optavc

Dr. Andreas V. Copan, Maddeline M Davis, Alexander G. Heide

May 25, 2021

CONTENTS:

CHAPTER

ONE

QUICKSTART

1.1 Background and Purpose

optavc was originally written by Dr. Copan to facilitate optimizations and hessian calculations by finite difference. It is primarily meant for internal use by the CCQC on our clusters and computing resources.

optavc is meant to run Psi4, Molpro, Cfour, and Orca QM calculations on the Sapelo and Vulcan clusters. These clusters have, over time, utilized the PBS/TORQUE, SLURM, and SGE queueing systems. Submission scripts for these queueing systems are autogenerated from program specific templates.

1.2 Dependencies

python >= 3.6

psi4 >= 1.3.2

1.2.1 loading psi4

optavc uses psi4's finite difference module to generate displacements. psi4 must be available in the environment as a python module to run optavc. This does not need to be the same version of psi4 that will be run on the cluster.

You can easily add psi4 to your environment with the *vulcan load psi4@master* or *module load PSI4/1.3.2_conda*

1.2.2 installing psi4

If you need to install psi4 and/or a new python: conda is the recommended installation process for dependencies. All CCQC members should familarize themselves with conda and git.

psi4.1.3.2:

```
conda create -n <new_env> psi4 psi4-rt -c psi4
```

psi4.1.4:

```
conda create -n <new_env> psi4 psi4-rt -c psi4/label/dev`
```

1.3 Installing optavc (for CCQC members)

There are no planned releases for optavc currently. For development purposes, please fork the gihub repository using the github website. For general usage feel free to just clone the master. Again all CCQC members need to familiarize themselves with git

clone the github repo to whatever directory you keep python libraries. For instance, in general I recommend:

```
mkdir ~/github
cd ~/github
git clone https://github.com/CCQC/optavc.git
```

Please do not install python libraries in *~/bin*. which is for binaries and executables not python modules. Please read https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard for more information on how the Linux FHS is organized.

add the following line to your *.bashrc* or the rc file for your shell of choice adjust to your shell as:

```
export PYTHONPATH=$PYTHONPATH:~/github/
```

1.4 Running Optavc

optavc is, fundamentally, a python module. Therefore there isn't really a input file for optavc. It is generally easiest; however, to write and then exectute a python script in your current working directory.

Basic Requirements:

> a python dict of options (see the options chapter for the bare required options).
>
> a template file. For the purposes of explanation optavc "only" RUNS this template file. The template file must be completely sufficient to run whatever singlepoint, gradient, or hessian calculation you are interested in running.

The easiest way to run optavc is as a background process on the login node of a cluster. This is not ideal, but optavc spends alot of time sleeping, and is not particularly resource intensive for the kinds of molecules we treat.

To run this way write a simple python script to call optavc like:

```
import optavc
options_kwargs = {
  'template_file_path': "template.dat",
  'energy_regex'      : r"\s*@DF-RHF Final Energy:\s+(-\d+\.\d+)",
  'program'           : "psi4",
  'maxiter'           : 20,
}

optavc.run_optavc("OPT", options_kwargs)
```

Then run this python script from cmd line as:

```
nohup python -u <name.py> &
```

nohup places all printing to stdout in nohup.out so all of optavcs output will be placed there. optavc makes calls to psi4 to perform the finite difference procedure, extrapolations, optimizations etc. All output produced from psi4 will be placed by default in output.dat

Directories like *HESS* and *STEP00* will be created in the current working direcotory which contain either input and output files or contain directories for each displacement which contain input and output files. Look there to check on how the calculations themselves are going.

`optavc.main.`**`run_optavc`**(*jobtype*, *options_dict*, *restart_iteration=0*, *xtpl_restart=None*, *sow=True*, *path='.'*, *test_input=False*, *molecule=None*)

Optavc driver to unify and simplify input. This is the only internal method of optavc a user should need to interact with. Creates any needed internal class objects based on the provided dictionary.

Performs an Optimization or Hessian calculation

Parameters

- **`jobtype`** (*`str`*) – upper and lowercase variants of HESS, FREQUENCY, FREQUENCIES, HESSIAN, OPT, and OPTIMIZATION are allowed
- **`options_dict`** (*`dict`*) – should contain BOTH optavc and psi4 options for optking and the finite difference proecedure. should not contain ANY options relevant for the calculations that optking will submit.
- **`restart_iteration`** (*`int`*) – For optimizations only. orresponds to the iteration we should begin trying to sow gradients for
- **`sow`** (*`bool`*) – For hessians only. if False optavc will reap the hessian.
- **`path`** (*`str`*) – for Hessians only. Specifies where to write displacement directories.
- **`test_input`** (*`bool`*) – Check all calculation in STEP00, HESS, or otherwise. This is a good way to test that the energy regex is working as expected. Raises an AssertionError if optavc considers any calculations to have failed.

Returns

- **result** (*list[int] or list[list[int]]*) – the gradient after optimization the hessian after a hessian calculation
- **energy** (*int*) – final energy or energy of non displaced point
- **molecule** (*molecule.Molecule*) – final molecule.

1.4.1 Warnings/Changelog - READ THIS!!

A few important changes and quirks that users should be notified about:

UNITS - defaults to Anstrom. Optavc does not pay attention to the units set in the template file make sure to set units if Bohr.

SUCCESS REGEXES - The corresponded keyword has been abandoned in favor of using the energy regex as the only indicator that a job has finished. This is because some programs don't have reliable, consistent success strings. If you're worried about your energy regex run a single gradient than *test_input=True* as seen in the documentation above for *run_optavc*

ENERGY REGEXES - In order to speedup optavc, optavc now run's by default in MULTILINE mode. ENERGY REGEXES must be matched then **from the begining of the line** this includes white space.

MEMORY - OPTAVC now only uses *#SBATCH –mem={memory}* for sapelo submit scripts. Please make sure to determine the total amount of memory your job uses if using a program such as molpro which specifies memory in a *per task* fashion.

1.4.2 Basic Functionality

Optavc can perform optimizations and hessian calculations through Psi4.

Optimizations can be performed with Analytic Gradients or by Finite Differences of Singlepoints

Hessians can be calculated Analytically or by Finite Differences of Gradients or Singlepoints.

Basis set extrapolation and additive corrections can be applied at the level of the Gradient and Hessian.

Aside from the actual calculation of the Energy, Gradient, or Hessian most functionality is obtained through Psi4 and QCDB.

