# Date:

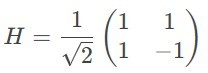
**TASK 3: Bell States and Entanglement Entropy**

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

1. **Construct all four Bell states** (|Φ⁺⟩, |Φ⁻⟩, |Ψ⁺⟩, |Ψ⁻⟩) using quantum gates (Hadamard and CNOT).
2. **Measure their entanglement entropy** to verify that they are maximally entangled (entropy

= 1).

1. **Compare with a product state** (|00⟩) to confirm it has zero entanglement (entropy = 0).
2. **Mathematical Model**
   1. **Quantum Gates Representation**
      1. **Hadamard Gate (H)**



* Transforms basis states:

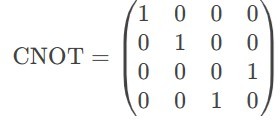


# Identity Gate (I)



* Leaves qubit states unchanged.

# CNOT Gate

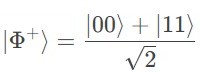


* Flips the target qubit if the control qubit is ∣1⟩.

# Bell States Construction

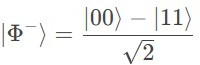
Bell states are constructed by applying *H* to the first qubit followed by CNOT:

**a.** |Φ+⟩



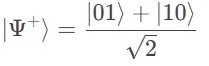
* Constructed from ∣00⟩.

**b.** |Φ–⟩



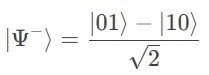
* Constructed from ∣10⟩.

**c.** |Ψ+⟩



* Constructed from ∣01⟩.

d. |Ψ–⟩



* Constructed from ∣11⟩.

# Partial Trace Operation

Given a density matrix *ρ* for a bipartite system *A*⊗*B*, the partial trace over subsystem *B* is



where {∣*k*⟩*B*} is a basis for *B.*

# Entanglement Entropy (Von Neumann Entropy)

For a pure bipartite state ∣*ψ*⟩*AB*, the entanglement entropy is the von Neumann entropy of the reduced density matrix *ρA*=Tr*B*(∣*ψ*⟩⟨*ψ*∣).



where *λi* are the eigenvalues of *ρA*.

# 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.

# 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(np.array([1, 0]), np.array([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(np.array([1, 0]), np.array([1, 0]))  # |00⟩

# state = np.kron(H, I) @ state

# state = CNOT @ state

# state[3] \*= -1  # Flip the sign of |11⟩

# return state

# @staticmethod

# def psi\_plus():

# """Construct |Ψ⁺⟩ = (|01⟩ + |10⟩)/√2"""

# state = np.kron(np.array([1, 0]), np.array([0, 1]))  # |01⟩

# state = np.kron(H, I) @ state

# return CNOT @ state

# @staticmethod

# def psi\_minus():

# """Construct |Ψ⁻⟩ = (|01⟩ - |10⟩)/√2"""

# state = np.kron(np.array([1, 0]), np.array([0, 1]))  # |01⟩

# state = np.kron(H, I) @ state

# state = CNOT @ state

# state[2] \*= -1  # Flip the sign of |10⟩

# return 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

# """

# if state.ndim == 1:

# rho = np.outer(state, state.conj())

# else:

# rho = state

# rho\_A = partial\_trace(rho, [2, 2], axis=1)

# eigvals = np.linalg.eigvalsh(rho\_A)

# entropy = 0.0

# for lamda in eigvals:

# if lamda > 1e-10:

# 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"\nBell 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}")

# # Entanglement entropy

# print(f"\nEntanglement 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}")

# # Product state

# product\_state = np.kron(np.array([1, 0]), np.array([1, 0]))  # |00⟩

# print(f"Entanglement entropy of |00⟩: {entanglement\_entropy(product\_state):.4f}")

# Output:

# 