Forte

Release 0.2.3

Forte Developers

Jan 16, 2023

CONTENTS:

Forte is an open-source suite of quantum chemistry methods for strongly correlated electrons.

CHAPTER

ONE

OVERVIEW

Forte is an open-source suite of state-of-the-art quantum chemistry methods applicable to chemical systems with strongly correlated electrons. The code is written as a plugin to Psi4 in C++ with C++17 functionality, and it takes advantage of shared memory parallelism throughout.

1.1 Capabilities

In general, Forte is composed of two types of methods: 1. Active space solvers 1. Dynamical correlation solvers

Active space solvers	Abbreviation
Full/complete active space configuration interaction	FCI/CASCI
Adaptive configuration interaction	ACI
Projector configuration interaction	PCI

Dynamical correlation solvers	Abbreviation
Driven similarity renormalization group	DSRG
Second-order DSRG multireference perturbation theory	DSRG-MRPT2
Third-order DSRG multireference perturbation theory	DSRG-MRPT3
Multireference DSRG with singles and doubles	MR-LDSRG(2)

Note that the active space solvers, notably FCI, ACI, and PCI can operate within the full orbital basis defined by the user-selected basis set. In this case, these methods also recover dynamical correlation.

1.2 Dependencies

In order to run Forte, the following are required: - A Recent version of Psi4 - CMake version 3.0 or higher - The tensor library Ambit

CHAPTER

TWO

TUTORIALS

2.1 Compiling and running Forte

2.1.1 Download and compilation of Forte

Prior to the compilation of Forte you must first check to make sure you have the following:

1. CMake version 3.0 or higher
2. An updated version of Psi4 (obtain it from https://github.com/psi4/psi4)
3. The tensor library Ambit (obtain it from https://github.com/jturney/ambit). Note that ambit is included in the conda distribution of psi4. So if you already have the latest version of psi4 installed there is no need to compile ambit.

Once you have the current versions of Psi4, CMake, and Ambit, follow the following instructions to install Forte.

Download Forte

1. Open a terminal and change the current working directory to the location where you want to clone the Forte directory. Let's assume this is the folder `src`.
2. Clone Forte from GitHub by pasting the following command:

```
git clone git@github.com:evangelistalab/forte.git
```

The repository will be cloned in the folder `src/forte`

1. Compilation via `setup.py` (recommended)

The most convenient way to compile forte is using the `setup.py` script. To compile Forte do the following:

1. From the `src` directory change to the forte directory `src/forte`
2. Tell `setup.py` where to find ambit, which can be done by creating the `src/forte/setup.cfg` file and adding the following lines

```
[CMakeBuild]
ambitpath=<ambit install dir>
```

or alternatively by setting the environmental variable `AMBITDIR` to point to the ambit install directory (note: there is no need to append `share/cmake/ambit`)

```
export AMBITPATH=<ambit install dir>
```

3. Compile forte by calling

```
python setup.py develop
```

or for Debug mode

```
python setup.py build_ext --debug develop
```

This procedure will register forte within pip and you should be able to see forte listed just by calling

```
pip list
```

You can test that the path to Forte is set correctly by running python and importing forte:

```
import forte
```

2. Compilation via CMake

Forte may also be compiled by directly invoking CMake by following these instructions:

1. Run psi4 in the Forte folder

```
psi4 --plugin-compile
```

Psi4 will generate a CMake command for building Forte that looks like: `cmake -C /usr/local/psi4/stage/usr/local/psi4/share/cmake/psi4/psi4PluginCache.cmake -DCMAKE_PREFIX_PATH=/usr/local/psi4/stage/usr/local/psi4 .`

2. Run the cmake command generated in 1. appending the location of Ambit's cmake files (via the `-Dambit_DIR` option):

```
cmake -C /usr/local/psi4/stage/usr/local/psi4/share/cmake/psi4/psi4PluginCache.cmake
      -DCMAKE_PREFIX_PATH=/usr/local/psi4/stage/usr/local/psi4 .
      -Dambit_DIR=<ambit-bin-dir>/share/cmake/ambit
```

3. Run make

```
make
```

Setting up the PYTHONPATH

If Forte is compiled with CMake, you will need to specify `PYTHONPATH` environment variable to make sure that it can be imported in python. Assuming that you cloned Forte from the folder `src` then you will have a folder named `src/forte`. Your `PYTHONPATH` should then include `src/forte`

```
# in bash
export PYTHONPATH=<homedir>/src/forte:$PYTHONPATH
```

This allows Forte to be imported correctly since the main `__init__.py` file for Forte is found at `src/forte/forte/__init__.py`

CMake script

The following script automates the Forte compilation process

```
#! /bin/tcsh

# Modify the following four parameters
set ambit_dir = /Users/fevange/Bin/ambit-Release/share/cmake/ambit/ # <- location of␣
↪ambit
set srcdir = /Users/fevange/Source/forte   # <- location of forte source
set build_type = Release # <- Release, Release, or RelWithDebInfo

# Run cmake
cd $srcdir

set cmake_psi4 = `psi4 --plugin-compile`

$cmake_psi4 \
-Dambit_DIR=$ambit_dir \ # remove this line if ambit is installed via conda
-DCMAKE_BUILD_TYPE=$build_type \
-DMAX_DET_ORB=128 \
-DPYTHON_EXECUTABLE=/opt/anaconda3/bin/python \
-DENABLE_ForteTests=TRUE \

make -j`getconf _NPROCESSORS_ONLN`
```

Advanced compilation options

Number of threads used to compile Forte

To speed up compilation of Forte specify the number of threads to use for compilation. This can be done in the `setup.cfg` file via

```
[CMakeBuild]
nprocs=<number of threads>
```

or when using CMake, compile Forte with the option -jn, for example, to compile with four threads

```
make -j4
```

Add configuration and build options

When using `setup.py` you can specify the `CMAKE_CONFIG_OPTIONS` and `CMAKE_BUILD_OPTIONS` passed internally to CMake in `setup.cfg`

```
[CMakeBuild]
cmake_config_options=...
cmake_build_options=...
```

These are convenient if you want to specify a different compiler from the one automatically detected by CMake.

Maximum number of orbitals in the ``Determinant`` class

By default, Forte is compiled assuming that the maximum number of orbitals that can be handled by codes that use the `Determinant` class is 64. To change this value modify the `setup.cfg` file to include

```
[CMakeBuild]
max_det_orb=<a multiple of 64>
```

or add the option

```
-DMAX_DET_ORB=<a multiple of 64>
```

if compiling with CMake.

Enabling code coverage

To enable compilation with code coverage activated, set the option `enable_codecov` to `ON` in the `setup.cfg` file `tcsh [CMakeBuild] enable_codecov=ON` or add the option `tcsh -DENABLE_CODECOV=ON` if compiling with CMake.

Compilation via `setup.py` (recommended)

The most convenient way to compile forte is using the `setup.py` script. To compile Forte do the following:

1. Tell `setup.py` where to find ambit, which can be done either by setting the environmental variable `AMBITDIR` to point to the ambit install directory (note: there is no need to append `share/cmake/ambit`)

```
export AMBITPATH=<ambit install dir>
```

or by modifying the <`fortedir`>`/setup.cfg` file to include

```
[CMakeBuild]
ambitpath=<ambit install dir>
```

2. Compile forte by calling in <`fortedir`>

```
python setup.py develop
```

or for Debug mode

```
fortedir> python setup.py build_ext --debug develop
```

This procedure will register forte within pip and you should be able to see forte listed just by calling

```
pip list
```

Compilation via CMake

Forte may also be compiled by directly invoking CMake by following these instructions:

1. Run psi4 in the Forte folder

```
psi4 --plugin-compile
```

Psi4 will generate a CMake command for building Forte that looks like: `cmake -C /usr/local/psi4/stage/usr/local/psi4/share/cmake/psi4/psi4PluginCache.cmake -DCMAKE_PREFIX_PATH=/usr/local/psi4/stage/usr/local/psi4 .`

2. Run the cmake command generated in 1. appending the location of Ambit's cmake files (via the `-Dambit_DIR` option):

```
cmake -C /usr/local/psi4/stage/usr/local/psi4/share/cmake/psi4/psi4PluginCache.cmake -
↪DCMAKE_PREFIX_PATH=/usr/local/psi4/stage/usr/local/psi4 . -Dambit_DIR=<ambit-bin-dir>/
↪share/cmake/ambit
```

3. Run make

```
make
```

The following script automates steps 1 and 2 of the forte compilation process

```
#! /bin/tcsh

# Modify the following four parameters
set ambit_dir = /Users/fevange/Bin/ambit-Release/share/cmake/ambit/ # <- location of␣
↪ambit
set srcdir = /Users/fevange/Source/forte   # <- location of forte source
set build_type = Release # <- Release, Release, or RelWithDebInfo

# Run cmake
cd $srcdir

set cmake_psi4 = `psi4 --plugin-compile`

$cmake_psi4 \
-Dambit_DIR=$ambit_dir \ # remove this line if ambit is installed via conda
-DCMAKE_BUILD_TYPE=$build_type \
-DMAX_DET_ORB=128 \
-DPYTHON_EXECUTABLE=/opt/anaconda3/bin/python \
-DENABLE_ForteTests=TRUE \
```

Advanced compilation options

- **Maximum number of orbitals in the `Determinant` class**. By default, Forte is compiled assuming that the maximum number of orbitals that can be handled by codes that use the `Determinant` class is 64. To change this value modify the `<fortedir>/setup.cfg` file to include

```
[CMakeBuild]
max_det_orb=<a multiple of 64>
```

or add the option

```
-DMAX_DET_ORB=<a multiple of 64>
```

if compiling with CMake.

- **Enabling code coverage**. To enable compilation with code coverage activated, set the option `enable_codecov` to `ON` in the `<fortedir>/setup.cfg` file

```
[CMakeBuild]
enable_codecov=ON
```

or add the option

```
-DENABLE_CODECOV=ON
```

if compiling with CMake.

2.2 Running Forte computations

In this section, we will look at the basics of running computations in Forte.

There are two ways one can run Forte:

1. Using the plugin interface in Psi4.
2. Using Forte's python API.

2.2.1 Running a FCI computation using the plugin interface

Let's start by looking at how one can run Forte as a plugin to Psi4. The following text input (see `examples/plugin/01_fci/input.dat`) can be used to run a FCI computation on the lithium dimer:

```
# examples/plugin/01_fci/input.dat

import forte

molecule {
0 1
Li 0.0 0.0 0.0
Li 0.0 0.0 3.0
units bohr
}

set {
```

(continues on next page)

(continued from previous page)

```
  basis sto-3g
  scf_type pk
  e_convergence 10
}

set forte {
  active_space_solver fci
}

energy('forte')
```

To understand the structure of the input file, we will go over each section of this input.

- The file start with the `import forte` command, which loads the Forte module.
- The next section specifies the molecular structure and the charge/multiplicity of the molecule. This section accepts inputs as specified in Psi4's Molecule and Geometry Specification documentation, and accepts both Cartesian and Z-matrix coordinates

```
molecule {
0 1
Li 0.0 0.0 0.0
Li 0.0 0.0 3.0
units bohr
}
```

- The options block that follows passes options to Psi4. Here we set the basis (`basis sto-3g`), the type of SCF integral algorithm (`scf_type pk`, which uses conventional integrals), and the energy convergence threshold (`e_convergence 10`, equivalent to 10^{-10} E_h)

```
set {
  basis sto-3g
  scf_type pk
  e_convergence 10
}
```

- The next section sets options specific to Forte. In a typical Forte job the user needs to specify two objects:
  - An **active space solver**, used to treat static correlation effects. The active space solver finds a solution to the Schrödinger equation in the subset of active orbitals.
  - A **dynamical correlation solver**, used to add dynamical electron correlation corrections on top of a wave function defined in the active space. To run a FCI computation, we only need to specify the active space solver, which is done by setting the option `active_space_solver`:

```
set forte {
  active_space_solver fci
}
```

- The last line of the input calls the Psi4 energy method specifing that we want to run the `forte` module

```
energy('forte')
```

To run this computation we invoke psi4 on the command line

```
>>>psi4 input.dat
```

This will run psi4 and produce the output file `output.dat`, a copy of which is available in the file `examples/plugin/01_fci/output.dat`. From this output, we can read the CI coefficient of the most important determinants written in occupation number representation

```
220 0 0 200 0 0      0.89740847 <-- coefficient
200 0 0 200 0 2     -0.29206218
200 0 0 200 2 0     -0.29206218
200 0 0 220 0 0     -0.14391931
```

and a summary of the total energy of a state and the expectation value of the spin squared operator ($\hat{S}^2$)

```
Multi.(2ms)  Irrep.  No.               Energy      <S^2>
-------------------------------------------------------
   1  (  0)    Ag     0      -14.595808852754  -0.000000
-------------------------------------------------------
```

2.2.2 Running a FCI computation using the python API

The following input runs the same FCI computation discussed above using the python API:

```
# examples/api/01_fci.py

import psi4
import forte

psi4.geometry("""
0 1
Li 0.0 0.0 0.0
Li 0.0 0.0 3.0
units bohr
""")

psi4.set_options({
    'basis': 'sto-3g',                    # <-- set the basis set
    'scf_type': 'pk',                     # <-- request conventional two-electron␣
↪integrals
    'e_convergence': 10,                  # <-- set the energy convergence
    'forte__active_space_solver' : 'fci'} # <-- specify the reference
    )

psi4.energy('forte')
```

This python file mirrors the psi4 input file.

- The file start with both the `import psi4` and `import forte` commands, to load both the psi4 and Forte modules.
- The next command creates a psi4 `Molecule` object calling the function `psi4.geometry`. This object is stored in a default memory location and automatically used by psi4

```
psi4.geometry("""
0 1
Li 0.0 0.0 0.0
Li 0.0 0.0 3.0
units bohr
""")
```

- The options block that follows passes options to both Psi4 and Forte. Here we pass options as a python dictionary, prefixing options that are specific to Forte with `forte__`:

```
psi4.set_options({
    'basis': 'sto-3g',                    # <-- set the basis set
    'scf_type': 'pk',                     # <-- request conventional two-electron␣
↪integrals
    'e_convergence': 10,                  # <-- set the energy convergence
    'forte__active_space_solver' : 'fci'} # <-- specify the active space solver
    )
```

- The last line of the python code calls the Psi4 energy method specifing that we want to run the `forte` module

```
psi4.energy('forte')
```

This computation is identical to the previous one and produces the exact same output (see `examples/plugin/01_fci.out`).

2.2.3 Passing options in Forte: psi4 interface vs. dictionaries (new)

In the previous sections, calcultation options were passed to Forte through psi4. An alternative way to pass options is illustrated in the following example (using the python API):

```
# examples/api/07_options_passing.py
"""Example of passing options as a dictionary in an energy call"""

import psi4
import forte

psi4.geometry("""
0 3
C
H 1 1.085
H 1 1.085 2 135.5
""")

psi4.set_options({
    'basis': 'DZ',
    'scf_type': 'pk',
    'e_convergence': 12,
    'reference': 'rohf',
})

forte_options = {
    'active_space_solver': 'fci',
```

(continues on next page)

(continued from previous page)

```
    'restricted_docc': [1, 0, 0, 0],
    'active': [3, 0, 2, 2],
    'multiplicity': 3,
    'root_sym': 2,
}

efci1 = psi4.energy('forte', forte_options=forte_options)

forte_options['multiplicity'] = 1
forte_options['root_sym'] = 0
forte_options['nroot'] = 2
forte_options['root'] = 1

efci2 = psi4.energy('forte', forte_options=forte_options)
```

- Note how in this file we create a python dictionary (`forte_options`) and pass it to the `energy` function as the parameter `forte_options`.
- Passing options via a dictionary takes priority over passing options via psi4. This means that **any option previously passed via psi4 is ignored**.
- This way of passing options is **safer** than the one based on psi4 because, unless the user intentionally passes the same dictionary in the energy call, there is no memory effect where previously defined options have an effect on all subsequent calls to `energy`.
- Note how later in the file we call `energy` again but this time we modify the options directly by modifying the dictionary

```
forte_options['multiplicity'] = 1
forte_options['root_sym'] = 0
forte_options['nroot'] = 2
forte_options['root'] = 1
```

Here we change the multiplicity and symmetry of the target state, and compute two roots, reporting the energy of the second one.

This computation is identical to the previous one and produces the exact same output (see `examples/plugin/01_fci.out`).

2.2.4 Test cases and Jupyter Tutorials

- **Test cases**. Forte provides test cases for most of all methods implemented. This is a good place to start if you are new to Forte. Test cases based on Psi4's plugin interface can be found in the `<fortedir>/tests/methods` folder. Test cases based on Forte's python API can be found in the `<fortedir>/tests/pytest` folder.
- **Jupyter Tutorials for Forte's Python API**. Forte is designed as a C++ library with a lot of the classes and functionality exposed in Python via the `pybind11` library. Tutorials on how to use Forte's API can be found here.

2.3 Specifying a wave function in Forte

To uniquely identify an electronic state, Forte needs to know the total number of electrons (N), the number of alpha and beta electrons (N_α/N_β), the spin state, and the symmetry of the state(s) computed. These quantities are specified via various options, as detailed below.

2.3.1 Charge and spin multiplicity

The molecular charge (Q), is defined as the sum of the charge of the nuclei (Z_A) minus the number of electrons

$$Q = \sum_A Z_A - N_{\text{el}}.$$

Recall that, in the absence of an external field, an electronic state is an eigenfunction of the spin squared operator $\hat{S}^2$ and the z component of the spin operator $\hat{S}_z$. The eigenvalues of these two operators are related to the spin quantum numbers S and M_S via (in atomic units)

$$\hat{S}^2|S, M_S\rangle = S(S+1)|S, M_S\rangle,$$

and

$$\hat{S}_z|S, M_S\rangle = M_S|S, M_S\rangle, \quad M_S = -S, -S+1, \ldots, S-1, S$$

The spin multiplicity is defined as

$$\text{MULTIPLICITY} = 2S + 1,$$

and this quantity is used instead of S to specify which spin state is being computed.

Unless otherwise specified, when a psi4 `Molecule` object is created, the molecular charge is assumed to be 0, and the spin multiplicity is assumed to be 1 (singlet state). These quantities may specified by the user in the first line of the geometry input:

```
molecule {
  0 1 # <-- charge and multiplicity (neutral,singlet)
  ...
}

molecule {
  1 4 # <-- (cation,quartet)
  ...
}
```

The charge and multiplicity specified in the molecule section of the input (and read by psi4) are stored in the `Wavefunction` object generated by psi4 and are read by Forte, where they are used in the rest of the computation.

To select a different charge and multiplicity in Forte, it is possible to specify the options `CHARGE` and `MULTIPLICITY` in the forte input section. These options override the value passed in via the `Wavefunction` object (and the `Wavefunction` object takes priority over the values specified in the molecular geometry input).

2.3.2 Specifying the z component of spin (M_S)

Most quantum chemistry codes take as an input the spin multiplicity and then select the highest possible value of M_S compatible with S, that is, $M_S = S$. Note that M_S is connected to the number of alpha/beta electrons via

$$M_S = \frac{1}{2}(N_\alpha - N_\beta)$$

and, therfore, together with the total number of electrons, it determines the value of N_α and N_β.

In Forte, modules will select either the lowest absolute or highest value of M_S compatible with S (as specified via `MULTIPLICITY`), depending on internal details of the implementation. For example, if `MULTIPLICITY = 3` and M_S is not specified, the FCI code in Forte will assume that the user is interested in the solution with $M_S = 0$.

However, Forte also allows the user to select electronic states with a well defined value of M_S. This quantity may be specified via the option `MS`. This option is of type `double`, so it should be specified as `0.0`, `-1.5`, etc.

For example, the following input requests the $M_S = 0$ component of a triplet state:

```
# triplet, m_s = 0
set forte {
    multiplicity = 3
    ms = 0.0
}
```

while the following gives the $M_S = -1$ component:

```
# triplet, m_s = 1
set forte {
    multiplicity = 3
    ms = -1.0
}
```

2.3.3 Point group symmetry

Forte takes advantage of symmetry, so it important to specify both the symmetry of the target electronic state and the orbital spaces that define a computation (see below). Forte supports only Abelian groups (C_1, C_s, C_i, C_2, C_{2h}, C_{2v}, D_2, D_{2h}). If a molecule has non-Abelian point group symmetry, the largest Abelian subgroup will be used. For a given group, the irreducible representations (irrep) are arranged according to Cotton's book (*Chemical Applications of Group Theory*). This ordering is reproduced in the following table and is the same as used in Psi4:

P oint g roup	I rrep 0	I rrep 1	I rrep 2	I rrep 3	I rrep 4	I rrep 5	I rrep 6	I rrep 7
:ma th:` C_1`	: math :*A*							
:ma th:` C_s`	:m ath: *A*’	:ma th:` A”`						
:ma th:` C_i`	: math :*A_ {g}*	: math :*A_ {u}*						
:ma th:` C_2`	: math :*A*	: math :*B*						
:m ath: *C_{ 2h}*	: math :*A_ {g}*	: math :*B_ {g}*	: math :*A_ {u}*	: math :*B_ {u}*				
:m ath: *C_{ 2v}*	: math :*A_ {1}*	: math :*B_ {1}*	: math :*A_ {2}*	: math :*B_ {2}*				
:ma th:` D_2`	: math :*A*	: math :*B_ {1}*	: math :*B_ {2}*	: math :*B_ {3}*				
:m ath: *D_{ 2h}*	: math :*A_ {g}*	:m ath: *B_{ 1g}*	:m ath: *B_{ 2g}*	:m ath: *B_{ 3g}*	: math :*A_ {u}*	:m ath: *B_{ 1u}*	:m ath: *B_{ 2u}*	:m ath: *B_{ 3u}*

By default, Forte targets a total symmetric state (e.g., A_1, A_g, ...). To specify a state with a different irreducible representation (irrep), provide the `ROOT_SYM` option. This option takes an integer argument that indicates the irrep in Cotton's ordering.

2.3.4 Definition of orbital spaces

Running a Forte computation requires specifying a partitioning of the molecular orbitals. Forte defines five types of elementary orbital spaces:

1. Frozen doubly occupied orbitals (`FROZEN_DOCC`). These orbitals are always doubly occupied.
2. Restricted doubly occupied orbitals (`RESTRICTED_DOCC`). Orbitals that are treated as doubly occupied by method for static correlation. Restricted doubly occupied orbitals are allowed to be excited in in methods that add dynamic electron correlation.
3. Active/generalized active orbitals (`ACTIVE/GASn`). Used to define active spaces or generalized active spaces for static correlation methods. These orbitals are partially occupied. Standard complete active spaces can be specified either via the `ACTIVE` or the `GAS1` orbital space. For generalized active spaces, the user must provide the number of orbitals in each irrep for all the GAS spaces required. `GAS1` through `GAS6` are currently supported.
4. Restricted unoccupied orbitals (`RESTRICTED_UOCC`). Also called virtuals, these orbitals are ignored by methods for static correlation but considered by dynamic correlation approaches.
5. Frozen unoccupied orbitals (`FROZEN_UOCC`). These orbitals are always unoccupied.

The following table summarizes the properties of these orbital spaces:

Space	Occupation in CAS/GAS	Occupation in correlated methods	Description
FROZEN_DOCC	2	2	Frozen doubly occupied orbitals
RESTRICTED_DOCC	2	0-2	Restricted doubly occupied orbitals
GAS1, GAS2, ...	0-2	0-2	Generalized active spaces
RESTRICTED_UOCC	0	0-2	Restricted unoccupied orbitals
FROZEN_UOCC	0	0	Frozen unoccupied orbitals

Note: Forte makes a distinction between elementary and `composite` orbital spaces. The spaces defined above are all elementary, except for `ACTIVE`, which is defined as the composite space of all the GAS spaces, that is, `ACTIVE = GAS1 | GAS2 | GAS3 | GAS4 | GAS5 | GAS6`. When the user specifies the value of a composite space like `ACTIVE`, then all the orbitals are by default assigned to the first space, which in the case of `ACTIVE` is `GAS1`. It is important also to note that when there is more than one irrep, the orbitals within a composite space are ordered **first** by irrep and then by elementary space. This is important to keep in mind when plotting orbitals or for developers writing code in forte.

2.3.5 Orbital space specification

Selecting the correct set of orbitals for a multireference computation is perhaps one of the most important steps in setting up an input file. To specify an orbital space, the user must provide the number of orbitals contained in each irrep (see Point group symmetry). Since Forte only supports Abelian groups, each orbital space can be specified by a vector of integers with at most eight entries. Note that irreps are arranged according to Cotton's book (*Chemical Applications of Group Theory*).

The following is an example of a computation on BeH_2. This system has 6 electrons. We freeze the Be 1s-like orbital, which has A_1 symmetry. The $2a_1$ orbital is restricted doubly occupied and the $3a_1/1b_2$ orbitals belong to the active space. The remaining orbitals belong to the `RESTRICTED_UOCC` set and no virtual orbitals are frozen:

```
set forte{
    #          A1 A2 B1 B2
    frozen_docc      [1 ,0 ,0 ,0]
    restricted_docc  [2 ,0 ,0 ,0]
    active           [1 ,0 ,0 ,1]
    restricted_uocc  [4 ,0 ,2 ,3]
    frozen_uocc      [0 ,0 ,0 ,0]
}
```

2.3.6 Partial specification of orbital spaces and space priority

Specifying all five orbital spaces for each computation is tedious and error prone. Forte can help reduce the number of orbital spaces that the user needs to specify by making certain assumptions. The class that controls orbital spaces (`MOSpaceInfo`) assumes that orbital spaces have the following priority:

```
GAS1 (= ACTIVE) > RESTRICTED_UOCC > RESTRICTED_DOCC > FROZEN_DOCC > FROZEN_UOCC > GAS2 >␣
↪...
```

When the input does not contain all five orbital spaces, Forte will infer the size of other orbital spaces. It first sums up all the orbitals specified by the user, and then assigns any remaining orbitals to the space not specified in the input that has the highest priority.

