Solving Partial Differential Equations with Variational Quantum Algorithms

Kevin Ngo
McMahon Lab, Department of Applied and Engineering Physics, Cornell University
(Dated: December 22, 2019)

Nonlinear partial differential equations have wide applications from fluid dynamics to biology systems. However, these equations can often be very complex without elegant analytical solutions. Quantum computation provides an opportunity to improve upon the efficiency of previously existing numerical methods. In this paper, we examine variational quantum algorithms in particular, in order to verify its ability to solve the nonlinear Schrodinger's equation. We deconstruct the algorithm presented by Lubasch et al. [1] and attempt to simulate a simplified version on a classical computer.

I. OVERVIEW OF VARIATIONAL QUANTUM ALGORITHMS

Variational quantum algorithms aim to solve problems according to the variational method in quantum mechanics. Broadly, the variational method seeks to find the lowest energy eigenstate of a system by initializing an ansatz parametrized on some values and then minimizing the expectation energy value based on those parameters. For variational quantum algorithms, this is typically done with a hybrid approach, evaluating the expectation energy value with a parametrized quantum circuit and then performing the optimization clasically.

A. Variational Quantum Eigensolver

A motivating example to apply a variational quantum algorithm is the Max-Cut problem. The Max-Cut problem can be described as follows: Given a graph $G = (V, E)$, select a subset $S \subseteq V$ such that the number of edges between S and the complement of S is maximized [2]. The weighted version of the problem is modified such that each edge has a weight and the total weight of the edges between S and the complement of S is maximized. For this problem, the dimension of the problem space grows exponentially (the number of configurations growing with respect to the number of vertices), which makes it well suited to be used with a quantum algorithm.

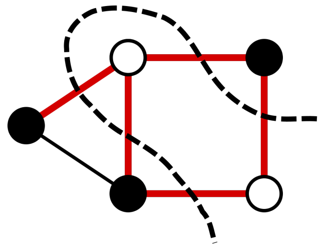


FIG. 1: Max-cut problem example, represented with graph colorings such that the white nodes are in S and the black nodes are in the complement of S

For a graph with n vertices, the 2^n configurations can be represented in a vector of length 2^n where each element corresponds to a different configuration. The energy states for these configurations are the sum of all edge weights where edges from S to the complement of S are negative and all other weights are positive. These can be represented in a Hamiltonian operator which may be a diagonal matrix with the corresponding energy values on the diagonal. The eigenvalues then correspond to the energy values and finding the eigenvector associated with the lowest eigenvalue is equivalent to finding the solution to the Max-Cut problem [3].

This can be solved using a variational quantum eigensolver, as the algorithm seeks to find the eigenvector associated with the lowest eigenvalue for a system. n qubits are required to encapsulate the 2^n configurations. An ansatz, $\psi(\lambda)$, is generated by a quantum circuit, parametrized on some values λ. The expectation value $\langle \psi(\lambda) | H | \psi(\lambda) \rangle$ is measured by sampling a quantum circuit. This value is minimized on a classical computer and the parameters λ are modified accordingly. This is repeated until it converges on a solution.

II. SOLVING THE NONLINEAR SCHRODINGER'S EQUATION

This project aimed to replicate the algorithm proposed by Lubasch et al. [1]. In their paper, they solve the one-dimensional Gross-Pitaevskii equation shown in Eq. (1) by minimizing the cost function in Eq. (2), where $\langle\!\langle K \rangle\!\rangle$, $\langle\!\langle P \rangle\!\rangle$, $\langle\!\langle I \rangle\!\rangle$ are the kinetic, potential, and interaction energies respectively.

$$\left[-\frac{1}{2} \frac{d^2}{dx^2} + V(x) + g|f(x)|^2 \right] f(x) = E f(x) \tag{1}$$

$$C = \langle\!\langle K \rangle\!\rangle + \langle\!\langle P \rangle\!\rangle + \langle\!\langle I \rangle\!\rangle \tag{2}$$

This equation is solved by applying the finite difference method to discretize the wave function on an interval $[a, b]$ into N equidistant grid points where h_N is the grid spacing. These grid points are represented with n qubits where $2^n = N$. The value of the wave function $\psi_k = \psi(x_k)$, where x_k is one of the $k \in \{0, ..., N-1\}$ grid points, is represented as the probability value of each of the N

configurations of qubits. This maintains the necessary normalization condition $\sum_{k=0}^{N-1} |\psi_k|^2 = 1$. The boundary condition $\psi_0 = \psi_{N-1}$ is also enforced. Thus, the energies for the discretized wave function are described in Eq. (3) below:

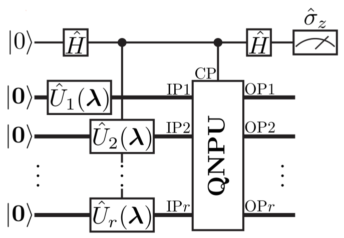
$$\langle\langle K \rangle\rangle = -\frac{1}{2}\frac{1}{h_N^2}\sum_{k=0}^{N-1}\psi_k^*(\psi_{k+1} - 2\psi_k + \psi_{k-1}), \tag{3a}$$

$$\langle\langle P \rangle\rangle = \sum_{k=0}^{N-1}[\psi_k^* V(x_k)\psi_k], \tag{3b}$$

$$\langle\langle I \rangle\rangle = \frac{1}{2}\frac{g}{h_N}\sum_{k=0}^{N-1}|\psi_k|^4, \tag{3c}$$

A. Circuit

The circuit presented in the Lubasch et al. paper generates the ansatz for the wave function through the $\hat{U}_j(\boldsymbol{\lambda})$ networks depicted in Fig. 3 such that $|\psi(\boldsymbol{\lambda})_j\rangle = \hat{U}_j(\boldsymbol{\lambda})|\mathbf{0}\rangle$. This ansatz is based on matrix product states. The wave function then passes through the quantum nonlinear processing unit (QNPU) where the cost function is evaluated. The variational method is applied here by minimizing this cost value on a classical computer to find optimal parameters $\boldsymbol{\lambda}$.



$|0\rangle$
$\hat{H}$
$\hat{H}$
$\hat{\sigma}_z$
$|\mathbf{0}\rangle$
$\hat{U}_1(\boldsymbol{\lambda})$
$\hat{U}_2(\boldsymbol{\lambda})$
$\hat{U}_r(\boldsymbol{\lambda})$
CP
IP1
IP2
IPr
OP1
OP2
OPr
QNPU

FIG. 2: Overview of circuit presented in the Lubasch et al. paper

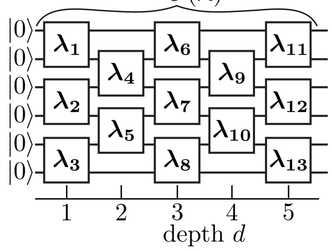


FIG. 3: Network generating ansatz for wave function

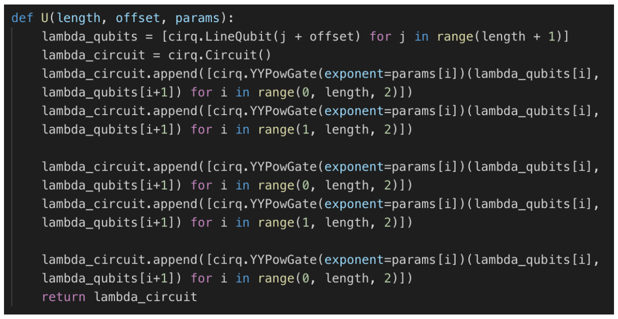
III. SIMULATION

To simulate the algorithm presented in the Lubasch et al. paper, the Cirq library was used. The library allows us to build and execute quantum circuits on a quantum computer. One detail that may need clarification is that occasionally, the qubits are indexed by a single integer, and at other times they are indexed by an ordered pair. This refers to the LineQubit and GridQubit objects in the Cirq library, which limit multi-qubit gate connections on physical quantum computers, but do not pose such constraints when simulating on a classical computer. They are used interchangeably in the following examples.