Finite Difference Displacements are generated by psi4 and calculations for each displacements. Thermochemical and Vibrational analysis is also performed by Psi4. Optimization is performed by the Optking 2.0 module in Psi4.

CHAPTER

TWO

OPTIONS

class optavc.options.**Options**(***kwargs*)

The only *required* options for a *standard* run of optavc are energy_regex and program. All others are either not required for all jobs or have default values.

The required options for the extrapolation procedure are xtpl_templates, xtpl_regexes, and xtpl_basis_sets. energy_regex and template_file_path are NOT allowed to fill in xtpl_regexes and xtpl_templates. program is allowed fill in xtpl_programs.

The required options for performing an additive correction are delta_templates and delta_regexes. again energy_regex and template_file_path are not allowed to fill in delta_regexes and delta_templates

Notes

This class is also responsible for setting options in Psi4 that are required for optavc to utilize Psi4 properly. Common keywords are max_force_g_convergence, findif_points, hessian_write, cart_hess_read. etc

These are keywords related to the finite difference procedure and geometry optimization. NOT keywords related to running a specific calculation: basis_set, e_convergence, etc.

template_file_path

default : 'template.dat'

string, interpretable as a path, to a template file

Type string

backup_template

default : None

string, interpretable as a path to a secondary template file if the first has failed. This is only invoked when performing a restart. This is invoked either with *run_optavc("HESS", sow=False)* or *run_optavc("OPT", restart_iteration=<xx>)* For optimizations this only applies to the iteration we restart. It will not persist, so the template_file_path should also be updated to match backup_template. (Only backup_template will overwrite the currently written template files).

Type string or None

energy_regex

python 'raw' string. Should work with multiline mode. Should contain a group which returns the energy itself.

Type string

self.**correction_regexes**

default = ''

python 'raw' string. Should work with multiline mode. Should contain a group which returns the energy. This is a simple additive correction to the energy. For instance (T) correlation energy if CCSD(T) is needed. Not usually needed.

Type string

`self.`**`deriv_regex`**

default = ''

python 'raw' string. Should work with multiline mode. Uses a permissive regex to grab the actual values of a gradient. deriv_regex should be written to match the few lines before the gradient. i.e.

Total Molecular Gradient:

Type string

`self.`**`cluster`**

default = setter Allowed names for the cluster are vulcan, sapelo, and sapelo_old which are SGE, SLURM, and PBS/TORQUE clusters. if not provided otpavc will use socket.gethostname() to determine what cluster the user is on.

Type string

`self.`**`nslots`**

default 4

How many threads or tasks the job should be run with. Used to create the cluster submission script

Type int

`self.`**`threads`**

default 1

WARNING. Experimental. For programs that CLAIM to use mixed MPI and OpenMP parallelism set threads seperately from taks

Type int

`self.`**`scratch`**

default 'lscratch'

Choose between running on local scratch or a network filesystem. Vulcan has only lscratch. Sapelo has both; however, lscratch space is limited.

Type string

`self.`**`program`**

default ""

name of the program to calculations with orca, molpro, psi4, or cfour The name should at least be sufficiently long to load the module with. i.e. cfour@2.0+mpi is sufficient for vulcan load psi4 is also sufficient isntead of psi4@master.

Type string

`self.`**`parallel`**

default = setter

For programs like cfour the submission script generated and module loaded by optavc depends on the version. Recognized values are mpi, serial, and mixed. serial encompases openMP parallelism. Mixed uses both and is experimental.

the setter will attempt first to determine the value for parallel from the name of the program. i.e. cfour2.0+mpi otherwise programs defaults are: mpi for orca and molpro. serial for psi4 and cfour.

Type string

`self.`**`time_limit`**
default 10:00:00

time limit on vulcan is automatically 2 weeks and this overrides optavc's default.

Type string

`self.`**`queue`**
default gen4.q or batch

Type string

`self.`**`email`**
Vulcan and Sapelo both use myid. If requred email is automatically determined by $USER EMAIL is only setup for the Sapelo cluster

Type bool or None

`self.`**`email_opts`**
default = 'END,FAIL'

See documentation for the clusters queueing system for valid values. Optavc will not check. The default is only value for SLURM

Type string

`self.`**`memory`**
default '32GB'

How much total memory is needed for the Job. Include units. Only used for SAPELO

Type string

`self.`**`name`**
default STEP

This name may be altered at various levels but is the main prefix for how optavc names jobs. This impacts both directories in the current working directory and job names on the cluster.

Type string

`self.`**`files_to_copy`**
default []

Copy files from the current working directory to the directories for each displacement. Useful for files like GENBAS which can be pulled by default from the cfour install but may need to be overridden

Type list[string]

`self.`**`input_name`**
default 'input.dat'

what should the template file be named when it is written into a directory. When running cfour the input file will eventually be named ZMAT but this may occur in a tmp directory

`self.`**`output_name`**
default 'output.dat'

Type string

`self.`**`input_units`**
default "angstrom"

Make sure to set to bohr if the molecule in your template file. issues will occur and may not be easily recognizable

Type string

`self.`**`point_group`**

default None optavc will call psi4.reset_point_group(point_group) to make sure the symmetry is being conserved.

Type string

`self.`**`dertype`**

default ENERGY

allowed values are ENERGY or 0, GRADIENT or 1, HESSIAN or 2. Optimizations can be done with 0, or 1. Hessians can be done with 0, 1, or 2

Type string

`self.`**`deriv_file`**

default 'output'

What file should we look in to fetch the gradient or hessian. if 'output' the output file is serched with deriv_regex. Otherwise optavc attempts to read a gradient, or hessian file like cfour's GRD file. If program is cfour optavc will force use of the GRD file.

Type string

`self.`**`mpi`**

default None

WARNING Experimental. Legacy, unmaintained code for submitting singlepoints to slurm as a single mpi process

Type bool

`self.`**`job_array`**

defualt False

Should calculations be submitted in an array or as individual jobs to the cluster. array submission is only supported if cluster is Vulcan. job_array will be set to False for Sapelo.

Type bool

`self.`**`resub`**

default False

Prevents shut down of optavc if a calculation has failed by resubmitting to the cluster's queue. This only affects calculations that are run through the FiniteDifferenceCalc classes.

Type bool

`self.`**`resub_max`**

default None

How many times should a singlepoint be resubmitted. Some old molpro issues on specific Sapelo nodes required ths to be set to a relatively large integer. Be careful

Type int

`self.`**`sleepy_sleep_time`**

default 60

Determines the frequency with which optavc will wake up and check the status of its jobs.