In the case of the BeH_2 example, it is necessary to specify only the FROZEN_DOCC, RESTRICTED_DOCC, and ACTIVE orbital spaces:

```
set forte{
    frozen_docc         [1 ,0 ,0 ,0]
    restricted_docc     [2 ,0 ,0 ,0]
    active              [1 ,0 ,0 ,1]

    # Forte will automatically assign the following:
    # restricted_uocc   [4 ,0 ,2 ,3]
    # frozen_uocc       [0 ,0 ,0 ,0]
    # gas1              [1 ,0 ,0 ,1]
    # gas2              [0 ,0 ,0 ,0]
    # gas3              [0 ,0 ,0 ,0]
    # gas4              [0 ,0 ,0 ,0]
    # gas5              [0 ,0 ,0 ,0]
    # gas6              [0 ,0 ,0 ,0]
```

}

the remaining 9 orbitals are automatically assigned to the RESTRICTED_UOCC space. This space, together with FROZEN_UOCC, was not specified in the input. However, RESTRICTED_UOCC has higher priority than the FROZEN_UOCC space, so Forte will assign all the remaining orbitals to the RESTRICTED_UOCC set.

A Forte input with no orbital space specified will assign all orbitals to the active space:

```
set forte{
    # Forte will automatically assign the following:
    # frozen_docc       [0 ,0 ,0 ,0]
    # restricted_docc   [0 ,0 ,0 ,0]
    # active            [7 ,0 ,2 ,4]
    # restricted_uocc   [0 ,0 ,0 ,0]
    # frozen_uocc       [0 ,0 ,0 ,0]
}
```

Note that except for computations with small basis sets, declaring all orbitals active might be unfeasible.

As a general rule, it is recommended that users run SCF computations and inspect the orbitals prior to selecting an active space.

2.3.7 Occupation numbers of GAS wave functions

General active space (GAS) wave functions are defined by partitioning the active space into subspaces and specifying constraints on the occupation of these subspaces. To specify a general active space (GAS) wave function, the user must select the GAS spaces (see Definition of orbital spaces) and the minimum and maximum occupation numbers of each GAS space. This is done by passing two list of integers for each GASN space, GASNMIN and GASNMAX. For example, the following input defines the orbitals associated with two GAS spaces (GAS1 and GAS2).

```
set forte{
    gas1        [2,0,0,0]
    gas2        [2,0,1,2]
    gas1min     [2]
    gas1max     [4]
}
```

The options GAS1MIN and GAS1MAX specify the minimum and maximum numbers allowed in the GAS1 space. This information is sufficient to determine all possible GAS occupations.

2.4 Specifying calculations of multiple states

2.4.1 Requesting multiple solutions of a given spin and symmetry

Codes that support excited states take the additional option NROOT, which can be used to specify the number of solutions (roots) of the charge, multiplicity, and symmetry specified by the user.

Assuming a C_{2v} molecular point group, the following example is for an input to compute three state of symmetry $^4A_2$ for a neutral molecule:

```
set forte {
  charge       0 # <-- neutral
  multiplicity 4 # <-- quartet
  root_sym     1 # <-- A_2
  nroot        3 # <-- three solutions
}
```

2.4.2 Requesting multiple solutions of different spin and symmetry

For certain types of multistate computations (e.g., state-averaged CASSCF), one may want to compute solutions of different spin and symmetry.

The simplest way to do so is by specifying the AVG_STATE option to define different sets of electronic states. This option is passed as a list of triplets of numbers `[[irrep1, multi1, nstates1], [irrep2, multi2, nstates2], ...]`, where `irrep`, `multi`, and `nstates` specify the irrep, multiplicity, and the number of states of each type requested.

For example, for a molecule with C_{2v} point group symmetry, the following input requests four $^3B_1$ states and two $^5B_2$ states:

```
set forte {
  avg_state [[2,3,4],[3,5,2]] # <-- [(B1, triplet, 4 states), (B2,quintet,2 states)]
}
```

When AVG_STATE is specified, each state is assigned a weight, which by default is $1/N$ where N is the total number of states computed. The weights of all the states can also be indicated with the AVG_WEIGHT option. This option is a list of lists of numbers that indicate the weight of each state in a triplet defined via AVG_STATE. This option takes the format `[[w1_1, w1_2, ..., w1_l], " [w2_1, w2_2, ..., w2_m], ...]`, where each sublist specifies the weights of states defined by a triplet `[irrep, multi, nstates]`.

Suppose we want to do a computation on a singlet and two triplet A_1 states, and assign a weight of 1/4 to the $^1A_1$ state and weights of 1/2 and 1/4 to the $^3A_1$ states. This computation can be specified by the input:

```
set forte {
  avg_state [[0,1,1],[0,3,2]]
  avg_weight [[0.25],[0.5,0.25]]
}
```

If the state weights do not add up to one, Forte will scale them, so the following input is an equivalent way to perform the same computation:

```
set forte {
  avg_state [[0,1,1],[0,3,2]]
  avg_weight [[1.],[2.,1.]]
}
```

2.4.3 Multistate GAS calculations

Multistate computations using a GAS partitioning (see :ref:`Occupation numbers of GAS wave functions`) can be used to generate even more nuanced electronic states. When the electronic states are specified via the `AVG_STATE` option, one can indicate states with different GAS occupations by setting the `GASNMIN` and `GASNMAX` options. For multistate computations, these are lists that specify the minimum and maximum occupation of each GAS space for each triplet that defines an electronic state.

For example, the test case `tests/methods/gasci-2` shows how to compute two electronic states of the water molecule of $^1A_1$ symmetry. These two states use different occupation restrictions. Specifically, the O 1s-like orbital ($1a_1$) has maximum occupation of 2 and 1 in the two electronic states:

```
set forte {
  gas1    [1,0,0,0]
  gas2    [3,0,1,2]
  gas1min [0,0]
  gas1max [2,1] # The second set of states is constrained to have at most 1 electron in␣
↪GAS1
  avg_state [[0,1,1],[0,1,1]] # 2 states of singlet A_1 symmetry and different GAS
}
```

While the first state is representative of the ground state of water, the second state corresponds to a core-excited state.

2.5 Examples of advanced Forte computations

2.5.1 Computing one or more FCI solutions of a given symmetry

The first example shows how to perform separate computations on different electronic states of methylene. We compute the lowest $^3B_1$ state and the first two $^1A_1$ states. All of these computations use ROHF orbitals optimized for the lowest $^3B_1$. Here we use the python API interface of Forte, but it is easy to translate these examples to a psi4 psithon input.

This input (see `examples/api/02_rohf_fci.py`) starts with the geometry specification:

```
# examples/api/02_rohf_fci.py

import psi4
import forte

mol = psi4.geometry("""
0 3 # triplet state
C
H 1 1.085
H 1 1.085 2 135.5
""")

psi4.set_options(
```

(continues on next page)

(continued from previous page)

```
    {
        'basis': 'DZ',
        'scf_type': 'pk', # <-- request conventional two-electron integrals
        'e_convergence': 12,
        'reference': 'rohf',
        'forte__active_space_solver': 'fci',
        'forte__restricted_docc': [1, 0, 0, 0],
        'forte__active': [3, 0, 2, 2],
        'forte__multiplicity': 3, # <-- triplet state
        'forte__root_sym': 2, # <-- B1 symmetry
    }
)
```

Note that all options prefixed with `forte__` are specific to the Forte computation. Here we specify a multiplicity equal to 3 and the B_1 irrep (`root_sym = 2`). In this example we keep the lowest A_1 MO doubly occupied (`'restricted_docc': [1, 0, 0, 0]`) and use an active space that contains three A_1 MOs, and two MOs each of B_1 and B_2 symmetry. Lastly, we run Forte:

```
psi4.energy('forte')
```

This input will run a CASCI computation (since we have not requested orbital optimization). An example of how to request orbital optimization can be found in the section *Computing a manifold of solutions of different symmetry*. The output will return the energy and show the composition of the wave function:

```
==> Root No. 0 <==

2b0 a0 20      0.70326213
2a0 b0 20     -0.70326213

Total Energy:     -38.924726774489, <S^2>: 2.000000

==> Energy Summary <==

Multi.(2ms)  Irrep.  No.               Energy      <S^2>
-----------------------------------------------------------
   3  (  0)    B1     0      -38.924726774489   2.000000
-----------------------------------------------------------
```

Next we change the options to compute the lowest two $1A_1$ states. We modify the multiplicity, the symmetry, and indicate that we want two roots (`NROOTS = 2`):

```
psi4.set_options({
    'forte__multiplicity': 1,
    'forte__root_sym': 0, # <-- A1 symmetry
    'forte__nroots' : 2}
)
psi4.energy('forte')
```

The results of this computation is:

```
==> Root No. 0 <==

220 00 20      0.92134189
```

(continues on next page)

(continued from previous page)

```
200 20 20     -0.37537841

Total Energy:     -38.866616413802, <S^2>: -0.000000

==> Root No. 1 <==

200 20 20     -0.89364609
220 00 20     -0.36032959
ab0 20 20     -0.13675846
ba0 20 20     -0.13675846

Total Energy:     -38.800424868719, <S^2>: -0.000000

==> Energy Summary <==

Multi.(2ms)  Irrep.  No.               Energy      <S^2>
-------------------------------------------------------------
   1  (  0)    A1     0      -38.866616413802  -0.000000
   1  (  0)    A1     1      -38.800424868719  -0.000000
-------------------------------------------------------------
```

2.5.2 State-averaged CASSCF with states of different symmetry

The next example shows how to perform a state-averaged CASSCF computation on two electronic states of different symmetries. We still consider methylene, and average the lowest $^3B_1$ and $^1A_1$ states. To begin, we use ROHF orbitals optimized for the lowest $^3B_1$. However, the final orbitals will optimize the average energy **:raw-latex:`\begin{equation} E_\mathrm{avg} = \frac{1}{2} \left(E_{^3B_1} + E_{^1A_1}\right). \end{equation}`** We use the same active space of the previous example, but here to specify the state, we set the `AVG_STATE` option:

```
# examples/api/03_sa-casscf.py

import psi4
import forte

psi4.geometry("""
0 3
C
H 1 1.085
H 1 1.085 2 135.5
""")

psi4.set_options({'basis': 'DZ', 'scf_type': 'pk', 'e_convergence': 12, 'reference':
↪'rohf',
        'forte__job_type': 'mcscf_two_step',
        'forte__active_space_solver': 'fci',
        'forte__restricted_docc': [1, 0, 0, 0],
        'forte__active': [3, 0, 2, 2],
        'forte__avg_state': [[2, 3, 1], [0, 1, 1]]
        # [(B1, triplet, 1 state), (A1,singlet,1 state)]
    }
)
```

(continues on next page)

(continued from previous page)

```
psi4.energy('forte')
```

The output of this computation (in `examples/api/03_sa-casscf.out`) shows the energy for both states in the following table:

```
==> Energy Summary <==

Multi.(2ms)  Irrep.  No.               Energy      <S^2>
-----------------------------------------------------------
   1  (  0)    A1     0      -38.900217662950   0.000000
-----------------------------------------------------------
   3  (  0)    B1     0      -38.960623289646   2.000000
-----------------------------------------------------------
```

2.5.3 Using different mean-field guesses in CASSCF computations

A common issue when running CASSCF computation problematic convergence due to a poor orbital guess. By default, Forte's CASSCF code uses a Hartree-Fock guess on a state with the same charge and multiplicity of the solution that we are seeking. The next example shows how to provide initial orbitals from states with different multiplicity, different charge and multiplicity, or obtained via DFT. Here we target the singlet state of methylene, using the same active space of the previous example.

Guess with a different multiplicity

In the first example, we will ROHF orbitals for the triplet state as a starting guess for CASSCF. To specify a triplet state we modify the geometry section. After the ROHF computation, we pass the option `forte__multiplicity` to instruct Forte to optimize a singlet state

```
# examples/api/04_casscf-triplet-guess.py

import psi4
import forte

psi4.geometry("""
0 3 # <-- here we specify a triplet state
C
H 1 1.085
H 1 1.085 2 135.5
""")

psi4.set_options({'basis': 'DZ', 'scf_type': 'pk', 'e_convergence': 12, 'reference':
↪'rohf'})

e, wfn = psi4.energy('scf',return_wfn=True)

psi4.set_options({
        'forte__job_type': 'mcscf_two_step',
        'forte__multiplicity' : 1, # <-- to override multiplicity = 2 assumed from␣
↪geometry
```

(continues on next page)

(continued from previous page)

```
        'forte__active_space_solver': 'fci',
        'forte__restricted_docc': [1, 0, 0, 0],
        'forte__active': [3, 0, 2, 2],
    }
)

psi4.energy('forte',ref_wfn=wfn)
```

Guess with different charge and multiplicity

In the second example, we will ROHF orbitals for the doublet cation as a starting guess for CASSCF. The relevant changes are made in the geometry section, where we indicate a charge of +1 and multiplicity equal to 2:

```
# examples/api/05_casscf-doublet-guess.py

# ...

psi4.geometry("""
1 2
C
H 1 1.085
H 1 1.085 2 135.5
""")
```

and we also add options to fully specify the values of charge, multiplicity, and M_S used to perform the CASSCF computation

```
psi4.set_options({
        'forte__job_type': 'mcscf_two_step',
        'forte__charge' : 0, # <-- to override charge = +1 assumed from geometry
        'forte__multiplicity' : 1, # <-- to override multiplicity = 2 assumed from␣
↪geometry
        'forte__ms' : 0, # <-- to override ms = 1/2 assumed from geometry
        'forte__active_space_solver': 'fci',
        'forte__restricted_docc': [1, 0, 0, 0],
        'forte__active': [3, 0, 2, 2],
    }
)
```

Guess based on DFT orbitals

In the last example, we pass DFT orbitals (triplet UB3LYP) as a starting guess:

```
# examples/api/06_casscf-dft-guess.py

# ...

psi4.geometry("""
0 3
C
```

(continues on next page)

(continued from previous page)

```
H 1 1.085
H 1 1.085 2 135.5
""")

psi4.set_options({'basis': 'DZ', 'scf_type': 'pk', 'e_convergence': 12, 'reference': 'uks
↪'})

e, wfn = psi4.energy('b3lyp',return_wfn=True)

psi4.set_options({
        'forte__job_type': 'mcscf_two_step',
        'forte__charge' : 0, # <-- to override charge = +1 assumed from geometry
        'forte__multiplicity' : 1, # <-- to override multiplicity = 2 assumed from␣
↪geometry
        'forte__ms' : 0, # <-- to override ms = 1/2 assumed from geometry
        'forte__active_space_solver': 'fci',
        'forte__restricted_docc': [1, 0, 0, 0],
        'forte__active': [3, 0, 2, 2],
    }
)
```

The following is a numerical comparison of the convergence pattern of these three computations and the default guess used by Forte (singlet RHF, in this case)

```
Singlet RHF guess (default guess)

                    Energy CI                    Energy Orbital
           ------------------------------  ------------------------------
    Iter.        Total Energy       Delta        Total Energy       Delta  Orb. Grad. ␣
↪Micro
    ------------------------------------------------------------------------------------
↪---
       1     -38.869758479716  0.0000e+00    -38.878577088058  0.0000e+00  6.0872e-07   ␣
↪ 9
       2     -38.894769773234 -2.5011e-02    -38.899566097104 -2.0989e-02  5.9692e-07   ␣
↪ 9
       3     -38.900644175245 -5.8744e-03    -38.900811142131 -1.2450e-03  2.8974e-07   ␣
↪ 7
       4     -38.900845440821 -2.0127e-04    -38.900853293556 -4.2151e-05  2.1094e-07   ␣
↪ 6
       5     -38.900856221599 -1.0781e-05    -38.900856382896 -3.0893e-06  3.5078e-07   ␣
↪ 5
       6     -38.900856468519 -2.4692e-07    -38.900856468929 -8.6033e-08  2.7648e-07   ␣
↪ 4
       7     -38.900856469070 -5.5024e-10    -38.900856469077 -1.4813e-10  3.4940e-08   ␣
↪ 3
    ------------------------------------------------------------------------------------
↪---


Triplet ROHF guess (04_casscf-triplet-guess.out)
```

(continues on next page)

(continued from previous page)

```
                    Energy CI                    Energy Orbital
           ------------------------------  ------------------------------
    Iter.        Total Energy       Delta        Total Energy       Delta  Orb. Grad.  Micro
    ----------------------------------------------------------------------------------------
       1    -38.866616410911  0.0000e+00   -38.877770272313  0.0000e+00  1.8365e-06   10
       2    -38.894804745194 -2.8188e-02   -38.899492369417 -2.1722e-02  1.8438e-06    9
       3    -38.900608150627 -5.8034e-03   -38.900797672192 -1.3053e-03  1.1877e-07    8
       4    -38.900840657824 -2.3251e-04   -38.900851568289 -5.3896e-05  2.8346e-07    6
       5    -38.900856107378 -1.5450e-05   -38.900856342345 -4.7741e-06  3.5145e-07    5
       6    -38.900856468806 -3.6143e-07   -38.900856469025 -1.2668e-07  2.6265e-07    4
       7    -38.900856469077 -2.7063e-10   -38.900856469079 -5.3831e-11  3.0108e-08    3
    ----------------------------------------------------------------------------------------


Doublet ROHF (examples/api/05_casscf-doublet-guess.out)

                    Energy CI                    Energy Orbital
           ------------------------------  ------------------------------
    Iter.        Total Energy       Delta        Total Energy       Delta  Orb. Grad.  Micro
    ----------------------------------------------------------------------------------------
       1    -38.819565876524  0.0000e+00   -38.862403924512  0.0000e+00  2.4525e-06   14
       2    -38.893973814690 -7.4408e-02   -38.899703281038 -3.7299e-02  2.7109e-06   11
       3    -38.900705647525 -6.7318e-03   -38.900828470211 -1.1252e-03  4.6263e-07    9
       4    -38.900850304213 -1.4466e-04   -38.900854820184 -2.6350e-05  2.4524e-07    8
       5    -38.900856305078 -6.0009e-06   -38.900856411175 -1.5910e-06  4.4290e-07    7
       6    -38.900856468122 -1.6304e-07   -38.900856468764 -5.7589e-08  1.6046e-07    7
       7    -38.900856469077 -9.5575e-10   -38.900856469079 -3.1550e-10  2.7174e-08    3
    ----------------------------------------------------------------------------------------


Unrestricted DFT (B3LYP) guess (examples/api/06_casscf-dft-guess.out)
```

(continues on next page)

(continued from previous page)

```
                    Energy CI                     Energy Orbital
           ------------------------------  ------------------------------
   Iter.        Total Energy       Delta        Total Energy       Delta  Orb. Grad.  
↪Micro
   ------------------------------------------------------------------------------------
↪---
      1    -38.864953251693  0.0000e+00    -38.878418350280  0.0000e+00  1.6735e-06   
↪11
      2    -38.893853205980 -2.8900e-02    -38.899336177723 -2.0918e-02  1.7239e-06   
↪ 9
      3    -38.900627125514 -6.7739e-03    -38.900811355470 -1.4752e-03  1.6239e-07   
↪ 8
      4    -38.900846596355 -2.1947e-04    -38.900853835219 -4.2480e-05  9.5895e-08   
↪ 7
      5    -38.900856239152 -9.6428e-06    -38.900856388795 -2.5536e-06  1.0132e-07   
↪ 6
      6    -38.900856468388 -2.2924e-07    -38.900856468891 -8.0096e-08  6.1277e-08   
↪ 5
      7    -38.900856469072 -6.8447e-10    -38.900856469078 -1.8658e-10  2.4832e-08   
↪ 3
   ------------------------------------------------------------------------------------
↪---
```

2.6 Selecting two-electron integral types

Forte can handle different types of exact and approximate two-electron integrals. This section describes the various options available and their properties/limitations. The selection of different integral types is controlled by the option `INT_TYPE`

2.6.1 Conventional integrals

Conventional integrals are the default choice for Forte. When this option is selected, Forte will compute and store the two-electron integrals in the molecular orbital (MO) basis ϕ_p.

$$\langle pq|rs\rangle = \int dx_1 dx_2 \phi_p^*(x_1)\phi_q^*(x_2) r_{12}^{-1} \phi_r(x_1)\phi_s(x_2)$$

These integrals are computed with Psi4's `IntegralTrasform` class and written to disk. Forte will store three copies of these integrals, the antisymmetrized alpha-alpha and beta-beta integrals

$$\langle p_\alpha q_\alpha \| r_\alpha s_\alpha\rangle, \langle p_\beta q_\beta \| r_\beta s_\beta\rangle,$$

and the alpha-beta integrals (not antisymmetrized)

$$\langle p_\alpha q_\beta | r_\alpha s_\beta\rangle,$$

for all values of p, q, r, s. Storage of these integrals has a memory cost equal to $3N^4$, where N is the number of orbitals that are correlated (frozen core and virtual orbitals excluded). Therefore, conventional integrals are viable for computations with at most 100-200 orbitals. For larger bases, density Fitting and Cholesky decomposition are instead recommended.

2.6.2 Density Fitting (DF) and Cholesky Decomposition (CD)

The density fitting and Cholesky decomposition methods approximate two-electron integrals as products of three-index tensors b_{pr}^{P}

$$\langle pq|rs\rangle = \sum_{P}^{M} b_{pr}^{P} b_{qs}^{P}$$

where M is a quantity of the order $3N$.

Note: The equations reported here use physicist notation for the two-electron integrals, but the DF/CD literature usually adopts chemist's notation. The main difference between DF and CD is in the way the B tensors are defined. In DF, the b tensor is defined as

$$b_{pq}^{Q} = \sum_{p} (pq|P)[(P|Q)^{-1/2}]_{PQ}$$

where the indices P and Q refer to the auxiliary basis set.

Two options control the type of density fitting basis used in forte. The auxiliary basis used in the correlated computations is defined via the Psi4 option `DF_BASIS_MP2`. The auxiliary basis used in CASSCF is defined via the Psi4 option `DF_BASIS_SCF`. These two options can be different, but this might lead to an unconsistent treatment of correlation effects.

In the CD approach, the b tensor is formed via Cholesky decomposition of the exact two-electron integrals in the atomic basis. The accuracy of this decomposition (and the resulting two-electron integrals) is determined by a user defined tolerance selected via the option `CHOLESKY_TOLERANCE`. Both the DF and CD algorithms store the b tensor in memory, and therefore, they require $MN^2 \approx 3N^3$ memory for storage. On a single node with 128 GB of memory, DF and CD computations allow to treat up to 1000 orbitals.

2.6.3 Disk-based Density Fitting (DiskDF)

Calculations with more than 1000 basis functions quickly become unfeasible as the memory requirements of density fitting grows as the cube of basis size. In this case, it is possible to switch to a disk-based implementation of DF, which assumes that the b tensor can be fully stored on disk.

2.6.4 Integrals from a FCIDUMP file

Most of Forte computations can also be executed using integrals read from a FCIDUMP file. To read integrals in the FCIDUMP format just use the option `INT_TYPE = FCIDUMP`. For example:

```
import forte
set forte {
  active_space_solver fci
  int_type            fcidump
  frozen_docc         [2 ,0 ,0 ,0]
  restricted_docc     [2 ,0 ,0 ,0]
  active              [2 ,2 ,2 ,2]
}
```

The default name of the FCIDUMP file is `INTDUMP`, but it can be changed via the option `FCIDUMP_FILE`. Forte will read the number of orbital, number of electrons, the multiplicity, and irrep from the FCIDUMP file. This information is then used to build a `StateInfo` object that contains all information regarding the electronic state that will be computed. The user can, however, select a different state by specifying the number of electrons (`NEL`), multiplicity (`MULTIPLICITY`), and irrep (`ROOT_SYM`) via the appropriate options.

2.6.5 Integral Selection Keywords

The following keywords control the integral class and affect all computations that run in Forte:

- **INT_TYPE** `INT_TYPE` selects the integral type used in the calculation
  - Type: string
  - Default: `CONVENTIONAL`
  - Possible Values: `CONVENTIONAL`, `DF`, `CHOLESKY`, `DISKDF`, `FCIDUMP`
- **CHOLESKY_TOLERANCE** The tolerance for the cholesky decomposition. This keyword determines the accuracy of the computation. A smaller tolerance is a more accurate computation. The tolerance for the cholesky decomposition:
  - Type: double in scientific notation (ie 1e-5 or 0.0001)
  - Default: `1.0e-6`
- **DF_BASIS_MP2** The basis set used for density fitting the integrals used in all correlated computations. This keyword needs to be placed in the globals section of a Psi4 input. This basis should be one of the RI basis sets designed for a given primary basis, for example, when using `BASIS = cc-pVDZ` you should use `DF_BASIS_MP2 = cc-pVDZ-RI`.
  - Type: string specifing basis set
  - Default: none
- **DF_BASIS_SCF** The basis set used for density fitting the integrals used in forte's CASSCF computations. This keyword needs to be placed in the globals section of a Psi4 input. This basis should be one of the JK basis sets designed for a given primary basis, for example, when using `BASIS = cc-pVDZ` you should use `DF_BASIS_SCF = cc-pVDZ-JKfit`.
  - Type: string specifing basis set
  - Default: none
- **FCIDUMP_FILE** `FCIDUMP_FILE` selects the file from which to read the integrals in the FCIDUMP format
  - Type: string
  - Default: `INTDUMP`

2.7 Reading integrals in a spin orbital basis

In this tutorial, you will learn how to read access integrals in a spin orbital basis from python. These integrals can be used in pilot implementations of quantum chemistry methods. By the end of this tutorial you will know how to read the integrals and compute the Hartree-Fock energy. For an implementation of MP2 based on the spin orbital integrals see the file `tests/pytest/helpers/test_spinorbital.py` in the forte directory.

