OPENFERMION/Q-CHEM VERSION 1.0

Yongbin Kim and Anna I. Krylov

Department of Chemistry
University of Southern California

OUTLINE

This document includes:

- Part1: General description of OpenFermion/Q-CHEM interface
- Part2: Working procedure of OpenFermion/Q-CHEM interface
  1. Perform electronic structure calculations using Q-CHEM
  2. Save molecular information of Q-CHEM into MolecularData object via OpenFermion/Q-CHEM plugin
  3. Construct, perform, and analyze quantum calculations via OpenFermion library

Part 1

OPENFERMION/Q-CHEM VERSION 1.0

GENERAL DESCRIPTION

OPENFERMION/Q-CHEM (V 1.0)

- OpenFermion[1] is an open-source library (licensed under Apache 2) for compiling and analyzing quantum algorithms which simulate fermionic systems.
- OpenFermion/Q-CHEM library allows the electronic structure package Q-CHEM[2] (Q-CHEM INC.) to interface with OpenFermion.

[1]Jarrod R McClean, Nicholas C Rubin, Kevin J Sung, Ian D Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, E Schuyler Fried, Craig Gidney, Brendan Gimby, Pranav Gokhale, Thomas Häner, Tarini Hardikar, Vojtěch Havlíček, Oscar Higgott, Cupjin Huang, Josh Izaac, Zhang Jiang, Xinle Liu, Sam McArdle, Matthew Neeley, Thomas O'Brien, Bryan O'Gorman, Isil Ozfidan, Maxwell D Radin, Jhonathan Romero, Nicolas P D Sawaya, Bruno Senjean, Kanav Setia, Sukin Sim, Damian S Steiger, Mark Steudtner, Qiming Sun, Wei Sun, Daochen Wang, Fang Zhang, and Ryan Babbush *OpenFermion: The Electronic Structure Package for Quantum Computers*. Quantum Science and Technology 5.3 (2020): 034014.

