pyopencap Documentation

pyopencap

Jun 09, 2021

CONTENTS

PyOpenCAP is the Python API for OpenCAP, an open-source software aimed at extending the functionality of quantum chemistry packages to describe resonances. PyOpenCAP uses the pybind11 library to expose C++ classes and methods, allowing calculations to be driven within a Python interpreter.

PyOpenCAP is currently capable of processing quantum chemistry data in order to perform 'perturbative' complex absorbing potential calculations on metastable electronic states. These calculations are able to extract resonance position and width at the cost of a single bound-state electronic structure calculation.

To get started, please see our *tutorial*.

If you have questions or need support, please open an issue on GitHub, or contact us directly at gayverjr@bu.edu.

PyOpenCAP is released under the MIT license.

CHAPTER

ONE

SUPPORTED PACKAGES

- OpenMolcas
- PySCF

CHAPTER

TWO

SUPPORTED POTENTIALS

- Box
- Smooth Voronoi

Please see the *keywords* section for more details.

CHAPTER

THREE

UPCOMING FEATURES

- automated trajectory analysis tools
- interface to Psi4

CHAPTER

FOUR

CONTENTS

4.1 Installation

4.1.1 Install with pip (recommended)

```
pip install pyopencap
# or
pip3 install pyopencap
```

Precompiled Python wheels are available on Pypi for almost all Linux systems and most MacOS systems, for Python versions 3.6 and later.

4.1.2 Build from source

Dependencies

Compiling PyOpenCAP from source requires first installing the following dependencies:

- C++ compiler with full C++17 language support and standard libraries (**Warning: Default Apple Clang on MacOS is not supported**)
- Python3 interpreter and development libraries: version >= 3.6
- CMake: version >= 3.12
- HDF5: hierarchical data format, version >= 1.10
- Eigen: linear algebra library, version >= 3.3

All of these dependencies are available through standard package managers such as Homebrew, Conda, and yum/apt-get on Linux.

Compiler

For Linux users, any compiler which fully supports the C++17 standard should work (e.g GCC 7.x or later). If you are unsure, try updating to the latest version of your compiler.

For Mac users, as of MacOS 10.15 Catalina, the Apple Clang provided by XCode will not work due to missing standard library features. We suggest installing the latest version of GCC (currently 10.2) from Homebrew, and then setting the following environment variables before attempting to build from source:

```
# for GCC 10 installed by brew

export CC=gcc-10
```

(continues on next page)

(continued from previous page)

```
export CXX=g++-10
```

Building the package

If your operating system/Python environment is not covered by any of our pre-built wheels, the command `pip install pyopencap` will download the tarball from Pypi and try to compile from source. You can also clone the repository and install a local version:

```
git clone https://github.com/gayverjr/opencap.git

cd opencap

pip install .
```

Compiling from source will take several minutes. To monitor your progress, you can run pip with the *–verbose* flag.

To ensure that the installation was successful, return to your home directory, start a Python shell, and type:

```
import pyopencap
```

If you cloned the repository, you can run the tests by entering the *pyopencap* directory, and running `pytest`. The following python packages are required to run the tests:

```
pip install h5py
pip install numpy
pip install pytest
pip install pyscf
```

4.2 Tutorial

This is a tutorial to get you started using PyOpenCAP. Here, we walk through the steps to generate the zeroth order Hamiltonian and the CAP matrix required to perform a perturbative CAP/XMS-CASPT2 calculation on the $^2\Pi_g$ shape resonance of N_2^-.

To follow along with the tutorial, install PyOpenCAP, clone the repository, and open a Python interpreter in the examples/pyopencap/openmolcas directory. Alternatively, copy the files "nosymm.rassi.h5" and "nosymm.out" located in the examples/opencap directory to your working directory, and set the "RASSI_FILE" and "OUTPUT_FILE" variables to the appropriate paths.

A notebook version of this tutorial can be found here.

Preliminary: Importing the module

In addition to PyOpenCAP, we'll import numpy to help us process the data. We'll also set the paths to the RASSI_FILE and OUTPUT_FILE generated by OpenMolcas, which we'll be processing to run our perturbative CAP calculation.

```
>>> import pyopencap
>>> import numpy as np
>>> RASSI_FILE = "../../opencap/nosymm.rassi.h5"
>>> OUTPUT_FILE = "../../opencap/nosymm.out"
```

Constructing the System object

The *System* object of PyOpenCAP contains the geometry and basis set information, as well as the overlap matrix. The constructor takes in a Python dictionary as an argument, and understands a specific set of keywords . There are

three equivalent ways of specifying the geometry and basis set: rassi_h5, molden, and inline. Here, we'll use the rassi_h5 file.

```
>>> sys_dict = {"molecule": "molcas_rassi","basis_file": RASSI_FILE}
>>> s = pyopencap.System(sys_dict)
>>> smat = s.get_overlap_mat()
>>> np.shape(smat)
Number of basis functions:119
(119, 119)
```

Constructing the CAP object

The CAP matrix is computed by the *CAP* object. The constructor requires a *System* object, a dictionary containing the CAP parameters, the number of states (10 in this case), and finally the string "openmolcas", which denotes the ordering of the atomic orbital basis set.

```
>>> cap_dict = {"cap_type": "box",
        "cap_x":"2.76",
        "cap_y":"2.76",
        "cap_z":"4.88",
        "Radial_precision": "14",
        "angular_points": "110"}
>>> pc = pyopencap.CAP(s,cap_dict,10,"openmolcas")
```

Parsing electronic structure data from file

The *read_data()* function can read in the effective Hamiltonian and densities in one-shot when passed a Python dictionary with the right keywords. For now, we'll retrieve the effective Hamiltonian and store it as h0 for later use.

```
>>> es_dict = {"method" : "ms-caspt2",
       "molcas_output":OUTPUT_FILE,
       "rassi_h5":     RASSI_FILE}
>>> pc.read_data(es_dict)
>>> h0 = pc.get_H()
```

Passing densities in RAM

Alternatively, one can load in the densities one at a time using the *add_tdms()* or *add_tdm()* functions. We load in the matrices from rassi.h5 using the h5py package, and then pass them as numpy arrays to the *CAP* object. In this example, the CAP matrix is made to be symmetric.

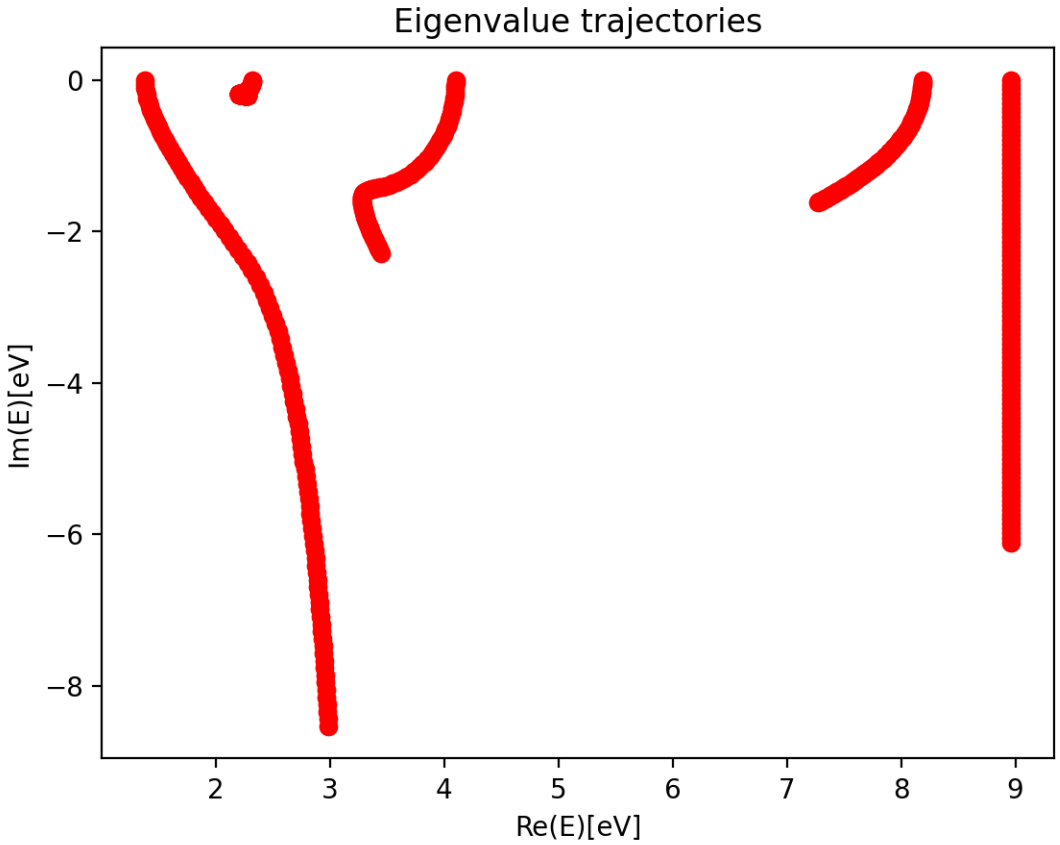
```
>>> import h5py
>>> f = h5py.File(RASSI_FILE, 'r')
>>> dms = f["SFS_TRANSITION_DENSITIES"]
>>> pc = pyopencap.CAP(s,cap_dict,10,"openmolcas")
>>> for i in range(0,10):
>>>     for j in range(i,10):
>>>         dm1 = np.reshape(dms[i][j],(119,119))
>>>         pc.add_tdm(dm1,i,j,"openmolcas",RASSI_FILE)
>>>         if i!=j:
>>>             pc.add_tdm(dm1,j,i,"openmolcas",RASSI_FILE)
```

Once all of the densities are loaded, the CAP matrix is computed using the *compute_perturb_cap()* function. The matrix can be retrieved using the *get_perturb_cap()* function.

```
>>> pc.compute_perturb_cap()
>>> W_mat=pc.get_perturb_cap()
```

