Psithon Documentation

Release 4.01

Psi4 Project

March 15, 2012

CONTENTS

Note: No recompile of the PSI program is necessary for changes made to files in `$PSIDATADIR`, including those described below.

Since quantum chemical methods in PSI4 are accessed through Python functions, and most important quantities are available as PSI variables, it is straightforward to create aliases to commonly run calculations or to define hybrid methods. The `$PSIDATADIR/python/aliases.py` file is intended for editing by the user for this purpose.

As an example, the MP2.5 method is the average of MP2 and MP3. The latter is available through the arbitrary order MPn code and returns all lower energies along with it in PSI variables. The following is basic code that will compute and return the MP2.5 energy.

```
def run_mp2_5(name, **kwargs):

    energy('mp3', **kwargs)
    e_scf = PsiMod.get_variable('SCF TOTAL ENERGY')
    ce_mp2 = PsiMod.get_variable('MP2 CORRELATION ENERGY')
    ce_mp3 = PsiMod.get_variable('MP3 CORRELATION ENERGY')

    ce_mp25 = 0.5 * (ce_mp2 + ce_mp3)
    e_mp25 = e_scf + ce_mp25

    print """  MP2.5 total energy:                      %16.8f\n""" % (e_mp25)
    print """  MP2.5 correlation energy:                %16.8f\n""" % (ce_mp25)

    return e_mp25
```

Compare the above to the method that resides in `aliases.py`. The rationale for the changes is indicated in the comments below.

```
def run_mp2_5(name, **kwargs):
    lowername = name.lower()  # handy variable with name keyword in lowercase
    kwargs = kwargs_lower(kwargs)  # removes case sensitivity in keyword names

    # Run detci calculation and collect conventional quantities
    energy('mp3', **kwargs)
    e_scf = PsiMod.get_variable('SCF TOTAL ENERGY')
    ce_mp2 = PsiMod.get_variable('MP2 CORRELATION ENERGY')
    ce_mp3 = PsiMod.get_variable('MP3 CORRELATION ENERGY')
    e_mp2 = e_scf + ce_mp2  # reform mp2 and mp3 total energies for printing
    e_mp3 = e_scf + ce_mp3

    # Compute quantities particular to MP2.5
    ce_mp25 = 0.5 * (ce_mp2 + ce_mp3)
    e_mp25 = e_scf + ce_mp25
    PsiMod.set_variable('MP2.5 CORRELATION ENERGY', ce_mp25)  # add new method's important results
    PsiMod.set_variable('MP2.5 TOTAL ENERGY', e_mp25)         #     to PSI variable repository
    PsiMod.set_variable('CURRENT CORRELATION ENERGY', ce_mp25)
    PsiMod.set_variable('CURRENT ENERGY', e_mp25)  # geometry optimizer tracks this variable, permit
                                                   #     MP2.5 finite difference optimizations
    # build string of title banner and print results
    banners = ''
    banners += """PsiMod.print_out('\\n')\n"""
    banners += """banner(' MP2.5 ')\n"""
    banners += """PsiMod.print_out('\\n')\n\n"""
    exec banners

    tables  = ''
    tables += """  SCF total energy:                                   %16.8f\n""" % (e_scf)
    tables += """  MP2 total energy:                                   %16.8f\n""" % (e_mp2)
    tables += """  MP2.5 total energy:                                 %16.8f\n""" % (e_mp25)
    tables += """  MP3 total energy:                                   %16.8f\n\n""" % (e_mp3)
    tables += """  MP2 correlation energy:                             %16.8f\n""" % (ce_mp2)
    tables += """  MP2.5 correlation energy:                           %16.8f\n""" % (ce_mp25)
    tables += """  MP3 correlation energy:                             %16.8f\n""" % (ce_mp3)
    PsiMod.print_out(tables)  # prints nice header and table of all involved quantities to output fi

    return e_mp25
```

One final step is necessary. At the end of the `aliases.py` file, add the following line.

```
procedures['energy']['mp2.5'] = run_mp2_5
```

This permits the newly defined MP2.5 method to be called in the input file with the following command.

```
energy('mp2.5')
```

CHAPTER

ONE

CREATING A DATABASE

A necessary consideration in constructing a database is the distinction between reagents and reactions. A reagent is a single molecular system (may be a dimer) whose geometry you are possession of and whose electronic energy may be of interest. A reaction is a combination of one or more reagent energies whose value you are interested in and a reference value for which you may or may not be in possession of. A few examples follow. In a database of interaction energies, the reagents are dimers and their component monomers (usually derived from the dimer geometry), and the reactions are the dimer less monomers energies. In a database of barrier heights, the reagents are reactants, products, and transition-state structures, and the reactions are the transition-states less minimum-energy structures. Possibly you may have a collection of structures to simply be acted upon in parallel, in which case the structures are both the reagents and the reactions. The role of the database.py file is to collect arrays and dictionaries that define the geometries of reagents (GEOS), their combination into reactions (RXNM & ACTV), available reference values for reactions (BIND), and brief comments for reagents and reactions (TAGL). The journey from reagent geometries to functional database.py file is largely automated, in a process described below.

- **Prepare geometry files** Assemble xyz files for all intended reagent systems in a directory. Follow the rules below for best results. The filename for each xyz file should be the name of the system. lowercase or MixedCase is preferable (according to Sherrill lab convention). Avoid dashes and dots in the name as python won't allow them. If you're determined to have dashes and dots, they must be replaced by other characters in the process_input line, then translated back in the GEOS section; see NBC10.py for an example.
  - The first line for each xyz file should be the number of atoms in the system.
  - The second line for each xyz file can be blank (interpreted as no comment), anything (interpreted as a comment), or two integers and anything (interpreted as charge, multiplicity, and remainder as comment).
  - The third and subsequent lines have four fields: the element symbol and the three cartesian coordinates in angstroms. The atom lines should not contain any dummy atoms (what's the use in cartesian form). For dimer systems, an algorithm is used to apportion the atoms into two fragments; thus the atoms need not be arranged with all fragmentA atoms before all fragmentB atoms. The algorithm will fail for very closely arranged fragments. For dimers, any charge and multiplicity from the second line will be applied to fragmentA (python); charge and multiplicity may need to be redistributed later in the editing step.
- Run script ixyz2database.pl

  Move into the directory where all your xyz files are located. Run the script in place, probably as `$PSIDATADIR/databases/ixyz2database.pl`. It will ask a number of questions about your intended database and generate a python file named for your database. Uppercase is preferable for database names (according to Sherrill lab convention). Note your choice for the route variable for the next step.
- Edit file database.py

According to your responses in to questions in the ixyz2database.pl script, several bullets will be printed of edits you necessarily or optionally should make. Copy your new database into `$PSIDATADIR/databases`.

- Thy python functions shall always have final argument **kwargs, that they may take in and pass on keywords meant for other functions. Yea, even the run_mcscf(), and run_ccsd() -type functions that have no use for kwargs. The exceptions are python functions that are only helpers called by a driver function.
- Python functions should read the kwargs dictionary and (possibly) add to it. Functions should not pop or remove keywords from kwargs, even those keywords meaningful only to itself. This will ensure that the complete kwargs is available for pickling and sow/reap procedures. The exception is the molecule argument, which is read by the first function that gets ahold of it. This first function activates the molecule and pops it out of kwargs, effectively setting molecule for all subsequent functions. The code below should suffice.

```
# Make sure the molecule the user provided is the active one
if 'molecule' in kwargs:
    activate(kwargs['molecule'])
    del kwargs['molecule']
molecule = PsiMod.get_active_molecule()
molecule.update_geometry()
```

- Preferrably, the python function signature (for functions intended to be called in input files) is `function(name, **kwargs)`. For functions that have other positional keywords, please bundle them into kwargs at earliest convenience (see **'Database'_** argument db_name for example).
- After the docstring, the first two lines of your function should be the ones below. The first provides a case insensitive handle to the name argument value. The second converts all the kwargs dictionary keys to lowercase versions of themselves, so that input files can be case insensitive.

```
lowername = name.lower()
kwargs = kwargs_lower(kwargs)
```

- Case sensitivity for kwargs dictionary values still needs to be handled. The first line below shows how to convert argument values to lowercase for matching. When not matching a whole value such that regular expressions are needed, the second line below performs a case insensitive match.

```
if (kwargs['db_mode'].lower() == 'continuous'):
if re.match(r'^sapt', name, flags=re.IGNORECASE):
```

- Match boolean keywords (db_cp in the example below) with expressions like the following, which allow case insensitive yes/true/on/1/no/false/off/0 user input. If your argument's value is a derivative level, similarly, use input.der0th, input.der1st, and input.der2nd.

```
if input.yes.match(str(db_cp)):
elif input.no.match(str(db_cp)):
```

- For keywords that might be used in other functions as well as your own, prepend the argument name with a short representation of your function name. For example, there are keywords cp_func, db_func, and opt_func to request what python function, if not energy(), is called by cp(), database(), and optimize().
- Upon checking in a new python file, edit the file `psi4/doc/userman/source/index.rst` and follow the instructions therein that your file may be autodocumented here.
- Write docstrings! For a major function intended for use in input files, start with the skeleton docstring in `psi4/lib/python/example_docstring` and replace anything that looks like `<this>`. For a behind-the-scenes function or if you don't want the bother of dealing with reStructuredText, just write an ordinary docstring. It will get slurped into the documentation in plain text.
- Your python function should follow PEP8 conventions (without the line-length restriction). I'm aiming for files to pass the line below, unless for good reason. The second line is for database Python files.

```
>>> pep8.py -r --ignore=E501 pythonfile.py
>>> pep8.py -r --ignore=E501,E221,E222,E241,E201,E202 databasefile.py
```

- Your python function should not prevent any test case (`make tests`, NOT `make longtests`) from passing. A test case(s) should be written and checked in for any major python function, so that others do not break your code. If most of your work was on the python (as opposed to c++) side, the test case prefix pywrap_ is suggested.
- Be sure to set any new PSI variables through lines like those below. Especially if the function returns an energy, set the 'current energy' variable. This last is needed to communicate with the optimizer.

```
PsiMod.set_variable('MP2.5 CORRELATION ENERGY', ce_mp25)
PsiMod.set_variable('MP2.5 TOTAL ENERGY', e_mp25)
PsiMod.set_variable('CURRENT ENERGY', e_mp25)
```

- Once your python function is fairly stable on its own, it's potential for interoperability with energy()/opt()/cp()/db()/cbs()/etc. should be evaluated. If it makes physical sense that it should work, you should strive to make that interoperability a reality. Some steps:
  - If any interoperability is possible, define an argument xx_func, where xx is a short name for your function. Add near the top of your function code like the below (less the final two lines). The net result of this code is that if the user specifies no *_func arguments, then energy() gets called. If the user defines xx_func, then its value gets called. If the user defines func, then its value gets reassigned to xx_func, func itself is deleted, and xx_func() gets called. Whatever is getting called is stored in func within the function.

```
# Establish function to call
if not('xx_func' in kwargs):
    if ('func' in kwargs):
        kwargs['xx_func'] = kwargs['func']
        del kwargs['func']
    else:
        kwargs['xx_func'] = energy
func = kwargs['xx_func']
if not func:
    raise ValidationError('Function \'%s\' does not exist to be called by wrapper counterpoi
if (func is db):
    raise ValidationError('Wrapper xx is unhappy to be calling function \'%s\'.' % (func.__n
```

  - If specific interoperabilities are known, code them in. For example, if xx shouldn't call db, add the last two lines above to the xx function. If db shouldn't call xx, add the following two lines below to the db function.

```
if (func is xx):
    raise ValidationError('Wrapper database is unhappy to be calling function \'%s\'.' % (fu
```

  - Create a multipart test case that runs some intercalls between your function and others (akin to pywrap_all). In trials, permute the order of calls a few times to expose any calls that don't clean up after themselves and need further attention.
  - When all is validated, add your findings to the great interoperability table in the documentation.

`wrappers.`**`complete_basis_set`**(*name*[, *scf_basis*, *scf_scheme*, *corl_wfn*, *corl_basis*, *corl_scheme*, *delta_wfn*, *delta_wfn_lesser*, *delta_basis*, *delta_scheme*, *delta2_wfn*, *delta2_wfn_lesser*, *delta2_basis*, *delta2_scheme*])

Function to define a multistage energy method from combinations of basis set extrapolations and delta corrections and condense the components into a minimum number of calculations.

Aliases cbs()

Returns (*float*) – Total electronic energy in Hartrees

Psi variables

`CBS TOTAL ENERGY`
`CBS REFERENCE ENERGY`
`CBS CORRELATION ENERGY`
`CURRENT ENERGY`
`CURRENT REFERENCE ENERGY`
`CURRENT CORRELATION ENERGY`

Caution: Some features are not yet implemented. Buy a developer a coffee.

- Methods beyond basic scf, mp2, ccsd, ccsd(t) not yet hooked in through PSI variables, df-mp2 in particular.
- No scheme defaults for given basis zeta number, so scheme must be specified explicitly.
- No way to tell function to boost fitting basis size for all calculations.
- No way to extrapolate def2 family basis sets
- Need to add more extrapolation schemes

As represented in the equation below, a CBS energy method is defined in four sequential stages (scf, corl, delta, delta2) covering treatment of the reference total energy, the correlation energy, a delta correction to the correlation energy, and a second delta correction. Each is activated by its stage_wfn keyword and is only allowed if all preceding stages are active.

$$E_{total}^{\text{CBS}} = \mathcal{F}_{\textbf{scf\_scheme}} \left(E_{total,\ \text{SCF}}^{\textbf{scf\_basis}}\right) + \mathcal{F}_{\textbf{corl\_scheme}} \left(E_{corl,\ \textbf{corl\_wfn}}^{\textbf{corl\_basis}}\right) + \delta_{\textbf{delta\_wfn\_lesser}}^{\textbf{delta\_wfn}} + \delta_{\textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_wfn}}$$

Here, $\mathcal{F}$ is an energy or energy extrapolation scheme, and the following also hold.

$$\delta_{\textbf{delta\_wfn\_lesser}}^{\textbf{delta\_wfn}} = \mathcal{F}_{\textbf{delta\_scheme}} \left(E_{corl,\ \textbf{delta\_wfn}}^{\textbf{delta\_basis}}\right) - \mathcal{F}_{\textbf{delta\_scheme}} \left(E_{corl,\ \textbf{delta\_wfn\_lesser}}^{\textbf{delta\_basis}}\right)$$

$$\delta_{\textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_wfn}} = \mathcal{F}_{\textbf{delta2\_scheme}} \left(E_{corl,\ \textbf{delta2\_wfn}}^{\textbf{delta2\_basis}}\right) - \mathcal{F}_{\textbf{delta2\_scheme}} \left(E_{corl,\ \textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_basis}}\right)$$

A translation of this ungainly equation to example [5] below is as follows. In words, this is a double- and triple-zeta 2-point Helgaker-extrapolated CCSD(T) coupled-cluster correlation correction appended to a triple- and quadruple-zeta 2-point Helgaker-extrapolated MP2 correlation energy appended to a SCF/aug-cc-pVQZ reference energy.

$$E_{total}^{\text{CBS}} = \mathcal{F}_{\text{highest\_1}} \left(E_{total,\ \text{SCF}}^{\text{aug-cc-pVQZ}}\right) + \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}} \left(E_{corl,\ \text{MP2}}^{\text{aug-cc-pV[TQ]Z}}\right) + \delta_{\text{MP2}}^{\text{CCSD(T)}}$$

$$\delta_{\text{MP2}}^{\text{CCSD(T)}} = \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}} \left(E_{corl,\ \text{CCSD(T)}}^{\text{aug-cc-pV[DT]Z}}\right) - \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}} \left(E_{corl,\ \text{MP2}}^{\text{aug-cc-pV[DT]Z}}\right)$$

- **Energy Methods** The presence of a stage_wfn keyword is the indicator to incorporate (and check for stage_basis and stage_scheme keywords) and compute that stage in defining the CBS energy.

The cbs() function requires, at a minimum, `name='scf'` and `scf_basis` keywords to be specified for reference-step only jobs and `name` and `corl_basis` keywords for correlated jobs.

Parameters

- **name** (*string*) – `'scf'` || `'ccsd'` || etc.

  First argument, usually unlabeled. Indicates the computational method for the correlation energy, unless only reference step to be performed, in which case should be `'scf'`. Overruled if stage_wfn keywords supplied.

- **corl_wfn** (*string*) – `'mp2'` || `'ccsd(t)'` || etc.

  Indicates the energy method for which the correlation energy is to be obtained. Can also be specified with `name` or as the unlabeled first argument to the function.

- **delta_wfn** (*string*) – `'ccsd'` || `'ccsd(t)'` || etc.

  Indicates the (superior) energy method for which a delta correction to the correlation energy is to be obtained.

- **delta_wfn_lesser** (*string*) – **|dl|** `'mp2'` **|dr|** || `'ccsd'` || etc.

  Indicates the inferior energy method for which a delta correction to the correlation energy is to be obtained.

- **delta2_wfn** (*string*) – `'ccsd'` || `'ccsd(t)'` || etc.

  Indicates the (superior) energy method for which a second delta correction to the correlation energy is to be obtained.

- **delta2_wfn_lesser** (*string*) – **|dl|** `'mp2'` **|dr|** || `'ccsd(t)'` || etc.

  Indicates the inferior energy method for which a second delta correction to the correlation energy is to be obtained.

•**Basis Sets** Currently, the basis set set through `set` commands have no influence on a cbs calculation.

Parameters

- **scf_basis** (*string*) – **|dl|** `corl_basis` **|dr|** || `'cc-pV[TQ]Z'` || `'jun-cc-pv[tq5]z'` || `'6-31G*'` || etc.

  Indicates the sequence of basis sets employed for the reference energy. If any correlation method is specified, `scf_basis` can default to `corl_basis`.

- **corl_basis** (*string*) – `'cc-pV[TQ]Z'` || `'jun-cc-pv[tq5]z'` || `'6-31G*'` || etc.

  Indicates the sequence of basis sets employed for the correlation energy.

- **delta_basis** (*string*) – `'cc-pV[TQ]Z'` || `'jun-cc-pv[tq5]z'` || `'6-31G*'` || etc.

  Indicates the sequence of basis sets employed for the delta correction to the correlation energy.

- **delta2_basis** (*string*) – `'cc-pV[TQ]Z'` || `'jun-cc-pv[tq5]z'` || `'6-31G*'` || etc.

  Indicates the sequence of basis sets employed for the second delta correction to the correlation energy.

•**Schemes** Transformations of the energy through basis set extrapolation for each stage of the CBS definition. A complaint is generated if number of basis sets in stage_basis does not exactly satisfy requirements of stage_scheme. An exception is the default, `'highest_1'`, which uses the best basis set available. See Extrapolation Schemes for all available schemes.

Parameters

- **scf_scheme** (*function*) – **|dl|** `highest_1` **|dr|** || `scf_xtpl_helgaker_3` || etc.

  Indicates the basis set extrapolation scheme to be applied to the reference energy.
- **corl_scheme** (*function*) – **|dl|** `highest_1` **|dr|** || `corl_xtpl_helgaker_2` || etc.

  Indicates the basis set extrapolation scheme to be applied to the correlation energy.
- **delta_scheme** (*function*) – **|dl|** `highest_1` **|dr|** || `corl_xtpl_helgaker_2` || etc.

  Indicates the basis set extrapolation scheme to be applied to the delta correction to the correlation energy.
- **delta2_scheme** (*function*) – **|dl|** `highest_1` **|dr|** || `corl_xtpl_helgaker_2` || etc.

  Indicates the basis set extrapolation scheme to be applied to the second delta correction to the correlation energy.

Examples

```
>>> # [1] replicates with cbs() the simple model chemistry scf/cc-pVDZ: set basis cc-pVDZ energy
>>> cbs('scf', scf_basis='cc-pVDZ')

>>> # [2] replicates with cbs() the simple model chemistry mp2/jun-cc-pVDZ: set basis jun-cc-pVD
>>> cbs('mp2', corl_basis='jun-cc-pVDZ')

>>> # [3] DTQ-zeta extrapolated scf reference energy
>>> cbs('scf', scf_basis='cc-pV[DTQ]Z', scf_scheme=scf_xtpl_helgaker_3)

>>> # [4] DT-zeta extrapolated mp2 correlation energy atop a T-zeta reference
>>> cbs('mp2', corl_basis='cc-pv[dt]z', corl_scheme=corl_xtpl_helgaker_2)

>>> # [5] a DT-zeta extrapolated coupled-cluster correction atop a TQ-zeta extrapolated mp2 corr
>>> cbs('mp2', corl_basis='aug-cc-pv[tq]z', corl_scheme=corl_xtpl_helgaker_2, delta_wfn='ccsd(t)

>>> # [6] a D-zeta ccsd(t) correction atop a DT-zeta extrapolated ccsd cluster correction atop a
>>> cbs('mp2', corl_basis='aug-cc-pv[tq]z', corl_scheme=corl_xtpl_helgaker_2, delta_wfn='ccsd',

>>> # [7] cbs() coupled with database()
>>> database('mp2', 'BASIC', subset=['h2o','nh3'], symm='on', func=cbs, corl_basis='cc-pV[tq]z',
```

CHAPTER

TWO

OUTPUT

At the beginning of a cbs() job is printed a listing of the individual energy calculations which will be performed. The output snippet below is from the example job [2] above. It shows first each model chemistry needed to compute the aggregate model chemistry requested through cbs(). Then, since, for example, an `energy('ccsd(t)')` yields CCSD(T), CCSD, MP2, and SCF energy values, the wrapper condenses this task list into the second list of minimum number of calculations which will actually be run.

```
Naive listing of computations required.
        scf / aug-cc-pvqz              for  SCF TOTAL ENERGY
        mp2 / aug-cc-pvtz              for  MP2 CORRELATION ENERGY
        mp2 / aug-cc-pvqz              for  MP2 CORRELATION ENERGY
    ccsd(t) / aug-cc-pvdz              for  CCSD(T) CORRELATION ENERGY
    ccsd(t) / aug-cc-pvtz              for  CCSD(T) CORRELATION ENERGY
        mp2 / aug-cc-pvdz              for  MP2 CORRELATION ENERGY
        mp2 / aug-cc-pvtz              for  MP2 CORRELATION ENERGY

Enlightened listing of computations required.
        mp2 / aug-cc-pvqz              for  MP2 CORRELATION ENERGY
    ccsd(t) / aug-cc-pvdz              for  CCSD(T) CORRELATION ENERGY
    ccsd(t) / aug-cc-pvtz              for  CCSD(T) CORRELATION ENERGY
```

At the end of a cbs() job is printed a summary section like the one below. First, in the components section, are listed the results for each model chemistry available, whether required for the cbs job (*) or not. Next, in the stages section, are listed the results for each extrapolation. The energies of this section must be dotted with the weightings in column Wt to get the total cbs energy. Finally, in the CBS section, are listed the results for each stage of the cbs procedure. The stage energies of this section sum outright to the total cbs energy.

```
==> Components <==

----------------------------------------------------------------------------------
               Method / Basis            Rqd   Energy [H]   Variable
----------------------------------------------------------------------------------
                  scf / aug-cc-pvqz        *  -1.11916375   SCF TOTAL ENERGY
                  mp2 / aug-cc-pvqz        *  -0.03407997   MP2 CORRELATION ENERGY
                  scf / aug-cc-pvdz           -1.11662884   SCF TOTAL ENERGY
                  mp2 / aug-cc-pvdz        *  -0.02881480   MP2 CORRELATION ENERGY
              ccsd(t) / aug-cc-pvdz        *  -0.03893812   CCSD(T) CORRELATION ENERGY
                 ccsd / aug-cc-pvdz           -0.03893812   CCSD CORRELATION ENERGY
                  scf / aug-cc-pvtz           -1.11881134   SCF TOTAL ENERGY
                  mp2 / aug-cc-pvtz        *  -0.03288936   MP2 CORRELATION ENERGY
              ccsd(t) / aug-cc-pvtz        *  -0.04201004   CCSD(T) CORRELATION ENERGY
                 ccsd / aug-cc-pvtz           -0.04201004   CCSD CORRELATION ENERGY
----------------------------------------------------------------------------------
```

```
==> Stages <==

-----------------------------------------------------------------------------------
 Stage         Method / Basis            Wt   Energy [H]   Scheme
-----------------------------------------------------------------------------------
   scf            scf / aug-cc-pvqz        1  -1.11916375   highest_1
  corl            mp2 / aug-cc-pv[tq]z     1  -0.03494879   corl_xtpl_helgaker_2
 delta        ccsd(t) / aug-cc-pv[dt]z     1  -0.04330347   corl_xtpl_helgaker_2
 delta            mp2 / aug-cc-pv[dt]z    -1  -0.03460497   corl_xtpl_helgaker_2
-----------------------------------------------------------------------------------

==> CBS <==

-----------------------------------------------------------------------------------
 Stage         Method / Basis                 Energy [H]   Scheme
-----------------------------------------------------------------------------------
   scf            scf / aug-cc-pvqz           -1.11916375   highest_1
  corl            mp2 / aug-cc-pv[tq]z        -0.03494879   corl_xtpl_helgaker_2
 delta  ccsd(t) - mp2 / aug-cc-pv[dt]z        -0.00869851   corl_xtpl_helgaker_2
 total            CBS                         -1.16281105
-----------------------------------------------------------------------------------
```

CHAPTER

THREE

EXTRAPOLATION SCHEMES

`wrappers.`**`highest_1`**`(`***largs*`)`

Scheme for total or correlation energies with a single basis or the highest zeta-level among an array of bases. Used by `wrappers.complete_basis_set()`.

$$E_{total}^X = E_{total}^X$$

`wrappers.`**`scf_xtpl_helgaker_2`**`(`***largs*`)`

Extrapolation scheme for reference energies with two adjacent zeta-level bases. Used by `wrappers.complete_basis_set()`.

$$E_{total}^X = E_{total}^\infty + \beta e^{-\alpha X}, \alpha = 1.63$$

`wrappers.`**`scf_xtpl_helgaker_3`**`(`***largs*`)`

Extrapolation scheme for reference energies with three adjacent zeta-level bases. Used by `wrappers.complete_basis_set()`.

$$E_{total}^X = E_{total}^\infty + \beta e^{-\alpha X}$$

`wrappers.`**`corl_xtpl_helgaker_2`**`(`***largs*`)`

Extrapolation scheme for correlation energies with two adjacent zeta-level bases. Used by `wrappers.complete_basis_set()`.

$$E_{corl}^X = E_{corl}^\infty + \beta X^{-3}$$

$$E_{total}^{\text{CBS}} = \mathcal{F}_{\textbf{scf\_scheme}}\left(E_{total,\ \text{SCF}}^{\textbf{scf\_basis}}\right) + \mathcal{F}_{\textbf{corl\_scheme}}\left(E_{corl,\ \textbf{corl\_wfn}}^{\textbf{corl\_basis}}\right) + \delta_{\textbf{delta\_wfn\_lesser}}^{\textbf{delta\_wfn}} + \delta_{\textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_wfn}}$$

Here, $\mathcal{F}$ is an energy or energy extrapolation scheme, and the following also hold.

$$\delta_{\textbf{delta\_wfn\_lesser}}^{\textbf{delta\_wfn}} = \mathcal{F}_{\textbf{delta\_scheme}}\left(E_{corl,\ \textbf{delta\_wfn}}^{\textbf{delta\_basis}}\right) - \mathcal{F}_{\textbf{delta\_scheme}}\left(E_{corl,\ \textbf{delta\_wfn\_lesser}}^{\textbf{delta\_basis}}\right)$$

$$\delta_{\textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_wfn}} = \mathcal{F}_{\textbf{delta2\_scheme}}\left(E_{corl,\ \textbf{delta2\_wfn}}^{\textbf{delta2\_basis}}\right) - \mathcal{F}_{\textbf{delta2\_scheme}}\left(E_{corl,\ \textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_basis}}\right)$$

A translation of this ungainly equation to example [5] below is as follows. In words, this is a double- and triple-zeta 2-point Helgaker-extrapolated CCSD(T) coupled-cluster correlation correction appended to a triple- and quadruple-zeta 2-point Helgaker-extrapolated MP2 correlation energy appended to a SCF/aug-cc-pVQZ reference energy.

$$E_{total}^{\text{CBS}} = \mathcal{F}_{\text{highest\_1}}\left(E_{total,\ \text{SCF}}^{\text{aug-cc-pVQZ}}\right) + \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}}\left(E_{corl,\ \text{MP2}}^{\text{aug-cc-pV[TQ]Z}}\right) + \delta_{\text{MP2}}^{\text{CCSD(T)}}$$

$$\delta_{\text{MP2}}^{\text{CCSD(T)}} = \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}}\left(E_{corl,\ \text{CCSD(T)}}^{\text{aug-cc-pV[DT]Z}}\right) - \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}}\left(E_{corl,\ \text{MP2}}^{\text{aug-cc-pV[DT]Z}}\right)$$

`wrappers.`**`cp`**(*name*[, *func*, *check_bsse*])

The cp function computes counterpoise-corrected two-body interaction energies for complexes composed of arbitrary numbers of monomers.