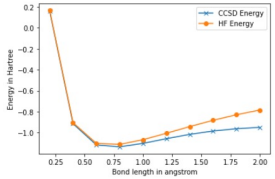
[2] Evgeny Epifanovsky, Andrew T. B. Gilbert, Xintian Feng, Joonho Lee, Yuezhi Mao, Narbe Mardirossian, Pavel Pokhilko, Alec F. White, Marc P. Coons, Adrian L. Dempwolff, Zhengting Gan, Diptarka Hait, Paul R. Horn, Leif D. Jacobson, Ilya Kaliman, Jörg Kussmann, Adrian W. Lange, Ka Un Lao, Daniel S. Levine, Jie Liu, Simon C. McKenzie, Adrian F. Morrison, Kaushik D. Nanda, Felix Plasser, Dirk R. Rehn, Marta L. Vidal, Zhi-Qiang You, Ying Zhu, Bushra Alam, Benjamin J. Albrecht, Abdulrahman Aldossary, Ethan Alguire, Josefine H. Andersen, Vishikh Athavale, Dennis Barton, Khadiza Begam, Andrew Behn, Nicole Bellonzi, Yves A. Bernard, Eric J. Berquist, Hugh G. A. Burton, Abel Carreras, Kevin Carter-Fenk, Romit Chakraborty, Alan D. Chien, Kristina D. Closser, Vale Cofer-Shabica, Saswata Dasgupta, Marc de Wergifosse, Jia Deng, Michael Diedenhofen, Hainam Do, Sebastian Ehlert, Po-Tung Fang, Shervin Fatehi, Qingguo Feng, Triet Friedhoff, James Gayvert, Qinghui Ge, Gergely Gidofalvi, Matthew Goldey, Joe Gomes, Cristina E. González-Espinoza, Sahil Gulania, Anastasia O. Gunina, Magnus W. D. Hanson-Heine, Phillip H. P. Harbach, Andreas Hauser, Michael F. Herbst, Mario Hernández Vera, Manuel Hodecker, Zachary C. Holden, Shannon Houck, Xunkun Huang, Kerwin Hui, Bang C. Huynh, Maxim Ivanov, Ádám Jász, Hyunjun Ji, Hanjie Jiang, Benjamin Kaduk, Sven Kähler, Kirill Khistyaev, Jaehoon Kim, Gergely Kis, Phil Klunzinger, Zsuzsanna Koczor-Benda, Joong Hoon Koh, Dimitri Kosenkov, Laura Koulias, Tim Kowalczyk, Caroline M. Krauter, Karl Kue, Alexander Kunitsa, Thomas Kus, István Ladjánszki, Arie Landau, Keith V. Lawler, Daniel Lefrancois, Susi Lehtola, Run R. Li, Yi-Pei Li, Jiashu Liang, Marcus Liebenthal, Hung-Hsuan Lin, You-Sheng Lin, Fenglai Liu, Kuan-Yu Liu, Matthias Loipersberger, Arne Luenser, Aaditya Manjanath, Prashant Manohar, Erum Mansoor, Sam F. Manzer, Shan-Ping Mao, Aleksandr V. Marenich, Thomas Markovich, Stephen Mason, Simon A. Maurer, Peter F. McLaughlin, Maximilian F. S. J. Menger, Jan-Michael Mewes, Stefanie A. Mewes, Pierpaolo Morgante, J. Wayne Mullinax, Katherine J. Oosterbaan, Garrette Paran, Alexander C. Paul, Suranjan K. Paul, Fabijan Pavošević, Zheng Pei, Stefan Prager, Emil I. Proynov, Ádám Rák, Eloy Ramos-Cordoba, Bhaskar Rana, Alan E. Rask, Adam Rettig, Ryan M. Richard, Fazle Rob, Elliot Rossomme, Tarek Scheele, Maximilian Scheurer, Matthias Schneider, Nickolai Sergueev, Shaama M. Sharada, Wojciech Skomorowski, David W. Small, Christopher J. Stein, Yu-Chuan Su, Eric J. Sundstrom, Zhen Tao, Jonathan Thirman, Gábor J. Tornai, Takashi Tsuchimochi, Norm M. Tubman, Srimukh Prasad Veccham, Oleg Vydrov, Jan Wenzel, Jon Witte, Atsushi Yamada, Kun Yao, Sina Yeganeh, Shane R. Yost, Alexander Zech, Igor Ying Zhang, Xing Zhang, Yu Zhang, Dmitry Zuev, Alán Aspuru-Guzik, Alexis T. Bell, Nicholas A. Besley, Ksenia B. Bravaya, Bernard R. Brooks, David Casanova, Jeng-Da Chai, Sonia Coriani, Christopher J. Cramer, György Cserey, A. Eugene DePrince III, Robert A. DiStasio Jr., Andreas Dreuw, Barry D. Dunietz, Thomas R. Furlani, William A. Goddard III, Sharon Hammes-Schiffer, Teresa Head-Gordon, Warren J. Hehre, Chao-Ping Hsu, Thomas-C. Jagau, Yousung Jung, Andreas Klamt, Jing Kong, Daniel S. Lambrecht, WanZhen Liang, Nicholas J. Mayhall, C. William McCurdy, Jeffrey B. Neaton, Christian Ochsenfeld, John A. Parkhill, Roberto Peverati, Vitaly A. Rassolov, Yihan Shao, Lyudmila V. Slipchenko, Tim Stauch, Ryan P. Steele, Joseph E. Subotnik, Alex J. W. Thom, Alexandre Tkatchenko, Donald G. Truhlar, Troy Van Voorhis, Tomasz A. Wesolowski, K. Birgitta Whaley, H. Lee Woodcock III, Paul M. Zimmerman, Shirin Faraji, Peter M. W. Gill, Martin Head-Gordon, John M. Herbert, *and* Anna I. Krylov, "Software for the frontiers of quantum chemistry: An overview of developments in the Q-Chem 5 package", J. Chem. Phys. 155, 084801 (2021)

OPENFERMION/Q-CHEM (V 1.0)