Type int

`self.`**`xtpl`**
default None

is an extrapolation to be performed.

Type bool

`self.`**`maxiter`**
default 20

This is optavc's internal maxiter. Optkings geom_maxiter should not be obeyed since gradients are being set manually

`self.`**`xtpl_basis_sets`**
This is a required keyword for performing an extrapolation. Optavc will fail is this is set unnecesarily without the other required keywords.

list of lists of cardinal numbers for a basis set extrapolation. See composite gradient section for more details

The first list corresponds to extrapolation of a correlated method, the second list is for HF. example [[4, 3], [5, 4, 3]]

Type None or List[List[int]]

`self.`**`xtpl_templates`**
This is a required keyword for performing an extrapolation. Optavc will fail is this is set unnecesarily without the other required keywords.

list of lists of strings for templates for a basis set extrapolation. See composite gradient section for more details The first list corresponds to extrapolation of a correlated method, the second list is for HF. example [['template.dat', 'template.dat], ['template1.dat', 'template2.dat']]

Type None or List[List[str]]

`self.`**`xtpl_regexes`**
This is a required keyword for performing an extrapolation. Optavc will fail is this is set unnecesarily without the other required keywords.

list of lists of raw strings . See composite gradient section for more details The first list corresponds to extrapolation of a correlated method, the second list is for HF. example : [[r'mp2_qz', r'mp2_tz'] [r'hf_5z, r'hf_qz'']]

Type None or List[List[str]

`self.`**`programs`**
default setter

If no value is given. self.program will be applied to every calculation in the extrapolation procedure The first list corresponds to extrapolation of a correlated method, the second list is for HF.

Type None or List[List[str]]

`self.`**`xtpl_names`**
default setter

list of names for each calculation that will be performed. Duplicates are allowed. Default names are dependent on the number of calculations being performed and templates supplied.

Type None or List[List[str]]

`self.`**`xtpl_dertypes`**
default [[self.dertype, self.dertype], [self.dertype, 'self.dertype]]

list of strings or integers to determine the type of calculations being submitted. Energy, Gradient or Hessian. If no values are provided all calculations are assumed to match self.dertype which itself defaults to energy

Type None or List[List[str]]

`self.`**`xtpl_queues`**

default [[self.queue, self.queue], [self.queue, self.queue]]

If no values are provided, falls back to self.queue which defaults to gen4.q or batch depending on self.cluster

Type None or List[List[str]]

`self.`**`xtpl_nslots`**

default [[self.nslots, self.nslots], [self.nslots, self.nslots]]

see self.nslots for its default value

Type None or List[List[int]]

`self.`**`xtpl_memories`**

default [[self.memory, self.memory], [self.memory, self.memory]]

see self.memory for its default

Type None or List[List[str]]

`self.`**`xtpl_parallels`**

default [[self.parallel, self.parallel], [self.parallel, self.parallel]]

see self.parallel for defaults

Type None or List[List[str]]

`self.`**`xtpl_time_limits`**

default [[self.time_limit, self.time_limit], [self.time_limit, self.time_limit]]

see self.time_limit for defaults

Type None or List[List[str]]

`self.`**`xtpl_scratches`**

default [[self.scratch, self.scratch], [self.scratch, self.scratch]]

self.scratch for default

Type None or List[List[str]]

`self.`**`xtpl_deriv_regexes`**

default [[self.deriv_regex, self.deriv_regex], [self.deriv_regex, self.deriv_regex]]

see.deriv_regex for default

Type None or List[List[str]]

`self.`**`xtpl_deriv_files`**

default [[self.deriv_file, self.deriv_file], [self.deriv_file, self.deriv_file]]

Type None or List[List[str]]

`self.`**`scf_xtpl`**

default False

If false the scf results will not be extrapolatied. The extrapolated correlated result will be added to the scf result of the highest cardinality. Performing HF calculations with smaller basis sets may still be required i.e. with gradients.

Type bool

CHAPTER

THREE

COMPOSITE GRADIENTS AND HESSIANS

3.1 Composite Options

There are many combinations of xtpl and delta options possible so a more detailed explanation of the options for composite calculations are given here than in the Options area.

3.1.1 Option Format

xtpl options and delta options are provided as a list of lists. With xtpl_option[0] corresponding to the options for each calculation including post-HF correlation and xtpl_option[1] containing options for the Hartree Fock calculations. The options in these two lists are given in order of decreasing basis set size. i.e:

```
"xtpl_templates": [[mp2_qz_template, mp2_tz_template], [hf_qz_template, hf_tz_
↪template]],
"basis set size": [[4, 3], [4, 3]],
```

For delta options. There may be an arbitrary number of lists each of which should be two dimensional i.e:

```
"delta_templates": [[core-valence], [spin-orbit], [dboc]]
```

Internally, each xtpl and delta option will be expanded to its full length; however, options will be broadcast automatically if possible. Broadcasting can occur in several ways.

If no value is given for an xtpl or delta keyword, the corresponding *standard* option will be used for every calculation in the composite gradient. This may be user provided or can be the default value for the *standard* option. If a single value is given for a xtpl or delta keyword that single value will be used for every calculation. If a single value is given for each part of an extrapolation and correction the value will be broadcast across the list.

To demonstrate, the following inputs are all equivalent

input 1:

```
molpro_mp2_regex = r''
molpro_ccsd(t)_regex = r''
molpro_hf_regex = r''

options = {
    "xtpl_templates": [['mp2_qz.dat', 'mp2_tz.dat'], ['mp2_qz.dat', 'mp2_tz.dat']],
    "xtpl_regexes": [[molpro_mp2_regex], [molpro_hf_regex]],
    "xtpl_basis_sets": [[4, 3], [4, 3]],
    "xtpl_memories": "12GB",
    "delta_templates": [['AE-mp2.dat', 'FC-mp2.dat'], ['ccsd(t).dat', 'FC-mp2.dat']],
    "delta_regexes": [[molpro_mp2_regex], [molpro_ccsd(t), molpro_mp2_regex]],
```

(continues on next page)

