SUMMER PROJECT REPORT

EFFICIENCY COMPARISON AND IMPLEMENTATION OF VQE, VQD, AND P-VQD ALGORITHMS

UNDER GUIDANCE OF

DR. KUNTAL ROY

(Assistant Professor, EECS Department)
Indian Institute of Science Education and Research Bhopal

SUBMITTED BY

ABHAY SAXENA

(BS-MS, 2021-26)
Indian Institute of Science Education and Research Kolkata

Acknowledgement

I am writing to extend my utmost gratitude to Dr. Kuntal Roy for his invaluable contributions to my summer project report. Dr. Roy's profound expertise and invaluable suggestions were paramount to the successful completion of this project, and his constructive feedback has been of immense value. I am deeply grateful to him for providing me with the opportunity to delve into the realm of Quantum computation, as well as for his guidance and mentorship that encompassed a comprehensive understanding from fundamentals to practical applications.

Contents

Section 1

Abstract

1.1 Objective

This report aims to perform a comprehensive and in-depth comparison of three quantum algorithms, namely Variational Quantum Eigensolver (VQE), Variational Quantum Deflation (VQD), and Projected Variational Quantum Dynamics (p-VQD), all implemented in the Qiskit framework. The primary objective is to rigorously evaluate, contrast, and gain insights into the performance, efficiency, and applicability of each algorithm in solving complex quantum chemistry problems, particularly focusing on electronic structure calculations. The assessment encompasses various quantum hardware and simulators available in Qiskit, enabling a thorough investigation of the algorithms' capabilities under different computational settings. By achieving a detailed understanding of the strengths and limitations of VQE, VQD, and p-VQD, this study endeavours to assist researchers, quantum enthusiasts, and industry professionals in making informed decisions while choosing the most suitable quantum algorithms for their specific quantum chemistry simulations and future quantum computing endeavours.

1.2 Roadmap

In pursuit of the defined objective, the report proceeds as follows:

Introduction:
The report begins by providing an overview of the significance of quantum algorithms in quantum chemistry simulations. It outlines the importance of VQE, VQD, and p-VQD as promising candidates for tackling electronic structure calculations.

Algorithm Principles:
Under this section, the fundamental principles of each algorithm, namely VQE, VQD, and p-VQD, are explained in detail. This subheading sheds light on their underlying concepts, mathematical foundations, and workflow.

Implementation and Setup:
We describe the implementation of the algorithms in the Qiskit framework, along with the necessary setup to run them on quantum hardware and simulators. This section also covers the selection criteria for the quantum chemistry problems considered in the comparison.

Performance Evaluation:
To evaluate the algorithms, we assess their accuracy, convergence speed, and resource requirements. This subheading also examines their performance under the influence of noise and other error sources commonly found in current quantum hardware.

Complexity and Scalability:
We analyze the complexity of implementing each algorithm in Qiskit, providing insights into the programming efforts and expertise required. Additionally, the report examines the scalability of VQE, VQD, and p-VQD concerning the size and complexity of quantum chemistry problems they can address.

Results and Discussion:
The section presents the comparative analysis of VQE, VQD, and p-VQD based on the performance and scalability evaluations. We discuss the strengths and limitations of each algorithm, providing a clear understanding of their respective applicability and potential in real-world quantum chemistry simulations.

Conclusion:
In the concluding part, we summarise the key findings of the comparison and offer insights into the suitability of each algorithm for different quantum chemistry use cases. The report concludes with a perspective on the future prospects and potential advancements in the field of quantum algorithms for solving electronic structure problems.

Through this comprehensive assessment, researchers, quantum enthusiasts, and industry professionals will gain valuable insights into the selection of appropriate quantum algorithms for their specific quantum chemistry simulations.

Section 2

Introduction

2.1 Importance of VQE, VQD, and p-VQD

By utilising the power of quantum algorithms, quantum computing has ushered in a time of previously unimaginable opportunities and the potential to change entire sectors. Variational Quantum Eigensolver (VQE), Variational Quantum Deflation (VQD), and Projected Variational Quantum Dynamics (p-VQD) stand out among these ground-breaking algorithms for their crucial roles in resolving complex issues that conventional computers find difficult to handle effectively, i.e provide with the quantum advantage and reduce the run time complexity.

Variational Quantum Eigensolver (VQE):

Its capacity to determine the ground state energy of molecular systems makes it extremely valuable.
Based in the Chebyshev's inequality you would need to do $O(1/\epsilon^2)$ to get within ϵ accuracy for each circuit in your experiment.

1. Applicability to Quantum Chemistry: For the purpose of understanding molecular characteristics and reactions, electronic structure calculations in particular are critical, and VQE is specifically developed to solve these challenges. This can potentially advance fields like drug discovery, materials science, and catalysis.
2. Hardware Flexibility: VQE's flexibility allows it to be implemented on various quantum hardware platforms, ranging from noisy intermediate-scale quantum (NISQ) devices to error-corrected quantum computers. This adaptability makes it accessible for researchers working with different quantum technologies.
3. Resource Efficiency: Compared to traditional quantum algorithms, VQE requires fewer qubits and gates, making it more resource-efficient. This is particularly advantageous in the NISQ era, where qubit coherence and gate fidelity are limited, and provide reliable results even in imperfect quantum systems.
4. Applications Beyond Quantum Chemistry: While initially designed for quantum chemistry, VQE's variational approach has applications in other domains, such as optimization, machine learning, and

finance. This versatility makes it a valuable tool in interdisciplinary research.

5. Accessible for Researchers: VQE is widely supported in quantum programming libraries, such as Qiskit, Forest SDK, and Cirq, making it accessible to researchers and developers in the quantum computing community.

Ref:1.c

Variational Quantum Deflation (VQD):

VQE laid the foundation for VQD, which is a logical advancement in the development of quantum algorithms. A variational approach through the quantum algorithm VQD is used to find the k eigenvalues of the Hamiltonian H of a given system. The algorithm determines the excited state energies of generalized Hamiltonians using a better cost function.

By Adding the parameter of overlap function to the cost function of the algorithm, It makes use of previously calculated States of an atom, to reduce computation for calculation of higher states. In this way, finding the ground state through VQE, and building upon it through VQD for finding higher states would be more efficient as we would see further.

The importance of VQD is not limited to quantum chemistry alone. Its deflation technique has the potential to find applications in other domains, such as quantum optimization and quantum machine learning. This versatility positions VQD as an indispensable asset in the quantum algorithm toolbox, supporting further advancements in quantum computing applications.

Ref 2.b

Projected Variational Quantum Dynamics (p-VQD):

p-VQD represents a notable milestone in the pursuit of simulating dynamic phenomena in quantum systems.

1. Combination of VQE and VQD: p-VQD combines the strengths of both VQE and VQD algorithms, enhancing its capabilities for dynamic simulations and quantum chemistry applications.
2. Real-time Simulations: The algorithm's ability to perform real-time simulations sets it apart, enabling the study of time-dependent quantum processes with unprecedented accuracy, Hence

completing to the real time simulation of molecules and enable researchers to study chemical reactions and their kinetics.
3. Projected Hamiltonian: By using a projected Hamiltonian, p-VQD optimises the representation of the quantum system, focusing computational resources on essential aspects of the dynamics.
4. Catalysis Studies: In the field of catalysis, p-VQD offers insights into reaction mechanisms and catalytic pathways, accelerating the development of efficient catalysts.
5. Ongoing research and development in p-VQD will unlock new possibilities, bringing us closer to realising the full potential of quantum computing in dynamic phenomena simulations.

Ref 3.b

2.2 Variational Quantum Algorithms

VQA is a subclass of quantum algorithms that employs quantum circuits and variational techniques to solve optimization problems.
Utilizing a hybrid approach, VQA adjusts quantum parameters to minimize an objective function, generating trial states with quantum circuits.
Ref 5.a

- Adaptability and Versatility:
  VQA's capacity to adapt to various optimization tasks, including combinatorial optimization, machine learning, and quantum chemistry, is a significant advantage.
  Its adaptability makes VQA a valuable tool in multiple fields, surpassing classical techniques in quantum-efficient ways.

- Potential for Breakthroughs:
  VQA holds the potential to unlock revolutionary breakthroughs, impacting industries and driving scientific research as quantum computing technologies and hardware advance.
  Its efficient solution-finding capability is a crucial element in harnessing the full potential of quantum computing.

- Leading Candidate for Quantum Advantage:
  Among near-term quantum computers, VQAs stand at the forefront as the primary candidate for achieving quantum advantage.
  They have been extensively developed and applied in diverse fields, addressing tasks like finding ground states of molecules, simulating quantum system dynamics, and solving linear systems of equations.

- Common Structure and Adaptability:
  VQAs share a common structure, where a parameterized cost function encodes the task, evaluated by a quantum computer, and optimized by a classical optimizer.
  The adaptive nature of VQAs suits the constraints and imperfections of near-term quantum computers, demonstrating promise in practical applications.

- Challenges and Progress:
  Implementing VQAs for large-scale problems presents challenges related to trainability, accuracy, and efficiency.
  Ongoing efforts are focused on developing strategies to address these challenges, enhancing VQA performance for real-world scenarios.

- Optimization for NISQ Computers:
  VQAs provide an optimization-based or learning-based strategy to handle constraints imposed by NISQ computers, offering a single approach for quantum problem-solving.

Variational Method

- We know that and eigenvector $|\psi_i\rangle$ of a matrix A, does not vary under transformation upto eigenvalue. $A|\psi_i\rangle = \lambda_i|\psi_i\rangle$
- Eigenvalue of any Hermitian matrix has property $\lambda_i = \lambda_i^*$
- $H = \sum_{i=1}^{N} \lambda_i|\psi_i\rangle\langle\psi_i|$
- $\langle H\rangle_\psi = \langle\psi|H|\psi\rangle = \langle\psi|(\sum_{i=1}^{N}\lambda_i|\psi_i\rangle\langle\psi_i|)|\psi\rangle = \sum_{i=1}^{N}\lambda_i\langle\psi|\psi_i\rangle\langle\psi_i|\psi\rangle$
  $= \sum_{i=1}^{N} \lambda_i|\langle\psi|\psi_i\rangle|^2$
- So it is clear, $\lambda_{min} \leq \langle H\rangle_\psi = \langle\psi|H|\psi\rangle = \sum_{i=1}^{N} |\langle\psi|\psi_i\rangle|^2$
- This is Known as Variational method for $\langle H\rangle_{\psi_{min}} = \lambda_{min}$

  Ref 5.a

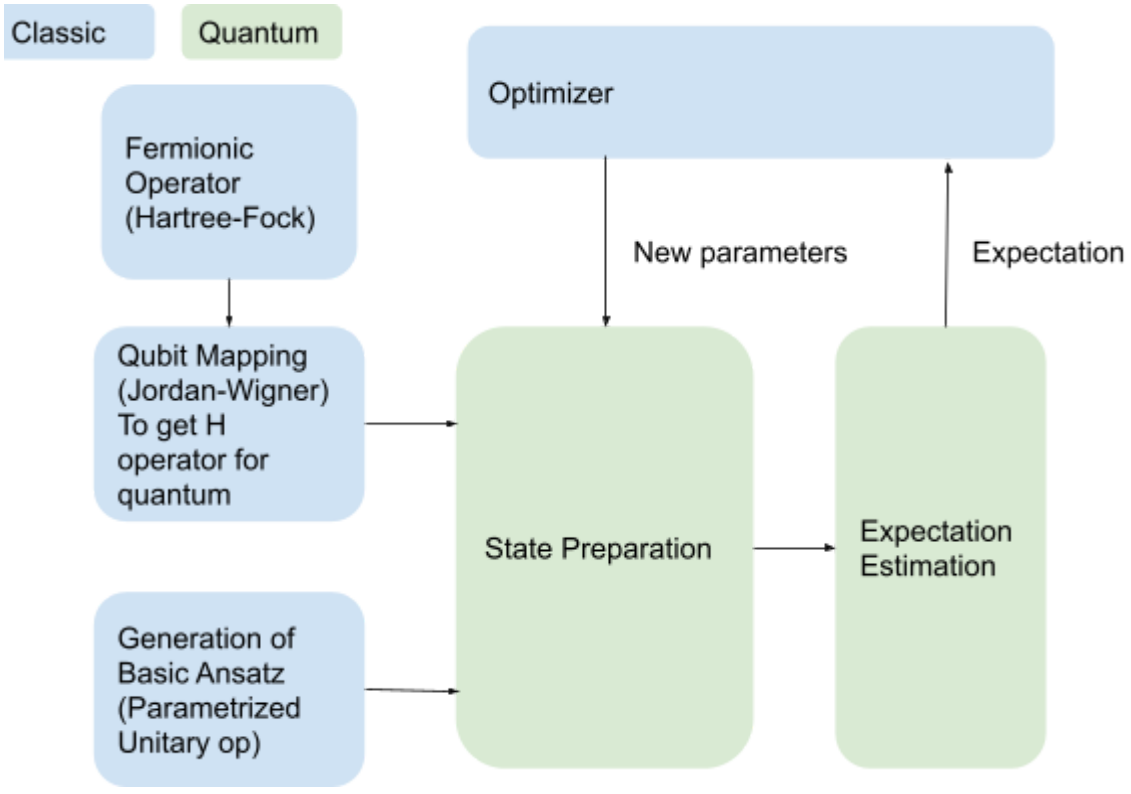
2.3 Algorithm Principles

Variational Quantum Eigensolver (VQE)

We define the VQE problem as follows:
definition:

$$E_{VQE} = min_{\theta}\langle\psi|U^{\dagger}(\{\theta\})\,\hat{H}\,U(\{\theta\})|\psi\rangle$$

Where U is our ansatz, i.e a parametrized unitary circuit which provides with our respective **ansatz wavefunction** upon applying certain set of parameters $\{\theta\}$ to ansatz circuit.
$|\psi\rangle$ is initial qubit register, generally initialized as $|0\rangle$.
$U(\theta)|\psi\rangle$ and $|\psi\rangle$ are both normalized wavefunction, with ansatz unitary introduced for sake of changing initial state to find eigenvector of hermitian H in order to find the eigenvalue.
Ref 1.a
Basic Structure:



Hamiltonian Construction and Representation: The first step involves defining the system for which the ground state is sought, represented by the Hamiltonian. The choice of basis functions and their representation significantly impacts the accuracy and computational cost of the VQE.

Encoding of Operators: Quantum computers measure observables in a Pauli basis. To use the Hamiltonian in VQE, it needs to be encoded into a set of Pauli operators. The transformation from fermionic operators to spin operators ensures that the fermionic antisymmetry is maintained.

Measurement Strategy and Grouping: To efficiently obtain expectation values, measurements are grouped and organized. Techniques such as commuting group identification and inference methods help minimize the number of measurements required.

Ansatz and State Preparation: The trial wavefunction, representing the ground state, is prepared using a parametrized quantum circuit called the ansatz. The choice of ansatz affects its expressibility, trainability, and circuit depth.

Parameter Optimization: The parameters of the ansatz are iteratively updated until convergence by sampling the Hamiltonian's expectation value. The choice of optimization method impacts the number of measurements and iterations required.

Error Mitigation: Quantum noise poses challenges for VQE on NISQ devices. Error mitigation techniques aim to reduce the impact of noise through post-processing of measurement data or trial wavefunctions.

The VQE pipeline is unique in its focus on finding eigenstates of quantum observables, distinguishing it from other variational quantum algorithms (VQAs) and Quantum Neural Networks (QNN) used for machine learning purposes.
Ref 1.a

Variational Quantum Deflation (VQD)

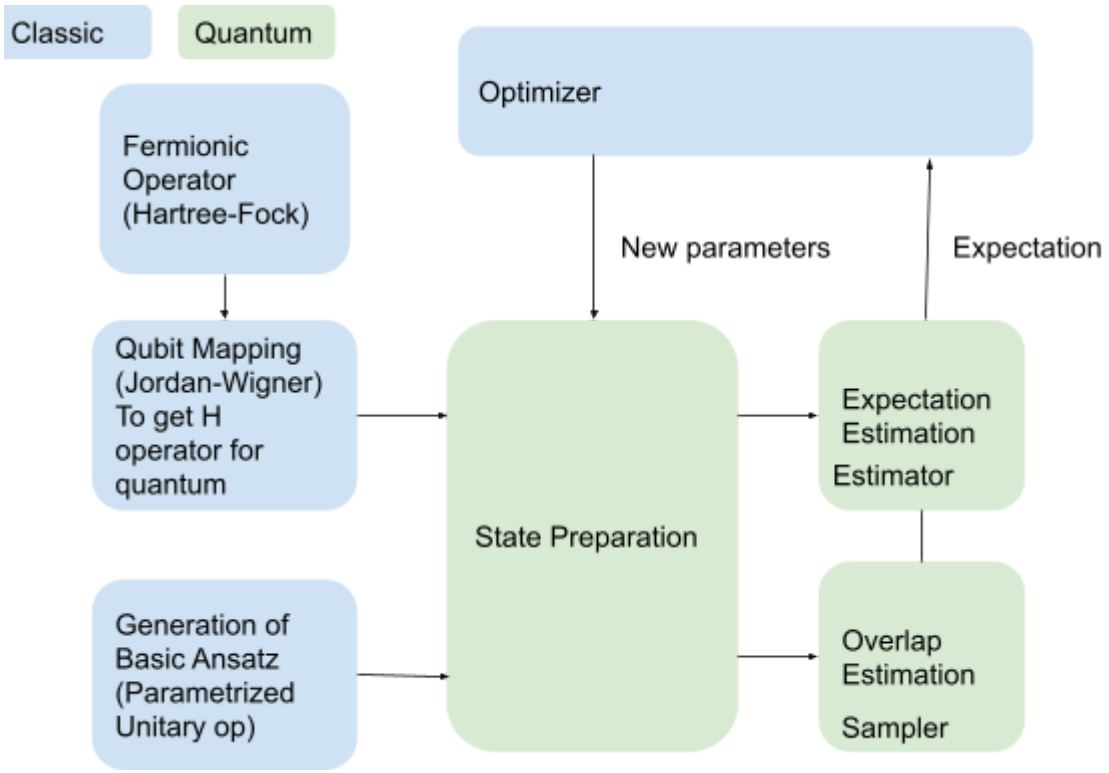
We define the VQE problem as follows:
definition:

$$E_{VQD} = min_{\theta} \quad (\langle \psi(\lambda_k)|U(\{\theta\})|\psi(\lambda_k)\rangle + \sum_{1=0}^{k-1} \beta_i |\langle \psi(\lambda_k)|\psi(\lambda_k)\rangle|^2)$$

Where U is our ansatz, i.e a parametrized unitary circuit which provides with our respective ansatz wavefunction upon applying certain set of parameters $\{\theta\}$ to ansatz circuit.
$|\psi\rangle$ is initial qubit register, generally initialized as $|0\rangle$.

$|\psi\rangle$ is normalized wavefunction with parameters λ_i.

And the second part is overlap weight with previously calculated states, which , by participation in cast function , makes the solution more efficient as all solution states are orthogonal.
Ref 2.a
Basic Structure:



Only thing new here from VQE is Overlap Estimation.
In VQD, the algorithm iteratively applies a deflation technique to find the next excited state of a quantum system. The deflation technique removes the contribution of the already-found lower-energy states, allowing the algorithm to target the next higher-energy state with improved efficiency and accuracy.

To apply the deflation technique correctly, VQD requires estimating the overlap between the current variational state and the already-found lower-energy states. This overlap estimation is essential for determining the correct updates to the quantum state during the optimization process.

Estimating the overlap involves calculating the inner product between the current trial state (obtained from the variational quantum algorithm) and the previously found eigenstates. This inner product measures the similarity or "overlap" between the states and provides information about

how much the current state is aligned with the known lower-energy states.

The accurate estimation of overlaps is critical for the success of the VQD algorithm. If the overlap estimation is not precise, it can lead to incorrect deflation of the lower-energy states, resulting in inaccurate excited state calculations.

To improve the efficiency and accuracy of overlap estimation, various techniques and methods can be employed, such as using state preparation techniques optimized for overlap calculations or employing quantum state tomography to reconstruct the trial state and accurately estimate overlaps with known eigenstates.
Ref 2.a

Projected Variational Quantum Dynamics

Time-Dependent Simulations: p-VQD is specifically designed to tackle time-dependent processes in quantum systems. It can simulate how quantum states evolve over time, making it suitable for studying dynamic phenomena.