- Once one generates a MolecularData instance, one can then call “run_qchem” module in order to update and save the MolecularData object by accessing the Q-CHEM calculations.
- The current version of the code works with common electronic structure calculation results (HF, MP2, and CCSD) which are molecular orbitals, 1- and 2-electron integrals, Hartree-Fock energy, MP2 energy, CCSD energy, and CCSD amplitudes, and more.
- A simple example illustrating how to store the energy of H_2 at various bond lengths.

```python
from openfermion.chem import MolecularData
from openfermionqchem import run_qchem

# Set molecule parameters.
# We do not run calculations through OpenFermion directly.
basis        = 'sto-3g'
multiplicity = 1

# Generate molecule at different bond lengths.
bond_length_interval = 0.2
n_points = 10
hf_energies   = []
ccsd_energies = []
bond_lengths  = []

for point in range(1, n_points + 1):
    bond_length    = bond_length_interval * float(point)
    bond_lengths  += [bond_length]
    geometry       = [('H', (0., 0., 0.)), ('H', (0., 0., bond_length))]

    # Generate a MolecularData instance
    molecule       = MolecularData(geometry, basis, multiplicity)

    # Run run_qchem module.
    # Users need to specify file directory with OpenFermion/Q-CHEM v0.0.
    file_directory = '/Users/yongbin/Desktop/openfermionqchem/molecules/h2/sto-3g/'+str(round(bond_length,2))+'/'
    # Update the MolecularData instance
    molecule       = run_qchem(molecule,file_directory=file_directory,output_name='test_qis')

    # Print out some results of calculation.
    print('\nAt bond length of {} angstrom, molecular hydrogen has:'.format(bond_ler
    print('Hartree-Fock energy of {} Hartree.'.format(molecule.hf_energy))
    print('MP2 energy of {} Hartree.'.format(molecule.mp2_energy))
    print('CCSD energy of {} Hartree.'.format(molecule.ccsd_energy))
    print('Nuclear repulsion energy between protons is {} Hartree.'.format(
        molecule.nuclear_repulsion))
    for orbital in range(molecule.n_orbitals):
        print('Spatial orbital {} has energy of {} Hartree.'.format(
            orbital, molecule.orbital_energies[orbital]))
    hf_energies   += [molecule.hf_energy]
    ccsd_energies += [molecule.ccsd_energy]

# Plot.
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(0)
plt.plot(bond_lengths, ccsd_energies, 'x-')
plt.plot(bond_lengths, hf_energies, 'o-')
plt.ylabel('Energy in Hartree')
plt.xlabel('Bond length in angstrom')
plt.show()
```

```
At bond length of 1.8 angstrom, molecular hydrogen has:
Hartree-Fock energy of -0.828848148 Hartree.
MP2 energy of -0.89788202 Hartree.
CCSD energy of -0.96181696 Hartree.
Nuclear repulsion energy between protons is 0.29398734 Hartree.
Spatial orbital 0 has energy of -0.299563221 Hartree.
Spatial orbital 1 has energy of 0.145960299 Hartree.

At bond length of 2.0 angstrom, molecular hydrogen has:
Hartree-Fock energy of -0.78379265 44 Hartree.
MP2 energy of -0.87251188 Hartree.
CCSD energy of -0.94864111 Hartree.
Nuclear repulsion energy between protons is 0.26458861 Hartree.
Spatial orbital 0 has energy of -0.269459223 Hartree.
Spatial orbital 1 has energy of 0.108997371 Hartree.
```

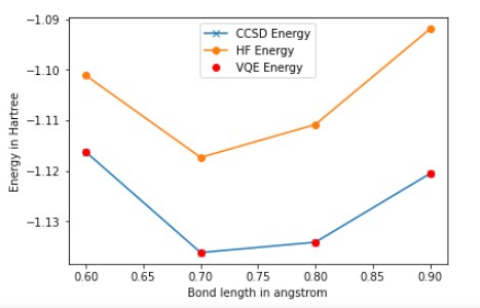


OPENFERMION/Q-CHEM (V 1.0)

