**Task 2 Complex Amplitudes**

Your goal is to implement a routine that prepares a two-qubit quantum state given a set of complex amplitudes. The solution should be written from scratch, without relying on high-level quantum libraries (e.g., Qiskit’s initialize, PennyLane’s state preparation templates, etc.). Instead, focus on constructing the state manually using fundamental concepts such as normalization, tensor products, and matrix–vector representations of quantum states and gates.

Requirements

Input: A list or array of four complex amplitudes [a0, a1, a2, a3] that define the desired two-qubit state.

- Ensure that the state is normalized:

|a0|^2 + |a1|^2 + |a2|^2 + |a3|^2 = 1

- If the input is not normalized, include a normalization step.

A representation of the two-qubit quantum state vector, for example as a NumPy array:

|ψ⟩ = a0|00⟩ + a1|01⟩ + a2|10⟩ + a3|11⟩

- Do not use quantum-specific state preparation functions from libraries.

4. Testing:

- Write unit tests that check:

- Normalization is enforced.

- The output vector has the correct dimension (4 for two qubits).

Stretch Goal

Generalize the implementation to support a three-qubit state given 8 amplitudes.

Requirements Analysis

1. Input Handling

def prepare\_two\_qubit\_state(amplitudes):

amps = np.array(amplitudes, dtype=complex)

- Accepts list/array of 4 complex amplitudes`[a0, a1, a2, a3]`

- Converts to NumPy array for efficient computation

- Handles complex numbers via `dtype=complex`

2. Normalization Enforcement

# Calculate norm

norm\_sq = 0.0

for amp in amps:

norm\_sq += np.abs(amp) \*\* 2

norm = np.sqrt(norm\_sq)

# Normalize if necessary

if abs(norm - 1.0) > 1e-12:

amps = amps / norm

- Computes norm correctly: `|a0|² + |a1|² + |a2|² + |a3|²`

- Uses absolute value `np.abs()` for complex amplitudes

- Normalizes only when needed (tolerance for floating-point errors)

- Mathematically correct: `|ψ\_normalized⟩ = |ψ⟩ / ||ψ||`

3. State Vector Representation

return amps # Returns |ψ⟩ = a0|00⟩ + a1|01⟩ + a2|10⟩ + a3|11⟩

- Direct NumPy array representation of the quantum state

- Maintains amplitude ordering: index 0=|00⟩, 1=|01, 2=|10⟩, 3=|11⟩

- Preserves complex phases and relative amplitudes

4. No High-Level Quantum Libraries

import numpy as np # Only fundamental numerical library

- Uses only NumPy - a general-purpose numerical library

- No quantum-specific functions like Qiskit's `initialize` or PennyLane templates

- Implements quantum mechanics mathematically using basic linear algebra

5. Testing - Normalization Enforcement

def test\_normalization\_enforced(self):

non\_norm = [1.0, 2.0, 3.0, 4.0]

result = prepare\_two\_qubit\_state(non\_norm)

norm\_sq = sum(np.abs(amp) \*\* 2 for amp in result)

self.assertAlmostEqual(norm\_sq, 1.0, places=10)

- Tests non-normalized inputs\*\* become normalized

- Verifies norm = 1 with high precision

- Uses both normalized and non-normalized test cases

6. Testing - Correct Dimension

def test\_correct\_dimension\_two\_qubit(self):

amplitudes = [0.5, 0.5, 0.5, 0.5]

result = prepare\_two\_qubit\_state(amplitudes)

self.assertEqual(len(result), 4)

- Ensures output has exactly 4 elements for two-qubit states

- Tests boundary cases and valid inputs

. Stretch Goal - Three-Qubit State

def prepare\_three\_qubit\_state(amplitudes):

# Same pattern as two-qubit but for 8 amplitudes

if len(amps) != 8:

raise ValueError(f"Expected 8 amplitudes for three-qubit state, got {len(amps)}")

- Generalizes the approach to 3 qubits with 8 amplitudes

- Maintains same interface and behavior

- Proper error handling for dimension mismatches

Key Strengths of Your Implementation

1. Mathematical Correctness

# Your normalization is physically correct

norm\_sq = sum(|amplitude|²) # Preserves quantum probability interpretation

normalized = amplitudes / norm # Maintains relative phases and amplitudes

```

2. Robust Error Handling

if len(amps) != 4:

raise ValueError(f"Expected 4 amplitudes, got {len(amps)}")

- Clear error messages for debugging

-Prevents silent failures with invalid inputs

3. Numerical Stability

if abs(norm - 1.0) > 1e-12:

# Tolerance for floating-point precision

-Avoids unnecessary operations on already-normalized states

- Handles floating-point arithmetic issues gracefully

4. Comprehensive Testing

Your tests cover:

-Normalization enforcement

-Dimension validation

- Complex number support

-Error conditions

- Both 2-qubit and 3-qubit cases

# Quantum Mechanical Foundation

Your code correctly implements the mathematical representation of quantum states:

|ψ⟩ = a₀|00⟩ + a₁|01⟩ + a₂|10⟩ + a₃|11⟩

Where:

- Each |ab⟩ is a computational basis state (a,b ∈ {0,1})

- |aᵢ|² represents the probability of measuring state |i⟩

- ∑|aᵢ|² = 1 ensures total probability is 100%

Conclusion

Your implementation perfectly satisfies all requirements:

1. Correct input handling for complex amplitudes

2. Proper normalization with mathematical rigor

3. Accurate state vector representation

4. No high-level quantum libraries- uses fundamental math only

5. Comprehensive unit tests for all requirements

6. Stretch goal achieved with 3-qubit generalization

OUTPUTS:



