
2556         StringTake[termN, {1, 2}],
2557         termN
2558     ],
2559     {termN, termNames}
2560 ];
2561
2562 (*Load the parameters from Carnall*)
2563 params = LoadParameters[ln, "Free Ion" -> False];
2564 (*Enforce the override to the spin-spin contribution to the
magnetic interactions*)
2565 params[[Sigma]SS] = If[spinspin, 1, 0];
2566 (*Everything that is not given is set to zero*)
2567 params = ParamPad[params, "Print" -> True];
2568
2569 {fstates, basis, symbolicMatrix} =
2570 SolveStates[params[nf], 0, params, "Return Symbolic Matrix" ->
True];
2571 symbolicMatrix =
2572     If[spinspin,
2573         ReplaceInSparseArray[symbolicMatrix, {[Sigma]SS -> 1}],
2574         ReplaceInSparseArray[symbolicMatrix, {[Sigma]SS -> 0}]
2575     ];
2576 fstates = ShiftedLevels[fstates];
2577 fstates = SortBy[fstates, First];
2578 cfenergies = First /@ fstates;
2579 cfenergies = Chop[cfenergies];
2580 If[OddQ[numE],
2581     (
2582         cfenergies = cfenergies[;; ;; 2];
2583     )
2584 ];
2585
2586 mainKey = StringTemplate["appendix:'Ln':Association"] [<|"Ln" -> ln
|>];
2587 lnData = Carnall[mainKey];
2588 carnalKeys = lnData // Keys;
2589 repetitions = Length[lnData[#]["Calc (1/cm)"]] & /@ carnalKeys;
2590 carnallAssignments =
2591     First /@ Carnall["appendix:" <> ln <> ":RawTable"];
2592
2593 carnalKey = StringTemplate["appendix:'Ln':Calculated"] [<|"Ln" ->
ln|>];
2594 gtEnergies = Sort[Carnall[carnalKey]];
2595 diffs = Sort[cfenergies][;; Length[gtEnergies]] - gtEnergies;
2596 rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
2597
2598 Return[{rmsDifference, gtEnergies, cfenergies, ln,
carnallAssignments, {fstates, basis, symbolicMatrix}}]
2599 )
2600
2601 FastIonSolverLaF3::usage =
2602     "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.
2603
2604 The function returns a list with seven elements {rmsDifference,
carnallEnergies, eigenEnergies, ln, carnallAssignments, eigensys,
basis}. Where:
2605
2606 rmsDifference is the root mean squared difference between the
calculated values and those quoted by Carnall
2607
2608 carnallEnergies are the quoted calculated values from Carnall;
2609
2610 eigenEnergies are the calculated energies (in the case of an odd
number of electrons the kramers degeneracy has been elided from
this list);
2611
2612 ln is simply a string labelling the corresponding lanthanide;
2613
2614 carnallAssignments is a list of strings providing the term
assignments that Carnall assumed,
2615
2616 eigensys is a list of tuples where the first element is the energy
corresponding to the eigenvector given as the second element;
2617
2618 basis a list that specifies the basis in which the Hamiltonian was
constructed and diagonalized.
2619 ";
2620 Options[FastIonSolverLaF3] = {
2621 "MakeNotebook" -> True,
2622 "NotebookSave" -> True,
2623 "HTMLSave" -> False,
2624 "eigenstateTruncationProbability" -> 0.9,
2625 "Include spin-spin" -> True,
2626 "Max Eigenstates in Table" -> 100,
2627 "Sparse" -> True,
2628 "PrintFun" -> Print,
2629 "SaveData" -> True,
2630 "paramFiddle" -> {},
2631 "Append to Filename" -> ""
2632 };
2633 FastIonSolverLaF3[numE_, OptionsPattern[]] := Module[{
2634 makeNotebook, eigenstateTruncationProbability, spinspin, host,
2635 ln, terms, termNames, carnallEnergies, eigenEnergies,
2636 simplerStateLabels,
2637 eigensys, basis, assignmentMatches, stateLabels,
2638 carnallAssignments},
2639 (
2640 PrintFun = OptionValue["PrintFun"];
2641 makeNotebook = OptionValue["MakeNotebook"];
2642 eigenstateTruncationProbability = OptionValue["
eigenstateTruncationProbability"];
2643 maxStatesInTable = OptionValue["Max Eigenstates in Table"];
2644 spinspin = OptionValue["Include spin-spin"];
2645 host = "LaF3";
2646 paramFiddle = OptionValue["paramFiddle"];

```



```

2645 ln = theLanthanides[[numE]];
2646 terms = AllowedNKSJTerms[Min[numE, 14 - numE]];
2647 termNames = First /@ terms;
2648 (* For labeling the states, the degeneracy in some of the terms is
    elided *)
2649 PrintFun["> Calculating simpler term labels ..."];
2650 termSimplifier =
2651   Table[termN -> If[StringLength[termN] == 3,
2652     StringTake[termN, {1, 2}],
2653     termN
2654   ],
2655   {termN, termNames}
2656 ];
2657
2658 (*Load the parameters from Carnall*)
2659 PrintFun["> Loading the fit parameters from Carnall ..."];
2660 params = LoadParameters[ln, "Free Ion" -> False];
2661 If[numE>7,
2662   (
2663     PrintFun["> Conjugating the parameters accounting for the hole
    -particle equivalence ..."];
2664     params = HoleElectronConjugation[params];
2665     params[t2Switch] = 0;
2666   ),
2667   params[t2Switch] = 1;
2668 ];
2669
2670 Do[params[key] = paramFiddle[key],
2671   {key, Keys[paramFiddle]}
2672 ];
2673
2674 (* Import the symbolic Hamiltonian *)
2675 PrintFun["> Loading the symbolic Hamiltonian for this
    configuration ..."];
2676 startTime = Now;
2677 numH = 14 - numE;
2678 numEH = Min[numE, numH];
2679 C2vsimplifier = {B12 -> 0, B14 -> 0, B16 -> 0, B34 -> 0, B36 -> 0,
2680   B56 -> 0,
2681   S12 -> 0, S14 -> 0, S16 -> 0, S22 -> 0, S24 -> 0, S26 -> 0,
2682   S34 -> 0, S36 -> 0,
2683   S44 -> 0, S46 -> 0, S56 -> 0, S66 -> 0, T11p -> 0, T11 -> 0,
2684   T12 -> 0, T14 -> 0, T15 -> 0,
2685   T16 -> 0, T18 -> 0, T17 -> 0, T19 -> 0};
2686 simpleHam = If[
2687   ValueQ[symbolicHamiltonians[numEH]],
2688   symbolicHamiltonians[numEH],
2689   SimplerSymbolicHamMatrix[numE, C2vsimplifier, "PrependToFilename
    " -> "C2v-", "Overwrite" -> False]
2690 ];
2691 endTime = Now;
2692 loadTime = QuantityMagnitude[endTime - startTime, "Seconds"];
2693 PrintFun[">> Loading the symbolic Hamiltonian took ", loadTime, "
    seconds."];
2694

```

```

2695 (*Enforce the override to the spin-spin contribution to the
2696 magnetic interactions*)
2697
2698 (*Everything that is not given is set to zero*)
2699 params = ParamPad[params, "Print" -> False];
2700 PrintFun[params];
2701 (* numHam = simpleHam /. params; *)
2702 numHam = ReplaceInSparseArray[simpleHam, params];
2703 If[Not[OptionValue["Sparse"]],
2704   numHam = Normal[numHam]
2705 ];
2706 PrintFun["> Calculating the SLJ basis ..."];
2707 basis = BasisLSJMJ[numE];
2708
2709 (*Remove numerical noise*)
2710 PrintFun["> Diagonalizing the numerical Hamiltonian ..."];
2711 startTime = Now;
2712 eigensys = Eigensystem[numHam];
2713 endTime = Now;
2714 diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
2715 PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
2716 eigensys = Chop[eigensys];
2717 eigensys = Transpose[eigensys];
2718
2719 (*Shift the baseline energy*)
2720 eigensys = ShiftedLevels[eigensys];
2721 (*Sort according to energy*)
2722 eigensys = SortBy[eigensys, First];
2723 (*Grab just the energies*)
2724 eigenEnergies = First /@ eigensys;
2725
2726 (*Energies are doubly degenerate in the case of odd number of
2727 electrons, keep only one*)
2728 If[OddQ[numE],
2729   (
2730     PrintFun["> Since there's an odd number of electrons energies
2731 come in pairs, taking just one for each pair ..."];
2732     eigenEnergies = eigenEnergies[;; ;; 2];
2733   )
2734 ];
2735
2736 (*Compare against the data quoted by Bill Carnall*)
2737 PrintFun["> Comparing against the data from Carnall ..."];
2738 mainKey = StringTemplate["appendix:'Ln':Association"]
2739 ][<|"Ln" -> ln|>];
2740 lnData = Carnall[mainKey];
2741 carnalKeys = lnData // Keys;
2742 repetitions = Length[lnData[#]["Calc (1/cm)"]] & /@
2743 carnalKeys;
2744 carnallAssignments = First /@ Carnall["appendix:" <> ln <> "":
2745 RawTable"];
2746 carnalKey = StringTemplate["appendix:'Ln':Calculated"] [<|
2747 "Ln" -> ln|>];
2748 carnallEnergies = Carnall[carnalKey];

```

```

2743 (* For the difference take as many energies as quoted by Bill*)
2744 eigenEnergies = eigenEnergies + carnallEnergies[[1]];
2745 diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] -
2746 carnallEnergies;
2747 (* Remove the differences where the appendix tables have elided
values*)
2748 rmsDifference = Sqrt[Mean[(Select[diffs, FreeQ[#, Missing[]] &)]
^2]];
2749 titleTemplate = StringTemplate[
2750 "Energy Level Diagram of \!\(\(*SuperscriptBox[\('ion'\), \(\(3\)\
\(+\)\)\]\)\)"];
2751 title = titleTemplate[<|"ion" -> ln|>];
2752 parsedStates = ParseStates[eigensys, basis];
2753 If[OddQ[numE],
2754   parsedStates = parsedStates[[;; ;; 2]]];
2755
2756 stateLabels = #[[-1]] & /@ parsedStates;
2757 simplerStateLabels = ((#[[2]] /. termSimplifier) <> ToString
#[[3]], InputForm) & /@ parsedStates;
2758
2759 PrintFun[">> Truncating eigenvectors to given probability ..."];
2760 startTime = Now;
2761 truncatedStates = ParseStatesByProbabilitySum[eigensys, basis,
2762   eigenstateTruncationProbability,
2763   0.01];
2764 endTime = Now;
2765 truncationTime = QuantityMagnitude[endTime - startTime, "Seconds"
];
2766 PrintFun[">>> Truncation took ", truncationTime, " seconds."];
2767
2768 If[makeNotebook,
2769   (
2770     PrintFun["> Putting together results in a notebook ..."];
2771     energyDiagram = Framed[
2772       EnergyLevelDiagram[eigensys, "Title" -> title,
2773         "Background" -> White]
2774       , Background -> White, FrameMargins -> 50];
2775     appToFname = OptionValue["Append to Filename"];
2776     PrintFun[">> Comparing the term assignments between qlanth and
Carnall ..."];
2777     assignmentMatches =
2778       If[StringContainsQ[#[[1]], #[[2]], "\[Checkmark]", "X"] & /@
2779         Transpose[{carnallAssignments, simplerStateLabels[[;; Length
[carnallAssignments]]]}];
2780     assignmentMatches = {{"\[Checkmark]",
2781       Count[assignmentMatches, "\[Checkmark]"]}, {"X",
2782       Count[assignmentMatches, "X"]}};
2783     labelComparison = (If[StringContainsQ[#[[1]], #[[2]], "\[
Checkmark]", "X"] & /@
2784       Transpose[{carnallAssignments,
2785         simplerStateLabels[[;; Length[carnallAssignments]]]}]);
2786     labelComparison =
2787       PadRight[labelComparison, Length[simplerStateLabels], "-"];
2788

```

