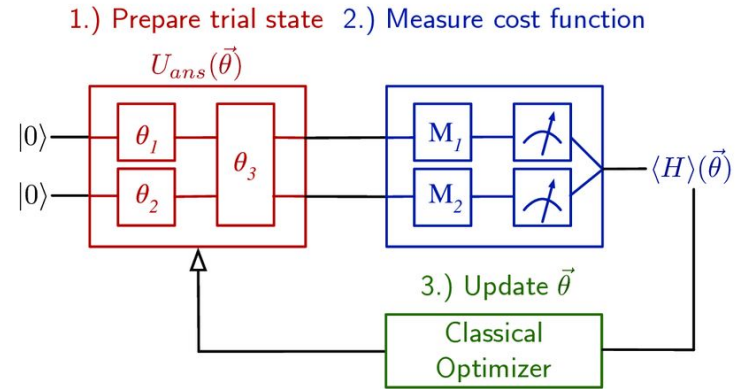


Variational Quantum Eigensolvers using the Hubbard Model of Interacting Fermions

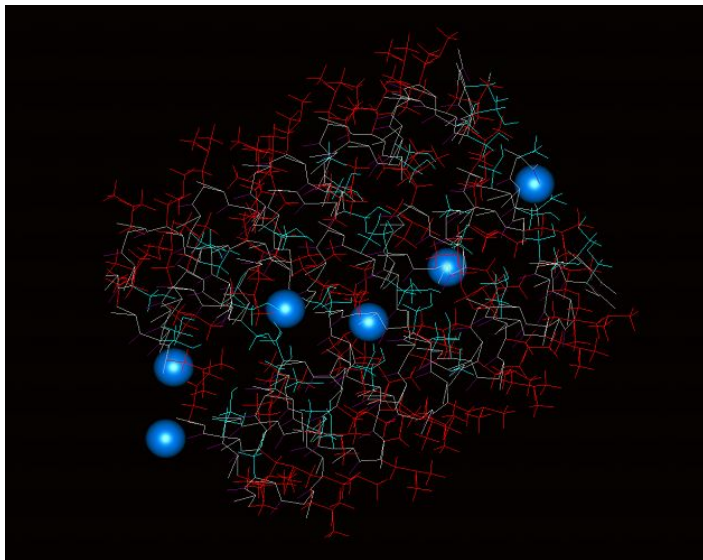
Danny Duke

What is a VQE

- A Variational Quantum Eigensolver is a quantum computing algorithm used to find the lowest possible eigenvalue of a hamiltonian in a given quantum system.
- Hamiltonian describes the total energy of a system.
- Eigenvalues are the possible energies of the hamiltonian given certain parameters.