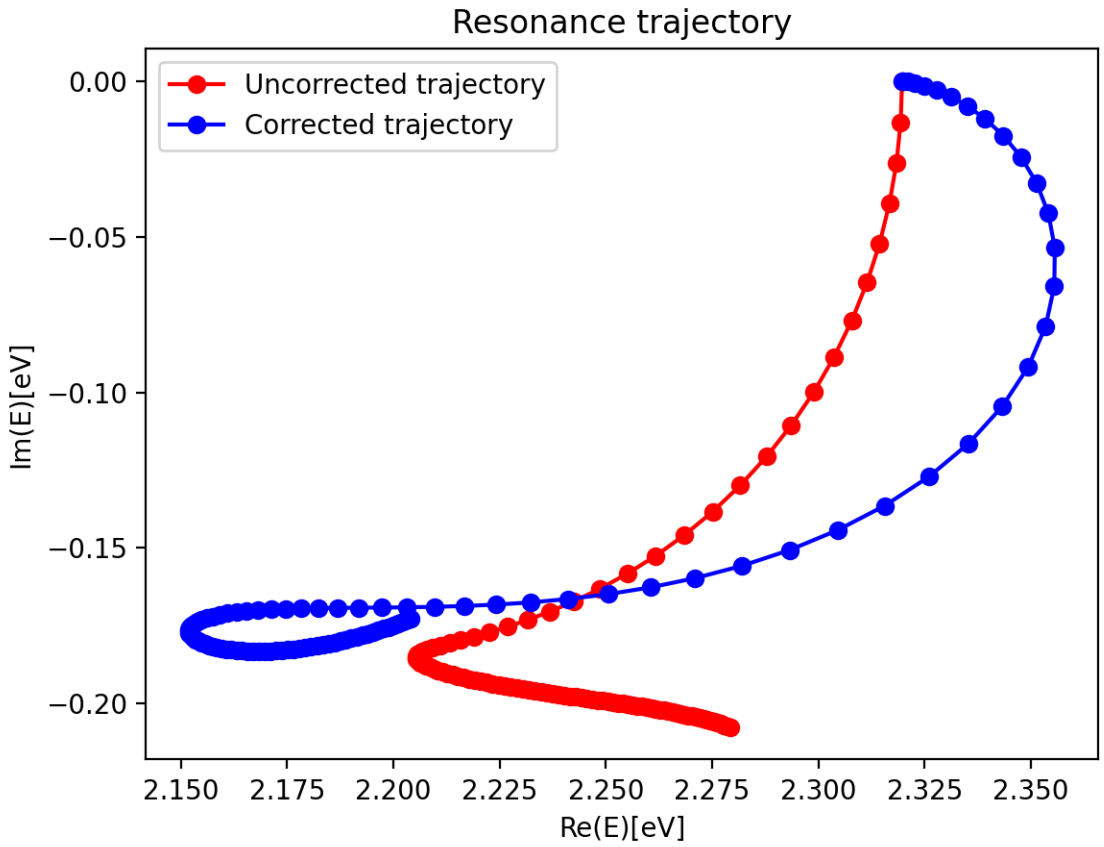
We now have our zeroth order Hamiltonian (stored in h0) and our CAP matrix(W_mat) in the state basis. Extracting resonance position and width requires analysis of the eigenvalue trajectories.

The script example.py runs this example and diagonalizes the CAP-augmented Hamiltonian $H^{CAP} = H_0 - i\eta W$ over a range of η-values. The reference energy was obtained in a separate calculation which computed the ground state of the neutral molecule with CASCI/CASPT2 using the optimized orbitals of the anionic state. The results are plotted below:



The resonance trajectory will vary slowest with the changing CAP strength. Zooming in on the trajectory near 2.2eV, we also plot the "corrected" trajectory, which is obtained by applying the first-order correction:

$U(\eta) = E(\eta) - \eta \frac{\partial E(\eta)}{\partial \eta}$.



Finally, the best estimate of resonance position and width are obtained at the stationary point

$\eta_{opt} = min \left| \eta^2 \frac{\partial^2 E}{\partial \eta^2} \right|$.

For this example, this yields a resonance energy of 2.15eV, and a width of 0.35eV.

4.3 Theory

4.3.1 Resonances and Non-Hermitian Quantum Mechanics

Electronic resonances are metastable electronic states with finite lifetimes embedded in the ionization/detachment continuum. Common examples include temporary anions formed by electron attachment, and core-excited and core-ionized states which can undergo Auger decay or similar relaxation pathways. These states are not part of the usual L^2 Hilbert space of square integrable functions, and instead belong to the continuous spectrum of the electronic Hamiltonian. Theoretical description of resonances is generally not possible by means of conventional bound-state quantum chemistry methods, and special techniques are required to obtain accurate energies and lifetimes.

Non-Hermitian quantum mechanics (NHQM) techniques provide an attractive approach that enables adaptation of existing quantum chemistry methodologies to treat metastable electronic states. In NHQM formalisms, a resonance appears as a single square-integrable eigenstate of a non-Hermitian Hamiltonian, associated with a with a complex eigenvalue:

$$E = E_{res} - i\Gamma/2.$$

The real part of the energy (E_{res}) is the resonance position. The imaginary part ($\Gamma/2$) is the half-width, which is inversely proportional to the lifetime of the state [Reinhardt1982].

4.3.2 Complex Absorbing Potential

Complex absorbing potentials (CAPs) are imaginary potentials added to the Hamiltonian, and they are routinely used for evaluation of resonance parameters. In this context, CAPs transform a resonance into a single square integrable state, rendering it accessible by means of standard bound-state techniques. To this end, the electronic Hamiltonian is augmented with an imaginary potential:

$$H^{CAP} = H - i\eta W$$

where η is the CAP strength parameter, and W is a real potential which vanishes in the vicinity of the molecular system and grows with distance [Riss1993].