Aliases counterpoise_correct(), counterpoise_correction()

Returns (*float*) Counterpoise-corrected interaction energy in kcal/mol

Psi variables

`CP-CORRECTED 2-BODY INTERACTION ENERGY`
`UNCP-CORRECTED 2-BODY INTERACTION ENERGY`

Caution: Some features are not yet implemented. Buy a developer a coffee.
- No values of func besides energy have been tested.
- Table print-out needs improving. Add some PSI variables.

Parameters

- **name** (*string*) – `'scf'` || `'ccsd(t)'` || etc.

  First argument, usually unlabeled. Indicates the computational method to be applied to the molecule. May be any valid argument to `driver.energy()`; however, SAPT is not appropriate.

- **func** (*function*) – **|dl|** `energy` **|dr|** || `optimize` || `cbs`

  Indicates the type of calculation to be performed on the molecule and each of its monomers. The default performs a single-point `energy('name')`, while `optimize` perfoms a geometry optimization on each system, and `cbs` performs a compound single-point energy. If a nested series of python functions is intended (see **'Function Intercalls'_**), use keyword `cp_func` instead of `func`.

- **check_bsse** (*bool*) – `'on'` || **|dl|** `'off'` **|dr|**

  Indicates whether to additionally compute un-counterpoise corrected monomers and thus obtain an estimate for the basis set superposition error.

Examples

```
>>> # [1] counterpoise-corrected mp2 interaction energy
>>> cp('dfmp2')
```

`wrappers.`**`database`**(*name*, *db_name*[, *func*, *mode*, *cp*, *rlxd*, *symm*, *zpe*, *benchmark*, *tabulate*, *subset*])
Function to access the molecule objects and reference energies of popular chemical databases.

Aliases db()

Returns (*float*) Mean absolute deviation of the database in kcal/mol

Psi variables

`db_name DATABASE MEAN SIGNED DEVIATION`
`db_name DATABASE MEAN ABSOLUTE DEVIATION`
`db_name DATABASE ROOT-MEAN-SQUARE DEVIATION`

Note: It is very easy to make a database from a collection of xyz files using the script `$PSIDATADIR/databases/ixyz2database.pl`. See **'Creating a New Database'_** for details.

Caution: Some features are not yet implemented. Buy a developer some coffee.
•In sow/reap mode, use only global options (e.g., the local option set by `set scf scf_type df` will not be respected).

Parameters

- **name** (*string*) – `'scf'` || `'sapt0'` || `'ccsd(t)'` || etc.

  First argument, usually unlabeled. Indicates the computational method to be applied to the database. May be any valid argument to `driver.energy()`.

- **db_name** (*string*) – `'BASIC'` || `'S22'` || `'HTBH'` || etc.

  Second argument, usually unlabeled. Indicates the requested database name, matching the name of a python file in `psi4/lib/databases`. Consult that directory for available databases and literature citations.

- **func** (*function*) – **|dl|** `energy` **|dr|** || `optimize` || `cbs`

  Indicates the type of calculation to be performed on each database member. The default performs a single-point `energy('name')`, while `optimize` perfoms a geometry optimization on each reagent, and `cbs` performs a compound single-point energy. If a nested series of python functions is intended (see **'Function Intercalls'_**), use keyword `db_func` instead of `func`.

- **mode** (*string*) – **|dl|** `'continuous'` **|dr|** || `'sow'` || `'reap'`

  Indicates whether the calculations required to complete the database are to be run in one file (`'continuous'`) or are to be farmed out in an embarrassingly parallel fashion (`'sow'`/`'reap'`). For the latter, run an initial job with `'sow'` and follow instructions in its output file.

- **cp** (*bool*) – `'on'` || **|dl|** `'off'` **|dr|**

  Indicates whether counterpoise correction is employed in computing interaction energies. Use this option and NOT the `wrappers.cp()` function for BSSE correction in database(). Option available (See Available Databases) only for databases of bimolecular complexes.

- **rlxd** (*bool*) – `'on'` || **|dl|** `'off'` **|dr|**

  Indicates whether correction for deformation energy is employed in computing interaction energies. Option available (See Available Databases) only for databases of bimolecular complexes with non-frozen monomers, e.g., HBC6.

- **symm** (*bool*) – **|dl|** `'on'` **|dr|** || `'off'`

Indicates whether the native symmetry of the database reagents is employed (`'on'`) or whether it is forced to C_1 symmetry (`'off'`). Some computational methods (e.g., SAPT) require no symmetry, and this will be set by database().

- **zpe** (*bool*) – `'on'` || **|dl|** `'off'` **|dr|**

  Indicates whether zero-point-energy corrections are appended to single-point energy values. Option valid only for certain thermochemical databases. Disabled until Hessians ready.

- **benchmark** (*string*) – **|dl|** `'default'` **|dr|** || `'S22A'` || etc.

  Indicates whether a non-default set of reference energies, if available (See Available Databases), are employed for the calculation of error statistics.

- **tabulate** (*array of strings*) – **|dl|** `[]` **|dr|** || `['scf total energy', 'natom']` || etc.

  Indicates whether to form tables of variables other than the primary requested energy. Available for any PSI variable.

- **subset** (*string or array of strings*) – Indicates a subset of the full database to run. This is a very flexible option and can be used in three distinct ways, outlined below. Note that two take a string and the last takes an array. See Available Databases for available values.

  – **`'small'` || `'large'` || `'equilibrium'`** Calls predefined subsets of the requested database, either `'small'`, a few of the smallest database members, `'large'`, the largest of the database members, or `'equilibrium'`, the equilibrium geometries for a database composed of dissociation curves.

  – **`'BzBz_S'` || `'FaOOFaON'` || `'ArNe'` || etc.** For databases composed of dissociation curves, individual curves can be called by name. Consult the database python files for available molecular systems. The choices for this keyword are case sensitive and must match the database python file

  – **`[1,2,5]` || `['1','2','5']` || `['BzMe-3.5', 'MeMe-5.0']` || etc.** Specify a list of database members to run. Consult the database python files for available molecular systems. The choices for this keyword are case sensitive and must match the database python file

Examples

```
>>> # [1] Two-stage SCF calculation on short, equilibrium, and long helium dimer
>>> db('scf','RGC10',cast_up='sto-3g',subset=['HeHe-0.85','HeHe-1.0','HeHe-1.5'], tabulate=['scf

>>> # [2] Counterpoise-corrected interaction energies for three complexes in S22
>>> #     Error statistics computed wrt an old benchmark, S22A
>>> database('dfmp2','S22',cp=1,subset=[16,17,8],benchmark='S22A')

>>> # [3] SAPT0 on the neon dimer dissociation curve
>>> db('sapt0',subset='NeNe',cp=0,symm=0,db_name='RGC10')

>>> # [4] Optimize system 1 in database S22, producing tables of scf and mp2 energy
>>> db('mp2','S22',db_func=optimize,subset=[1], tabulate=['mp2 total energy','current energy'])

>>> # [5] CCSD on the smallest systems of HTBH, a hydrogen-transfer database
>>> database('ccsd','HTBH',subset='small', tabulate=['ccsd total energy', 'mp2 total energy'])
```

CHAPTER

FOUR

OUTPUT

At the beginning of a database job is printed a listing of the individual system calculations which will be performed. The output snippet below is from the example job [1] above. It shows each reagent required for the subset of database reactions requested. Note that this is an un-counterpoise-corrected example, and the wrapper is smart enough to compute only once the monomer whose energy will be subtracted from each of the three dimers.

```
RGC1-HeHe-0.85-dimer
RGC1-He-mono-unCP
RGC1-HeHe-1.0-dimer
RGC1-HeHe-1.5-dimer
```

At the end of the job, the Requested Energy table is printed that gives the total energies for the requested model chemistry for each reagent and each reaction, as well as the stoichoimetric weights by which the reagent energies are transfromed into the reaction energy. In this case, the dimer is +1 and the monomer is -2, indicating the the interaction energy is computed from dimer less first monomer less second (identical) monomer. Error statistics are computed with respect to the reference energies stored in the database. One of these, the mean absolute deviation, is returned by the wrapper as an ordinary Python variable. (For databases without a stored reference energy, e.g., BASIC, large and meaningless numbers are printed for error.) The other two tables tabulate the PSI variables requested through keyword `tabulate`, in this case the total SCF energy and the number of atoms in each reagent.

```
==> Scf Total Energy <==

-----------------------------------------------------------------------------------
         Reaction          Reaction Value               Reagent 1       Reagent 2
                                                          Value Wt        Value Wt
-----------------------------------------------------------------------------------
   RGC1-HeHe-0.85              0.00011520         -5.71020576  1   -2.85516048 -2
    RGC1-HeHe-1.0              0.00000153         -5.71031943  1   -2.85516048 -2
    RGC1-HeHe-1.5             -0.00000000         -5.71032096  1   -2.85516048 -2
-----------------------------------------------------------------------------------

==> Natom <==

-----------------------------------------------------------------------------------
         Reaction          Reaction Value               Reagent 1       Reagent 2
                                                          Value Wt        Value Wt
-----------------------------------------------------------------------------------
   RGC1-HeHe-0.85              0.00000000          2.00000000  1    1.00000000 -2
    RGC1-HeHe-1.0              0.00000000          2.00000000  1    1.00000000 -2
    RGC1-HeHe-1.5              0.00000000          2.00000000  1    1.00000000 -2
-----------------------------------------------------------------------------------

==> Requested Energy <==
-----------------------------------------------------------------------------------
         Reaction     Reaction Energy      Error       Reagent 1       Reagent 2
                         Ref     Calc [kcal/mol]          [H] Wt          [H] Wt
-----------------------------------------------------------------------------------
   RGC1-HeHe-0.85     0.0376   0.0723     0.0347  -5.71020576  1  -2.85516048 -2
    RGC1-HeHe-1.0    -0.0219   0.0010     0.0228  -5.71031943  1  -2.85516048 -2
    RGC1-HeHe-1.5    -0.0029  -0.0000     0.0029  -5.71032096  1  -2.85516048 -2
-----------------------------------------------------------------------------------
      Minimal Dev                         0.0029
      Maximal Dev                         0.0347
  Mean Signed Dev                         0.0201
Mean Absolute Dev                         0.0201
          RMS Dev                         0.0240
-----------------------------------------------------------------------------------
```

CHAPTER FIVE

AVAILABLE DATABASES

Below are documented for particular databases the availibility of the generic database function options **cp**, **rlxd**, **benchmark**, and the string options for **subset**. The full reagent member list, which can also be used in conjunction with **subset**, is not included here for consideration of space and may be found in the database file. The database Python files are very readable and should be consulted for more particular questions.

ACENES

Database of Ed and Rob's favorite linear acene dimers.
Geometries from nowhere special, and reference energies undefined.

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`
- **subset**
  - `'small'`
  - `'large'`
  - `'FIRST3'` benzene, napthalene, and anthracene dimers
  - `'FIRST5'` benzene - pentacene dimers
  - `'FIRST10'` benzene - decacene dimers

BAKERJCC93

Database of molecules that are challenging to optimize.
Geometries from Baker J. Comput. Chem. 14 1085 (1993), as reported in Bakken and Helgaker, J. Chem. Phys. 117, 9160 (2002), with a few further corrections.
No reference energies defined.

- **cp** `'off'`
- **rlxd** `'off'`
- **subset**

  - 'small'
  - 'large'

BASIC

Database of simple molecules, mostly for testing.
Geometries from nowhere special, and no reference energies defined.

- **cp** 'off'
- **rlxd** 'off'
- **subset** ['h2o','nh3','ch4']

BBI

Database (Merz) of protein backbone-backbone interactions.
Geometries from Kenneth Merz Group, Univ. of Florida.
Reference interaction energies from Sherrill group, Georgia Tech.

- **cp** 'off' || 'on'
- **rlxd** 'off'

CFLOW

Database of extended conjugated bimolecular systems.
Geometries and Reference interaction energies from the following articles:
    Polyene geometries from Marshall et al. JCTC 6 3681 (2010).
    Polyene reference interaction energies from Sherrill group by ccsd(t**)-f12b/heavy-aug-cc-pvdz.
    Acene geometries (except benzene) from Sherrill group by df-mp2/cc-pvtz c.2011.
    Benzene geometry from NBC10 database and citations therein.
    Acene reference interaction energies (incl. benzene dimer) from Sherrill group by ccsd(t**)-f12b/aug-cc-pvdz.
    Buckybowl (Pulay-labeled) geometries from Sherrill group by PBE1PBE/6-31G*, following Pulay's instructions in Janowski et al. CPL 512 155 (2011).
    Buckybowl (Pulay-labeled) reference interaction energies from Janowski et al. CPL 512 155 (2011).
    Buckyware (Grimme-labeled) geometries from Grimme PCCP 12 7091 (2010).
    Buckyware (Grimme-labeled) reference interaction energies from Grimme PCCP 12 7091 (2010) by B97-D2/TZVP.
    Collection into CFLOW by Parrish et al. XXX XXX XXXXXX (2012).

- **cp** 'off' || 'on'
- **rlxd** 'off'
- **subset**

- 'small'
- 'large'
- 'equilibrium'
- 'Polyenes' equilibrium for linear polyene dimers for 2 through 16 monomer carbons
- 'cBzBz' 5-point dissociation curve for benzene dimer
- 'c2BzBz' 5-point dissociation curve for napthalene-benzene complex
- 'c2Bz2Bz' 5-point dissociation curve for napthalene dimer
- 'c3Bz2Bz' 5-point dissociation curve for anthracene-napthalene complex
- 'c3Bz3Bz' 5-point dissociation curve for anthracene dimer
- 'c4Bz3Bz' 5-point dissociation curve for tetracene-anthracene complex
- 'Arenes' equilibrium for benzene dimer through tetracene-anthracene complex linear arenes
- 'cArenes' 5-point curves around benzene dimer through tetracene-anthracene complex linear arenes
- 'cPulay' 4-point dissociation curve for bowl-in-bowl corannulene dimer
- 'Pulay' Pulay bowl-in-bowl corannulene dimer dissociation curve and extra point
- 'Grimme60' Grimme corannulene dimer, C60 @ buckybowl, and C60 @ buckycatcher
- 'Grimme70' Grimme C70 @ buckycatcher at three orientations
- 'Paper' linear polyene dimers, equilibrium arene complexes, Pulay corannulene dimer curve, and Grimme corannulene dimer and C60 complexes
- 'cPaper' linear polyene dimers, arene complex curves, Pulay corannulene dimer curve, and Grimme corannulene dimer and C60 complexes

---

CORE

Database of Pulay corannulene structures. Subsumed into CFLOW.

- **cp** 'off' || 'on'
- **rlxd** 'off'

G2

Database of thermodynamic reactions.
WIP

- **cp** 'off'
- **rlxd** 'off'

HBC6

Database (Sherrill) of interaction energies for dissociation curves of doubly hydrogen-bonded bimolecular complexes.

Geometries from Thanthiriwatte et al. JCTC 7 88 (2011).
Reference interaction energies from Marshall et al. JCP 135 194102 (2011).

- **cp** `'off'` || `'on'`
- **rlxd** `'off'` || `'on'`
- **benchmark**
  - `'HBC60'` Thanthiriwatte et al. JCTC 7 88 (2011).
  - |dl| `'HBC6A'` |dr| Marshall et al. JCP 135 194102 (2011).
  - `'HBC6ARLX'` Sherrill group, unpublished.
- **subset**
  - `'small'`
  - `'large'`
  - `'equilibrium'`

HSG

Database (Merz) of interaction energies for bimolecular complexes from protein-indinavir reaction site.
Geometries from Faver et al. JCTC 7 790 (2011).
Reference interaction energies from Marshall et al. JCP 135 194102 (2011).

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`
- **benchmark**
  - `'HSG0'` Faver et al. JCTC 7 790 (2011).
  - |dl| `'HSGA'` |dr| Marshall et al. JCP 135 194102 (2011).
- **subset**
  - `'small'`
  - `'large'`

HTBH

Database (Truhlar) of hydrogen-transfer barrier height reactions.
Geometries from Truhlar and coworkers at site
http://t1.chem.umn.edu/misc/database_group/database_therm_bh/raw_geom.cgi .
Reference energies from Zhao et al. JPCA, 109 2012-2018 (2005) doi: 10.1021/jp045141s [in supporting information].

- **cp** `'off'`
- **rlxd** `'off'`

- **subset**
  - `'small'`
  - `'large'`

JSCH

Database (Hobza) of interaction energies for nucelobase pairs.
Geometries and reference interaction energies from Jurecka et al. PCCP 8 1985 (2006).
Corrections implemented from footnote 92 of Burns et al., JCP 134 084107 (2011).

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`
- **subset**
  - `'small'`
  - `'large'`

NBC10

Database (Sherrill) of interaction energies for dissociation curves of dispersion-bound bimolecular complexes.
Geometries and Reference interaction energies from the following articles:
    Benzene Dimers from Sherrill et al. JPCA 113 10146 (2009).
    Benzene-Hydrogen Sulfide from Sherrill et al. JPCA 113 10146 (2009).
    Benzene-Methane from Sherrill et al. JPCA 113 10146 (2009).
    Methane Dimer from Takatani et al. PCCP 9 6106 (2007).
    Pyridine Dimers from Hohenstein et al. JPCA 113 878 (2009).
    Collection into NBC10 from Burns et al. JCP 134 084107 (2011).
    Reference from Marshall et al. JCP 135 194102 (2011).

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`
- **benchmark**
  - `'NBC100'` Burns et al. JCP 134 084107 (2011).
  - **|dl|** `'NBC10A'` **|dr|** Marshall et al. JCP 135 194102 (2011).
- **subset**
  - `'small'`
  - `'large'`
  - `'equilibrium'`
  - `'BzBz_S'` dissociation curve for benzene dimer, sandwich
  - `'BzBz_T'` dissociation curve for benzene dimer, t-shaped

  - `'BzBz_PD34'` dissociation curve for benzene dimer, parallel displaced by 3.4A
  - `'BzH2S'` dissociation curve for benzene-H2S
  - `'BzMe'` dissociation curve for benzene-methane
  - `'MeMe'` dissociation curve for methane dimer
  - `'PyPy_S2'` dissociation curve for pyridine dimer, sandwich
  - `'PyPy_T3'` dissociation curve for pyridine dimer, t-shaped
  - `'BzBz_PD32'` dissociation curve for benzene dimer, parallel displaced by 3.2A
  - `'BzBz_PD36'` dissociation curve for benzene dimer, parallel displaced by 3.6A

NHTBH

Database (Truhlar) of non-hydrogen-transfer barrier height reactions.
Geometries and Reaction energies from Truhlar and coworkers at site
http://t1.chem.umn.edu/misc/database_group/database_therm_bh/non_H.htm.

- **cp** `'off'`
- **rlxd** `'off'`
- **subset**
  - `'small'`
  - `'large'`

RGC10

Database (Sherrill) of interaction energies for dissociation curves of rare-gas biatomic complexes.
Geometries and reference interaction energies from Tang et al. JCP 118 4976 (2003).

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`
- **subset**
  - `'small'`
  - `'large'`
  - `'equilibrium'`
  - `'HeHe'` 18-point dissociation curve for helium dimer
  - `'HeNe'` 18-point dissociation curve for helium-neon complex
  - `'HeAr'` 18-point dissociation curve for helium-argon complex
  - `'HeKr'` 18-point dissociation curve for helium-krypton complex
  - `'NeNe'` 18-point dissociation curve for neon dimer

   - `'NeAr'` 18-point dissociation curve for neon-argon complex
   - `'NeKr'` 18-point dissociation curve for neon-krypton complex
   - `'ArAr'` 18-point dissociation curve for argon dimer
   - `'ArKr'` 18-point dissociation curve for argon-krypton complex
   - `'KrKr'` 18-point dissociation curve for krypton dimer

S22

Database (Hobza) of interaction energies for bimolecular complexes.
Geometries from Jurecka et al. PCCP 8 1985 (2006).
Reference interaction energies from Marshall et al. JCP 135 194102 (2011).

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`
- **benchmark**
   - `'S220'` Jurecka et al. PCCP 8 1985 (2006).
   - `'S22A'` Takatani et al. JCP 132 144104 (2010).
   - |dl| `'S22B'` |dr| Marshall et al. JCP 135 194102 (2011).
- **subset**
   - `'small'` water dimer, methane dimer, ethene-ethine
   - `'large'` adenine-thymine
   - `'HB'` hydrogen-bonded systems
   - `'MX'` mixed-influence systems
   - `'DD'` dispersion-dominated systems

S22by5

Database (Hobza) of interaction energies for dissociation curves of bimolecular complexes.
Geometries and reference interaction energies from Grafova et al. JCTC 6 2365 (2010).
Note that the S22by5-N-1.0 members are essentially the same geometries as S22-N (there's trivial round-off error) but the reference interaction energies for S22by5 are of lower quality than those of S22.

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`
- **subset**
   - `'small'`
   - `'large'`
   - `'equilibrium'`

- `'mol1'` five-point (0.9, 1.0, 1.2, 1.5, 2.0) $\times R_{eq}$ dissociation curve for molecule 1
- ...
- `'mol22'` five-point (0.9, 1.0, 1.2, 1.5, 2.0) $\times R_{eq}$ dissociation curve for molecule 22

S66

Database (Hobza) of interaction energies for bimolecular complexes.
Geometries and reference energies from Rezac et al. JCTC 7 2427 (2011).

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`
- **subset**
  - `'small'`
  - `'large'`
  - `'HB'` hydrogen-bonded systems
  - `'MX'` mixed-influence systems
  - `'DD'` dispersion-dominated systems

SSI

Database (Merz) of interaction energies for protein sidechain-sidechain interactions.
Geometries from Kenneth Merz Group, Univ. of Florida.
Reference interaction energies from <Reference>.

- **cp** `'off'` || `'on'`
- **rlxd** `'off'`

For many of the PSI4 Python functions described above, it makes scientific sense that they could be called in combination. For instance, one could optimize all the reagents in a database or compute a counterpoise-corrected interaction energy with an extrapolated method. The table below outlines permitted intercalls between functions, showing that db(opt(cbs(energy()))) is allowed, while db(cp(energy())) is not. This table is not yet validated for calls with cp().

Caller	**Callee**				
	cp	**db**	**opt**	**cbs**	**energy**
cp		—	Y	Y	Y
db	—		Y	Y	Y
opt	—	—		Y	Y
cbs	—	—	—		Y
energy	—	—	—	—	

- The command db(opt(cbs(energy()))) is actually expressed as `db(..., db_func=opt, opt_func=cbs)`. The perhaps expected final argument of `cbs_func=energy` is not necessary since energy() is always the function called by default. Also, the outermost internal function call (`db_func` above can be called as just `func`. Several examples of intercalls between Python functions can be found in sample input *pywrap_all*.
- All keyword arguments are passed along to each function traversed in the

Python driver, so there should be no concern for separating them, grouping them, or designating them for a particular function when undertaking a nested calculation. Where the same keyword is used by multiple functions, prefixes are added, e.g., **db_mode** and **opt_mode**.

- Function intercalls should not be used in sow/reap mode.