Forte assumes that the spin orbital basis $\{\psi_p\}$ is organized as follows

$$\underbrace{\phi_{0,\alpha}}_{\psi_0}, \underbrace{\phi_{0,\beta}}_{\psi_1}, \underbrace{\phi_{1,\alpha}}_{\psi_2}, \underbrace{\phi_{1,\beta}}_{\psi_3}, \cdots$$

To read the one-electron integrals $h_{pq} = \langle\psi_p|\hat{h}|\psi_q\rangle$ we use the function `spinorbital_oei`. This function takes as arguments a `ForteIntegrals` object and two lists of integers, `p` and `q`, that specify the indices of the bra and ket **spatial orbitals**. For example, if we want the integrals over the bra functions ψ_0, ψ_1, ψ_3 and ket functions ψ_5, ψ_6 we can write the following code

```
p = [0,1,3]
q = [5,6]
h = forte.spinorbital_oei(ints, p, q)
```

To read the two-electron antisymmetrized integrals in physicist notation $\langle pq||rs\rangle$ we use the function `spinorbital_tei`, passing four list that corresponds to the range of the indices `p`, `q`, `r`, and `s`.

```
p = [0,1]
q = [0,1]
r = [2,3]
s = [2,3]
v = forte.spinorbital_tei(ints, p, q, r, s)
```

To compute the SCF energy we evaluate the expression

$$E = V_{\mathrm{NN}} + \sum_{i}^{\mathrm{docc}} h_{ii} + \frac{1}{2}\sum_{ij}^{\mathrm{docc}} \langle ij||ij\rangle$$

where V_{NN} is the nuclear repulsion energy. To evaluate this expression we only need the one- and two-electron integral blocks that corresponds to the doubly occupied orbitals.

2.7.1 Preparing the orbitals via the `utils.psi4_scf` helper function

To prepare an integral object it is necessary to first run a HF or CASSCF computation.

Forte provides helper functions to run these computations using psi4. By default **this function uses conventional integrals**.

```
import math
import numpy as np
import forte
import forte.utils

geom = """
O
H 1 1.0
H 1 1.0 2 104.5
"""

escf_psi4, wfn = forte.utils.psi4_scf(geom=geom, basis='6-31G', reference='RHF')

# grab the orbital occupation
doccpi = wfn.doccpi().to_tuple()
soccpi = wfn.soccpi().to_tuple()

print(f'The SCF energy is {escf_psi4} [Eh]')
print(f'SCF doubly occupied orbitals per irrep: {doccpi}')
print(f'SCF singly occupied orbitals per irrep: {soccpi}')
```

```
The SCF energy is -75.98015792193438 [Eh]
SCF doubly occupied orbitals per irrep: (3, 0, 1, 1)
SCF singly occupied orbitals per irrep: (0, 0, 0, 0)
```

2.7.2 Preparing the integral object

To prepare the integrals, we use the helper function `utils.prepare_forte_objects`. We pass the psi4 wave function object (`wfn`) and specify the number of doubly occupied orbitals using the SCF occupation from psi4. Virtual orbitals are automatically determined.

```
mo_spaces={'RESTRICTED_DOCC' : doccpi, 'ACTIVE' : soccpi}
forte_objects = forte.utils.prepare_forte_objects(wfn,mo_spaces)
```

The `forte_objects` returned is a dictionary, and we can access the `ForteIntegral` object using the key `ints`. We store this object in the variable `ints`. We will also use the `MOSpaceInfo` object, which is stored with the key `mo_space_info`.

```
ints = forte_objects['ints']
mo_space_info = forte_objects['mo_space_info']
```

2.7.3 Preparing list of doubly occupied orbitals

From the `MOSpaceInfo` object we can find the list of doubly occupied orbitals

```
rdocc = mo_space_info.corr_absolute_mo('RESTRICTED_DOCC')
print(f'List of doubly occupied orbitals: {rdocc}')
```

```
List of doubly occupied orbitals: [0, 1, 2, 7, 9]
```

2.7.4 Preparing the core blocks of the Hamiltonian

Here we call the functions that return the integrals in the spin orbital basis. We store those in two variables, `h` and `v`.

```
h = forte.spinorbital_oei(ints, rdocc, rdocc)
v = forte.spinorbital_tei(ints, rdocc, rdocc, rdocc, rdocc)

with np.printoptions(precision=2, suppress=True):
    print(h)
```

```
[[-32.98   0.    -0.58   0.    -0.19   0.     0.     0.     0.     0.  ]
 [  0.   -32.98   0.    -0.58   0.    -0.19   0.     0.     0.     0.  ]
 [ -0.58   0.    -7.78   0.    -0.3    0.     0.     0.     0.     0.  ]
 [  0.    -0.58   0.    -7.78   0.    -0.3    0.     0.     0.     0.  ]
 [ -0.19   0.    -0.3    0.    -6.8    0.     0.     0.     0.     0.  ]
 [  0.    -0.19   0.    -0.3    0.    -6.8    0.     0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.    -7.07   0.     0.     0.  ]
 [  0.     0.     0.     0.     0.     0.     0.    -7.07   0.     0.  ]
 [  0.     0.     0.     0.     0.     0.     0.     0.    -6.5    0.  ]
 [  0.     0.     0.     0.     0.     0.     0.     0.     0.    -6.5 ]]
```

2.7.5 Evaluating the energy expression

Here we add the three contributions to the energy and check the SCF energy computed with psi4 and the one recomputed here

```
escf = ints.nuclear_repulsion_energy()
escf += np.einsum('ii->', h)
escf += 0.5 * np.einsum('ijij->', v)

print(f'The SCF energy is {escf_psi4} [Eh] (psi4)')
print(f'The SCF energy is {escf} [Eh] (spin orbital integrals)')
print(f'The difference is {escf_psi4 - escf} [Eh]')
assert math.isclose(escf, escf_psi4)
```

```
The SCF energy is -75.98015792193442 [Eh] (psi4)
The SCF energy is -75.98015792193439 [Eh] (spin orbital integrals)
The difference is -2.8421709430404e-14 [Eh]
```

CHAPTER

THREE

HOWTOS

3.1 Full configuration interaction

3.1.1 Running the test cases

Forte provides test cases for most of all methods implemented. This is a good place to start if you are new to Forte. After compiling and setting up `PYTHONPATH`, you can run the test cases:

```
cd tests/methods
python run_forte_tests_travis.py
```

3.2 ACI: Adaptive Configuration Interaction

3.2.1 Theory

The Adaptive Configuration Interaction Method (ACI) is an iterative selected CI method that optimizes a space of determinants such that the total error in the energy is controlled by a user-defined parameter, σ,

$$|E_{\text{CASCI}} - E_{\text{ACI}}| \approx \sigma.$$

The ACI algorithm grows a set of reference determinants (P) and screens its first-order interacting space using perturbative energy estimates. This screening is done in a cumulative fasion to produce an approximation to the total correlation energy ignored. The space of reference (P) and selected determinants (Q) define the ACI model space (M). The Hamiltonian is diagonalized in this space to produce the ACI energy and wave function,

$$|\Psi_M\rangle = \sum_{\Phi_\mu} C_\mu |\Phi_\mu\rangle.$$

The algorithm proceeds with a pruning step to get a new (P) space to start the next iteration. The iterations end when the ACI energy is satisfactorily converged, which produces a total error that matches $sigma$ very closely. Additionally, the perturbative estimates of the determinants excluded from the model space can be used as a perturbative correction, which we denote as the ACI+PT2 energy.

3.2.2 A Few Practical Notes

- In Forte, ACI wave functions are defined only in the active orbitals.
- The ACI wave function is defined is a set of Slater Determinants, so it is not guaranteed to be a pure spin eigenfunction. As a result, we augment ACI wave functions throughout the procedure to ensure each intermediate space of determinants is spin complete.
- The initial P space is defined from a small CAS wave function so that there are fewer than 1000 determinants. This can be enlarged to improve convergence if needed using the ACTIVE_GUESS_SIZE option.
- This portion of the manual will discuss ACI usage generally, but all content is transferrable to the case where ACI is used as a reference for DSRG computations. If that is the case, the option CAS_TYPE ACI needs to be set.

3.2.3 A First Example

The simplest input for an ACI calculation involves specifying values for σ.

```
import forte

molecule h2o {
0 1
  O
  H 1 0.96
  H 1 0.96 2 104.5
}

set {
    basis sto-3g
    reference rhf
}

set forte {
    job_type aci
    sigma 0.001
}
E_scf, scf_wfn = energy('scf', return_wfn=True)
energy('forte', ref_wfn=scf_wfn)
```

Though not required, it is good practice to also specify the number of roots, multiplicity, symmetry, and charge. The output contains information about the sizes and energies of the P and M spaces at each step of the iteration, and provides a summary of the converged wave function:

```
==> ACI Summary <==

Iterations required:                         3
Dimension of optimized determinant space:    24

* Adaptive-CI Energy Root   0        = -75.012317069484 Eh =   0.0000 eV
* Adaptive-CI Energy Root   0 + EPT2 = -75.013193884201 Eh =   0.0000 eV

==> Wavefunction Information <==
```

(continues on next page)

(continued from previous page)

```
Most important contributions to root   0:
  0  -0.987158 0.974480589          16 |2220220>
  1   0.076700 0.005882905          15 |2220202>
  2  -0.046105 0.002125685          13 |22-+2+->
  3  -0.046105 0.002125685          14 |22+-2-+>
  4   0.044825 0.002009273          12 |2202220>
  5   0.043438 0.001886853          11 |2222200>
  6   0.040971 0.001678638          10 |2200222>
  7   0.033851 0.001145896           9 |22--2++>
  8   0.033851 0.001145896           8 |22++2-->
  9   0.032457 0.001053488           7 |2+-2220>

Spin state for root 0: S^2 = 0.000000, S = 0.000, singlet

==> Computing Coupling Lists <==
--------------------------------
               0.000186 s
               0.000186 s
              0.000333 s
              0.000307 s
              0.000866 s
--------------------------------
1-RDM  took 0.000107 s (determinant)

==> NATURAL ORBITALS <==

      1A1     2.000000      1B1     1.998476      2A1     1.998399
      3A1     1.977478      1B2     1.974442      2B2     0.025891
      4A1     0.025314


RDMS took 0.002290

Adaptive-CI ran in : 0.067389 s
```

For ground state computations, very few additional options are required unless very large determinants spaces are considered. In this case, memory efficient screening and diagonalization algorithms can be chosen.

3.2.4 Computing Excited States with ACI

3.2.5 ACI Options

Basic Options

NROOT

Number of CI roots to find. If energy('aci') is used, energy criteria will be computed for each root with respect to a trial wavefunction. The maximum value among each root will then be used for evaluation with τ_q.

- Type: int
- Default: 1

SELECT_TYPE

Specifies whether second order PT theory energy correction, or first order amplitude is used in selecting the Q space.

- Type: string
- Options: AMP, ENERGY, AIMED_AMP, AIMED_ENERGY
- Default: AMP

TAUP

Threshold used to prune the $P + Q$ space

- Type: double
- Default: 0.01

TAUQ

Threshold used to select the Q space

- Type: double
- Default: 0.000001

Expert Options

DIAG_ALGORITHM

The algorithm used in all diagonalizations. This option is only needed for calculations with very large configuration spaces.

- Type: string
- Options: DAVIDSON, FULL, DAVIDSON_LIST
- Default: DAVIDSON

SMOOTH

This option implements a smoothing function for the Hamiltonian that makes the energy an everywhere-differentiable function of a geometric coordinate by gradually gradually decoupling the determinant of least importance. This function is useful for correcting discontinuities in potential energy curves, but it can yeild non-physical curves if the discontinuities are large.

- Type: bool
- Default: False

SMOOTH_THRESHOLD

The threshold for smoothing the Hamiltonian

- Type: double
- Default: 0.01

EXCITED_ALGORITHM

This option determines the algorithm to compute excited states. Currently the only options implemented are "STATE_AVERAGE" which means that a function of the criteria among the excited states of interest are used to build the configuraiton space, and "ROOT_SELECT" where the determinant space is constructed with respect to a single root.

- Type: string

- Options: “STATE_AVERAGE”, “ROOT_SELECT”
- Default: “STATE_AVERAGE”

PERTURB_SELECT

Option defines τ_q as either MP2 estimate or estimate derived from 2D diagonalization. True uses the MP2 estimation.

- Type: bool
- Default: false

POST_DIAGONALIZE

Option to re-diagonalize Hamiltonian in final CI space. This can be is useful to compute more roots.

- Type: bool
- Default: False

POST_ROOT

Number of roots to compute on post-diagonalization. For this option to be used, post-diagonalize must be true.

- Type: int
- Default: 1

PQ_FUNCTION

Option that selects the function of energy estimates per root and the expansion coefficients per root. This option is only meaningful if more than one root is desired.

- Type: string
- Options: “MAX”, “AVERAGE”
- Default: “MAX”

Q_REFERENCE

Reference state type to be used when computing estimates of the energy difference between two states. The estimation of the change in energy gap a determinant introduces can be done for all excited states with respect to the ground state (GS), or with respect to the nearest, lower state.

- Type: string
- Options: “GS”, “ADJACENT”
- Default: “GS”

Q_REL

Rather than using the absolute energy to define the importance of a determinant, an energy gap between two states can be used. This allows the determinant space to be constructed such that the energy difference between to states is optimized.

- Type: bool
- Default: False

REF_ROOT

Option that selects the desired root that is used to build the determinant space. This option should only be used when the EXCITED_ALGORITHM is set to “ROOT_SELECT”.

- Type: int
- Default: 0

SPIN_TOL

For all of the algorithms in EX_ACI, roots are only used to build determinant spaces if their spin multiplicity is within a given tolerance of the input spin multiplicity. This option defines that spin tolerance. NOTE: the multiplicity must be defined within the EX_ACI scope. For poorly behaved systems, it may be useful to increase this to an arbitrarily large value such that the lowest-energy multiplicities can be confirmed

- Type: double
- Default: 1.0e-4

3.3 MCSCF: Multi-Configuration Self-Consistent Field

3.3.1 Theory

The Multi-Configuration Self-Consistent Field (MCSCF) tries to optimize the orbitals and the CI coefficients for a multi-configuration wave function:

$$|\Psi\rangle = \sum_{I}^{N_{\mathrm{det}}} c_I |\Phi_I\rangle,$$

where c_I is the coefficient for Slater determinant Φ_I. In MCSCF, the molecular orbitals (MOs) are generally separated into three subsets: core ($\mathbf{C}$, doubly occupied), active ($\mathbf{A}$), and virtual ($\mathbf{V}$, unoccuied). The set of determinants are formed by arranging the number of active electrons (i.e., the total number of electrons minus twice the number of core orbitals) in the active orbitals. There are many ways to pick the determinant basis, including complete active space (CAS), restricted active space (RAS), generalized active space (GAS), and other selective configuration interaction schemes (such as ACI).

For convenience, we first introduce the following convention for orbital indices: i, j for core orbitals, t, u, v, w for active orbitals, and a, b for virtual orbitals. General orbitals (core, active, or virtual) are denoted using indices p, q, r, s.

The MCSCF energy can be expressed as

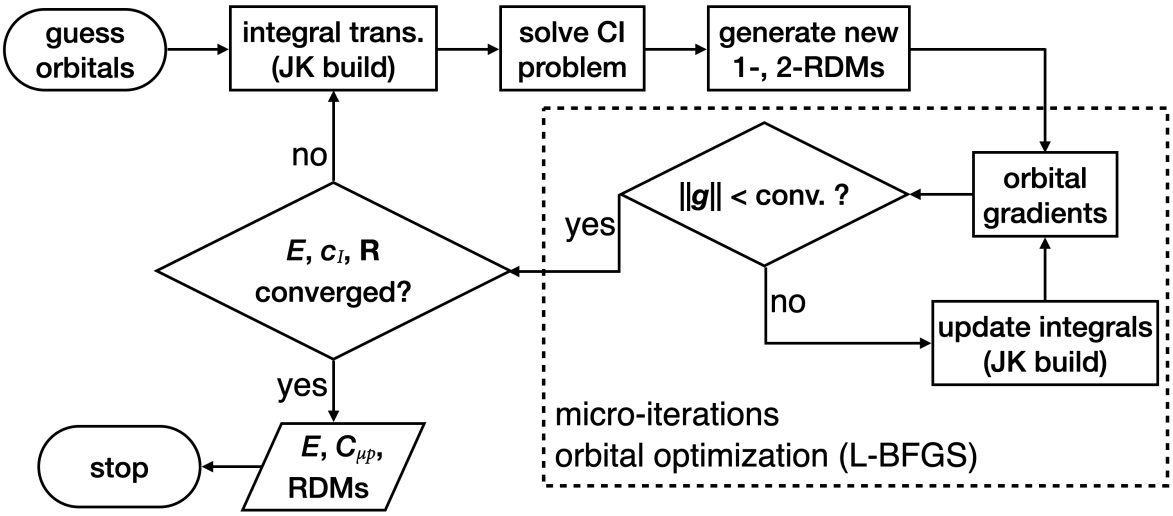
$$E = \sum_{tu}^{\mathbf{A}} f^{\mathrm{c}}_{tu} D_{tu} + \frac{1}{2} \sum_{tuvw}^{\mathbf{A}} (tu|vw) \overline{D}_{tu,vw} + E_{\mathrm{c}} + E_{\mathrm{nuc}},$$

where $f^{\mathrm{c}}_{pq} = h_{pq} + \sum_i^{\mathbf{C}} [2(pq|ii) - (pi|iq)]$ are the closed-shell Fock matrix elements and $(pq|rs)$ are the MO two-electron integrals in chemists' notation. The term $E_{\mathrm{c}} = \sum_j^{\mathbf{C}} (h_{jj} + f^{\mathrm{c}}_{jj})$ is the closed-shell energy and E_{nuc} is the nuclear repulsion energy. We have also used the 1- and 2-body reduced density matrices (RDMs) defined respectively as $D_{tu} = \sum_{IJ} c_I c_J \langle \Phi_I | \hat{E}_{tu} | \Phi_J \rangle$ and $D_{tu,vw} = \sum_{IJ} c_I c_J \langle \Phi_I | \hat{E}_{tu,vw} | \Phi_J \rangle$, where the unitary group generators are defined as $\hat{E}_{tu} = \sum_\sigma^{\uparrow\downarrow} a^\dagger_{t_\sigma} a_{u_\sigma}$ and $\hat{E}_{tu,vw} = \sum_{\sigma\tau}^{\uparrow\downarrow} a^\dagger_{t_\sigma} a_{u_\sigma} a^\dagger_{v_\tau} a_{w_\tau}$. Moreover, we use the symmetrized 2-RDM in the MCSCF energy expression such that it has the same 8-fold symmetry as the two-electron integrals: $\overline{D}_{tu,vw} = \frac{1}{2}(D_{tu,vw} + D_{ut,vw})$.

There are then two sets of parameters in MCSCF: 1) CI coefficients $\{c_I | I = 1, 2, \ldots, N_{\mathrm{det}}\}$, and 2) MO coefficients $\{C_{\mu p} | p = 1, 2, \ldots, N_{\mathrm{MO}}\}$ with $|\phi_p\rangle = \sum_\mu^{\mathrm{AO}} C_{\mu p} |\chi_\mu\rangle$. The goal of MCSCF is then to optimize both sets of parameters to minimize the energy, subject to orthonormal molecular orbitals $|\phi^{\mathrm{new}}_p\rangle = \sum_s |\phi^{\mathrm{old}}_s\rangle U_{sp}$, $\mathbf{U} = \exp(\mathbf{R})$ with $\mathbf{R}^\dagger = -\mathbf{R}$. It is then straightforward to see the two steps in MCSCF: CI optimization (for given orbitals) and orbital optimization (for given RDMs).

3.3.2 Implementation

In Forte, we implement the atomic-orbital-driven two-step MCSCF algorithm based on JK build. We largely follow the article by Hohenstein et al. [J. Chem. Phys. 142, 224103 (2015)] with exceptions on the orbital diagonal Hessian which can be found in Theor. Chem. Acc. 97, 88-95 (1997) (non-redundant rotaions) and J. Chem. Phys. 152, 074102 (2020) (active-active rotaions). The difference is that we improve the orbital optimization step via L-BFGS iterations to obtain a better approxiamtion to the orbital Hessian. The optimization procedure is shown in the following figure:



All types of integrals available in Forte are supported for energy computations.

Note: External integrals read from a FCIDUMP file (`CUSTOM`) are supported, but their use in the current code is very inefficient, which requires further optimization.

Besides MCSCF energies, we have also implement analytic MCSCF energy gradients. Frozen orbitals are allowed for computing both the energy and gradients, although these frozen orbitals must come from canonical Hartree-Fock in order to compute analytic gradients.

Warning: The density-fitted (`DF`, `DISKDF`) and Cholesky-decomposed (`CHOLESKY`) integrals are fully supported for energy computations. However, there is a small discrepancy for gradients between analytic results and finite difference. This is caused by the DF derivative integrals in Psi4.

Meanwhile, analytic gradient calculations are not available for FCIDUMP (`CUSTOM`) integrals.

3.3.3 Input Example

The following performs an MCSCF calculation on CO molecule. Specifically, this is a CASSCF(6,6)/cc-pCVDZ calculation with 2 frozen-core orbitals.

```
import forte

molecule CO{
  0 1
  C
```

(continues on next page)

(continued from previous page)

```
  O  1 1.128
}

set {
  basis                 cc-pcvdz
  reference             rhf
  scf_type              pk
  maxiter               300
  e_convergence         10
  d_convergence         8
  docc                  [5,0,1,1]
}

set forte {
  job_type              mcscf_two_step
  frozen_docc           [2,0,0,0]
  frozen_uocc           [0,0,0,0]
  restricted_docc       [2,0,0,0]
  active                [2,0,2,2]
  e_convergence         8  # energy convergence of the FCI iterations
  r_convergence         8  # residual convergence of the FCI iterations
  casscf_e_convergence  8  # energy convergence of the MCSCF iterations
  casscf_g_convergence  6  # gradient convergence of the MCSCF iterations
  casscf_micro_maxiter  4  # do at least 4 micro iterations per macro iteration
}

Eforte = energy('forte')
```

Near the end of the output, we can find a summary of the MCSCF iterations:

```
==> MCSCF Iteration Summary <==

                  Energy CI                    Energy Orbital
         ----------------------------  ----------------------------
  Iter.        Total Energy       Delta        Total Energy       Delta  Orb. Grad. ␣
↪Micro
  ------------------------------------------------------------------------------------
↪-
     1    -112.799334478817  0.0000e+00   -112.835855509518  0.0000e+00  1.9581e-03     4
     2    -112.843709831147 -4.4375e-02   -112.849267918030 -1.3412e-02  5.8096e-03     4
     3    -112.867656057839 -2.3946e-02   -112.871626476542 -2.2359e-02  5.4580e-03     4
     4    -112.871805690190 -4.1496e-03   -112.871829079776 -2.0260e-04  9.6326e-04     4
     5    -112.871833833468 -2.8143e-05   -112.871834596898 -5.5171e-06  1.0716e-04     4
     6    -112.871834848100 -1.0146e-06   -112.871834858812 -2.6191e-07  1.4395e-05     4
     7    -112.871834862835 -1.4735e-08   -112.871834862936 -4.1231e-09  1.1799e-06     3
     8    -112.871834862954 -1.1940e-10   -112.871834862958 -2.2439e-11  1.4635e-07     2
  ------------------------------------------------------------------------------------
↪-
```

The last column shows the number of micro iterations used in a given macro iteration.

To obtain the analytic energy gradients, just replace the last line of the above input to

```
gradient('forte')
```

The output prints out all the components that contribute to the energy first derivatives:

```
-Nuclear Repulsion Energy 1st Derivatives:
   Atom            X                  Y                  Z
  ------   -----------------  -----------------  -----------------
     1        0.000000000000     0.000000000000    10.563924863908
     2        0.000000000000     0.000000000000   -10.563924863908

-Core Hamiltonian Gradient:
   Atom            X                  Y                  Z
  ------   -----------------  -----------------  -----------------
     1        0.000000000000     0.000000000000   -25.266171481954
     2        0.000000000000     0.000000000000    25.266171481954

-Lagrangian contribution to gradient:
   Atom            X                  Y                  Z
  ------   -----------------  -----------------  -----------------
     1        0.000000000000     0.000000000000     0.763603330124
     2        0.000000000000     0.000000000000    -0.763603330124

-Two-electron contribution to gradient:
   Atom            X                  Y                  Z
  ------   -----------------  -----------------  -----------------
     1        0.000000000000     0.000000000000    13.964810830002
     2        0.000000000000     0.000000000000   -13.964810830002

-Total gradient:
   Atom            X                  Y                  Z
  ------   -----------------  -----------------  -----------------
     1        0.000000000000     0.000000000000     0.026167542081
     2        0.000000000000     0.000000000000    -0.026167542081
```

The `Total gradient` can be compared with that from finite-difference calculations:

```
1     0.00000000000000     0.00000000000000     0.02616749349810
2     0.00000000000000     0.00000000000000    -0.02616749349810
```

obtained from input

```
set findif{
  points 5
}
gradient('forte', dertype=0)
```

Here the difference between finite difference and analytic formalism is 4.8E-8, which is reasonable as our energy only converges to 1.0E-8. Note that only the *total* gradient is available for finite-difference calculations.

The geometry optimization is invoked by

```
optimize('forte')                                    # Psi4 optimization procedure
```

(continues on next page)

(continued from previous page)

```
mol = psi4.core.get_active_molecule()                    # grab the optimized geoemtry
print(mol.to_string(dtype='psi4', units='angstrom'))     # print geometry to screen
```

Assuming the initial geometry is close to the equilibrium, we can also pass the MCSCF converged orbitals of the initial geometry as an initial orbital guess for subsequent geometries along the optimization steps

```
Ecas, ref_wfn = energy('forte', return_wfn=True)         # energy at initial geometry
Eopt = optimize('forte', ref_wfn=ref_wfn)                # Psi4 optimization procedure

mol = psi4.core.get_active_molecule()                    # grab optimized geometry
print(mol.to_string(dtype='psi4', units='angstrom'))     # print geometry to screen
```

Similarly, we can also optimize geometries using finite difference technique:

```
Ecas, ref_wfn = energy('forte', return_wfn=True)         # energy at initial geometry
Eopt = optimize('forte', ref_wfn=ref_wfn, dertype=0)     # Psi4 optimization procedure
```

Warning: After optimization, the input `ref_wfn` no longer holds the data of the initial geometry!

