Xuenan Wang(xw336)

Rutgers University – New Brunswick

CS672 Quantum Computing: Programs and Systems

Programming Assignment 2: VQE on Qiskit

1. Warmup: Bell state entangled qubits on real quantum machines

Interpret the results from the Python program's command line output. What is the expected distribution of answers if there were infinitely many samples taken from measuring the quantum circuit output? Do the observed results match the expected distribution? Why or why not? Submit your answers in your writeup.

Following are the result I got from command line:

```
(env) xw336@ilab3:~/QuantumComputing/Qiskit$ python hello_quantum.py
BasicAer backends:  [<QasmSimulatorPy('qasm_simulator') from BasicAer()>, <StatevectorSimu
latorPy('statevector_simulator') from BasicAer()>, <UnitarySimulatorPy('unitary_simulator'
) from BasicAer()>]
{'00': 535, '11': 489}
/ilab/users/xw336/QuantumComputing/Qiskit/env/lib/python3.6/site-packages/qiskit/providers
/ibmq/ibmqfactory.py:192: UserWarning: Timestamps in IBMQ backend properties, jobs, and jo
b results are all now in local time instead of UTC.
  warnings.warn('Timestamps in IBMQ backend properties, jobs, and job results '
Remote backends:  [<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='ope
n', project='main')>, <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project=
'main')>, <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open', project='
main')>, <IBMQBackend('ibmq_vigo') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQBackend('ibmq_ourense') from IBMQ(hub='ibm-q', group='open', project='main')>, <IBMQB
ackend('ibmq_valencia') from IBMQ(hub='ibm-q', group='open', project='main')>, <IBMQBacken
d('ibmq_armonk') from IBMQ(hub='ibm-q', group='open', project='main')>, <IBMQBackend('ibmq
_athens') from IBMQ(hub='ibm-q', group='open', project='main')>, <IBMQBackend('ibmq_santia
go') from IBMQ(hub='ibm-q', group='open', project='main')>]
Running on current least busy device:  ibmq_athens
Counts:  {'00': 510, '01': 30, '10': 14, '11': 470}
```

Figure 1. Output of hello_quantum.py

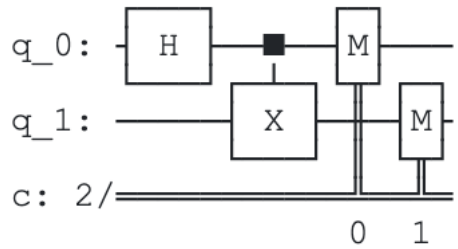

Figure 2. Circuit of hello_quantum.py

The expected result if there were infinitely many samples is {‘00’: 512, ‘11’:512} because we have total 1024 shots. The result I have from the simulator is {‘00’: 535, ‘11’:489}, which I’d say pretty close to our ideal result. The result I got from IBM Qiskit Athens

quantum processor is {‘00’: 510, ‘01’: 30, ‘10’: 14, ‘11’:470}. It’s basically following my expectation. However, there shouldn’t be any $|01>$ and $|10>$. This could because some physical implementation errors of IBM Qiskit machine.

2. Quantum chemistry ground state estimation experiments on simulated quantum computers

A. LiH ground state energy estimation at known bond length

Interpret the results from the Python program's command line output. How many qubits are involved in the quantum circuit for the VQE ansatz? Submit your answers in your writeup.

This experiment is trying to simulate LiH ground state energy using STO-3G basis. Following is my result: (see next page)