```

2789     statesTable =
2790     Grid[Prepend[{Round[#[[1]]], #[[2]]} & /@
2791     truncatedStates[;;Min[Length[eigensys],maxStatesInTable
]]], {"Energy/!\(\(*SuperscriptBox[\(cm\), \(-1\)]\)\"",
2792     "\[Psi]"}], Frame -> All, Spacings -> {2, 2},
2793     FrameStyle -> Blue,
2794     Dividers -> {{False, True, False}, {True, True}}];
2795     DefaultIfMissing[expr_] := If[FreeQ[expr, Missing[]], expr, "NA"
];
2796     PrintFun[">> Rounding the energy differences for table
presentation ..."];
2797     roundedDiffs = Round[diffs, 0.1];
2798     roundedDiffs = PadRight[roundedDiffs, Length[
simplerStateLabels], "-"];
2799     roundedDiffs = DefaultIfMissing /@ roundedDiffs;
2800     diffs = PadRight[diffs, Length[simplerStateLabels], "-"];
2801     diffs = DefaultIfMissing /@ diffs;
2802     diffTableData = Transpose[{simplerStateLabels, eigenEnergies,
2803     labelComparison,
2804     PadRight[carnallAssignments, Length[simplerStateLabels], "
-"],
2805     DefaultIfMissing/@PadRight[carnallEnergies, Length[
simplerStateLabels], "-"],
2806     roundedDiffs}];
2807     diffTable =
2808     TableForm[diffTableData,
2809     TableHeadings -> {None, {"qlanth",
2810     "E/!\(\(*SuperscriptBox[\(cm\), \(-1\)]\)\"", "", "Carnall",
2811     "E/!\(\(*SuperscriptBox[\(cm\), \(-1\)]\)\"",
2812     "\[CapitalDelta]E/!\(\(*SuperscriptBox[\(cm\), \(-1\)]\)\"",
}}];
2813
2814     diffs = Sort[eigenEnergies][;; Length[carnallEnergies]] -
carnallEnergies;
2815     notBad = FreeQ[#,Missing[]]&/@diffs;
2816     diffs = Pick[diffs,notBad];
2817     diffHistogram =
2818     Histogram[diffs, Frame -> True, ImageSize -> 800,
2819     AspectRatio -> 1/3, FrameStyle -> Directive[16],
2820     FrameLabel -> {"(qlanth-carnall)/Ky", "Freq"}];
2821     rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
2822     labelTemplate =
2823     StringTemplate[
2824     "\!\(\(*SuperscriptBox[\(\'ln\'\", \(\(3\)\)\(+\)\)]\)\""];
2825     diffData = diffs;
2826     diffLabels = simplerStateLabels[;;Length[notBad]];
2827     diffLabels = Pick[diffLabels, notBad];
2828     diffPlot = Framed[
2829     ListLabelPlot[diffData,
2830     diffLabels,
2831     Frame -> True,
2832     PlotRange -> All,
2833     ImageSize -> 1200,
2834     AspectRatio -> 1/3,
2835     FrameLabel -> {"",

```

```

2836         "(qlanth-carnall) / \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"
2837     },
2838     PlotMarkers -> "OpenMarkers",
2839     PlotLabel ->
2840         Style[labelTemplate[<| "ln" -> ln|>] <> " | " <> "\[Sigma]="
2841         <> ToString[Round[rmsDifference, 0.01]] <>
2842         " \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)\n", 20],
2843     Background -> White
2844 ],
2845 Background -> White,
2846 FrameMargins -> 50
2847 ];
2848 nb = CreateDocument[{
2849     TextCell[Style[DisplayForm[SuperscriptBox[host <> ":" <> ln,
2850 "3+"]]], "Title", TextAlignment -> Center],
2851     TextCell["Energy Diagram", "Section", TextAlignment ->
2852 Center],
2853     TextCell[energyDiagram, TextAlignment -> Center],
2854     TextCell["Multiplet Assignments & Energy Levels", "Section",
2855 TextAlignment -> Center],
2856     TextCell[diffHistogram, TextAlignment -> Center],
2857     TextCell[diffPlot, "Output", TextAlignment -> Center],
2858     TextCell[assignmentMatches, "Output", TextAlignment ->
2859 Center],
2860     TextCell[diffTable, "Output", TextAlignment -> Center],
2861     TextCell["Truncated Eigenstates", "Section", TextAlignment
2862 -> Center],
2863     TextCell["These are some of the resultant eigenstates which
2864 add up to at least a total probability of " <> ToString[
2865 eigenstateTruncationProbability] <> ".", "Text", TextAlignment ->
2866 Center],
2867     TextCell[statesTable, "Output", TextAlignment -> Center]
2868 },
2869 WindowSelected -> True,
2870 WindowTitle -> ln <> " in " <> "LaF3" <> appToFname,
2871 WindowSize -> {1600, 800}];
2872 If[OptionValue["SaveData"],
2873 (
2874     exportFname = FileNameJoin[{moduleDir, "calcs", ln <> " in
2875 " <> "LaF3" <> appToFname <> ".m"}];
2876     SelectionMove[nb, After, Notebook];
2877     NotebookWrite[nb, Cell["Reload Data", "Section",
2878 TextAlignment -> Center]];
2879     NotebookWrite[nb, Cell[(
2880         "{rmsDifference, carnallEnergies, eigenEnergies, ln,
2881 carnallAssignments, simplerStateLabels, eigensys, basis,
2882 truncatedStates} = Import[FileNameJoin[{NotebookDirectory[], \"\" <>
2883 StringSplit[exportFname, \"/"]][[-1]] <> \"\"];\"
2884     ), "Input"]];
2885     NotebookWrite[nb, Cell[(
2886         "Manipulate[First[MinimalBy[truncatedStates, Abs[First
2887 [#] - energy] &]], {energy, 0}]"
2888     ), "Input"]];
2889     SelectionMove[nb, Before, Notebook];

```

```

2875         Export[exportFname, {rmsDifference, carnallEnergies,
eigenEnergies, ln, carnallAssignments, simplerStateLabels,
eigensys, basis, truncatedStates}];
2876         tinyexportFname = FileNameJoin[{moduleDir, "calcs", ln <> "
in " <> "LaF3" <> appToFname <> "- tiny.m"}];
2877         tinyExport = <|"ln"->ln,
2878             "carnallEnergies"->carnallEnergies,
2879             "rmsDifference"-> rmsDifference,
2880             "eigenEnergies"-> eigenEnergies,
2881             "carnallAssignments"-> carnallAssignments,
2882             "simplerStateLabels" -> simplerStateLabels
|>;
2883         Export[tinyexportFname, tinyExport];
2884     )
2885 ];
2886 If[OptionValue["NotebookSave"],
2887     (
2888         nbFname = FileNameJoin[{moduleDir, "calcs", ln <> " in " <>
"LaF3" <> appToFname <> ".nb"}];
2889         PrintFun[">> Saving notebook to ", nbFname, " ..."];
2890         NotebookSave[nb, nbFname];
2891     )
2892 ];
2893 If[OptionValue["HTMLSave"],
2894     (
2895         htmlFname = FileNameJoin[{moduleDir, "calcs", "html", ln <>
" in " <> "LaF3" <> appToFname <> ".html"}];
2896         PrintFun[">> Saving html version to ", htmlFname, " ..."];
2897         Export[htmlFname, nb];
2898     )
2899 ];
2900 )
2901 ];
2902
2903 Return[{rmsDifference, carnallEnergies, eigenEnergies, ln,
carnallAssignments, simplerStateLabels, eigensys, basis,
truncatedStates}];
2904 )
2905 ];
2906
2907 ShiftedLevels::usage = "
2908 ShiftedLevels[originalLevels] takes a list of levels of the form
2909 {{energy_1, coeff_vector_1},
2910 {energy_2, coeff_vector_2},
2911 ...}}
2912 and returns the same input except that now to every energy the
minimum of all of them has been subtracted.";
2913 ShiftedLevels[originalLevels_] :=
2914 Module[{groundEnergy, shifted},
2915     groundEnergy = Sort[originalLevels][[1,1]];
2916     shifted = Map[{-#[[1]] - groundEnergy, #[[2]]} &,
originalLevels];
2917     Return[shifted];
2918 ]
2919

```

```

2920 (* #####
2921 *)
2922 (* ##### Eigensystem analysis
2923 ##### *)
2924 PrettySaundersSLJmJ::usage = "PrettySaundersSLJmJ[{SL, J, mJ}]
2925 produces a human-redeable symbol for the given basis vector {SL, J
2926 , mJ}."
2927 PrettySaundersSLJmJ[{SL_, J_, mJ_}] := (If[
2928 StringQ[SL],
2929 ({S, L} = FindSL[SL];
2930 L = StringTake[SL, {2, -1}];
2931 ),
2932 {S, L} = SL];
2933 Return[
2934 RowBox[{AdjustmentBox[Style[2*S + 1, Smaller],
2935 BoxBaselineShift -> -1, BoxMargins -> 0],
2936 AdjustmentBox[PrintL[L], BoxMargins -> -0.2],
2937 AdjustmentBox[
2938 Style[{InputForm[J], mJ}, Small, FontTracking -> "Narrow"],
2939 BoxBaselineShift -> 1,
2940 BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]]] // DisplayForm]]
2941
2942 BasisVecInRusselSaunders::usage = "BasisVecInRusselSaunders[basisVec
2943 ] takes a basis vector in the format {LSstring, Jval, mJval} and
2944 returns a human-readable symbol for the corresponding Russel-
2945 Saunders term."
2946 BasisVecInRusselSaunders[basisVec_] := (
2947 {LSstring, Jval, mJval} = basisVec;
2948 Ket[PrettySaunders[LSstring, Jval], mJval]
2949 )
2950
2951 LSJMJTemplate =
2952 StringTemplate[
2953 "\\!\\(\\*TemplateBox[{\\nRowBox[{\\n'LS'", \\n,\\n, \\nRowBox[{\\n'J'", \\n, \\n=\\n, \\n'J'\\n"}], \\n,\\n, \\nRowBox[{\\n'mJ'", \\n=\\n, \\n'mJ'\\n"}]}],\\n\\n\\nKet\\n}\\n)\\n"];
2954
2955 BasisVecInLSJMJ::usage = "BasisVecInLSJMJ[basisVec] takes a basis
2956 vector in the format {{LSstring, Jval}, mJval}, nucSpin} and
2957 returns a human-readable symbol for the corresponding LSJMJ term
2958 in the form |LS, J=..., mJ=...>."
2959 BasisVecInLSJMJ[basisVec_] := (
2960 {LSstring, Jval, mJval} = basisVec;
2961 LSJMJTemplate[<|
2962 "LS" -> LSstring,
2963 "J" -> ToString[Jval, InputForm],
2964 "mJ" -> ToString[mJval, InputForm]|>]
2965 );
2966
2967 ParseStates::usage = "ParseStates[states, basis] takes a list of
2968 eigenstates in terms of their coefficients in the given basis and
2969 returns a list of the same states in terms of their energy, LSJMJ
2970 symbol, J, mJ, S, L, LSJ symbol, and LS symbol. The LS symbol
2971 returned corresponds to the term with the largest coefficient in

```

