Calculation of derivative properties in XMS-CASPT2

Peter John Cherry
Shiozaki group meeting
October 3rd 2017

Calculation of matrix elements for many electron operators

- Need a generic approach for evaluating terms of the form

$$\sum_{\substack{ijkl\\wxyz}} \sum_{J} T^{\dagger}_{ijkl} g_{wxyz} q_{uv} \langle I | a_i a_j a^{\dagger}_k a^{\dagger}_l a^{\dagger}_w a^{\dagger}_x a_y a_z a^{\dagger}_u a_v | J \rangle c_J$$

- SMITH3 generates code of evaluating these integrals.

- Jae developed and implemented and algorithm such that the most costly term is

$$\sum_{ijklmn}^{act} \sum_{J} \langle I | a^{\dagger}_i a^{\dagger}_j a^{\dagger}_k a_l a_m a_n | J \rangle c_J A_{ijklmn}$$

- This is highly efficient, but unfortunately, it is not as well suited to the cases with spin-flipping interactions.

Outline of new approach

- Aim to produce routine which can calculate arbitrary BraKet:

$$\langle M|\hat{A}\hat{B}\hat{C}....|N\rangle$$

- Should take advantage of symmetries, and range constraints.

- The program uses the following information to construct an object associated with each operator:

    - Number of indices.
    - Position of creation and annihilation operators.
    - Range of indices.
    - List of symmetry functions and constraints.
    - If the operator couples spin and spatial components.

- A braket object is then built from a vector of such objects.

- The program then uses this information to build a list of operations (tensor contractions and additions) which need to be performed.

Distinction from current approach

- Cannot calculate CI-derivatives for four component wavefunctions using the current approach.
- New approach should incorporate symmetry more effectively, particularly symmetries associated with spin.
- New approach repeatedly reshuffles the indexes in order to take advantage of constraints on the indices.
- Final expressions are not normal ordered, and a different algorithm is used for evaluating the CI derivatives.
- Using a generic algorithm, not generated code.

Outline of method

Step 1 : Get operator information.

- Split input tensors up into blocks.
- Generate all possible contractions of these blocks.
- Identify which blocks/contracted blocks are equivalent.

Step 2 : Manipulate indexes.

- Rearrange indexes to normal order, applying range constraints.
- Put indexes into alternating order and group common indexes.

Step 3 : Generate computational proceedure.

- Merge all contributions from different brakets.
- Generate a task list to obtain contracted A-tensors.

Step 4 : Execute computational proceedure.

- Contract A-tensors with gamma matrices.

Step 5 : Loop over spin sectors and states.

- Repeat for all necessary spin sectors.

Outline of method

Step 1 : Get operator information.

- Split input tensors up into blocks.
- Generate all possible contractions of these blocks.
- Identify which blocks/contracted blocks are equivalent.

Step 2 : Manipulate indexes.

- Rearrange indexes to normal order, applying range constraints.
- Put indexes into alternating order and group common indexes.

Step 3 : Generate computational proceedure.

- Merge all contributions from different brakets.
- Generate a task list to obtain contracted A-tensors.

Step 4 : Execute computational proceedure. ← Working here now

- Contract A-tensors with gamma matrices.

Step 5 : Loop over spin sectors and states.

- Repeat for all necessary spin sectors.

Outline of method

Step 1 : Get operator information.

- Split input tensors up into blocks.
- Generate all possible contractions of these blocks.
- Identify which blocks/contracted blocks are equivalent.

Step 2 : Manipulate indexes.

- Rearrange indexes to normal order, applying range constraints. $\longleftarrow$ Problem area.
- Put indexes into alternating order and group common indexes.

Step 3 : Generate computational proceedure.

- Merge all contributions from different brakets.
- Generate a task list to obtain contracted A-tensors. $\longleftarrow$ Problem area.

Step 4 : Execute computational proceedure. $\longleftarrow$ Working here now

- Contract A-tensors with gamma matrices.

Step 5 : Loop over spin sectors and states.

- Repeat for all necessary spin sectors.