A. Replicating circuit components

1. $\hat{U}(\boldsymbol{\lambda})$ network

The $\hat{U}(\boldsymbol{\lambda})$ networks were generated such that there can be a variable length of n qubits with an offset in order to be connected with other components of the circuit. At the moment, the depth is always $d = 5$ with only n parameters (these are replicated as much as necessary), but may be modified in the future in order to improve the ansatz.

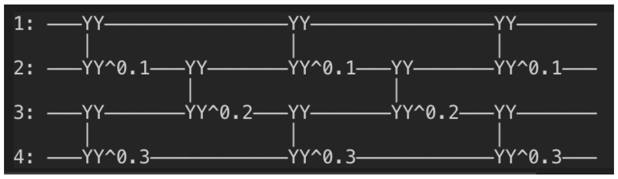


```
def U(length, offset, params):
    lambda_qubits = [cirq.LineQubit(j + offset) for j in range(length + 1)]
    lambda_circuit = cirq.Circuit()
    lambda_circuit.append([cirq.YYPowGate(exponent=params[i])(lambda_qubits[i],
    lambda_qubits[i+1]) for i in range(0, length, 2)])
    lambda_circuit.append([cirq.YYPowGate(exponent=params[i])(lambda_qubits[i],
    lambda_qubits[i+1]) for i in range(1, length, 2)])

    lambda_circuit.append([cirq.YYPowGate(exponent=params[i])(lambda_qubits[i],
    lambda_qubits[i+1]) for i in range(0, length, 2)])
    lambda_circuit.append([cirq.YYPowGate(exponent=params[i])(lambda_qubits[i],
    lambda_qubits[i+1]) for i in range(1, length, 2)])

    lambda_circuit.append([cirq.YYPowGate(exponent=params[i])(lambda_qubits[i],
    lambda_qubits[i+1]) for i in range(0, length, 2)])
    return lambda_circuit
```

FIG. 4: Code snippet for generating $\hat{U}(\boldsymbol{\lambda})$ network

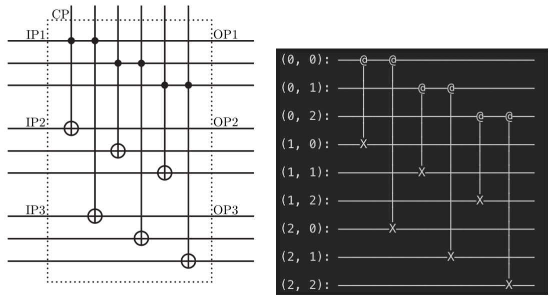


```
1: ───YY─────────────────YY─────────────────YY────────
      │                  │                  │
2: ───YY^0.1───YY────────YY^0.1───YY────────YY^0.1───
               │                  │
3: ───YY───────YY^0.2────YY───────YY^0.2────YY────────
      │                  │                  │
4: ───YY^0.3─────────────YY^0.3─────────────YY^0.3───
```

FIG. 5: $\hat{U}(\boldsymbol{\lambda})$ network generated for $n = 4$, offset = 1, and parameters = [0.1, 0.2, 0.3]

2. Interaction energy term

Each of the energy terms are found within the QNPU unit depicted in Fig. 2. The nonlinear term $|\psi|^4$ is calculated through a sequence of CNOT gates shown below.