Projected Hamiltonian: In p-VQD, a projected Hamiltonian is used to represent the quantum system. The algorithm focuses computational resources on crucial parts of the system, optimizing the representation for efficient simulations.

Combining VQE and VQD: By combining elements from VQE and VQD, p-VQD achieves better performance in simulating dynamic quantum systems compared to using either algorithm independently.

Efficiency and Accuracy: p-VQD aims to strike a balance between computational efficiency and simulation accuracy. By utilizing the strengths of VQE and VQD, it provides a powerful tool for time-dependent quantum simulations.
Ref 2.b
So, basically defining the p-VQD problem as :

Definition:

A time operator for small step

$$\delta t \in \mathbf{R} \text{ is } e^{-i\hat{H}\delta t}$$

$|\psi_{w(t)}\rangle$:let this describe quantum state of the system for time t.

p-VQD problem for dω and dt is thus-

$$\arg\max_{dw\in\mathbb{R}^p} |\langle\phi(\delta t)|\psi_{w+dw}\rangle|^2$$

Optimally best next step to find the parameters is to minimise step-infidelity defined by:

$$L(dw,\delta t) = \frac{1-|\langle 0|C^\dagger(w)e^{i\hat{H}\delta t}C(w+dw)|0\rangle|^2}{\delta t^2}$$

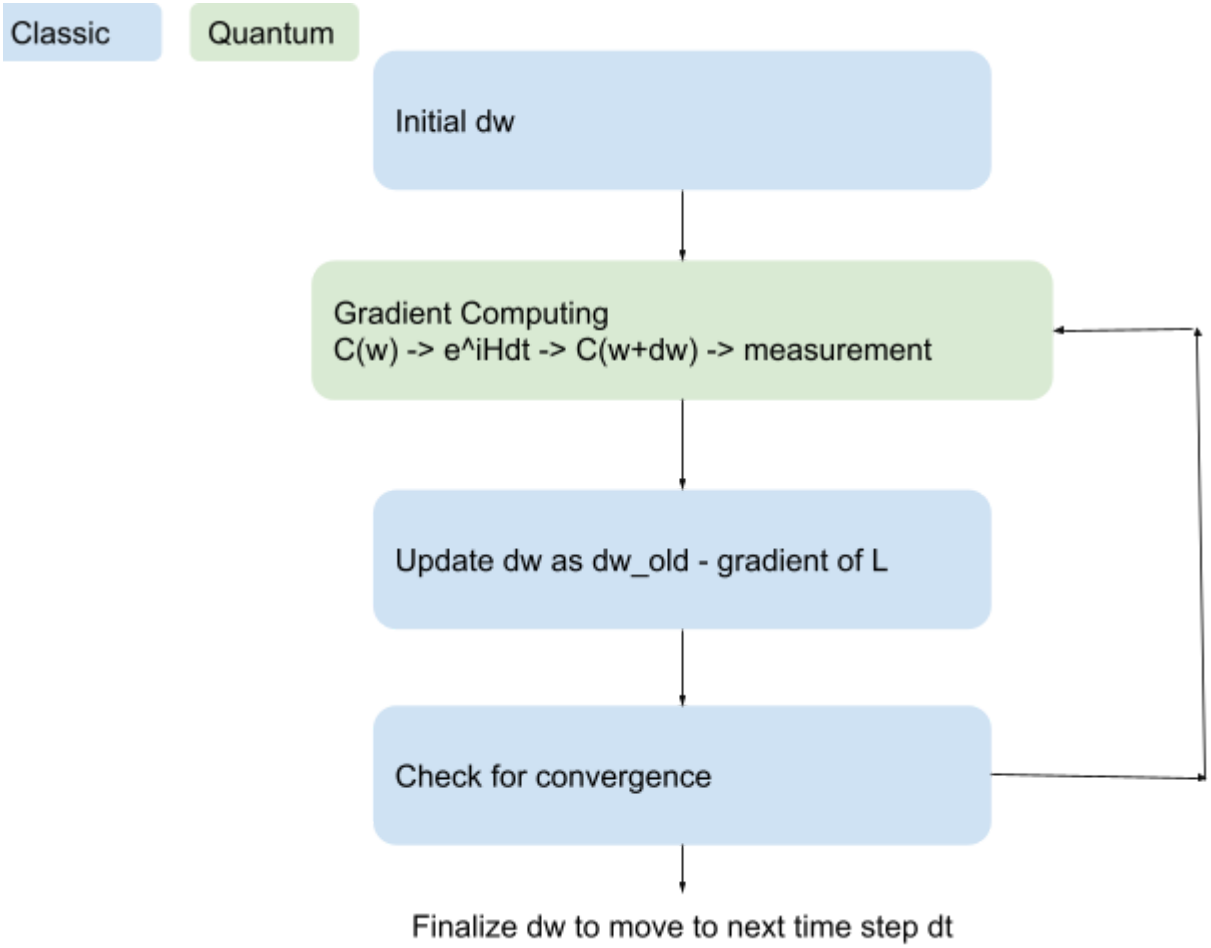
Where,

$$|\psi_w\rangle = C(w)|0\rangle$$

Basically, this method projects the time step onto a variational form (ansatz) and uses a Trotter formula (given by the evolution argument) to calculate the next state for each timestep. By applying a traditional optimization procedure, the projection is found by maximising the fidelity of the Trotter-evolved state and the ansatz.

Ref 2.a

Basic Structure:



Ref 2.a

2.4 Implementation

```
from qiskit_ibm_runtime import QiskitRuntimeService
service = QiskitRuntimeService(
    channel="ibm_quantum", token="XXXXXXX")
backend = service.get_backend("ibmq_quito")
```

Is required in all the following implementations to connect to qiskit cloud computing quantum computers (in our case ibm quito).
Ref 4.j

Defining Hamiltonian

```
import qiskit_nature
from qiskit_nature.second_q.formats.molecule_info import MoleculeInfo
from qiskit_nature.second_q.drivers import Psi4Driver
from qiskit_nature.second_q.mappers import ParityMapper
qiskit_nature.settings.use_pauli_sum_op = False
```

```
def qubit_operator(dist):
    mol = MoleculeInfo(
        # Coordinates in Angstrom
        symbols=["H", "H"],
        coords=([0.0, 0.0, 0.0], [dist, 0.0, 0.0]),
        multiplicity=1,
        charge=0,
    )

    problem = Psi4Driver.from_molecule(mol).run()

    spatial_orbitals_count = problem.num_spatial_orbitals
    particle_count = problem.num_particles

    mapper = ParityMapper(num_particles=particle_count)  # Set Mapper

    hamiltonian = mapper.map(problem.second_q_ops()[0])  # Set
Hamiltonian

    return hamiltonian, particle_count, spatial_orbitals_count,
problem, mapper
```

So here we see that we define molecule data using MoleculeInfo and convert it to Fermionic Operator using Psi4, an opensource suite designed for efficient, high-accuracy simulations of molecular properties.

It is then mapped to qubits using parity mapper, responsible for The Parity fermion-to-qubit mapping.

Problem is Fermionic version, and hamiltonian is the mapped version(variational form).

For Hydrogen, at optimal distance of 0.735 armstrongs, we obtain,

```
hamiltonian = SparsePauliOp.from_list(
    [
        ("II", -1.052373245772859),
        ("IZ", 0.39793742484318045),
        ("ZI", -0.39793742484318045),
        ("ZZ", -0.01128010425623538),
        ("XX", 0.18093119978423156),
    ]
```

```
)
```

Which returns a matrix,

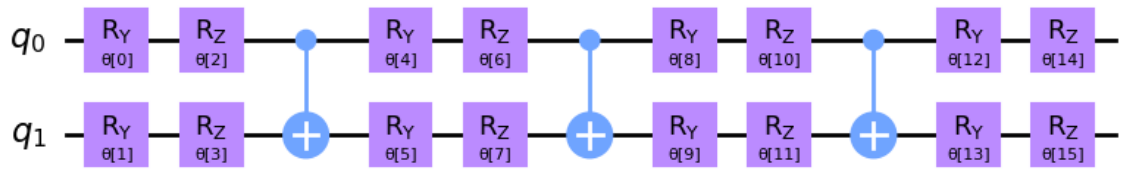
(-1.064+0j)	0j	0j	(0.181+0j)
0j	(-1.837+0j)	(0.181+0j)	0j
0j	(0.181+0j)	(-0.245+0j)	0j
(0.181+0j)	0j	0j	(-1.064+0j)

Ref 4.k

Ansatz

```
from qiskit.circuit.library import EfficientSU2
#ansatz = TwoLocal(rotation_blocks=['ry', 'rz'],
entanglement_blocks='cz')
ansatz = EfficientSU2(hamiltonian.num_qubits)
ansatz.decompose().draw('mpl')
```

Here, we create an ansatz from previously set library circuit consisting of layers of single qubit operations spanned by SU(2) and CX entanglements. SU(2) or special unitary groups for degree 2 can be pauli Y and Z gates.



[Output circuit]

Ref 4.k

VQE (customised implementation)

Cost function

```
def cost_func(params, ansatz, hamiltonian, estimator):
    energy = estimator.run(ansatz, hamiltonian,
                           parameter_values=params).result().values[0]
    return energy
```

Estimator is an object of class Estimator from qiskit_ibm_runtime.Estimator

Quantum circuit and observable expectation values are estimated via the Qiskit Runtime Estimator basic service.

Ref 4.m

Initial parameters of ansatz x0:

```
import numpy as np
x0 = 2 * np.pi * np.random.random(num_params)
#hartree fork can be used to reduce no. of iterations needed
```

When we will start optimizing the circuit we will have some initial parameters, to start from , we take them as array of 0's.

