qlanth

January 29, 2024

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

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

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

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

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

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

REFERENCES:

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

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

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

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

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

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

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

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

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

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

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

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

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


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

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

paramAtlas = "
E0: linear combination of F_k, see eqn. (2-80) in Wybourne 1965
E1: linear combination of F_k
E2: linear combination of F_k
E3: linear combination of F_k,

ζ: spin-orbit strength parameter.

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

```

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

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

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

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

gs: electronic gyromagnetic ratio

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

B02: crystal field parameter B_0^2
B04: crystal field parameter B_0^4
B06: crystal field parameter B_0^6
B12: crystal field parameter B_1^2
B14: crystal field parameter B_1^4

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

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

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

S24: crystal field parameter S_2^4
S26: crystal field parameter S_2^6
S34: crystal field parameter S_3^4
S36: crystal field parameter S_3^6
```

```

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

\[Epsilon]: ground level baseline shift
t2Switch: controls the usage of the t2 operator beyond f7
wChErrA: If 1 then the type-A errors in Chen are used, if 0 then not.
wChErrB: If 1 then the type-B errors in Chen are used, if 0 then not.
";
paramSymbols = StringSplit[paramAtlas, "\n"];
paramSymbols = Select[paramSymbols, # != ""& ];
paramSymbols = ToExpression[StringSplit[#, ":"][[1]]] & /@ paramSymbols;
Protect /@ paramSymbols;

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

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

AllowedJ;
AllowedMforJ;
AllowedNKSLJMforJMTerms;
AllowedNKSLJMforJTerms;

AllowedNKSLJTerms;
AllowedNKSLTerms;
AllowedNKSLforJTerms;
AllowedSLJMTerms;
AllowedSLJTerms;

AllowedSLTerms;
BasisLSJMJ;
Bqk;
CFP;
CFPAssoc;

CFPTable;
CFPTerms;
Carnall;
CasimirG2;
CasimirSO7;

Cqk;
CrystalField;
Dk;
ElectrostaticConfigInteraction;
Electrostatic;

ElectrostaticTable;
EnergyLevelDiagram;
EnergyStates;
BasisTableGenerator;
EtoF;

FastIonSolverLaF3;
FindNKLSTerm;
FindSL;

```

```
FtoE;
GG2U;
GSO7W;
GenerateCFP;
GenerateCFPAssoc;

GenerateCFPTable;
GenerateCrystalFieldTable;
GenerateElectrostaticTable;
GenerateReducedUkTable;
GenerateReducedV1kTable;

GenerateSOOandECSOLSTable;
GenerateSOOandECSOTable;
GenerateSpinOrbitTable;
GenerateSpinSpinTable;
GenerateT22Table;

GenerateThreeBodyTables;
GenerateThreeBodyTablesUsingCFP;
Generator;
HamMatrixAssembly;
HamiltonianForm;

HamiltonianMatrixPlot;
HoleElectronConjugation;
IonSolverLaF3;
JJBlockMatrix;
JJBlockMatrixFileName;

JJBlockMatrixTable;
LabeledGrid;
LoadAll;
LoadCFP;
LoadCarnall;

LoadChenDeltas;
LoadElectrostatic;
LoadGuillotParameters;
LoadParameters;
LoadSOOandECSO;

LoadSOOandECSOLS;
LoadSpinOrbit;
LoadSpinSpin;
LoadSymbolicHamiltonians;
LoadT11;

LoadT22;
LoadTermLabels;
LoadThreeBody;
LoadUk;
LoadVk1;

MagneticInteractions;
MaxJ;
MinJ;
NKCFPPhase;

ParamPad;
ParseStates;
ParseStatesByNumBasisVecs;
```

```
ParseStatesByProbabilitySum;
ParseTermLabels;

Phaser;
PrettySaunders;
PrettySaundersSLJ;
PrettySaundersSLJmJ;
PrintL;

PrintSLJ;
PrintSLJM;
ReducedSOOandECSOinf2;
ReducedSOOandECSOinfn;
ReducedT11inf2;

ReducedT22inf2;
ReducedUk;
ReducedUkTable;
ReducedV1kTable;
Reducedt11inf2;

ReplaceInSparseArray;
RobustMissingQ;
SOOandECSO;
SOOandECSOTable;
Seniority;

ShiftedLevels;
SixJay;
SpinOrbit;
SpinSpin;
SpinSpinTable;

Sqk;
SquarePrimeToNormal;
T11n;
T22n;
TPO;

TabulateJJBlockMatrixTable;
TabulateManyJJBlockMatrixTables;
TextBasedProgressBar;
ScalarOperatorProduct;
ThreeBodyTable;

ThreeBodyTables;
ThreeJay;
TotalCFIters;
chenDeltas;
fK;

fnTermLabels;
moduleDir;
symbolicHamiltonians;

(* this selects the function that is applied
to calculated matrix elements *)
SimplifyFun = Expand;

Begin["`Private`"]

  moduleDir = DirectoryName[$InputFileName];
```

```
frontEndAvailable = (Head[$FrontEnd] === FrontEndObject);

(* ######################################################################## *)
(* ################################# MISC ################################# *)

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

TPO::usage="Two plus one.";
TPO[args__] := Times @@ ((2*# + 1) & /@ {args});

Phaser::usage = "Phaser[x] returns (-1)^x";
Phaser[exponent_] := ((-1)^exponent);

TriangleCondition[a_, b_, c_] := (Abs[b - c] <= a <= (b + c));

TriangleAndSumCondition[a_, b_, c_] := (And[Abs[b - c] <= a <= (b + c), IntegerQ[a + b + c
  ]]);

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

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.
  ";
SquarePrimeToNormal[squarePrime_] :=
(
  radical = Product[Prime[idx1 - 1] ^ Part[squarePrime, idx1], {idx1, 2, Length[squarePrime
  ]}];
  radical = radical /. {"A" -> 10, "B" -> 11, "C" -> 12, "D" -> 13};
  val = squarePrime[[1]] * Sqrt[radical];
  Return[val];
);

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

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