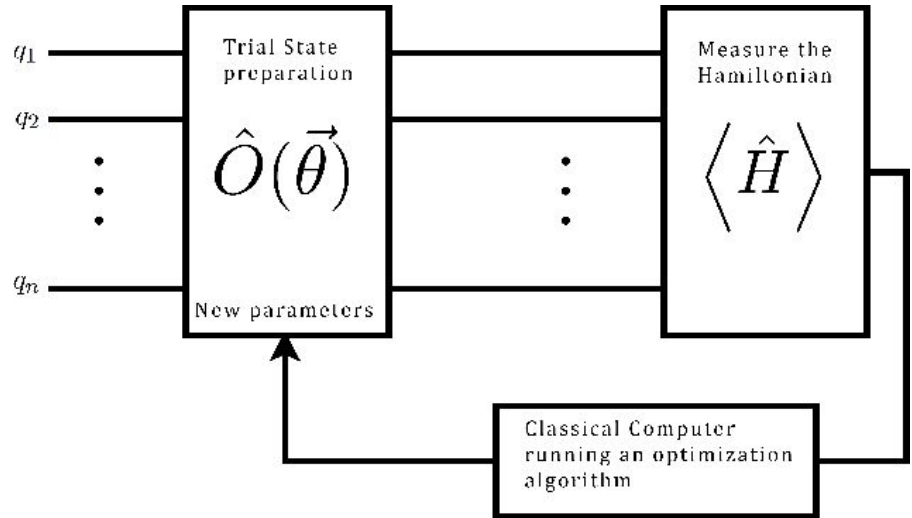
Why VQE



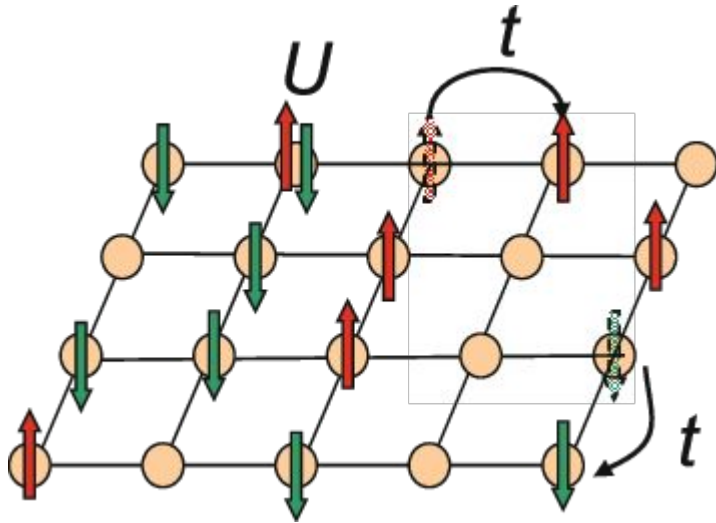
- When you map the properties of a molecule onto a qubit hamiltonian, you can use a VQE to find the lowest possible Eigenvalue of the hamiltonian, which in this case would represent the ground state of the molecule.
- This principle can be especially useful when simulating a molecule that is not viable to physically obtain.
- Once you know the ground state of a molecule is known, it is easy to derive many other properties such as electromagnetic dipoles, polarizability, and other forces. Additionally, properties of the reaction between two molecules can be determined by the difference in their energies.
- Materials science requires the testing of many different materials of uses such as advanced medicine, research into superconductivity, and creating more durable and sustainable materials that meet the ever growing needs of society. Not having to physically obtain and test every material we use is a necessary step to designing the bricks with which we build the future.

How does a VQE Work?

- Starts by applying a parameterized transformation to the hamiltonian which puts it in a state that the energy can be measured.
- Measures the energy based on the parameter theta.
- Classical computer uses this information to prepare a new guess for parameter theta



Hubbard Model of Interacting Fermions



- A lattice of sites that represent electron orbitals.
- Electrons are stored with their spin in the lattice, and are able to move from one orbital to a neighboring orbital,
- Pauli exclusion principle applies.
- t is the hopping energy for the electron
- U is the “Hubbard term” which describes the energy it takes for any two electrons in the same orbital to have opposite spins (Coulomb energy).



Hamiltonian of the System Described by the Hubbard Model

$$H = -t \sum_{\sigma} \sum_{\langle ij \rangle} a_{j,\sigma}^{\dagger} a_{i,\sigma} + U \sum_i \hat{n}_{i,\uparrow} \hat{n}_{i,\downarrow},$$

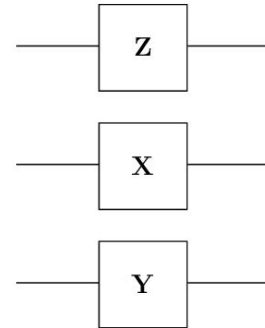
- The sum of the hopping energy and the coulomb energy.
- t is the hopping integral.
- There is one t term for each pair of neighboring sites.
- $a_{j,\sigma}^{\dagger} a_{i,\sigma}$ is the term for creation and annihilation at neighboring indices i and j .
- U represents the Coulomb energy for each orbital that contains 2 electrons, in a higher energy state due to their necessity to have opposite spins. This is repeated for each index of the lattice

Mapping Hamiltonian to Qubits: Jordan-Wigner

- To interact with the Hamiltonian in the Quantum Computer, it must be mapped onto gates that can be applied to qubits.

$$H = -t \sum_{\sigma} \sum_{\langle ij \rangle} a_{j,\sigma}^{\dagger} a_{i,\sigma} + U \sum_i \hat{n}_{i,\uparrow} \hat{n}_{i,\downarrow}$$

→





Pauli Matrices

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$




Jordan Wigner Cont'd

- The Jordan Wigner approach to mapping the Hamiltonian works by defining the creation and annihilation operators of the Hubbard model in terms of basic matrices/quantum gates.

$$a_{i+1}^\dagger a_i = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$X - iY = \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix}$$

$$X + iY = \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix}$$


$$a_{i+1}^\dagger a_i = \begin{pmatrix} \frac{X - iY}{2} \\ 1 \end{pmatrix} \begin{pmatrix} \frac{X + iY}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$



Developing the Hamiltonian (non-interacting)

$$\begin{aligned}a_{i+1}^\dagger a_i &= \left(\frac{X_{i+1} - iY_{i+1}}{2}\right) Z_i \cdots Z_0 \left(\frac{X_i + iY_i}{2}\right) Z_{i-1} \cdots Z_0 \\&= \left(\frac{X_{i+1} - iY_{i+1}}{2}\right) \left(\frac{Z_i X_i + iZ_i Y_i}{2}\right) = \left(\frac{X_{i+1} - iY_{i+1}}{2}\right) \left(\frac{X_i + iY_i}{2}\right) \\&= \frac{1}{4} (iX_{i+1} Y_i + X_{i+1} X_i + Y_{i+1} Y_i - iY_{i+1} X_i).\end{aligned}$$

$$a_i^\dagger a_{i+1} = \frac{1}{4} (iX_i Y_{i+1} + X_i X_{i+1} + Y_i Y_{i+1} - iY_i X_{i+1}).$$

$$a_{i+1}^\dagger a_i + a_i^\dagger a_{i+1} = \frac{1}{2} (X_i X_{i+1} + Y_i Y_{i+1}).$$



Developing the Hamiltonian (Interacting)

$$a_i^\dagger a_i = \left(\frac{X_i - iY_i}{2}\right) Z_{i-1} \cdots Z_0 \left(\frac{X_i + iY_i}{2}\right) Z_{i-1} \cdots Z_0 = \frac{1}{2}(I - Z_i),$$

$$a_{i\uparrow}^\dagger a_{i\uparrow} a_{i\downarrow}^\dagger a_{i\downarrow} = \frac{1}{4}(I - Z_{i\uparrow} - Z_{i\downarrow} + Z_{i\uparrow} Z_{i\downarrow}).$$



Time Evolution in terms of Basic Gates

$$U(\theta_j) = \exp(i\theta_j(H_0 + H_U)) = \exp(i\theta_{j,xe}H_{xe}) \exp(i\theta_{j,xo}H_{xo}) \exp(i\theta_{j,ye}H_{ye}) \exp(i\theta_{j,yo}H_{yo}) \exp(i\theta_{j,z}H_U),$$

$$\exp(i\theta_{j,xe}H_{xe}) = \Pi_{\sigma} \Pi_{i=2k} \exp(-\frac{t}{2}i\theta_{j,xe}X_{i\sigma}X_{i+1\sigma}),$$

$$\exp(i\theta_{j,ye}H_{ye}) = \Pi_{\sigma} \Pi_{i=2k} \exp(-\frac{t}{2}i\theta_{j,ye}Y_{i\sigma}Y_{i+1\sigma}),$$

$$\exp(i\theta_{j,z}H_U) = \Pi_i \exp(\frac{U}{4}i\theta_{j,z}(I - Z_{i\uparrow} - Z_{i\downarrow} + Z_{i\uparrow}Z_{i\downarrow})).$$