```

2960   the given basis.";
2961 ParseStates[states_, basis_, OptionsPattern[]] := Module[{
2962   parsedStates},
2963   (
2964     parsedStates = Table[(
2965       {energy, eigenVec} = state;
2966       maxTermIndex = Ordering[Abs[eigenVec]][[-1]];
2967       {LSstring, Jval, mJval} = basis[[maxTermIndex]];
2968       LSJsymbol = Subscript[LSstring, {Jval, mJval}];
2969       LSJMJsymbol = LSstring <> ToString[Jval, InputForm];
2970       {S, L} = FindSL[LSstring];
2971       {energy, LSstring, Jval, mJval, S, L, LSJsymbol, LSJMJsymbol}
2972     ),
2973     {state, states}];
2974   Return[parsedStates]
2975 )
2976 ]
2977
2978 ParseStatesByNumBasisVecs::usage = "ParseStatesByNumBasisVecs[states
2979 , basis, numBasisVecs] takes a list of eigenstates in terms of
2980 their coefficients in the given basis and returns a list of the
2981 same states in terms of their energy and the coefficients of the
2982 numBasisVecs most significant basis vectors.";
2983 ParseStatesByNumBasisVecs[states_, basis_, numBasisVecs_, roundTo_ :
2984 0.01] := (
2985   parsedStates = Table[(
2986     {energy, eigenVec} = state;
2987     energy = Chop[energy];
2988     probs = Round[Abs[eigenVec^2], roundTo];
2989     amplitudes = Round[eigenVec, roundTo];
2990     ordering = Ordering[probs];
2991     chosenIndices = ordering[[-numBasisVecs ;;]];
2992     majorComponents = basis[[chosenIndices]];
2993     majorProbabilities = amplitudes[[chosenIndices]];
2994     majorComponents = BasisVecInLSJMJ /@ majorComponents;
2995     majorRep = majorProbabilities . majorComponents;
2996     {energy, majorRep}
2997   ),
2998   {state, fstates}];
2999   Return[parsedStates]
3000 )
3001
3002 FindThresholdPosition::usage = "FindThresholdPosition[list,
3003 threshold] returns the position of the first element in list that
3004 is greater than threshold. If no such element exists, it returns
3005 the length of list. The elements of the given list must be in
3006 ascending order.";
3007 FindThresholdPosition[list_, threshold_] :=
3008 Module[{position},
3009   position = Position[list, _?(# > threshold &), 1, 1];
3010   thrPos = If[Length[position] > 0,
3011     position[[1, 1]],
3012     Length[list]];
3013   If[thrPos == 0, Return[1], Return[thrPos+1]]
3014 ]

```



```

3004 ParseStateByProbabilitySum[{energy_, eigenVec_}, probSum_, roundTo_
3005 :0.01, maxParts_:20] := Compile[
3006   {{energy, _Real, 0},{eigenVec, _Complex, 1},{probSum, _Real, 0},
3007   {roundTo, _Real, 0},{maxParts, _Integer, 0}},
3008   Module[
3009     {numStates, state, amplitudes, probs, ordering,
3010     orderedProbs, truncationIndex, accProb, thresholdIndex,
3011     chosenIndices, majorComponents,
3012     majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
3013   (
3014     numStates      = Length[eigenVec];
3015     (*Round them up*)
3016     amplitudes      = Round[eigenVec, roundTo];
3017     probs           = Round[Abs[eigenVec^2], roundTo];
3018     ordering        = Reverse[Ordering[probs]];
3019     (*Order the probabilities from high to low*)
3020     orderedProbs    = probs[[ordering]];
3021     (*To speed up Accumulate, assume that only as much as maxParts
3022     will be needed*)
3023     truncationIndex = Min[maxParts, Length[orderedProbs]];
3024     orderedProbs    = orderedProbs[[;;truncationIndex]];
3025     (*Accumulate the probabilities*)
3026     accProb         = Accumulate[orderedProbs];
3027     (*Find the index of the first element in accProb that is greater
3028     than probSum*)
3029     thresholdIndex  = Min[Length[accProb], FindThresholdPosition[
3030     accProb, probSum]];
3031     (*Grab all the indices up till that one*)
3032     chosenIndices   = ordering[[;;thresholdIndex]];
3033     (*Select the corresponding elements from the basis*)
3034     majorComponents = basis[[chosenIndices]];
3035     (*Select the corresponding amplitudes*)
3036     majorAmplitudes = amplitudes[[chosenIndices]];
3037     (*Take their absolute value*)
3038     absMajorAmplitudes = Abs[majorAmplitudes];
3039     (*Make sure that there are no effectively zero contributions*)
3040     notnullAmplitudes = Flatten[Position[absMajorAmplitudes, x_ /; x
3041     != 0]];
3042     (* majorComponents = PrettySaundersSLJmJ
3043     [{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents; *)
3044     majorComponents = PrettySaundersSLJmJ /@ majorComponents;
3045     majorAmplitudes = majorAmplitudes[[notnullAmplitudes]];
3046     (*Make them into Kets*)
3047     majorComponents = Ket /@ majorComponents[[notnullAmplitudes]];
3048     (*Multiply and add to build the final Ket*)
3049     majorRep        = majorAmplitudes . majorComponents;
3050   );
3051   Return[{energy, majorRep}]
3052 ],
3053   CompilationTarget -> "C",
3054   RuntimeAttributes -> {Listable},
3055   Parallelization -> True,
3056   RuntimeOptions -> "Speed"
3057 ];

```

```

3051 ParseStatesByProbabilitySum::usage = "ParseStatesByProbabilitySum[
3052   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.";
3053 ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ :
   0.01, maxParts_: 20] := Module[
3054   {parsedByProb, numStates, state, energy, eigenVec, amplitudes,
   probs, ordering,
3055   orderedProbs, truncationIndex, accProb, thresholdIndex,
   chosenIndices, majorComponents,
3056   majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
3057   (
3058     numStates      = Length[eigensys];
3059     parsedByProb = Table[(
3060       state        = eigensys[[idx]];
3061       {energy, eigenVec} = state;
3062       (*Round them up*)
3063       amplitudes    = Round[eigenVec, roundTo];
3064       probs         = Round[Abs[eigenVec^2], roundTo];
3065       ordering      = Reverse[Ordering[probs]];
3066       (*Order the probabilities from high to low*)
3067       orderedProbs  = probs[[ordering]];
3068       (*To speed up Accumulate, assume that only as much as maxParts
   will be needed*)
3069       truncationIndex = Min[maxParts, Length[orderedProbs]];
3070       orderedProbs    = orderedProbs[[;;truncationIndex]];
3071       (*Accumulate the probabilities*)
3072       accProb         = Accumulate[orderedProbs];
3073       (*Find the index of the first element in accProb that is greater
   than probSum*)
3074       thresholdIndex = Min[Length[accProb], FindThresholdPosition[
   accProb, probSum]];
3075       (*Grab all the indices up till that one*)
3076       chosenIndices  = ordering[[;; thresholdIndex]];
3077       (*Select the corresponding elements from the basis*)
3078       majorComponents = basis[[chosenIndices]];
3079       (*Select the corresponding amplitudes*)
3080       majorAmplitudes = amplitudes[[chosenIndices]];
3081       (*Take their absolute value*)
3082       absMajorAmplitudes = Abs[majorAmplitudes];
3083       (*Make sure that there are no effectively zero contributions*)
3084       notnullAmplitudes = Flatten[Position[absMajorAmplitudes, x_ /;
   x != 0]];
3085       (* majorComponents = PrettySaundersSLJmJ
   [{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents; *)
3086       majorComponents = PrettySaundersSLJmJ /@ majorComponents;
3087       majorAmplitudes = majorAmplitudes[[notnullAmplitudes]];
3088       (*Make them into Kets*)
3089       majorComponents = Ket /@ majorComponents[[notnullAmplitudes
   ]];
3090       (*Multiply and add to build the final Ket*)
3091       majorRep        = majorAmplitudes . majorComponents;
3092       {energy, majorRep}

```

```

3093     ), {idx, numStates}];
3094 Return[parsedByProb]
3095 )
3096 ];
3097
3098 (* ##### Eigensystem analysis
3099 ##### *)
3100 (*
3101 #####
3102 *)
3103 (* ##### Misc
3104 ##### *)
3105
3106 SymbToNum::usage = "SymbToNum[expr, numAssociation] takes an
3107 expression expr and returns what results after making the
3108 replacements defined in the given replacementAssociation. If
3109 replacementAssociation doesn't define values for expected keys,
3110 they are taken to be zero.";
3111 SymbToNum[expr_, replacementAssociation_] := (
3112     includedKeys = Keys[replacementAssociation];
3113     (*If a key is not defined, make its value zero.*)
3114     fullAssociation = Table[(
3115         If[MemberQ[includedKeys, key],
3116             ToExpression[key]->replacementAssociation[key],
3117             ToExpression[key]->0
3118         ],
3119     ),
3120     {key, paramSymbols}];
3121 Return[expr /. fullAssociation];
3122 )
3123
3124 SimpleConjugate::usage = "SimpleConjugate[expr] takes an expression
3125 and applies a simplified version of the conjugate in that all it
3126 does is that it replaces the imaginary unit I with -I. It assumes
3127 that every other symbol is real so that it remains the same under
3128 complex conjugation. Among other expressions it is valid for any
3129 rational or polynomial expression with complex coefficients and
3130 real variables.";
3131 SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;
3132
3133 ExportMZip::usage="ExportMZip[\"dest.[zip,m]\"] saves a compressed
3134 version of expr to the given destination.";
3135 ExportMZip[filename_, expr_] := Module[{baseName, exportName,
3136     mImportName, zipImportName},
3137 (
3138     baseName      = FileBaseName[filename];
3139     exportName     = StringReplace[filename, ".m" -> ".zip"];
3140     mImportName    = StringReplace[exportName, ".zip" -> ".m"];
3141     If[FileExistsQ[mImportName],
3142     (
3143         PrintTemporary[mImportName <> " exists already, deleting"];

```

