Variational Quantum Eigensolver

Variational Quantum Eigensolver :-

- Hybrid Quantum Classical Algorithm

Hartree-Fock Solution
$|\varphi_{HF}\rangle$

Mapping of Hamiltonian to qubits.

$$\boldsymbol{H = \sum_k \alpha_k \prod_i \sigma_i^k \;\; ; \sigma_i^k = \{I, X, Y, Z\}}$$

initialize reference HF state to qubits

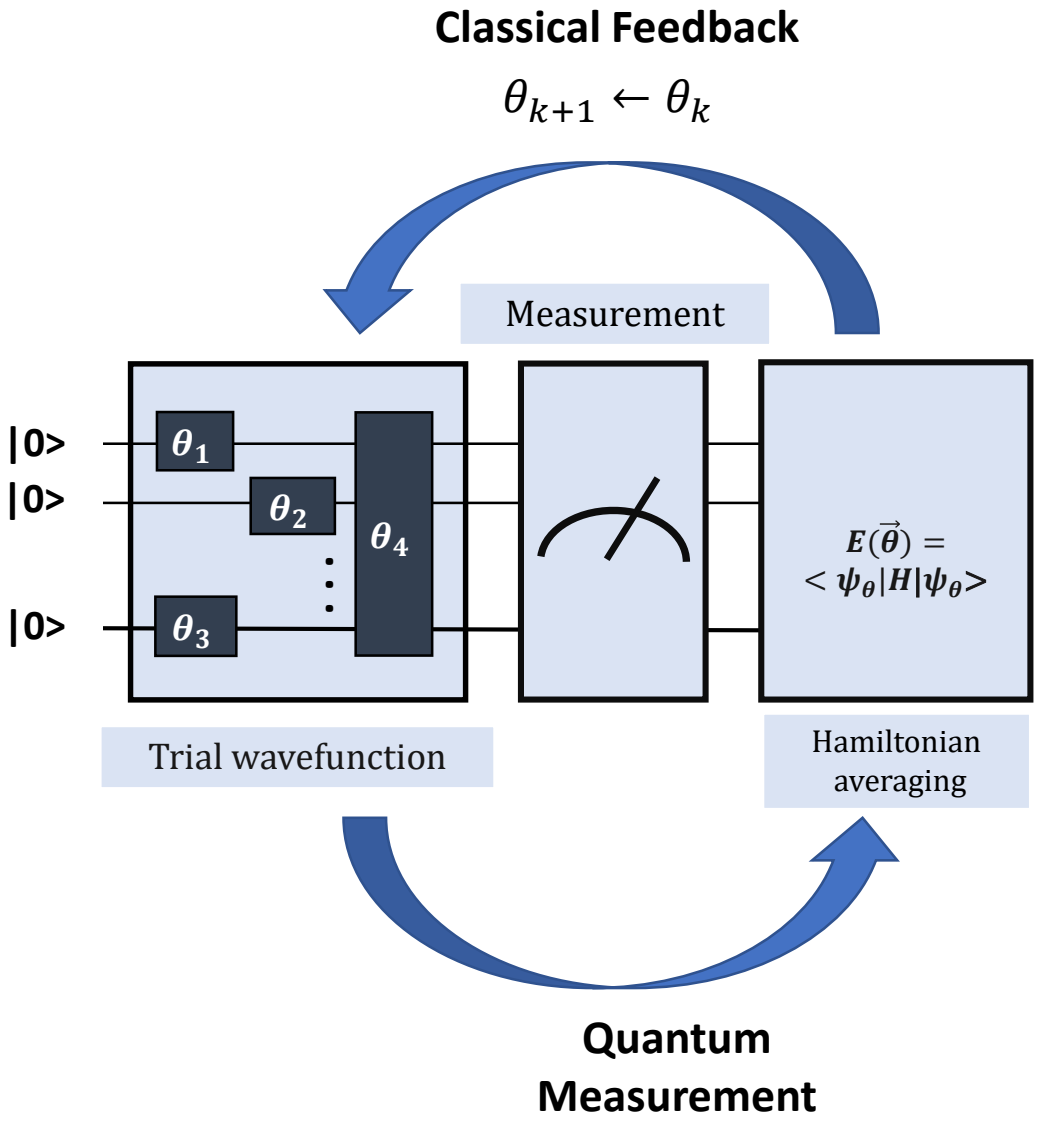
$$|\varphi_{HF}\rangle = |0101\rangle \; ; \quad for\ H_2$$