Generation of contraction task list

- Most terms are too complicated to evaluate directly, e.g.,

$$\sum_{\substack{ijkl\\wxyz\\uv}} \sum_{J} T^{\dagger}_{ijkl} g_{wxyz} q_{uv} \langle I | a_i a_j a^{\dagger}_k a^{\dagger}_l a^{\dagger}_w a^{\dagger}_x a_y a_z a^{\dagger}_u a_v | J \rangle c_J$$

- Reordering indexes to represent as sum of contractions:

$$\rightarrow \sum_{\substack{ijkl\\wxyz\\uv}} \gamma^{I}_{ijklwxyzuv} A_{ijklwxyzuv} + \sum_{abcdefgh} \gamma^{I}_{abcdefgh} A_{abcdefgh}$$

$$+ \sum_{abcdef} \gamma^{I}_{abcdef} A_{abcdef} + \sum_{ab} \gamma^{I}_{ab} A_{ab}$$

Generation of contraction task list

- Each A-tensor is defined by the original operators, $T^{\dagger}_{ijkl}$, g_{wxyz}, q_{uv} , and a set, X , of contraction indices and factors:

$$A_{abcd} = \sum_{x}^{x \in X} \sum_{efghmn} \delta_{ef}\delta_{gh}\delta_{mn}T^{\dagger}_{ijkl}g_{wxyz}q_{uv}$$

$$X = \{(\{(a_1, b_1), (c_1, d_1), (e_1, f_1)\}, s_1), (\{(a_2, b_2), (c_2, d_2), (e_2, f_2)\}, s_2), ...\}$$

- Each contribution, A^x , to the A-tensor is obtained by performing a recursive sequence of operations:

$$A^x_{abcd} = B(A^x_{abcdefgh}, (g, h)) =$$

$$= B(B(A^x_{abcdefgh}, (g, h)), (e, f)) = B(B(B(A^x_{abcdefgh}, (g, h)), (e, f)), (c, d))$$

Generation of contraction task list

- The sequence of the operations is the task list.
- How the task list is built can significantly influence the computational cost.
- Currently using the following tactics to reduce cost:
  - Standardize order in which contractions are performed (facilitates term reuse).
  - Reorder indexes so that simlar indexes are together (facilitates term reuse).
  - Perform all intra-tensor contractions before any inter tensor contractions (reduces maximum dimension of tensor to be contracted).
  - Never store product tensors.

Contraction tree

$$T^{\dagger}_{ijkl}g_{wxyz}q_{uv}$$

Contraction tree

$$\begin{tikzcd}
 & T^{\dagger}_{ijkl}g_{wxyz}q_{uv} \arrow[dl, "\delta_{lz}"'] \arrow[dr, "\delta_{ky}"] & \\
A_{ijkwxyuv} & & A_{ijlwxzuv}
\end{tikzcd}$$

Contraction tree

$$\begin{tikzcd}
 & & T^{\dagger}_{ijkl} g_{wxyz} q_{uv} \arrow[dl, "\delta_{lz}"'] \arrow[dr, "\delta_{ky}"] & & \\
 & A_{ijkwxyuv} \arrow[dl, "\delta_{iw}"'] \arrow[d, "\delta_{kv}"'] & & A_{ijlwxzuv} \arrow[d, "\delta_{iw}"] \arrow[dr, "\delta_{jx}"] & \\
A_{jkxyuv} & A_{ijwxyu} & & A_{jlxzuv} & A_{ilwzuv}
\end{tikzcd}$$

Contraction tree

$$\begin{tikzcd}
 & & & T^{\dagger}_{ijkl} g_{wxyz} q_{uv} \arrow[dl, "\delta_{lz}"'] \arrow[drr, "\delta_{ky}"] & & & & \\
 & & A_{ijkwxyuv} \arrow[dl, "\delta_{iw}"'] \arrow[d, "\delta_{kv}"'] & & & A_{ijlwxzuv} \arrow[d, "\delta_{iw}"] \arrow[dr, "\delta_{jx}"] & & \\
 & A_{jkxyuv} \arrow[dl, "\delta_{jx}"'] \arrow[d, "\delta_{ky}"'] & A_{ijwxyu} \arrow[d, "\delta_{jx}"'] \arrow[dr, "\delta_{yu}"'] & & & A_{jlxzuv} \arrow[dl, "\delta_{ju}"] \arrow[d, "\delta_{jx}"] & A_{ilwzuv} \arrow[d, "\delta_{iw}"'] \arrow[dr, "\delta_{zu}"] & \\
A_{kyuv} & A_{jxuv} & A_{iwyu} & A_{ijwx} & A_{lxzu} & A_{lzuv} & A_{lzuv} & A_{ilwv}
\end{tikzcd}$$

Contraction tree

$$\begin{tikzcd}
 & & & T^{\dagger}_{ijkl} g_{wxyz} q_{uv} \arrow[dl, "\delta_{lz}"'] \arrow[drr, "\delta_{ky}"] & & & & \\
 & & A_{ijkwxyuv} \arrow[dl, "\delta_{iw}"'] \arrow[d, "\delta_{kv}"'] & & & A_{ijlwxzuv} \arrow[d, "\delta_{iw}"] \arrow[dr, "\delta_{jx}"] & & \\
 & A_{jkxyuv} \arrow[dl, "\delta_{jx}"'] \arrow[d, "\delta_{ky}"'] & A_{ijwxyu} \arrow[d, "\delta_{jx}"'] \arrow[dr, "\delta_{yu}"] & & & A_{jlxzuv} \arrow[dl, "\delta_{ju}"] \arrow[d, "\delta_{jx}"] & A_{ilwzuv} \arrow[d, "\delta_{iw}"'] \arrow[dr, "\delta_{zu}"] & \\
A_{kyuv} & A_{jxuv} & A_{iwyu} & A_{ijwx} & A_{lxzu} & \underline{A_{lzuv}} & \underline{A_{lzuv}} & A_{ilwv}
\end{tikzcd}$$

Generation of contraction task list

- Each A-tensor is defined by the original operators, $T^{\dagger}_{ijkl}$, g_{wxyz}, q_{uv} , and a set, X , of contraction indices and factors:

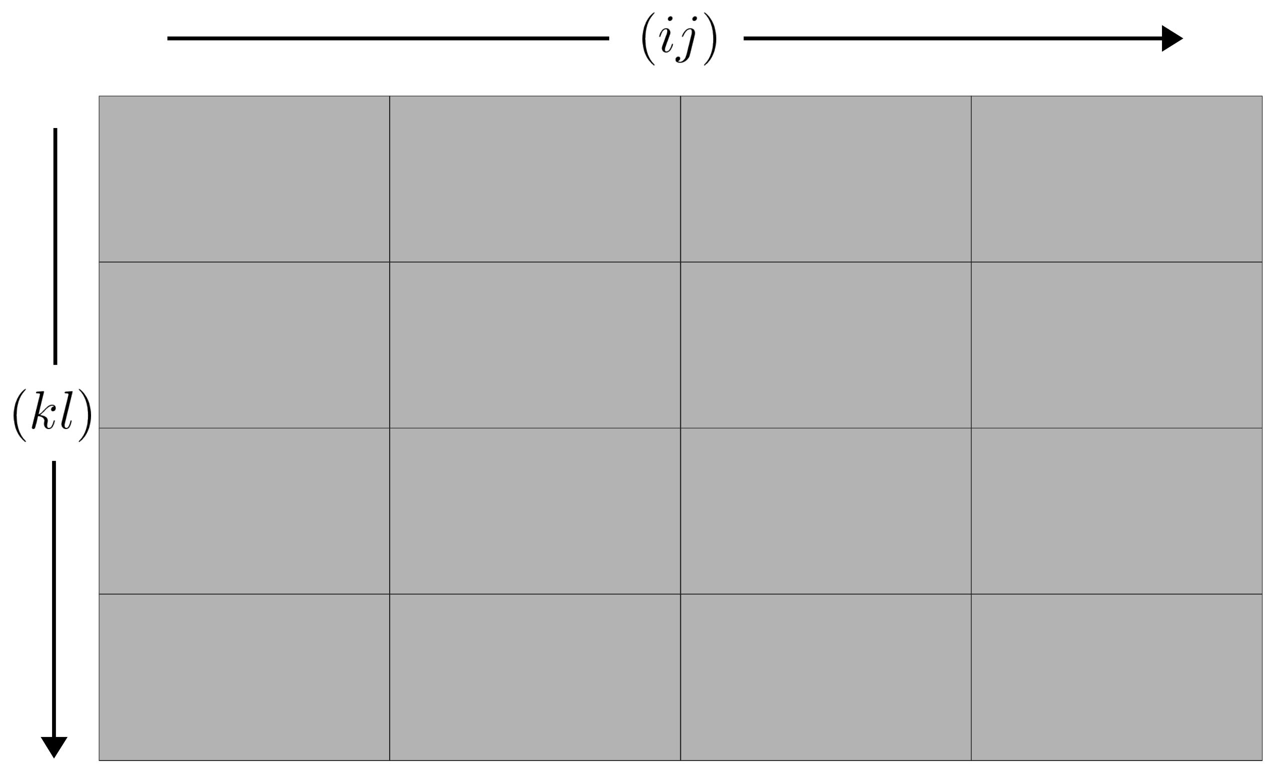
$$A_{abcd} = \sum_{x}^{x \in X} \sum_{efghmn} \delta_{ef}\delta_{gh}\delta_{mn}T^{\dagger}_{ijkl}g_{wxyz}q_{uv}$$