```

```
ReducedUk::usage = "ReducedUk[n, l, SL, SpLp, k] gives the reduced matrix element of the
  symmetric unit tensor operator U^(k). See equation 11.53 in Cowan (1981).";
ReducedUk[numE_, l_, SL_, SpLp_, k_] :=
  Module[{orbital, Uk, S, L, Sp, Lp, Sb, Lb, parentSL, cfpSL, cfpSpLp, Ukval, SLparents,
  SLpparents, commonParents, phase},
    {spin, orbital} = {1/2, 3};
    {S, L} = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    If[Not[S == Sp],
      Return[0]
    ];
    cfpSL = CFP[{numE, SL}];
    cfpSpLp = CFP[{numE, SpLp}];
    SLparents = First /@ Rest[cfpSL];
    SLpparents = First /@ Rest[cfpSpLp];
    commonParents = Intersection[SLparents, SLpparents];
    Uk = Sum[(
      {Sb, Lb} = FindSL[\[Psi]b];
      Phaser[Lb] *
      CFPAssoc[{numE, SL, \[Psi]b}] *
      CFPAssoc[{numE, SpLp, \[Psi]b}] *
      SixJay[{orbital, k, orbital}, {L, Lb, Lp}]
    ),
    {\[Psi]b, commonParents}
    ];
    phase = Phaser[orbital + L + k];
    prefactor = numE * phase * Sqrt[TPO[L,Lp]];
    Ukval = prefactor*Uk;
    Return[Ukval];
]

Ck::usage = "Diagonal reduced matrix element <l||C^(k)||l> where the Subscript[C, q]^(k)
  are reduced spherical harmonics. See equation 11.23 in Cowan (1981) with l=l'.";
Ck[orbital_, k_] := (-1)^orbital * TPO[orbital] * ThreeJay[{orbital, 0}, {k, 0}, {orbital,
  0}]

fk::usage = "Slater integral. See equation 12.17 in Cowan (1981).";
fk[numE_, orbital_, NKSL_, NKSLp_, k_] := Module[
  {terms, S, L, Sp, Lp, termsWithSameSpin, SL, fkVal, spinMultiplicity,
  prefactor, summand1, summand2},
  {S, L}   = FindSL[NKSL];
  {Sp, Lp} = FindSL[NKSLp];
  terms = AllowedNKSLTerms[numE];
  (* sum for summand1 is over terms with same spin *)
  spinMultiplicity  = 2*S + 1;
  termsWithSameSpin = StringCases[terms, ToString[spinMultiplicity] ~~ __];
  termsWithSameSpin = Flatten[termsWithSameSpin];
  If[Not[{S, L} == {Sp, Lp}],
  Return[0]
  ];
  prefactor = 1/2 * Abs[Ck[orbital, k]]^2;
  summand1 = Sum[(
      ReducedUkTable[{numE, orbital, SL, NKSL,  k}] *
      ReducedUkTable[{numE, orbital, SL, NKSLp, k}]
      ),
    {SL, termsWithSameSpin}
  ];
  summand1 = 1 / TPO[L] * summand1;
  summand2 = (
    KroneckerDelta[NKSL, NKSLp] *
      (numE *(4*orbital + 2 - numE)) /
      ((2*orbital + 1) * (4*orbital + 1))
    );
  fkVal = prefactor*(summand1 - summand2);
  Return[fkVal];
]

fK::usage = "Non-reduced Slater integral f^k = Subscript[f, k] * Subscript[D, k]";
fK[numE_, orbital_, NKSL_, NKSLp_ ,k_]:= (Dk[k] * fk[numE, orbital, NKSL, NKSLp, k])

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)."
  ;
Dk[k_] := {1, 225, 1089, 184041/25}[[k/2+1]]

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

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

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

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

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

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

GenerateReducedV1kTable::usage = "GenerateReducedV1kTable[nmax, export calculates values
 for Vk1 and returns an association where the keys are lists of the form {n, SL, SpLp, 1}.
  If the option \"Export\" is set to True then the resulting data is saved to ./data/
 ReducedV1kTable.m."
Options[GenerateReducedV1kTable] = {"Export" -> True, "Progress" -> True};
GenerateReducedV1kTable[numEmax_Integer:7, OptionsPattern[]]:= (
  numValues = Total[Length[AllowedNKSLTerms[#]]*Length[AllowedNKSLTerms[#]]&/@Range[1,
 numEmax]];
  Print["Calculating " <> ToString[numValues] <> " values for Vk1."];
  counter = 1;
  If[And[OptionValue["Progress"], frontEndAvailable],
  progBar = PrintTemporary[
      Dynamic[Row[{ProgressIndicator[counter, {0, numValues}], " ",
        counter}]]]
    ];
  ReducedV1kTable = Table[
    (
      counter = counter+1;
      {n, SL, SpLp, 1} -> SimplifyFun[ReducedV1k[n, SL, SpLp, 1]]
    ),
    {n, 1, numEmax},
    {SL, AllowedNKSLTerms[n]},
    {SpLp, AllowedNKSLTerms[n]}
  ];
  ReducedV1kTable = Association[ReducedV1kTable];
  If[And[OptionValue["Progress"], frontEndAvailable],
    NotebookDelete[progBar]
  ];
  exportFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m"}];
  If[OptionValue["Export"],
    (
      Print["Exporting to file "<>ToString[exportFname]];
      Export[exportFname, ReducedV1kTable];
    )
  ];
  Return[ReducedV1kTable];
)

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

(* ######################################################################### *)
(* ############################# Electrostatic ############################# *)

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).";
Electrostatic[numE_, NKSL_, NKSLp_]:= Module[
  {f0, f2, f4, f6, e0, e1, e2, e3, eMatrixVal, orbital},
  orbital = 3;
  Ek = {E0, E1, E2, E3};
  f0 = fK[numE, orbital, NKSL, NKSLp, 0];
  f2 = fK[numE, orbital, NKSL, NKSLp, 2];
  f4 = fK[numE, orbital, NKSL, NKSLp, 4];
  f6 = fK[numE, orbital, NKSL, NKSLp, 6];
  e0 = f0;
  e1 = 9/7*f0 + f2/42 + f4/77 + f6/462;
  e2 = 143/42*f2 - 130/77*f4 + 35/462*f6;
  e3 = 11/42*f2 + 4/77*f4 - 7/462*f6;
  eMatrixVal = e0*E0 + e1*E1 + e2*E2 + e3*E3;
  Return[eMatrixVal];
]

GenerateElectrostaticTable::usage = "GenerateElectrostaticTable[numEmax] can be used to
  generate the table for the electrostatic interaction from f^1 to f^numEmax. If the option
   \"Export\" is set to True then the resulting data is saved to ./data/ElectrostaticTable.
  m.";
Options[GenerateElectrostaticTable] = {"Export" -> True};
GenerateElectrostaticTable[numEmax_Integer:7, OptionsPattern[]]:= (
  ElectrostaticTable = Table[
    {numE, SL, SpLp} -> SimplifyFun[Electrostatic[numE, SL, SpLp]],
    {numE, 1, numEmax},
    {SL, AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]}
  ];
  ElectrostaticTable = Association[Flatten[ElectrostaticTable]];
  If[OptionValue["Export"],
    Export[FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}],
    ElectrostaticTable];
  ];
  Return[ElectrostaticTable];
)

(* ############################# Electrostatic ############################# *)
(* ########################################################################## *)

(* ########################################################################## *)
(* ################################# Bases ################################# *)

BasisTableGenerator::usage = "BasisTableGenerator[numE] returns an association whose keys
  are triples of the form {numE, J} and whose values are lists having the basis elements
  that correspond to {numE, J}.";
BasisTableGenerator[numE_] := Module[{energyStatesTable}, (
    energyStatesTable = <||>;
    allowedJ = AllowedJ[numE];
    Do[
    (
      energyStatesTable[{numE, J}] = EnergyStates[numE, J];
      ),
    {Jp, allowedJ},
    {J,  allowedJ}];
    Return[energyStatesTable]
    )
  ];

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}.";
BasisLSJMJ[numE_] := Module[{energyStatesTable, basis, idx1},
  (
    energyStatesTable = BasisTableGenerator[numE];
    basis = Table[
      energyStatesTable[{numE, AllowedJ[numE][[idx1]]}],
      {idx1, 1, Length[AllowedJ[numE]]}];
    basis = Flatten[basis, 1];
    Return[basis]
    )
  ];

(* ################################# Bases ################################# *)
(* ######################################################################### *)

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

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.";
Options[GenerateCFP] = {"Export" -> True, "PhaseFunction"-> "NK"};
GenerateCFP[OptionsPattern[]]:= (
  CFP = Table[
    {numE, NKSL} -> First[CFPTerms[numE, NKSL]],
    {numE, 1, 7},
    {NKSL, AllowedNKSLTerms[numE]}];
  CFP = Association[CFP];
  (* Go all the way to f14 *)
  CFP = CFPExpander["Export" -> False, "PhaseFunction"-> OptionValue["PhaseFunction"]];
  If[OptionValue["Export"],
    Export[FileNameJoin[{moduleDir, "data", "CFPs.m"}], CFP];
  ];
  Return[CFP];
)

JuddCFPPhase::usage="Phase between conjugate coefficients of fractional parentage according
   to Velkov's thesis, page 40.";
JuddCFPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_, parentSeniority_,
  daughterSeniority_] := (
    {spin, orbital} = {1/2, 3};
    expo = (
        (parentS + parentL + daughterS + daughterL) -
        (orbital + spin) +
        1/2 * (parentSeniority + daughterSeniority - 1)
    );
    phase = Phaser[-expo];
    Return[phase];
)

NKCFPPhase::usage="Phase between conjugate coefficients of fractional parentage according
  to Nielson and Koster page viii. Note that there is a typo on there the expression for
  zeta should be (-1)^((v-1)/2) instead of (-1)^(v - 1/2).";
NKCFPPhase[parent_, parentS_, parentL_, daughterS_, daughterL_, parentSeniority_,
  daughterSeniority_] := (
    {spin, orbital} = {1/2, 3};
    expo = (
        (parentS + parentL + daughterS + daughterL) -
        (orbital + spin)
    );
    phase = Phaser[-expo];
    If[parent == 2*orbital,
        phase = phase * Phaser[(daughterSeniority-1)/2]];
    Return[phase];
)

Options[CFPExpander] = {"Export" -> True, "PhaseFunction" -> "NK"};
CFPExpander::usage="Using the coefficients of fractional parentage up to f7 this function
  calculates them up to f14.

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.";
CFPExpander[OptionsPattern[]]:=(
    orbital    = 3;
    halfFilled = 2 * orbital + 1;
    fullShell  = 2 * halfFilled;
    parentMax  = 2 * orbital;

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

    complementaryCFPs[{14, "1S"}] = {"1S", {"2F",1}};
    extendedCFPs         = Join[CFP, complementaryCFPs];
    If[OptionValue["Export"];,
    (
        exportFname = FileNameJoin[{moduleDir, "data", "CFPs_extended.m"}];
        Print["Exporting to ", exportFname];
        Export[exportFname, extendedCFPs];
    )
    ];
    Return[extendedCFPs];
)

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";
Options[GenerateCFPTable] = {"Export" -> True};
GenerateCFPTable[OptionsPattern[]]:= (
  CFPtextData = Import[FileNameJoin[{moduleDir, "data", "B1F_ALL.TXT"}]];
  fConfigs = StringSplit[CFPtextData, "[ONE PARTICLE FRACTIONAL PARENTAGE COEFFICIENTS "];
  CFPTable = {};

  (* This table parses the text file with the one-body coefficients of fractional parentage
   *)
  CFPTable = Table[
  (
    fNx = StringReplace[Part[fConfigs, idx1], "-" -> " -"];
    daughterLabelSpots = StringPosition[fNx,
      Shortest[StartOfLine ~~ DigitCharacter ~~ LetterCharacter ~~ ___ ~~ "["],
      Overlaps -> False];
    daughterLabels = Map[StringDrop[#, -1] &, StringTake[fNx, daughterLabelSpots]];
    daughterLabelLines = StringPosition[fNx,
      Shortest[StartOfLine ~~ DigitCharacter ~~ LetterCharacter ~~ __ ~~
        EndOfLine], Overlaps -> False];
    startDaughters = Map[Last, daughterLabelLines + 2];
    stopDaughters = Delete[Append[Map[First, daughterLabelLines - 2], StringLength[fNx]],
 1];
    daughterLines = Join[Partition[startDaughters, 1], Partition[stopDaughters, 1], 2];
    testing = Map[StringSplit,
      StringSplit[StringTake[fNx, daughterLines], EndOfLine]];
    testing2 = Map[DeleteCases[#, {}] &, testing];
    ToIntegerOrString[list_] := Map[If[StringMatchQ[#, NumberString], ToExpression[#], #]
 &, list];
    CFPs = Table[(
      tt = Part[testing2, mm];
      pLabels = Map[Extract[#, 1] &, tt];
      pValues = Map[SquarePrimeToNormal, Map[ToIntegerOrString[Drop[#, 2]] &, tt]];
      Join[Partition[pLabels, 1], Partition[pValues, 1], 2]
    ),
    {mm,1, Length[testing2]}
    ];
    CFPconfig = Join[Partition[daughterLabels, 1], CFPs, 2];
    CFPconfig
  ),
  {idx1, 2, 7}
  ];
  CFPTable = Join[{{{"2F", {"1S", 1}}}}, CFPTable];
  If[OptionValue["Export"],
    (
    CFPTablefname = FileNameJoin[{moduleDir, "data", "CFPTable.m"}];
    Export[CFPTablefname, CFPTable];
    )
  ];
  Return[CFPTable];
)

GenerateCFPAssoc::usage = "GenerateCFPAssoc[export] converts the coefficients of fractional
   parentage into an association in which zero values are explicit. If \"Export\" is set to
   True, the association is exported to the file /data/CFPAssoc.m.";
Options[GenerateCFPAssoc] = {"Export" -> True};
GenerateCFPAssoc[OptionsPattern[]]:= (
  CFPAssoc = Association[];
  Do[
    (daughterTerms = AllowedNKSLTerms[numE];
    parentTerms    = AllowedNKSLTerms[numE - 1];
    Do[
      (
      cfps = CFP[{numE, daughter}];
      cfps = cfps[[2 ;;]];
      parents = First /@ cfps;
      Do[
        (
        key = {numE, daughter, parent};
        cfp = If[
          MemberQ[parents, parent],
          (
            idx = Position[parents, parent][[1, 1]];
            cfps[[idx]][[2]]
          ),
          0
          ];
        CFPAssoc[key] = cfp;
        ),
        {parent, parentTerms}
        ]
      ),
      {daughter, daughterTerms}
      ]
    ),
    {numE, 1, 14}
    ];
  If[OptionValue["Export"],
    (
    CFPAssocfname = FileNameJoin[{moduleDir, "data", "CFPAssoc.m"}];
    Export[CFPAssocfname, CFPAssoc];
    )
  ];
  Return[CFPAssoc];
)

CFPTerms::usage = "CFPTerms[numE] gives all the daughter and parent terms, together with
  the corresponding coefficients of fractional parentage, that correspond to the the f^n
  configuration.

CFPTerms[numE, SL] gives all the daughter and parent terms, together with the corresponding
   coefficients of fractional parentage, that are compatible with the given string SL in
  the f^n configuration.

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

In all cases the output is in the shape of a list with enclosed lists having the format {
  daughter_term, {parent_term_1, CFP_1}, {parent_term_2, CFP_2}, ...}.
Only the one-body coefficients for f-electrons are provided.
In all cases it must be that 1 <= n <= 7.
";
CFPTerms[numE_] := Part[CFPTable, numE]
CFPTerms[numE_, SL_] :=
  Module[
    {NKterms, CFPconfig},
    NKterms = {{}};
    CFPconfig = 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 Fracional Parentage ################## *)
(* ########################################################################## *)

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

SpinOrbit::usage = "SpinOrbit[numE, SL, SpLp, J] returns the LSJ reduced matrix element ζ <
  SL, J|L.S|SpLp, J>. These are given as a function of ζ. This function requires that the
  association ReducedV1kTable be defined.";
SpinOrbit[numE_, SL_, SpLp_, J_]:= Module[
  {S, L, Sp, Lp, orbital, sign, 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
  this package is in. It requires ReducedV1kTable to be defined.";
GenerateSpinOrbitTable[nmax_:7, export_:False]:= (
  SpinOrbitTable =
    Table[
      {numE, SL, SpLp, J} -> SpinOrbit[numE, SL, SpLp, J],
    {numE, 1, nmax},
    {J, MinJ[numE], MaxJ[numE]},
    {SL, Map[First, AllowedNKSLforJTerms[numE, J]]},
    {SpLp, Map[First, AllowedNKSLforJTerms[numE, J]]}
    ];
  SpinOrbitTable = Association[SpinOrbitTable];

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

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

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

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

  (* Parse table 1 from Judd 1984 *)
  judd1984Fname1 = FileNameJoin[{moduleDir, "data", "Judd1984-1.csv"}];
  data = Import[judd1984Fname1, "CSV", "Numeric" -> False];
  headers = data[[1]];
  data = data[[2 ;;]];
  data = Transpose[data];
  \[Psi] = Select[data[[1]], # != "" &];
  \[Psi]p = Select[data[[2]], # != "" &];
  matrixKeys = Transpose[{\[Psi], \[Psi]p}];
  data = data[[3 ;;]];
  cols = Table[ParseJuddTab1 /@ Select[col, # != "" &], {col, data}];
  cols = Select[cols, Length[#] == 21 &];
  tab1 = Prepend[Prepend[cols, \[Psi]p], \[Psi]];
  tab1 = Transpose[Prepend[Transpose[tab1], headers]];

  (* Parse table 2 from Judd 1984 *)
  judd1984Fname2 = FileNameJoin[{moduleDir, "data", "Judd1984-2.csv"}];
  data = Import[judd1984Fname2, "CSV", "Numeric" -> False];
  headers = data[[1]];
```

```
    data = data[[2 ;;]];
    data = Transpose[data];
    {operatorLabels, WUlabels, multiFactorSymbols, multiFactorValues} = data[[;; 4]];
    multiFactorValues = ParseJuddTab1 /@ multiFactorValues;
    multiFactorValues = AssociationThread[multiFactorSymbols -> multiFactorValues];

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

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

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

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

  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.";
  Options[GenerateThreeBodyTables] = {"Export" -> False};
  GenerateThreeBodyTables[nmax_ : 7, OptionsPattern[]] := (
    tiKeys = {"t_{2}"     ,"t_{2}^{'}",
              "t_{3}",
              "t_{4}"     , "t_{6}",
              "t_{7}"     , "t_{8}",
              "t_{11}"    , "t_{11}^{'}",
              "t_{12}",
              "t_{14}"    , "t_{15}",
              "t_{16}"    , "t_{17}",
              "t_{18}"    , "t_{19}"};
    TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
    juddOperators = ParseJudd1984[];

    op3MatrixElement::usage =
    "op3MatrixElement[SL, SpLp, opSymbol] returns the value for the reduced matrix element of the operator opSymbol for the terms {SL, SpLp} in the f^3 configuration.

  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.\"";
    op3MatrixElement[SL_, SpLp_, opSymbol_] := (
      jOP = juddOperators[{3, opSymbol}];
      key = {SL, SpLp};
      val = If[MemberQ[Keys[jOP], key],
              jOP[key],
              0];
      Return[val];
      );

    ti::usage = "This is the implementation of formula (2) in Judd & Suskin 1984.";
    ti[n_, SL_, SpLp_, tiKey_, opOrder_:3] := Module[
      {nn, S, L, Sp, Lp, cfpSL, cfpSpLp, parentSL, parentSpLp, tnk, tnks},
      {S, L} = FindSL[SL];
      {Sp, Lp} = FindSL[SpLp];
      If[S == Sp && L == Lp,
      (
        cfpSL   = CFP[{n, SL}];
        cfpSpLp = CFP[{n, SpLp}];
        tnks    = Table[(
          parentSL = cfpSL[[nn, 1]];
          (
            cfpSL[[nn, 2]] *
            cfpSpLp[[mm, 2]] *
            tktable[{n - 1, parentSL, cfpSpLp[[mm,1]], tiKey}]
          )
          ),
          {nn, 2, Length[cfpSL]},
          {mm, 2, Length[cfpSpLp]}
          ];
        tnk = Total[Flatten[tnks]];
      ),
        tnk = 0;
      ];
      Return[ n / (n - opOrder) * tnk];
      ];

    (*Calculate the matrix elements of t^i for n up to 7*)
    tktable = <||>;
    Do[
      (
        Do[
          (
            tkValue =
              Which[
                numE <= 2, (* Initialize n=1,2 with zeros *)
                0,
                numE == 3, (* Grab matrix elem in f^3 from Judd 1984 *)
                SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
                True,
                SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2, 3]]]
              ];
            tktable[{numE, SL, SpLp, opKey}] = tkValue;
          ),
        {SL,    AllowedNKSLTerms[numE]},
        {SpLp,  AllowedNKSLTerms[numE]},
        {opKey, Append[tiKeys, "e_{3}"]}
        ];
        PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " configuration complete"]];
      ),
    {numE, 1, nmax}
    ];
    (*Now use those matrix elements to determine their sum as weighted by their corresponding
     strengths Ti*)
    partialTiKeys = {"t_{2}^{'}",
          "t_{3}",
          "t_{4}"     , "t_{6}",
          "t_{7}"     , "t_{8}",
          "t_{11}"    , "t_{11}^{'}",
          "t_{12}",
          "t_{14}"    , "t_{15}",
          "t_{16}"    , "t_{17}",
          "t_{18}"    , "t_{19}"};
    ThreeBodyTable = <||>;
    Do[
      Do[(
          flipSign                 = -1;
          holeElectronSelector     = ((1 + isE)/2 + (1 - isE)/2 * flipSign);
          ThreeBodyTable[{numE, SL, SpLp}] =
            (
              holeElectronSelector *
              Sum[(tktable[{numE, SL, SpLp, tiKey}] * TSymbolsAssoc[tiKey]),
              {tiKey, partialTiKeys}
              ]
            )
        );,
      {SL, AllowedNKSLTerms[numE]},
      {SpLp, AllowedNKSLTerms[numE]}
      ];
      PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " matrix complete"]];,
    {numE, 1, nmax}
    ];
```

```

  (* Calculate the matrix elements of t2 in f^n*)
  LoadUk[];
  Do[(
    terms = AllowedNKSLTerms[numE];
    Do[
      (
        electro = Electrostatic[numE, term1, term2];
        electro = electro /. {E0 -> 0, E1 -> 0, E2 -> 0, E3 -> 1};
        prefactor = (
          (numE - 2)/(70 Sqrt[2]) (1 + isE)/2 +
          ((14 - numE) - 2)/(70 Sqrt[2]) (1 - isE)/2
        );
        prefactor = Simplify[prefactor];
        braSeniority = Seniority[SL];
        ketSeniority = Seniority[SpLp];
        onePlus\[CapitalDelta]vHalf = Simplify[1 + (braSeniority - ketSeniority)/2];
        flipSign      = If[EvenQ[onePlus\[CapitalDelta]vHalf], 1, -1];
        holeElectronSelector           = ((1 + isE)/2 + (1 - isE)/2 * flipSign);
        t2primeVal = holeElectronSelector*tktable[{numE, term1, term2, "t_{2}^{'}"}];
        t2value = If[numE==2,
         prefactor * electro,
         prefactor * electro + t2primeVal
        ];
        t2value    = T2 * Simplify[t2value];
        ThreeBodyTable[{numE, term1, term2}] += t2value;
      ),
      {term1, terms},
      {term2, terms}
      ];
  ),
  {numE, 2, nmax}
  ];

  ThreeBodyTables =
  Table[
    (
      terms = AllowedNKSLTerms[numE];
      singleThreeBodyTable =
        Table[
          {SL, SLp} -> ThreeBodyTable[{numE, SL, SLp}],
        {SL, terms},
        {SLp, terms}
        ];
      singleThreeBodyTable = Flatten[singleThreeBodyTable];
      singleThreeBodyTables =
        Table[(
          notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
          reps = ConstantArray[0, Length[TSymbols]];
          reps[[notNullPosition]] = 1;
          rep = AssociationThread[TSymbols -> reps];
          notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
          ),
        {notNullSymbol, TSymbols}
        ];
      singleThreeBodyTables = Association[singleThreeBodyTables];
      numE -> singleThreeBodyTables
    ),
  {numE, 1, nmax}
  ];
  ThreeBodyTables = Association[ThreeBodyTables];