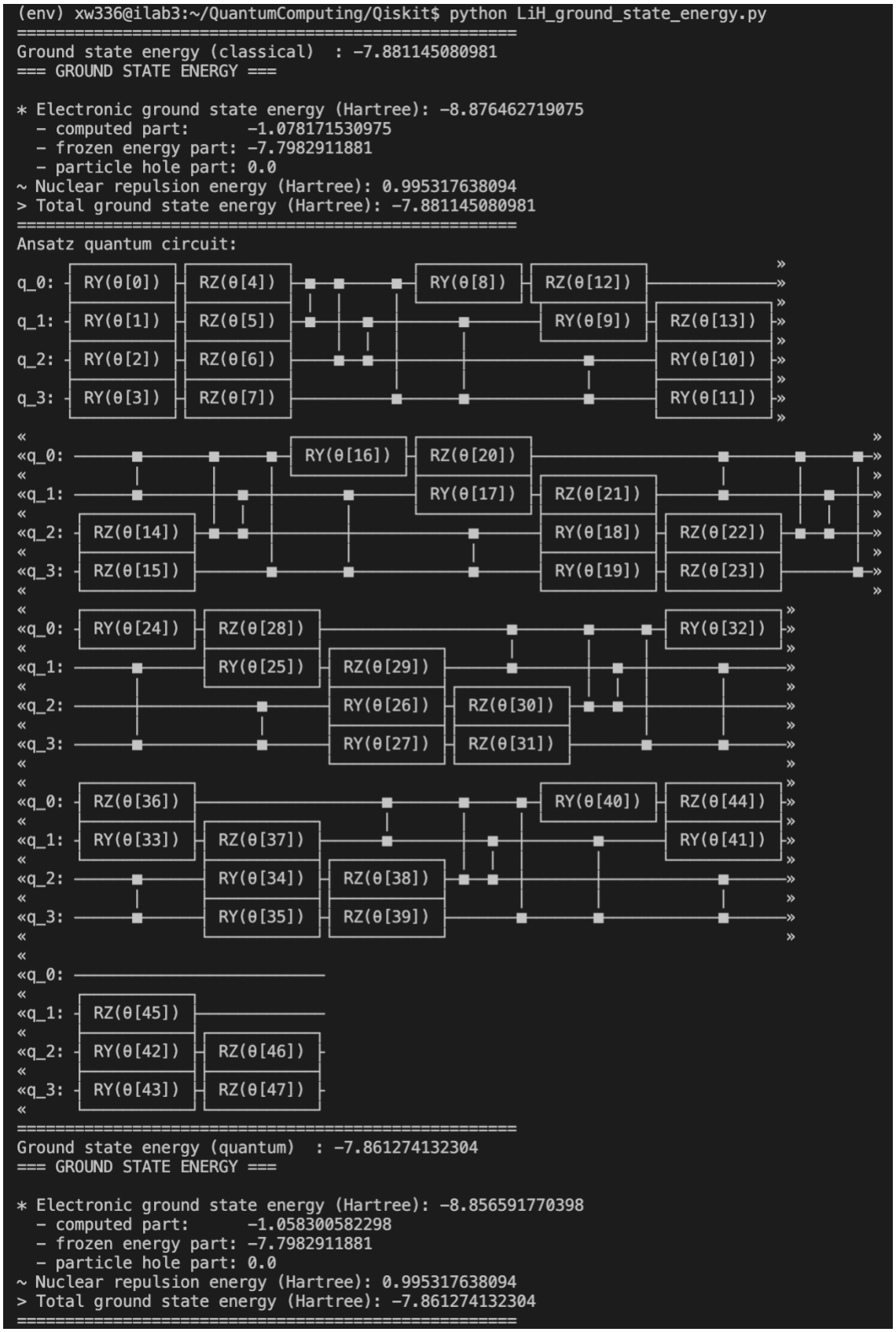

Figure 3. Output of LiH_ground_state_energy.py

The ground state energy computed from classical way is -7.881145080981. Quantum circuit using to simulate this problem is shown above. 4 qubits involved.
The result from quantum simulation is -7.861274132304 and it's very close to the classical computation.

B. Hydrogen molecule bond length determination by finding ground state energy minimum

Include your plot in your writeup. Interpret the plot. What is the hydrogen molecule's bond length, and what is its ground state energy? Submit your answers in your writeup.

This experiment is trying to simulate H_2 ground state energy using Qiskit Chemistry and STO-3G basis over a range of interatomic distances with VQE and TwoLocal. Following is my result:

```
(env) xw336@ilab3:~/QuantumComputing/Qiskit$ python H2_energy_curve.py
Processing step  4 --- complete
Distances:  [0.5   0.625 0.75  0.875 1.   ]
Energies: [[-1.055156   -1.1238725  -1.1355422  -1.12429005 -1.10109341]
 [-1.05515979 -1.12416092 -1.13711707 -1.12467175 -1.10115033]]
Hartree-Fock energies: [-1.04299627 -1.10814999 -1.11615145 -1.09745432 -1.06610865]
```

Figure 4. Output of H2_energy_curve.py

Result of VQE algorithm is [-1.055156 -1.1238725 -1.1355422 -1.12429005 -1.10109341]
Result of NumPyMinimumEigensolver is [-1.05515979 -1.12416092 -1.13711707 -1.12467175 -1.10115033]
Result of Hartree-Fork is [-1.04299627 -1.10814999 -1.11615145 -1.09745432 -1.06610865]
It's clear on the above plot that the curve generated by VQE and NumPyEigensolver is basically that same and Hartree-Fock is not quite the same be very similar.