Since the CAP-augmented Hamiltonian depends on the strength of the CAP, a choice has to be made on the optimal value of η that provides best estimates of the resonance position and width. In a complete one-electron basis, the exact resonance position and width are obtained in the limit of an infinitesimally weak CAP ($\eta \rightarrow 0^+$). In practice when finite bases are used, an optimal CAP strength η_{opt} is found by locating a stationary point on the eigenvalue trajectory E(η). A commonly used criterion is the minimum of the logarithmic velocity ($|\eta \frac{dE}{d\eta}| \rightarrow min$) [Riss1993].

4.3.3 Perturbative or "Projected" CAP

There are multiple strategies for how to incorporate CAPs into an electronic structure calculation. The most straightforward implementation is to engage the one-electron CAP term starting at the lowest level of theory (e.g. Hartree-Fock). While conceptually simple, this requires modification of electronic structure routines to handle the complex objects. Additionally, this approach requires a unique calculation for each η along the eigenvalue trajectory, which can become prohibitively expensive for larger systems or dynamical simulation.

An efficient alternative is to treat the CAP as a first order perturbation, considering only a small subset of the eigenstates of the real Hamiltonian [Sommerfeld2001]. In this case, the CAP will be introduced in the basis of the reduced subset of states:

$$W_{uv} = \langle u|W|v\rangle$$

where u and v are eigenstates of the real Hamiltonian. Since the CAP is a one-particle operator, these expressions can easily be evaluated using the CAP matrix in atomic orbital basis evaluated separately, the one-electron reduced density matrices (ρ) for each state, and the set of transition density matrices (γ) between each pair of states that are obtained from the bound-state calculation.

$$W_{uv} = \begin{Bmatrix} Tr\left[W^{AO}\gamma^{uv}\right], & u \neq v \\ Tr\left[W^{AO}\rho^{u}\right], & u = v \end{Bmatrix}$$

Once CAP matrix is evaluated the CAP-augmented Hamiltonian is constructed as follows:

$$H^{CAP} = H_0 - i\eta W$$

where H_0 is an appropriate zeroth order Hamiltonian obtained from the electronic structure calculation, and W is the CAP represented in the subspace. Diagonalization of this CAP-augmented Hamiltonian yields η-depdendent eigenvalues that are used to extract resonance position and width. Importantly, as only a small number of states in considered (typically less than 30), finding the eigenvalues of the CAP-augmented Hamiltonian has negligible cost in comparison to the bound-state electronic structure calculation required to get the initial set of states (u,v,..). Thus, although this perturbative or projected approach introduces another parameter (number of eigenstates), the overall cost is essentially reduced to that of a single electronic structure calculation.

With the zeroth order Hamiltonian and the CAP matrix, eigenvalue trajectories can be generated by means of simple external scripts, and estimates of resonances positions and widths can be obtained from analysis of the trajectories.

4.3.4 References

4.4 Interfaces

PyOpenCAP officially supports interfaces with the OpenMolcas and PySCF software packages. For Q-Chem developers, we have also developed an interface with the EOM-CCSD methods in Q-Chem. Please contact us directly by writing to gayverjr@bu.edu if you are interested in using OpenCAP in tandem with Q-Chem.

4.4.1 OpenMolcas

OpenMolcas is an open-source quantum chemistry package which specializes in multiconfigurational approaches to electronic structure. OpenMolcas can be used in tandem with PyOpenCAP to perform complex absorbing potential (extended)multi-state complete active space second order perturbation theory [**CAP/(X)MS-CASPT2**] calculations, which have been shown to yield accurate energies and lifetimes for metastable electronic states. Here, we outline the steps of performing these calculations using OpenMolcas and PyOpenCAP. Some suggested readings are provided at the bottom of the page.

Preliminary: Prepare input orbitals

As with any multi-reference calculation, the choice of active space is crucial for CAP/(X)MS-CASPT2, and is most often guided by chemical intuition. We refer the reader to the OpenMolcas manual for how to prepare input orbitals for a state-averaged RASSCF calculation.

Step 1: Running the OpenMolcas calculation

State-averaged RASSCF

In order to utilize the Perturbative CAP approach, a multi-state excited state calculation must be performed. In the RASSCF module, the keyword 'CIROOT' is used to activate state-averaged RASSCF calculations.

```
&RASSCF
CIROOT = 10 10 1
```

Export transition densities with RASSI

To generate the one-particle densities required to construct the CAP matrix, the RASSI module must be executed with the TRD1 keyword activated. This keyword saves one-particle transition density matrices between each pair of RASSCF states as well as the one-particle density matrices for each state to a file titled $Jobname.rassi.h5.

```
&RASSI
TRD1
```

Generate effective Hamiltonian with (X)MS-CASPT2

The (X)MS-CASPT2 approach is required to generate an appropriate zeroth Hamiltonian for the perturbative CAP method. To activate (X)MS-CASPT2 in OpenMolcas, use the Multistate keyword in the CASPT2 module.

```
&CASPT2
Multistate = all
# or
Xmultistate = all
```

Reference energy

There are multiple strategies for obtaining the reference energy used to define the resonance position. For anionic resonances, one such strategy is to add an additional diffuse orbital to the active space in order to mimic ionization, which obtains the resonance and the ground state of the neutral molecule in a single calculation [Kunitsa2017]. Another strategy (which was used in the *tutorial*) is to calculate the ground state of the neutral molecule with CASCI/CASPT2 using the optimized orbitals of the anionic state.

Step 2: Importing the data to PyOpenCAP

System object

To run a PyOpenCAP calculation, the geometry and basis set must be imported into a *System* object. The constructor takes in a Python dictionary as an argument. The relevant keywords are discussed here, and more information is provided in the *keywords* page.