Tip: We could use this code to perform FCI analytic energy gradients (and thus geometry optimizations). The trick is to set all correalted orbitals as active. In test case `casscf-opt-3`, we optimize the geometry of HF molecule at the FCI/3-21G level of theory with frozen 1s orbital of F. Note that frozen orbitals will be kept as they are in the original geometry and therefore the final optimized geometry will be slightly different if a different starting geometry is used.

3.3.4 Options

Basic Options

CASSCF_MAXITER

The maximum number of macro iterations.

- Type: int
- Default: 100

CASSCF_MICRO_MAXITER

The maximum number of micro iterations.

- Type: int
- Default: 40

CASSCF_MICRO_MINITER

The minimum number of micro iterations.

- Type: int
- Default: 6

CASSCF_E_CONVERGENCE

The convergence criterion for the energy (two consecutive energies).

- Type: double
- Default: 1.0e-8

CASSCF_G_CONVERGENCE

The convergence criterion for the orbital gradient (RMS of gradient vector). This value should be roughly in the same order of magnitude as CASSCF_E_CONVERGENCE. For example, given the default energy convergence (1.0e-8), set CASSCF_G_CONVERGENCE to 1.0e-7 – 1.0e-8 for a better convergence behavior.

- Type: double
- Default: 1.0e-7

CASSCF_MAX_ROTATION

The max value allowed in orbital update vector. If a value in the orbital update vector is greater than this number, the update vector will be scaled by this number / max value.

- Type: double
- Default: 0.2

CASSCF_DIIS_START

The iteration number to start DIIS on orbital rotation matrix R. DIIS will not be used if this number is smaller than 1.

- Type: int
- Default: 15

CASSCF_DIIS_MIN_VEC

The minimum number of DIIS vectors allowed for DIIS extrapolation.

- Type: int
- Default: 3

CASSCF_DIIS_MAX_VEC

The maximum number of DIIS vectors, exceeding which the oldest vector will be discarded.

- Type: int
- Default: 8

CASSCF_DIIS_FREQ

How often to do a DIIS extrapolation. For example, 1 means do DIIS every iteration and 2 is for every other iteration, etc.

- Type: int
- Default: 1

CASSCF_CI_SOLVER

Which active space solver to be used.

- Type: string
- Options: CAS, FCI, ACI, PCI
- Default: CAS

CASSCF_DEBUG_PRINTING

Whether to enable debug printing.

- Type: Boolean
- Default: False

CASSCF_FINAL_ORBITAL

What type of orbitals to be used for redundant orbital pairs for a converged calculation.

- Type: string
- Options: CANONICAL, NATURAL, UNSPECIFIED
- Default: CANONICAL

CASSCF_NO_ORBOPT

Turn off orbital optimization procedure if true.

- Type: Boolean
- Default: False

CASSCF_DIE_IF_NOT_CONVERGED

Stop Forte if MCSCF did not converge.

- Type: Boolean
- Default: True

Expert Options

CASSCF_INTERNAL_ROT

Whether to enable pure internal (GASn-GASn) orbital rotations.

- Type: Boolean
- Default: False

CASSCF_ZERO_ROT

Zero the optimization between orbital pairs. Format: [[irrep1, mo1, mo2], [irrep1, mo3, mo4], ...] where irreps are 0-based, while MO indices are 1-based and relative within the irrep. For example, zeroing the mixing of 3A1 and 2A1 translates to [[0, 3, 2]].

- Type: array
- Default: No Default

CASSCF_ACTIVE_FROZEN_ORBITAL

A list of active orbitals to be frozen in the casscf optimization. Active orbitals contain all GAS1, GAS2, ..., GAS6 orbitals. Orbital indices are zero-based and in Pitzer ordering. For example, GAS1 [1,0,0,1]; GAS2 [1,2,2,1]; CASSCF_ACTIVE_FROZEN_ORBITAL [2,6] means we freeze the first A2 orbital in GAS2 and the B2 orbital in GAS1. This option is useful when doing core-excited state computations.

- Type: array
- Default: No Default

CPSCF Options

CPSCF_MAXITER

Max number of iterations for solving coupled perturbed SCF equation

- Type: int
- Default: 50

CPSCF_CONVERGENCE

Convergence criterion for the CP-SCF equation

- Type: double
- Default: 1.0e-8

3.4 Driven Similarity Renormalization Group

Important: Any publication utilizing the DSRG code should acknowledge the following articles:

- F. A. Evangelista, *J. Chem. Phys.* **141**, 054109 (2014).
- C. Li and F. A. Evangelista, *Annu. Rev. Phys. Chem.* **70**, 245-273 (2019).

Depending on the features used, the user is encouraged to cite the corresponding articles listed *here*.

Caution: The examples used in this manual are written based on the spin-integrated code. To make the spin-integrated code work properly for molecules with **even** multiplicities [S * (S + 1) = 2, 4, 6, ...], the user should specify the following keyword:

```
spin_avg_density    true       # use spin-summed reduced density matrices
```

to invoke the use of spin-free densities. The spin-free densities are computed by averaging all spin multiplets (e.g., Ms = 1/2 or -1/2 for doublets). For odd multiplicities [S * (S + 1) = 1, 3, 5, ...], there is no need to do so. Please check test case *dsrg-mrpt2-13* for details.

Note: The latest version of Forte also has the spin-adapted MR-DSRG implemented for DSRG-MRPT2, DSRG-MRPT3, and MR-LDSRG(2) (and its variants). To invoke the spin-adated implementation, the user needs to specify the following keywords:

```
correlation_solver  sa-mrdsrg  # spin-adapted DSRG computation
corr_level          ldsrg2     # spin-adapted theories: PT2, PT3, LDSRG2_QC, LDSRG2
```

The spin-adapted version should be at least 2-3 times faster than the corresponding spin-integrated code, and it also saves some memory. Note that the spin-adapted code will ignore the `spin_avg_density` keyword and always treat it as `true`.

3.4.1 Basics of DSRG

1. Overview of DSRG Theory

Driven similarity renormalization group (DSRG) is a numerically robust approach to treat dynamical (or weak) electron correlation. Specifically, the DSRG performs a *continuous* similarity transformation of the bare Born-Oppenheimer Hamiltonian $\hat{H}$,

$$\bar{H}(s) = e^{-\hat{S}(s)} \hat{H} e^{\hat{S}(s)},$$

where s is the flow parameter defined in the range $[0, +\infty)$. The value of s controls the amount of dynamical correlation included in $\bar{H}(s)$, with $s = 0$ corresponding to no correlation included. The operator $\hat{S}$ can be any operator in general. For example, if $\hat{S} = \hat{T}$ is the coupled cluster substitution operator, the DSRG $\bar{H}(s)$ is identical to coupled-cluster (CC) similarity transformed Hamiltonian except for the s dependence. See *Table I* for different flavours of $\hat{S}$.

Table 1: Table I. Connections of DSRG to CC theories when using different types of $\hat{S}$.

$\hat{S}$	Explanation	CC Theories
$\hat{T}$	cluster operator	traditional CC
$\hat{A}$	$\hat{A} = \hat{T} - \hat{T}^\dagger$	unitary CC, CT
$\hat{G}$	general operator	generalized CC

In the current implementation, we choose the **anti-hermitian** parametrization, i.e., $\hat{S} = \hat{A}$.

The DSRG transformed Hamiltonian $\bar{H}(s)$ contains many-body (> 2-body) interactions in general. We can express it as

$$\bar{H} = \bar{h}_0 + \bar{h}^p_q \{a^q_p\} + \frac{1}{4} \bar{h}^{pq}_{rs} \{a^{rs}_{pq}\} + \frac{1}{36} \bar{h}^{pqr}_{stu} \{a^{stu}_{pqr}\} + ...$$

where $a^{pq\cdots}_{rs\cdots} = a^\dagger_p a^\dagger_q \dots a_s a_r$ is a string of creation and annihilation operators and $\{\cdot\}$ represents normal-ordered operators. In particular, we use Mukherjee-Kutzelnigg normal ordering [see J. Chem. Phys. 107, 432 (1997)] with respect to a general multideterminantal reference Ψ_0. Here we also assume summations over repeated indices for brevity. Also note that $\bar{h}_0$ is the energy dressed by dynamical correlation effects.

In DSRG, we require the off-diagonal components of $\bar{H}$ gradually go to zero (from $\hat{H}$) as s grows (from 0). By off-diagonal components, we mean $\bar{h}^{ij\cdots}_{ab\cdots}$ and $\bar{h}^{ab\cdots}_{ij\cdots}$ where $i, j, \dots$ indicates hole orbitals and $a, b, \dots$ labels particle orbitals. There are in principle infinite numbers of ways to achieve this requirement. The current implementation chooses the following parametrization,

$$\bar{h}^{ij\cdots}_{ab\cdots} = [\bar{h}^{ij\cdots}_{ab\cdots} + \Delta^{ij\cdots}_{ab\cdots} t^{ij\cdots}_{ab\cdots}] e^{-s(\Delta^{ij\cdots}_{ab\cdots})^2},$$

where $\Delta^{ij\cdots}_{ab\cdots} = \epsilon_i + \epsilon_j + \cdots - \epsilon_a - \epsilon_b - \dots$ is the Møller-Plesset denominator defined by orbital energies ϵ_p and $t^{ij\cdots}_{ab\cdots}$ are the cluster amplitudes. This equation is called the DSRG flow equation, which suggests a way how the off-diagonal Hamiltonian components evolves as s changes. We can now solve for the cluster amplitudes since $\bar{H}$ is a function of $\hat{T}$ using the Baker–Campbell–Hausdorff (BCH) formula.

Since we choose $\hat{S} = \hat{A}$, the corresponding BCH expansion is thus non-terminating. Approximations have to be introduced and different treatments to $\bar{H}$ leads to various levels of DSRG theories. Generally, we can treat it either in a perturbative or non-perturbative manner. For non-perturbative theories, the **only** widely tested scheme so far is the recursive single commutator (RSC) approach, where every single commutator is truncated to contain at most two-body contributions for a nested commutator. For example, a doubly nested commutator is computed as

$$\frac{1}{2}[[\hat{H}, \hat{A}], \hat{A}] \approx \frac{1}{2}[[\hat{H}, \hat{A}]_{1,2}, \hat{A}]_{0,1,2},$$

where 0, 1, 2 indicate scalar, 1-body, and 2-body contributions. We term the DSRG method that uses RSC as LDSRG(2).

Alternatively, we can perform a perturbative analysis on the **approximated** BCH equation of $\bar{H}$ and obtain various DSRG perturbation theories [e.g., 2nd-order (PT2) or 3rd-order (PT3)]. Note we use the RSC approximated BCH equation for computational cost considerations. As such, the implemented DSRG-PT3 is **not** a formally complete PT3, but a numerically efficient companion theory to the LDSRG(2) method.

To conclude this subsection, we discuss the computational cost and current implementation limit, which are summarized in *Table II*.

Table 2: Table II. Cost and maximum system size for the DSRG methods implemented in Forte.

Method	Computational Cost	Conventional 2-el. integrals	Density-fitted/Cholesky (DF/CD)
PT2	one-shot N^5	$\sim$ 250 basis functions	$\sim$ 1800 basis functions
PT3	one-shot N^6	$\sim$ 250 basis functions	$\sim$ 700 basis functions
LDSRG(2)	iterative N^6	$\sim$ 200 basis functions	$\sim$ 550 basis functions

2. Input Examples

Minimal Example - DSRG-MPT2 energy of HF

Let us first see an example with minimal keywords. In particular, we compute the energy of hydrogen fluoride using DSRG multireference (MR) PT2 using a complete active space self-consistent field (CASSCF) reference.

```
import forte

molecule mol{
  0 1
  F
  H  1 R
}
mol.R = 1.50  # this is a neat way to specify H-F bond lengths

set globals{
   basis                  cc-pvdz
   reference              rhf
   scf_type               pk
   d_convergence          8
   e_convergence          10
   restricted_docc        [2,0,1,1]
   active                 [2,0,0,0]
}

set forte{
   active_space_solver    fci
   correlation_solver     dsrg-mrpt2
   dsrg_s                 0.5
   frozen_docc            [1,0,0,0]
   restricted_docc        [1,0,1,1]
   active                 [2,0,0,0]
}
```

(continues on next page)

(continued from previous page)

```
Emcscf, wfn = energy('casscf', return_wfn=True)
energy('forte', ref_wfn=wfn)
```

There are three blocks in the input:

1. The `molecule` block specifies the geometry, charge, multiplicity, etc.
2. The second block specifies Psi4 options (see Psi4 manual for details).
3. The last block shows options specifically for Forte.

In this example, we use Psi4 to compute CASSCF reference. Psi4 provides the freedom to specify the core (a.k.a. internal) and active orbitals using `RESTRICTED_DOCC` and `ACTIVE` options, but *it is generally the user's responsibility to select and verify correct orbital ordering*. The `RESTRICTED_DOCC` array `[2,0,1,1]` indicates two a_1, zero a_2, one b_1, and one b_2 doubly occupied orbitals. There are four irreps because the computation is performed using C_{2v} point group symmetry.

The computation begins with the execution of Psi4's CASSCF code, invoked by `Emcscf, wfn = energy('casscf', return_wfn=True)`. This function call returns the energy and CASSCF wave function. In the second call to the energy function, `energy('forte', ref_wfn=wfn)`, we ask the Psi4 driver to call Forte. The wave function stored in `wfn` will is passed to Forte via argument `ref_wfn`.

Forte generally recomputes the reference using the provided wave function parameters. To perform a DSRG computation, the user is expected to specify the following keywords:

- `ACTIVE_SPACE_SOLVER`: Here we use `FCI` to perform a CAS configuration interaction (CASCI), i.e., a full CI within the active orbitals.
- `CORRELATION_SOLVER`: This option determines which code to run. The four well-tested DSRG solvers are: `DSRG-MRPT2`, `THREE-DSRG-MRPT2`, `DSRG-MRPT3`, and `MRDSRG`. The density-fitted DSRG-MRPT2 is implemented in `THREE-DSRG-MRPT2`. The `MRDSRG` is mainly designed to perform MR-LDSRG(2) computations.
- `DSRG_S`: This keyword specifies the DSRG flow parameter in a.u. For general MR-DSRG computations, the user should change the value to $0.5 \sim 1.5$ a.u. Most of our computations in *References* are performed using 0.5 or 1.0 a.u.

  Caution: By default, `DSRG_S` is set to 0.5 a.u. The user should always set this keyword by hand! Non-perturbative methods may not converge for large values of flow parameter.

- Orbital spaces: Here we also specify frozen core orbitals besides core and active orbitals. Note that in this example, we optimize the 1s-like core orbital in CASSCF but later freeze it in the DSRG treatments of dynamical correlation. Details regarding to orbital spaces can be found in the section sec:mospaceinfo.

  Tip: To perform a single-reference (SR) DSRG computation, set the array `ACTIVE` to zero. In the above example, the SR DSRG-PT2 energy can be obtained by modifying `RESTRICTED_DOCC` to `[2,0,1,1]` and `ACTIVE` to `[0,0,0,0]`. The MP2 energy can be reproduced if we further change `DSRG_S` to very large values (e.g., 10^8 a.u.).

The output of the above example consists of several parts:

- The active-space FCI computation:

```
==> Root No. 0 <==

  20     -0.95086442
  02      0.29288371
```

(continues on next page)

(continued from previous page)

```
  Total Energy:       -99.939316382616340

==> Energy Summary <==

  Multi.  Irrep.  No.               Energy
  ----------------------------------------
     1      A1     0       -99.939316382616
  ----------------------------------------
```

Forte prints out the largest determinants in the CASCI wave function and its energy. Since we read orbitals from Psi4's CASSCF, this energy should coincide with Psi4's CASSCF energy.

- The computation of 1-, 2-, and 3-body reduced density matrices (RDMs) of the CASCI reference:

```
==> Computing RDMs for Root No. 0 <==

  Timing for 1-RDM: 0.000 s
  Timing for 2-RDM: 0.000 s
  Timing for 3-RDM: 0.000 s
```

- Canonicalization of the orbitals:

```
==> Checking Fock Matrix Diagonal Blocks <==

  Off-Diag. Elements       Max           2-Norm
  -------------------------------------------------
  Fa actv              0.0000000000   0.0000000000
  Fb actv              0.0000000000   0.0000000000
  -------------------------------------------------
  Fa core              0.0000000000   0.0000000000
  Fb core              0.0000000000   0.0000000000
  -------------------------------------------------
  Fa virt              0.0000000000   0.0000000000
  Fb virt              0.0000000000   0.0000000000
  -------------------------------------------------
Orbitals are already semicanonicalized.
```

All DSRG procedures require the orbitals to be canonicalized. In this basis, the core, active, and virtual diagonal blocks of the average Fock matrix are diagonal. Forte will test if the orbitals provided are canonical, and if not it will perform a canonicalization. In this example, since Psi4's CASSCF orbitals are already canonical, Forte just tests the Fock matrix but does not perform an actual orbital rotation.

- Computation of the DSRG-MRPT2 energy:
  - The output first prints out a summary of several largest amplitudes and possible intruders:

```
==> Excitation Amplitudes Summary <==

Active Indices:    1    2
...  # ommit output for T1 alpha, T1 beta, T2 alpha-alpha, T2 beta-beta
Largest T2 amplitudes for spin case AB:
       _       _               _       _               _       _
   i   j   a   b           i   j   a   b           i   j   a   b
```

(continues on next page)

(continued from previous page)

```
-------------------------------------------------------------------------------
[  1   2   2   4] 0.055381 [  0   0   1   1]-0.053806 [  1   2   1   4] 0.048919
[  1  14   1  15] 0.047592 [  1  10   1  11] 0.047592 [  2   2   4   4]-0.044138
[  2  14   1  15] 0.042704 [  2  10   1  11] 0.042704 [  1  10   1  12]-0.040985
[  1  14   1  16]-0.040985 [  2   2   1   4] 0.040794 [  1   1   1   5] 0.040479
[  1  14   2  15] 0.036004 [  1  10   2  11] 0.036004 [  2  10   2  12]-0.035392
-------------------------------------------------------------------------------
Norm of T2AB vector: (nonzero elements: 1487)                0.369082532477979.
-------------------------------------------------------------------------------
```

Here, {i, j} are generalized hole indices and {a, b} indicate generalized particle indices. The active indices are given at the beginning of this printing block. Thus, the largest amplitude in this case [(1,2) -> (2,4)] is a semi-internal excitation from (active, active) to (active, virtual). In general, semi-internal excitations tend to be large and they are suppressed by DSRG.

- An energy summary is given later in the output:

```
==> DSRG-MRPT2 Energy Summary <==

  E0 (reference)                 =    -99.939316382616383
  <[F, T1]>                      =     -0.010942204196708
  <[F, T2]>                      =      0.011247157867728
  <[V, T1]>                      =      0.010183611834684
  <[V, T2]> (C_2)^4              =     -0.213259856801491
  <[V, T2]> C_4 (C_2)^2 HH       =      0.002713363798054
  <[V, T2]> C_4 (C_2)^2 PP       =      0.012979097502477
  <[V, T2]> C_4 (C_2)^2 PH       =      0.027792466274407
  <[V, T2]> C_6 C_2              =     -0.003202673882957
  <[V, T2]>                      =     -0.172977603109510
  DSRG-MRPT2 correlation energy  =     -0.162489037603806
  DSRG-MRPT2 total energy        =   -100.101805420220188
  max(T1)                        =      0.097879100308377
  max(T2)                        =      0.055380911136950
  ||T1||                         =      0.170534584213259
  ||T2||                         =      0.886328961933259
```

Here we show all contributions to the energy. Specifically, those labeled by C_4 involve 2-body density cumulants, and those labeled by C_6 involve 3-body cumulants.

A More Advanced Example - MR-LDSRG(2) energy of HF

Here we look at a more advanced example of MR-LDSRG(2) using the same molecule.

```
# We just show the input block of Forte here.
# The remaining input is identical to the previous example.

set forte{
   active_space_solver     fci
   correlation_solver      mrdsrg
   corr_level              ldsrg2
   frozen_docc             [1,0,0,0]
   restricted_docc         [1,0,1,1]
   active                  [2,0,0,0]
   dsrg_s                  0.5
```

(continues on next page)

(continued from previous page)

```
   e_convergence           1.0e-8
   dsrg_rsc_threshold      1.0e-9
   relax_ref               iterate
}
```

Warning: This example takes a long time to finish (~3 min on a 2018 15-inch MacBook Pro).

There are several things to notice.

1. To run a MR-LDSRG(2) computation, we need to change `CORRELATION_SOLVER` to `MRDSRG`. Additionally, the `CORR_LEVEL` should be specified as `LDSRG2`. There are other choices of `CORR_LEVEL` but they are mainly for testing new ideas.
2. We specify the energy convergence keyword `E_CONVERGENCE` and the RSC threshold `DSRG_RSC_THRESHOLD`, which controls the truncation of the recursive single commutator (RSC) approximation of the DSRG Hamiltonian. In general, the value of `DSRG_RSC_THRESHOLD` should be smaller than that of `E_CONVERGENCE`. Making `DSRG_RSC_THRESHOLD` larger will stop the BCH series earlier and thus saves some time. It is OK to leave `DSRG_RSC_THRESHOLD` as the default value, which is 10^{-12} a.u.
3. The MR-LDSRG(2) method includes reference relaxation effects. There are several variants of reference relaxation levels (see *Theoretical Variants and Technical Details*). Here we use the fully relaxed version, which is done by setting `RELAX_REF` to `ITERATE`.

Note: The reference relaxation procedure is performed in a tick-tock way (see *Theoretical Variants and Technical Details*), by alternating the solution of the DSRG amplitude equations and the diagonalization of the DSRG Hamiltonian. This procedure may not monotonically converge and is potentially numerically unstable. We therefore suggest using a moderate energy threshold ($\geq 10^{-8}$ a.u.) for the iterative reference relaxation, which is controlled by the option `RELAX_E_CONVERGENCE`.

For a given reference wave function, the output prints out a summary of:

1. The iterations for solving the amplitudes, where each step involves building a DSRG transformed Hamiltonian.
2. The MR-LDSRG(2) energy:

```
==> MR-LDSRG(2) Energy Summary <==

  E0 (reference)                 =     -99.939316382616383
  MR-LDSRG(2) correlation energy =      -0.171613035562048
  MR-LDSRG(2) total energy       =    -100.110929418178429
```

3. The MR-LDSRG(2) converged amplitudes:

```
==> Final Excitation Amplitudes Summary <==

  Active Indices:    1    2
  ...  # ommit output for T1 alpha, T1 beta, T2 alpha-alpha, T2 beta-beta
  Largest T2 amplitudes for spin case AB:
       _       _               _       _               _       _
     i   j   a   b             i   j   a   b             i   j   a   b
  ------------------------------------------------------------------------------
  [  0   0   1   1]-0.060059 [  1   2   2   4] 0.046578 [  1  10   1  11] 0.039502
```

(continues on next page)

(continued from previous page)

```
[  1  14   1  15] 0.039502 [  0   0   1   2]-0.038678 [  1   1   1   5] 0.037546
[  2   2   4   4]-0.033871 [  1   2   1   4] 0.033125 [  1  14   2  15] 0.032868
[  1  10   2  11] 0.032868 [  1  10   1  12]-0.032602 [  1  14   1  16]-0.032602
[ 14  14  15  15]-0.030255 [ 10  10  11  11]-0.030255 [  2  14   1  15] 0.029241
-------------------------------------------------------------------------------
Norm of T2AB vector: (nonzero elements: 1487)                0.330204946109119.
-------------------------------------------------------------------------------
```

At the end of the computation, Forte prints a summary of the energy during the reference relaxation iterations:

```
=> MRDSRG Reference Relaxation Energy Summary <=

                     Fixed Ref. (a.u.)                 Relaxed Ref. (a.u.)
        ----------------------------------  ----------------------------------
Iter.           Total Energy          Delta          Total Energy          Delta
-------------------------------------------------------------------------------
    1     -100.110929418178 (a) -1.001e+02     -100.114343552853 (b) -1.001e+02
    2     -100.113565563124 (c) -2.636e-03     -100.113571036112      7.725e-04
    3     -100.113534597590      3.097e-05     -100.113534603824      3.643e-05
    4     -100.113533334887      1.263e-06     -100.113533334895      1.269e-06
    5     -100.113533290863      4.402e-08     -100.113533290864      4.403e-08
    6     -100.113533289341      1.522e-09     -100.113533289341 (d)  1.522e-09
-------------------------------------------------------------------------------
```

Let us introduce the nomenclature for reference relaxation.

Name	Example Value	Description
a) Unrelaxed	-100.110929418178	1st iter.; fixed CASCI ref.
b) Partially Relaxed	-100.114343552853	1st iter.; relaxed CASCI ref.
c) Relaxed	-100.113565563124	2nd iter.; fixed ref.
d) Fully Relaxed	-100.113533289341	last iter.; relaxed ref.

The unrelaxed energy is a diagonalize-then-perturb scheme, while the partially relaxed energy corresponds to a diagonalize-then-perturb-then-diagonalize method. In this example, the fully relaxed energy is well reproduced by the relaxed energy with a small error ($< 10^{-4}$ a.u.).

Other Examples

There are plenty of examples in the tests/method folder. A complete list of the DSRG test cases can be found *here*.

3. General DSRG Options

CORR_LEVEL

Correlation level of MR-DSRG.

- Type: string
- Options: PT2, PT3, LDSRG2, LDSRG2_QC, LSRG2, SRG_PT2, QDSRG2
- Default: PT2

DSRG_S

The value of the flow parameter s.