```

```
    If[OptionValue["Export"],
    (
      threeBodyTablefname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
      Export[threeBodyTablefname, ThreeBodyTable];
      threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
      Export[threeBodyTablesfname, ThreeBodyTables];
    )
    ];
    Return[{ThreeBodyTable, ThreeBodyTables}];
    )

Options[GenerateThreeBodyTablesUsingCFP] = {"Export" -> False};
GenerateThreeBodyTablesUsingCFP::usage="This function generates the matrix elements for the
    three body operators using the coefficients of fractional parentage, including those
    beyond f^7.";
GenerateThreeBodyTablesUsingCFP[nmax_Integer : 14, OptionsPattern[]] := (
  tiKeys = {"t_{2}", "t_{2}^{'}", "t_{3}", "t_{4}", "t_{6}", "t_{7}",
    "t_{8}", "t_{11}", "t_{11}^{'}", "t_{12}", "t_{14}", "t_{15}",
    "t_{16}", "t_{17}", "t_{18}", "t_{19}"};
  TSymbolsAssoc = AssociationThread[tiKeys -> TSymbols];
  juddOperators = ParseJudd1984[];
  op3MatrixElement::usage = "op3MatrixElement[SL, SpLp, opSymbol] returns the value for the
    reduced matrix element of the operator opSymbol for the terms {SL, SpLp} in the f^3
    configuration.";
  op3MatrixElement[SL_, SpLp_, opSymbol_] := (
    jOP = juddOperators[{3, opSymbol}];
    key = {SL, SpLp};
    val = If[MemberQ[Keys[jOP], key],
      jOP[key],
      0];
    Return[val];
    );
  ti::usage = "This is the implementation of formula (2) in Judd & Suskin 1984. It computes
    the matrix elements of ti in f^n by using the matrix elements in f3 and the coefficients
    of fractional parentage. If the option \"Fast\" is set to True then the values for n>7
    are simply computed as the negatives of the values in the complementary configuration;
    this except for t2 and t11 which are treated as special cases.";
  Options[ti] = {"Fast" -> True};
  ti[nE_, SL_, SpLp_, tiKey_, opOrder_ : 3, OptionsPattern[]] :=
   Module[{nn, S, L, Sp, Lp,
      cfpSL, cfpSpLp,
      parentSL, parentSpLp, tnk, tnks},
    {S, L}   = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    fast     = OptionValue["Fast"];
    numH = 14 - nE;
    If[fast && Not[MemberQ[{"t_{2}","t_{11}"},tiKey]] && nE > 7,
      Return[-tktable[{numH, SL, SpLp, tiKey}]]
    ];
    If[(S == Sp && L == Lp),
     (
      cfpSL    = CFP[{nE, SL}];
      cfpSpLp = CFP[{nE, SpLp}];
      tnks = Table[(
          parentSL    = cfpSL[[nn, 1]];
          parentSpLp = cfpSpLp[[mm, 1]];
          cfpSL[[nn, 2]] * cfpSpLp[[mm, 2]] *
          tktable[{nE - 1, parentSL, parentSpLp, tiKey}]
          ),
          {nn, 2, Length[cfpSL]},
          {mm, 2, Length[cfpSpLp]}
          ];
      tnk = Total[Flatten[tnks]];
      ),
     tnk = 0;
     ];
    Return[ nE / (nE - opOrder) * tnk];];

  (*Calculate the matrix elements of t^i for n up to nmax*)
  tktable = <||>;
  Do[(
    Do[(
      tkValue = Which[numE <= 2,
        (*Initialize n=1,2 with zeros*)
        0,
        numE == 3,
        (*Grab matrix elem in f^3 from Judd 1984*)
        SimplifyFun[op3MatrixElement[SL, SpLp, opKey]],
        True,
        SimplifyFun[ti[numE, SL, SpLp, opKey, If[opKey == "e_{3}", 2, 3]]]
        ];
      tktable[{numE, SL, SpLp, opKey}] = tkValue;
      ),
    {SL, AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]},
    {opKey, Append[tiKeys, "e_{3}"]}
    ];
    PrintTemporary[StringJoin["\[ScriptF]", ToString[numE], " configuration complete"]];
    ),
  {numE, 1, nmax}
  ];

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

  ThreeBodyTables = Table[(
    terms = AllowedNKSLTerms[numE];
    singleThreeBodyTable =
      Table[
        {SL, SLp} -> ThreeBodyTable[{numE, SL, SLp}],
        {SL, terms},
        {SLp, terms}
      ];
    singleThreeBodyTable  = Flatten[singleThreeBodyTable];
    singleThreeBodyTables = Table[(
        notNullPosition = Position[TSymbols, notNullSymbol][[1, 1]];
        reps = ConstantArray[0, Length[TSymbols]];
        reps[[notNullPosition]] = 1;
        rep = AssociationThread[TSymbols -> reps];
        notNullSymbol -> Association[(singleThreeBodyTable /. rep)]
        ),
      {notNullSymbol, TSymbols}
      ];
    singleThreeBodyTables = Association[singleThreeBodyTables];
    numE -> singleThreeBodyTables),
    {numE, 1, 7}];

  ThreeBodyTables = Association[ThreeBodyTables];
  If[OptionValue["Export"], (
    threeBodyTablefname = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
    Export[threeBodyTablefname, ThreeBodyTable];
    threeBodyTablesfname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
    Export[threeBodyTablesfname, ThreeBodyTables];
    )
   ];
  Return[{ThreeBodyTable, ThreeBodyTables}];)

  ScalarOperatorProduct::usage="ScalarOperatorProduct[op1, op2, numE] calculated the
    innerproduct between the two scalar operators op1 and op2.";
  ScalarOperatorProduct[op1_, op2_, numE_] := Module[
    {terms, S, L, factor, term1, term2},
    (
    terms = AllowedNKSLTerms[numE];
    Simplify[
      Sum[(
        {S, L} = FindSL[term1];
        factor = TPO[S, L];
        factor * op1[{term1, term2}] * op2[{term2, term1}]
        ),
      {term1, terms},
      {term2, terms}
      ]
    ]
    )
  ];

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

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

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

  ReducedT11inf2::usage="ReducedT11inf2[SL, SpLp] returns the reduced matrix element of the
    scalar component of the double tensor T11 for the given SL terms SL, SpLp.
  Data used here for m0, m2, m4 is from Table II of Judd, BR, HM Crosswhite, and Hannah
    Crosswhite. Intra-Atomic Magnetic Interactions for f Electrons. Physical Review 169, no.
    1 (1968): 130.
  ";
  ReducedT11inf2[SL_, SpLp_] :=
    Module[{T11inf2},
    T11inf2 = <|
      {"1S", "3P"} -> 6 M0 + 2 M2 + 10/11 M4,
      {"3P", "3P"} -> -36 M0 - 72 M2 - 900/11 M4,
      {"3P", "1D"} -> -Sqrt[(2/15)] (27 M0 + 14 M2 + 115/11 M4),
      {"1D", "3F"} -> Sqrt[2/5] (23 M0 + 6 M2 - 195/11 M4),
      {"3F", "3F"} -> 2 Sqrt[14] (-15 M0 - M2 + 10/11 M4),
      {"3F", "1G"} -> Sqrt[11] (-6 M0 + 64/33 M2 - 1240/363 M4),
      {"1G", "3H"} -> Sqrt[2/5] (39 M0 - 728/33 M2 - 3175/363 M4),
      {"3H", "3H"} -> 8/Sqrt[55] (-132 M0 + 23 M2 + 130/11 M4),
      {"3H", "1I"} -> Sqrt[26] (-5 M0 - 30/11 M2 - 375/1573 M4)
      |>;
    Which[
      MemberQ[Keys[T11inf2],{SL,SpLp}],
        Return[T11inf2[{SL,SpLp}]],
      MemberQ[Keys[T11inf2],{SpLp,SL}],
        Return[T11inf2[{SpLp,SL}]],
      True,
        Return[0]
    ]
    ];

  MagneticInteractions[{numE_, SLJ_, SLJp_, J_}, chenDelta_] :=
    (
      key = {numE, SLJ, SLJp, J};
      ss = \[Sigma]SS * SpinSpinTable[key];
      sooandecso = SOOandECSOTable[key];
      total = ss + sooandecso;
      (* In the type A errors the wrong values are different *)
      If[MemberQ[Keys[chenDeltas["A"]], {numE, SLJ, SLJp}],
        (
          {S, L} = FindSL[SLJ];
          {Sp, Lp} = FindSL[SLJp];
          phase    = Phaser[Sp + L + J];
          Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
          Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
          {M0v, M2v, M4v, P2v, P4v, P6v} = chenDeltas["A"][{numE, SLJ, SLJp}]["wrong"];
          total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
          total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
          total  = total /. Prescaling;
          total  = wChErrA * total + (1 - wChErrA) * (ss + sooandecso)
        )
      ];
      (* In the type B errors the wrong values are zeros all around *)
      If[MemberQ[chenDeltas["B"], {numE, SLJ, SLJp}],
        (
          {S, L} = FindSL[SLJ];
          {Sp, Lp} = FindSL[SLJp];
          phase = Phaser[Sp + L + J];
          Msixjay = SixJay[{Sp, Lp, J},{L, S, 2}];
          Psixjay = SixJay[{Sp, Lp, J},{L, S, 1}];
          {M0v, M2v, M4v, P2v, P4v, P6v} = {0, 0, 0, 0, 0, 0};
          total  = phase * Msixjay(M0v*M0 + M2v*M2 + M4v*M4);
          total += phase * Psixjay(P2v*P2 + P4v*P4 + P6v*P6);
          total  = total /. Prescaling;
          total  = wChErrB * total + (1 - wChErrB) * (ss + sooandecso)
        )
      ];
      Return[total];
    )

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

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

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.

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.\"
";
T11n[numE_, SL_, SpLp_]:= Module[
  {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp, parentSL, parentSpLp, Sb, Lb
  , Tnkk, phase, Sbp, Lbp},
  {spin, orbital} = {1/2, 3};
  {S, L}    = FindSL[SL];
  {Sp, Lp} = FindSL[SpLp];
  t = 1;
  cfpSL     = CFP[{numE, SL}];
  cfpSpLp   = CFP[{numE, SpLp}];
  Tnkk =
    Sum[(
      parentSL = cfpSL[[idx2, 1]];
      parentSpLp = cfpSpLp[[idx1, 1]];
      {Sb, Lb} = FindSL[parentSL];
      {Sbp, Lbp} = FindSL[parentSpLp];
      phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
      (
        phase *
        cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
        SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
        SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
        T11Table[{numE - 1, parentSL, parentSpLp}]
      )
    ),
    {idx1, 2, Length[cfpSpLp]},
    {idx2, 2, Length[cfpSL]}
    ];
  Tnkk *= numE / (numE - 2) * Sqrt[TPO[S, Sp, L, Lp]];
  Return[Tnkk];
  ];

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

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

The T11 operator corresponds to the spin-other-orbit interaction, and the t11 operator (
  associated with electrostatically-correlated spin-orbit) originates from configuration
  interaction analysis. To their sum the a facor proportional to operator z13 is subtracted
   since its effect is seen as redundant to the spin-orbit interaction. The factor of 1/6
  is not on Judd's 1966 paper, but it is on \"Chen, Xueyuan, Guokui Liu, Jean Margerie, and
   Michael F Reid. \"A Few Mistakes in Widely Used Data Files for Fn Configurations
  Calculations.\" Journal of Luminescence 128, no. 3 (2008): 421-27\".

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

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

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

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

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

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

   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.
   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.\"
   ";
   T22n[numE_, SL_, SpLp_]:= Module[
    {spin, orbital, t, idx1, idx2, S, L, Sp, Lp, cfpSL, cfpSpLp, parentSL, parentSpLp, Sb, Lb
    , Tnkk, phase, Sbp, Lbp},
    {spin, orbital} = {1/2, 3};
    {S, L}   = FindSL[SL];
    {Sp, Lp} = FindSL[SpLp];
    t = 2;
    cfpSL    = CFP[{numE, SL}];
    cfpSpLp  = CFP[{numE, SpLp}];
    Tnkk =
      Sum[(
        parentSL = cfpSL[[idx2, 1]];
        parentSpLp = cfpSpLp[[idx1, 1]];
        {Sb, Lb} = FindSL[parentSL];
        {Sbp, Lbp} = FindSL[parentSpLp];
        phase = Phaser[Sb + Lb + spin + orbital + Sp + Lp];
        (
          phase *
          cfpSpLp[[idx1, 2]] * cfpSL[[idx2, 2]] *
          SixJay[{S, t, Sp}, {Sbp, spin, Sb}] *
          SixJay[{L, t, Lp}, {Lbp, orbital, Lb}] *
          T22Table[{numE - 1, parentSL, parentSpLp}]
        )
      ),
      {idx1, 2, Length[cfpSpLp]},
      {idx2, 2, Length[cfpSL]}
      ];
    Tnkk *= numE / (numE - 2) * Sqrt[TPO[S,Sp,L,Lp]];
    Return[Tnkk];
    ];

   Options[GenerateT22Table] = {"Export" -> True, "Progress" -> True};
   GenerateT22Table[nmax_Integer, OptionsPattern[]]:= (
    If[And[OptionValue["Progress"], frontEndAvailable],
      (
        numItersai = Association[Table[numE->Length[AllowedNKSLTerms[numE]]^2, {numE, 1, nmax
    }]];
        counters = Association[Table[numE->0, {numE, 1, nmax}]];
        totalIters = Total[Values[numItersai[[1;;nmax]]]];
        template1 = StringTemplate["Iteration ‘numiter‘ of ‘totaliter‘"];
        template2 = StringTemplate["‘remtime‘ min remaining"];template3 = StringTemplate["
    Iteration speed = ‘speed‘ ms/it"];
        template4 = StringTemplate["Time elapsed = ‘runtime‘ min"];
```

```
        progBar = PrintTemporary[
          Dynamic[
            Pane[
              Grid[{{Superscript["f", numE]},
                    {template1[<|"numiter"->numiter, "totaliter"->totalIters|>]},
                    {template4[<|"runtime"->Round[QuantityMagnitude[UnitConvert[(Now-
  startTime), "min"]], 0.1]|>]},
                    {template2[<|"remtime"->Round[QuantityMagnitude[UnitConvert[(Now-
  startTime)/(numiter)*(totalIters-numiter), "min"]], 0.1]|>]},
                    {template3[<|"speed"->Round[QuantityMagnitude[Now-startTime, "ms"]/(
  numiter), 0.01]|>]},
                    {ProgressIndicator[Dynamic[numiter], {1, totalIters}]}},
                    Frame->All],
                    Full,
                    Alignment->Center]
                  ]
                ];
      )
    ];
    T22Table = <||>;
    startTime = Now;
    numiter = 1;
    Do[
      (
        numiter+= 1;
        T22Table[{numE, SL, SpLp}] = Which[
          numE==1,
          0,
          numE==2,
          SimplifyFun[ReducedT22inf2[SL, SpLp]],
          True,
          SimplifyFun[T22n[numE,  SL, SpLp]]
        ];
      ),
    {numE, 1, nmax},
    {SL,   AllowedNKSLTerms[numE]},
    {SpLp, AllowedNKSLTerms[numE]}
    ];
    If[And[OptionValue["Progress"],frontEndAvailable],
      NotebookDelete[progBar]
    ];
    If[OptionValue["Export"],
