## 1 background

In experiments we measure "observable values". Denote one such variable as $x$. To be clear $x$ is a number (not a matrix, operator, function etc.). Each observable corresponds to an operator $\hat{X}$. The hat $\hat{}$ means this is an operator[1].

Experimental systems are said to be in different "states". We try to describe state $M$ with a "wavefunction" $\Psi_M$ . Note this wavefuction describes the behavious of all the particles in the system (electrons in this case). The basic properties of these states are defined in the StateInfo object, and also in CASPT2_ALT.

The wavefunction is described as a linear combination of other wavefunctions:

$$\Psi_M = \sum_I c_I \Phi_I, \tag{1}$$

Each $\Phi$ corresponds to a different configuration of N particles, and the are $c_I$ are coefficients defining how much each configuration contributes to the total wavefunction,$Psi$[2]. It is very important to note that the total wavefunction $\Psi$ *cannot* usually be described by any single configuration of particles.

To get the value of observable $x_M$ associated with state $\Psi_M$ we evaluate.

$$\langle \Psi_M | \hat{X} | \Psi_M \rangle \tag{2}$$

The $\langle|$ part is the "Bra" member of the GammaInfo object, whilst the "Ket" is the $|\rangle$. [3].

The $\langle||\rangle$ is associated with the method used to evaluate these terms. Do not worry about it, just bear in mind that $\hat{X}|\Psi_M\rangle$ is *not* $\hat{X}$ times $\Psi_M$; neither $\hat{X}$ nor $\Psi$ have any defined numerical representation. The idea is that provided we represent $\hat{X}$ and $\Psi$ in a manner which preserves their logical structure and relationship to one another, the result of the above expression is the same.

This has informed how the program is written; in general all classes with info in their name contain *no* actual data about the representation of the wavefunction or operators, they only contain information about their structure, and instructions how

[1]Instantiations of the TensOp class contain information about the operators
[2]The array with elements $c_I$ is the CIVec object
[3]These obtained results are stored in the scalar_results_map

to best evaluate terms which involve them. Determining these instructions is what is done in the functions in the gamma_generator, TensOp, MultiTensOp, CtrTensOp and CtrMultiTensOp classes, whilst all the actual work is carried out in the more ???_computer and ????_arithmetic classes.

To understand how each of these classes relate to one another we need to discuss the physics a bit more.

Most observables can be boiled down to interactions between pairs of electrons. This means we can rewrite the above expression as

$$\sum_{\substack{electron\\pairs}} \langle \Psi_M | \hat{X}^{2el} | \Psi_M \rangle \tag{3}$$

Where $\hat{X}^{2el}$ is a two electron operator, i.e., we calculate the contributions to the observable from each pair of electrons seperately and then sum them up. In the code I refer to these are BraKets, and information about them is stored in the BraKet class.

However, as mentioned above, the wavefunction $\Psi_M$ does not correspond to single cofiguration of particles, so it is not clear how the summation over electron pairs can be accomplished. In order to do so we must use the representation defined in equation (1). Substituting equation (1) into (3) we obtain:

$$\sum_{\substack{electron\\pairs}} \sum_{IJ} c_I^M \langle \Phi_I | \hat{X}^{2el} | \Phi_J \rangle c_J^M \tag{4}$$

Now, instead of calculating a single interaction between two wavefunctions with ill-defined particles, we calculate a lot of interactions between particles with well defined particles. This can now be rewritten:

$$\sum_{ijkl} \sum_{IJ} c_M^I \langle \Phi_I | \hat{a}_i^\dagger \hat{a}_j^\dagger \hat{a}_k \hat{a}_l | \Phi_J \rangle c_J^M X_{ijkl} \tag{5}$$

This is a bit nasty looking, but it is the basic kind of expression I am evaluating, so it is worth going through properly:

- Here $\hat{a}_l$ "annihlates" particle $l$ in $\Phi_M$ whilst $\hat{a}_i^\dagger$ "creates" particle $i$. So we annihilate two particles in $\Phi_J$ and then create two in $\Phi_I$. In other words we are exciting from I to J. These $\hat{a}$ operators are referred to as aops in the code. They have no numerical repsentation, but describe how configurations relate to one another.

- The $X_{ijkl}$ are elements of a tensor (here, a 4-index array of floats) which describes how the operator $\hat{X}$ influences the interaction of particles $i$, $j$, $k$ and $l$. [4]

- The matrix with elements $\sum_{IJ} c_I^M \langle \Phi_I | \hat{a}_i^\dagger \hat{a}_j^\dagger \hat{a}_k \hat{a}_l | \Phi_J \rangle c_J^M$ is referred to as the (2-electron reduced) density matrix, and defines how the different configurations interact. For various reasons, I will refer to them as gammas, here and in the code. [5].

Writing expressions like this has several advantages due to properties of the configurations $Phi_J$ and the operators $\hat{a}$. By definition

$$\langle \Phi_I | \Phi_J = \begin{cases} 1 & \text{if } \Phi_I | \Phi_J \\ 0 & \text{otherwise .} \end{cases} \tag{6}$$

In otherwords, if $\Phi_I | \Phi_J$ aren't the same, the term vanishes. However, in equantion there are several particle creation and destruction operators between the two $\Phi_I | \Phi_J$, and these operators act to transform configurations into one another. For example suppose I have a configuration $\Phi(1,2,3)$, which contains particles 1,2, and 3, and another configuration $\Phi(1,2,4)$ containing particles 1,2 and 4. We can use the creation and annhiliation operators, $\hat{a}_4$ and $\hat{a}_3^\dagger$ to transform between these two configurations;

$$|\Phi(1,2,3)\rangle = \hat{a}_3^\dagger \hat{a}_4 | \Phi(1,2,4)\rangle. \tag{7}$$

What's happening here is that the operator $\hat{a}_4$ destroys particle 4 in $\Phi(1,2,4)$, and then operator $\hat{a}_3^\dagger$ fills this whole with particle 3. So if we now revist equation (6) we can now get the relation

$$\langle \Phi(1,2,4) | \hat{a}_i^\dagger \hat{a}_j \Phi(1,2,4) = \begin{cases} 1 & \text{if } i = 3 \quad \text{and} \quad j = 4 \\ 0 & \text{otherwise .} \end{cases} \tag{8}$$

As we can see now, which terms contribute to depend on which configurations $\Phi$ we are using to define the total wavefunciton $\Phi$, and which particles our operator $\hat{X}$ acts on. For example, we may know that our observable is only significanlty influenced by those electrons in the outermost shells, enabling us to skip many terms in the

[4]These tensors are stored in TensOp_data_map, note that the corresponding in TensOp object does not contain this data, it only has a name so it can be fetched from the map.

[5]The elements of gammas are stored in Gamma_data_map, are calculated in the gamma_computer class. Which gammas we need is for a given expressions is determined in the gammagenerator class, and information about the a given gamma is defined in a gamma_info object.

summaiton. Furthermore, our operator $\hat{X}$ may have internal symmetries, reducing the number of unique elements $X_{ijkl}$ we need to store.

The purpose of my code is to take advantage of these symmetries and range constraints. It should do it automatically, and for any number of operators, whose symmetries and range constraints are specified by functions input by the used. The 4-index term above is not actually problematic. But to calculate the things I am interested in I have a dozen or so terms looking like this

$$\sum_{IJ} c_{NI} \langle \Phi_N | \hat{a}_i \hat{a}_j \hat{a}_k^\dagger \hat{a}_l^\dagger \hat{a}_m^\dagger \hat{a}_n \hat{a}_o^\dagger \hat{a}_p^\dagger \hat{a}_q \hat{a}_s | \Phi_J \rangle c_{JM} X^*_{klij} Y_{mn} Z_{opqs} \tag{9}$$

We can never store the relevant arrays; they are too large. However, the creation and annhilation operators possess a permutation symmetry which enables us to rewrite the above 10-index term, as several (often 100 or so) terms with fewer indexes: =

$$\sum_{\nu}^{\nu\in\{\Omega_\nu^{(1)}\}} \sum_{ijklmnopqr} \gamma_{k'l'm'n'o'p'q'r'} X^*_{klij} Y_{mn} Z_{opqs} \delta_{i'j'} \tag{10}$$

$$+ \sum_{\nu}^{\nu\in\{\Omega_\nu^{(2)}\}} \sum_{ijklmnopqr} \gamma_{m'n'o'p'q'r'} X^*_{klij} Y_{mn} Z_{opqs} \delta_{i'j'} \delta_{k'l'} \tag{11}$$

$$+ \sum_{\nu}^{\nu\in\{\Omega_\nu^{(3)}\}} \sum_{ijklmnopqr} \gamma_{o'p'q'r'} X^*_{klij} Y_{mn} Z_{opqs} \delta_{i'j'} \delta_{k'l'} \delta_{m'n'} \tag{12}$$

$$+ \sum_{\nu}^{\nu\in\{\Omega_\nu^{(4)}\}} \sum_{ijklmnopqr} \gamma_{q'r'} X^*_{klij} Y_{mn} Z_{opqs} \delta_{i'j'} \delta_{k'l'} \delta_{m'n'} \delta_{o'p'} \tag{13}$$

$$\sum_{\nu}^{\nu\in\{\Omega_\nu^{(5)}\}} \sum_{ijklmnopqr} X^*_{klij} Y_{mn} Z_{opqs} \delta_{i'j'} \delta_{k'l'} \delta_{m'n'} \delta_{o'p'} \delta_{q'r'} \tag{14}$$

where

$$\gamma_{ijkl} = \sum_{IJ} c_I^M \langle \Phi_I | \hat{a}_i^\dagger \hat{a}_j^\dagger \hat{a}_k \hat{a}_l | \Phi_J \rangle c_J^M \tag{15}$$

The primed indices are obtained by applying the permutation operation $\Omega$ to the set of unprimed indexes, i.e.,

$$\{i'j'k'l'm'n'o'p'q'r'\} = \Omega\{ijklmnopqr\}. \tag{16}$$

The outer summation in equation 14 runs over different permutations (there is a way of figuring out which ones you need). So this outer summation is a sum over possible relabellings of the indexes, not values of the indexes themselves.

The $\delta_{x'y'} = 0$ if $x' \neq y'$ and 1 otherwise. Hence, the permutation in the outer summation combined with the delta functions is expresses that each different term in the summation involves the same baisc tensors, but with different indexes set equal. For example, in the four index term;

$$\sum_{\nu}^{\nu \in \{\Omega_{\nu}^{(5)}\}} \sum_{ijklmnopqr} \gamma_{o'p'q'r'} X^{*}_{klij} Y_{mn} Z_{opqs} \delta_{i',j'} \delta_{k'l'} \delta_{m'n'} \tag{17}$$

one contribution might have $i' = k$ , $j' = k$, $k' = q$, $l' = s$, $m' = j$ and $n' = m$;

$$\sum_{ijklmnopqr} \gamma_{ilnr} X^{*}_{klij} Y_{mn} Z_{opqs} \delta_{ko} \delta_{qs} \delta_{jm} = \sum_{ilnr} \gamma_{ilnr} \sum_{jkmoqs} X^{*}_{klij} Y_{mn} Z_{opqs} \delta_{ko} \delta_{qs} \delta_{jm} = \tag{18}$$

So in equation (14) some of the indexes of X, Y and Z are being forced to be the same, which reduces the dimensions of everything involved and splits things up into two smaller operations. Setting two indexes the same like this is referred to as a contraction. In the code I often refer to pair of numbers $\{i, j\}$ as a contraction, a ctr or a delta.

What my code does is reorders the code to reduce the number of distinct combinations of contractions, whilst also trying to reduce the dimension of the problem as much as possible. It then generates an list of operations which need to be performed to evaluate these expressions; the difficulty is that there are numerous different orders in which these expressions can be evaluated, some of which are much more efficient than others. So basically the code contains an automatic equation rearranger. This is necessary, as the number of terms in the rearranged expression is large, so it cannot be done by hand for all different quantities.

How to best rearrange the terms is dependent on the range constraints on the indexes and the internal symmetries of the operators. Getting it to take advantage of this is the hard part. Furthermore, there are often lots of different BraKets being calculated. The contributions to each BraKet are different but ultimately many can be merged together, and some can be known to be equivalent or vanish due to properties of the system. Accounting for this can be tricky.

The meat of all this is rearrangement is done in CtrTensOp and gamma_generator.

Information about a block of tensor is stored in a CtrTensOp object (CTP = contracted tensor part). For example, $X_{ijkl}$ are the elements of a CTP. If I have multiple tensors in one expression ($XYZ$), I stored this information in CtrMultiTensOp object (CMTP = contracted multi tensor part). For example, $T_{qrst}X_{ijkl}$ are elements of a CTP. Contracting indexes on two different tensors takes us from a CMTP to a CTP, e.g., $\sum_t T_{qrst}X_{ijkl}\delta_{ti} = A_{qrsjkl}$. The effect of contracting indexes on different tensors effectively fuses those tensors so they can no longer be seperated, and their symmetries and contraints must be combined. Generally speaking I start with a CMTP object, and then must contract it to get a CTP object, and then contract this in lots of different ways. This is done in a full_contract in CtrTensOp.

It is important to note that the tensors are split up into blocks corresponding to different ranges (these range blocks are nothing to do with the parallelism). Each CTP and CMTP object only corresponds to a certain range block. So there are several different CMTP objects for every tensor. A list of all possible ranges for a given tensor is specfied in the all_ranges member of the TensOp and MultiTensOp classes.

Which terms are includes in equation14 are dependent on the range blocks. The list of contributing terms, with their range blocks, is determined in gamma generator. Which terms need to be calculated (there may be intermediate terms, some terms can be combined etc.,) is determined in the full contract functions in CtrTensOp.

CtrTensOp generate a task list, which is executed by expression computer. Expression computer then calls tensor arithmetic, and gamma computer, The former contains the acual linear algebra routines for contracting the tensors, whilst the latter contains routines for calculating the $\gamma_{ijkl} = \sum_{IJ} c_{NI}\langle\Phi_N|\hat{a}_i^{\dagger}\hat{a}_j\hat{a}_k^{\dagger}\hat{a}_l|\Phi_J\rangle c_{JM}$ type terms.

In conclusion, the basic proceedure is

- Things start in src/smith/caspt/CASPT2_ALT.cc, but most of the code is in /src/smith/wicktool .
- First instantiate a Sys_Op object which contains loads of background information.
- Put information about the wavefunction (states_info objects) and operators (TensOp objects) into the Sys_Op object.
- TensOp will define all possible sub blocks and contractions, and identify equivalencies, but will not do any actual calculations.

- Instantiate a Expression_Info object which contains the actual mathematical expressions I want to calculate.
- Expression Info will then generate some BraKet objects, each of which corresponds to a term like .
- The Bracket will then run the gamma generator to see which terms contribute.
- The gamma_generator will produce the gamma_info objects, and also CtrTenOp objects containing information about the terms we need to calculate.
- Full contract (in CtrTensOp) then runs on each of these CTP and CMTP objects, which will provide a list of the linear algebra operations we need to do.
- Each gamma object will have a seperate task list (ACompute_list), which contains all the contracted operators which need to be contracted with that gamma (n.b. matrix multiplication is a special case of contraction). Each of these lists is put into the G_to_A_map; the key is the name of the gamma matrix, whilst the mapped object is the ACompute list.
- Then the initial data about the wavefunction and operators to generate an expression_computer member of Sys_Op.
- This expression_computer object is then used to execute the various lists as described above.