- Once the properties of a molecule are saved, the modules of OpenFermion can be called to prepare or perform quantum simulations.
- A simple example of H_2 bond dissociation curve with HF, CCSD, and VQE/UCCSD.

```python
import numpy as np
import os
from scipy.optimize import minimize

import cirq
from openfermion.transforms import get_fermion_operator, jordan_wigner, bravyi_kitaev
from openfermion.linalg import get_sparse_operator, get_ground_state
from openfermion.chem import MolecularData
from openfermionqchem import run_qchem
from openfermion.ops import FermionOperator, InteractionOperator

from openfermionprojectq import *
from projectq.ops import X, All, Measure
from projectq.backends import CommandPrinter, CircuitDrawer
```

```python
# Generate molecule at different bond lengths.
bond_length_interval = 0.1
n_points       = 4
hf_energies    = []
ccsd_energies  = []
vqe_energies   = []
bond_lengths   = []

for point in range(1, n_points + 1):
    bond_length     = 0.5 + bond_length_interval * float(point)
    bond_lengths  += [bond_length]
    geometry        = [('H', (0., 0., 0.)), ('H', (0., 0., bond_length))]

    # Generate a MolecularData instance
    molecule        = MolecularData(geometry, basis, multiplicity)

    # Run run_qchem module.
    # Users need to specify file directory with OpenFermion/Q-CHEM v0.0.
    file_directory = '/Users/yongbin/Desktop/openfermionqchem/molecules/h2/sto-3g/'+str(round(bond_length,2))+'/'
    # Update the MolecularData instance
    molecule        = run_qchem(molecule,file_directory=file_directory,output_name='test_qis')


    hamiltonian         = InteractionOperator(molecule.nuclear_repulsion,molecule.one_body_integrals,0.25*molecule.two_b
    ground_energy, _    = get_ground_state(get_sparse_operator(hamiltonian))
    qubit_hamiltonian   = jordan_wigner(hamiltonian)
    qubit_hamiltonian.compress()
    compiler_engine     = uccsd_trotter_engine()
    initial_amplitudes = [0, 0.01]

    # Run VQE Optimization to find new CCSD parameters

    opt_result = minimize(energy_objective, initial_amplitudes,
                          method="CG", options={'disp':False})
    opt_energy, opt_amplitudes = opt_result.fun, opt_result.x

    hf_energies.append(molecule.hf_energy)
    ccsd_energies.append(molecule.ccsd_energy)
    vqe_energies.append(opt_energy)

# Plot.
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(0)
plt.plot(bond_lengths, ccsd_energies, 'x-', label='CCSD Energy')
plt.plot(bond_lengths, hf_energies, 'o-', label='HF Energy')
plt.plot(bond_lengths, vqe_energies, 'o', label='VQE Energy', color='red')
plt.ylabel('Energy in Hartree')
plt.xlabel('Bond length in angstrom')
plt.legend()
plt.show()
```



OPENFERMION/Q-CHEM (V 1.0)

- To use OpenFermion/Q-CHEM, you need to personally install Q-CHEM. To install the latest versions of OpenFermion and Openfermion/Q-CHEM:
  - python –m pip install openfermionqchem (need GitHub repository)
- The current version is compatible with Q-CHEM 5.4+.
  - It reads molecule's information from text files provided by Q-CHEM calculations.
- Plans for version 1.1
  - Version 1.1 will use new journaling feature of Q-CHEM to update and save the MolecularData object.

Part 2

OPENFERMION/Q-CHEM VERSION 1.0 WORKING PROCEDURE

1. **Perform electronic structure calculations using Q-CHEM**
2. Save molecular information of Q-CHEM into MolecularData object via OpenFermion/Q-CHEM plugin
3. Construct/perform/analyze quantum calculations via OpenFermion library

Q-CHEM PART

```
$comment
OPENFERMION/Q-CHEM example
H2/STO-3G
$end

$molecule
0  1
   H  0.00  0.00  0.00
   H  0.0 0 0.00  0.75
$end

$rem
    BASIS                = STO-3G
    METHOD               = CCSD
    GUI                  = 2
    N_FROZEN_CORE = 0
    PRINT_QIS            = TRUE
$end
```

Request Q-CHEM checkpoint file

All electrons will be active; Q-CHEM provides two-electron integrals for core electrons as well

Request for Q-CHEM to provide electron integrals, molecular orbital energies, CCSD amplitudes in text files

Q-CHEM PART

List of files from Q-CHEM

1. Q-CHEM output file (Default)
2. Q-CHEM checkpoint file (need GUI=2)
3. mo_ints_for_qis.dat (Fock matrix elements)
4. mo_ene_for_qis.dat (Molecular orbital energies)
5. two_body_int_for_qis.dat (Two-electron integrals)
6. In the case of CCSD calculations
    1. cc_t1_for_qis.dat (CCSD t1 amplitudes)
    2. cc_t2_for_qis.dat (CCSD t2 amplitudes)

