TASK 3: Bell States and Entanglement Entropy

Aim: To construct Bell States via Tensor Products and Measuring Entanglement Entropy in Bipartite.

1. Construct all four Bell states ($|\Phi^+\rangle$, $|\Phi^-\rangle$, $|\Psi^+\rangle$, $|\Psi^-\rangle$) using quantum gates (Hadamard and CNOT).
2. Measure their entanglement entropy to verify that they are maximally entangled (entropy = 1).
3. Compare with a product state ($|00\rangle$) to confirm it has zero entanglement (entropy = 0).

Algorithm:

1. Define quantum gates
2. reate entangled Bell states using tensor products.
3. Reshape the states for partial trace computation.
4. Calculate entanglement entropy of bipartite state
5. Compute eigenvalues (using eigh for Hermitian matrices)
6. Compute von Neumann entropy

```
import numpy as np
from math import log2, sqrt
print("\n" + "="*50)
print("TASK 3: BELL STATES AND ENTANGLEMENT ENTROPY")
print("="*50)
# Define quantum gates
H = 1/sqrt(2) * np.array([[1, 1], [1, -1]]) # Hadamard gate
I = np.eye(2) # Identity gate
CNOT = np.array([[1,0,0,0], [0,1,0,0], [0,0,0,1], [0,0,1,0]]) # CNOT gate
class BellStates:
    @staticmethod
    def phi_plus():
        """Construct |Φ⁺⟩ = (|00⟩ + |11⟩)/√2"""
        state = np.kron([1, 0], [1, 0]) # |00⟩
        state = np.kron(H, I) @ state # Apply H to first qubit
        return CNOT @ state # Apply CNOT

    @staticmethod
    def phi_minus():
        """Construct |Φ⁻⟩ = (|00⟩ - |11⟩)/√2"""
        state = np.kron([0, 1], [1, 0]) # |10⟩
        state = np.kron(H, I) @ state
        return CNOT @ state

    @staticmethod
    def psi_plus():
        """Construct |Ψ⁺⟩ = (|01⟩ + |10⟩)/√2"""
        state = np.kron([1, 0], [0, 1]) # |01⟩
```

```
        state = np.kron(H, I) @ state
        return CNOT @ state

    @staticmethod
    def psi_minus():
        """Construct |Ψ⁻⟩ = (|01⟩ - |10⟩)/√2"""
        state = np.kron([0, 1], [0, 1]) # |11⟩
        state = np.kron(H, I) @ state
        return CNOT @ state
def partial_trace(rho, dims, axis=0):
    """
    Compute partial trace of density matrix rho
    dims: list of dimensions of each subsystem [dA, dB]
    axis: 0 for tracing out B, 1 for tracing out A
    """
    dA, dB = dims
    if axis == 0: # Trace out B
        rho_reduced = np.zeros((dA, dA), dtype=complex)
        for i in range(dA):
            for j in range(dA):
                for k in range(dB):
                    rho_reduced[i,j] += rho[i*dB + k, j*dB + k]
    else: # Trace out A
        rho_reduced = np.zeros((dB, dB), dtype=complex)
        for i in range(dB):
            for j in range(dB):
                for k in range(dA):
                    rho_reduced[i,j] += rho[k*dB + i, k*dB + j]
    return rho_reduced
def entanglement_entropy(state):
    """
    Calculate entanglement entropy of bipartite state
    Input: state vector or density matrix
    Output: entanglement entropy
    """
    # Convert state to density matrix if it's a state vector
    if state.ndim == 1:
        rho = np.outer(state, state.conj())
    else:
        rho = state

    # Partial trace over subsystem B (assuming 2-qubit system)
    rho_A = partial_trace(rho, [2, 2], axis=1)

    # Compute eigenvalues (using eigh for Hermitian matrices)
    eigvals = np.linalg.eigvalsh(rho_A)

    # Calculate von Neumann entropy
    entropy = 0.0
    for lamda in eigvals:
        if lamda > 1e-10: # avoid log(0)
```

```
                entropy -= lamda * log2(lamda)

        return entropy
    # Example usage
    if __name__ == "__main__":
        # Construct Bell states
        phi_p = BellStates.phi_plus()
        phi_m = BellStates.phi_minus()
        psi_p = BellStates.psi_plus()
        psi_m = BellStates.psi_minus()
        print(f"Bell state |Φ⁺⟩ =", phi_p)
        print(f"Bell state |Φ⁻⟩ =", phi_m)
        print(f"Bell state |Ψ⁺⟩ =", psi_p)
        print(f"Bell state |Ψ⁻⟩ =", psi_m)
        # Verify entanglement entropy (should be 1 for maximally entangled states)
        print(f"Entanglement entropy of |Φ⁺⟩: {entanglement_entropy(phi_p):.4f}")
        print(f"Entanglement entropy of |Φ⁻⟩: {entanglement_entropy(phi_m):.4f}")
        print(f"Entanglement entropy of |Ψ⁺⟩: {entanglement_entropy(psi_p):.4f}")
        print(f"Entanglement entropy of |Ψ⁻⟩: {entanglement_entropy(psi_m):.4f}")

        # Verify product state has zero entanglement entropy
        product_state = np.kron([1, 0], [1, 0]) # |00⟩
        print(f"Entanglement entropy of |00⟩: {entanglement_entropy(product_state):.4f}")
```

```
==================================================
TASK 3: BELL STATES AND ENTANGLEMENT ENTROPY
==================================================
Bell state |Φ⁺⟩ = [0.70710678 0.         0.         0.70710678]
Bell state |Φ⁻⟩ = [ 0.70710678  0.          0.         -0.70710678]
Bell state |Ψ⁺⟩ = [0.         0.70710678 0.70710678 0.        ]
Bell state |Ψ⁻⟩ = [ 0.          0.70710678 -0.70710678  0.        ]
Entanglement entropy of |Φ⁺⟩: 1.0000
Entanglement entropy of |Φ⁻⟩: 1.0000
Entanglement entropy of |Ψ⁺⟩: 1.0000
Entanglement entropy of |Ψ⁻⟩: 1.0000
Entanglement entropy of |00⟩: 0.0000
```

Result:

Bell states were constructed and their entanglement entropy was accurately calculated