Parametrized trial state/ansatz Preparation : -

$$|\Psi(\theta)\rangle = U(\theta)|\varphi_{HF}\rangle$$

- Rayleigh-Ritz Variational principle

$$\boldsymbol{E = \langle \Psi(\theta)|H|\Psi(\theta)\rangle \geq E_0}$$



Parametrized Trial Quantum state/Ansatz Preparation :-

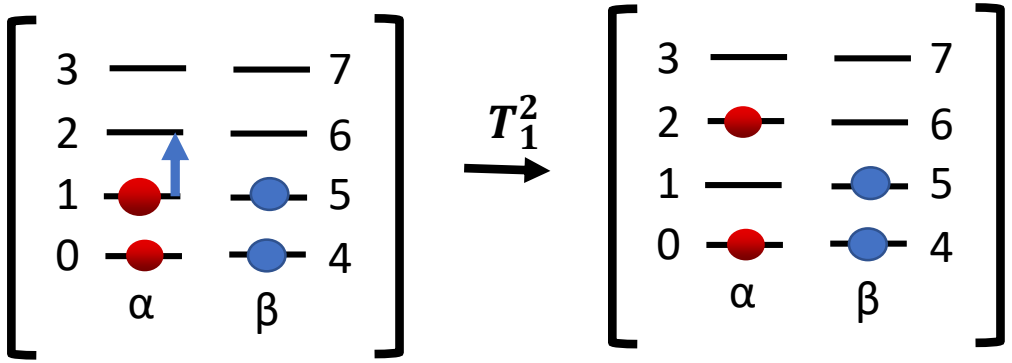
Chemistry-inspired Ansatz (UCC)

Coupled Cluster Ansatz :-

$$|\Psi_{cc}\rangle = e^{T}|\varphi_{HF}\rangle \; ; T = T_1 + T_2 + T_3 \ldots$$

$$T_1 = \sum_{ia} t_i^a a_a^+ a_i \, , T_2 = \sum_{ijab} t_{ij}^{ab} a_a^+ a_b^+ a_i a_j \, , \ldots$$

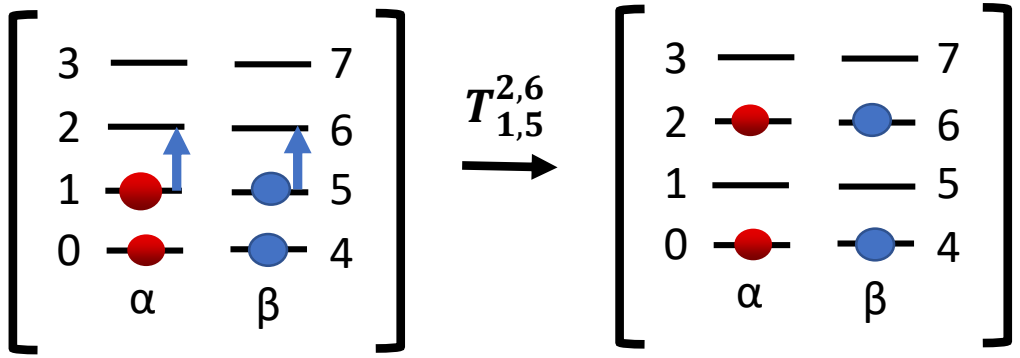
i,j: occupied orbitals; a,b: unoccupied orbitals.
Implementation on Quantum Computer is not possible



HF State

Unitary Coupled Cluster Ansatz :-

$$|\Psi_{UCC}\rangle = e^{\tau}|\varphi_{HF}\rangle \; ; \tau = \tau_1 + \tau_2 + \tau_3 \ldots$$
$$\tau = T - T^{\dagger}$$
$$E = \langle\varphi_{HF}|e^{-\tau}He^{\tau}|\varphi_{HF}\rangle$$



HF State

UCC Theory

$$U(\theta) = \exp(\tau) = \exp(\sum_n T_n - T_n^\dagger)$$

$$E_{UCC} = \langle \varphi_{HF} | e^{-\tau} H e^{\tau} | \varphi_{HF} \rangle = \left\langle \varphi_{HF} \middle| (H + [H,\tau] + \frac{1}{2}[[H,\tau],\tau] + \cdots) \middle| \varphi_{HF} \right\rangle$$

T_n's commute , $T_n^\dagger$'s breaks the commutativity. BCH expansion does not terminate.
To solve UCC on a classical computer, we have to truncate BCH expansion at certain order.
But, we can construct quantum circuit for e^τ on a quantum computer.