Callback

```
def callback_builder(ansatz, hamiltonian, estimator, callbackd):
    def callback( nfev, parameters, value, stepsize, accepted):
        #number of function evaluations, the parameters, the function
value, the stepsize, whether the step was accepted.
        callbackd["iters"] += 1
        # Set the prev_vector to the latest one
        callbackd["prev_vector"] = parameters
        # Compute the value of the cost function at the current vector
        callbackd["cost_history"].append(
            value
        )
        # Grab the current time
        # Find the total time of the execute (after the 1st iteration)
        job = service.jobs(backend_name="ibmq_quito")
        callbackd['times'].append(job.metrics()['usage']['seconds'])
        callbackd['step_size'].append(stepsize)
        # Print to screen on single line
        print(
            "Iters. done: {} [Avg. time per iter:
{}]".format(callback_dict["iters"], time_str),
            end="\r",
            flush=True,
        )

    return callback
```

On every iteration of SPSA, it returns some parameters to callback which can be stored in a dictionary for future references. The dictionary is :-

```

callbackd = {
    'n_iters':none,
```

```
    'pvector':[],
    'cost_old':[],
    'final_time':[],
    'step_size':[],
    'cc':[]#concurrency
}
```

job.metrics()['usage']['seconds'] for returns the usage time of an algorithm, Where job is job = service.jobs(backend_name="ibmq_quito"), i.e last run job in our computer used.

Ref 4.l

Starting a session:

```
from qiskit_ibm_runtime import Session
from qiskit.algorithms.optimizers import SPSA
with Session(backend=backend):
    estimator = Estimator(options={"shots": int(1e4)})
    def costfspsa(params):
        return cost_func(params, ansatz, hamiltonian, estimator)

    callback = callback_builder(ansatz, hamiltonian, estimator,
callbackd)
    spsa = SPSA(callback=callback, maxiter=125)
    spsa.minimize(
        costfspsa,
        x0
    )
```

Session: You can organize a set of iterative calls to the quantum computer into a Qiskit Runtime session. When the first job within a session begins, the session has begun. The scheduler gives subsequent jobs in the session a higher priority. To save needless overhead, data used during a session, such as transpiled circuits, is also cached.

Ref 4.j

SPSA: Simultaneous perturbation stochastic approximation is an algorithmic method for optimizing systems with multiple unknown parameters. It is a type of stochastic approximation algorithm. Stochastic approximation methods are a family of iterative methods typically used for root-finding problems or for optimization problems instead of using gradients like SLSQP and COBYLA (scipy methods for ideal no noise systems).

Costfspsa: is the generated cost function for spsa,using parameters, spsa provides along with other parameters required to run estimator in cost_func.

Shots: Estimator's number of shots tell how many times an algorithm is run to get a probability distribution of results. 1e4 i.e 10000 shots is ideal for a noisy real quantum computer.
Ref 4.l
Adding Concurrency:

```
from qiskit.quantum_info import Statevector, concurrence
for j in callbackd['pvector']:
    bound_ansatz = ansatz.bind_parameters(j)
    i = Statevector.from_instruction(bound_ansatz)
    callbackd['cc'].append(concurrence(i))
```

We shall discuss the use of concurrency in innovation section of this report.

Output for basic custom VQE implementation is shared under Results section 3.1
Ref 4.i

VQD

```
from qiskit.primitives import Sampler, Estimator
from qiskit.algorithms.state_fidelities import ComputeUncompute

estimator = Estimator()
sampler = Sampler()
fidelity = ComputeUncompute(sampler)
```

ComputeUnCompute uses the sampler primitive to calculate the state fidelity of two quantum circuits following the compute-uncompute method
Ref 4.n
fedility (state overlap): $|\langle\psi(x)|\phi(y)\rangle|^2$

The initial release of Qiskit Runtime includes two primitives:

Sampler: Generates quasi-probability distribution from input circuits.

Estimator: Calculates expectation values from input circuits and observables.

session:

```
from qiskit.algorithms.eigensolvers import VQD



vqd = VQD(estimator, fidelity, ansatz, optimizer, k=k, betas=betas,
callback=callback)
result = vqd.compute_eigenvalues(operator = hamiltonian)
vqd_values = result.eigenvalues
```

Where callback is

```
counts = []
values = []
steps = []

def callback(eval_count, params, value, meta, step):
    counts.append(eval_count)
    values.append(value)
    steps.append(step)
```

And

```
k = 4
betas = [33, 33, 33, 33]
```

Here k is number of states to calculate eigenvalue for, and betas are respective β_i in

$$E_{VQD} = min_{\theta} \quad (\langle\psi(\lambda_k)|U(\{\theta\})|\psi(\lambda_k)\rangle + \sum_{1=0}^{k-1} \beta_i |\langle\psi(\lambda_k)|\psi(\lambda_k)\rangle|^2)$$

Actual solutions for comparison

```
from qiskit.algorithms.eigensolvers import NumPyEigensolver



exact_solver = NumPyEigensolver(k=4)
exact_result = exact_solver.compute_eigenvalues(hamiltonian)
ref_values = exact_result.eigenvalues
```

NumpyEigenSolver returns correct solutions using classical computer to compare values to as got from VQD.

Ref 4.c

P-VQD

Almost same as all terms explained in VQD, p-VQD uses

```
sampler = Sampler()
fidelity = ComputeUncompute(sampler)
estimator = Estimator()
hamiltonian = 0.1 * SparsePauliOp(["ZZ", "IX", "XI"])
observable = Pauli("ZZ")
ansatz = EfficientSU2(2, reps=1)
initial_parameters = np.zeros(ansatz.num_parameters)
```

Where we have taken easier hamiltonian which serves different purpose of time simulation.

Optimizer

```
time = 1
optimizer = L_BFGS_B()
```

A differentiable scalar function f's value is minimized via the Limited-memory Broyden-Fletcher-Goldfarb-Shanno Bound (L-BFGS-B) algorithm.
This optimizer uses a quasi-Newton technique, which means that while attempting to determine f's minimal value, it does not require f's Hessian (the matrix of f's second derivatives), unlike Newton's method.

We setup algorithm from preexisting qiskit.algorithms.time_evolver.PVQD as-

```
# setup the algorithm
pvqd = PVQD(
    fidelity,
    ansatz,
    initial_parameters,
    estimator,
    num_timesteps=100,
    optimizer=optimizer,
)
```

Ref 4.o

And run the time evolution as:

```
# specify the evolution problem
```

```
problem = TimeEvolutionProblem(
    hamiltonian, time, aux_operators=[hamiltonian, observable]
)

# and evolve!
result = pvqd.evolve(problem)
```

Ref 4.p

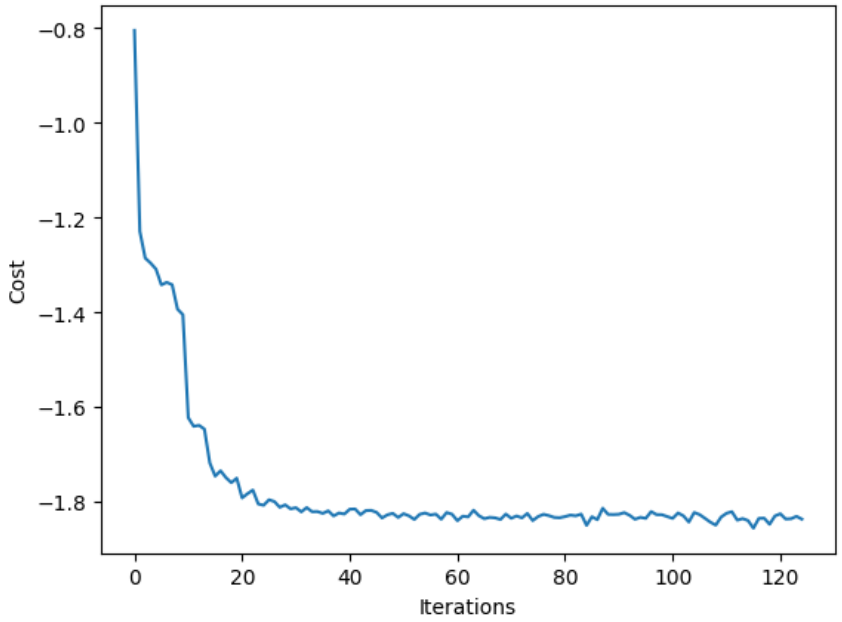
Section 3

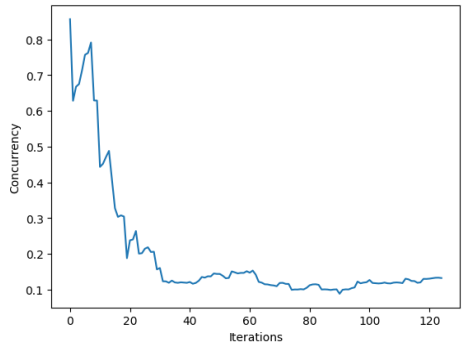
Results

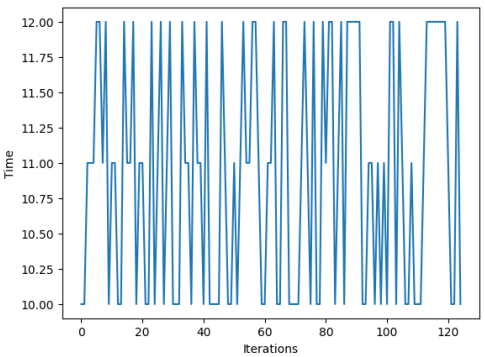
3.1 Plots and discussion

Basic Implimentation's output

The basic **custom VQE** implementation discussed earlier had the following output







We could see how VQE's cost function is minimised by SPSA, by changing the parameters of ansatz circuit . The curve approaches near least value much faster than our simulators, shared ahead, not because they are inefficient, but because, the number of shots is higher here (10^4).