```

3130     DeleteFile[mImportName];
3131     Pause[2];
3132 )
3133 ];
3134 Export[exportName, (baseName<>".m") -> expr]
3135 )
3136 ];
3137
3138 Options[ImportMZip]={"Leave Uncompressed" -> True};
3139 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 uncompressed version of the object in the same
folder of filename";
3140 ImportMZip[filename_String, OptionsPattern[]] := Module[
3141 {baseName, importKey, zipImportName, mImportName, imported},
3142 (
3143     baseName      = FileNameBaseName[filename];
3144     (*Function allows for the filename to be .m or .zip*)
3145     importKey      = baseName <> ".m";
3146     zipImportName  = StringReplace[filename, ".m"->".zip"];
3147     mImportName    = StringReplace[zipImportName, ".zip"->".m"];
3148     If[FileExistsQ[mImportName],
3149     (
3150         PrintTemporary[".m version exists already, importing that
instead ..."];
3151         Return[Import[mImportName]];
3152     )
3153     ];
3154     imported = Import[zipImportName, importKey];
3155     If[OptionValue["Leave Uncompressed"],
3156     Export[mImportName, imported]
3157     ];
3158     Return[imported]
3159 )
3160 ];
3161
3162 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.";
3163 ReplaceInSparseArray[s_SparseArray, rule_] := (With[{
3164     elem = s["NonzeroValues"]/.rule,
3165     def  = s["Background"]/.rule
3166 },
3167     (* Return[{elem,def}]; *)
3168     srep = SparseArray[Automatic,
3169         s["Dimensions"],
3170         def,
3171         {1, {s["RowPointers"], s["ColumnIndices"]}}, elem}
3172     ];
3173 ];
3174 Return[srep];
3175 );
3176

```

```

3177 Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
3178 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.";
3179 ParseTeXLikeSymbol[bigString_, OptionsPattern[]] := (
3180     form = OptionValue["Form"];
3181     (*parse greek*)
3182     symbols = Table[(
3183         str = StringReplace[string, {"\\alpha" -> "\alpha",
3184             "\\beta" -> "\beta",
3185             "\\gamma" -> "\gamma",
3186             "\\psi" -> "\[Psi]"}];
3187         symbol = Which[
3188             StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"]],
3189             (
3190                 (*yes sub no sup*)
3191                 mainSymbol = StringSplit[str, "_"][[1]];
3192                 mainSymbol = ToExpression[mainSymbol];
3193
3194                 subPart =
3195                     StringCases[str,
3196                         RegularExpression@"\\{(.*)\\}" -> "$1"][[1]];
3197                 Subscript[mainSymbol, subPart]
3198             ),
3199             Not[StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
3200             (
3201                 (*no sub yes sup*)
3202                 mainSymbol = StringSplit[str, "^"][[1]];
3203                 mainSymbol = ToExpression[mainSymbol];
3204
3205                 supPart =
3206                     StringCases[str,
3207                         RegularExpression@"\\{(.*)\\}" -> "$1"][[1]];
3208                 Superscript[mainSymbol, supPart]],
3209             StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
3210             (
3211                 (*yes sub yes sup*)
3212                 mainSymbol = StringSplit[str, "_"][[1]];
3213                 mainSymbol = ToExpression[mainSymbol];
3214                 {subPart, supPart} =
3215                     StringCases[str, RegularExpression@"\\{(.*)\\}" -> "$1"];
3216                 Subsuperscript[mainSymbol, subPart, supPart]
3217             ),
3218             True,
3219             ((*no sup or sub*)
3220             str)
3221         ];
3222     symbol
3223 ),
3224 {string, StringSplit[bigString, " "]}];
3225 Which[

```

```

3226     form == "Row",
3227     Return[Row[symbols]],
3228     form == "List",
3229     Return[symbols]
3230 ]
3231 );
3232
3233 (* ##### Misc
3234 ##### *)
3235
3236 (* #####
3237 ##### *)
3238
3239 (* ##### Some Plotting Routines
3240 ##### *)
3241
3242 EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes states
3243 and produces a visualization of its energy spectrum.
3244 The resultant visualization can be navigated by clicking and
3245 dragging to zoom in on a region, or by clicking and dragging
3246 horizontally while pressing Ctrl. Double-click to reset the view."
3247 ;
3248 Options[EnergyLevelDiagram] = {
3249     "Title" -> "",
3250     "ImageSize" -> 1000,
3251     "AspectRatio" -> 1/8,
3252     "Background" -> "Automatic",
3253     "Epilog" -> {}
3254 };
3255 EnergyLevelDiagram[states_, OptionsPattern[]]:= (
3256     energies = First/@states;
3257     epi = OptionValue["Epilog"];
3258     ExploreGraphics@ListPlot[Tooltip[{{#, 0}, {#, 1}}, {Quantity
3259 [#/8065.54429, "eV"], Quantity[#, 1/"Centimeters"]}] &/@ energies,
3260     Joined -> True,
3261     PlotStyle -> Black,
3262     AspectRatio -> OptionValue["AspectRatio"],
3263     ImageSize -> OptionValue["ImageSize"],
3264     Frame -> True,
3265     PlotRange -> {All, {0, 1}},
3266     FrameTicks -> {{None, None}, {Automatic, Automatic}},
3267     FrameStyle -> Directive[15, Dashed, Thin],
3268     PlotLabel -> Style[OptionValue["Title"], 15, Bold],
3269     Background -> OptionValue["Background"],
3270     FrameLabel -> {"\!\(\(*FractionBox[\(E\), SuperscriptBox[\(cm\)]
3271 , \((-1\)]\)\)"},
3272     Epilog -> epi]
3273 )
3274
3275 ExploreGraphics::usage =
3276     "Pass a Graphics object to explore it. Zoom by clicking and
3277     dragging a rectangle. Pan by clicking and dragging while pressing

```

```

3268 Ctrl. Click twice to reset view.
3269 Based on ZeitPolizei @ https://mathematica.stackexchange.com/
3270 questions/7142/how-to-manipulate-2d-plots";
3271
3272 OptAxesRedraw::usage =
3273 "Option for ExploreGraphics to specify redrawing of axes. Default
3274 False.";
3275 Options[ExploreGraphics] = {OptAxesRedraw -> False};
3276
3277 ExploreGraphics[graph_Graphics, opts : OptionsPattern[]] := With[
3278 {gr = First[graph],
3279  opt = DeleteCases[Options[graph],
3280    PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _],
3281  plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
3282  ar = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
3283  ao = AbsoluteOptions[AxesOrigin],
3284  rectangle = {Dashing[Small],
3285    Line[{#1,
3286      {First[#2], Last[#1]},
3287      #2,
3288      {First[#1], Last[#2]},
3289      #1]}] &,
3290  optAxesRedraw = OptionValue[OptAxesRedraw]},
3291 DynamicModule[
3292 {dragging=False, first, second, rx1, rx2, ry1, ry2,
3293  range = plr},
3294 {{rx1, rx2}, {ry1, ry2}} = plr;
3295 Panel@
3296 EventHandler[
3297   Dynamic@Graphics[
3298     If[dragging, {gr, rectangle[first, second]}, gr],
3299     PlotRange -> Dynamic@range,
3300     AspectRatio -> ar,
3301     AxesOrigin -> If[optAxesRedraw,
3302       Dynamic@Mean[range\[Transpose]], ao],
3303     Sequence @@ opt],
3304   {"MouseDown", 1} :> (
3305     first = MousePosition["Graphics"]
3306   ),
3307   {"MouseDragged", 1} :> (
3308     dragging = True;
3309     second = MousePosition["Graphics"]
3310   ),
3311   "MouseClicked" :> (
3312     If[CurrentValue@"MouseClicked"]==2,
3313     range = plr];
3314   ),
3315   {"MouseUp", 1} :> If[dragging,
3316     dragging = False;
3317
3318     range = {{rx1, rx2}, {ry1, ry2}} =
3319       Transpose@{first, second};
3320     range[[2]] = {0, 1},
3321   {"MouseDown", 2} :> (
3322     first = {sx1, sy1} = MousePosition["Graphics"]

```

```

3320     ),
3321     {"MouseDown", 2} :> (
3322         second = {sx2, sy2} = MousePosition["Graphics"];
3323         rx1 = rx1 - (sx2 - sx1);
3324         rx2 = rx2 - (sx2 - sx1);
3325         ry1 = ry1 - (sy2 - sy1);
3326         ry2 = ry2 - (sy2 - sy1);
3327         range = {{rx1, rx2}, {ry1, ry2}};
3328         range[[2]] = {0, 1};
3329     )]]];
3330
3331 Options[LabeledGrid]={
3332     ItemSize->Automatic,
3333     Alignment->Center,
3334     Frame->All,
3335     "Separator"->"," ,
3336     "Pivot"->""
3337 };
3338 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.";
3339 LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]]:=
    Module[
3340     {gridList=data,rowHeads=rowHeaders,colHeads=columnHeaders},
3341     (
3342     separator=OptionValue["Separator"];
3343     pivot=OptionValue["Pivot"];
3344     gridList=Table[
3345         Tooltip[
3346             data[[rowIdx,colIdx]],
3347             DisplayForm[
3348                 RowBox[{rowHeads[[rowIdx]],
3349                     separator,
3350                     colHeads[[colIdx]]}
3351             ]
3352         ],
3353         {rowIdx,Dimensions[data][[1]]},
3354         {colIdx,Dimensions[data][[2]]}];
3355     gridList=Transpose[Prepend[gridList,colHeads]];
3356     rowHeads=Prepend[rowHeads,pivot];
3357     gridList=Prepend[gridList,rowHeads]//Transpose;
3358     Grid[gridList,
3359         Frame->OptionValue[Frame],
3360         Alignment->OptionValue[Alignment],
3361         Frame->OptionValue[Frame],
3362         ItemSize->OptionValue[ItemSize]
3363     ]
3364 )
3365 ]
3366
3367 Options[HamiltonianForm]={ "Separator"->"," , "Pivot"->""}
3368

```