      (
        fname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
        Export[fname, T22Table];
      )
    ];
    Return[T22Table];
  );

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

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

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

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

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

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

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

  Prescaling = {P2 -> P2/225, P4 -> P4/1089, P6 -> 25 * P6 / 184041};
  GenerateSOOandECSOTable::usage="GenerateSOOandECSOTable[nmax] generates the matrix elements
     in the |LSJ> basis for the (spin-other-orbit + electrostatically-correlated-spin-orbit)
    operator. It returns an association where the keys are of the form {n, SL, SpLp, J}. If
    the option \"Export\" is set to True then the resulting object is saved to the data
    folder. Since this is a scalar operator, there is no MJ dependence. This dependence only
    comes into play when the crystal field contribution is taken into account.";
  Options[GenerateSOOandECSOTable] = {"Export"->False}
  GenerateSOOandECSOTable[nmax_, OptionsPattern[]]:= (
    SOOandECSOTable = <||>;
    Do[
      SOOandECSOTable[{numE, SL, SpLp, J}] = (SOOandECSO[numE, SL, SpLp, J] /. Prescaling);,
    {numE, 1, nmax},
    {J, MinJ[numE], MaxJ[numE]},
    {SL,   First /@ AllowedNKSLforJTerms[numE, J]},
    {SpLp, First /@ AllowedNKSLforJTerms[numE, J]}
    ];
    If[OptionValue["Export"],
    (
      fname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
      Export[fname, SOOandECSOTable];
    )
    ];
    Return[SOOandECSOTable];
  );

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

  (* ######################################################################### *)
  (* ############################ Crystal Field ############################## *)

  Cqk::usage = "Cqk[numE, q, k, NKSL, J, M, NKSLp, Jp, Mp].";
  Cqk[numE_, q_, k_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := Module[
    {S, Sp, L, Lp, orbital, val},
    orbital = 3;
    {S, L}   = FindSL[NKSL];
    {Sp, Lp} = FindSL[NKSLp];
    f1  = ThreeJay[{J, -M}, {k, q}, {Jp, Mp}];
    val =
      If[f1==0,
        0,
        (
          f2 = SixJay[{L, J, S}, {Jp, Lp, k}] ;
          If[f2==0,
            0,
            (
              f3 = ReducedUkTable[{numE, orbital, NKSL, NKSLp, k}];
              If[f3==0,
                0,
                (
                  (
                    Phaser[J - M + S + Lp + J + k] *
                    Sqrt[TPO[J, Jp]] *
                    f1 *
                    f2 *
```

```
                     f3 *
                     Ck[orbital, k]
                   )
                 )
               ]
             )
           ]
         )
       ];
     val
     ]

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

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

  CrystalField::usage = "CrystalField[n, NKSL, J, M, NKSLp, Jp, Mp] gives the general
    expression for the matrix element of the crystal field Hamiltonian parametrized with Bqk
    and Sqk coefficients as a sum over spherical harmonics Cqk.";
  CrystalField[numE_, NKSL_, J_, M_, NKSLp_, Jp_, Mp_] := (
    Sum[
      (
        cqk  = Cqk[numE,  q, k, NKSL, J, M, NKSLp, Jp, Mp];
        cmqk = Cqk[numE, -q, k, NKSL, J, M, NKSLp, Jp, Mp];
        Bqk[q, k]   * (cqk + (-1)^q * cmqk) +
        I*Sqk[q, k] * (cqk - (-1)^q * cmqk)
        ),
    {k, {2, 4, 6}},
    {q, 0, k}
    ]
  )

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

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

  (* ############################ Crystal Field ############################### *)
  (* ######################################################################### *)

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

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

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

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

  GSO7W::usage = "GSO7W is an association whose keys are labels for the irreducible
    representations of group R7 and whose values are the eigenvalues of the corresponding
    Casimir operator.
  Reference: Wybourne, \"Spectroscopic Properties of Rare Earths\", table 2-7.";
  GSO7W := Association[
    {
      "000" -> 0,
      "100" -> 3/5,
      "110" -> 5/5,
      "111" -> 6/5,
      "200" -> 7/5,
      "210" -> 9/5,
      "211" -> 10/5,
      "220" -> 12/5,
      "221" -> 13/5,
      "222" -> 15/5
    }
  ];

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

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

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

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

  Options[JJBlockMatrix] = {"Sparse"->True, "ChenDeltas"->False};
  JJBlockMatrix::usage = "For given J, J' in the f^n configuration JJBlockMatrix[numE, J, J']
     determines all the SL S'L' terms that may contribute to them and using those it provides
     the matrix elements <J, LS | H | J', LS'>. H having contributions from the following
    interactions: Coulomb, spin-orbit, spin-other-orbit, electrostatically-correlated-spin-
    orbit, spin-spin, three-body interactions, and crystal-field.";
  JJBlockMatrix[numE_, J_, Jp_, CFTable_, OptionsPattern[]]:= Module[
    {NKSLJMs, NKSLJMps, NKSLJM, NKSLJMp,
    SLterm, SpLpterm,
    MJ, MJp,
    subKron, matValue, eMatrix},
    (
      NKSLJMs  = AllowedNKSLJMforJTerms[numE, J];
      NKSLJMps = AllowedNKSLJMforJTerms[numE, Jp];
      eMatrix =
        Table[
          (*Condition for a scalar matrix op*)
          SLterm   =  NKSLJM[[1]];
          SpLpterm = NKSLJMp[[1]];
          MJ       =  NKSLJM[[3]];
          MJp      = NKSLJMp[[3]];
          subKron  =
            (
              KroneckerDelta[J, Jp] *
              KroneckerDelta[MJ, MJp]
            );
          matValue =
            If[subKron==0,
              0,
                (
                  ElectrostaticTable[{numE, SLterm, SpLpterm}] +
                  ElectrostaticConfigInteraction[{SLterm, SpLpterm}] +
                  SpinOrbitTable[{numE, SLterm, SpLpterm, J}] +
                  MagneticInteractions[{numE, SLterm, SpLpterm, J}, OptionValue["ChenDeltas"
    ]] +
                  ThreeBodyTable[{numE, SLterm, SpLpterm}]
                )
            ];
          matValue += CFTable[{numE, SLterm, J, MJ, SpLpterm, Jp, MJp}];
          matValue,
        {NKSLJMp, NKSLJMps},
        {NKSLJM , NKSLJMs}
        ];
    If[OptionValue["Sparse"],
      eMatrix = SparseArray[eMatrix]
    ];
    Return[eMatrix]
  )
  ];

  EnergyStates::usage = "Alias for AllowedNKSLJMforJTerms. At some point may be used to
    redefine states used in basis.";
  EnergyStates[numE_, J_]:= AllowedNKSLJMforJTerms[numE, J];

  JJBlockMatrixFileName::usage = "JJBlockMatrixFileName[numE] gives the filename for the
    energy matrix table for an atom with numE f-electrons. The function admits an optional
    parameter \"FilenameAppendix\" which can be used to modify the filename.";
  Options[JJBlockMatrixFileName] = {"FilenameAppendix" -> ""}
  JJBlockMatrixFileName[numE_Integer, OptionsPattern[]] := (
    fileApp = OptionValue["FilenameAppendix"];
    fname = FileNameJoin[{moduleDir,
        "hams",
        StringJoin[{"f", ToString[numE], "_JJBlockMatrixTable", fileApp ,".m"}]}];
    Return[fname];
    );

  Options[TabulateJJBlockMatrixTable] = {"Sparse"->True, "ChenDeltas"->False};
  TabulateJJBlockMatrixTable::usage = "TabulateJJBlockMatrixTable[numE, I] returns a list
    with three elements {JJBlockMatrixTable, EnergyStatesTable, AllowedM}. JJBlockMatrixTable
     is an association with keys equal to lists of the form {numE, J, Jp}. EnergyStatesTable
    is an association with keys equal to lists of the form {numE, J}. AllowedM is another
    association with keys equal to lists of the form {numE, J} and values equal to lists
    equal to the corresponding values of MJ. It's unnecessary (and it won't work in this
    implementation) to give numE > 7 given the equivalency between electron and hole
    configurations.";
  TabulateJJBlockMatrixTable[numE_, CFTable_, OptionsPattern[]]:= (
    JJBlockMatrixTable = <||>;
    EnergyStatesTable = <||>;
    AllowedM = <||>;
    totalIterations = Length[AllowedJ[numE]]^2;
    template1 = StringTemplate["Iteration `numiter` of `totaliter`"];
    template2 = StringTemplate["`remtime` min remaining"];
    template4 = StringTemplate["Time elapsed = `runtime` min"];
```

```
    numiter   = 0;
    startTime = Now;
    If[$FrontEnd =!= Null,
      (
        temp = PrintTemporary[
          Dynamic[
            Grid[
              {
                {template1[<|"numiter"->numiter, "totaliter"->totalIterations|>]},
                {template2[<|"remtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime)/(
  Max[1,numiter])*(totalIterations-numiter), "min"]], 0.1]|>]},
                {template4[<|"runtime"->Round[QuantityMagnitude[UnitConvert[(Now-startTime),
  "min"]], 0.1]|>]},
                {ProgressIndicator[numiter, {1, totalIterations}]}
              }
            ]
        ]
        ];
      )
    ];
    Do[
      (
        JJBlockMatrixTable[{numE, J, Jp}] = JJBlockMatrix[numE, J, Jp, CFTable, "Sparse"->
  OptionValue["Sparse"], "ChenDeltas" -> OptionValue["ChenDeltas"]];
        EnergyStatesTable[{numE, J}]  = EnergyStates[numE, J];
        AllowedM[{numE, J}]              = Table[M, {J, MinJ[numE], MaxJ[numE]}, {M, -J, J
  }];
        numiter += 1;
      ),
    {Jp, AllowedJ[numE]},
    {J, AllowedJ[numE]}
    ];
    If[$FrontEnd =!= Null,
      NotebookDelete[temp]
    ];
    Return[{JJBlockMatrixTable, EnergyStatesTable, AllowedM}];
  )

  Options[TabulateManyJJBlockMatrixTables] = {"Overwrite"->False, "Sparse"->True, "ChenDeltas
  "->False, "FilenameAppendix"-> ""};
  TabulateManyJJBlockMatrixTables::usage = "TabulateManyJJBlockMatrixTables[{n1, n2, ...}]
  calculates the tables of matrix elements for the requested f^n_i configurations. The
  function does not return the matrices themselves. It instead returns an association whose
   keys are numE and whose values are the filenames where the output of
  TabulateJJBlockMatrixTables was saved to. When these files are loaded with Get, the
  following three symbols are thus defined: JJBlockMatrixTable, EnergyStatesTable, and
  AllowedM.
  JJBlockMatrixTable is an association whose keys are of the form {n, J, Jp} and whose values
   are matrix elements.";
  TabulateManyJJBlockMatrixTables[ns_, OptionsPattern[]]:= (
    overwrite = OptionValue["Overwrite"];
    fNames = <||>;
    fileApp = OptionValue["FilenameAppendix"];
    Do[
      (
        CFdataFilename = FileNameJoin[{moduleDir, "data", "CrystalFieldTable_f"<>ToString[
  numE]<>".zip"}];
        PrintTemporary["Importing CrystalFieldTable from ", CFdataFilename, " ..."];
        CrystalFieldTable = ImportMZip[CFdataFilename];

        PrintTemporary["#------- numE = ", numE, " -------#"];
        exportFname = JJBlockMatrixFileName[numE, "FilenameAppendix" -> fileApp];
        fNames[numE] = exportFname;
        If[FileExistsQ[exportFname] && Not[overwrite],
          Continue[]
        ];
        {JJBlockMatrixTable, EnergyStatesTable, AllowedM} = TabulateJJBlockMatrixTable[numE,
   CrystalFieldTable, "Sparse"->OptionValue["Sparse"], "ChenDeltas" -> OptionValue["
   ChenDeltas"]];
        If[FileExistsQ[exportFname]&&overwrite,
          DeleteFile[exportFname]
        ];
        Save[exportFname, {JJBlockMatrixTable, EnergyStatesTable, AllowedM}];

        ClearAll[CrystalFieldTable];
      ),
    {numE, ns}
    ];
  Return[fNames];
  )

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

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

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

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

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

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

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

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

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

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

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

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

  MinJ::usage = "MinJ[numE] gives the minimum J = S+L that corresponds to the configuration f
```

```
  ^numE.";
MinJ[numE_] := Min[Map[Abs[Part[#, 1] - Part[#, 2]] &, AllowedSLTerms[Min[numE, 14-numE]]]]

AllowedSLJTerms::usage = "AllowedSLJTerms[numE] returns a list with the allowed {S, L, J}
  terms in the f^n configuration, the terms are given as lists in the format {S, L, J}.
  This list may have repeated elements which account for possible degeneracies of the
  related term.";
AllowedSLJTerms[numE_] :=
  Module[{idx1, allowedSL, allowedSLJ},
    allowedSL = AllowedSLTerms[numE];
    allowedSLJ = {};
    For[
      idx1 = 1,
      idx1 <= Length[allowedSL],
      termSL = allowedSL[[idx1]];
      termsSLJ =
        Table[
          {termSL[[1]], termSL[[2]], J},
        {J,Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
        ];
      allowedSLJ = Join[allowedSLJ, termsSLJ];
      idx1++
    ];
    SortBy[allowedSLJ, Last]
  ]

AllowedNKSLJTerms::usage = "AllowedNKSLJTerms[numE] returns a list with the allowed {SL, J}
   terms in the f^n configuration, the terms are given as lists in the format {SL, J} where