Time Evolution in terms of Basic Gates

$$e^{i\theta X_i X_j} = \begin{pmatrix} \cos(\theta) & 0 & 0 & i \sin(\theta) \\ 0 & \cos(\theta) & i \sin(\theta) & 0 \\ 0 & i \sin(\theta) & \cos(\theta) & 0 \\ i \sin(\theta) & 0 & 0 & \cos(\theta) \end{pmatrix}. \quad e^{i\theta(I - Z_{i\uparrow} - Z_{i\downarrow} + Z_{i\uparrow} Z_{i\downarrow})} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$$

$$e^{i\theta Y_i Y_j} = \begin{pmatrix} \cos(\theta) & 0 & 0 & -i \sin(\theta) \\ 0 & \cos(\theta) & i \sin(\theta) & 0 \\ 0 & i \sin(\theta) & \cos(\theta) & 0 \\ -i \sin(\theta) & 0 & 0 & \cos(\theta) \end{pmatrix}$$



Time Evolution in terms of Basic Gates

$$e^{i\theta X_i X_j} = CX_i \cdot Rx_i(\theta) \otimes I_j \cdot CX_i, \quad e^{i\theta(I - Z_{i\uparrow} - Z_{i\downarrow} + Z_{i\uparrow} Z_{i\downarrow})} = cRz(\theta),$$

$$e^{i\theta Y_i Y_j} = CX_i \cdot I_i \otimes H_j \cdot CX_j \cdot I_i \otimes Rx_j(-\theta) \cdot CX_j \cdot I_i \otimes H_j \cdot CX_i.$$



VQE - Initializing Circuit

- Define molecule you are describing with properties such as number of spin up and down electrons, lattice sites, hopping energy, and Coulomb energy.
- Circuit is initialized using twice as many qubits as there are lattice sites to accommodate both up and down spins.

```
import numpy as np
from copy import deepcopy

from qiskit import *







M = 5      # number of lattice sites
t = 1.0    # hopping energy
U = 0.1    # Coulomb energy
Nu = 3     # number of spin-up electrons
Nd = 3     # number of spin-down electrons
```

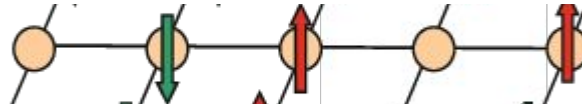

Creating the Molecule in terms of Qubits

```
# compose quantum circuit

def compose_initial_circuit(circuit, _M, _Nu, _Nd):
    for i in range(_Nu):
        circuit.x(i)
    for i in range(_Nd):
        circuit.x(_M+i)
    return circuit

circ_initial = compose_initial_circuit(deepcopy(circ_empty), M, Nu, Nd)
circ_initial.draw()
```

q_0: |0> 
q_1: |0> 
q_2: |0> 
q_3: |0> _____
q_4: |0> _____
q_5: |0> 
q_6: |0> 
q_7: |0> 
q_8: |0> _____
q_9: |0> _____



- Linear Hubbard model
- X gates to initialize electrons in at different indices
- Top half of Qubits represent spin up, bottom half represent spin down.

