

qlanth

January 29, 2024

```

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  $t_i$ .
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 + Judd, BR, and MA Suskin. "Complete Set of Orthogonal Scalar

71 Operators for the Configuration f^3 ." JOSA B 1, no. 2 (1984):

72 261-65. <https://doi.org/10.1364/JOSAB.1.000261>.

73

74 + Carnall, W. T., G. L. Goodman, K. Rajnak, and R. S. Rana. "A

75 Systematic Analysis of the Spectra of the Lanthanides Doped into

76 Single Crystal LaF_3 ." The Journal of Chemical Physics 90, no. 7

77 (1989): 3443-57. <https://doi.org/10.1063/1.455853>.

78

79 + Hansen, JE, BR Judd, and Hannah Crosswhite. "Matrix Elements of

80 Scalar Three-Electron Operators for the Atomic f-Shell." Atomic Data

81 and Nuclear Data Tables 62, no. 1 (1996): 1-49.

82 <https://doi.org/10.1006/adnd.1996.0001>.

83

84 + Velkov, Dobromir. "Multi-Electron Coefficients of Fractional

85 Parentage for the p, d, and f Shells." John Hopkins University,

86 2000.

87

88 + Dodson, Christopher M., and Rashid Zia. "Magnetic Dipole and

89 Electric Quadrupole Transitions in the Trivalent Lanthanide Series:

90 Calculated Emission Rates and Oscillator Strengths." Physical Review

91 B 86, no. 12 (September 5, 2012): 125102.

92 <https://doi.org/10.1103/PhysRevB.86.125102>.

93

94

95 ----- *)

96

97 `BeginPackage["qlanth"];`

98 `Needs["qconstants"];`

99 `Needs["qplotter"];`

100

101 `paramAtlas = "`

102 `E0: linear combination of F_k , see eqn. (2-80) in Wybourne 1965`

103 `E1: linear combination of F_k`

104 `E2: linear combination of F_k`

105 `E3: linear combination of F_k ,`

106

107 `ζ : spin-orbit strength parameter.`

108

109 `F0: Direct Slater integral F^0 , produces an overall shift of all energy levels.`

110 `F2: Direct Slater integral F^2`

111 `F4: Direct Slater integral F^4 , possibly constrained by ratio to F^2`

112 `F6: Direct Slater integral F^6 , possibly constrained by ratio to F^2`

```

113
114 M0: 0th Marvin integral
115 M2: 2nd Marvin integral
116 M4: 4th Marvin integral
117 \[Sigma]SS: spin-spin override, if 0 spin-spin omitted, and 1 if included
118
119 T2: three-body effective operator parameter T^2
120 T2p: three-body effective operator parameter T^2'
121 T3: three-body effective operator parameter T^3
122 T4: three-body effective operator parameter T^4
123 T6: three-body effective operator parameter T^6
124 T7: three-body effective operator parameter T^7
125 T8: three-body effective operator parameter T^8
126
127 T11: three-body effective operator parameter T^11
128 T11p: three-body effective operator parameter T^11'
129 T12: three-body effective operator parameter T^12
130 T14: three-body effective operator parameter T^14
131 T15: three-body effective operator parameter T^15
132 T16: three-body effective operator parameter T^16
133 T17: three-body effective operator parameter T^17
134 T18: three-body effective operator parameter T^18
135 T19: three-body effective operator parameter T^19
136
137 P0: 0th parameter for the two-body electrostatically correlated spin-orbit interaction
138 P2: 2nd parameter for the two-body electrostatically correlated spin-orbit interaction
139 P4: 4th parameter for the two-body electrostatically correlated spin-orbit interaction
140 P6: 6th parameter for the two-body electrostatically correlated spin-orbit interaction
141
142 gs: electronic gyromagnetic ratio
143
144  $\alpha$ : Trees' parameter  $\alpha$  describing configuration interaction via the Casimir operator of SO(3)
145  $\beta$ : Trees' parameter  $\beta$  describing configuration interaction via the Casimir operator of G(2)
146  $\gamma$ : Trees' parameter  $\gamma$  describing configuration interaction via the Casimir operator of SO(7)
147
148 B02: crystal field parameter B_0^2
149 B04: crystal field parameter B_0^4
150 B06: crystal field parameter B_0^6
151 B12: crystal field parameter B_1^2
152 B14: crystal field parameter B_1^4
153
154 B16: crystal field parameter B_1^6
155 B22: crystal field parameter B_2^2
156 B24: crystal field parameter B_2^4
157 B26: crystal field parameter B_2^6
158 B34: crystal field parameter B_3^4
159
160 B36: crystal field parameter B_3^6
161 B44: crystal field parameter B_4^4
162 B46: crystal field parameter B_4^6
163 B56: crystal field parameter B_5^6
164 B66: crystal field parameter B_6^6
165
166 S12: crystal field parameter S_1^2
167 S14: crystal field parameter S_1^4
168 S16: crystal field parameter S_1^6
169 S22: crystal field parameter S_2^2
170
171 S24: crystal field parameter S_2^4
172 S26: crystal field parameter S_2^6
173 S34: crystal field parameter S_3^4
174 S36: crystal field parameter S_3^6

```

```

175
176 S44: crystal field parameter S_4^4
177 S46: crystal field parameter S_4^6
178 S56: crystal field parameter S_5^6
179 S66: crystal field parameter S_6^6
180
181 \[Epsilon]: ground level baseline shift
182 t2Switch: controls the usage of the t2 operator beyond f7
183 wChErrA: If 1 then the type-A errors in Chen are used, if 0 then not.
184 wChErrB: If 1 then the type-B errors in Chen are used, if 0 then not.
185 ";
186 paramSymbols = StringSplit[paramAtlas, "\n"];
187 paramSymbols = Select[paramSymbols, # != "" & ];
188 paramSymbols = ToExpression[StringSplit[#, ":"][[1]]] & /@ paramSymbols;
189 Protect /@ paramSymbols;
190
191 (* Parameter families*)
192 cfSymbols = {B02, B04, B06, B12, B14, B16, B22, B24, B26, B34, B36,
193             B44, B46, B56, B66, S12, S14, S16, S22, S24, S26, S34, S36, S44,
194             S46, S56, S66};
195
196 TSymbols = {T2, T2p, T3, T4, T6, T7, T8, T11, T11p, T12, T14, T15, T16, T17, T18, T19};
197
198 AllowedJ;
199 AllowedMforJ;
200 AllowedNKSLJMforJMTerms;
201 AllowedNKSLJMforJTerms;
202
203 AllowedNKSLJTerms;
204 AllowedNKSLTerms;
205 AllowedNKSLforJTerms;
206 AllowedSLJMTerms;
207 AllowedSLJTerms;
208
209 AllowedSLTerms;
210 BasisLSJMJ;
211 Bqk;
212 CFP;
213 CFPAssoc;
214
215 CFPTable;
216 CFPTerms;
217 Carnall;
218 CasimirG2;
219 CasimirS07;
220
221 Cqk;
222 CrystalField;
223 Dk;
224 ElectrostaticConfigInteraction;
225 Electrostatic;
226
227 ElectrostaticTable;
228 EnergyLevelDiagram;
229 EnergyStates;
230 BasisTableGenerator;
231 EtoF;
232
233 FastIonSolverLaF3;
234 FindNKLSLTerm;
235 FindSL;
236

```

```

237 FtoE;
238 GG2U;
239 GS07W;
240 GenerateCFP;
241 GenerateCFPAassoc;
242
243 GenerateCFPTable;
244 GenerateCrystalFieldTable;
245 GenerateElectrostaticTable;
246 GenerateReducedUkTable;
247 GenerateReducedV1kTable;
248
249 GenerateS00andECS0LSTable;
250 GenerateS00andECS0Table;
251 GenerateSpinOrbitTable;
252 GenerateSpinSpinTable;
253 GenerateT22Table;
254
255 GenerateThreeBodyTables;
256 GenerateThreeBodyTablesUsingCFP;
257 Generator;
258 HamMatrixAssembly;
259 HamiltonianForm;
260
261 HamiltonianMatrixPlot;
262 HoleElectronConjugation;
263 IonSolverLaF3;
264 JJBBlockMatrix;
265 JJBBlockMatrixFileName;
266
267 JJBBlockMatrixTable;
268 LabeledGrid;
269 LoadAll;
270 LoadCFP;
271 LoadCarnall;
272
273 LoadChenDeltas;
274 LoadElectrostatic;
275 LoadGuillotParameters;
276 LoadParameters;
277 LoadS00andECS0;
278
279 LoadS00andECS0LS;
280 LoadSpinOrbit;
281 LoadSpinSpin;
282 LoadSymbolicHamiltonians;
283 LoadT11;
284
285 LoadT22;
286 LoadTermLabels;
287 LoadThreeBody;
288 LoadUk;
289 LoadVk1;
290
291 MagneticInteractions;
292 MaxJ;
293 MinJ;
294 NKCFPPPhase;
295
296 ParamPad;
297 ParseStates;
298 ParseStatesByNumBasisVecs;

```

```

299 ParseStatesByProbabilitySum;
300 ParseTermLabels;
301
302 Phaser;
303 PrettySaunders;
304 PrettySaundersSLJ;
305 PrettySaundersSLJmJ;
306 PrintL;
307
308 PrintSLJ;
309 PrintSLJM;
310 ReducedS00andECS0inf2;
311 ReducedS00andECS0infn;
312 ReducedT11inf2;
313
314 ReducedT22inf2;
315 ReducedUk;
316 ReducedUkTable;
317 ReducedV1kTable;
318 Reducedt11inf2;
319
320 ReplaceInSparseArray;
321 RobustMissingQ;
322 S00andECS0;
323 S00andECS0Table;
324 Seniority;
325
326 ShiftedLevels;
327 SixJay;
328 SpinOrbit;
329 SpinSpin;
330 SpinSpinTable;
331
332 Sqq;
333 SquarePrimeToNormal;
334 T11n;
335 T22n;
336 TP0;
337
338 TabulateJJBlockMatrixTable;
339 TabulateManyJJBlockMatrixTables;
340 TextBasedProgressBar;
341 ScalarOperatorProduct;
342 ThreeBodyTable;
343
344 ThreeBodyTables;
345 ThreeJay;
346 TotalCFIters;
347 chenDeltas;
348 fK;
349
350 fnTermLabels;
351 moduleDir;
352 symbolicHamiltonians;
353
354 (* this selects the function that is applied
355 to calculated matrix elements *)
356 SimplifyFun = Expand;
357
358 Begin["Private"]
359
360 moduleDir = DirectoryName[$InputFileName];

```

```

361 frontEndAvailable = (Head[$FrontEnd] === FrontEndObject);
362
363 (* ##### *)
364 (* ##### MISC ##### *)
365
366 RobustMissingQ[expr_] := (FreeQ[expr, _Missing] == False);
367
368 TP0::usage="Two plus one.";
369 TP0[args_] := Times @@ ((2*# + 1) & /@ {args});
370
371 Phaser::usage = "Phaser[x] returns (-1)^x";
372 Phaser[exponent_] := ((-1)^exponent);
373
374 TriangleCondition[a_, b_, c_] := (Abs[b - c] <= a <= (b + c));
375
376 TriangleAndSumCondition[a_, b_, c_] := (And[Abs[b - c] <= a <= (b + c), IntegerQ[a + b + c
  ]]);
377
378 TextBasedProgressBar[progress_, totalIterations_, prefix_:""] := Module[
379   {progMessage},
380   progMessage = ToString[progress] <> "/" <> ToString[totalIterations];
381   If[progress < totalIterations,
382     WriteString["stdout", StringJoin[prefix, progMessage, "\r"]],
383     WriteString["stdout", StringJoin[prefix, progMessage, "\n"]]]
384 ];
385 ];
386
387 SquarePrimeToNormal::usage = "Given a list with the parts corresponding to the squared
  prime representation of a number, this function parses the result into standard notation.
  ";
388 SquarePrimeToNormal[squarePrime_] :=
389 (
390   radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {idx1, 2, Length[squarePrime
  ]}];
391   radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" -> 13};
392   val = squarePrime[[1]] * Sqrt[radical];
393   Return[val];
394 );
395
396 ParamPad::usage = "ParamPad[params] takes an association params whose keys are a subset of
  paramSymbols. The function returns a new association where all the keys not present in
  paramSymbols, will now be included in the returned association with their values set to
  zero.
  The function additionally takes an option \"Print\" that if set to True, will print the
  symbols that were not present in the given association.";
397
398 Options[ParamPad] = {"Print" -> True}
399 ParamPad[params_, OptionsPattern[]] := (
400   notPresentSymbols = Complement[paramSymbols, Keys[params]];
401   If[OptionValue["Print"],
402     Print["Symbols not in given params: ",
403       notPresentSymbols]
404   ];
405   newParams = Transpose[{paramSymbols, ConstantArray[0, Length[paramSymbols]]}];
406   newParams = (#[[1]] -> #[[2]]) & /@ newParams;
407   newParams = Association[newParams];
408   newParams = Join[newParams, params];
409   Return[newParams];
410 )
411
412 (* ##### *)
413 (* ##### Racah Algebra ##### *)
414

```

```

415 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 Cowan (1981).";
416 ReducedUk[numE_, l_, SL_, SpLp_, k_] :=
417 Module[{orbital, Uk, S, L, Sp, Lp, Sb, Lb, parentSL, cfpSL, cfpSpLp, Ukval, SLparents,
SLpparents, commonParents, phase},
418 {spin, orbital} = {1/2, 3};
419 {S, L} = FindSL[SL];
420 {Sp, Lp} = FindSL[SpLp];
421 If[Not[S == Sp],
422 Return[0]
423 ];
424 cfpSL = CFP[{numE, SL}];
425 cfpSpLp = CFP[{numE, SpLp}];
426 SLparents = First /@ Rest[cfpSL];
427 SLpparents = First /@ Rest[cfpSpLp];
428 commonParents = Intersection[SLparents, SLpparents];
429 Uk = Sum[(
430 {Sb, Lb} = FindSL[\[Psi]b];
431 Phaser[Lb] *
432 CFPAssoc[{numE, SL, \[Psi]b}] *
433 CFPAssoc[{numE, SpLp, \[Psi]b}] *
434 SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
435 ),
436 {\[Psi]b, commonParents}
437 ];
438 phase = Phaser[orbital + L + k];
439 prefactor = numE * phase * Sqrt[TPO[L, Lp]];
440 Ukval = prefactor*Uk;
441 Return[Ukval];
442 ]
443
444 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 Cowan (1981) with l=1'.";
445 Ck[orbital_, k_] := (-1)^orbital * TPO[orbital] * ThreeJay[{orbital, 0}, {k, 0}, {orbital,
0}]
446
447 fk::usage = "Slater integral. See equation 12.17 in Cowan (1981).";
448 fk[numE_, orbital_, NKSL_, NKSLp_, k_] := Module[
449 {terms, S, L, Sp, Lp, termsWithSameSpin, SL, fkVal, spinMultiplicity,
450 prefactor, summand1, summand2},
451 {S, L} = FindSL[NKSL];
452 {Sp, Lp} = FindSL[NKSLp];
453 terms = AllowedNKSLTerms[numE];
454 (* sum for summand1 is over terms with same spin *)
455 spinMultiplicity = 2*S + 1;
456 termsWithSameSpin = StringCases[terms, ToString[spinMultiplicity] ~~ __];
457 termsWithSameSpin = Flatten[termsWithSameSpin];
458 If[Not[{S, L} == {Sp, Lp}],
459 Return[0]
460 ];
461 prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
462 summand1 = Sum[(
463 ReducedUkTable[{numE, orbital, SL, NKSL, k}] *
464 ReducedUkTable[{numE, orbital, SL, NKSLp, k}]
465 ),
466 {SL, termsWithSameSpin}
467 ];
468 summand1 = 1 / TPO[L] * summand1;
469 summand2 = (
470 KroneckerDelta[NKSL, NKSLp] *
471 (numE *(4*orbital + 2 - numE)) /
472 ((2*orbital + 1) * (4*orbital + 1))

```



```

473     );
474     fkVal = prefactor*(summand1 - summand2);
475     Return[fkVal];
476 ]
477
478 fK::usage = "Non-reduced Slater integral f^k = Subscript[f, k] * Subscript[D, k]";
479 fK[numE_, orbital_, NKSL_, NKSLp_, k_] := (Dk[k] * fk[numE, orbital, NKSL, NKSLp, k])
480
481 Dk::usage = "Ratio between the reduced and non-reduced Slater direct (Subscript[F, k] and F
    ^k) and exchange(Subscript[G, k] and G^k) integrals. Subscript[D, k] := (Subscript[G, k](ff
    ))/(G^k (ff)) = (Subscript[F, k](ff))/(F^k (ff)). k must be even. See table 6-3 in Cowan
    (1981), and also section 2-7 of Wybourne (1965). See also equation 6.41 in Cowan (1981).";
482 ;
483 Dk[k_] := {1, 225, 1089, 184041/25}[[k/2+1]]
484
485 FtoE::usage = "FtoE[F0, F2, F4, F6] calculates the corresponding {E0, E1, E2, E3} values.
    See eqn. 2-80 in Wybourne. Note that in that equation the subscripted Slater integrals are
    used but since this function assumes the the input values are superscripted Slater
    integrals, it is necessary to convert them using Dk.";
486 FtoE[F0_, F2_, F4_, F6_] := (Module[ (*Necessary here since Ei are protected.*)
487     {E0, E1, E2, E3},
488     E0 = (F0 - 10*F2/Dk[2] - 33*F4/Dk[4] - 286*F6/Dk[6]);
489     E1 = (70*F2/Dk[2] + 231*F4/Dk[4] + 2002*F6/Dk[6])/9;
490     E2 = (F2/Dk[2] - 3*F4/Dk[4] + 7*F6/Dk[6])/9;
491     E3 = (5*F2/Dk[2] + 6*F4/Dk[4] - 91*F6/Dk[6])/3;
492     Return[{E0, E1, E2, E3}];
493 ]
494 );
495
496 EtoF::usage = "EtoF[E0, E1, E2, E3] calculates the corresponding {F0, F2, F4, F6} values.
    The inverse of EtoF.";
497 EtoF[E0_, E1_, E2_, E3_] := (Module[ (*Necessary here since Fi are protected.*)
498     {F0, F2, F4, F6},
499     F0 = 1/7 (7 E0 + 9 E1);
500     F2 = 75/14 (E1 + 143 E2 + 11 E3);
501     F4 = 99/7 (E1 - 130 E2 + 4 E3);
502     F6 = 5577/350 (E1 + 35 E2 - 7 E3);
503     Return[{F0, F2, F4, F6}];
504 ]
505 );
506
507 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.";
508 SixJay[{j1_, j2_, j3_}, {j4_, j5_, j6_}] := (
509     sixJayval =
510     Which[
511     Not[TriangleAndSumCondition[j1, j2, j3]],
512     0,
513     Not[TriangleAndSumCondition[j1, j5, j6]],
514     0,
515     Not[TriangleAndSumCondition[j4, j2, j6]],
516     0,
517     Not[TriangleAndSumCondition[j4, j5, j3]],
518     0,
519     True,
520     SixJSymbol[{j1, j2, j3}, {j4, j5, j6}]];
521     SixJay[{j1, j2, j3}, {j4, j5, j6}] = sixJayval);
522
523 ThreeJay::usage = "ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] gives the value of the Wigner 3j-
    symbol and memorizes the computed value.";
524 ThreeJay[{j1_, m1_}, {j2_, m2_}, {j3_, m3_}] := (
525     threejval = Which[

```

```

526     Not[(m1 + m2 + m3) == 0],
527     0,
528     Not[TriangleCondition[j1,j2,j3]],
529     0,
530     True,
531     ThreeJSymbol[{j1, m1}, {j2, m2}, {j3, m3}]
532 ];
533 ThreeJay[{j1, m1}, {j2, m2}, {j3, m3}] = threejval);
534
535 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.";
536 ReducedV1k[numE_, SL_, SpLp_, k_] := Module[
537     {Vk1, S, L, Sp, Lp, Sb, Lb, spin, orbital, cfpSL, cfpSpLp,
538     SLparents, SpLpparents, commonParents, prefactor},
539     {spin, orbital} = {1/2, 3};
540     {S, L} = FindSL[SL];
541     {Sp, Lp} = FindSL[SpLp];
542     cfpSL = CFP[{numE, SL}];
543     cfpSpLp = CFP[{numE, SpLp}];
544     SLparents = First /@ Rest[cfpSL];
545     SpLpparents = First /@ Rest[cfpSpLp];
546     commonParents = Intersection[SLparents, SpLpparents];
547     Vk1 = Sum[(
548         {Sb, Lb} = FindSL[\[Psi]b];
549         Phaser[(Sb + Lb + S + L + orbital + k - spin)] *
550         CFPAssoc[{numE, SL, \[Psi]b}] *
551         CFPAssoc[{numE, SpLp, \[Psi]b}] *
552         SixJay[{S, Sp, 1}, {spin, spin, Sb}] *
553         SixJay[{L, Lp, k}, {orbital, orbital, Lb}]
554     ),
555     {\[Psi]b, commonParents}
556 ];
557 prefactor = numE * Sqrt[spin * (spin + 1) * TPO[spin, S, L, Sp, Lp] ];
558 Return[prefactor * Vk1];
559 ]
560
561 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.";
562 Options[GenerateReducedUkTable] = {"Export" -> True, "Progress" -> True};
563 GenerateReducedUkTable[numEmax_Integer:7, OptionsPattern[]]:= (
564     numValues = Total[Length[AllowedNKSLTerms[#]]*Length[AllowedNKSLTerms[#]]&/@Range[1,
numEmax]] * 4;
565     Print["Calculating " <> ToString[numValues] <> " values for Uk k=0,2,4,6."];
566     counter = 1;
567     If[And[OptionValue["Progress"], frontEndAvailable],
568     progBar = PrintTemporary[
569         Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
570             counter}]]]
571 ];
572 ReducedUkTable = Table[
573     (
574         counter = counter+1;
575         {numE, 3, SL, SpLp, k} -> SimplifyFun[ReducedUk[numE, 3, SL, SpLp, k]]
576     ),
577     {numE, 1, numEmax},
578     {SL, AllowedNKSLTerms[numE]},
579     {SpLp, AllowedNKSLTerms[numE]},
580     {k, {0, 2, 4, 6}}
581 ];
582 ReducedUkTable = Association[Flatten[ReducedUkTable]];

```

```

583 ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "ReducedUkTable.m"}];
584 If[And[OptionValue["Progress"], frontEndAvailable],
585   NotebookDelete[progBar]
586 ];
587 If[OptionValue["Export"],
588   (
589     Print["Exporting to file " <> ToString[ReducedUkTableFname]];
590     Export[ReducedUkTableFname, ReducedUkTable];
591   )
592 ];
593 Return[ReducedUkTable];
594 )
595
596 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."
597 Options[GenerateReducedVikTable] = {"Export" -> True, "Progress" -> True};
598 GenerateReducedVikTable[numEmax_Integer:7, OptionsPattern[]]:= (
599   numValues = Total[Length[AllowedNKSLTerms[#]]*Length[AllowedNKSLTerms[#]]&/@Range[1,
  numEmax]];
600   Print["Calculating " <> ToString[numValues] <> " values for Vk1."];
601   counter = 1;
602   If[And[OptionValue["Progress"], frontEndAvailable],
603     progBar = PrintTemporary[
604       Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
605         counter}]]]
606   ];
607   ReducedVikTable = Table[
608     (
609       counter = counter+1;
610       {n, SL, SpLp, 1} -> SimplifyFun[ReducedVik[n, SL, SpLp, 1]]
611     ),
612     {n, 1, numEmax},
613     {SL, AllowedNKSLTerms[n]},
614     {SpLp, AllowedNKSLTerms[n]}
615   ];
616   ReducedVikTable = Association[ReducedVikTable];
617   If[And[OptionValue["Progress"], frontEndAvailable],
618     NotebookDelete[progBar]
619   ];
620   exportFname = FileNameJoin[{moduleDir, "data", "ReducedVikTable.m"}];
621   If[OptionValue["Export"],
622     (
623       Print["Exporting to file " <> ToString[exportFname]];
624       Export[exportFname, ReducedVikTable];
625     )
626   ];
627   Return[ReducedVikTable];
628 )
629
630 (* ##### Racah Algebra ##### *)
631 (* ##### *)
632
633 (* ##### *)
634 (* ##### Electrostatic ##### *)
635
636 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).";
637 Electrostatic[numE_, NKSL_, NKSLp_] := Module[
638   {f0, f2, f4, f6, e0, e1, e2, e3, eMatrixVal, orbital},

```