   SL is a string in spectroscopic notation.";
AllowedNKSLJTerms[numE_] :=
  Module[{allowedSL, allowedNKSL, allowedSLJ, nn},
    allowedNKSL = AllowedNKSLTerms[numE];
    allowedSL = AllowedSLTerms[numE];
    allowedSLJ = {};
    For[
      nn = 1,
      nn <= Length[allowedSL],
      (
        termSL = allowedSL[[nn]];
        termNKSL = allowedNKSL[[nn]];
        termsSLJ =
          Table[{termNKSL, J},
          {J, Abs[termSL[[1]] - termSL[[2]]], Total[termSL]}
          ];
        allowedSLJ = Join[allowedSLJ, termsSLJ];
        nn++
      )
    ];
    SortBy[allowedSLJ, Last]
  ]

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.";
AllowedNKSLforJTerms[numE_, J_] := Module[
    {allowedSL, allowedNKSL, allowedSLJ, nn, termSL, termNKSL, termsSLJ},
    allowedNKSL = AllowedNKSLTerms[numE];
    allowedSL = AllowedSLTerms[numE];
    allowedSLJ = {};
    For[
      nn = 1,
      nn <= Length[allowedSL],
      (
        termSL = allowedSL[[nn]];
        termNKSL = allowedNKSL[[nn]];
        termsSLJ = If[Abs[termSL[[1]] - termSL[[2]]] <= J <= Total[termSL],
          {{termNKSL, J}},
          {}
          ];
        allowedSLJ = Join[allowedSLJ, termsSLJ];
        nn++
      )
    ];
    Return[allowedSLJ]
  ];

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}.";
AllowedSLJMTerms[numE_] := Module[
    {allowedSLJ, allowedSLJM, termSLJ, termsSLJM, nn},
    allowedSLJ = AllowedSLJTerms[numE];
    allowedSLJM = {};
    For[
      nn = 1,
      nn <= Length[allowedSLJ],
      nn++,
      (
        termSLJ = allowedSLJ[[nn]];
        termsSLJM =
          Table[{termSLJ[[1]], termSLJ[[2]], termSLJ[[3]], M},
          {M, - termSLJ[[3]], termSLJ[[3]]}
          ];
        allowedSLJM = Join[allowedSLJM, termsSLJM];
      )
    ];
    Return[SortBy[allowedSLJM, Last]];
    ]

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}.";
AllowedNKSLJMforJMTerms[numE_, J_, MJ_] :=
  Module[{allowedSL, allowedNKSL, allowedSLJM, nn},
    allowedNKSL = AllowedNKSLTerms[numE];
    allowedSL   = AllowedSLTerms[numE];
    allowedSLJM = {};
    For[
      nn = 1,
      nn <= Length[allowedSL],
      termSL = allowedSL[[nn]];
      termNKSL = allowedNKSL[[nn]];
      termsSLJ = If[(Abs[termSL[[1]] - termSL[[2]]]
                    <= J
                    <= Total[termSL]
                    && (Abs[MJ] <= J)
                    ),
                    {{termNKSL, J, MJ}},
                    {}];
      allowedSLJM = Join[allowedSLJM, termsSLJ];
      nn++
    ];
```

```
      Return[allowedSLJM];
    ]

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

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

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

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

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

  Options[ParseTermLabels] = {"Export" -> True};
  ParseTermLabels::usage="ParseTermLabels[] parses the labels for the terms in the f^n
    configurations based on the labels for the f6 and f7 configurations. The function returns
     a list whose elements are of the form {LS, seniority, W, U}.";
  ParseTermLabels[OptionsPattern[]] := Module[
    {labelsTextData, fNtextLabels, nielsonKosterLabels, seniorities, RacahW, RacahU},
  (
    labelsTextData = FileNameJoin[{moduleDir, "data", "NielsonKosterLabels_f6_f7.txt"}];
    fNtextLabels    = Import[labelsTextData];
    nielsonKosterLabels = Partition[StringSplit[fNtextLabels], 3];
    termLabels = Map[Part[#, {1}] &, nielsonKosterLabels];
    seniorities = Map[ToExpression[Part[# , {2}]] &, nielsonKosterLabels];
    racahW =
      Map[
        StringTake[
          Flatten[StringCases[Part[# , {3}],
            "(" ~~ DigitCharacter ~~ DigitCharacter ~~ DigitCharacter ~~ ")"]],
          {2, 4}
        ] &,
      nielsonKosterLabels];
    racahU =
      Map[
        StringTake[
          Flatten[StringCases[Part[# , {3}],
            "(" ~~ DigitCharacter ~~ DigitCharacter ~~ ")"]],
          {2, 3}
        ] &,
      nielsonKosterLabels];
    fnTermLabels = Join[termLabels, seniorities, racahW, racahU, 2];
    fnTermLabels = Sort[fnTermLabels];
    If[OptionValue["Export"],
      (
        broadFname = FileNameJoin[{moduleDir,"data","fnTerms.m"}];
        Export[broadFname, fnTermLabels];
      )
    ];
    Return[fnTermLabels];
  )
  ]

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

  Options[LoadParameters] = {
      "Source"->"Carnall",
      "Free Ion"->False,
      "gs"->2.002319304386
      };
  LoadParameters::usage="LoadParameters[ln] takes a string with the symbol the element of a
    trivalent lanthanide ion and returns model parameters for it. It is based on the data for
     LaF3. If the option \"Free Ion\" is set to True then the function sets all crystal field
     parameters to zero. Through the option \"gs\" it allows modyfing the electronic
    gyromagnetic ratio.";
  LoadParameters[Ln_String, OptionsPattern[]]:=
    Module[{source, params},
    (
      source = OptionValue["Source"];
      params = Which[source=="Carnall",
              (Association[Carnall["data"][Ln]])
              ];
      (*If a free ion then all the parameters from the crystal field are set to zero*)
      If[OptionValue["Free Ion"],
        Do[params[cfSymbol] = 0,
        {cfSymbol, cfSymbols}
        ]
      ];
      params[F0] = 0;
      params[M2] = 0.56 * params[M0]; (* See Carnall 1989, Table I, caption, probably fixed
    based on HF values*)
      params[M4] = 0.31 * params[M0]; (* See Carnall 1989, Table I, caption, probably fixed
    based on HF values*)
      params[P0] = 0;
      params[P4] = 0.5 * params[P2];  (* See Carnall 1989, Table I, caption, probably fixed based on HF values*)
      params[P6] = 0.1 * params[P2];  (* See Carnall 1989, Table I, caption, probably fixed based on HF values*)
      params[gs] = OptionValue["gs"];
      {params[E0], params[E1], params[E2], params[E3]} = FtoE[params[F0], params[F2], params[F4], params[F6]];
      params[E0] = 0;
      Return[params];
    )
  ];

  HoleElectronConjugation::usage = "HoleElectronConjugation[params] takes the parameters (as an association) that define a configuration and converts them so that they may be interpreted as corresponding to a complentary hole configuration. Some of this can be simply done by changing the sign of the model parameters. In the case of the effective three body interaction the relationship is more complex and is controlled by the value of the isE variable.";

  HoleElectronConjugation[params_] :=
    Module[{newparams = params},
      (
        flipSignsOf = {ζ, T2, T3, T4, T6, T7, T8};
        flipSignsOf = Join[flipSignsOf, cfSymbols];
        flipped =
          Table[(flipper -> - newparams[flipper]),
          {flipper, flipSignsOf}
          ];
        nonflipped =
          Table[(flipper -> newparams[flipper]),
          {flipper, Complement[Keys[newparams], flipSignsOf]}
          ];
        flippedParams = Association[Join[nonflipped, flipped]];
        Return[flippedParams];
      )
    ]


  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.
  This function requires files for pre-computed energy matrix tables that provide the symbols JJBlockMatrixTable[_, _, _, _, _].
  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.
  To account for configurations f^n with n > 7, particle-hole dualities are enforced for ζ and T_i.
  The unit for the returned energies is cm^-1.

  Parameters
  ----------
  nf (int) : Number of f-electrons.
  params (association) : Parameters of the ion under study.

  Returns
  -------
  {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[JJBlockMatrixFileName[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]] = JJBlockMatrixTable[{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.*)
```

```
    (*Parse the results for the eigenvectors in terms of the ordered basis being used.*)
    basis = {};
    Do[basis = Join[basis, EnergyStatesTable[{n, AllowedJ[n][[nn]]}]],
    {nn, 1, Length[AllowedJ[n]]}
    ];
    levels = {};
    Do[levels = Join[levels, {{EigenvalueJM[[nn]], EigenvectorJM[[nn]]}}];,
    {nn, 1, Length[EigenvalueJM]}
    ];
    If[OptionValue["Return Symbolic Matrix"],
    Return[{levels, basis, symbolicMatrix}]
    ];
    Return[{levels, basis}];
  ];

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

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

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

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

    (*In Carnall's approach the crystal field is assumed to have C_{2v} symmetry, which is a
    simplification from the actual point symmetry of C_2*)
    simplifier = {
      B12 -> 0, B14 -> 0, B16 -> 0, B34 -> 0, B36 -> 0, B56 -> 0,
      S12 -> 0, S14 -> 0, S16 -> 0, S22 -> 0, S24 -> 0, S26 -> 0,
      S34 -> 0, S36 -> 0, S44 -> 0, S46 -> 0, S56 -> 0, S66 -> 0,
      T11 -> 0, T12 -> 0, T14 -> 0, T15 -> 0, T16 -> 0, T18 -> 0, T11p -> 0,
      T17 -> 0, T19 -> 0
      };
    eTofs = (#[[1]] -> #[[2]]) & /@ Transpose[{{E0, E1, E2, E3}, FtoE[F0, F2, F4, F6]}];
    ham = Normal[HamMatrixAssembly[numE, 0]];
    simpleHam = ham /. simplifier;
    simpleHam = simpleHam /. eTofs;
    hamParams = DeleteDuplicates[Flatten[Variables /@ simpleHam]];
    ham = Normal[HamMatrixAssembly[numE, 0]];
    termNames = First /@ terms;
    termSimplifier =
      Table[
        termN -> If[StringLength[termN] == 3,
          StringTake[termN, {1, 2}],
          termN
          ],
      {termN, termNames}
      ];

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

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

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

    carnalKey  = StringTemplate["appendix:`Ln`:Calculated"][<|"Ln" -> ln|>];
    gtEnergies = Sort[Carnall[carnalKey]];
    diffs = Sort[cfenergies][[;; Length[gtEnergies]]] - gtEnergies;
    rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];

    Return[{rmsDifference, gtEnergies, cfenergies, ln, carnallAssignments, {fstates, basis,
   symbolicMatrix}}]
    )

  FastIonSolverLaF3::usage =
    "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.

    The function returns a list with seven elements {rmsDifference, carnallEnergies,
   eigenEnergies, ln, carnallAssignments, eigensys, basis}. Where:

    rmsDifference is the root mean squared difference between the calculated values and those
     quoted by Carnall

    carnallEnergies are the quoted calculated values from Carnall;

    eigenEnergies are the calculated energies (in the case of an odd number of electrons the
   kramers degeneracy has been elided from this list);

    ln is simply a string labelling the corresponding lanthanide;

    carnallAssignments is a list of strings providing the term assignments that Carnall
   assumed,

    eigensys is a list of tuples where the first element is the energy corresponding to the
   eigenvector given as the second element;

    basis a list that specifies the basis in which the Hamiltonian was constructed and
   diagonalized.
  ";
  Options[FastIonSolverLaF3] = {
    "MakeNotebook" -> True,
    "NotebookSave" -> True,
    "HTMLSave" -> False,
    "eigenstateTruncationProbability" -> 0.9,
    "Include spin-spin" -> True,
    "Max Eigenstates in Table" -> 100,
    "Sparse" -> True,
    "PrintFun" -> Print,
    "SaveData" -> True,
    "paramFiddle" -> {},
    "Append to Filename" -> ""
    };
  FastIonSolverLaF3[numE_, OptionsPattern[]] := Module[{
    makeNotebook, eigenstateTruncationProbability, spinspin, host,
    ln, terms, termNames, carnallEnergies, eigenEnergies,simplerStateLabels,
    eigensys, basis, assignmentMatches, stateLabels, carnallAssignments},
  (
    PrintFun = OptionValue["PrintFun"];
    makeNotebook = OptionValue["MakeNotebook"];
    eigenstateTruncationProbability = OptionValue["eigenstateTruncationProbability"];
    maxStatesInTable = OptionValue["Max Eigenstates in Table"];
    spinspin = OptionValue["Include spin-spin"];
    host = "LaF3";
    paramFiddle = OptionValue["paramFiddle"];
    ln = theLanthanides[[numE]];
    terms = AllowedNKSLJTerms[Min[numE, 14 - numE]];
    termNames = First /@ terms;
    (* For labeling the states, the degeneracy in some of the terms is elided *)
    PrintFun["> Calculating simpler term labels ..."];
    termSimplifier =
      Table[termN -> If[StringLength[termN] == 3,
        StringTake[termN, {1, 2}],
        termN
        ],
      {termN, termNames}
      ];

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

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

    (*Import the symbolic Hamiltonian*)
    PrintFun["> Loading the symbolic Hamiltonian for this configuration ..."];
    startTime = Now;
    numH = 14 - numE;
    numEH = Min[numE, numH];
    simpleHam =
      If[ValueQ[symbolicHamiltonians],
        (
          If[MemberQ[Keys[symbolicHamiltonians], numEH],
            symbolicHamiltonians[numEH],
            Import["./hams/SymbolicMatrix-f" <> ToString[numEH] <> ".m"]
          ]
        ),
        Import["./hams/SymbolicMatrix-f" <> ToString[numEH] <> ".m"]
      ];
    endTime   = Now;
    loadTime = QuantityMagnitude[endTime - startTime, "Seconds"];
    PrintFun[">> Loading the symbolic Hamiltonian took ", loadTime, " seconds."];

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

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

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

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