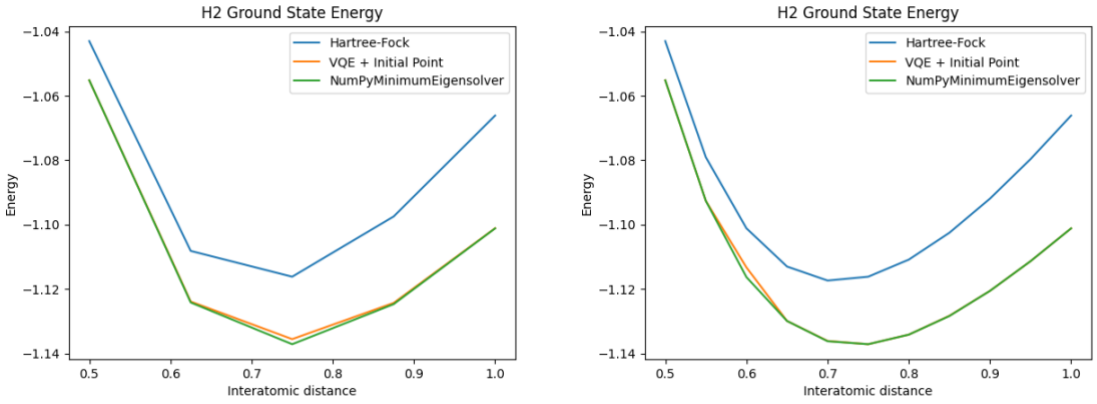

Figure 5. H_2 Energy over Interatomic Distance (0.5 – 1.0), 4 steps (left) and 10 steps (right)

We can plot the energy over interatomic distance. Above are the graphs. We can see that with steps going up, which means the curve being smoother, the curve still follows the same pattern.

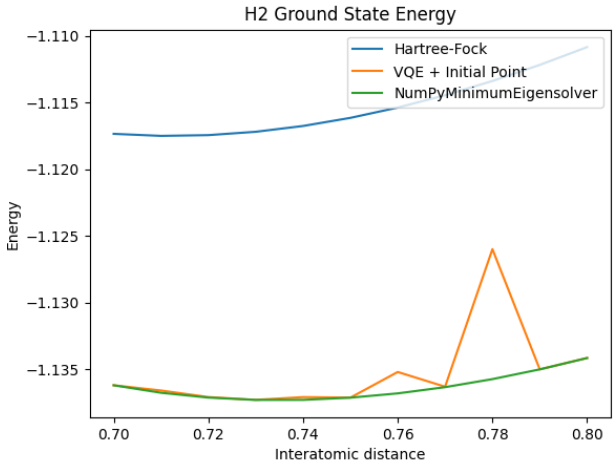

Figure 6. H_2 Energy over Interatomic Distance (0.7 – 0.8), 10 steps

When I amplify the lowest range of [0.7, 0.8], the VQE seems different. Although I don't understand these unexpected spurs, I guess it's because some implementation error or issue of VQE algorithm.

C. Plotting the LiH energy curve

Include your plot in your writeup. Interpret the plot. What is lithium hydride's bond length, and what is its ground state energy? Submit your answers and source code in your writeup.

According to experimental data for LiH from cccbdb.nist.gov, the bond length is 1.595 and ground state energy is -7.900.
I modified H2_energy_curve.py based on LiH_ground_state_energy.py, the modified file is named LiH_energy_curve.py (see appendix).
Following are my results:

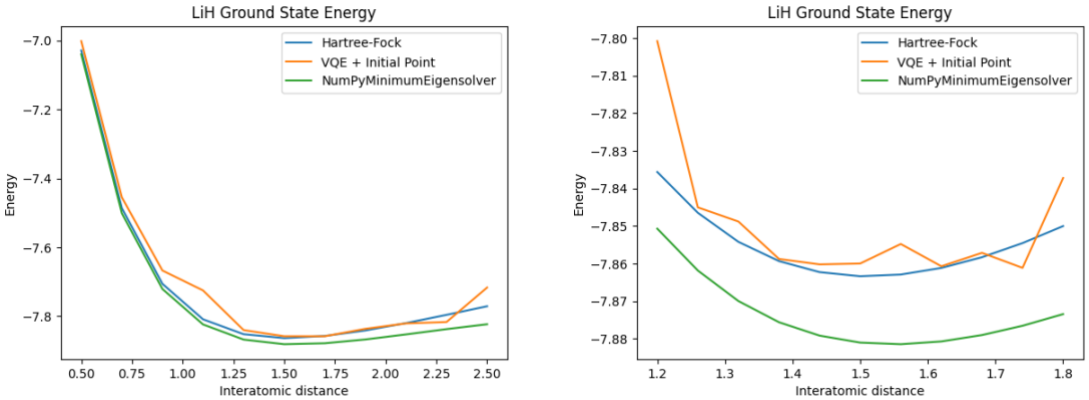

Figure 7. LiH Energy over Interatomic Distance with 10 steps, [0.5, 2.5] (left), [1.2, 1.8] (right)

It's clear that the bond length is between 1.5 and 1.6, which is correct. And the ground state energy is about -7.88 and it's also as expected. However, the VQE curve is not as smooth as

Hartree-Fock and NumPyEigensolver. I double-checked with H2 model and when I zoom in small range, the VQE curve also not smooth, so I think it's just because the VQE algorithm and implementations instead of my implementation.
For all above experiment, I used 256 iteration for SPSA optimizer. I wonder if this is the reason for VQE non-smooth curve. So I did following experiment:

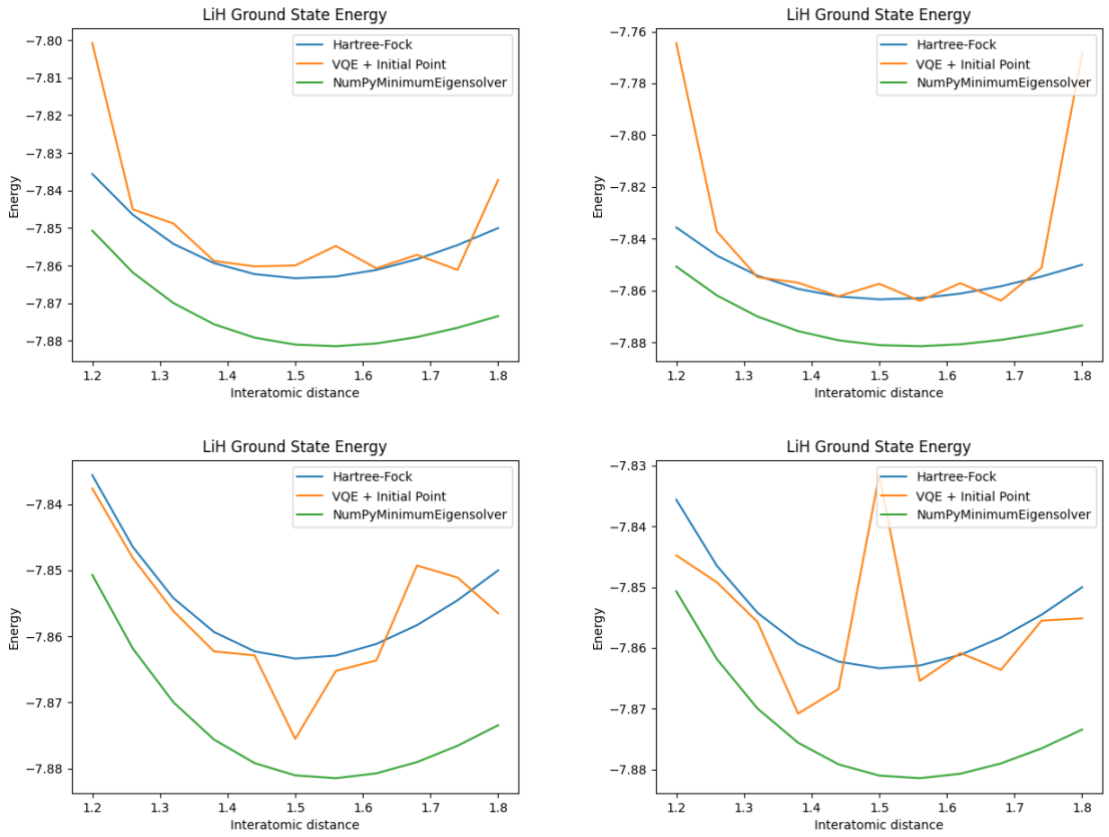

Figure 8. LiH Energy Curve with different iteration
256(top-left), 512(top-right), 1024(down-left), 2048(down-right)

