Psi4 Research Update

Lori Burns, Sherrill Group
14 November 2014
Circa November 2014, basis set handling in PSI4 has been revamped. Fear not that the beloved libmints BasisSet object has changed- rather, the user specification of basis sets, programmer’s API to BasisSet constructors, and the construction of BasisSet objects has been changed.

**Advantages to New Scheme (aka Why)**

- Defaults for fitting basis sets set on a per-atom basis (e.g., DF-SCF on metal-organic with cc-pVDZ uses cc-pVDZ-JKFIT for the organic and Def2-tzvpp (or something) for the metal) so that the user shouldn’t experience a failed job on account of incomplete fitting basis sets.
- All default info for auxiliary basis sets in one place. Programmer when calling for a new auxiliary BasisSet gives the fitting role if defaults need to be computed (e.g., JKFIT) and the orbital basis to compute defaults off of (e.g., get_option(BASIS)). This eliminates all the “corresponding_jkfit” boilerplate in proc.py and also means defaults can be assigned for non-uniform orbital basis sets.
- Assignment of basis sets to atoms proceeds through “all”, “by_symbol” (e.g., “Co”), or “by_label” (e.g., H1 or Co_mine). There is no assignment to atoms by number (except a bit internally where it’s safe) which can be ambiguous when the Molecule has been fragmented as for SAPT.
- Users don’t need to “set basis basename” after every molecule {...} definition or activation because basis sets are not attached to the molecule at time option is set but at time BasisSet is built. Similarly, once can define a basis basename {...} block and use it for multiple molecules.

BasisSet gives the option name where any user intentions as to proper value may be found (DF, which the new basis can be recalled (get_str(‘DF_BASIS_SCF’)), the fitting role if defaults need to be...
Can set basis sets before molecule

Can re-use basis sets for multiple molecules
Custom basis sets can be set before molecule, too

When named, custom basis sets can be recalled
molecule watdim {
  0 1
  O  -1.551007  -0.114520   0.000000
  H  -1.934259   0.762503   0.000000
  H_hb -0.599677   0.040712   0.000000
--
  0 1
  O   1.350625   0.111469   0.000000
  H   1.680398  -0.373741  -0.758561
  H   1.680398  -0.373741   0.758561
}

set basis minebasfile
energy('scf')  # water dimer in minebas

set basis cc-pvdz
energy('scf')  # water dimer in dz

molecule ammdim {
  0 1
  N  -1.578718  -0.046611   0.000000
  H  -2.158621   0.136396  -0.809565
  H  -2.158621   0.136396   0.809565
  H_hb -0.849471   0.658193   0.000000
--
  0 1
  N   1.578718   0.046611   0.000000
  H   2.158621  -0.136396  -0.809565
  H   0.849471  -0.658193   0.000000
  H   2.158621  -0.136396   0.809565
}

set basis minebasfile
energy('scf')  # ammonia dimer in minebas

>>> cat minebasfile.gbs

spherical
!
! H_hb is aug-cc-pvtz
! H is cc-pvtz
! other are aug-cc-pvdz
!
****
H_hb 
0
S   3   1.00
...
  0.2470000              1.0000000
****
H 
0
S   3   1.00
  33.8700000              0.0060680
...
  1.0570000              1.0000000
****
N 
0
S   8   1.00
  9046.0000000              0.0007000
...
  0.2300000              1.0000000
****
O 
0
S   8   1.00
...
  0.3320000              1.0000000
****

That same custom basis set can be stored in a file

Files now allow label specs
Above is basis {} block, below is custom gbs file

See (with print > 1) exactly what gets loaded from where

Results in identical orbital basis sets, but df_basis_scf differ b/c basis {...} block preserves info
Programmer’s Access to BasisSet Objects

To get a BasisSet object into your module, just call `pyconstruct` where formerly you called `construct`. There are two flavors, one for orbital basis sets and one for auxiliary basis sets. There’s no difference in the BasisSet objects they return or even the code used to assemble them— the two flavors are just for sane argument naming and to establish different signatures for Boost binding.

**Orbital Basis**

Give the function a Molecule object for which to build basis, a label for the basis (generally, BASIS), and a hint for finding the basis. This last argument gets used to find a python function by that name camouflaged (that’s what `basis {...}` blocks in the input file get translated into) or failing that a string to find a gbs file defining the basis.

```cpp
// simple
boost::shared_ptr<BasisSet> primary = BasisSet::pyconstruct_orbital(molecule, "BASIS", "CC-PVDZ");

// self-contained
boost::shared_ptr<BasisSet> primary = BasisSet::pyconstruct_orbital(Process::environment.molecule(), "BASIS", Process::environment.options.get_str("BASIS");
```

**Auxiliary Basis**

Give the function a Molecule object for which to build basis, a label for the basis, a hint for finding the basis, a fitting role to apply if defaults need to be generated, and a hint for finding the orbital basis to build defaults against.

```cpp
// simple
boost::shared_ptr<BasisSet> auxiliary = BasisSet::pyconstruct_auxiliary(molecule, "DF_BASIS_SCF", "", "JKFIT", "CC-PVDZ");

// self-contained and force Spherical
boost::shared_ptr<BasisSet> auxiliary = BasisSet::pyconstruct_auxiliary(Process::environment.molecule(), "DF_BASIS_SCF", Process::environment.options.get_str("DF_BASIS_SCF"), "JKFIT", Process::environment.options.get_str("BASIS"), 1);
```