NAME PsiMod

FILE (built-in)

CLASSES

Boost.Python.enum(__builtin__.int) DiagonalizeOrder PsiReturnType

Boost.Python.instance(__builtin__.object) Arguments BasisSet BasisSetParser

Gaussian94BasisSetParser

CdSalcList Checkpoint DFChargeFitter Environment ExternalPotential FittingMetric Functional GridProp IO IOManager IntVector Matrix MatrixFactory MintsHelper MoldenWriter Molecule MultipoleSymmetry NBOWriter OEProp PetiteList PointGroup Process PseudoTrial SOBasisSet SuperFunctional Symmetry-Operation Vector Vector3 Wavefunction

HF RHF(HF, Wavefunction)

matrix_vector

class Arguments(Boost.Python.instance)

Method resolution order:
Arguments
Boost.Python.instance
__builtin__.object

Methods defined here:

__getitem__(...)
__getitem__((Arguments)arg1, (int)arg2) -> str :
docstring

__init__(...)
__init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

Data and other attributes defined here:

__instance_size__ = 40

—————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

—————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class BasisSet(Boost.Python.instance)

docstring

Method resolution order:
    BasisSet
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__reduce__ = <unnamed Boost.Python function>(...)

max_am(...)
    max_am((BasisSet)arg1) -> int :
        docstring

nao(...)
    nao((BasisSet)arg1) -> int :
        docstring

nbf(...)
    nbf((BasisSet)arg1) -> int :
        docstring

nprimitive(...)
    nprimitive((BasisSet)arg1) -> int :
        docstring

nshell(...)
    nshell((BasisSet)arg1) -> int :
        docstring

print_detail_out(...)
    print_detail_out((BasisSet)arg1) -> None :

docstring

print_out(...)
print_out((BasisSet)arg1) -> None :
docstring

Static methods defined here:

construct(...)
construct((BasisSetParser)arg1, (Molecule)arg2, (str)arg3) -> BasisSet :
docstring

make_filename(...)
make_filename((str)arg1) -> str :
docstring

Data and other attributes defined here:

__init__ = <built-in function __init__>
Raises an exception
This class cannot be instantiated from Python

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

class BasisSetParser(Boost.Python.instance)

docstring

Method resolution order:
BasisSetParser
Boost.Python.instance
__builtin__.object

Methods defined here:

__reduce__ = <unnamed Boost.Python function>(...)

————————————————————————————-

Data and other attributes defined here:

__init__ = <built-in function __init__>

    Raises an exception
    This class cannot be instantiated from Python

————————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

————————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>

    T.__new__(S, ...) -> a new object with type S, a subtype of T

class CdSalcList(Boost.Python.instance)

docstring

Method resolution order:

    CdSalcList
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__reduce__ = <unnamed Boost.Python function>(...)

matrix(...)

    matrix((CdSalcList)arg1) -> Matrix :
        docstring

print_out(...)

    print_out((CdSalcList)arg1) -> None :
        docstring

————————————————————————————-

Data and other attributes defined here:

__init__ = <built-in function __init__>

    Raises an exception

This class cannot be instantiated from Python

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class Checkpoint(Boost.Python.instance)

docstring

Method resolution order:
    Checkpoint
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__((object)arg1, (IO)arg2, (int)arg3) -> None

__reduce__ = <unnamed Boost.Python function>(...)

Static methods defined here:

shared_object(...)
    shared_object() -> Checkpoint :
        docstring

Data descriptors defined here:

disp
    docstring

e_t
    docstring

eccsd

docstring

ecorr
docstring

efzc
docstring

emp2
docstring

enuc
docstring

eref
docstring

escf
docstring

etot
docstring

label
docstring

———————————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

———————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

class DFChargeFitter(Boost.Python.instance)

docstring

```
Method resolution order:
    DFChargeFitter
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__( (object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

d(...)
    d( (DFChargeFitter)arg1) -> Vector :
        docstring

fit(...)
    fit( (DFChargeFitter)arg1) -> Vector :
        docstring

setAuxiliary(...)
    setAuxiliary( (DFChargeFitter)arg1, (BasisSet)arg2) -> None :
        docstring

setD(...)
    setD( (DFChargeFitter)arg1, (Matrix)arg2) -> None :
        docstring

setPrimary(...)
    setPrimary( (DFChargeFitter)arg1, (BasisSet)arg2) -> None :
        docstring

----------------------------------------------------------------------
Data and other attributes defined here:

__instance_size__ = 32

----------------------------------------------------------------------
Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

----------------------------------------------------------------------
Data and other attributes inherited from Boost.Python.instance:
```

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class DiagonalizeOrder(Boost.Python.enum)

docstring

Method resolution order:
    DiagonalizeOrder
    Boost.Python.enum
    __builtin__.int
    __builtin__.object

Data and other attributes defined here:

Ascending = PsiMod.DiagonalizeOrder.Ascending

Descending = PsiMod.DiagonalizeOrder.Descending

names = {‘Ascending’: PsiMod.DiagonalizeOrder.Ascending, ‘Descending’:...

values = {1: PsiMod.DiagonalizeOrder.Ascending, 3: PsiMod.DiagonalizeO...

———————————————————————-

Methods inherited from Boost.Python.enum:

__repr__(...)
    x.__repr__() <==> repr(x)

__str__(...)
    x.__str__() <==> str(x)

———————————————————————-

Data descriptors inherited from Boost.Python.enum:

name

———————————————————————-

Methods inherited from __builtin__.int:

__abs__(...)
    x.__abs__() <==> abs(x)

__add__(...)
    x.__add__(y) <==> x+y

__and__(...)
    x.__and__(y) <==> x&y

__cmp__(...)
    x.__cmp__(y) <==> cmp(x,y)

__coerce__(...)
    x.__coerce__(y) <==> coerce(x, y)

__div__(...)
    x.__div__(y) <==> x/y

__divmod__(...)
    x.__divmod__(y) <==> divmod(x, y)

__float__(...)
    x.__float__() <==> float(x)

__floordiv__(...)
    x.__floordiv__(y) <==> x//y

__format__(...)

__getattribute__(...)
    x.__getattribute__('name') <==> x.name

__getnewargs__(...)

__hash__(...)
    x.__hash__() <==> hash(x)

__hex__(...)
    x.__hex__() <==> hex(x)

__index__(...)
    x[y:z] <==> x[y.__index__():z.__index__()]

__int__(...)
    x.__int__() <==> int(x)

__invert__(...)
    x.__invert__() <==> ~x

__long__(...)
    x.__long__() <==> long(x)

__lshift__(...)
    x.__lshift__(y) <==> x<<y

__mod__(...)
x.__mod__(y) <==> x%y

__mul__(...)
x.__mul__(y) <==> x*y

__neg__(...)
x.__neg__() <==> -x

__nonzero__(...)
x.__nonzero__() <==> x != 0

__oct__(...)
x.__oct__() <==> oct(x)

__or__(...)
x.__or__(y) <==> x|y

__pos__(...)
x.__pos__() <==> +x

__pow__(...)
x.__pow__(y[, z]) <==> pow(x, y[, z])

__radd__(...)
x.__radd__(y) <==> y+x

__rand__(...)
x.__rand__(y) <==> y&x

__rdiv__(...)
x.__rdiv__(y) <==> y/x

__rdivmod__(...)
x.__rdivmod__(y) <==> divmod(y, x)

__rfloordiv__(...)
x.__rfloordiv__(y) <==> y//x

__rlshift__(...)
x.__rlshift__(y) <==> y<<x

__rmod__(...)
x.__rmod__(y) <==> y%x

__rmul__(...)
x.__rmul__(y) <==> y*x

__ror__(...)
    x.__ror__(y) <==> y|x

__rpow__(...)
    y.__rpow__(x[, z]) <==> pow(x, y[, z])

__rrshift__(...)
    x.__rrshift__(y) <==> y>>x

__rshift__(...)
    x.__rshift__(y) <==> x>>y

__rsub__(...)
    x.__rsub__(y) <==> y-x

__rtruediv__(...)
    x.__rtruediv__(y) <==> y/x

__rxor__(...)
    x.__rxor__(y) <==> y^x

__sub__(...)
    x.__sub__(y) <==> x-y

__truediv__(...)
    x.__truediv__(y) <==> x/y

__trunc__(...)
    Truncating an Integral returns itself.

__xor__(...)
    x.__xor__(y) <==> x^y

bit_length(...)
    int.bit_length() -> int

    Number of bits necessary to represent self in binary.
    >>> bin(37)
    '0b100101'
    >>> (37).bit_length()
    6

conjugate(...)
    Returns self, the complex conjugate of any int.

———————————————————————————————-

Data descriptors inherited from __builtin__.int:

denominator
    the denominator of a rational number in lowest terms

imag
    the imaginary part of a complex number

numerator
    the numerator of a rational number in lowest terms

real
    the real part of a complex number

Data and other attributes inherited from __builtin__.int:

__new__ = <built-in method __new__ of type object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class Environment(Boost.Python.instance)

Method resolution order:
    Environment
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__getitem__(...)
    __getitem__((Environment)arg1, (str)arg2) -> str :
        docstring

__init__(...)
    __init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

Data and other attributes defined here:

__instance_size__ = 352

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class ExternalPotential(Boost.Python.instance)

docstring

Method resolution order:
    ExternalPotential
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

addBasis(...)
    addBasis((ExternalPotential)arg1, (BasisSet)arg2, (Vector)arg3) -> None :
        docstring

addCharge(...)
    addCharge((ExternalPotential)arg1, (float)arg2, (float)arg3, (float)arg4, (float)arg5) -> None :
        docstring

clear(...)
    clear((ExternalPotential)arg1) -> None :
        docstring

computePotentialMatrix(...)
    computePotentialMatrix((ExternalPotential)arg1, (BasisSet)arg2) -> Matrix :
        docstring

print_out(...)
    print_out((ExternalPotential)arg1) -> None :
        docstring

setName(...)
    setName((ExternalPotential)arg1, (str)arg2) -> None :
        docstring

———————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

———————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class FittingMetric(Boost.Python.instance)

docstring

Method resolution order:
    FittingMetric
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

form_QR_inverse(...)
    form_QR_inverse((FittingMetric)arg1, (float)arg2) -> None :
        docstring

form_cholesky_inverse(...)
    form_cholesky_inverse((FittingMetric)arg1) -> None :
        docstring

form_eig_inverse(...)
    form_eig_inverse((FittingMetric)arg1, (float)arg2) -> None :
        docstring

form_fitting_metric(...)
    form_fitting_metric((FittingMetric)arg1) -> None :

docstring

form_full_inverse(...)
  form_full_inverse((FittingMetric)arg1) -> None :
    docstring

get_algorithm(...)
  get_algorithm((FittingMetric)arg1) -> str :
    docstring

get_metric(...)
  get_metric((FittingMetric)arg1) -> Matrix :
    docstring

get_pivots(...)
  get_pivots((FittingMetric)arg1) -> IntVector :
    docstring

get_reverse_pivots(...)
  get_reverse_pivots((FittingMetric)arg1) -> IntVector :
    docstring

is_inverted(...)
  is_inverted((FittingMetric)arg1) -> bool :
    docstring

is_poisson(...)
  is_poisson((FittingMetric)arg1) -> bool :
    docstring

———————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

———————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
  T.__new__(S, ...) -> a new object with type S, a subtype of T

class Functional(Boost.Python.instance)

docstring

Method resolution order:

Functional
Boost.Python.instance
__builtin__.object

Methods defined here:

__reduce__ = <unnamed Boost.Python function>(...)

computeRKSFunctional(...)
computeRKSFunctional((Functional)arg1, (object)arg2) -> None :
docstring

computeUKSFunctional(...)
computeUKSFunctional((Functional)arg1, (object)arg2) -> None :
docstring

get_citation(...)
get_citation((Functional)arg1) -> str :
docstring

get_density_cutoff(...)
get_density_cutoff((Functional)arg1) -> float :
docstring

get_deriv(...)
get_deriv((Functional)arg1) -> int :
docstring

get_description(...)
get_description((Functional)arg1) -> str :
docstring

get_name(...)
get_name((Functional)arg1) -> str :
docstring

get_npoints(...)
get_npoints((Functional)arg1) -> int :
docstring

get_parameters(...)
get_parameters((Functional)arg1) -> object :
docstring

get_parameters_string(...)
: get_parameters_string((Functional)arg1) -> str :
: docstring

is_gga(...)
: is_gga((Functional)arg1) -> bool :
: docstring

is_meta(...)
: is_meta((Functional)arg1) -> bool :
: docstring

set_citation(...)
: set_citation((Functional)arg1, (str)arg2) -> None :
: docstring

set_density_cutoff(...)
: set_density_cutoff((Functional)arg1, (float)arg2) -> None :
: docstring

set_deriv(...)
: set_deriv((Functional)arg1, (int)arg2) -> None :
: docstring

set_description(...)
: set_description((Functional)arg1, (str)arg2) -> None :
: docstring

set_name(...)
: set_name((Functional)arg1, (str)arg2) -> None :
: docstring

set_npoints(...)
: set_npoints((Functional)arg1, (int)arg2) -> None :
: docstring

set_parameter(...)
: set_parameter((Functional)arg1, (str)arg2, (float)arg3) -> None :
: docstring

set_parameters(...)
: set_parameters((Functional)arg1, (object)arg2) -> None :
: docstring

Static methods defined here:

available_functionals(...)
    available_functionals() -> str :
        docstring

available_names(...)
    available_names() -> object :
        docstring

create_functional(...)
    create_functional((str)arg1, (int)arg2, (int)arg3) -> Functional :
        docstring

———————————————————————————-

Data and other attributes defined here:

__init__ = <built-in function __init__>
    Raises an exception
    This class cannot be instantiated from Python

———————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class Gaussian94BasisSetParser(BasisSetParser)

docstring

Method resolution order:
    Gaussian94BasisSetParser
    BasisSetParser
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__((object)arg1) -> None

```
    __reduce__ = <unnamed Boost.Python function>(...)

    ----------------------------------------------------------------------
    Data and other attributes defined here:

    __instance_size__ = 32

    ----------------------------------------------------------------------
    Data descriptors inherited from Boost.Python.instance:

    __dict__

    __weakref__

    ----------------------------------------------------------------------
    Data and other attributes inherited from Boost.Python.instance:

    __new__ = <built-in method __new__ of Boost.Python.class object>
        T.__new__(S, ...) -> a new object with type S, a subtype of T
```

class GridProp(Boost.Python.instance)

```
    docstring

    Method resolution order:
        GridProp
        Boost.Python.instance
        __builtin__.object

    Methods defined here:

    __init__(...)
        __init__( (object)arg1) -> None

    __reduce__ = <unnamed Boost.Python function>(...)

    add(...)
        add( (GridProp)arg1, (str)arg2) -> None :
            docstring

    add_alpha_mo(...)
        add_alpha_mo( (GridProp)arg1, (int)arg2, (int)arg3) -> None :
            docstring

    add_basis_fun(...)
        add_basis_fun( (GridProp)arg1, (int)arg2, (int)arg3) -> None :
            docstring
```

add_beta_mo(...)
  add_beta_mo((GridProp)arg1, (int)arg2, (int)arg3) -> None :
    docstring

build_grid_overages(...)
  build_grid_overages((GridProp)arg1, (float)arg2) -> None :
    docstring

compute(...)
  compute((GridProp)arg1) -> None :
    docstring

get_l(...)
  get_l((GridProp)arg1, (int)arg2) -> float :
    docstring

get_n(...)
  get_n((GridProp)arg1, (int)arg2) -> int :
    docstring

get_o(...)
  get_o((GridProp)arg1, (int)arg2) -> float :
    docstring

set_caxis(...)
  set_caxis((GridProp)arg1, (float)arg2, (float)arg3) -> None :
    docstring

set_filename(...)
  set_filename((GridProp)arg1, (str)arg2) -> None :
    docstring

set_format(...)
  set_format((GridProp)arg1, (str)arg2) -> None :
    docstring

set_l(...)
  set_l((GridProp)arg1, (float)arg2, (float)arg3, (float)arg4) -> None :
    docstring

set_n(...)
  set_n((GridProp)arg1, (int)arg2, (int)arg3, (int)arg4) -> None :
    docstring

set_o(...)
  set_o((GridProp)arg1, (float)arg2, (float)arg3, (float)arg4) -> None :
    docstring

——————————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

——————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

——————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class HF(Wavefunction)

docstring

Method resolution order:
    HF
    Wavefunction
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__reduce__ = <unnamed Boost.Python function>(...)

——————————————————————————-

Data and other attributes defined here:

__init__ = <built-in function __init__>
    Raises an exception
    This class cannot be instantiated from Python

——————————————————————————-

Methods inherited from Wavefunction:

Ca(...)
    Ca((Wavefunction)arg1) -> Matrix :
        docstring

Cb(...)

Cb((Wavefunction)arg1) -> Matrix :
docstring

Da(...)
Da((Wavefunction)arg1) -> Matrix :
docstring

Db(...)
Db((Wavefunction)arg1) -> Matrix :
docstring

Fa(...)
Fa((Wavefunction)arg1) -> Matrix :
docstring

Fb(...)
Fb((Wavefunction)arg1) -> Matrix :
docstring

add_postiteration_callback(...)
add_postiteration_callback((Wavefunction)arg1, (object)arg2) -> None :
docstring

add_preiteration_callback(...)
add_preiteration_callback((Wavefunction)arg1, (object)arg2) -> None :
docstring

basisset(...)
basisset((Wavefunction)arg1) -> BasisSet :
docstring

energy(...)
energy((Wavefunction)arg1) -> float :
docstring

epsilon_a(...)
epsilon_a((Wavefunction)arg1) -> Vector :
docstring

epsilon_b(...)
epsilon_b((Wavefunction)arg1) -> Vector :
docstring

frequencies(...)
frequencies((Wavefunction)arg1) -> Vector :
docstring

gradient(...)
    gradient((Wavefunction)arg1) -> Matrix :
        docstring

nirrep(...)
    nirrep((Wavefunction)arg1) -> int :
        docstring

nmo(...)
    nmo((Wavefunction)arg1) -> int :
        docstring

nso(...)
    nso((Wavefunction)arg1) -> int :
        docstring

sobasisset(...)
    sobasisset((Wavefunction)arg1) -> SOBasisSet :
        docstring

———————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class IO(Boost.Python.instance)

docstring

Method resolution order:
    IO
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

close(...)
    close((IO)arg1, (int)arg2, (int)arg3) -> None :
        docstring

open(...)
    open((IO)arg1, (int)arg2, (int)arg3) -> None :
        docstring

open_check(...)
    open_check((IO)arg1, (int)arg2) -> int :
        docstring

rehash(...)
    rehash((IO)arg1, (int)arg2) -> None :
        docstring

state(...)
    state((IO)arg1) -> int :
        docstring

tocclean(...)
    tocclean((IO)arg1, (int)arg2, (str)arg3) -> None :
        docstring

tocprint(...)
    tocprint((IO)arg1, (int)arg2) -> None :
        docstring

tocwrite(...)
    tocwrite((IO)arg1, (int)arg2) -> None :
        docstring

———————————————————————-

Static methods defined here:

change_file_namespace(...)
    change_file_namespace((int)arg1, (str)arg2, (str)arg3) -> None :
        docstring

get_default_namespace(...)
    get_default_namespace() -> str :
        docstring

set_default_namespace(...)
    set_default_namespace((str)arg1) -> None :
        docstring

shared_object(...)
    shared_object() -> IO

---

Data and other attributes defined here:

__instance_size__ = 32

---

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

---

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class IOManager(Boost.Python.instance)

docstring

Method resolution order:
    IOManager
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

crashclean(...)
    crashclean((IOManager)arg1) -> None :
        docstring

get_default_path(...)
    get_default_path((IOManager)arg1) -> str :
        docstring

get_file_path(...)
    get_file_path((IOManager)arg1, (int)arg2) -> str :

```
            docstring

mark_file_for_retention(...)
        mark_file_for_retention( (IOManager)arg1, (str)arg2, (bool)arg3) -> None :
            docstring

print_out(...)
        print_out( (IOManager)arg1) -> None :
            docstring

psiclean(...)
        psiclean( (IOManager)arg1) -> None :
            docstring

set_default_path(...)
        set_default_path( (IOManager)arg1, (str)arg2) -> None :
            docstring

set_specific_path(...)
        set_specific_path( (IOManager)arg1, (int)arg2, (str)arg3) -> None :
            docstring

set_specific_retention(...)
        set_specific_retention( (IOManager)arg1, (int)arg2, (bool)arg3) -> None :
            docstring

write_scratch_file(...)
        write_scratch_file( (IOManager)arg1, (str)arg2, (str)arg3) -> None :
            docstring

----------------------------------------------------------------------
Static methods defined here:

shared_object(...)
        shared_object() -> IOManager :
            docstring

----------------------------------------------------------------------
Data and other attributes defined here:

__instance_size__ = 32

----------------------------------------------------------------------
Data descriptors inherited from Boost.Python.instance:

__dict__
```

```
    __weakref__

    ----------------------------------------------------------------------
    Data and other attributes inherited from Boost.Python.instance:

    __new__ = <built-in method __new__ of Boost.Python.class object>
        T.__new__(S, ...) -> a new object with type S, a subtype of T
```

class IntVector(Boost.Python.instance)

```
    docstring

    Method resolution order:
        IntVector
        Boost.Python.instance
        __builtin__.object

    Methods defined here:

    __init__(...)
        __init__( (object)arg1) -> None

        __init__( (object)arg1, (int)arg2) -> None

    __reduce__ = <unnamed Boost.Python function>(...)

    dim(...)
        dim( (IntVector)arg1, (int)arg2) -> int :
            docstring

    get(...)
        get( (IntVector)arg1, (int)arg2, (int)arg3) -> int :
            docstring

    nirrep(...)
        nirrep( (IntVector)arg1) -> int :
            docstring

    print_out(...)
        print_out( (IntVector)arg1) -> None :
            docstring

    set(...)
        set( (IntVector)arg1, (int)arg2, (int)arg3, (int)arg4) -> None :
            docstring

    ----------------------------------------------------------------------
    Data and other attributes defined here:
```

__instance_size__ = 32

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class Matrix(Boost.Python.instance)

docstring

Method resolution order:
    Matrix
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__getitem__(...)
    __getitem__((Matrix)arg1, (tuple)arg2) -> float :
        docstring

__init__(...)
    __init__((object)arg1) -> None

    __init__((object)arg1, (int)arg2, (int)arg3) -> None

__reduce__ = <unnamed Boost.Python function>(...)

__setitem__(...)
    __setitem__((Matrix)arg1, (tuple)arg2, (float)arg3) -> None :
        docstring

accumulate_product(...)
    accumulate_product((Matrix)arg1, (Matrix)arg2, (Matrix)arg3) -> None :
        docstring

add(...)
    add((Matrix)arg1, (Matrix)arg2) -> None :

docstring

back_transform(...)
back_transform((Matrix)arg1, (Matrix)arg2, (Matrix)arg3) -> None :
docstring

cholesky_factorize(...)
cholesky_factorize((Matrix)arg1) -> None :
docstring

cols(...)
cols((Matrix)arg1, (int)arg2) -> int :
docstring

copy_lower_to_upper(...)
copy_lower_to_upper((Matrix)arg1) -> None :
docstring

copy_upper_to_lower(...)
copy_upper_to_lower((Matrix)arg1) -> None :
docstring

diagonalize(...)
diagonalize((Matrix)arg1, (Matrix)arg2, (Vector)arg3, (DiagonalizeOrder)arg4) -> None :
docstring

exp(...)
exp((Matrix)arg1) -> None :
docstring

gemm(...)
gemm((Matrix)arg1, (bool)arg2, (bool)arg3, (float)arg4, (Matrix)arg5, (Matrix)arg6, (float)arg7) -> None :
docstring

get(...)
get((Matrix)arg1, (int)arg2, (int)arg3 [, (int)arg4]) -> float :
docstring

identity(...)
identity((Matrix)arg1) -> None :
docstring

invert(...)
invert((Matrix)arg1) -> None :
docstring

load(...)
: load((Matrix)arg1, (str)arg2) -> None :
    docstring

name(...)
: name((Matrix)arg1) -> str :
    docstring

nirrep(...)
: nirrep((Matrix)arg1) -> int :
    docstring

partial_cholesky_factorize(...)
: partial_cholesky_factorize((Matrix)arg1, (float)arg2, (bool)arg3) -> Matrix :
    docstring

power(...)
: power((Matrix)arg1, (float)arg2, (float)arg3) -> None :
    docstring

print_out(...)
: print_out((Matrix)arg1) -> None :
    docstring

project_out(...)
: project_out((Matrix)arg1, (Matrix)arg2) -> None :
    docstring

remove_symmetry(...)
: remove_symmetry((Matrix)arg1, (Matrix)arg2, (Matrix)arg3) -> None :
    docstring

rms(...)
: rms((Matrix)arg1) -> float :
    docstring

rows(...)
: rows((Matrix)arg1, (int)arg2) -> int :
    docstring

save(...)
: save((Matrix)arg1, (str)arg2, (bool)arg3, (bool)arg4, (bool)arg5) -> None :
    docstring

scale(...)
: scale((Matrix)arg1, (float)arg2) -> None :
    docstring

scale_column(...)

 scale_column((Matrix)arg1, (int)arg2, (int)arg3, (float)arg4) -> None :

  docstring

scale_row(...)

 scale_row((Matrix)arg1, (int)arg2, (int)arg3, (float)arg4) -> None :

  docstring

set(...)

 set((Matrix)arg1, (int)arg2, (int)arg3, (float)arg4) -> None :

  docstring

 set((Matrix)arg1, (int)arg2, (int)arg3, (int)arg4, (float)arg5) -> None :

  docstring

 set((Matrix)arg1, (list)arg2) -> None :

  docstring

set_name(...)

 set_name((Matrix)arg1, (str)arg2) -> None :

  docstring

subtract(...)

 subtract((Matrix)arg1, (Matrix)arg2) -> None :

  docstring

sum_of_squares(...)

 sum_of_squares((Matrix)arg1) -> float :

  docstring

symmetry(...)

 symmetry((Matrix)arg1) -> int :

  docstring

trace(...)

 trace((Matrix)arg1) -> float :

  docstring

transform(...)

 transform((Matrix)arg1, (Matrix)arg2) -> None :

  docstring

 transform((Matrix)arg1, (Matrix)arg2 [, (Matrix)arg3]) -> None :

  docstring

vector_dot(...)

vector_dot((Matrix)arg1, (Matrix)arg2) -> float :
docstring

zero(...)
zero((Matrix)arg1) -> None :
docstring

zero_diagonal(...)
zero_diagonal((Matrix)arg1) -> None :
docstring

zero_lower(...)
zero_lower((Matrix)arg1) -> None :
docstring

zero_upper(...)
zero_upper((Matrix)arg1) -> None :
docstring

———————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

———————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

class MatrixFactory(Boost.Python.instance)

docstring

Method resolution order:
MatrixFactory
Boost.Python.instance
__builtin__.object

Methods defined here:

```
__init__(...)
    __init__( (object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

create_matrix(...)
    create_matrix( (MatrixFactory)arg1) -> Matrix :
        docstring

    create_matrix( (MatrixFactory)arg1, (str)arg2) -> Matrix :
        docstring

----------------------------------------------------------------------
Static methods defined here:

shared_object(...)
    shared_object() -> MatrixFactory :
        docstring

----------------------------------------------------------------------
Data and other attributes defined here:

__instance_size__ = 32

----------------------------------------------------------------------
Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

----------------------------------------------------------------------
Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T
```

class MintsHelper(Boost.Python.instance)

```
docstring

Method resolution order:
    MintsHelper
    Boost.Python.instance
    __builtin__.object

Methods defined here:
```

__init__(...)
    __init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

ao_angular_momentum(...)
    ao_angular_momentum((MintsHelper)arg1) -> matrix_vector :
        docstring

ao_erf_eri(...)
    ao_erf_eri((MintsHelper)arg1, (float)arg2) -> Matrix :
        docstring

ao_eri(...)
    ao_eri((MintsHelper)arg1) -> Matrix :
        docstring

ao_kinetic(...)
    ao_kinetic((MintsHelper)arg1) -> Matrix :
        docstring

ao_nabla(...)
    ao_nabla((MintsHelper)arg1) -> matrix_vector :
        docstring

ao_overlap(...)
    ao_overlap((MintsHelper)arg1) -> Matrix :
        docstring

ao_potential(...)
    ao_potential((MintsHelper)arg1) -> Matrix :
        docstring

basisset(...)
    basisset((MintsHelper)arg1) -> BasisSet :
        docstring

cdsalcs(...)
    cdsalcs((MintsHelper)arg1, (int)arg2, (bool)arg3, (bool)arg4) -> CdSalcList :
        docstring

factory(...)
    factory((MintsHelper)arg1) -> MatrixFactory :
        docstring

integrals(...)
    integrals((MintsHelper)arg1) -> None :

docstring

one_electron_integrals(...)
one_electron_integrals((MintsHelper)arg1) -> None :
docstring

one_electron_integrals((MintsHelper)arg1) -> None :
docstring

petite_list(...)
petite_list((MintsHelper)arg1) -> PetiteList :
docstring

play(...)
play((MintsHelper)arg1) -> None :
docstring

so_angular_momentum(...)
so_angular_momentum((MintsHelper)arg1) -> matrix_vector :
docstring

so_dipole(...)
so_dipole((MintsHelper)arg1) -> matrix_vector :
docstring

so_kinetic(...)
so_kinetic((MintsHelper)arg1) -> Matrix :
docstring

so_nabla(...)
so_nabla((MintsHelper)arg1) -> matrix_vector :
docstring

so_overlap(...)
so_overlap((MintsHelper)arg1) -> Matrix :
docstring

so_potential(...)
so_potential((MintsHelper)arg1) -> Matrix :
docstring

so_quadrupole(...)
so_quadrupole((MintsHelper)arg1) -> matrix_vector :
docstring

so_traceless_quadrupole(...)
so_traceless_quadrupole((MintsHelper)arg1) -> matrix_vector :

docstring

sobasisset(...)
sobasisset((MintsHelper)arg1) -> SOBasisSet :
docstring

———————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

———————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

class MoldenWriter(Boost.Python.instance)

docstring

Method resolution order:
MoldenWriter
Boost.Python.instance
__builtin__.object

Methods defined here:

__init__(...)
__init__((object)arg1, (Wavefunction)arg2) -> None

__reduce__ = <unnamed Boost.Python function>(...)

write(...)
write((MoldenWriter)arg1, (str)arg2) -> None :
docstring

———————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

class Molecule(Boost.Python.instance)

docstring

Method resolution order:
Molecule
Boost.Python.instance
__builtin__.object

Methods defined here:

Z(...)
Z((Molecule)arg1, (int)arg2) -> float :
docstring

__init__(...)
__init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

activate_all_fragments(...)
activate_all_fragments((Molecule)arg1) -> None :
docstring

add_atom(...)
add_atom((Molecule)arg1, (int)arg2, (float)arg3, (float)arg4, (float)arg5, (str)arg6, (float)arg7, (float)arg8, (int)arg9) -> None :
docstring

atom_at_position(...)
atom_at_position((Molecule)arg1, (float)arg2, (float)arg3) -> int :
docstring

center_of_mass(...)
center_of_mass((Molecule)arg1) -> Vector3 :
docstring

charge(...)
charge((Molecule)arg1, (int)arg2) -> float :

docstring

deactivate_all_fragments(...)
: deactivate_all_fragments((Molecule)arg1) -> None :
: docstring

extract_subsets(...)
: extract_subsets((Molecule)arg1, (list)arg2, (list)arg3) -> Molecule :
: docstring

extract_subsets((Molecule)arg1, (list)arg2, (int)arg3) -> Molecule :
: docstring

extract_subsets((Molecule)arg1, (int)arg2, (list)arg3) -> Molecule :
: docstring

extract_subsets((Molecule)arg1, (int)arg2, (int)arg3) -> Molecule :
: docstring

extract_subsets((Molecule)arg1, (list)arg2) -> Molecule :
: docstring

extract_subsets((Molecule)arg1, (int)arg2) -> Molecule :
: docstring

find_point_group(...)
: find_point_group((Molecule)arg1, (float)arg2) -> PointGroup :
: docstring

fix_orientation(...)
: fix_orientation((Molecule)arg1, (bool)arg2) -> None :
: docstring

form_symmetry_information(...)
: form_symmetry_information((Molecule)arg1, (float)arg2) -> None :
: docstring

get_variable(...)
: get_variable((Molecule)arg1, (str)arg2) -> float :
: docstring

init_with_checkpoint(...)
: init_with_checkpoint((Molecule)arg1, (Checkpoint)arg2) -> None :
: docstring

init_with_io(...)
: init_with_io((Molecule)arg1, (IO)arg2) -> None :

docstring

is_variable(...)
is_variable((Molecule)arg1, (str)arg2) -> bool :
docstring

label(...)
label((Molecule)arg1, (int)arg2) -> str :
docstring

mass(...)
mass((Molecule)arg1, (int)arg2) -> float :
docstring

molecular_charge(...)
molecular_charge((Molecule)arg1) -> int :
docstring

move_to_com(...)
move_to_com((Molecule)arg1) -> None :
docstring

multiplicity(...)
multiplicity((Molecule)arg1) -> int :
docstring

name(...)
name((Molecule)arg1) -> str :
docstring

natom(...)
natom((Molecule)arg1) -> int :
docstring

nfragments(...)
nfragments((Molecule)arg1) -> int :
docstring

nuclear_repulsion_energy(...)
nuclear_repulsion_energy((Molecule)arg1) -> float :
docstring

point_group(...)
point_group((Molecule)arg1) -> PointGroup :
docstring

print_in_input_format(...)

print_in_input_format((Molecule)arg1) -> None :
docstring

print_out(...)
print_out((Molecule)arg1) -> None :
docstring

print_out((Molecule)arg1) -> None :
docstring

print_out_in_bohr(...)
print_out_in_bohr((Molecule)arg1) -> None :
docstring

reinterpret_coordentry(...)
reinterpret_coordentry((Molecule)arg1, (bool)arg2) -> None :
docstring

reset_point_group(...)
reset_point_group((Molecule)arg1, (str)arg2) -> None :
docstring

save_string_xyz(...)
save_string_xyz((Molecule)arg1) -> str :
docstring

save_to_checkpoint(...)
save_to_checkpoint((Molecule)arg1, (Checkpoint)arg2, (str)arg3) -> None :
docstring

save_xyz(...)
save_xyz((Molecule)arg1, (str)arg2) -> None :
docstring

schoenflies_symbol(...)
schoenflies_symbol((Molecule)arg1) -> str :
docstring

set_active_fragment(...)
set_active_fragment((Molecule)arg1, (int)arg2) -> None :
docstring

set_active_fragments(...)
set_active_fragments((Molecule)arg1, (list)arg2) -> None :
docstring

set_basis_all_atoms(...)

set_basis_all_atoms((Molecule)arg1, (str)arg2, (str)arg3) -> None :
docstring

set_basis_by_label(...)
set_basis_by_label((Molecule)arg1, (str)arg2, (str)arg3, (str)arg4) -> None :
docstring

set_basis_by_number(...)
set_basis_by_number((Molecule)arg1, (int)arg2, (str)arg3, (str)arg4) -> None :
docstring

set_basis_by_symbol(...)
set_basis_by_symbol((Molecule)arg1, (str)arg2, (str)arg3, (str)arg4) -> None :
docstring

set_geometry(...)
set_geometry((Molecule)arg1, (Matrix)arg2) -> None :
docstring

set_ghost_fragment(...)
set_ghost_fragment((Molecule)arg1, (int)arg2) -> None :
docstring

set_ghost_fragments(...)
set_ghost_fragments((Molecule)arg1, (list)arg2) -> None :
docstring

set_molecular_charge(...)
set_molecular_charge((Molecule)arg1, (int)arg2) -> None :
docstring

set_multiplicity(...)
set_multiplicity((Molecule)arg1, (int)arg2) -> None :
docstring

set_name(...)
set_name((Molecule)arg1, (str)arg2) -> None :
docstring

set_point_group(...)
set_point_group((Molecule)arg1, (PointGroup)arg2) -> None :
docstring

set_variable(...)
set_variable((Molecule)arg1, (str)arg2, (float)arg3) -> None :
docstring

symbol(...)

    symbol((Molecule)arg1, (int)arg2) -> str :

        docstring

translate(...)

    translate((Molecule)arg1, (Vector3)arg2) -> None :

        docstring

update_geometry(...)

    update_geometry((Molecule)arg1) -> None :

        docstring

    update_geometry((Molecule)arg1) -> None :

        docstring

x(...)

    x((Molecule)arg1, (int)arg2) -> float :

        docstring

y(...)

    y((Molecule)arg1, (int)arg2) -> float :

        docstring

z(...)

    z((Molecule)arg1, (int)arg2) -> float :

        docstring

Static methods defined here:

create_molecule_from_string(...)

    create_molecule_from_string((str)arg1) -> Molecule :

        docstring

Data and other attributes defined here:

__instance_size__ = 32

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class MultipoleSymmetry(Boost.Python.instance)

docstring

Method resolution order:
    MultipoleSymmetry
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__((object)arg1, (int)arg2, (Molecule)arg3, (object)arg4, (MatrixFactory)arg5) -> None

__reduce__ = <unnamed Boost.Python function>(...)

create_matrices(...)
    create_matrices((MultipoleSymmetry)arg1, (str)arg2) -> matrix_vector :
        docstring

——————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

——————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class NBOWriter(Boost.Python.instance)

docstring

Method resolution order:
    NBOWriter
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)

        __init__((object)arg1, (Wavefunction)arg2) -> None

    __reduce__ = <unnamed Boost.Python function>(...)

    write(...)
        write((NBOWriter)arg1, (str)arg2) -> None :
            docstring

    ———————————————————————————————-

    Data descriptors inherited from Boost.Python.instance:

    __dict__

    __weakref__

    ———————————————————————————————-

    Data and other attributes inherited from Boost.Python.instance:

    __new__ = <built-in method __new__ of Boost.Python.class object>
        T.__new__(S, ...) -> a new object with type S, a subtype of T

class OEProp(Boost.Python.instance)

    docstring

    Method resolution order:
        OEProp
        Boost.Python.instance
        __builtin__.object

    Methods defined here:

    __init__(...)
        __init__((object)arg1) -> None

    __reduce__ = <unnamed Boost.Python function>(...)

    add(...)
        add((OEProp)arg1, (str)arg2) -> None :
            docstring

    compute(...)
        compute((OEProp)arg1) -> None :
            docstring

    set_title(...)
        set_title((OEProp)arg1, (str)arg2) -> None :
            docstring

————————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class PetiteList(Boost.Python.instance)

docstring

Method resolution order:
    PetiteList
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__reduce__ = <unnamed Boost.Python function>(...)

aotoso(...)
    aotoso((PetiteList)arg1) -> Matrix :
        docstring

print(...)
    print((PetiteList)arg1, (object)arg2) -> None :
        docstring

sotoao(...)
    sotoao((PetiteList)arg1) -> Matrix :
        docstring

————————————————————————-

Data and other attributes defined here:

__init__ = <built-in function __init__>

```
          Raises an exception
          This class cannot be instantiated from Python

     ----------------------------------------------------------------------
     Data descriptors inherited from Boost.Python.instance:

     __dict__

     __weakref__

     ----------------------------------------------------------------------
     Data and other attributes inherited from Boost.Python.instance:

     __new__ = <built-in method __new__ of Boost.Python.class object>
          T.__new__(S, ...) -> a new object with type S, a subtype of T
```

class PointGroup(Boost.Python.instance)

```
     docstring

     Method resolution order:
          PointGroup
          Boost.Python.instance
          __builtin__.object

     Methods defined here:

     __init__(...)
          __init__( (object)arg1) -> None

          __init__( (object)arg1, (str)arg2) -> None

     __reduce__ = <unnamed Boost.Python function>(...)

     symbol(...)
          symbol( (PointGroup)arg1) -> str :
               docstring

     ----------------------------------------------------------------------
     Data and other attributes defined here:

     __instance_size__ = 32

     ----------------------------------------------------------------------
     Data descriptors inherited from Boost.Python.instance:

     __dict__
```

```
__weakref__

----------------------------------------------------------------------
Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T
```

class Process(Boost.Python.instance)

```
Method resolution order:
    Process
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__( (object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

----------------------------------------------------------------------
Data descriptors defined here:

environment

----------------------------------------------------------------------
Data and other attributes defined here:

__instance_size__ = 24

----------------------------------------------------------------------
Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

----------------------------------------------------------------------
Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T
```

class PseudoTrial(Boost.Python.instance)

```
docstring

Method resolution order:
        PseudoTrial
        Boost.Python.instance
        __builtin__.object

Methods defined here:

__init__(...)
        __init__( (object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

getA(...)
        getA( (PseudoTrial)arg1) -> Matrix :
                docstring

getI(...)
        getI( (PseudoTrial)arg1) -> Matrix :
                docstring

getIPS(...)
        getIPS( (PseudoTrial)arg1) -> Matrix :
                docstring

getQ(...)
        getQ( (PseudoTrial)arg1) -> Matrix :
                docstring

getR(...)
        getR( (PseudoTrial)arg1) -> Matrix :
                docstring

----------------------------------------------------------------------
Data and other attributes defined here:

__instance_size__ = 32

----------------------------------------------------------------------
Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

----------------------------------------------------------------------
Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
```

T.__new__(S, ...) -> a new object with type S, a subtype of T

class PsiReturnType(Boost.Python.enum)

docstring

Method resolution order:
    PsiReturnType
    Boost.Python.enum
    __builtin__.int
    __builtin__.object

Data and other attributes defined here:

Balk = PsiMod.PsiReturnType.Balk

EndLoop = PsiMod.PsiReturnType.EndLoop

Failure = PsiMod.PsiReturnType.Failure

Success = PsiMod.PsiReturnType.Success

names = {'Balk': PsiMod.PsiReturnType.Balk, 'EndLoop': PsiMod.PsiRetur...

values = {0: PsiMod.PsiReturnType.Success, 1: PsiMod.PsiReturnType.Fai...

Methods inherited from Boost.Python.enum:

__repr__(...)
    x.__repr__() <==> repr(x)

__str__(...)
    x.__str__() <==> str(x)

Data descriptors inherited from Boost.Python.enum:

name

Methods inherited from __builtin__.int:

__abs__(...)
    x.__abs__() <==> abs(x)

__add__(...)
    x.__add__(y) <==> x+y

__and__(...)
    x.__and__(y) <==> x&y

__cmp__(...)
    x.__cmp__(y) <==> cmp(x,y)

__coerce__(...)
    x.__coerce__(y) <==> coerce(x, y)

__div__(...)
    x.__div__(y) <==> x/y

__divmod__(...)
    x.__divmod__(y) <==> divmod(x, y)

__float__(...)
    x.__float__() <==> float(x)

__floordiv__(...)
    x.__floordiv__(y) <==> x//y

__format__(...)

__getattribute__(...)
    x.__getattribute__('name') <==> x.name

__getnewargs__(...)

__hash__(...)
    x.__hash__() <==> hash(x)

__hex__(...)
    x.__hex__() <==> hex(x)

__index__(...)
    x[y:z] <==> x[y.__index__():z.__index__()]

__int__(...)
    x.__int__() <==> int(x)

__invert__(...)
    x.__invert__() <==> ~x

__long__(...)
    x.__long__() <==> long(x)

__lshift__(...)
    x.__lshift__(y) <==> x<<y

__mod__(...)
    x.__mod__(y) <==> x%y

__mul__(...)
    x.__mul__(y) <==> x*y

__neg__(...)
    x.__neg__() <==> -x

__nonzero__(...)
    x.__nonzero__() <==> x != 0

__oct__(...)
    x.__oct__() <==> oct(x)

__or__(...)
    x.__or__(y) <==> x|y

__pos__(...)
    x.__pos__() <==> +x

__pow__(...)
    x.__pow__(y[, z]) <==> pow(x, y[, z])

__radd__(...)
    x.__radd__(y) <==> y+x

__rand__(...)
    x.__rand__(y) <==> y&x

__rdiv__(...)
    x.__rdiv__(y) <==> y/x

__rdivmod__(...)
    x.__rdivmod__(y) <==> divmod(y, x)

__rfloordiv__(...)
    x.__rfloordiv__(y) <==> y//x

__rlshift__(...)
    x.__rlshift__(y) <==> y<<x

__rmod__(...)
    x.__rmod__(y) <==> y%x

__rmul__(...)
    x.__rmul__(y) <==> y*x

__ror__(...)
    x.__ror__(y) <==> y|x

__rpow__(...)
    y.__rpow__(x[, z]) <==> pow(x, y[, z])

__rrshift__(...)
    x.__rrshift__(y) <==> y>>x

__rshift__(...)
    x.__rshift__(y) <==> x>>y

__rsub__(...)
    x.__rsub__(y) <==> y-x

__rtruediv__(...)
    x.__rtruediv__(y) <==> y/x

__rxor__(...)
    x.__rxor__(y) <==> y^x

__sub__(...)
    x.__sub__(y) <==> x-y

__truediv__(...)
    x.__truediv__(y) <==> x/y

__trunc__(...)
    Truncating an Integral returns itself.

__xor__(...)
    x.__xor__(y) <==> x^y

bit_length(...)
    int.bit_length() -> int

    Number of bits necessary to represent self in binary.
    >>> bin(37)
    '0b100101'
    >>> (37).bit_length()
    6

conjugate(...)

Returns self, the complex conjugate of any int.

———————————————————————————————-

Data descriptors inherited from __builtin__.int:

denominator
the denominator of a rational number in lowest terms

imag
the imaginary part of a complex number

numerator
the numerator of a rational number in lowest terms

real
the real part of a complex number

———————————————————————————————-

Data and other attributes inherited from __builtin__.int:

__new__ = <built-in method __new__ of type object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

class RHF(HF, Wavefunction)

docstring

Method resolution order:
RHF
HF
Wavefunction
Boost.Python.instance
__builtin__.object

Methods defined here:

__reduce__ = <unnamed Boost.Python function>(...)

———————————————————————————————-

Data and other attributes defined here:

__init__ = <built-in function __init__>
Raises an exception
This class cannot be instantiated from Python

———————————————————————————————-

Methods inherited from Wavefunction:

Ca(...)
: Ca((Wavefunction)arg1) -> Matrix :
  docstring

Cb(...)
: Cb((Wavefunction)arg1) -> Matrix :
  docstring

Da(...)
: Da((Wavefunction)arg1) -> Matrix :
  docstring

Db(...)
: Db((Wavefunction)arg1) -> Matrix :
  docstring

Fa(...)
: Fa((Wavefunction)arg1) -> Matrix :
  docstring

Fb(...)
: Fb((Wavefunction)arg1) -> Matrix :
  docstring

add_postiteration_callback(...)
: add_postiteration_callback((Wavefunction)arg1, (object)arg2) -> None :
  docstring

add_preiteration_callback(...)
: add_preiteration_callback((Wavefunction)arg1, (object)arg2) -> None :
  docstring

basisset(...)
: basisset((Wavefunction)arg1) -> BasisSet :
  docstring

energy(...)
: energy((Wavefunction)arg1) -> float :
  docstring

epsilon_a(...)
: epsilon_a((Wavefunction)arg1) -> Vector :
  docstring

epsilon_b(...)
: epsilon_b((Wavefunction)arg1) -> Vector :
  docstring

frequencies(...)
 frequencies((Wavefunction)arg1) -> Vector :
  docstring

gradient(...)
 gradient((Wavefunction)arg1) -> Matrix :
  docstring

nirrep(...)
 nirrep((Wavefunction)arg1) -> int :
  docstring

nmo(...)
 nmo((Wavefunction)arg1) -> int :
  docstring

nso(...)
 nso((Wavefunction)arg1) -> int :
  docstring

sobasisset(...)
 sobasisset((Wavefunction)arg1) -> SOBasisSet :
  docstring

———————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
 T.__new__(S, ...) -> a new object with type S, a subtype of T

class SOBasisSet(Boost.Python.instance)

docstring

Method resolution order:
 SOBasisSet
 Boost.Python.instance
 __builtin__.object

Methods defined here:

__reduce__ = <unnamed Boost.Python function>(...)

petite_list(...)
    petite_list((SOBasisSet)arg1) -> PetiteList :
        docstring

———————————————————————————-

Data and other attributes defined here:

__init__ = <built-in function __init__>
    Raises an exception
    This class cannot be instantiated from Python

———————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class SuperFunctional(Boost.Python.instance)

docstring

Method resolution order:
    SuperFunctional
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__init__(...)
    __init__((object)arg1) -> None

__reduce__ = <unnamed Boost.Python function>(...)

computeRKSFunctional(...)
    computeRKSFunctional((SuperFunctional)arg1, (object)arg2) -> None :
        docstring

computeUKSFunctional(...)

computeUKSFunctional((SuperFunctional)arg1, (object)arg2) -> None :
docstring

get_citation(...)
get_citation((SuperFunctional)arg1) -> str :
docstring

get_composition(...)
get_composition((SuperFunctional)arg1) -> str :
docstring

get_dash_d(...)
get_dash_d((SuperFunctional)arg1) -> object :
docstring

get_deriv(...)
get_deriv((SuperFunctional)arg1) -> int :
docstring

get_description(...)
get_description((SuperFunctional)arg1) -> str :
docstring

get_exact_exchange(...)
get_exact_exchange((SuperFunctional)arg1) -> float :
docstring

get_functional(...)
get_functional((SuperFunctional)arg1, (int)arg2) -> Functional :
docstring

get_name(...)
get_name((SuperFunctional)arg1) -> str :
docstring

get_npoints(...)
get_npoints((SuperFunctional)arg1) -> int :
docstring

get_omega(...)
get_omega((SuperFunctional)arg1) -> float :
docstring

get_pt2(...)
get_pt2((SuperFunctional)arg1) -> float :
docstring

get_size(...)
    get_size((SuperFunctional)arg1) -> int :
        docstring

get_weight(...)
    get_weight((SuperFunctional)arg1, (int)arg2) -> float :
        docstring

is_dash_d(...)
    is_dash_d((SuperFunctional)arg1) -> bool :
        docstring

is_double_hybrid(...)
    is_double_hybrid((SuperFunctional)arg1) -> bool :
        docstring

is_gga(...)
    is_gga((SuperFunctional)arg1) -> bool :
        docstring

is_hybrid(...)
    is_hybrid((SuperFunctional)arg1) -> bool :
        docstring

is_meta(...)
    is_meta((SuperFunctional)arg1) -> bool :
        docstring

is_range_corrected(...)
    is_range_corrected((SuperFunctional)arg1) -> bool :
        docstring

set_citation(...)
    set_citation((SuperFunctional)arg1, (str)arg2) -> None :
        docstring

set_dash_d(...)
    set_dash_d((SuperFunctional)arg1, (object)arg2, (float)arg3) -> None :
        docstring

set_deriv(...)
    set_deriv((SuperFunctional)arg1, (int)arg2) -> None :
        docstring

set_description(...)
    set_description((SuperFunctional)arg1, (str)arg2) -> None :
        docstring

set_exact_exchange(...)
    set_exact_exchange((SuperFunctional)arg1, (float)arg2) -> None :
        docstring

set_name(...)
    set_name((SuperFunctional)arg1, (str)arg2) -> None :
        docstring

set_npoints(...)
    set_npoints((SuperFunctional)arg1, (int)arg2) -> None :
        docstring

set_omega(...)
    set_omega((SuperFunctional)arg1, (float)arg2) -> None :
        docstring

set_parameter(...)
    set_parameter((SuperFunctional)arg1, (str)arg2, (str)arg3, (float)arg4) -> None :
        docstring

set_pt2(...)
    set_pt2((SuperFunctional)arg1, (float)arg2) -> None :
        docstring

set_size(...)
    set_size((SuperFunctional)arg1) -> int :
        docstring

———————————————————————————-

Static methods defined here:

available_names(...)
    available_names() -> object :
        docstring

available_superfunctionals(...)
    available_superfunctionals() -> str :
        docstring

build_superfunctional(...)
    build_superfunctional((str)arg1, (int)arg2, (int)arg3) -> SuperFunctional :
        docstring

create_superfunctional(...)
    create_superfunctional((str)arg1, (int)arg2, (int)arg3) -> SuperFunctional :
        docstring

————————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

————————————————————————-

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class SymmetryOperation(Boost.Python.instance)

docstring

Method resolution order:
    SymmetryOperation
    Boost.Python.instance
    __builtin__.object

Methods defined here:

E(...)
    E((SymmetryOperation)arg1) -> None :
        docstring

__init__(...)
    __init__((object)arg1) -> None

    __init__((object)arg1, (SymmetryOperation)arg2) -> None

__reduce__ = <unnamed Boost.Python function>(...)

c2_x(...)
    c2_x((SymmetryOperation)arg1) -> None :
        docstring

c2_y(...)
    c2_y((SymmetryOperation)arg1) -> None :
        docstring

i(...)
    i((SymmetryOperation)arg1) -> None :
        docstring

operate(...)
    operate((SymmetryOperation)arg1, (SymmetryOperation)arg2) -> SymmetryOperation :
        docstring

rotate_n(...)
    rotate_n((SymmetryOperation)arg1, (int)arg2) -> None :
        docstring

rotate_theta(...)
    rotate_theta((SymmetryOperation)arg1, (float)arg2) -> None :
        docstring

sigma_xy(...)
    sigma_xy((SymmetryOperation)arg1) -> None :
        docstring

sigma_xz(...)
    sigma_xz((SymmetryOperation)arg1) -> None :
        docstring

sigma_yz(...)
    sigma_yz((SymmetryOperation)arg1) -> None :
        docstring

trace(...)
    trace((SymmetryOperation)arg1) -> float :
        docstring

transform(...)
    transform((SymmetryOperation)arg1, (SymmetryOperation)arg2) -> SymmetryOperation :
        docstring

transpose(...)
    transpose((SymmetryOperation)arg1) -> None :
        docstring

unit(...)
    unit((SymmetryOperation)arg1) -> None :
        docstring

zero(...)
    zero((SymmetryOperation)arg1) -> None :

```
                docstring

        ----------------------------------------------------------------------
        Data and other attributes defined here:

        __instance_size__ = 96

        ----------------------------------------------------------------------
        Data descriptors inherited from Boost.Python.instance:

        __dict__

        __weakref__

        ----------------------------------------------------------------------
        Data and other attributes inherited from Boost.Python.instance:

        __new__ = <built-in method __new__ of Boost.Python.class object>
            T.__new__(S, ...) -> a new object with type S, a subtype of T
```

class Vector(Boost.Python.instance)

```
        docstring

        Method resolution order:
            Vector
            Boost.Python.instance
            __builtin__.object

        Methods defined here:

        __getitem__(...)
            __getitem__( (Vector)arg1, (int)arg2) -> float :
                docstring

            __getitem__( (Vector)arg1, (tuple)arg2) -> float :
                docstring

        __init__(...)
            __init__( (object)arg1) -> None

            __init__( (object)arg1, (int)arg2) -> None

        __reduce__ = <unnamed Boost.Python function>(...)

        __setitem__(...)
            __setitem__( (Vector)arg1, (int)arg2, (float)arg3) -> None :
                docstring
```