```

  (*Energies are doubly degenerate in the case of odd number of electrons, keep only one*)
  If[OddQ[numE],
    (
      PrintFun["> Since there's an odd number of electrons energies come in pairs, taking
 just one for each pair ..."];
      eigenEnergies = eigenEnergies[[;; ;; 2]];
    )
  ];

  (*Compare against the data quoted by Bill Carnall*)
  PrintFun["> Comparing against the data from Carnall ..."];
  mainKey             = StringTemplate["appendix:`Ln`:Association"][<|"Ln" -> ln|>];
  lnData              = Carnall[mainKey];
  carnalKeys          = lnData // Keys;
  repetitions         = Length[lnData[#]["Calc (1/cm)"]] & /@ carnalKeys;
  carnallAssignments = First /@ Carnall["appendix:" <> ln <> ":RawTable"];
  carnalKey           = StringTemplate["appendix:`Ln`:Calculated"][<|"Ln" -> ln|>];
  carnallEnergies     = Carnall[carnalKey];

  (* For the difference take as many energies as quoted by Bill*)
  eigenEnergies = eigenEnergies + carnallEnergies[[1]];
  diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] - carnallEnergies;
  (* Remove the differences where the appendix tables have elided values*)
  rmsDifference = Sqrt[Mean[(Select[diffs, FreeQ[#, Missing[]] &])^2]];
  titleTemplate = StringTemplate[
    "Energy Level Diagram of \!\(\*SuperscriptBox[\(`ion`\), \(\(3\)\(+\)\)]\)"];
  title = titleTemplate[<|"ion" -> ln|>];
  parsedStates = ParseStates[eigensys, basis];
  If[OddQ[numE],
    parsedStates = parsedStates[[;; ;; 2]]];

  stateLabels = #[[-1]] & /@ parsedStates;
  simplerStateLabels = ((#[[2]] /. termSimplifier) <> ToString[#[[3]], InputForm]) & /@
 parsedStates;

  PrintFun[">> Truncating eigenvectors to given probability ..."];
  startTime = Now;
  truncatedStates = ParseStatesByProbabilitySum[eigensys, basis,
      eigenstateTruncationProbability,
      0.01];
  endTime = Now;
  truncationTime = QuantityMagnitude[endTime - startTime, "Seconds"];
  PrintFun[">>> Truncation took ", truncationTime, " seconds."];

  If[makeNotebook,
    (
      PrintFun["> Putting together results in a notebook ..."];
      energyDiagram = Framed[
        EnergyLevelDiagram[eigensys, "Title" -> title,
        "Background" -> White]
        , Background -> White, FrameMargins -> 50];
      appToFname = OptionValue["Append to Filename"];
      PrintFun[">> Comparing the term assignments between qlanth and Carnall ..."];
      assignmentMatches =
      If[StringContainsQ[#[[1]], #[[2]]], "\[Checkmark]", "X"] & /@
        Transpose[{carnallAssignments, simplerStateLabels[[;; Length[carnallAssignments
 ]]]}];
      assignmentMatches = {{"\[Checkmark]",
        Count[assignmentMatches, "\[Checkmark]"]}, {"X",
        Count[assignmentMatches, "X"]}};
      labelComparison = (If[StringContainsQ[#[[1]], #[[2]]], "\[Checkmark]", "X"] & /@
        Transpose[{carnallAssignments,
          simplerStateLabels[[;; Length[carnallAssignments]]]}]);
      labelComparison =
      PadRight[labelComparison, Length[simplerStateLabels], "-"];

      statesTable =
      Grid[Prepend[{Round[#[[1]]], #[[2]]} & /@
          truncatedStates[[;;Min[Length[eigensys],maxStatesInTable]]], {"Energy/\!\(\*
  SuperscriptBox[\(cm\), \(-1\)]\)",
          "\[Psi]"}], Frame -> All, Spacings -> {2, 2},
        FrameStyle -> Blue,
        Dividers -> {{False, True, False}, {True, True}}];
      DefaultIfMissing[expr_]:= If[FreeQ[expr, Missing[]], expr,"NA"];
      PrintFun[">> Rounding the energy differences for table presentation ..."];
      roundedDiffs = Round[diffs, 0.1];
      roundedDiffs = PadRight[roundedDiffs, Length[simplerStateLabels], "-"];
      roundedDiffs = DefaultIfMissing /@ roundedDiffs;
      diffs = PadRight[diffs, Length[simplerStateLabels], "-"];
      diffs = DefaultIfMissing /@ diffs;
      diffTableData = Transpose[{simplerStateLabels, eigenEnergies,
          labelComparison,
          PadRight[carnallAssignments, Length[simplerStateLabels], "-"],
          DefaultIfMissing/@PadRight[carnallEnergies, Length[simplerStateLabels], "-"],
          roundedDiffs}];
      diffTable =
      TableForm[diffTableData,
        TableHeadings -> {None, {"qlanth",
          "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)", "", "Carnall",
          "E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)",
          "\[CapitalDelta]E/\!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"}}];

      diffs = Sort[eigenEnergies][[;; Length[carnallEnergies]]] - carnallEnergies;
      notBad = FreeQ[#,Missing[]]&/@diffs;
      diffs = Pick[diffs,notBad];
      diffHistogram =
      Histogram[diffs, Frame -> True, ImageSize -> 800,
        AspectRatio -> 1/3, FrameStyle -> Directive[16],
        FrameLabel -> {"(qlanth-carnall)/Ky", "Freq"}];
      rmsDifference = Sqrt[Total[diffs^2/Length[diffs]]];
      labelTempate =
      StringTemplate[
        "\!\(\*SuperscriptBox[\(`ln`\), \(\(3\)\(+\)\)]\)"];
      diffData = diffs;
      diffLabels = simplerStateLabels[[;;Length[notBad]]];
      diffLabels = Pick[diffLabels, notBad];
      diffPlot = Framed[
        ListLabelPlot[diffData,
        diffLabels,
        Frame -> True,
        PlotRange -> All,
        ImageSize -> 1200,
        AspectRatio -> 1/3,
        FrameLabel -> {"",
          "(qlanth-carnall) / \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)"},
        PlotMarkers -> "OpenMarkers",
        PlotLabel ->
          Style[labelTempate[<|"ln" -> ln|>] <> " | " <> "\[Sigma]=" <>
            ToString[Round[rmsDifference, 0.01]] <>
            " \!\(\*SuperscriptBox[\(cm\), \(-1\)]\)\n", 20],
        Background -> White
        ],
        Background -> White,
        FrameMargins -> 50
        ];
      nb = CreateDocument[{
        TextCell[Style[DisplayForm[SuperscriptBox[host <> ":" <> ln, "3+"]]], "Title", TextAlignment -> Center],
        TextCell["Energy Diagram", "Section", TextAlignment -> Center],
        TextCell[energyDiagram, TextAlignment -> Center],
        TextCell["Multiplet Assignments & Energy Levels", "Section", TextAlignment -> Center],
        TextCell[diffHistogram, TextAlignment -> Center],
        TextCell[diffPlot, "Output", TextAlignment -> Center],
        TextCell[assignmentMatches, "Output", TextAlignment -> Center],
        TextCell[diffTable, "Output", TextAlignment -> Center],
        TextCell["Truncated Eigenstates", "Section", TextAlignment -> Center],
        TextCell["These are some of the resultant eigenstates which add up to at least a total probability of " <> ToString[eigenstateTruncationProbability] <> ".", "Text", TextAlignment -> Center],
        TextCell[statesTable, "Output", TextAlignment -> Center]
        },
      WindowSelected -> True,
      WindowTitle -> ln <> " in " <> "LaF3" <> appToFname,
      WindowSize -> {1600, 800}];
      If[OptionValue["SaveData"],
        (
          exportFname = FileNameJoin[{moduleDir,"calcs", ln <> " in " <> "LaF3" <> appToFname <> ".m"}];
          SelectionMove[nb, After, Notebook];
          NotebookWrite[nb, Cell["Reload Data", "Section", TextAlignment -> Center]];
          NotebookWrite[nb, Cell[(
            "{rmsDifference, carnallEnergies, eigenEnergies, ln, carnallAssignments, simplerStateLabels, eigensys, basis, truncatedStates} = Import[FileNameJoin[{NotebookDirectory[],\"" <> StringSplit[exportFname,"/"][[-1]] <> "\"}]];"
            ),"Input"]];
          NotebookWrite[nb, Cell[(
            "Manipulate[First[MinimalBy[truncatedStates, Abs[First[#] - energy] &]], {energy,0}]"
            ),"Input"]];
          SelectionMove[nb, Before, Notebook];
          Export[exportFname, {rmsDifference, carnallEnergies, eigenEnergies, ln, carnallAssignments, simplerStateLabels, eigensys, basis, truncatedStates}];
          tinyexportFname = FileNameJoin[{moduleDir,"calcs", ln <> " in " <> "LaF3" <> appToFname <> "- tiny.m"}];
          tinyExport = <|"ln"->ln,
                        "carnallEnergies"->carnallEnergies,
                        "rmsDifference"-> rmsDifference,
                        "eigenEnergies"-> eigenEnergies,
                        "carnallAssignments"-> carnallAssignments,
                        "simplerStateLabels" -> simplerStateLabels|>;
          Export[tinyexportFname, tinyExport];
        )
      ];
      If[OptionValue["NotebookSave"],
        (
          nbFname = FileNameJoin[{moduleDir,"calcs", ln <> " in " <> "LaF3" <> appToFname <> ".nb"}];
          PrintFun[">> Saving notebook to ", nbFname, " ..."];
          NotebookSave[nb, nbFname];
        )
      ];
      If[OptionValue["HTMLSave"],
        (
          htmlFname = FileNameJoin[{moduleDir,"calcs", "html", ln <> " in " <> "LaF3" <>
```

```
  appToFname <> ".html"}];
          PrintFun[">> Saving html version to ", htmlFname, " ..."];
          Export[htmlFname, nb];
        )
      ];
    )
  ];

  Return[{rmsDifference, carnallEnergies, eigenEnergies, ln, carnallAssignments,
  simplerStateLabels, eigensys, basis, truncatedStates}];
  )
];

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

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

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

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

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

LSJMJTemplate =
  StringTemplate[
  "\!\(\*TemplateBox[{\nRowBox[{\"`LS`\", \",\", \nRowBox[{\"J\", \
\"=\", \"`J`\"}], \",\", \nRowBox[{\"mJ\", \"=\", \"`mJ`\"}]}]},\n\
\"Ket\"]\)"];
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=...>."
BasisVecInLSJMJ[basisVec_] := (
  {LSstring, Jval, mJval} = basisVec;
  LSJMJTemplate[<|
    "LS" -> LSstring,
    "J" -> ToString[Jval, InputForm],
    "mJ" -> ToString[mJval, InputForm]|>]
  );

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.";
ParseStates[states_, basis_, OptionsPattern[]] := Module[{parsedStates},
(
  parsedStates = Table[(
    {energy, eigenVec} = state;
    maxTermIndex = Ordering[Abs[eigenVec]][[-1]];
    {LSstring, Jval, mJval} = basis[[maxTermIndex]];
    LSJsymbol = Subscript[LSstring, {Jval, mJval}];
    LSJMJsymbol = LSstring <> ToString[Jval, InputForm];
    {S, L} = FindSL[LSstring];
    {energy, LSstring, Jval, mJval, S, L, LSJsymbol, LSJMJsymbol}
    ),
    {state, states}];
  Return[parsedStates]
  )
]

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.";
ParseStatesByNumBasisVecs[states_, basis_, numBasisVecs_, roundTo_ : 0.01] := (
  parsedStates = Table[(
    {energy, eigenVec} = state;
    energy = Chop[energy];
    probs = Round[Abs[eigenVec^2], roundTo];
    amplitudes = Round[eigenVec, roundTo];
    ordering = Ordering[probs];
    chosenIndices = ordering[[-numBasisVecs ;;]];
    majorComponents = basis[[chosenIndices]];
    majorProbabilities = amplitudes[[chosenIndices]];
    majorComponents = BasisVecInLSJMJ /@ majorComponents;
    majorRep = majorProbabilities . majorComponents;
    {energy, majorRep}
    ),
    {state, fstates}];
  Return[parsedStates]
  )