OpenFermion/
Q-CHEM

MolecularData

Part 2

OPENFERMION/Q-CHEM VERSION 1.0 WORKING PROCEDURE

1. Perform electronic structure calculations using Q-CHEM
2. **Save molecular information of Q-CHEM into MolecularData object via OpenFermion/Q-CHEM plugin**
3. Construct/perform/analyze quantum calculations via OpenFermion library

OPENFERMION-QCHEM PART

- Parsing checkpoint file (general information of a molecule) → OpenFermion MolecularData object

Q-CHEM Checkpoint File

```
SP        R                                STO-3G
Number of atoms                               I                2
Charge                                        I                0
Multiplicity                                  I                1
Number of electrons                           I                2
Number of alpha electrons                     I                1
Number of beta electrons                      I                1
Current cartesian coordinates                 R   N=           6
  0.00000000E+00  0.00000000E+00 -7.08647297E-01  0.00000000E+00  0.00000000E+00
  7.08647297E-01
Nuclear charges                               R   N=           2
  1.00000000E+00  1.00000000E+00
Number of basis functions                     I                2
Core Hamiltonian Matrix                       R   N=           3
 -1.11450934E+00 -9.47668283E-01 -1.11450934E+00
Alpha MO coefficients                         R   N=           4
 -5.49926604E-01 -5.49926604E-01 -1.20096196E+00  1.20096196E+00
```

MolecularData

```python
print("Basis set                  : {}".format(molecule.basis))
print("Number of atoms            : {}".format(molecule.n_atoms))
print("Charge                     : {}".format(molecule.charge))
print("Multiplicity               : {}".format(molecule.multiplicity))
print("Number of electrons        : {}".format(molecule.n_electrons))
print("Atoms                      : {}".format(molecule.atoms))
print("Nuclear number             : {}".format(molecule.protons))
print("Number of spatial orbitals: {}".format(molecule.n_orbitals))
print("Molecular geometry         : {}".format(molecule.geometry))
print("Core Hamiltonian           : {}".format(molecule.h_core))
print("Canonical orbitals         : {}".format(molecule.canonical_orbitals))
```

```
Basis set                  : sto-3g
Number of atoms            : 2
Charge                     : 0
Multiplicity               : 1
Number of electrons        : 2
Atoms                      : ['H', 'H']
Nuclear number             : [1, 1]
Number of spatial orbitals: 2
Molecular geometry         : [('H', (0.0, 0.0, -0.375)), ('H', (0.0, 0.0, 0.375))]
Core Hamiltonian           : [[-1.11450934  -0.947668283]
 [-0.947668283 -1.11450934 ]]
Canonical orbitals         : [[-0.549926604 -1.20096196 ]
 [-0.549926604  1.20096196 ]]
```

OPENFERMION-QCHEM PART

- Parsing output file (HF, MP2, CCSD energies) → OpenFermion MolecularData object

Q-CHEM Output File

```
Nuclear Repulsion Energy                  =  0.70556961 hartrees
Total energy in the final basis set = -1.1161514490
MP2 energy                                = -1.12952457
CCSD total energy                         = -1.13711722
```

→ **MolecularData**

```
print("Nuclear repulsion energy: {}".format(molecule.nuclear_repulsion))
print("HF energy                : {}".format(molecule.hf_energy))
print("MP2 energy               : {}".format(molecule.mp2_energy))
print("CCSD energy              : {}".format(molecule.ccsd_energy))
```

```
Nuclear repulsion energy: 0.70556961
HF energy                : -1.116151449
MP2 energy               : -1.12952457
CCSD energy              : -1.13711722
```

OPENFERMION-QCHEM PART

- Using core Hamiltonian and molecular orbital coefficients, OpenFermion/Q-CHEM computes one-electron integrals → OpenFermion MolecularData object

One-electron Integrals

$$h_{pq} = \langle MO | H_{core} | MO \rangle$$

Only with α orbital coefficients

Spatial to spin orbital representation

αβαβαβ… order

MolecularData

```
print("one-body integrals: \n{}".format(molecule.one_body_integrals))
```

```
one-body integrals:
[[-1.2472845018  0.            0.            0.          ]
 [ 0.           -1.2472845018  0.            0.          ]
 [ 0.            0.           -0.4812729262  0.          ]
 [ 0.            0.            0.           -0.4812729262]]
```

