Contents

Chapter 1: Overview

The program is intended to help with evaluation of the many index expressions which arise in multi-reference perturbation theory. Essentially, it aims to evaluate one of the following three kinds of expressions, the simplest of which is

$$\langle M|\hat{X}\hat{Y}...|N\rangle, \tag{1.1}$$

here $\hat{X}$ and $\hat{Y}$ are some arbitrary operators, and $|N\rangle$ is a multireference wavefunction represented as a linear combination of determinants, $|I\rangle$;

$$|N\rangle = \sum_I c_I^N |I\rangle. \tag{1.2}$$

Written using second quantization (1.1) would be

$$\sum_{\substack{x_1x_2...\\ y_1y_2...\\ ...}} X_{x_1x_2...}Y_{y_1y_2...}... \sum_I \sum_J \langle I|a_{x_1}^\dagger a_{x_2}...a_{y_1}^\dagger a_{y_2}....|J\rangle c_I^{M\dagger} c_J^N. \tag{1.3}$$

Here, $X_{x_1,x_2...}$ and $Y_{y_1,y_2...}$ are just representations of operators $\hat{X}$ and $\hat{Y}$ in the molecular orbital basis.

It is also possible to calculate derivatives of (1.3) with respect to CI-coefficients, c_I^N;

$$\sum_{\substack{x_1x_2...\\ y_1y_2...\\ ...}} X_{x_1x_2...}Y_{y_1y_2...}... \sum_J \langle I|a_{x_1}^\dagger a_{x_2}...a_{y_1}^\dagger a_{y_2}....|J\rangle c_J^N \tag{1.4}$$

In a similar vein, it is possible to evaluate expressions of form:

$$\sum_{\substack{x_1x_2...\\ y_1y_2...\\ ...}} X_{x_1x_2...}Y_{y_1y_2...}... \sum_\Omega \sum_I \sum_J \langle I|\hat{E}_\Omega^\dagger a_{x_1}^\dagger a_{x_2}...a_{y_1}^\dagger a_{y_2}....|J\rangle c_I^{M\dagger} c_J^N$$

$$= \sum_{\substack{x_1 x_2 \ldots \\ y_1 y_2 \ldots \\ \ldots}} X_{x_1 x_2 \ldots} Y_{y_1 y_2 \ldots} \ldots \sum_I \sum_J \langle I | a_{\omega_1} a_{\omega_2} \ldots a^\dagger_{x_1} a_{x_2} \ldots a^\dagger_{y_1} a_{y_2} \ldots . | J \rangle c_I^{M\dagger} c_J^N \qquad (1.5)$$

where the $\hat{E}^\dagger_\Omega$ is a projection operator which excites electrons into the virtual space, from the orbitals used to construct the space in which the reference wavefunctions, $\{|N\rangle\}$, were originally defined.

The program is intended to be flexible; and instead of asking for specific perturbation theories or properties the user has the option of specifying algebraic expressions directly via the input file.

It is important to note that the program is intended to facilitate property calculations on wavefunctions obtained from multireference calculations. Hence it is necessary to first run a CASSCF calculation before calling the program. Furthermore, the program does not contain routines for evaluation of molecular orbital integrals; when such integrals are needed it calls to the pre-existing routines in the Bagel [1] code.

1.1 Structural outline

The program is split into three main components; an algebraic manipulator, an FCI iand linear algebra library (mainly consisting of routines for performing tensor contractions), and a task translation module for facilitating communication between .

The algebraic manipulator interprets the expressions defined in the input, and restructures these into a number of quantities which are more suited to computational evaluation. To accomplish this the algebraic manipulator makes extensive use of the commutation relations of the creation and annihilation operators, as well as the physical symmetries of the operators, in order to minimize the number of terms, and avoid the need for calculation of terms which are likely to be computationally expensive to evaluate (e.g., those with large numbers of indexes). This series of expressions is used to form an algebraic task list; a sequence of generic mathematical operations which are evaluated by other parts of the program. It is important to note that the algebraic manipulator is entirely based around symbolic manipulation, and does not handle any of the large data structures, e.g., CI-vectors, molecular orbital integral tensors, etc.. The expressions in the algebraic task list do not in any way specify what kind of data structures need to be used to store the quantities necessary for their evaluation.

The FCI and tensor libraries contain generic routines for calculating density matrices

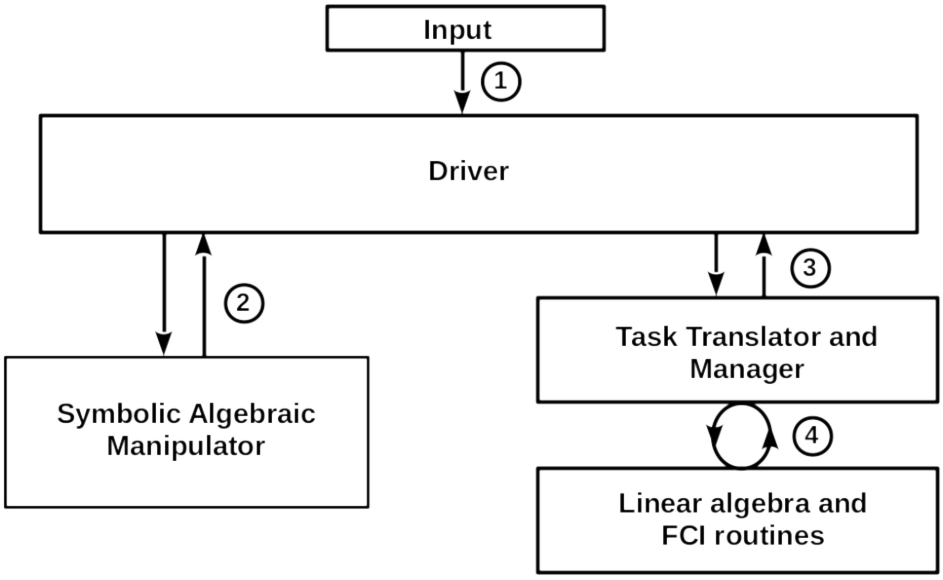

Figure 1.1: *Outline of program structure. Arrows represent the flow of data, numbers represent the initial order in which the components are used (looping may occur in the case of iterations). The intention is that each component, with the exception of the driver, functions independently of the others.*

and their derivatives, and performing tensor contractions (and other various tensor manipulations). The tensor and FCI library are completely distinct from the algebraic manipulator, and operate totally independently. The routines within the FCI library have a recursive structure, which enables calculation of density matrices (and derivatives) of arbitrary order. In a similar vein, the tensor library can operate on tensors of arbitrary order, with each dimension being a different size. At present the FCI library uses the Determinant class defined in Bagel [1]. The tensor libraries make use of the SMITH::Tensor class defined in Bagel, which in turn makes use of the TiledArray [2] libraries for parallel storage of the data. The routines in the tensor arithmetic library are built using a combination of calls to LAPACK [3], and the tensor transposition routine defined in TiledArray[1].

The final major component of the program is the task translator, which faciliates communication between the algebraic manipulator and the FCI and tensor libraries. This is necessary, as neither the arithmetic component has knowledge of the classes used in the algebraic manipulator, and vice versa. Whilst this has complicated the design slightly, it should enable greater portability, as well as making the program substantially easier to extend and upgrade.

[1]To clarify, whilst btas has some tensor contraction routines, these were found to not be suitable for the current purpose, and are not used.

In a sense there is a fourth component, a driving routine, which performs some communication between the three routines, parsing of the input, and some initialization. However, whilst this component is practically important, and not particularly small, it is rather simple and of little interest, so I will not discuss it significantly.

1.2 Specific components

The functioning of the program is best explained by example. In the following, use of italicized words indicate that word is a class defined in the program. The "highest level" class is the *equation*, which determines what the final output of the program is. For example, the user may ask the program to find the $\hat{T}^{LN}$ which satisfy

$$\sum_N \langle M|\hat{E}^\dagger_\Omega(\hat{H}-\epsilon_L)\hat{T}^{LN}|N\rangle + \langle M|\hat{E}^\dagger_\Omega\hat{H}|L\rangle = 0 \qquad \forall \quad M. \tag{1.6}$$

To accomplish this, the user will specify the equation in the input file, and the program will use the information to construct an *equation* object. This is *equation* is then used to generate a number of *expression* objects, which can be evaluated independently. For the equation specified by (1.6) these *expression* objects may correspond to quantities such as

$$\sum_N \langle M|\hat{E}^\dagger_\Omega(\hat{H}-\epsilon_L)\hat{T}^{LN}|N\rangle + \langle M|\hat{E}^\dagger_\Omega\hat{H}|L\rangle. \tag{1.7}$$

Each *equation* contains a number of such *expression* objects, along with information regarding how they relate to one another and the quantity requested by the user. In a single instance of the *expression* class corresponding to (1.7), the values of L and M would be set, whilst the range of summation over N would be defined. Note that unlike the equation the expression contains no information about what the expression should be equal to. Typically, an *equation* will contain many *expressions*, but these will only differ from one another by the values of the indices, or the constraints placed on the ranges of summations. The motivation for this is to aid in parallelization; as seperate expressions may be evaluated entirely independently from one another.

Each *expression* is further broken down into a number of *term*s. *Terms* differ from *expressions* in that any state index summation in in a *term* applies to the entire term. Hence (1.7) cannot be represented by a single term object, and instead requires two:

$$\sum_N \langle M|\hat{E}^\dagger_\Omega(\hat{H}-\epsilon_L)\hat{T}^{LN}|N\rangle \tag{1.8}$$

$$\langle M|\hat{E}_{\Omega}^{\dagger}\hat{H}|L\rangle \tag{1.9}$$

Without this separation (1.9) would be summed over N times, despite the fact N does not occur. Whilst this constraint seems needless and annoying there are justifications for them stemming from the strategies, to be elaborated upon later in this document, used to merge different *term*s together.

Each *term* is further broken down into a list of *braket*s. The *term* in (1.8) would be broken down into $2N$ *brakets*:

$$\langle M|\hat{E}_{\Omega}^{\dagger}\hat{H}\hat{T}^{LN}|N\rangle \tag{1.10}$$

$$\langle M|\hat{E}_{\Omega}^{\dagger}(-\epsilon_L)\hat{T}^{LN}|N\rangle \tag{1.11}$$

Within a *braket* the values of all the indexes are specified. A single *braket* consists of a bra and a ket, surrounding some many electron operator formed from the product of an arbitrary number of 1- and 2-electron operators. The individual *braket*s comprising a *term* are never[2] evaluated independently; the arithmetic operations to evaluate *braket*s with common bra and ket will be merged together, provided these bra and ket exist within the same term. Similarly, it is also possible to merge the compute lists associated with different *term*s together, though not the *term* objects themselves. However, this is not done by default, as typically it is useful to store and evaluate *term*s independently.

The remaining fundamental classes are the *TensOps*, which contain information regarding the second-quantized presentation of the operators, e.g., the tensor $\mathbf{H}$, with elements H_{ijkl}, where i, j, k and l are molecular orbital indices. And the *StatesInfo*, *CIVecInfo* classes, which contains information about which ci-sectors are used to represent the wavefunction, and which orbitals were used to construct these ci-sectors, respectively.

The above described components are the targets upon which the algebraic manipulation routines act to generate the algebraic task list.

[2]Excepting the situation where a *term* consists of a single *braket*.

Chapter 2: Example Input

To illustrate the capabilities of the program below is an example input for calculating the T-amplitudes in an MS-CASPT2 calculation :

```
{
  "title" : "proptool",
  "nfrozencore" : 0,
  "nfrozenvirt" : 0,
  "nact" : 4,
  "nclosed" : 3,

  "variables" : {
     "factors" : [ { "name" : "shift", "value" :  0.2 } ],
     "ranges" :  [ { "name" : "states", "range_vec" : [ 0, 1, 2 ] } ,
                   { "name" : "act" ,   "range_vec" : [ 3, 4, 5, 6 ] } ]
   },

  "operators" : [
    { "name"   : "T", "factor" : 1.0, "TimeSymm" : "none", "HermConj" : 0,
      "idxs"   : ["T0", "T1", "T2", "T3"],
      "ranges" : [["act", "vir"], ["act", "vir"], ["cor", "act"], ["cor", "act"]],
      "aops"   : [ 1, 1, 0, 0 ] ,
      "state_dependence" : 2 } ] ,

  "terms" : [
    { "name" : "X(H-E-Es)T", "type" : "orb_exc_deriv", "Target" : "T'",
        "brakets" : [
          { "ops"    : [ { "name" : "T'",   "ids" : [ "none" ] },
                         { "name" : "H-E", "ids" : [ "N" ] },
                         { "name" : "T",   "ids" : [ "L", "N" ] } ] ,
                           "bra" : "M", "ket" : "N" },
          { "ops"    : [ { "name" : "T'" , "ids" : [ "none" ] },
                         { "name" : "T" , "ids" : [ "L", "N" ] } ] ,
                           "factor" : "shift", "bra" : "M", "ket" : "N" } ] },
    { "name" : "-X(H-E)X", "type" : "orb_exc_deriv",
      "brakets" : [
        { "ops"    : [ { "name" : "X",   "ids" : [ "none" ] },
                       { "name" : "H-E", "ids" : [ "N" ] },
                       { "name" : "X",   "ids" : [ "none" ] } ] ,
                         "bra" : "M", "ket" : "M" } ] } ],

  "equations" :  [
     { "name" : "T_update", "type" : "LinearRM" , "target" : "T", "target_indexes" : [ "L", "N" ],
       "expressions" : [
         { "name"  : "residual",
           "terms" : [
             { "term" : "X(H-E-Es)T", "factor" : "one",
               "indexes" : [ { "name" : "M", "range" : "states", "sum" : false },
                             { "name" : "L", "range" : "states", "sum" : false },
                             { "name" : "N", "range" : "states", "sum" : false } ] } ] }
         { "name"  : "denominator",
           "terms" : [
             { "term" : "X(H-E)X", "factor" : "one",
               "indexes" : [ { "name" : "M", "range" : "states", "sum" : false },
                             { "name" : "L", "range" : "states", "sum" : false },
                             { "name" : "N", "range" : "states", "sum" : false } ] } ] } ]
    } ]
}
```

Important Note : As the program is still undergoing development and debugging,

the input structure is highly volatile. Accordingly, the above input is unlikely to be compatible with the current version of the program.

The input makes use of the JavaScript Object Notation [4] format, which enables specification of relatively complicated objects through use of nested arrays. Currently, the interface is rather prolix, and will be substantially reduced in complexity as the program nears full release. Nonetheless, the above form remains useful for demonstrating the program's basic functionality.

To explain the above input it is best to discuss each section individually. The first section is largely to help connection to the rest of Bagel:

```
"frozen" : false,
"nfrozenvirt" : 0,
"nact" : 4,
"nclosed" : 3,
```

These keywords just specify the number of the various kinds of orbitals used in the CASSCF. The `nfrozencore` and `nfrozenvirt` specify the number of core or virtual orbitals to be "frozen"; such orbitals are not excited either from or to.

```
"variables" : {
   "factors" : [ { "name" : "shift", "value" :  0.2 } ],
   "ranges" :  [ { "name" : "states", "range_vec" : [ 0, 1, 2 ] } ,
                 { "name" : "act" ,   "range_vec" : [ 3, 4, 5, 6 ] } ,
                 { "name" : "virt" ,  "range_vec" : [ 7, 8, 9, 10, 11, 12, 13, 14, 15 ] } ]
 },
```

This section is used to define ranges and variables which may be used in later definitions. For example, `shift` is specified here to be 0.2, and is the level shift[1] and will be used later on to specify some of the terms we wish to calculate.

```
"operators" : [
  { "name"     : "T",
    "factor" : 1.0,
    "TimeSymm" : "none",
    "HermConj" : true,
    "idxs"     : ["T0", "T1", "T2", "T3"],
    "ranges"   : [["act", "vir"], ["act", "vir"], ["cor", "act"], ["cor", "act"]],
    "aops"     : [ 1, 1, 0, 0 ] ,
    "state_dependence" : 2 } ] ,
```

This section enables the user to define their own operator, and specify some of its symmetries. For example, here `TimeSymm` and `HermConj` determine whether this operator has time reversal symmetry and whether or not it is Hermitian. The symmetry functions are currently hard coded, but there is the facility for user defined ones to be added with relative ease. The `aops` specifies the ordering of the creation and annihilation operators; 1 is creation, 0 is annihilation. Here it is specified that the T tensor

[1] Units are Hartrees, currently this cannot be changed.

has four indices. The `ranges` array specifies the orbitals on which the operator acts. Each subarray in the `ranges` section specifies a different sub range used to divide up the T tensor into blocks. For example, in one possible block of the T operator the first index would be active, the second index could be virtual, whilst the third and fourth blocks would be core. Note that in the aforementioned `variables` section the user is able to specify the exact orbital indexes to which each range corresponds, enabling tight control over the block decomposition of the tensor, as well as the ranges on which an operator acts. The `state dependence` section determines how many state indexes the operator has.

```
"terms" : [
  { "name" : "residual", "type" : "orb_exc_deriv", "Target" : "T'",
      "brakets" : [
        { "ops"    : [ { "name" : "T'",   "ids" : [ "none" ], "transform" : "inverse" },
                       { "name" : "H-E", "ids" : [ "L" ] },
                       { "name" : "T",    "ids" : [ "L", "N" ] } ] ,
                         "bra" : "M", "ket" : "N" },

        { "ops"    : [ { "name" : "T'" , "ids" : [ "none" ], "transform" : "inverse" },
                       { "name" : "T" , "ids" : [ "L", "N" ] } ] ,
                         "factor" : "shift", "bra" : "M", "ket" : "N" } ] },

 { "name" : "-X(H-E)X", "type" : "value", "target" : "T'",
   "brakets" : [
     { "ops"    : [ { "name" : "X",   "ids" : [ "none" ] },
                    { "name" : "H-E", "ids" : [ "L" ] },
                    { "name" : "X",   "ids" : [ "none" ] } ] ,
                      "bra" : "N", "ket" : "N" } ] } ],
```

This `terms` section defines quantities to facilitate construct the expressions the user wishes the program to evaluate. The `name` is used by the program to identify the term internally, and in printing the results. It must be unique. The `type : orb_exc_deriv` specifies the kind of term; in this case the partial derivative of the terms is taken with respect to the elements of the tensor specified by `Target : T'`.

It is worth considering the `brakets` array seperately:

```
"brakets" : [
  { "ops"    : [ { "name" : "T'",   "ids" : [ "none" ] },
                 { "name" : "H-E", "ids" : [ "L" ] },
                 { "name" : "T",    "ids" : [ "L", "N" ] } ] ,
                   "bra" : "M",
                   "ket" : "N"                          },
  { "ops"    : [ { "name" : "T'" , "ids" : [ "none" ] },
                 { "name" : "T" , "ids" : [ "L", "N" ] } ] ,
                   "factor" : "shift",
                   "bra" : "M",
                   "ket" : "N" } ]
```

The above represents the mathematical term:

$$\langle\Psi_M|\hat{T'}_{t_0't_1't_2't_3'}(\hat{H}-E_L^0)\hat{T}^{LN}_{t_0t_1t_2t_3}|\Psi_N\rangle + E_{shift}\langle\Psi_M|\hat{T'}_{t_0't_1't_2't_3'}\hat{T}_{ijkl}\hat{T}^{LN}_{t_0t_1t_2t_3}|\Psi_N\rangle \qquad (2.1)$$

Note that there are no summations specified in the `brakets` array; the summations to be performed are determined from the `type` specified in `term`, and the final `equation` input. The `ops` specifies the operators used in the `braket`. Most relevant operators

and combinations of operators with scalar arrays, e.g., `H-E`, are predefined in the code. The `ids` specifies the names of the state indexes for these operators. This naming aids the program in determinating the correct ranges of the various summations, and also to enables indexes on different operators to be tied to one another. For example the first index, L, on the `T` tensor is the same in both brakets. The `none` indicates that the operator is state independent. The `factor` defines the factor associated by which the braket is multiplied, and this defaults to 1.0. The `Bra` and `Ket` specify the name of the state index of the Bra and Ket.

```
"equations" : [
  { "name" : "T_update", "type" : "LinearRM" , "target" : "T", "target_indexes" : [ "L", "N" ],
   "expressions" : [
      { "name"  : "residual",
        "terms" : [
          { "term" : "residual", "factor" : "mone",
            "indexes" : [ { "name" : "M", "range" : "states", "sum" : false },
                          { "name" : "L", "range" : "states", "sum" : false },
                          { "name" : "N", "range" : "states", "sum" : true } ] },
      { "name"  : "denominator",
        "terms" : [
          { "term" : "X(H-E)X", "factor" : "one",
            "indexes" : [ { "name" : "M", "range" : "states", "sum" : false },
                          { "name" : "L", "range" : "states", "sum" : false },
                          { "name" : "N", "range" : "states", "sum" : true } ] } ] } ]
  } ]
 } ]
```

The `equations` section is used to specify what is actually calculated. The above corresponds to finding the `T` which minimizes the following

$$\frac{\langle \Psi_M | \hat{X}_\Omega (\hat{H} - E_L^0 - E_{shift}) \hat{T}^{LN}_{\Omega'} | \Psi_N \rangle}{\langle \Psi_M | \hat{X}^\dagger_\Omega (\hat{H} - E_L^0) \hat{X}_{\Omega'} | \Psi_M \rangle} \tag{2.2}$$

It should be noted that for reasons of simplicity, the various transformations and orthogonalizations associated with XMS-CASPT2 and the internally contracted wavefunctions are not included here. Ultimately, XMS-CASPT2 will be assumed to be the default, and the corresponding transformations will be performed automatically, however this will not substantially impact the input.

The `equations` section first specifies the `type` of equation to be solved, in this case `LinearRM` (linear residual minimization). The `type` dictates what other keywords must appear. Due to the choice of `LinearRM`, it is necessary to specify a `target` tensor for which we are solving, as well as a `residual` and `denominator`. These are two `expressions` which are defined by a list of `terms`, which were themselves defined by the user in the previous section. It is at this stage that the user specifies the summations over the state indexes which appear in the terms.

An expression may be composed of multiple different terms even though, in principle, these could be merged. However, breaking down the expression into smaller, independent parts facilitates identification of quantities which may be reused, and eases control of summations over different state indexes.

Chapter 3: Algebraic task generation

The introduction section illustrated how the input is broken down to yield a number of smaller and more manageable *terms*, generally corresponding to a single braket. Each of these *terms* is fed into the algebraic manipulator. This component of the program generates a set of instructions for evaluating each of these *terms*, which the task translator then converts into calls to the FCI and linear algebra routines. Construction of this task list is split up into two major parts; the tensor contraction task list generator, and the "gamma generator". Before these individual components are discussed in detail it is necessary to discuss the qualities the final task list should possess, and what distinguishes and motivates the current approach.

3.1 Core functionality

At its core, the basic operation of the program is to evaluate terms such as

$$\sum_{ijkl}\sum_{mnop}\sum_{IJ}\langle I|i^\dagger j^\dagger klm^\dagger n^\dagger op|J\rangle c_I^M c_J^N Y_{ijkl}^{ML} Z_{mnop}^{NP}, \tag{3.1}$$

where I and J each denote a Slater determinant, and L, M, N, P are state indexes. $\mathbf{Y}^{ML}$ and $\mathbf{Z}^{NP}$ are representations of operators and $\hat{Y}$ and $\hat{Z}$ in the molecular orbital basis. For the time being the state dependence of the operator representations will be ignored, but this has important consequences for exploitation of symmetry, and will be discussed at length later.

The computational cost of directly evaluating terms such as (3.1) can be prove prohibitive, particularly if the ranges of the indexes i, j, k, l, m, n, o, and p are large (e.g., if they range over virtual and core orbitals). The algebraic manipulator uses various commutation and symmetry relations to rearrange expressions, such as (3.1), into a

sum of terms in which the orbital indexes only range over the active orbitals, e.g.,

$$\sum_{stuvwxyz} \gamma_{stuvwxyz} A_{stuvwxyz} + \sum_{stuvxy} \gamma_{stuvwx} A_{stuvwx}$$

$$+ \sum_{stuv} \sum_{mnop} \gamma_{stuv} A_{stuv}, + \sum_{st} \gamma_{st} A_{st} + A. \tag{3.2}$$

Here the tensors $A_{ijkl\ldots}$ are formed by performing contractions between and re-orderings of the indexes of tensors on Y and Z as determined from the commutation relations of the creation and annihilation operations, e.g.,

$$A_{stwx} = \sum_{r}^{R} \hat{\wp}_r \sum_{\{u,y\}}^{c1} \delta_{uy} \sum_{\{v,z\}}^{c2} \delta_{vz} Y_{ijkl} Z_{mnop},$$

where the $c1$ and $c2$ are sets of pairs of indexes to be contracted, e.g.,

$$c2 = \{\{i,k\}\},\{j,k\}\},\{l,m\}\}.....\} \tag{3.3}$$

and $\hat{\wp}_r$ which transforms the ordered set of indexes $\{u,v,s,t,w,x,y,z\}$ into some permutation, $r \in R$, of the ordered set of indexes $\{i,j,k,l,m,n,o,p\}$, whilst also acting to multiply the result of the summation by an appropriate factor.

The γ and Γ are defined by[1]:

$$\gamma_{swtzuyvx} = \sum_{IJ} \langle I|s^\dagger w t^\dagger z u^\dagger y v^\dagger x|J\rangle c_I c_J^*, \tag{3.4}$$

$$\Gamma^I_{swtzuyvx} = \sum_{stuvwxyz} \sum_{J} \langle I|s^\dagger w t^\dagger z u^\dagger y v^\dagger x|J\rangle c_J^*, \tag{3.5}$$

A key feature of the program is that it is able to use decompose the $\mathbf{A}$ and γ objects up into smaller sub-blocks, and use symmetries existing between these blocks, as well as number of other physical constraints on the possibility of creation and annhilation operations, to inform the generation of the task list. Not only does this facilitate parallelization of the resulting task list, as operations concerning different blocks can be assigned to different nodes, but it may also substantially reduce the number of distinct operations which need to be performed. This kind of decomposition is particularly advantageous for relativistic systems, where the seperate treatment of α and β electrons can result in much larger tensors. For example, if the range of each index on an eight index tensor is doubled, then the number of elements increases by a factor of 256.

[1]It is worth noting that these are not normal ordered. The significance of this will be explained in due course.

The challenge faced by the algebraic manipulator is that whilst there are several different series of form (3.2) which are equal to (3.1), these different series often differ wildly in the amount of computational effort their evaluation requires. Hence the algebraic manipulator must accomplish two things. First, it must decide on the series of form (3.2) that is least computationally expensive to evaluate. Second, it must produce a sequence of instructions for evaluating the terms in this series. This second point may seem trivial, but is complicated substantially by the decomposition of the tensors, and the need to take advantage of symmetry in an effective manner[2].

The algebraic manipulator accomplishes this by performing the operations as outlined in figure 3.1. Both the γ tensors, as well as the the tensors contracted to produce the $\mathbf{A}$ tensors, are split up into blocks. As calculation and storage of the γ and $\mathbf{\Gamma}^I$ is typically more computationally demanding than obtaining the $\mathbf{A}$, minimization of the number of distinct γ and $\mathbf{\Gamma}^I$ blocks is prioritized, and each such block corresponds to a different task list, which are typically evaluated seperately.

Chapter 5 focuses on the machinery (called the "gamma generator") for chosing the the series of form (3.2). Chapter 4 focuses on evaluation of terms contained therein. The reason for this seemingly backwards order is that the decisions made in the former and wholly dependent on the factors governing the efficiency of the task lists generated in the latter.

[2]Whilst the algebraic manipulator can also generate task lists for derivative expressions, the task lists remain confined to evaluation of linear combinations of terms with one of the following forms :

$$\sum_{\substack{ijkl\\mnop}} \gamma_{ijklmnop} A_{ijklmnop}, \qquad \sum_{ijkl} \gamma_{imjo} A_{imjoknlp}, \qquad \text{or} \qquad \sum_{\substack{ijkl\\mnop}} \Gamma^I_{ijklmnop} A_{ijklmnop}. \tag{3.6}$$

and the distinctions between these terms do not impact this preliminary discussion.

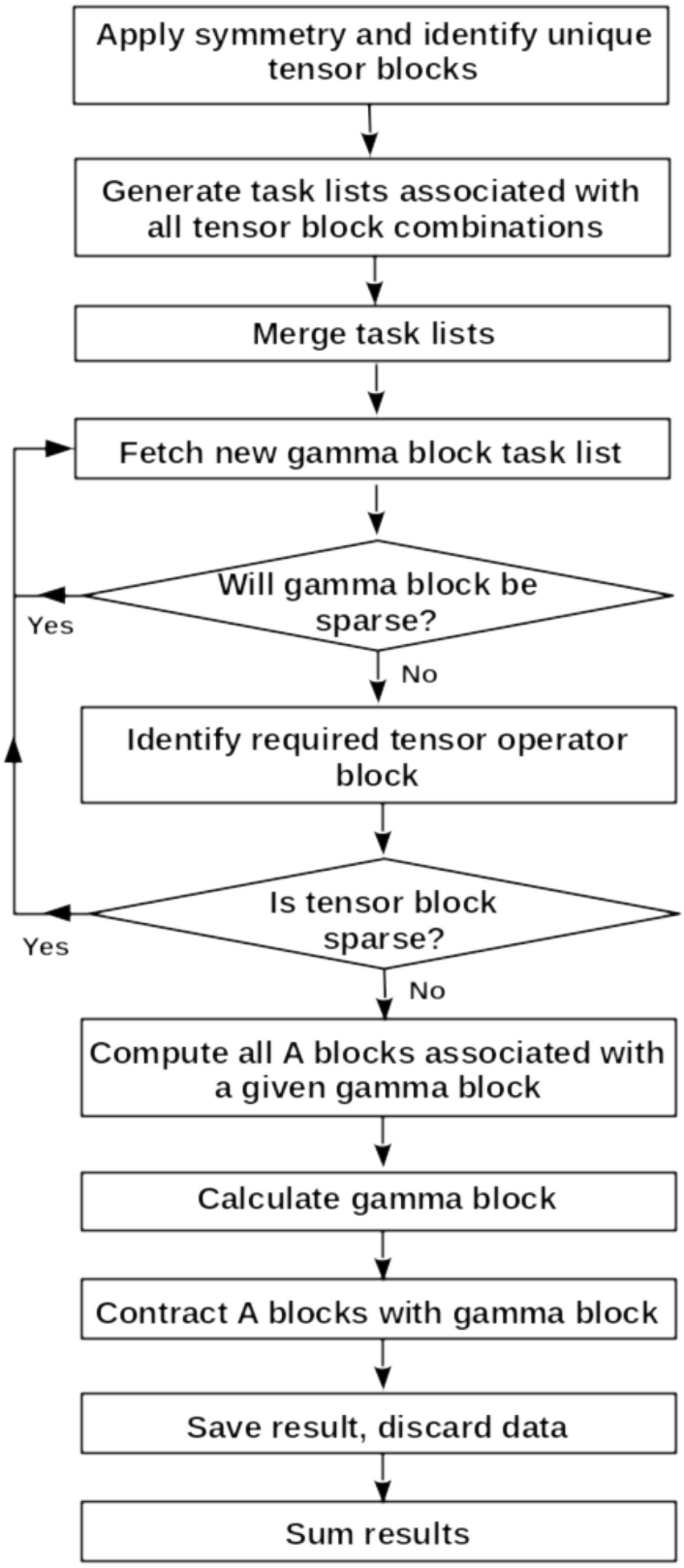

Figure 3.1: *Outline of the algebraic manipulator.*

Chapter 4: Tensor decomposition and contraction

4.1 Tensor Decomposition

In most cases the ranges of the molecular orbital indexes can be split into a number of different subranges, which enable us to define each of the operator tensors as the sum of a number of different blocks, each one associated with a different combination of these subranges. Decomposing the tensors in this manner reduces the maximum size of the data structure the computer has to deal with at any one time. Block decomposition also facilitates the implementation of symmetry, as many blocks are either equivalent to one another or connected via some simple transformation. This is particularly relevant in the relativistic case, where Kramers symmetry can potentially be used to ensure that the size of the blocks we deal with is no larger than those encountered in the non-relativistic case.

Before proceeding further it is worth mentioning that the design of the basic tensor operator object is heavily informed by the functionality required of the algebraic manipulation routines. For these manipulation routines to make sense it is necessary to be very precise about the definition of the tensors. Consequently, some of the exposition will seem tedious, and the design choices unnatural, at least until the algorithms in the main algebraic manipulator are discussed.

The design of this object is easiest explained by example. Consider an operator; $\hat{Q}$, which may be written in second quantized form as

$$\hat{Q} = \sum_{i}^{R^i} \sum_{j}^{R^j} \sum_{k}^{R^k} \sum_{l}^{R^l} \hat{a}_i^\dagger \hat{a}_j^\dagger \hat{a}_k \hat{a}_l Q_{ijkl}. \tag{4.1}$$

The coefficients, Q_{ijkl}, can be thought of as elements of a tensor $\mathbf{Q}$, which is the

representation of the operator $\hat{Q}$ in the molecular orbital basis. I shall refer to such tensors as operator tensors. The R^i is just the set of values over which index i runs (note the range is index specific). This R^i can be broken down into N_{block} disjoint subsets or sub ranges, i.e.,

$$R^i = r^i_1 \cup r^i_2 \cup r^i_3 \cup ... = \bigcup_{\mu}^{N_{block}} r^i_\mu \tag{4.2}$$

A range block, $\mathbf{B}$, specifies the combinations of indexes which lie within some specified ranges. For example, the (ordered) set of sub ranges, $\{r_\mu,\ r_\nu,\ r_\xi,\ r_\chi\}$, defines range block $B^{\mu\nu\xi\chi}$, which specifies the set of ranges.

$$B^{\mu\nu\xi\chi} := \{\{i,j,k,l\} \mid i \in\ r^i_\mu,\ j \in r^j_\nu,\ k \in r^k_\xi,\ l \in r^l_\chi\}. \tag{4.3}$$

The components, Q^b_{ijkl}, of a tensor block, $\mathbf{Q}^b$ are those blocks which satisfy

$$Q_{ijkl} \in \mathbf{Q}^b \quad \text{iff} \quad \{i,j,k,l\} \in b \tag{4.4}$$

using this notation we can rewrite (4.1) as

$$\hat{Q} = \sum_{b}^{\{B\}} \sum_{\{i,j,k,l\}}^{b} \hat{a}^\dagger_i \hat{a}^\dagger_j \hat{a}_k \hat{a}_l Q_{ijkl}. \tag{4.5}$$

For each operator $\hat{Q}$, we store a single *TensOp* object, which corresponds to a molecular orbital tensor $\mathbf{Q}$. This object contains a list of blocks, Q^b, information about which is stored in *CtrTensorPart* objects. To facilitate the exploitation of symmetry and sparsity, the task list generated by the program only contains instructions referring to tensor blocks, never the full operators themselves.

4.2 Block symmetry and sparsity

Some blocks of the tensor can be obtained by transforming other blocks. How the program exploits this is best explained through a series of examples.

Consider two tensor blocks Q^{b1} and Q^{b2} between which there is the relation

$$Q^{b1}_{ijkl} = Q^{b2}_{ijlk}\rho, \tag{4.6}$$

where $b1 = B^{\xi\xi\chi\xi}$, $b2 := B^{\xi\xi\xi\chi}$, and ρ is some constant factor. This is a special case

where the ranges r^ξ and r^χ have the same length:

$$\text{card}(r^\xi) = \text{card}(r^\chi). \tag{4.7}$$

This case regularly occurs when dealing with spin symmetry. Here an element of block Q^{b1} is equal to an element of block Q^{b2} with the last two indexes interchanged. Therefore, we do not need to store both blocks; operations on Q^{b2} are equivalent to operations on Q^{b1} preceded by a transformation.

Now consider a third, unrelated tensor block R^{b3}, where $b3 := \{\xi, \xi, \chi, \xi, \nu, \nu\}$. This tensor block is large, and r^ν has far greater extent than r^χ, i.e.,

$$\text{card}(r^\nu) \gg \text{card}(r^\chi) \tag{4.8}$$

For example, consider calculation of elements of a tensor $\mathbf{X}$ from contractions between tensors $\mathbf{Q}$ and $\mathbf{R}$;

$$X_{mn} = \sum_{ijkl} Q^{b1}_{ijkl} R_{ijklmn} + \sum_{ijkl} R^{b3}_{ijlkmn} Q^{b2}_{ijkl}. \tag{4.9}$$

Note that the l and k indexes on R have been interchanged.

A straight forward way to calculate this is by the following two tasks:

$$\text{task A1}: \qquad X_{mn} = \sum_{ijkl} Q^{b1}_{ijkl} R^{b3}_{ijklmn},$$

$$\text{task A2}: \qquad X_{mn} += \sum_{ijkl} Q^{b2}_{ijlk} R^{b3}_{ijklmn}.$$

where the indexes on Q^{b2} have been swapped to avoid the swapping the indexes on R. Applying symmetry we end up with three tasks, but don't need to store Q^{b2}, i.e.,

$$\text{task B1}: \qquad X_{mn} = \sum_{ijkl} Q^{b1}_{ijkl} R^{b3}_{ijklmn}$$

$$\text{task B2}: \qquad Q^{b2}_{ijkl} = -(\text{swap}(k,l)[Q^{b1}_{ijkl}])$$

$$\text{task B3}: \qquad X_{mn} += \sum_{ijkl} Q^{b2}_{ijkl} R^{b3}_{ijlkmn}$$

However, this saves on storage, not computation time. A better approach would be

$$\text{task D1}: \qquad A^{b1}_{ijkl} = Q^{b1}_{ijkl} + \rho(\text{swap}(k,l)[Q^{b1}_{ijkl}])$$

$$\text{task D2}: \qquad X_{mn} = \sum_{ijkl} A^{b1}_{ijkl} R^{b3}_{ijklmn}.$$

This could be improved even further to

$$\text{task E1}: \qquad X_{mn} = \sum_{ijkl} (1+\rho) Q^{b1}_{ijkl} R^{b3}_{ijklmn}.$$

Getting to this final task list, containing just task E1, can be extremely beneficial, not only because we can often replace several transposes and summations with a single scalar multiplication, but also because if $\rho = -1$, then no computation needs to be performed at all.

To go from tasks $A1$ and $A2$ to tasks $D1$ and $D2$ all that is required is that whenever the program wants the data for tensor block Q^{b2}, it instead fetches the data for Q^{b2} and performs the appropriate transformation.

Ensuring that tasks $D1$ and $D2$ are replaced by task $E1$ is a bit more involved, as it requires that symmetry is accounted for during the construction of the task list. The full details of this will wait until the next section, but the basic ideas governing how symmetry is handled can be discussed now.

The set, B, of all the blocks of a tensor can be generated by applying the appropriate transformations to some subset of these blocks;

$$S : \tilde{B} \to B \qquad \tilde{B} \subset B$$

$$\tilde{b} \mapsto b \qquad \tilde{b} \in \tilde{B} \qquad b \in B$$

$$b = t\tilde{b} \tag{4.10}$$

Only the data corresponding to the minimal set, $\tilde{b}$, of blocks is stored. Each *TensOp* object contains a list of *rangeblock* objects, which define this mapping between the block (in the full set, B), and the master range block (from the minimal set, $\tilde{B}$). Loosely speaking, the algebraic manipulation routines just replace all appearances of the original blocks with master blocks, plus some transformation.

It is important that the minimal set of blocks is consistent, so that if the same operator appears at multiple places in the same expression, only subject to different transformations, no unnecessary blocks are stored. For example, λ, should correspond to the same minimal set as λ^T. Accordingly, the transformations routines are written such that multiple transformations can be applied to the same block; in this case there is a transpose operation, followed by the transformation into the minimal set associated with λ. Note that actual data corresponding to the tensor representations (i.e., the large, many index arrays) is not being transformed, only the algebraic objects which represent this data using the task list construction.

4.3 MultiTensOp

Another important matter is combinations of multiple operators, e.g.,

$$\langle I|\hat{\lambda}\hat{H}\hat{T}|J\rangle. \tag{4.11}$$

The symmetry operations associated with these operators are combined; the increase in the saving associated with block symmetry is multiplicative, and so increases steeply with the number of concatenated operators. Furthermore, there is a canonical operator ordering, so as to avoid storing multiple terms which differ only by some transpose, e.g.,

$$\lambda^{b1}H^{b1}T^{b1} = T^{b1}H^{b1}\lambda^{b1}. \tag{4.12}$$

4.4 CtrTensOp and CtrMultiTensOp

The task list generator often produces terms which are formed from contractions between indexes either on the same,

$$H_{il}^{(b1,jk)} = \sum_{jk} H_{ijkl}^{b1}\delta jk,$$

or different tensors,

$$[HT]_{ilxy}^{(b1,b2,jz,kw)} = \sum_{jz}\sum_{kw} H_{ijkl}^{b1}T_{wxyz}^{b2}\delta_{jz}\delta_{kw}. \tag{4.13}$$

These correspond to the CtrTensOp object; note that once the contraction between H and T has occurred, they are usually inextricable (short of performing some potentially expensive decomposition), and so are treated just like a normal tensor block. In terms

involving two tensors which are not contracted all operations on the product tensor should be decomposed into operations of these two individual tensors[1] (note that decomposing the operation does not, in the cases relevant here, carry the same cost as decomposing the tensors).

It is vital that all transformations are taken account of before the task list of required contractions is generated. Hence, the RHS in (4.13) above may well be replaced with something like this in the algebraic task list

$$[HT]^{(b1,b2,jz,kw)}_{ilxy} \mapsto \sum_{ix}\sum_{ly} \rho(H^{\tilde{b1}}_{jilk} T^{\tilde{b2}}_{yzwx} \delta_{ix}\delta_{ly}). \tag{4.14}$$

Here the original blocks, $b1$ and $b2$, have been replaced with blocks, $\tilde{b1}$ and $\tilde{b2}$, from the minimal set, the indexes reshuffled, and a factor ρ associated with the various transformations has appeared.

4.5 Contraction procedure

In most contexts, the tensor contractions are performed using a recursive sequence of binary contractions. For example,

$$A_{kwzm} = [\tilde{B}\tilde{C}\tilde{D}]^{(jy,lx,no,pi)}_{kwzm} = \sum_{jy}\sum_{no}\sum_{lx}\sum_{ip} B_{ijkl} C_{mnop} D_{wxyz} \delta_{jy}\delta_{no}\delta_{lx}\delta_{ip} \tag{4.15}$$

The far LHS is just the output tensor. The term $[\tilde{B}\tilde{C}\tilde{D}]^{(jy,lx,no,pi)}_{kwzm}$, indicates that the tensor blocks $\tilde{B}$, $\tilde{C}$, and $\tilde{D}$ have been contracted over the indexes in the superscript, whilst the indexes in the subscript remain uncontracted. The tilde indicates that these are all master blocks (a choice made for the sake of simplifying the example). The tensor A_{kwzm} would be obtained through the following sequence of operations:

$$\text{Task1} \;:\; [\tilde{C}]^{(no)}_{mp} = \sum_{no} \tilde{C}_{mnop}\delta_{no}. \tag{4.16}$$

$$\text{Task2} \;:\; [\tilde{B}\tilde{C}]^{(ip,no)}_{jklm} = \sum_{pi} \tilde{B}_{ijkl}[\tilde{C}]^{(no)}_{mp}\delta_{ip}. \tag{4.17}$$

$$\text{Task3} \;:\; [\tilde{B}\tilde{C}\tilde{D}]^{(ip,no,jy)}_{klmwxq} = \sum_{jy} [\tilde{B}\tilde{C}]^{(ip,no)}_{jklm} \tilde{D}_{wxqy}\delta_{jy}. \tag{4.18}$$

$$\text{Task4} \;:\; [\tilde{B}\tilde{C}\tilde{D}]^{(ip,no,jy,lx)}_{kmwq} = \sum_{lx} [\tilde{B}\tilde{C}\tilde{D}]^{(ip,no,jy)}_{klmwxq}\delta_{lx}. \tag{4.19}$$

[1] this is currently not done everywhere in the code

A task list for performing the necessary contractions, transpositions, multiplications, etc., is generated from the information stored in the CtrMultiTensOp corresponding to $[BCD]^{(jy,lx,no,pi)}_{kwzm}$. The design of the algorithm for generating the task list is motivated by five basic principles:

- Avoid storing or multiplying large tensor blocks. The size of a tensor block is assumed to be directly correlated with its order.
- Identify if an intermediate result in one task list may be reused in another.
- Task lists should be comparable; if two different task lists will produce results which will cancel, or which can be connected by symmetry, this should be recognized and taken advantage of.
- Every step of the task list contains information about what inputs it needs, and what outputs it produces.
- The algebraic task list is purely symbolic, and the tasks themselves should not be dependent on how the tensor data is stored.

These principals lead to the following, more specific rules:

Rule 1 : Contractions between indexes on the same tensor are performed first; this will help reduce the rank of the largest tensor to be stored to a minimum. For example, if the first two tasks, (4.16) and (4.17), were reversed;

$$\text{Task1'} \quad : \quad [BC]^{(ip)}_{jklmno} = \sum_{pi} B_{ijkl}[C]^{(no)}_{mp}\delta_{ip}.$$

$$\text{Task2'} \quad : \quad [BC]^{(ip,no)}_{jklm} = \sum_{pi} [BC]^{(ip)}_{jklmno}\delta_{no}.$$

the most expensive operation will be contraction of two 4-index tensors, whereas in original formulation, the most expensive is contraction of a 4-index tensor with a 2-index tensor.

Rule 2 : The contraction list is generated in accordance with an ordering principle, such that the order in which the contractions are performed is always the same, e.g., the task lists for $[\tilde{B}\tilde{C}\tilde{D}]^{(jy,lx,no,pi)}_{kwzm}$ and $[\tilde{B}\tilde{C}\tilde{D}]^{(kw,lz,no,pi)}_{jxym}$ will always perform the contraction over (n,o) first, followed by the contraction over (i,p). This has two advantages; first the "intermediate" tensor (intermediate in the sense it is not the result we seek), $[\tilde{B}\tilde{C}]^{(ip,no)}_{jklm}$ can be reused, and the first two tasks need not be repeated. Furthermore,

this consistent ordering facilitates merging of contraction lists.

Rule 3 : A canonical ordering is defined for the indexes and tensors, and any transpositions are defined relative to this canonical order. For example, tensor B always appears to the left of C, and index i always appears to the left of j. This is also to facilitate merging of task lists.

Rule 4 : Each "intermediate" tensor has a "use count", which defines the number of remaining tasks which use this tensor (the same intermediate result may be used in many different task lists). When there is no more use for the tensor, it should be deleted[2].

Rule 5 : The contraction task list is always expressed in terms of operations on the master blocks. The transformations connecting the master blocks to the general blocks have two components; a multiplicative factor[3], and an index transposition. Rather than transposing the indexes of the tensor blocks, and then executing the task list, the indexes over which the contractions are performed are altered to reflect the index transpositions, and the actual index transpositions are performed as the final stage of the task list, i.e., after all contractions have been performed. To illustrate why this approach is taken please consider the following example, suppose we would like to calculate

$$A_{kwzm} = [BCD]_{inwz}^{((b1,b2,b3),(jy,lx,mo,kp))} = \sum_{jy}\sum_{no}\sum_{lx}\sum_{ip} B_{ijkl}C_{mnop}D_{wxyz}\delta_{jy}\delta_{no}\delta_{lx}\delta_{ip} \tag{4.20}$$

whilst taking advantage of the following symmetry operations:

$$B_{ijkl}^{b1} = B_{klij}^{\tilde{b1}} \quad (\ 0123 \rightarrow 2301\ )$$

$$C_{mnop}^{b2} = C_{nmop}^{\tilde{b2}} \quad (\ 0123 \rightarrow 1023\ )$$

$$D_{wxyz}^{b3} = D_{zxyw}^{\tilde{b3}} \quad (\ 0123 \rightarrow 3120\ )$$

To simplify the notation, I shall now use $\tilde{D}$ to denote $D^{\tilde{b3}}$. The following tasks would be performed,

$$\text{Task1} \ : \quad [\tilde{C}]_{mp}^{(no)} = \sum_{no} \tilde{C}_{mnop}\delta_{no} \qquad \delta_{mo} \rightarrow \delta_{no}\ .$$

[2]The count is implemented, but the deletion is not.

[3]Or possibly a complex conjugation, which reduces to a factor when using real arithmetic.

$$\text{Task2} \;:\quad [\tilde{B}\tilde{C}]^{(ip,no)}_{jklm} = \sum_{pi} \tilde{B}_{ijkl}[\tilde{C}]^{(no)}_{mp}\delta_{ip} \qquad \delta_{kp} \rightarrow \delta_{ip}\ .$$

$$\text{Task3} \;:\quad [\tilde{B}\tilde{C}\tilde{D}]^{(ip,ly,no)}_{klmwxz} = \sum_{ly} [\tilde{B}\tilde{C}]^{(ip,no)}_{jklm}\tilde{D}_{wxyz}\delta_{ly}. \qquad \delta_{lx} \rightarrow \delta_{ly}\ .$$

$$\text{Task4} \;:\quad [\tilde{B}\tilde{C}\tilde{D}]^{(ip,ly,mx,no)}_{klwz} = \sum_{mx} [\tilde{B}\tilde{C}\tilde{D}]^{(ip,no,ly)}_{jkmwxz}\delta_{mx}. \qquad \delta_{jy} \rightarrow \delta_{mx}\ .$$

$$\text{Task5} \;:\quad [BCD]^{(jy,lx,mo,kp)}_{inzw} = swap(2,3)[\tilde{B}\tilde{C}\tilde{D}]^{(ip,jy,mx,no)}_{klwz}$$

where changes to contractions versus those that would be performed on the untransformed blocks are noted on the right. By waiting until the last possible moment to apply the index transpositions, only one index transposition is required; the swapping of the indexes 2 and 3. Furthermore, it enables the program to identify that the $[\tilde{B}\tilde{C}]^{(ip,no)}_{jklm}$, obtained in Task 2 (4.17) of the previous example task list, can be reused.

Rule 6 : All factors arising from symmetry transformations are applied after all necessary contractions have been performed. This keeps the size of the array which needs to be scaled to a minimum, and also facilitates the tensor reuse and task list comparison described above.

Chapter 5: Task list generation

Consider a term

$$\sum_{ijkl}\sum_{mnop}\sum_{IJ}\langle I|i^\dagger j^\dagger klm^\dagger n^\dagger op|J\rangle c_I^M c_J^N Y_{ijkl}^{ML} Z_{mnop}^{NP}, \tag{5.1}$$

where I and J each denote a Slater determinant, and L, M, N, P are state indexes. $\mathbf{Y}^{ML}$ and $\mathbf{Z}^{NP}$ are representations of operators and $\hat{Y}$ and $\hat{Z}$ in the molecular orbital basis. For the time being the state dependence of the operator representations will be ignored, but has important consequences for exploitation of symmetry, and will be discussed at length later.

The procedure most commonly employed by code generators is to use Wick's theorem to rearrange expressions, such as (5.1), into a series of normal ordered terms, e.g.,

$$\begin{aligned}
&= \sum_{stuvwxyz}\sum_{IJ}\langle I|s^\dagger t^\dagger u^\dagger v^\dagger wzyx|J\rangle c_I c_J^* A_{stuvwxyz},\\
&+ \sum_{stuwxy}\sum_{IJ}\langle I|s^\dagger t^\dagger u^\dagger wzy|J\rangle c_I c_J^* A_{stuwxy},\\
&+ \sum_{stwx}\sum_{mnop}\sum_{IJ}\langle I|s^\dagger t^\dagger wz|J\rangle c_I c_J^* A_{stwx},\\
&+ \sum_{sw}\sum_{mnop}\sum_{IJ}\langle I|s^\dagger w|J\rangle c_I c_J^* A_{sw}\\
&+ \sum_{sw}\sum_{mnop} A.
\end{aligned}$$

Where the tensors $A_{ijkl\ldots}$ are formed by performing the contractions between and reorderings of the indexes of tensors on Y and Z as determined from the commutation relations of the creation and annihilation operations, e.g.,

$$A_{stwx} = \sum_{r}^{R} \hat{\wp}_r \sum_{\{u,y\}}^{c1} \delta_{uy} \sum_{\{v,z\}}^{c2} \delta_{vz} Y_{ijkl} Z_{mnop},$$

where the $c1$ and $c2$ are sets of pairs of indexes to be contracted, e.g.,

$$c2 = \{\{i,k\}\},\{j,k\}\},\{l,m\}\}.....\} \tag{5.2}$$

and $\hat{\wp}_r$ which transforms the ordered set of indexes $\{u,v,s,t,w,x,y,z\}$ into some permutation, $r \in R$, of the ordered set of indexes $\{i,j,k,l,m,n,o,p\}$, whilst also acting to multiply the result of the summation by an appropriate factor.

A major advantage of this is that it enables constraints to be on the values over which the molecular orbital indexes range; all indexes on annihilation operators must correspond to orbitals occupied in $|J\rangle$, and creation operators must correspond to orbitals occupied in $|I\rangle$. This is a consequence of how determinants are constructed in active space based methods. A generic determinant can be written corresponding to the case where the, molecular orbitals, $\{\psi_{x_i}\}$, with indexes $X = \{x_0,...,x_n\}$ are occupied can be written

$$\Psi_K = \sum_{\zeta}^{\zeta \in S_X} \epsilon_\zeta \bigotimes_{i=0}^{n} \psi_{x_i}, \tag{5.3}$$

where X is a set of molecular orbital indexes $\{x_0,...,x_n\}$, ϵ is the nth rank Levi-Cevita tensor, ζ is a member of the symmetric group S_X, consisting of all possible permutations on the set of indexes X.
In CI based methods it is common to use the fact that each distinct ordered set, X, of indexes corresponds to a different determinant (given all sets being compared have their indexes ordered according to some standard set of rules). In active space based methods the set of determinants used to define the wavefunction is constrained to only those determinants which corresponding $X \in F^{CAS}$, where

$$F^{CAS} = \{X | (X = X^c \cup X^a) \wedge (X^a \subset P) \wedge (\text{card}(X) = n_{act})\}$$

where P is the list of all "active" molecular orbital indexes, X^a is a subset of active orbitals occupied in the determinant defined by X, and where X^c is the list of closed orbital indexes, all of which are occupied in all determinants defined from members of F^{CAS}.

Unfortunately, F^{CAS} can get very large, leading to computational difficulties. One way of trying to manage this is to decompose the set of active orbital indexes, P, up into

n_p subsets;

$$P = \bigcup_{j=0}^{n_p} P_j, \tag{5.4}$$

and then splitting F^{CAS} up into subsets, $F_{o_1,\ldots,o_{n_p}}$, based on the number, o_j, of orbitals in each subspace, P_j, which are occupied :

$$F_{o_1,\ldots,o_{n_p}} = \left\{ X = X_c \cup \left[\bigcup_{j=0}^{n_p} X_j \right] \middle| (X_j \subset P_j) \wedge \left(\sum_j^{n_p} o_j = n_{act} \right) \right\}. \tag{5.5}$$

Here X_j is the subset of P_j with cardinality o_j. The second constraint is just the requirement that the sum of all the occupancy numbers, o_j, associated with each of the different subspaces, P_j, adds up to the total number of occupied electrons.

To illustrate this consider the case where P is split into two subspaces;

$$P = P_\mu \cup P_\nu. \tag{5.6}$$

For the case where

$$\text{card}(P_\mu) = 2, \quad \text{card}(P_\nu) = 4, \quad \text{and} \quad n_{act} = 3$$

The space, F^{CAS}, can then be split up according to how many γ and ν electrons are occupied in a given element;

$$F^{CAS} = F^{CAS}_{3,0} \cup F^{CAS}_{2,1} \cup F^{CAS}_{1,2}$$

$$F^{CAS}_{3,0} = \{X| \wedge (X^a_\mu \subset P_{mu}) \wedge (\text{card}(X)_\mu = n_{act}\}$$

$$F^{CAS}_{2,1} = \{X| \wedge (X^a_\mu \subset P_{mu}) \wedge (\text{card}(X)_\mu = n_{act} - 1\} \wedge (X^a_\nu \subset P_{nu}) \wedge (\text{card}(X)_\nu = 1\}$$

$$F^{CAS}_{1,2} = \{X| \wedge (X^a_\mu \subset P_{\mu}) \wedge (\text{card}(X)_\mu = n_{act} - 2\} \wedge (X^a_\nu \subset P_{nu}) \wedge (\text{card}(X)_\nu = 2\}$$

$$X = X^c \cup X^a_\mu \cup X^a_\nu.$$

I shall hereafter refer to these subspaces of F^{CAS} as CI-sectors (and in specific cases, spin-sectors). Decomposing the active space in this manner has a number of benefits, as the different CI-sectors may not interact, or have their interaction limited in some way. The most prominent example of this is the non-interaction of CI-sectors with different numbers of occupied α and β orbitals in the non-relativistic framework. However, even if they do, the decomposition of the active space makes it is possible to handle this interaction in a piecewise manner, i.e., deal with interactions between two CI-sectors

at a time.
One major advantage is that decomposition of the active-space into different components facilitates the block wise decomposition of the reduced density matrices (RDMs), as a block of an RDM can being defined by the CI-sectors to which the Bra and Ket belong. Using the above decomposition of the active space as an example;

$$\sum_{J}^{\in\mathcal{F}_{all}} \sum_{I}^{\in\mathcal{F}_{all}} \langle I|i^\dagger j^\dagger m^\dagger n^\dagger klop|J\rangle c_I^M c_J^N$$

$$\sum_{\mathcal{F}_{sub}}^{\in\mathcal{F}_{all}} \sum_{\mathcal{F}_{sub}}^{\in\mathcal{F}_{all}} \sum_{I}^{\in\mathcal{F}_{sub}} \sum_{J}^{\in\mathcal{F}_{sub}} \langle I|i^\dagger j^\dagger m^\dagger n^\dagger klop|J\rangle c_I^M c_J^N,$$

where $\mathcal{F}_{sub}$ is one of the sub-spaces into which the total Fock space $\mathcal{F}_{all}$ has been decomposed.

Unfortunately, this decomposition brings with it a number of complexities which are not present for undecomposed active spaces. However, use of such decomposition techniques can lead to significant gains in computational efficiency, particularly for methods which much handle multi-reference, relativistic wavefunctions.

Allowance for the fact that the Bra and Ket may belong to different CI-sectors, the blockwise decomposition of the molecular orbital tensors, and the exploitation of symmetry which exists within and between different blocks *and* different CI-sectors, are the distinguishing features of the program.

5.1 Blockwise handling of terms

That different Bra and Ket may have different CI-sectors has important ramifications for the program structure, however, before discussing these in depth it is necessary to discuss the basics of the task list construction.

In the previous section we discussed the construction of the task list used for calculating contractions between multiple different tensors, e.g., the task list for obtaining

$$A_{kwzm} = [\tilde{B}\tilde{C}\tilde{D}]_{kwzm}^{(jy,lx,no,pi)} = \sum_{jy}\sum_{no}\sum_{lx}\sum_{ip} B_{ijkl}C_{mnop}D_{wxyz}\delta_{jy}\delta_{no}\delta_{lx}\delta_{ip}. \tag{5.7}$$

The construction of this "contraction task list" (referred to as *A_compute_list* in the code, is effectively independent from the task list to be discussed now, which concerns

determination of which contracted tensors, e.g., which A_{kwzm}, that need to be calculated in order to obtain the expectation value or derivative.

As stated in the program overview, the total equation (or expression) to be solved (or evaluated), is broken down into a number of *Terms*, e.g.,

$$\langle M|\hat{Y}\hat{Z}|N\rangle,$$

which using second quantization as

$$\sum_{ijkl}\sum_{mnop}\sum_{IJ}\langle I|i^\dagger j^\dagger klm^\dagger n^\dagger op|J\rangle c_I^M c_J^N Y_{ijkl}^{ML} Z_{mnop}^{NP}. \tag{5.8}$$

In (5.8) the sum over the orbital indexes $\{i,j,k,l\}$ and $\{m,n,o,p\}$ runs over all indexes specified by tensors $\mathbf{Y}$ and $\mathbf{Z}$ respectively. However, we can rewrite the above in terms of summations over the CI-sectors and blocks into which these tensors may be decomposed;

$$\sum_{B^Y}\sum_{B^Z}\sum_{ijkl}^{B^Y}\sum_{mnop}^{B^Z}\sum_{IJ}\langle I|i^\dagger j^\dagger klm^\dagger n^\dagger op|J\rangle c_I^M c_J^N Y_{ijkl}^{ML} Z_{mnop}^{NP}. \tag{5.9}$$

Here B^Y and B^Z are blocks of $\mathbf{Y}$ and $\mathbf{Z}$ which define the ranges over which the summations of the molecular orbital indexes are to occur.

Many of the contributions from these various tensor blocks will vanish, but exactly which will vanish is dependent on the CI-sector to which $|I\rangle$ and $|J\rangle$ belong. For example, if

$$|I\rangle \in \mathcal{F}_{\mu=2,\nu=1} \qquad \text{and} \qquad |J\rangle \in \mathcal{F}_{\mu=3,\nu=0},$$

then the cumulative action of the creation and annihilation operators within the BraKet must be to destroy one electron in range r^μ, and create one in r^ν. All terms corresponding to blocks which do not meet this criterion can immediately be discarded. This idea can be taken further: If the expression is rearranged into normal order,

$$\sum_{B^Y}\sum_{B^Z}\sum_{ijkl}^{B^Y}\sum_{mnop}^{B^Z}\sum_{IJ}\langle I|i^\dagger j^\dagger m^\dagger n^\dagger klop|J\rangle c_I^M c_J^N Y_{ijkl}^{ML} Z_{mnop}^{NP}. \tag{5.10}$$

then not only do we retain the above constraint, but it is also known that terms corresponding to a combination of blocks, B^Y and B^Z, where *any* of the indexes k, l, o, p are in range ν, will vanish. An analogous constraint involving constraints on the

ranges of the creation operators is obtained by putting the expression into anti-normal ordering;

$$\sum_{B^Y}\sum_{B^Z}\sum_{ijkl}^{B^Y}\sum_{mnop}^{B^Z}\sum_{IJ}\langle I|klopi^\dagger j^\dagger m^\dagger n^\dagger|J\rangle c_I^M c_J^N Y_{ijkl}^{ML} Z_{mnop}^{NP}. \tag{5.11}$$

By applying this procedure we ensure that only indexes within the active space are left within the BraKet, which is computationally significant as the dimension of the active orbital subrange, r^a, is typically much smaller than the other subranges, such as those corresponding to core, r^c, or virtual, r_v, orbitals.

Whilst this rearrangement will generate new terms in accordance with the commutation relations of the creation and annihilation operators, all of these new terms will be of lower rank than the original. This fact, combined with the range constraints and elimination of terms means that the increase in the number of computational tasks to be performed is generally well worth it.

Perhaps most importantly of all this means calculations can be done in a piece-wise fashion, treating only one combination of CI-sectors at a time. This is particularly useful when calculating expressions requiring reduced density matrix derivatives,

$$\Gamma^I_{ijklmn}\sum_J \langle I|i^\dagger j^\dagger m^\dagger n^\dagger opkl|J\rangle c_J, \tag{5.12}$$

which, due to the lack of a summation over one of the CI-indexes, I, can prove extremely large. This could prove highly problematic for implementations of energy derivative expressions in the relativistic framework; the treatment of α and β electrons effectively doubles the size of the active space. However, the block decomposition of molecular orbital tensors, the decomposition of the active space, and the separation of blocks into real and imaginary components, ensures that the peak memory requirements of the calculations need not drastically exceed those for similar expressions in a non-relativistic framework.

A disadvantage is that the algebraic operations of rearranging the tensors must be performed for every possible block, of which there can in principal be millions[1]. Fortunately, there are a number of ways to mitigate this, but it is not so trivial an issue that it can be completely ignored.

The reordering sequence specified above; normal ordering followed by anti-normal or-

[1]If each range is divided in to six, and we have ten indexes, $6^{10} = 60466176$.

dering, encapsulates the most important principals used by the algebraic manipulator. However, more efficient task lists can be generated by following these two initial reorderings with others, the structure of which are determined by the properties of the block, the term being calculated[2]. In order to understand why this approach is taken, and how these reorderings are chosen and generated, it is necessary to discuss the implementation of symmetry.

5.2 Handling of block and spin-sector symmetry

One of the defining features of the program is that symmetry is incorporated at the stage of algebraic manipulation, and precedes the task list construction, i.e., it is only after all expressions have been rewritten to involve some minimal set of distinct terms that the task list is generated. Thus if two terms are equivalent, one of these terms will never appear at any stage of the final task list. This is primarily to facilitate the identification of terms which can be merged or reused (to save memory), and terms which can be completely separated (to facilitate parallelization).

Whilst the previous section discussed how symmetry was applied in execution and representation of the contraction task list, it did not discuss how symmetry was used to decide what quantities that task list was attempting to calculated. The focus of this section is the latter of these two points.

Broadly speaking, the algorithm should function such that given the following equivalences:

$$\hat{T}^{KM} = \hat{T}^{KP} \ , \quad \hat{T}^{-1} = T^{\dagger} \ , \text{ and } \quad \mathbf{T}^{KM}_{b1} = \mathbf{T}^{KP}_{b2}$$

$$\mathbf{H}_{b3} = \mathbf{H}_{b4}$$

$$\mathbf{c}^{P}_{K} = -\boldsymbol{\sigma_{y,2\times 2}}\mathbf{c}^{M*}_{J}, |K\rangle = |\overline{J}\rangle, \quad \text{where} \quad \boldsymbol{\sigma_{y,2\times 2}} = \begin{bmatrix} \boldsymbol{\sigma_y} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\sigma_y}, \end{bmatrix}$$

any contribution written using the terms on the left hand side (LHS) of the above can be rewritten as one involving the terms on the right hand side (RHS) of the above. For example,

$$\sum_{ijkl}^{b1}\sum_{mnop}^{b2} \langle K|\hat{i}^{\dagger}\hat{j}^{\dagger}\hat{k}\hat{l}\hat{m}^{\dagger}\hat{n}^{\dagger}\hat{o}\hat{p}|K\rangle \mathbf{c}^{P}_{K}\mathbf{c}^{M}_{I}T^{KM,-1,b1}_{ijkl}H^{b3}_{mnop}$$

[2]For example, terms involving CI-derivatives use different reorderings to terms involving molecular orbital tensor derivatives.

is re-expressed as this term

$$\sum_{ijkl}^{b1}\sum_{mnop}^{b3}\langle I|\hat{m}^\dagger\hat{n}^\dagger\hat{o}\hat{p}\hat{\bar{i}}^\dagger\hat{\bar{j}}^\dagger\hat{\bar{k}}\hat{\bar{l}}|\overline{J}\rangle\mathbf{c}_J^{M*}\mathbf{c}_I^M H_{mnop}^{b4}T_{lkji}^{KM,b2}$$

plus some sequence of transformations, and potentially a number of lower order terms which will arise due to the switching of the order of $\hat{H}$ and $\hat{T}$. Here the notations $\bar{i}$ is the index corresponding to the Kramer's pair of orbital i.

The key thing to note in the above is that the transformations of the blocks range limits on the summations are not transformed, whilst those on the molecular orbital tensors are. Furthermore, the indexes inside the BraKet are not transformed with the representations of the molecular orbital tensors, but are transformed due to the time reversal symmetry operation resulting from the substitution $|K\rangle = |\overline{I}\rangle$, and the Hermitian conjugation operation applied to $\hat{T}$.

This is because transformations of the operator, such as inversion and time-reversal, are fundamentally different to transformations of the representation of an operator in the molecular orbital basis, e.g., mapping $T_{ijkl}^{b1} \rightarrow T_{klij}^{b2}$. Transformations to the operator itself will impact the ranges on which the creation and annihilation operators used to define that operator act, e.g., in the above case time-reversal alters the ranges on which the creation and annihilation operators act, and hence alters the ranges within the BraKet. Conversely, symmetries arising from properties of the molecular orbital indexes on the molecular orbital tensors do not influence the ranges on which the creation and annihilation operators act, as these are symmetries arising from the molecular orbital basis chosen to represent the operator.

Clearly, there is a very close link between the two, but they are distinct. By way of example consider an operator, $\hat{W}$, which is split into two blocks

$$b1 = \{\alpha, \alpha, \alpha, \alpha\} \qquad \text{and} \qquad b2 = \{\beta, \beta, \beta, \beta\}$$

ranges on indexes. Suppose that ranges r_α and r_β have equal extent, and that the representation of $\hat{W}$ within each of these blocks is equivalent, i.e.,

$$W_{ijkl}^{b1} = W_{ijkl}^{b2}. \tag{5.13}$$

Now consider a 2-electron wavefunction, formed from a linear combination of determi-

nants, in each of which two alpha orbitals are occupied

$$|M\rangle = \sum_I c_I^M |I(n_\alpha = 2, n_\beta = 0\rangle. \tag{5.14}$$

We can now write

$$\langle M|\hat{W}|M\rangle = \langle M|\hat{W}^{b1}|M\rangle + \langle M|\hat{W}^{b2}|M\rangle$$

$$= \sum_{ijkl}^{b1} \sum_{IJ} \langle I|\hat{i}^\dagger \hat{j}^\dagger \hat{k}\hat{l}|J\rangle c_I c_J W_{ijkl}^{b1} + \sum_{ijkl}^{b2} \sum_{IJ} \langle I|\hat{i}^\dagger \hat{j}^\dagger \hat{k}\hat{l}|J\rangle c_I c_J W_{ijkl}^{b2},$$

which then using $W_{ijkl}^{b1} = W_{ijkl}^{b2}$ leads to

$$= \sum_{ijkl}^{b1} \sum_{IJ} \langle I|\hat{i}^\dagger \hat{j}^\dagger \hat{k}\hat{l}|J\rangle c_I c_J W_{ijkl}^{b1} + \sum_{ijkl}^{b2} \sum_{IJ} \langle I|\hat{i}^\dagger \hat{j}^\dagger \hat{k}\hat{l}|J\rangle c_I c_J W_{ijkl}^{b1}.$$

However, it is clear from the definition of $|M\rangle$ that the second term must always be zero, as there are no β electrons to destroy, hence,

$$\neq 2 \sum_{ijkl}^{b1} \sum_{IJ} \langle I|\hat{i}^\dagger \hat{j}^\dagger \hat{k}\hat{l}|J\rangle c_I c_J W_{ijkl}^{b1}.$$

This means that whilst we can transform all occurrences of $\mathbf{W}^{b2}$ into occurrences of $\mathbf{W}^{b1}$, we are typically unable to apply the same transformation to the block ranges and the molecular orbital indexes which appear withing the braket. Transformations of these indexes only arise when performing transformations on the physical action of the operator, such as time-reversal, or inversion.

It is for this reason that symmetries are not dealt with by using a single $i < j < k < l$ type rule, but instead are subdivided into three distinct types, which can be combined to guide construction of the task list. These three basic symmetry types are : operator symmetry, representation (or block) symmetry, and CI-sector symmetry.

Operator symmetry refers to transformations on the action of the operator, and equivalences between operators which act on different combinations of states (dictated by time reversal, or potentially spatial symmetry). For example,

$$\hat{T}^{LM} = \hat{T}_{ijkl}^{\overline{LM}} \tag{5.15}$$

where $\overline{L}$ is the index referring to the Kramers pair of the state with index L. Another

kind of operator symmetry is

$$T^{LM,b}_{ijkl} = T^{\overline{LM},\bar{b}}_{ijkl} \tag{5.16}$$

Representation or block symmetries are relations such as

$$\mathbf{T}^{LM,b1}_{ijkl} = \rho \mathbf{T}^{LM,b2}_{klij} \tag{5.17}$$

where $b2$ is some index block which may be obtained by performing a permutation operation of the indexes of $b1$, and ρ is some multiplicative factor. Another kind of common block symmetry is

$$\mathbf{T}^{LM,b}_{ijkl} = \rho \mathbf{T}^{\overline{LM},\bar{b}}_{ijkl} * \tag{5.18}$$

$\bar{b}$ is the range block defined from the index ranges of the which are Kramers pairs of the molecular orbital index ranges which define block b. Similarly, $\overline{L}$ and $\overline{M}$ are the indexes of the states which are Kramers pairs of L and M respectively.

Finally, an example of CI-sector symmetry would be

$$\mathbf{c}^P_K = -\boldsymbol{\sigma}_{\boldsymbol{y},\mathbf{2\times 2}} \mathbf{c}^{M*}_J, |K\rangle = |\overline{J}\rangle, \quad \text{where} \quad \boldsymbol{\sigma}_{\boldsymbol{y},\mathbf{2\times 2}} = \begin{bmatrix} \boldsymbol{\sigma}_{\boldsymbol{y}} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\sigma}_{\boldsymbol{y}} \end{bmatrix}$$

This is just time reversal symmetry.

Initially, the state dependence of the operators seems peculiar; the molecular orbitals themselves do not differ between states, hence the state dependence of the representations suggests that it is not possible to interpret these operators as describing interactions between 1, 2, or n-electrons. In fact, such state dependent operators do not correspond to physical interactions per se, but are instead a tool used to aid in the description of representation of a perturbed state in terms via interaction of a number of unperturbed states. An important consequence of this is that the symmetry of the operator representations is determined by the form of the equations from which they are obtained, rather than from consideration of the form of operators themselves.

Chapter 6: Theoretical supplement

Note : This section outlines some of theoretical background on which the program is based. However, it does not contain any new developments; its primary purpose is as a quick reference, and to clarify some of the notation used in the main body of this document.

Begin by defining a space s, split into a reference, p, and external, **q**, space;

$$\mathrm{s} = \mathrm{p} + \mathrm{s} = \{|p\rangle\} \cup \{|q\rangle\} \tag{6.1}$$

Now define projection operators

$$\hat{P} = \sum_{p} |p\rangle\langle p| \qquad \hat{Q} = \sum_{q} |q\rangle\langle q| \tag{6.2}$$

$$\hat{P} + \hat{Q} = \hat{I} \qquad \hat{P} - \hat{I} = \hat{Q} \tag{6.3}$$

Then have zeroth order Hamiltonian, $\hat{H}_0$, and perturbation, $\hat{V}$. The spaces p and q are such that

$$\langle p|\hat{H}_0|q\rangle = \langle q|\hat{H}_0|p\rangle = \langle p|\hat{V}|p\rangle = 0 \quad \forall \;\; p,\, q. \tag{6.4}$$

Write two equations for the zeroth order, $\hat{H}_0$, and full, $\hat{H} = \hat{H}_0 + \hat{V}$, Hamiltonians:

$$\hat{H}_0|p\rangle = \epsilon_p|p\rangle \tag{6.5}$$

$$\hat{H}|\Psi_p\rangle = (\hat{H}_0 + \hat{V})|\Psi_p\rangle = (\epsilon_p + E_p)|\Psi_p\rangle \tag{6.6}$$

Here, $|\Psi_p\rangle$, is being treated as though it were a perturbation of state $|p\rangle$. Initially, assume that there is only one state in the space p. It is now more convenient to write this as

$$(\epsilon_p - \hat{H}_0)|\Psi_p\rangle = (\hat{V} - E_p)|\Psi_p\rangle \tag{6.7}$$

The wavefunction $|\Psi_p\rangle$ may be expanded in p and q;

$$|\Psi_p\rangle = \sum_p C_p|p\rangle + \sum_q T_q|q\rangle. \tag{6.8}$$

Hence,

$$\hat{P}|\Psi_p\rangle = \sum_p C_p|p\rangle \qquad \text{and} \qquad \hat{Q}|\Psi_p\rangle = \sum_q T_q|q\rangle. \tag{6.9}$$

This, combined with the identities associated with the projectors $\hat{P}$ and $\hat{Q}$, and the attributes of the space, s, stated in (6.4) leads to the following statement

$$\hat{P}(\epsilon_p - \hat{H}_0)(\hat{P} + \hat{Q})|\Psi_p\rangle = \hat{P}(\hat{V} - E_p)(\hat{P} + \hat{Q})|\Psi_p\rangle. \tag{6.10}$$

Taking each side separately:

$$\hat{P}(\epsilon_p - \hat{H}_0)(\hat{P} + \hat{Q})|\Psi_p\rangle = 0, \tag{6.11}$$

$$\hat{P}(\hat{V} - E_p)(\hat{P} + \hat{Q})|\Psi_p\rangle = \hat{P}\hat{V}\hat{Q}|\Psi_p\rangle + \hat{Q}E_p\hat{Q}|\Psi_p\rangle. \tag{6.12}$$

Substituting (6.11) and (6.12) back into (6.10) and rearranging we obtain

$$\hat{P}\hat{V}\hat{Q}|\Psi_p\rangle = \hat{P}E_p\hat{P}|\Psi_p\rangle. \tag{6.13}$$

Assuming the members of p are orthonormal and taking advantage of the consequent idempotency of $\hat{P}$ leads to

$$\hat{P}E_p\hat{P}|\Psi_p\rangle = E_p\hat{P}|\Psi_p\rangle, \tag{6.14}$$

hence

$$\sum_{p'}\langle p'|E_p\hat{P}|\Psi_p\rangle = \sum_{p'}\langle p'|\hat{V}\hat{Q}|\Psi_p\rangle. \tag{6.15}$$

Whilst the primary intention of the MRPTool module is to tackle multistate, highly degenerate cases, it helpful[1] to first consider the singlestate case, i.e., where the dimension of p is 1. If intermediate normalization, $\langle p|\Psi_p\rangle = 1$, is assumed then

$$\hat{P}E_p\hat{P}|\Psi_p\rangle = E_p, \tag{6.16}$$

which leads to

$$E_p = \langle p|\hat{V}\hat{Q}|\Psi_p\rangle. \tag{6.17}$$

If in (6.10) $\hat{Q}$ instead of $\hat{P}$ was applied from the left the following expressions would

[1] for me, at least

result:

$$\hat{Q}(\hat{V} - E_p)(\hat{P} + \hat{Q})|\Psi_p\rangle = \hat{Q}\hat{V}|\Psi_p\rangle - \hat{Q}E_p\hat{Q}|\Psi_p\rangle, \tag{6.18}$$

$$\hat{Q}(\epsilon - \hat{H}_0)(\hat{P} + \hat{Q})|\Psi_p\rangle = \hat{Q}\epsilon_p\hat{Q}|\Psi\rangle - \hat{Q}\hat{H}_0\hat{Q}|\Psi_p\rangle. \tag{6.19}$$

However, noting that the RHS of (6.11) is 0 it is possible to write

$$\hat{Q}(\epsilon_p - \hat{H}_0)(\hat{P} + \hat{Q})|\Psi_p\rangle = (\hat{P} + \hat{Q})(\epsilon_p - \hat{H}_0)(\hat{P} + \hat{Q})|\Psi_p\rangle = (\epsilon_p - \hat{H}_0)|\Psi_p\rangle,$$

hence

$$(\epsilon - \hat{H}_0)|\Psi_p\rangle = \hat{Q}\epsilon_p\hat{Q}|\Psi\rangle - \hat{Q}\hat{H}_0\hat{Q}|\Psi_p\rangle. \tag{6.20}$$

Equating the RHS's of (6.20) and (6.18)

$$(\epsilon_p - \hat{H}_0)|\Psi_p\rangle = \hat{Q}\hat{V}|\Psi_p\rangle - \hat{Q}E_p\hat{Q}|\Psi_p\rangle.$$

If there is only one state in p, then it is possible to substitute in from (6.17) to obtain

$$(\epsilon_p - \hat{H}_0)|\Psi_p\rangle = \hat{Q}\hat{V}|\Psi_p\rangle - \hat{Q}[\langle p|\hat{V}\hat{Q}|\Psi_p\rangle]\hat{Q}|\Psi_p\rangle,$$

noting that $[\langle p|\hat{V}\hat{Q}|\Psi_p\rangle] = [\langle p|\hat{V}|\Psi_p\rangle]$, which is just a number, and that Q is idempotent, this can be rewritten

$$(\epsilon_p - \hat{H}_0)|\Psi_p\rangle = \hat{Q}\hat{V}|\Psi_p\rangle - \hat{Q}|\Psi_p\rangle[\langle p|\hat{V}|\Psi_p\rangle]. \tag{6.21}$$

It is useful to define a "wave operator" $\hat{\Omega}^p$ which can be used to obtain the perturbed state, $|\Psi_p\rangle$, from a state $\{|p\rangle\}$ within the reference space p;

$$|\Psi_p\rangle = \hat{\Omega}_p|p\rangle. \tag{6.22}$$

This may be used to rewrite (6.21) as

$$(\epsilon_p - \hat{H}_0)\hat{\Omega}^p|p\rangle = \hat{Q}\hat{V}\hat{\Omega}^p|p\rangle - \hat{Q}\hat{\Omega}^p|p\rangle[\langle p|\hat{V}\hat{\Omega}^p|p\rangle]. \tag{6.23}$$

CASPT2 and related theories discussed in the following can be thought of as techniques for finding approximate methods for finding Ω_p.

6.1 CASPT2

The wave operator can be written as a series expansion;

$$\hat{\Omega}^p = \hat{\Omega}^{p,0} + \hat{\Omega}^{p,1} + \hat{\Omega}^{p,2} + ... \tag{6.24}$$

where

$$\hat{\Omega}^{p,0}|p\rangle = |p^0\rangle = |p\rangle, \quad \hat{\Omega}^{p,1}|p\rangle = |p^1\rangle, \quad \hat{\Omega}^{p,2}|p\rangle = |p^2\rangle, \text{ etc..} \tag{6.25}$$

Where $\langle p^i|p^j\rangle = \delta_{ij}$, hence $(\hat{Q}\hat{\Omega}^{p,0}) = 0$. Expanding out the energy in a similar manner, substituting the above expansion into (6.23) and equating first order terms leads to the first order wave equation,

$$(\epsilon_p - \hat{H}_0)\Omega^{p,1}|p\rangle = \hat{Q}\hat{H}|p\rangle, \tag{6.26}$$

which makes use of the fact that $\hat{Q}\hat{V}|p\rangle = \hat{Q}\hat{H}|p\rangle$. A similar logic may be used to obtain an expression for the second order energy;

$$E_p^{(2)} = E_p^{(1)} + \langle p|\hat{H}\hat{\Omega}^{p,1}|p\rangle. \tag{6.27}$$

Provided the above orthogonality constraints are met there is considerable freedom in choosing the form of the terms in the series (6.25). A natural choice is to have $\hat{\Omega}^{p,1}$, be the contribution to $|\Psi_p\rangle$ from single and doubly excited states, $\hat{\Omega}^{p,2}$, the contribution from triply and quadruply excited states, and so on:

$$\hat{\Omega}^{p,0} = \sum_m |m\rangle\langle m|, \quad m \in \mathrm{p}$$

$$\hat{\Omega}^{p,1} = \sum_r |r\rangle\langle p|T_r^p \quad r \in \mathrm{q}^{sd}$$

$$\hat{\Omega}^{p,2} = \sum_x |x\rangle\langle p|T_x^p \quad x \in \mathrm{q}^{tq}, \text{ etc.. .}$$

The above makes use of the fact that the space, q, may be split up into mutually orthogonal subspaces corresponding to singly and doubly excited states, triply and quadruply excited states, etc..,

$$\mathrm{q} = \mathrm{q}^{sd} + \mathrm{q}^{tq} + ... \tag{6.28}$$

The T_r^p are amplitude coefficients indicating how much $|r\rangle \in q^{(st)}$ contributes to $|\Psi_p\rangle$.

In the internally contracted framework projection into these excited spaces is achieved via excitation operators, e.g., $\hat{E}_\omega$, which excite the appropriate number of electrons from the closed or active orbitals specified by ω;

$$\hat{E}^{sd}_\omega|p\rangle = |r_p\rangle \qquad |r\rangle \in q^{sd}. \tag{6.29}$$

Note that the state onto which the operator $\hat{E}^{sd}_\omega$ projects is dependent on the state $|p\rangle$ upon which it acts. So in principal, it is possible to replace the index denoting orbital excitations with a subscripted index, ω_p, where ω_p is the state onto which the action of $\hat{E}_\omega$ on $|p\rangle$ projects, i.e.,

$$\hat{E}_\omega|p\rangle \rightarrow \hat{E}_{r_p}|p\rangle = |r_p\rangle\langle p|p\rangle. \tag{6.30}$$

Though needlessly prolix when the subspace p contains only one state, this notation will come in use later on when dealing with the multistate case. However, until that point, the subscript on p shall be omitted.

Rewriting (6.26) using these definitions gives

$$\sum_{r'}(\epsilon_p - \hat{H}_0)T^{st}_{r'}\Omega^{p,1}|p\rangle = \sum_r \hat{Q}^{st}_r\hat{H}|p\rangle. \tag{6.31}$$

$$\sum_{r}^{r\in q^{st}}(\epsilon_p - \hat{H}_0)T^{p}_{r}\hat{E}_r|p\rangle = \sum_{r}^{r\in q^{st}}|r\rangle\langle r|\hat{H}|p\rangle.$$

Rearranging and left multiplying by $\langle r| \in q^{st}$ gives

$$\langle r'|\sum_{r}^{r\in q^{st}}(\epsilon_p - \hat{H}_0)T^{p}_{r}|r\rangle - \langle r'|\sum_{r}^{r\in q^{st}}|r\rangle\langle r|\hat{H}|p\rangle = 0$$

$$\langle r'|\sum_{r}^{r\in q^{st}}(\epsilon_p - \hat{H}_0)T^{p}_{r}|r\rangle - \langle r'|\hat{H}|p\rangle = 0 \tag{6.32}$$

The derivative with respect T should vanish, so at convergence,

$$\sum_{r}^{r\in q^{st}}\langle r'|(\epsilon_p - \hat{H}_0)T^{p}_{r}|r\rangle = G = 0 \tag{6.33}$$

Using the definition of the functional derivative;

$$\frac{G[T^p_r + \delta T] - G[T^p_r]}{\delta T^p_r} = 0,$$

$$\frac{\delta G[T_r^p]}{\delta T_r^p} = \langle r|(\epsilon_p - \hat{H}_0)|r\rangle. \tag{6.34}$$

This can now be used to obtain the T update equation

$$\Delta T_r = \left(\frac{\delta G[T_r^p]}{\delta T_r^p}\right)^{-1} G[T_r^p] = \frac{\sum_r^{r\in q^{st}} \langle r'|(\epsilon_p - \hat{H}_0)T_r^p|r\rangle}{\langle r'|(\epsilon_p - \hat{H}_0)|r'\rangle}. \tag{6.35}$$

This expression is iterated over until the threshold is reached.

6.2 Multistate cases

As mentioned above, the case where p contains more than one state is less straightforward, and there a number of approaches to dealing with it (MS-CASPT2, XMS-CASPT2, GVVPT2 etc.,). To explain the issues associated with the degenerate multistate case, and to illustrate the different in motivation and structure of these approaches, the above expressions are now rewritten using an explicit basis for spaces p and q:

$$\hat{P} = \sum_p |p\rangle\langle p| \qquad \hat{Q} = \sum_q |q\rangle\langle q|,$$

$$\hat{P}|\Psi_p\rangle = \sum_p \sum_l |p\rangle\langle p|l\rangle X_l^p \qquad \hat{Q}|\Psi_p\rangle = \sum_q \sum_r |q\rangle\langle q|r\rangle T_r^p,$$

where X_l^p and T_r^p are coefficients. The X_l^p may be interpreted as the coefficients describing how the perturbation acts to mix together the states within the reference space, p. If the states within the reference space are well separated energetically, then it is reasonable to assume that this mixing is small, and $X_l^p \to \delta_l^p$, however, this is not the case, by definition, in the vicinity of conical intersections.

In the following initial discussion of approaches to this problem it shall be assumed that the sets of $\{|l\rangle\}$ and $\{|r\rangle\}$ being summed over are identical to the respective sets $\{|p\rangle\}$ and $\{|q\rangle\}$. This assumption is not valid in the case of internally contracted methods, which will be discussed in due course.

Rewriting (6.11) with this basis gives

$$\sum_{mnl} |m\rangle\langle m|(\epsilon_p - \hat{H}_0)|n\rangle\langle n|l\rangle X_l^p + \sum_{mqr} |m\rangle\langle m|(\epsilon_p - \hat{H}_0)|q\rangle\langle q|r\rangle T_r^p$$

$$\sum_{mnl} |m\rangle\langle m|(\epsilon_p - \hat{H}_0)|l\rangle X_l^p + \sum_{mqr} |m\rangle\langle m|(\epsilon_p - \hat{H}_0)|r\rangle T_r^p$$

Multiplying from the left by state $\langle\Psi_p| = \sum_k \langle k|{X_k^p}^\dagger$ yields

$$\sum_{kmnl} X_k^{p\dagger}\langle k|m\rangle\langle m|(\epsilon_p - \hat{H}_0)|l\rangle X_l^p + \sum_{kmqr} X_k^{p\dagger}\langle k|(\epsilon_p - \hat{H}_0)|r\rangle T_r^p$$

$$= \sum_{kl} X_k^{p\dagger}\langle k|(\epsilon_p - \hat{H}_0)|l\rangle X_l^p \tag{6.36}$$

The useful expression (6.17) for the perturbation energy, E_p, may be obtained because the RHS of (6.11), the single state analogue of (6.36), vanishes. Unfortunately, this does not happen here, a fact which will later prove significant. There is more than one way of handling this issue, but to summarize them properly it is worth first discussing the generalized Bloch equation.

6.3 Generalized Bloch Equation

Consider the matrix equation,

$$\mathbf{H\Psi} = \begin{bmatrix} \mathbf{H_{PP}} & \mathbf{H_{PQ}} \\ \mathbf{H_{QP}} & \mathbf{H_{QQ}} \end{bmatrix} \begin{bmatrix} \mathbf{\Psi}_P \\ \mathbf{\Psi}_Q \end{bmatrix} = E \begin{bmatrix} \mathbf{\Psi}_P \\ \mathbf{\Psi}_Q \end{bmatrix} \tag{6.37}$$

Now introduce a transformation operator, $\mathbf{S}$, which can be used to obtain the full vector, $\mathbf{\Psi}$, from just the component, $\mathbf{\Psi}_P$, within the p subspace, i.e.,

$$\mathbf{S(T)\Psi}_0 = \begin{bmatrix} \mathbf{I}_{PP} & 0 \\ \mathbf{T} & \mathbf{I}_{QQ} \end{bmatrix} \begin{bmatrix} \mathbf{\Psi}_P \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{\Psi}_P \\ \mathbf{T\Psi}_P \end{bmatrix} = \mathbf{\Psi} \tag{6.38}$$

The matrix $\mathbf{S(T)}$ has the useful property

$$\mathbf{S(T}_1) + \mathbf{S(T}_2) = \mathbf{S(T}_1 + \mathbf{T}_2), \tag{6.39}$$

which has the corollary $\mathbf{S(T)}^{-1} = \mathbf{S}(-\mathbf{T})$. Noting this it is possible to write

$$\mathbf{S(T)}^{-1}\mathbf{HS}(-\mathbf{T})\mathbf{\Psi}_0 = \tag{6.40}$$

$$\begin{bmatrix} \mathbf{I}_{PP} & \mathbf{0} \\ -\mathbf{T} & \mathbf{I}_{QQ} \end{bmatrix} \begin{bmatrix} \mathbf{H}_{PP} & \mathbf{H}_{PQ} \\ \mathbf{H}_{QP} & \mathbf{H}_{QQ} \end{bmatrix} \begin{bmatrix} \mathbf{I}_{PP} & \mathbf{0} \\ \mathbf{T} & \mathbf{I}_{QQ} \end{bmatrix} \begin{bmatrix} \mathbf{\Psi}_P \\ \mathbf{0} \end{bmatrix} = E \begin{bmatrix} \mathbf{\Psi}_P \\ \mathbf{0} \end{bmatrix} \tag{6.41}$$

The blocks of the transformed matrix $\tilde{\mathbf{H}} = \mathbf{S(T)}^{-1}\mathbf{HS}(-\mathbf{T})$ are

$$\tilde{\mathbf{H}}_{PP} = \mathbf{H}_{PP} + \mathbf{H}_{PQ}\mathbf{T} \tag{6.42}$$

$$\tilde{\mathbf{H}}_{PQ} = \mathbf{H}_{PQ} \tag{6.43}$$

$$\tilde{\mathbf{H}}_{QP} = \mathbf{H}_{QP} + \mathbf{H}_{QQ}\mathbf{T} - \mathbf{H}_{PP}\mathbf{T} - \mathbf{T}\mathbf{H}_{PQ}\mathbf{T} \tag{6.44}$$

$$\tilde{\mathbf{H}}_{QQ} = \mathbf{H}_{QQ} - \mathbf{T}\mathbf{H}_{PQ} \tag{6.45}$$

The expression (6.42) defines the effective Bloch Hamiltonian, and can be directly connected to the Bloch equation specified in (6.23). Requiring that $\mathbf{T}$ is such that (6.44) is zero will decouple the H_{PP} block rest of the blocks of the transformed matrix, enabling the obtain eigenvalues associated with the PP block to be obtained separately from those associated with the rest of the matrix;

$$\tilde{\mathbf{H}} \begin{bmatrix} \boldsymbol{\psi_P} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{H}}_{\mathbf{PP}} & \tilde{\mathbf{H}}_{\mathbf{PQ}} \\ 0 & \tilde{\mathbf{H}}_{\mathbf{QQ}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Psi_P} \\ \mathbf{0} \end{bmatrix} = E \begin{bmatrix} \boldsymbol{\Psi_P} \\ \mathbf{0} \end{bmatrix}. \tag{6.46}$$

This can now be connected to the Bloch equation stated earlier by making use of the notation;

$$\mathbf{P} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad \boldsymbol{\Omega} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{F} & \mathbf{0} \end{bmatrix}, \quad \boldsymbol{\Psi}^0 = \begin{bmatrix} \boldsymbol{\Psi}_P \\ \mathbf{0} \end{bmatrix}, \tag{6.47}$$

Using this notation the Bloch Hamiltonian, $\mathbf{H}^{Bloch}$, may be written

$$\mathbf{H}^{Bloch} = \mathbf{P}\mathbf{H}\boldsymbol{\Omega}\mathbf{P}\boldsymbol{\Psi}. \tag{6.48}$$

As stated earlier, if $\mathbf{T}$, and hence $\boldsymbol{\Omega}$, is defined such that (6.44) vanishes, then the eigenvalues of this effective Hamiltonian will be a subset of the eigenvalues of the full Hamiltonian. It is worth highlighting the identity $\boldsymbol{\Omega}\mathbf{P} = \boldsymbol{\Omega}$, i.e., the wave-operator Ω ensures that the effective Hamiltonian only acts on the components of states which are within the p subspace.

This effective Hamiltonian is the main workhorse of the multireference perturbation theories, but it is worth continuing with some further analysis and discussion of the Bloch equation so as to facilitate later discussions of approximate methods. The generalized Bloch equation is

$$\boldsymbol{\Omega}\mathbf{H}\boldsymbol{\Omega} = \mathbf{H}\boldsymbol{\Omega} \tag{6.49}$$

To see this more directly it can be written out using the matrices specified above;

$$= \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{H}_{PP} & \mathbf{H}_{PQ} \\ \mathbf{H}_{QP} & \mathbf{H}_{QQ} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{T} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{PP} & \mathbf{H}_{PQ} \\ \mathbf{H}_{QP} & \mathbf{H}_{QQ} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{T} & \mathbf{0} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{H}_{PP} + \mathbf{H}_{PQ}\mathbf{T} & \mathbf{0} \\ \mathbf{T}\mathbf{H}_{PP} + \mathbf{T}\mathbf{H}_{PQ}\mathbf{T} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{PP} + \mathbf{H}_{PQ}\mathbf{T} & \mathbf{0} \\ \mathbf{H}_{QP} + \mathbf{H}_{QQ}\mathbf{T} & \mathbf{0} \end{bmatrix}$$

to see that the last two lines are equivalent note use the fact that $\mathbf{T}$ is such that $\tilde{\mathbf{H}}_{QP}$ as defined in (6.44) vanishes. Rearranging (6.44) yields

$$\mathbf{H}_{QP} + \mathbf{H}_{QQ}\mathbf{T} = \mathbf{H}_{PP}\mathbf{T} + \mathbf{T}\mathbf{H}_{PQ}\mathbf{T}.$$

Hence the two sides of (6.3) are equivalent. It should be noted that it is in the failure of the choice of $\mathbf{T}$ to satisfy the condition specified in (6.3) that many errors originate.

Whilst (6.49) is the perhaps the most generic form of the Bloch equation, it is not the one most commonly seen in the CASPT2 literature, which is

$$(E_0 - \mathbf{H}_0)\mathbf{\Omega}\mathbf{\Psi}_0 = \mathbf{Q}\mathbf{V}\mathbf{\Omega}\mathbf{\Psi}_0 - \mathbf{Q}\mathbf{\Omega}\mathbf{P}\mathbf{V}\mathbf{\Omega}\mathbf{\Psi}_0 \tag{6.50}$$

To obtain this expression introduce the notation

$$\mathbf{H}_0 = \begin{bmatrix} \mathbf{H}_{PP} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_{QQ} \end{bmatrix}, \qquad \mathbf{V} = \begin{bmatrix} \mathbf{0} & \mathbf{H}_{PQ} \\ \mathbf{H}_{QP} & \mathbf{0} \end{bmatrix}. \tag{6.51}$$

Where,

$$\mathbf{H}_0\mathbf{\Psi}_0 = \mathbf{E}^{(0)}\mathbf{\Psi}_0 \tag{6.52}$$

and

$$(\mathbf{H}_0 + \mathbf{V})\mathbf{\Psi} = (\mathbf{E}^0 + \mathbf{E}^{'})\mathbf{\Psi} \tag{6.53}$$

Writing $\mathbf{\Psi}$ as $\mathbf{\Omega}\mathbf{\Psi}_0$, and using employing the generalized Bloch equation (6.49) leads to

$$\mathbf{\Omega}(\mathbf{H}_0 + \mathbf{V})\mathbf{\Omega}\mathbf{\Psi}_0 = \mathbf{\Omega}(\mathbf{E}^0 + \mathbf{E}^{'})\mathbf{\Omega}\mathbf{\Psi}_0 \tag{6.54}$$

and expression for $\mathbf{E}^{'}$ (for those states within the P-block only);

$$\mathbf{E}^{'} = \mathbf{\Omega}\mathbf{P}\mathbf{V}\mathbf{\Omega}\mathbf{\Psi}_0. \tag{6.55}$$

Similarly, it is also possible to write

$$(\mathbf{H}_0 + \mathbf{V})\mathbf{\Omega}\mathbf{\Psi}_0 = (\mathbf{E}^0 + \mathbf{E}^{'})\mathbf{\Omega}\mathbf{\Psi}_0$$

$$\rightarrow (\mathbf{E}^0 - \mathbf{H}_0)\mathbf{\Omega}\mathbf{\Psi}_0 = (\mathbf{V} - \mathbf{E}^{'})\mathbf{\Omega}\mathbf{\Psi}_0 \tag{6.56}$$

Using the expression for $\mathbf{E}^{'}$ in (6.55) gives

$$(\mathbf{E}^0 - \mathbf{H}_0)\mathbf{\Omega\Psi}_0 = \mathbf{V\Omega\Psi}_0 - \mathbf{\Omega PV\Omega\Psi}_0. \tag{6.57}$$

The RHS can now be multiplied by $(\mathbf{P} + \mathbf{Q})$, and upon noting that

$$\mathbf{PV\Omega\Psi}_0 = \mathbf{P\Omega PV\Omega\Psi}_0.$$

this term can be cancelled, leaving

$$(\mathbf{E}^0 - \mathbf{H}_0)\mathbf{\Omega\Psi}_0 = \mathbf{QV\Omega\Psi}_0 - \mathbf{Q\Omega PV\Omega\Psi}_0, \tag{6.58}$$

which is the form in which the Bloch equation appears much of the perturbation theory literature.

It is important to note that whilst (6.58) is the principal equation, for it to hold it (given the above definitions) requires that $\mathbf{T}$ is such that (6.44) vanishes. Often, approximations are used in the definition of the matrix components which appear in (6.44), and how these approximations impact the accuracy of methods based on (6.58) is an important topic which we return to in due course.

6.4 Multistate CASTP2 and Extended Multistate CASPT2

Multistate methods generally start from the effective Bloch Hamiltonian defined in (6.48) as

$$\hat{H}^{Bloch} = \hat{P}\hat{H}\hat{\Omega}\hat{P} \tag{6.59}$$

as in the perturbation theory for the single state case the wave operator, $\hat{\Omega}$, is expanded as a series of which we take the only the first two terms

$$\hat{\Omega} = \hat{\Omega} = 1 + \hat{\Omega}^{sd} + \hat{\Omega}^{qt} + \tag{6.60}$$

$$\hat{H}^{Bloch,2} = \hat{P}\hat{H}\hat{P} + \hat{P}\hat{H}\hat{\Omega}^{sd}\hat{P}. \tag{6.61}$$

The eigenvalues of this effective Hamiltonian are the energies of the perturbed wave-function.

To reconnect with the previous discussion around (6.36) this may be be written out

with the summations made explicit,

$$\hat{H}^{Bloch,2} = \sum_{mn} |m\rangle\langle m|\hat{H}|n\rangle\langle n| + \sum_{mn} |m\rangle\langle m|\hat{H}\hat{\Omega}_r^{sd}|n\rangle\langle n| \tag{6.62}$$

Hereafter, $\hat{H}^{eff} = \hat{H}^{Bloch,2}$, unless otherwise stated. Using the definitions

$$|\Psi_i\rangle = \sum_{l}^{l\in \mathrm{p}} X_l^i |l\rangle \quad \text{and} \quad \hat{\Omega}^{sd}|k\rangle = \sum_r \hat{\Omega}_k = \left[\sum_r^{r\in q^{sd}} T_r^k |r\rangle\langle k|\right]|k\rangle, \tag{6.63}$$

an element of the effective Hamiltonian, H_{ij}^{eff}, may be written,

$$\langle\Psi_i|\hat{H}^{Bloch,2}|\Psi_j\rangle =$$

$$\sum_{mn}^{\in \mathbf{p}} X_m^{i\dagger}\langle m|\hat{H}|n\rangle X_n^j + \sum_{mnkl}^{\in \mathbf{p}} \sum_r^{\in \mathrm{q}^{sd}} X_m^{i\dagger}\langle m|\hat{H}T_r^l|r\rangle\langle n|n\rangle X_n^j \tag{6.64}$$

The above assumes that the members of p are orthogonal; $\langle n|k\rangle X_k^i = X_n^i \delta_{nk}$.

Typically, members, $\{|p\rangle\}$, of subspace $\mathbf{p}$ are chosen such that they are eigenvectors of $\hat{H}_0$, i.e., $\langle m|n\rangle X_n^i = \delta_{mn}$, which simplifies the above expression further still;

$$H_{ij}^{eff} = \langle i|\hat{H}|j\rangle + \sum_r^{\in \mathrm{q}^{sd}} \langle i|\hat{H}T_r^j|r\rangle\langle j|j\rangle \tag{6.65}$$

The diagonal elements are the single state CASPT2 energies, and have contributions from both terms. The off diagonal elements ($i \neq j$) only have contributions from the second term, and are zero in the absence of the perturbation.

To examine the interactions between different states more closely it is useful to partition the Hamiltonian into blocks;

$$\hat{H} = \hat{P}\hat{H}\hat{P} + \hat{Q}\hat{H}\hat{Q} + \hat{P}\hat{H}\hat{Q} + \hat{Q}\hat{H}\hat{P}.$$

Two of the most common definitions and requirements in perturbation theories are

$$\hat{H} = \hat{H}^0 + \hat{V} \quad \text{and} \quad \hat{H}_0|m\rangle = \epsilon_m^0|m\rangle \quad m \in \mathrm{p}$$

which when combined with the third and fourth requirements gives

$$\langle m|\hat{V}|n\rangle = 0 \quad \text{and} \quad \langle m|\hat{H}|r\rangle = 0 \quad \forall m, n \in \mathrm{p}, \ \forall r \in \mathrm{q}^{sd} \tag{6.66}$$

suggests the following definitions of $\hat{H}^0$ and $\hat{V}$,

$$\hat{H}^0 = \hat{P}\hat{H}\hat{P} + \hat{Q}\hat{H}\hat{Q} \quad \text{and} \quad \hat{V} = \hat{P}\hat{H}\hat{Q} + \hat{Q}\hat{H}\hat{P} \tag{6.67}$$

which may be used to rewrite the expression for H_{ij}^{eff} in the perhaps more familiar form

$$H_{ij}^{eff} = \langle i|\hat{H}_0|j\rangle + \sum_{mn}^{\in \mathrm{p}} \sum_{r}^{\in \mathrm{q}^{sd}} \langle i|\hat{V}T_r^j|r\rangle\langle j|j\rangle. \tag{6.68}$$

In the single state case, the second term of (6.68) reduces to the expression for the second order energy found in (6.27). Furthermore, by comparing the definitions in (6.67) and that in (6.51), the connection between this expression and the generalized Bloch equation will hopefully become more apparent.

6.5 MRPT2 energy derivatives

The conditions specified in (6.66) clearly pertain to a very specific situation, but are still often employed as part of a method, albeit an indirect one, of expanding the basis used to describe the wavefunction.

This is particularly useful when calculating the linear response of the energy to the perturbation of a parameter about zero, e.g.,

$$\left.\frac{\delta E}{\delta \mathbf{R}}\right|_{\mathbf{R}=\mathbf{0}} \quad \text{where} \quad \hat{H} = \hat{H}^0[\mathbf{R}] + \hat{V}[\mathbf{R}] \quad \text{and} \quad \hat{V}[\mathbf{R}=0] = 0 \tag{6.69}$$

Where $\mathbf{R} = 0$ is just some equilibrium value for $\mathbf{R}$, i.e., the derivative is being taken about the point where the off diagonal blocks of the Hamiltonian are zero $\hat{H}_{PQ} = 0$.

In the many cases, notably where $\mathbf{R}$ is a nuclear coordinate, these gradients can be strongly influenced by the correlation with highly energetic orbitals not included within the active space, and MS-CASPT2 is a way of taking this into account. Provided the states in the model space are well separated the interaction between them due to the perturbation is often small enough, e.g., smaller than the sum total of their interaction with all states in the external space q, to be safely neglected.

Unfortunately, taking derivatives of the effective Hamiltonian with respect to a pertur-

bation parameter is not straight forward. Consider the derivative of (6.64)

$$\frac{\delta H_{ij}^{eff}}{\delta \mathbf{R}} = \frac{\delta}{\delta \mathbf{R}}\left[\sum_{mn} X_m^{i\dagger}\langle m|(\hat{H}^0+\hat{V})|n\rangle X_n^j\right]\frac{\delta}{\delta \mathbf{R}}+\left[\sum_{mn}^{\in \mathrm{p}}\sum_{r}^{\in \mathrm{q}^{sd}} X_m^{i\dagger}\langle i|(\hat{H}^0+\hat{V})T_r^n|r\rangle\langle n|n\rangle X_n^{j\dagger}\right]. \tag{6.70}$$

The dependence of T_r^j on $\mathbf{R}$ is complicated; if $|j\rangle$ is a CASSCF wavefunction, then T_r^j is dependent on the ci-coefficients and orbital coefficients used to define $|j\rangle$, which are dependent on $\hat{H}$, which is dependent on $\mathbf{R}$. This dependence of T_r^j on the j prevents the Hellman-Feynman theorem being used. An alternative might be to use the chain rule, and determine the derivatives of the ci and orbital coefficients with respect to $\mathbf{R}$, but this is not feasible in terms of computational cost. Instead a Lagrangian approach is adopted, which incorporates all the various constraints associated with the optimization of the T amplitudes, ci and orbital coefficients.

Before proceeding it is worth noting that any resulting theory should be invariant with respect to transformation in the reference space. Previously, it was stated that $\mathbf{X}_n^j$ can effectively be set to δ_{nj} by requiring that the model space is comprised of eigenvectors of the zeroth order Hamiltonian $\hat{H}^0$. However, this is somewhat problematic when dealing with degeneracies; the eigenstates are not uniquely defined. For example, in the case of a conical intersection two states are degenerate, but may have very different nuclear energy gradients. If these gradients are to be consistently defined for the two states there cannot be arbitrary rotations occurring between the different eigenvectors.

Extended multistate CASPT2 (XMS-CASPT2) handles this issue by defining the zeroth order Hamiltonian as

$$\hat{H}_{0,XMS} = \hat{P}\hat{f}\hat{P} + \hat{Q}\hat{f}\hat{Q}, \tag{6.71}$$

where $\hat{f}$ is the state averaged Fock operator defined by

$$\hat{f} = \sum_{cd} f_{cd} = \sum_{cd} \hat{a}_c^\dagger \hat{a}_d \left[h_{cd} + \sum_{ab} d_{ab}\frac{1}{2}(2J_{ab}^{ij} - K_{ab}^{ij})\right] \tag{6.72}$$

where a,b,c and d are molecular orbital indices, $J_{ab}^{cd} = (ij|ab)$ and $K_{ab}^{cd} = (ca|db)$ are two electron integrals, and d_{ab} are elements of the reduced, one-electron, state-averaged density matrix. The states $\{|p\rangle\}$ are unitary transformed into a set $\{|\tilde{p}\rangle\}$, which are eigenvectors of $\hat{H}_{0,XMS}$ with eigenvalues $\tilde{\epsilon}_p$;

$$|\tilde{p} = \sum_l |l\rangle U_{lp} \tag{6.73}$$

The state averaging ensures that the elements f_{cd}, and their derivative with respect to any perturbing parameter, are invariant under rotations between states. This will result in a consistent definition of the energy derivatives, even at conical intersections.

Another advantage is that choosing the zeroth order energies ϵ_i^0 to be the eigenvalues of the state averaged Fock operator greatly reduces the chances of a zero occurring in the denominator of the expression (6.35) used to calculate the T-amplitudes. One added complexity in XMS-CASPT2 is that the eigenvectors of $\hat{f}^{sa}$, defined by coefficients X_n^j, have an implicit dependence on $\mathbf{R}$. This can be handled by adding another constraint to the Lagrangian, but must be borne in mind. Hereafter, the XMS subscript on $\hat{H}_{0,XMS}$ will be omitted.

Bibliography

[1] T. Shiozaki. Bagel: Brilliantly advanced general electronic-structure library. *WIREs Computational Molecular Science*, 8:1331, 2018.

[2] J. A. Calvin and E. F. Valeev. "TiledArray: A general-purpose scalable block-sparse tensor framework".

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[4] Javascript object notation. urlhttps://www.json.org/. Accessed: 2018-06-29.