Rassi.h5

The rassi.h5 file which contains the one-particle densities also contains the geometry and basis set information. To read in from rassi, "molcas_rassi" must set as the value to the key "molecule", and the path to the file must be set as the value to the key "basis_file". Here is an example:

```
sys_dict = {"molecule": "molcas_rassi","basis_file": "path/to/rassi.h5"}
my_system = pycap.System(sys_dict)
```

Molden

Molden files generated by OpenMolcas contain the geometry and basis set information. To read in from molden, "molden" must be set as the value to the key "molecule", and the path to the file must be set as the value to the key "basis_file". Here is an example:

```
sys_dict = {"molecule": "molden","basis_file": "path/to/file.molden"}
my_system = pycap.System(sys_dict)
```

Inline(not recommended)

The molecule and basis set can also be specified manually. The "molecule" keyword must be set to "read", and then an additional keyword "geometry:" must be specified, with a string that contains the geometry in xyz format. The "basis_file" keyword must be set to a path to a basis set file formatted in Psi4 style, which can be downloaded from the MolSSI BSE. Other optional keyword for this section include "bohr_coordinates" and cart_bf. Please see the *keywords* section for more details. Up to G-type functions are supported.

```
sys_dict = {"geometry":    '''N  0  0   1.039
                           N  0  0   -1.039
                           X   0  0   0.0''',
                     "molecule" : "read",
                     "basis_file":"path/to/basis.bas",
                     "cart_bf":"d",
                     "bohr_coordinates:": "true"}
my_system = pycap.System(sys_dict)
```

One particle densities/zeroth order Hamiltonian

The CAP matrix is computed by the *CAP* object. The constructor requires a *System*, a dictionary containing the CAP parameters, the number of states, and finally the string "openmolcas", which denotes the ordering of the atomic orbital basis set. An example is provided below. Please see the keywords section for more information on the CAP parameters.

```
cap_dict = {"cap_type": "box",
            "cap_x":"2.76",
            "cap_y":"2.76",
            "cap_z":"4.88",
            "Radial_precision": "14",
            "angular_points": "110"}
pc = pycap.CAP(my_system,cap_dict,10,"openmolcas")
```

Before we can compute the CAP matrix in the state basis, we must load in the density matrices. There are two ways of doing this. The first is to use the *read_data()* function. As shown below, we define a dictionary which contains the following keys: "method" (electronic structure method chosen), "rassi_h5"(density matrices), and "molcas_output"(output file containing effective Hamiltonian). The effective Hamiltonian can be retrieved using the *get_H()* function of the *CAP* object. Currently, only the effective Hamiltonians from (X)MS-CASPT2 calculations can be parsed from an OpenMolcas output file. We note that when *read_data()* is used, our code symmetrizes the CAP matrix in the state basis.

```
es_dict = {"method" : "ms-caspt2",
       "molcas_output":"path/to/output.out",
       "rassi_h5":"path/to/rassi.h5"}
pc.read_data(es_dict)
# save the effective Hamiltonian for later use
h0 = pc.get_H()
```

Alternatively, one can load in the densities one at a time using *add_tdm()*. In our examples below, we load in the matrices from rassi.h5 using the h5py package, and then pass them as numpy arrays to the *CAP* object.

```
import h5py
f = h5py.File('path/to/rassi.h5', 'r')
dms = f["SFS_TRANSITION_DENSITIES"]
# spin traced
nbasis = int(np.sqrt(dms.shape[2]))
for i in range(0,10):
    for j in range(i,10):
        dm = 0.5*np.reshape(dms[i][j],(nbasis,nbasis))
        pc.add_tdm(dm,i,j,"openmolcas","path/to/rassi.h5")
        # usually a good idea to symmetrize
        if i!=j:
            pc.add_tdm(dm,,j,i,"openmolcas","path/to/rassi.h5")
```

Step 3: Computing the CAP matrix

Once all of the densities are loaded, the CAP matrix is computed using *compute_perturb_cap()*. The matrix can be retrieved using *get_perturb_cap()*.

```
pc.compute_perturb_cap()
W_mat=pc.get_perturb_cap()
```

Note:

When using cartesian d, f, or g-type basis functions, special care must be taken to ensure that the normalization conventions match what is used by OpenMolcas. In these cases, *compute_ao_cap()* and then *renormalize()* or *renormalize_cap()* should be invoked before calling *compute_perturb_cap()*.

```
pc.compute_ao_cap()
pc.renormalize()
pc.compute_perturb_cap()
```

Step 4: Generate eigenvalue trajectories

Extracting resonance position and width requires analysis of the eigenvalue trajectories. Template scripts are provided in the repository. Development of automated tools for trajectory analysis is a subject of future work.

Officially supported methods

The following methods have been benchmarked, and the *read_data()* function is capable of parsing output files to obtain the zeroth order Hamiltonian.

- MS-CASPT2
- XMS-CASPT2

Untested (use at your own risk!)

The following methods are capable of dumping densities using the TRD1 keyword of the RASSI module, but have not been benchmarked for any systems, and the zeroth order Hamiltonian cannot be parsed from the output file using the *read_data()* function. Use at your own caution, and please contact us if you find success using any of these methods so we can add official support!