    __setitem__((Vector)arg1, (tuple)arg2, (float)arg3) -> None :
        docstring

dim(...)
    dim((Vector)arg1, (int)arg2) -> int :
        docstring

get(...)
    get((Vector)arg1, (int)arg2) -> float :
        docstring

    get((Vector)arg1, (int)arg2, (int)arg3) -> float :
        docstring

nirrep(...)
    nirrep((Vector)arg1) -> int :
        docstring

print_out(...)
    print_out((Vector)arg1) -> None :
        docstring

scale(...)
    scale((Vector)arg1, (float)arg2) -> None :
        docstring

set(...)
    set((Vector)arg1, (int)arg2, (float)arg3) -> None :
        docstring

    set((Vector)arg1, (int)arg2, (int)arg3, (float)arg4) -> None :
        docstring

———————————————————————-

Data and other attributes defined here:

__instance_size__ = 32

———————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

———————————————————————-

Data and other attributes inherited from Boost.Python.instance:

```
__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T
```

class Vector3(Boost.Python.instance)

```
docstring

Method resolution order:
    Vector3
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__add__(...)
    __add__( (Vector3)arg1, (Vector3)arg2) -> object

__getitem__(...)
    __getitem__( (Vector3)arg1, (int)arg2) -> float :
        docstring

__iadd__(...)
    __iadd__( (object)arg1, (Vector3)arg2) -> object

__imul__(...)
    __imul__( (object)arg1, (float)arg2) -> object

__init__(...)
    __init__( (object)arg1) -> None

    __init__( (object)arg1, (float)arg2) -> None

    __init__( (object)arg1, (float)arg2, (float)arg3, (float)arg4) -> None

    __init__( (object)arg1, (Vector3)arg2) -> None

__isub__(...)
    __isub__( (object)arg1, (Vector3)arg2) -> object

__neg__(...)
    __neg__( (Vector3)arg1) -> object

__reduce__ = <unnamed Boost.Python function>(...)

__str__(...)
    __str__( (Vector3)arg1) -> str :
```

docstring

__sub__(...)
__sub__((Vector3)arg1, (Vector3)arg2) -> object

cross(...)
cross((Vector3)arg1, (Vector3)arg2) -> Vector3 :
docstring

distance(...)
distance((Vector3)arg1, (Vector3)arg2) -> float :
docstring

dot(...)
dot((Vector3)arg1, (Vector3)arg2) -> float :
docstring

norm(...)
norm((Vector3)arg1) -> float :
docstring

normalize(...)
normalize((Vector3)arg1) -> None :
docstring

Data and other attributes defined here:

__instance_size__ = 40

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

class Wavefunction(Boost.Python.instance)

docstring

Method resolution order:

```
        Wavefunction
        Boost.Python.instance
        __builtin__.object

Methods defined here:

Ca(...)
        Ca( (Wavefunction)arg1) -> Matrix :
                docstring

Cb(...)
        Cb( (Wavefunction)arg1) -> Matrix :
                docstring

Da(...)
        Da( (Wavefunction)arg1) -> Matrix :
                docstring

Db(...)
        Db( (Wavefunction)arg1) -> Matrix :
                docstring

Fa(...)
        Fa( (Wavefunction)arg1) -> Matrix :
                docstring

Fb(...)
        Fb( (Wavefunction)arg1) -> Matrix :
                docstring

__reduce__ = <unnamed Boost.Python function>(...)

add_postiteration_callback(...)
        add_postiteration_callback( (Wavefunction)arg1, (object)arg2) -> None :
                docstring

add_preiteration_callback(...)
        add_preiteration_callback( (Wavefunction)arg1, (object)arg2) -> None :
                docstring

basisset(...)
        basisset( (Wavefunction)arg1) -> BasisSet :
                docstring

energy(...)
        energy( (Wavefunction)arg1) -> float :
                docstring
```

epsilon_a(...)
    epsilon_a((Wavefunction)arg1) -> Vector :
        docstring

epsilon_b(...)
    epsilon_b((Wavefunction)arg1) -> Vector :
        docstring

frequencies(...)
    frequencies((Wavefunction)arg1) -> Vector :
        docstring

gradient(...)
    gradient((Wavefunction)arg1) -> Matrix :
        docstring

nirrep(...)
    nirrep((Wavefunction)arg1) -> int :
        docstring

nmo(...)
    nmo((Wavefunction)arg1) -> int :
        docstring

nso(...)
    nso((Wavefunction)arg1) -> int :
        docstring

sobasisset(...)
    sobasisset((Wavefunction)arg1) -> SOBasisSet :
        docstring

—————————————————————————-

Data and other attributes defined here:

__init__ = <built-in function __init__>
    Raises an exception
    This class cannot be instantiated from Python

—————————————————————————-

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

class matrix_vector(Boost.Python.instance)

docstring

Method resolution order:
    matrix_vector
    Boost.Python.instance
    __builtin__.object

Methods defined here:

__contains__(...)
    __contains__((matrix_vector)arg1, (object)arg2) -> bool

__delitem__(...)
    __delitem__((matrix_vector)arg1, (object)arg2) -> None

__getitem__(...)
    __getitem__((object)arg1, (object)arg2) -> object

__init__(...)
    __init__((object)arg1) -> None

__iter__(...)
    __iter__((object)arg1) -> object

__len__(...)
    __len__((matrix_vector)arg1) -> int

__reduce__ = <unnamed Boost.Python function>(...)

__setitem__(...)
    __setitem__((matrix_vector)arg1, (object)arg2, (object)arg3) -> None

append(...)
    append((matrix_vector)arg1, (object)arg2) -> None

extend(...)
    extend((matrix_vector)arg1, (object)arg2) -> None

Data and other attributes defined here:

__instance_size__ = 40

Data descriptors inherited from Boost.Python.instance:

__dict__

__weakref__

Data and other attributes inherited from Boost.Python.instance:

__new__ = <built-in method __new__ of Boost.Python.class object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

FUNCTIONS

DASUM(...)

DASUM((int)arg1, (int)arg2, (Vector)arg3, (int)arg4) -> float : docstring

DAXPY(...)

DAXPY((int)arg1, (int)arg2, (float)arg3, (Vector)arg4, (int)arg5, (Vector)arg6, (int)arg7) -> None : docstring

DCOPY(...)

DCOPY((int)arg1, (int)arg2, (Vector)arg3, (int)arg4, (Vector)arg5, (int)arg6) -> None : docstring

DDOT(...)

DDOT((int)arg1, (int)arg2, (Vector)arg3, (int)arg4, (Vector)arg5, (int)arg6) -> float : docstring

DGBMV(...)

DGBMV((int)arg1, (str)arg2, (int)arg3, (int)arg4, (int)arg5, (int)arg6, (float)arg7, (Matrix)arg8, (int)arg9, (Vector)ar
docstring

DGEEV(...)

DGEEV((int)arg1, (str)arg2, (str)arg3, (int)arg4, (Matrix)arg5, (int)arg6, (Vector)arg7, (Vector)arg8, (Matrix)arg9,
docstring

DGEMM(...)

DGEMM((int)arg1, (str)arg2, (str)arg3, (int)arg4, (int)arg5, (int)arg6, (float)arg7, (Matrix)arg8, (int)arg9, (Matrix)a
docstring

DGEMV(...)

DGEMV((int)arg1, (str)arg2, (int)arg3, (int)arg4, (float)arg5, (Matrix)arg6, (int)arg7, (Vector)arg8, (int)arg9, (float)
docstring

DGER(...)

DGER((int)arg1, (int)arg2, (int)arg3, (float)arg4, (Vector)arg5, (int)arg6, (Vector)arg7, (int)arg8, (Matrix)arg9, (int)
docstring

DGETRF(...)

DGETRF((int)arg1, (int)arg2, (int)arg3, (Matrix)arg4, (int)arg5, (IntVector)arg6) -> int :
docstring

DGETRI(...)

DGETRI((int)arg1, (int)arg2, (Matrix)arg3, (int)arg4, (IntVector)arg5, (Vector)arg6, (int)arg7) -> int :
docstring

DGETRS(...)

DGETRS((int)arg1, (str)arg2, (int)arg3, (int)arg4, (Matrix)arg5, (int)arg6, (IntVector)arg7, (Matrix)arg8, (int)arg9)
docstring

DNRM2(...)

DNRM2((int)arg1, (int)arg2, (Vector)arg3, (int)arg4) -> float : docstring

DPOTRF(...)

DPOTRF((int)arg1, (str)arg2, (int)arg3, (Matrix)arg4, (int)arg5) -> int : docstring

DPOTRI(...)

DPOTRI((int)arg1, (str)arg2, (int)arg3, (Matrix)arg4, (int)arg5) -> int : docstring

DPOTRS(...)

DPOTRS((int)arg1, (str)arg2, (int)arg3, (int)arg4, (Matrix)arg5, (int)arg6, (Matrix)arg7, (int)arg8) -> int :
docstring

DROT(...)

DROT((int)arg1, (int)arg2, (Vector)arg3, (int)arg4, (Vector)arg5, (int)arg6, (float)arg7, (float)arg8) -> None :
docstring

DSBMV(...)

DSBMV((int)arg1, (str)arg2, (int)arg3, (int)arg4, (float)arg5, (Matrix)arg6, (int)arg7, (Vector)arg8, (int)arg9, (float)
docstring

DSCAL(...)

DSCAL((int)arg1, (int)arg2, (float)arg3, (Vector)arg4, (int)arg5) -> None : docstring

DSWAP(...)

DSWAP((int)arg1, (int)arg2, (Vector)arg3, (int)arg4, (Vector)arg5, (int)arg6) -> None : docstring

DSYEV(...)

DSYEV((int)arg1, (str)arg2, (str)arg3, (int)arg4, (Matrix)arg5, (int)arg6, (Vector)arg7, (Vector)arg8, (int)arg9) -> in
docstring

DSYMM(...)

DSYMM((int)arg1, (str)arg2, (str)arg3, (int)arg4, (int)arg5, (float)arg6, (Matrix)arg7, (int)arg8, (Matrix)arg9, (int)a
docstring

DSYMV(...)

DSYMV((int)arg1, (str)arg2, (int)arg3, (float)arg4, (Matrix)arg5, (int)arg6, (Vector)arg7, (int)arg8, (float)arg9, (Vect
docstring

DSYR(...)

DSYR((int)arg1, (str)arg2, (int)arg3, (float)arg4, (Vector)arg5, (int)arg6, (Matrix)arg7, (int)arg8) -> None :
docstring

DSYR2(...)

DSYR2((int)arg1, (str)arg2, (int)arg3, (float)arg4, (Vector)arg5, (int)arg6, (Vector)arg7, (int)arg8, (Matrix)arg9, (int
docstring

DSYR2K(...)

DSYR2K((int)arg1, (str)arg2, (str)arg3, (int)arg4, (int)arg5, (float)arg6, (Matrix)arg7, (int)arg8, (Matrix)arg9, (int)a
docstring

DSYRK(...)

DSYRK((int)arg1, (str)arg2, (str)arg3, (int)arg4, (int)arg5, (float)arg6, (Matrix)arg7, (int)arg8, (float)arg9, (Matrix)
docstring

DSYSV(...)

DSYSV((int)arg1, (str)arg2, (int)arg3, (int)arg4, (Matrix)arg5, (int)arg6, (IntVector)arg7, (Matrix)arg8, (int)arg9, (
docstring

DTBMV(...)

DTBMV((int)arg1, (str)arg2, (str)arg3, (str)arg4, (int)arg5, (int)arg6, (Matrix)arg7, (int)arg8, (Vector)arg9, (int)arg
docstring

DTBSV(...)

DTBSV((int)arg1, (str)arg2, (str)arg3, (str)arg4, (int)arg5, (int)arg6, (Matrix)arg7, (int)arg8, (Vector)arg9, (int)arg1
docstring

DTRMM(...)

DTRMM((int)arg1, (str)arg2, (str)arg3, (str)arg4, (str)arg5, (int)arg6, (int)arg7, (float)arg8, (Matrix)arg9, (int)arg1
docstring

DTRMV(...)

DTRMV((int)arg1, (str)arg2, (str)arg3, (str)arg4, (int)arg5, (Matrix)arg6, (int)arg7, (Vector)arg8, (int)arg9) -> None
docstring

DTRSM(...)

DTRSM((int)arg1, (str)arg2, (str)arg3, (str)arg4, (str)arg5, (int)arg6, (int)arg7, (float)arg8, (Matrix)arg9, (int)arg10,
docstring

DTRSV(...)

DTRSV((int)arg1, (str)arg2, (str)arg3, (str)arg4, (int)arg5, (Matrix)arg6, (int)arg7, (Vector)arg8, (int)arg9) -> None
docstring

IDAMAX(...)

IDAMAX((int)arg1, (int)arg2, (Vector)arg3, (int)arg4) -> int : docstring

adc(...)

adc() -> float : docstring

add_user_basis_file(...)

add_user_basis_file((str)arg1) -> None : docstring

benchmark_blas1(...)

benchmark_blas1((int)arg1, (float)arg2) -> None : docstring

benchmark_blas2(...)

benchmark_blas2((int)arg1, (float)arg2) -> None : docstring

benchmark_blas3(...)

benchmark_blas3((int)arg1, (float)arg2, (int)arg3) -> None : docstring

benchmark_disk(...)

benchmark_disk((int)arg1, (float)arg2) -> None : docstring

benchmark_integrals(...)

benchmark_integrals((int)arg1, (float)arg2) -> None : docstring

benchmark_math(...)

benchmark_math((float)arg1) -> None : docstring

ccdensity(...)

ccdensity() -> float : docstring

ccenergy(...)

ccenergy() -> float : docstring

cceom(...)

cceom() -> float : docstring

cchbar(...)

cchbar() -> float : docstring

cclambda(...)

cclambda() -> float : docstring

ccresponse(...)

ccresponse() -> float : docstring

ccsort(...)

ccsort() -> float : docstring

cctriples(...)

cctriples() -> float : docstring

clean(...)

clean() -> None : Function to remove scratch files. Call between independent jobs.

close_outfile(...)

close_outfile() -> None : docstring

dcft(...)

dcft() -> float : docstring

deriv(...)

deriv() -> int : docstring

detci(...)

detci() -> float : docstring

dfcc(...)

dfcc() -> float : docstring

dfmp2(...)

dfmp2() -> float : docstring

fd_1_0(...)

fd_1_0((list)arg1) -> PsiReturnType : docstring

fd_freq_0(...)

fd_freq_0((list)arg1, (int)arg2) -> PsiReturnType : docstring

fd_freq_1(...)

fd_freq_1((list)arg1, (int)arg2) -> PsiReturnType : docstring

fd_geoms_1_0(...)

fd_geoms_1_0() -> matrix_vector : docstring

fd_geoms_freq_0(...)

fd_geoms_freq_0((int)arg1) -> matrix_vector : docstring

fd_geoms_freq_1(...)

fd_geoms_freq_1((int)arg1) -> matrix_vector : docstring

fd_geoms_hessian_0(...)

fd_geoms_hessian_0() -> matrix_vector : docstring

fd_hessian_0(...)

fd_hessian_0((list)arg1) -> PsiReturnType : docstring

flush_outfile(...)

flush_outfile() -> None : docstring

get_active_molecule(...)

get_active_molecule() -> Molecule : docstring

get_global_option(...)

get_global_option((str)arg1) -> object : docstring

get_global_option_list(...)

get_global_option_list() -> list : docstring

get_gradient(...)

get_gradient() -> Matrix : docstring

get_input_directory(...)

get_input_directory() -> str : docstring

get_local_option(...)

get_local_option((str)arg1, (str)arg2) -> object : docstring

get_memory(...)

get_memory() -> int : docstring

get_option(...)

get_option((str)arg1) -> object : docstring

get_variable(...)

get_variable((str)arg1) -> float : docstring

has_global_option_changed(...)

has_global_option_changed((str)arg1) -> bool : docstring

has_local_option_changed(...)

has_local_option_changed((str)arg1, (str)arg2) -> bool : docstring

has_option_changed(...)

has_option_changed((str)arg1) -> bool : docstring

libfock(...)

libfock() -> int : docstring

lmp2(...)

lmp2() -> float : docstring

mcscf(...)

mcscf() -> float : docstring

me(...)

me() -> int : docstring

mints(...)

mints() -> int : docstring

mp2(...)

mp2() -> float : docstring

mrcc_generate_input(...)

mrcc_generate_input((dict)arg1) -> PsiReturnType : docstring

mrcc_load_densities(...)

mrcc_load_densities((dict)arg1) -> PsiReturnType : docstring

nproc(...)

nproc() -> int : docstring

nthread(...)

nthread() -> int : docstring

nuclear_dipole(...)

nuclear_dipole((Molecule)arg1) -> Vector : docstring

omp2(...)

omp2() -> int : docstring

opt_clean(...)

opt_clean() -> None : docstring

optking(...)

optking() -> int : docstring

outfile_name(...)

outfile_name() -> str : docstring

plugin(...)

plugin((str)arg1) -> int : docstring

plugin_close(...)

plugin_close((str)arg1) -> None : docstring

plugin_close_all(...)

plugin_close_all() -> None : docstring

plugin_load(...)

plugin_load((str)arg1) -> int : docstring

prepare_options_for_module(...)

prepare_options_for_module((str)arg1) -> None : docstring

print_global_options(...)

print_global_options() -> None : docstring

print_options(...)

print_options() -> None : docstring

print_out(...)

print_out((str)arg1) -> None : docstring

print_variables(...)

print_variables() -> None : docstring

psi_top_srcdir(...)

psi_top_srcdir() -> str : docstring

psimrcc(...)

psimrcc() -> float : docstring

reference_wavefunction(...)

reference_wavefunction() -> Wavefunction : docstring

reopen_outfile(...)

reopen_outfile() -> None : docstring

revoke_global_option_changed(...)

revoke_global_option_changed((str)arg1) -> None : docstring

revoke_local_option_changed(...)

revoke_local_option_changed((str)arg1, (str)arg2) -> None : docstring

revoke_option_changed(...)

revoke_option_changed((str)arg1) -> None : docstring

sapt(...)

> **sapt() -> float :** docstring

scf(...)

> **scf((object)arg1, (object)arg2) -> float :** docstring
>
> **scf() -> float :** docstring

set_active_molecule(...)

> **set_active_molecule((Molecule)arg1) -> None :** docstring

set_global_option(...)

> **set_global_option((str)arg1, (str)arg2) -> bool :** docstring
>
> **set_global_option((str)arg1, (float)arg2) -> bool :** docstring
>
> **set_global_option((str)arg1, (int)arg2) -> bool :** docstring
>
> set_global_option((str)arg1, (list)arg2 [, (object)arg3]) -> bool

set_global_option_python(...)

> **set_global_option_python((str)arg1, (object)arg2) -> bool :** docstring

set_gradient(...)

> **set_gradient((Matrix)arg1) -> None :** docstring

set_local_option(...)

> **set_local_option((str)arg1, (str)arg2, (str)arg3) -> bool :** docstring
>
> **set_local_option((str)arg1, (str)arg2, (float)arg3) -> bool :** docstring
>
> **set_local_option((str)arg1, (str)arg2, (int)arg3) -> bool :** docstring
>
> set_local_option((str)arg1, (str)arg2, (list)arg3 [, (object)arg4]) -> bool

set_local_option_python(...)

> **set_local_option_python((str)arg1, (object)arg2) -> None :** docstring

set_memory(...)

> **set_memory((int)arg1) -> None :** docstring

set_nthread(...)

> **set_nthread((int)arg1) -> None :** docstring

set_variable(...)

> **set_variable((str)arg1, (float)arg2) -> None :** docstring

transqt(...)

> **transqt() -> float :** docstring

transqt2(...)

> **transqt2() -> float :** docstring

version(...)

> **version() -> str :** docstring

DATA Ascending = PsiMod.DiagonalizeOrder.Ascending Balk = PsiMod.PsiReturnType.Balk Descending = PsiMod.DiagonalizeOrder.Descending EndLoop = PsiMod.PsiReturnType.EndLoop Failure = PsiMod.PsiReturnType.Failure Success = PsiMod.PsiReturnType.Success

Warning: Python naming practices of file_that_includes_function.function_name() are followed below. In psi4 input files, it is only necessary to call the function name alone. That is, use `energy('scf')`, not `driver.energy('scf')`.

Note: The options documented below are placed as arguments in the command that calls the Python function, not in the `set globals` block or with any other `set` command.

Note: Psithon keyword names and values are insensitive to case. The few exceptions are documented for the `database()` function, where case structure must match the database file.

Note: Boolean arguments can be specified by `yes`, `on`, `true`, or `1` for affirmative and `no`, `off`, `false`, or `0` for negative, all insensitive to case.

Note: Certain convergence and tolerance keywords, of type *double* (real numbers), may be specified using either a real numberor an integer; an integer *X* is then treated as the number of converged decimal digits required. For example, to request an energy converged to $10^{-6} E_h$, the user may set the `e_convergence` keyword to 0.000001, 1.0e-6, or 6.

Note: The derivative level type for `driver.optimize()` and `driver.frequency()` functions can be specified by `energy`, `none`, or `0` for 0th derivative, `gradient`, `first`, or `1` for 1st derivative, and `hessian`, `second`, or `2` for 2nd derivative.

Note: The derivative level type for `driver.optimize()` and `driver.frequency()` functions can be specified by `energy`, `none`, or `0` for 0th derivative, `gradient`, `first`, or `1` for 1st derivative, and `hessian`, `second`, or `2` for 2nd derivative.

`driver.`**`optimize`**`(`*name*, ***kwargs*`)`

Function to perform a geometry optimization.

Aliases opt()

Returns (*float*) Total electronic energy of optimized structure in Hartrees.

Psi variables

`CURRENT ENERGY`

Note: Analytic gradients area available for all methods in the table below. Optimizations with other methods in the energy table proceed by finite differences.

Caution: Some features are not yet implemented. Buy a developer a coffee.

- Need to check that all methods do return electronic energy. I think gradient got changed at one point.

name	calls method
scf	Hartree–Fock (HF) or density functional theory (DFT)
mp2	2nd-order Moller-Plesset perturbation theory (MP2)
ccsd	coupled cluster singles and doubles (CCSD)
ccsd(t)	CCSD with perturbative triples
eom-ccsd	equation of motion (EOM) CCSD

Parameters

- **name** (*string*) – `'scf'` || `'df-mp2'` || `'ci5'` || etc.

  First argument, usually unlabeled. Indicates the computational method to be applied to the database. May be any valid argument to `driver.energy()`.

- **func** (*function*) – **|dl|** `gradient` **|dr|** || `energy` || `cbs`

  Indicates the type of calculation to be performed on the molecule. The default dertype accesses‘‘’gradient’‘‘ or `'energy'`, while `'cbs'` performs a multistage finite difference calculation. If a nested series of python functions is intended (see **‘Function Intercalls‘_**), use keyword `opt_func` instead of `func`.

- **mode** (*string*) – **|dl|** `'continuous'` **|dr|** || `'sow'` || `'reap'`

  For a finite difference of energies optimization, indicates whether the calculations required to complete the optimization are to be run in one file (`'continuous'`) or are to be farmed out in an embarrassingly parallel fashion (`'sow'`/`'reap'`). For the latter, run an initial job with `'sow'` and follow instructions in its output file.

- **dertype** (*dertype*) – `'gradient'` || `'energy'`

  Indicates whether analytic (if available) or finite difference optimization is to be performed.

Examples

```
>>> # [1] Analytic scf optimization
>>> optimize('scf')

>>> # [2] Finite difference mp3 optimization
>>> opt('mp3')

>>> # [3] Forced finite difference ccsd optimization
>>> optimize('ccsd', dertype=1)
```

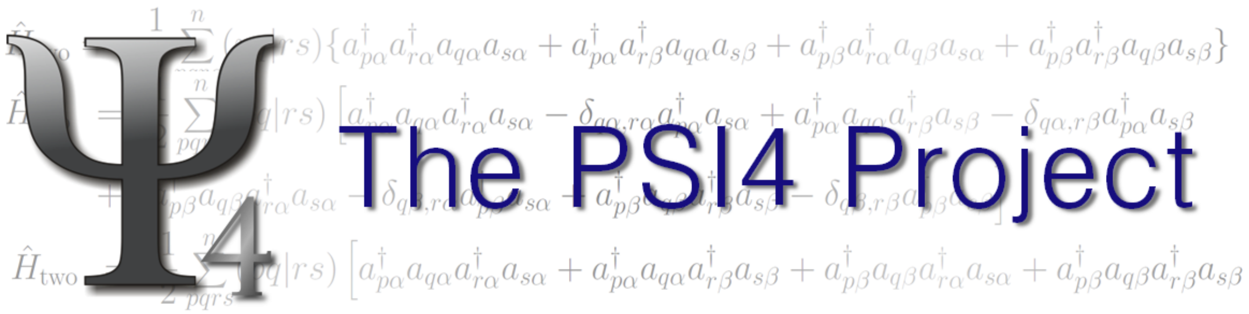
Many of the tasks automated by Python wrappers consist of a number of independent psi4 calculations and are thus suited to an embarrassingly parallel mode of operation. In Psithon, these have been dubbed sow/reap procedures and have the following general structure.

- Prepare an input file, simply adding `mode='sow'` to the argument list of an available Python function. Run this quick job to produce input files for lengthier calculations.
- According to the instructions in the output file of the above step, run the generated input files in any order on any variety of computers and architectures. This is the time-intensive portion of the calculation.
- The ‘sow’ stage also produces a *master* input file (with a `mode='reap'` directive). When all the jobs in the above step are completed, place their output files in the same location as the *master* input, and run this last, quick job to collect the results.
- Sow/reap procedures are governed by the **mode** keyword, choices being `'continuous'`, `'sow'`, and `'reap'`. Only `'sow'` is likely to be used by the user, as `'continuous'` is always the default, and input files with `'reap'` are autogenerated.