```

3369 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.";
3370 HamiltonianForm[hamMatrix_, basisLabels_List, OptionsPattern[]]:= (
3371     braLabels=DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[
    RightBracketingBar]"}]] & /@ basisLabels;
3372     ketLabels=DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[
    RightAngleBracket]"}]] & /@ basisLabels;
3373     LabeledGrid[hamMatrix, braLabels, ketLabels, "Separator" ->
    OptionValue["Separator"], "Pivot" -> OptionValue["Pivot"]]
3374 )
3375
3376 Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover"
    -> True, "Overlay Values" -> True}];
3377 HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts :
    OptionsPattern[]] := (
3378     braLabels = DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[
    RightBracketingBar]"}]] & /@ basisLabels;
3379     ketLabels = DisplayForm[Rotate[RowBox[{"\[LeftBracketingBar]", #,
    "\[RightAngleBracket]"}]], \[Pi]/2]] & /@ basisLabels;
3380     ketLabelsUpright = DisplayForm[RowBox[{"\[LeftBracketingBar]", #,
    "\[RightAngleBracket]"}]] & /@ basisLabels;
3381     numRows = Length[hamMatrix];
3382     numCols = Length[hamMatrix[[1]]];
3383     epiThings = Which[
3384         And[OptionValue["Hover"], Not[OptionValue["Overlay Values"]]],
3385         Flatten[
3386             Table[
3387                 Tooltip[
3388                     {
3389                         Transparent,
3390                         Rectangle[
3391                             {j - 1, numRows - i},
3392                             {j - 1, numRows - i} + {1, 1}
3393                         ]
3394                     },
3395                     Row[{braLabels[[i]], ketLabelsUpright[[j]], "=", hamMatrix[[i,
    j]]}]
3396                 ],
3397                 {i, 1, numRows},
3398                 {j, 1, numCols}
3399             ]
3400         ],
3401         And[OptionValue["Hover"], OptionValue["Overlay Values"]],
3402         Flatten[
3403             Table[
3404                 Tooltip[
3405                     {
3406                         Transparent,
3407                         Rectangle[
3408                             {j - 1, numRows - i},

```

```

3409         {j - 1, numRows - i} + {1, 1}
3410     ]
3411 },
3412     DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i]],
"\[LeftBracketingBar]", basisLabels[[j]], "\[RightAngleBracket]"
}]]
3413 ],
3414     {i, numRows},
3415     {j, numCols}
3416 ]
3417 ],
3418     True,
3419     {}
3420 ];
3421 textOverlay = If[OptionValue["Overlay Values"],
3422     (
3423         Flatten[
3424             Table[
3425                 Text[hamMatrix[[i, j]],
3426                     {j - 1/2, numRows - i + 1/2}
3427             ],
3428             {i, 1, numRows},
3429             {j, 1, numCols}
3430         ]
3431     ),
3432     {}
3433 ];
3434 epiThings = Join[epiThings, textOverlay];
3435 MatrixPlot[hamMatrix,
3436     FrameTicks -> {
3437         {Transpose[{Range[Length[braLabels]], braLabels}], None},
3438         {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}
3439     },
3440     Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
3441     Epilog -> epiThings
3442 ];
3443
3444
3445
3446 (* ##### Some Plotting Routines
##### *)
3447 (*
#####
*)
3448
3449 (*
#####
*)
3450 (* ##### Load Functions
##### *)
3451
3452 LoadAll::usage="LoadAll[] executes all Load* functions.";
3453 LoadAll[]:=(
3454     LoadTermLabels[];
3455     LoadCFP[];

```

```

3456     LoadUk [];
3457     LoadV1k [];
3458     LoadT22 [];
3459     LoadS00andECSOLS [];
3460
3461     LoadElectrostatic [];
3462     LoadSpinOrbit [];
3463     LoadS00andECSO [];
3464     LoadSpinSpin [];
3465     LoadThreeBody [];
3466     LoadChenDeltas [];
3467     LoadCarnall [];
3468 )
3469
3470 LoadTermLabels::usage="LoadTermLabels[] loads into the session the
3471 labels for the terms in the f^n configurations.";
3472 LoadTermLabels[]:= (
3473     If[ValueQ[fnTermLabels], Return[]];
3474     PrintTemporary["Loading data for state labels in the f^n
3475 configurations..."];
3476     fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
3477     fnTermLabels::usage = "This list contains the labels of f^n
3478 configurations. Each element of the list has four elements {LS,
3479 seniority, W, U}. At first sight this seems to only include the
3480 labels for the f^6 and f^7 configuration, however, all is included
3481 in these two.";
3482     If[!FileExistsQ[fnTermsFname],
3483         (PrintTemporary[">> fnTerms.m not found, generating ..."];
3484         fnTermLabels = ParseTermLabels["Export"->True];
3485         ),
3486         fnTermLabels = Import[fnTermsFname];
3487 ];
3488 )
3489
3490 Carnall::usage = "Association of data from Carnall et al (1989) with
3491 the following keys: {data, annotations, paramSymbols,
3492 elementNames, rawData, rawAnnotations, annnotatedData, appendix:Pr
3493 :Association, appendix:Pr:Calculated, appendix:Pr:RawTable,
3494 appendix:Headings}";
3495 LoadCarnall::usage="LoadCarnall[] loads data for trivalent
3496 lanthanides in LaF3 using the data from Bill Carnall's 1989 paper.
3497 ";
3498 LoadCarnall[]:= (
3499     If[ValueQ[Carnall], Return[]];
3500     carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
3501     If[!FileExistsQ[carnallFname],
3502         (PrintTemporary[">> Carnall.m not found, generating ..."];
3503         Carnall = ParseCarnall[];
3504         ),
3505         Carnall = Import[carnallFname];
3506 ];
3507 )

```

```

3498 LoadChenDeltas::usage="LoadChenDeltas[] loads the differences noted
      by Chen.";
3499 LoadChenDeltas[]:=(
3500   If[ValueQ[chenDeltas], Return[]];
3501   PrintTemporary["Loading the association of discrepancies found by
      Chen ..."];
3502   chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"
      }];
3503   If[!FileExistsQ[chenDeltasFname],
3504     (PrintTemporary[">> chenDeltas.m not found, generating ..."];
3505      chenDeltas = ParseChenDeltas[];
3506     ),
3507     chenDeltas = Import[chenDeltasFname];
3508   ];
3509 );
3510
3511 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";
3512 Options[ParseChenDeltas] = {"Export" -> True};
3513 ParseChenDeltas[OptionsPattern[]]:= (
3514   chenDeltasRaw = Import[FileNameJoin[{moduleDir, "data", "the-chen-
      deltas-A.csv"}]];
3515   chenDeltasRaw = chenDeltasRaw[[2 ;;]];
3516   chenDeltas = <|>;
3517   chenDeltasA = <|>;
3518   Off[Power::infy];
3519   Do[
3520     ({right, wrong} = {chenDeltasRaw[[row]][[4 ;;]],
3521      chenDeltasRaw[[row + 1]][[4 ;;]]};
3522     key = chenDeltasRaw[[row]][[1 ;; 3]];
3523     repRule = (#[[1]] -> #[[2]]*#[[1]]) & /@
3524       Transpose[{{M0, M2, M4, P2, P4, P6}, right/wrong}];
3525     chenDeltasA[key] = <|"right" -> right, "wrong" -> wrong,
3526       "repRule" -> repRule|>;
3527     chenDeltasA[{key[[1]], key[[3]], key[[2]]}] = <|"right" -> right
3528       ,
3529       "wrong" -> wrong, "repRule" -> repRule|>;
3530     {row, 1, Length[chenDeltasRaw], 2}];
3531   chenDeltas["A"] = chenDeltasA;
3532
3533   chenDeltasRawB = Import[FileNameJoin[{moduleDir, "data", "the-chen-
      deltas-B.csv"}], "Text"];
3534   chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
3535   chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
3536   chenDeltasB = {ToExpression[StringTake[#[[1]], {2}]], #[[2]],
      #[[3]]} & /@ chenDeltasB;
3537   chenDeltas["B"] = chenDeltasB;
3538   On[Power::infy];
3539   If[OptionValue["Export"],
3540     (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.
      m"}]);
3541     Export[chenDeltasFname, chenDeltas];

```

```

3542     )
3543   ];
3544   Return[chenDeltas];
3545 )
3546
3547 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.m";
3548 Options[ParseCarnall] = {"Export" -> True};
3549 ParseCarnall[] := (
3550   ions          = {"Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho", "Er",
3551     "Tm"};
3552   templates     = StringTemplate/@StringSplit["appendix:'ion':
  Association appendix:'ion':Calculated appendix:'ion':RawTable
  appendix:'ion':Headings", " "];
3553
3554   (* How many unique eigenvalues, after removing Kramer's degeneracy
  *)
3555   fullSizes     = AssociationThread[ions, {91, 182, 1001, 1001, 3003,
  1716, 3003, 1001, 1001, 182, 91}];
3556   carnall       = Import[FileNameJoin[{moduleDir, "data", "Carnall.xls"
  }]][[2]];
3557   carnallErr    = Import[FileNameJoin[{moduleDir, "data", "Carnall.xls"
  }]][[3]];
3558
3559   elementNames = carnall[[1]][[2;;]];
3560   carnall       = carnall[[2;;]];
3561   carnallErr    = carnallErr[[2;;]];
3562   carnall       = Transpose[carnall];
3563   carnallErr    = Transpose[carnallErr];
3564   paramNames   = ToExpression/@carnall[[1]][[1;;]];
3565   carnall       = carnall[[2;;]];
3566   carnallErr    = carnallErr[[2;;]];
3567   carnallData   = Table[(
3568     data        = carnall[[i]];
3569     data        = (#[[1]]->#[[2]])&/@Select[Transpose
  [{paramNames, data}], #[[2]]!="&"];
3570     elementNames[[i]]->data
3571   ),
3572     {i, 1, 13}
3573   ];
3574   carnallData   = Association[carnallData];
3575   carnallNotes  = Table[(
3576     data        = carnallErr[[i]];
3577     elementName = elementNames[[i]];
3578     dataFun     = (
3579       #[[1]] -> If[#[[2]]=="[",
3580         "Not allowed to vary in fitting.",
3581         If[#[[2]]=="[R]",
3582           "Ratio constrained by: " <> <|"Eu"->"F4/F2
  =0.713; F6/F2=0.512",
3583           "Gd"->"F4/F2=0.710]",
3584           "Tb"->"F4/F2=0.707"|>[elementName],
3585           If[#[[2]]=="i",
3586             "Interpolated",

```

```

3586         #[[2]]
3587     ]
3588 ]
3589 ] &;
3590 data = dataFun /@ Select[Transpose[{paramNames,
data}],#[[2]]!="&];
3591     elementName->data
3592 ),
3593 {i,1,13}
3594 ];
3595 carnallNotes = Association[carnallNotes];
3596
3597 annotatedData = Table[
3598     If[NumberQ[#[[1]]],Tooltip[#[[1]],#[[2]]],""] & /@
Transpose[{paramNames/. carnallData[element],
3599     paramNames/. carnallNotes[element]
3600     }],
3601     {element,elementNames}
3602 ];
3603 annotatedData = Transpose[annotatedData];
3604
3605 Carnall = <|"data"      -> carnallData,
3606     "annotations"     -> carnallNotes,
3607     "paramSymbols"    -> paramNames,
3608     "elementNames"    -> elementNames,
3609     "rawData"         -> carnall,
3610     "rawAnnotations"  -> carnallErr,
3611     "includedTableIons" -> ions,
3612     "annnotatedData"  -> annotatedData
3613 |>;
3614
3615 Do[(
3616     carnallData = Import[FileNameJoin[{moduleDir,"data","Carnall
.xls"}]][[i]];
3617     headers = carnallData[[1]];
3618     calcIndex = Position[headers,"Calc (1/cm)"][[1,1]];
3619     headers = headers[[2;;]];
3620     carnallLabels = carnallData[[1]];
3621     carnallData = carnallData[[2;;]];
3622     carnallTerms = DeleteDuplicates[First/@carnallData];
3623     parsedData = Table[(
3624         rows = Select[carnallData,#[[1]]==term&];
3625         rows = #[[2;;]]&/@rows;
3626         rows = Transpose[rows];
3627         rows = Transpose[{headers,rows}];
3628         rows = Association[(#[[1]]->#[[2]])&/@rows];
3629         term->rows
3630     ),
3631     {term,carnallTerms}
3632 ];
3633     carnallAssoc = Association[parsedData];
3634     carnallCalcEnergies = #[[calcIndex]]&/@carnallData;
3635     carnallCalcEnergies = If[NumberQ[#],#,Missing[]]&/
@carnallCalcEnergies;
3636     ion = ions[[i-3]];

```

```

3637     carnallCalcEnergies = PadRight[carnallCalcEnergies, fullSizes[
ion], Missing[]];
3638     keys                = #[<|"ion"->ion|>]&/@templates;
3639     Carnall[keys[[1]]]  = carnallAssoc;
3640     Carnall[keys[[2]]]  = carnallCalcEnergies;
3641     Carnall[keys[[3]]]  = carnallData;
3642     Carnall[keys[[4]]]  = headers;
3643     ),
3644     {i,4,14}
3645 ];
3646
3647 goodions = Select[ions, #!="Pm"&];
3648 expData  = Select[Transpose[Carnall["appendix:"<>#<>":RawTable"
]][[1+Position[Carnall["appendix:"<>#<>":Headings"], "Exp (1/cm)"
]][[1,1]]], NumberQ]&/@goodions;
3649 Carnall["All Experimental Data"]=AssociationThread[goodions,
expData];
3650 If[OptionValue["Export"],
3651 (
3652     carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
3653     Print["Exporting to "<>exportFname];
3654     Export[carnallFname, Carnall];
3655 )
3656 ];
3657 Return[Carnall];
3658 )
3659
3660 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";
3661 CFPAssoc::usage = " CFPAssoc is an association where keys are of
lists of the form {num_electrons, daughterTerm, 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[].";
3662 LoadCFP::usage="LoadCFP[] loads CFP, CFPAssoc, and CFPTable into the
session.";
3663 LoadCFP[]:= (
3664     If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]], Return[]];
3665
3666     PrintTemporary["Loading CFPTable ..."];
3667     CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
3668     If[!FileExistsQ[CFPTablefname],
3669         (PrintTemporary[">> CFPTable.m not found, generating ..."];
3670         CFPTable = GenerateCFPTable["Export"->True];
3671         ),
3672     CFPTable = Import[CFPTablefname];
3673 ];
3674
3675 PrintTemporary["Loading CFPs.m ..."];
3676 CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
3677 If[!FileExistsQ[CFPfname],

```

```

3678     (PrintTemporary[">> CFPs.m not found, generating ..."];
3679     CFP = GenerateCFP["Export"->True];
3680 ),
3681 CFP = Import[CFPfname];
3682 ];
3683
3684 PrintTemporary["Loading CFPAssoc.m ..."];
3685 CFPAffname = FileNameJoin[{moduleDir, "data", "CFPAff.m"}];
3686 If[!FileExistsQ[CFPAffname],
3687   (PrintTemporary[">> CFPAssoc.m not found, generating ..."];
3688     CFPAssoc = GenerateCFPAff["Export"->True];
3689   ),
3690   CFPAssoc = Import[CFPAffname];
3691 ];
3692 );
3693
3694 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[.].";
3695 LoadUk::usage="LoadUk[] loads into session the reduced matrix
    elements for unit tensor operators.";
3696 LoadUk[]:=(
3697   If[ValueQ[ReducedUkTable], Return[]];
3698   PrintTemporary["Loading the association of reduced matrix elements
    for unit tensor operators ..."];
3699   ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "
    ReducedUkTable.m"}];
3700   If[!FileExistsQ[ReducedUkTableFname],
3701     (PrintTemporary[">> ReducedUkTable.m not found, generating ..."]
3702   );
3703     ReducedUkTable = GenerateReducedUkTable[7];
3704   ),
3705   ReducedUkTable = Import[ReducedUkTableFname];
3706 ];
3707 );
3708
3709 ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}]
    provides the calculated result of Electrostatic[{n, SL, SpLp}].
    Load using LoadElectrostatic[.].";
3710 LoadElectrostatic::usage="LoadElectrostatic[] loads the reduced
    matrix elements for the electrostatic interaction.";
3711 LoadElectrostatic[]:=(
3712   If[ValueQ[ElectrostaticTable], Return[]];
3713   PrintTemporary["Loading the association of matrix elements for the
    electrostatic interaction ..."];
3714   ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "
    ElectrostaticTable.m"}];
3715   If[!FileExistsQ[ElectrostaticTablefname],
3716     (PrintTemporary[">> ElectrostaticTable.m not found, generating
    ..."]);
3717     ElectrostaticTable = GenerateElectrostaticTable[7];
3718   ),
3719   ElectrostaticTable = Import[ElectrostaticTablefname];
3720 ];

```



```

3720 );
3721
3722 LoadV1k::usage="LoadV1k[] loads into session the matrix elements of
3723 V1k.";
3724 LoadV1k[]:=(
3725   If[ValueQ[ReducedV1kTable], Return[]];
3726   PrintTemporary["Loading the association of matrix elements for V1k
3727   ..."];
3728   ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "
3729   ReducedV1kTable.m"}];
3730   If[!FileExistsQ[ReducedV1kTableFname],
3731     (PrintTemporary[">> ReducedV1kTable.m not found, generating ..."]
3732     );
3733     ReducedV1kTable = GenerateReducedV1kTable[7];
3734   ),
3735   ReducedV1kTable = Import[ReducedV1kTableFname];
3736 ];
3737 );
3738
3739 LoadSpinOrbit::usage="LoadSpinOrbit[] loads into session the matrix
3740 elements of the spin-orbit interaction.";
3741 LoadSpinOrbit[]:=(
3742   If[ValueQ[SpinOrbitTable], Return[]];
3743   PrintTemporary["Loading the association of matrix elements for
3744   spin-orbit ..."];
3745   SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "
3746   SpinOrbitTable.m"}];
3747   If[!FileExistsQ[SpinOrbitTableFname],
3748     (PrintTemporary[">> SpinOrbitTable.m not found, generating ..."]
3749     );
3750     SpinOrbitTable = GenerateSpinOrbitTable[7, True];
3751   ),
3752   SpinOrbitTable = Import[SpinOrbitTableFname];
3753 ];
3754 );
3755
3756 LoadS00andECSOLS::usage="LoadS00andECSOLS[] loads into session the
3757 LS reduced matrix elements of the S00-ECS0 interaction.";
3758 LoadS00andECSOLS[]:=(
3759   If[ValueQ[S00andECSOLSTable], Return[]];
3760   PrintTemporary["Loading the association of LS reduced matrix
3761   elements for S00-ECS0 ..."];
3762   S00andECSOLSTableFname = FileNameJoin[{moduleDir, "data", "
3763   ReducedS00andECSOLSTable.m"}];
3764   If[!FileExistsQ[S00andECSOLSTableFname],
3765     (PrintTemporary[">> ReducedS00andECSOLSTable.m not found,
3766     generating ..."];
3767     S00andECSOLSTable = GenerateS00andECSOLSTable[7];
3768   ),
3769   S00andECSOLSTable = Import[S00andECSOLSTableFname];
3770 ];
3771 );
3772
3773 LoadS00andECS0::usage="LoadS00andECS0[] loads into session the LSJ
3774 reduced matrix elements of spin-other-orbit and electrostatically-

```

```

correlated-spin-orbit.";
3762 LoadS00andECS0[]:=(
3763   If[ValueQ[S00andECS0TableFname], Return[]];
3764   PrintTemporary["Loading the association of matrix elements for
spin-other-orbit and electrostatically-correlated-spin-orbit ..."]
];
3765   S00andECS0TableFname = FileNameJoin[{moduleDir, "data", "
S00andECS0Table.m"}];
3766   If[!FileExistsQ[S00andECS0TableFname],
3767     (PrintTemporary[">> S00andECS0Table.m not found, generating ..."]
);
3768     S00andECS0Table = GenerateS00andECS0Table[7, "Export"->True];
3769   ),
3770   S00andECS0Table = Import[S00andECS0TableFname];
3771 ];
3772 );
3773
3774 LoadT22::usage="LoadT22[] loads into session the matrix elements of
T22.";
3775 LoadT22[]:=(
3776   If[ValueQ[T22Table], Return[]];
3777   PrintTemporary["Loading the association of reduced T22 matrix
elements ..."];
3778   T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.
m"}];
3779   If[!FileExistsQ[T22TableFname],
3780     (PrintTemporary[">> ReducedT22Table.m not found, generating ..."]
);
3781     T22Table = GenerateT22Table[7];
3782   ),
3783   T22Table = Import[T22TableFname];
3784 ];
3785 );
3786
3787 LoadSpinSpin::usage="LoadSpinSpin[] loads into session the matrix
elements of spin-spin.";
3788 LoadSpinSpin[]:=(
3789   If[ValueQ[SpinSpinTable], Return[]];
3790   PrintTemporary["Loading the association of matrix elements for
spin-spin ..."];
3791   SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "
SpinSpinTable.m"}];
3792   If[!FileExistsQ[SpinSpinTableFname],
3793     (PrintTemporary[">> SpinSpinTable.m not found, generating ..."];
3794     SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
3795   ),
3796   SpinSpinTable = Import[SpinSpinTableFname];
3797 ];
3798 );
3799
3800 LoadThreeBody::usage="LoadThreeBody[] loads into session the matrix
elements of three-body configuration-interaction effects.";
3801 LoadThreeBody[]:=(
3802   If[ValueQ[ThreeBodyTable], Return[]];

```

```

3803     PrintTemporary["Loading the association of matrix elements for
3804     three-body configuration-interaction effects ..."];
3805     ThreeBodyFname = FileNameJoin[{moduleDir, "data", "
3806     ThreeBodyTable.m"}];
3807     ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "
3808     ThreeBodyTables.m"}];
3809     If[!FileExistsQ[ThreeBodyFname],
3810       (PrintTemporary[">> ThreeBodyTable.m not found, generating ..."
3811       ];
3812       {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTables
3813       [14, "Export" -> True];
3814       ),
3815       ThreeBodyTable = Import[ThreeBodyFname];
3816       ThreeBodyTables = Import[ThreeBodiesFname];
3817     ];
3818   );
3819
3820   (* ##### Load Functions
3821   ##### *)
3822   (*
3823   #####
3824   *)
3825 End[]
3826
3827 LoadTermLabels[];
3828 LoadCFP[];
3829
3830 EndPackage[]

```

## 5 qonstants.m

```

1 BeginPackage["qonstants"];
2
3 (* Physical Constants*)
4 bohrRadius = 5.29177210903 * 10^-9;
5 ee = 1.602176634 * 10^-19;
6
7 (* Spectroscopic niceties*)
8 theLanthanides = {"Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy",
9   "Ho", "Er", "Tm", "Yb"};
10 theActinides = {"Ac", "Th", "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk",
11   "Cf", "Es", "Fm", "Md", "No", "Lr"};
12 theTrivalents = {"Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho",
13   "Er", "Tm"};
14 specAlphabet = "SPDFGHIKLMNOQRTUV";
15
16 EndPackage[];

```

## 6 qplotter.m

```

1 BeginPackage["qplotter"];
2
3
4 GetColor;
5 IndexMappingPlot;
6 ListLabelPlot;
7 AutoGraphicsGrid;
8
9 Begin["Private"];
10
11 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.";
12 Options[AutoGraphicsGrid] = Options[GraphicsGrid];
13 AutoGraphicsGrid[graphsList_, opts : OptionsPattern[]] :=
14 (
15     numGraphs = Length[graphsList];
16     width = Floor[Sqrt[numGraphs]];
17     height = Ceiling[numGraphs/width];
18     groupedGraphs = Partition[graphsList, width, width, 1, Null];
19     GraphicsGrid[groupedGraphs, opts]
20 )
21
22 Options[IndexMappingPlot] = Options[Graphics];
23 IndexMappingPlot::usage =
24 "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.";
25 IndexMappingPlot[pairs_, opts : OptionsPattern[]] := Module[{width,
    height}, (
26     width = Max[First /@ pairs];
27     height = width/3;
28     Return[
29         Graphics[{{Tooltip[Point[{#[[1]], 0}],#[[1]]}], Tooltip[Point
    [{#[[2]], height}],#[[2]]],
30             Line[{#[[1]], 0}, {#[[2]], height}]}]} & /@ pairs, opts,
    ImageSize -> 800]]
31 )
32 ]
33
34 TickCompressor[fTicks_] :=
35 Module[{avgTicks, prevTickLabel, groupCounter, groupTally, idx,
36     tickPosition, tickLabel, avgPosition, groupLabel}, {avgTicks = {}};
37 prevTickLabel = fTicks[[1, 2]];
38 groupCounter = 0;
39 groupTally = 0;
40 idx = 1;
41 Do[({tickPosition, tickLabel} = tick;
42     If[
43         tickLabel === prevTickLabel,
44         (groupCounter += 1;
45         groupTally += tickPosition;

```

```

46     groupLabel = tickLabel;),
47     (
48         avgPosition = groupTally/groupCounter;
49         avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
50         groupCounter = 1;
51         groupTally = tickPosition;
52         groupLabel = tickLabel;
53     )
54 ];
55 If[idx != Length[fTicks],
56     prevTickLabel = tickLabel;
57     idx += 1;]
58 ), {tick, fTicks}];
59 If[Or[Not[prevTickLabel === tickLabel], groupCounter > 1],
60     (
61         avgPosition = groupTally/groupCounter;
62         avgTicks = Append[avgTicks, {avgPosition, groupLabel}];
63     )
64 ];
65 Return[avgTicks];)]
66
67 GetColor[s_Style] := s /. Style[_ , c_] :> c
68 GetColor[_] := Black
69
70 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.";
71 Options[ListLabelPlot] = Append[Options[ListPlot], "TickCompression"->
    True];
72 ListLabelPlot[data_, labels_, opts : OptionsPattern[]] := Module[
73     {uniqueLabels, pallete, groupedByTerm, groupedKeys, scatterGroups,
74     groupedColors, frameTicks, compTicks, bottomTicks, topTicks},
75     (
76         uniqueLabels = DeleteDuplicates[labels];
77         pallete = Table[ColorData["Rainbow", i], {i, 0, 1,
78             1/(Length[uniqueLabels] - 1)}];
79         uniqueLabels = (#[[1]] -> #[[2]]) & /@ Transpose[{RandomSample[
80             uniqueLabels], pallete}];
81         uniqueLabels = Association[uniqueLabels];
82         groupedByTerm = GroupBy[Transpose[{labels, Range[Length[data]],
83             data}], First];
84         groupedKeys = Keys[groupedByTerm];
85         scatterGroups = Transpose[Transpose[#][[2 ;; 3]] & /@ Values[
86             groupedByTerm];
87         groupedColors = uniqueLabels[#] & /@ groupedKeys;
88         frameTicks = {Transpose[{Range[Length[data]],
89             Style[Rotate[#, 0], uniqueLabels[#]] & /@ labels}],
90             Automatic};
91         If[OptionValue["TickCompression"], (
92             compTicks = TickCompressor[frameTicks[[1]]];
93             bottomTicks =
94                 MapIndexed[
95                     If[EvenQ[First[#2]], {#1[[1]],

```

```

93         Tooltip[Style["\[SmallCircle]", GetColor
94         [#1[[2]]], #1[[2]]], #1] &, compTicks];
95     topTicks =
96         MapIndexed[
97             If[OddQ[First[#2]], {#1[[1]],
98                 Tooltip[Style["\[SmallCircle]", GetColor
99                 [#1[[2]]], #1[[2]]], #1] &, compTicks];
100         frameTicks = {{Automatic, Automatic}, {bottomTicks, topTicks
101         }};
102     ];
103     ListPlot[scatterGroups,
104         opts,
105         Frame->True,
106         PlotStyle -> groupedColors,
107         FrameTicks -> frameTicks]
108     )
109 ]
110 End[];
111
112 EndPackage[];

```

## 7 misc.m

```

1 BeginPackage["misc`"];
2
3 ExportToH5;
4 FlattenBasis;
5 RecoverBasis;
6 FlowMatching;
7 SuperIdentity;
8
9 GreedyMatching;
10 HelperNotebook;
11 StochasticMatching;
12 ExtractSymbolNames;
13 GetModificationDate;
14 ToPythonSparseFunction;
15
16 FirstOrderPerturbation;
17 SecondOrderPerturbation;
18
19 ToPythonSymPyExpression;
20
21 Begin["`Private`"];
22
23 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  $A + \beta$  perMatrix are to first order
equal to  $\lambda + \Delta_i \beta$ , where the  $\Delta_i$  are the
returned values. The eigenvalues and eigenvectors are assumed to
be given in the same order, i.e. the  $i$ th eigenvalue corresponds to
the  $i$ th eigenvector. This assuming that the eigenvalues are non-
degenerate.";
24 FirstOrderPerturbation[eigenValues_, eigenVectors_,
25   perMatrix_] := (Diagonal[
26     eigenVectors . perMatrix . Transpose[eigenVectors]])
27
28 SecondOrderPerturbation::usage="Given the eigenValues and eigenVectors
of a matrix A (which doesn't need to be given) together with a
corresponding perturbation matrix perMatrix, this function
calculates the second derivative of the eigenvalues with respect
to the scale factor of the perturbation matrix. In the sense that
the eigenvalues of the matrix  $A + \beta$  perMatrix are to second order
equal to  $\lambda + \Delta_i \beta + \Delta_i^{(2)} \beta^2 / 2$ , 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
 $i$ th eigenvalue corresponds to the  $i$ th eigenvector. This assuming
that the eigenvalues are non-degenerate.";
29 SecondOrderPerturbation[eigenValues_, eigenVectors_, perMatrix_] := (
30   dim = Length[perMatrix];
31   eigenBras = Conjugate[eigenVectors];
32   eigenKets = eigenVectors;
33   matV = Abs[eigenBras . perMatrix . Transpose[eigenKets]]^2;
34   OneOver[x_, y_] := If[x == y, 0, 1/(x - y)];
35   eigenDiffs = Outer[OneOver, eigenValues, eigenValues, 1];
36   pProduct = Transpose[eigenDiffs]*matV;
37   Return[2*(Total /@ Transpose[pProduct])];
38 )
39
40 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.";
41 SuperIdentity[args___] := {args};
42
43 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."
;
44 FlattenBasis[basis_] := Module[{flatbasis, flatbasisLS, flatbasisNums
},
45   (
46     flatbasis = Flatten[basis];
47     flatbasisLS = flatbasis[[1 ;; 4]];
48     flatbasisNums = Select[flatbasis, Not[StringQ[#]] &];
49     Return[{flatbasisLS, flatbasisNums}]
50   )
51 ];
52

```

```

53 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}.";
54 RecoverBasis[{flatbasisLS_, flatbasisNums_}] := Module[{recBasis},
55   (
56     recBasis = {{#[[1]], #[[2]]}, #[[3]], #[[4]]} & /@ (Flatten /@
57       Transpose[{flatbasisLS,
58         Partition[Round[2*#]/2 & /@ flatbasisNums, 3]]});
59     Return[recBasis];
60   )
61 ]
62
63 ExtractSymbolNames[expr_Hold] := Module[
64   {strSymbols},
65   strSymbols = ToString[expr, InputForm];
66   StringCases[strSymbols, RegularExpression["\\w+"]][[2 ;;]]
67 ]
68
69 ExportToH5::usage =
70   "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.
71   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.";
72 Options[ExportToH5] = {"Overwrite" -> True};
73 ExportToH5[fname_String, symbols_Hold, OptionsPattern[]] := (
74   If[And[FileExistsQ[fname], OptionValue["Overwrite"]],
75     (
76       Print["File already exists, overwriting ..."];
77       DeleteFile[fname];
78     )
79   ];
80   symbolNames = ExtractSymbolNames[symbols];
81   Do[(Print[symbolName];
82     Export[fname, ToExpression[symbolName], {"Datasets", symbolName},
83       OverwriteTarget -> "Append"]
84     ), {symbolName, symbolNames}]
85   )
86
87 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.";
88 Options[GreedyMatching] = {

```