- (QD/SS)DMRG-(PC/SC)NEVPT2
- SS-CASPT2
- MC-PDFT

Suggested reading

4.4.2 PySCF

PySCF is an ab initio computational chemistry program natively implemented in Python. The major advantage of using Pyscf in tandem with OpenCAP is that calculations can be performed in one-shot within the same python script. Since PySCF allows direct control over data structures such as density matrices, the interface between PySCF and OpenCAP is seamless. Currently, only FCI has been benchmarked, and here we outline how to perform a calculation using this module.

Preliminary: Running the PySCF calculation

Please consult the PySCF documentation for how run calculations with PySCF. An example script using FCI is provided in our repository. For FCI, the zeroth order Hamiltonian is a diagonal matrix whose entries are the energies of the FCI states.

Step 1: Defining the System object

Molden(recommended)

The best way to construct the *System* object is to import the geometry and basis set from molden.

```
molden_dict = {"basis_file":"molden_in.molden","molecule": "molden"}
pyscf.tools.molden.from_scf(myhf,"molden_in.molden")
s = pyopencap.System(molden_dict)
```

Inline

The molecule and basis set can also be specified inline. The "molecule" keyword must be set to "read", and then an additional keyword "geometry" must be specified, with a string that contains the geometry in xyz format. The "basis_file" keyword must be set to a path to a basis set file formatted in Psi4 style, which can be downloaded from the MolSSI BSE. Other optional keyword for this section include "bohr_coordinates" and "cart_bf". Please see the *keywords* section for more details. It is recommended to check the overlap matrix to ensure that the ordering and normalization matches. Up to G-type functions are supported.

```
pyscf_smat = scf.hf.get_ovlp(mol)
sys_dict = {"geometry":    '''N  0  0   1.039
                           N  0  0   -1.039
                           X   0  0   0.0''',
                      "molecule" : "read",
                      "basis_file":"path/to/basis.bas",
                      "cart_bf":"d",
                      "bohr_coordinates:": "true"}
s.check_overlap_mat(pyscf_smat,"pyscf")
```

Step 1: Defining the CAP object

The CAP matrix is computed by the *CAP* object. The constructor requires a *System* object, a dictionary containing the CAP parameters, the number of states, and finally the string "pyscf", which denotes the ordering of the atomic orbital basis set. An example is provided below. Please see the keywords section for more information on the CAP parameters.

```
cap_dict = {"cap_type": "box",
            "cap_x":"2.76",
            "cap_y":"2.76",
            "cap_z":"4.88",
            "Radial_precision": "14",
            "angular_points": "110"}
pc = pycap.CAP(my_system,cap_dict,10,"pyscf")
```

Step 2: Passing the density matrices

For FCI and related modules, transition densities can be obtained using the `trans_rdm1()` function of the FCI module:

```
fs = fci.FCI(mol, myhf.mo_coeff)
e, c = fs.kernel()
# tdm between ground and 1st excited states
dm1 = fs.trans_rdm1(fs.ci[0],fs.ci[1],myhf.mo_coeff.shape[1],mol.nelec)
```

Importantly, trans_rdm1 returns the density matrix in **MO basis**. Thus before passing it to PyOpenCAP, it **must be transformed into AO basis**:

```
dm1_ao = np.einsum('pi,ij,qj->pq', myhf.mo_coeff, dm1, myhf.mo_coeff.conj())
```

Densities are loaded in one at a time using *add_tdm()*. Ensure that the indices of each state match those of the zeroth order Hamiltonian.

```
for i in range(0,len(fs.ci)):
    for j in range(0,len(fs.ci)):
```

(continues on next page)

(continued from previous page)

```
        dm1 = fs.trans_rdm1(fs.ci[i],fs.ci[j],myhf.mo_coeff.shape[1],mol.nelec)
        dm1_ao = np.einsum('pi,ij,qj->pq', myhf.mo_coeff, dm1, myhf.mo_coeff.conj())
        pc.add_tdm(dm1_ao,i,j,"pyscf")
```

Step 3: Computing the CAP matrix

Once all of the densities are loaded, the CAP matrix is computed using the *compute_perturb_cap()* function. The matrix can be retrieved using the *get_perturb_cap()* function.

```
pc.compute_perturb_cap()
W_mat=pc.get_perturb_cap()
```

Note:

When using cartesian d, f, or g-type basis functions, special care must be taken to ensure that the normalization conventions match what is used by OpenMolcas. In these cases, *compute_ao_cap()* and then *renormalize()* or *renormalize_cap()* should be invoked before calling *compute_perturb_cap()*.

```
pc.compute_ao_cap()
pc.renormalize_cap(pyscf_smat,"pyscf")
pc.compute_perturb_cap()
```

Step 4: Generate eigenvalue trajectories

Extracting resonance position and width requires analysis of the eigenvalue trajectories. A template trajectory analysis script is provided in the repository. Development of automated tools for trajectory analysis is a subject of future work.

Officially supported methods

- Full CI

Coming (hopefully) soon

- EOM-CCSD
- ADC

Untested (use at your own risk!)

Any module which one particle transition densities available can be supported. This includes all methods which can utilize the trans_rdm1 function, including but not limited to:

- MRPT
- CISD
- TDDFT

4.5 API

PyOpenCAP exposes two OpenCAP classes to Python: *System* and *CAP*. All data returned by the methods of these objects is passed as a copy, and the underlying C++ code retains ownership of the original data. Data passed to these objects from Python is passed by reference, but is not modified in any way by the underlying C++ code.