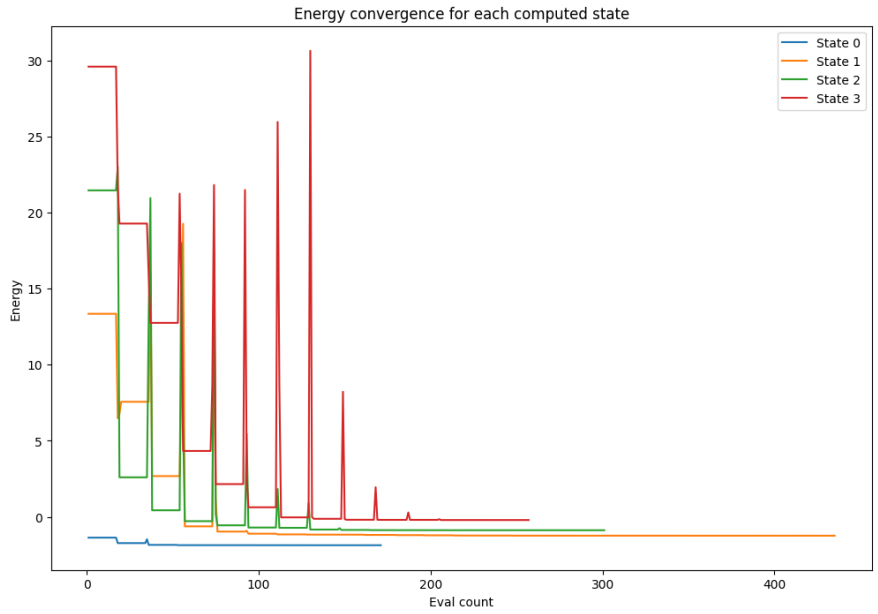
Just as Expected, Cost function reduces and comes very close to original eigenvalue of 0.98

Time taken by each Iteration is also plotted, as returned under jobs.matrics() by **ibmq_quito,** plotted as time vs iterations graph.

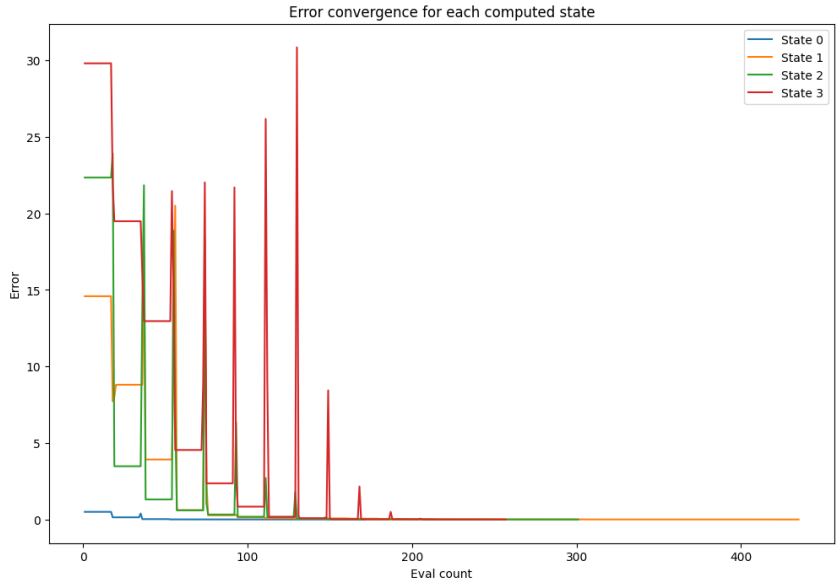
Concurrency is also mentioned here and is an entanglement monotone, thus is useful to compare efficiency of each step in terms of quantum advantage it provides since, each step

Note: Rest of the processing shall not be run on quantum computer as they are equally well represented by Fake Provider Noise models on local simulation, and queue length along with time taken for quantum computers to initialise, with the api is unfeasible for running the processes.

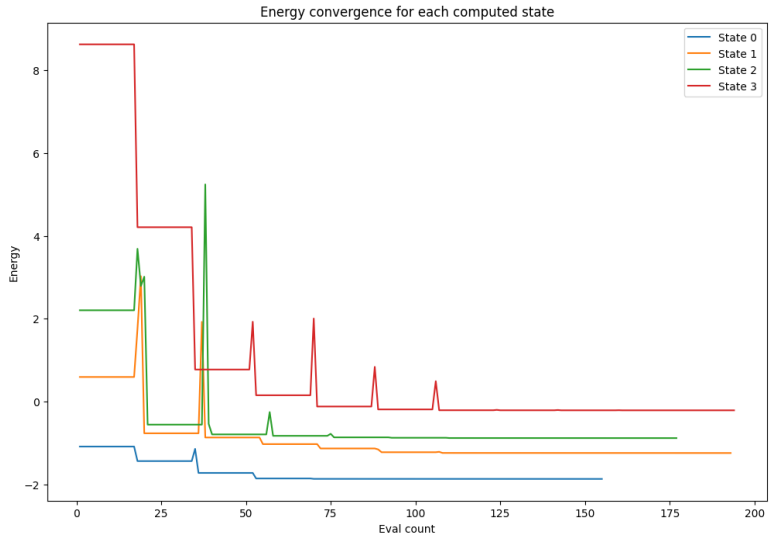
Vqd output:(basic implementation)
Similarly, we can see the output for basic implementation of VQD as follows:



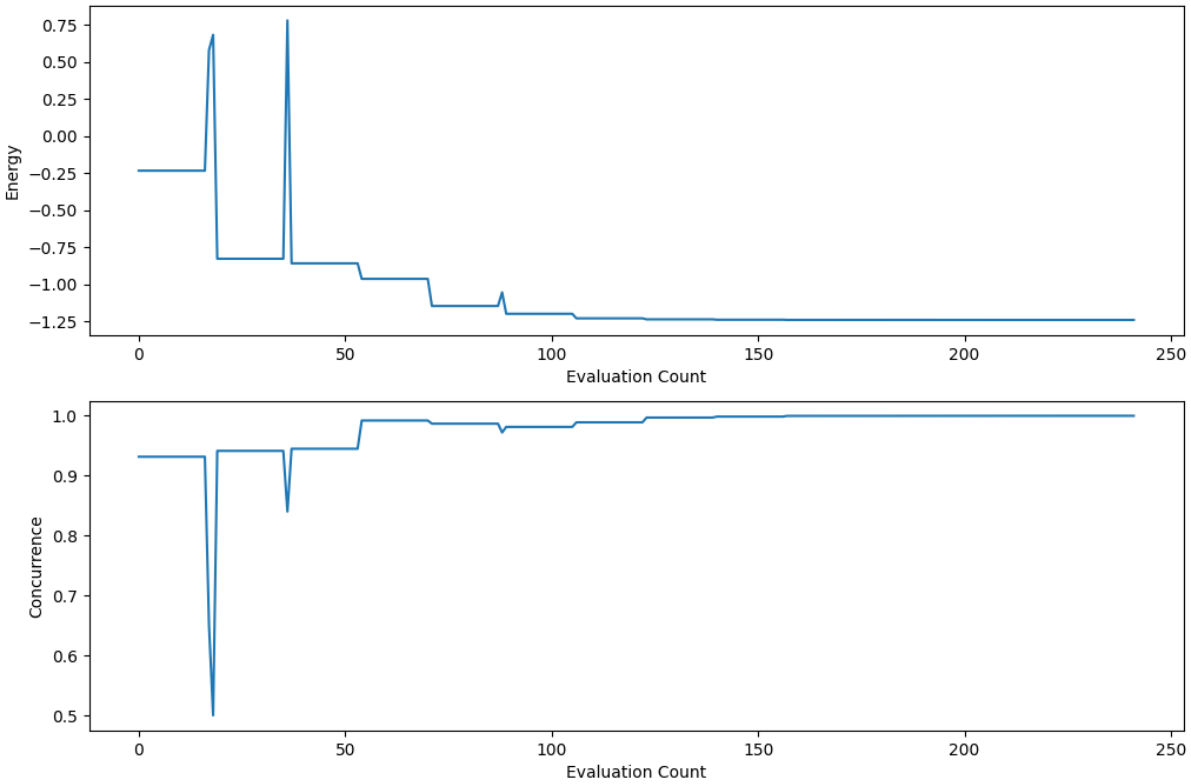
We can also plot the difference of each state to it's expected true value as per Numpy EigenSolver as-



i.e a plot of error.
We can see, Error seems to be higher, for these state, but when the calculations are run with lower sample rate, so is not the case:



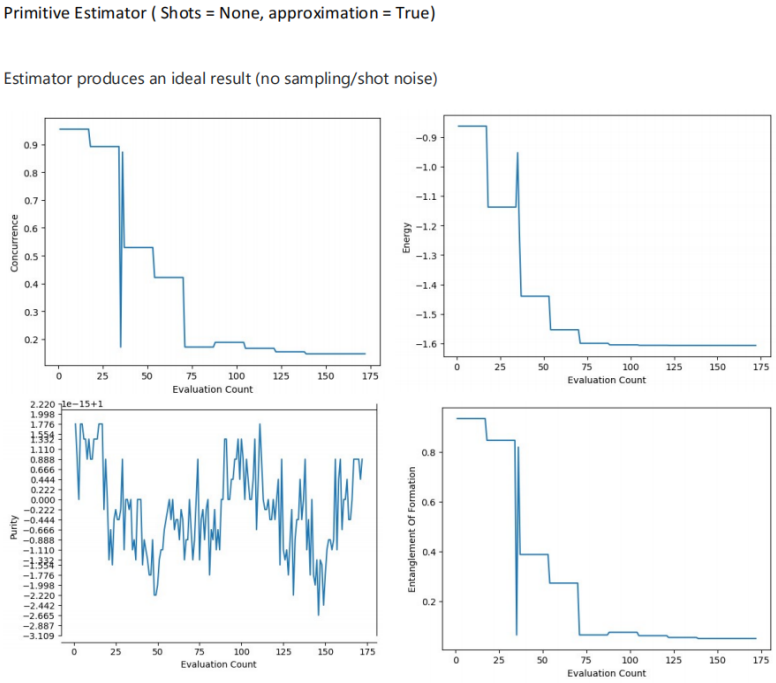
We can see, the amount of initial deviation is dependent on coefficient betas of Sampling function, i.e overlap, in VQD.
However, β must not be smaller than the next state to be solved, to launch the sampler we above, or else, it will be stuck to a lower energy eigenvalue solution.



Concurrence of second State in VQD

If, we notice, the bottom most state 0 , is actually a VQE simulation itself, for ideal simulator, as this graphs is for the same case. Basically, state 0 in VQD is calculated using VQE only, and rest states are build upon it, taking advantage of previous calculations, to comparatively, reduce computation.