Above for graph are LiH energy curve over interatomic distance in same range with same steps but different SPSA iteration times. Observe that with iteration number going up, the VQE curve is closer to Hartree-Fock curve(relatively). However, the curve with iteration of 2048 doesn't looks as good as the curve with iteration of 1024. It's not clear why did this happen but again, comparing to the curve with iteration of 256, this is a huge improvement.

3. Your choice of quantum chemistry experiment

> 3. Use classical simulation of noisy quantum executions to investigate the impact of noise on the VQE algorithm.

I implemented a VQE algorithm both with noise model and without noise model for H_2 molecule. Following is my result:

```
(env) xw336@ilab2:~/QuantumComputing/Qiskit$ python H2_noise_VQE.py
Exact Result: [-1.13722138]
VQE Result with noise: -1.1013389795826005
VQE Result without noise: -1.1325100462877604
```

Figure 9. VQE Result both with noise and without noise

As shown above, the result with noise is not as accurate as non-noise one. However, they are both not falling within chemical accuracy (defined as being within 0.0016 Hartree of the exact result).

Appendix: Source Code

hello_quantum.py

```
# This code is part of Qiskit.
#
# (C) Copyright IBM 2017, 2018.
#
# This code is licensed under the Apache License, Version 2.0. You may
# obtain a copy of this license in the LICENSE.txt file in the root directory
# of this source tree or at http://www.apache.org/licenses/LICENSE-2.0.
#
# Any modifications or derivative works of this code must retain this
# copyright notice, and modified files need to carry a notice indicating
# that they have been altered from the originals.

"""Example used in the README. In this example a Bell state is made."""

# Import Qiskit
from qiskit import QuantumCircuit
from qiskit import execute, IBMQ, BasicAer
from qiskit.providers.ibmq import least_busy

provider =
IBMQ.save_account('897bcc7b8ed8bb1ab9783d9e57d4d5551741efac0deb606de146c28afabec10853c751104509a8
856afbd58a2052ac034ef72baddc8a3b127b71e2ac80b751f3', overwrite=True)

# Create a Quantum Circuit
qc = QuantumCircuit(2, 2)

# Add a H gate on qubit 0, putting this qubit in superposition.
qc.h(0)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 1, putting
# the qubits in a Bell state.
qc.cx(0, 1)
# Add a Measure gate to see the state.
qc.measure([0, 1], [0, 1])
```

```
# See a list of available local simulators
print("BasicAer backends: ", BasicAer.backends())
backend_sim = BasicAer.get_backend('qasm_simulator')

# Compile and run the Quantum circuit on a simulator backend
job_sim = execute(qc, backend_sim)
result_sim = job_sim.result()

# Show the results
print(result_sim.get_counts(qc))

# Authenticate for access to remote backends
try:
    provider = IBMQ.load_account()
except:
    print("""WARNING: No valid IBMQ credentials found on disk.
             You must store your credentials using IBMQ.save_account(token, url).
             For now, there's only access to local simulator backends...""")
    exit(0)

# see a list of available remote backends
ibmq_backends = provider.backends()

print("Remote backends: ", ibmq_backends)
# Compile and run the Quantum Program on a real device backend
# select those with at least 2 qubits
try:
    least_busy_device = least_busy(provider.backends(
        filters=lambda x: x.configuration().n_qubits >= 2, simulator=False))
except:
    print("All devices are currently unavailable.")

print("Running on current least busy device: ", least_busy_device)

#running the job
job_exp = execute(qc, least_busy_device, shots=1024, max_credits=10)
```

```
result_exp = job_exp.result()


# Show the results
print('Counts: ', result_exp.get_counts(qc))
```