```

639     orbital = 3;
640     Ek = {E0, E1, E2, E3};
641     f0 = fK[numE, orbital, NKSL, NKSLp, 0];
642     f2 = fK[numE, orbital, NKSL, NKSLp, 2];
643     f4 = fK[numE, orbital, NKSL, NKSLp, 4];
644     f6 = fK[numE, orbital, NKSL, NKSLp, 6];
645     e0 = f0;
646     e1 = 9/7*f0 + f2/42 + f4/77 + f6/462;
647     e2 = 143/42*f2 - 130/77*f4 + 35/462*f6;
648     e3 = 11/42*f2 + 4/77*f4 - 7/462*f6;
649     eMatrixVal = e0*E0 + e1*E1 + e2*E2 + e3*E3;
650     Return[eMatrixVal];
651 ]
652
653 GenerateElectrostaticTable::usage = "GenerateElectrostaticTable[numE] can be used to
generate the table for the electrostatic interaction from f1 to fnumE. If the option
\"Export\" is set to True then the resulting data is saved to ./data/ElectrostaticTable.
m.";
654 Options[GenerateElectrostaticTable] = {"Export" -> True};
655 GenerateElectrostaticTable[numE_Integer:7, OptionsPattern[]]:= (
656     ElectrostaticTable = Table[
657         {numE, SL, SpLp} -> SimplifyFun[Electrostatic[numE, SL, SpLp]],
658         {numE, 1, numE},
659         {SL, AllowedNKSLTerms[numE]},
660         {SpLp, AllowedNKSLTerms[numE]}
661     ];
662     ElectrostaticTable = Association[Flatten[ElectrostaticTable]];
663     If[OptionValue["Export"],
664         Export[FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}],
665             ElectrostaticTable];
666     ];
667     Return[ElectrostaticTable];
668 )
669
670 (* ##### Electrostatic ##### *)
671 (* ##### *)
672
673 (* ##### *)
674 (* ##### Bases ##### *)
675
676 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}.";
677 BasisTableGenerator[numE_] := Module[{energyStatesTable}, (
678     energyStatesTable = <||>;
679     allowedJ = AllowedJ[numE];
680     Do[
681         (
682             energyStatesTable[{numE, J}] = EnergyStates[numE, J];
683         ),
684         {Jp, allowedJ},
685         {J, allowedJ}];
686     Return[energyStatesTable]
687 )
688 ];
689
690 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}.";
691 BasisLSJMJ[numE_] := Module[{energyStatesTable, basis, idx1},
692     (

```

```

693     energyStatesTable = BasisTableGenerator[numE];
694     basis = Table[
695         energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
696         {idx1, 1, Length[AllowedJ[numE]]}];
697     basis = Flatten[basis, 1];
698     Return[basis]
699 )
700 ];
701
702 (* ##### Bases ##### *)
703 (* ##### *)
704
705 (* ##### *)
706 (* ##### Coefficients of Fractional Parentage ##### *)
707
708 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.";
709 Options[GenerateCFP] = {"Export" -> True, "PhaseFunction" -> "NK"};
710 GenerateCFP[OptionsPattern[]]:= (
711     CFP = Table[
712         {numE, NKSL} -> First[CFPTerms[numE, NKSL]],
713         {numE, 1, 7},
714         {NKSL, AllowedNKSLTerms[numE]}];
715     CFP = Association[CFP];
716     (* Go all the way to f14 *)
717     CFP = CFPExpander["Export" -> False, "PhaseFunction" -> OptionValue["PhaseFunction"]];
718     If[OptionValue["Export"],
719         Export[FileNameJoin[{moduleDir, "data", "CFPs.m"}], CFP];
720 ];
721     Return[CFP];
722 )
723
724 JuddCFPPPhase::usage="Phase between conjugate coefficients of fractional parentage according
to Velkov's thesis, page 40.";
725 JuddCFPPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_, parentSeniority_,
daughterSeniority_] := (
726     {spin, orbital} = {1/2, 3};
727     expo = (
728         (parentS + parentL + daughterS + daughterL) -
729         (orbital + spin) +
730         1/2 * (parentSeniority + daughterSeniority - 1)
731     );
732     phase = Phaser[-expo];
733     Return[phase];
734 )
735
736 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).";
737 NKCFPPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_, parentSeniority_,
daughterSeniority_] := (
738     {spin, orbital} = {1/2, 3};
739     expo = (
740         (parentS + parentL + daughterS + daughterL) -
741         (orbital + spin)
742     );
743     phase = Phaser[-expo];
744     If[parent == 2*orbital,

```

```

745     phase = phase * Phaser[(daughterSeniority-1)/2]];
746     Return[phase];
747 )
748
749 Options[CFPExpander] = {"Export" -> True, "PhaseFunction" -> "NK"};
750 CFPExpander::usage="Using the coefficients of fractional parentage up to f7 this function
    calculates them up to f14.
751
752 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.";
753 CFPExpander[OptionsPattern[]]:=
754     orbital      = 3;
755     halfFilled   = 2 * orbital + 1;
756     fullShell    = 2 * halfFilled;
757     parentMax    = 2 * orbital;
758
759     PhaseFun     = <|
760         "Judd" -> JuddCFPPPhase,
761         "NK"  -> NKCFPPPhase|>[OptionValue["PhaseFunction"]];
762     PrintTemporary["Calculating CFPs using the phase system from ", PhaseFun];
763     (* Initialize everything with lists to be filled in the next Do*)
764     complementaryCFPs =
765         Table[
766             ({numE, term} -> {term}),
767             {numE, halfFilled + 1, fullShell - 1, 1},
768             {term, AllowedNKSLTerms[numE]}
769         ];
770     complementaryCFPs = Association[Flatten[complementaryCFPs]];
771     Do[(
772         daughter      = parent + 1;
773         conjugateDaughter = fullShell - parent;
774         conjugateParent = conjugateDaughter - 1;
775         parentTerms    = AllowedNKSLTerms[parent];
776         daughterTerms  = AllowedNKSLTerms[daughter];
777         Do[
778             (
779                 parentCFPs      = Rest[CFP[{daughter, daughterTerm}]];
780                 daughterSeniority = Seniority[daughterTerm];
781                 {daughterS, daughterL} = FindSL[daughterTerm];
782                 Do[
783                     (
784                         {parentTerm, parentCFPval} = parentCFP;
785                         {parentS, parentL}        = FindSL[parentTerm];
786                         parentSeniority            = Seniority[parentTerm];
787                         phase = PhaseFun[parent, parentS, parentL,
788                             daughterS, daughterL,
789                             parentSeniority, daughterSeniority];
790                         prefactor = (daughter * TPO[daughterS, daughterL]) /
791                             (conjugateDaughter * TPO[parentS, parentL]);
792                         prefactor = Sqrt[prefactor];
793                         newCFPval = phase * prefactor * parentCFPval;
794                         key = {conjugateDaughter, parentTerm};
795                         complementaryCFPs[key] = Append[complementaryCFPs[key], {daughterTerm,
newCFPval}]]
796                     ),
797                     {parentCFP, parentCFPs}
798                 ]
799             ),
800             {daughterTerm, daughterTerms}

```

```

801     ]
802   ),
803   {parent, 1, parentMax}
804 ];
805
806 complementaryCFPs[{14, "1S"}] = {"1S", {"2F", 1}};
807 extendedCFPs = Join[CFP, complementaryCFPs];
808 If[OptionValue["Export"],,
809 (
810   exportFname = FileNameJoin[{moduleDir, "data", "CFPs_extended.m"}];
811   Print["Exporting to ", exportFname];
812   Export[exportFname, extendedCFPs];
813 )
814 ];
815 Return[extendedCFPs];
816 )
817
818 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";
819 Options[GenerateCFPTable] = {"Export" -> True};
820 GenerateCFPTable[OptionsPattern[]]:= (
821   CFPtextData = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
822   fConfigs = StringSplit[CFPtextData, "[ONE PARTICLE FRACTIONAL PARENTAGE COEFFICIENTS]";
823   CFPTable = {};
824
825   (* This table parses the text file with the one-body coefficients of fractional parentage
*)
826   CFPTable = Table[
827   (
828     fNx = StringReplace[Part[fConfigs, idx1], "-" -> " -"];
829     daughterLabelSpots = StringPosition[fNx,
830       Shortest[StartOfLine ~~ DigitCharacter ~~ LetterCharacter ~~ ___ ~~ "[",
831       Overlaps -> False];
832     daughterLabels = Map[StringDrop[#, -1] &, StringTake[fNx, daughterLabelSpots]];
833     daughterLabelLines = StringPosition[fNx,
834       Shortest[StartOfLine ~~ DigitCharacter ~~ LetterCharacter ~~ __ ~~
835       EndOfLine], Overlaps -> False];
836     startDaughters = Map[Last, daughterLabelLines + 2];
837     stopDaughters = Delete[Append[Map[First, daughterLabelLines - 2], StringLength[fNx]],
838     1];
839     daughterLines = Join[Partition[startDaughters, 1], Partition[stopDaughters, 1], 2];
840     testing = Map[StringSplit,
841       StringSplit[StringTake[fNx, daughterLines], EndOfLine]];
842     testing2 = Map[DeleteCases[#, {}] &, testing];
843     ToIntegerOrString[list_] := Map[If[StringMatchQ[#, NumberString], ToExpression[#, #]
&, list];
844     CFPs = Table[(
845       tt = Part[testing2, mm];
846       pLabels = Map[Extract[#, 1] &, tt];
847       pValues = Map[SquarePrimeToNormal, Map[ToIntegerOrString[Drop[#, 2]] &, tt]];
848       Join[Partition[pLabels, 1], Partition[pValues, 1], 2]
849     ),
850     {mm, 1, Length[testing2]}
851   ];
852   CFPconfig = Join[Partition[daughterLabels, 1], CFPs, 2];
853   CFPconfig
854 ),
855 {idx1, 2, 7}
856 ];
857 CFPTable = Join[{{{ "2F", {"1S", 1}}}}, CFPTable];
858 If[OptionValue["Export"],

```

```

858     (
859     CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
860     Export[CFPTablefname, CFPTable];
861     )
862 ];
863 Return[CFPTable];
864 )
865
866 GenerateCFPAAssoc::usage = "GenerateCFPAAssoc[export] converts the coefficients of fractional
      parentage into an association in which zero values are explicit. If \"Export\" is set to
      True, the association is exported to the file /data/CFPAAssoc.m.";
867 Options[GenerateCFPAAssoc] = {"Export" -> True};
868 GenerateCFPAAssoc[OptionsPattern[]]:= (
869     CFPAAssoc = Association[];
870     Do[
871         (daughterTerms = AllowedNKSLTerms[numE];
872          parentTerms    = AllowedNKSLTerms[numE - 1];
873          Do[
874              (
875              cfps = CFP[{numE, daughter}];
876              cfps = cfps[[2 ;;]];
877              parents = First /@ cfps;
878              Do[
879                  (
880                  key = {numE, daughter, parent};
881                  cfp = If[
882                      MemberQ[parents, parent],
883                      (
884                          idx = Position[parents, parent][[1, 1]];
885                          cfps[[idx]][[2]]
886                      ),
887                      0
888                  ];
889                  CFPAAssoc[key] = cfp;
890                  ),
891                  {parent, parentTerms}
892              ]
893          ),
894          {daughter, daughterTerms}
895      ]
896     ),
897     {numE, 1, 14}
898 ];
899 If[OptionValue["Export"],
900     (
901     CFPAAssocfname = FileNameJoin[{moduleDir, "data", "CFPAAssoc.m"}];
902     Export[CFPAAssocfname, CFPAAssoc];
903     )
904 ];
905 Return[CFPAAssoc];
906 )
907
908 CFPTerms::usage = "CFPTerms[numE] gives all the daughter and parent terms, together with
      the corresponding coefficients of fractional parentage, that correspond to the the f^n
      configuration.
909
910 CFPTerms[numE, SL] gives all the daughter and parent terms, together with the corresponding
      coefficients of fractional parentage, that are compatible with the given string SL in
      the f^n configuration.
911
912 CFPTerms[numE, L, S] gives all the daughter and parent terms, together with the
      corresponding coefficients of fractional parentage, that correspond to the given total

```


orbital angular momentum L and total spin S in the f^n configuration. L being an integer, and S being integer or half-integer.

In all cases the output is in the shape of a list with enclosed lists having the format {daughter_term, {parent_term_1, CFP_1}, {parent_term_2, CFP_2}, ...}.

Only the one-body coefficients for f -electrons are provided.

In all cases it must be that $1 \leq n \leq 7$.

";

CFPTerms[numE_] := Part[CFPTable, numE]

CFPTerms[numE_, SL_] :=

Module[

{NKterms, CFPconfig},

NKterms = {{}};

CFPconfig = Part[CFPTable, numE];

Map[

If[StringFreeQ[First[#], SL],

Null,

NKterms = Join[NKterms, {#}, 1]

] &,

CFPconfig

];

NKterms = DeleteCases[NKterms, {}]

]

CFPTerms[numE_, L_, S_] :=

Module[

{NKterms, SL, CFPconfig},

SL = StringJoin[ToString[2 S + 1], PrintL[L]];

NKterms = {{}};

CFPconfig = Part[CFPTable, numE];

Map[

If[StringFreeQ[First[#], SL],

Null,

NKterms = Join[NKterms, {#}, 1]

] &,

CFPconfig

];

NKterms = DeleteCases[NKterms, {}]

]

(* ##### Coefficients of Fractional Parentage ##### *)

(* ##### *)

(* ##### *)

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

SpinOrbit::usage = "SpinOrbit[numE, SL, SpLp, J] returns the LSJ reduced matrix element $\zeta \langle SL, J | L.S | SpLp, J \rangle$. These are given as a function of ζ . This function requires that the association ReducedV1kTable be defined.";

SpinOrbit[numE_, SL_, SpLp_, J_] := Module[

{S, L, Sp, Lp, orbital, sign, prefact},

orbital = 3;

{S, L} = FindSL[SL];

{Sp, Lp} = FindSL[SpLp];

prefact = Sqrt[orbital*(orbital+1)*(2*orbital+1)] * SixJay[{L, Lp, 1}, {Sp, S, J}];

sign = Phaser[J + L + Sp];

Return[sign * prefact * ζ * ReducedV1kTable[{numE, SL, SpLp, 1}]];

]

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

```

967   this package is in. It requires ReducedV1kTable to be defined.";
GenerateSpinOrbitTable[nmax_:7, export_:False]:= (
968   SpinOrbitTable =
969     Table[
970       {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
971       {numE, 1, nmax},
972       {J, MinJ[numE], MaxJ[numE]},
973       {SL, Map[First, AllowedNKSLforJTerms[numE, J]]},
974       {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
975     ];
976   SpinOrbitTable = Association[SpinOrbitTable];
977
978   exportFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
979   If[export,
980     (
981       Print["Exporting to file "<>ToString[exportFname]];
982       Export[exportFname, SpinOrbitTable];
983     )
984   ];
985   Return[SpinOrbitTable];
986 )
987
988 (* ##### Spin Orbit ##### *)
989 (* ##### *)
990
991 (* ##### *)
992 (* ##### Three Body Operators ##### *)
993
994 Options[ParseJudd1984] = {"Export" -> False};
995 ParseJudd1984::usage="This function parses the data from tables 1 and 2 of Judd from Judd,
BR, and MA Suskin. \"Complete Set of Orthogonal Scalar Operators for the Configuration f
^3\". JOSA B 1, no. 2 (1984): 261-65.\"";
996 ParseJudd1984[OptionsPattern[]]:= (
997   export = OptionValue["Export"];
998   ParseJuddTab1[str_] := (
999     strR = ToString[str];
1000     strR = StringReplace[strR, ".5" -> "^(1/2)"];
1001     num = ToExpression[strR];
1002     sign = Sign[num];
1003     num = sign*Simplify[Sqrt[num^2]];
1004     If[Round[num] == num, num = Round[num]];
1005     Return[num]);
1006
1007   (* Parse table 1 from Judd 1984 *)
1008   judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"}];
1009   data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
1010   headers = data[[1]];
1011   data = data[[2 ;;]];
1012   data = Transpose[data];
1013   \[Psi] = Select[data[[1]], # != "" &];
1014   \[Psi]p = Select[data[[2]], # != "" &];
1015   matrixKeys = Transpose[{\[Psi], \[Psi]p}];
1016   data = data[[3 ;;]];
1017   cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data}];
1018   cols = Select[cols, Length[#] == 21 &];
1019   tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
1020   tab1 = Transpose[Prepend[Transpose[tab1], headers]];
1021
1022   (* Parse table 2 from Judd 1984 *)
1023   judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"}];
1024   data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
1025   headers = data[[1]];

```

```

1026 data = data[[2 ;;]];
1027 data = Transpose[data];
1028 {operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} = data[[;; 4]];
1029 multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
1030 multiFactorValues = AssociationThread[multiFactorSymbols -> multiFactorValues];
1031
1032 (*scale values of table 1 given the values in table 2*)
1033 oppyS = {};
1034 normalTable =
1035   Table[header = col[[1]];
1036     If[StringContainsQ[header, " "],
1037       (
1038         multiplierSymbol = StringSplit[header, " "][[1]];
1039         multiplierValue = multiFactorValues[multiplierSymbol];
1040         operatorSymbol = StringSplit[header, " "][[2]];
1041         oppyS = Append[oppyS, operatorSymbol];
1042       ),
1043       (
1044         multiplierValue = 1;
1045         operatorSymbol = header;
1046       )
1047     ];
1048     normalValues = 1/multiplierValue*col[[2 ;;]];
1049     Join[{operatorSymbol}, normalValues], {col, tab1[[3 ;;]]}
1050   ];
1051
1052 (*Create an association for the matrix elements in the f^3 config*)
1053 juddOperators = Association[];
1054 Do[(
1055   col      = normalTable[[colIndex]];
1056   opLabel  = col[[1]];
1057   opValues = col[[2 ;;]];
1058   opMatrix = AssociationThread[matrixKeys -> opValues];
1059   Do[(
1060     opMatrix[Reverse[mKey]] = opMatrix[mKey]
1061   ),
1062     {mKey, matrixKeys}
1063   ];
1064   juddOperators[{3, opLabel}] = opMatrix,
1065   {colIndex, 1, Length[normalTable]}
1066 ];
1067
1068 (* special case of t2 in f3 *)
1069 (* this is the same as getting the matrix elements from Judd 1966 *)
1070 numE = 3;
1071 e30p = juddOperators[{3, "e_{3}"}];
1072 t2prime = juddOperators[{3, "t_{2}^{'}"}];
1073 prefactor = 1/(70 Sqrt[2]);
1074 t20p = (# -> (t2prime[#] + prefactor*e30p[#])) & /@ Keys[t2prime];
1075 t20p = Association[t20p];
1076 juddOperators[{3, "t_{2}"}] = t20p;
1077
1078 (*Special case of t11 in f3*)
1079 t11 = juddOperators[{3, "t_{11}"}];
1080 eβprimeOp = juddOperators[{3, "e_{\\beta}^{'}"}];
1081 t11primeOp = (# -> (t11[#] + Sqrt[3/385] eβprimeOp[#])) & /@ Keys[t11];
1082 t11primeOp = Association[t11primeOp];
1083 juddOperators[{3, "t_{11}^{'}"}] = t11primeOp;
1084 If[export,
1085   (
1086     (*export them*)
1087     PrintTemporary["Exporting ..."];

```

```

1088     exportFname = FileNameJoin[{moduleDir, "data", "juddOperators.m"}];
1089     Export[exportFname, juddOperators];
1090 )
1091 ];
1092 Return[juddOperators];
1093 )
1094
1095 GenerateThreeBodyTables::usage = "GenerateThreeBodyTables[nmax] computes the LS reduced
    matrix elements for the three-body operators in f^n configurations up to n = nmax. The
    function returns an association whose keys are lists of the form {n, SL, SpLp}. By
    default the resulting data is not exported to disk, to do so set the option \"Export\" to
    True.";
1096 Options[GenerateThreeBodyTables] = {"Export" -> False};
1097 GenerateThreeBodyTables[nmax_ : 7, OptionsPattern[]] := (
1098     tiKeys = {"t_{2}"          , "t_{2}^{'}",
1099             "t_{3}"          ,
1100             "t_{4}"          , "t_{6}"          ,
1101             "t_{7}"          , "t_{8}"          ,
1102             "t_{11}"         , "t_{11}^{'}",
1103             "t_{12}"         ,
1104             "t_{14}"         , "t_{15}"         ,
1105             "t_{16}"         , "t_{17}"         ,
1106             "t_{18}"         , "t_{19}"};
1107     TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
1108     juddOperators = ParseJudd1984[];
1109
1110 op3MatrixElement::usage =
1111 "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.
1112
1113 Data used here was taken from tables 1 and 2 from Judd, BR, and MA Suskin. \"Complete Set
    of Orthogonal Scalar Operators for the Configuration f^3\". JOSA B 1, no. 2 (1984):
    261-65.\"";
1114 op3MatrixElement[SL_, SpLp_, opSymbol_] := (
1115     jOP = juddOperators[{3, opSymbol}];
1116     key = {SL, SpLp};
1117     val = If[MemberQ[Keys[jOP], key],
1118             jOP[key],
1119             0];
1120     Return[val];
1121 );
1122
1123 ti::usage = "This is the implementation of formula (2) in Judd & Suskin 1984.";
1124 ti[n_, SL_, SpLp_, tiKey_, opOrder_:3] := Module[
1125     {nn, S, L, Sp, Lp, cfpSL, cfpSpLp, parentSL, parentSpLp, tnk, tnks},
1126     {S, L} = FindSL[SL];
1127     {Sp, Lp} = FindSL[SpLp];
1128     If[S == Sp && L == Lp,
1129     (
1130         cfpSL = CFP[{n, SL}];
1131         cfpSpLp = CFP[{n, SpLp}];
1132         tnks = Table[(
1133             parentSL = cfpSL[[nn, 1]];
1134             (
1135                 cfpSL[[nn, 2]] *
1136                 cfpSpLp[[mm, 2]] *
1137                 tktable[{n - 1, parentSL, cfpSpLp[[mm, 1]], tiKey}]
1138             )
1139         ),
1140         {nn, 2, Length[cfpSL]},
1141         {mm, 2, Length[cfpSpLp]}
1142     ];

```

```

1143     tnk = Total[Flatten[tnks]];
1144   ),
1145   tnk = 0;
1146 ];
1147 Return[ n / (n - opOrder) * tnk];
1148 ];
1149
1150 (*Calculate the matrix elements of t^i for n up to 7*)
1151 tktable = <||>;
1152 Do[
1153   (
1154     Do[
1155       (
1156         tkValue =
1157           Which[
1158             numE <= 2, (* Initialize n=1,2 with zeros *)
1159             0,
1160             numE == 3, (* Grab matrix elem in f^3 from Judd 1984 *)
1161             SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
1162             True,
1163             SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2, 3]]]
1164           ];
1165         tktable[{numE, SL, SpLp, opKey}] = tkValue;
1166       ),
1167       {SL, AllowedNKSLTerms[numE]},
1168       {SpLp, AllowedNKSLTerms[numE]},
1169       {opKey, Append[tiKeys, "e_{3}"]}
1170     ];
1171     PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " configuration complete"]];
1172   ),
1173   {numE, 1, nmax}
1174 ];
1175 (*Now use those matrix elements to determine their sum as weighted by their corresponding
1176 strengths Ti*)
1177 partialTiKeys = {"t_{2}~{'",
1178   "t_{3}",
1179   "t_{4}"      , "t_{6}",
1180   "t_{7}"      , "t_{8}",
1181   "t_{11}"     , "t_{11}~{'",
1182   "t_{12}",
1183   "t_{14}"     , "t_{15}",
1184   "t_{16}"     , "t_{17}",
1185   "t_{18}"     , "t_{19}"};
1186 ThreeBodyTable = <||>;
1187 Do[
1188   Do[(
1189     flipSign = -1;
1190     holeElectronSelector = ((1 + isE)/2 + (1 - isE)/2 * flipSign);
1191     ThreeBodyTable[{numE, SL, SpLp}] =
1192       (
1193         holeElectronSelector *
1194         Sum[(tktable[{numE, SL, SpLp, tiKey}] * TSymbolsAssoc[tiKey]),
1195           {tiKey, partialTiKeys}
1196         ]
1197       );
1198     {SL, AllowedNKSLTerms[numE]},
1199     {SpLp, AllowedNKSLTerms[numE]}
1200   ];
1201   PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix complete"]];,
1202   {numE, 1, nmax}
1203 ];

```

```

1204
1205 (* Calculate the matrix elements of t2 in f^n*)
1206 LoadUk[];
1207 Do[(
1208     terms = AllowedNKSLTerms[numE];
1209     Do[
1210         (
1211             electro = Electrostatic[numE, term1, term2];
1212             electro = electro /. {E0 -> 0, E1 -> 0, E2 -> 0, E3 -> 1};
1213             prefactor = (
1214                 (numE - 2)/(70 Sqrt[2]) (1 + isE)/2 +
1215                 ((14 - numE) - 2)/(70 Sqrt[2]) (1 - isE)/2
1216             );
1217             prefactor = Simplify[prefactor];
1218             braSeniority = Seniority[SL];
1219             ketSeniority = Seniority[SpLp];
1220             onePlus\[CapitalDelta]vHalf = Simplify[1 + (braSeniority - ketSeniority)/2];
1221             flipSign = If[EvenQ[onePlus\[CapitalDelta]vHalf], 1, -1];
1222             holeElectronSelector = ((1 + isE)/2 + (1 - isE)/2 * flipSign);
1223             t2primeVal = holeElectronSelector*tktable[{numE, term1, term2, "t_{2}~{'}"}];
1224             t2value = If[numE==2,
1225                 prefactor * electro,
1226                 prefactor * electro + t2primeVal
1227             ];
1228             t2value = T2 * Simplify[t2value];
1229             ThreeBodyTable[{numE, term1, term2}] += t2value;
1230         ),
1231         {term1, terms},
1232         {term2, terms}
1233     ];
1234 ),
1235 {numE, 2, nmax}
1236 ];
1237
1238 ThreeBodyTables =
1239 Table[
1240     (
1241         terms = AllowedNKSLTerms[numE];
1242         singleThreeBodyTable =
1243             Table[
1244                 {SL, SLp} -> ThreeBodyTable[{numE, SL, SLp}],
1245                 {SL, terms},
1246                 {SLp, terms}
1247             ];
1248         singleThreeBodyTable = Flatten[singleThreeBodyTable];
1249         singleThreeBodyTables =
1250             Table[(
1251                 notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
1252                 reps = ConstantArray[0, Length[TSymbols]];
1253                 reps[[notNullPosition]] = 1;
1254                 rep = AssociationThread[TSymbols -> reps];
1255                 notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
1256             ),
1257             {notNullSymbol, TSymbols}
1258         ];
1259         singleThreeBodyTables = Association[singleThreeBodyTables];
1260         numE -> singleThreeBodyTables
1261     ),
1262     {numE, 1, nmax}
1263 ];
1264 ThreeBodyTables = Association[ThreeBodyTables];
1265

```

```

1266 If[OptionValue["Export"],
1267 (
1268     threeBodyTablefname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
1269     Export[threeBodyTablefname, ThreeBodyTable];
1270     threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
1271     Export[threeBodyTablesfname, ThreeBodyTables];
1272 )
1273 ];
1274 Return[{ThreeBodyTable, ThreeBodyTables}];
1275 )
1276
1277 Options[GenerateThreeBodyTablesUsingCFP] = {"Export" -> False};
1278 GenerateThreeBodyTablesUsingCFP::usage="This function generates the matrix elements for the
    three body operators using the coefficients of fractional parentage, including those
    beyond f^7.";
1279 GenerateThreeBodyTablesUsingCFP[nmax_Integer : 14, OptionsPattern[]] := (
1280     tiKeys = {"t_{2}", "t_{2}^{'}", "t_{3}", "t_{4}", "t_{6}", "t_{7}",
1281         "t_{8}", "t_{11}", "t_{11}^{'}", "t_{12}", "t_{14}", "t_{15}",
1282         "t_{16}", "t_{17}", "t_{18}", "t_{19}"};
1283     TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
1284     juddOperators = ParseJudd1984[];
1285     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.";
1286     op3MatrixElement[SL_, SpLp_, opSymbol_] := (
1287         jOP = juddOperators[{3, opSymbol}];
1288         key = {SL, SpLp};
1289         val = If[MemberQ[Keys[jOP], key],
1290             jOP[key],
1291             0];
1292         Return[val];
1293     );
1294     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.";
1295     Options[ti] = {"Fast" -> True};
1296     ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
1297         Module[{nn, S, L, Sp, Lp,
1298             cfpSL, cfpSpLp,
1299             parentSL, parentSpLp, tnk, tnks},
1300             {S, L} = FindSL[SL];
1301             {Sp, Lp} = FindSL[SpLp];
1302             fast = OptionValue["Fast"];
1303             numH = 14 - nE;
1304             If[fast && Not[MemberQ[{"t_{2}", "t_{11}"}, tiKey]] && nE > 7,
1305                 Return[-tktable[{numH, SL, SpLp, tiKey}]]
1306             ];
1307             If[(S == Sp && L == Lp),
1308                 (
1309                     cfpSL = CFP[{nE, SL}];
1310                     cfpSpLp = CFP[{nE, SpLp}];
1311                     tnks = Table[
1312                         parentSL = cfpSL[[nn, 1]];
1313                         parentSpLp = cfpSpLp[[mm, 1]];
1314                         cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
1315                         tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
1316                     ],
1317                     {nn, 2, Length[cfpSL]},
1318                     {mm, 2, Length[cfpSpLp]}
1319                 );

```

```

1320     tnk = Total[Flatten[tnks]];
1321   ),
1322   tnk = 0;
1323 ];
1324 Return[ nE / (nE - opOrder) * tnk];];
1325
1326 (*Calculate the matrix elements of  $t^i$  for n up to nmax*)
1327 tktable = <||>;
1328 Do[(
1329   Do[(
1330     tkValue = Which[numE <= 2,
1331       (*Initialize n=1,2 with zeros*)
1332       0,
1333       numE == 3,
1334       (*Grab matrix elem in  $f^3$  from Judd 1984*)
1335       SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
1336       True,
1337       SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2, 3]]]
1338     ];
1339     tktable[{numE, SL, SpLp, opKey}] = tkValue;
1340   ),
1341   {SL, AllowedNKSLTerms[numE]},
1342   {SpLp, AllowedNKSLTerms[numE]},
1343   {opKey, Append[tiKeys, "e_{3}"]}
1344 ];
1345 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " configuration complete"]];
1346 ),
1347 {numE, 1, nmax}
1348 ];
1349
1350 (* Now use those matrix elements to determine their sum as weighted by their corresponding
1351 strengths  $T_i$  *)
1352 ThreeBodyTable = <||>;
1353 Do[
1354   Do[
1355     (
1356       ThreeBodyTable[{numE, SL, SpLp}] = (
1357         Sum[(
1358           If[tiKey == "t_{2}", t2Switch, 1] *
1359           tktable[{numE, SL, SpLp, tiKey}] *
1360           TSymbolsAssoc[tiKey] +
1361           If[tiKey == "t_{2}", 1 - t2Switch, 0] *
1362           (-tktable[{14 - numE, SL, SpLp, tiKey}]) *
1363           TSymbolsAssoc[tiKey]
1364         ),
1365         {tiKey, tiKeys}
1366       )
1367     ),
1368     {SL, AllowedNKSLTerms[numE]},
1369     {SpLp, AllowedNKSLTerms[numE]}
1370 ];
1371 PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix complete"]];,
1372 {numE, 1, 7}
1373 ];
1374
1375 ThreeBodyTables = Table[(
1376   terms = AllowedNKSLTerms[numE];
1377   singleThreeBodyTable =
1378     Table[
1379       {SL, SLp} -> ThreeBodyTable[{numE, SL, SLp}],
1380       {SL, terms},

```



```

1381     {SLp, terms}
1382 ];
1383 singleThreeBodyTable = Flatten[singleThreeBodyTable];
1384 singleThreeBodyTables = Table[{
1385     notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
1386     reps = ConstantArray[0, Length[TSymbols]];
1387     reps[[notNullPosition]] = 1;
1388     rep = AssociationThread[TSymbols -> reps];
1389     notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
1390 },
1391 {notNullSymbol, TSymbols}
1392 ];
1393 singleThreeBodyTables = Association[singleThreeBodyTables];
1394 numE -> singleThreeBodyTables),
1395 {numE, 1, 7}];
1396
1397 ThreeBodyTables = Association[ThreeBodyTables];
1398 If[OptionValue["Export"], (
1399     threeBodyTablefname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
1400     Export[threeBodyTablefname, ThreeBodyTable];
1401     threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
1402     Export[threeBodyTablesfname, ThreeBodyTables];
1403 )
1404 ];
1405 Return[{ThreeBodyTable, ThreeBodyTables}];)
1406
1407 ScalarOperatorProduct::usage="ScalarOperatorProduct[op1, op2, numE] calculated the
1408 innerproduct between the two scalar operators op1 and op2.";
1409 ScalarOperatorProduct[op1_, op2_, numE_] := Module[
1410     {terms, S, L, factor, term1, term2},
1411     (
1412     terms = AllowedNKSLTerms[numE];
1413     Simplify[
1414         Sum[(
1415             {S, L} = FindSL[term1];
1416             factor = TPO[S, L];
1417             factor * op1[{term1, term2}] * op2[{term2, term1}]
1418         ),
1419         {term1, terms},
1420         {term2, terms}
1421     ]
1422     )
1423 ];
1424
1425 (* ##### Three Body Operators ##### *)
1426 (* ##### *)
1427
1428 (* ##### *)
1429 (* ##### Magnetic Interactions ##### *)
1430
1431 ReducedT22inf2::usage="ReducedT22inf2[SL, SpLp] returns the reduced matrix element of the
1432 scalar component of the double tensor T22 for the terms SL, SpLp in f^2.
1433 Data used here for m0, m2, m4 is from Table I of Judd, BR, HM Crosswhite, and Hannah
1434 Crosswhite. Intra-Atomic Magnetic Interactions for f Electrons. Physical Review 169, no.
1435 1 (1968): 130.
1436 ";
1437 ReducedT22inf2[SL_, SpLp_] :=
1438 Module[{statePosition, PsiPspipStates, m0, m2, m4, Tkk2m},
1439     T22inf2 = <|
1440     {"3P", "3P"} -> -12 M0 - 24 M2 - 300/11 M4,
1441     {"3P", "3F"} -> 8/Sqrt[3] (3 M0 + M2 - 100/11 M4),

```

```

1439 {"3F", "3F"} -> 4/3 Sqrt[14] (-M0 + 8 M2 - 200/11 M4),
1440 {"3F", "3H"} -> 8/3 Sqrt[11/2] (2 M0 - 23/11 M2 - 325/121 M4),
1441 {"3H", "3H"} -> 4/3 Sqrt[143] (M0 - 34/11 M2 - (1325/1573) M4)
1442 |>;
1443 Which[
1444   MemberQ[Keys[T22inf2],{SL,SpLp}],
1445   Return[T22inf2[{SL,SpLp}]],
1446   MemberQ[Keys[T22inf2],{SpLp,SL}],
1447   Return[T22inf2[{SpLp,SL}]],
1448   True,
1449   Return[0]
1450 ];
1451 ];
1452
1453 ReducedT11inf2::usage="ReducedT11inf2[SL, SpLp] returns the reduced matrix element of the
1454 scalar component of the double tensor T11 for the given SL terms SL, SpLp.
1455 Data used here for m0, m2, m4 is from Table II of Judd, BR, HM Crosswhite, and Hannah
1456 Crosswhite. Intra-Atomic Magnetic Interactions for f Electrons. Physical Review 169, no.
1457 1 (1968): 130.
1458 ";
1459 ReducedT11inf2[SL_, SpLp_] :=
1460 Module[{T11inf2},
1461   T11inf2 = <|
1462     {"1S", "3P"} -> 6 M0 + 2 M2 + 10/11 M4,
1463     {"3P", "3P"} -> -36 M0 - 72 M2 - 900/11 M4,
1464     {"3P", "1D"} -> -Sqrt[(2/15)] (27 M0 + 14 M2 + 115/11 M4),
1465     {"1D", "3F"} -> Sqrt[2/5] (23 M0 + 6 M2 - 195/11 M4),
1466     {"3F", "3F"} -> 2 Sqrt[14] (-15 M0 - M2 + 10/11 M4),
1467     {"3F", "1G"} -> Sqrt[11] (-6 M0 + 64/33 M2 - 1240/363 M4),
1468     {"1G", "3H"} -> Sqrt[2/5] (39 M0 - 728/33 M2 - 3175/363 M4),
1469     {"3H", "3H"} -> 8/Sqrt[55] (-132 M0 + 23 M2 + 130/11 M4),
1470     {"3H", "1I"} -> Sqrt[26] (-5 M0 - 30/11 M2 - 375/1573 M4)
1471   |>;
1472   Which[
1473     MemberQ[Keys[T11inf2],{SL,SpLp}],
1474     Return[T11inf2[{SL,SpLp}]],
1475     MemberQ[Keys[T11inf2],{SpLp,SL}],
1476     Return[T11inf2[{SpLp,SL}]],
1477     True,
1478     Return[0]
1479   ]
1480 ];
1481
1482 MagneticInteractions[{numE_, SLJ_, SLJp_, J_}, chenDelta_] :=
1483 (
1484   key = {numE, SLJ, SLJp, J};
1485   ss = \[Sigma]SS * SpinSpinTable[key];
1486   sooandecso = S00andECS0Table[key];
1487   total = ss + sooandecso;
1488   (* In the type A errors the wrong values are different *)
1489   If[MemberQ[Keys[chenDeltas["A"]], {numE, SLJ, SLJp}],
1490     (
1491       {S, L} = FindSL[SLJ];
1492       {Sp, Lp} = FindSL[SLJp];
1493       phase = Phaser[Sp + L + J];
1494       Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
1495       Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
1496       {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ, SLJp}][ "wrong"];
1497       total = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1498       total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1499       total = total /. Prescaling;
1500       total = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)

```

```

1498 )
1499 ];
1500 (* In the type B errors the wrong values are zeros all around *)
1501 If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
1502 (
1503   {S, L} = FindSL[SLJ];
1504   {Sp, Lp} = FindSL[SLJp];
1505   phase = Phaser[Sp + L + J];
1506   Msixjay = SixJay[{Sp, Lp, J}, {L, S, 2}];
1507   Psixjay = SixJay[{Sp, Lp, J}, {L, S, 1}];
1508   {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
1509   total = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
1510   total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
1511   total = total /. Prescaling;
1512   total = wChErrB * total + (1 - wChErrB) * (ss + sooandecso)
1513 )
1514 ];
1515 Return[total];
1516 )
1517
1518 (* ##### Magnetic Interactions ##### *)
1519 (* ##### *)
1520
1521 (* ##### *)
1522 (* ##### Reduced S00 and ECS0 ##### *)
1523
1524 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.
1525
1526 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.\"
1527 ";
1528 T11n[numE_, SL_, SpLp_] := Module[
1529   {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp, parentSL, parentSpLp, Sb, Lb
, Tnkk, phase, Sbp, Lbp},
1530   {spin, orbital} = {1/2, 3};
1531   {S, L} = FindSL[SL];
1532   {Sp, Lp} = FindSL[SpLp];
1533   t = 1;
1534   cfpSL = CFP[{numE, SL}];
1535   cfpSpLp = CFP[{numE, SpLp}];
1536   Tnkk =
1537     Sum[(
1538       parentSL = cfpSL[[idx2, 1]];
1539       parentSpLp = cfpSpLp[[idx1, 1]];
1540       {Sb, Lb} = FindSL[parentSL];
1541       {Sbp, Lbp} = FindSL[parentSpLp];
1542       phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1543       (
1544         phase *
1545         cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1546         SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1547         SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1548         T11Table[{numE - 1, parentSL, parentSpLp}]
1549       )
1550     ),
1551     {idx1, 2, Length[cfpSpLp]},
1552     {idx2, 2, Length[cfpSL]}
1553 ];

```

```

1554   Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1555   Return[Tnkk];
1556 ];
1557
1558 Reducedt11inf2::usage="Reducedt11inf2[SL, SpLp] returns the reduced matrix element in f^2
of the double tensor operator t11 for the corresponding terms {SL, SpLp}.
1559 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.\"
1560 \"
1561 Reducedt11inf2[SL_, SpLp_] := Module[
1562   {t11inf2},
1563   t11inf2 = <|
1564     {\"1S\", \"3P\"} -> -2 P0 - 105 P2 - 231 P4 - 429 P6,
1565     {\"3P\", \"3P\"} -> -P0 - 45 P2 - 33 P4 + 1287 P6,
1566     {\"3P\", \"1D\"} -> Sqrt[15/2] (P0 + 32 P2 - 33 P4 - 286 P6),
1567     {\"1D\", \"3F\"} -> Sqrt[10] (-P0 - 9/2 P2 + 66 P4 - 429/2 P6),
1568     {\"3F\", \"3F\"} -> Sqrt[14] (-P0 + 10 P2 + 33 P4 + 286 P6),
1569     {\"3F\", \"1G\"} -> Sqrt[11] (P0 - 20 P2 + 32 P4 - 104 P6),
1570     {\"1G\", \"3H\"} -> Sqrt[10] (-P0 + 55/2 P2 - 23 P4 - 65/2 P6),
1571     {\"3H\", \"3H\"} -> Sqrt[55] (-P0 + 25 P2 + 51 P4 + 13 P6),
1572     {\"3H\", \"1I\"} -> Sqrt[13/2] (P0 - 21 P4 - 6 P6)
1573   |>;
1574   Which[
1575     MemberQ[Keys[t11inf2],{SL,SpLp}],
1576     Return[t11inf2[{SL,SpLp}]],
1577     MemberQ[Keys[t11inf2],{SpLp,SL}],
1578     Return[t11inf2[{SpLp,SL}]],
1579     True,
1580     Return[0]
1581   ]
1582 ]
1583
1584 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.
1585
1586 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 factor 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\".
1587
1588 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.\";
1589 ReducedS00andECS0inf2[SL_, SpLp_] :=
1590 Module[{pairPosition, f2TermPairs, a13, z13, redS00andECS0inf2},
1591   f2TermPairs = {
1592     {\"1S\", \"3P\"}, {\"3P\", \"1S\"},
1593     {\"3P\", \"3P\"}, {\"3P\", \"1D\"},
1594     {\"1D\", \"3P\"}, {\"1D\", \"3F\"},
1595     {\"3F\", \"1D\"}, {\"3F\", \"3F\"},
1596     {\"3F\", \"1G\"}, {\"1G\", \"3F\"},
1597     {\"1G\", \"3H\"}, {\"3H\", \"1G\"},
1598     {\"3H\", \"3H\"}, {\"3H\", \"1I\"},
1599     {\"1I\", \"3H\"}};
1600   a13 = (-33 M0 + 3 M2 + 15/11 M4 -
1601     6 P0 + 3/2 (35 P2 + 77 P4 + 143 P6));
1602   z13 = {2, 2,

```

```

1603     1,
1604     1/Sqrt[1080] (-90),
1605     1/Sqrt[1080] (-90),
1606     Sqrt[2/405] 45,
1607     Sqrt[2/405] 45,
1608     Sqrt[14],
1609     1/Sqrt[891] (-99),
1610     1/Sqrt[891] (-99),
1611     990/Sqrt[98010],
1612     990/Sqrt[98010],
1613     55/Sqrt[55],
1614     -2574/Sqrt[1019304],
1615     -2574/Sqrt[1019304]};
1616 pairPosition = Position[f2TermPairs, {SL, SpLp}];
1617 If[Length[pairPosition] == 0,
1618     Return[0],
1619     pairPosition = pairPosition[[1, 1]]
1620 ];
1621
1622 redS00andECS0inf2 = (
1623     ReducedT11inf2[SL, SpLp] +
1624     Reducedt11inf2[SL, SpLp] -
1625     a13 / 6 * z13[[pairPosition]]
1626 );
1627 redS00andECS0inf2 = SimplifyFun[redS00andECS0inf2];
1628 Return[redS00andECS0inf2];
1629 ];
1630
1631 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.
";
1632
1633 ReducedS00andECS0infn[numE_, SL_, SpLp_] := Module[
1634     {spin, orbital, t, S, L, Sp, Lp, idx1, idx2, cfpSL, cfpSpLp, parentSL, Sb, Lb, Sbp, Lbp,
parentSpLp, funval},
1635     {spin, orbital} = {1/2, 3};
1636     {S, L} = FindSL[SL];
1637     {Sp, Lp} = FindSL[SpLp];
1638     t = 1;
1639     cfpSL = CFP[{numE, SL}];
1640     cfpSpLp = CFP[{numE, SpLp}];
1641     funval =
1642         Sum[
1643             (
1644                 parentSL = cfpSL[[idx2, 1]];
1645                 parentSpLp = cfpSpLp[[idx1, 1]];
1646                 {Sb, Lb} = FindSL[parentSL];
1647                 {Sbp, Lbp} = FindSL[parentSpLp];
1648                 phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1649                 (
1650                     phase *
1651                     cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1652                     SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1653                     SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1654                     S00andECS0LSTable[{numE - 1, parentSL, parentSpLp}]
1655                 )
1656             ),
1657     {idx1, 2, Length[cfpSpLp]},
1658     {idx2, 2, Length[cfpSL]}

```

