

Qiskit Advocate Mentorship Program - Final Showcase

<https://github.com/qiskit-advocate/qamp-fall-21/issues/28>

Variational Quantum Algorithms for Excited States

The Team

- Mentors

- Pauline Ollitrault
- Steve Wood

- Mentees

- Abhishek Jayachandran
- Vishnu Ajith



Aim

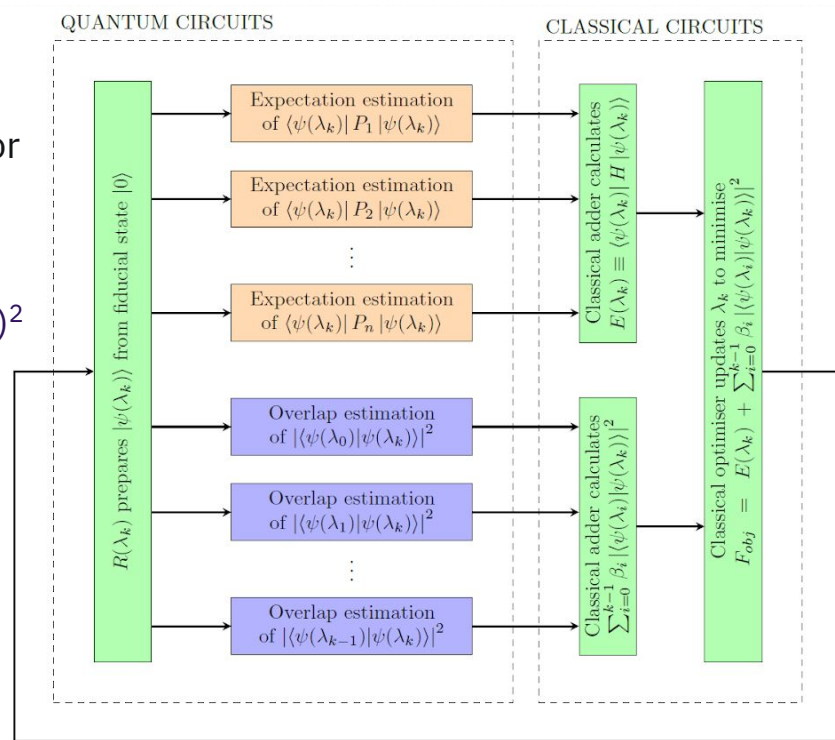
- Implement a generalized variational quantum method for predicting higher energy states of hamiltonians.
- Identify algorithms with reasonable performance for the task
- Add the `kstateVQE()` algorithm to qiskit terra that implements the `EigenSolver()` interface.
- Benchmark performance of the methods
- Write a tutorial explaining use of the method

VQD

- The VQD extends the VQE to calculate the k^{th} excited state by optimizing the parameters θ_k for the ansatz state $|\Psi(\theta_k)\rangle$ such that the cost function:

$$F(\theta_k) := \langle \Psi(\theta_k) | H | \Psi(\theta_k) \rangle + \sum_{i=0}^{k-1} \beta_i \langle \Psi(\theta_k) | \Psi(\theta_i) \rangle^2$$

is minimized.



Code Review

```
expect_op, expectation = self.construct_expectation(
    self._ansatz_params, operator, return_expectation=True
)

for state in range(step):

    prev_circ = self.ansatz.bind_parameters(prev_states[state])
    overlap_op.append(~CircuitStateFn(prev_circ)@CircuitStateFn(self.ansatz))

def energy_evaluation(parameters):
    parameter_sets = np.reshape(parameters, (-1, num_parameters))
    # Create dict associating each parameter with the lists of parameterization values for it
    param_bindings = dict(zip(self._ansatz_params, parameter_sets.transpose().tolist()))

    start_time = time()
    sampled_expect_op = self._circuit_sampler.convert(expect_op, params=param_bindings)
    mean = np.real(sampled_expect_op.eval())[0]

    for state in range(step):
        sampled_final_op = self._circuit_sampler.convert(overlap_op[state], params=param_bindings)
        cost = sampled_final_op.eval()
        mean += np.real(self.betas[state] * np.conj(cost) * cost)

    return mean
```

Outcomes

- Learned and understood in detail about variational quantum algorithms and in particular their implementation in Qiskit
- Learned the inner workings of qiskit enough to develop and integrate an algorithm to the terra codebase.
- Implemented the Variational Quantum Deflation Algorithm

```
from qiskit import QuantumCircuit, execute
from qiskit import Aer, IBMQ
from qiskit.providers.aer.noise import NoiseModel

# Choose a real device to simulate from IBMQ provider
provider = IBMQ.load_account()
backend = provider.get_backend('ibmq_igro')
coupling_map = backend.configuration().coupling_map

# Generate an Aer noise model for device
noise_model = NoiseModel.from_backend(backend)
basis_gates = noise_model.basis_gates

# Generate 3-qubit QMC state
num_qubits = 3
circuit = QuantumCircuit(3, 3)
circuit.h(0)
circuit.cx(0, 1)
circuit.cx(1, 2)
circuit.measure([0, 1, 2], [0, 1, 2])

# Perform noisy simulation
backend = Aer.get_backend('qasm_simulator')
job = execute(circuit, backend,
              config={
                  'noise_model': noise_model,
                  'basis_gates': basis_gates
              })
result = job.result()
print(result.get_counts(0))
```

Road Ahead

- Conduct unit tests on code and push to Qiskit Terra
- Write a tutorial explaining use of the `kstateVQE` and add it to the Qiskit tutorials

Demo