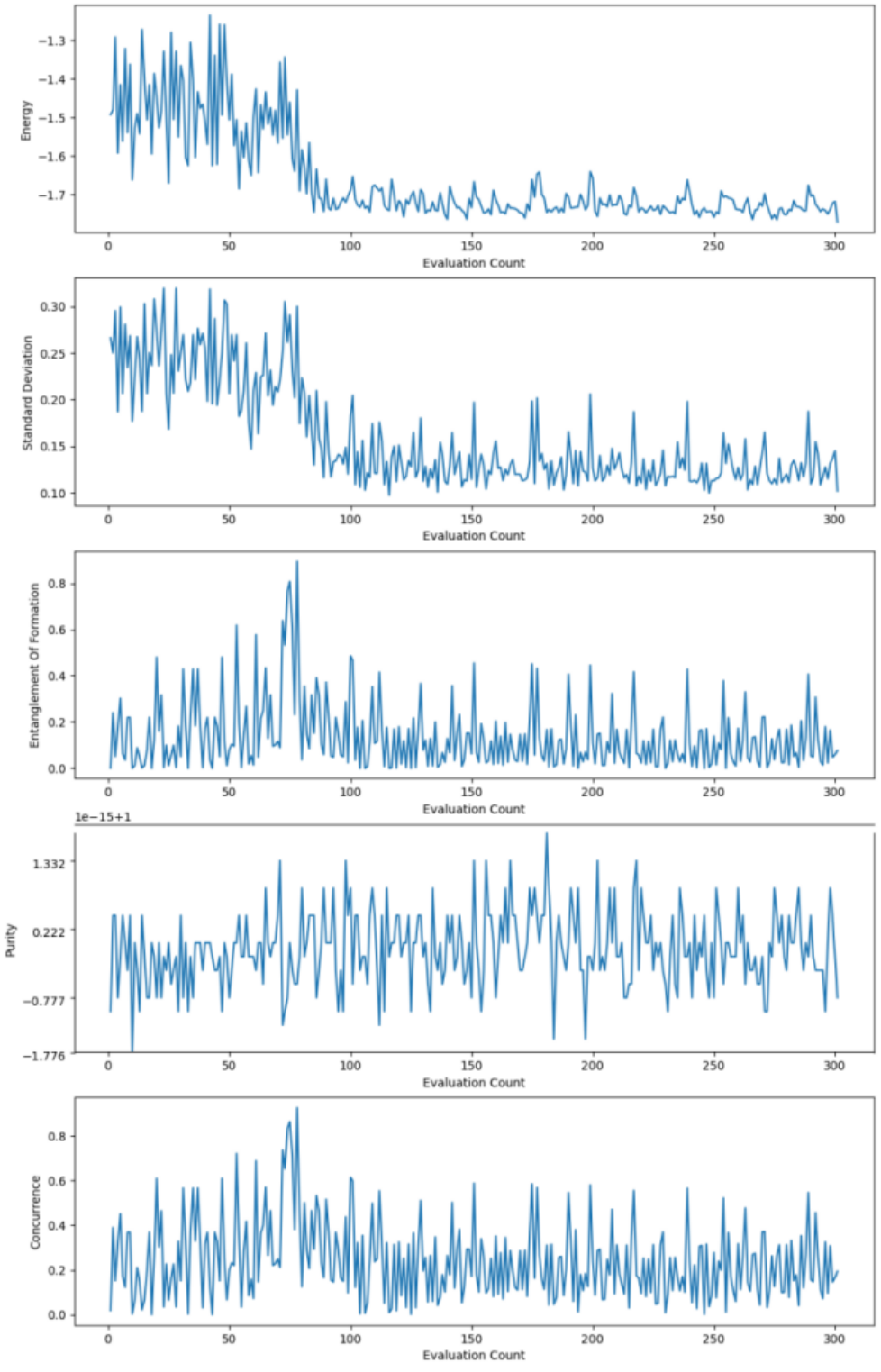
Various cases for VQE:



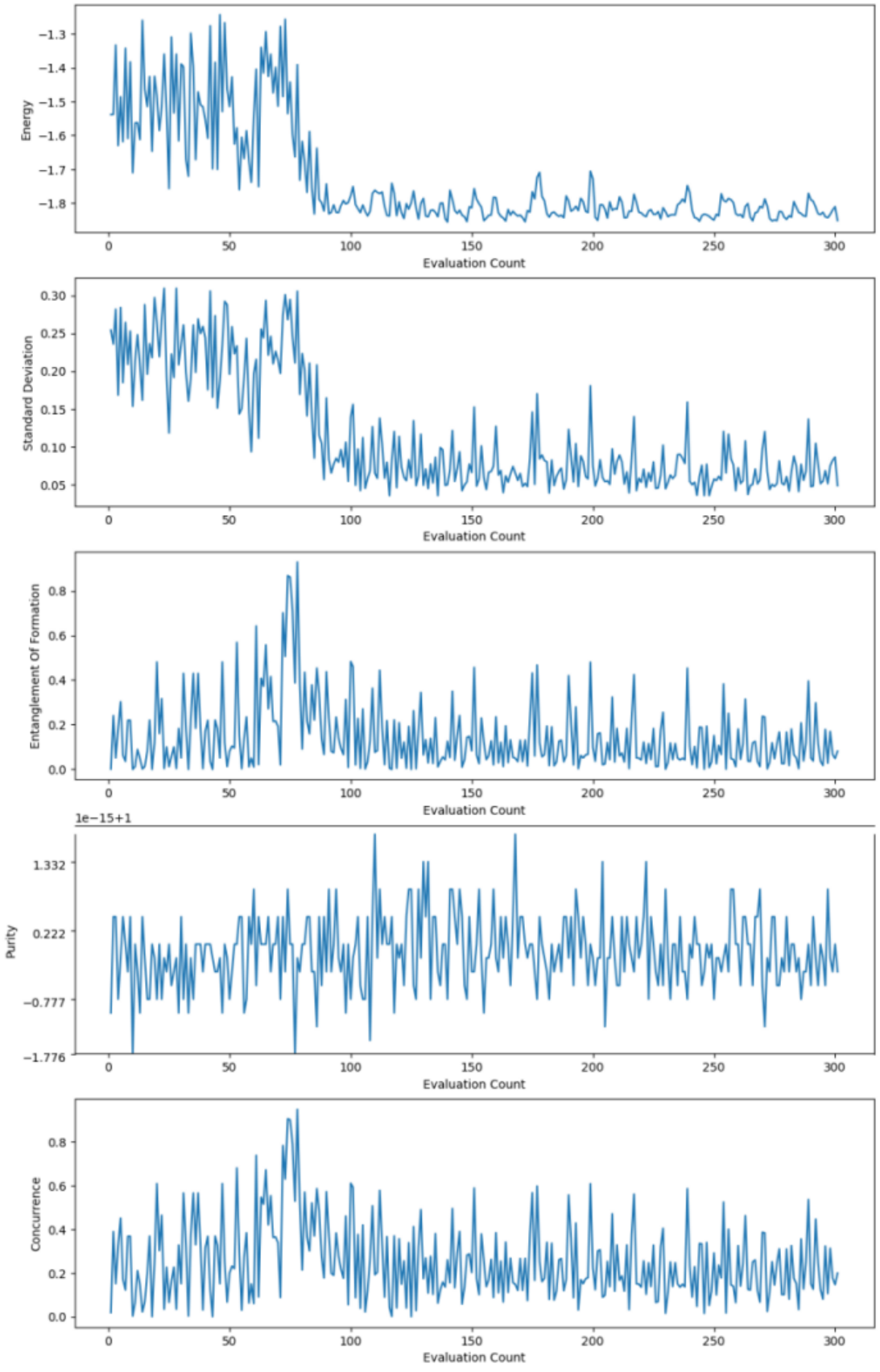
We can see How Concurrence reduces as the energy reduces. This signifies the role played by entanglement in the way that it shows the working of the algorithm itself. If the entanglement is low, then amount of quantum advantage is low. Hence it is a nice way to quantify advantage. We also see it's corelation with entanglement of formation, which is same for 2 qubit systems.

Noisy Estimator: shots = 1024

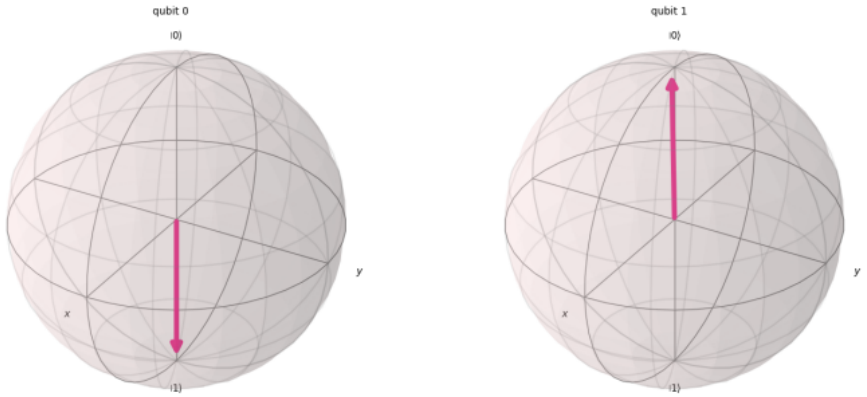
- (Standard Deviation is also added as solution is not ideal, and metadata is thus provided)
- expectation values with sampling error and also simulated error from quantum computer.
- VQE on Aer qasm simulator(with noise): -1.85160
- Delta from reference energy value is 0.00567



Noiseless Estimator, shots = 1024



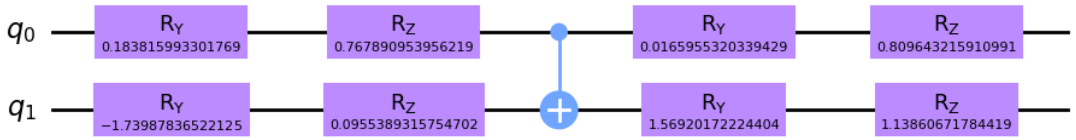
Final state:

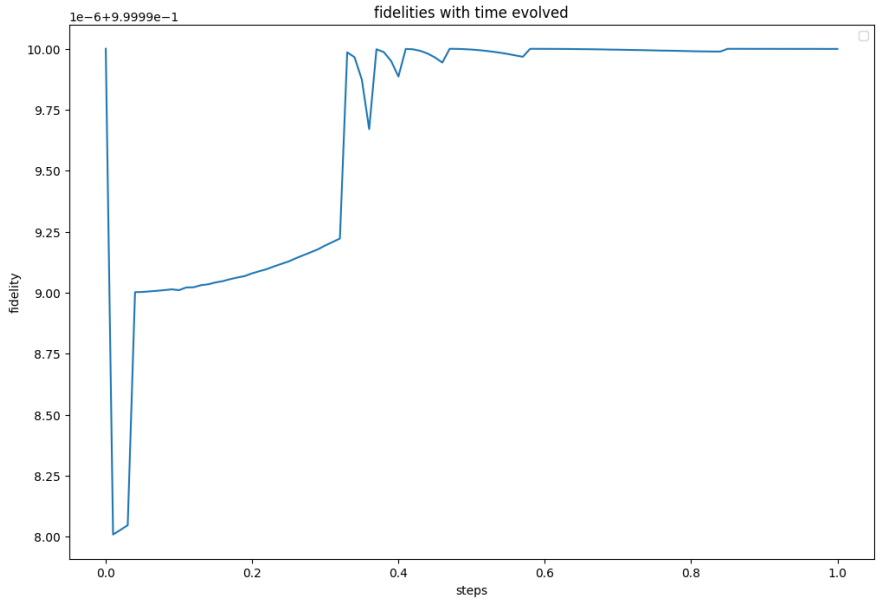


Upon the completion of VQE, we see the final state vector(ansatz eigenvector) to find the hamiltonian's EigenValue.

pVQD

Evolved state ansatz:





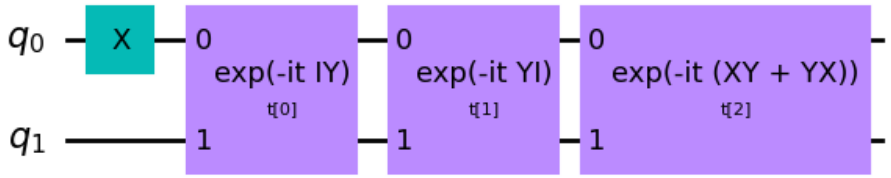
Fidelities as we may remember stand for state overlaps.
Remember we are trying to maximize fidelity, as we minimize infidelity:

$$L(dw, \delta t) = \frac{1 - |\langle 0|C^\dagger(w)e^{i\hat{H}\delta t}C(w+dw)|0\rangle|^2}{\delta t^2}$$

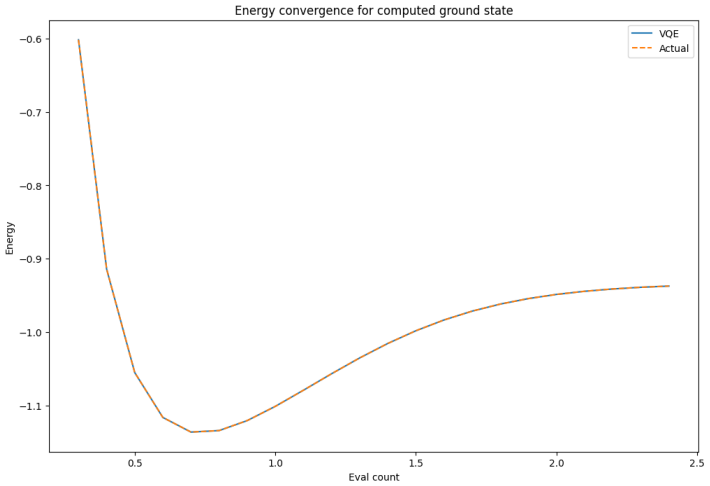
Molecule Simulations

UCCSD ansatz for VQE Molecule simulation

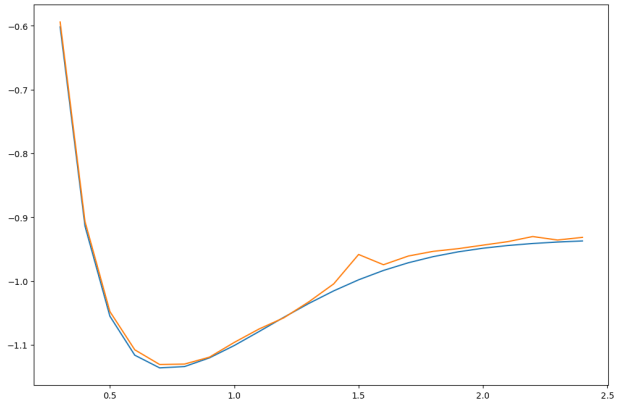
UCCSD (Unitary Coupled Cluster Singles and Doubles): is a quantum algorithm used to solve electronic structure problems in molecular systems. It employs a parameterized quantum circuit (ansatz) to represent the trial wavefunction. By applying "singles" and "doubles" excitations, it captures electronic correlations. The algorithm iteratively optimizes the ansatz parameters to minimize the molecular Hamiltonian's energy, providing an estimate of the ground state energy. UCCSD belongs to the Variational Quantum Eigensolver (VQE) family and holds potential for computational chemistry and materials science applications, but challenges with quantum noise and hardware limitations persist. Research aims to enhance the efficiency and accuracy of VQE algorithms on near-term quantum devices.



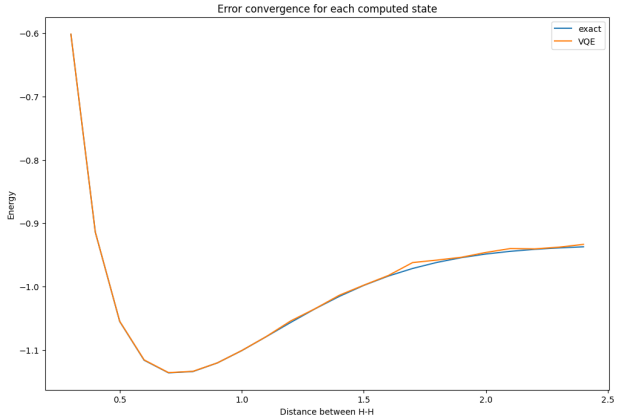
t[0]	-7.450580585082699e-09
t[1]	-2.8041880121786598e-08
t[2]	-0.6719440148165818