```

1659 ];
1660 funval *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1661 Return[funval];
1662 ];
1663
1664 Options[GenerateS00andECSOLSTable] = {"Progress" -> True, "Export" -> True};
1665 GenerateS00andECSOLSTable[nmax_Integer, OptionsPattern[]]:= (
1666   If[And[OptionValue["Progress"], frontEndAvailable],
1667     (
1668       numItersai = Association[Table[numE->Length[AllowedNKSLTerms[numE]]^2, {numE, 1, nmax
1669     }]];
1669     counters = Association[Table[numE->0, {numE, 1, nmax}]];
1670     totalIters = Total[Values[numItersai[[1;;nmax]]]];
1671     template1 = StringTemplate["Iteration 'numiter' of 'totaliter'"];
1672     template2 = StringTemplate["'remtime' min remaining"]; template3 = StringTemplate["
Iteration speed = 'speed' ms/it"];
1673     template4 = StringTemplate["Time elapsed = 'runtime' min"];
1674     progBar = PrintTemporary[
1675       Dynamic[
1676         Pane[
1677           Grid[{
1678             {Superscript["f", numE]},
1679             {template1[<|"numiter"->numiter, "totaliter"->totalIters|>]},
1680             {template4[<|"runtime"->Round[QuantityMagnitude[UnitConvert[(Now-
startTime), "min"]], 0.1]|>]},
1681             {template2[<|"remtime"->Round[QuantityMagnitude[UnitConvert[(Now-
startTime)/(numiter)*(totalIters-numiter), "min"]], 0.1]|>]},
1682             {template3[<|"speed"->Round[QuantityMagnitude[Now-startTime, "ms"]/(
numiter), 0.01]|>]}, {ProgressIndicator[Dynamic[numiter], {1, totalIters}]}
1683           ],
1684           Frame->All
1685         ],
1686         Full,
1687         Alignment->Center
1688       ]
1689     ]
1690   ];
1691 )
1692 ];
1693 S00andECSOLSTable = <||>;
1694 numiter = 1;
1695 startTime = Now;
1696 Do[
1697   (
1698     numiter+= 1;
1699     S00andECSOLSTable[{numE, SL, SpLp}] = Which[
1700       numE==1,
1701       0,
1702       numE==2,
1703       SimplifyFun[ReducedS00andECS0inf2[SL, SpLp]],
1704       True,
1705       SimplifyFun[ReducedS00andECS0infn[numE, SL, SpLp]]
1706     ];
1707   ),
1708   {numE, 1, nmax},
1709   {SL, AllowedNKSLTerms[numE]},
1710   {SpLp, AllowedNKSLTerms[numE]}
1711 ];
1712 If[And[OptionValue["Progress"], frontEndAvailable],
1713   NotebookDelete[progBar]];
1714 If[OptionValue["Export"],
1715   (fname = FileNameJoin[{moduleDir, "data", "ReducedS00andECSOLSTable.m"}]);

```

```

1716     Export[fname, S00andECS0LSTable];
1717 )
1718 ];
1719 Return[S00andECS0LSTable];
1720 );
1721
1722 (* ##### Reduced S00 and ECS0 ##### *)
1723 (* ##### *)
1724
1725 (* ##### *)
1726 (* ##### Spin-Spin ##### *)
1727
1728 T22n::usage="T22n[n, SL, SpLp] calculates the reduced matrix element of the T22 operator
for the f^n configuration corresponding to the terms SL and SpLp. This is the operator
corresponding to the inter-electron between spin.
1729 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.\"
";
1730
1731 T22n[numE_, SL_, SpLp_] := Module[
1732 {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp, parentSL, parentSpLp, Sb, Lb
, Tnkk, phase, Sbp, Lbp},
1733 {spin, orbital} = {1/2, 3};
1734 {S, L} = FindSL[SL];
1735 {Sp, Lp} = FindSL[SpLp];
1736 t = 2;
1737 cfpSL = CFP[{numE, SL}];
1738 cfpSpLp = CFP[{numE, SpLp}];
1739 Tnkk =
1740 Sum[(
1741 parentSL = cfpSL[[idx2, 1]];
1742 parentSpLp = cfpSpLp[[idx1, 1]];
1743 {Sb, Lb} = FindSL[parentSL];
1744 {Sbp, Lbp} = FindSL[parentSpLp];
1745 phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
1746 (
1747 phase *
1748 cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
1749 SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
1750 SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
1751 T22Table[{numE - 1, parentSL, parentSpLp}]
1752 )
1753 ),
1754 {idx1, 2, Length[cfpSpLp]},
1755 {idx2, 2, Length[cfpSL]}
1756 ];
1757 Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
1758 Return[Tnkk];
1759 ];
1760
1761 Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
1762 GenerateT22Table[nmax_Integer, OptionsPattern[]] := (
1763 If[And[OptionValue["Progress"], frontEndAvailable],
1764 (
1765 numItersai = Association[Table[numE->Length[AllowedNKSLTerms[numE]]^2, {numE, 1, nmax
}]];
1766 counters = Association[Table[numE->0, {numE, 1, nmax}]];
1767 totalIters = Total[Values[numItersai[[1;;nmax]]]];
1768 template1 = StringTemplate["Iteration 'numiter' of 'totaliter'"];
1769 template2 = StringTemplate["'remtime' min remaining"]; template3 = StringTemplate["
Iteration speed = 'speed' ms/it"];
1770 template4 = StringTemplate["Time elapsed = 'runtime' min"];

```

```

1771     progBar = PrintTemporary[
1772         Dynamic[
1773             Pane[
1774                 Grid[{{Superscript["f", numE]},
1775                     {template1[<|"numiter"->numiter, "totaliter"->totalIters|>]},
1776                     {template4[<|"runtime"->Round[QuantityMagnitude[UnitConvert[(Now-
1777 startTime), "min"]], 0.1]|>]},
1778                     {template2[<|"remtime"->Round[QuantityMagnitude[UnitConvert[(Now-
1779 startTime)/(numiter)*(totalIters-numiter), "min"]], 0.1]|>]},
1780                     {template3[<|"speed"->Round[QuantityMagnitude[Now-startTime, "ms"]]/(
1781 numiter), 0.01]|>]},
1782                     {ProgressIndicator[Dynamic[numiter], {1, totalIters}]}}},
1783                 Frame->All],
1784                 Full,
1785                 Alignment->Center]
1786             ]
1787         ];
1788     T22Table = <||>;
1789     startTime = Now;
1790     numiter = 1;
1791     Do[
1792         (
1793             numiter+= 1;
1794             T22Table[{numE, SL, SpLp}] = Which[
1795                 numE==1,
1796                 0,
1797                 numE==2,
1798                 SimplifyFun[ReducedT22inf2[SL, SpLp]],
1799                 True,
1800                 SimplifyFun[T22n[numE, SL, SpLp]]
1801             ];
1802         {numE, 1, nmax},
1803         {SL, AllowedNKSLTerms[numE]},
1804         {SpLp, AllowedNKSLTerms[numE]}
1805     ];
1806     If[And[OptionValue["Progress"], frontEndAvailable],
1807         NotebookDelete[progBar]
1808     ];
1809     If[OptionValue["Export"],
1810         (
1811             fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
1812             Export[fname, T22Table];
1813         )
1814     ];
1815     Return[T22Table];
1816 ];
1817
1818 SpinSpin::usage="SpinSpin[n, SL, SpLp, J] returns the matrix element <|SL,J|spin-spin|SpLp,
1819 J|> for the spin-spin operator within the configuration f^n. This matrix element is
1820 independent of MJ. This is obtained by querying the relevant reduced matrix element by
1821 querying the association T22Table and putting in the adequate phase and 6-j symbol.
1822
1823 This is calculated according to equation (3) in \"Judd, BR, HM Crosswhite, and Hannah
1824 Crosswhite. \"Intra-Atomic Magnetic Interactions for f Electrons.\" Physical Review 169,
1825 no. 1 (1968): 130.\"
1826 \".
1827 ";
1828 SpinSpin[numE_, SL_, SpLp_, J_] := Module[
1829     {S, L, Sp, Lp,  $\alpha$ , val},

```



```

1825      $\alpha$  = 2;
1826     {S, L} = FindSL[SL];
1827     {Sp, Lp} = FindSL[SpLp];
1828     val = (
1829         Phaser[Sp + L + J] *
1830         SixJay[{Sp, Lp, J}, {L, S,  $\alpha$ }] *
1831         T22Table[{numE, SL, SpLp}]
1832     );
1833     Return[val]
1834 ];
1835
1836 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.";
1837 Options[GenerateSpinSpinTable] = {"Export"->False};
1838 GenerateSpinSpinTable[nmax_, OptionsPattern[]] :=
1839 (
1840     SpinSpinTable = <||>;
1841     PrintTemporary[Dynamic[numE]];
1842     Do[
1843         SpinSpinTable[{numE, SL, SpLp, J}] = (SpinSpin[numE, SL, SpLp, J]);,
1844         {numE, 1, nmax},
1845         {J, MinJ[numE], MaxJ[numE]},
1846         {SL, First /@ AllowedNKSLforJTerms[numE, J]},
1847         {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
1848     ];
1849     If[OptionValue["Export"],
1850         (fname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
1851          Export[fname, SpinSpinTable];
1852         )
1853     ];
1854     Return[SpinSpinTable];
1855 );
1856
1857 (* ##### *)
1858 (* ##### Spin-Spin ##### *)
1859
1860 (* ##### *)
1861 (* ##### Spin-Other-Orbit and Electrostatically-Correlated-Spin-Orbit ##### *)
1862
1863 S00andECS0::usage="S00andECS0[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
S00andECS0LSTable and putting in the adequate phase and 6-j symbol. The S00andECS0LSTable
puts together the reduced matrix elements from three operators.
1864
1865 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.\".
1866 ";
1867 S00andECS0[numE_, SL_, SpLp_, J_] := Module[
1868     {S, Sp, L, Lp,  $\alpha$ , val},
1869      $\alpha$  = 1;
1870     {S, L} = FindSL[SL];
1871     {Sp, Lp} = FindSL[SpLp];
1872     val = (
1873         Phaser[Sp + L + J] *

```

```

1874         SixJay[{Sp, Lp, J}, {L, S,  $\alpha$ }] *
1875         S00andECS0LSTable[{numE, SL, SpLp}]
1876     );
1877     Return[val];
1878 ]
1879
1880 Prescaling = {P2 -> P2/225, P4 -> P4/1089, P6 -> 25 * P6 / 184041};
1881 GenerateS00andECS0Table::usage="GenerateS00andECS0Table[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.";
1882 Options[GenerateS00andECS0Table] = {"Export"->False}
1883 GenerateS00andECS0Table[nmax_, OptionsPattern[]]:= (
1884     S00andECS0Table = <||>;
1885     Do[
1886         S00andECS0Table[{numE, SL, SpLp, J}] = (S00andECS0[numE, SL, SpLp, J] /. Prescaling);,
1887         {numE, 1, nmax},
1888         {J, MinJ[numE], MaxJ[numE]},
1889         {SL, First /@ AllowedNKSLforJTerms[numE, J]},
1890         {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
1891     ];
1892     If[OptionValue["Export"],
1893     (
1894         fname = FileNameJoin[{moduleDir, "data", "S00andECS0Table.m"}];
1895         Export[fname, S00andECS0Table];
1896     )
1897 ];
1898     Return[S00andECS0Table];
1899 );
1900
1901 (* ##### Spin-Other-Orbit and Electrostatically-Correlated-Spin-Orbit ##### *)
1902 (* ##### *)
1903
1904 (* ##### *)
1905 (* ##### Crystal Field ##### *)
1906
1907 Cqk::usage = "Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp].";
1908 Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
1909     {S, Sp, L, Lp, orbital, val},
1910     orbital = 3;
1911     {S, L} = FindSL[NKSL];
1912     {Sp, Lp} = FindSL[NKSLp];
1913     f1 = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
1914     val =
1915         If[f1==0,
1916             0,
1917             (
1918                 f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
1919                 If[f2==0,
1920                     0,
1921                     (
1922                         f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
1923                         If[f3==0,
1924                             0,
1925                             (
1926                                 (
1927                                     Phaser[J - M + S + Lp + J + k] *
1928                                     Sqrt[TP0[J, Jp]] *
1929                                     f1 *
1930                                     f2 *

```

```

1931         f3 *
1932         Ck[orbital, k]
1933     )
1934 )
1935 ]
1936 )
1937 ]
1938 )
1939 ];
1940 val
1941 ]
1942
1943 Bqk[q_, 2] := {B02/2, B12, B22}[[q + 1]];
1944 Bqk[q_, 4] := {B04/2, B14, B24, B34, B44}[[q + 1]];
1945 Bqk[q_, 6] := {B06/2, B16, B26, B36, B46, B56, B66}[[q + 1]];
1946
1947 Sqk[q_, 2] := {Sm22, Sm12, S02, S12, S22}[[q + 3]];
1948 Sqk[q_, 4] := {Sm44, Sm34, Sm24, Sm14, S04, S14, S24, S34, S44}[[q + 5]];
1949 Sqk[q_, 6] := {Sm66, Sm56, Sm46, Sm36, Sm26, Sm16, S06, S16, S26, S36, S46, S56, S66}[[q +
    7]];
1950
1951 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.";
1952 CrystalField[numE_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := (
1953     Sum[
1954         (
1955             cqk = Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp];
1956             cmqk = Cqk[numE, -q, k, NKSL, J, M, NKSLp, Jp, Mp];
1957             Bqk[q, k] * (cqk + (-1)^q * cmqk) +
1958             I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
1959         ),
1960         {k, {2, 4, 6}},
1961         {q, 0, k}
1962     ]
1963 )
1964
1965 TotalCFIters::usage = "TotalIters[i, j] returns total number of function evaluations for
    calculating all the matrix elements for the  $\backslash\backslash(*\text{SuperscriptBox}[\backslash(f\backslash), \backslash(i\backslash)]\backslash)$  to the
     $\backslash\backslash(*\text{SuperscriptBox}[\backslash(f\backslash), \backslash(j\backslash)]\backslash)$  configurations.";
1966 TotalCFIters[i_, j_] := (
1967     numIters = {196, 8281, 132496, 1002001, 4008004, 9018009, 11778624};
1968     Return[Total[numIters[[i ; j]]]];
1969 )
1970
1971 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.";
1972 Options[GenerateCrystalFieldTable] = {"Export" -> False, "Progress" -> True, "Compress" ->
    True}
1973 GenerateCrystalFieldTable[numEs_List:{1,2,3,4,5,6,7}, OptionsPattern[]]:= (
1974     ExportFun =
1975     If[OptionValue["Compress"],
1976         ExportMZip,
1977         Export
1978     ];
1979     numiter = 1;
1980     template1 = StringTemplate["Iteration 'numiter' of 'totaliter'"];

```

```

1981 template2 = StringTemplate["'remtime' min remaining"];
1982 template3 = StringTemplate["Iteration speed = 'speed' ms/it"];
1983 template4 = StringTemplate["Time elapsed = 'runtime' min"];
1984 totalIter = Total[TotalCFIters[#, #] & /@ numEs];
1985 freebies = 0;
1986 startTime = Now;
1987 If[And[OptionValue["Progress"], frontEndAvailable],
1988   progBar = PrintTemporary[
1989     Dynamic[
1990       Pane[
1991         Grid[
1992           {
1993             {Superscript["f", numE]},
1994             {template1[<|"numiter" -> numiter, "totaliter" -> totalIter|>]},
1995             {template4[<|"runtime" -> Round[QuantityMagnitude[UnitConvert[(Now -
1996 startTime), "min"]], 0.1]|>]},
1997             {template2[<|"remtime" -> Round[QuantityMagnitude[UnitConvert[(Now -
1998 startTime)/(numiter - freebies) * (totalIter - numiter), "min"]], 0.1]|>]},
1999             {template3[<|"speed" -> Round[QuantityMagnitude[Now - startTime, "ms"]/(
2000 numiter-freebies), 0.01]|>]},
2001             {ProgressIndicator[Dynamic[numiter], {1, totalIter}]}
2002           ],
2003           Frame -> All
2004         ],
2005         Full,
2006         Alignment -> Center
2007       ]
2008     ];
2009   ];
2010 Do[
2011   (
2012     exportFname = FileNameJoin[{moduleDir, "data", "CrystalFieldTable_f"<>ToString[numE
2013 ]<>".zip"}];
2014     If[FileExistsQ[exportFname],
2015       CrystalFieldTable = Import[exportFname];
2016       Print["File exists, skipping ..."];
2017       numiter+= TotalCFIters[numE, numE];
2018       freebies+= TotalCFIters[numE, numE];
2019       Continue[];
2020     ];
2021     CrystalFieldTable = <||>;
2022     Do[
2023       (
2024         numiter+= 1;
2025         CrystalFieldTable[{numE, NKSL, J, M, NKSLp, Jp, Mp}] = CrystalField[numE, NKSL, J
2026 , M, NKSLp, Jp, Mp];
2027       ),
2028       {J, MinJ[numE], MaxJ[numE]},
2029       {Jp, MinJ[numE], MaxJ[numE]},
2030       {M, AllowedMforJ[J]},
2031       {Mp, AllowedMforJ[Jp]},
2032       {NKSL, First /@ AllowedNKSLforJTerms[numE, J]},
2033       {NKSLp, First /@ AllowedNKSLforJTerms[numE, Jp]}
2034     ];
2035     If[And[OptionValue["Progress"], frontEndAvailable],
2036       NotebookDelete[progBar]
2037     ];
2038     If[OptionValue["Export"],
2039       (
2040         Print["Exporting to file "<>ToString[exportFname]];
2041         ExportFun[exportFname, CrystalFieldTable];
2042       )
2043     ];
2044   ];

```

```

2038         )
2039     ];
2040 ),
2041 {numE, numEs}
2042 ]
2043 )
2044
2045 (* ##### Crystal Field ##### *)
2046 (* ##### *)
2047
2048 (* ##### *)
2049 (* ##### Configuration-Interaction via Casimir Operators ##### *)
2050
2051 CasimirS03::usage = "CasimirS03[SL, SpLp] returns LS reduced matrix element of the
configuration interaction term corresponding to the Casimir operator of R3.";
2052 CasimirS03[{SL_, SpLp_}] := (
2053     {S, L} = FindSL[SL];
2054     If[SL == SpLp,
2055          $\alpha$  * L * (L + 1),
2056         0
2057     ]
2058 )
2059
2060 GG2U::usage = "GG2U is an association whose keys are labels for the irreducible
representations of group G2 and whose values are the eigenvalues of the corresponding
Casimir operator.
Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\", table 2-6.";
2061 GG2U = Association[{
2062     "00" -> 0,
2063     "10" -> 6/12,
2064     "11" -> 12/12,
2065     "20" -> 14/12,
2066     "21" -> 21/12,
2067     "22" -> 30/12,
2068     "30" -> 24/12,
2069     "31" -> 32/12,
2070     "40" -> 36/12}
2071 ];
2072
2073
2074 CasimirG2::usage = "CasimirG2[SL, SpLp] returns LS reduced matrix element of the
configuration interaction term corresponding to the Casimir operator of G2.";
2075 CasimirG2[{SL_, SpLp_}] := (
2076     Ulabel = FindNKLSTerm[SL][[1]][[4]];
2077     If[SL==SpLp,
2078          $\beta$  * GG2U[Ulabel],
2079         0
2080     ]
2081 )
2082
2083 GS07W::usage = "GS07W is an association whose keys are labels for the irreducible
representations of group R7 and whose values are the eigenvalues of the corresponding
Casimir operator.
Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\", table 2-7.";
2084 GS07W := Association[
2085     {
2086         "000" -> 0,
2087         "100" -> 3/5,
2088         "110" -> 5/5,
2089         "111" -> 6/5,
2090         "200" -> 7/5,
2091         "210" -> 9/5,
2092         "211" -> 10/5,
2093

```

```

2094     "220" -> 12/5,
2095     "221" -> 13/5,
2096     "222" -> 15/5
2097 }
2098 ];
2099
2100 CasimirS07::usage = "CasimirS07[SL, SpLp] returns the LS reduced matrix element of the
    configuration interaction term corresponding to the Casimir operator of R7.";
2101 CasimirS07[{SL_, SpLp_}] := (
2102     Wlabel = FindNKLSTerm[SL][[1]][[3]];
2103     If[SL==SpLp,
2104          $\gamma$  * GS07W[Wlabel],
2105         0
2106     ]
2107 )
2108
2109 ElectrostaticConfigInteraction::usage = "ElectrostaticConfigInteraction[{SL, SpLp}] returns
    the matrix element for configuration interaction as approximated by the Casimir
    operators of the groups R3, G2, and R7. SL and SpLp are strings that represent terms
    under LS coupling.";
2110 ElectrostaticConfigInteraction[{SL_, SpLp_}] := Module[
2111     {S, L, val},
2112     {S, L} = FindSL[SL];
2113     val = (
2114         If[SL == SpLp,
2115             CasimirS03[{SL, SL}] +
2116             CasimirS07[{SL, SL}] +
2117             CasimirG2[{SL, SL}],
2118             0
2119         ]
2120     );
2121     ElectrostaticConfigInteraction[{S, L}] = val;
2122     Return[val];
2123 ]
2124
2125 (* ##### Configuration-Interaction via Casimir Operators ##### *)
2126 (* ##### *)
2127
2128 (* ##### *)
2129 (* ##### Block assembly ##### *)
2130
2131 Options[JJBBlockMatrix] = {"Sparse"->True, "ChenDeltas"->False};
2132 JJBBlockMatrix::usage = "For given J, J' in the f^n configuration JJBBlockMatrix[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.";
2133 JJBBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]]:= Module[
2134     {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
2135     SLterm, SpLpterm,
2136     MJ, MJp,
2137     subKron, matValue, eMatrix},
2138     (
2139         NKSLJMs = AllowedNKSLJMforJTerms[numE, J];
2140         NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
2141         eMatrix =
2142             Table[
2143                 (*Condition for a scalar matrix op*)
2144                 SLterm = NKSLJM[[1]];
2145                 SpLpterm = NKSLJMp[[1]];
2146                 MJ = NKSLJM[[3]];
2147                 MJp = NKSLJMp[[3]];

```

```

2148         subKron =
2149             (
2150                 KroneckerDelta[J, Jp] *
2151                 KroneckerDelta[MJ, MJp]
2152             );
2153         matValue =
2154             If[subKron==0,
2155                 0,
2156                 (
2157                     ElectrostaticTable[{numE, SLterm, SpLpterm}] +
2158                     ElectrostaticConfigInteraction[{SLterm, SpLpterm}] +
2159                     SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
2160                     MagneticInteractions[{numE, SLterm, SpLpterm, J}, OptionValue["ChenDeltas"
]] +
2161                     ThreeBodyTable[{numE, SLterm, SpLpterm}]
2162                 )
2163             ];
2164         matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp}];
2165         matValue,
2166         {NKSLJMp, NKSLJMps},
2167         {NKSLJM, NKSLJMs}
2168     ];
2169     If[OptionValue["Sparse"],
2170         eMatrix = SparseArray[eMatrix]
2171     ];
2172     Return[eMatrix]
2173 )
2174 ];
2175
2176 EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some point may be used to
    redefine states used in basis.";
2177 EnergyStates[numE_, J_] := AllowedNKSLJMforJTerms[numE, J];
2178
2179 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.";
2180 Options[JJBBlockMatrixFileName] = {"FilenameAppendix" -> ""}
2181 JJBBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
2182     fileApp = OptionValue["FilenameAppendix"];
2183     fname = FileNameJoin[{moduleDir,
2184         "hams",
2185         StringJoin[{"f", ToString[numE], "_JJBBlockMatrixTable", fileApp, ".m"}]};
2186     Return[fname];
2187 );
2188
2189 Options[TabulateJJBBlockMatrixTable] = {"Sparse" -> True, "ChenDeltas" -> False};
2190 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}. EnergyStatesTable
    is an association with keys equal to lists of the form {numE, J}. AllowedM is another
    association with keys equal to lists of the form {numE, J} and values equal to lists
    equal to the corresponding values of MJ. It's unnecessary (and it won't work in this
    implementation) to give numE > 7 given the equivalency between electron and hole
    configurations.";
2191 TabulateJJBBlockMatrixTable[numE_, CFTable_, OptionsPattern[]] := (
2192     JJBBlockMatrixTable = <||>;
2193     EnergyStatesTable = <||>;
2194     AllowedM = <||>;
2195     totalIterations = Length[AllowedJ[numE]]^2;
2196     template1 = StringTemplate["Iteration 'numiter' of 'totaliter'"];
2197     template2 = StringTemplate["'remtime' min remaining"];
2198     template4 = StringTemplate["Time elapsed = 'runtime' min"];

```