OPENFERMION-QCHEM PART

- Parsing two_body_int_for_qis.dat (two-electron integrals) → OpenFermion MolecularData object

Two-electron integrals

```
0000
4 2 2 2 2
   0.0000000000000000e+00    0.0000000000000000e+00    0.0000000000000000e+00
   0.0000000000000000e+00    0.0000000000000000e+00    6.7284794691618166e-01
  -6.7284794691618166e-01    0.0000000000000000e+00    0.0000000000000000e+00
  -6.7284794691618166e-01    6.7284794691618166e-01    0.0000000000000000e+00
   0.0000000000000000e+00    0.0000000000000000e+00    0.0000000000000000e+00
   0.0000000000000000e+00
000V
4 2 2 2 2
   0.0000000000000000e+00    0.0000000000000000e+00    0.0000000000000000e+00
   0.0000000000000000e+00    0.0000000000000000e+00    0.0000000000000000e+00
   0.0000000000000000e+00    0.0000000000000000e+00    0.0000000000000000e+00
   0.0000000000000000e+00    0.0000000000000000e+00    0.0000000000000000e+00
   0.0000000000000000e+00    0.0000000000000000e+00    0.0000000000000000e+00
   0.0000000000000000e+00
```

MolecularData

OPENFERMION-QCHEM PART

Two-electron integrals

- Spin orbital representation
- OOOO, OOOV, OOVV, OVOV, OVVV, VVVV blocks written in 1D array (O: Occupied V: Virtual)
  - OpenFermion/Q-CHEM transforms 1D → 4D array (n_orbs x norbs x norbs x norbs)
    - Consider all possible permutations
    - idx = (p*qmax*rmax*smax) + (q*rmax*smax) + (r*smax) + s
    - V[p][q][r][s] = QCHEM_TWO_BODY[idx]
- Q-CHEM's orbital order is αααα…αβββ… β
  - OpenFermion/Q-CHEM reorders them into αβαβ..αβ in molecular orbital energy ascending order.

```python
print("two-body integrals: \n{}".format(molecule.two_body_integrals))
```

```
two-body integrals:
[[[[ 0.            0.            0.            0.          ]
   [ 0.            0.            0.            0.          ]
   [ 0.            0.            0.            0.          ]
   [ 0.            0.            0.            0.          ]]

  [[ 0.           -0.6728479469  0.            0.          ]
   [ 0.6728479469  0.            0.            0.          ]
   [ 0.            0.            0.           -0.1817715366]
   [ 0.            0.            0.1817715366  0.          ]]

  [[ 0.            0.           -0.4802057229  0.          ]
   [ 0.            0.            0.            0.          ]
   [ 0.4802057229  0.            0.            0.          ]
   [ 0.            0.            0.            0.          ]]

  [[ 0.            0.            0.           -0.6619772594]
   [ 0.            0.            0.1817715366  0.          ]
   [ 0.           -0.1817715366  0.            0.          ]
   [ 0.6619772594  0.            0.            0.          ]]]
```

...

OPENFERMION-QCHEM PART

- Parsing cc_t1_for_qis.dat and cc_t2_for_qis.dat (CCSD amplitudes) → OpenFermion MolecularData object

CCSD amplitudes

- Follow exactly same procedure as two body integrals
  - 1D → 2D (t1 amps) and 1D → 4D (t2 amps)
  - Reordering in terms of orbital energies

```
print("CCSD singles: \n{}".format(molecule.ccsd_single_amps))
```

```
CCSD singles amplitudes:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
print("CCSD doubles: \n{}".format(molecule.ccsd_double_amps))
```

```
CCSD doubles:
[[[ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]
  [ 0.             -0.0576706721    0.              0.            ]]

 [[ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]
  [ 0.0576706721    0.              0.              0.            ]]

 [[ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]]

 [[ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]
  [ 0.              0.              0.              0.            ]]]
```

OPENFERMION-QCHEM PART

Once one generates the MolecularData object via OpenFermion/Q-CHEM plugin, then one can call “run_test” module to check whether the electron integral matrices are stored properly.