$$X = \{(\{(a_1, b_1), (c_1, d_1), (e_1, f_1)\}, s_1), (\{(a_2, b_2), (c_2, d_2), (e_2, f_2)\}, s_2), ...\}$$

- Each path from top to bottom corresponds to a member of X .
- Order of contractions is not important.
- Reorder the individual members of X to get the most efficient tree.

Application of block symmetry

Example : CASPT2 perturbation coefficients $T_{(ij),(kl)}$.



Possible transitions

$closed \rightarrow active$
$closed \rightarrow virtual$
$active \rightarrow active$
$active \rightarrow virtual$

Application of block symmetry

Example : CASPT2 perturbation coefficients $T_{(ij),(kl)}$.

	$c \rightarrow a$	$c \rightarrow v$	$a \rightarrow a$	$a \rightarrow v$
$c \rightarrow a$	▨	▨	▨	▨
$c \rightarrow v$	▨	▨	▨	▨
$a \rightarrow a$	▨	▨	▨	▨
$a \rightarrow v$	▨	▨	▨	▨

Possible transitions

$closed \rightarrow active$

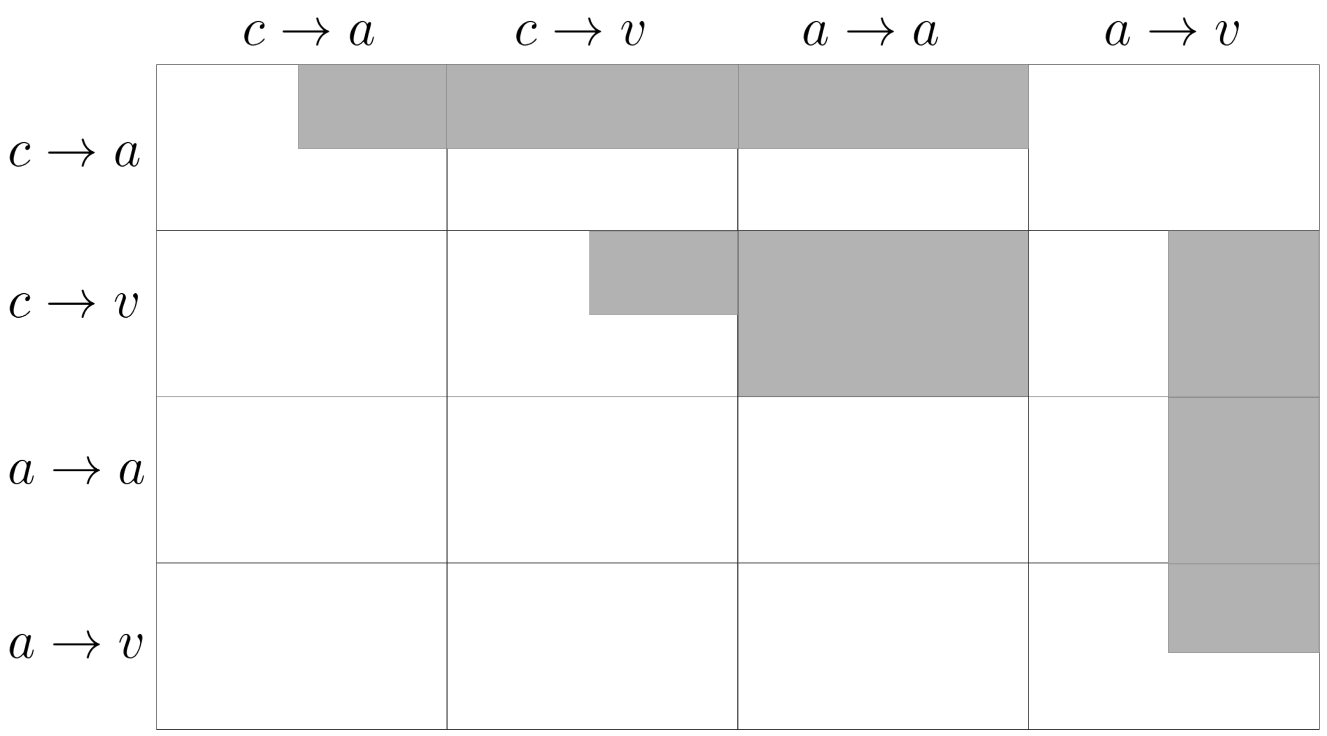
$closed \rightarrow virtual$

$active \rightarrow active$

$active \rightarrow virtual$

Application of symmetry

Example : CASPT2 perturbation coefficients $T_{(ij),(kl)}$.