```

2199 numiter = 0;
2200 startTime = Now;
2201 If[$FrontEnd != Null,
2202 (
2203     temp = PrintTemporary[
2204         Dynamic[
2205             Grid[
2206                 {
2207                     {template1[<|"numiter"->numiter, "totaliter"->totalIterations|>]},
2208                     {template2[<|"remtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime)/(
2209 Max[1,numiter])*(totalIterations-numiter), "min"]], 0.1]|>]},
2210                     {template4[<|"runtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime),
2211 "min"]], 0.1]|>]},
2212                     {ProgressIndicator[numiter, {1, totalIterations}]}
2213                 ]
2214             ];
2215         )
2216     ];
2217     Do[
2218     (
2219         JJBBlockMatrixTable[{numE, J, Jp}] = JJBBlockMatrix[numE, J, Jp, CFTable, "Sparse"->
2220 OptionValue["Sparse"], "ChenDeltas" -> OptionValue["ChenDeltas"]];
2221         EnergyStatesTable[{numE, J}] = EnergyStates[numE, J];
2222         AllowedM[{numE, J}] = Table[M, {J, MinJ[numE], MaxJ[numE]}, {M, -J, J
2223 }];
2224         numiter += 1;
2225     ),
2226     {Jp, AllowedJ[numE]},
2227     {J, AllowedJ[numE]}
2228 ];
2229 If[$FrontEnd != Null,
2230     NotebookDelete[temp]
2231 ];
2232 Return[{JJBBlockMatrixTable, EnergyStatesTable, AllowedM}];
2233 )
2234
2235 Options[TabulateManyJJBBlockMatrixTables] = {"Overwrite"->False, "Sparse"->True, "ChenDeltas
2236 "->False, "FilenameAppendix"-> ""};
2237
2238 TabulateManyJJBBlockMatrixTables::usage = "TabulateManyJJBBlockMatrixTables[{n1, n2, ...}]
2239 calculates the tables of matrix elements for the requested f^n_i configurations. The
2240 function does not return the matrices themselves. It instead returns an association whose
2241 keys are numE and whose values are the filenames where the output of
2242 TabulateJJBBlockMatrixTables was saved to. When these files are loaded with Get, the
2243 following three symbols are thus defined: JJBBlockMatrixTable, EnergyStatesTable, and
2244 AllowedM.
2245
2246 JJBBlockMatrixTable is an association whose keys are of the form {n, J, Jp} and whose values
2247 are matrix elements.";
2248
2249 TabulateManyJJBBlockMatrixTables[ns_, OptionsPattern[]]:= (
2250     overwrite = OptionValue["Overwrite"];
2251     fName = <|>;
2252     fileApp = OptionValue["FilenameAppendix"];
2253     Do[
2254     (
2255         CFdataFilename = FileNameJoin[{moduleDir, "data", "CrystalFieldTable_f"<>ToString[
2256 numE]<>".zip"}];
2257         PrintTemporary["Importing CrystalFieldTable from ", CFdataFilename, " ..."];
2258         CrystalFieldTable = ImportMZip[CFdataFilename];
2259
2260         PrintTemporary["#----- numE = ", numE, " -----#"];
2261         exportFname = JJBBlockMatrixFileName[numE, "FilenameAppendix" -> fileApp];

```



```

2248     fName = exportFname;
2249     If[FileExistsQ[exportFname] && Not[overwrite],
2250       Continue[]
2251     ];
2252     {JJBBlockMatrixTable, EnergyStatesTable, AllowedM} = TabulateJJBBlockMatrixTable[numE,
CrystalFieldTable, "Sparse" -> OptionValue["Sparse"], "ChenDeltas" -> OptionValue["
ChenDeltas"]];
2253     If[FileExistsQ[exportFname] && overwrite,
2254       DeleteFile[exportFname]
2255     ];
2256     Save[exportFname, {JJBBlockMatrixTable, EnergyStatesTable, AllowedM}];
2257
2258     ClearAll[CrystalFieldTable];
2259   },
2260   {numE, ns}
2261 ];
2262 Return[fNames];
2263 )
2264
2265 HamMatrixAssembly::usage="HamMatrixAssembly[numE] returns the Hamiltonian matrix for the f^
n_i configuration. The matrix is returned as a SparseArray."
2266 Options[HamMatrixAssembly] = {"FilenameAppendix" -> ""};
2267 HamMatrixAssembly[nf_, OptionsPattern[]] := Module[
2268   {numE, ii, jj, howManyJs, Js, blockHam},
2269   (*****
2270   (*hole-particle equivalence enforcement*)
2271   numE = nf;
2272   allVars = {E0, E1, E2, E3, ζ, F0, F2, F4, F6, M0, M2, M4, T2, T2p,
2273     T3, T4, T6, T7, T8, P0, P2, P4, P6, gs,
2274     α, β, γ, B02, B04, B06, B12, B14, B16,
2275     B22, B24, B26, B34, B36, B44, B46, B56, B66, S12, S14, S16, S22,
2276     S24, S26, S34, S36, S44, S46, S56, S66, T11, T11p, T12, T14, T15, T16,
2277     T17, T18, T19};
2278   params0 = AssociationThread[allVars, allVars];
2279   If[nf > 7,
2280     (
2281       numE = 14 - nf;
2282       params = HoleElectronConjugation[params0];
2283     ),
2284     params = params0;
2285   ];
2286   (* Load symbolic expressions for LS,J,J' energy sub-matrices. *)
2287   emFname = JJBBlockMatrixFileName[numE, "FilenameAppendix" -> OptionValue["FilenameAppendix
2288   "]];
2289   Get[emFname];
2290   (*Patch together the entire matrix representation using J,J' blocks.*)
2291   PrintTemporary["Patching JJ blocks ..."];
2292   Js = AllowedJ[numE];
2293   howManyJs = Length[Js];
2294   blockHam = ConstantArray[0, {howManyJs, howManyJs}];
2295   Do[
2296     blockHam[[jj, ii]] = JJBBlockMatrixTable[{numE, Js[[ii]], Js[[jj]]}];,
2297     {ii, 1, howManyJs},
2298     {jj, 1, howManyJs}
2299   ];
2300   (* Once the block form is created flatten it *)
2301   blockHam = ArrayFlatten[blockHam];
2302   blockHam = ReplaceInSparseArray[blockHam, params];
2303   Return[blockHam];
2304 ]
2305 (* ***** Block assembly ***** *)

```

```

2306 (* ##### *)
2307
2308 (* ##### *)
2309 (* ##### Printers and Labels ##### *)
2310
2311 PrintL::usage = "PrintL[L] give the string representation of a given angular momentum.";
2312 PrintL[L_] := If[StringQ[L], L, StringTake[specAlphabet, {L + 1}]]
2313
2314 FindSL::usage = "FindSL[LS] gives the spin and orbital angular momentum that corresponds to
the provided string LS.";
2315 FindSL[SL_] := (
2316   FindSL[SL] =
2317     If[StringQ[SL],
2318       {
2319         (ToExpression[StringTake[SL, 1]]-1)/2,
2320         StringPosition[specAlphabet, StringTake[SL, {2}]][[1, 1]]-1
2321       },
2322       SL
2323     ]
2324 )
2325
2326 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.";
2327 PrintSLJ[SLJ_] :=
2328   RowBox[{SuperscriptBox[" ", 2 SLJ[[1]] + 1,
2329     SubscriptBox[PrintL[SLJ[[2]]], SLJ[[3]]]}] // DisplayForm;
2330
2331 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.";
2332 PrintSLJM[SLJM_] :=
2333   RowBox[{SuperscriptBox[" ", 2 SLJM[[1]] + 1,
2334     SubscriptBox[PrintL[SLJM[[2]]], {SLJM[[3]], SLJM[[4]]}]}] //
2335   DisplayForm;
2336
2337 (* ##### Printers and Labels ##### *)
2338 (* ##### *)
2339
2340 (* ##### *)
2341 (* ##### Term management ##### *)
2342
2343 AllowedSLTerms::usage = "AllowedSLTerms[numE] returns a list with the allowed terms in the
f^numE configuration, the terms are given as lists in the format {S, L}. This list may
have redundancies which are compatible with the degeneracies that might correspond to the
given case.";
2344 AllowedSLTerms[numE_] := Map[FindSL[First[#]] &, CFPTerms[Min[numE, 14-numE]]]
2345
2346 AllowedNKSLTerms::usage = "AllowedNKSLTerms[numE] returns a list with the allowed terms in
the f^numE configuration, the terms are given as strings in spectroscopic notation. The
integers in the last positions are used to distinguish cases with degeneracy.";
2347 AllowedNKSLTerms[numE_] := (Map[First, CFPTerms[Min[numE, 14-numE]]])
2348 AllowedNKSLTerms[0] = {"1S"};
2349 AllowedNKSLTerms[14] = {"1S"};
2350
2351 MaxJ::usage = "MaxJ[numE] gives the maximum J = S+L that corresponds to the configuration f
^numE.";
2352 MaxJ[numE_] := Max[Map[Total, AllowedSLTerms[Min[numE, 14-numE]]]]
2353
2354 MinJ::usage = "MinJ[numE] gives the minimum J = S+L that corresponds to the configuration f

```

```

^numE.";
2355 MinJ[numE_] := Min[Map[Abs[Part[#, 1] - Part[#, 2]] &, AllowedSLTerms[Min[numE, 14-numE]]]]
2356
2357 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.";
2358 AllowedSLJTerms[numE_] :=
2359 Module[{idx1, allowedSL, allowedSLJ},
2360   allowedSL = AllowedSLTerms[numE];
2361   allowedSLJ = {};
2362   For[
2363     idx1 = 1,
2364     idx1 <= Length[allowedSL],
2365     termSL = allowedSL[[idx1]];
2366     termsSLJ =
2367       Table[
2368         {termSL[[1]], termSL[[2]], J},
2369         {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2370       ];
2371     allowedSLJ = Join[allowedSLJ, termsSLJ];
2372     idx1++
2373   ];
2374   SortBy[allowedSLJ, Last]
2375 ]
2376
2377 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.";
2378 AllowedNKSLJTerms[numE_] :=
2379 Module[{allowedSL, allowedNKSL, allowedSLJ, nn},
2380   allowedNKSL = AllowedNKSLTerms[numE];
2381   allowedSL = AllowedSLTerms[numE];
2382   allowedSLJ = {};
2383   For[
2384     nn = 1,
2385     nn <= Length[allowedSL],
2386     (
2387       termSL = allowedSL[[nn]];
2388       termNKSL = allowedNKSL[[nn]];
2389       termsSLJ =
2390         Table[{termNKSL, J},
2391           {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
2392         ];
2393       allowedSLJ = Join[allowedSLJ, termsSLJ];
2394       nn++
2395     )
2396   ];
2397   SortBy[allowedSLJ, Last]
2398 ]
2399
2400 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.";
2401 AllowedNKSLforJTerms[numE_, J_] := Module[
2402   {allowedSL, allowedNKSL, allowedSLJ, nn, termSL, termNKSL, termsSLJ},
2403   allowedNKSL = AllowedNKSLTerms[numE];
2404   allowedSL = AllowedSLTerms[numE];
2405   allowedSLJ = {};
2406   For[
2407     nn = 1,

```

```

2408     nn <= Length[allowedSL],
2409     (
2410         termSL = allowedSL[[nn]];
2411         termNKSL = allowedNKSL[[nn]];
2412         termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[termSL],
2413             {{termNKSL, J}},
2414             {}
2415         ];
2416         allowedSLJ = Join[allowedSLJ, termsSLJ];
2417         nn++
2418     )
2419 ];
2420 Return[allowedSLJ]
2421 ];
2422
2423 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}.";
2424 AllowedSLJMTerms[numE_] := Module[
2425     {allowedSLJ, allowedSLJM, termSLJ, termsSLJM, nn},
2426     allowedSLJ = AllowedSLJTerms[numE];
2427     allowedSLJM = {};
2428     For[
2429         nn = 1,
2430         nn <= Length[allowedSLJ],
2431         nn++,
2432         (
2433             termSLJ = allowedSLJ[[nn]];
2434             termsSLJM =
2435                 Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
2436                     {M, -termSLJ[[3]], termSLJ[[3]]}
2437             ];
2438             allowedSLJM = Join[allowedSLJM, termsSLJM];
2439         )
2440     ];
2441     Return[SortBy[allowedSLJM, Last]];
2442 ]
2443
2444 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}.";
2445 AllowedNKSLJMforJMTerms[numE_, J_, MJ_] :=
2446     Module[{allowedSL, allowedNKSL, allowedSLJM, nn},
2447         allowedNKSL = AllowedNKSLTerms[numE];
2448         allowedSL = AllowedSLTerms[numE];
2449         allowedSLJM = {};
2450         For[
2451             nn = 1,
2452             nn <= Length[allowedSL],
2453             termSL = allowedSL[[nn]];
2454             termNKSL = allowedNKSL[[nn]];
2455             termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
2456                 <= J
2457                 <= Total[termSL]
2458                 && (Abs[MJ] <= J)
2459             ),
2460                 {{termNKSL, J, MJ}},
2461                 {}];
2462             allowedSLJM = Join[allowedSLJM, termsSLJ];
2463             nn++
2464         ];

```

```

2465     Return[allowedSLJM];
2466 ]
2467
2468 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}.";
2469 AllowedNKSLJMforJTerms[numE_, J_] :=
2470 Module[{MJs, labelsAndMomenta, termsWithJ},
2471 (
2472   MJs = AllowedMforJ[J];
2473   (* Pair LS labels and their {S,L} momenta *)
2474   labelsAndMomenta = ({#, FindSL[#]}) & /@ AllowedNKSLTerms[numE];
2475   (* A given term will contain J if |L-S|<=J<=L+S *)
2476   ContainsJ[{SL_String, {S_, L_}}] := (Abs[S - L] <= J <= (S + L));
2477   (* Keep just the terms that satisfy this condition *)
2478   termsWithJ = Select[labelsAndMomenta, ContainsJ];
2479   (* We don't want to keep the {S,L} *)
2480   termsWithJ = {#[[1]], J} & /@ termsWithJ;
2481   (* This is just a quick way of including up all the MJ values *)
2482   Return[Flatten /@ Tuples[{termsWithJ, MJs}]]
2483 )
2484 ]
2485
2486 AllowedMforJ::usage = "AllowedMforJ[J] is shorthand for Range[-J, J, 1].";
2487 AllowedMforJ[J_] := Range[-J, J, 1];
2488
2489 AllowedJ::usage = "AllowedJ[numE] returns the total angular momenta J that appear in the f^
numE configuration.";
2490 AllowedJ[numE_] := Table[J, {J, MinJ[numE], MaxJ[numE]}];
2491
2492 Seniority::usage="Seniority[LS] returns the seniority of the given term."
2493 Seniority[LS_] := FindNKLSTerm[LS][[1, 2]]
2494
2495 FindNKLSTerm::usage = "Given the string LS FindNKLSTerm[SL] returns all the terms that are
compatible with it. This is only for f^n configurations. The provided terms might belong
to more than one configuration. The function returns a list with elements of the form {LS
, seniority, W, U}.";
2496 FindNKLSTerm[SL_] := Module[
2497   {NKterms, n},
2498   n = 7;
2499   NKterms = {};
2500   Map[
2501     If[! StringFreeQ[First[#], SL],
2502       If[ToExpression[Part[# , 2]] <= n,
2503         NKterms = Join[NKterms, {#}, 1]
2504       ]
2505     ] &,
2506     fnTermLabels
2507   ];
2508   NKterms = DeleteCases[NKterms, {}];
2509   NKterms]
2510
2511 Options[ParseTermLabels] = {"Export" -> True};
2512 ParseTermLabels::usage="ParseTermLabels[] parses the labels for the terms in the f^n
configurations based on the labels for the f6 and f7 configurations. The function returns
a list whose elements are of the form {LS, seniority, W, U}.";
2513 ParseTermLabels[OptionsPattern[]] := Module[
2514   {labelsTextData, fNtextLabels, nielsonKosterLabels, seniorities, RacahW, RacahU},
2515   (
2516     labelsTextData = FileNameJoin[{moduleDir, "data", "NielsonKosterLabels_f6_f7.txt"}];

```

```

2517 fnTextLabels = Import[labelsTextData];
2518 nielsonKosterLabels = Partition[StringSplit[fnTextLabels], 3];
2519 termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
2520 seniorities = Map[ToExpression[Part[#, {2}]] &, nielsonKosterLabels];
2521 racahW =
2522   Map[
2523     StringTake[
2524       Flatten[StringCases[Part[#, {3}],
2525         {" ~ ~ DigitCharacter ~ ~ DigitCharacter ~ ~ DigitCharacter ~ ~ "}],
2526       {2, 4}
2527     ] &,
2528     nielsonKosterLabels];
2529 racahU =
2530   Map[
2531     StringTake[
2532       Flatten[StringCases[Part[#, {3}],
2533         {" ~ ~ DigitCharacter ~ ~ DigitCharacter ~ ~ "}],
2534       {2, 3}
2535     ] &,
2536     nielsonKosterLabels];
2537 fnTermLabels = Join[termLabels, seniorities, racahW, racahU, 2];
2538 fnTermLabels = Sort[fnTermLabels];
2539 If[OptionValue["Export"],
2540   (
2541     broadFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
2542     Export[broadFname, fnTermLabels];
2543   )
2544 ];
2545 Return[fnTermLabels];
2546 )
2547 ]
2548
2549 (* ##### Term management ##### *)
2550 (* ##### *)
2551
2552 Options[LoadParameters] = {
2553   "Source" -> "Carnall",
2554   "Free Ion" -> False,
2555   "gs" -> 2.002319304386
2556 };
2557 LoadParameters::usage = "LoadParameters[ln] takes a string with the symbol the element of a
trivalent lanthanide ion and returns model parameters for it. It is based on the data for
LaF3. If the option \"Free Ion\" is set to True then the function sets all crystal field
parameters to zero. Through the option \"gs\" it allows modifying the electronic
gyromagnetic ratio.";
2558 LoadParameters[Ln_String, OptionsPattern[]]:=
2559 Module[{source, params},
2560   (
2561     source = OptionValue["Source"];
2562     params = Which[source=="Carnall",
2563       (Association[Carnall["data"][Ln]])
2564     ];
2565     (*If a free ion then all the parameters from the crystal field are set to zero*)
2566     If[OptionValue["Free Ion"],
2567       Do[params[cfSymbol] = 0,
2568         {cfSymbol, cfSymbols}
2569       ]
2570     ];
2571     params[F0] = 0;
2572     params[M2] = 0.56 * params[M0]; (* See Carnall 1989, Table I, caption, probably fixed
based on HF values*)
2573     params[M4] = 0.31 * params[M0]; (* See Carnall 1989, Table I, caption, probably fixed

```

```

based on HF values*)
2574     params[P0] = 0;
2575     params[P4] = 0.5 * params[P2]; (* See Carnall 1989, Table I, caption, probably fixed
based on HF values*)
2576     params[P6] = 0.1 * params[P2]; (* See Carnall 1989, Table I, caption, probably fixed
based on HF values*)
2577     params[gs] = OptionValue["gs"];
2578     {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[F0], params[F2], params[
F4], params[F6]];
2579     params[E0] = 0;
2580     Return[params];
2581 )
2582 ];
2583
2584 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.";
2585
2586 HoleElectronConjugation[params_] :=
2587     Module[{newparams = params},
2588     (
2589         flipSignsOf = {ζ, T2, T3, T4, T6, T7, T8};
2590         flipSignsOf = Join[flipSignsOf, cfSymbols];
2591         flipped =
2592             Table[{flipper -> - newparams[flipper]},
2593                 {flipper, flipSignsOf}
2594             ];
2595         nonflipped =
2596             Table[{flipper -> newparams[flipper]},
2597                 {flipper, Complement[Keys[newparams], flipSignsOf]}
2598             ];
2599         flippedParams = Association[Join[nonflipped, flipped]];
2600         Return[flippedParams];
2601     )
2602 ];
2603
2604
2605 SolveStates::usage = "SolveStates[nf, params] solves the energy values and states for an
atom with nf f-electrons. params is an association with the parameters of the specific
ion under study.
2606 This function requires files for pre-computed energy matrix tables that provide the symbols
JJBlockMatrixTable[_, _, _, _, _].
2607 The optional parameter \"maxEigenvalues\" (default: \"All\") specifies the number of
eigenvalues to be returned. If maxE is \"All\" then all eigenvalues are returned. If maxE
is positive then the k largest (in absolute value) eigenvalues are returned. If maxE is
negative then the k smallest (in absolute value) eigenvalues are returned.
2608 To account for configurations f^n with n > 7, particle-hole dualities are enforced for ζ
and T_i.
2609 The unit for the returned energies is cm^-1.
2610
2611 Parameters
2612 -----
2613 nf (int) : Number of f-electrons.
2614 params (association) : Parameters of the ion under study.
2615
2616 Returns
2617 -----
2618 {eigenstates, basis} (list): eigenstates is a list wher each element is a list with two
elements, the first element being the energy eigenvalue and the second being a list that

```

represents the eigenvector in the computational basis. basis is a list of lists that represent the computational basis. The elements of the basis are lists of the form {{SL, J}, mJ}, I}, where SL is given a string.

Options

```

\ "Return Symbolic Matrix\ " (bool) : If True then the function returns instead a list with
the three elements {levels, basis, symbolicMatrix}.
\ "maxEigenvalues\ " (int) : Number of eigenvalues to be returned. If \ "All\ " then all
eigenvalues are returned. If positive then the k largest (in absolute value) eigenvalues
are returned. If negative then the k smallest (in absolute value) eigenvalues are
returned.

```

References:

1. Sign inversion for ζ : Wybourne, Spectroscopic Properties of Rare Earths.
2. Sign inversion for {T2, T3, T4, T6, T7, T8}: Hansen and Judd, Matrix Elements of Scalar Three Electron Operators for the Atomic f Shell."