LiH_ground_state_energy.py

```
# Use PySCF, a classical computational chemistry software
# package, to compute the one-body and two-body integrals in
# molecular-orbital basis, necessary to form the Fermionic operator
from qiskit.chemistry.drivers import PySCFDriver, UnitsType
import warnings
warnings.filterwarnings('ignore')
driver = PySCFDriver(atom='Li .0 .0 .0; H .0 .0 1.595',
                     unit=UnitsType.ANGSTROM,
                     basis='sto3g')
molecule = driver.run()

# Build the qubit operator, which is the input to the VQE algorithm in Aqua
from qiskit.chemistry.core import Hamiltonian, TransformationType, QubitMappingType
operator =  Hamiltonian(
                transformation=TransformationType.FULL,
                qubit_mapping=QubitMappingType.PARITY, # Other choices: JORDAN_WIGNER, BRAVYI_KITAEV
                two_qubit_reduction=True,
                freeze_core=True,
                orbital_reduction=[-3, -2])
qubit_op, _ = operator.run(molecule)

# Control group baseline results
from qiskit.aqua.algorithms import NumPyMinimumEigensolver
result = NumPyMinimumEigensolver(qubit_op).run()
result = operator.process_algorithm_result(result)
print("=================================================")
print('Ground state energy (classical)  : {:.12f}'.format(result.energy))
print(result)
print("=================================================")

# setup a classical optimizer for VQE
from qiskit.aqua.components.optimizers import SPSA
optimizer = SPSA()

# setup the variational form for VQE
```

```
from qiskit.circuit.library import TwoLocal
var_form = TwoLocal(qubit_op.num_qubits, ['ry', 'rz'], 'cz', reps=5, entanglement='full')
print("Ansatz quantum circuit:")
print(var_form)

# setup and run VQE
from qiskit.aqua.algorithms import VQE
algorithm = VQE(qubit_op, var_form, optimizer)

# set the backend for the quantum computation
from qiskit import Aer
backend = Aer.get_backend('statevector_simulator')

result = algorithm.run(backend)
result = operator.process_algorithm_result(result)
print("=====================================================")
print('Ground state energy (quantum)  : {:.12f}'.format(result.energy))
print(result)
print("=====================================================")
```

H2_enerfy_curve.py

```
# Based on IBM Qiskit community tutorials
# https://github.com/qiskit-community/qiskit-community-tutorials/blob/master/chemistry/h2_vqe_initial_point.ipynb

import numpy as np
import matplotlib.pyplot as plt

from qiskit.chemistry.drivers import PySCFDriver, UnitsType
from qiskit.chemistry.core import Hamiltonian, TransformationType, QubitMappingType
from qiskit.circuit.library import TwoLocal
from qiskit.aqua.algorithms import NumPyMinimumEigensolver, VQE

from qiskit import BasicAer
from qiskit.aqua import QuantumInstance
from qiskit.aqua.components.optimizers import SPSA

import warnings
warnings.filterwarnings('ignore')

# Specify nuclei positions and species (atoms)
atoms = 'H .0 .0 -{0}; H .0 .0 {0}'
algorithms = [{'name': 'VQE'},
              {'name': 'NumPyMinimumEigensolver'}]
titles= ['VQE + Initial Point', 'NumPyMinimumEigensolver']

start = 0.7  # Minimum atomic separation to try
by    = 0.1  # Total range of atomic separations to try
steps = 10   # Number of steps to increase by
energies = np.empty([len(algorithms), steps+1])
hf_energies = np.empty(steps+1)
distances = np.empty(steps+1)

print('Processing step __', end='')
for i in range(steps+1):
    print('\b\b{:2d}'.format(i), end='', flush=True)
    d = start + i*by/steps
```

```
    # Given a atomic geometry specification, obtain molecular Hamiltonian
    driver = PySCFDriver(atom=atoms.format(d/2),
                         unit=UnitsType.ANGSTROM,
                         basis='sto3g')
    molecule = driver.run()

    # Build the qubit operator, which is the input to the VQE algorithm in Aqua
    operator =  Hamiltonian(
        transformation=TransformationType.FULL,
        qubit_mapping=QubitMappingType.PARITY, # Other choices: JORDAN_WIGNER, BRAVYI_KITAEV
        two_qubit_reduction=True,
        freeze_core=False,
        orbital_reduction=None)
    qubit_op, _ = operator.run(molecule)

    for j in range(len(algorithms)):

        if algorithms[j]['name'] == 'NumPyMinimumEigensolver':
            result = NumPyMinimumEigensolver(qubit_op).run()
        else:
            # Choice of classical optimizer
            optimizer = SPSA(max_trials=256)
            # Choice of ansatz
            var_form = TwoLocal(qubit_op.num_qubits, ['ry', 'rz'], 'cz', reps=3, entanglement='full')
            algo = VQE(qubit_op, var_form, optimizer, max_evals_grouped=1)
            result = algo.run(QuantumInstance(BasicAer.get_backend('statevector_simulator')))
            if j == 0:
                algorithms[j]['initial_point'] = result.optimal_point.tolist()

        result = operator.process_algorithm_result(result)
        energies[j][i] = result.energy
        hf_energies[i] = result.hartree_fock_energy

    distances[i] = d
print(' --- complete')

print('Distances: ', distances)
```

```
print('Energies:', energies)
print('Hartree-Fock energies:', hf_energies)


plt.plot(distances, hf_energies, label='Hartree-Fock')
for j in range(len(algorithms)):
    plt.plot(distances, energies[j], label=titles[j])
plt.xlabel('Interatomic distance')
plt.ylabel('Energy')
plt.title('H2 Ground State Energy')
plt.legend(loc='upper right')
fileName = "H2_Ground_State_Energy_start" + str(start) + "by" + str(by) + "steps" + str(steps) + ".png"
plt.savefig(fileName)
```