Possible transitions

$closed \to active$

$closed \to virtual$

$active \to active$

$active \to virtual$

Application of block symmetry

Gaunt (or Breit) term:

	$\alpha \rightarrow \alpha$	$\alpha \rightarrow \beta$	$\beta \rightarrow \alpha$	$\beta \rightarrow \beta$
$\alpha \rightarrow \alpha$				
$\alpha \rightarrow \beta$				
$\beta \rightarrow \alpha$				
$\beta \rightarrow \beta$				

Possible transitions

$\alpha \rightarrow \alpha$

$\alpha \rightarrow \beta$

$\beta \rightarrow \alpha$

$\beta \rightarrow \beta$

Application of block symmetry

Example : Gaunt (or Breit) term: $H^{Gaunt}_{(ij),(kl)}$

	$\alpha \rightarrow \alpha$	$\alpha \rightarrow \beta$	$\beta \rightarrow \alpha$	$\beta \rightarrow \beta$
$\alpha \rightarrow \alpha$	▨	▨		
$\alpha \rightarrow \beta$	▨	▨		
$\beta \rightarrow \alpha$				
$\beta \rightarrow \beta$				

Possible transitions

$\alpha \rightarrow \alpha$

$\alpha \rightarrow \beta$

$\beta \rightarrow \alpha$

$\beta \rightarrow \beta$

Contraction of tensor blocks

Contract over all possible pairs of indexes, e.g.,

$$\hat{H}\hat{T} \rightarrow \sum_{wxyz}\sum_{ijkl} H_{wxyz} a_w^\dagger a_x^\dagger a_y a_z T_{ijkl} a_i^\dagger a_j^\dagger a_k a_l$$

$$A_{abcdef}^{HT,st} = \sum_{wxyz}\sum_{ijkl} H_{wxyz} T_{ijkl} \delta_{st}$$

$$s := w, x, i, \text{ or } \; j$$
$$t := y, z, k \text{ or } \; l$$

Use the block range constraints to rule out possible contractions, e.g.

$$A_{abcdef}^{HT,st} = \begin{cases} A_{abcdef}^{HT,st} & \text{if} \quad rng(s) = rng(t) \\ 0 & \text{otherwise} \end{cases}$$

Contraction list is different for *every* range block.

Contraction of tensor blocks

- Distinct range blocks may become equivalent following contraction, e.g.

$$A^{HT}_{\mathbf{wxyjkl}} = \sum_{\mathbf{wxyz}} \sum_{\mathbf{ijkl}} H_{\mathbf{wxyz}} T_{\mathbf{ijkl}} \delta_{\mathbf{zi}}$$ **Active**

$$A^{HT}_{\mathbf{wxyjkl}} = \sum_{\mathbf{wxyz}} \sum_{\mathbf{ijkl}} H_{\mathbf{wxyz}} T_{\mathbf{ijkl}} \delta_{\mathbf{zi}}$$ **Core**

$$A^{HT}_{\mathbf{wxyjkl}} = \sum_{\mathbf{wxyz}} \sum_{\mathbf{ijkl}} H_{\mathbf{wxyz}} T_{\mathbf{ijkl}} \delta_{\mathbf{zi}}$$ **Virtual**

- Following the contraction the range block is the same.

- Important to combine contraction lists following this; otherwise can potentially increase the complexity.

Generation of contraction list

Step 1 : First take creation and annihilation operators in order specified by input operator, e.g.,

$$\langle I|a_l a_i^\dagger a_m a_j^\dagger a_n a_k^\dagger|J\rangle$$

Step 2 : Rewrite in normal order:

$$= \langle I|a_i^\dagger a_j^\dagger a_k^\dagger a_l a_m a_n|J\rangle + (s_{m'n'}\delta_{m'n'} + ...)\langle I|a_{i'}^\dagger a_{j'}^\dagger a_{k'} a_{l'}|J\rangle$$
$$+(s_{k'l'm'n'}\delta_{m'n'}\delta_{k'l'} + ...)\langle I|a_{i'}^\dagger a_{j'}^\dagger|J\rangle + ...$$

Step 3 : Remove terms which known to be zero from index ranges.