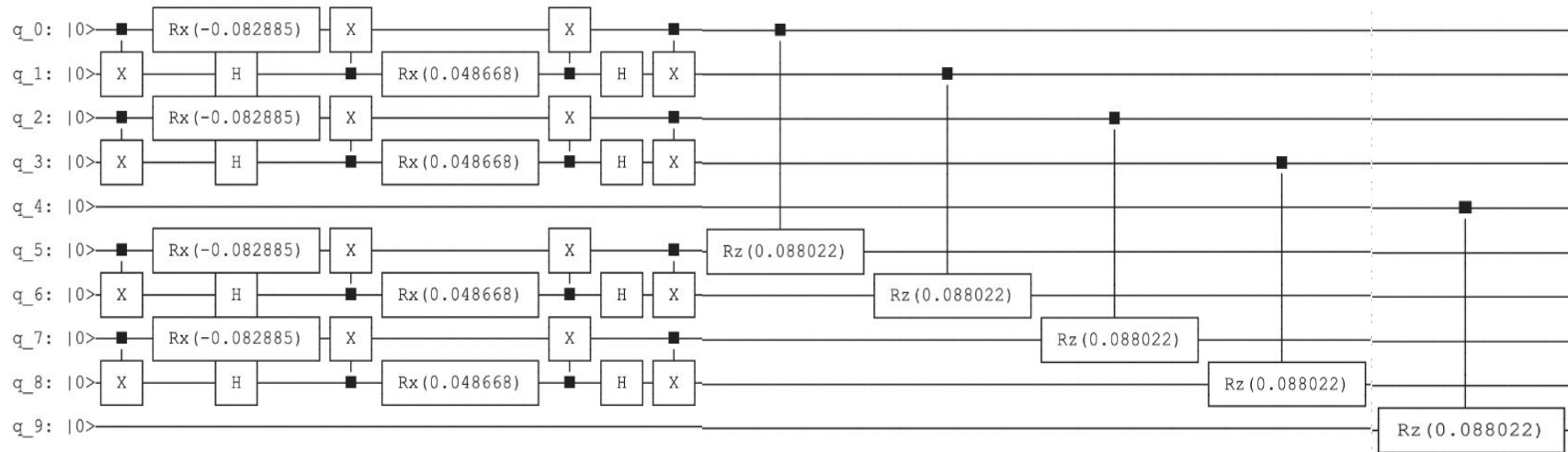
VQE - Creating Ansatz Function

```
def compose_ansatz_circuit(circuit, _M, _S, _theta):
    # scale theta based on t and U
    _theta[:, :4] = -t/2.0*_theta[:, :4]
    _theta[:, 4] = U*_theta[:, 4]
    # loop over S Trotter-Suzuki steps
    for j in range(_S):
        # XX and YY interactions: loop over even and odd terms
        for parity in [0]:
            # loop over orbitals
            for i in range(_M-1):
                if i%2 == parity:
                    # loop over spin up and down qubits
                    for spin_index in [0, _M]:
                        circuit.cx(spin_index+i, spin_index+i+1)
                        circuit.rx(_theta[j, 2*parity+0], spin_index+i) # theta_j,x(e)
                        circuit.h(spin_index+i+1)
                        circuit.cx(spin_index+i+1, spin_index+i)
                        circuit.rx(_theta[j, 2*parity+1], spin_index+i+1) # theta_j,y(e)
                        circuit.cx(spin_index+i+1, spin_index+i)
                        circuit.h(spin_index+i+1)
                        circuit.cx(spin_index+i, spin_index+i+1)

            # barrier
            circuit.barrier()
            # ZZ interactions
            # loop over orbitals
            for i in range(_M):
                circuit.crz(_theta[j, 4], i, _M+i)
    return circuit

circ_ansatz = compose_ansatz_circuit(deepcopy(circ_empty), M, S, theta)
circ_ansatz.draw()
```

- Quantum aspect of the algorithm.
- Implement the series of gates and rotations necessary to mimic the hamiltonian given an input parameter θ .
- Gates are determined by time evolution.





Measurement Function

- Circuit for measuring the energy of the ansatz, i.e. the eigenvalue of the hamiltonian.

