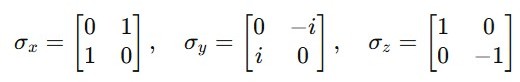
# Date:

**TASK 4: Commutation Relations and Euler Decomposition**

**Aim:** To verify Pauli matrix commutation relations and decompose a gate using Euler angles.

1. Verify the fundamental commutation and anti-commutation relations of Pauli matrices (X, Y, Z)
2. Implement and validate Z-Y-Z Euler angle decomposition for arbitrary single-qubit gates
3. Demonstrate the decomposition on standard quantum gates (X, Y, Z, H, S, T) and Cirq operations
4. **Mathematical Model**
   1. **Pauli Matrix Commutation & Anti-Commutation Relations**
      * **Pauli Matrices**

****

* + - **Commutator**

The commutator of two operators A,B is



For Pauli matrices



where ϵijk is the Levi-Civita symbol.

Example:

# Anti-Commutator

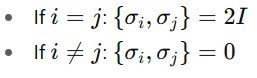
The anti-commutator is



For Pauli matrices



where δij is the Kronecker delta and I is the 2×2 identity. This means



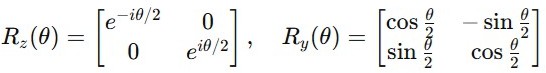
# Z–Y–Z Euler Decomposition for Single-Qubit Gates

Any single-qubit unitary U∈SU(2) can be written (up to a global phase eiϕ) as:



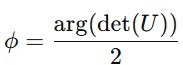
where

* + - ϕ is a global phase.
    - α,β,γ are rotation angles.
    - Rotation operators



# Decomposition steps:

1. Extract global phase



1. Remove the global phase



1. From U0, solve for β using



1. Solve for α and γ from the argument (phase) of elements U00 and U01.

# Algorithm

* 1. **Pauli Matrix Verification**

1. Symbolically define Pauli matrices using SymPy
2. Compute commutators [A,B] = AB-BA and verify [σi,σj] = 2iεijkσk
3. Compute anti-commutators {A,B} = AB+BA and verify {σi,σj} = 2δijI

# Z-Y-Z Decomposition

1. Check matrix unitarity: UTU = I
2. Extract global phase from determinant
3. Solve for Euler angles (α, β, γ) in:
4. U = e^iφ Rz(α)Ry(β)Rz(γ)
5. Handle special cases when β ≈ 0 or π
6. Reconstruct matrix to validate decomposition

# Testing

1. Standard gates: X, Y, Z, Hadamard (H), Phase (S), π/8 (T)
2. Random unitary matrices
3. Optional Cirq integration for hardware verification

**3.PROGRAM**

import numpy as np

import cmath

print("\n" + "="\*50)

print("TASK 4: COMMUTATION RELATIONS AND EULER ANGLES")

print("="\*50)

# --- Part 1: Verify Pauli commutation & anti-commutation with SymPy ---

import sympy as sp

I = sp.eye(2)

sx = sp.Matrix([[0, 1], [1, 0]])

sy = sp.Matrix([[0, -sp.I], [sp.I, 0]])

sz = sp.Matrix([[1, 0], [0, -1]])

paulis = {'X': sx, 'Y': sy, 'Z': sz}

def comm(A, B):

    return sp.simplify(A \* B - B \* A)

def anti(A, B):

    return sp.simplify(A \* B + B \* A)

print("\n=== Commutation relations ===")

for (a, b, k) in [('X', 'Y', 'Z'), ('Y', 'Z', 'X'), ('Z', 'X', 'Y')]:

    lhs = comm(paulis[a], paulis[b])

    rhs = 2 \* sp.I \* paulis[k]

    print(f"[{a},{b}] =\n{lhs}\nExpected:\n{rhs}\n")

print("\n=== Anti-commutation relations ===")

for i in ['X', 'Y', 'Z']:

    for j in ['X', 'Y', 'Z']:

        lhs = anti(paulis[i], paulis[j])

        rhs = 2 \* (1 if i == j else 0) \* I

        print(f"{{{i},{j}}} =\n{lhs}  Expected:\n{rhs}\n")

# --- Part 2: Z–Y–Z Euler decomposition ---

def is\_unitary(U, tol=1e-8):

    return np.allclose(U.conj().T @ U, np.eye(2), atol=tol)

def decompose\_zyz(U, tol=1e-8):

    """Return (phi, alpha, beta, gamma) such that

       U = e^{i phi} Rz(alpha) Ry(beta) Rz(gamma)

    """

    U = np.array(U, dtype=complex)

    if not is\_unitary(U):

        raise ValueError("Matrix is not unitary.")

    detU = np.linalg.det(U)

    phi = cmath.phase(detU) / 2

    U0 = U \* np.exp(-1j \* phi)

    detU0 = np.linalg.det(U0)

    U0 = U0 / np.sqrt(detU0)

    a = U0[0, 0]

    b = U0[0, 1]

    beta = 2 \* np.arccos(min(1.0, max(0.0, abs(a))))

    if np.isclose(np.sin(beta / 2), 0, atol=tol):

        alpha = 2 \* (-cmath.phase(a))

        gamma = 0.0

    else:

        phi1 = -cmath.phase(a)

        phi2 = -cmath.phase(-b)

        alpha = phi1 + phi2

        gamma = phi1 - phi2

    return float(phi), float(alpha), float(beta), float(gamma)

def Rz(theta):

    return np.array([[np.exp(-1j \* theta / 2), 0],

                     [0, np.exp(1j \* theta / 2)]], dtype=complex)

def Ry(theta):

    return np.array([[np.cos(theta / 2), -np.sin(theta / 2)],

                     [np.sin(theta / 2), np.cos(theta / 2)]],

dtype=complex)

def reconstruct(phi, alpha, beta, gamma):

    return np.exp(1j \* phi) @ (Rz(alpha) @ Ry(beta) @ Rz(gamma))

# --- Part 3: Test examples ---

def Rx(theta):

    return np.cos(theta / 2) \* np.eye(2) - 1j \* np.sin(theta / 2) \* sx

examples = {

    "Rx(pi/3)": Rx(np.pi / 3),

    "Ry(pi/4)": Ry(np.pi / 4),

    "Rz(pi/2)": Rz(np.pi / 2),

    "H": (1 / np.sqrt(2)) \* np.array([[1, 1], [1, -1]],

dtype=complex),

    "S": np.array([[1, 0], [0, 1j]], dtype=complex),

    "T": np.array([[1, 0], [0, np.exp(1j \* np.pi / 4)]],

dtype=complex),

}

print("\n=== Z–Y–Z Euler Decomposition ===")

for name, U in examples.items():

    phi, alpha, beta, gamma = decompose\_zyz(U)

    print(f"{name}:\n  φ={phi:.6f}, α={alpha:.6f}, β={beta:.6f}, γ={gamma:.6f}\n")

# Optional: Use Cirq if available

try:

    import cirq

    print("\nCirq example decomposition for H gate:")

    # Create a qubit and turn H into an operation

    q = cirq.LineQubit(0)

    H\_op = cirq.H(q)

    # Extract the unitary matrix of H

    U = cirq.unitary(H\_op)

    # Perform Z–Y–Z decomposition

    phi, alpha, beta, gamma = decompose\_zyz(U)

    print(f"Cirq H: φ={phi:.6f}, α={alpha:.6f}, β={beta:.6f}, γ={gamma:.6f}")

except ImportError:

    print("\nCirq not installed. Skipping Cirq exa”)

OUTPUT:

