1 Tensor Operator (TensOp)

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

$$\hat{Q} = \sum_{i}^{R^i} \sum_{j}^{R^j} \sum_{k}^{R^k} \sum_{l}^{R^l} \hat{a}_i^\dagger \hat{a}_j^\dagger \hat{a}_k \hat{a}_l Q_{ijkl}. \tag{1}$$

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.,

$$R^i = r_1^i \cup r_2^i \cup r_3^i \cup ... = \bigcup_{\mu}^{N_{block}} r_\mu^i \tag{2}$$

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.

$$B^{\mu\nu\xi\chi} := \{\{i,j,k,l\} \mid i \in\ r_\mu^i,\ j \in r_\nu^j,\ k \in r_\xi^k,\ l \in r_\chi^l\}. \tag{3}$$

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

$$Q_{ijkl} \in \mathbf{Q}^b \quad \text{iff} \quad \{i,j,k,l\} \in b \tag{4}$$

using this notation we can rewrite (1) as

$$\hat{Q} = \sum_{b}^{\{B\}} \sum_{\{i,j,k,l\}}^{b} \hat{a}_i^\dagger \hat{a}_j^\dagger \hat{a}_k \hat{a}_l Q_{ijkl}. \tag{5}$$

For each operator $\hat{Q}$, we store a single *TensOp* object, which corresponding 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.

1.1 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

$$Q^{b1}_{ijkl} = Q^{b2}_{ijlk}\rho, \tag{6}$$

where $b1 = B^{\xi\xi\chi\xi}$, $b2 := B^{\xi\xi\xi\chi}$, and ρ is some constant factor. This is a special case were the ranges r^ξ and r^χ have the same length:

$$\mathrm{card}(r^\xi) = \mathrm{card}(r^\chi). \tag{7}$$

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} preceeded 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.,

$$\mathrm{card}(r^\nu) \gg \mathrm{card}(r^\chi) \tag{8}$$

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

$$X_{mn} = \sum_{ijkl} Q^{b1}_{ijkl} R_{ijklmn} + \sum_{ijkl} R^{b3}_{ijlkmn} Q^{b2}_{ijkl}. \tag{9}$$

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 it the basic ideas governing how symmetry is handled can be discussed now.
The set of all the blocks of a tensor, B, can be generated by applying the appropriate trans-

formations to some subset of these blocks;

$$S : \tilde{B} \rightarrow B \quad \tilde{B} \subset B$$

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

$$b = t\tilde{b} \tag{10}$$

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 original block (in the full set, B), and the unique range block (from the minimal set, $\tilde{B}$). Loosely speaking, the algebraic manipulation routines just replace all appearances of the original blocks with unique 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.
It is important to note that the tensor blocks themselves are

1.2 MultiTensOp

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

$$\langle I|\hat{lambda}\hat{H}\hat{T}|J\rangle. \tag{11}$$

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.,

$$lambda^{b1} H^{b1} T^{b1} = T^{b1} H^{b1} \lambda^{b1}. \tag{12}$$

1.3 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,

$$[HT]_{ilxy}^{(b1,b2,jz,kw)} = \sum_{jz} \sum_{kw} H_{ijkl}^{b1} T_{wxyz}^{b2} \delta jz \delta kw. \tag{13}$$

These correspond to the CtrTensOp object; note that once the contraction between H and T has occured, 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 (**??**) above may well be replaced with something like this in the algebraic task list

$$[HT]_{ilxy}^{(b1,b2,jz,kw)} \mapsto \sum_{ix} \sum_{ly} \rho(H_{jilk}^{\tilde{b1}} T_{yzwx}^{\tilde{b2}} \delta ix \delta ly). \tag{14}$$

Here the orignal 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.

1.4 Contraction procedure

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

$$A_{kwzm} = [BCD]_{kwzm}^{(jy,lx,no,pi)} = \sum_{jy} \sum_{no} \sum_{lx} \sum_{ip} B_{ijkl} C_{mnop} D_{wxyz} \delta_{jy} \delta_{no} \delta_{lx} \delta_{ip} \tag{15}$$

Would be obtained through the following sequence of operations:

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

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

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

$$\text{Task3} \;:\quad [BCD]^{(ip,no,jy)}_{klmwxq} = \sum_{jy} [BC]^{(ip,no)}_{jklm} D_{wxqy} \delta_{jy}.$$

$$\text{Task4} \;:\quad [BCD]^{(ip,no,jy,lx)}_{kmwq} = \sum_{lx} [BC]^{(ip,no)}_{klmwxq} D_{wxqy} \delta_{lx}.$$

The task list is generated from the information stored in the CtrMultiTensOp corresponding to $[BCD]^{(jy,lx,no,pi)}_{kwzm}$, and is done in accordance with the following principles:

Principle 1 : Contractions between indexes on the same tensor are performed first; this is to keep the rank of the largest tensor to be stored to a minimum. For example, if the first two tasks were reversed:

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

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

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.

The ordering of the contractions is always the same, e.g., the task lists for $[BCD]^{(jy,lx,no,pi)}_{kwzm}$ and $[BCD]^{(kw,lz,no,pi)}_{jxym}$ will always perform the contraction over (n, o) first, followed by the contraction over (i, p). This means that the $[BC]^{(ip,no)}_{jklm}$ can be reused, and the first two tasks need not be repeated.

Principle 2 : Item Each of these "intermediate" tensors is given a use number; which increases and decreases as the operations are performed, so that it can be deleted once it is no longer needed[2].

Principle 3 : A canonical ordering is defined for the indexes, and any transpositions are defined relative to this canonical order. e.g., tensor B always appears to the left of C, and index i always appears to the left of j.

Principle 4 : Further to the above point, the contraction task list is performed on using the unique blocks (blocks from $\tilde{B} \subset B$). The transformations connecting the unique blocks to the original blocks have two components, a multiplicative factor [3], and an index rearrangement. Rather than transforming the tensor blocks and then executing the task list, the task list is constructed so that the indexes over which the contractions are performed are altered to reflect the necessary transformations, and only once the contraction list is executed are the necessary transpositions performed. For example, suppose we would like to calculate

$$A_{kwzm} = [BCD]^{((b1,b2,b3),(jy,lx,mo,kp))}_{inwz} = \sum_{jy} \sum_{no} \sum_{lx} \sum_{ip} B_{ijkl} C_{mnop} D_{wxyz} \delta_{jy} \delta_{no} \delta_{lx} \delta_{ip} \qquad (16)$$

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

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

whilst taking advantage of the following symmetry operations:

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

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

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

The following tasks would be performed,

$$\text{Task1} \;\; : \;\; [C]^{(\tilde{b2},no)}_{mp} = \sum_{no} C^{\tilde{b2}}_{mnop}\delta_{no} \qquad \delta_{mo} \rightarrow \delta_{no} \; .$$

$$\text{Task2} \;\; : \;\; [BC]^{((\tilde{b1},tilde_{b2}),(ip,no))}_{jklm} = \sum_{pi} B_{ijkl}[C]^{(no)}_{mp}\delta_{ip} \qquad \delta_{kp} \rightarrow \delta_{ip} \; .$$

$$\text{Task3} \;\; : \;\; [BCD]^{(ip,ly,no)}_{klmwxz} = \sum_{ly} [BC]^{(ip,no)}_{jklm} D_{wxyz}\delta_{ly}. \qquad \delta_{lx} \rightarrow \delta_{ly} \; .$$

$$\text{Task4} \;\; : \;\; [BCD]^{(ip,ly,mx,no)}_{klwz} = \sum_{mx} [BC]^{(ip,no)}_{klmwxz} D_{wxyz}\delta_{mx}. \qquad \delta_{jy} \rightarrow \delta_{mx} \; .$$

$$\text{Task5} \;\; : \;\; [BCD]^{((b1,b2,b3),(jy,lx,mo,kp))}_{inwz} = swap(0,1)[BCD]^{(ip,ly,mx,no)}_{klwz}$$

where changes to contractions versus those that would be performed on the untransformed blocks are noted on the right. Note that by waiting until the last possible moment to apply the symmetry transformations to the blocks, only one block transposition is required (not quite, fix this), and it becomes possible to reuse the $[BC]^{((\tilde{b1},tilde_{b2}),(ip,no))}_{jklm}$ obtained from earlier contraction lists.

Principle 5 : 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, and also because different task lists which use the same tensor may involve different factors.