```

89     "alistLabels" -> {},
90     "blistLabels" -> {}];
91 GreedyMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{
92     aValues = aValues0,
93     bValues = bValues0,
94     bValuesOriginal = bValues0,
95     bestLabels, bestMatches,
96     bestLabel, aElement, givenLabels,
97     aLabels, aLabel,
98     diffs, minDiff,
99     bLabels,
100     minDiffPosition, bestMatch},
101 (
102     aLabels      = OptionValue["alistLabels"];
103     bLabels      = OptionValue["blistLabels"];
104     bestMatches = {};
105     bestLabels   = {};
106     givenLabels = (Length[aLabels] > 0);
107     Do[
108         (
109             aElement      = aValues[[idx]];
110             diffs         = Abs[bValues - aElement];
111             minDiff       = Min[diffs];
112             minDiffPosition = Position[diffs, minDiff][[1, 1]];
113             bestMatch     = bValues[[minDiffPosition]];
114             bestMatches   = Append[bestMatches, {aElement, bestMatch}];
115             If[givenLabels,
116                 (
117                     aLabel      = aLabels[[idx]];
118                     bestLabel   = bLabels[[minDiffPosition]];
119                     bestLabels = Append[bestLabels, {aLabel, bestLabel}];
120                     bLabels     = Drop[bLabels, {minDiffPosition}];
121                 )
122             ];
123             bValues = Drop[bValues, {minDiffPosition}];
124             If[Length[bValues] == 0, Break[]];
125         ),
126         {idx, 1, Length[aValues]}
127     ];
128     pairedIndices = MapIndexed[{#2[[1]], Position[bValuesOriginal,
129 #1[[2]]][[1, 1]]} &, bestMatches];
130     If[givenLabels,
131         Return[{bestMatches, bestLabels, pairedIndices}],
132         Return[{bestMatches, pairedIndices}]
133     ]
134 ]
135
136 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.\";
137 Options[StochasticMatching] = {\"alistLabels\" -> {},
138 \"blistLabels\" -> {}};
139 StochasticMatching[aValues0_, bValues0_, numShuffles_ : 200,
OptionsPattern[]] := Module[{
140 aValues = aValues0,
141 bValues = bValues0,
142 matchingLabels, ranger, matches, noShuff, bestMatch, highestCost,
lowestCost, dev, sorter, bestValues,
143 pairedIndices, bestLabels, matchedIndices, shuffler
144 },
145 (
146 matchingLabels = (Length[OptionValue[\"alistLabels\"]] > 0);
147 ranger = Range[1, Length[aValues]];
148 matches = If[Not[matchingLabels], (
149 Table[{
150 shuffler = If[i == 1, ranger, RandomSample[ranger]];
151 {bestValues, matchedIndices} =
152 GreedyMatching[aValues[[shuffler]], bValues];
153 cost = Total[Abs#[[1]] - #[[2]]] & /@ bestValues];
154 {cost, {bestValues, matchedIndices}}
155 ), {i, 1, numShuffles}]
156 ),
157 Table[{
158 shuffler = If[i == 1, ranger, RandomSample[ranger]];
159 {bestValues, bestLabels, matchedIndices} =
160 GreedyMatching[aValues[[shuffler]], bValues,
161 \"alistLabels\" -> OptionValue[\"alistLabels\"][[shuffler]],
162 \"blistLabels\" -> OptionValue[\"blistLabels\"]];
163 cost = Total[Abs#[[1]] - #[[2]]] & /@ bestValues];
164 {cost, {bestValues, bestLabels, matchedIndices}}
165 ), {i, 1, numShuffles}]
166 ];
167 noShuff = matches[[1, 1]];
168 matches = SortBy[matches, First];
169 bestMatch = matches[[1, 2]];
170 highestCost = matches[[-1, 1]];
171 lowestCost = matches[[1, 1]];
172 dev = StandardDeviation[First /@ matches];
173 Print[lowestCost, \" <-> \", highestCost, \" | \[Sigma]=\", dev,
174 \" | N=\", numShuffles, \" | null=\", noShuff];
175 If[matchingLabels,
176 (
177 {bestValues, bestLabels, matchedIndices} = bestMatch;
178 sorter = Ordering[First /@ bestValues];
179 bestValues = bestValues[[sorter]];
180 bestLabels = bestLabels[[sorter]];
181 pairedIndices =
182 MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
183 bestValues];
184 Return[{bestValues, bestLabels, pairedIndices}]

```

```

185     ),
186     (
187         {bestValues, matchedIndices} = bestMatch;
188         sorter = Ordering[First /@ bestValues];
189         bestValues = bestValues[[sorter]];
190         pairedIndices =
191             MapIndexed[{#2[[1]], Position[bValues, #1[[2]]][[1, 1]]} &,
192                 bestValues];
193         Return[{bestValues, pairedIndices}]
194     )
195 ];
196 )
197 ]
198
199 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.";
200 Options[FlowMatching] = {"alistLabels" -> {}, "blistLabels" -> {}, "
notMatched" -> 0, "CostFun"-> (Abs[#1-#2] &)};
201 FlowMatching[aValues0_, bValues0_, OptionsPattern[]] := Module[{
202     aValues = aValues0, bValues = bValues0, edgesSourceToA,
203     capacitySourceToA, nA, nB,
204     costSourceToA, midLayer, midLayerEdges, midCapacities,
205     midCosts, edgesBtoSink, capacityBtoSink, costBtoSink,
206     allCapacities, allCosts, allEdges, graph,
207     flow, bestValues, bestLabels, cFun,
208     aLabels, bLabels, pairedIndices, matchingLabels},
209     (
210         matchingLabels = (Length[OptionValue["alistLabels"]] > 0);
211         aLabels = OptionValue["alistLabels"];
212         bLabels = OptionValue["blistLabels"];
213         cFun = OptionValue["CostFun"];
214         nA = Length[aValues];
215         nB = Length[bValues];
216         (*Build up the edges costs and capacities*)
217         (*From source to the nodes representing the values of the first \
list*)
218         edgesSourceToA = ("source" \[DirectedEdge] {"A", #}) & /@ Range[1,
nA];
219         capacitySourceToA = ConstantArray[1, nA];
220         costSourceToA = ConstantArray[0, nA];

```

```

221
222 (*From all the elements of A to all the elements of B*)
223 midLayer = Table[{{"A", i} \[DirectedEdge] {"B", j}}, 1, cFun[
224   aValues[[i]], bValues[[j]]], {i, 1, nA}, {j, 1, nB}];
225 midLayer = Flatten[midLayer, 1];
226 {midLayerEdges, midCapacities, midCosts} = Transpose[midLayer];
227
228 (*From the elements of B to the sink*)
229 edgesBtoSink = {"B", #} \[DirectedEdge] "sink" & /@ Range[1, nB];
230 capacityBtoSink = ConstantArray[1, nB];
231 costBtoSink = ConstantArray[0, nB];
232
233 (*Put it all together*)
234 allCapacities = Join[capacitySourceToA, midCapacities,
235   capacityBtoSink];
236 allCosts = Join[costSourceToA, midCosts, costBtoSink];
237 allEdges = Join[edgesSourceToA, midLayerEdges, edgesBtoSink];
238 graph = Graph[allEdges, EdgeCapacity -> allCapacities,
239   EdgeCost -> allCosts];
240
241 (*Solve it*)
242 flow = FindMinimumCostFlow[graph, "source", "sink", nA -
243   OptionValue["notMatched"], "OptimumFlowData"];
244 (*Collect the pairs of matched indices*)
245 pairedIndices = Select[flow["EdgeList"], And[Not[#[[1]] === "source"
246   ], Not[#[[2]] === "sink"]] &];
247 pairedIndices = {#[[1, 2]], #[[2, 2]]} & /@ pairedIndices;
248 (*Collect the pairs of matched values*)
249 bestValues = {aValues[[#[[1]]]], bValues[[#[[2]]]]} & /@
250   pairedIndices;
251 (*Account for having been given labels*)
252 If[matchingLabels,
253   (
254     bestLabels = {aLabels[[#[[1]]]], bLabels[[#[[2]]]]} & /@
255     pairedIndices;
256     Return[{bestValues, bestLabels, pairedIndices}]
257   ),
258   (
259     Return[{bestValues, pairedIndices}]
260   )
261 ];
262 ]
263
264 HelperNotebook::usage="HelperNotebook[nbName] creates a separate
265   notebook and returns a function that can be used to print to the
266   bottom of it. The name of the notebook, nbName, is optional and
267   defaults to OUT.";
268
269 HelperNotebook[nbName_:"OUT"] :=
270   Module[{screenDims, screenWidth, screenHeight, nbWidth, leftMargin,
271     PrintToOutputNb}, (
272     screenDims =
273       SystemInformation["Devices", "ScreenInformation"][[1, 2, 2]];
274     screenWidth = screenDims[[1, 2]];
275     screenHeight = screenDims[[2, 2]];

```

```

267 nbWidth = Round[screenWidth/3];
268 leftMargin = screenWidth - nbWidth;
269 outputNb = CreateDocument[{}, WindowTitle -> nbName,
270     WindowMargins -> {{leftMargin, Automatic}, {Automatic,
271     Automatic}}, WindowSize -> {nbWidth, screenHeight}];
272 PrintToOutputNb[text_] :=
273 (
274     SelectionMove[outputNb, After, Notebook];
275     NotebookWrite[outputNb, Cell[BoxData[ToBoxes[text]], "Output"
276     ]];
277     Return[PrintToOutputNb]
278 )
279 ]
280
281 GetModificationDate::usage="GetModificationDate[fname] returns the
282     modification date of the given file.";
283 GetModificationDate[theFileName_] := FileDate[theFileName, "
284     Modification"];
285
286 (*Helper function to convert Mathematica expressions to standard form
287 *)
288 StandardFormExpression[expr0_] := Module[{expr=expr0}, ToString[expr,
289     InputForm]];
290
291 (*Helper function to translate to Python/SymPy expressions*)
292 ToPythonSymPyExpression::usage="ToPythonSymPyExpression[expr] converts
293     a Mathematica expression to a SymPy expression. This is a little
294     iffy and might break if the expression includes Mathematica
295     functions that haven't been given a SymPy equivalent.";
296 ToPythonSymPyExpression[expr0_] := Module[{standardForm, expr=expr0},
297     standardForm = StandardFormExpression[expr];
298     StringReplace[standardForm, {
299         "Power[" -> "Pow(",
300         "Sqrt[" -> "sqrt(",
301         "[" -> "(",
302         "]" -> ")",
303         "\\\" -> "\"",
304         (*Remove special Mathematica backslashes*)
305         "/" -> "/" (*Ensure division is represented with a slash*)}]]];
306
307 ToPythonSparseFunction[sparseArray_SparseArray, funName_] :=
308     Module[{data, rowPointers, columnIndices, dimensions, pyCode, vars,
309         varList, dataPyList,
310         colIndicesPyList}, (*Extract unique symbolic variables from the \
311     SparseArray*)
312     vars = Union[Cases[Normal[sparseArray], _Symbol, Infinity]];
313     varList = StringRiffle[ToString /@ vars, ", "];
314     (*varList=ToPythonSymPyExpression/@varList;*)
315     (*Convert data to SymPy compatible strings*)
316     dataPyList =
317         StringRiffle[
318             ToPythonSymPyExpression /@ Normal[sparseArray["NonzeroValues"]],
319             ", "];
320     colIndicesPyList =

```

```

314 StringRiffle[
315   ToPythonSymPyExpression /@ (Flatten[
316     Normal[sparseArray["ColumnIndices"] - 1]), ", "];
317 (*Extract sparse array properties*)
318 rowPointers = Normal[sparseArray["RowPointers"]];
319 dimensions = Dimensions[sparseArray];
320 (*Create Python code string*)pyCode = StringJoin[
321   "#!/usr/bin/env python3\n\n",
322   "from scipy.sparse import csr_matrix\n",
323   "from sympy import *\n",
324   "import numpy as np\n",
325   "\n",
326   "sqrt = np.sqrt\n",
327   "\n",
328   "def ", funName, "(",
329   varList,
330   "):\n",
331   "    data = np.array([", dataPyList, "])\n",
332   "    indices = np.array([",
333   colIndicesPyList,
334   "])\n",
335   "    indptr = np.array([",
336   StringRiffle[ToString /@ rowPointers, ", ", "]\n",
337   "    shape = (", StringRiffle[ToString /@ dimensions, ", ", "],
338   ")\n",
339   "    return csr_matrix((data, indices, indptr), shape=shape)"];
340 pyCode
341 ];
342
343 End[];
344 EndPackage[];

```