Trotterization

$$U_{UCCSD} = \exp\left(\sum_{n=1}^{2} T_n - T_n^\dagger\right) \approx \left[\prod_{ia} e^{\frac{\theta_i^a \tau_i^a}{k}} \prod_{ijab} e^{\frac{\theta_{ij}^{ab} \tau_{ij}^{ab}}{k}}\right]^k ; \tau_i^a = a_a^\dagger a_i - a_i^\dagger a_a$$

$$\exp(\theta_i^a (a_a^\dagger a_i - a_i^\dagger a_a)) \xrightarrow{JWT} \exp(\frac{i}{2}\theta_i^a (\prod_k \sigma_k - \prod_l \sigma_l))$$

$$\approx \exp(\tfrac{i}{2}\theta_i^a \textstyle\prod_k \sigma_k) \times \exp(-\tfrac{i}{2}\theta_i^a \prod_l \sigma_l)$$

Quantum circuit formation for exponentiated products of Pauli Operators :-

$$|\Psi(\boldsymbol{t})\rangle = \boldsymbol{U(\theta)}|\Psi_0\rangle$$

$$|\Psi(\boldsymbol{t})\rangle = \boldsymbol{e^{-iP\alpha}}|\Psi_0\rangle$$
$$\boldsymbol{P = Z_0 Z_1}$$
$$|\Psi(\boldsymbol{t})\rangle = \boldsymbol{e^{-i\alpha Z_0 Z_1}}\ |\Psi_0\rangle$$

$$|\Psi_0\rangle = |a_0 a_1\rangle$$
$$e^{-i\alpha Z_0 Z_1}|a_0 a_1\rangle = \{\cos(\alpha)I - isin(\alpha)(Z_0 \otimes Z_1)\}|a_0 a_1\rangle$$

Where a_i's can be either 0 or 1
If $|a_0 a_1\rangle$ contains even no of 1's then parity of the string will be 0 (even parity) and if there is odd no of 1's then parity will be 1 (odd parity).

Even parity :- $(Z_0 \otimes Z_1)|a_0 a_1\rangle = |a_0 a_1\rangle$
Odd parity :- $(Z_0 \otimes Z_1)|a_0 a_1\rangle = -|a_0 a_1\rangle$

Final state

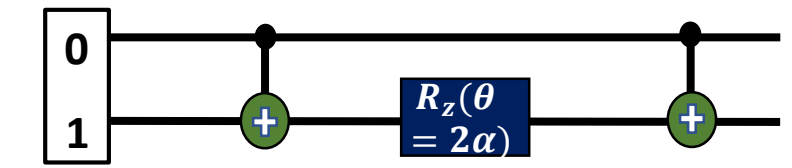
Even Parity :- $|\Psi(\boldsymbol{t})\rangle = \boldsymbol{e^{-i\alpha}}\ |\boldsymbol{a_0 a_1}\rangle$
Odd parity :- $|\Psi(\boldsymbol{t})\rangle = \boldsymbol{e^{i\alpha}}\ |\boldsymbol{a_0 a_1}\rangle$

$$\boldsymbol{R_Z(\theta)} = \begin{pmatrix} \boldsymbol{e^{-i\frac{\theta}{2}}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{e^{i\frac{\theta}{2}}} \end{pmatrix}$$

$$R_z(\theta)\left|0\right\rangle = e^{-i\theta/2}\left|0\right\rangle$$
$$R_z(\theta)\left|1\right\rangle = e^{i\theta/2}\left|1\right\rangle$$

How to get parity through circuit

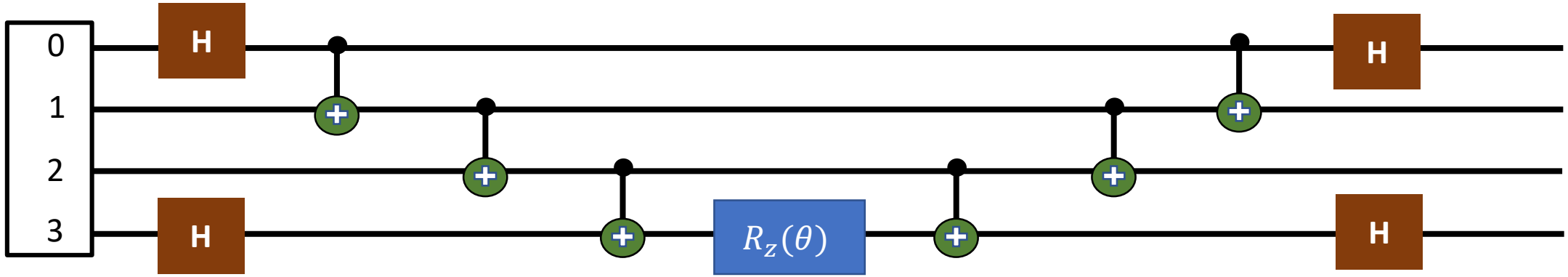
$$CNOT(0,1)|00\rangle = |00\rangle$$
$$CNOT(0,1)|01\rangle = |01\rangle$$
$$CNOT(0,1)|10\rangle = |11\rangle$$
$$CNOT(0,1)|11\rangle = |10\rangle$$



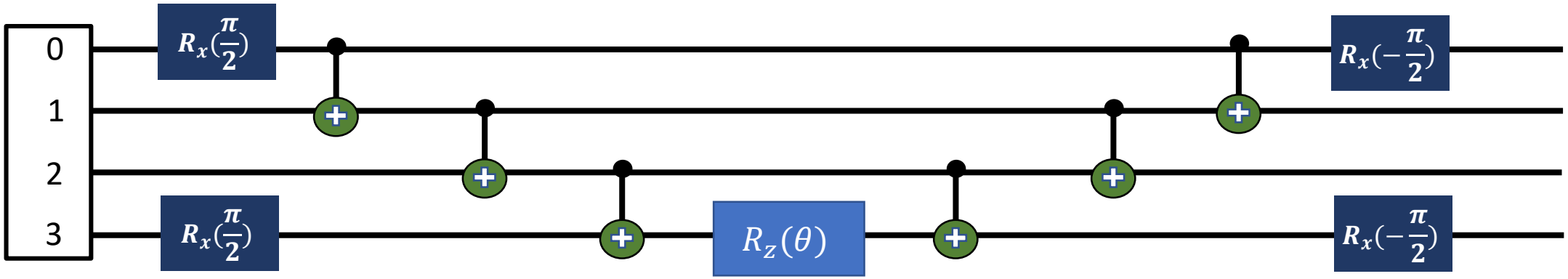
$|\Psi(t)\rangle$

$|\Psi_0\rangle$

Circuit for $e^{-i\frac{\theta}{2}X_0Z_1Z_2X_3}$: —



Circuit for $e^{-i\frac{\theta}{2}Y_0Z_1Z_2Y_3}$: —



Implementation of VQE algorithm for H_2 on Qiskit

In [ ]:

```
from qiskit_nature.drivers.second_quantization import PySCFDriver
from qiskit_nature.problems.second_quantization.electronic import ElectronicStructureProblem

from qiskit_nature.converters.second_quantization import QubitConverter
from qiskit_nature.mappers.second_quantization import JordanWignerMapper
from qiskit_nature.circuit.library import HartreeFock, UCC
from qiskit.algorithms.optimizers import COBYLA, CG, SLSQP, L_BFGS_B
from qiskit.algorithms import VQE
from qiskit_nature.algorithms.ground_state_solvers.minimum_eigensolver_factories import NumPyMinimumEigensolverFactory
from qiskit import Aer
import numpy as np
from qiskit_nature.algorithms.ground_state_solvers import GroundStateEigensolver
from qiskit.circuit import Parameter, QuantumCircuit, QuantumRegister
```

In [ ]:

```
coord = 'H 0.0 0.0 0.0; H 0.0 0.0 0.735'
driver = PySCFDriver(atom=coord, charge=0, spin=0, basis='sto3g')
es_problem = ElectronicStructureProblem(driver)

# obtaining qubit Hamiltonian
mapper = JordanWignerMapper()
converter = QubitConverter(mapper=mapper, two_qubit_reduction=False)
second_q_op = es_problem.second_q_ops()
print(second_q_op[0])
qubit_op = converter.convert(second_q_op[0])
print(qubit_op)
```

Fermionic Operator
register length=4, number terms=14
(0.18093119978423106+0j) * (+_0 -_1 +_2 -_3)
+ (-0.18093119978423128+0j) * (+_0 -_1 -_2 +_3)
+ (-0.18093119978423128+0j) * (-_0 +_1 +_2 -_3)
+ (0.18093119978423144+0j) * (-_0 +_1 -_2 +_3 ...

-0.8105479805373281 * IIII
- 0.22575349222402358 * ZIII
+ 0.17218393261915543 * IZII
+ 0.12091263261776633 * ZZII
- 0.22575349222402358 * IIZI....

In [3]:
```
es_particle_number = es_problem.grouped_property_transformed.get_property('ParticleNumber')
num_particles = (es_particle_number.num_alpha, es_particle_number.num_beta)
num_spin_orbitals = es_particle_number.num_spin_orbitals
es_energy = es_problem.grouped_property_transformed.get_property('ElectronicEnergy')
nuclear_repulsion_energy = es_energy.nuclear_repulsion_energy
shift = nuclear_repulsion_energy
print('Number Of Particles: ',num_particles)
print('Number of Spin Orbitals: ', num_spin_orbitals)
print('Nuclear Repulsion Energy: ', nuclear_repulsion_energy)
```

Number Of Particles: (1, 1)
Number of Spin Orbitals: 4
Nuclear Repulsion Energy: 0.7199689944489797

In [4]:
```
#initial_state
init_state = HartreeFock(num_spin_orbitals, num_particles, converter)

#Create dummy parametrized circuit
theta = Parameter('a')
n = qubit_op.num_qubits
qc = QuantumCircuit(qubit_op.num_qubits)
qc.rz(theta*0,0)
ansatz = qc
ansatz.compose(init_state, front=True, inplace=True)

#Pass it through VQE
algorithm = VQE(ansatz,quantum_instance=backend)
result = algorithm.compute_minimum_eigenvalue(qubit_op).eigenvalue
print('HF Energy is',np.real(result)+shift)
```

HF Energy is -1.116998996754004

In [5]:
```
var_form = UCC(num_particles=num_particles,num_spin_orbitals=num_spin_orbitals,excitations='sd',initial_state=init_state,
               qubit_converter=converter)
excitation_list = var_form._get_excitation_list()
print('no of parameters',var_form.num_parameters)
print(excitation_list)
```

no of parameters 3
Excitation list is [((0,), (1,)), ((2,), (3,)), ((0, 2), (1, 3))]

In [6]:
```
optimizer = COBYLA(maxiter=10000)
backend = Aer.get_backend('statevector_simulator')
counts = list()
values = list()
def store_intermediate_result(eval_count, parameters, mean, std):
    counts.append(eval_count)
    values.append(mean)
vqe = VQE(var_form, optimizer=optimizer, callback = store_intermediate_result, quantum_instance=backend)
vqe_result = vqe.compute_minimum_eigenvalue(qubit_op)
E1 = np.real(vqe_result.eigenvalue)+shift
print(E1)
```

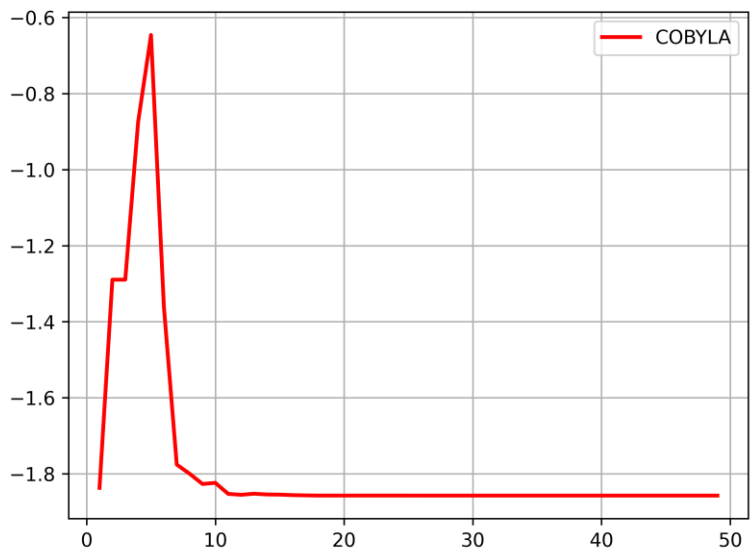
VQE Optimized UCCSD Energy is -1.1373060333734046