- Available at present for **‘Database‘_** and finite difference operation of **‘Optimize‘_**.

Caution: Some features are not yet implemented. Buy a developer a coffee.

- Local options (e.g., `set scf e_convergence 9`) will not get transmitted to the child jobs.
- Array options (e.g., `set states_per_irrep [2, 1]`) will not get transmitted to the child jobs.
- Function intercalls (e.g., db(opt())) are not tested with sow/reap procedures.



CHAPTER

SIX

GENERAL

To allow arbitrarily complex computations to be performed, PSI4 is built upon the Python interpreter, with modifications termed Psithon. Chapter 3 of the User's Manual describes the non-standard Python associated with clean molecule, basis, and option specification in the PSI4 input file. This documentation addresses the pure Python side-what functions allow the efficient compiled code to be run, what functions post-process and interact with that output, and how the ordinary (or ambitious) user can extent PSI4's functionality.

Warning: Python naming practices of file_that_includes_function.function_name() are followed below. In psi4 input files, it is only necessary to call the function name alone. That is, use `energy('scf')`, not `driver.energy('scf')`.

Note: The options documented below are placed as arguments in the command that calls the Python function, not in the `set globals` block or with any other `set` command.

Note: Psithon keyword names and values are insensitive to case. The few exceptions are documented for the `database()` function, where case structure must match the database file.

Note: Boolean arguments can be specified by `yes`, `on`, `true`, or `1` for affirmative and `no`, `off`, `false`, or `0` for negative, all insensitive to case.

Note: Certain convergence and tolerance keywords, of type *double* (real numbers), may be specified using either a real numberor an integer; an integer *X* is then treated as the number of converged decimal digits required. For example, to request an energy converged to $10^{-6}E_h$, the user may set the `e_convergence` keyword to 0.000001, 1.0e-6, or 6.

Note: The derivative level type for `driver.optimize()` and `driver.frequency()` functions can be specified by `energy`, `none`, or `0` for 0th derivative, `gradient`, `first`, or `1` for 1st derivative, and `hessian`, `second`, or `2` for 2nd derivative.

CHAPTER

SEVEN

SYMMETRY-ADAPTED PERTURBATION THEORY

Symmetry-adapted perturbation theory (SAPT) provides a means of directly computing the noncovalent interaction between two molecules, that is, the interaction energy is determined without computing the total energy of the monomers or dimer. In addition, SAPT provides a decomposition of the interaction energy into physically meaningful components: *i.e.,* electrostatic, exchange, induction, and dispersion terms. In SAPT, the Hamiltonian of the dimer is partitioned into contributions from each monomer and the interaction.

$$H = F_A + W_A + F_B + W_B + V$$

Here, the Hamiltonian is written as a sum of the usual monomer Fock operators, F, the fluctuation potential of each monomer, W, and the interaction potential, V. The monomer Fock operators, $F_A + F_B$, are treated as the zeroth-order Hamiltonian and the interaction energy is evaluated through a perturbative expansion of V, W_A, and W_B. Through first-order in V, electrostatic and exchange interactions are included; induction and dispersion first appear at second-order in V. For a complete description of SAPT, the reader is referred to the excellent review by Jeziorski, Moszynski, and Szalewicz [Jeziorski:1994:1887].

Several truncations of the SAPT expansion are available in the SAPT module of PSI4. The simplest truncation of SAPT is denoted SAPT0.

$$E_{SAPT0} = E^{(10)}_{elst} + E^{(10)}_{exch} + E^{(20)}_{ind,resp} + E^{(20)}_{exch-ind,resp} + E^{(20)}_{disp} + E^{(20)}_{exch-disp}$$

In this notation, $E^{(vw)}$ defines the order in V and in $W_A + W_B$; the subscript, $resp$, indicates that orbital relaxation effects are included.

$$E_{SAPT2} = E_{SAPT0} + E^{(12)}_{elst,resp} + E^{(11)}_{exch} + E^{(12)}_{exch} + {}^{t}E^{(22)}_{ind} + {}^{t}E^{(22)}_{exch-ind}$$

$$E_{SAPT2+} = E_{SAPT2} + E^{(21)}_{disp} + E^{(22)}_{disp}$$

$$E_{SAPT2+(3)} = E_{SAPT2+} + E^{(13)}_{elst,resp} + E^{(30)}_{disp}$$

$$E_{SAPT2+3} = E_{SAPT2+(3)} + E^{(30)}_{exch-disp} + E^{(30)}_{ind-disp} + E^{(30)}_{exch-ind-disp}$$

A thorough analysis of the performance of these truncations of SAPT can be found in a review by Hohenstein and Sherrill [Hohenstein:2012:WIREs].

The SAPT module relies entirely on the density-fitting approximation of the two-electron integrals. The factorization of the SAPT energy expressions, as implemented in PSI4, assumes the use of density-fitted two-electron integrals, therefore, the SAPT module cannot be run with exact integrals. In practice, we have found that the density-fitting approximation introduces negligable errors into the SAPT energy and greatly improves efficiency.

7.1 A First Example

The following is the simplest possible input that will perform all available SAPT computations (normally, you would pick one of these methods).

```
molecule water_dimer {
     0 1
     O  -1.551007  -0.114520   0.000000
     H  -1.934259   0.762503   0.000000
     H  -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

     units angstrom
     no_reorient
     symmetry c1
}

set globals {
    basis         aug-cc-pvdz
}

energy('sapt0')
energy('sapt2')
energy('sapt2+')
energy('sapt2+(3)')
energy('sapt2+3')
```

The SAPT module uses the standard PSI4 partitioning of the dimer into monomers. Additionally, the `no_reorient` flag must be included and the use of spatial symmetry disabled by setting the molecule option `symmetry c1`. A final note is that the SAPT module is only capable of performing SAPT comuptations for interactions between closed-shell singlets.

The example input shown above would not be used in practice. To exploit the efficiency of the density-fitted SAPT implementation in PSI4, the SCF computations should also be performed with density-fitted (DF) integrals.

```
set globals {
    basis         aug-cc-pvdz
    df_basis_scf  aug-cc-pvdz-jkfit
    df_basis_sapt aug-cc-pvdz-ri
    guess         sad
    scf_type      df
}

set sapt {
    print         1
}
```

These options will perform the SAPT computation with DF-HF and a superposition-of-atomic-densities guess. This is the preferred method of running the SAPT module.

7.2 SAPT0

Generally speaking, SAPT0 should be applied to large systems or large data sets. The performance of SAPT0 relies entirely on error cancellation, which seems to be optimal with a truncated aug-cc-pVDZ basis, namely, jun-cc-pVDZ (which we have referred to in previous work as aug-cc-pVDZ'). The SAPT module has been used to perform SAPT0 computations with over 200 atoms and 2800 basis functions; this code should be scalable to 4000 basis functions. Publications resulting from the use of the SAPT0 code should cite the following publications: [Hohenstein:2010:184111] and [Hohenstein:2011:174107].

7.2.1 Basic SAPT0 Keywords

SAPT_LEVEL The level of theory for SAPT.

- **Type**: string
- **Possible Values**: SAPT0, SAPT2, SAPT2+, SAPT2+3
- **Default**: SAPT0

BASIS The basis set used to describe the monomer molecular orbitals.

- **Type**: string
- **Possible Values**: Basis Sets
- **Default**: none

DF_BASIS_SAPT The fitting basis to use for all two-electron integrals in the SAPT computation. PSI4 will attempt to pick a reasonable fitting basis if one is not provided.

- **Type**: string
- **Default**: none

DF_BASIS_ELST Optionally, a different fitting basis can be used for the $E^{(10)}_{elst}$ and $E^{(10)}_{exch}$ terms. This may be important if heavier elements are involved.

- **Type**: string
- **Default**: none

FREEZE_CORE Sets the number of core orbitals to freeze in the evaluation of the $E^{(20)}_{disp}$ and $E^{(20)}_{exch-disp}$ terms. It is recommended to freeze core in all SAPT computations.

- **Type**: string
- **Possible Values**: TRUE, FALSE, SMALL, LARGE
- **Default**: FALSE

D_CONVERGENCE Convergence of the residual of the CPHF coefficients needed for the $E^{(20)}_{ind,resp}$.

- **Type**: conv double
- **Default**: 1.0×10^{-8}

E_CONVERGENCE Convergence of the energy change in the $E^{(20)}_{ind,resp}$ term during the solution of the CPHF equations (in hartrees).

- **Type**: conv double
- **Default**: 1.0×10^{-10}

MAXITER The maximum number of CPHF iterations.

- **Type**: integer
- **Default**: 50

PRINT The print level for the SAPT module. If set to 0, only the header and final results are printed. If set to 1, some intermediate quantities are also printed. For large SAPT computations, it is advisable to set to 1 so the progress of the computation can be tracked.

- **Type**: integer
- **Default**: 1

7.2.2 Advanced SAPT0 Keywords

AIO-CPHF Do disk I/O asynchronously during the solution of the CPHF equations. This option may speed up the computation slightly, however its use will cause PSI4 to spawn an additional thread.

- **Type**: boolean
- **Default**: FALSE

AIO_DF_INTS Do disk I/O asynchronously during the formation of the DF integrals. This option may speed up the computation slightly, however its use will cause PSI4 to spawn an additional thread.

- **Type**: boolean
- **Default**: FALSE

NO_RESPONSE Don't solve the CPHF equations, evaluate $E^{(20)}_{ind}$ and $E^{(20)}_{exch-ind}$ instead of their response-including counterparts. Only turn on this option if you are not going to use the induction energy.

- **Type**: boolean
- **Default**: FALSE

INTS_TOLERANCE All three-index DF integrals and those contributing to four-index integrals that fall below this Schwarz bound will be neglected. The default is very conservative, however, there isn't much to gain from loosening it.

- **Type**: conv double
- **Default**: 1.0×10^{-12}

DENOMINATOR_DELTA The SAPT module uses approximate energy denominators for most of the $E^{(20)}_{disp}$ and $E^{(20)}_{exch-disp}$ evaluation. This option controls the maximum allowable error norm in the energy denominator tensor.

- **Type**: double
- **Default**: 1.0×10^{-6}

DENOMINATOR_ALGORITHM Should the energy denominators be approximated with Laplace transformations or a Cholesky decomposition? We have found Laplace transformations to be slightly more efficient.

- **Type**: string
- **Possible Values**: LAPLACE, CHOLESKY
- **Default**: LAPLACE

SAPT_OS_SCALE The SAPT module will print a decomposition of the $E^{(20)}_{disp}$ and $E^{(20)}_{exch-disp}$ terms into same-spin and opposite-spin contributions, in analogy to the SCS-MP2 method of Stefan Grimme. This option controls the scaling of the opposite-spin contributions.

- **Type**: double
- **Default**: 6/5

SAPT_SS_SCALE This option controls the scaling of the same-spin contributions.

- **Type**: double
- **Default**: 1.0/3.0

DEBUG Print lots of intermediate quantities that are not usually interesting. For SAPT, it will also do additional work (which is not optimized for large systems) so don't turn it on.

- **Type**: integer
- **Default**: 0

7.3 Higher-Order SAPT

For smaller systems (up to the size of a nucleic acid base pair), more accurate interaction energies can be obtained through higher-order SAPT computations. The SAPT module can perform density-fitted evaluations of SAPT2, SAPT2+, SAPT2+(3), and SAPT2+3 energies. Publications resulting from the use of the higher-order SAPT code should cite the following: [Hohenstein:2010:014101].

A brief note on memory usage: the higher-order SAPT code assumes that certain quantities can be held in core. This code requires sufficient memory to hold $3o^2v^2 + v^2N_{aux}$ arrays in core. With this requirement computations on the adenine-thymine complex can be performed with an aug-cc-pVTZ basis in less than 64GB of memory.

Higher-order SAPT is treated separately from the higly optimized SAPT0 code, therefore, higher-order SAPT uses a separate set of keywords. The following keywords are relevant for higher-order SAPT.

7.3.1 Basic Keywords for Higher-order SAPT

Skipped for mock-up.

7.3.2 Advanced Keywords for Higher-order SAPT

Skipped for mock-up.

7.4 MP2 Natural Orbitals

One of the unique features of the SAPT module is its ability to use MP2 natural orbitals (NOs) to speed up the evaluation of the triples contribution to disperison. By transforming to the MP2 NO basis, we can throw away virtual orbitals that are expected to contribute little to the dispersion energy. Speedups in excess of $50\times$ are possible. In practice, this approximation is very good and should always be applied. Publications resulting from the use of MP2 NO-based approximations should cite the following: [Hohenstein:2010:104107].

7.4.1 Basic Keywords Controlling MP2 NO Approximations

Skipped for mock-up.

7.4.2 Advanced Keywords Controlling MP2 NO Approximations

Skipped for mock-up.

7.5 Charge-Transfer in SAPT

It is possible to obtain the stabilization energy of a complex due to charge-transfer effects from a SAPT computation. The charge-transfer energy can be computed with the SAPT module as described by Stone and Misquitta [Misquitta:2009:201].

Charge-transfer energies can be obtained from the following calls to the energy function.

```
energy('sapt0-ct')
energy('sapt2-ct')
energy('sapt2+-ct')
energy('sapt2+(3)-ct')
energy('sapt2+3-ct')
```

A SAPT charge-transfer analysis will perform 5 HF computations: the dimer in the dimer basis, monomer A in the dimer basis, monomer B in the dimer basis, monomer A in the monomer A basis, and monomer B in the monomer B basis. Next, it performs two SAPT computations, one in the dimer basis and one in the monomer basis. Finally, it will print a summary of the charge-transfer results:

```
  SAPT Charge Transfer Analysis
-----------------------------------------------------------------------------
  SAPT Induction (Dimer Basis)         -2.0970 mH       -1.3159 kcal mol^-1
  SAPT Induction (Monomer Basis)       -1.1396 mH       -0.7151 kcal mol^-1
  SAPT Charge Transfer                 -0.9574 mH       -0.6008 kcal mol^-1
```

These results are for the water dimer geometry shown above computed with SAPT0/aug-cc-pVDZ.

7.6 Interpreting SAPT Results

We will examine the results of a SAPT2+3/aug-cc-pVDZ computation on the water dimer. This computation can be performed with the following input:

```
molecule water_dimer {
     0 1
     O  -1.551007  -0.114520   0.000000
     H  -1.934259   0.762503   0.000000
     H  -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
     units angstrom
     no_reorient
     symmetry c1
}

set globals {
    basis          aug-cc-pvdz
    guess          sad
    scf_type       df
}

set sapt {
    print          1
    nat_orbs       true
    freeze_core    true
}

energy('sapt2+3')
```

To reiterate some of the options mentioned above: the *NAT_ORBS* option will compute MP2 natural orbitals and use them in the evaluation of the triples correction to dispersion, and the *FREEZE_CORE* option will freeze the core throughout the SAPT computation. This SAPT2+3/aug-cc-pVDZ computation produces the following results:

```
  SAPT Results
--------------------------------------------------------------------------
  Electrostatics            -13.06429805 mH      -8.19797114 kcal mol^-1
    Elst10,r                -13.37543274 mH      -8.39321111 kcal mol^-1
    Elst12,r                  0.04490253 mH       0.02817676 kcal mol^-1
    Elst13,r                  0.26623216 mH       0.16706321 kcal mol^-1

  Exchange                   13.41793548 mH       8.41988199 kcal mol^-1
    Exch10                   11.21823471 mH       7.03954885 kcal mol^-1
    Exch10(S^2)              11.13803867 mH       6.98922508 kcal mol^-1
    Exch11(S^2)               0.04558910 mH       0.02860760 kcal mol^-1
    Exch12(S^2)               2.15411167 mH       1.35172554 kcal mol^-1

  Induction                  -3.91333155 mH      -2.45565272 kcal mol^-1
    Ind20,r                  -4.57531220 mH      -2.87105187 kcal mol^-1
    Ind30,r                  -4.91715479 mH      -3.08556135 kcal mol^-1
    Ind22                    -0.83761074 mH      -0.52560870 kcal mol^-1
    Exch-Ind20,r              2.47828867 mH       1.55514969 kcal mol^-1
    Exch-Ind30,r              4.33916816 mH       2.72286924 kcal mol^-1
    Exch-Ind22                0.45370482 mH       0.28470409 kcal mol^-1
    delta HF,r (2)           -1.43240211 mH      -0.89884593 kcal mol^-1
    delta HF,r (3)           -0.85441547 mH      -0.53615383 kcal mol^-1

  Dispersion                 -3.62061213 mH      -2.27196851 kcal mol^-1
    Disp20                   -3.54292109 mH      -2.22321664 kcal mol^-1
    Disp30                    0.05959981 mH       0.03739945 kcal mol^-1
    Disp21                    0.11216179 mH       0.07038259 kcal mol^-1
    Disp22 (SDQ)             -0.17924270 mH      -0.11247650 kcal mol^-1
    Disp22 (T)               -0.47692549 mH      -0.29927528 kcal mol^-1
    Est. Disp22 (T)          -0.54385253 mH      -0.34127263 kcal mol^-1
    Exch-Disp20               0.64545652 mH       0.40503010 kcal mol^-1
    Exch-Disp30              -0.01823411 mH      -0.01144207 kcal mol^-1
    Ind-Disp30               -0.91816995 mH      -0.57616037 kcal mol^-1
    Exch-Ind-Disp30           0.76459013 mH       0.47978757 kcal mol^-1

  Total HF                   -5.68662366 mH      -3.56841037 kcal mol^-1
  Total SAPT0                -8.58408823 mH      -5.38659691 kcal mol^-1
  Total SAPT2                -6.72339084 mH      -4.21899163 kcal mol^-1
  Total SAPT2+               -7.26739725 mH      -4.56036082 kcal mol^-1
  Total SAPT2+(3)            -6.94156528 mH      -4.35589816 kcal mol^-1
  Total SAPT2+3              -7.11337921 mH      -4.46371303 kcal mol^-1
```

At the bottom of this output are the total SAPT energies (defined above), they are composed of subsets of the individual terms printed above. The individual terms are grouped according to the component of the interaction to which they contribute. The total component energies (*i.e.,* electrostatics, exchange, induction, and dispersion) represent what we regard as the best estimate available at a given level of SAPT computed from a subset of the terms of that grouping. The groupings shown above are not unique and are certainly not rigorously defined. We regard the groupings used in PSI4 as a "chemist's grouping" as opposed to a more mathematically based grouping, which would group all exchange terms (*i.e.* $E^{(20)}_{exch-ind,resp}$, $E^{(20)}_{exch-disp}$, *etc.* in the exchange component. A final note is that both `Disp22(T)` and `Est.Disp22(T)` results appear if MP2 natural orbitals are used to evaluate the triples correction to dispersion. The `Disp22(T)` result is the triples correction as computed in the truncated NO basis; `Est.Disp22(T)` is a scaled result that attempts to recover the effect of the truncated virtual space. The `Est.Disp22(T)` value used in the SAPT energy and dispersion component (see[Hohenstein:2010:104107]_ for details).

CHAPTER

EIGHT

ENERGY

`driver.`**`energy`**(*name*, ***kwargs*)

Function to compute the single-point electronic energy.

Returns (*float*) Total electronic energy in Hartrees. SAPT returns interaction energy.

Psi variables

CURRENT ENERGY
CURRENT REFERENCE ENERGY
CURRENT CORRELATION ENERGY

name	calls method
scf	Hartree–Fock (HF) or density functional theory (DFT) `manual`
mp2	2nd-order Moller-Plesset perturbation theory (MP2)
df-mp2	MP2 with density fitting
dcft	density cumulant functional theory
mcscf	multiconfigurational self consistent field (SCF)
dfcc	coupled cluster with density fitting
mp2c	coupled MP2 (MP2C)
mp2-drpa	random phase approximation?
sapt0	0th-order symmetry adapted perturbation theory (SAPT)
sapt2	2nd-order SAPT, traditional definition
sapt2+	SAPT including all 2nd-order terms
sapt2+(3)	SAPT including perturbative triples
sapt2+3	
sapt0-ct	0th-order SAPT plus charge transfer (CT) calculation
sapt2-ct	SAPT2 plus CT
sapt2+-ct	SAPT2+ plus CT
sapt2+(3)-ct	SAPT2+(3) plus CT
sapt2+3-ct	SAPT2+3 plus CT
cc2	approximate coupled cluster singles and doubles (CC2)
ccsd	coupled cluster singles and doubles (CCSD)
bccd	Brueckner coupled cluster doubles (BCCD)
cc3	approximate coupled cluster singles, doubles, and triples (CC3)
ccsd(t)	CCSD with perturbative triples
bccd(t)	BCCD with perturbative triples
ccenergy	**expert** full control over ccenergy module
mp *n*	*n* th-order Moller–Plesset perturbation theory
zapt *n*	*n* th-order z-averaged perturbation theory (ZAPT)
cisd	configuration interaction (CI) singles and doubles (CISD)

Continued on next page

Table 8.1 – continued from previous page

name	calls method
cisdt	CI singles, doubles, and triples (CISDT)
cisdtq	CI singles, doubles, triples, and quadruples (CISDTQ)
ci *n*	*n* th-order CI
fci	full configuration interaction (FCI)
detci	**expert** full control over detci module
cphf	coupled-perturbed Hartree-Fock?
cpks	coupled-perturbed Kohn-Sham?
cis	CI singles (CIS)
tda	Tamm-Dankoff approximation (TDA)
tdhf	time-dependent HF (TDHF)
tddft	time-dependent DFT (TDDFT)
adc	2nd-order algebraic diagrammatic construction (ADC)
eom-cc2	EOM-CC2
eom-ccsd	equation of motion (EOM) CCSD
eom-cc3	EOM-CC3

name	calls method in Kallay's MRCC program
mrccsd	CC through doubles
mrccsdt	CC through triples
mrccsdtq	CC through quadruples
mrccsdtqp	CC through quintuples
mrccsdtqph	CC through sextuples
mrccsd(t)	CC through doubles with perturbative triples
mrccsdt(q)	CC through triples with perturbative quadruples
mrccsdtq(p)	CC through quadruples with pertubative quintuples
mrccsdtqp(h)	CC through quintuples with pertubative sextuples
mrccsd(t)_l	
mrccsdt(q)_l	
mrccsdtq(p)_l	
mrccsdtqp(h)_l	
mrccsdt-1a	
mrccsdtq-1a	
mrccsdtqp-1a	
mrccsdtqph-1a	
mrccsdt-1b	
mrccsdtq-1b	
mrccsdtqp-1b	
mrccsdtqph-1b	
mrcc2	
mrcc3	
mrcc4	
mrcc5	
mrcc6	
mrccsdt-3	
mrccsdtq-3	
mrccsdtqp-3	
mrccsdtqph-3	

Parameters

- **name** (*string*) – `'scf'` || `'df-mp2'` || `'ci5'` || etc.

  First argument, usually unlabeled. Indicates the computational method to be applied to the system.

- **bypass_scf** (*bool*) – `'on'` || ⇒ `'off'` ⇐

  Indicates whether, for *name* values built atop of scf calculations, the scf step is skipped. Suitable when special steps are taken to get the scf to converge in an explicit preceeding scf step.

Examples

```
>>> # [1] Coupled-cluster singles and doubles calculation with psi code
>>> energy('ccsd')

>>> # [2] Charge-transfer SAPT calculation with scf projection from small into requested basis
>>> energy('sapt0-ct',cast_up=True)

>>> # [3] Arbitrary-order MPn calculation
>>> energy('mp4')
```

CHAPTER

NINE

OPTIMIZE

Note: The derivative level type for `driver.optimize()` and `driver.frequency()` functions can be specified by `energy`, `none`, or `0` for 0th derivative, `gradient`, `first`, or `1` for 1st derivative, and `hessian`, `second`, or `2` for 2nd derivative.

`driver.`**`optimize`**(*name*, ***kwargs*)

Function to perform a geometry optimization.

Aliases opt()

Returns (*float*) Total electronic energy of optimized structure in Hartrees.

Psi variables

`CURRENT ENERGY`

Note: Analytic gradients area available for all methods in the table below. Optimizations with other methods in the energy table proceed by finite differences.

Caution: Some features are not yet implemented. Buy a developer a coffee.

•Need to check that all methods do return electronic energy. I think gradient got changed at one point.

name	calls method
scf	Hartree–Fock (HF) or density functional theory (DFT)
mp2	2nd-order Moller-Plesset perturbation theory (MP2)
ccsd	coupled cluster singles and doubles (CCSD)
ccsd(t)	CCSD with perturbative triples
eom-ccsd	equation of motion (EOM) CCSD

Parameters

- **name** (*string*) – `'scf'` || `'df-mp2'` || `'ci5'` || etc.

  First argument, usually unlabeled. Indicates the computational method to be applied to the database. May be any valid argument to `driver.energy()`.

- **func** (*function*) – ⇒ `gradient` ⇐ || `energy` || `cbs`

  Indicates the type of calculation to be performed on the molecule. The default dertype accesses‘‘’gradient’‘‘ or `'energy'`, while `'cbs'` performs a multistage finite difference calculation. If a nested series of python functions is intended (see Function Intercalls), use keyword `opt_func` instead of `func`.

- **mode** (*string*) – ⇒ `'continuous'` ⇐ || `'sow'` || `'reap'`

  For a finite difference of energies optimization, indicates whether the calculations required to complete the optimization are to be run in one file (`'continuous'`) or are to be farmed out in an embarrassingly parallel fashion (`'sow'`/`'reap'`). For the latter, run an initial job with `'sow'` and follow instructions in its output file.

- **dertype** (*dertype*) – `'gradient'` || `'energy'`

  Indicates whether analytic (if available) or finite difference optimization is to be performed.

Examples

```
>>> # [1] Analytic scf optimization
>>> optimize('scf')

>>> # [2] Finite difference mp3 optimization
>>> opt('mp3')

>>> # [3] Forced finite difference ccsd optimization
>>> optimize('ccsd', dertype=1)
```

CHAPTER

TEN

RESPONSE

`driver.`**`response`**`(`*name*, ***kwargs*`)`

Function to compute linear response properties.

Returns (*float*) Total electronic energy in Hartrees.

Caution: Some features are not yet implemented. Buy a developer a coffee.
- Check that energy is actually being returned.
- Check if ther're some PSI variables that ought to be set.

name	calls method
cc2	2nd-order approximate CCSD
ccsd	coupled cluster singles and doubles (CCSD)

Parameters **name** (*string*) – `'ccsd'` || etc.

First argument, usually unlabeled. Indicates the computational method to be applied to the system.

