

Date: 12/08/25

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

1. Mathematical Model

1.1 Pauli Matrix Commutation & Anti-Commutation Relations

- Pauli Matrices

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

- Commutator

The commutator of two operators A,B is

$$[A, B] = AB - BA$$

For Pauli matrices

$$[\sigma_i, \sigma_j] = 2i \epsilon_{ijk} \sigma_k$$

where ϵ_{ijk} is the Levi-Civita symbol.

$$[\sigma_x, \sigma_y] = \sigma_x \sigma_y - \sigma_y \sigma_x = 2i \sigma_z$$

Example:

- Anti-Commutator

The anti-commutator is

$$\{A, B\} = AB + BA$$

For Pauli matrices

$$\{\sigma_i, \sigma_j\} = 2\delta_{ij} I$$

where δ_{ij} is the Kronecker delta and I is the 2×2 identity.

This means

- If $i = j$: $\{\sigma_i, \sigma_j\} = 2I$
- If $i \neq j$: $\{\sigma_i, \sigma_j\} = 0$

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

Any single-qubit unitary $U \in \text{SU}(2)$ can be written (up to a global phase $e^{i\phi}$) as:

$$U = e^{i\phi} R_z(\alpha) R_y(\beta) R_z(\gamma)$$

where

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

$$R_z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

- Decomposition steps:

1. Extract global phase

$$\phi = \frac{\arg(\det(U))}{2}$$

2. Remove the global phase

$$U_0 = U e^{-i\phi}$$

3. From U_0 , solve for β using

$$\beta = 2 \cos^{-1}(|U_{00}|)$$

4. Solve for α and γ from the argument (phase) of elements U_{00} and U_{01} .

2. Algorithm

2.1 Pauli Matrix Verification

- Symbolically define Pauli matrices using SymPy
- Compute commutators $[A, B] = AB - BA$ and verify $[\sigma_i, \sigma_j] = 2i\epsilon_{ijk}\sigma_k$
- Compute anti-commutators $\{A, B\} = AB + BA$ and verify $\{\sigma_i, \sigma_j\} = 2\delta_{ij}I$

2.2 Z-Y-Z Decomposition

- Check matrix unitarity: $U^T U = I$
- Extract global phase from determinant
- Solve for Euler angles (α, β, γ) in:
- $U = e^{i\phi} R_z(\alpha) R_y(\beta) R_z(\gamma)$
- Handle special cases when $\beta \approx 0$ or π
- Reconstruct matrix to validate decomposition

2.3 Testing

- Standard gates: X, Y, Z, Hadamard (H), Phase (S), $\pi/8$ (T)
- Random unitary matrices
- 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"\n{{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"\n{name}: 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 examples.")

```

Output:

=====

TASK 4: COMMUTATION RELATIONS AND EULER ANGLES

=====

== Commutation relations ==

[X,Y] =

Matrix([[2*I, 0], [0, -2*I]])

Expected:

Matrix([[2*I, 0], [0, -2*I]])

[Y,Z] =

Matrix([[0, 2*I], [2*I, 0]])

Expected:

Matrix([[0, 2*I], [2*I, 0]])

[Z,X] =

Matrix([[0, 2], [-2, 0]])

Expected:

Matrix([[0, 2], [-2, 0]])

== Anti-commutation relations ==

{X,X} =

Matrix([[2, 0], [0, 2]])

Expected:

Matrix([[2, 0], [0, 2]])

{X,Y} =

Matrix([[0, 0], [0, 0]])

Expected:

Matrix([[0, 0], [0, 0]])

{X,Z} =

Matrix([[0, 0], [0, 0]])

Expected:

Matrix([[0, 0], [0, 0]])

{Y,X} =

Matrix([[0, 0], [0, 0]])

Expected:

Matrix([[0, 0], [0, 0]])

{Y,Y} =
Matrix([[2, 0], [0, 2]])

Expected:
Matrix([[2, 0], [0, 2]])
{Y,Z} =
Matrix([[0, 0], [0, 0]])

Expected:
Matrix([[0, 0], [0, 0]])
{Z,X} =
Matrix([[0, 0], [0, 0]])

Expected:
Matrix([[0, 0], [0, 0]])
{Z,Y} =
Matrix([[0, 0], [0, 0]])

Expected:
Matrix([[0, 0], [0, 0]])
==== Z-Y-Z Euler Decomposition ====
Rx(pi/3):

$\phi=0.000000, \alpha=-1.570796, \beta=1.047198, \gamma=1.570796$

Ry(pi/4):
 $\phi=0.000000, \alpha=0.000000, \beta=0.785398, \gamma=-0.000000$

Rz(pi/2):
 $\phi=0.000000, \alpha=1.570796, \beta=0.000000, \gamma=0.000000$

H:
 $\phi=1.570796, \alpha=0.000000, \beta=1.570796, \gamma=3.141593$

S:
 $\phi=0.785398, \alpha=1.570796, \beta=0.000000, \gamma=0.000000$

T:
 $\phi=0.392699, \alpha=0.785398, \beta=0.000000, \gamma=0.000000$

Cirq example decomposition for H gate:

Cirq H: $\phi=1.570796$, $\alpha=0.000000$, $\beta=1.570796$, $\gamma=3.141593$

4. Result

Commutation properties and Euler angle decomposition were successfully demonstrated.