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.
  ";
FindThresholdPosition[list_, threshold_] :=
Module[{position},
  position = Position[list, _?(# > threshold &), 1, 1];
  thrPos = If[Length[position] > 0,
    position[[1, 1]],
    Length[list]];
  If[thrPos == 0, Return[1], Return[thrPos+1]]
  ]

ParseStateByProbabilitySum[{energy_, eigenVec_}, probSum_, roundTo_:0.01, maxParts_:20] :=
  Compile[
  {{energy, _Real, 0},{eigenVec, _Complex, 1}, {probSum, _Real, 0}, {roundTo, _Real, 0}, {
  maxParts, _Integer, 0}},
  Module[
  {numStates, state, amplitudes, probs, ordering,
  orderedProbs, truncationIndex, accProb, thresholdIndex, chosenIndices, majorComponents,
  majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
(
  numStates    = Length[eigenVec];
  (*Round them up*)
  amplitudes          = Round[eigenVec, roundTo];
  probs               = Round[Abs[eigenVec^2], roundTo];
  ordering            = Reverse[Ordering[probs]];
  (*Order the probabilities from high to low*)
  orderedProbs        = probs[[ordering]];
  (*To speed up Accumulate, assume that only as much as maxParts will be needed*)
  truncationIndex     = Min[maxParts, Length[orderedProbs]];
  orderedProbs        = orderedProbs[[;;truncationIndex]];
  (*Accumulate the probabilities*)
  accProb             = Accumulate[orderedProbs];
  (*Find the index of the first element in accProb that is greater than probSum*)
  thresholdIndex      = Min[Length[accProb], FindThresholdPosition[accProb, probSum]];
  (*Grab all the indicees up till that one*)
  chosenIndices       = ordering[[;; thresholdIndex]];
  (*Select the corresponding elements from the basis*)
  majorComponents     = basis[[chosenIndices]];
  (*Select the corresponding amplitudes*)
  majorAmplitudes     = amplitudes[[chosenIndices]];
  (*Take their absolute value*)
  absMajorAmplitudes = Abs[majorAmplitudes];
  (*Make sure that there are no effectively zero contributions*)
  notnullAmplitudes  = Flatten[Position[absMajorAmplitudes, x_ /; x != 0]];
  (* majorComponents    = PrettySaundersSLJmJ[{#[[1]],#[[2]],#[[3]]}] & /@ majorComponents;
   *)
  majorComponents     = PrettySaundersSLJmJ /@ majorComponents;
  majorAmplitudes     = majorAmplitudes[[notnullAmplitudes]];
  (*Make them into Kets*)
  majorComponents     = Ket /@ majorComponents[[notnullAmplitudes]];
  (*Multiply and add to build the final Ket*)
  majorRep            = majorAmplitudes . majorComponents;
    );
Return[{energy, majorRep}]
],
  CompilationTarget -> "C",
  RuntimeAttributes -> {Listable},
  Parallelization -> True,
  RuntimeOptions -> "Speed"
];

ParseStatesByProbabilitySum::usage = "ParseStatesByProbabilitySum[eigensys, basis, probSum]
   takes a list of eigenstates in terms of their coefficients in the given basis and
  returns a list of the same states in terms of their energy and the coefficients of the
  basis vectors that sum to at least probSum.";
ParseStatesByProbabilitySum[eigensys_, basis_, probSum_, roundTo_ : 0.01, maxParts_: 20] :=
   Module[
  {parsedByProb, numStates, state, energy, eigenVec, amplitudes, probs, ordering,
  orderedProbs, truncationIndex, accProb, thresholdIndex, chosenIndices, majorComponents,
  majorAmplitudes, absMajorAmplitudes, notnullAmplitudes, majorRep},
(
  numStates    = Length[eigensys];
  parsedByProb = Table[(
    state               = eigensys[[idx]];
    {energy, eigenVec} = state;
    (*Round them up*)
    amplitudes          = Round[eigenVec, roundTo];
    probs               = Round[Abs[eigenVec^2], roundTo];
    ordering            = Reverse[Ordering[probs]];
    (*Order the probabilities from high to low*)
    orderedProbs        = probs[[ordering]];
    (*To speed up Accumulate, assume that only as much as maxParts will be needed*)
    truncationIndex     = Min[maxParts, Length[orderedProbs]];
    orderedProbs        = orderedProbs[[;;truncationIndex]];
    (*Accumulate the probabilities*)
    accProb             = Accumulate[orderedProbs];
    (*Find the index of the first element in accProb that is greater than probSum*)
    thresholdIndex      = Min[Length[accProb], FindThresholdPosition[accProb, probSum]];
    (*Grab all the indicees up till that one*)
    chosenIndices       = ordering[[;; thresholdIndex]];
    (*Select the corresponding elements from the basis*)
    majorComponents     = basis[[chosenIndices]];
    (*Select the corresponding amplitudes*)
    majorAmplitudes     = amplitudes[[chosenIndices]];
    (*Take their absolute value*)
    absMajorAmplitudes = Abs[majorAmplitudes];
    (*Make sure that there are no effectively zero contributions*)
    notnullAmplitudes  = Flatten[Position[absMajorAmplitudes, x_ /; x != 0]];
    (* majorComponents     = PrettySaundersSLJmJ[{#[[1]],#[[2]],#[[3]]}] & /@
  majorComponents; *)
    majorComponents     = PrettySaundersSLJmJ /@ majorComponents;
    majorAmplitudes     = majorAmplitudes[[notnullAmplitudes]];
    (*Make them into Kets*)
    majorComponents     = Ket /@ majorComponents[[notnullAmplitudes]];
    (*Multiply and add to build the final Ket*)
    majorRep            = majorAmplitudes . majorComponents;
    {energy, majorRep}
    ), {idx, numStates}];
Return[parsedByProb]
)
];

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

(* ###################################################################### *)
(* ################################# Misc ################################ *)

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.";
SymbToNum[expr_, replacementAssociation_]:= (
  includedKeys = Keys[replacementAssociation];
  (*If a key is not defined, make its value zero.*)
  fullAssociation = Table[(
    If[MemberQ[includedKeys, key],
      ToExpression[key]->replacementAssociation[key],
      ToExpression[key]->0
    ]
  ),
  {key, paramSymbols}];
  Return[expr/.fullAssociation];
)

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

ExportMZip::usage =
  "ExportMZip[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.";
ExportMZip[filename_, object_] := (
  zipTemplate = StringTemplate["cd \"`sourceDir`\"; zip \"`dest`\" \"`source`\""];
  delTemplate = StringTemplate["rm \"`rmFname`\""];
  Export[filename, object];
  zipFilename = StringReplace[filename, ".m" -> ".zip"];
  splitName = FileNameSplit[zipFilename];
  zipFilename = splitName[[-1]];
  sourceDir = FileNameJoin[splitName[[1 ;; -2]]];
  zipCmd =
    zipTemplate[<|"sourceDir" -> sourceDir, "dest" -> zipFilename,
      "source" -> FileNameSplit[filename][[-1]]|>];
  delCmd = delTemplate[<|"rmFname" -> filename|>];
  Run[zipCmd];
  Run[delCmd];
  );

ImportMZip::usage =
  "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.";
ImportMZip[filename_] := (
  splitName     = FileNameSplit[filename];
  sourceDir     = FileNameJoin[splitName[[1 ;; -2]]];
  delTemplate   = StringTemplate["rm \"`rmFname`\""];
  unzipTemplate = StringTemplate["cd \"`sourceDir`\"; unzip \"`source`\""];
  unzipCmd      = unzipTemplate[<|"sourceDir" -> sourceDir,
                                  "source" -> FileNameSplit[filename][[-1]]|>];
  Run[unzipCmd];
  mFilename = StringReplace[filename, ".zip" -> ".m"];
  imported  = Import[mFilename];
  delCmd    = delTemplate[<|"rmFname" -> mFilename|>];
  Run[delCmd];
  Return[imported];
  );
```

```

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

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

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

        supPart =
          StringCases[str,
            RegularExpression@"\\{(.*?)\\}" -> "$1"][[1]];
        Superscript[mainSymbol, supPart]),
        StringContainsQ[str, "_"] && StringContainsQ[str, "^"],
        (
        (*yes sub yes sup*)
        mainSymbol = StringSplit[str, "_"][[1]];
        mainSymbol = ToExpression[mainSymbol];
        {subPart, supPart} =
          StringCases[str, RegularExpression@"\\{(.*?)\\}" -> "$1"];
        Subsuperscript[mainSymbol, subPart, supPart]
```

```
          ),
          True,
          ((*no sup or sub*)
          str)
          ];
        symbol
        ),
      {string, StringSplit[bigString, " "]}];
    Which[
      form == "Row",
      Return[Row[symbols]],
      form == "List",
      Return[symbols]
      ]
    );

  (* ################################# Misc ################################## *)
  (* ###################################################################### *)

  (* ###################################################################### *)
  (* ######################## Some Plotting Routines ######################## *)

  EnergyLevelDiagram::usage = "EnergyLevelDiagram[states] takes states and produces a
   visualization of its energy spectrum.
  The resultant visualization can be navigated by clicking and dragging to zoom in on a
   region, or by clicking and dragging horizontally while pressing Ctrl. Double-click to
   reset the view.";
  Options[EnergyLevelDiagram] = {
    "Title"->"",
    "ImageSize"->1000,
    "AspectRatio" -> 1/8,
    "Background"->"Automatic",
    "Epilog"->{}
    };
  EnergyLevelDiagram[states_, OptionsPattern[]]:= (
    energies = First/@states;
    epi = OptionValue["Epilog"];
    ExploreGraphics@ListPlot[Tooltip[{{#, 0}, {#, 1}}, {Quantity[#/8065.54429, "eV"],
   Quantity[#, 1/"Centimeters"]}] &/@ energies,
      Joined        -> True,
      PlotStyle     -> Black,
      AspectRatio   -> OptionValue["AspectRatio"],
      ImageSize     -> OptionValue["ImageSize"],
      Frame         -> True,
      PlotRange     -> {All, {0, 1}},
      FrameTicks    -> {{None, None}, {Automatic, Automatic}},
      FrameStyle    -> Directive[15, Dashed, Thin],
      PlotLabel     -> Style[OptionValue["Title"], 15, Bold],
      Background    -> OptionValue["Background"],
      FrameLabel    -> {"\!\(\*FractionBox[\(E\), SuperscriptBox[\(cm\), \(-1\)]]\)"},
      Epilog        -> epi]
  )

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

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

```

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

        range = {{rx1, rx2}, {ry1, ry2}} =
          Transpose@{first, second};
        range[[2]] = {0, 1}],
      {"MouseDown", 2} :> (
        first = {sx1, sy1} = MousePosition["Graphics"]
        ),
      {"MouseDragged", 2} :> (
        second = {sx2, sy2} = MousePosition["Graphics"];
        rx1 = rx1 - (sx2 - sx1);
        rx2 = rx2 - (sx2 - sx1);
        ry1 = ry1 - (sy2 - sy1);
        ry2 = ry2 - (sy2 - sy1);
        range = {{rx1, rx2}, {ry1, ry2}};
        range[[2]] = {0, 1};
        )}]]];

Options[LabeledGrid]={
    ItemSize->Automatic,
    Alignment->Center,
    Frame->All,
    "Separator"->",",
    "Pivot"->""
};
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.";
LabeledGrid[data_,rowHeaders_,columnHeaders_,OptionsPattern[]]:=Module[
    {gridList=data,rowHeads=rowHeaders,colHeads=columnHeaders},
    (
    separator=OptionValue["Separator"];
    pivot=OptionValue["Pivot"];
    gridList=Table[
            Tooltip[
              data[[rowIdx,colIdx]],
              DisplayForm[
                RowBox[{rowHeads[[rowIdx]],
                        separator,
                        colHeads[[colIdx]]}
                       ]
                       ]
                     ],
        {rowIdx,Dimensions[data][[1]]},
        {colIdx,Dimensions[data][[2]]}];
    gridList=Transpose[Prepend[gridList,colHeads]];
    rowHeads=Prepend[rowHeads,pivot];
    gridList=Prepend[gridList,rowHeads]//Transpose;
    Grid[gridList,
        Frame->OptionValue[Frame],
        Alignment->OptionValue[Alignment],
        Frame->OptionValue[Frame],
        ItemSize->OptionValue[ItemSize]
        ]
)
]

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

Options[HamiltonianMatrixPlot] = Join[Options[MatrixPlot], {"Hover" -> True, "Overlay
  Values" -> True}];
HamiltonianMatrixPlot[hamMatrix_, basisLabels_, opts : OptionsPattern[]] := (
   braLabels = DisplayForm[RowBox[{"\[LeftAngleBracket]", #, "\[RightBracketingBar]"}]] & /@
    basisLabels;
   ketLabels = DisplayForm[Rotate[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"
  }],\[Pi]/2]] & /@ basisLabels;
   ketLabelsUpright = DisplayForm[RowBox[{"\[LeftBracketingBar]", #, "\[RightAngleBracket]"
  }]] & /@ basisLabels;
   numRows = Length[hamMatrix];
   numCols = Length[hamMatrix[[1]]];
```

```
    epiThings = Which[
      And[OptionValue["Hover"], Not[OptionValue["Overlay Values"]]],
      Flatten[
        Table[
          Tooltip[
            {
              Transparent,
              Rectangle[
              {j - 1, numRows - i},
              {j - 1, numRows - i} + {1, 1}
              ]
            },
          Row[{braLabels[[i]],ketLabelsUpright[[j]],"=",hamMatrix[[i, j]]}]
          ],
        {i, 1, numRows},
        {j, 1, numCols}
        ]
      ],
      And[OptionValue["Hover"], OptionValue["Overlay Values"]],
      Flatten[
        Table[
          Tooltip[
            {
              Transparent,
              Rectangle[
              {j - 1, numRows - i},
              {j - 1, numRows - i} + {1, 1}
              ]
            },
          DisplayForm[RowBox[{"\[LeftAngleBracket]", basisLabels[[i]], "\[LeftBracketingBar]"
    , basisLabels[[j]], "\[RightAngleBracket]"}]]
          ],
        {i, numRows},
        {j, numCols}
        ]
      ],
      True,
      {}
      ];
    textOverlay = If[OptionValue["Overlay Values"],
      (
        Flatten[
        Table[
          Text[hamMatrix[[i, j]],
            {j - 1/2, numRows - i + 1/2}
          ],
        {i, 1, numRows},
        {j, 1, numCols}
        ]
        ]
      ),
      {}
      ];
    epiThings = Join[epiThings, textOverlay];
    MatrixPlot[hamMatrix,
      FrameTicks -> {
        {Transpose[{Range[Length[braLabels]], braLabels}], None},
        {None, Transpose[{Range[Length[ketLabels]], ketLabels}]}
        },
      Evaluate[FilterRules[{opts}, Options[MatrixPlot]]],
      Epilog -> epiThings
    ]
```

```
  );

(* ######################## Some Plotting Routines ######################### *)
(* ######################################################################### *)

(* ######################################################################### *)
(* ############################ Load Functions ############################ *)

LoadAll::usage="LoadAll[] executes all Load* functions.";
LoadAll[]:=(
    LoadTermLabels[];
    LoadCFP[];
    LoadUk[];
    LoadVk1[];
    LoadT22[];
    LoadSOOandECSOLS[];

    LoadElectrostatic[];
    LoadSpinOrbit[];
    LoadSOOandECSO[];
    LoadSpinSpin[];
    LoadThreeBody[];
    LoadChenDeltas[];
    LoadCarnall[];
  )

LoadTermLabels::usage="LoadTermLabels[] loads into the session the labels for the terms in
  the f^n configurations.";
LoadTermLabels[]:= (
  If[ValueQ[fnTermLabels], Return[]];
  PrintTemporary["Loading data for state labels in the f^n configurations..."];
  fnTermsFname = FileNameJoin[{moduleDir, "data", "fnTerms.m"}];
  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.";
  If[!FileExistsQ[fnTermsFname],
    (PrintTemporary[">> fnTerms.m not found, generating ..."];
      fnTermLabels = ParseTermLabels["Export"->True];
    ),
    fnTermLabels = Import[fnTermsFname];
  ];
)

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

LoadChenDeltas::usage="LoadChenDeltas[] loads the differences noted by Chen.";
```