Step 4 : Repeat process, but reorder to alternating order

$$\langle I|a_i^\dagger a_k^\dagger a_j^\dagger a_l a_m a_n|J\rangle \rightarrow \langle I|a_i^\dagger a_l a_j^\dagger a_m a_k^\dagger a_n|J\rangle$$
$$= \gamma_{ijklmn}^{IJ}$$

Grouping of indexes

- Takes too long in relativistic framework;

$$N_{blocks} \approx N_{range}^{N_{id}} = 6^{10}$$

- Furthermore, applying spin symmetry was difficult.
- Rewrote contraction list generation routines to avoid this problem.
- Gamma generation routine now takes range constraints.
- Possible to immediately rule out majority of ranges.

Grouping of indexes

- Introduced grouping of indexes to reduce number of distinct terms, e.g.,

$$\langle I|a^\dagger_{\mathbf{i}} a_{\mathbf{j}} a^\dagger_{\mathbf{k}} a_{\mathbf{l}} a^\dagger_{\mathbf{m}} a_{\mathbf{n}} a^\dagger_{\mathbf{o}} a_{\mathbf{p}}|J\rangle c_J \rightarrow \langle I|a^\dagger_{\mathbf{i}} a_{\mathbf{n}} a^\dagger_{\mathbf{o}} a_{\mathbf{l}} a^\dagger_{\mathbf{m}} a_{\mathbf{j}} a^\dagger_{\mathbf{k}} a_{\mathbf{p}}|J\rangle c_J$$

$$\gamma_{\mathbf{ij}}\gamma_{\mathbf{kl}}\gamma_{\mathbf{mn}}\gamma_{\mathbf{op}} \rightarrow \gamma_{\mathbf{in}}\gamma_{\mathbf{ol}}\gamma_{\mathbf{mj}}\gamma_{\mathbf{kp}}$$

- Colours denote different ranges.
- Immediately useful for spin, but can be used in other contexts, e.g., point symmetry.

Contraction of γ^{I}_{ijklmn} and A_{ijklmn}

Perform contraction in two steps:

$$\sum_{\sigma_3}\sum_{ijklmn}\gamma^{IJ\sigma_3}_{ij,kl,mn}A^{\sigma_3}_{ij,kl,mn}c_J=\sum_{\sigma_1}\sum_{ij}\sum_{K}\gamma^{IK\sigma_1}_{ij}\tilde{\gamma}^{K\sigma_1}_{ij}$$

- Where

$$\tilde{\gamma}^{K\sigma_1}_{ij}=\sum_{\sigma_2}\sum_{LJ}\sum_{klmn}\gamma^{KJ\sigma_2}_{kl,mn}A'^{\sigma_1\otimes\sigma_2}_{ij,kl,mn}$$

and

$$\sigma_3=\sigma_1\cup\sigma_1\cup\sigma_1=\{s_1s_2s_3s_4s_5s_6\}$$

- This step should be comparable in speed to the most expensive step of the current algorithm.

Contraction of γ^{I}_{ijklmn} and A_{ijklmn}

Contract with A-tensor whilst generating the gamma matrices:

$$\sum_{ijklmn} \gamma^{IJ}_{ij,kl,mn} A_{ij,kl,mn} c_J$$

$$= \sum_{ij}\sum_{K} \langle I|a^\dagger_i a_j|K\rangle \sum_{kl}\sum_{L} \langle K|a^\dagger_k a_l|L\rangle \sum_{mn}\sum_{J} \langle L|a^\dagger_m a_n|J\rangle A_{ij,kl,mn} c_J$$

$$= \sum_{ij}\sum_{K} \langle I|a^\dagger_i a_j|K\rangle \sum_{kl}\sum_{L} \langle K|a^\dagger_k a_l|L\rangle \tilde{A}^{L}_{ij,kl}$$

$$= \sum_{ij}\sum_{K} \langle I|a^\dagger_i a_j|K\rangle \tilde{A}^{K}_{ij}$$

- Minimizes the number of gamma matrices which need to be generated and stored.