(continued from previous page)

```
    "delta_nslots": [[8], [16, 4]],
    "delta_memories": [['16GB', '8GB'], ['32GB', '8GB']],
}

run_optavc("OPT", options)
```

input 2:

```
molpro_mp2_regex = r''
molpro_ccsd(t)_regex = r''
molpro_hf_regex = r''

options = {
    "xtpl_templates": [['mp2_qz.dat', 'mp2_tz.dat'], ['mp2_qz.dat', 'mp2_tz.dat']],
    "xtpl_regexes": [[molpro_mp2_regex, molpro_mp2_regex], [[molpro_hf_regex, molpro_
↪hf_regex]],
    "xtpl_basis_sets": [[4, 3], [4, 3]],
    "xtpl_nslots": [[4, 4], [4, 4]],
    "xtpl_memories": [['12GB', '12GB'], ['12GB', '12GB']],
    "delta_templates": [['AE-mp2.dat', 'FC-mp2.dat'], ['ccsd(t).dat', 'FC-mp2.dat']],
    "delta_regexes": [[molpro_mp2_regex, molpro_mp2_regex], [molpro_ccsd(t), molpro_
↪mp2_regex]],
    "delta_nslots": [[8, 8], [16, 4]],
    "delta_memories": [['16GB', '8GB'], ['32GB', '8GB']],
}

run_optavc("OPT", options)
```

In the above example. no value was provided for *xtpl_nslots* so the default *nslots* value was used to broadcast to the full form. For *xtpl_regexes* only one value was given for the correlation and scf portions. These were therefore broadcast across the sublist. For *delta_regexes* and *delta_nslots* the same broadcast occured, here the sublist defines a correction. For *xtpl_memories* a single value is given for the entire option so it is broadcast across all lists

xtpl_names and *delta_names* are unique in that they have a special and custom default. All other options fall back to the corresponding *standard* option if no value is provided.

3.1.2 Required Options

To run an extrapolated calculation *xtpl_regexes*, *xtpl_basis_sets*, and *xtpl_templates* are requied keywords. To run a composite calculation the above three keywords are required as well as *delta_regexes*, and *delta_templates*. All or None of these calculations must be set or an Error will be raised.

The behavior of the Delta and Xtpl classes is dictated almost entirely from these three keywords. All other keywords are for cluster interaction.

For a given sublist if only 1 value is given for *templates* or *regexes* the other MUST contain two or more values. Consider the previous example again:

```
molpro_mp2_regex = r''
molpro_ccsd(t)_regex = r''
molpro_hf_regex = r''


options = {
    "xtpl_templates": [['mp2_qz.dat', 'mp2_tz.dat'], ['mp2_qz.dat', 'mp2_tz.dat']],
    "xtpl_regexes": [[molpro_mp2_regex], [[molpro_hf_regex]],
```

(continues on next page)

(continued from previous page)

```
    "xtpl_basis_sets": [[4, 3], [4, 3]],
    "xtpl_memories": "12GB",
    "delta_templates": [['AE-mp2.dat', 'FC-mp2.dat'], ['ccsd(t).dat', 'FC-mp2.dat']],
    "delta_regexes": [[molpro_mp2_regex], [molpro_ccsd(t), molpro_mp2_regex]],
    "delta_nslots": [[8], [16, 4]],
    "delta_memories": [['16GB', '8GB'], ['32GB', '8GB']],
}

run_optavc("OPT", options)
```

This calculation is run using the default dertype - “ENERGY”. The user should know that molpro will print the ccsd(t) mp2 and hartree fock energies in the same output file in the course of running ccsd(t). This means only a handful of jobs need to be run 2 for the extrapolation and 3 additional jobs for the corrections instead of 8 total.

The length of *<option>_templates* and *<option>_regexes* will be (in general) inversely proportional. Optavc expects this even if the full specification is given as in input 2 only a certain number of unique templates and regexes are expected. This can be overspecified and optavc will run more jobs than necessary but optavc will quit if not enough are provided. For the extrapolation portion in the example above two unique calculations are performed based on the unique entires in *xtpl_templates*. mp2_qz.dat is run once. mp2_tz.dat is run once. *molpro_mp2_regex* and *molpro_hf_regex* are used to get energies from both output files. Similar behavior occurs for *delta_templates* and *delta_regexes* The lengths of all required options are compared as a sanity check. For a given sublist if only 1 value is given for *templates* or *regexes* the other MUST contain two or more values.

CHAPTER

FOUR

EXAMPLES

This will be the sole area for optavc examples

4.1 Optimizations

4.1.1 Basic

A bare bones input file:

```
import optavc

options = {
    # These two keywords would be sufficient to make OPTAVC run
    'program': 'cfour@2.0+mpi',
    'energy_regex': r"\s*CCSD\(T\)\senergy\s*(-?\d*\.\d*)",

    # required to modify psi4
    'max_force_g_convergence': 1e-7,
    'findif_points': 5,

}

optavc.run_optavc('OPT', options, restart_iteration=0)
```

4.1.2 Bad Input

An input file with uneeded input that few clusters can support well:

```
import optavc

options = {
    # These two keywords would be sufficient to make OPTAVC run
    'program': 'cfour@2.0+mpi',
    'energy_regex': r"\s*CCSD\(T\)\senergy\s*(-?\d*\.\d*)",

    # required to modify psi4
    'max_force_g_convergence': 1e-7,
    'findif_points': 5,

    # These are overiding defaults
```

(continues on next page)

(continued from previous page)

```
    'queue': 'batch_30d',
    'nslots': 8,
    'memory': '92GB',
    'cluster': 'Sapelo'
}

optavc.run_optavc('OPT', options, restart_iteration=0)
```

First there's no need to use the cluster keyword. optavc will autodetect the cluster. Second If you're gonna be doing finite differences, the cluster will never be able to run so many jobs with these memory requirements. Computer hours matters just as much as Human hours. Analytic Gradients offer better convergence behavior anyway - see every geometry optimization paper from the 90s.