Examples

```
>>> # [1] CCSD-LR properties calculation
>>> response('ccsd')
```

CHAPTER

ELEVEN

FREQUENCY

`driver.`**`frequency`**`(`*name*, ***kwargs*`)`

Function to compute harmonic vibrational frequencies.

Aliases frequencies(), freq()

Returns (*float*) Total electronic energy in Hartrees.

Caution: Some features are not yet implemented. Buy a developer a coffee.

- RAK, why are you adding OPTKING options as GLOBALS? And shouldn't they be Py-side not C-side options?
- Make frequency look analogous to gradient, especially in matching derivative levels. Make dertype actually a dertype type.

Parameters

- **name** (*string*) – `'scf'` || `'df-mp2'` || `'ci5'` || etc.

  First argument, usually unlabeled. Indicates the computational method to be applied to the system.

- **dertype** (*dertype*) – $\Rightarrow$ `'hessian'` $\Leftarrow$ || `'gradient'` || `'energy'`

  Indicates whether analytic (if available- they're not), finite difference of gradients (if available) or finite difference of energies is to be performed.

- **irrep** (*int or string*) – $\Rightarrow$ `-1` $\Leftarrow$ || `1` || `'b2'` || `'App'` || etc.

  Indicates which symmetry block (Cotton ordering) of vibrational frequencies to be computed. `1`, `'1'`, or `'a1'` represents a_1, requesting only the totally symmetric modes. `-1` indicates a full frequency calculation.

Examples

```
>>> # [1] <example description>
>>> <example python command>
```

```
>>> # [2] Frequency calculation for b2 modes through finite difference of gradients
>>> frequencies('scf', dertype=1, irrep=4)
```

CHAPTER

TWELVE

COUNTERPOISE CORRECT

`wrappers.`**`cp`**`(`*name*[, *func*, *check_bsse*]`)`

The cp function computes counterpoise-corrected two-body interaction energies for complexes composed of arbitrary numbers of monomers.

Aliases counterpoise_correct(), counterpoise_correction()

Returns (*float*) Counterpoise-corrected interaction energy in kcal/mol

Psi variables

`CP-CORRECTED 2-BODY INTERACTION ENERGY`
`UNCP-CORRECTED 2-BODY INTERACTION ENERGY`

Caution: Some features are not yet implemented. Buy a developer a coffee.

- No values of func besides energy have been tested.
- Table print-out needs improving. Add some PSI variables.

Parameters

- **name** (*string*) – `'scf'` || `'ccsd(t)'` || etc.

  First argument, usually unlabeled. Indicates the computational method to be applied to the molecule. May be any valid argument to `driver.energy()`; however, SAPT is not appropriate.

- **func** (*function*) – ⇒ `energy` ⇐ || `optimize` || `cbs`

  Indicates the type of calculation to be performed on the molecule and each of its monomers. The default performs a single-point `energy('name')`, while `optimize` perfoms a geometry optimization on each system, and `cbs` performs a compound single-point energy. If a nested series of python functions is intended (see Function Intercalls), use keyword `cp_func` instead of `func`.

- **check_bsse** (*bool*) – `'on'` || ⇒ `'off'` ⇐

  Indicates whether to additionally compute un-counterpoise corrected monomers and thus obtain an estimate for the basis set superposition error.

Examples

```
>>> # [1] counterpoise-corrected mp2 interaction energy
>>> cp('dfmp2')
```

CHAPTER

THIRTEEN

DATABASE

CHAPTER

FOURTEEN

COMPLETE BASIS SET

`wrappers.complete_basis_set(name[, scf_basis, scf_scheme, corl_wfn, corl_basis, corl_scheme, delta_wfn, delta_wfn_lesser, delta_basis, delta_scheme, delta2_wfn, delta2_wfn_lesser, delta2_basis, delta2_scheme ])`

Function to define a multistage energy method from combinations of basis set extrapolations and delta corrections and condense the components into a minimum number of calculations.

Aliases cbs()

Returns (*float*) – Total electronic energy in Hartrees

Psi variables

CBS TOTAL ENERGY
CBS REFERENCE ENERGY
CBS CORRELATION ENERGY
CURRENT ENERGY
CURRENT REFERENCE ENERGY
CURRENT CORRELATION ENERGY

Caution: Some features are not yet implemented. Buy a developer a coffee.

- Methods beyond basic scf, mp2, ccsd, ccsd(t) not yet hooked in through PSI variables, df-mp2 in particular.
- No scheme defaults for given basis zeta number, so scheme must be specified explicitly.
- No way to tell function to boost fitting basis size for all calculations.
- No way to extrapolate def2 family basis sets
- Need to add more extrapolation schemes

As represented in the equation below, a CBS energy method is defined in four sequential stages (scf, corl, delta, delta2) covering treatment of the reference total energy, the correlation energy, a delta correction to the correlation energy, and a second delta correction. Each is activated by its stage_wfn keyword and is only allowed if all preceding stages are active.

$$E_{total}^{\text{CBS}} = \mathcal{F}_{\textbf{scf\_scheme}} \left(E_{total,\ \text{SCF}}^{\textbf{scf\_basis}}\right) + \mathcal{F}_{\textbf{corl\_scheme}} \left(E_{corl,\ \textbf{corl\_wfn}}^{\textbf{corl\_basis}}\right) + \delta_{\textbf{delta\_wfn\_lesser}}^{\textbf{delta\_wfn}} + \delta_{\textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_wfn}}$$

Here, $\mathcal{F}$ is an energy or energy extrapolation scheme, and the following also hold.

$$\delta_{\textbf{delta\_wfn\_lesser}}^{\textbf{delta\_wfn}} = \mathcal{F}_{\textbf{delta\_scheme}} \left(E_{corl,\ \textbf{delta\_wfn}}^{\textbf{delta\_basis}}\right) - \mathcal{F}_{\textbf{delta\_scheme}} \left(E_{corl,\ \textbf{delta\_wfn\_lesser}}^{\textbf{delta\_basis}}\right)$$

$$\delta_{\textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_wfn}} = \mathcal{F}_{\textbf{delta2\_scheme}}\left(E_{corl,\ \textbf{delta2\_wfn}}^{\textbf{delta2\_basis}}\right) - \mathcal{F}_{\textbf{delta2\_scheme}}\left(E_{corl,\ \textbf{delta2\_wfn\_lesser}}^{\textbf{delta2\_basis}}\right)$$

A translation of this ungainly equation to example [5] below is as follows. In words, this is a double- and triple-zeta 2-point Helgaker-extrapolated CCSD(T) coupled-cluster correlation correction appended to a triple- and quadruple-zeta 2-point Helgaker-extrapolated MP2 correlation energy appended to a SCF/aug-cc-pVQZ reference energy.

$$E_{total}^{\text{CBS}} = \mathcal{F}_{\text{highest\_1}}\left(E_{total,\ \text{SCF}}^{\text{aug-cc-pVQZ}}\right) + \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}}\left(E_{corl,\ \text{MP2}}^{\text{aug-cc-pV[TQ]Z}}\right) + \delta_{\text{MP2}}^{\text{CCSD(T)}}$$

$$\delta_{\text{MP2}}^{\text{CCSD(T)}} = \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}}\left(E_{corl,\ \text{CCSD(T)}}^{\text{aug-cc-pV[DT]Z}}\right) - \mathcal{F}_{\text{corl\_xtpl\_helgaker\_2}}\left(E_{corl,\ \text{MP2}}^{\text{aug-cc-pV[DT]Z}}\right)$$

•**Energy Methods** The presence of a stage_wfn keyword is the indicator to incorporate (and check for stage_basis and stage_scheme keywords) and compute that stage in defining the CBS energy.

The cbs() function requires, at a minimum, `name='scf'` and `scf_basis` keywords to be specified for reference-step only jobs and `name` and `corl_basis` keywords for correlated jobs.

Parameters

- **name** (*string*) – `'scf'` || `'ccsd'` || etc.

  First argument, usually unlabeled. Indicates the computational method for the correlation energy, unless only reference step to be performed, in which case should be `'scf'`. Overruled if stage_wfn keywords supplied.

- **corl_wfn** (*string*) – `'mp2'` || `'ccsd(t)'` || etc.

  Indicates the energy method for which the correlation energy is to be obtained. Can also be specified with `name` or as the unlabeled first argument to the function.

- **delta_wfn** (*string*) – `'ccsd'` || `'ccsd(t)'` || etc.

  Indicates the (superior) energy method for which a delta correction to the correlation energy is to be obtained.

- **delta_wfn_lesser** (*string*) – $\Rightarrow$ `'mp2'` $\Leftarrow$ || `'ccsd'` || etc.

  Indicates the inferior energy method for which a delta correction to the correlation energy is to be obtained.

- **delta2_wfn** (*string*) – `'ccsd'` || `'ccsd(t)'` || etc.

  Indicates the (superior) energy method for which a second delta correction to the correlation energy is to be obtained.

- **delta2_wfn_lesser** (*string*) – $\Rightarrow$ `'mp2'` $\Leftarrow$ || `'ccsd(t)'` || etc.

  Indicates the inferior energy method for which a second delta correction to the correlation energy is to be obtained.

•**Basis Sets** Currently, the basis set set through `set` commands have no influence on a cbs calculation.

Parameters

- **scf_basis** (*string*) – ⇒ `corl_basis` ⇐ || `'cc-pV[TQ]Z'` || `'jun-cc-pv[tq5]z'` || `'6-31G*'` || etc.

  Indicates the sequence of basis sets employed for the reference energy. If any correlation method is specified, `scf_basis` can default to `corl_basis`.

- **corl_basis** (*string*) – `'cc-pV[TQ]Z'` || `'jun-cc-pv[tq5]z'` || `'6-31G*'` || etc.

  Indicates the sequence of basis sets employed for the correlation energy.

- **delta_basis** (*string*) – `'cc-pV[TQ]Z'` || `'jun-cc-pv[tq5]z'` || `'6-31G*'` || etc.

  Indicates the sequence of basis sets employed for the delta correction to the correlation energy.

- **delta2_basis** (*string*) – `'cc-pV[TQ]Z'` || `'jun-cc-pv[tq5]z'` || `'6-31G*'` || etc.

  Indicates the sequence of basis sets employed for the second delta correction to the correlation energy.

•**Schemes** Transformations of the energy through basis set extrapolation for each stage of the CBS definition. A complaint is generated if number of basis sets in stage_basis does not exactly satisfy requirements of stage_scheme. An exception is the default, `'highest_1'`, which uses the best basis set available. See Extrapolation Schemes for all available schemes.

Parameters

- **scf_scheme** (*function*) – ⇒ `highest_1` ⇐ || `scf_xtpl_helgaker_3` || etc.

  Indicates the basis set extrapolation scheme to be applied to the reference energy.

- **corl_scheme** (*function*) – ⇒ `highest_1` ⇐ || `corl_xtpl_helgaker_2` || etc.

  Indicates the basis set extrapolation scheme to be applied to the correlation energy.

- **delta_scheme** (*function*) – ⇒ `highest_1` ⇐ || `corl_xtpl_helgaker_2` || etc.

  Indicates the basis set extrapolation scheme to be applied to the delta correction to the correlation energy.

- **delta2_scheme** (*function*) – ⇒ `highest_1` ⇐ || `corl_xtpl_helgaker_2` || etc.

  Indicates the basis set extrapolation scheme to be applied to the second delta correction to the correlation energy.

Examples

```
>>> # [1] replicates with cbs() the simple model chemistry scf/cc-pVDZ: set basis cc-pVDZ energy
>>> cbs('scf', scf_basis='cc-pVDZ')

>>> # [2] replicates with cbs() the simple model chemistry mp2/jun-cc-pVDZ: set basis jun-cc-pVD
>>> cbs('mp2', corl_basis='jun-cc-pVDZ')

>>> # [3] DTQ-zeta extrapolated scf reference energy
>>> cbs('scf', scf_basis='cc-pV[DTQ]Z', scf_scheme=scf_xtpl_helgaker_3)

>>> # [4] DT-zeta extrapolated mp2 correlation energy atop a T-zeta reference
>>> cbs('mp2', corl_basis='cc-pv[dt]z', corl_scheme=corl_xtpl_helgaker_2)

>>> # [5] a DT-zeta extrapolated coupled-cluster correction atop a TQ-zeta extrapolated mp2 corr
>>> cbs('mp2', corl_basis='aug-cc-pv[tq]z', corl_scheme=corl_xtpl_helgaker_2, delta_wfn='ccsd(t)
>>> # [6] a D-zeta ccsd(t) correction atop a DT-zeta extrapolated ccsd cluster correction atop a
>>> cbs('mp2', corl_basis='aug-cc-pv[tq]z', corl_scheme=corl_xtpl_helgaker_2, delta_wfn='ccsd',

>>> # [7] cbs() coupled with database()
>>> database('mp2', 'BASIC', subset=['h2o','nh3'], symm='on', func=cbs, corl_basis='cc-pV[tq]z',
```

14.1 Output

At the beginning of a cbs() job is printed a listing of the individual energy calculations which will be performed. The output snippet below is from the example job [2] above. It shows first each model chemistry needed to compute the aggregate model chemistry requested through cbs(). Then, since, for example, an `energy('ccsd(t)')` yields CCSD(T), CCSD, MP2, and SCF energy values, the wrapper condenses this task list into the second list of minimum number of calculations which will actually be run.

```
Naive listing of computations required.
        scf / aug-cc-pvqz              for  SCF TOTAL ENERGY
        mp2 / aug-cc-pvtz              for  MP2 CORRELATION ENERGY
        mp2 / aug-cc-pvqz              for  MP2 CORRELATION ENERGY
    ccsd(t) / aug-cc-pvdz              for  CCSD(T) CORRELATION ENERGY
    ccsd(t) / aug-cc-pvtz              for  CCSD(T) CORRELATION ENERGY
        mp2 / aug-cc-pvdz              for  MP2 CORRELATION ENERGY
        mp2 / aug-cc-pvtz              for  MP2 CORRELATION ENERGY

Enlightened listing of computations required.
        mp2 / aug-cc-pvqz              for  MP2 CORRELATION ENERGY
    ccsd(t) / aug-cc-pvdz              for  CCSD(T) CORRELATION ENERGY
    ccsd(t) / aug-cc-pvtz              for  CCSD(T) CORRELATION ENERGY
```

At the end of a cbs() job is printed a summary section like the one below. First, in the components section, are listed the results for each model chemistry available, whether required for the cbs job (*) or not. Next, in the stages section, are listed the results for each extrapolation. The energies of this section must be dotted with the weightings in column Wt to get the total cbs energy. Finally, in the CBS section, are listed the results for each stage of the cbs procedure. The stage energies of this section sum outright to the total cbs energy.

```
==> Components <==

---------------------------------------------------------------------------------
               Method / Basis            Rqd   Energy [H]   Variable
---------------------------------------------------------------------------------
                  scf / aug-cc-pvqz        *  -1.11916375   SCF TOTAL ENERGY
                  mp2 / aug-cc-pvqz        *  -0.03407997   MP2 CORRELATION ENERGY
                  scf / aug-cc-pvdz           -1.11662884   SCF TOTAL ENERGY
                  mp2 / aug-cc-pvdz        *  -0.02881480   MP2 CORRELATION ENERGY
              ccsd(t) / aug-cc-pvdz        *  -0.03893812   CCSD(T) CORRELATION ENERGY
                 ccsd / aug-cc-pvdz           -0.03893812   CCSD CORRELATION ENERGY
                  scf / aug-cc-pvtz           -1.11881134   SCF TOTAL ENERGY
                  mp2 / aug-cc-pvtz        *  -0.03288936   MP2 CORRELATION ENERGY
              ccsd(t) / aug-cc-pvtz        *  -0.04201004   CCSD(T) CORRELATION ENERGY
                 ccsd / aug-cc-pvtz           -0.04201004   CCSD CORRELATION ENERGY
---------------------------------------------------------------------------------

==> Stages <==

---------------------------------------------------------------------------------
 Stage         Method / Basis             Wt   Energy [H]   Scheme
---------------------------------------------------------------------------------
```

```
   scf             scf / aug-cc-pvqz        1  -1.11916375   highest_1
  corl             mp2 / aug-cc-pv[tq]z     1  -0.03494879   corl_xtpl_helgaker_2
 delta         ccsd(t) / aug-cc-pv[dt]z     1  -0.04330347   corl_xtpl_helgaker_2
 delta             mp2 / aug-cc-pv[dt]z    -1  -0.03460497   corl_xtpl_helgaker_2
---------------------------------------------------------------------------------

==> CBS <==

---------------------------------------------------------------------------------
 Stage         Method / Basis                  Energy [H]   Scheme
---------------------------------------------------------------------------------
   scf             scf / aug-cc-pvqz           -1.11916375   highest_1
  corl             mp2 / aug-cc-pv[tq]z        -0.03494879   corl_xtpl_helgaker_2
 delta  ccsd(t) - mp2 / aug-cc-pv[dt]z         -0.00869851   corl_xtpl_helgaker_2
 total             CBS                         -1.16281105
---------------------------------------------------------------------------------
```

14.2 Extrapolation Schemes

`wrappers.`**`highest_1`**`(`***largs*`)`

Scheme for total or correlation energies with a single basis or the highest zeta-level among an array of bases. Used by `wrappers.complete_basis_set()`.

$$E_{total}^X = E_{total}^X$$

`wrappers.`**`scf_xtpl_helgaker_2`**`(`***largs*`)`

Extrapolation scheme for reference energies with two adjacent zeta-level bases. Used by `wrappers.complete_basis_set()`.

$$E_{total}^X = E_{total}^\infty + \beta e^{-\alpha X}, \alpha = 1.63$$

`wrappers.`**`scf_xtpl_helgaker_3`**`(`***largs*`)`

Extrapolation scheme for reference energies with three adjacent zeta-level bases. Used by `wrappers.complete_basis_set()`.

$$E_{total}^X = E_{total}^\infty + \beta e^{-\alpha X}$$

`wrappers.`**`corl_xtpl_helgaker_2`**`(`***largs*`)`

Extrapolation scheme for correlation energies with two adjacent zeta-level bases. Used by `wrappers.complete_basis_set()`.

$$E_{corl}^X = E_{corl}^\infty + \beta X^{-3}$$

CHAPTER

FIFTEEN

FRACTIONAL OCCUPATION

`frac.`**`frac_nuke`**(*mol*, ***kwargs*)

`frac.`**`frac_traverse`**(*mol*, ***kwargs*)

`frac.`**`ip_fitting`**(*mol*, *omega_l*, *omega_r*, ***kwargs*)

CHAPTER

SIXTEEN

BEGINNER PSITHON PROGRAMMING

Note: No recompile of the PSI program is necessary for changes made to files in `$PSIDATADIR`, including those described below.

16.1 Defining a Method Alias

Since quantum chemical methods in PSI4 are accessed through Python functions, and most important quantities are available as PSI variables, it is straightforward to create aliases to commonly run calculations or to define hybrid methods. The `$PSIDATADIR/python/aliases.py` file is intended for editing by the user for this purpose.

As an example, the MP2.5 method is the average of MP2 and MP3. The latter is available through the arbitrary order MPn code and returns all lower energies along with it in PSI variables. The following is basic code that will compute and return the MP2.5 energy.

```
def run_mp2_5(name, **kwargs):

    energy('mp3', **kwargs)
    e_scf = PsiMod.get_variable('SCF TOTAL ENERGY')
    ce_mp2 = PsiMod.get_variable('MP2 CORRELATION ENERGY')
    ce_mp3 = PsiMod.get_variable('MP3 CORRELATION ENERGY')

    ce_mp25 = 0.5 * (ce_mp2 + ce_mp3)
    e_mp25 = e_scf + ce_mp25

    print """  MP2.5 total energy:                      %16.8f\n""" % (e_mp25)
    print """  MP2.5 correlation energy:                %16.8f\n""" % (ce_mp25)

    return e_mp25
```

Compare the above to the method that resides in `aliases.py`. The rationale for the changes is indicated in the comments below.

```
def run_mp2_5(name, **kwargs):
    lowername = name.lower()  # handy variable with name keyword in lowercase
    kwargs = kwargs_lower(kwargs)  # removes case sensitivity in keyword names

    # Run detci calculation and collect conventional quantities
    energy('mp3', **kwargs)
    e_scf = PsiMod.get_variable('SCF TOTAL ENERGY')
    ce_mp2 = PsiMod.get_variable('MP2 CORRELATION ENERGY')
    ce_mp3 = PsiMod.get_variable('MP3 CORRELATION ENERGY')
    e_mp2 = e_scf + ce_mp2  # reform mp2 and mp3 total energies for printing
    e_mp3 = e_scf + ce_mp3

    # Compute quantities particular to MP2.5
    ce_mp25 = 0.5 * (ce_mp2 + ce_mp3)
    e_mp25 = e_scf + ce_mp25
    PsiMod.set_variable('MP2.5 CORRELATION ENERGY', ce_mp25)  # add new method's important results
    PsiMod.set_variable('MP2.5 TOTAL ENERGY', e_mp25)         #     to PSI variable repository
    PsiMod.set_variable('CURRENT CORRELATION ENERGY', ce_mp25)
    PsiMod.set_variable('CURRENT ENERGY', e_mp25)  # geometry optimizer tracks this variable, permit
                                                   #     MP2.5 finite difference optimizations
    # build string of title banner and print results
    banners = ''
    banners += """PsiMod.print_out('\\n')\n"""
    banners += """banner(' MP2.5 ')\n"""
    banners += """PsiMod.print_out('\\n')\n\n"""
    exec banners

    tables  = ''
    tables += """  SCF total energy:                        %16.8f\n""" % (e_scf)
    tables += """  MP2 total energy:                        %16.8f\n""" % (e_mp2)
    tables += """  MP2.5 total energy:                      %16.8f\n""" % (e_mp25)
    tables += """  MP3 total energy:                        %16.8f\n\n""" % (e_mp3)
    tables += """  MP2 correlation energy:                  %16.8f\n""" % (ce_mp2)
    tables += """  MP2.5 correlation energy:                %16.8f\n""" % (ce_mp25)
    tables += """  MP3 correlation energy:                  %16.8f\n""" % (ce_mp3)
    PsiMod.print_out(tables)  # prints nice header and table of all involved quantities to output fi

    return e_mp25
```

One final step is necessary. At the end of the aliases.py file, add the following line.

```
procedures['energy']['mp2.5'] = run_mp2_5
```

This permits the newly defined MP2.5 method to be called in the input file with the following command.

```
energy('mp2.5')
```

16.2 Creating a Database

A necessary consideration in constructing a database is the distinction between reagents and reactions. A reagent is a single molecular system (may be a dimer) whose geometry you are possession of and whose electronic energy may be of interest. A reaction is a combination of one or more reagent energies whose value you are interested in and a reference value for which you may or may not be in possession of. A few examples follow. In a database of interaction energies, the reagents are dimers and their component monomers (usually derived from the dimer geometry), and the reactions are the dimer less monomers energies. In a database of barrier heights, the reagents are reactants, products, and transition-state structures, and the reactions are the transition-states less minimum-energy structures. Possibly you may have a collection of structures to simply be acted upon in parallel, in which case the structures are both the reagents and the reactions. The role of the database.py file is to collect arrays and dictionaries that define the geometries of reagents (GEOS), their combination into reactions (RXNM & ACTV), available reference values for reactions (BIND), and brief comments for reagents and reactions (TAGL). The journey from reagent geometries to functional database.py file is largely automated, in a process described below.

- **Prepare geometry files** Assemble xyz files for all intended reagent systems in a directory. Follow the rules below for best results. The filename for each xyz file should be the name of the system. lowercase or MixedCase is preferable (according to Sherrill lab convention). Avoid dashes and dots in the name as python won't allow them. If you're determined to have dashes and dots, they must be replaced by other characters in the process_input line, then translated back in the GEOS section; see NBC10.py for an example.

  - The first line for each xyz file should be the number of atoms in the system.
  - The second line for each xyz file can be blank (interpreted as no comment), anything (interpreted as a comment), or two integers and anything (interpreted as charge, multiplicity, and remainder as comment).
  - The third and subsequent lines have four fields: the element symbol and the three cartesian coordinates in angstroms. The atom lines should not contain any dummy atoms (what's the use in cartesian form). For dimer systems, an algorithm is used to apportion the atoms into two fragments; thus the atoms need not be arranged with all fragmentA atoms before all fragmentB atoms. The algorithm will fail for very closely arranged fragments. For dimers, any charge and multiplicity from the second line will be applied to fragmentA (python); charge and multiplicity may need to be redistributed later in the editing step.

- Run script ixyz2database.pl

  Move into the directory where all your xyz files are located. Run the script in place, probably as `$PSIDATADIR/databases/ixyz2database.pl`. It will ask a number of questions about your intended database and generate a python file named for your database. Uppercase is preferable for database names (according to Sherrill lab convention). Note your choice for the route variable for the next step.

- Edit file database.py

  According to your responses in to questions in the ixyz2database.pl script, several bullets will be printed of edits you necessarily or optionally should make. Copy your new database into `$PSIDATADIR/databases`.

CHAPTER

SEVENTEEN

FUNCTION INTERCALLS

For many of the PSI4 Python functions described above, it makes scientific sense that they could be called in combination. For instance, one could optimize all the reagents in a database or compute a counterpoise-corrected interaction energy with an extrapolated method. The table below outlines permitted intercalls between functions, showing that db(opt(cbs(energy()))) is allowed, while db(cp(energy())) is not. This table is not yet validated for calls with cp().

Caller	**Callee**				
	cp	**db**	**opt**	**cbs**	**energy**
cp		—	Y	Y	Y
db	—		Y	Y	Y
opt	—	—		Y	Y
cbs	—	—	—		Y
energy	—	—	—	—	

- The command db(opt(cbs(energy()))) is actually expressed as `db(..., db_func=opt, opt_func=cbs)`. The perhaps expected final argument of `cbs_func=energy` is not necessary since energy() is always the function called by default. Also, the outermost internal function call (`db_func` above can be called as just `func`. Several examples of intercalls between Python functions can be found in sample input *pywrap_all*.
- All keyword arguments are passed along to each function traversed in the

Python driver, so there should be no concern for separating them, grouping them, or designating them for a particular function when undertaking a nested calculation. Where the same keyword is used by multiple functions, prefixes are added, e.g., **db_mode** and **opt_mode**.

- Function intercalls should not be used in sow/reap mode.

CHAPTER

EIGHTEEN

EMBARRASSING PARALLELISM

Many of the tasks automated by Python wrappers consist of a number of independent psi4 calculations and are thus suited to an embarrassingly parallel mode of operation. In Psithon, these have been dubbed sow/reap procedures and have the following general structure.

