Introductory Workshop on Quantum Computational Chemistry

Oxford Quantum Information Society,
Cambridge Quantum Computing

February 22, 2020

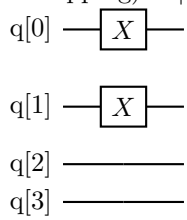
For these exercises, you will need a Python 3.6 or 3.7 environment with pip. To install all of the required packages, run the following in your terminal:

```
pip install pytket pytket-qiskit openfermion scipy
```

We recommend using the `oxfordQIS.py` script as a starting point for these exercises (`https://github.com/CQCL/pytket/blob/master/examples/oxfordQIS.py`).

Hartree–Fock energy

In order to construct a UCCSD ansatz, the first step is to initialise the state to the Hartree-Fock state. For the H_2 molecule in the minimal basis (STO-3G) the Hartree—Fock reference state (with the Jordan-Wigner qubit mapping) is $|1100\rangle$ which can be generated by the following circuit:



Create this circuit using pytket and calculate the Hartree—Fock energy. That is, compute the expectation value of the H_2 STO-3G Hamiltonian using the `AerBackend`. The Hamiltonian operator of the H_2 is provided in the `oxfordQIS.py` file. In addition, the `h2_JW_sto3g_ansatz()` function creates the full UCCSD ansatz circuit for H_2. Set all the parameters of the UCCSD ansatz to zero and compute the expectation value of the Hamiltonian. How does this compare to the Hartree—Fock energy and why?

Plotting Convergence of VQE

The `minimize` method in `scipy` can take an optional argument `callback`. If you pass a function in here, the optimiser will call your function after each iteration of the optimisation with the current parameter vector.

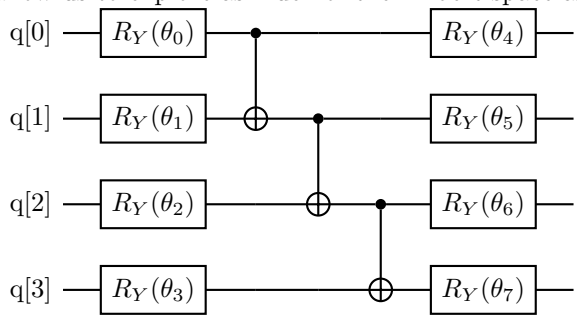
Use the `callback` option (or any other means) to track the energy value over time and plot this against number of calls to the objective function. Change the method used by the `minimize` function from `'Nelder-Mead'` to `'BFGS'` and compare the convergence rate.

Expectation Values from Statevector Simulations

The current objective function in `oxfordQIS.py` expects the backend to return a summary of measurement counts. For small systems, using a statevector simulation can work faster and give a more accurate result since there is no sampling error. Adapt the objective function to compute the expectation value $\langle\psi|\hat{H}|\psi\rangle$ by multiplication of the state $|\psi\rangle$ with the given Hamiltonian $\hat{H}$ and test it by changing the `backend` to `AerStateBackend`.

Hardware-Efficient Ansatz

The UCCSD ansatz is physically-motivated, where each of the parameters corresponds to an excitation operator. Unfortunately, these excitation operators are costly to perform with primitive gates. Hardware-efficient ansätze are shallow circuits that pack in as many parameterised gates as possible. The goal here is to allow us to explore as much of the Hilbert space as possible in a small circuit.



The shallow circuit gives more reliable and accurate measurements, but the large parameter count makes the classical optimisation of VQE more expensive and the structure allows for the generation of unphysical states (according to the chemical model).

Implement the above ansatz and run the same VQE routine. How does the rate of convergence differ?

Number operator

The number of electrons in the molecule is a conserved physical quantity. The number operator can be given by fermionic creation and annihilation operators:

$$\hat{N} = a_0^\dagger a_0 + a_1^\dagger a_1 + a_2^\dagger a_2 + a_3^\dagger a_3$$

Using the UCCSD ansatz for H_2 provided, compute the expectation value of number operator. In the `oxfordQIS.py` script the number operator is expressed with Openfermion's fermion operators. (Hint: Openfermion has a Jordan–Wigner transformation function.)

Try varying the UCCSD parameters and investigate how the expectation value changes. Compute the expectation value of the number operator with the Hardware Efficient Ansatz (HAE). Try different set of arbitrary parameters for the HAE again. What difference can you observe between the expectation values computed by the HAE and UCCSD ansatz?

Run a VQE optimization with the hardware efficient ansatz. Verify that the ground state obtained by HAE is physical. (Hint: for H_2 one expects one spin-up and one spin-down electron.)

Error Mitigation

Readout error of a single qubit system can be characterised by a probability matrix $P = \begin{pmatrix} p_{0,0} & p_{0,1} \\ p_{1,0} & p_{1,1} \end{pmatrix}$ where $p_{i,j}$ is the probability that i is measured when state $|j\rangle$ is prepared. A perfect device would have $p_{0,0} = p_{1,1} = 1$. This extends to multiple qubit systems in the obvious way.

These errors rarely act in a symmetric way, i.e. $p_{0,0} \neq p_{1,1}$. This can introduce a systematic error in results. We can symmetrise these errors by performing an X gate just before the measurement and correcting the result classically for half of the shots. Write a function that takes a circuit and a backend, and appends X gates to each qubit and corrects the readouts afterwards. Compare the readout distributions for a circuit between a noiseless simulation, a noisy simulation, and applying this technique to half of the shots in the noisy simulation to symmetrise the errors.

To simulate a noisy device, we can add different noise models to the Aer simulator. For this example, we can focus on just adding readout error:

```
from qiskit.providers.aer.noise import NoiseModel
```

```
my_noise_model = NoiseModel()
my_noise_model.add_all_qubit_readout_error([[0.9, 0.1], [0.2, 0.8]])
backend = AerBackend(my_noise_model)
```

Measurement Reduction

Currently, we are running a different circuit for each term from the Hamiltonian in the expectation value measurement. Measurement reduction techniques aim to reduce the number of circuit evaluations needed by using a single set of measurement results to compute multiple terms.

For example, the measurement circuit we presented for the term $Z \otimes Z$ simultaneously measures the terms $Z \otimes I$ and $I \otimes Z$. We can obtain either of these single-qubit measurements by ignoring one of the measurement results from the $Z \otimes Z$ measurement circuit.

Write some code which takes a counts dictionary (map from readout to frequency) and projects out a subset of the bits and use this to calculate an expectation value for the following Hamiltonian by executing only two circuits on the backend:

$$\hat{H} = 0.5IIII + 0.2ZIII - 0.2IZII + 0.2IIZI - 0.2IIIZ + 0.3ZIZI + 0.4IZIZ - 0.6ZZZZ + 0.1XXII + 0.1IIYY$$

Executing on an IBM-Q Device

If you don't already have credentials for accessing the IBM-Q devices, you can set this up for free by following the instructions in the "Access IBM Quantum Systems" section of `https://qiskit.org/documentation/install.html`.

Log in online and select a device you want to run your circuits on (for example, `ibmq_london`). Swap out the `AerBackend` for an `IBMQBackend` and use this to compute a single expectation value. This is likely to be significantly slower than the simulator because of the latency of sending the job off to the device and waiting for it to reach the front of the queue.

```
from pytket.backends.ibm import IBMQBackend

# Create the backend that connects to the device
backend = IBMQBackend('ibmq_london')
# Send the circuits off to run
backend.process_circuits(all_circuits, n_shots=2000)
counts = backend.get_counts(all_circuits[0])
# Process data...
```