```

Options[SolveStates] = {"Return Symbolic Matrix" -> False,
                        "maxEigenvalues" -> "All"};
SolveStates[nf_, params0_, OptionsPattern[]]:= Module[
  {n, ii, jj, JMvals},
  maxEigen = OptionValue["maxEigenvalues"];
  (*****
  (*hole-particle equivalence enforcement*)
  n = nf;
  If[nf>7,
    (
      n = 14 - nf;
      params = HoleElectronConjugation[params0];
    ),
    params = params0;
  ];
  (*hole-particle equivalence enforcement*)
  (*****
  (*Load symbolic expressions for energy sub-matrices.*)
  Get[JJBBlockMatrixFileName[n, "FilenameAppendix" -> fileApp]];
  (*Patch together the entire matrix representation in block-diagonal form.*)
  ThisEnergyMatrix = ConstantArray[0, {Length[AllowedJ[n]], Length[AllowedJ[n]]}];
  Do[ThisEnergyMatrix[[jj, ii]] = JJBBlockMatrixTable[{n, AllowedJ[n][[ii]], AllowedJ[n][[jj]]}],
    {ii, 1, Length[AllowedJ[n]]},
    {jj, 1, Length[AllowedJ[n]]}
  ];
  ThisEnergyMatrix = ArrayFlatten[ThisEnergyMatrix];
  symbolicMatrix = ThisEnergyMatrix;
  ThisEnergyMatrix = ReplaceInSparseArray[ThisEnergyMatrix, params];
  problemSize = Dimensions[ThisEnergyMatrix][[1]];
  If[maxEigen!="All",
    (
      If[Abs[maxEigen]>problemSize, maxEigen="All"]
    )
  ];
  PrintTemporary["The energy matrix has dimensions:", Dimensions[ThisEnergyMatrix]];
  (*Solve for eigenvalues and eigenvectors.*)
  {EigenvalueJM, EigenvectorJM} =
    If[maxEigen=="All",
      Eigensystem[ThisEnergyMatrix],
      Eigensystem[ThisEnergyMatrix, maxEigen]
    ];
  EigenvalueJM = Re[EigenvalueJM];
  (*There might be a very small imaginary part.*)

```



```

2672 (*Parse the results for the eigenvectors in terms of the ordered basis being used.*)
2673 basis = {};
2674 Do[basis = Join[basis, EnergyStatesTable[{n, AllowedJ[n][[nn]]}],
2675 {nn, 1, Length[AllowedJ[n]]}
2676 ];
2677 levels = {};
2678 Do[levels = Join[levels, {{EigenvalueJM[[nn]], EigenvectorJM[[nn]]}}];,
2679 {nn, 1, Length[EigenvalueJM]}
2680 ];
2681 If[OptionValue["Return Symbolic Matrix"],
2682 Return[{levels, basis, symbolicMatrix}]
2683 ];
2684 Return[{levels, basis}];
2685 ];
2686
2687 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.
2688
2689 Parameters
2690 -----
2691 numE (int) : Number of f-electrons.
2692
2693 Options
2694 -----
2695 \"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.
2696
2697 Returns
2698 -----
2699 {rmsDifference, gtEnergies, cfenergies, ln, carnallAssignments, {fstates, basis,
symbolicMatrix}} (list): with
2700 rmsDifference (float) : The root-mean-square difference between the calculated values
from Carnall and the ones computed here.
2701 gtEnergies (list) : The calculated values for the energy levels as quoted by Carnall.
2702 cfenergies (list) : The calculated values for the energy levels as calculated here.
2703 ln (string) : The symbol of the lanthanide ion.
2704 carnallAssignments (list) : The assignments of the energy levels as quoted by Carnall.
2705 {fstates, basis, symbolicMatrix} (list) : The eigenstates, basis and symbolic matrix as
calculated here.
2706 ";
2707 Options[IonSolverLaF3] = {"Include Spin-Spin" -> True};
2708 IonSolverLaF3[numE_, OptionsPattern[]] := (
2709     spinspace = OptionValue["Include Spin-Spin"];
2710     host = "LaF3";
2711     ln = StringSplit["Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb"][[numE]];
2712     terms = AllowedNKSLJTerms[Min[numE, 14 - numE]];
2713     expData = Flatten[#["Exp (1/cm)"] & /@ Values[Carnall["appendix:" <> ln <> ":Association"
]]];
2714
2715 (*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*})
2716 simplifier = {
2717     B12 -> 0, B14 -> 0, B16 -> 0, B34 -> 0, B36 -> 0, B56 -> 0,
2718     S12 -> 0, S14 -> 0, S16 -> 0, S22 -> 0, S24 -> 0, S26 -> 0,
2719     S34 -> 0, S36 -> 0, S44 -> 0, S46 -> 0, S56 -> 0, S66 -> 0,
2720     T11 -> 0, T12 -> 0, T14 -> 0, T15 -> 0, T16 -> 0, T18 -> 0, T11p -> 0,
2721     T17 -> 0, T19 -> 0
2722 };
2723 eTofs = (#[[1]] -> #[[2]]) & /@ Transpose[{{E0, E1, E2, E3}, FtoE[F0, F2, F4, F6]}}];
2724 ham = Normal[HamMatrixAssembly[numE, 0]];

```

```

2725 simpleHam = ham /. simplifier;
2726 simpleHam = simpleHam /. eTofs;
2727 hamParams = DeleteDuplicates[Flatten[Variables /@ simpleHam]];
2728 ham = Normal[HamMatrixAssembly[numE, 0]];
2729 termNames = First /@ terms;
2730 termSimplifier =
2731   Table[
2732     termN -> If[StringLength[termN] == 3,
2733       StringTake[termN, {1, 2}],
2734       termN
2735     ],
2736     {termN, termNames}
2737   ];
2738
2739 (*Load the parameters from Carnall*)
2740 params = LoadParameters[ln, "Free Ion" -> False];
2741 (*Enforce the override to the spin-spin contribution to the magnetic interactions*)
2742 params[[SigmaSS]] = If[spinspin, 1, 0];
2743 (*Everything that is not given is set to zero*)
2744 params = ParamPad[params, "Print" -> True];
2745
2746 {fstates, basis, symbolicMatrix} =
2747   SolveStates[params[nf], 0, params, "Return Symbolic Matrix" -> True];
2748 symbolicMatrix =
2749   If[spinspin,
2750     ReplaceInSparseArray[symbolicMatrix, {\[Sigma]SS -> 1}],
2751     ReplaceInSparseArray[symbolicMatrix, {\[Sigma]SS -> 0}]
2752   ];
2753 fstates = ShiftedLevels[fstates];
2754 fstates = SortBy[fstates, First];
2755 cfenergies = First /@ fstates;
2756 cfenergies = Chop[cfenergies];
2757 If[OddQ[numE],
2758   (
2759     cfenergies = cfenergies[;; ;; 2]];
2760 )
2761 ];
2762
2763 mainKey = StringTemplate["appendix:'Ln':Association"] [<|"Ln" -> ln|>];
2764 lnData = Carnall[mainKey];
2765 carnalKeys = lnData // Keys;
2766 repetitions = Length[lnData[#]["Calc (1/cm)"]] & /@ carnalKeys;
2767 carnallAssignments =
2768   First /@ Carnall["appendix:" <> ln <> ":RawTable"];
2769
2770 carnalKey = StringTemplate["appendix:'Ln':Calculated"] [<|"Ln" -> ln|>];
2771 gtEnergies = Sort[Carnall[carnalKey]];
2772 diffs = Sort[cfenergies][;; Length[gtEnergies]] - gtEnergies;
2773 rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
2774
2775 Return[{rmsDifference, gtEnergies, cfenergies, ln, carnallAssignments, {fstates, basis,
symbolicMatrix}}]
2776 )
2777
2778 FastIonSolverLaF3::usage =
2779   "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.
2780
2781 The function returns a list with seven elements {rmsDifference, carnallEnergies,
eigenEnergies, ln, carnallAssignments, eigensys, basis}. Where:

```

```

2782 rmsDifference is the root mean squared difference between the calculated values and those
2783 quoted by Carnall
2784
2785 carnallEnergies are the quoted calculated values from Carnall;
2786
2787 eigenEnergies are the calculated energies (in the case of an odd number of electrons the
2788 kramers degeneracy has been elided from this list);
2789
2790 ln is simply a string labelling the corresponding lanthanide;
2791
2792 carnallAssignments is a list of strings providing the term assignments that Carnall
2793 assumed,
2794
2795 eigensys is a list of tuples where the first element is the energy corresponding to the
2796 eigenvector given as the second element;
2797
2798 basis a list that specifies the basis in which the Hamiltonian was constructed and
2799 diagonalized.
2800
2801 ";
2802 Options[FastIonSolverLaF3] = {
2803   "MakeNotebook" -> True,
2804   "NotebookSave" -> True,
2805   "HTMLSave" -> False,
2806   "eigenstateTruncationProbability" -> 0.9,
2807   "Include spin-spin" -> True,
2808   "Max Eigenstates in Table" -> 100,
2809   "Sparse" -> True,
2810   "PrintFun" -> Print,
2811   "SaveData" -> True,
2812   "paramFiddle" -> {},
2813   "Append to Filename" -> ""
2814 };
2815 FastIonSolverLaF3[numE_, OptionsPattern[]] := Module[{
2816   makeNotebook, eigenstateTruncationProbability, spinspin, host,
2817   ln, terms, termNames, carnallEnergies, eigenEnergies, simplerStateLabels,
2818   eigensys, basis, assignmentMatches, stateLabels, carnallAssignments},
2819 (
2820   PrintFun = OptionValue["PrintFun"];
2821   makeNotebook = OptionValue["MakeNotebook"];
2822   eigenstateTruncationProbability = OptionValue["eigenstateTruncationProbability"];
2823   maxStatesInTable = OptionValue["Max Eigenstates in Table"];
2824   spinspin = OptionValue["Include spin-spin"];
2825   host = "LaF3";
2826   paramFiddle = OptionValue["paramFiddle"];
2827   ln = theLanthanides[[numE]];
2828   terms = AllowedNKSLJTerms[Min[numE, 14 - numE]];
2829   termNames = First /@ terms;
2830   (* For labeling the states, the degeneracy in some of the terms is elided *)
2831   PrintFun["> Calculating simpler term labels ..."];
2832   termSimplifier =
2833     Table[termN -> If[StringLength[termN] == 3,
2834       StringTake[termN, {1, 2}],
2835       termN
2836     ],
2837     {termN, termNames}
2838   ];
2839
2840   (*Load the parameters from Carnall*)
2841   PrintFun["> Loading the fit parameters from Carnall ..."];
2842   params = LoadParameters[ln, "Free Ion" -> False];
2843   If[numE>7,

```

```

2839     (
2840         PrintFun["> Conjugating the parameters accounting for the hole-particle equivalence
..."];
2841         params = HoleElectronConjugation[params];
2842         params[t2Switch] = 0;
2843     ),
2844     params[t2Switch] = 1;
2845 ];
2846
2847 Do[params[key] = paramFiddle[key],
2848     {key, Keys[paramFiddle]}
2849 ];
2850
2851 (*Import the symbolic Hamiltonian*)
2852 PrintFun["> Loading the symbolic Hamiltonian for this configuration ..."];
2853 startTime = Now;
2854 numH = 14 - numE;
2855 numEH = Min[numE, numH];
2856 simpleHam =
2857     If[ValueQ[symbolicHamiltonians],
2858         (
2859             If[MemberQ[Keys[symbolicHamiltonians], numEH],
2860                 symbolicHamiltonians[numEH],
2861                 Import["./hams/SymbolicMatrix-f" <> ToString[numEH] <> ".m"]
2862             ]
2863         ),
2864         Import["./hams/SymbolicMatrix-f" <> ToString[numEH] <> ".m"]
2865     ];
2866 endTime = Now;
2867 loadTime = QuantityMagnitude[endTime - startTime, "Seconds"];
2868 PrintFun[">> Loading the symbolic Hamiltonian took ", loadTime, " seconds."];
2869
2870 (*Enforce the override to the spin-spin contribution to the magnetic interactions*)
2871 params[[Sigma]SS] = If[spinspin, 1, 0];
2872
2873 (*Everything that is not given is set to zero*)
2874 params = ParamPad[params, "Print" -> False];
2875 PrintFun[params];
2876 (* numHam = simpleHam /. params; *)
2877 numHam = ReplaceInSparseArray[simpleHam, params];
2878 If[Not[OptionValue["Sparse"]],
2879     numHam = Normal[numHam]
2880 ];
2881 PrintFun["> Calculating the SLJ basis ..."];
2882 basis = BasisLSJMJ[numE];
2883
2884 (*Remove numerical noise*)
2885 PrintFun["> Diagonalizing the numerical Hamiltonian ..."];
2886 startTime = Now;
2887 eigensys = Eigensystem[numHam];
2888 endTime = Now;
2889 diagonalTime = QuantityMagnitude[endTime - startTime, "Seconds"];
2890 PrintFun[">> Diagonalization took ", diagonalTime, " seconds."];
2891 eigensys = Chop[eigensys];
2892 eigensys = Transpose[eigensys];
2893
2894 (*Shift the baseline energy*)
2895 eigensys = ShiftedLevels[eigensys];
2896 (*Sort according to energy*)
2897 eigensys = SortBy[eigensys, First];
2898 (*Grab just the energies*)
2899 eigenEnergies = First /@ eigensys;

```

```

2900
2901 (*Energies are doubly degenerate in the case of odd number of electrons, keep only one*)
2902 If[OddQ[numE],
2903   (
2904     PrintFun["> Since there's an odd number of electrons energies come in pairs, taking
just one for each pair ..."];
2905     eigenEnergies = eigenEnergies[;; ;; 2];
2906   )
2907 ];
2908
2909 (*Compare against the data quoted by Bill Carnall*)
2910 PrintFun["> Comparing against the data from Carnall ..."];
2911 mainKey = StringTemplate["appendix: 'Ln': Association"] [ <|"Ln" -> ln|>];
2912 lnData = Carnall[mainKey];
2913 carnalKeys = lnData // Keys;
2914 repetitions = Length[lnData[#]["Calc (1/cm)"]] & /@ carnalKeys;
2915 carnallAssignments = First /@ Carnall["appendix:" <> ln <> ":RawTable"];
2916 carnalKey = StringTemplate["appendix: 'Ln': Calculated"] [ <|"Ln" -> ln|>];
2917 carnallEnergies = Carnall[carnalKey];
2918
2919 (* For the difference take as many energies as quoted by Bill*)
2920 eigenEnergies = eigenEnergies + carnallEnergies[[1]];
2921 diffs = Sort[eigenEnergies][;; Length[carnallEnergies]] - carnallEnergies;
2922 (* Remove the differences where the appendix tables have elided values*)
2923 rmsDifference = Sqrt[Mean[(Select[diffs, FreeQ[#, Missing[]] &])^2]];
2924 titleTemplate = StringTemplate[
2925   "Energy Level Diagram of \!\\(*SuperscriptBox[\\('ion'\\), \\(\\(3\\)\\(\\(\\)\\)\\)\\)"];
2926 title = titleTemplate[ <|"ion" -> ln|>];
2927 parsedStates = ParseStates[eigensys, basis];
2928 If[OddQ[numE],
2929   parsedStates = parsedStates[;; ;; 2]];
2930
2931 stateLabels = #[[-1]] & /@ parsedStates;
2932 simplerStateLabels = ((#[[2]] /. termSimplifier) <> ToString[#[[3]], InputForm]) & /@
parsedStates;
2933
2934 PrintFun[">> Truncating eigenvectors to given probability ..."];
2935 startTime = Now;
2936 truncatedStates = ParseStatesByProbabilitySum[eigensys, basis,
2937   eigenstateTruncationProbability,
2938   0.01];
2939 endTime = Now;
2940 truncationTime = QuantityMagnitude[endTime - startTime, "Seconds"];
2941 PrintFun[">>> Truncation took ", truncationTime, " seconds."];
2942
2943 If[makeNotebook,
2944   (
2945     PrintFun["> Putting together results in a notebook ..."];
2946     energyDiagram = Framed[
2947       EnergyLevelDiagram[eigensys, "Title" -> title,
2948         "Background" -> White]
2949       , Background -> White, FrameMargins -> 50];
2950     appToFname = OptionValue["Append to Filename"];
2951     PrintFun[">> Comparing the term assignments between qlanth and Carnall ..."];
2952     assignmentMatches =
2953     If[StringContainsQ[#[[1]], #[[2]], "\[Checkmark]", "X"] & /@
2954       Transpose[{carnallAssignments, simplerStateLabels[;; Length[carnallAssignments
]]}];
2955     assignmentMatches = {"\[Checkmark]",
2956       Count[assignmentMatches, "\[Checkmark]"], {"X",
2957       Count[assignmentMatches, "X"]}};
2958     labelComparison = (If[StringContainsQ[#[[1]], #[[2]], "\[Checkmark]", "X"] & /@

```

```

2959 Transpose[{carnallAssignments,
2960   simplrStateLabels[;; Length[carnallAssignments]]]}];
2961 labelComparison =
2962 PadRight[labelComparison, Length[simplrStateLabels], "-"];
2963
2964 statesTable =
2965 Grid[Prepend[{Round#[[1]], #[[2]]} & /@
2966   truncatedStates[;; Min[Length[eigensys], maxStatesInTable]]], {"Energy/\!\(\*
SuperscriptBox[\(cm\), \(-1\)]\)\"",
2967   "\[Psi]"}], Frame -> All, Spacings -> {2, 2},
2968   FrameStyle -> Blue,
2969   Dividers -> {{False, True, False}, {True, True}}];
2970 DefaultIfMissing[expr_] := If[FreeQ[expr, Missing[]], expr, "NA"];
2971 PrintFun[">> Rounding the energy differences for table presentation ..."];
2972 roundedDiffs = Round[diffs, 0.1];
2973 roundedDiffs = PadRight[roundedDiffs, Length[simplrStateLabels], "-"];
2974 roundedDiffs = DefaultIfMissing /@ roundedDiffs;
2975 diffs = PadRight[diffs, Length[simplrStateLabels], "-"];
2976 diffs = DefaultIfMissing /@ diffs;
2977 diffTableData = Transpose[{simplrStateLabels, eigenEnergies,
2978   labelComparison,
2979   PadRight[carnallAssignments, Length[simplrStateLabels], "-"],
2980   DefaultIfMissing /@ PadRight[carnallEnergies, Length[simplrStateLabels], "-"],
2981   roundedDiffs}];
2982 diffTable =
2983 TableForm[diffTableData,
2984   TableHeadings -> {None, {"qlanth",
2985     "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)\"", "", "Carnall",
2986     "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)\"",
2987     "\[CapitalDelta]E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"}];
2988
2989 diffs = Sort[eigenEnergies][;; Length[carnallEnergies]] - carnallEnergies;
2990 notBad = FreeQ[#, Missing[]] & /@ diffs;
2991 diffs = Pick[diffs, notBad];
2992 diffHistogram =
2993 Histogram[diffs, Frame -> True, ImageSize -> 800,
2994   AspectRatio -> 1/3, FrameStyle -> Directive[16],
2995   FrameLabel -> {"(qlanth-carnall)/Ky", "Freq"}];
2996 rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
2997 labelTemplate =
2998 StringTemplate[
2999   "\!\(\*SuperscriptBox[\(‘ln‘), \(\{3\}\(+\)\)]\)");
3000 diffData = diffs;
3001 diffLabels = simplrStateLabels[;; Length[notBad]];
3002 diffLabels = Pick[diffLabels, notBad];
3003 diffPlot = Framed[
3004   ListLabelPlot[diffData,
3005     diffLabels,
3006     Frame -> True,
3007     PlotRange -> All,
3008     ImageSize -> 1200,
3009     AspectRatio -> 1/3,
3010     FrameLabel -> {"",
3011       "(qlanth-carnall) / \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)\"",
3012     PlotMarkers -> "OpenMarkers",
3013     PlotLabel ->
3014       Style[labelTemplate[<|"ln" -> ln|>] <> " | " <> "\[Sigma]=" <>
3015         ToString[Round[rmsDifference, 0.01]] <>
3016         " \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)n", 20],
3017     Background -> White
3018   ],
3019   Background -> White,

```

```

3020     FrameMargins -> 50
3021 ];
3022 nb = CreateDocument[{
3023     TextCell[Style[DisplayForm[SuperscriptBox[host <> ":" <> ln, "3+" ]]], "Title",
TextAlignment -> Center],
3024     TextCell["Energy Diagram", "Section", TextAlignment -> Center],
3025     TextCell[energyDiagram, TextAlignment -> Center],
3026     TextCell["Multiplet Assignments & Energy Levels", "Section", TextAlignment ->
Center],
3027     TextCell[diffHistogram, TextAlignment -> Center],
3028     TextCell[diffPlot, "Output", TextAlignment -> Center],
3029     TextCell[assignmentMatches, "Output", TextAlignment -> Center],
3030     TextCell[diffTable, "Output", TextAlignment -> Center],
3031     TextCell["Truncated Eigenstates", "Section", TextAlignment -> Center],
3032     TextCell["These are some of the resultant eigenstates which add up to at least a
total probability of " <> ToString[eigenstateTruncationProbability] <> ".", "Text",
TextAlignment -> Center],
3033     TextCell[statesTable, "Output", TextAlignment -> Center]
3034 },
3035 WindowSelected -> True,
3036 WindowTitle -> ln <> " in " <> "LaF3" <> appToFname,
3037 WindowSize -> {1600, 800}];
3038 If[OptionValue["SaveData"],
3039 (
3040     exportFname = FileNameJoin[{moduleDir,"calcs", ln <> " in " <> "LaF3" <>
appToFname <> ".m"}];
3041     SelectionMove[nb, After, Notebook];
3042     NotebookWrite[nb, Cell["Reload Data", "Section", TextAlignment -> Center]];
3043     NotebookWrite[nb, Cell[(
3044         "{rmsDifference, carnallEnergies, eigenEnergies, ln, carnallAssignments,
simplerStateLabels, eigensys, basis, truncatedStates} = Import[FileNameJoin[{
NotebookDirectory[],\"\" <> StringSplit[exportFname,\"/"][[-1]] <> \"\"];\"
3045         ),\"Input\"]];
3046         NotebookWrite[nb, Cell[(
3047             \"Manipulate[First[MinimalBy[truncatedStates, Abs[First[#] - energy] &]], {
energy,0}] \"
3048             ),\"Input\"]];
3049             SelectionMove[nb, Before, Notebook];
3050             Export[exportFname, {rmsDifference, carnallEnergies, eigenEnergies, ln,
carnallAssignments, simplerStateLabels, eigensys, basis, truncatedStates}];
3051             tinyexportFname = FileNameJoin[{moduleDir,\"calcs\", ln <> " in " <> "LaF3" <>
appToFname <> "- tiny.m"}];
3052             tinyExport = <|\"ln\"->ln,
3053                 \"carnallEnergies\"->carnallEnergies,
3054                 \"rmsDifference\"-> rmsDifference,
3055                 \"eigenEnergies\"-> eigenEnergies,
3056                 \"carnallAssignments\"-> carnallAssignments,
3057                 \"simplerStateLabels\" -> simplerStateLabels|>;
3058             Export[tinyexportFname, tinyExport];
3059         )
3060     ];
3061     If[OptionValue["NotebookSave"],
3062     (
3063         nbFname = FileNameJoin[{moduleDir,\"calcs\", ln <> " in " <> "LaF3" <> appToFname
<> ".nb"}];
3064         PrintFun[">> Saving notebook to ", nbFname, " ..."];
3065         NotebookSave[nb, nbFname];
3066     )
3067 ];
3068 If[OptionValue["HTMLSave"],
3069 (
3070     htmlFname = FileNameJoin[{moduleDir,\"calcs\", \"html\", ln <> " in " <> "LaF3" <>

```