- Prepare an input file, simply adding `mode='sow'` to the argument list of an available Python function. Run this quick job to produce input files for lengthier calculations.
- According to the instructions in the output file of the above step, run the generated input files in any order on any variety of computers and architectures. This is the time-intensive portion of the calculation.
- The 'sow' stage also produces a *master* input file (with a `mode='reap'` directive). When all the jobs in the above step are completed, place their output files in the same location as the *master* input, and run this last, quick job to collect the results.
- Sow/reap procedures are governed by the **mode** keyword, choices being `'continuous'`, `'sow'`, and `'reap'`. Only `'sow'` is likely to be used by the user, as `'continuous'` is always the default, and input files with `'reap'` are autogenerated.
- Available at present for Database and finite difference operation of Optimize.

Caution: Some features are not yet implemented. Buy a developer a coffee.

- Local options (e.g., `set scf e_convergence 9`) will not get transmitted to the child jobs.
- Array options (e.g., `set states_per_irrep [2, 1]`) will not get transmitted to the child jobs.
- Function intercalls (e.g., db(opt())) are not tested with sow/reap procedures.

CHAPTER

NINETEEN

PSITHON PROGRAMMING BEST PRACTICES

- Thy python functions shall always have final argument **kwargs, that they may take in and pass on keywords meant for other functions. Yea, even the run_mcscf(), and run_ccsd() -type functions that have no use for kwargs. The exceptions are python functions that are only helpers called by a driver function.
- Python functions should read the kwargs dictionary and (possibly) add to it. Functions should not pop or remove keywords from kwargs, even those keywords meaningful only to itself. This will ensure that the complete kwargs is available for pickling and sow/reap procedures. The exception is the molecule argument, which is read by the first function that gets ahold of it. This first function activates the molecule and pops it out of kwargs, effectively setting molecule for all subsequent functions. The code below should suffice.

```
# Make sure the molecule the user provided is the active one
if 'molecule' in kwargs:
    activate(kwargs['molecule'])
    del kwargs['molecule']
molecule = PsiMod.get_active_molecule()
molecule.update_geometry()
```

- Preferrably, the python function signature (for functions intended to be called in input files) is `function(name, **kwargs)`. For functions that have other positional keywords, please bundle them into kwargs at earliest convenience (see Database argument db_name for example).
- After the docstring, the first two lines of your function should be the ones below. The first provides a case insensitive handle to the name argument value. The second converts all the kwargs dictionary keys to lowercase versions of themselves, so that input files can be case insensitive.

```
lowername = name.lower()
kwargs = kwargs_lower(kwargs)
```

- Case sensitivity for kwargs dictionary values still needs to be handled. The first line below shows how to convert argument values to lowercase for matching. When not matching a whole value such that regular expressions are needed, the second line below performs a case insensitive match.

```
if (kwargs['db_mode'].lower() == 'continuous'):
if re.match(r'^sapt', name, flags=re.IGNORECASE):
```

- Match boolean keywords (db_cp in the example below) with expressions like the following, which allow case insensitive yes/true/on/1/no/false/off/0 user input. If your argument's value is a derivative level, similarly, use input.der0th, input.der1st, and input.der2nd.

```
if input.yes.match(str(db_cp)):
elif input.no.match(str(db_cp)):
```

- For keywords that might be used in other functions as well as your own, prepend the argument name with a short representation of your function name. For example, there are keywords cp_func, db_func, and opt_func to request what python function, if not energy(), is called by cp(), database(), and optimize().
- Upon checking in a new python file, edit the file `psi4/doc/userman/source/index.rst` and follow the instructions therein that your file may be autodocumented here.
- Write docstrings! For a major function intended for use in input files, start with the skeleton docstring in `psi4/lib/python/example_docstring` and replace anything that looks like `<this>`. For a behind-the-scenes function or if you don't want the bother of dealing with reStructuredText, just write an ordinary docstring. It will get slurped into the documentation in plain text.
- Your python function should follow PEP8 conventions (without the line-length restriction). I'm aiming for files to pass the line below, unless for good reason. The second line is for database Python files.

```
>>> pep8.py -r --ignore=E501 pythonfile.py
>>> pep8.py -r --ignore=E501,E221,E222,E241,E201,E202 databasefile.py
```

- Your python function should not prevent any test case (`make tests`, NOT `make longtests`) from passing. A test case(s) should be written and checked in for any major python function, so that others do not break your code. If most of your work was on the python (as opposed to c++) side, the test case prefix pywrap_ is suggested.
- Be sure to set any new PSI variables through lines like those below. Especially if the function returns an energy, set the 'current energy' variable. This last is needed to communicate with the optimizer.

```
PsiMod.set_variable('MP2.5 CORRELATION ENERGY', ce_mp25)
PsiMod.set_variable('MP2.5 TOTAL ENERGY', e_mp25)
PsiMod.set_variable('CURRENT ENERGY', e_mp25)
```

- Once your python function is fairly stable on its own, it's potential for interoperability with energy()/opt()/cp()/db()/cbs()/etc. should be evaluated. If it makes physical sense that it should work, you should strive to make that interoperability a reality. Some steps:
  - If any interoperability is possible, define an argument xx_func, where xx is a short name for your function. Add near the top of your function code like the below (less the final two lines). The net result of this code is that if the user specifies no *_func arguments, then energy() gets called. If the user defines xx_func, then its value gets called. If the user defines func, then its value gets reassigned to xx_func, func itself is deleted, and xx_func() gets called. Whatever is getting called is stored in func within the function.

    ```
    # Establish function to call
    if not('xx_func' in kwargs):
        if ('func' in kwargs):
            kwargs['xx_func'] = kwargs['func']
            del kwargs['func']
        else:
            kwargs['xx_func'] = energy
    func = kwargs['xx_func']
    if not func:
        raise ValidationError('Function \'%s\' does not exist to be called by wrapper counterpoi
    if (func is db):
        raise ValidationError('Wrapper xx is unhappy to be calling function \'%s\'.' % (func.__n
    ```

  - If specific interoperabilities are known, code them in. For example, if xx shouldn't call db, add the last two lines above to the xx function. If db shouldn't call xx, add the following two lines below to the db function.

    ```
    if (func is xx):
        raise ValidationError('Wrapper database is unhappy to be calling function \'%s\'.' % (fu
    ```

- Create a multipart test case that runs some intercalls between your function and others (akin to pywrap_all). In trials, permute the order of calls a few times to expose any calls that don't clean up after themselves and need further attention.
- When all is validated, add your findings to the great interoperability table in the documentation.

CHAPTER

TWENTY

EXPERT: PSIMOD MODULE

20.1 A Secondary Header

CHAPTER

TWENTYONE

EXPERT: PYTHON MODULES

21.1 aliases

Module with functions that call upon those in modules `proc`, `driver`, and `wrappers`.

Place in this file quickly defined procedures such as

- aliases for complex methods
- simple modifications to existing methods

`aliases.`**`run_cim`**(*name*, ***kwargs*)
Eugene's CIM driven by Python

`aliases.`**`run_mp2_5`**(*name*, ***kwargs*)
Function that computes MP2.5 energy from results of a DETCI MP3 calculation.

```
>>> energy('mp2.5')
```

`aliases.`**`run_plugin_ccsd_serial`**(*name*, ***kwargs*)
Function encoding sequence of PSI module and plugin calls so that Eugene DePrince's ccsd_serial plugin can be called via `driver.energy()`.

```
>>> energy('plugin_ccsd_serial')
```

`aliases.`**`run_plugin_omega`**(*name*, ***kwargs*)
Function encoding sequence of PSI module and plugin calls, as well as typical options, to access Rob Parrish's omega plugin.

```
>>> energy('plugin_omega')
```

`aliases.`**`sherrillgroup_gold_standard`**(*name='mp2'*, ***kwargs*)
Function to call the quantum chemical method known as 'Gold Standard' in the Sherrill group. Uses `wrappers.complete_basis_set()` to evaluateo the following expression. Two-point extrapolation of the correlation energy performed according to `wrappers.corl_xtpl_helgaker_2()`.

$$E_{total}^{\text{Au\_std}} = E_{total,\ \text{SCF}}^{\text{aug-cc-pVQZ}} + E_{corl,\ \text{MP2}}^{\text{aug-cc-pV[TQ]Z}} + \delta_{\text{MP2}}^{\text{CCSD(T)}}\big|_{\text{aug-cc-pVTZ}}$$

```
>>> energy('sherrillgroup_gold_standard')
```

21.2 driver

Module with a *procedures* dictionary specifying available quantum chemical methods and functions driving the main quantum chemical functionality, namely single-point energies, geometry optimizations, response properties, and vibrational frequency calculations.

`driver.`**`gradient`**(*name*, ***kwargs*)
: Function complementary to optimize(). Carries out one gradient pass, deciding analytic or finite difference.

`driver.`**`hessian`**(*name*, ***kwargs*)
: Function to compute force constants. Presently identical to frequency().

`driver.`**`parse_arbitrary_order`**(*name*)
: Function to parse name string into a method family like CI or MRCC and specific level information like 4 for CISDTQ or MRCCSDTQ.

`driver.`**`parse_cotton_irreps`**(*irrep*)
: Function to return validated Cotton ordering index from string or integer irreducible representation *irrep*.

21.3 input

Module import

`input.`**`process_input`**(*raw_input*, *print_level=1*)

21.4 molutil

Module with utility functions that act on molecule objects.

`molutil.`**`activate`**(*mol*)
: Function to set molecule object *mol* as the current active molecule.

`molutil.`**`dynamic_variable_bind`**(*cls*)
: Function to bind PsiMod.Molecule class.

`molutil.`**`extract_cluster_indexing`**(*mol*, *cluster_size=0*)
: Function to returns a LIST of all subclusters of the molecule *mol* of real size *cluster_size*. If *cluster_size* = 0, returns all possible combinations of cluster size.

`molutil.`**`extract_clusters`**(*mol*, *ghost=True*, *cluster_size=0*)
: Function to return all subclusters of the molecule *mol* of real size *cluster_size* and all other atoms ghosted if *ghost* equals true, all other atoms discarded if *ghost* is false. If *cluster_size* = 0, returns all possible combinations of cluster size.

`molutil.`**`geometry`**(*geom*, *name='default'*)
: Function to create a molecule object of name *name* from the geometry in string *geom*.

`molutil.`**`new_get_attr`**(*self*, *name*)
: Function to redefine get_attr method of molecule class.

`molutil.`**`new_set_attr`**(*self*, *name*, *value*)
: Function to redefine set_attr method of molecule class.

21.5 physconst

21.6 proc

Module with functions that encode the sequence of PSI module calls for each of the *name* values of the energy(), optimize(), response(), and frequency() function.

proc.**run_adc**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for an algebraic diagrammatic construction calculation.

Caution: Get rid of active molecule lines- should be handled in energy.

proc.**run_bccd**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a Brueckner CCD calculation.

proc.**run_bccd_t**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a Brueckner CCD(T) calculation.

proc.**run_cc_gradient**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a CCSD and CCSD(T) gradient calculation.

proc.**run_cc_response**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a CC2 and CCSD calculation.

proc.**run_ccenergy**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a CCSD, CC2, and CC3 calculation.

proc.**run_dcft**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a density cumulant functional theory calculation.

proc.**run_detci**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a configuration interaction calculation, namely FCI, CIn, MPn, and ZAPTn.

proc.**run_dfcc**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a density-fitted coupled-cluster calculation.

proc.**run_dfmp2**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a density-fitted MP2 calculation.

Caution: Get rid of madness-era restart file

proc.**run_eom_cc**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for an EOM-CC calculation, namely EOM-CC2, EOM-CCSD, and EOM-CC3.

proc.**run_eom_cc_gradient**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for an EOM-CCSD gradient calculation.

proc.**run_libfock**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a calculation through libfock, namely RCPHF, RCIS, RTDHF, RTDA, and RTDDFT.

proc.**run_mcscf**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a multiconfigurational self-consistent-field calculation.

proc.**run_mp2**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a MP2 calculation.

proc.**run_mp2_gradient**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a MP2 gradient calculation.

proc.**run_mp2c**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a coupled MP2 calculation.

proc.**run_mp2drpa**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a MP2-DRPA calculation.

proc.**run_mrcc**(*name*, ***kwargs*)
Function that prepares environment and input files for a calculation calling Kallay's MRCC code.

proc.**run_psimrcc**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a PSIMRCC computation using a reference from the MC-SCF module

proc.**run_psimrcc_scf**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a PSIMRCC computation using a reference from the SCF module

proc.**run_sapt**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a SAPT calculation of any level.

proc.**run_sapt_ct**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a charge-transfer SAPT calcuation of any level.

proc.**run_scf**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a self-consistent-field theory (HF & DFT) calculation.

proc.**run_scf_gradient**(*name*, ***kwargs*)
Function encoding sequence of PSI module calls for a SCF gradient calculation.

proc.**scf_helper**(*name*, ***kwargs*)
Function serving as helper to SCF, choosing whether to cast up or just run SCF with a standard guess. This preserves previous SCF options set by other procedures (e.g., SAPT output file types for SCF).

21.7 procutil

Module with utility functions used by several Python functions.

procutil.**format_kwargs_for_input**(*filename*, *lmode=1*, ***kwargs*)
Function to pickle to file *filename* the options dictionary *kwargs*. Mode *lmode* =2 pickles appropriate settings for reap mode. Used to capture Python options information for distributed (sow/reap) input files.

procutil.**format_molecule_for_input**(*mol*)
Function to return a string of the output of `input.process_input()` applied to the XYZ format of molecule *mol*. Used to capture molecule information for distributed (sow/reap) input files.

procutil.**format_options_for_input**()
Function to return a string of commands to replicate the current state of user-modified options. Used to capture C++ options information for distributed (sow/reap) input files.

Caution: Some features are not yet implemented. Buy a developer a coffee.
- Does not cover local (as opposed to global) options.
- Does not work with array-type options.

procutil.**get_psifile**(*fileno*, *pidspace='32433'*)
Function to return the full path and filename for psi file *fileno* (e.g., psi.32) in current namespace *pidspace*.

procutil.**kwargs_lower**(*kwargs*)
Function to rebuild and return *kwargs* dictionary with all keys made lowercase. Should be called by every function that could be called directly by the user.

21.8 psiexceptions

Module with non-generic exceptions classes.

exception psiexceptions.**PsiException**
Error class for Psi.

exception psiexceptions.**ValidationError**(*msg*)
Error called for problems with the input file. Prints error message *msg* to standard output stream and output file.

21.9 pubchem

Queries the PubChem database using a compound name (i.e. 1,3,5-hexatriene) to obtain a molecule string that can be passed to Molecule.

```
results = getPubChemObj("1,3,5-hexatriene")

Results is an array of results from PubChem matches to your query.
  for entry in results:
     entry["CID"]         => PubChem compound identifer
     entry["IUPAC"]       => IUPAC name for the resulting compound
     entry["PubChemObj"]  => instance of PubChemObj for this compound

     entry["PubChemObj"].getMoleculeString()   => returns a string compatible
                                                  with PSI4's Molecule creation
```

class pubchem.**PubChemObj**(*cid*, *mf*, *iupac*)

getCartesian()
Function to return a string of the atom symbol and XYZ coordinates of the PubChem object.

getMoleculeString()
Function to obtain a molecule string through getCartesian() or fail.

getSDF()
Function to return the SDF (structure-data file) of the PubChem object.

getXYZFile()
Function to obtain preferentially a molecule string through getCartesian() or a query string otherwise.

name()
Function to return the IUPAC name of the PubChem object.

pubchem.**getPubChemResults**(*name*)
Function to query the PubChem database for molecules matching the input string. Builds a PubChem object if found.

21.10 qmmm

Module with classes to integrate MM charges into a QM calculation.

class `qmmm.`**`Diffuse`**(*molecule*, *basisname*, *ribasisname*)

> **`fitGeneral`**()
> Function to perform a general fit of diffuse charges to wavefunction density.
>
> **`fitScf`**()
> Function to run scf and fit a system of diffuse charges to resulting density.
>
> **`populateExtern`**(*extern*)

class `qmmm.`**`QMMM`**

> **`addChargeAngstrom`**(*Q*, *x*, *y*, *z*)
> Function to add a point charge of magnitude *Q* at position (*x*, *y*, *z*) Angstroms.
>
> **`addChargeBohr`**(*Q*, *x*, *y*, *z*)
> Function to add a point charge of magnitude *Q* at position (*x*, *y*, *z*) Bohr.
>
> **`addDiffuse`**(*diffuse*)
> Function to add a diffuse charge field *diffuse*.
>
> **`populateExtern`**()
> Function to define a charge field external to the molecule through point and diffuse charges.

21.11 text

Module with utility classes and functions related to data tables and text.

class `text.`**`Table`**(*rows=()*, *row_label_width=10*, *row_label_precision=4*, *cols=()*, *width=16*, *precision=10*)
Class defining a flexible Table object for storing data.

> **`absolute_to_relative`**(*Factor=627.5095*)
> Function to shift the data of each column of the Table object such that the lowest value is zero. A scaling factor of *Factor* is applied.
>
> **`copy`**()
> Function to return a copy of the Table object.
>
> **`format_label`**()
> Function to pad the width of Table object labels.
>
> **`format_values`**(*values*)
> Function to pad the width of Table object data cells.
>
> **`save`**(*file*)
> Function to save string of the Table object to *file*.
>
> **`scale`**(*Factor=627.5095*)
> Function to apply a scaling factor *Factor* to the data of the Table object.

`text.`**`banner`**(*text*, *type=1*, *width=35*)
Function to print *text* to output file in a banner of minimum width *width* and minimum three-line height for *type* = 1 or one-line height for *type* = 2.

`text.`**`print_stderr`**(*stuff*)
Function to print *stuff* to standard error stream.

text.**print_stdout**(*stuff*)
    Function to print *stuff* to standard output stream.

21.12 util

Module with utility functions for use in input files.

util.**compare_integers**(*expected*, *computed*, *label*)
    Function to compare two integers. Prints `util.success()` when value *computed* matches value *expected*. Performs a system exit on failure. Used in input files in the test suite.

util.**compare_matrices**(*expected*, *computed*, *digits*, *label*)
    Function to compare two matrices. Prints `util.success()` when elements of matrix *computed* match elements of matrix *expected* to number of *digits*. Performs a system exit on failure to match symmetry structure, dimensions, or element values. Used in input files in the test suite.

util.**compare_strings**(*expected*, *computed*, *label*)
    Function to compare two strings. Prints `util.success()` when string *computed* exactly matches string *expected*. Performs a system exit on failure. Used in input files in the test suite.

util.**compare_values**(*expected*, *computed*, *digits*, *label*)
    Function to compare two values. Prints `util.success()` when value *computed* matches value *expected* to number of *digits*. Performs a system exit on failure. Used in input files in the test suite.

util.**compare_vectors**(*expected*, *computed*, *digits*, *label*)
    Function to compare two vectors. Prints `util.success()` when elements of vector *computed* match elements of vector *expected* to number of *digits*. Performs a system exit on failure to match symmetry structure, dimension, or element values. Used in input files in the test suite.

util.**get_memory**()
    Function to return the total memory allocation.

util.**get_num_threads**()
    Function to return the number of threads to parallelize across.

util.**set_memory**(*bytes*)
    Function to reset the total memory allocation.

util.**set_num_threads**(*nthread*)
    Function to reset the number of threads to parallelize across.

util.**success**(*label*)
    Function to print a '*label*...PASSED' line to screen. Used by `util.compare_values()` family when functions pass.

21.13 wrappers

Module with functions that call the four main `driver` functions: `driver.energy`, `driver.optimize`, `driver.response`, and `driver.frequency`.

wrappers.**call_function_in_1st_argument**(*funcarg*, ***largs*)
    Function to make primary function call to energy(), opt(), etc. with options dictionary *largs*. Useful when *funcarg* to call is stored in variable.

wrappers.**drop_duplicates**(*seq*)
    Function that given an array *seq*, returns an array without any duplicate entries. There is no guarantee of which duplicate entry is dropped.

`wrappers.`**`n_body`**(*name*, ***kwargs*)

`wrappers.`**`reconstitute_bracketed_basis`**(*needarray*)
Function to reform a bracketed basis set string from a sequential series of basis sets (e.g, form 'cc-pv[q5]z' from array [cc-pvqz, cc-pv5z]). The basis set array is extracted from the *f_basis* field of a *NEED* dictionary in `wrappers.complete_basis_set()`. Result is used to print a nicely formatted basis set string in the results table.

`wrappers.`**`split_menial`**(*menial*)
Function used by `wrappers.complete_basis_set()` to separate *menial* 'scftot' into [scf, tot] and 'mp2corl' into [mp2, corl].

`wrappers.`**`tblhead`**(*tbl_maxrgt*, *tbl_delimit*, *ttype*)
Function that prints the header for the changable-width results tables in db(). *tbl_maxrgt* is the number of reagent columns the table must plan for. *tbl_delimit* is a string of dashes of the correct length to set off the table. *ttype* is 1 for tables comparing the computed values to the reference or 2 for simple tabulation and sum of the computed values.

`wrappers.`**`validate_bracketed_basis`**(*basisstring*)
Function to transform and validate basis sets for cbs(). A basis set with no paired square brackets is passed through with zeta level 0 (e.g., '6-31+G(d,p)' is returned as [6-31+G(d,p)] and [0]). A basis set with square brackets is checked for sensible sequence and Dunning-ness and returned as separate basis sets (e.g., 'cc-pV[Q5]Z' is returned as [cc-pVQZ, cc-pV5Z] and [4, 5]). Note that this function has no communication with the basis set library to check if the basis actually exists. Used by `wrappers.complete_basis_set()`.

`wrappers.`**`validate_scheme_args`**(*functionname*, ***largs*)
Function called by each extrapolation scheme in `wrappers.complete_basis_set()`. Checks that all the input arguments are present and suitable so that the scheme function can focus on defining the extrapolation.

CHAPTER

TWENTYTWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

CHAPTER

TWENTYTHREE

REFERENCES

CHAPTER

TWENTYFOUR

GLOSSARY

24.1 SAPT

DEBUG Print lots of intermediate quantities that are not usually interesting. For SAPT, it will also do additional work (which is not optimized for large systems) so don't turn it on.

- **Type**: integer
- **Default**: 0

XXX xxxxx

- **Type**:
- **Possible Values**:
- **Default**:

AIO-CPHF Do disk I/O asynchronously during the solution of the CPHF equations. This option may speed up the computation slightly, however its use will cause PSI4 to spawn an additional thread.

- **Type**: boolean
- **Default**: FALSE

AIO_DF_INTS Do disk I/O asynchronously during the formation of the DF integrals. This option may speed up the computation slightly, however its use will cause PSI4 to spawn an additional thread.

- **Type**: boolean
- **Default**: FALSE

BASIS The basis set used to describe the monomer molecular orbitals.

- **Type**: string
- **Possible Values**: Basis Sets
- **Default**: none

D_CONVERGENCE Convergence of the residual of the CPHF coefficients needed for the $E^{(20)}_{ind,resp}$.

- **Type**: conv double
- **Default**: 1.0×10^{-8}

DENOMINATOR_ALGORITHM Should the energy denominators be approximated with Laplace transformations or a Cholesky decomposition? We have found Laplace transformations to be slightly more efficient.

- **Type**: string

- **Possible Values**: LAPLACE, CHOLESKY
- **Default**: LAPLACE

DENOMINATOR_DELTA The SAPT module uses approximate energy denominators for most of the $E^{(20)}_{disp}$ and $E^{(20)}_{exch-disp}$ evaluation. This option controls the maximum allowable error norm in the energy denominator tensor.

- **Type**: double
- **Default**: 1.0×10^{-6}

DF_BASIS_ELST Optionally, a different fitting basis can be used for the $E^{(10)}_{elst}$ and $E^{(10)}_{exch}$ terms. This may be important if heavier elements are involved.

- **Type**: string
- **Default**: none

DF_BASIS_SAPT The fitting basis to use for all two-electron integrals in the SAPT computation. PSI4 will attempt to pick a reasonable fitting basis if one is not provided.

- **Type**: string
- **Default**: none

E_CONVERGENCE Convergence of the energy change in the $E^{(20)}_{ind,resp}$ term during the solution of the CPHF equations (in hartrees).

- **Type**: conv double
- **Default**: 1.0×10^{-10}

FREEZE_CORE Sets the number of core orbitals to freeze in the evaluation of the $E^{(20)}_{disp}$ and $E^{(20)}_{exch-disp}$ terms. It is recommended to freeze core in all SAPT computations.

- **Type**: string
- **Possible Values**: TRUE, FALSE, SMALL, LARGE
- **Default**: FALSE

INTS_TOLERANCE All three-index DF integrals and those contributing to four-index integrals that fall below this Schwarz bound will be neglected. The default is very conservative, however, there isn't much to gain from loosening it.

- **Type**: conv double
- **Default**: 1.0×10^{-12}

MAXITER The maximum number of CPHF iterations.

- **Type**: integer
- **Default**: 50

XXX xxxxx

- **Type**:
- **Possible Values**:
- **Default**:

XXX xxxxx

- **Type**:
- **Possible Values**:

- **Default**:

NO_RESPONSE Don't solve the CPHF equations, evaluate $E_{ind}^{(20)}$ and $E_{exch-ind}^{(20)}$ instead of their response-including counterparts. Only turn on this option if you are not going to use the induction energy.

- **Type**: boolean
- **Default**: FALSE

XXX xxxxx

- **Type**:
- **Possible Values**:
- **Default**:

PRINT The print level for the SAPT module. If set to 0, only the header and final results are printed. If set to 1, some intermediate quantities are also printed. For large SAPT computations, it is advisable to set to 1 so the progress of the computation can be tracked.

- **Type**: integer
- **Default**: 1

SAPT_LEVEL The level of theory for SAPT.

- **Type**: string
- **Possible Values**: SAPT0, SAPT2, SAPT2+, SAPT2+3
- **Default**: SAPT0

XXX xxxxx

- **Type**:
- **Possible Values**:
- **Default**:

SAPT_OS_SCALE The SAPT module will print a decomposition of the $E_{disp}^{(20)}$ and $E_{exch-disp}^{(20)}$ terms into same-spin and opposite-spin contributions, in analogy to the SCS-MP2 method of Stefan Grimme. This option controls the scaling of the opposite-spin contributions.

- **Type**: double
- **Default**: 6/5

SAPT_SS_SCALE This option controls the scaling of the same-spin contributions.

- **Type**: double
- **Default**: 1.0/3.0

24.2 DUMMY

DEBUG Print lots of intermediate quantities that are not usually interesting. For SAPT, it will also do additional work (which is not optimized for large systems) so don't turn it on.

- **Type**: integer
- **Default**: 0

XXX xxxxx

- **Type**:
- **Possible Values**:
- **Default**:

PYTHON MODULE INDEX

INDEX