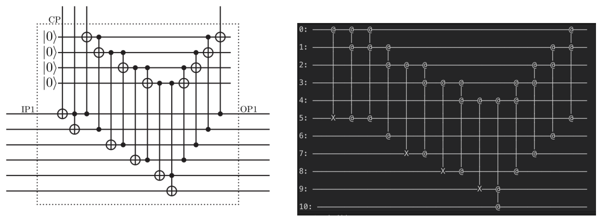


(a) Reference (b) Cirq simulation

FIG. 6: Circuit for interaction energy term for $n = 3$

3. *Kinetic energy term*

The kinetic energy term is currently coded statically for $n = 6$, unlike the previous two components which are parametrized for a variable n. This is because it is currently unclear to us how the circuit corresponds to the kinetic energy term in Eq. 3a, but will be extended in the future once a better understanding is gained.



(a) Reference (b) Cirq simulation

FIG. 7: Circuit for kinetic energy term for $n = 6$

4. *Potential energy term*

The potential energy term consists of another network of CNOT gates as well as a component for the external potential V. This has not been implemented, but it is the final component of the QNPU.

B. Simplifying and solving

In order to begin testing the efficacy of the algorithm, Eq. 1 was modified to get rid of the potential and interaction terms as shown in Eq. 4. In conjunction with reducing to a single $\hat{U}(\boldsymbol{\lambda})$ network, this simplified the circuit to only two components: a $\hat{U}(\boldsymbol{\lambda})$ network and the kinetic energy term as shown in Fig. 8.

$$-\frac{1}{2}\frac{d^2}{dx^2}f(x) = Ef(x) \tag{4}$$

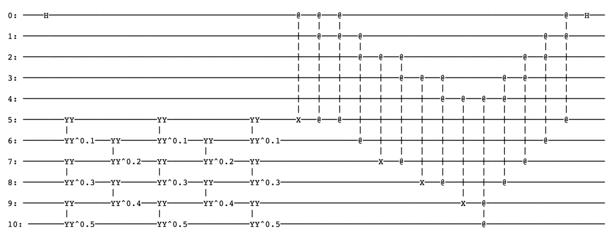


FIG. 8: Circuit for reduced Schrodinger's eguation

The goal of reducing the equation was to begin running the algorithm and to compare the results against a known solution. For this simplified problem, the solution should be $\psi(x) = \sqrt{2}\sin(\pi x)$ [4]. This would be the result of the $\hat{U}(\boldsymbol{\lambda})$ network when parametrized with the values of the converged solution. To gain a better understanding of the ansatz initially generated by the $\hat{U}(\boldsymbol{\lambda})$ network, the wave function was plotted with random parameters in Fig. 9.

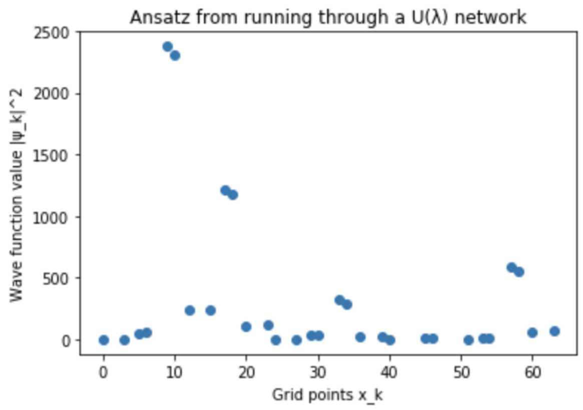
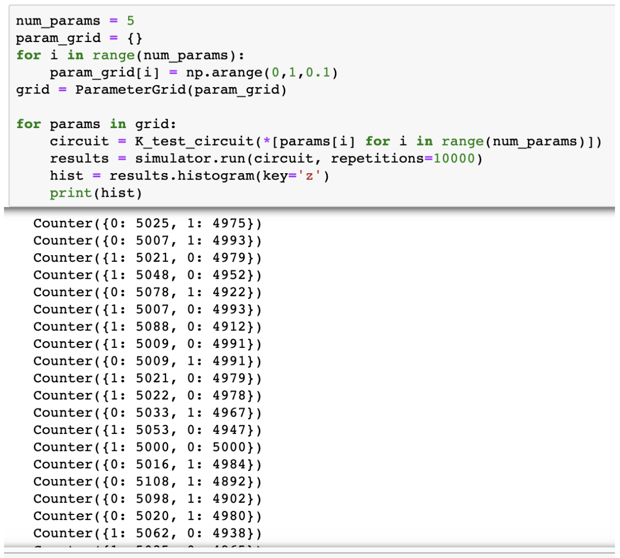


