Coulomb and Exchange algorithms

The Molecular Sciences Software Institute Daniel G. A. Smith

Coulomb and Exchange matrices show up many times in Quantum Mechanics. Several places that they can occur (but not limited to!) are SCF, SCF linear response, SCF Hessians, MP2 gradients, Symmetry-Adapted Perturbation Theory, ... It is therefore quite advantageous to spend time optimizing these algorithms for under various approximations and for specific hardware. As discussed these take the very general form of:

$$J[D_{\lambda\sigma}]_{\mu\nu} = g_{\mu\nu\lambda\sigma} D_{\lambda\sigma} \tag{1}$$

$$K[D_{\lambda\sigma}]_{\mu\nu} = g_{\mu\lambda\nu\sigma} D_{\lambda\sigma} \tag{2}$$

Where $D_{\lambda\sigma}$ can take a great many forms and does not necessarily represent a canonical SCF density.

I. THE PK SUPERMATRIX ALGORITHMS

The PK Supermatrix algorithm[1] effectively attempts to align the data of ERI tensors to that of the underlying density. In essence,

$$J[D_{\lambda\sigma}]_{\mu\nu} = g_{\mu\nu\lambda\sigma} D_{\lambda\sigma} \tag{3}$$

$$K[D_{\lambda\sigma}]_{\mu\nu} = g^{k}_{\mu\nu\lambda\sigma} D_{\lambda\sigma} \tag{4}$$

$$g^{k}_{\mu\nu\lambda\sigma} = g_{\mu\nu\lambda\sigma} \tag{5}$$

Where the g^k tensor is a permuted form of g so that when building the K matrix the data is aligned. Please note we will talk a great deal about alignment in the C++ section of the course.

At this point we have effectively doubled the space of our ERI tensor as the g^k tensor needs to be stored as well. We need to explain one more curiosity of the g tensor for this particular method to gain a good deal of needed. Consider that the ERI tensor has the following symmetry,

$$g_{\mu\lambda\nu\sigma} \equiv g_{\mu\lambda\sigma\nu} \equiv g_{\lambda\mu\nu\sigma} \equiv g_{\lambda\mu\sigma\nu} \equiv g_{\nu\sigma\mu\lambda} \equiv g_{\sigma\nu\mu\lambda} \equiv g_{\nu\sigma\lambda\mu} \equiv g_{\sigma\nu\lambda\mu} \tag{6}$$

In the NumPy based formalism presented it is difficult to exploit the full 8-fold symmetry of the ERI tensor; however, do try to exploit some of the symmetry. A particularly useful function is the *numpy.triu$_i$ndices* for acquiring and indexing the lower triangular part of the ERI tensor. Do not forget to consider that the student working on SCF response will have a non-symmetric density.

II. DENSITY-FITTED COULOMB AND EXCHANGE ALGORITHMS

A. Theory

With density fitting[2–5] the two-electron integrals are represented by the following

$$g_{\mu\nu\lambda\sigma} \approx (\widetilde{\mu\nu|P})[J^{-1}]_{PQ}(\widetilde{Q|\lambda\sigma}) \tag{7}$$

where the Coulomb metric $[J]_{PQ}$ and the three center overlap integral $(\widetilde{Q|\lambda\sigma})$ are defined as

$$[J]_{PQ} = \int P(\mathbf{r}_1)\frac{1}{\mathbf{r}_{12}}Q(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2 \tag{8}$$

$$(\widetilde{Q|\lambda\sigma}) = \int Q(\mathbf{r}_1)\frac{1}{r_{12}}\lambda(\mathbf{r}_2)\sigma(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2 \tag{9}$$

To simplify the density fitting notation, the inverse Coulomb metric is typically folded into the three center overlap tensor

$$(P|\lambda\sigma) = [J^{-\frac{1}{2}}]_{PQ}(\widetilde{Q|\lambda\sigma}) \tag{10}$$

$$g_{\mu\nu\lambda\sigma} \approx (\mu\nu|P)(P|\lambda\sigma) \tag{11}$$

B. Coulomb Matrix

The Coulomb matrix can then be computed in $\mathcal{O}(\mathrm{N}^2\mathrm{N}_{aux})$ operations:

$$\chi_P = (P|\lambda\sigma)D_{\lambda\sigma} \tag{12}$$

$$J[D_{\lambda\sigma}]_{\mu\nu} = (\mu\nu|P)\chi_P \tag{13}$$

C. Exchange Matrix

The Exchange matrix can be computed in $\mathcal{O}(\mathrm{N}^3\mathrm{N}_{aux})$ operations:

$$\zeta_{P\nu\lambda} = (P|\nu\sigma)D_{\lambda\sigma} \tag{14}$$

$$K[D_{\lambda\sigma}]_{\mu\nu} = (\mu\lambda|P)\zeta_{P\nu\lambda} \tag{15}$$

however, considering the form of the density matrix, we can reduce this to $\mathcal{O}(p\mathrm{N}^2\mathrm{N}_{aux})$

$$D_{\lambda\sigma} = C_{p\sigma}C_{p\lambda} \tag{16}$$

$$\zeta^1_{P\mu p} = (\mu\sigma|P)C_{p\sigma} \tag{17}$$

$$\zeta^2_{P\nu p} = (P|\nu\lambda)C_{p\lambda} \tag{18}$$

$$K[D_{\lambda\sigma}]_{\mu\nu} = \zeta^1_{P\mu p}\zeta^2_{P\nu p} \tag{19}$$

where p is a generalized MO index that can span any space. This technique is especially beneficial when p is an inactive or active index as these are typically small relative to the full N space. It should be noted that Eqs. 16-19 pertain to the generalized case where $C_{p\sigma} \neq C_{p\lambda}$, if both C matrices are identical, only one ζ intermediate needs to be built. The $C_{p\sigma} \neq C_{p\lambda}$ case often arises in SCF theory when either rotated, transition, or generalized density matrices are used.

D. Benefits

Computation of the Coulomb and Exchange matrices through conventional means costs N^4, here we see that Coulomb builds are rank reduced, but Exchange builds are of the same rank. The real benefit of density-fitted integrals for K builds comes from data locality and data storage. As N_{aux} is typically twice the size of N, we can imagine a case with 3000 AO basis functions. Conventional 4-index two-electron integrals would use 81 TB if the 8-fold symmetry of the tensor is exploited, on the other hand, the 3-index density-fitted tensor would only take up 216 GB if the 2-fold symmetry is exploited, a reduction of 375 fold.

E. Code snippets

In order to compute the necessary quantities several new code techniques need to be first introduced. If you observe the $[J]_{PQ}$ and $(\widetilde{Q|\lambda}\sigma)$ integrals carefully you should notice that these are conventional ERI integrals without two or one centers, respectively. Therefore, we compute these integrals just like ERI's, but with a so-called "zero-basis" which, in effect, does introduce a new center.

First we need to build our basis sets and MintsHelper object,

```
# Get orbital basis from a Wavefunction object
orb = wfn.basisset()

# Build the complementary JKFIT basis for the aug-cc-pVDZ basis (for example)
aux = psi4.core.BasisSet.build(mol, fitrole="JKFIT", other="aug-cc-pVDZ")

# The zero basis set
zero_bas = psi4.core.BasisSet.zero_ao_basis_set()

# Build instance of MintsHelper
```

```
mints = psi4.core.MintsHelper(orb)
```

The MintsHelper object can also accept a series of basis objects indicating which basis each index should be. The $\widetilde{(Q|\lambda\sigma)}$ and $[J]_{PQ}$ quantities can then be built like the following,

```
# Build (P|pq) raw 3-index ERIs, dimension (1, Naux, nbf, nbf)
Qls_tilde = mints.ao_eri(zero_bas, aux, orb, orb)
Qls_tilde = np.squeeze(Qls_tilde) # remove the 1-dimensions


# Build & invert Coulomb metric, dimension (1, Naux, 1, Naux)
metric = mints.ao_eri(zero_bas, aux, zero_bas, aux)
metric.power(-0.5, 1.e-14)
metric = np.squeeze(metric) # remove the 1-dimensions
```

From here the $(Q|\lambda\sigma)$ can be assembled and then contracted with C or D to build the coulomb and exchange matrices.

REFERENCES

[1]R. C. Raffenetti, Chem. Phys. Lett. **20**, 335 (1973).

[2]F. Weigend, Phys. Chem. Chem. Phys. **4**, 4285 (2002).

[3]O. Vahtras, J. Almlöf, and M. W. Feyereisen, Chem. Phys. Lett. **213**, 514 (1993).

[4]B. I. Dunlap, J. W. D. Connolly, and J. R. Sabin, J. Chem. Phys. **71**, 3396 (1979).

[5]J. L. Whitten, J. Chem. Phys. **58**, 4496 (1973).