In [7]:
```
# NumPyMinimumEigensolver
solver = NumPyMinimumEigensolverFactory()
calc = GroundStateEigensolver(converter, solver)
numpy_result = calc.solve(es_problem)
exact_energy = np.real(numpy_result.eigenenergies[0]) + shift
print('EXACT ENERGY: ', exact_energy)
```

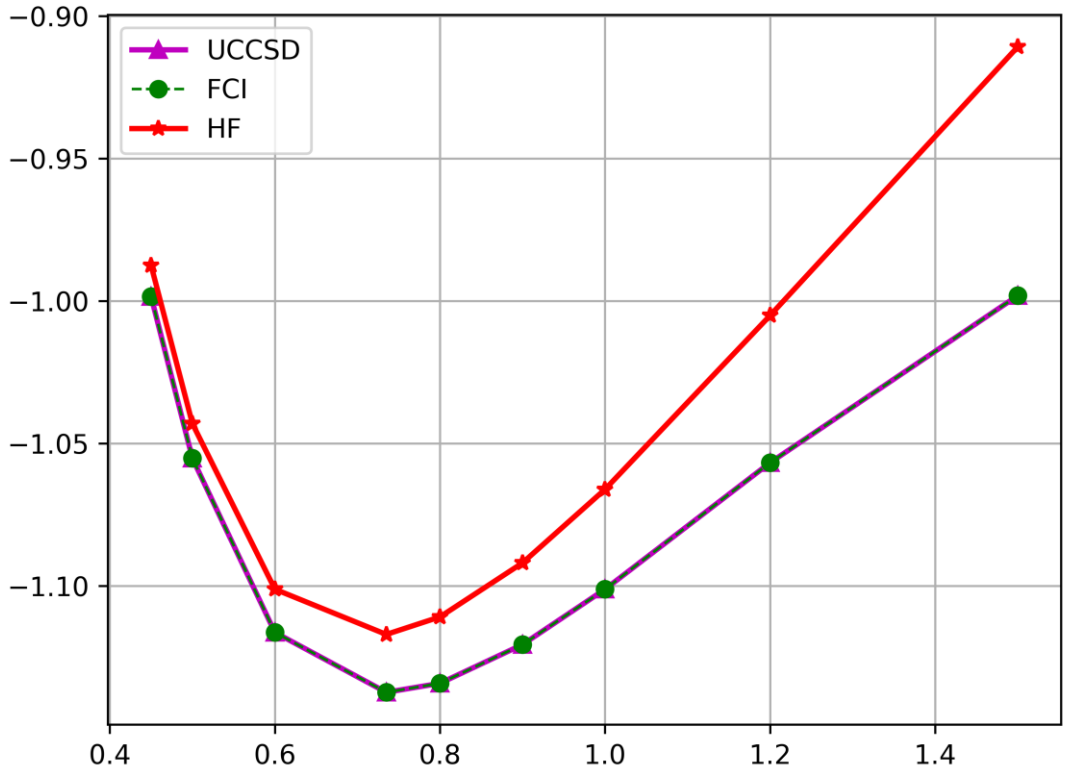
EXACT ENERGY: -1.1373060357534013

In [ ]:
```
import matplotlib.pyplot as plt

plt.plot(counts,values,c='r',linewidth=2,label='COBYLA')
plt.legend()
plt.grid()
plt.show()
```



Potential Energy Surface for H_2 :-



Try yourself

Questions

- Draw the Optimization pattern for different optimizer for LiH.
- Draw the potential energy surface LiH (HF, UCCSD, FCI).
- Draw the potential energy surface for H_2 (HF, UCCD, FCI).
- Draw the potential energy surface for LiH (HF, UCCD, UCCS, UCCSD, FCI).

rb.gy/vcabgp