Unit Test

```
from openfermionqchem import run_test

run_test()
```

```
==================================================================
Molecule:       H2/STO-3G
Bond distance: 0.75
FCI energy in Hartees for reference:          -1.13711707
Direct diagonalization through OpenFermion:   -1.13711707
fidelity; FCI / Direct diagonalization:        1.0
==================================================================
Molecule:       H2/3-21G
Bond distance: 0.75
FCI energy in Hartees for reference:          -1.14787738
Direct diagonalization through OpenFermion:   -1.14787739
fidelity; FCI / Direct diagonalization:        1.0
==================================================================
Molecule:       H2/6-311G
Bond distance: 0.75
FCI energy in Hartees for reference:          -1.15349121
Direct diagonalization through OpenFermion:   -1.15349122
fidelity; FCI / Direct diagonalization:        1.0
==================================================================
```

If fidelity != 1.0, the electron integral transformation schemes are wrong!

run_test module for unit test

```
from openfermionqchem import test_files
def run_test():
    """
    This function checks code consistencies.

    Prints:
        H2/STO-3G FCI vs Direct Diagonalization energy comparisons.
        H2/3-21G  FCI vs Direct Diagonalization energy comparisons.
        H2/6-311G FCI vs Direct Diagonalization energy comparisons.
    """
    basis        = '3-21g'
    multiplicity = 1
    charge       = 0
    bond_length  = 0.75
    geometry     = [('H', (0., 0., 0.)), ('H', (0., 0., bond_length))]
    molecule     = MolecularData(geometry, basis, multiplicity, charge)
    path         = str(test_files.__path__._path[0])+'/'
    molecule1    = run_qchem(molecule,file_directory=path+'h2-sto3g/',output_name='test_qis')
    molecule2    = run_qchem(molecule,file_directory=path+'h2-321g/',output_name='test_qis')
    molecule3    = run_qchem(molecule,file_directory=path+'h2-6311g/',output_name='test_qis')

    h2_sto3g_fci = -1.1371170673457311
    h2_321g_fci  = -1.1478773770642743
    h2_6311g_fci  = -1.1534912078971873

    H1 = InteractionOperator(molecule1.nuclear_repulsion,molecule1.one_body_integrals,0.25*molecule1.two_body_integrals)
    H2 = InteractionOperator(molecule2.nuclear_repulsion,molecule2.one_body_integrals,0.25*molecule2.two_body_integrals)
    H3 = InteractionOperator(molecule3.nuclear_repulsion,molecule3.one_body_integrals,0.25*molecule3.two_body_integrals)

    h2_sto3g_direct, _ = get_ground_state(get_sparse_operator(H1))
    h2_321g_direct, _  = get_ground_state(get_sparse_operator(H2))
    h2_6311g_direct, _  = get_ground_state(get_sparse_operator(H3))

    fidelity1 = round(h2_sto3g_direct / h2_sto3g_fci, 7)
    fidelity2 = round(h2_321g_direct / h2_321g_fci, 7)
    fidelity3 = round(h2_6311g_direct / h2_6311g_fci, 7)
```

OpenFermion/Q-CHEM provides test molecule (H_2) with 3 different basis sets

Generate MolecularData objects

FCI energies for references

Obtain ground state energies by direct diagonalization

Compare them with reference FCI energies

Part 2

OPENFERMION/Q-CHEM VERSION 1.0 WORKING PROCEDURE

1. Perform electronic structure calculations using Q-CHEM
2. Save molecular information of Q-CHEM into MolecularData object via OpenFermion/Q-CHEM plugin
3. **Construct/perform/analyze quantum calculations via OpenFermion library**

OPENFERMION PART

- **Use Openfermion module with the constructed MolecularData class**

e.g) Jordan-Wigner transformation

```
from openfermion.ops import InteractionOperator
from openfermion.transforms import jordan_wigner

nuclear_repulsion = molecule.nuclear_repulsion
one_body          = molecule.one_body_integrals
two_body          = molecule.two_body_integrals
hamiltonian       = InteractionOperator(nuclear_repulsion,one_body,0.25*two_body)
jw_hamiltonian    = jordan_wigner(hamiltonian)
print("Jordan-Wigner Hamiltonian: \n{}".format(jw_hamiltonian))
```

