Psi4’s DFT and SAPT(DFT) Design Architecture: Successes, Failures, and Lessons

Daniel G. A. Smith
The Molecular Sciences Software Institute
**SAPT(DFT) Requirements**

**Requirements:**
- New DFT XC kernels
- DFT XC kernel 3rd-order derivatives
- Gradient-regularized asymptotic correction
- SCF/DFT – Linear response
- SCF/DFT – Time-dependent response

**Pickups:**
- VV10
- CAM/DSD functionals
- "Modern" functionals (wB97M-V)
- DFT Hessians
- TDDFT
- Generalized solvers

"Make it faster"
DFT Overview

\[ FC = S \epsilon \]

HF: \[ F = H + 2J - K \]

DFT: \[ F = H + 2J - \alpha K - \beta K^\omega + V \]

V Permutations:
- X Functionals
- C Functionals
- VV10
- GRAC
- Range-Separated Parameter
- ...

Other Effects:
- MP2
- D3/D4 Dispersion
- DAS Dispersion
- GCP Corrections
- BSIE Corrections
- ...

“John Von Neumann would just look at this and sigh”
SCF Class Structure

\[
\text{HF: } F = H + 2J - K
\]
\[
\text{DFT: } F = H + 2J - \alpha K - \beta K^{\omega} + V
\]

Superfunctional
- Parameters
- X/C Functionals
- Post-processing

Wavefunction
- HF
  - RHF
  - ROHF
  - CUHF
  - UHF

Superfunctional
- KS
  - RKS
  - UKS
Consider: Functional Programming

rhhf_cphf_solve(...) 
uhhf_cphf_solve(...) 

c1_deep_copy() 
new HF
def build_b2plyp_d3m(name, npoints, deriv, restricted):
    
    # Call this first
    sup = core.SuperFunctional.blank()
    sup.set_max_points(npoints)
    sup.set_deriv(deriv)

    # => User-Customization <= #

    # Add member functionals
    becke = core.LibXCFFunctional('XC_GGA_X_B88', restricted)
    becke.set_alpha(0.47)
    sup.add_x_functional(becke)
    lyp = core.LibXCFFunctional('XC_GGA_C_LYP', restricted)
    lyp.set_alpha(0.73)
    sup.add_c_functional(lyp)

    # Set GKS up after adding functionals
    sup.set_x_alpha(0.53)
    sup.set_c_alpha(0.27)

    # => End User-Customization <= #

    # Call this last
    sup.allocate()
    return (sup, '-D3M')

psi4.energy("SCF", dft_functional=build_b2plyp_d3m)
LibXC Kernels

• 250+ DFT Primitives
• First, second, and third derivatives
• Up-to-date:
  • HSE
  • ωB97M-V
  • SCAN
  • MN15
• ~10 contributors
• Implemented in 23+ codes
• Fortran and C++ interfaces
• Actively developed (some issues)
V Builds [grids]

\[ F = H + 2J - \alpha K - \beta K^\alpha + V \]

\[ V = f^{xc}[\rho(\hat{r})] \quad E_x^{LDA}[\rho] = -\frac{3}{4} \left( \frac{3}{\pi} \right)^{\frac{1}{3}} \int \rho(\hat{r})^{\frac{4}{3}} d\hat{r} \]

We also need the derivatives!

---

Quadrature Grid

Collocation

Density on a grid

\[ \phi_\mu^P D_{\mu\nu} \phi_\nu^P \rightarrow \rho^P \]

Arbitrary density
V Integration

Cast AO density to grid
\[ \rho \phi^{P} \rightarrow \phi^{P} D_{\mu \nu} \phi^{P} \rightarrow \rho^{P} \]

Co-location is sparse and is broken into blocks
Perfect place to thread!

Evaluate all XC kernels and required derivatives
\[ E_{x}^{GGA}, \delta \rho \delta \gamma^{x}, \delta \rho \delta \gamma^{y}, \ldots, \delta \gamma \delta \gamma \rho \]

Cast partial derivatives back to AO space
\[ V_{\mu \nu} \leftarrow \phi^{P}_{\mu} f^{xc}_{\delta \gamma}[\rho(\hat{\gamma})] \phi^{P}_{\nu} \]

Partials and chain rule can get out of hand
DFT Update Benefits

• Hundreds of new functionals (more than Q-Chem/Gaussian/NWChem/GAMESS)
  • GRAC, VV10, CAM, DSD

• V builds are threaded and optimized, up to 12x speedup on 6 cores

• Overall DFT can be up to 10x faster for meta functionals

• Time dependent/independent Hessian–vector elements are available

• DFT linear response implemented
  • TDDFT just a step away

• Second–order KS convergence (SOUKS)
  • Stability analysis could be done easily
DFT Update Lessons

Successes:
- XC Kernel generation is offloaded
- New object-level threading model for DFT
- LibXCFunctional derived from Functional

Failures:
- CP–SCF is implemented on HF derived Wavefunctions
- Missing SG–0/SG–1/SG–2 grids (I think)
- Optimized before profiling (collocation matrix rate limiting)
- An enormous amount of potential currently wasted

Lessons:
- Large vertical class hierarchy gets messy and restrictive
  - What happens when a method is only implemented on Wavefunction?
- Pre–factors matter more than scaling in our normal size regime
  \[
  \frac{4 \pi^3}{3 \cdot 4} \approx 0.529...
  \]
SAPT Introduction

- Obtains interaction energies by an intermolecular perturbation
- Four fundamental forces:
  - Electrostatics – Charge-charge interaction
  - Exchange – Pauli repulsion
  - Induction – Relaxation of a monomer in the field of the other
  - Dispersion – Correlation between monomers
- SAPT Hamiltonian:

\[
H = F_A + F_B + \zeta V + \xi W_A + \eta W_B
\]

Fock Operators

Intramonomer Correlation

Intermonomer Correlation
This article is copyrighted as indicated in the article. Reuse of AIP content is subject to the terms at: http://scitation.aip.org/termsconditions.

This is done by means of suitable symmetry projection operators and leads to a class of methods known under the common name of symmetry-adapted perturbation theory (SAPT).

The zeroth-order wave function in SAPT is assumed in a form which allows a straightforward calculation of the interaction energy, the perturbational approach must take into account its polarization part, its correlation effects.

We present an extension of many-body symmetry-adapted perturbation theory (SAPT) with the aim of providing a basis for the calculation of the interaction energy of interacting systems containing more than a few electrons. To solve this difficulty, a multiple-perturbation approach has been developed. This approach, termed many-body perturbation series, eliminates the need for a hybrid SAPT approach, involving supermolecular Hartree-Fock calculations.

In the SAPT approach, the Born-Oppenheimer Hamiltonian of the interacting system is the operator of the intermolecular interaction, and exchange energies—which can be related to specific meaningful quantities—the electrostatic, induction, dispersion, and exchange energies—are taken into account by a many-body perturbation series, as

\[ H = F_A + F_B + \zeta V + \xi W_A + \eta W_B \]

where

- \( E_{\text{elst}}^1 = D^A_{\mu\nu}(\mu\nu|\lambda\sigma)D^B_{\lambda\sigma} + D^A_{\mu\nu}V^B_{\mu\nu} + D^B_{\mu\nu}V^A_{\mu\nu} + E_{\text{nuc}}^{\text{int}} \)
- \( E_{\text{exch}}^1(S^2) = \text{tr} \left( D^A S D^B S P^A \omega_A + D^B S D^A S P^B \omega_B + P^A S D^B \cdot K[D^A S P^B] \right) \)

SAPT Terms

What does it really take?