4.5.1 pyopencap.System

The System class is used to store the molecular geometry and the basis set. Upon construction, it automatically computes the overlap matrix which can be accessed and used to verify the the ordering of the atomic orbital basis set.

class pyopencap.**System**

__init__(*self: pyopencap.pyopencap_cpp.System*, *arg0: dict*) → None
Constructs System object.

get_overlap_mat(*self: pyopencap.pyopencap_cpp.System*) → numpy.ndarray[float64[m, n]]
Returns overlap matrix.

check_overlap_mat(*self: pyopencap.pyopencap_cpp.System*, *smat: numpy.ndarray[float64[m, n]]*, *ordering: str*, *basis_file: str = ''*) → bool
Compares input overlap matrix to internal overlap to check basis set ordering.

get_basis_ids(*self: pyopencap.pyopencap_cpp.System*) → str
Returns a string of the basis function ids. Each ID has the following format:atom index,shell number,l,m

4.5.2 pyopencap.CAP

The *CAP* class is used to compute the CAP matrix first in AO basis, and then in wave function basis using the one-particle densities which are passed in. It is also capable of parsing OpenMolcas output files to obtain the zeroth order Hamiltonian and return it to the user.

class pyopencap.**CAP**

__init__(*self: pyopencap.pyopencap_cpp.CAP*, *arg0: pyopencap.pyopencap_cpp.System*, *arg1: dict*, *arg2: int*, *arg3: str*) → None
Constructs CAP object.

add_tdm(*self: pyopencap.pyopencap_cpp.CAP*, *tdm: numpy.ndarray[float64[m, n]]*, *initial_idx: int*, *final_idx: int*, *ordering: str*, *basis_file: str = ''*) → None
Adds spin-traced tdm to CAP object at specified indices. The optional argument basis_file is required when using the OpenMolcas interface, and it must point to the path to the rassi.5 file.

add_tdms(*self: pyopencap.pyopencap_cpp.CAP*, *alpha_density: numpy.ndarray[float64[m, n]]*, *beta_density: numpy.ndarray[float64[m, n]]*, *initial_idx: int*, *final_idx: int*, *ordering: str*, *basis_file: str = ''*) → None
Adds alpha/beta tdms to CAP object at specified indices. The optional argument basis_file is required when using the OpenMolcas interface, and it must point to the path to the rassi.5 file.

compute_ao_cap(*self: pyopencap.pyopencap_cpp.CAP*) → None
Computes CAP matrix in AO basis.

compute_perturb_cap(*self: pyopencap.pyopencap_cpp.CAP*) → None
Computes CAP matrix in state basis using transition density matrices.

get_H (*self: pyopencap.pyopencap_cpp.CAP*) → numpy.ndarray[float64[m, n]]
Returns zeroth order Hamiltonian read from file.

get_ao_cap (*self: pyopencap.pyopencap_cpp.CAP*) → numpy.ndarray[float64[m, n]]
Returns CAP matrix in AO basis.

get_perturb_cap (*self: pyopencap.pyopencap_cpp.CAP*) → numpy.ndarray[float64[m, n]]
Returns CAP matrix in state basis.

read_data (*self: pyopencap.pyopencap_cpp.CAP*, *es_dict: dict*) → None
Reads electronic structure data specified in dictionary.

renormalize (*self: pyopencap.pyopencap_cpp.CAP*) → None
Re-normalizes AO CAP using electronic structure data.

renormalize_cap (*self: pyopencap.pyopencap_cpp.CAP*, *smat: numpy.ndarray[float64[m, n]]*, *ordering: str*, *basis_file: str = ''*) → None
Re-normalizes AO CAP matrix using input overlapmatrix.

4.6 Keywords

PyOpenCAP uses Python dictionaries which contain key/value pairs to specify the parameters of the calculation. Here, we outline the valid key/value combinations. Importantly, **all key value pairs should be specified as strings**.

4.6.1 System keywords

The *System* object contains the basis set and geometry information, which can be obtained in a few different ways.

Keyword	Required	Valid values	Description
molecule	yes	molden,qchem_fchk,rassi_h5,inline	Specifies which format to read the molecular geometry. If "inline" is chosen, the "geometry" keyword is also required.
geometry	no	See below	Specifies the geometry in an inline format described below. Required when the "molecule" field is set to "inline".
basis_file	yes	path to basis file	Specifies the path to the basis file. When "molecule" is set to "molden","rassi_h5", or "qchem_fchk", this field should be set to a path to a file of the specified type. When "molecule" is set to "inline", this field should be set to a path to a basis set file formatted in "Psi4" style.
cart_bf	no	'd', 'df', 'dfg' 'dg', 'f', 'g', 'fg'	Controls the use of pure or Cartesian angular forms of GTOs. The letters corresponding to the angular momenta listed in this field will be expanded in cartesians, those not listed will be expanded in pure GTOs. For example, "df" means d and f-type functions will be cartesian, and all others will be pure harmonic. This keyword is only active when "molecule" is set to "inline".
bohr_coordinates	no	"True" or "False"	Set this keyword to true when the coordinates specified in "geometry" keyword are in bohr units. This keyword is only active when "molecule" is set to "inline".

When specifying the geometry inline, use the following format:

```
atom1 x-coordinate y-coordinate z-coordinate

atom2 x-coordinate y-coordinate z-coordinate ...
```

Ghost centers with zero nuclear charge can be specified using the symbol "X".