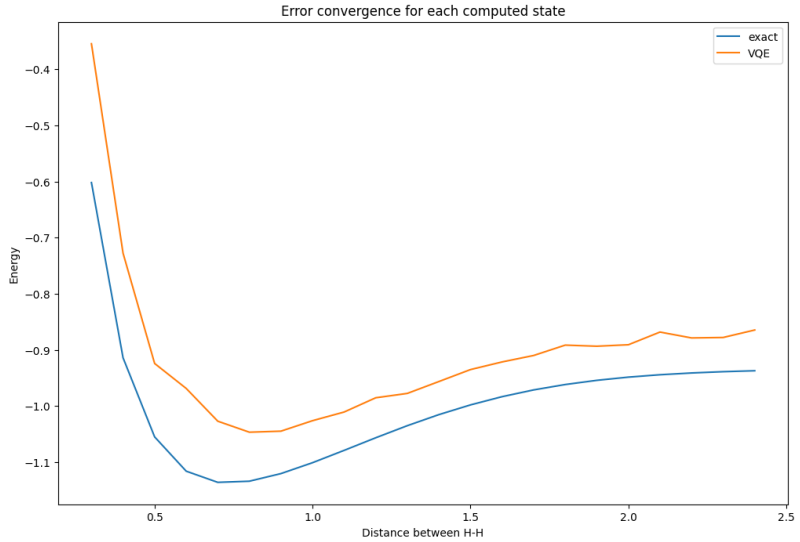
VQE on qiskit.primitive.Estimator



VQE on AerSimulator shots:2048

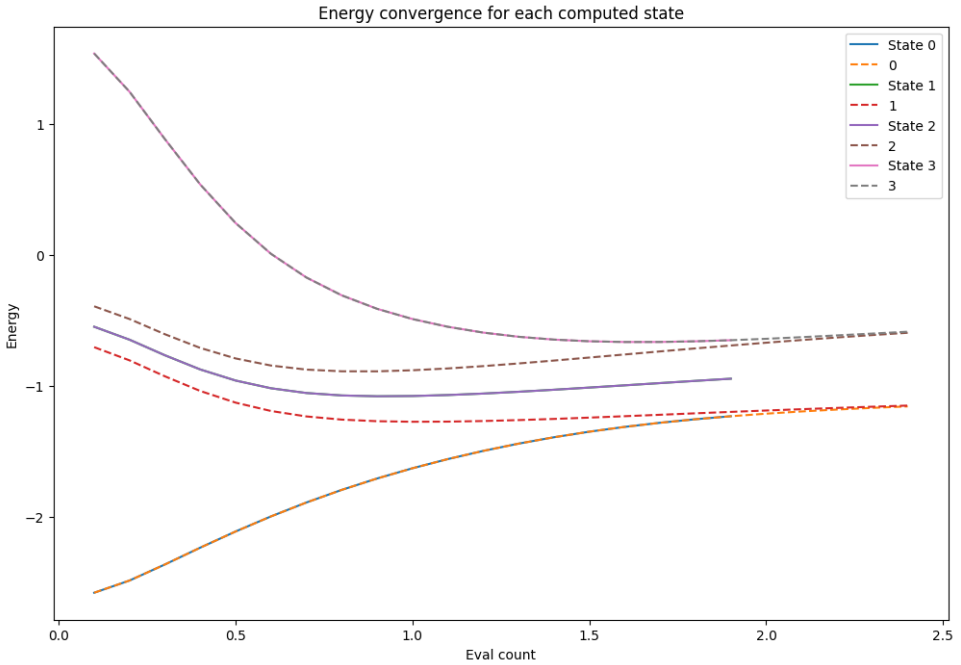


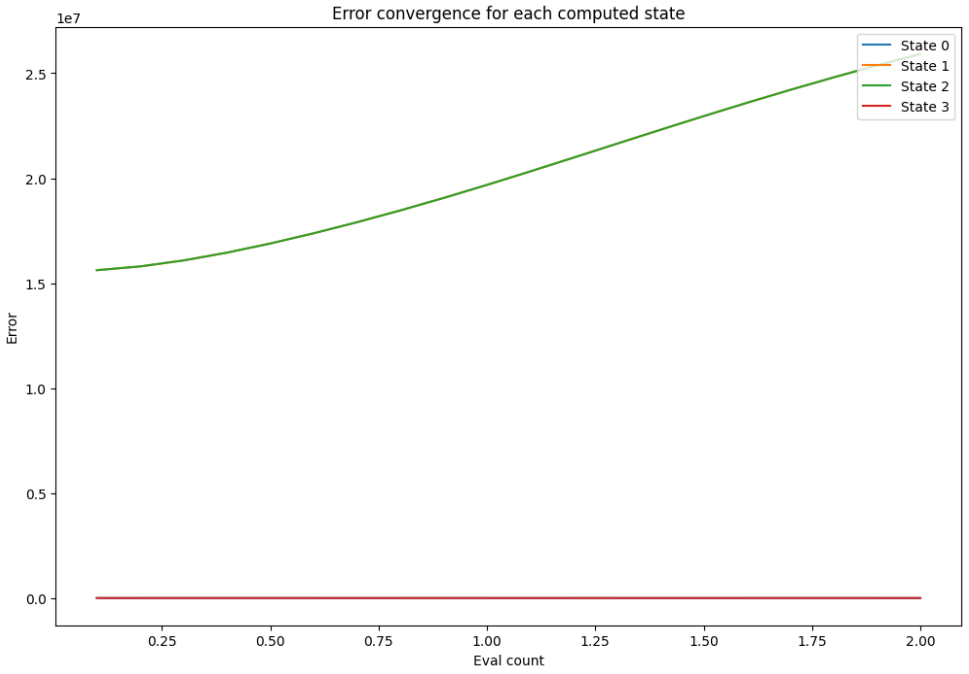
VQE on shots:0, approximation=True(ideal simulator, no noise)



VQE on shots:2048, noise model: FakeVigo

Notice in the VQD Energy Convergence graph below are only talking about electron electron interaction, i.e eigenvalue, we have to approximate to hartree fock, by taking molecule problem, and providing it with eigen value to calculate nuclear interaction to be acounted to see the single dip at around 0.7 demoting least energy





State 2, is visibly showing some error.

3.2 Overview

The most critical takeaway from this study is the paramount importance of selecting the most suitable algorithm among VQE, p-VQD, and VQD for specific quantum simulations. Each algorithm exhibits distinct strengths and limitations, making their appropriateness contingent on the nature of the quantum problem being addressed.

For accurate ground state determination in quantum chemistry, VQE stands out as a reliable choice. Its precision and widespread use in molecular systems make it a preferred option for such applications. On the other hand, VQD emerges as an excellent solution for simulations involving dynamic phenomena, offering real-time insights into time-dependent processes.

The revolutionary p-VQD represents a significant milestone, combining the best features of VQE and VQD to efficiently simulate chemical reactions and molecular dynamics with quantum accuracy. Its unique abilities make it an attractive choice for specific quantum simulations with time-dependent properties.

When implementing these algorithms, it is crucial to consider the nature of the quantum problem, the computational resources available, and the desired level of accuracy. By making the right algorithmic choice, researchers can optimize their simulations and achieve remarkable results with quantum speedup.

In conclusion, the efficiency and success of quantum simulations heavily depend on selecting the most suitable algorithm among VQE, p-VQD, and VQD. This decision is pivotal for achieving accurate results and unlocking the full potential of quantum computing in diverse scientific and industrial domains. As the field progresses, a nuanced understanding of these algorithms' strengths will be instrumental in driving quantum computing advancements and discoveries.

Section 4

Use of Concurrence

Concurrence is a measure used in quantum information theory to quantify the entanglement between two qubits in a quantum system. It plays a crucial role in assessing the efficiency of quantum algorithms, particularly those designed for quantum computations involving entangled states. Here's how concurrence is used to measure the efficiency of quantum algorithms:

Quantum Entanglement: In quantum mechanics, entanglement is a phenomenon where the quantum states of two or more particles become correlated in such a way that their properties are inseparable, regardless of the physical distance between them. Entanglement is a valuable resource in quantum computing, as it enables the representation and processing of information in a way that is not possible classically.

Quantifying Entanglement with Concurrence: Concurrence is a specific measure used to quantify the degree of entanglement between two qubits in a quantum system. It ranges from 0 to 1, where 0 indicates no entanglement (separable states), and 1 indicates maximum entanglement (maximally entangled states).

Efficiency of Quantum Algorithms: In quantum algorithms, particularly those involving quantum circuits with entangled states, the amount of entanglement present can impact the efficiency of the algorithm. Highly entangled states can lead to enhanced quantum parallelism and computational power, enabling quantum algorithms to outperform classical counterparts for certain tasks.

Entanglement and Quantum Speedup: Entanglement is one of the factors that contribute to quantum speedup, where quantum algorithms demonstrate exponential speedup over classical algorithms. Algorithms that can generate and effectively utilize entanglement are more likely to achieve quantum speedup for specific problem instances.

Optimization of Entanglement: In the development and implementation of quantum algorithms, researchers and algorithm designers often seek to optimize and control the level of entanglement present in the quantum states involved. This optimization can involve choosing appropriate

quantum circuits, designing suitable ansatzes for variational algorithms, or finding the right resources for quantum error correction.

Trade-off with Quantum Resources: While entanglement can be advantageous for some quantum algorithms, it can also introduce challenges. Highly entangled states may require more quantum resources, such as additional qubits and complex quantum gates, which can increase the computational cost and error susceptibility of the algorithm.

Overall, concurrence and entanglement play a crucial role in the efficiency and performance of quantum algorithms. Striking the right balance between the amount of entanglement and the required quantum resources is essential for harnessing the full potential of quantum computing and achieving quantum advantage for specific computational tasks.

The concurrence of $|\psi\rangle$ is given by

$$C(|\psi\rangle) = \sqrt{2(1 - Tr[\rho_0^2])}$$

Here $\rho_0 = Tr_1[|\psi\rangle\langle\psi|]$

ρ_0 = reduced state :Partial trace of the input state

To Apply Concurrence :

```
from qiskit.quantum_info import Statevector, concurrence
for j in data['pm']:
    bound_ansatz = ansatz.bind_parameters(j)
    i = Statevector.from_instruction(bound_ansatz)
    data['cc'].append(concurrence(i))
```

Where data[‘pm’] represents list of all parameters, and data[‘cc’] is where it is stored.
We see that We first put all the parameters into the ansatz, and then get the StateVector from it.
We find the concurrency of state vector using qiskit.quantum_info.concurrence, and plot it for every step.
Ref 4.i

Section 5

Summary

This study delved into a comprehensive comparison of VQE, VQD, and p-VQD quantum algorithms, focusing on their efficiency and implementation in quantum computing. The findings emphasised the critical importance of selecting the most suitable algorithm for specific quantum simulations.

VQE, VQD, and p-VQD are promising Variational Quantum Algorithms (VQAs) with wide-ranging applications in electronic structure problems, molecular dynamics, and materials science. VQE excels in accurate ground state determination, making it prevalent in quantum chemistry. VQD offers real-time insights into dynamic phenomena, while p-VQD combines the strengths of VQE and VQD for efficient simulations of chemical reactions and molecular dynamics.

Implementation results showcased the efficiency of each algorithm in various scenarios. VQD handled diverse cases effectively, while VQE demonstrated proficiency in specific problem domains. p-VQD performed well in simulating molecular systems with time-dependent properties.

Concurrence, a measure of entanglement between qubits, along with Standard Deviation and absolute error provided insights into algorithm efficiency and quantum speedup.

In conclusion, the choice of algorithm significantly impacts computational efficiency and accuracy in quantum simulations. As quantum hardware evolves, ongoing research and innovations will enhance VQE, VQD, and p-VQD, making them indispensable for scientific exploration and practical applications.

Section 6

References

1. VQE:
    a. [2111.05176] The Variational Quantum Eigensolver: a review of methods and best practices (arxiv.org)
    b. Fermion-to-qubit mappings with varying resource requirements for quantum simulation - IOPscience
    c. [2103.08505] VQE Method: A Short Survey and Recent Developments (arxiv.org)
    d. (PDF) Optimization of the Variational Quantum Eigensolver for Quantum Chemistry Applications (researchgate.net)
2. VQD:
    a. [1805.08138] Variational Quantum Computation of Excited States (arxiv.org)
    b. Variational Quantum Deflation (VQD) Algorithm — Qiskit 0.43.3 documentation
3. p-VQD
    a. [2101.04579] An efficient quantum algorithm for the time evolution of parameterized circuits (arxiv.org)
    b. Cost function dependent barren plateaus in shallow parametrized quantum circuits | Nature Communications
4. Qiskit
    a. https://qiskit.org/ecosystem/nature/tutorials/06_qubit_mappers.html
    b. https://qiskit.org/documentation/stable/0.31/stubs/qiskit.chemistry.drivers.PySCFDriver.html
    c. https://qiskit.org/documentation/stubs/qiskit.algorithms.MinimumEigensolverResult.html
    d. https://qiskit.org/documentation/stubs/qiskit.algorithms.VQE.compute_minimum_eigenvalue.html
    e. https://qiskit.org/documentation/stubs/qiskit.algorithms.VQE.html
    f. https://qiskit.org/ecosystem/nature/tutorials/01_electronic_structure.html
    g. https://qiskit.org/ecosystem/nature/getting_started.html
    h. https://cloud.ibm.com/docs/quantum-computing?topic=quantum-computing-migrate-qiskit-alg

i. https://qiskit.org/documentation/stubs/qiskit.quantum_info.concurrence.html
j. QiskitRuntimeService — Qiskit Runtime IBM Client 0.11.2 documentation
k. qiskit-community-tutorials/chemistry/h2_vqe_initial_point.ipynb at master · qiskit-community/qiskit-community-tutorials · GitHub
l. SPSA — Qiskit 0.43.3 documentation
m. Estimator — Qiskit Runtime IBM Client 0.11.2 documentation
n. ComputeUncompute — Qiskit 0.43.3 documentation
o. PVQD — Qiskit 0.43.3 documentation
p. https://qiskit.org/documentation/stubs/qiskit.algorithms.TimeEvolutionProblem.html

5. VQA:
   a. Variational quantum algorithms | Nature Reviews Physics