- \( e^-/p^+ \) attraction
- \( e^-/e^- \) repulsion
- \( p^+/p^+ \) repulsion

More complex, but still densities.
SAPT Terms

SCF Wavefunctions

Cache the Wavefunctions and densities

```python
scf_eA, scf_wfnA = psi4.energy("B3LYP/cc-pVDZ", return_wfn=True, molecule=monomerA)
scf_eB, scf_wfnB = psi4.energy("B3LYP/cc-pVDZ", return_wfn=True, molecule=monomerB)

cache = {}
cache["D_A"] = scf_wfnA.Da()
cache["Cocc_A"] = scf_wfnA.Ca_subset("AO", "OCC")
cache["D_B"] = scf_wfnB.Da()
```

Cache any integrals

```python
mintsA = psi4.core.MintsHelper(scf_wfnA.basisset())
mintsB = psi4.core.MintsHelper(scf_wfnB.basisset())

cache["V_A"] = mintsA.ao_potential()
cache["V_B"] = mintsB.ao_potential()
```
SAPT Terms
Computing Electrostatics

\[ E_{elst}^{1} = D_{\mu \nu}^{A} (\mu \nu | \lambda \sigma) D_{\lambda \sigma}^{B} + D_{\mu \nu}^{A} V_{\mu \nu}^{B} + D_{\mu \nu}^{B} V_{\mu \nu}^{A} + E_{\text{int}}^{\text{nuc}} \]

# Coulomb contraction
jk.clear()
jk.add_C_left(cache["Cocc_A"])
jk.compute()

J_A = jk.J()[0]

# Contract terms
Elst10 = cache["D_B"].vector_dot(J_A)
Elst10 += cache["D_A"].vector_dot(cache["V_B"])
Elst10 += cache["D_B"].vector_dot(cache["V_A"])
Elst10 += cache["nuclear_repulsion_energy"]
SAPT Induction
Psi4 Wavefunction Object

- Induction amplitudes are recovered from coupled-perturbed (CP) quantities. \((N^4)\)
- Reference can be:
  - RHF, UHF, RKS, UKS
  - Fractions of exact exchange
  - Range-separated functionals
  - Asymptotic corrections

\[
H^{(2)} x^A = \omega^B
\]

\[
E_{\text{ind}}^{(2)} = -x^A \omega^B - x^B \omega^A
\]

Electrostatic Potential
CP Response

```python
scf_e, scf_wfn = psi4.energy("CAM-B3LYP/cc-pVDZ", return_wfn=True)
X_A = scf_wfn.cphf_solve(cache["W_B"], **options)
print(-1.0 * X_A.vector_dot(cache["W_B"]))
```
SAPT(DFT) Update Lessons

**Successes:**
- Elst and Exch based on functional programming
  - Valid for any SCF theory
- Reusing some Disp and Exch–Disp from F–SAPT through Python and functional–like programming

**Failures:**
- Third (!) SAPTO solver implemented.
  - All still active, but tested to get matching results.
  - Documentationless at the moment

**Lessons:**
- A discussion about deprecation and switching backends needs to be held
- Start mixing in functional–style programming
- If it can be written in Python without performance degradation, do that
A diagram for modularity
Each area is of an interest to specific groups

More modules than any one group would want to work on
Free as in a puppy

• We are hindered by technical debt
  • Software will continuously evolve
  • Hardware will continuously evolve
• We are used to many warts
  • Aging disk I/O
  • Little tensor support
• We are missing some easy tricks
  • DFT MP2 Hessians
  • TDDFT
• Development has become spotty again
  • 2–3 people focusing on Psi
  • Large oscillations in effort (trackable on git, happened since the 80’s)

  • This is ok if Psi wants to maintain the status quo
  • This is not ok if Psi wants to take over the world

• SCF convergence issues
• Large duplication of effort
• MM (Hessians, opts, etc)
• Visualization access

• Aging disk I/O
• Little tensor support
• DFT MP2 Hessians
• TDDFT
• SCF convergence issues
• Large duplication of effort
• MM (Hessians, opts, etc)
• Visualization access
Psi4 Versioning

[Sorry Lori]

• Psi 1, 2, 3, 4 Major Version means a language change
  • Psi4 beta —> C++ program with some Python bindings
  • Psi4 1.0 —> Python API is a bit of an afterthought
  • Psi4 1.1 —> Inverted, recent Python API
• Python API is constrained by our fully C++ past
  • Psithon is invaluable
  • Increasingly large population using Psi4 as a library
• If we want a really good API, we probably need to break it, and we need to think about it very hard
• Psi5 —> A Python program with a C++ backend for heavy lifting
I WANT YOU FOR Psi4

[github.com/psi4]