LiH_energy_curve.py

```
# Based on IBM Qiskit community tutorials
# https://github.com/qiskit-community/qiskit-community-tutorials/blob/master/chemistry/h2_vqe_initial_point.ipynb

import numpy as np
import matplotlib.pyplot as plt

from qiskit.chemistry.drivers import PySCFDriver, UnitsType
from qiskit.chemistry.core import Hamiltonian, TransformationType, QubitMappingType
from qiskit.circuit.library import TwoLocal
from qiskit.aqua.algorithms import NumPyMinimumEigensolver, VQE

from qiskit import BasicAer
from qiskit.aqua import QuantumInstance
from qiskit.aqua.components.optimizers import SPSA

import warnings
warnings.filterwarnings('ignore')

# Specify nuclei positions and species (atoms)
atoms = 'Li .0 .0 .0; H .0 .0 {0}'
algorithms = [{'name': 'VQE'},
              {'name': 'NumPyMinimumEigensolver'}]
titles= ['VQE + Initial Point', 'NumPyMinimumEigensolver']

start = 1.2  # Minimum atomic separation to try
by    = 0.6  # Total range of atomic separations to try
steps = 10   # Number of steps to increase by
trials = 2048
energies = np.empty([len(algorithms), steps+1])
hf_energies = np.empty(steps+1)
distances = np.empty(steps+1)

print('Processing step __', end='')
for i in range(steps+1):
    print('\b\b{:2d}'.format(i), end='', flush=True)
    d = start + i*by/steps
```

```
    # Given a atomic geometry specification, obtain molecular Hamiltonian
    driver = PySCFDriver(atom=atoms.format(d),
                         unit=UnitsType.ANGSTROM,
                         basis='sto3g')
    molecule = driver.run()

    # Build the qubit operator, which is the input to the VQE algorithm in Aqua
    operator =  Hamiltonian(
        transformation=TransformationType.FULL,
        qubit_mapping=QubitMappingType.PARITY, # Other choices: JORDAN_WIGNER, BRAVYI_KITAEV
        two_qubit_reduction=True,
        freeze_core=True,
        orbital_reduction=[-3, -2])
    qubit_op, _ = operator.run(molecule)

    for j in range(len(algorithms)):

        if algorithms[j]['name'] == 'NumPyMinimumEigensolver':
            result = NumPyMinimumEigensolver(qubit_op).run()
        else:
            # Choice of classical optimizer
            optimizer = SPSA(max_trials=trials)
            # Choice of ansatz
            var_form = TwoLocal(qubit_op.num_qubits, ['ry', 'rz'], 'cz', reps=3, entanglement='full')
            algo = VQE(qubit_op, var_form, optimizer, max_evals_grouped=1)
            result = algo.run(QuantumInstance(BasicAer.get_backend('statevector_simulator')))
            if j == 0:
                algorithms[j]['initial_point'] = result.optimal_point.tolist()

        result = operator.process_algorithm_result(result)
        energies[j][i] = result.energy
        hf_energies[i] = result.hartree_fock_energy

    distances[i] = d
print(' --- complete')
```

```
print('Distances: ', distances)
print('Energies:', energies)
print('Hartree-Fock energies:', hf_energies)

plt.plot(distances, hf_energies, label='Hartree-Fock')
for j in range(len(algorithms)):
    plt.plot(distances, energies[j], label=titles[j])
plt.xlabel('Interatomic distance')
plt.ylabel('Energy')
plt.title('LiH Ground State Energy')
plt.legend(loc='upper right')
fileName = "LiH_Ground_State_Energy_start" + str(start) + "by" + str(by) + "steps" + str(steps) + "-trials" + str(trials) +
".png"
plt.savefig(fileName)
```

