1 Overview

1.1 Purpose

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

$$\langle M|\hat{X}\hat{Y}...|N\rangle, \tag{1}$$

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

$$|N\rangle = \sum_I c_I^N |I\rangle. \tag{2}$$

There are three main types of terms the program can calculate. The simplest is :

$$\sum_{\substack{x_1x_2...\\ y_1y_2...\\ ...}} X_{x_1x_2...}Y_{y_1y_2...}... \sum_I \sum_J \langle I|a^\dagger_{x_1}a_{x_2}...a^\dagger_{y_1}a_{y_2}....|J\rangle c_I^{M\dagger}c_J^N \tag{3}$$

Which is just (1) written using second quantization; $X_{x_1,x_2...}$ is a representation of operator $\hat{X}$ in a basis of molecular orbitals.

It is also possible to calculate derivatives of (3) with respect to ci-coefficients, c_I^N;

$$\sum_{\substack{x_1x_2...\\ y_1y_2...\\ ...}} X_{x_1x_2...}Y_{y_1y_2...}... \sum_J \langle I|a^\dagger_{x_1}a_{x_2}...a^\dagger_{y_1}a_{y_2}....|J\rangle c_J^N \tag{4}$$

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

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

$$= \sum_{\substack{x_1x_2...\\ y_1y_2...\\ ...}} X_{x_1x_2...}Y_{y_1y_2...}... \sum_I \sum_J \langle I|a_{\omega_1}a_{\omega_2}...a^\dagger_{x_1}a_{x_2}...a^\dagger_{y_1}a_{y_2}....|J\rangle c_I^{M\dagger}c_J^N \tag{5}$$

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

The program is intended to be flexible; and instead of asking for specific perturbation theories

or properties, the user has the option of specifing algebraic expressions directly via the input file.

1.2 Structural outline

The program is split into three main components; an algebraic manipulator, an fci and tensor contraction library, and an task manager (make figure).

The algebraic manipulator should the input expression, and rearranges it into a series of expressions which are more suited to computational evaluation. To accomplish this it 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 mathematical operations which need to be evaluated. It is important to note that the algebraic manipulator is entirely symbolic manipulation, and does not handle any of the large data structures, e.g., civectors, matrices of molecular orbital indices used. 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 contains generic routines for calculating density matrices and their derivatives, and performing tensor contractions (and other various tensor manipulations). The tensor and FCI library are completely disctinct from the algebraic manipulator, and operate totally independently. The FCI library is designed such that it can calculate density matrices (and derivatives) or arbitrary order. In a similar vein, the tensor library can operate on tensor of arbitrary rank, with each dimension being an arbitrary and different size. At present both the FCI uses the dvec and determinant classes defined in bagel, but this should be phased out. Similarly, the tensor contraction makes use of the SMITH::Tensor_ class.

The final component of the program is the task manager which is intended to faciliate communication between the routines. This is necessary, as the neither the arithmetic component has now 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.

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

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 could be something like find the $\hat{T}^{LN}$, for all desired values of L and N, which satisfy

$$\sum_N \langle M|\hat{E}^\dagger_\Omega(\hat{H}-\epsilon_L)\hat{T}^{LN}|N\rangle + \langle M|\hat{E}^\dagger_\Omega\hat{H}|L\rangle = 0 \qquad \forall \quad M. \tag{6}$$

This equation for is broken down into a number of components, or *expression*s, which can be evaluated independently:

$$\sum_N \langle M|\hat{E}^\dagger_\Omega(\hat{H}-\epsilon_L)\hat{T}^{LN}|N\rangle + \langle M|\hat{E}^\dagger_\Omega\hat{H}|L\rangle \tag{7}$$

The *equation* contains a number of *expression*s, along with information regarding how they relate to one another and the solution to the equation. In an expression of the above type 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, a given equation will have several similar expressions, which only different from one another by the values of the indexes or the ranges of the index summation.

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 (7) cannot be represented by a single term object, and instead requires two: is a valid term, whilst

$$\sum_N \langle M|\hat{E}^\dagger_\Omega(\hat{H}-\epsilon_L)\hat{T}^{LN}|N\rangle \tag{8}$$

$$\langle M|\hat{E}^\dagger_\Omega\hat{H}|L\rangle \tag{9}$$

Without this seperation (9) would be summed over N times, despite the fact N does not occur. Whilst this constraints seems needless and annoying there are good practical reasons for it, mainly 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 (8) would be broken down into $2N$ *brakets*:

$$\langle M|\hat{E}^\dagger_\Omega\hat{H}\hat{T}^{LN}|N\rangle \tag{10}$$

$$\langle M|\hat{E}^\dagger_\Omega(-\epsilon_L)\hat{T}^{LN}|N\rangle \tag{11}$$

Within a *braket* the values of all the indexes are specified. A single *braket* consists of a bra and a ket, surrounding some operator formed from the product of an arbitrary number of other

operators. The individual *braket*s comprising a *term* are never[1] 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-quatized 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 describe components are the targets upon which the algebraic manipulation routines act to generate the algebraic task list. I shall now describe the proceedure, and introduce further classes as necessary.

It is important to note that due to the nature of the program I shall not describe the program in the order in which it is executed, but instead start from the simplest operations it performs, and then gradually build up to show how these can be linked together to generate the task lists necessary for solution of the equations.

2.1 Operator definition, formation, blocking and symmetry

Key classes: *TensOp/MultiTensOp, CtrTensorPart/CtrMultiTensorPart, RangeBlock/SplitRangeBlock*

2.2 Braket Formation

Key classes: *Expression, BraKet, GammaGenerator, GammaInfo, StateInfo, CIVecInfo*

2.3 Algebraic Task List generation

Key classes: *Expression, GammaGenerator, AContribInfo, CtrTensOp/CtrMultiTensOp*

2.4 Equation solver

Key classes: *Expression, TensOp/MultiTensOp, SystemInfo, Computer classes,*

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