Date:

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).

1. Mathematical Model

1.1 Quantum Gates Representation

a. **Hadamard Gate (H)**

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

- Transforms basis states:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

b. **Identity Gate (I)**

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

- Leaves qubit states unchanged.

c. **CNOT Gate**

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Flips the target qubit if the control qubit is $|1\rangle$.

1.2 Bell States Construction

Bell states are constructed by applying H to the first qubit followed by CNOT:

a. $|\Phi^+\rangle$

$$|\Phi^{+}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

- Constructed from $|00\rangle$.

b. $|\Phi^{-}\rangle$

$$|\Phi^{-}\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

- Constructed from $|10\rangle$.

c. $|\Psi^{+}\rangle$

$$|\Psi^{+}\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$

- Constructed from $|01\rangle$.

d. $|\Psi^{-}\rangle$

$$|\Psi^{-}\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

- Constructed from $|11\rangle$.

1.3 Partial Trace Operation

Given a density matrix ρ for a bipartite system $A \otimes B$, the partial trace over subsystem B is

$$\rho_A = \mathrm{Tr}_B(\rho) = \sum_k (I_A \otimes \langle k|_B)\rho(I_A \otimes |k\rangle_B)$$

where $\{|k\rangle B\}$ is a basis for B.

1.4 Entanglement Entropy (Von Neumann Entropy)

For a pure bipartite state $|\psi\rangle AB$, the entanglement entropy is the von Neumann entropy of the reduced density matrix $\rho_A = \mathrm{Tr}_B(|\psi\rangle\langle\psi|)$.

$$S(\rho_A) = -\mathrm{Tr}(\rho_A \log_2 \rho_A) = -\sum_i \lambda_i \log_2 \lambda_i$$

where λi are the eigenvalues of ρA.

2. Algorithm

- Define quantum gates
- Create entangled Bell states using tensor products.
- Reshape the states for partial trace computation.
- Calculate entanglement entropy of bipartite state
- Compute eigenvalues (using eigh for Hermitian matrices)
- Compute von Neumann entropy.

3. Program

```
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"""
```

```python
        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:
```

```python
        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}")
```

4. Result

Bell states were constructed and their entanglement entropy was accurately calculated.