- Type: double
- Default: 0.5

DSRG_MAXITER

Max iterations for MR-DSRG amplitudes update.

- Type: integer
- Default: 50

DSRG_RSC_NCOMM

The maximum number of commutators in the recursive single commutator approximation to the BCH formula.

- Type: integer
- Default: 20

DSRG_RSC_THRESHOLD

The threshold of considering the BCH expansion converged based on the recursive single commutator approximation.

- Type: double
- Default: 1.0e-12

R_CONVERGENCE

The convergence criteria for the amplitudes.

- Type: double
- Default: 1.0e-6

NTAMP

The number of largest amplitudes printed in the amplitudes summary.

- Type: integer
- Default: 15

INTRUDER_TAMP

A threshold for amplitudes that are considered as intruders for printing.

- Type: double
- Default: 0.1

TAYLOR_THRESHOLD

A threshold for small energy denominators that are computed using Taylor expansion (instead of direct reciprocal of the energy denominator). For example, 3 means Taylor expansion is performed if denominators are smaller than 1.0e-3.

- Type: integer
- Default: 3

DSRG_DIIS_START

The minimum iteration to start storing DIIS vectors for MRDSRG amplitudes. Any number smaller than 1 will turn off the DIIS procedure.

- Type: int
- Default: 2

DSRG_DIIS_FREQ

How often to do a DIIS extrapolation in MRDSRG iterations. For example, 1 means do DIIS every iteration and 2 is for every other iteration, etc.

- Type: int
- Default: 1

DSRG_DIIS_MIN_VEC

Minimum number of error vectors stored for DIIS extrapolation in MRDSRG.

- Type: int
- Default: 3

DSRG_DIIS_MAX_VEC

Maximum number of error vectors stored for DIIS extrapolation in MRDSRG.

- Type: int
- Default: 8

3.4.2 Theoretical Variants and Technical Details

1. Reference Relaxation

For MR methods, it is necessary to consider reference relaxation effects due to coupling between static and dynamical correlation. This can be introduced by requiring the reference wave function, Ψ_0 to be the eigenfunction of $\bar{H}(s)$. The current implementation uses the uncoupled two-step (tick-tock) approach, where the DSRG transformed Hamiltonian $\bar{H}(s)$ is built using the RDMs of a given Ψ_0, and then diagonalize $\bar{H}(s)$ within the active space yielding a new Ψ_0. These two steps can be iteratively performed until convergence.

Denoting the i-th iteration of reference relaxation by superscript $[i]$, the variants of reference relaxation procedure introduced above can be expressed as

Name	Energy Expression
Unrelaxed	$\langle \Psi_0^{[0]} \vert \bar{H}^{[0]}(s) \vert \Psi_0^{[0]} \rangle$
Partially Relaxed	$\langle \Psi_0^{[1]}(s) \vert \bar{H}^{[0]}(s) \vert \Psi_0^{[1]}(s) \rangle$
Relaxed	$\langle \Psi_0^{[1]}(s) \vert \bar{H}^{[1]}(s) \vert \Psi_0^{[1]}(s) \rangle$
Fully Relaxed	$\langle \Psi_0^{[n]}(s) \vert \bar{H}^{[n]}(s) \vert \Psi_0^{[n]}(s) \rangle$

where $[0]$ uses the original reference wave function and $[n]$ suggests converged results.

By default, `MRDSRG` only performs an unrelaxed computation. To obtain partially relaxed energy, the user needs to change `RELAX_REF` to `ONCE`. For relaxed energy, `RELAX_REF` should be switched to `TWICE`. For fully relaxed energy, `RELAX_REF` should be set to `ITERATE`.

For other DSRG solvers aimed for perturbation theories, only the unrelaxed and partially relaxed energies are available. In the literature, we term the partially relaxed version as the default DSRG-MRPT, while the unrelaxed version as uDSRG-MRPT.

Tip: These energies can be conveniently obtained in the input file. For example, `Eu = variable("UNRELAXED ENERGY")` puts unrelaxed energy to a variable `Eu`. The avaible keys are `"UNRELAXED ENERGY"`, `PARTIALLY RELAXED ENERGY`, `"RELAXED ENERGY"`, and `"FULLY RELAXED ENERGY"`.

---

2. Orbital Rotations

The DSRG equations are defined in the semicanonical orbital basis, and thus it is not generally orbital invariant. All DSRG solvers, except for `THREE-DSRG-MRPT2`, automatically rotates the integrals to semicanonical basis even if the input integrals are not canonicalized (if keyword `SEMI_CANONICAL` is set to `FALSE`). However, it is recommended a careful inspection to the printings regarding to the semicanonical orbitals. An example printing of orbital canonicalization can be found in *Minimal Example*.

3. Sequential Transformation

In the sequential transformation ansatz, we compute $\bar{H}$ sequentially as

$$\bar{H}(s) = e^{-\hat{A}_n} \cdots e^{-\hat{A}_2} e^{-\hat{A}_1} \hat{H} e^{\hat{A}_1} e^{\hat{A}_2} \cdots e^{\hat{A}_n}$$

instead of the traditional approach:

$$\bar{H}(s) = e^{-\hat{A}_1 - \hat{A}_2 - \cdots - \hat{A}_n} \hat{H} e^{\hat{A}_1 + \hat{A}_2 + \cdots + \hat{A}_n}$$

For clarity, we ignore the indication of s dependence on $\bar{H}(s)$ and $\hat{A}(s)$. In the limit of $s \to \infty$ and no truncation of $\hat{A}(s)$, both the traditional and sequential MR-DSRG methods can approach the full configuration interaction limit. The difference between their truncated results are also usually small.

In the sequential approach, $e^{-\hat{A}_1} \hat{H} e^{\hat{A}_1}$ is computed as a unitary transformation to the bare Hamiltonian, which is very efficient when combined with integral factorization techniques (scaling reduction).

4. Non-Interacting Virtual Orbital Approximation

In the non-interacting virtual orbital (NIVO) approximation, we neglect the operator components of all rank-4 intermediate tensors and $\bar{H}$ with three or more virtual orbital indices ($\mathbf{VVVV}$, $\mathbf{VCVV}$, $\mathbf{VVVA}$, etc.). Consequently, the number of elements in the intermediates are reduced from $\mathcal{O}(N^4)$ to $\mathcal{O}(N^2 N_{\mathbf{H}}^2)$, which is of similar size to the $\hat{T}_2$ amplitudes. As such, the memory requirement of MR-LDSRG(2) is significantly reduced when we apply NIVO approximation and combine with integral factorization techniques with a batched algorithm for tensor contractions.

Since much less number of tensor elements are involved, NIVO approximation dramatically reduces computation time. However, the overall time scaling of MR-LDSRG(2) remain unchanged (prefactor reduction). The error introduced by the NIVO approximation is usually negligible.

Note: If conventional two-electron integrals are used, NIVO starts from the bare Hamiltonian term (i.e., $\hat{H}$ and all the commutators in the BCH expansion of $\bar{H}$ are approximated). For DF or CD intregrals, however, NIVO will start from the first commutator $[\hat{H}, \hat{A}]$.

5. Zeroth-order Hamiltonian of DSRG-MRPT2 in MRDSRG Class

DSRG-MRPT2 is also implemented in the MRDSRG class for testing other zeroth-order Hamiltonian. The general equation for all choices is to compute the summed second-order Hamiltonian:

$$\bar{H}^{[2]} = \hat{H} + [\hat{H}, \hat{A}^{(1)}] + [\hat{H}^{(0)}, \hat{A}^{(2)}] + \frac{1}{2}[[\hat{H}^{(0)}, \hat{A}^{(1)}], \hat{A}^{(1)}]$$

where for brevity the (s) notation is ignored and the superscripts of parentheses indicate the orders of perturbation. We have implemented the following choices for the zeroth-order Hamiltonian.

Diagonal Fock operator (Fdiag)

This choice contains the three diagonal blocks of the Fock matrix, that is, core-core, active-active, and virtual-virtual. Due to its simplicity, $\bar{H}^{[2]}$ can be obtained in a non-iterative manner in the semicanonical basis.

Fock operator (Ffull)

This choice contains all the blocks of the Fock matrix. Since Fock matrix contains non-diagonal contributions, $[\hat{H}^{(0)}, \hat{A}^{(2)}]$ can contribute to the energy. As such, both first- and second-order amplitudes are solved iteratively.

Dyall Hamiltonian (Fdiag_Vactv)

This choice contains the diagonal Fock matrix and the part of V labeled only by active indices. We solve the first-order amplitudes iteratively. However, $[\hat{H}^{(0)}, \hat{A}]$ will neither contribute to the energy nor the active part of the $\bar{H}^{[2]}$.

Fink Hamiltonian (Fdiag_Vdiag)

This choice contains all the blocks of Dyall Hamiltonian plus other parts of V that do not change the excitation level. For example, these additional blocks include: cccc, aaaa, vvvv, caca, caac, acac, acca, cvcv, cvvc, vcvc, vccv, avav, avva, vava, and vaav. The computation procedure is similar to that of Dyall Hamiltonian.

To use different types of zeroth-order Hamiltonian, the following options are needed

```
correlation_solver      mrdsrg
corr_level              pt2
dsrg_pt2_h0th           Ffull
```

Warning: The implementation of DSRG-MRPT2 in `correlation_solver mrdsrg` is different from the one in `correlation_solver dsrg-mrpt2`. For the latter, the $\hat{H}^{(0)}$ is **assumed** being Fdiag and diagonal such that $[\hat{H}^{(0)}, \hat{A}^{(1)}]$ can be written in a compact form using semicanonical orbital energies. For `mrdsrg`, $[\hat{H}^{(0)}, \hat{A}^{(1)}]$ is evaluated without any assumption to the form of $\hat{H}^{(0)}$. These two approaches are equivalent for DSRG based on a CASCI reference.

However, they will give different energies when there are multiple GAS spaces (In DSRG, all GAS orbitals are treated as ACTIVE). In this case, semicanonical orbitals are defined as those that make the diagonal blocks of the Fock matrix diagonal: core-core, virtual-virtual, GAS1-GAS1, GAS2-GAS2, ..., GAS6-GAS6. Then it is

equivalent to say that `dsrg-mrpt2` uses all the diagonal blocks of the Fock matrix as zeroth-order Hamiltonian. In order to correctly treat the GAS m - GAS n ($m \neq n$) part of Fock matrix as first-order Hamiltonian, one need to invoke internal excitations (i.e., active-active excitations). Contrarily, `mrdsrg` takes the entire active-active block of Fock matrix as zeroth-order Hamiltonian, that is all blocks of GAS m - GAS n ($m, n \in \{1, 2, \cdots, 6\}$).

The spin-adapted code `correlation_solver sa-mrdsrg` with `corr_level pt2` has the same behavior to the `dsrg-mrpt2` implementaion.

6. Restart iterative MRDSRG from a previous computation

The convergence of iterative MRDSRG [e.g., MR-LDSRG(2)] can be greatly improved if it starts from good initial guesses (e.g., from loosely converged amplitudes or those of a near-by geometry). The amplitudes can be dumped to the current working directory on disk for later use by turning on the `DSRG_DUMP_AMPS` keyword. These amplitudes are stored in a binary file using Ambit (version later than 06/30/2020). For example, T1 amplitudes are stored as `forte.mrdsrg.spin.t1.bin` for the spin-integrated code and `forte.mrdsrg.adapted.t1.bin` for spin-adapted code (i.e., *correlation_solver* set to *sa-mrdsrg*). To read amplitudes in the current directory (must follow the same file name convention), the user needs to invoke the `DSRG_READ_AMPS` keyword.

Note: In general, we should make sure the orbital phases are consistent between reading and writing amplitudes. For example, the following shows part of the input to ensure the coefficient of the first AO being positive for all MOs.

```
...
Escf, wfn = energy('scf', return_wfn=True)

# fix orbital phase
Ca = wfn.Ca().clone()
nirrep = wfn.nirrep()
rowdim, coldim = Ca.rowdim(), Ca.coldim()
for h in range(nirrep):
    for i in range(coldim[h]):
        v = Ca.get(h, 0, i)
        if v < 0:
            for j in range(rowdim[h]):
                Ca.set(h, j, i, -1.0 * Ca.get(h, j, i))
wfn.Ca().copy(Ca)

energy('forte', ref_wfn=wfn)
```

For reference relaxation, initial amplitudes are obtained from the previous converged values by default. To turn this feature off (not recommended), please set `DSRG_RESTART_AMPS` to `False`.

7. Examples

Here we slightly modify the more advanced example in *General DSRG Examples* to adopt the sequential transformation and NIVO approximation.

```
# We just show the input block of Forte here.

set forte{
   active_space_solver     fci
   correlation_solver      mrdsrg
   corr_level              ldsrg2
   frozen_docc             [1,0,0,0]
   restricted_docc         [1,0,1,1]
   active                  [2,0,0,0]
   dsrg_s                  0.5
   e_convergence           1.0e-8
   dsrg_rsc_threshold      1.0e-9
   relax_ref               iterate
   dsrg_nivo               true
   dsrg_hbar_seq           true
}
```

Note: Since the test case is very small, invoking these two keywords does not make the computation faster. A significant speed improvement can be observed for a decent amout of basis functions (~ 100).

8. Related Options

RELAX_REF

Different approaches for MR-DSRG reference relaxation.

- Type: string
- Options: NONE, ONCE, TWICE, ITERATE
- Default: NONE

RELAX_E_CONVERGENCE

The energy convergence criteria for MR-DSRG reference relaxation.

- Type: double
- Default: 1.0e-8

MAXITER_RELAX_REF

Max macro iterations for MR-DSRG reference relaxation.

- Type: integer
- Default: 15

DSRG_DUMP_RELAXED_ENERGIES

Dump the energies after each reference relaxation step to JSON. The energies include all computed states and the averaged DSRG "Fixed" and "Relaxed" energies for every reference relaxation step.

- Type: Boolean

- Default: False

DSRG_RESTART_AMPS

Use converged amplitudes from the previous step as initial guesses of the current amplitudes.

- Type: Boolean
- Default: True

SEMI_CANONICAL

Semicanonicalize orbitals after solving the active-space eigenvalue problem.

- Type: Boolean
- Default: True

DSRG_HBAR_SEQ

Apply the sequential transformation algorithm in evaluating the transformed Hamiltonian $\bar{H}(s)$, i.e.,

$$\bar{H}(s) = e^{-\hat{A}_n(s)} \cdots e^{-\hat{A}_2(s)} e^{-\hat{A}_1(s)} \hat{H} e^{\hat{A}_1(s)} e^{\hat{A}_2(s)} \cdots e^{\hat{A}_n(s)}.$$

- Type: Boolean
- Default: False

DSRG_NIVO

Apply non-interacting virtual orbital (NIVO) approximation in evaluating the transformed Hamiltonian.

- Type: Boolean
- Default: False

DSRG_PT2_H0TH

The zeroth-order Hamiltonian used in the MRDSRG code for computing DSRG-MRPT2 energy.

- Type: string
- Options: FDIAG, FFULL, FDIAG_VACTV, FDIAG_VDIAG
- Default: FDIAG

DSRG_DUMP_AMPS

Dump amplitudes to the current directory for a MRDSRG method. File names for T1 and T2 amplitudes are `forte.mrdsrg.CODE.t1.bin` and `forte.mrdsrg.CODE.t2.bin`, respectively. Here, `CODE` will be `adapted` if using the spin-adapted implementation, while `spin` if using the spin-integrated code.

- Type: Boolean
- Default: False

DSRG_READ_AMPS

Read amplitudes from the current directory for iterative MRDSRG methods. File format and content should match those with `DSRG_DUMP_AMPS`.

- Type: Boolean
- Default: False

3.4.3 Density Fitted (DF) and Cholesky Decomposition (CD) Implementations

1. Theory

Integral factorization, as it suggests, factorizes the two-electron integrals into contractions of low-rank tensors. In particular, we use density fitting (DF) or Cholesky decomposition (CD) technique to express two-electron integrals as

$$\langle ij||ab\rangle = \sum_{Q}^{N_{\text{aux}}} (B_{ia}^{Q} B_{jb}^{Q} - B_{ib}^{Q} B_{ja}^{Q})$$

where Q runs over auxiliary indices. Note that we use physicists' notation here but the DF/CD literature use chemist notation.

The main difference between DF and CD is how the B tensor is formed. In DF, the B tensor is defined as

$$B_{pq}^{Q} = \sum_{P}^{N_{\text{aux}}} (pq|P)(P|Q)^{-1/2}.$$

In the CD approach, the B tensor is formed by performing a pivoted incomplete Cholesky decomposition of the 2-electron integrals. The accuracy of this decomposition is determined by a user defined tolerance, which directly determines the accuracy of the 2-electron integrals.

2. Limitations

There are several limitations of the current implementation.

We store the entire three-index integrals in memory by default. Consequently, we can treat about 1000 basis functions. For larger systems, please use the `DiskDF` keyword where these integrals are loaded to memory only when necessary. In general, we can treat about 2000 basis functions (with DiskDF) using DSRG-MRPT2.

Density fitting is more suited to spin-adapted equations while the current code uses spin-integrated equations.

We have a more optimized code of DF-DSRG-MRPT2. The batching algorithms of DSRG-MRPT3 (manually tuned) and MR-LDSRG(2) (Ambit) are currently not ideal.

3. Examples

Tip: For DSRG-MRPT3 and MR-LDSRG(2), DF/CD will automatically turn on if `INT_TYPE` is set to `DF`, `CD`, or `DISKDF`. For DSRG-MRPT2 computations, please set the `CORRELATION_SOLVER` keyword to `THREE-DSRG-MRPT2` besides the `INT_TYPE` option.

The following input performs a DF-DSRG-MRPT2 calculation on nitrogen molecule. This example is modified from the df-dsrg-mrpt2-4 test case.

```
import forte

memory 500 mb

molecule N2{
  0 1
  N
  N  1 R
```

(continues on next page)

(continued from previous page)

```
  R = 1.1
}

set globals{
   reference               rhf
   basis                   cc-pvdz
   scf_type                df
   df_basis_mp2            cc-pvdz-ri
   df_basis_scf            cc-pvdz-jkfit
   d_convergence           8
   e_convergence           10
}

set forte {
   active_space_solver     cas
   int_type                df
   restricted_docc         [2,0,0,0,0,2,0,0]
   active                  [1,0,1,1,0,1,1,1]
   correlation_solver      three-dsrg-mrpt2
   dsrg_s                  1.0
}

Escf, wfn = energy('scf', return_wfn=True)
energy('forte', ref_wfn=wfn)
```

To perform a DF computation, we need to specify the following options:

1. Psi4 options: `SCF_TYPE`, `DF_BASIS_SCF`, `DF_BASIS_MP2`

Warning: In test case df-dsrg-mrpt2-4, `SCF_TYPE` is specified to `PK`, which is incorrect for a real computation.

2. Forte options: `CORRELATION_SOLVER`, `INT_TYPE`

Attention: Here we use different basis sets for `DF_BASIS_SCF` and `DF_BASIS_MP2`. There is no consensus on what basis sets should be used for MR computations. However, there is one caveat of using inconsistent DF basis sets in Forte due to orbital canonicalization: Frozen orbitals are left unchanged (i.e., canonical for `DF_BASIS_SCF`) while DSRG (and orbital canonicalization) only reads `DF_BASIS_MP2`. This inconsistency leads to slight deviations to the frozen-core energies ($< 10^{-4}$ a.u.) comparing to using identical DF basis sets.

The output produced by this input:

```
==> DSRG-MRPT2 (DF/CD) Energy Summary <==

  E0 (reference)                 =   -109.023295547673101
  <[F, T1]>                      =     -0.000031933175984
  <[F, T2]>                      =     -0.000143067308999
  <[V, T1]>                      =     -0.000183596694872
  <[V, T2]> C_4 (C_2)^2 HH       =      0.003655752832132
  <[V, T2]> C_4 (C_2)^2 PP       =      0.015967613107776
  <[V, T2]> C_4 (C_2)^2 PH       =      0.017515091046864
```

(continues on next page)

(continued from previous page)

```
  <[V, T2]> C_6 C_2              =     -0.000194156963250
  <[V, T2]> (C_2)^4              =     -0.265179563137787
  <[V, T2]>                      =     -0.228235263114265
  DSRG-MRPT2 correlation energy  =     -0.228593860294120
  DSRG-MRPT2 total energy        =   -109.251889407967226
  max(T1)                        =      0.002234583100143
  ||T1||                         =      0.007061738508652
```

Note: THREE-DSRG-MRPT2 currently does not print a summary for the largest amplitudes.

To use Cholesky integrals, set INT_TYPE to CHOLESKY and specify CHOLESKY_TOLERANCE. For example, a CD equivalence of the above example is

```
# same molecule input ...

set globals{
   reference               rhf
   basis                   cc-pvdz
   scf_type                cd                  # <=
   cholesky_tolerance      5                   # <=
   d_convergence           8
   e_convergence           10
}

set forte {
   active_space_solver     cas
   int_type                cholesky            # <=
   cholesky_tolerance      1.0e-5              # <=
   restricted_docc         [2,0,0,0,0,2,0,0]
   active                  [1,0,1,1,0,1,1,1]
   correlation_solver      three-dsrg-mrpt2
   dsrg_s                  1.0
}

Escf, wfn = energy('scf', return_wfn=True)
energy('forte', ref_wfn=wfn)
```

The output energies are:

```
E0 (reference)                 =   -109.021897967354022
DSRG-MRPT2 total energy        =   -109.250407455691658
```

The energies computed using conventional integrals are:

```
E0 (reference)                 =   -109.021904986168678
DSRG-MRPT2 total energy        =   -109.250416722481461
```

The energy error of using CD integrals (threshold = 10^{-5} a.u.) is thus around $\sim 10^{-5}$ a.u.. In general, comparing to conventional 4-index 2-electron integrals, the use of CD integrals yields energy errors to the same decimal points as CHOLESKY_TOLERANCE.

Caution: The cholesky algorithm, as currently written, does not allow applications to large systems (> 1000 basis functions).

4. Related Options

For basic options of factorized integrals, please check sec:integrals.

CCVV_BATCH_NUMBER

Manually specify the number of batches for computing `THREE-DSRG-MRPT2` energies. By default, the number of batches are automatically computed using the remaining memory estimate.

- Type: integer
- Default: -1

3.4.4 MR-DSRG Approaches for Excited States

There are several MR-DSRG methods available for computing excited states.

Warning: The current only supports SA-DSRG due to the revamp of Forte structure. MS-, XMS-, DWMS-DSRG will be available soon.

1. State-Averaged Formalism

In state-averaged (SA) DSRG, the MK vacuum is an ensemble of electronic states, which are typically obtained by an SA-CASSCF computation. For example, we want to study two states, Φ_1 and Φ_2, described qualitatively by a CASCI with SA-CASSCF orbitals. The ensemble of states (assuming equal weights) is characterized by the density operator

$$\hat{\rho} = \frac{1}{2}|\Phi_1\rangle\langle\Phi_1| + \frac{1}{2}|\Phi_2\rangle\langle\Phi_2|$$

Note that Φ_1 and Φ_2 are just two of the many states (say, n) in CASCI.

The bare Hamiltonian and cluster operators are normal ordered with respect to this ensemble, whose information is embedded in the state-averaged densities. An effective Hamiltonian $\bar{H}$ is then built by solving the DSRG cluster amplitudes. In this way, the dynamical correlation is described for all the states lying in the ensemble. Here, the DSRG solver and correlation levels remain the same to those of state-specific cases. For example, we use `DSRG-MRPT3` to do SA-DSRG-PT3.

Now we have many ways to proceed and obtain the excited states, two of which have been implemented.

- One approach is to diagonalize $\bar{H}$ using Φ_1 and Φ_2. As such, the new states are just linear combinations of states in the ensemble and the CI coefficients are then constrained to be combined using Φ_1 and Φ_2. We term this approach constrained SA, with a letter "c" appended at the end of a method name (e.g., SA-DSRG-PT2c). and in Forte we use the option `SA_SUB` to specify this SA variant.
- The other approach is to diagonalize $\bar{H}$ using all configurations in CASCI, which allows all CI coefficients to relax. This approach is the default SA-DSRG approach, which is also the default in Forte. The corresponding option is `SA_FULL`.

For both approaches, one could iterate these two-step (DSRG + diagoanlization) procedure till convergence is reached.

Note: For SA-DSRG, a careful inspection of the output CI coefficients is usually necessary. This is because the ordering of states may change after dynamical correlation is included. When that happens, a simple fix is to include more states in the ensemble, which may reduce the accuracy yet usually OK if only a few low-lying states are of interest.

2. Multi-State, Extended Multi-State Formalisms

Warning: Not available at the moment.

Note: Only support at the PT2 level of theory.

In multi-state (MS) DSRG, we adopt the single-state parametrization where the effective Hamiltonian is built as

$$H_{MN}^{\text{eff}} = \langle\Phi_M|\hat{H}|\Phi_N\rangle + \frac{1}{2}\left[\langle\Phi_M|\hat{T}_M^\dagger\hat{H}|\Phi_N\rangle + \langle\Phi_M|\hat{H}\hat{T}_N|\Phi_N\rangle\right],$$

where $\hat{T}_M$ is the state-specific cluster amplitudes for state M, that is, we solve DSRG-PT2 amplitudes $\hat{T}_M$ normal ordered to $|\Phi_M\rangle$. The MS-DSRG-PT2 energies are then obtained by diagonalizing this effective Hamiltonian. However, it is known this approach leaves wiggles on the potential energy surface (PES) near the strong coupling region of the reference wave functions.

A simple way to cure these artificial wiggles is to use the extended MS (XMS) approach. In XMS DSRG, the reference states $\tilde{\Phi}_M$ are linear combinations of CASCI states Φ_M such that the Fock matrix is diagonal. Specifically, the Fock matrix is built according to

$$F_{MN} = \langle\Phi_M|\hat{F}|\Phi_N\rangle,$$

where $\hat{F}$ is the state-average Fock operator. Then in the mixed state basis, we have $\langle\tilde{\Phi}_M|\hat{F}|\tilde{\Phi}_N\rangle = 0$, if $M \neq N$. The effective Hamiltonian is built similarly to that of MS-DSRG-PT2, except that $\tilde{\Phi}_M$ is used.