Units are assumed to be Angstroms unless the bohr_coordinates keyword is set to True.

Example:

```
sys_dict = {"geometry":    '''N  0  0   1.039
                           N  0  0   -1.039
                           X   0  0   0.0''',
                       "molecule" : "read",
                       "basis_file":"path/to/basis.bas",
                       "cart_bf":"d",
                       "bohr_coordinates:": "true"}
```

4.6.2 CAP keywords

PyOpenCAP supports Voronoi and Box-type absorbing potentials. We also allow some customization of the numerical grid used for integration. Please see https://github.com/dftlibs/numgrid for more details on the radial_precision and angular_points keywords.

General Keywords

Keyword	Required	Default/valid values	Description
cap_type	yes	"box" or "voronoi"	Type of absorbing potential.
radial_precision	no	"14"	Radial precision for numerical integration grid. A precision of 1x10^(-N), where N is the value specified is used.
angular_points	no	"590"	Number of angular points used for the grid. See https://github.com/dftlibs/numgrid for allowed numbers of points.

Box CAP

A quadratic potential which encloses the system in a 3D rectangular box.

$$W = W_x + W_y + W_z$$

$$W_\alpha = \begin{Bmatrix} 0 & |r_\alpha| < R_\alpha^0 \\ (r_\alpha - R_\alpha^0)^2 & |r_\alpha| > R_\alpha^0 \end{Bmatrix}$$

Keyword	Description
cap_x	Onset of CAP in x-direction. Specify in bohr units.
cap_y	Onset of CAP in y-direction. Specify in bohr units.
cap_y	Onset of CAP in z-direction. Specify in bohr units.

Smooth Voronoi CAP

A quadratic potential which uniformly wraps around the system at a specified cutoff radius. The edges between between Voronoi cells are smoothed out to make the potential more amenable to numerical integration [Sommerfeld2015].

$$W(\vec{r}) = \begin{Bmatrix} 0 & r_{WA} \leq r_{cut} \\ (r_{WA} - r_{cut})^2 & r_{WA} > r_{cut} \end{Bmatrix}$$

$$r_{WA}(\vec{r}) = \sqrt{\frac{\sum_i w_i |\vec{r} - \vec{R}_i|^2}{\sum_i w_i}}$$

$$w_i = \frac{1}{(|\vec{r} - \vec{R}_i|^2 - r_{min}^2 + 1a.u.)^2}$$

$$r_{min} = \min_i |\vec{r} - \vec{R}_i|$$

Keyword	Description
r_cut	Cutoff radius for Voronoi CAP. Specify in bohr units.

Example

```
cap_dict = {"cap_type": "box",
            "cap_x":"2.76",
            "cap_y":"2.76",
            "cap_z":"4.88",
            "Radial_precision": "14",
            "angular_points": "110"}
```

Electronic structure keywords

The *read_data()* function is able to parse the zeroth order Hamiltonian and load the densities when supplied with an appropriate formatted dictionary. All keywords must be specified to use this function. Currently, this is only supported for calculations using the OpenMolcas interface.

Keyword	Description
method	Electronic structure method used in the calculation. Valid options are "MS-CASPT2" and "XMS-CASPT2".
molcas_output	Path to OpenMolcas output file.
rassi_h5	Path to OpenMolcas rassi.h5 file.

Example:

```
es_dict = {"method" : "ms-caspt2",
       "molcas_output":"path/to/output.out",
       "rassi_h5":"path/to/rassi.h5"}
pc.read_data(es_dict)
```

4.7 References

BIBLIOGRAPHY

[Riss1993] Riss, U. V.; Meyer, H. D. Calculation of Resonance Energies and Widths Using the Complex Absorbing Potential Method. *J. Phys. B At. Mol. Opt. Phys.* **1993**, 26 (23), 4503–4535.

[Sommerfeld2001] Sommerfeld, T.; Santra, R. Efficient Method to Perform CAP/CI Calculations for Temporary Anions. *Int. J. Quantum Chem.* **2001**, 82 (5), 218–226.

[Reinhardt1982] Reinhardt, W. P. Complex Coordinates in the Theory of Atomic and Molecular Structure and Dynamics. *Annu. Rev. Phys. Chem.* **1982**, 33 (1), 223–255.

[Phung2020] Phung, Q. M.; Komori, Y.; Yanai, T.; Sommerfeld, T.; Ehara, M. Combination of a Voronoi-Type Complex Absorbing Potential with the XMS-CASPT2 Method and Pilot Applications. *J. Chem. Theory Comput.* **2020**, 16 (4), 2606–2616.

[Kunitsa2017] Kunitsa, A. A.; Granovsky, A. A.; Bravaya, K. B. CAP-XMCQDPT2 Method for Molecular Electronic Resonances. *J. Chem. Phys.* **2017**, 146 (18), 184107.

[Al-Saadon2019] Al-Saadon, R.; Shiozaki, T.; Knizia, G. Visualizing Complex-Valued Molecular Orbitals. *J. Phys. Chem. A* **2019**, 123 (14), 3223–3228.

[Sommerfeld2015] Sommerfeld, T.; Ehara, M. Complex Absorbing Potentials with Voronoi Isosurfaces Wrapping Perfectly around Molecules. *J. Chem. Theory Comput.* **2015**, 11 (10), 4627–4633.

INDEX

Symbols

A

C

G

R

S