```

appToFname <> ".html"]];
3071   PrintFun[">> Saving html version to ", htmlFname, " ..."];
3072   Export[htmlFname, nb];
3073   )
3074 ];
3075 )
3076 ];
3077
3078   Return[{rmsDifference, carnallEnergies, eigenEnergies, ln, carnallAssignments,
3079   simplifierStateLabels, eigensys, basis, truncatedStates}];
3080 ];
3081
3082 ShiftedLevels::usage = "
3083 ShiftedLevels[originalLevels] takes a list of levels of the form
3084 {{energy_1, coeff_vector_1},
3085 {energy_2, coeff_vector_2},
3086 ...}}
3087 and returns the same input except that now to every energy the minimum of all of them has
3088 been subtracted.";
3089 ShiftedLevels[originalLevels_] :=
3090   Module[{groundEnergy, shifted},
3091     groundEnergy = Sort[originalLevels][[1,1]];
3092     shifted      = Map[{#[[1]] - groundEnergy, #[[2]]} &, originalLevels];
3093     Return[shifted];
3094   ]
3095
3096 (* ##### Eigensystem analysis ##### *)
3097
3098 PrettySaunders::usage = "PrettySaunders[SL, J] produces a human-redeable symbol for the
3099 given Russel-Saunders term."
3100 PrettySaundersSLJ[{{SL_, J_}, MJ_}] := (
3101   If[StringQ[SL],
3102     {S, L} = FindSL[SL],
3103     {S, L} = SL
3104   ];
3105   Return[RowBox[
3106     {AdjustmentBox[Style[2*S + 1, Smaller], BoxBaselineShift -> -1,
3107       BoxMargins -> 0], AdjustmentBox[PrintL[L], BoxMargins -> -0.2],
3108     AdjustmentBox[
3109       Style[InputForm[J], Small, FontTracking -> "Narrow"],
3110       BoxBaselineShift -> 1, BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]
3111     }] // DisplayForm]]
3112
3113 PrettySaundersSLJmJ[{{SL_, J_}, mJ_}] := (If[
3114   StringQ[SL],
3115   ({S, L} = FindSL[SL];
3116     L = StringTake[SL, {2, -1}];
3117   ),
3118   {S, L} = SL];
3119   Return[
3120     RowBox[{AdjustmentBox[Style[2*S + 1, Smaller],
3121       BoxBaselineShift -> -1, BoxMargins -> 0],
3122       AdjustmentBox[PrintL[L], BoxMargins -> -0.2],
3123       AdjustmentBox[
3124         Style[{InputForm[J], mJ}, Small, FontTracking -> "Narrow"],
3125         BoxBaselineShift -> 1,
3126         BoxMargins -> {{0.7, 0}, {0.4, 0.4}}]] // DisplayForm]]
3127
3128 BasisVecInRusselSaunders::usage = "BasisVecInRusselSaunders[basisVec] takes a basis vector
3129 in the format {LSstring, Jval, mJval} and returns a human-readable symbol for the

```



```

corresponding Russel-Saunders term."
3128 BasisVecInRusselSaunders[basisVec_] := (
3129   {LSstring, Jval, mJval} = basisVec;
3130   Ket[PrettySaunders[LSstring, Jval], mJval]
3131 )
3132
3133 LSJMJTemplate =
3134   StringTemplate[
3135     "\!\(\*TemplateBox[{\nRowBox[{\\"LS\'\", \",\", \nRowBox[{\\"J\'\", \
3136     \\"=\", \\"J\'\"}], \",\", \nRowBox[{\\"mJ\'\", \\"=\", \\"mJ\'\"}]]}],\n\
3137     \\"Ket\[\]\"]];
3138 BasisVecInLSJMJ::usage = "BasisVecInLSJMJ[basisVec] takes a basis vector in the format {{
    LSstring, Jval}, mJval}, nucSpin} and returns a human-readable symbol for the
    corresponding LSJMJ term in the form |LS, J=..., mJ=...>."
3139 BasisVecInLSJMJ[basisVec_] := (
3140   {LSstring, Jval, mJval} = basisVec;
3141   LSJMJTemplate[<|
3142     "LS" -> LSstring,
3143     "J" -> ToString[Jval, InputForm],
3144     "mJ" -> ToString[mJval, InputForm]|>]
3145 );
3146
3147 ParseStates::usage = "ParseStates[states, basis] takes a list of eigenstates in terms of
    their coefficients in the given basis and returns a list of the same states in terms of
    their energy, LSJMJ symbol, J, mJ, S, L, LSJ symbol, and LS symbol. The LS symbol
    returned corresponds to the term with the largest coefficient in the given basis.";
3148 ParseStates[states_, basis_, OptionsPattern[]] := Module[{parsedStates},
3149   (
3150     parsedStates = Table[(
3151       {energy, eigenVec} = state;
3152       maxTermIndex = Ordering[Abs[eigenVec]][[-1]];
3153       {LSstring, Jval, mJval} = basis[[maxTermIndex]];
3154       LSJsymbol = Subscript[LSstring, {Jval, mJval}];
3155       LSJMJsymbol = LSstring <> ToString[Jval, InputForm];
3156       {S, L} = FindSL[LSstring];
3157       {energy, LSstring, Jval, mJval, S, L, LSJsymbol, LSJMJsymbol}
3158     ),
3159     {state, states}];
3160   Return[parsedStates]
3161 )
3162 ]
3163
3164 ParseStatesByNumBasisVecs::usage = "ParseStatesByNumBasisVecs[states, basis, numBasisVecs]
    takes a list of eigenstates in terms of their coefficients in the given basis and returns
    a list of the same states in terms of their energy and the coefficients of the
    numBasisVecs most significant basis vectors.";
3165 ParseStatesByNumBasisVecs[states_, basis_, numBasisVecs_, roundTo_ : 0.01] := (
3166   parsedStates = Table[(
3167     {energy, eigenVec} = state;
3168     energy = Chop[energy];
3169     probs = Round[Abs[eigenVec^2], roundTo];
3170     amplitudes = Round[eigenVec, roundTo];
3171     ordering = Ordering[probs];
3172     chosenIndices = ordering[[-numBasisVecs ;;]];
3173     majorComponents = basis[[chosenIndices]];
3174     majorProbabilities = amplitudes[[chosenIndices]];
3175     majorComponents = BasisVecInLSJMJ /@ majorComponents;
3176     majorRep = majorProbabilities . majorComponents;
3177     {energy, majorRep}
3178   ),
3179   {state, fstates}];
3180   Return[parsedStates]

```

```

3181 )
3182
3183 FindThresholdPosition::usage = "FindThresholdPosition[list, threshold] returns the position
    of the first element in list that is greater than threshold. If no such element exists,
    it returns the length of list. The elements of the given list must be in ascending order.
    ";
3184 FindThresholdPosition[list_, threshold_] :=
3185 Module[{position},
3186   position = Position[list, _?(# > threshold &), 1, 1];
3187   thrPos = If[Length[position] > 0,
3188     position[[1, 1]],
3189     Length[list]];
3190   If[thrPos == 0, Return[1], Return[thrPos+1]]
3191 ]
3192
3193 ParseStateByProbabilitySum[{energy_, eigenVec_}, probSum_, roundTo_:0.01, maxParts_:20] :=
    Compile[
3194   {{energy, _Real, 0},{eigenVec, _Complex, 1}, {probSum, _Real, 0}, {roundTo, _Real, 0}, {
    maxParts, _Integer, 0}},
3195   Module[
3196     {numStates, state, amplitudes, probs, ordering,
3197     orderedProbs, truncationIndex, accProb, thresholdIndex, chosenIndices, majorComponents,
3198     majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
3199   (
3200     numStates      = Length[eigenVec];
3201     (*Round them up*)
3202     amplitudes      = Round[eigenVec, roundTo];
3203     probs           = Round[Abs[eigenVec^2], roundTo];
3204     ordering         = Reverse[Ordering[probs]];
3205     (*Order the probabilities from high to low*)
3206     orderedProbs    = probs[[ordering]];
3207     (*To speed up Accumulate, assume that only as much as maxParts will be needed*)
3208     truncationIndex = Min[maxParts, Length[orderedProbs]];
3209     orderedProbs    = orderedProbs[;;truncationIndex];
3210     (*Accumulate the probabilities*)
3211     accProb         = Accumulate[orderedProbs];
3212     (*Find the index of the first element in accProb that is greater than probSum*)
3213     thresholdIndex  = Min[Length[accProb], FindThresholdPosition[accProb, probSum]];
3214     (*Grab all the indices up till that one*)
3215     chosenIndices   = ordering[;; thresholdIndex];
3216     (*Select the corresponding elements from the basis*)
3217     majorComponents = basis[[chosenIndices]];
3218     (*Select the corresponding amplitudes*)
3219     majorAmplitudes = amplitudes[[chosenIndices]];
3220     (*Take their absolute value*)
3221     absMajorAmplitudes = Abs[majorAmplitudes];
3222     (*Make sure that there are no effectively zero contributions*)
3223     notnullAmplitudes = Flatten[Position[absMajorAmplitudes, x_ /; x != 0]];
3224     (* majorComponents      = PrettySaundersSLJmJ[{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents;
    *)
3225     majorComponents = PrettySaundersSLJmJ /@ majorComponents;
3226     majorAmplitudes = majorAmplitudes[[notnullAmplitudes]];
3227     (*Make them into Kets*)
3228     majorComponents = Ket /@ majorComponents[[notnullAmplitudes]];
3229     (*Multiply and add to build the final Ket*)
3230     majorRep        = majorAmplitudes . majorComponents;
3231   );
3232   Return[{energy, majorRep}]
3233 ],
3234   CompilationTarget -> "C",
3235   RuntimeAttributes -> {Listable},
3236   Parallelization -> True,

```

```

3237 RuntimeOptions -> "Speed"
3238 ];
3239
3240 ParseStatesByProbabilitySum::usage = "ParseStatesByProbabilitySum[eigensys, basis, probSum]
    takes a list of eigenstates in terms of their coefficients in the given basis and
    returns a list of the same states in terms of their energy and the coefficients of the
    basis vectors that sum to at least probSum.";
3241 ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ : 0.01, maxParts_ : 20] :=
    Module[
3242 {parsedByProb, numStates, state, energy, eigenVec, amplitudes, probs, ordering,
3243 orderedProbs, truncationIndex, accProb, thresholdIndex, chosenIndices, majorComponents,
3244 majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
3245 (
3246 numStates = Length[eigensys];
3247 parsedByProb = Table[(
3248 state = eigensys[[idx]];
3249 {energy, eigenVec} = state;
3250 (*Round them up*)
3251 amplitudes = Round[eigenVec, roundTo];
3252 probs = Round[Abs[eigenVec^2], roundTo];
3253 ordering = Reverse[Ordering[probs]];
3254 (*Order the probabilities from high to low*)
3255 orderedProbs = probs[[ordering]];
3256 (*To speed up Accumulate, assume that only as much as maxParts will be needed*)
3257 truncationIndex = Min[maxParts, Length[orderedProbs]];
3258 orderedProbs = orderedProbs[[;;truncationIndex]];
3259 (*Accumulate the probabilities*)
3260 accProb = Accumulate[orderedProbs];
3261 (*Find the index of the first element in accProb that is greater than probSum*)
3262 thresholdIndex = Min[Length[accProb], FindThresholdPosition[accProb, probSum]];
3263 (*Grab all the indices up till that one*)
3264 chosenIndices = ordering[[;; thresholdIndex]];
3265 (*Select the corresponding elements from the basis*)
3266 majorComponents = basis[[chosenIndices]];
3267 (*Select the corresponding amplitudes*)
3268 majorAmplitudes = amplitudes[[chosenIndices]];
3269 (*Take their absolute value*)
3270 absMajorAmplitudes = Abs[majorAmplitudes];
3271 (*Make sure that there are no effectively zero contributions*)
3272 notnullAmplitudes = Flatten[Position[absMajorAmplitudes, x_ /; x != 0]];
3273 (* majorComponents = PrettySaundersSLJmJ[{#[[1]],#[[2]],#[[3]]}] & /@
majorComponents; *)
3274 majorComponents = PrettySaundersSLJmJ /@ majorComponents;
3275 majorAmplitudes = majorAmplitudes[[notnullAmplitudes]];
3276 (*Make them into Kets*)
3277 majorComponents = Ket /@ majorComponents[[notnullAmplitudes]];
3278 (*Multiply and add to build the final Ket*)
3279 majorRep = majorAmplitudes . majorComponents;
3280 {energy, majorRep}
3281 ), {idx, numStates}];
3282 Return[parsedByProb]
3283 )
3284 ];
3285
3286 (* ##### Eigensystem analysis ##### *)
3287 (* ##### *)
3288
3289 (* ##### *)
3290 (* ##### Misc ##### *)
3291
3292 SymbToNum::usage = "SymbToNum[expr, numAssociation] takes an expression expr and returns
    what results after making the replacements defined in the given replacementAssociation.

```

```

    If replacementAssociation doesn't define values for expected keys, they are taken to be
    zero.";
3293 SymbToNum[expr_, replacementAssociation_] := (
3294     includedKeys = Keys[replacementAssociation];
3295     (*If a key is not defined, make its value zero.*)
3296     fullAssociation = Table[(
3297         If[MemberQ[includedKeys, key],
3298             ToExpression[key]->replacementAssociation[key],
3299             ToExpression[key]->0
3300         ]
3301     ),
3302     {key, paramSymbols}];
3303     Return[expr/.fullAssociation];
3304 )
3305
3306 SimpleConjugate::usage = "SimpleConjugate[expr] takes an expression and applies a
    simplified version of the conjugate in that all it does is that it replaces the imaginary
    unit I with -I. It assumes that every other symbol is real so that it remains the same
    under complex conjugation. Among other expressions it is valid for any rational or
    polynomial expression with complex coefficients and real variables.";
3307 SimpleConjugate[expr_] := expr /. Complex[a_, b_] :> a - I b;
3308
3309 ExportMZip::usage =
3310     "ExportMZip[filename, object] exports the given object and compresses it. It first
    exports in .m format, it then compresses that file in to a zip file, and finally the .m
    file is deleted. The filename must be a full path and end with .m. This probably won't
    work on a PC.";
3311 ExportMZip[filename_, object_] := (
3312     zipTemplate = StringTemplate["cd \"'sourceDir'\"; zip \"'dest'\\" \"'source'\"];
3313     delTemplate = StringTemplate["rm \"'rmFname'\"];
3314     Export[filename, object];
3315     zipFilename = StringReplace[filename, ".m" -> ".zip"];
3316     splitName = FileNameSplit[zipFilename];
3317     zipFilename = splitName[[-1]];
3318     sourceDir = FileNameJoin[splitName[[1 ;; -2]]];
3319     zipCmd =
3320         zipTemplate[<|"sourceDir" -> sourceDir, "dest" -> zipFilename,
3321             "source" -> FileNameSplit[filename][[-1]]|>];
3322     delCmd = delTemplate[<|"rmFname" -> filename|>];
3323     Run[zipCmd];
3324     Run[delCmd];
3325 );
3326
3327 ImportMZip::usage =
3328     "ImportMZip[filename] decompresses the provided filename, and imports the enclosed .m
    file that it is assumed to contain. After being imported the uncompressed file is deleted
    from disk. The provided filename must be a full path and end with .zip. This probably
    won't work on a PC.";
3329 ImportMZip[filename_] := (
3330     splitName = FileNameSplit[filename];
3331     sourceDir = FileNameJoin[splitName[[1 ;; -2]]];
3332     delTemplate = StringTemplate["rm \"'rmFname'\"];
3333     unzipTemplate = StringTemplate["cd \"'sourceDir'\"; unzip \"'source'\"];
3334     unzipCmd = unzipTemplate[<|"sourceDir" -> sourceDir,
3335         "source" -> FileNameSplit[filename][[-1]]|>];
3336     Run[unzipCmd];
3337     mFilename = StringReplace[filename, ".zip" -> ".m"];
3338     imported = Import[mFilename];
3339     delCmd = delTemplate[<|"rmFname" -> mFilename|>];
3340     Run[delCmd];
3341     Return[imported];
3342 );

```

```

3343
3344 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.";
3345 ReplaceInSparseArray[s_SparseArray, rule_] := (With[{
3346     elem = s["NonzeroValues"]/.rule,
3347     def  = s["Background"]/.rule
3348 },
3349 (* Return[{elem,def}]; *)
3350 srep = SparseArray[Automatic,
3351     s["Dimensions"],
3352     def,
3353     {1, {s["RowPointers"], s["ColumnIndices"]}}, elem}
3354 ];
3355 ];
3356 Return[srep];
3357 );
3358
3359 Options[ParseTeXLikeSymbol] = {"Form" -> "List"};
3360 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.";
3361 ParseTeXLikeSymbol[bigString_, OptionsPattern[]] := (
3362     form = OptionValue["Form"];
3363     (*parse greek*)
3364     symbols = Table[(
3365         str = StringReplace[string, {"\\alpha" -> "\alpha",
3366             "\\beta" -> "\beta",
3367             "\\gamma" -> "\gamma",
3368             "\\psi" -> "\[Psi]"}];
3369         symbol = Which[
3370             StringContainsQ[str, "_"] && Not[StringContainsQ[str, "^"]],
3371             (
3372                 (*yes sub no sup*)
3373                 mainSymbol = StringSplit[str, "_"][[1]];
3374                 mainSymbol = ToExpression[mainSymbol];
3375
3376                 subPart =
3377                     StringCases[str,
3378                         RegularExpression@"\\{(.*)\\}" -> "$1"][[1]];
3379                 Subscript[mainSymbol, subPart]
3380             ),
3381             Not[StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
3382             (
3383                 (*no sub yes sup*)
3384                 mainSymbol = StringSplit[str, "^"][[1]];
3385                 mainSymbol = ToExpression[mainSymbol];
3386
3387                 supPart =
3388                     StringCases[str,
3389                         RegularExpression@"\\{(.*)\\}" -> "$1"][[1]];
3390                 Superscript[mainSymbol, supPart]),
3391             StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
3392             (
3393                 (*yes sub yes sup*)
3394                 mainSymbol = StringSplit[str, "_"][[1]];
3395                 mainSymbol = ToExpression[mainSymbol];
3396                 {subPart, supPart} =
3397                     StringCases[str, RegularExpression@"\\{(.*)\\}" -> "$1"];
3398                 Subsuperscript[mainSymbol, subPart, supPart]

```

```

3399         ),
3400         True,
3401         ((*no sup or sub*)
3402         str)
3403     ];
3404     symbol
3405     ),
3406     {string, StringSplit[bigString, " "]]];
3407 Which[
3408     form == "Row",
3409     Return[Row[symbols]],
3410     form == "List",
3411     Return[symbols]
3412 ]
3413 );
3414
3415 (* ##### Misc ##### *)
3416 (* ##### *)
3417
3418 (* ##### *)
3419 (* ##### Some Plotting Routines ##### *)
3420
3421 EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes states and produces a
3422 visualization of its energy spectrum.
3423 The resultant visualization can be navigated by clicking and dragging to zoom in on a
3424 region, or by clicking and dragging horizontally while pressing Ctrl. Double-click to
3425 reset the view.";
3426 Options[EnergyLevelDiagram] = {
3427     "Title" -> "",
3428     "ImageSize" -> 1000,
3429     "AspectRatio" -> 1/8,
3430     "Background" -> "Automatic",
3431     "Epilog" -> {}
3432 };
3433 EnergyLevelDiagram[states_, OptionsPattern[]]:= (
3434     energies = First/@states;
3435     epi = OptionValue["Epilog"];
3436     ExploreGraphics@ListPlot[Tooltip[{{#, 0}, {#, 1}}, {Quantity[#/8065.54429, "eV"],
3437 Quantity[#, 1/"Centimeters"]}]] &/@ energies,
3438     Joined -> True,
3439     PlotStyle -> Black,
3440     AspectRatio -> OptionValue["AspectRatio"],
3441     ImageSize -> OptionValue["ImageSize"],
3442     Frame -> True,
3443     PlotRange -> {All, {0, 1}},
3444     FrameTicks -> {{None, None}, {Automatic, Automatic}},
3445     FrameStyle -> Directive[15, Dashed, Thin],
3446     PlotLabel -> Style[OptionValue["Title"], 15, Bold],
3447     Background -> OptionValue["Background"],
3448     FrameLabel -> {"\!\(\*FractionBox[\(E\), SuperscriptBox[\(cm\), \(-1\)]]\)"},
3449     Epilog -> epi]
3450 )
3451
3452 ExploreGraphics::usage =
3453 "Pass a Graphics object to explore it. Zoom by clicking and dragging a rectangle. Pan by
3454 clicking and dragging while pressing Ctrl. Click twice to reset view.
3455 Based on ZeitPolizei @ https://mathematica.stackexchange.com/questions/7142/how-to-
3456 manipulate-2d-plots";
3457
3458 OptAxesRedraw::usage =
3459 "Option for ExploreGraphics to specify redrawing of axes. Default False.";
3460 Options[ExploreGraphics] = {OptAxesRedraw -> False};

```

```

3455
3456 ExploreGraphics[graph_Graphics, opts : OptionsPattern[]] := With[
3457   {gr = First[graph],
3458    opt = DeleteCases[Options[graph],
3459     PlotRange -> PlotRange | AspectRatio | AxesOrigin -> _],
3460    plr = PlotRange /. AbsoluteOptions[graph, PlotRange],
3461    ar = AspectRatio /. AbsoluteOptions[graph, AspectRatio],
3462    ao = AbsoluteOptions[AxesOrigin],
3463    rectangle = {Dashing[Small],
3464     Line[{#1,
3465      {First[#2], Last[#1]},
3466      #2,
3467      {First[#1], Last[#2]},
3468      #1]}} &,
3469    optAxesRedraw = OptionValue[OptAxesRedraw]],
3470   DynamicModule[
3471    {dragging=False, first, second, rx1, rx2, ry1, ry2,
3472     range = plr},
3473    {{rx1, rx2}, {ry1, ry2}} = plr;
3474   Panel@
3475   EventHandler[
3476    Dynamic@Graphics[
3477     If[dragging, {gr, rectangle[first, second]}, gr],
3478     PlotRange -> Dynamic@range,
3479     AspectRatio -> ar,
3480     AxesOrigin -> If[optAxesRedraw,
3481      Dynamic@Mean[range\[Transpose]], ao],
3482     Sequence @@ opt],
3483    {"MouseDown", 1} :> (
3484     first = MousePosition["Graphics"]
3485    ),
3486    {"MouseDragged", 1} :> (
3487     dragging = True;
3488     second = MousePosition["Graphics"]
3489    ),
3490    "MouseClicked" :> (
3491     If[CurrentValue@"MouseClicked"==2,
3492      range = plr];
3493    ),
3494    {"MouseUp", 1} :> If[dragging,
3495     dragging = False;
3496
3497     range = {{rx1, rx2}, {ry1, ry2}} =
3498      Transpose@{first, second};
3499     range[[2]] = {0, 1}],
3500    {"MouseDown", 2} :> (
3501     first = {sx1, sy1} = MousePosition["Graphics"]
3502    ),
3503    {"MouseDragged", 2} :> (
3504     second = {sx2, sy2} = MousePosition["Graphics"];
3505     rx1 = rx1 - (sx2 - sx1);
3506     rx2 = rx2 - (sx2 - sx1);
3507     ry1 = ry1 - (sy2 - sy1);
3508     ry2 = ry2 - (sy2 - sy1);
3509     range = {{rx1, rx2}, {ry1, ry2}};
3510     range[[2]] = {0, 1};
3511    )]]];
3512
3513 Options[LabeledGrid]={
3514   ItemSize->Automatic,
3515   Alignment->Center,
3516   Frame->All,

```