4.1.3 Gradient

optimization using cfour analytic CCSD(T) gradients:

```
import optavc

cfour_grad_regex = r"\s*Molecular\s*gradient\s*-+\s*"
# "([A-Z]+\s#[0-9]+\s[xyz]\s*-?\d+\.\d+\s*)+"

options = {
    'program': 'cfour@2.0+mpi',
    'template_file_path': 'template.dat',
    'energy_regex': r"\s*CCSD\(T\)\senergy\s*(-?\d*\.\d*)",
    'deriv_regex': cfour_grad_regex,
    'dertype': 'gradient',
    'queue': 'gen6.q',
    'max_force_g_convergence': 1e-7
}

optavc.run_optavc('OPT', options, restart_iteration=0)
```

4.1.4 Transition State

An example of a transition state optimization followed by frequencies analysis to verify the stationary point character:

```
import optavc
import os

os.system(f'cp output.default.hess output.default.{os.getpid().hess}')

cfour_grad_regex = r"\s*Molecular\s*gradient\s*-+\s*"
c4_hess = r"\s*Ex\s*Ey\s*Ez"


options = {
    'program': 'cfour@2.0+mpi',
    'energy_regex': r"\s*CCSD\(T\)\senergy\s*(-?\d*\.\d*)",
    'deriv_regex': cfour_grad_regex,
    'dertype': 'gradient',
    'queue': 'gen6.q',
```

(continues on next page)

(continued from previous page)

```
    'max_force_g_convergence': 1e-7,
    'cart_hess_read': True,
    'opt_type': 'TS'
}

gradient, energy, molecule = optavc.run_optavc('OPT', options, restart_iteration=0)

options.update({'deriv_regex': c4_hess,
                'template_file_path': 'template2.dat',
                'dertype': 'hessian',
                'hessian_write': True})

optavc.run_optavc('HESS', options, molecule=molecule)
```

There are some optking options left in the input, psi4 will set these options, but no calls will be made to optking so there's no need to remove them.

4.1.5 Extrapolation

The following example demonstrates a simple two point extrapolation of gradients via singlepoints:

```
import optavc

molpro_ccsdt_regex = r''
molpro_scf_regex = r''

options_kwargs = {
    "program" : "molpro",
    "xtpl_basis_sets" : [[4, 3], [4, 3]],
    "xtpl_energy_regexes" : [[molpro_ccsdt_regex], [[molpro_scf_regex]],
    "xtpl_templates" : [[molpro_qz.dat, molpro_tz.dat], [molpro_qz.dat, molpro_tz.
↪dat]]
}

optavc.run_optavc("OPT", options_kwargs)
```

All other options will resort to default values as described elsewhere.

4.1.6 Composite

A *real* example of a two point mp2 exatrapolation using analytic gradients with a ccsd(t) correction at a dz basis set where some keywords are expanded more than necessary and some are left to be broadcast:

```
import os
import optavc

energy_regex = r"\s*\s\sTotal\sEnergy\s*=\s*(-\d*.\d*)"
mp2_reg = r"\s*DF-MP2\sTotal\sEnergy\s\(a\.u\.\)\s*:\s*(-\d*.\d*)"
psi4_grad = r"\s*-Total\s*Gradient:\n\s*Atom[XYZ\s]*[-\s]*" # This is just the header␣
↪i.e.
ccsdt = r"\s*CCSD\(T\)\senergy\s*(-?\d*\.\d*)"
c4_grad = r"\s*Molecular\s*gradient\s*-+\s*"
c4_mp2 = r"\s*The\sfinal\selectronic\senergy\sis\s*(-\d*.\d*)"
```

(continues on next page)

(continued from previous page)

```
options_kwargs = {
    'program'                  : "psi4@master",
    'maxiter'                  : 100,
    'files_to_copy'            : ['GENBAS'],
    'deriv_regex'              : psi4_grad,
    'nslots'                   : 4,
    'max_force_g_convergence'  : 1e-7,
    'ensure_bt_convergence'    : True,
    'xtpl_templates'           : [["mp2_qz.dat", "mp2_tz.dat"], ["scf_qz.dat", "scf_tz.
↪dat"]],
    'xtpl_names'               : [['PR2a_mp2qz', 'PR2a_mp2tz'], ['PR2a_scfqz', "PR2a_
↪scftz"]],
    'xtpl_regexes'             : [[mp2_reg], [energy_regex]],
    'xtpl_dertypes'            : [['gradient'], ['gradient']],
    'xtpl_queues'              : [['gen4.q', 'gen3.q'], ['gen3.q']],
    'xtpl_memories'            : [['30GB', '16GB'], ['16GB', '16GB']],
    'xtpl_basis_sets'          : [[4, 3], [4, 3]],
    'delta_templates'          : [["ccsdpT.dat", "mp2_dz.dat"]],
    'delta_regexes'            : [[ccsdt, c4_mp2]],
    'delta_programs'           : [["cfour@2.0+mpi"]],
    'delta_names'              : [["PR2a_CC", "PR2a_mp2dz"]],
    'delta_deriv_regexes'      : [[c4_grad, c4_grad]],
    'delta_dertypes'           : [['gradient', 'gradient']],
    'delta_parallels'          : [['mpi', 'serial']],
    'delta_memories'           : [['60GB', '30GB']],
    'delta_queues'             : [['gen6.q', 'gen4.q']],
}

gradient, energy, molecule = optavc.run_optavc('opt', options_kwargs, restart_
↪iteration=0)
```

4.2 Hessians

4.2.1 Basic

Bare bones hessian calculation:

```
import optavc

options = {
    # These two keywords would be sufficient to make OPTAVC run
    'program': 'cfour@2.0+mpi',
    'energy_regex': r"\s*CCSD\(T\)\senergy\s*(-?\d*\.\d*)",
    'findif_points': 5,
    'hessian_write': True
}

optavc.run_optavc('HESSIAN', options, sow=True)
```

4.2.2 Composite

This is an example of a compsite hessian calculation using analytic hessians from cfour and analytic gradients in psi4:

```
import os
import optavc

energy_regex = r"\s*\s\sTotal\sEnergy\s*=\s*(-\d*.\d*)"
mp2_reg = r"\s*DF-MP2\sTotal\sEnergy\s\(a\.u\.\)\s*:\s*(-\d*.\d*)"
psi4_grad = r"\s*-Total\s*Gradient:\n\s*Atom[XYZ\s]*[-\s]*" # This is just the header␣
↪i.e.
ccsdt = r"\s*CCSD\(T\)\senergy\s*(-?\d*\.\d*)"
c4_mp2 = r"\s*The\sfinal\selectronic\senergy\sis\s*(-\d*.\d*)"
c4_hess = r"\s*Ex\s*Ey\s*Ez"

options_kwargs = {
    'program'                   : "psi4@master",
    'maxiter'                   : 100,
    'files_to_copy'             : ['GENBAS'],
    'deriv_regex'               : psi4_grad,
    'nslots'                    : 4,
    'max_force_g_convergence'   : 1e-7,
    'ensure_bt_convergence'     : True,
    'xtpl_templates'            : [["mp2_qz.dat", "mp2_tz.dat"], ["scf_qz.dat", "scf_tz.
↪dat"]],
    'xtpl_names'                : [['PR2a_mp2qz', 'PR2a_mp2tz'], ['PR2a_scfqz', "PR2a_
↪scftz"]],
    'xtpl_regexes'              : [[mp2_reg], [energy_regex]],
    'xtpl_dertypes'             : [['gradient'], ['gradient']],
    'xtpl_queues'               : [['gen4.q', 'gen3.q'], ['gen3.q']],
    'xtpl_memories'             : [['30GB', '16GB'], ['16GB', '16GB']],
    'xtpl_basis_sets'           : [[4, 3], [4, 3]],
    'delta_templates'           : [["ccsdpT.dat", "mp2_dz.dat"]],
    'delta_regexes'             : [[ccsdt, c4_mp2]],
    'delta_programs'            : [["cfour@2.0+mpi"]],
    'delta_names'               : [["PR2a_CC", "PR2a_mp2dz"]],
    'delta_deriv_regexes'       : [[c4_hess]],
    'delta_dertypes'            : [['hessian']],
    'delta_parallels'           : [['mpi', 'serial']],
    'delta_memories'            : [['60GB', '30GB']],
    'delta_queues'              : [['gen6.q', 'gen4.q']],
    'hessian_write'             : True
}

gradient, energy, molecule = optavc.run_optavc('FREQUENCIES', options_kwargs,␣
↪sow=True)
```