3. Dynamically Weighted Multi-State Formalism

Warning: Not available at the moment.

Note: Only support at the PT2 level of theory.

As shown by the XMS approach, mixing states is able to remove the wiggles on the PES. Dynamically weighted MS (DWMS) approach provides an alternative way to mix zeroth-order states. The idea of DWMS is closely related to SA-DSRG. In DWMS, we choose an ensemble of zeroth-order reference states, where the weights are automatically determined according to the energy separations between these reference states. Specifically, the weight for target state M is given by

$$\omega_{MN}(\zeta) = \frac{e^{-\zeta(E_M^{(0)} - E_N^{(0)})^2}}{\sum_{P=1}^{n} e^{-\zeta(E_M^{(0)} - E_P^{(0)})^2}},$$

where $E_M^{(0)} = \langle \Phi_M | \hat{H} | \Phi_M \rangle$ is the zeroth-order energy of state M and ζ is a parameter to be set by the user. Then we follow the MS approach to form an effective Hamiltonian where the amplitudes are solved for the ensemble tuned to that particular state.

For a given value of $zeta$, the weights of two reference states Φ_M and Φ_N will be equal if they are degenerate in energy. On the other limit where they are energetically far apart, the ensemble used to determine $\hat{T}_M$ mainly consists of Φ_M with a little weight on Φ_N, and vice versa.

For two non-degenerate states, by sending ζ to zero, both states in the ensemble have equal weights (general for n states), which is equivalent to the SA formalism. If we send ζ to ∞, then the ensemble becomes state-specific. Thus, parameter ζ can be understood as how drastic between the transition from MS to SA schemes.

Caution: It is not guaranteed that the DWMS energy (for one adiabatic state) lies in between the MS and SA values. When DWMS energies go out of the bounds of MS and SA, a small ζ value is preferable to avoid rather drastic energy changes in a small geometric region.

4. Examples

A simple example is to compute the lowest two states of LiF molecule using SA-DSRG-PT2.

```
import forte

molecule {
  0 1
  Li
  F  1 R
  R = 10.000

  units bohr
}

basis {
  assign Li Li-cc-pvdz
  assign F  aug-cc-pvdz
[ Li-cc-pvdz ]
spherical
****
Li     0
S   8   1.00
   1469.0000000              0.0007660
    220.5000000              0.0058920
     50.2600000              0.0296710
     14.2400000              0.1091800
      4.5810000              0.2827890
      1.5800000              0.4531230
      0.5640000              0.2747740
      0.0734500              0.0097510
S   8   1.00
   1469.0000000             -0.0001200
    220.5000000             -0.0009230
     50.2600000             -0.0046890
     14.2400000             -0.0176820
```

(continues on next page)

(continued from previous page)

```
      4.5810000             -0.0489020
      1.5800000             -0.0960090
      0.5640000             -0.1363800
      0.0734500              0.5751020
S   1   1.00
      0.0280500              1.0000000
P   3   1.00
      1.5340000              0.0227840
      0.2749000              0.1391070
      0.0736200              0.5003750
P   1   1.00
      0.0240300              1.0000000
D   1   1.00
      0.1239000              1.0000000
****
}

set globals{
  reference           rhf
  scf_type            pk
  maxiter             300
  e_convergence       10
  d_convergence       10
  docc                [4,0,1,1]
  restricted_docc     [3,0,1,1]
  active              [2,0,0,0]
  mcscf_r_convergence 7
  mcscf_e_convergence 10
  mcscf_maxiter       250
  mcscf_diis_start    25
  num_roots           2
  avg_states          [0,1]
}

set forte{
  active_space_solver cas
  correlation_solver  dsrg-mrpt2
  frozen_docc        [2,0,0,0]
  restricted_docc    [1,0,0,0]
  active             [3,0,2,2]
  dsrg_s             0.5
  avg_state          [[0,1,2]]
  dsrg_multi_state   sa_full
  calc_type          sa
}

Emcscf, wfn = energy('casscf', return_wfn=True)
energy('forte',ref_wfn=wfn)
```

Here, we explicitly specify the cc-pVDZ basis set of Li since Psi4 uses seg-opt basis (at least at some time). For simplicity, we do an SA-CASSCF(2,2) computation in Psi4 but the active space in Forte is CASCI(8e,7o), which should be clearly stated in the publication if this kind of special procedure is used.

To perform an SA-DSRG-PT2 computation, the following keywords should be specified (besides those already mentioned in the state-specific DSRG-MRPT2):

- `CALC_TYPE`: The type of computation should be set to state averaging, i.e., SA. Multi-state and dynamically weighted computations should be set correspondingly.
- `AVG_STATE`: This specifies the states to be averaged, given in arrays of triplets [[A1, B1, C1], [A2, B2, C2], ...]. Each triplet corresponds to the *state irrep*, *state multiplicity*, and the *nubmer of states*, in sequence. The number of states are counted from the lowest energy one in the given symmetry.
- `DSRG_MULTI_STATE`: This options specifies the methods used in DSRG computations. By default, it will use `SA_FULL`.

The output of this example will print out the CASCI(8e,7o) configurations

```
==> Root No. 0 <==

  ba0 20 20         -0.6992227471
  ab0 20 20         -0.6992227471
  200 20 20         -0.1460769052

  Total Energy:   -106.772573855919561


==> Root No. 1 <==

  200 20 20          0.9609078151
  b0a 20 20          0.1530225853
  a0b 20 20          0.1530225853
  ba0 20 20         -0.1034194675
  ab0 20 20         -0.1034194675

  Total Energy:   -106.735798144523812
```

Then the 1-, 2-, and 3-RDMs for each state are computed and then sent to orbital canonicalizer. The DSRG-PT2 computation will still print out the energy contributions, which now correspond to the corrections to the average of the ensemble.

```
==> DSRG-MRPT2 Energy Summary <==

  E0 (reference)                 =   -106.754186000221665
  <[F, T1]>                      =     -0.000345301150943
  <[F, T2]>                      =      0.000293904835970
  <[V, T1]>                      =      0.000300892512596
  <[V, T2]> (C_2)^4              =     -0.246574892923286
  <[V, T2]> C_4 (C_2)^2 HH       =      0.000911300780649
  <[V, T2]> C_4 (C_2)^2 PP       =      0.002971830422787
  <[V, T2]> C_4 (C_2)^2 PH       =      0.010722949661906
  <[V, T2]> C_6 C_2              =      0.000099208259233
  <[V, T2]>                      =     -0.231869603798710
  DSRG-MRPT2 correlation energy  =     -0.231620107601087
  DSRG-MRPT2 total energy        =   -106.985806107822754
```

Finally, a CASCI is performed using DSRG-PT2 dressed integrals.

```
==> Root No. 0 <==

  200 20 20          0.8017660337
  ba0 20 20          0.4169816393
  ab0 20 20          0.4169816393

  Total Energy:   -106.990992362637314


==> Root No. 1 <==

  200 20 20         -0.5846182713
  ba0 20 20          0.5708699624
  ab0 20 20          0.5708699624

  Total Energy:   -106.981903302649229
```

Here we observe the ordering of states changes by comparing the configurations. In fact, it is near the avoided crossing region and we see the CI coefficients between these two states are very similar (comparing to the original CASCI coefficients). An automatic way to correspond states before and after DSRG treatments for dynamical correlation is not implemented. A simple approach is to compute the overlap, which should usually suffice.

At the end, we print the energy summary of the states of interest.

```
==> Energy Summary <==

  Multi.  Irrep.  No.              Energy
  ----------------------------------------
     1      A1     0      -106.990992362637
     1      A1     1      -106.981903302649
  ----------------------------------------
```

Tip: It is sometimes cumbersome to grab the energies of all the computed states from the output file, especially when multiple reference relaxation steps are performed. Here, one could use the keyword **DSRG_DUMP_RELAXED_ENERGIES** where a JSON file `dsrg_relaxed_energies.json` is created. In the above example, the file will read

```
{
    "0": {
        "ENERGY ROOT 0 1A1": -106.7725738559195,
        "ENERGY ROOT 1 1A1": -106.7357981445238
    },
    "1": {
        "DSRG FIXED": -106.98580610782275,
        "DSRG RELAXED": -106.98644783264328,
        "ENERGY ROOT 0 1A1": -106.99099236263731,
        "ENERGY ROOT 1 1A1": -106.98190330264923
    }
}
```

The printing for SA-DSRG-PT2c (set `DSRG_MULTI_STATE` to `SA_SUB`) is slightly different from above. After the DSRG-PT2 computation, we build the effective Hamiltonian using the original CASCI states.

```
==> Building Effective Hamiltonian for Singlet A1 <==

Computing  1RDMs (0 Singlet A1 - 0 Singlet A1) ... Done. Timing        0.001090 s
Computing  2RDMs (0 Singlet A1 - 0 Singlet A1) ... Done. Timing        0.001884 s
Computing 1TrDMs (0 Singlet A1 - 1 Singlet A1) ... Done. Timing        0.001528 s
Computing 2TrDMs (0 Singlet A1 - 1 Singlet A1) ... Done. Timing        0.002151 s
Computing  1RDMs (1 Singlet A1 - 1 Singlet A1) ... Done. Timing        0.001114 s
Computing  2RDMs (1 Singlet A1 - 1 Singlet A1) ... Done. Timing        0.001757 s

==> Effective Hamiltonian for Singlet A1 <==

## Heff Singlet A1 (Symmetry 0) ##
Irrep: 1 Size: 2 x 2

               1                   2

  1  -106.98637816344888     0.00443421124030
  2     0.00443421124030  -106.98523405219674

## Eigen Vectors of Heff for Singlet A1 with eigenvalues ##

         1           2

  1  -0.7509824  -0.6603222
  2   0.6603222  -0.7509824

   -106.9902771-106.9813351
```

Here, we see a strong coupling between the two states at this geometry: The SA-DSRG-PT2c ground state is $0.75|\Phi_1\rangle - 0.66|\Phi 2\rangle$.

5. Related Options

DSRG_MULTI_STATE

Algorithms to compute excited states.

- Type: string
- Options: SA_FULL, SA_SUB, MS, XMS
- Default: SA_FULL

DWMS_ZETA

Automatic Gaussian width cutoff for the density weights.

- Type: double
- Default: 0.0

Note: Add options when DWMS is re-enabled.

3.4.5 TODOs

0. Re-enable MS, XMS, and DWMS

These are disabled due to an infrastructure change.

1. DSRG-MRPT2 Analytic Energy Gradients

This is an ongoing project.

2. MR-DSRG(T) with Perturbative Triples

This is an ongoing project.

3.4.6 A Complete List of DSRG Teset Cases

Acronyms used in the following text:

- Integrals

  DF: density fitting; DiskDF: density fitting (disk algorithm); CD: Cholesky decomposition;

- Reference Relaxation

  U: unrelaxed; PR: partially relaxed; R: relaxed; FR: fully relaxed;

- Single-State / Multi-State

  SS: state-specific; SA: state-averaged; SAc: state-averaged with constrained reference; MS: multi-state; XMS: extended multi-state; DWMS: dynamically weighted multi-state;

- Theoretical Variants

  QC: commutator truncated to doubly nested level (i.e., $\bar{H} = \hat{H} + [\hat{H}, \hat{A}] + \frac{1}{2}[[\hat{H}, \hat{A}], \hat{A}]$); SQ: sequential transformation; NIVO: non-interacting virtual orbital approximation;

- Run Time:

  long: > 30 s to finish; Long: > 5 min to finish; LONG: > 20 min to finish;

1. DSRG-MRPT2 Test Cases

Name	Variant	Molecule	Notes
dsrg-mrpt2-1	SS, U	BeH_2	large s value, user defined basis set
dsrg-mrpt2-2	SS, U	HF	
dsrg-mrpt2-3	SS, U	H_4 (rectangular)	
dsrg-mrpt2-4	SS, U	N_2	
dsrg-mrpt2-5	SS, U	benzyne C_6H_4	
dsrg-mrpt2-6	SS, PR	N_2	
dsrg-mrpt2-7-casscf-natorbs	SS, PR	N_2	CASSCF natural orbitals
dsrg-mrpt2-8-sa	SA, SAc	LiF	lowest two singlet states, user defined basis set
dsrg-mrpt2-9-xms	MS, XMS	LiF	lowest two singlet states
dsrg-mrpt2-10-CO	SS, PR	CO	dipole moment (not linear response)
dsrg-mrpt2-11-C2H4	SA	ethylene C_2H_4	lowest three singlet states
dsrg-mrpt2-12-localized-actv	SA	butadiene C_4H_6	long, localized active orbitals
dsrg-mrpt2-13	SS	N_2 and N atom	size-consistency check
aci-dsrg-mrpt2-1	SS, U	N_2	ACI($\sigma = 0$)
aci-dsrg-mrpt2-2	SS, U	H_4 (rectangular)	ACI($\sigma = 0$)
aci-dsrg-mrpt2-3	SS, PR	H_4 (rectangular)	ACI($\sigma = 0$)
aci-dsrg-mrpt2-4	SS, U	octatetraene C_8H_{10}	DF, ACI($\sigma = 0.001$), ACI batching
aci-dsrg-mrpt2-5	SS, PR	octatetraene C_8H_{10}	long, DF, ACI($\sigma = 0.001$), ACI batching

2. DF/CD-DSRG-MRPT2 Test Cases

Name	Variant	Molecule	Notes
cd-dsrg-mrpt2-1	SS, U	BeH_2	CD($\sigma = 10^{-14}$)
cd-dsrg-mrpt2-2	SS, U	HF	CD($\sigma = 10^{-14}$)
cd-dsrg-mrpt2-3	SS, U	H_4 (rectangular)	CD($\sigma = 10^{-14}$)
cd-dsrg-mrpt2-4	SS, U	N_2	CD($\sigma = 10^{-12}$)
cd-dsrg-mrpt2-5	SS, U	benzyne C_6H_4	CD($\sigma = 10^{-11}$)
cd-dsrg-mrpt2-6	SS, PR	BeH_2	CD($\sigma = 10^{-14}$)
cd-dsrg-mrpt2-7-sa	SA	LiF	CD($\sigma = 10^{-14}$)
df-dsrg-mrpt2-1	SS, U	BeH_2	
df-dsrg-mrpt2-2	SS, U	HF	
df-dsrg-mrpt2-3	SS, U	H_4 (rectangular)	
df-dsrg-mrpt2-4	SS, U	N_2	
df-dsrg-mrpt2-5	SS, U	benzyne C_6H_4	
df-dsrg-mrpt2-6	SS, PR	N_2	
df-dsrg-mrpt2-7-localized-actv	SA	butadiene C_4H_6	long, localized active orbitals
df-dsrg-mrpt2-threading1	SS, U	benzyne C_6H_4	
df-dsrg-mrpt2-threading2	SS, U	benzyne C_6H_4	
df-dsrg-mrpt2-threading4	SS, U	benzyne C_6H_4	
diskdf-dsrg-mrpt2-1	SS, U	BeH_2	
diskdf-dsrg-mrpt2-2	SS, U	HF	
diskdf-dsrg-mrpt2-3	SS, U	H_4 (rectangular)	
diskdf-dsrg-mrpt2-4	SS, PR	N_2	
diskdf-dsrg-mrpt2-5	SS, U	benzyne C_6H_4	
diskdf-dsrg-mrpt2-threading1	SS, U	benzyne C_6H_4	
diskdf-dsrg-mrpt2-threading4	SS, U	benzyne C_6H_4	
df-aci-dsrg-mrpt2-1	SS, U	benzyne C_6H_4	ACI($\sigma = 0$)
df-aci-dsrg-mrpt2-2	SS, U	HF	ACI($\sigma = 0.0001$)

3. DSRG-MRPT3 Test Cases

Name	Variant	Molecule	Notes
dsrg-mrpt3-1	SS, PR	HF	
dsrg-mrpt3-2	SS, PR	HF	CD($\sigma = 10^{-8}$)
dsrg-mrpt3-3	SS, PR	N_2	CD($\sigma = 10^{-8}$), long, time printing
dsrg-mrpt3-4	SS, PR	N_2	
dsrg-mrpt3-5	SA	LiF	CAS(2e,2o), default cc-pVDZ of Li is seg-opt
dsrg-mrpt3-6-sa	SA	LiF	CAS(8e,7o), user defined cc-pVDZ for Li
dsrg-mrpt3-7-CO	SS, PR	CO	dipole moment (not linear response)
dsrg-mrpt3-8-sa-C2H4	SA	ethylene C_2H_4	long, lowest three singlet states
dsrg-mrpt3-9	SS, PR	HF	CD($\sigma = 10^{-14}$), batching
aci-dsrg-mrpt3-1	SS, PR	N_2	ACI($\sigma = 0$)

4. MR-DSRG Test Cases

Name	Variant	Molecule	Notes
mrdsrg-pt2-1	SS, U	BeH_2	PT2
mrdsrg-pt2-2	SS, PR	BeH_2	PT2
mrdsrg-pt2-3	SS, FR	BeH_2	long, PT2
mrdsrg-pt2-4	SS, FR	HF	PT2
mrdsrg-pt2-5	SS, R	HF	long, PT2, DIIS, 0th-order Hamiltonian
mrdsrg-srgpt2-1	SS, U	BeH_2	Long, SRG_PT2
mrdsrg-srgpt2-2	SS, U	BeH_2	LONG, SRG_PT2, Dyall Hamiltonian
mrdsrg-ldsrg2-1	SS, U	N_2	long, read amplitudes
mrdsrg-ldsrg2-df-1	SS, R	BeH_2	CD, long
mrdsrg-ldsrg2-df-2	SS, R	HF	CD, long
mrdsrg-ldsrg2-df-3	SS, U	H_4 (rectangular)	CD, long
mrdsrg-ldsrg2-df-4	SS, PR	H_2	CD
mrdsrg-ldsrg2-df-seq-1	SS, PR, SQ	BeH_2	CD, Long
mrdsrg-ldsrg2-df-seq-2	SS, R, SQ	HF	CD, Long
mrdsrg-ldsrg2-df-seq-3	SS, U, SQ	H_4 (rectangular)	CD, long
mrdsrg-ldsrg2-df-seq-4	SS, FR, SQ	H_4 (rectangular)	CD, Long
mrdsrg-ldsrg2-df-nivo-1	SS, PR, NIVO	BeH_2	CD, long
mrdsrg-ldsrg2-df-nivo-2	SS, R, NIVO	HF	CD, long
mrdsrg-ldsrg2-df-nivo-3	SS, U, NIVO	H_4 (rectangular)	CD, long
mrdsrg-ldsrg2-df-seq-nivo-1	SS, PR, SQ, NIVO	BeH_2	CD, long
mrdsrg-ldsrg2-df-seq-nivo-2	SS, R, SQ, NIVO	HF	CD, Long
mrdsrg-ldsrg2-df-seq-nivo-3	SS, U, SQ, NIVO	H_4 (rectangular)	CD, long
mrdsrg-ldsrg2-qc-1	SS, FR, QC	HF	long
mrdsrg-ldsrg2-qc-2	SS, U, QC	HF	long
mrdsrg-ldsrg2-qc-df-2	SS, U, QC	HF	CD, long

5. DWMS-DSRG-PT2 Test Cases

Add test cases when DWMS is back to life.

6. Spin-Adapted MR-DSRG Test Cases

Name	Variants	Molecule	Notes
mrdsrg-spin-adapted-1	SS, U	HF	LDSRG(2) truncated to 2-nested commutator
mrdsrg-spin-adapted-2	SS, PR	HF	long, LDSRG(2), non-semicanonical orbitals
mrdsrg-spin-adapted-3	SS, R, SQ, NIVO	HF	long, CD, LDSRG(2)
mrdsrg-spin-adapted-4	SS, U	N_2	long, CD, LDSRG(2), non-semicanonical, zero ccvv
mrdsrg-spin-adapted-5	SS, U	N_2	long, read/dump amplitudes
mrdsrg-spin-adapted-pt2-1	SS, U	HF	CD
mrdsrg-spin-adapted-pt2-2	SS, U	HF	CD, non-semicanonical orbitals, zero ccvv source
mrdsrg-spin-adapted-pt2-3	SS, PR	p-benzyne	DiskDF
mrdsrg-spin-adapted-pt2-4	SS, R	O_2	triplet ground state, CASSCF(8e,6o)
mrdsrg-spin-adapted-pt2-5	SA, R	C_2	CASSCF(8e,8o), zero 3 cumulant
mrdsrg-spin-adapted-pt2-6	SA	benzene	Exotic state-average weights
mrdsrg-spin-adapted-pt3-1	SS, PR	HF	CD
mrdsrg-spin-adapted-pt3-2	SA	ethylene	lowest three singlet states

3.4.7 References

The seminal work of DSRG is given in:

- "A driven similarity renormalization group approach to quantum many-body problems", F. A. Evangelista, *J. Chem. Phys.* **141**, 054109 (2014). (doi: 10.1063/1.4890660).

A general and pedagogical discussion of MR-DSRG is presented in:

- "Multireference Theories of Electron Correlation Based on the Driven Similarity Renormalization Group", C. Li and F. A. Evangelista, *Annu. Rev. Phys. Chem.* **70**, 245-273 (2019). (doi: 10.1146/annurev-physchem-042018-052416).

The theories of different DSRG correlation levels are discussed in the following articles:

DSRG-MRPT2 (without reference relaxation):

- "Multireference Driven Similarity Renormalization Group: A Second-Order Perturbative Analysis", C. Li and F. A. Evangelista, *J. Chem. Theory Compt.* **11**, 2097-2108 (2015). (doi: 10.1021/acs.jctc.5b00134).

DSRG-MRPT3 and variants of reference relaxations:

- "Driven similarity renormalization group: Third-order multireference perturbation theory", C. Li and F. A. Evangelista, *J. Chem. Phys.* **146**, 124132 (2017). (doi: 10.1063/1.4979016). Erratum: **148**, 079902 (2018). (doi: 10.1063/1.5023904).

MR-LDSRG(2):

- "Towards numerically robust multireference theories: The driven similarity renormalization group truncated to one- and two-body operators", C. Li and F. A. Evangelista, *J. Chem. Phys.* **144**, 164114 (2016). (doi: 10.1063/1.4947218). Erratum: **148**, 079903 (2018). (doi: 10.1063/1.5023493).

The DSRG extensions for excited state are discussed in the following articles:

SA-DSRG framework and its PT2 and PT3 applications:

- "Driven similarity renormalization group for excited states: A state-averaged perturbation theory", C. Li and F. A. Evangelista, *J. Chem. Phys.* **148**, 124106 (2018). (doi: 10.1063/1.5019793).

MS-DSRG and DWMS-DSRG:

- "Dynamically weighted multireference perturbation theory: Combining the advantages of multi-state and state- averaged methods", C. Li and F. A. Evangelista, *J. Chem. Phys.* **150**, 144107 (2019). (doi: 10.1063/1.5088120).

The DSRG analytic energy gradients are described in the following series of papers:

Single reference DSRG-PT2:

- "Analytic gradients for the single-reference driven similarity renormalization group second-order perturbation theory", S. Wang, C. Li, and F. A. Evangelista, *J. Chem. Phys.* **151**, 044118 (2019). (doi: 10.1063/1.5100175).

The integral-factorized implementation of DSRG is firstly achieved in:

- "An integral-factorized implementation of the driven similarity renormalization group second-order multireference perturbation theory", K. P. Hannon, C. Li, and F. A. Evangelista, *J. Chem. Phys.* **144**, 204111 (2016). (doi: 10.1063/1.4951684).

The sequential variant of MR-LDSRG(2) and NIVO approximation are described in:

- "Improving the Efficiency of the Multireference Driven Similarity Renormalization Group via Sequential Transformation, Density Fitting, and the Noninteracting Virtual Orbital Approximation", T. Zhang, C. Li, and F. A. Evangelista, *J. Chem. Theory Compt.* **15**, 4399-4414 (2019). (doi: 10.1021/acs.jctc.9b00353).

Combination between DSRG and adaptive configuration interaction with applications to acenes:

- "A Combined Selected Configuration Interaction and Many-Body Treatment of Static and Dynamical Correlation in Oligoacenes", J. B. Schriber, K. P. Hannon, C. Li, and F. A. Evangelista, *J. Chem. Theory Compt.* **14**, 6295-6305 (2018). (doi: 10.1021/acs.jctc.8b00877).

Benchmark of state-specific unrelaxed DSRG-MRPT2 (tested 34 active orbitals):

- "A low-cost approach to electronic excitation energies based on the driven similarity renormalization group", C. Li, P. Verma, K. P. Hannon, and F. A. Evangelista, *J. Chem. Phys.* **147**, 074107 (2017). (doi: 10.1063/1.4997480).

3.5 Active Space Embedding Theory

3.5.1 Simple active space frozen-orbital embedding

This embedding procedure provides an automatic way to embed one fragment into an environment, by an active space embedding theory that allows multireference method embedded in single-reference or multireference environment, for example, DSRG-MRPT2-in-CASSCF.

The input file should at least include two fragment:

```
molecule {
  0 1 # Fragment 1, system A
  ...
  --
  0 1 # Fragment 2, environment or bath B
  ...

  symmetry c1 # Currently it is suggested to disable symmetry for embedding calculations
}
```

In the forte options, turn on embedding procedure by adding options to forte:

```
set forte{
  embedding    true
  embedding_cutoff_method    threshold # threshold/cum_threshold/num_of_orbitals
  embedding_threshold    0.5 # threshold t
}
```

This is the minimum input required to run the embedding calculation. The embedding procedure will update the wavefunction coefficients and the MOSpaceInfo before running general forte calculations.

Four examples are available in test cases. Note that the program will by default semi-canonicalize frozen and active orbitals, if this is not intended, one can disable this semi-canonicalization with corresponding options.