H2_noise_VQE.py

```
# Based on IBM Qiskit community tutorials
# https://qiskit.org/textbook/ch-applications/vqe-molecules.html#Running-VQE-on-a-Noisy-Simulator

from qiskit.aqua.algorithms import VQE, NumPyEigensolver
import matplotlib.pyplot as plt
import numpy as np
from qiskit.chemistry.components.variational_forms import UCCSD
from qiskit.chemistry.components.initial_states import HartreeFock
from qiskit.circuit.library import EfficientSU2
from qiskit.aqua.components.optimizers import COBYLA, SPSA, SLSQP
from qiskit.aqua.operators import Z2Symmetries
from qiskit import IBMQ, BasicAer, Aer
from qiskit.chemistry.drivers import PySCFDriver, UnitsType
from qiskit.chemistry import FermionicOperator
from qiskit import IBMQ
from qiskit.aqua import QuantumInstance
from qiskit.ignis.mitigation.measurement import CompleteMeasFitter
from qiskit.providers.aer.noise import NoiseModel

import warnings
```

```
warnings.filterwarnings('ignore')

# Perpare the qubit operator representing the molecule's Hamiltonian
driver = PySCFDriver(atom='H .0 .0 -0.3625; H .0 .0 0.3625', unit=UnitsType.ANGSTROM, charge=0, spin=0,
basis='sto3g')
molecule = driver.run()
num_particles = molecule.num_alpha + molecule.num_beta
qubitOp = FermionicOperator(h1=molecule.one_body_integrals,
h2=molecule.two_body_integrals).mapping(map_type='parity')
qubitOp = Z2Symmetries.two_qubit_reduction(qubitOp, num_particles)

# Load a device coupling map and noise model from the IBMQ provider and create a quantum instance
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = Aer.get_backend("qasm_simulator")
device = provider.get_backend("ibmq_vigo")
coupling_map = device.configuration().coupling_map
noise_model = NoiseModel.from_backend(device.properties())
quantum_noise_instance = QuantumInstance(backend=backend,
                                         shots=8192,
                                         noise_model=noise_model,
                                         coupling_map=coupling_map,
                                         measurement_error_mitigation_cls=CompleteMeasFitter,
                                         cals_matrix_refresh_period=30)
quantum_non_noise_instance = QuantumInstance(backend=backend,
                                             shots=8192,
                                             coupling_map=coupling_map,
                                             measurement_error_mitigation_cls=CompleteMeasFitter,
                                             cals_matrix_refresh_period=30)

# Configure the optimizer, the variational form, and the VQE instance
exact_solution = NumPyEigensolver(qubitOp).run()
print("Exact Result:", np.real(exact_solution.eigenvalues) + molecule.nuclear_repulsion_energy)
optimizer = SPSA(maxiter=100)
var_form = EfficientSU2(qubitOp.num_qubits, entanglement="linear")
vqe = VQE(qubitOp, var_form, optimizer=optimizer)
noise_ret = vqe.run(quantum_noise_instance)
```

```
non_noise_ret = vqe.run(quantum_non_noise_instance)
noise_vqe_result = np.real(noise_ret['eigenvalue']+ molecule.nuclear_repulsion_energy)
non_noise_vqe_result = np.real(non_noise_ret['eigenvalue']+ molecule.nuclear_repulsion_energy)
print("VQE Result with noise:", noise_vqe_result)
print("VQE Result without noise:", non_noise_vqe_result)
```