```

3517     "Separator" -> ", ",
3518     "Pivot" -> ""
3519 };
3520 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.";
3521 LabeledGrid[data_, rowHeaders_, columnHeaders_, OptionsPattern[]]:=Module[
3522     {gridList=data, rowHeads=rowHeaders, colHeads=columnHeaders},
3523     (
3524     separator=OptionValue["Separator"];
3525     pivot=OptionValue["Pivot"];
3526     gridList=Table[
3527         Tooltip[
3528             data[[rowIdx, colIdx]],
3529             DisplayForm[
3530                 RowBox[{rowHeads[[rowIdx]],
3531                     separator,
3532                     colHeads[[colIdx]]}
3533                 ]
3534             ],
3535             {rowIdx, Dimensions[data][[1]]},
3536             {colIdx, Dimensions[data][[2]]}];
3537     gridList=Transpose[Prepend[gridList, colHeads]];
3538     rowHeads=Prepend[rowHeads, pivot];
3539     gridList=Prepend[gridList, rowHeads]//Transpose;
3540     Grid[gridList,
3541         Frame->OptionValue[Frame],
3542         Alignment->OptionValue[Alignment],
3543         Frame->OptionValue[Frame],
3544         ItemSize->OptionValue[ItemSize]
3545     ]
3546 )
3547 ]
3548
3549 Options[HamiltonianForm]={ "Separator" -> ", ", "Pivot" -> ""}
3550 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.";
3552 HamiltonianForm[hamMatrix_, basisLabels_List, OptionsPattern[]]:= (
3553     braLabels=DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]] & /@
    basisLabels;
3554     ketLabels=DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"}]] & /@
    basisLabels;
3555     LabeledGrid[hamMatrix, braLabels, ketLabels, "Separator" -> OptionValue["Separator"], "Pivot"
    -> OptionValue["Pivot"]]
3556 )
3557
3558 Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover" -> True, "Overlay
    Values" -> True}];
3559 HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts : OptionsPattern[]] := (
3560     braLabels = DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]] & /@
    basisLabels;
3561     ketLabels = DisplayForm[Rotate[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"
    }], \[Pi]/2]] & /@ basisLabels;
3562     ketLabelsUpright = DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"
    }]] & /@ basisLabels;
3563     numRows = Length[hamMatrix];
3564     numCols = Length[hamMatrix[[1]]];

```



```

3565 epiThings = Which[
3566     And[OptionValue["Hover"], Not[OptionValue["Overlay Values"]]],
3567     Flatten[
3568         Table[
3569             Tooltip[
3570                 {
3571                     Transparent,
3572                     Rectangle[
3573                         {j - 1, numRows - i},
3574                         {j - 1, numRows - i} + {1, 1}
3575                     ]
3576                 },
3577                 Row[{braLabels[[i]], ketLabelsUpright[[j]], "=", hamMatrix[[i, j]]}]
3578             ],
3579             {i, 1, numRows},
3580             {j, 1, numCols}
3581         ]
3582     ],
3583     And[OptionValue["Hover"], OptionValue["Overlay Values"]],
3584     Flatten[
3585         Table[
3586             Tooltip[
3587                 {
3588                     Transparent,
3589                     Rectangle[
3590                         {j - 1, numRows - i},
3591                         {j - 1, numRows - i} + {1, 1}
3592                     ]
3593                 },
3594                 DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i]], "\[LeftBracketingBar]"
3595     , basisLabels[[j]], "\[RightAngleBracket]"}]]
3596             ],
3597             {i, numRows},
3598             {j, numCols}
3599         ]
3600     ],
3601     True,
3602     {}
3603 ];
3604 textOverlay = If[OptionValue["Overlay Values"],
3605 (
3606     Flatten[
3607         Table[
3608             Text[hamMatrix[[i, j]],
3609                 {j - 1/2, numRows - i + 1/2}
3610             ],
3611             {i, 1, numRows},
3612             {j, 1, numCols}
3613         ]
3614     ),
3615     {}
3616 ];
3617 epiThings = Join[epiThings, textOverlay];
3618 MatrixPlot[hamMatrix,
3619     FrameTicks -> {
3620         {Transpose[{Range[Length[braLabels]], braLabels}], None},
3621         {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}
3622     },
3623     Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
3624     Epilog -> epiThings
3625 ]

```

```

3626 );
3627
3628 (* ##### Some Plotting Routines ##### *)
3629 (* ##### *)
3630
3631 (* ##### *)
3632 (* ##### Load Functions ##### *)
3633
3634 LoadAll::usage="LoadAll[] executes all Load* functions.";
3635 LoadAll[]:=(
3636     LoadTermLabels[];
3637     LoadCFP[];
3638     LoadUk[];
3639     LoadVk1[];
3640     LoadT22[];
3641     LoadS00andECSOLS[];
3642
3643     LoadElectrostatic[];
3644     LoadSpinOrbit[];
3645     LoadS00andECSO[];
3646     LoadSpinSpin[];
3647     LoadThreeBody[];
3648     LoadChenDeltas[];
3649     LoadCarnall[];
3650 )
3651
3652 LoadTermLabels::usage="LoadTermLabels[] loads into the session the labels for the terms in
the f^n configurations.";
3653 LoadTermLabels[]:= (
3654     If[ValueQ[fnTermLabels], Return[]];
3655     PrintTemporary["Loading data for state labels in the f^n configurations..."];
3656     fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
3657     fnTermLabels::usage = "This list contains the labels of f^n configurations. Each element
of the list has four elements {LS, seniority, W, U}. At first sight this seems to only
include the labels for the f^6 and f^7 configuration, however, all is included in these
two.";
3658     If[!FileExistsQ[fnTermsFname],
3659         (PrintTemporary[">> fnTerms.m not found, generating ..."];
3660          fnTermLabels = ParseTermLabels["Export"->True];
3661         ),
3662         fnTermLabels = Import[fnTermsFname];
3663     ];
3664 )
3665
3666 LoadCarnall::usage="LoadCarnall[] loads data for trivalent lanthanides in LaF3 using the
data from Bill Carnall's 1989 paper.";
3667 LoadCarnall[]:=(
3668     If[ValueQ[Carnall], Return[]];
3669     carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
3670     If[!FileExistsQ[carnallFname],
3671         (PrintTemporary[">> Carnall.m not found, generating ..."];
3672          Carnall = ParseCarnall[];
3673         ),
3674         Carnall = Import[carnallFname];
3675     ];
3676     Carnall::usage = "Association of data from Carnall et al (1989) with the following keys:
{data, annotations, paramSymbols, elementNames, rawData, rawAnnotations, annnotatedData,
appendix:Pr:Association, appendix:Pr:Calculated, appendix:Pr:RawTable, appendix:Headings}
";
3677 )
3678
3679 LoadChenDeltas::usage="LoadChenDeltas[] loads the differences noted by Chen.";

```

```

3680 LoadChenDeltas[]:=(
3681   If[ValueQ[chenDeltas], Return[]];
3682   PrintTemporary["Loading the association of discrepancies found by Chen ..."];
3683   chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"}];
3684   If[!FileExistsQ[chenDeltasFname],
3685     (PrintTemporary[">> chenDeltas.m not found, generating ..."];
3686      chenDeltas = ParseChenDeltas[];
3687     ),
3688     chenDeltas = Import[chenDeltasFname];
3689   ];
3690 );
3691
3692 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";
3693 Options[ParseChenDeltas] = {"Export" -> True};
3694 ParseChenDeltas[OptionsPattern[]]:= (
3695   chenDeltasRaw = Import[FileNameJoin[{moduleDir, "data", "the-chen-deltas-A.csv"}]];
3696   chenDeltasRaw = chenDeltasRaw[[2 ;;]];
3697   chenDeltas = <||>;
3698   chenDeltasA = <||>;
3699   Off[Power::infy];
3700   Do[
3701     ({right, wrong} = {chenDeltasRaw[[row]][[4 ;;]],
3702      chenDeltasRaw[[row + 1]][[4 ;;]]};
3703     key = chenDeltasRaw[[row]][[1 ;; 3]];
3704     repRule = (#[[1]] -> #[[2]]*#[[1]]) & /@
3705       Transpose[{{M0, M2, M4, P2, P4, P6}, right/wrong}];
3706     chenDeltasA[key] = <|"right" -> right, "wrong" -> wrong,
3707       "repRule" -> repRule|>;
3708     chenDeltasA[{key[[1]], key[[3]], key[[2]]}] = <|"right" -> right,
3709       "wrong" -> wrong, "repRule" -> repRule|>;
3710   ],
3711   {row, 1, Length[chenDeltasRaw], 2}];
3712   chenDeltas["A"] = chenDeltasA;
3713
3714   chenDeltasRawB = Import[FileNameJoin[{moduleDir, "data", "the-chen-deltas-B.csv"}], "Text
    "];
3715   chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
3716   chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
3717   chenDeltasB = {ToExpression[StringTake[#[[1]], {2}]], #[[2]], #[[3]]} & /@ chenDeltasB;
3718   chenDeltas["B"] = chenDeltasB;
3719   On[Power::infy];
3720   If[OptionValue["Export"],
3721     (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"}];
3722      Export[chenDeltasFname, chenDeltas];
3723     )
3724   ];
3725   Return[chenDeltas];
3726 );
3727
3728 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";
3729 Options[ParseCarnall] = {"Export" -> True};
3730 ParseCarnall[] := (
3731   ions = {"Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho", "Er", "Tm"};
3732   templates = StringTemplate/@StringSplit["appendix:'ion':Association appendix:'ion':
    Calculated appendix:'ion':RawTable appendix:'ion':Headings", " "];
3733
3734   (* How many unique eigenvalues, after removing Kramer's degeneracy *)
3735   fullSizes = AssociationThread[ions, {91, 182, 1001, 1001, 3003, 1716, 3003, 1001,

```

```

1001, 182, 91}];
3736 carnall      = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[2]];
3737 carnallErr   = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[3]];
3738
3739 elementNames = carnall[[1]][[2;;]];
3740 carnall      = carnall[[2;;]];
3741 carnallErr   = carnallErr[[2;;]];
3742 carnall      = Transpose[carnall];
3743 carnallErr   = Transpose[carnallErr];
3744 paramNames   = ToExpression/@carnall[[1]][[1;;]];
3745 carnall      = carnall[[2;;]];
3746 carnallErr   = carnallErr[[2;;]];
3747 carnallData  = Table[(
3748     data      = carnall[[i]];
3749     data      = (#[[1]]->#[[2]])&/@Select[Transpose[{paramNames,data
}],#[[2]]!="&];
3750     elementNames[[i]]->data
3751   ),
3752   {i,1,13}
3753 ];
3754 carnallData  = Association[carnallData];
3755 carnallNotes = Table[(
3756     data      = carnallErr[[i]];
3757     elementName = elementNames[[i]];
3758     dataFun    = (
3759       #[[1]] -> If[#[[2]]=="[",
3760         "Not allowed to vary in fitting.",
3761         If[#[[2]]=="[R]",
3762           "Ratio constrained by: " <> <|"Eu"->"F4/F2=0.713; F6/F2=0.512",
3763           "Gd"->"F4/F2=0.710] ",
3764           "Tb"->"F4/F2=0.707" |>[elementName],
3765           If[#[[2]]=="i",
3766             "Interpolated",
3767             #[[2]]
3768         ]
3769       ]
3770     ) &;
3771     data = dataFun /@ Select[Transpose[{paramNames,data}],#[[2]]!="&];
3772     elementName->data
3773   ),
3774   {i,1,13}
3775 ];
3776 carnallNotes = Association[carnallNotes];
3777
3778 annotatedData = Table[
3779   If[NumberQ[#[[1]]],Tooltip[#[[1]],#[[2]]],""] & /@ Transpose[{paramNames
/.carnallData[element],
3780   paramNames/.carnallNotes[element]
3781   }],
3782   {element,elementNames}
3783 ];
3784 annotatedData = Transpose[annotatedData];
3785
3786 Carnall = <|"data"      -> carnallData,
3787   "annotations"      -> carnallNotes,
3788   "paramSymbols"     -> paramNames,
3789   "elementNames"     -> elementNames,
3790   "rawData"          -> carnall,
3791   "rawAnnotations"    -> carnallErr,
3792   "includedTableIons" -> ions,
3793   "annnotatedData"    -> annotatedData
3794 |>;

```

```

3795
3796 Do[(
3797     carnallData = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[i]];
3798     headers = carnallData[[1]];
3799     calcIndex = Position[headers,"Calc (1/cm)"][[1,1]];
3800     headers = headers[[2;;]];
3801     carnallLabels = carnallData[[1]];
3802     carnallData = carnallData[[2;;]];
3803     carnallTerms = DeleteDuplicates[First/@carnallData];
3804     parsedData = Table[(
3805         rows = Select[carnallData,#[[1]]==term&];
3806         rows = #[[2;;]]&/@rows;
3807         rows = Transpose[rows];
3808         rows = Transpose[{headers,rows}];
3809         rows = Association[(#[[1]]->#[[2]])&/@rows];
3810         term->rows
3811     ),
3812     {term,carnallTerms}
3813 ];
3814     carnallAssoc = Association[parsedData];
3815     carnallCalcEnergies = #[[calcIndex]]&/@carnallData;
3816     carnallCalcEnergies = If[NumberQ[#],#,Missing[]]&/@carnallCalcEnergies;
3817     ion = ions[[i-3]];
3818     carnallCalcEnergies = PadRight[carnallCalcEnergies, fullSizes[ion], Missing[]];
3819     keys = #["ion"->ion]&/@templates;
3820     Carnall[keys[[1]]] = carnallAssoc;
3821     Carnall[keys[[2]]] = carnallCalcEnergies;
3822     Carnall[keys[[3]]] = carnallData;
3823     Carnall[keys[[4]]] = headers;
3824 ),
3825 {i,4,14}
3826 ];
3827
3828 goodions = Select[ions,#!="Pm"&];
3829 expData = Select[Transpose[Carnall["appendix:"<>#<>":RawTable"]][[1+Position[Carnall["
appendix:"<>#<>":Headings"],"Exp (1/cm)"][[1,1]]],NumberQ]&/@goodions;
3830 Carnall["All Experimental Data"]=AssociationThread[goodions,expData];
3831 If[OptionValue["Export"],
3832 (
3833     carnallFname = FileNameJoin[{moduleDir,"data","Carnall.m"}];
3834     Print["Exporting to "<>exportFname];
3835     Export[carnallFname,Carnall];
3836 )
3837 ];
3838 Return[Carnall];
3839 )
3840
3841 LoadSymbolicHamiltonians::usage="LoadSymbolicHamiltonians[numEs] loads into the session the
symbolic Hamiltonians for the list numEs of given number of f-electrons. The default is
All, which loads all of them from 2 to 7. It loads into session the symbolicHamiltonians
symbol, which corresponds to an association that has keys equal to number of f-electrons
and values equal to corresponding symbolic Hamiltonian matrices provided as SparseArray."
;
3842 Options[LoadSymbolicHamiltonians] = {"Reload" -> False};
3843 LoadSymbolicHamiltonians[numEs_:All, OptionsPattern[]]:=
3844 (If[numEs === All,
3845     numEs = {2, 3, 4, 5, 6, 7}];
3846 If[Not[ValueQ[symbolicHamiltonians]],symbolicHamiltonians = <|>];
3847 Do[
3848 (
3849     If[And[
3850         MemberQ[Keys[symbolicHamiltonians], numE],

```

```

3851         Not[OptionValue["Reload"]]],
3852         Continue[]
3853     ];
3854     PrintTemporary["Loading symbolic Hamiltonian for f" <> ToString[numE] <> " ..." ];
3855     symbolicHamiltonians[numE] = Import["./hams/SymbolicMatrix-f" <> ToString[numE] <> ".m"
];
3856     ),
3857     {numE, numEs}
3858 ]
3859 )
3860
3861 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";
3862 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[.];
3863 LoadCFP::usage="LoadCFP[] loads CFP, CFPAssoc, and CFPTable into the session.";
3864 LoadCFP[]:= (
3865     If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]], Return[]];
3866
3867     PrintTemporary["Loading CFPTable ..."];
3868     CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
3869     If[!FileExistsQ[CFPTablefname],
3870         (PrintTemporary[">> CFPTable.m not found, generating ..."];
3871             CFPTable = GenerateCFPTable["Export"->True];
3872         ),
3873         CFPTable = Import[CFPTablefname];
3874 ];
3875
3876     PrintTemporary["Loading CFPs.m ..."];
3877     CFPfname = FileNameJoin[{moduleDir, "data", "CFPs.m"}];
3878     If[!FileExistsQ[CFPfname],
3879         (PrintTemporary[">> CFPs.m not found, generating ..."];
3880             CFP = GenerateCFP["Export"->True];
3881         ),
3882         CFP = Import[CFPfname];
3883 ];
3884
3885     PrintTemporary["Loading CFPAssoc.m ..."];
3886     CFPAfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
3887     If[!FileExistsQ[CFPAfname],
3888         (PrintTemporary[">> CFPAssoc.m not found, generating ..."];
3889             CFPAssoc = GenerateCFPAssoc["Export"->True];
3890         ),
3891         CFPAssoc = Import[CFPAfname];
3892 ];
3893 );
3894
3895 ReducedUkTable::usage = "ReducedUkTable[{n, l = 3, SL, SpLp, k}] provides reduced matrix
elements of the spherical tensor operator Uk. See Cowan (1981) section 11-9 \"Unit Tensor
Operators\". Loaded using LoadUk[.];
3896 LoadUk::usage="LoadUk[] loads into session the reduced matrix elements for unit tensor
operators.";
3897 LoadUk[]:= (
3898     If[ValueQ[ReducedUkTable], Return[]];
3899     PrintTemporary["Loading the association of reduced matrix elements for unit tensor
operators ..."];
3900     ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "ReducedUkTable.m"}];
3901     If[!FileExistsQ[ReducedUkTableFname],

```

```

3902     (PrintTemporary[">> ReducedUkTable.m not found, generating ..."];
3903     ReducedUkTable = GenerateReducedUkTable[7];
3904 ),
3905     ReducedUkTable = Import[ReducedUkTableFname];
3906 ];
3907 );
3908
3909 ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}] provides the calculated
    result of Electrostatic[n, SL, SpLp]. Load using LoadElectrostatic[.];
3910 LoadElectrostatic::usage="LoadElectrostatic[] loads the reduced matrix elements for the
    electrostatic interaction.";
3911 LoadElectrostatic[]:= (
3912     If[ValueQ[ElectrostaticTable], Return[]];
3913     PrintTemporary["Loading the association of matrix elements for the electrostatic
    interaction ..."];
3914     ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}];
3915     If[!FileExistsQ[ElectrostaticTablefname],
3916         (PrintTemporary[">> ElectrostaticTable.m not found, generating ..."];
3917         ElectrostaticTable = GenerateElectrostaticTable[7];
3918         ),
3919         ElectrostaticTable = Import[ElectrostaticTablefname];
3920     ];
3921 );
3922
3923 LoadVk1::usage="LoadVk1[] loads into session the matrix elements of Vk1.";
3924 LoadVk1[]:= (
3925     If[ValueQ[ReducedVikTable], Return[]];
3926     PrintTemporary["Loading the association of matrix elements for Vk1 ..."];
3927     ReducedVikTableFname = FileNameJoin[{moduleDir, "data", "ReducedVikTable.m"}];
3928     If[!FileExistsQ[ReducedVikTableFname],
3929         (PrintTemporary[">> ReducedVikTable.m not found, generating ..."];
3930         ReducedVikTable = GenerateReducedVikTable[7];
3931         ),
3932         ReducedVikTable = Import[ReducedVikTableFname];
3933     ]
3934 );
3935
3936 LoadSpinOrbit::usage="LoadSpinOrbit[] loads into session the matrix elements of the spin-
    orbit interaction.";
3937 LoadSpinOrbit[]:= (
3938     If[ValueQ[SpinOrbitTable], Return[]];
3939     PrintTemporary["Loading the association of matrix elements for spin-orbit ..."];
3940     SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
3941     If[!FileExistsQ[SpinOrbitTableFname],
3942         (PrintTemporary[">> SpinOrbitTable.m not found, generating ..."];
3943         SpinOrbitTable = GenerateSpinOrbitTable[7, True];
3944         ),
3945         SpinOrbitTable = Import[SpinOrbitTableFname];
3946     ]
3947 );
3948
3949 LoadS00andECSOLS::usage="LoadS00andECSOLS[] loads into session the LS reduced matrix
    elements of the S00-ECSO interaction.";
3950 LoadS00andECSOLS[]:= (
3951     If[ValueQ[S00andECSOLSTable], Return[]];
3952     PrintTemporary["Loading the association of LS reduced matrix elements for S00-ECSO ..."];
3953     S00andECSOLSTableFname = FileNameJoin[{moduleDir, "data", "ReducedS00andECSOLSTable.m"}];
3954     If[!FileExistsQ[S00andECSOLSTableFname],
3955         (PrintTemporary[">> ReducedS00andECSOLSTable.m not found, generating ..."];
3956         S00andECSOLSTable = GenerateS00andECSOLSTable[7];
3957         ),
3958         S00andECSOLSTable = Import[S00andECSOLSTableFname];

```

```

3959 ];
3960 );
3961
3962 LoadS00andECS0::usage="LoadS00andECS0[] loads into session the LSJ reduced matrix elements
of spin-other-orbit and electrostatically-correlated-spin-orbit.";
3963 LoadS00andECS0[]:=(
3964   If[ValueQ[S00andECS0TableFname], Return[]];
3965   PrintTemporary["Loading the association of matrix elements for spin-other-orbit and
electrostatically-correlated-spin-orbit ..."];
3966   S00andECS0TableFname = FileNameJoin[{moduleDir, "data", "S00andECS0Table.m"}];
3967   If[!FileExistsQ[S00andECS0TableFname],
3968     (PrintTemporary[">> S00andECS0Table.m not found, generating ..."];
3969     S00andECS0Table = GenerateS00andECS0Table[7, "Export"->True];
3970   ),
3971   S00andECS0Table = Import[S00andECS0TableFname];
3972 ];
3973 );
3974
3975 LoadT22::usage="LoadT22[] loads into session the matrix elements of T22.";
3976 LoadT22[]:=(
3977   If[ValueQ[T22Table], Return[]];
3978   PrintTemporary["Loading the association of reduced T22 matrix elements ..."];
3979   T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
3980   If[!FileExistsQ[T22TableFname],
3981     (PrintTemporary[">> ReducedT22Table.m not found, generating ..."];
3982     T22Table = GenerateT22Table[7];
3983   ),
3984   T22Table = Import[T22TableFname];
3985 ];
3986 );
3987
3988 LoadSpinSpin::usage="LoadSpinSpin[] loads into session the matrix elements of spin-spin.";
3989 LoadSpinSpin[]:=(
3990   If[ValueQ[SpinSpinTable], Return[]];
3991   PrintTemporary["Loading the association of matrix elements for spin-spin ..."];
3992   SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
3993   If[!FileExistsQ[SpinSpinTableFname],
3994     (PrintTemporary[">> SpinSpinTable.m not found, generating ..."];
3995     SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
3996   ),
3997   SpinSpinTable = Import[SpinSpinTableFname];
3998 ];
3999 );
4000
4001 LoadThreeBody::usage="LoadThreeBody[] loads into session the matrix elements of three-body
configuration-interaction effects.";
4002 LoadThreeBody[]:=(
4003   If[ValueQ[ThreeBodyTable], Return[]];
4004   PrintTemporary["Loading the association of matrix elements for three-body configuration-
interaction effects ..."];
4005   ThreeBodyFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
4006   ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
4007   If[!FileExistsQ[ThreeBodyFname],
4008     (PrintTemporary[">> ThreeBody.m not found, generating ..."];
4009     {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTablesUsingCFP[14, "Export" ->
True];
4010   ),
4011   ThreeBodyTable = Import[ThreeBodyFname];
4012   ThreeBodyTables = Import[ThreeBodiesFname];
4013 ];
4014 );
4015

```



```
4016 (* ##### Load Functions ##### *)
4017 (* ##### *)
4018
4019 End []
4020
4021 LoadTermLabels [];
4022 LoadCFP [];
4023
4024 EndPackage []
```