EMBEDDING Options

EMBEDDING

Turn on/off embedding procedure.

- Type: bool
- Default: false

EMBEDDING_CUTOFF_METHOD

The choices of embedding cutoff methods. THRESHOLD: simple threshold CUM_THRESHOLD: cumulative threshold NUM_OF_ORBITALS: fixed number of orbitals

- Type: string
- Options: THRESHOLD, CUM_THRESHOLD, NUM_OF_ORBITALS
- Default: THRESHOLD

EMBEDDING_THRESHOLD

The threshold t of embedding cutoff. Do nothing when EMBEDDING_CUTOFF_METHOD is NUM_OF_ORBITALS

- Type: double
- Default: 0

EMBEDDING_REFERENCE

The reference wavefunction, do not need to specify unless using special active space treatment. Default is CASSCF with an well-defined active space including occupied and virtual orbitals.

- Type: string
- Default: CASSCF

EMBEDDING_SEMICANONICALIZE_ACTIVE

Turn on/off the semi-canonicalization of active space.

- Type: bool
- Default: true

EMBEDDING_SEMICANONICALIZE_ACTIVE

Turn on/off the semi-canonicalization of frozen core and virtual space. This will create a set of well-defined frozen orbitals.

- Type: bool
- Default: true

NUM_A_DOCC

The number of occupied orbitals fixed to system A, only function when EMBEDDING_CUTOFF_METHOD is NUM_OF_ORBITALS.

- Type: int
- Default: 0

NUM_A_UOCC

The number of virtual orbitals fixed to system A, only function when EMBEDDING_CUTOFF_METHOD is NUM_OF_ORBITALS.

- Type: int
- Default: 0

3.6 Atomic Valence Active Space (AVAS)

3.6.1 Overview

This AVAS procedure provides an automatic way to generate an active space for correlation computations by projecting MOs to an AO subspace, computing and sorting the overlaps for a new set of rotated MOs and a suitable active space.

Given a projector $\hat{P}$, AVAS builds the projected overlap matrices for doubly occupied and virtual orbitals separately from an restricted Hartree-Fock wave function

$$S_{ij} = \langle i|\hat{P}|j\rangle = \sum_{\mu\nu} C_{\mu i} P_{\mu\nu} C_{\nu j}, \quad i,j \in \{\text{DOCC}\},$$

$$\bar{S}_{ab} = \langle a|\hat{P}|b\rangle = \sum_{\mu\nu} C_{\mu a} P_{\mu\nu} C_{\nu b}, \quad a,b \in \{\text{UOCC}\},$$

where the projector matrix is given by

$$P_{\mu\nu} = \sum_{pq} \langle \mu | p \rangle (\rho^{-1})_{pq} \langle q | \nu \rangle, \quad p, q \in \{\text{Target Valence Atomic Orbitals}\}.$$

The matrix ρ^{-1} is the inverse of target AO overlap matrix $\rho_{pq} = \langle p | q \rangle$.

Note: Target AOs are selected from the MINAO basis.

If the option `AVAS_DIAGONALIZE` is `TRUE`, AVAS will diagonalize matrices S_{ij} and $\bar{S}_{ab}$ and rotate orbitals separately such that the Hartree-Fock energy is unaffected:

$$\mathbf{SU} = \mathbf{U}\sigma_{\mathrm{DOCC}}, \quad \tilde{C}_{\mu i} = \sum_j C_{\mu j} U_{ji},$$
$$\bar{\mathbf{S}}\bar{\mathbf{U}} = \bar{\mathbf{U}}\sigma_{\mathrm{UOCC}}, \quad \tilde{C}_{\mu a} = \sum_b C_{\mu b} \bar{U}_{ba}.$$

The two sets of eigenvalues are combined $\sigma = \sigma_{\mathrm{DOCC}} \oplus \sigma_{\mathrm{UOCC}}$ and subsequently sorted in descending order. If `AVAS_DIAGONALIZE` is set to `FALSE`, the "eigenvalues" will be directly grabbed from the diagonal elements of the projected overlap matrices and no orbital rotation is performed.

Depending on the selection scheme, part of the orbitals with nonzero eigenvalues are selected as active orbitals. We then semi-canonicalize all four subsets of orbitals separately. The final orbitals are arranged such that those considered as active lie in between the inactive occupied and inactive virtual orbitals.

Warning: The code does not support UHF reference at present. For ROHF reference, our implementation does not touch any singly occupied orbitals, which are all considered as active orbitals and assumed in canonical form.

3.6.2 Input example

In this example, we use AVAS to find an active space for formaldehyde that spans the $2p_x$ orbitals of the C and O atoms, followed by a CASCI computation.

```
import forte
molecule H2CO{
0 1
C           -0.000000000000    -0.000000000006    -0.599542970149
O           -0.000000000000     0.000000000001     0.599382404096
H           -0.000000000000    -0.938817812172    -1.186989139808
H            0.000000000000     0.938817812225    -1.186989139839
noreorient  # ask Psi4 to not reorient the xyz coordinate
}

set {
  basis         cc-pvdz
  reference     rhf
  scf_type      pk
  e_convergence 12
}

set forte {
```

(continues on next page)

(continued from previous page)

```
  job_type            none                  # no energy computation
  subspace            ["C(2px)","O(2px)"]   # target AOs from 2px orbitals of C and O
  avas                true                  # turn on AVAS
  avas_diagonalize    true                  # diagonalize the projected overlaps
  avas_sigma          1.0                   # fraction of eigenvalues included as active
}
Ezero, wfn = energy('forte', return_wfn=True)

set forte {
  job_type            newdriver  # compute some forte energy
  active_space_solver fci        # use FCI solver
  print               1          # print level
  restricted_docc     [5,0,0,2]  # from AVAS
  active              [0,0,3,0]  # from AVAS
}
Ecasci = energy('forte', ref_wfn=wfn)
```

Note: The keyword `noreorient` in the `molecule` section is very important if certain orientations of orbitals are selected in the subspace (e.g., $2pz$ of C). Otherwise, the subspace orbital selection may end up the wrong direction.

The AVAS procedure outputs:

```
Sum of eigenvalues: 1.98526975

==> AVAS MOs Information <==

  ---------------------------------------
                     A1    A2    B1    B2
  ---------------------------------------
  DOCC INACTIVE       5     0     0     2
  DOCC ACTIVE         0     0     1     0
  SOCC ACTIVE         0     0     0     0
  UOCC ACTIVE         0     0     2     0
  UOCC INACTIVE      13     3     4     8
  ---------------------------------------
  RESTRICTED_DOCC     5     0     0     2
  ACTIVE              0     0     3     0
  RESTRICTED_UOCC    13     3     4     8
  ---------------------------------------

==> Atomic Valence MOs (Active Marked by *) <==

  ==============================
   Irrep    MO  Occ.  <phi|P|phi>
  ------------------------------
  *  B1      0    2      0.970513
  *  B1      1    0      0.992548
  *  B1      2    0      0.022209
  ==============================
```

The `Sum of eigenvalues` is the sum of traces of projected overlap matrices $\mathbf{S}$ and $\bar{\mathbf{S}}$. We see that AVAS generates

three active orbitals of B1 symmetry. We then use this guess of active orbitals to compute the CASCI energy:

```
==> Root No. 0 <==

  200     -0.98014601
  020      0.18910986

  Total Energy:      -113.911667467206598

==> Energy Summary <==

  Multi.(2ms)  Irrep.  No.               Energy
  ------------------------------------------------
     1  (  0)    A1     0     -113.911667467207
  ------------------------------------------------
```

Note: Currently, the procedure is not automated enough so that two Forte computations need to be carried out. First perform an AVAS and check the output guess of active orbitals. Then put `RESTRICTED_DOCC` and `ACTIVE` in the input for another round of Forte computation.

For more examples, see `avas-1` to `avas-6` in the `tests/methods` folder. In particular, `avas-6` is a practical example on ferrocene.

3.6.3 Defining the molecular plane for orbitals

Molecular systems with conjugated bonds are often arranged into planar geometries. For such systems, it often desirable to select an active space that includes orbitals perpendicular to the plane. Each orbital is a linear combination of atomic p orbitals, which are also perpendicular to the plane. However, unlike the case of formaldehyde, where it easy to select the appropriate and * orbitals, in the more general case a orbital is a linear combination of $2p_x$, $2p_y$, and $2p_z$ orbitals. The approach described in the previous section is not flexible enough to treat general systems like molecules containing multiple systems or systems that are not aligned with a specific molecular axis.

There are two ways to fix this problem. One is to reorient the molecule such that the molecular plane lying in yz plane. However, this approach is not flexible enough to treat multiple systems in a molecule. The other approach is to use all p_x, p_y, p_z orbitals as basis, using which the p orbital perpendicular to the plane can be defined. To do this, we need to specify two keywords: `SUBSPACE` and `SUBSPACE_PI_PLANES`. The option `SUBSPACE_PI_PLANES` takes a list of atoms (3 or more) that form a plane, and in this case is used to define the plane. Note that this option uses the same syntax as `SUBSPACE`, whereby indicating an element (e.g., H) includes all the hydrogen atoms in the list that defines the plane. The option `SUBSPACE`, is used to select all the 2p orbitals, because neither of the three directions is perpendicular to the plane. This leads to the following input section of AVAS:

```
set forte {
  subspace           ["C(2p)","O(2p)"]  # must include all 2p orbitals!
  subspace_pi_planes [["C","O","H"]]    # only one plane, defined by all C, O and H atoms
  avas               true
  avas_diagonalize   true
  avas_sigma         1.0
}
```

and the output is now identical to the very first example

```
==> Atomic Valence MOs (Active Marked by *) <==

  ================================
   Irrep    MO  Occ.  <phi|P|phi>
  --------------------------------
  *   A      0    2      0.970513
  *   A      8    0      0.992548
  *   A      9    0      0.022209
  ================================
```

Some comments on the expressions of SUBSPACE_PI_PLANES are necessary. Possible expressions to define the planes include:

```
- [['C', 'H', 'O']]              # only one plane consisting all C, H, and O atoms of␣
↪the molecule.
- [['C1-6'], ['N1-2', 'C9-11']]  # plane 1 with the first six C atoms of the molecule,
                                 # plane 2 with C atoms #9, #10, and #11, and N atoms #1␣
↪and #2.
- [['C1-4'], ['C1-2', 'C5-6']]   # plane 1 with the first four C atoms of the molecule,
                                 # plane 2 with C atoms #1, #2, #5, and #6.
                                 # Two planes share C1 and C2!
```

This syntax follows the one used by SUBSPACE:

```
- ["C"]              # all carbon atoms
- ["C","N"]          # all carbon and nitrogen atoms
- ["C1"]             # carbon atom #1
- ["C1-3"]           # carbon atoms #1, #2, #3
- ["C(2p)"]          # the 2p subset of all carbon atoms
- ["C(1s)","C(2s)"]  # the 1s/2s subsets of all carbon atoms
- ["C1-3(2s)"]       # the 2s subsets of carbon atoms #1, #2, #3
- ["Ce(4fzx2-zy2)"]  # the 4f zxx-zyy orbital of all Ce atoms
```

Essentially, SUBSPACE_PI_PLANES defines a list of planes and the code attaches each atom of the plane with the plane unit normal $\mathbf{n} = (n_x, n_y, n_z)$. A complete subset of atomic p orbitals (p_x, p_y, p_z) are projected onto that vector so that the target p orbital becomes $|p\rangle = \sum_{i\in\{x,y,z\}} n_i|p_i\rangle$. This means we attach a coefficient to every subspace orbital, where the coefficient of the p_i orbital on the atom of the plane is n_i, while the coefficient for all other subspace AOs is 1.0. The projector is then modified as

$$P_{\mu\nu} = \sum_{r's'} \langle\mu|r'\rangle(\rho^{-1})_{r's'}\langle s'|\nu\rangle, \quad r', s' \in \{\text{Target Valence Atomic Orbitals}\},$$

where $|r'\rangle = \sum_r C_{rr'}|r\rangle$ and $|r\rangle$ are the AOs from the MINAO basis set. The coefficient matrix $C_{rr'}$ is given by

$$C_{rr'} = \begin{cases} n_i, & r' \in \{p \text{ orbitals on plane atoms if planes are defined}\}, \\ 1.0, & r' \in \{\text{other AOs in the subspace chosen by the user}\}, \\ 0, & \text{otherwise.} \end{cases}$$

Note: It is very important to include a complete set of p orbitals in SUBSPACE if planes are defined. Otherwise, the code will follow the directions given by SUBSPACE.

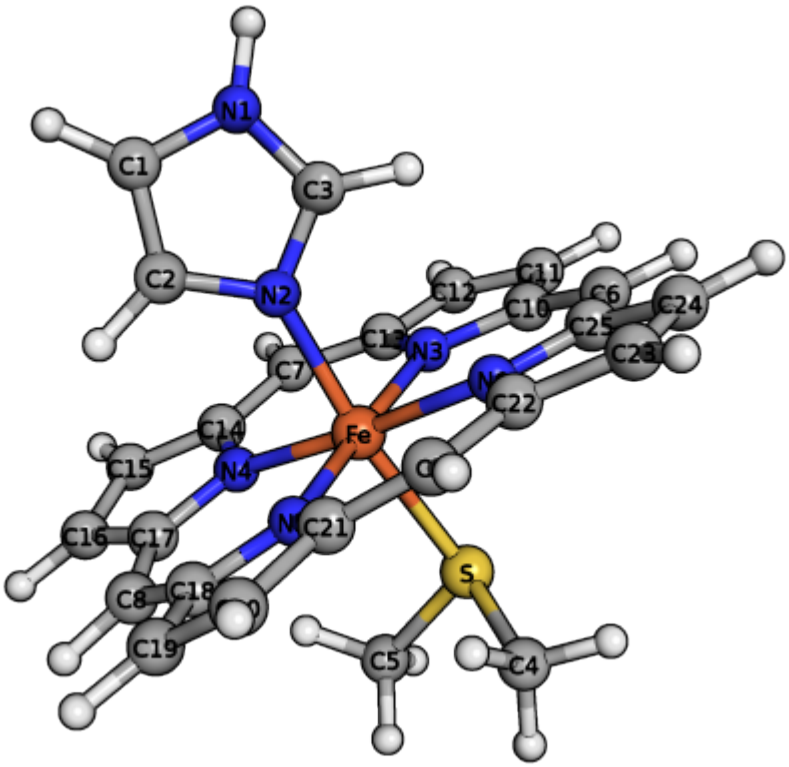
Tip: The code is flexible enough to treat double active spaces (e.g., double or double d-shell). For example, the double- active space of formaldehyde can be obtained via

```
set forte {
  minao_basis        double-shell
  subspace           ["C(2p)","C(3p)","O(2p)","O(3p)"]
  subspace_pi_planes [["C","O","H"]]
  avas               true
  avas_diagonalize   true
  avas_cutoff        0.5
}
```

Here I prepare a basis called "double-shell.gbs", which includes the 2p and 3p orbitals of C and O atoms. You can also prepare your own MINAO basis by truncating the the cc-pVTZ or ANO-RCC-VTZP basis sets.

3.6.4 Systems with multiple systems

For a more realistic example, consider the following iron porphyrin related molecule:



By checking the geometry, we see that the molecule contains two systems, namely, porphyrin and imidazole. The orbitals are perpendicular to the corresponding planes. However, the porphyrin is not a perfect plane and we assume the orbitals are perpendicular to the averaged plane formed by the porphyrin backbone. The iron 3d orbitals may interact with the orbitals of porphyrin and imidazole rings and the sulfur 3p orbitals. We would like to ask AVAS to select these orbitals as active, which can be achieved by the following

```
set forte {
  avas               true
  avas_diagonalize   true
  avas_cutoff        0.5
```

(continues on next page)

(continued from previous page)

```
  minao_basis         cc-pvtz-minao
  subspace            ["Fe(3d)","C6-25(2p)","N(2p)","S(3p)","C1-3(2p)"]
  subspace_pi_planes  [["Fe","C6-25","N3-6"], ["N1-2","C1-3"]]
}
```

Here, the porphyrin plane is defined by the iron atom, carbon atoms #6 to #25, and nitrogen atoms #3 to #6. The imidazole plane is defined by the first two nitrogen atoms and the first three carbon atoms. The atom ordering is consistent with the one used in the molecule section of the input (see the figure). The AVAS output selects exactly 37 orbitals we wanted.

```
==> AVAS MOs Information <==

  --------------------
                     A
  --------------------
  DOCC INACTIVE     106
  DOCC ACTIVE        22
  SOCC ACTIVE         0
  UOCC ACTIVE        15
  UOCC INACTIVE     462
  --------------------
  RESTRICTED_DOCC   106
  ACTIVE             37
  RESTRICTED_UOCC   462
  --------------------

==> Atomic Valence MOs (Active Marked by *) <==

  ==============================
   Irrep    MO  Occ.  <phi|P|phi>
  ------------------------------
  *   A      0    2      0.999085
  *   A      1    2      0.998642
  ...
  *   A     19    2      0.974919
  *   A     20    2      0.855068
  *   A     21    2      0.747171
      A     22    2      0.215276
      A     23    2      0.175599
  ...
      A     36    2      0.000408
  *   A    128    0      0.999163
  *   A    129    0      0.997849
  ...
  *   A    140    0      0.943277
  *   A    141    0      0.824388
  *   A    142    0      0.784721
      A    143    0      0.252635
      A    144    0      0.144740
  ...
      A    164    0      0.000898
  ==============================
```

The code is also flexible enough treat planes that share some atoms. Let's assume atom A is shared by planes P_1 and P_2 with the plane unit normals $\mathbf{n}_1$ and $\mathbf{n}_2$, respectively. The positive direction of $\mathbf{n}_i$ $(i = 1, 2)$ is taken such that $\mathbf{n}_i \cdot \mathbf{d} \geq 0$, where $\mathbf{d}$ is the vector from the centroid of the molecule to the centroid of the plane P_i. The vector attached to atom A is then a normalized vector sum given by $\mathbf{n}_{\mathrm{A}} = (\mathbf{n}_1 + \mathbf{n}_2)/||\mathbf{n}_1 + \mathbf{n}_2||$. Based on this feature, we may ask AVAS to pick the active space for C20 fullerene (see test case avas-8). For C20 fullerene, there are 12 planes forming the cage and we would like to make the target valence AOs pointing outwards the cage sphere. The planes can be specified manually or figured out using the nearest and second nearest neighbors of an atom.

3.6.5 Options

AVAS

Turn on the AVAS procedure or not.

- Type: Boolean
- Default: False

AVAS_DIAGONALIZE

Diagonalize the projected overlap matrices or not.

- Type: Boolean
- Default: True

AVAS_EVALS_THRESHOLD

Threshold smaller than which is considered as zero for an eigenvalue of the projected overlap matrices.

- Type: double
- Default: 1.0e-6

AVAS_SIGMA

Cumulative threshold to the eigenvalues of the projected overlap matrices to control the output number of active orbitals. Orbitals will be added to the active subset starting from that of the largest σ value and stopped when $\sum_u^{\mathrm{ACTIVE}} \sigma_u / \sum_p^{\mathrm{ALL}} \sigma_p$ is larger than the threshold.

- Type: double
- Default: 0.98

AVAS_CUTOFF

The threshold greater than which to the eigenvalues of the projected overlap matrices will be considered as active orbitals. If not equal to 1.0, it takes priority over the sigma threshold selection.

- Type: double
- Default: 1.0

AVAS_NUM_ACTIVE

The total number of orbitals considered as active for doubly occupied and virtual orbitals (singly occupied orbitals not included). If not equal to 0, it will take priority over the sigma or cutoff selections.

- Type: int
- Default: 0

AVAS_NUM_ACTIVE_OCC

The number of doubly occupied orbitals considered as active. If not equal to 0, it will take priority over the selection schemes based on sigma and cutoff selections and the total number of active orbitals.

- Type: int
- Default: 0

AVAS_NUM_ACTIVE_VIR

The number of virtual orbitals considered as active. If not equal to 0, it will take priority over the selection schemes based on sigma and cutoff selections and the total number of active orbitals.

- Type: int
- Default: 0

3.6.6 Citation Reference

Automated Construction of Molecular Active Spaces from Atomic Valence Orbitals J. Chem. Theory Comput. 13, 4063-4078 (2017).

3.7 Plotting MOs

Forte includes a set of utilities for plotting molecular orbitals saved in the cube file format directly from a Jupyter notebook. A good place to start is this: Tutorial_02.00_plotting_mos.ipynb.

3.7.1 Generating cube files

The `forte.utils.psi4_cubeprop` function offers a convenient way to generate cube files from the information stored in a psi4 `Wavefunction` object:

```
import forte.utils
forte.utils.psi4_cubeprop(wfn,path='cubes',nocc=4,nvir=4)
```

By default this function plots the HOMO-2 to the LUMO+2 orbitals, but in this example we specifically indicate that we want 4 occupied and 4 virtual orbitals. This function can also take a list of the orbitals to plot via the `orbs` option and it can return a set of CubeFile objects:

```
cubes = forte.utils.psi4_cubeprop(wfn, path = '.', orbs = [3,4,5,6], load = True)
```

3.7.2 Reading, manipulating, and saving cube files

Cube files can be read from disk via the `CubeFile` class. To read a cube file instantiate a `CubeFile` object by passing the file name:

```
cube = forte.CubeFile('cubes/Psi_a_15_15-A.cube')
```

From this object, we can plot the cube file or extract useful information:

```
# number of atoms
print(f'cube.natoms() -> {cube.natoms()}')

# number of grid points along each direction
print(f'cube.num() -> {cube.num()}')
```

The `CubeFile` class supports three type of operations:

`scale(double factor)`: scale all the values on the grid by `factor`
$\phi(\mathbf{r}_i) \leftarrow \texttt{factor} * \phi(\mathbf{r}_i)$

`add(CubeFile cube)`: add to this cube file the grid values stored in cube
$\phi(\mathbf{r}_i) \leftarrow \phi(\mathbf{r}_i) + \psi(\mathbf{r}_i)$

`pointwise_product(CubeFile cube)`: multiply each point of this cube file with the values stored in cube
$\phi(\mathbf{r}_i) \leftarrow \phi(\mathbf{r}_i) * \psi(\mathbf{r}_i)$

For example, we can compute the density of an orbital by taking the pointwise product with itself:

```
cube = forte.CubeFile('cubes/Psi_a_15_15-A.cube')
dens = forte.CubeFile(cube)
dens.pointwise_product(dens)
```

To write a `CubeFile` object to disk for later use just call the `save` function:

```
dens.save('cubes/dens.cube')
```

3.7.3 Plotting cube files

Forte includes a low-level 3D renderer based on `pythreejs` and a simple interface to this renderer, the `CubeViewer` class. We can tell the CubeViewer class to look for cube files in a specific path (via the path option):

```
cv = forte.utils.CubeViewer(path='cubes')
```

Alternatively, we can pass a list of cube files to load (via the cubes options). Here we specify two files and we also change the color scheme:

```
cv2 = forte.utils.CubeViewer(cubes=['cubes/Psi_a_13_13-A.cube','cubes/Psi_a_16_16-A.cube
↪'],colorscheme='electron')
```

3.7.4 Generating cube files

The `forte.utils.psi4_cubeprop` function offers a convenient way to generate cube files from the information stored in a psi4 `Wavefunction` object:

3.8 Forte Python API Tutorial (NEW)

Forte's new python API allows the user to express a calculation as a computational graph. Nodes on this graph do one of the following - Provide inputs - Take inputs from other nodes and perform a computational task

3.8.1 Creating the input node

The starting point for a Forte computation is an input object (`Input`). The input can be created via a factory function (`forte.solvers.solver_factory`)

```
from forte.solvers import solver_factory

# define the molecular geometry (H2, r = 1 Å)
zmat = """
H
H 1 1.0
"""

# create the input node using the zmat geometry and the cc-pVDZ basis set
input = solver_factory(molecule=zmat, basis='cc-pVDZ')
```

The object returned by `solver_factory` (`input`) is an input node that contains a `MolecularModel` attribute responsible for generating the Hamiltonian of this molecule/basis set combination. The `input` object can now be passed to a `Solver` node that will perform a computation.

3.8.2 Hartree–Fock theory

To run a Hartree–Fock (HF) computation on the molecule defined above the user has to do the following:

1. Specify the electronic state
2. Create a Hartree–Fock solver object
3. Call the `run()` function

Here is an example that shows the full input for a HF computation:

```
from forte.solvers import solver_factory, HF

xyz = """
H 0.0 0.0 0.0
H 0.0 0.0 1.0
"""
input = solver_factory(molecule=xyz, basis='cc-pVDZ')

# 1. singlet Ag state of H2 (neutral)
state = input.state(charge=0, multiplicity=1, sym='ag')

# 2. create the HF object
hf = HF(input, state=state)

# 3. run the computation
hf.run()
```

The output of this computation can be found in the `output.dat` file. However, the results of this computation are also stored in the HF object. For example, the HF energy can be accessed via `hf.value('hf energy')`.

3.8.3 FCI and CASCI

Forte implements several solvers that diagonalize the Hamiltonian in a (small) space of orbitals, including FCI, selected CI methods, and generalized active space (GAS). To perform one of these computations just pass an object that can provide molecular orbitals to an `ActiveSpaceSolver` object. For example, we can perform a CASCI computation on the same molecule as above by passing the `HF` node to an `ActiveSpaceSolver` node

```
from forte.solvers import solver_factory, HF, ActiveSpaceSolver

xyz = """
H 0.0 0.0 0.0
H 0.0 0.0 1.0
"""
input = solver_factory(molecule=xyz, basis='cc-pVDZ')

state = input.state(charge=0, multiplicity=1, sym='ag')

# create the HF object
hf = HF(input, state=state)

# specify the active space
# we pass an array that specifies the number of active MOs per irrep
# We use Cotton ordering, so this selects one MO from irrep 0 (Ag) and one from irrep 5␣
↪(B1u)
mo_spaces = input.mo_spaces(active=[1, 0, 0, 0, 0, 1, 0, 0])

# initialize a FCI solver and pass the HF object, the target electronic state, and the␣
↪MO space information
fci = ActiveSpaceSolver(hf, type='FCI', states=state, mo_spaces=mo_spaces)

# call run() on the FCI node
fci.run()
```

The CASCI energy can be accessed via the `value` function on the FCI object. In this case it returns a vector containing the energy of all the states computed:

```
fci.value('active space energy')[state] -> [-1.1083377195359851]
```

To compute two ^1A_g states we can simply pass a dictionary that maps states to number of desired solutions

```
fci = ActiveSpaceSolver(hf, type='FCI', states={state : 2}, mo_spaces=mo_spaces)
```

The energy of the two ^1A_g states can still be retrieved with the `value` function:

```
fci.value('active space energy')[state] -> [-1.1083377195359851, -0.2591786932627466]
```

CHAPTER

FOUR

PROGRAMMER'S MANUAL

4.1 Writing Forte's documentation

4.1.1 Location and structure of Forte's documentation

Forte uses sphinx to generate its documentation. The documentation is written in part in sphinx, with some of the content generated from Jupyter notebooks. The documentation is contained in the directory `docs`, which has the following structure:

```
docs
├── notebooks
└── source
```

`source` contains the restructured text files (rst) that are compiled by sphinx. The directory `notebooks` contains Jupyter notebooks that are used to generate some of the rst files. Restructured text file prefixed with `nb_` that live in `source` are generated from jupyter notebooks contained in the `notebooks` directory.

Note that the location of these converted jupyter notebooks reflects the relative location in the `notebooks` directory. For example, the file `docs/source/nb_00_overview.rst` is generated from the file `docs/notebooks/nb_00_overview.ipynb`.

4.1.2 Compiling the documentation

To compile the documentation on your local machine, from a terminal change to the `docs` folder and type

```
docs> make html
```

This command will run sphinx and generate the documentation in the folder `docs/build/html`. The documentation main page can be accessed via web browser using the url `docs/build/html/index.html`

4.1.3 Contributing to the documentation

To modify a section of Forte's documentation you should first identify which file to modify. If a `rst` file begins with `nb_`, then you should edit the corresponding jupyter notebook located in `docs/notebooks` or one of its subdirectories.

If you modified notebook files, you can update the corresponding rst files using the `update_rst.py` script in the `docs` directory:

```
docs> python update_rst.py
```

Since Jupyter facilitates the editing and rendering of the documentation, it is recommended to do all edits of Jupyter documents in Jupyter, and only at the end (for example, before a commit) to convert the content to rst files.

4.2 Psi4

4.2.1 Symmetry and the `Dimension` class

In Forte, the irreducible representations (irreps) of Abelian point groups are represented using a zero-based integer. The Cotton ordering of irreps is used, which can be found here. This ordering is convenient because the direct product of two irreps can be computed using the XOR operator. For example, consider ${C_2v} symmetry. if `ha``=A1 and ``hb , then their direct product can be computed as:

```
// Assume C2v symmetry
// Cotton ordering: [A1, A2, B1, B2]
int ha = 1; // 1 = 01 = A2
int hb = 3; // 3 = 11 = B2
int hab = ha ^ hb; // 10 = 2 = B1
```

4.2.2 The `Vector` and `Matrix` classes

CHAPTER

FIVE

FORTE'S PYTHON API

CHAPTER

SIX

LIST OF FORTE OPTIONS

ACI_ADD_AIMED_DEGENERATE

Add degenerate determinants not included in the aimed selection

- Type: Boolean
- Default value: True

ACI_ADD_EXTERNAL_EXCITATIONS

Adds external single excitations to the final wave function

- Type: Boolean
- Default value: False

ACI_ADD_SINGLES

Adds all active single excitations to the final wave function

- Type: Boolean
- Default value: False

ACI_APPROXIMATE_RDM

Approximate the RDMs

- Type: Boolean
- Default value: False

ACI_AVERAGE_OFFSET

Offset for state averaging

- Type: Integer
- Default value: 0

ACI_BATCHED_SCREENING

Control batched screeing

- Type: Boolean
- Default value: False

ACI_CONVERGENCE

ACI Convergence threshold

- Type: Double

- Default value: 0.000000

ACI_DIRECT_RDMS

Computes RDMs without coupling lists

- Type: Boolean
- Default value: False

ACI_ENFORCE_SPIN_COMPLETE

Enforce determinant spaces to be spin-complete

- Type: Boolean
- Default value: True

ACI_EXCITED_ALGORITHM

The excited state algorithm

- Type: String
- Default value: ROOT_ORTHOGONALIZE

ACI_EXTERNAL_EXCITATION_ORDER

Order of external excitations to add

- Type: String
- Default value: SINGLES

ACI_EXTERNAL_EXCITATION_TYPE

Type of external excitations to add

- Type: String
- Default value: ALL

ACI_EX_TYPE

Type of excited state to compute

- Type: String
- Default value: CONV

ACI_FIRST_ITER_ROOTS

Compute all roots on first iteration?

- Type: Boolean
- Default value: False

ACI_INITIAL_SPACE

The initial reference space

- Type: String
- Default value: CAS

ACI_LOW_MEM_SCREENING

Use low-memory screening algorithm

- Type: Boolean

- Default value: False

ACI_MAX_CYCLE

Maximum number of cycles

- Type: Integer
- Default value: 20

ACI_MAX_MEM

Sets max memory for batching algorithm (MB)

- Type: Integer
- Default value: 1000

ACI_MAX_RDM

Order of RDM to compute

- Type: Integer
- Default value: 1

ACI_MAX_RDM

Order of RDM to compute

- Type: Integer
- Default value: 1

ACI_MAX_RDM

Order of RDM to compute

- Type: Integer
- Default value: 1

ACI_NBATCH

Number of batches in screening

- Type: Integer
- Default value: 1

ACI_NFROZEN_CORE

Number of orbitals to freeze for core excitations

- Type: Integer
- Default value: 0

ACI_NO

Computes ACI natural orbitals

- Type: Boolean
- Default value: False

ACI_NO_THRESHOLD

Threshold for active space prediction

- Type: Double

- Default value: 0.020000

ACI_NROOT

Number of roots for ACI computation

- Type: Integer
- Default value: 1

ACI_N_AVERAGE

Number of roots to averag

- Type: Integer
- Default value: 1

ACI_PERTURB_SELECT

Type of energy selection

- Type: Boolean
- Default value: False

ACI_PQ_FUNCTION

Function for SA-ACI

- Type: String
- Default value: AVERAGE

ACI_PREITERATIONS

Number of iterations to run SA-ACI before SS-ACI

- Type: Integer
- Default value: 0

ACI_PRESCREEN_THRESHOLD

The SD space prescreening threshold

- Type: Double
- Default value: 0.000000

ACI_PRINT_NO

Print the natural orbitals

- Type: Boolean
- Default value: True

ACI_PRINT_REFS

Print the P space

- Type: Boolean
- Default value: False

ACI_PRINT_WEIGHTS

Print weights for active space prediction

- Type: Boolean

- Default value: False

ACI_PROJECT_OUT_SPIN_CONTAMINANTS

Project out spin contaminants in Davidson-Liu's algorithm

- Type: Boolean
- Default value: True

ACI_QUIET_MODE

Print during ACI procedure

- Type: Boolean
- Default value: False

ACI_REF_RELAX

Do reference relaxation in ACI

- Type: Boolean
- Default value: False

ACI_RELAX_SIGMA

Sigma for reference relaxation

- Type: Double
- Default value: 0.010000

ACI_ROOT

Root for single-state computations

- Type: Integer
- Default value: 0

ACI_ROOTS_PER_CORE

Number of roots to compute per frozen occupation

- Type: Integer
- Default value: 1

ACI_SAVE_FINAL_WFN

Print final wavefunction to file

- Type: Boolean
- Default value: False

ACI_SCALE_SIGMA

Scales sigma in batched algorithm

- Type: Double
- Default value: 0.500000

ACI_SELECT_TYPE

The energy selection criteria

- Type: String

- Default value: AIMED_ENERGY

ACI_SIZE_CORRECTION

Perform size extensivity correction

- Type: String
- Default value:

ACI_SPIN_ANALYSIS

Do spin correlation analysis

- Type: Boolean
- Default value: False

ACI_SPIN_PROJECTION

Type of spin projection

- Type: Integer
- Default value: 0

ACI_SPIN_TOL

Tolerance for S^2 value

- Type: Double
- Default value: 0.020000

ACI_STREAMLINE_Q

Do streamlined algorithm

- Type: Boolean
- Default value: False

ACI_TEST_RDMS

Run test for the RDMs

- Type: Boolean
- Default value: False

ACTIVE_REF_TYPE

Initial guess for active space wave functions

- Type: String
- Default value: CAS

AO_DSRG_MRPT2

Do AO-DSRG-MRPT2 if true (not available)

- Type: Boolean
- Default value: False

AVAS_DIAGONALIZE

Allow the users to specifydiagonalization of Socc and SvirIt takes priority over thethreshold based selection.

- Type: Boolean

- Default value: True

AVAS_NUM_ACTIVE

Allows the user to specify the total number of active orbitals. It takes priority over the threshold based selection.

- Type: Integer
- Default value: 0

AVAS_NUM_ACTIVE_OCC

Allows the user to specify the number of active occupied orbitals. It takes priority over the threshold based selection.

- Type: Integer
- Default value: 0

AVAS_NUM_ACTIVE_VIR

Allows the user to specify the number of active occupied orbitals. It takes priority over the threshold based selection.

- Type: Integer
- Default value: 0

AVAS_SIGMA

Threshold that controls the size of the active space

- Type: Double
- Default value: 0.980000

CCVV_ALGORITHM

Algorithm to compute the CCVV term in DSRG-MRPT2 (only used in three-dsrg-mrpt2 code)

- Type: String
- Default value: FLY_AMBIT
- Allowed values: CORE, FLY_AMBIT, FLY_LOOP, BATCH_CORE, BATCH_VIRTUAL, BATCH_CORE_GA, BATCH_VIRTUAL_GA, BATCH_VIRTUAL_MPI, BATCH_CORE_MPI, BATCH_CORE_REP, BATCH_VIRTUAL_REP

CCVV_BATCH_NUMBER

Batches for CCVV_ALGORITHM

- Type: Integer
- Default value: -1

CCVV_SOURCE

Special treatment for the CCVV term in DSRG-MRPT2 (used in three-dsrg-mrpt2 code)

- Type: String
- Default value: NORMAL
- Allowed values: ZERO, NORMAL

CHOLESKY_TOLERANCE

The tolerance for cholesky integrals

- Type: Double
- Default value: 0.000001

CINO

Do a CINO computation?

- Type: Boolean
- Default value: False

CINO_AUTO

Allow the users to choosewhether pass frozen_doccactice_docc and restricted_doccor not

- Type: Boolean
- Default value: False

CINO_NROOT

The number of roots computed

- Type: Integer
- Default value: 1

CINO_ROOTS_PER_IRREP

The number of excited states per irreducible representation

- Type: Array
- Default value: []

CINO_THRESHOLD

The fraction of NOs to include in the active space

- Type: Double
- Default value: 0.990000

CINO_TYPE

The type of wave function.

- Type: String
- Default value: CIS
- Allowed values: CIS, CISD

CORR_LEVEL

Correlation level of MR-DSRG (used in mrdsrg code, LDSRG2_P3 and QDSRG2_P3 not implemented)

- Type: String
- Default value: PT2
- Allowed values: PT2, PT3, LDSRG2, LDSRG2_QC, LSRG2, SRG_PT2, QDSRG2, LDSRG2_P3, QDSRG2_P3

DL_GUESS_SIZE

Set the initial guess space size for DL solver

- Type: Integer
- Default value: 100

DSRGPT

Renormalize (if true) the integrals (only used in toy code mcsrgpt2)

- Type: Boolean
- Default value: True

DSRG_DIPOLE

Compute (if true) DSRG dipole moments

- Type: Boolean
- Default value: False

DSRG_HBAR_SEQ

Evaluate H_bar sequentially if true

- Type: Boolean
- Default value: False

DSRG_MAXITER

Max iterations for MR-DSRG amplitudes update

- Type: Integer
- Default value: 50

DSRG_MRPT2_DEBUG

Excssive printing for three-dsrg-mrpt2

- Type: Boolean
- Default value: False

DSRG_MULTI_STATE

Multi-state DSRG options (MS and XMS recouple states after single-state computations)

- Type: String
- Default value: SA_FULL
- Allowed values: SA_FULL, SA_SUB, MS, XMS

DSRG_OMIT_V3

Omit blocks with >= 3 virtual indices if true

- Type: Boolean
- Default value: False

DSRG_TRANS_TYPE

DSRG transformation type

- Type: String
- Default value: UNITARY
- Allowed values: UNITARY, CC

DWMS_ALGORITHM

DWMS algorithms

- Type: String
- Default value: DWMS-0

- Allowed values: DWMS-0, DWMS-1, DWMS-AVG0, DWMS-AVG1

DWMS_ZETA

Gaussian width cutoff for the density weights

- Type: Double
- Default value: 0.000000

ESNOS

Compute external single natural orbitals

- Type: Boolean
- Default value: False

ESNO_MAX_SIZE

Number of external orbitals to correlate

- Type: Integer
- Default value: 0

FCIMO_ACTV_TYPE

The active space type

- Type: String
- Default value: COMPLETE
- Allowed values: COMPLETE, CIS, CISD, DOCI

FCIMO_CISD_NOHF

Ground state: HF; Excited states: no HF determinant in CISD space

- Type: Boolean
- Default value: True

FCIMO_IAO_ANALYSIS

Intrinsic atomic orbital analysis

- Type: Boolean
- Default value: False

FCIMO_IPEA

Generate IP/EA CIS/CISD space

- Type: String
- Default value: NONE
- Allowed values: NONE, IP, EA

FCIMO_LOCALIZE_ACTV

Localize active orbitals before computation

- Type: Boolean
- Default value: False

FCIMO_PRINT_CIVEC

The printing threshold for CI vectors

- Type: Double
- Default value: 0.050000

FCI_MAXITER

Maximum number of iterations for FCI code

- Type: Integer
- Default value: 30

FCI_MAX_RDM

The number of trial guess vectors to generate per root

- Type: Integer
- Default value: 1

FCI_NROOT

The number of roots computed

- Type: Integer
- Default value: 1

FCI_NTRIAL_PER_ROOT

The number of trial guess vectors to generate per root

- Type: Integer
- Default value: 10

FCI_PRINT_NO

Print the NO from the rdm of FCI

- Type: Boolean
- Default value: False

FCI_ROOT

The root selected for state-specific computations

- Type: Integer
- Default value: 0

FCI_TEST_RDMS

Test the FCI reduced density matrices?

- Type: Boolean
- Default value: False

FORM_HBAR3

Form 3-body Hbar (only used in dsrg-mrpt2 with SA_SUB for testing)

- Type: Boolean
- Default value: False

FORM_MBAR3

Form 3-body mbar (only used in dsrg-mrpt2 for testing)

- Type: Boolean
- Default value: False

GAMMA

The reference space selection threshold

- Type: Double
- Default value: 1.000000

H0TH

Zeroth-order Hamiltonian of DSRG-MRPT (used in mrdsrg code)

- Type: String
- Default value: FDIAG
- Allowed values: FDIAG, FFULL, FDIAG_VACTV, FDIAG_VDIAG

INTEGRAL_SCREENING

The screening for JK builds and DF libraries

- Type: Double
- Default value: 0.000000

INTERNAL_AMP

Include internal amplitudes for VCIS/VCISD-DSRG

- Type: String
- Default value: NONE
- Allowed values: NONE, SINGLES_DOUBLES, SINGLES, DOUBLES

INTERNAL_AMP_SELECT

Excitation types considered when internal amplitudes are included

- Type: String
- Default value: AUTO
- Allowed values: AUTO, ALL, OOVV

INTRUDER_TAMP

Threshold for amplitudes considered as intruders for warning

- Type: Double
- Default value: 0.100000

INT_TYPE

The integral type

- Type: String
- Default value: CONVENTIONAL
- Allowed values: CONVENTIONAL, DF, CHOLESKY, DISKDF, DISTDF, ALL, OWNINTEGRALS

ISA_B

Intruder state avoidance parameter when use ISA to form amplitudes (only used in toy code mcsrgpt2)

- Type: Double
- Default value: 0.020000

JOB_TYPE

Specify the job type

- Type: String
- Default value: NONE
- Allowed values: NONE, ACI, PCI, CAS, DMRG, SR-DSRG, SR-DSRG-ACI, SR-DSRG-PCI, TENSORSRG, TENSORSRG-CI, DSRG-MRPT2, DSRG-MRPT3, MR-DSRG-PT2, THREE-DSRG-MRPT2, SOMRDSRG, MRDSRG, MRDSRG_SO, CASSCF, ACTIVE-DSRGPT2, DWMS-DSRGPT2, DSRG_MRPT, TASKS, CC, NOJOB, DOCUMENTATION

MAXITER_RELAX_REF

Max macro iterations for DSRG reference relaxation

- Type: Integer
- Default value: 15

MINAO_BASIS

The basis used to define an orbital subspace

- Type: String
- Default value: STO-3G

MRCINO

Do a MRCINO computation?

- Type: Boolean
- Default value: False

MRCINO_AUTO

Allow the users to choosewhether pass frozen_doccactice_docc and restricted_doccor not

- Type: Boolean
- Default value: False

MRCINO_NROOT

The number of roots computed

- Type: Integer
- Default value: 1

MRCINO_ROOTS_PER_IRREP

The number of excited states per irreducible representation

- Type: Array
- Default value: []

MRCINO_THRESHOLD

The fraction of NOs to include in the active space

- Type: Double
- Default value: 0.990000

MRCINO_TYPE

The type of wave function.

- Type: String
- Default value: CIS
- Allowed values: CIS, CISD

MRPT2

Compute full PT2 energy

- Type: Boolean
- Default value: False

MS

Projection of spin onto the z axis

- Type: Double
- Default value: 0.000000

NTAMP

Number of amplitudes printed in the summary

- Type: Integer
- Default value: 15

N_GUESS_VEC

Number of guess vectors for Sparse CI solver

- Type: Integer
- Default value: 10

PCI_ADAPTIVE_BETA

Use an adaptive time step?

- Type: Boolean
- Default value: False

PCI_CHEBYSHEV_ORDER

The order of Chebyshev truncation

- Type: Integer
- Default value: 5

PCI_COLINEAR_THRESHOLD

The minimum norm of orthogonal vector

- Type: Double

- Default value: 0.000001

PCI_DL_COLLAPSE_PER_ROOT

The number of trial vector to retain after Davidson-Liu collapsing

- Type: Integer
- Default value: 2

PCI_DL_SUBSPACE_PER_ROOT

The maxim number of trial Davidson-Liu vectors

- Type: Integer
- Default value: 8

PCI_DYNAMIC_PRESCREENING

Use dynamic prescreening

- Type: Boolean
- Default value: False

PCI_ENERGY_ESTIMATE_FREQ

Iterations in between variational estimation of the energy

- Type: Integer
- Default value: 1

PCI_ENERGY_ESTIMATE_THRESHOLD

The threshold with which we estimate the variational energy. Note that the final energy is always estimated exactly.

- Type: Double
- Default value: 0.000001

PCI_EVAR_MAX_ERROR

The max allowed error for variational energy

- Type: Double
- Default value: 0.000000

PCI_E_CONVERGENCE

The energy convergence criterion

- Type: Double
- Default value: 0.000000

PCI_FAST_EVAR

Use a fast (sparse) estimate of the energy

- Type: Boolean
- Default value: False

PCI_FUNCTIONAL

The functional for determinant coupling importance evaluation

- Type: String

- Default value: MAX
- Allowed values: MAX, SUM, SQUARE, SQRT, SPECIFY-ORDER

PCI_FUNCTIONAL_ORDER

The functional order of PCI_FUNCTIONAL is SPECIFY-ORDER

- Type: Double
- Default value: 1.000000

PCI_GENERATOR

The propagation algorithm

- Type: String
- Default value: WALL-CHEBYSHEV
- Allowed values: LINEAR, QUADRATIC, CUBIC, QUARTIC, POWER, TROTTER, OLSEN, DAVIDSON, MITRUSHENKOV, EXP-CHEBYSHEV, WALL-CHEBYSHEV, CHEBYSHEV, LANCZOS, DL

PCI_GUESS_SPAWNING_THRESHOLD

The determinant importance threshold

- Type: Double
- Default value: -1.000000

PCI_INITIATOR_APPROX

Use initiator approximation

- Type: Boolean
- Default value: False

PCI_INITIATOR_APPROX_FACTOR

The initiator approximation factor

- Type: Double
- Default value: 1.000000

PCI_KRYLOV_ORDER

The order of Krylov truncation

- Type: Integer
- Default value: 5

PCI_MAXBETA

The maximum value of beta

- Type: Double
- Default value: 1000.000000

PCI_MAX_DAVIDSON_ITER

The maximum value of Davidson generator iteration

- Type: Integer
- Default value: 12

PCI_MAX_GUESS_SIZE

The maximum number of determinants used to form the guess wave function

- Type: Double
- Default value: 10000.000000

PCI_NROOT

The number of roots computed

- Type: Integer
- Default value: 1

PCI_PERTURB_ANALYSIS

Do result perturbation analysis

- Type: Boolean
- Default value: False

PCI_POST_DIAGONALIZE

Do a post diagonalization?

- Type: Boolean
- Default value: False

PCI_PRINT_FULL_WAVEFUNCTION

Print full wavefunction when finish

- Type: Boolean
- Default value: False

PCI_REFERENCE_SPAWNING

Do spawning according to reference

- Type: Boolean
- Default value: False

PCI_SCHWARZ_PRESCREENING

Use schwarz prescreening

- Type: Boolean
- Default value: False

PCI_SIMPLE_PRESCREENING

Prescreen the spawning of excitations

- Type: Boolean
- Default value: False

PCI_SPAWNING_THRESHOLD

The determinant importance threshold

- Type: Double
- Default value: 0.001000

PCI_STOP_HIGHER_NEW_LOW

Stop iteration when higher new low detected

- Type: Boolean
- Default value: False

PCI_SYMM_APPROX_H

Use Symmetric Approximate Hamiltonian

- Type: Boolean
- Default value: False

PCI_TAU

The time step in imaginary time (a.u.)

- Type: Double
- Default value: 1.000000

PCI_USE_INTER_NORM

Use intermediate normalization

- Type: Boolean
- Default value: False

PCI_USE_SHIFT

Use a shift in the exponential

- Type: Boolean
- Default value: False

PCI_VAR_ESTIMATE

Estimate variational energy during calculation

- Type: Boolean
- Default value: False

PI_ACTIVE_SPACE

Active space type

- Type: Boolean
- Default value: False

PRINT_1BODY_EVALS

Print eigenvalues of 1-body effective H

- Type: Boolean
- Default value: False

PRINT_DENOM2

Print (if true) renormalized denominators in DSRG-MRPT2

- Type: Boolean
- Default value: False

PRINT_IAOS

Print IAOs

- Type: Boolean
- Default value: True

PRINT_INTS

Print the one- and two-electron integrals?

- Type: Boolean
- Default value: False

PRINT_TIME_PROFILE

Print detailed timings in dsrg-mrpt3

- Type: Boolean
- Default value: False

PT2_MAX_MEM

Maximum size of the determinant hash (GB)

- Type: Double
- Default value: 1.000000

RELAX_REF

Relax the reference for MR-DSRG (used in dsrg-mrpt2/3, mrdsrg)

- Type: String
- Default value: NONE
- Allowed values: NONE, ONCE, TWICE, ITERATE

R_CONVERGENCE

Convergence criteria for amplitudes

- Type: Double
- Default value: 0.000001

SAVE_FINAL_WFN

Save the final wavefunction to a file

- Type: Boolean
- Default value: False

SIGMA

The energy selection threshold

- Type: Double
- Default value: 0.010000

SMART_DSRG_S

Automatic adjust the flow parameter according to denominators

- Type: String

- Default value: DSRG_S
- Allowed values: DSRG_S, MIN_DELTA1, MAX_DELTA1, DAVG_MIN_DELTA1, DAVG_MAX_DELTA1

SOURCE

Source operator used in DSRG (AMP, EMP2, LAMP, LEMP2 only available in toy code mcsrgpt2)

- Type: String
- Default value: STANDARD
- Allowed values: STANDARD, LABS, DYSON, AMP, EMP2, LAMP, LEMP2

SPIN_BASIS

Basis for spin analysis

- Type: String
- Default value: LOCAL

SPIN_MAT_TO_FILE

Save spin correlation matrix to file

- Type: Boolean
- Default value: False

SPIN_PROJECT_FULL

Project solution in full diagonalization algorithm

- Type: Boolean
- Default value: False

SUBSPACE

A list of orbital subspaces

- Type: Array
- Default value: []

T1_AMP

The way of forming T1 amplitudes (used in toy code mcsrgpt2)

- Type: String
- Default value: DSRG
- Allowed values: DSRG, SRG, ZERO

TAYLOR_THRESHOLD

Taylor expansion threshold for small denominator

- Type: Integer
- Default value: 3

THREEPDC_ALGORITHM

Algorithm for evaluating 3-body cumulants in three-dsrg-mrpt2

- Type: String
- Default value: CORE

- Allowed values: CORE, BATCH

THREE_MRPT2_TIMINGS

Detailed printing (if true) in three-dsrg-mrpt2

- Type: Boolean
- Default value: False

T_ALGORITHM

The way of forming amplitudes (DSRG_NOSEMI, SELEC, ISA only available in toy code mcsrgpt2)

- Type: String
- Default value: DSRG
- Allowed values: DSRG, DSRG_NOSEMI, SELEC, ISA

UNPAIRED_DENSITY

Compute unpaired electron density

- Type: Boolean
- Default value: False