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

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

  chenDeltasRawB = Import[FileNameJoin[{moduleDir, "data", "the-chen-deltas-B.csv"}], "Text
  "];
  chenDeltasB = StringSplit[chenDeltasRawB, "\n"];
  chenDeltasB = StringSplit[#, ","] & /@ chenDeltasB;
  chenDeltasB = {ToExpression[StringTake[#[[1]], {2}]], #[[2]], #[[3]]} & /@ chenDeltasB;
  chenDeltas["B"] = chenDeltasB;
  On[Power::infy];
  If[OptionValue["Export"],
    (chenDeltasFname = FileNameJoin[{moduleDir, "data", "chenDeltas.m"}];
    Export[chenDeltasFname, chenDeltas];
    )
    ];
  Return[chenDeltas];
)

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";
Options[ParseCarnall] = {"Export" -> True};
ParseCarnall[] := (
  ions         = {"Pr","Nd","Pm","Sm","Eu","Gd","Tb","Dy","Ho","Er","Tm"};
  templates    = StringTemplate/@StringSplit["appendix:`ion`:Association appendix:`ion`:
  Calculated appendix:`ion`:RawTable appendix:`ion`:Headings"," "];

  (* How many unique eigenvalues, after removing Kramer's degeneracy *)
  fullSizes    = AssociationThread[ions, {91, 182, 1001, 1001, 3003, 1716, 3003, 1001,
 1001, 182, 91}];
  carnall      = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[2]];
  carnallErr   = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[3]];

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

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

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

```

  Do[(
      carnallData   = Import[FileNameJoin[{moduleDir,"data","Carnall.xls"}]][[i]];
      headers       = carnallData[[1]];
      calcIndex     = Position[headers,"Calc (1/cm)"][[1,1]];
      headers       = headers[[2;;]];
      carnallLabels = carnallData[[1]];
      carnallData   = carnallData[[2;;]];
      carnallTerms  = DeleteDuplicates[First/@carnallData];
      parsedData    = Table[(
                          rows = Select[carnallData,#[[1]]==term&];
                          rows = #[[2;;]]&/@rows;
                          rows = Transpose[rows];
                          rows = Transpose[{headers,rows}];
                          rows = Association[(#[[1]]->#[[2]])&/@rows];
                          term->rows
                          ),
                      {term,carnallTerms}
                      ];
      carnallAssoc        = Association[parsedData];
      carnallCalcEnergies = #[[calcIndex]]&/@carnallData;
      carnallCalcEnergies = If[NumberQ[#],#,Missing[]]&/@carnallCalcEnergies;
      ion                 = ions[[i-3]];
      carnallCalcEnergies = PadRight[carnallCalcEnergies, fullSizes[ion], Missing[]];
      keys                = #[<|"ion"->ion|>]&/@templates;
      Carnall[keys[[1]]]  = carnallAssoc;
      Carnall[keys[[2]]]  = carnallCalcEnergies;
      Carnall[keys[[3]]]  = carnallData;
      Carnall[keys[[4]]]  = headers;
      ),
  {i,4,14}
  ];

  goodions  = Select[ions,#!="Pm"&];
  expData   = Select[Transpose[Carnall["appendix:"<>#<>":RawTable"]][[1+Position[Carnall["
 appendix:"<>#<>":Headings"],"Exp (1/cm)"][[1,1]]]],NumberQ]&/@goodions;
  Carnall["All Experimental Data"]=AssociationThread[goodions,expData];
  If[OptionValue["Export"],
    (
      carnallFname = FileNameJoin[{moduleDir, "data", "Carnall.m"}];
      Print["Exporting to "<>exportFname];
      Export[carnallFname, Carnall];
    )
    ];
  Return[Carnall];
)

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."
  ;
Options[LoadSymbolicHamiltonians] = {"Reload" -> False};
LoadSymbolicHamiltonians[numEs_:All, OptionsPattern[]]:=(
  If[numEs === All,
  numEs = {2, 3, 4, 5, 6, 7}];
  If[Not[ValueQ[symbolicHamiltonians]],symbolicHamiltonians = <||>];
  Do[
    (
    If[And[
        MemberQ[Keys[symbolicHamiltonians], numE],
        Not[OptionValue["Reload"]]],
      Continue[]
      ];
    PrintTemporary["Loading symbolic Hamiltonian for f" <> ToString[numE] <> " ..." ];
    symbolicHamiltonians[numE] = Import["./hams/SymbolicMatrix-f" <> ToString[numE] <> ".m"
  ];
    ),
    {numE, numEs}
  ]
)

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";
CFPAssoc::usage = " CFPAssoc is an association where keys are of lists of the form {
  num_electrons, daugherTerm, parentTerm} and values are the corresponding coefficients of
  fractional parentage. The terms given in string-spectroscopic notation. If a certain
  daughter term does not have a parent term, the value is 0. Loaded using LoadCFP[].";
LoadCFP::usage="LoadCFP[] loads CFP, CFPAssoc, and CFPTable into the session.";
LoadCFP[]:=(
  If[And[ValueQ[CFP], ValueQ[CFPTable], ValueQ[CFPAssoc]],Return[]];

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

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

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

ReducedUkTable::usage = "ReducedUkTable[{n, l = 3, SL, SpLp, k}] provides reduced matrix
  elements of the spherical tensor operator Uk. See Cowan (1981) section 11-9 \"Unit Tensor
   Operators\". Loaded using LoadUk[].";
LoadUk::usage="LoadUk[] loads into session the reduced matrix elements for unit tensor
  operators.";
LoadUk[]:=(
  If[ValueQ[ReducedUkTable], Return[]];
  PrintTemporary["Loading the association of reduced matrix elements for unit tensor
  operators ..."];
  ReducedUkTableFname = FileNameJoin[{moduleDir, "data", "ReducedUkTable.m"}];
  If[!FileExistsQ[ReducedUkTableFname],
    (PrintTemporary[">> ReducedUkTable.m not found, generating ..."];
      ReducedUkTable = GenerateReducedUkTable[7];
    ),
    ReducedUkTable = Import[ReducedUkTableFname];
  ];
);

ElectrostaticTable::usage = "ElectrostaticTable[{n, SL, SpLp}] provides the calculated
  result of Electrostatic[n, SL, SpLp]. Load using LoadElectrostatic[].";
LoadElectrostatic::usage="LoadElectrostatic[] loads the reduced matrix elements for the
  electrostatic interaction.";
LoadElectrostatic[]:=(
  If[ValueQ[ElectrostaticTable], Return[]];
  PrintTemporary["Loading the association of matrix elements for the electrostatic
  interaction ..."];
  ElectrostaticTablefname = FileNameJoin[{moduleDir, "data", "ElectrostaticTable.m"}];
  If[!FileExistsQ[ElectrostaticTablefname],
    (PrintTemporary[">> ElectrostaticTable.m not found, generating ..."];
      ElectrostaticTable = GenerateElectrostaticTable[7];
    ),
    ElectrostaticTable = Import[ElectrostaticTablefname];
  ];
);

LoadVk1::usage="LoadVk1[] loads into session the matrix elements of Vk1.";
LoadVk1[]:=(
  If[ValueQ[ReducedV1kTable], Return[]];
  PrintTemporary["Loading the association of matrix elements for Vk1 ..."];
  ReducedV1kTableFname = FileNameJoin[{moduleDir, "data", "ReducedV1kTable.m"}];
  If[!FileExistsQ[ReducedV1kTableFname],
    (PrintTemporary[">> ReducedV1kTable.m not found, generating ..."];
      ReducedV1kTable = GenerateReducedV1kTable[7];
    ),
    ReducedV1kTable = Import[ReducedV1kTableFname];
  ]
);

LoadSpinOrbit::usage="LoadSpinOrbit[] loads into session the matrix elements of the spin-
  orbit interaction.";
LoadSpinOrbit[]:=(
  If[ValueQ[SpinOrbitTable], Return[]];
  PrintTemporary["Loading the association of matrix elements for spin-orbit ..."];
  SpinOrbitTableFname = FileNameJoin[{moduleDir, "data", "SpinOrbitTable.m"}];
  If[!FileExistsQ[SpinOrbitTableFname],
    (PrintTemporary[">> SpinOrbitTable.m not found, generating ..."];
      SpinOrbitTable = GenerateSpinOrbitTable[7, True];
    ),
    SpinOrbitTable = Import[SpinOrbitTableFname];
  ]
);

LoadSOOandECSOLS::usage="LoadSOOandECSOLS[] loads into session the LS reduced matrix
  elements of the SOO-ECSO interaction.";
LoadSOOandECSOLS[]:=(
  If[ValueQ[SOOandECSOLSTable], Return[]];
  PrintTemporary["Loading the association of LS reduced matrix elements for SOO-ECSO ..."];
  SOOandECSOLSTableFname = FileNameJoin[{moduleDir, "data", "ReducedSOOandECSOLSTable.m"}];
  If[!FileExistsQ[SOOandECSOLSTableFname],
    (PrintTemporary[">> ReducedSOOandECSOLSTable.m not found, generating ..."];
      SOOandECSOLSTable = GenerateSOOandECSOLSTable[7];
    ),
    SOOandECSOLSTable = Import[SOOandECSOLSTableFname];
  ];
);

LoadSOOandECSO::usage="LoadSOOandECSO[] loads into session the LSJ reduced matrix elements
  of spin-other-orbit and electrostatically-correlated-spin-orbit.";
LoadSOOandECSO[]:=(
  If[ValueQ[SOOandECSOTableFname], Return[]];
  PrintTemporary["Loading the association of matrix elements for spin-other-orbit and
  electrostatically-correlated-spin-orbit ..."];
  SOOandECSOTableFname = FileNameJoin[{moduleDir, "data", "SOOandECSOTable.m"}];
  If[!FileExistsQ[SOOandECSOTableFname],
    (PrintTemporary[">> SOOandECSOTable.m not found, generating ..."];
      SOOandECSOTable = GenerateSOOandECSOTable[7, "Export"->True];
    ),
    SOOandECSOTable = Import[SOOandECSOTableFname];
  ];
);

LoadT22::usage="LoadT22[] loads into session the matrix elements of T22.";
LoadT22[]:=(
  If[ValueQ[T22Table], Return[]];
  PrintTemporary["Loading the association of reduced T22 matrix elements ..."];
  T22TableFname = FileNameJoin[{moduleDir, "data", "ReducedT22Table.m"}];
  If[!FileExistsQ[T22TableFname],
    (PrintTemporary[">> ReducedT22Table.m not found, generating ..."];
      T22Table = GenerateT22Table[7];
    ),
    T22Table = Import[T22TableFname];
  ];
);

LoadSpinSpin::usage="LoadSpinSpin[] loads into session the matrix elements of spin-spin.";
LoadSpinSpin[]:=(
  If[ValueQ[SpinSpinTable], Return[]];
  PrintTemporary["Loading the association of matrix elements for spin-spin ..."];
  SpinSpinTableFname = FileNameJoin[{moduleDir, "data", "SpinSpinTable.m"}];
  If[!FileExistsQ[SpinSpinTableFname],
    (PrintTemporary[">> SpinSpinTable.m not found, generating ..."];
      SpinSpinTable = GenerateSpinSpinTable[7, "Export" -> True];
    ),
    SpinSpinTable = Import[SpinSpinTableFname];
  ];
);

LoadThreeBody::usage="LoadThreeBody[] loads into session the matrix elements of three-body
  configuration-interaction effects.";
LoadThreeBody[]:=(
  If[ValueQ[ThreeBodyTable], Return[]];
  PrintTemporary["Loading the association of matrix elements for three-body configuration-
  interaction effects ..."];
  ThreeBodyFname   = FileNameJoin[{moduleDir, "data", "ThreeBodyTable.m"}];
  ThreeBodiesFname = FileNameJoin[{moduleDir, "data", "ThreeBodyTables.m"}];
  If[!FileExistsQ[ThreeBodyFname],
    (PrintTemporary[">> ThreeBody.m not found, generating ..."];
      {ThreeBodyTable, ThreeBodyTables} = GenerateThreeBodyTablesUsingCFP[14, "Export" ->
  True];
    ),
    ThreeBodyTable = Import[ThreeBodyFname];
    ThreeBodyTables = Import[ThreeBodiesFname];
  ];
);

```

```
  (* ############################# Load Functions ############################## *)
  (* ########################################################################### *)

End[]

LoadTermLabels[];
LoadCFP[];

EndPackage[]
```