Problem 5

This problem covers concepts about cost functions, optimizers, instances, and applications.

Problem 5a

Build SSVQE's cost function using the following observable:

$$ O = 2II-2XX+3YY-3ZZ $$
This cost function should take in parameters as an input. Use the included variational form, with reference states $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ and weights $w_{00}=50, w_{01}=40, w_{10}= 30, w_{11}=20$.

▶ Hint 1

▶ Hint 2

In [18]:

```
from qiskit import QuantumCircuit
from qiskit import IBMQ, Aer
from qiskit_ibm_runtime import QiskitRuntimeService, Estimator
from qiskit.circuit.library import TwoLocal
from qiskit.quantum_info import SparsePauliOp
from qiskit.primitives import Estimator
import numpy as np
from qc_grader.challenges.algorithm_design import grade_problem_5a
from math import pi
```

In [22]:

```
from qiskit import QuantumCircuit
from qiskit.circuit.library import TwoLocal
from qiskit.quantum_info import SparsePauliOp
from qiskit.primitives import Estimator
import numpy as np
from qc_grader.challenges.algorithm_design import grade_problem_5a
ansatz_list = TwoLocal(2, rotation_blocks=['rz', 'ry'], entanglement_blocks='cx'
weight_vector = [50, 40, 30, 20]
estimator = Estimator()
hamiltonian = SparsePauliOp.from_list([("II", 2), ("XX", -2), ("YY", 3), ("ZZ",
def cost_function(theta: list[float],k:int, ansatz_list:TwoLocal, weight_vector:
    hamiltonian = SparsePauliOp.from_list([("II", 2), ("XX", -2), ("YY", 3), ("Z
    reference_circuits = []
    for i in range(2):
        for j in range(2):
            qc = QuantumCircuit(2)
            if j == 1:
                qc.x(0)
            if i == 1:
                qc.x(1)
            reference_circuits.append(qc)


    ansatz_list = TwoLocal(2, rotation_blocks=['rz', 'ry'], entanglement_blocks=
```

```
    values = 0
    weight_vector = [50, 40, 30, 20]
    for i in range(k+1):
        reference_circuits[i]=reference_circuits[i].compose(ansatz_list)
        estimator = Estimator()
        job = estimator.run(reference_circuits[i], hamiltonian, theta)
        values += weight_vector[i]*job.result().values

    return values # TODO: return the right value given the input parameters


grade_problem_5a(cost_function,k,ansatz_list,weight_vector,hamiltonian,estimator
```

```
Submitting your answer. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
Your score is 3.
```

Problem 5b

After you have your `cost_function`, use a classical optimizer to calculate the optimal parameters and eigenvalues for the several ansatze.

In [25]:

```
from qiskit.algorithms.optimizers import COBYLA
from qiskit.circuit.library import TwoLocal
from qiskit.quantum_info import SparsePauliOp
from qiskit.primitives import Estimator
import numpy as np
from qc_grader.challenges.algorithm_design import grade_problem_5b
reference_circuits = []
for i in range(2):
    for j in range(2):
        qc = QuantumCircuit(2)
        if j == 1:
            qc.x(0)
        if i == 1:
            qc.x(1)
        reference_circuits.append(qc)
ansatz_list = TwoLocal(2, rotation_blocks=['rz', 'ry'], entanglement_blocks='cx'
eigenvalues = []
initial_theta = np.ones(8)
optimizer = COBYLA()

def cost_fun_ssvqe(theta:list[float]):
    return cost_function(theta,k, ansatz_list, weight_vector,hamiltonian, estima

optimizer_result = optimizer.minimize(fun=cost_fun_ssvqe, x0=initial_theta)

optimal_parameters = optimizer_result.x

for i in range(4):
    reference_circuits[i]=reference_circuits[i].compose(ansatz_list)
    estimator = Estimator()
    job = estimator.run(reference_circuits[i], hamiltonian, optimal_parameters)
    eigenvalues.append(job.result().values)
print(eigenvalues)


grade_problem_5b(eigenvalues)
```

```
[array([-5.99999998]), array([4.]), array([4.00000002]), array([5.99999997])]
Submitting your answer. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
Your score is 2.
```

Return to the assessment to earn your badge!