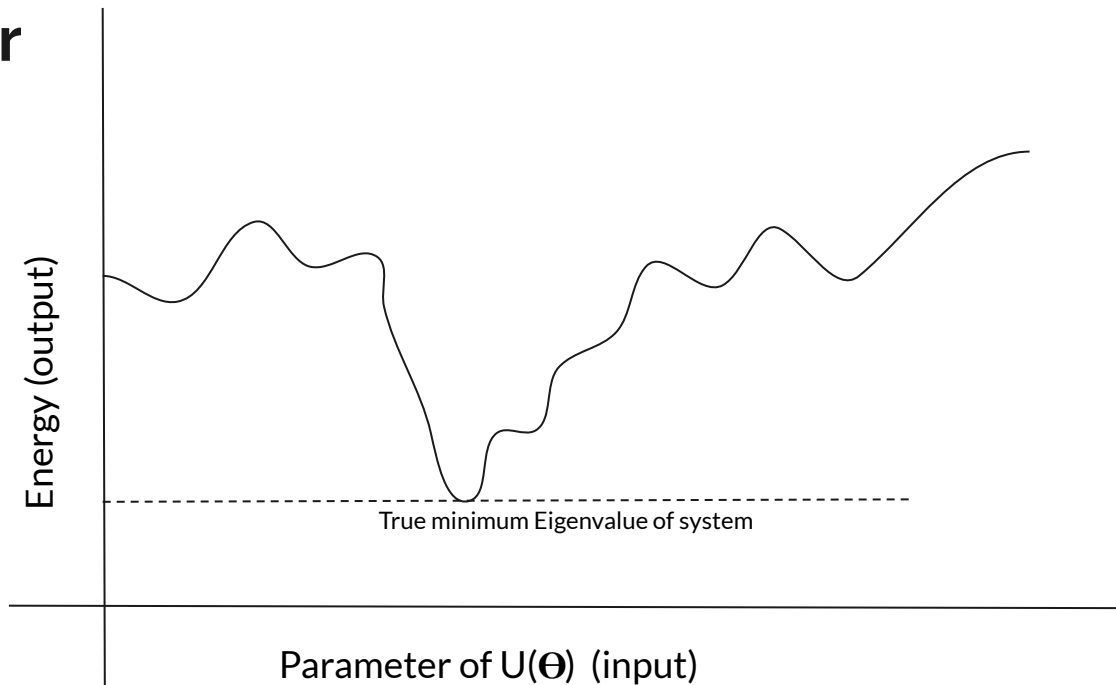
```
# compose quantum circuit

def compose_measurement_circuit(circuit, _M, which_term):
    if which_term == "xe":
        for i in range(_M-1):
            circuit.x(i)
            circuit.x(_M+i)
    elif which_term == "xo":
        for i in range(_M-1):
            circuit.x(i+1)
            circuit.x(_M+i+1)
    elif which_term == "ye":
        for i in range(_M-1):
            circuit.y(i)
            circuit.y(_M+i)
    elif which_term == "yo":
        for i in range(_M-1):
            circuit.y(i+1)
            circuit.y(_M+i+1)
    else:
        pass
    return circuit

circ_measurement = compose_measurement_circuit(deepcopy(circ_empty), M, "xe")
circ_measurement.draw()
```

Classical Optimizer

- VQE is a hybrid algorithm (relies on both classical and quantum computers).
- Basic machine learning
- Takes the energy output of the measurement circuit and the parameter θ that resulted in that energy, and then produces an alternate guess for θ that will result in a lower energy until it approaches the absolute minima.





Any Questions?



References

Smfarzaneh. "Vqe-Tutorials/Hubbard-Circuit.ipynb at Master · Smfarzaneh/Vqe-Tutorials." GitHub, github.com/smfarzaneh/vqe-tutorials/blob/master/hubbard-circuit.ipynb.

arXiv:2111.05176 [quant-ph]

arXiv:2103.12097 [cond-mat.str-el]

Bradben. "Jordan-Wigner Representation - Azure Quantum." Azure Quantum | Microsoft Docs, docs.microsoft.com/en-us/azure/quantum/user-guide/libraries/chemistry/concepts/jordan-wigner.

Hal Tasaki 1998 J. Phys.: Condens. Matter 10 4353

"Taylor Series." From Wolfram MathWorld, mathworld.wolfram.com/TaylorSeries.html.

"SuzukiTrotter¶." SuzukiTrotter - Qiskit 0.37.2 Documentation, qiskit.org/documentation/stubs/qiskit.synthesis.SuzukiTrotter.html.

arXiv:2206.08798 [quant-ph]