E91 Protocol

Author: Tymon Dydowicz
Environment: Qiskit, Python 3.10.11
Student ID: 151936

Project Overview

This project demonstrates implementation and verification of E91 Protocol

Key Tasks

1. **Single-qubit measurements** — using Z, X, and H operations
2. **Quantum state tomography** — measuring target qubit across X, Y, Z bases
3. **CHSH simulation** — preparing singlets, measuring Alice & Bob in random bases, and evaluating correlations
4. **Key reconstruction** — filtering results where Alice's and Bob's bases are appropriate

All simulations use:

- **2-qubit singlet states** for CHSH
- **4-qubit system** for prior measurements
- **1024 singlets** per CHSH simulation
- **1 shot per measurement**
- **Random seed:** 151936

Methodology

CHSH Measurement Code

The simulation constructs the singlet state (|\psi^-\rangle = (|01\rangle - |10\rangle)/\sqrt{2}) and performs random measurements:

```
SINGLETS = 1024
SHOTS = 1
SEED = 151936
BACKEND = AerSimulator()

b = [random.choice(['X','Y','Z']) for _ in range(SINGLETS)]
b_prime = [random.choice(['X','Y','Z']) for _ in range(SINGLETS)]

results = measureSinglets(num_singlets=SINGLETS, shots=SHOTS, seed=SEED,
backend=BACKEND)
```

Alice and Bob's qubits are measured according to their randomly chosen bases, using `measureAlice` and `measureBob` functions, with rotations applied according to the presentation.

```
def prepareSinglet():
    qr = QuantumRegister(2, 'qr')
    cr = ClassicalRegister(4, 'cr')
    circ = QuantumCircuit(qr, cr)

    circ.x(0)
    circ.x(1)
    circ.h(0)
    circ.cx(0, 1)

    return circ

def measureAlice(circ, basis, target_qubit=0, classical_bit=0):
    formated_basis = basis.upper()
    match formated_basis:
        case 'X':
            circ.h(target_qubit)
        case 'Y':
            circ.s(target_qubit)
            circ.h(target_qubit)
            circ.t(target_qubit)
            circ.h(target_qubit)
        case 'Z':
            pass
        case _:
            raise ValueError("Basis must be 'X', 'Y', or 'Z'")

    circ.measure(target_qubit, classical_bit)

def measureBob(circ, basis, target_qubit=1, classical_bit=1):
    formated_basis = basis.upper()
    match formated_basis:
        case 'X':
            circ.s(target_qubit)
            circ.h(target_qubit)
            circ.t(target_qubit)
            circ.h(target_qubit)
        case 'Y':
            pass
        case 'Z':
            circ.s(target_qubit)
            circ.h(target_qubit)
            circ.tdg(target_qubit)
            circ.h(target_qubit)
        case _:
            raise ValueError("Basis must be 'X', 'Y', or 'Z'")

    circ.measure(target_qubit, classical_bit)
```

Key Reconstruction

Matching basis measurements are filtered using `compareBasis`, with Bob's bits corrected to match Alice's basis conventions. Mismatches are counted to evaluate experimental fidelity.

```
def revealBasis(results):
    alice_key = []
    bob_key = []
    mismatch_count = 0

    for i in range(len(results)):
        alices_basis, bob_basis, alice_bit, bob_bit = results[i].values()
        matching = compareBasis(alices_basis, bob_basis)
        if matching:
            alice_key.append(alice_bit)
            bob_bit_corrected = 1 - bob_bit
            bob_key.append(bob_bit_corrected)

            if alice_bit != bob_bit_corrected:
                mismatch_count += 1

    return alice_key, bob_key, mismatch_count

alice_key, bob_key, mismatches = revealBasis(results)
```

CHSH Value Calculation

Correlation expectations are calculated per setting:

```
def groupResults(results):
    grouped = defaultdict(list)

    for r in results:
        b = BASIS_MAP[r["alice_basis"]]
        b_p = BASIS_MAP[r["bob_basis"]]
        a = 1 - 2 * r["alice_bit"]
        a_p = 1 - 2 * r["bob_bit"]

        grouped[(b, b_p)].append((a, a_p))

    return grouped

def countGroup(group):
    counts = {
        (1,1): 0,
        (1,-1): 0,
        (-1,1): 0,
        (-1,-1): 0
    }
```

```
    for pair in group:
        a, a_p = pair
        counts[(a, a_p)] += 1

    return counts

def calculateExpectation(counts):
    total = sum(counts.values())
    expectation = 0.0

    for (a, a_p), count in counts.items():
        expectation += a * a_p * (count / total)

    return expectation


def calculateCHSH(results):
    grouped = groupResults(results)

    E_XW = calculateExpectation(countGroup(grouped[(1,1)]))
    E_XV = calculateExpectation(countGroup(grouped[(1,3)]))
    E_ZW = calculateExpectation(countGroup(grouped[(3,1)]))
    E_ZV = calculateExpectation(countGroup(grouped[(3,3)]))

    S = E_XW - E_XV + E_ZW + E_ZV

    return S
```

This includes:

- (E(X,W)), (E(X,V)), (E(Z,W)), (E(Z,V))
- CHSH value (S = E(X,W) - E(X,V) + E(Z,W) + E(Z,V))
- Comparison against classical limit

Detailed Results Table

The following table summarizes measurement outcomes, probabilities, contributions, and expectations for the CHSH test.

Measurement	(b,b')	(a,a')	n_ij	p_ij	p·(a·a')	Expectation
X ⊗ W	(1, 1)	(1, 1)	7	0.0625	0.0625	-0.8036
X ⊗ W	(1, 1)	(1, -1)	48	0.4286	-0.4286	-0.8036
X ⊗ W	(1, 1)	(-1, 1)	53	0.4732	-0.4732	-0.8036
X ⊗ W	(1, 1)	(-1, -1)	4	0.0357	0.0357	-0.8036
X ⊗ V	(1, 3)	(1, 1)	59	0.5175	0.5175	0.7895
X ⊗ V	(1, 3)	(1, -1)	6	0.0526	-0.0526	0.7895
X ⊗ V	(1, 3)	(-1, 1)	6	0.0526	-0.0526	0.7895

Measurement	(b,b')	(a,a')	n_ij	p_ij	p·(a·a')	Expectation
X ⊗ V	(1, 3)	(-1, -1)	43	0.3772	0.3772	0.7895
Z ⊗ W	(3, 1)	(1, 1)	6	0.0588	0.0588	-0.7451
Z ⊗ W	(3, 1)	(1, -1)	54	0.5294	-0.5294	-0.7451
Z ⊗ W	(3, 1)	(-1, 1)	35	0.3431	-0.3431	-0.7451
Z ⊗ W	(3, 1)	(-1, -1)	7	0.0686	0.0686	-0.7451
Z ⊗ V	(3, 3)	(1, 1)	5	0.0407	0.0407	-0.7886
Z ⊗ V	(3, 3)	(1, -1)	56	0.4553	-0.4553	-0.7886
Z ⊗ V	(3, 3)	(-1, 1)	54	0.4390	-0.4390	-0.7886