4.2.3 Hess-Opt-Hess

Final example:

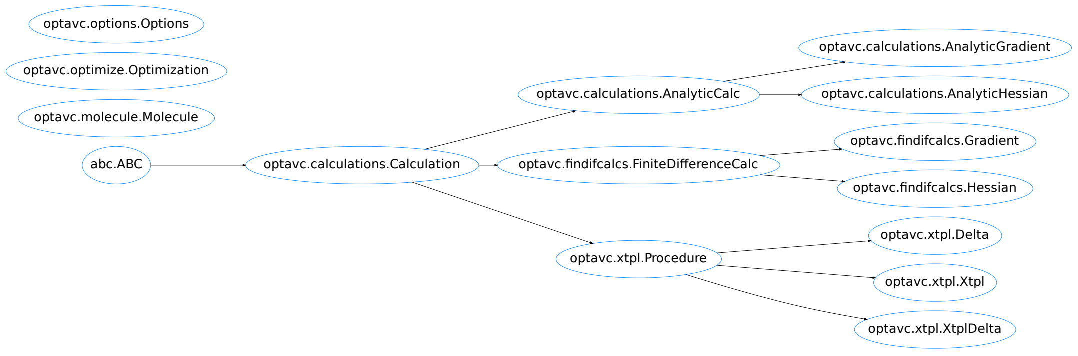
```
import optavc
import os

cfour_grad_regex = r"\s*Molecular\s*gradient\s*-+\s*"
c4_hess = r"\s*Ex\s*Ey\s*Ez"


options = {
    'program': 'cfour@2.0+mpi',
    'energy_regex': r"\s*CCSD\(T\)\senergy\s*(-?\d*\.\d*)",
    'deriv_regex': cfour_hess,
    'dertype': 'hessian',
    'queue': 'gen6.q',
    'max_force_g_convergence': 1e-7,
    'memory': '64GB',
    'hessian_write': True
}

hessian, energy, molecule = optavc.run_optavc('HESS', options)

options.update({'deriv_regex': c4_grad_regex,
                'template_file_path': 'template2.dat',
                'dertype': 'gradient',
                'cart_hess_red': True})   # Single PID for entire run no need to copy␣
↪hessian

optavc.run_optavc('opt', options, molecule=molecule)

options.update({"deriv_regex": cfour_grad_regex,
                "dertype": 'hessian',
                "template_file_path": 'template.dat'})

hessian, energy, molecule = optavc.run_optavc('HESS', options)
```

CHAPTER

FIVE

HOW DOES OPTAVC WORK?

5.1 Class Structure


optavc.options.Options
optavc.optimize.Optimization
optavc.molecule.Molecule
abc.ABC
optavc.calculations.Calculation
optavc.calculations.AnalyticCalc
optavc.calculations.AnalyticGradient
optavc.calculations.AnalyticHessian
optavc.findifcalcs.FiniteDifferenceCalc
optavc.findifcalcs.Gradient
optavc.findifcalcs.Hessian
optavc.xtpl.Procedure
optavc.xtpl.Delta
optavc.xtpl.Xtpl
optavc.xtpl.XtplDelta

5.2 Calculations

class optavc.calculations.**AnalyticCalc**(*molecule*, *inp_file_obj*, *options*, *path='.'*, *key=None*)

Parent class for the three real types of calculations that can be run. All other child classes will have one or more instances of AnalyticCalc or instances of classes that have instances of AnalyticCalc.

This class contains the necessary code to perform the actual execution of Gradients, Singlepoints, and Hessians. For result collection please see AnalyticGradient and Singlepoint child classes

Still cannot be instantiated: the abstract method get_result is not implemented here.

self.**cluster**

a class that manages how a Calculation is able to interact with the Cluster queueing a submission system

Type cluster.Cluster

self.**resub_count**

keeps track of whether we have exceed Options.resub_max for each Calculation

Type int

run()

This is the base implementation of run. calls cluster.submit or tries to execute the program on the host in a somewhat standard fashion (not guarranteed to work). The working directory must be changed to where the input has been written (the path of the Calculation) and then returns to the directory of the Parent Calculation. Returns the job_id (or zero) if no cluster id Parent implemenations generally just call this method for each of their Calculation objects.

write_input()

This is the base implementation of write_input. Updates the input file object with this calculations molecule information, then writes it to the location given to this calculation by its Parent Calculation. Copy any other files needed. Parent implemenations generally just call this method for each of their Calculation objects.

wait_for_calculation()

uses Options.sleepy_sleep_time to hold until this specific calculation is finished

check_status(*status_str*, *return_text=False*)

Check for status_str within a output_file. This is how optavc finds failed jobs. This information is necessary but not sufficient to resubmit a job on sapelo (job_state must also be confirmed via queuing system)

Parameters **status_str** (*str*) – generally options.success_regex or options.fail_regex

Returns

- **bool** (*True if the string was found.*) – Read if check_status(status_string): statements with care
- **output_text** (*str*) – if requested

Raises **FileNotFoundError** –

Notes

method should called through get_energy_from_output in standard workflow

compute_result()

Wrapper method to run a calculation from scratch. Write inputs. Run calculations. Collect results for any and all Calculations AnalyticCalc and FindifCalc reimplement this with better ways to wait. The Procedure class does simply inherit this. :meta private:

class optavc.calculations.**AnalyticGradient**(*molecule*, *inp_file_obj*, *options*, *disp_num=None*, *path='.'*, *key=None*)

This class was implemented in order to use CFour's analytic gradients with the psi4 CBS procedure. CCSD(T) gradients from CFour are not available through the Psi4/CFour interface.

Uses psi4.qcdb to rotate / align the gradient and molecule back into optavc's molecular orientation if using cfour gradients.

get_result()

This is the first class in the hierarchy that is really meant to just be run without management by a Parent class. Therefore if the calculation is not finished, optavc will check the queue for this job at intervals. returns a numpy array

get_reference_energy()

All classes at a higher level than this in the hierarchy will now need the energy seperate from the result

str_to_ndarray(*grad_output*)

Take string with possible header from the output file or a specified gradient file and convert to psi4 matrix

Notes

ignores any header grabs the last three columns of the last 'natom' lines and converts to a psi4 matrix via numpy array

class `optavc.calculations.`**`AnalyticHessian`**(*molecule*, *inp_file_obj*, *options*, *disp_num=None*, *path='.'*, *key=None*)

This class was implemented in order to use CFour's analytic hessians with the psi4 CBS

This class performs the same functionality as AnalyticGradient. See docs.

`get_reference_energy`()

Get reference energy from gradient calculation

Returns

Return type float

`hessian_file_lookup`()

Try to get the hesian from a special written file. Assuming a naive format.

Use psi4 qcdb machinery to get a hessian from cfour and ensure proper orientation

`rotate_hessian`(*hessian*)

only needed for cfour. Same procedure as in psi4. Use grad file to rotate the internal molecular orientaiton back to the psi4 (user) orientation. Then rotate the hessian

`str_to_ndarray`(*grad_output*)

Take string with possible header from the output file or a specified gradient file and convert to psi4 matrix

Notes

ignores any header grabs the last three columns of the last 'natom' lines and converts to a psi4 matrix via numpy array

class `optavc.calculations.`**`Calculation`**(*molecule*, *options*, *path='.'*, *key=None*)

This is the Base class for everything calculation related in optavc. Its purpose is really to define the API for the write_input, run, get_result, and compute_result methods.

The basic Hierarchy of Calculation is:

Optimizations can contain one or more Procedure, FiniteDifferenceCalc, or AnalyticCalc (only 1 at a time)

Procedures can contain one or more FiniteDifferenceCalc or AnalyticCalc.

FiniteDifferenceCalc can itself contain one or more AnalyticCalc (always of the same kind)

In this hierarchy any Calculation that consists of other Calculations lower in the hierarchy is able to run all Calculations in parallel through the write_inputs, run, and get_result methods. Calling compute_result repetively will result in non parallel executation of the sub calculations.

molecule [molecule.Molecule] optavc's custom very basic molecule class that can be interconverted to psi4's molecule

options [opions.Object] At every level of the Calculation Hierarchy this will become a copy of the higher levels options with (potentially) modifications made by the higher level Calculation.

path [str] path-like representation of the directory where the Calculation itself will be run changes throughout the Calculation hierarchy

write_input() write ALL input files for the Calculation class.

run() run ALL input files contained in the Calculation class. A simple submission of all jobs (at once) to the cluster.

get_result() get ALL results for the Calculation class.

self.reap compute_result() calls write_input, run, and get_result all together. This requires a waiting period between running the calculations and getting the results of course.

`class_name``()`
return AnalyticGradient from <class 'optavc.calculations.AnalyticGradient'>

`compute_result``()`
Wrapper method to run a calculation from scratch. Write inputs. Run calculations. Collect results for any and all Calculations AnalyticCalc and FindifCalc reimplement this with better ways to wait. The Procedure class does simply inherit this. :meta private:

`to_dict``()`
Not generally used. Serialize all object attributes and add in subset of keywords

`class` `optavc.calculations.`**`SinglePoint`**`(`*molecule*, *inp_file_obj*, *options*, *path='.'*, *disp_num=1*, *key=None*`)`
Handles lookup of the result for a Singlepoint. Extends AnalyticCalculation by adding the get_result behavior for a Singlepoint.

`self.`**`disp_num`**

Type int

`get_result``()`
get the energy (with a correction if needed) from an output file.

5.3 Finite Difference

`class` `optavc.findifcalcs.`**`FiniteDifferenceCalc`**`(`*molecule*, *inp_file_obj*, *options*, *path='.'*`)`
A Calculation consisting of a series of AnalyticCalc objects with needed machinery to submit, collect and assemble these calculations.

One of two basic child classes of Calculation that holds a list of Calculations. This class is interfaced with Psi4's finite difference machinary in order to create calculations for each needed displacement.

Most of the complexity of the Finite Difference Algorithms exists within this class due to the difficulty encountered in resubmitting jobs on the Sapelo cluster.

Provides basic functionality for the Gradient and Hessian child classes.

`calculations`
all required singlepoints or gradients

Type List[AnalyticCalc]

`failed`
singlepoint or gradient for which the result could not be found. calculations are added and removed with each collect_failures call

Type List[AnalyticCalc]

`job_ids`
collection of job IDs from the cluster. job IDs are added whenever run is called. Removed whenever resub is called. Resub will replace an ID if the singlepoint is rerun.

Type List[str]

check_resub_count()
Update self.failed to remvoe any calculations which have already been submitted the maximum number of times.

collect_failures(*raise_error=False*)
Collect all jobs which did not successfully exit in self.failed.

Returns **bool**

Return type False if unable to find any failures

compute_result()
Wrapper method to run a calculation from scratch. Write inputs. Run calculations. Collect results for any and all Calculations AnalyticCalc and FindifCalc reimplement this with better ways to wait. The Procedure class does simply inherit this. :meta private:

get_reference_energy()
Get result stored in the finite difference dictionary

Returns

Return type float

make_calculations()
Use psi4's finite difference machinery to create dispalcements and singlepoint or gradient calculations

reap(*force_resub=False*)
Collect results for all calculations. Resub if necessary and if allowed This method assumes that we have already told optavc to sleep after submitting all of our jobs. This is the required downtime between submission and checking the cluster, which can result in error. Not the down time for intermitently checking whether the job has finished. If called and jobs have not finished. Keep waiting

resub(*force_resub=False*)
Rerun each calculation in self.failed as an individual job.

run_individual()
Run analytic calculations for finite difference procedure

Returns **list[str]** – job ids (from AnalyticCalc.run) for all job ids that were just run.

Return type List[str]

class optavc.findifcalcs.**Gradient**(*molecule*, *input_obj*, *options*, *path*)
Machinary to create inputs and collect results for running a gradient using Singlepoints.

As noted else where, as one of the Parent Classes, Gradient uses the write_input, run, and get_result methods for the Child class to perform the Singlepoint calculations in parallel.

Compatible with the Finite Difference machinary for Psi4 < 1.3.2 and > 1.4

build_findif_dict()
Gradient can only take energies

findif_methods()
Can only do finite differences by energies.

Returns

- **create** (*func*) – function to create the displacements for a gradient
- **compute** (*func*) – function to collect singlepoints and calculate a gradient
- **constructor** (*func*) – constructor for creating the correct AnalyticCalc (Singlepoint)

reap (*force_resub=False*)
Collect results for all calculations. Resub if necessary and if allowed This method assumes that we have already told optavc to sleep after submitting all of our jobs. This is the required downtime between submission and checking the cluster, which can result in error. Not the down time for intermitently checking whether the job has finished. If called and jobs have not finished. Keep waiting

class `optavc.findifcalcs.`**Hessian** (*molecule*, *input_obj*, *options*, *path*)
Machinery to create inputs and collect results for a Hessian for any type of AnalyticCalc.

findif_methods ()
Use options obejct to determine how finite differences will be performed and what AnalyticCalc objects need to be made

get_reference_energy ()
Get result stored in the finite difference dictionary

Returns

Return type float

reap (*force_resub=False*)
Collect results for all calculations. Resub if necessary and if allowed This method assumes that we have already told optavc to sleep after submitting all of our jobs. This is the required downtime between submission and checking the cluster, which can result in error. Not the down time for intermitently checking whether the job has finished. If called and jobs have not finished. Keep waiting

5.4 Composite Procedures

class `optavc.xtpl.`**Delta** (*job_type*, *molecule*, *procedure_options*, *path='./HESS'*, *iteration=0*)
Child class of Procedure to represent performing an arbitrary number of additive corrections to gradients and hessians.

This class is not really meant to be a called without Xtpl. Please use XtplDelta.

`self.`**delta_options_list**
All of the delta_<option> options in the standard format of a list of two dimensional corrections. These options will be used to create a series of Options objects by setting the corresponding standard options with the delta options entries.

Assumes the Options object contins only properly formatted options.

Type List

`self.`**return_result** ()
Takes a simple difference between the first item in a two dimensional correction and the second. All corrections are then summed together and the sum is taken as the result

class `optavc.xtpl.`**Procedure** (*job_type*, *molecule*, *procedure_options*, *path='./HESS'*, *iteration=0*)
A Procedure may be thought of as a list of Calculations with a matching list of instructions 'SOW' and 'REAP' to enable calculating a series of unique Calculations.

Creates AnalyticGradient, AnalyticHessian, Gradient (Finite Difference), and Hessian (Finite Difference) Calculation objects and runs them in parallel run.

This is a really just a framework for the Xtpl and Delta classes below. Would need to be generalized to implement new Procedures.

get_reference_energy ()
Called reference to match findif method name reference in terms of displacements

class optavc.xtpl.**Xtpl**(*job_type*, *molecule*, *procedure_options*, *path='./HESS'*, *iteration=0*)

A child class of Procedure.

Represents a basis set extrapolation of gradients or hessians. This creates the Calculation objects needed to perform each individual calculation.

self.**xtpl_option_list**

all xtpl_<option> options. These will be used to create more specific Options objects for the individual calculations

Type list

get_result(*force_resub=False*)

collects all gradients or hessians in self.calculations and performs the basis set extrapolation as described by Options.xtpl_basis_sets and Options.xtpl_scf. Always performs a two point correlation energy extrapolation and can be either 2 point, 3 point or largest (additive) basis set extrapolation.

get_reference_energy()

Called reference to match findif method name reference in terms of displacements

class optavc.xtpl.**XtplDelta**(*job_type*, *molecule*, *procedure_options*, *path='./HESS'*, *iteration=0*)

A Procedure written to run the Xtpl and Delta Procedures in parallel. Creates the Xtpl and Delta

Seqentially calls write_input, run, and get_result for the Xtpl and Delta procedures. Adds the results from Xtpl and Delta together.

#TODO this should be rewritten to utlilize the _reap_sow_ordering and unique_calculations methods in the base class to eliminiate redundant calculations between the Xtpl and Delta classes

5.5 Optimization

class optavc.optimize.**Optimization**(*molecule*, *input_obj*, *options*, *xtpl_inputs=None*)

Run AnalyticGradient and Gradient calculations or any procedures which contain and only contain those object types

copy_old_steps(*restart_iteration*, *xtpl_restart*)

If a directory is going to be overwritten by a restart copy the directories to backup and prevent steps > restart_iteration are not immediately submitted.

create_opt_gradient(*iteration*)

Create Gradient with path and name updated by iteration

Parameters **iteration** (*int*) –

Returns **grad_obj**

Return type *findifcalcs.Gradient*

enforce_unique_paths(*grad_obj*)

Ensure that for an optimization no two gradients can be run in the same directory. Keep a list of paths used in a optimization up to date.

Method should be called whenever a new Gradient in created and before it is run

Parameters **grad_obj** (Gradient) – The newly created Gradient object

Notes

This is a warning for future development: If the gradient's path is not updated with each step, collect_failures() will allow new steps to be taken in the same directory. When optavc checks the relevant output files, it will find them to be completed and run another calculation this will dump new jobs onto the cluster each time.

Raises **`ValueError`** – This should not be caught. Indicates that changes:

`run_gradient`(*iteration*, *restart_iteration*, *grad_obj*, *force_resub=True*)

run a single gradient for an optimization. Reaping if iteration, restart_iteration, and xtpl_reap indicate this is possible

Parameters

- **`iteration`** (*`int`*) –
- **`restart_iteration`** (*`int`*) – indicates (with iteration) whether to sow/run or just reap previous energies
- **`grad_obj`** (`findifcalcs.Gradient`) –
- **`force_resub`** (*`bool, optional`*) – If the FIRST reap fails, resubmit all jobs to cluster for any gradient that "should" have already been completed based on iteration, restart_iteration, and xtpl_reap

Returns grad

Return type psi4.core.Matrix

Notes

Reassemble gradients as possible. Always restart based on iteration number If the we're rechecking the same input file in xtpl_procedure, xtpl_reap must be set

PYTHON MODULE INDEX

INDEX