Tqr

November 5, 2024

[ ]:
```python
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver
import matplotlib.pyplot as plt
from qiskit.quantum_info import Z2Symmetries, SparsePauliOp, Operator
driver = PySCFDriver(
    atom="He 0 0 0; H+ 0 0 0.772",
    basis="sto-3g",
    charge=1,
    spin=0,
)

problem = driver.run()
#print(problem)
```

[ ]:
```python
fermionic_op = problem.hamiltonian.second_q_op()
#print(fermionic_op)
print('spatial orbitals: ', problem.num_spatial_orbitals)
print('spin orbitals: ', problem.num_spin_orbitals)
print('num particles: ', problem.num_particles)
```

[ ]:
```python
from qiskit_nature.second_q.mappers import JordanWignerMapper, ParityMapper
from IPython.display import display, Math

mapper = JordanWignerMapper()
qubit_jw_op = mapper.map(fermionic_op)
print(qubit_jw_op)
# Convert the operator to a matrix
operator = Operator(qubit_jw_op)
matrix = operator.data

# Function to convert a matrix to a LaTeX string
def matrix_to_latex(matrix, chunk_size=10):
    latex_str = "\\begin{bmatrix}\n"
    for i, row in enumerate(matrix):
        latex_str += " & ".join([f"{elem:.3f}" for elem in row]) + " \\\\\n"
        if (i + 1) % chunk_size == 0:
            latex_str += "\\end{bmatrix}\n\\begin{bmatrix}\n"
    latex_str += "\\end{bmatrix}"
```

```python
    return latex_str

# Convert the matrix to a LaTeX string in chunks
latex_str = matrix_to_latex(matrix)

# Display the LaTeX string
display(Math(latex_str))

Operator(qubit_jw_op).draw('latex')
```

[ ]:
```python
def reduce_operator(op, reduced=False):
    if reduced:
        z2_symmetries = Z2Symmetries.find_z2_symmetries(op)
        newOp = Z2Symmetries(z2_symmetries.symmetries, z2_symmetries.sq_paulis,␣
 ↪z2_symmetries.sq_list)
        tapered_ops = newOp.taper(op)
        print(tapered_ops)
        first_tapered_op = tapered_ops[1]
        return first_tapered_op
    else:
        return op

operator = reduce_operator(qubit_jw_op, reduced=True)


#print('sq_list: ',Z2Symmetries.find_z2_symmetries(qubit_jw_op).sq_list)
#print('sq_paulis: ',Z2Symmetries.find_z2_symmetries(qubit_jw_op).sq_paulis)
#print('Symmetries: ',Z2Symmetries.find_z2_symmetries(qubit_jw_op).symmetries)

Operator(operator).draw('latex')
```

[ ]:
```python
cost_history_dict = {
    "prev_vector": None,
    "iters": 0,
    "cost_history": [],
}
```

[ ]:
```python
from qiskit_algorithms.optimizers import SLSQP,SPSA,COBYLA, L_BFGS_B
from qiskit_algorithms import VQE
from qiskit.circuit.library import RealAmplitudes
from qiskit.primitives import StatevectorEstimator
estimator = StatevectorEstimator()
ansatz = RealAmplitudes(operator.num_qubits, reps=1)
optimizer = SLSQP()
ansatz.decompose().draw('mpl')
```

[ ]:
```python
def cost_func_vqe(parameters, ansatz, hamiltonian, estimator):
    """Return estimate of energy from estimator

    Parameters:
        params (ndarray): Array of ansatz parameters
        ansatz (QuantumCircuit): Parameterized ansatz circuit
        hamiltonian (SparsePauliOp): Operator representation of Hamiltonian
        estimator (Estimator): Estimator primitive instance

    Returns:
        float: Energy estimate
    """

    estimator_job = estimator.run([(ansatz, hamiltonian, [parameters])])
    estimator_result = estimator_job.result()[0]

    cost = estimator_result.data.evs[0]
    cost_history_dict["iters"] += 1
    cost_history_dict["prev_vector"] = parameters
    cost_history_dict["cost_history"].append(cost)
    return cost
```

[ ]:
```python
from scipy.optimize import minimize
import numpy as np
initial_params = np.random.uniform(low= -np.pi, high=np.pi, size=ansatz.
 ↪num_parameters)
```

[ ]:
```python
result = minimize(cost_func_vqe, initial_params, args=(ansatz, operator,␣
 ↪estimator), method="COBYLA",options={'maxiter': 12000})
print(result)
```

[ ]:
```python
fig, ax = plt.subplots()
ax.plot(range(cost_history_dict["iters"]), cost_history_dict["cost_history"])
ax.set_xlabel("Iterations")
ax.set_ylabel("Cost")
plt.draw()
```