**Adding Basis Option to Code**

- Register new basis keyword with `psi4/src/bin/psi4/read_options.cc` (of course). The default should be the empty string.

```cpp
options.add_str("DF_BASIS_ELST", "");
```

- Register new basis keyword with the input parser `psi4/lib/python/inputparser.py`. In the main function `process_input`, add it to the regex below. This ensures that users can define `basis_keyword basis_name {...}` blocks where the contents of the block get associated with `basis_name` and assigned to your `basis_keyword`.

```cpp
basis_block = re.compile(r'^\s*?\{basis|df_basis_scf|df_basis_mp2|df_basis_cc|df_basis_sapt\}=\s*\{\w*\}\s*\{\s*\}\s*\{\s*\}',
    re.MULTILINE | re.DOTALL | re.IGNORECASE)
```
Approaching QC Tasks with the Right Tools

Python Crushing a Gnu
Antoine-Louis Barye, c. 1860
## Interoperability: the Best of Two Codes

<table>
<thead>
<tr>
<th></th>
<th>Psi4</th>
<th>Cfour</th>
<th>Psi4 + Cfour</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFT Grad.</strong></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Anharm. Freq.</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>DFT Anharm. Freq.</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Restartable, parallel
- Any P4 or C4 grad. mtd.
- Options/basis either prog.
- Values avail. electronically, not just text output file

### Example Code

```python
memory 4 gb
molecule h2o {
    O  -1.51  -0.13   0.00
    H  -1.95   0.73   0.00
    H  -0.56   0.09   0.00
    units angstrom
}
set basis aug-cc-pvdz
vpt2('b97-d3')
```

Best SCF-level vibrational spectrum from **Psi4**, from **Cfour**, and **together**.

thanks to Prof. John Stanton & Devin Matthews for help with interface
Interoperability: Pooling Capabilities

**Psi4** + **Molpro**

MP2 Interaction E. - SAPT0 Dispersion E. + TDDFT Dispersion E. = MP2C Interaction E.

---

**Psi4**

**Molpro**

\[ \approx \]

---

time

---

memory 4 gb

molecule {
 0 1
 0 -1.55 -0.11 0.00
 H -1.93 0.76 0.00
 H -0.60 0.04 0.00
--
 0 1
 0 1.35 0.11 0.00
 H 1.68 -0.37 -0.76
 H 1.68 -0.37 0.76
}

set basis aug-cc-pvdz

energy('mp2c')

---

**Psi4**

**Molpro**

\[ \times 2 \]

\[ \times 3 \]

---

**Psi4** + **Molpro**

---

---

---

---

Psi4-style input for all programs

Molecule and BasisSet get translated into appropriate format

options get translated into appropriate format but also selected psi4 options (e.g., scf convergence) get translated into driven program’s options (with appropriate non-clobbering)

drive external program and extract results all through python

extract results into PsiVariables, PsiMatrices, transform results back to psi4 input orientation

now results available to any driver wrapper or data writer

segregate interface python code between those that call the Boost python module psi4 (lib/python/interface_cfour.py, lib/python/wrapper_cfour.py) and the remainder that do the translation/extraction/orientation (lib/python/qcdb)

All this is preparatory to separating qcdb and the non-Boost-module-calling portions of the PSI4 driver into common driver which calls psi4 among other programs. ok?
PSI Variables

- QC language
  Communicate to user in text form in output file

- Communicate to user in data form through python dicts available after most wrappers

- Communicate QC results to driver/wrapper.
  Once non-psi4 output translated to PsiVariables, MP2 correlation energy from Cfour looks like MP2 correlation energy from Psi4.

- New program acts on arrays of PsiVariable data (through pandas) and applies all the little QC equalities we all know to assemble all available model chemistries.
Implement this in psi4 so that programmers only need define minimum psi variables and remainder are derived/validated. ok?

Extend PsiVariables to PsiArrays for gradients, etc.
import sys
sys.path.append('/Users/loriab/linux/qcdb')
import qcdb

asdf = qcdb.Database(['s22'])
asdf.load_qcdata_byproject('pt2')

print asdf
print asdf.mcs.keys()

asdf.plot_modelchems(['MP2-CP-atz', 'MP2-CP-aqz', 'CCSDT-CP-adz', 'CCSDT-CP-atqzadz'])
Public Service Announcements

- New docs target “make sphinxmini” that bypasses most of the autodocing steps to build quickly.

Interest in a “state variable” for options (beyond the “energy(‘scf’)” runtime Options object) that incorporates options settings from user (check), driver, read_option (check), and module complex-defaulting?
Psi4 Logos: SMALLCAPS and Implications

Do we want “Psi” in small caps? If so, what happens with the “4”?
Psi4 Logos: SMALLCAPS and Implications

Current
Psi4 Logos: SMALLCAPS and Implications
Databases built into Psi4 simplify job execution and result collection

- Database is simple python (text) file
- Build by script from xyz files
- Restartable, parallel
- Tabulate QC values
- Access any model chemistry
- Couple with other wrappers (e.g., basis extrapolation, geometry opt.)

Each colored dot characterizes a noncovalent interaction in the database $SSI$, as computed by the method $SAPT0$.

Faver, Burns, Marshall, Zheng, Sherrill, & Merz, in preparation.