Import Openfermion module to transform fermion to qubit representation

Call the molecular information from the MolecularData class

Q-CHEM employs anti-symmetrized two-body operator

```
Jordan-Wigner Hamiltonian:
-0.1097305523646723 [] +
-0.04544288414672032 [X0 X1 Y2 Y3] +
0.04544288414672032 [X0 Y1 Y2 X3] +
0.04544288414672032 [Y0 X1 X2 Y3] +
-0.04544288414672032 [Y0 Y1 X2 X3] +
0.16988451861151188 [Z0] +
0.16821198672904542 [Z0 Z1] +
0.1200514307157089 [Z0 Z2] +
0.16549431486242921 [Z0 Z3] +
0.16988451861151188 [Z1] +
0.16549431486242921 [Z1 Z2] +
0.1200514307157089 [Z1 Z3] +
-0.21886307025041057 [Z2] +
0.17395378775714795 [Z2 Z3] +
-0.21886307025041057 [Z3]
```

OPENFERMION PART

- Difference:
  - anti-symmetrized vs symmetrized operator
  - Spin orbital vs spatial orbital representation
    - Q-CHEM saves one step

```
from openfermion.ops import InteractionOperator
from openfermion.transforms import jordan_wigner

nuclear_repulsion = molecule.nuclear_repulsion
one_body          = molecule.one_body_integrals
two_body          = molecule.two_body_integrals
hamiltonian       = InteractionOperator(nuclear_repulsion,one_body,0.25*two_body)
jw_hamiltonian    = jordan_wigner(hamiltonian)
print("Jordan-Wigner Hamiltonian: \n{}".format(jw_hamiltonian))
```

```
Jordan-Wigner Hamiltonian:
-0.1097305523646723 [] +
-0.04544288414672032 [X0 X1 Y2 Y3] +
0.04544288414672032 [X0 Y1 Y2 X3] +
0.04544288414672032 [Y0 X1 X2 Y3] +
-0.04544288414672032 [Y0 Y1 X2 X3] +
0.16988451861151188 [Z0] +
0.16821198672904542 [Z0 Z1] +
0.1200514307157089 [Z0 Z2] +
0.16549431486242921 [Z0 Z3] +
0.16988451861151188 [Z1] +
0.16549431486242921 [Z1 Z2] +
0.1200514307157089 [Z1 Z3] +
-0.21886307025041057 [Z2] +
0.17395378775714795 [Z2 Z3] +
-0.21886307025041057 [Z3]
```

Q-CHEM

```
import pyscf
import openfermionpyscf
from openfermionpyscf import run_pyscf
from openfermion import get_fermion_operator
```

```
geometry              = [('H', (0., 0., -0.375)), ('H', (0., 0., 0.375))]
basis                 = 'sto-3g'
multiplicity          = 1
molecule              = MolecularData(geometry, basis, multiplicity)
```

```
# Run pyscf.
molecule = run_pyscf(molecule,run_scf=run_scf,run_ccsd=run_ccsd,run_fci=run_fci)
```

```
molecular_hamiltonian = molecule.get_molecular_hamiltonian()
```

```
fermion_hamiltonian = get_fermion_operator(molecular_hamiltonian)
jw_hamiltonian      = jordan_wigner(fermion_hamiltonian)
print('The Jordan-Wigner Hamiltonian: \n{}'.format(jw_hamiltonian))
```

```
The Jordan-Wigner Hamiltonian:
(-0.10973055606700677+0j) [] +
(-0.04544288414432619+0j) [X0 X1 Y2 Y3] +
(0.04544288414432619+0j) [X0 Y1 Y2 X3] +
(0.04544288414432619+0j) [Y0 X1 X2 Y3] +
(-0.04544288414432619+0j) [Y0 Y1 X2 X3] +
(0.1698845202794037+0j) [Z0] +
(0.16821198673715723+0j) [Z0 Z1] +
(0.1200514307254603+0j) [Z0 Z2] +
(0.1654943148697865+0j) [Z0 Z3] +
(0.1698845202794037+0j) [Z1] +
(0.1654943148697865+0j) [Z1 Z2] +
(0.1200514307254603+0j) [Z1 Z3] +
(-0.2188630678121964+0j) [Z2] +
(0.1739537877649413+0j) [Z2 Z3] +
(-0.21886306781219642+0j) [Z3]
```

PYSCF