FIG. 9: Ansatz initialized from a $\hat{U}(\boldsymbol{\lambda})$ network with $n = 6$, $d = 5$, and parameters $[0.1, 0.2, 0.3, 0.4, 0.5]$

After optimizing the parameters, the graph in Fig. 9 should ideally resemble $\psi(x) = \sqrt{2}\sin(\pi x)$. However, upon attempting this through a simple grid search, it was found that any modification to the parameters resulted in trivial changes to the measured cost as shown in Fig. 10. There are a handful of reasons why this might be the case. One reason could be that the ancilla qubit on the top line of Fig. 8 does not encode the cost as currently believed. Another reason could be that the parameters for the $\hat{U}(\boldsymbol{\lambda})$ network are insufficient (in scale/number). These reasons will be investigated in the future.



```
num_params = 5
param_grid = {}
for i in range(num_params):
    param_grid[i] = np.arange(0,1,0.1)
grid = ParameterGrid(param_grid)

for params in grid:
    circuit = K_test_circuit(*[params[i] for i in range(num_params)])
    results = simulator.run(circuit, repetitions=10000)
    hist = results.histogram(key='z')
    print(hist)
```

```
  Counter({0: 5025, 1: 4975})
  Counter({0: 5007, 1: 4993})
  Counter({1: 5021, 0: 4979})
  Counter({1: 5048, 0: 4952})
  Counter({0: 5078, 1: 4922})
  Counter({1: 5007, 0: 4993})
  Counter({1: 5088, 0: 4912})
  Counter({1: 5009, 0: 4991})
  Counter({0: 5009, 1: 4991})
  Counter({1: 5021, 0: 4979})
  Counter({1: 5022, 0: 4978})
  Counter({0: 5033, 1: 4967})
  Counter({1: 5053, 0: 4947})
  Counter({1: 5000, 0: 5000})
  Counter({0: 5016, 1: 4984})
  Counter({0: 5108, 1: 4892})
  Counter({0: 5098, 1: 4902})
  Counter({0: 5020, 1: 4980})
  Counter({1: 5062, 0: 4938})
```

FIG. 10

IV. NEXT STEPS

For next steps, the immediate goal is to resolve the optimization issues and to obtain a working solution for the simplified equation. Once this is complete and all components from the paper are added, the aim is to adapt the circuit toward other PDEs. There are also opportunities to improve upon the ansatz, as well as more efficient methods of classical optimization rather than a simple grid search.

ACKNOWLEDGMENTS

I would like to thank Professor McMahon as well as my group members Leeseok, Rithesh, Anthony, and Thomas for assisting with my understanding of the paper and the many relevant background concepts.

[1] M. Lubasch, J. Joo, P. Moinier, M. Kiffner, and D. Jaksch, Variational quantum algorithms for nonlinear problems (2019), arXiv:1907.09032 [quant-ph].

[2] W. Contributors, Maximum cut, `https://en.wikipedia.org/wiki/Maximum_cut` (2019).

[3] JavaFXpert, The variational quantum eigensolver, `https://medium.com/qiskit/the-variational-quantum-eigensolver-43f7718c2747` (2019).

[4] Marojević, E. Göklü, and C. Lämmerzahl, Energy eigenfunctions of the 1d gross–pitaevskii equation, Computer Physics Communications **184**, 1920–1930 (2013).

[5] Variational-quantum-eigensolver (vqe), `https://grove-docs.readthedocs.io/en/latest/vqe.html`, accessed: 2019-12-22.