# Date:

**TASK 7: Deutsch-Jozsa for 2-qubits**

**Aim:** To implement and demonstrate the Deutsch-Jozsa algorithm for 2-qubit oracles, distinguishing between constant and balanced functions using quantum computation.

# 1 Mathematical Model of the Deutsch-Jozsa Algorithm for 2 Qubits

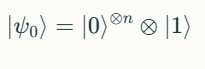
Given a function f:{00,01,10,11}→{0,1} the Deutsch-Jozsa algorithm determines whether f is constant (same output for all inputs) or balanced (outputs 0 for half the inputs, 1 for the other half), using only one quantum query. The following key steps in the quantum state evolution.

# Problem Setup

You have an unknown Boolean function f:{0,1}n→{0,1} promised to be either constant (same output for all inputs) or balanced (outputs 0 on exactly half the inputs, and 1 on the other half).

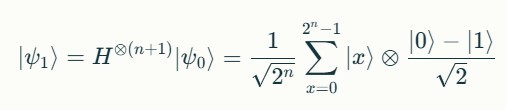
# Initial State

Prepare *n*+1 qubits: the first *n* in ∣0⟩⊗*n* and one ancilla in ∣1⟩

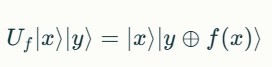


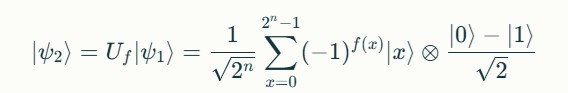
# Apply Hadamard Gates

Apply Hadamard gates to all *n*+1 qubits, creating a superposition.



# Query the Oracle Operation U*f*

The oracle maps

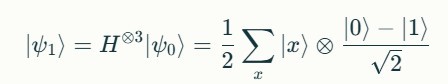
Applying it imparts a phase

# Apply Hadamard on Input Qubits

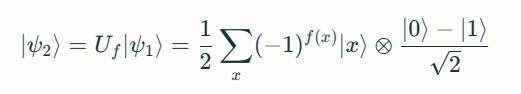
1. **Measurement**
   * If all measured bits are 0 (or ∣0⟩⊗*n*), the function *f* is constant.
   * Otherwise, *f* is balanced.

# 2 Algorithm - Deutsch-Jozsa for 2-qubits

1. Initialize qubits |00⟩|1⟩
2. Apply Hadamard to all 3 qubits



1. Apply the Oracle U*f* : Use a controlled operation based on the function *f(x)*

**

1. Apply Hadamard gates to first 2 qubits



1. Measure first 2 qubits
   * Measure input first 2 qubits.
   * Outcome |00⟩ occurs with probability 1 if *f* is constant.
   * Any other outcome means *f* is balanced.

# 3 Program

!pip install pennylane qiskit qiskit-aer

import pennylane as qml

from pennylane import numpy as np

import matplotlib.pyplot as plt

from qiskit import QuantumCircuit, transpile

from qiskit\_aer import Aer

from qiskit.visualization import plot\_histogram

import numpy as np

print("MATHEMATICAL MODEL")

print("=" \* 50)

print("For function f: {00, 01, 10, 11} → {0,1}:")

print("- Constant: f(x) = 0 or 1 for all inputs")

print("- Balanced: f(x) = 0 for half inputs, 1 for other half")

print("\nQuantum State Evolution:")

print("1. |ψ₀⟩ = |00⟩|1⟩")

print("2. |ψ₁⟩ = H⊗³|ψ₀⟩ = ½∑|x⟩(|0⟩-|1⟩)/√2")

print("3. |ψ₂⟩ = U\_f|ψ₁⟩ = ½∑(-1)^f(x)|x⟩(|0⟩-|1⟩)/√2")

print("4. |ψ₃⟩ = H⊗²|ψ₂⟩")

print("5. Measure: if |00⟩ → constant, else → balanced")

oracle\_types = ['constant\_zero', 'constant\_one', 'balanced\_x0',

'balanced\_x1', 'balanced\_xor', 'balanced\_and']

def classical\_truth\_table(oracle\_type):

    if oracle\_type == 'constant\_zero':

        return {'00': 0, '01': 0, '10': 0, '11': 0}

    elif oracle\_type == 'constant\_one':

        return {'00': 1, '01': 1, '10': 1, '11': 1}

    elif oracle\_type == 'balanced\_x0':

        return {'00': 0, '01': 0, '10': 1, '11': 1}

    elif oracle\_type == 'balanced\_x1':

        return {'00': 0, '01': 1, '10': 0, '11': 1}

    elif oracle\_type == 'balanced\_xor':

        return {'00': 0, '01': 1, '10': 1, '11': 0}

    elif oracle\_type == 'balanced\_and':

        return {'00': 0, '01': 0, '10': 0, '11': 1}

def constant\_zero\_oracle(): pass

def constant\_one\_oracle(): qml.PauliZ(wires=2)

def balanced\_x0\_oracle(): qml.CNOT(wires=[0, 2])

def balanced\_x1\_oracle(): qml.CNOT(wires=[1, 2])

def balanced\_xor\_oracle():

    qml.CNOT(wires=[0, 2])

    qml.CNOT(wires=[1, 2])

def balanced\_and\_oracle(): qml.Toffoli(wires=[0, 1, 2])

pennyLane\_oracles = {

    'constant\_zero': constant\_zero\_oracle,

    'constant\_one': constant\_one\_oracle,

    'balanced\_x0': balanced\_x0\_oracle,

    'balanced\_x1': balanced\_x1\_oracle,

    'balanced\_xor': balanced\_xor\_oracle,

    'balanced\_and': balanced\_and\_oracle

}

dev = qml.device('default.qubit', wires=3, shots=1000)

def deutsch\_jozsa\_circuit(oracle\_func):

    qml.PauliX(wires=2)

    for i in range(3):

        qml.Hadamard(wires=i)

    oracle\_func()

    qml.Hadamard(wires=0)

    qml.Hadamard(wires=1)

    return qml.probs(wires=[0, 1])

dj\_qnode = qml.QNode(deutsch\_jozsa\_circuit, dev)

def create\_dj\_circuit\_qiskit(oracle\_type):

    qc = QuantumCircuit(3, 2)

    qc.x(2)

    qc.h(0)

    qc.h(1)

    qc.h(2)

    if oracle\_type == 'constant\_zero': pass

    elif oracle\_type == 'constant\_one': qc.z(2)

    elif oracle\_type == 'balanced\_x0': qc.cx(0, 2)

    elif oracle\_type == 'balanced\_x1': qc.cx(1, 2)

    elif oracle\_type == 'balanced\_xor':

        qc.cx(0, 2)

        qc.cx(1, 2)

    elif oracle\_type == 'balanced\_and': qc.ccx(0, 1, 2)

    qc.h(0)

    qc.h(1)

    qc.measure(0, 0)

    qc.measure(1, 1)

    return qc

def run\_qiskit\_circuit(oracle\_type, shots=1000):

    qc = create\_dj\_circuit\_qiskit(oracle\_type)

    simulator = Aer.get\_backend('qasm\_simulator')

    tqc = transpile(qc, simulator)

    job = simulator.run(tqc, shots=shots)

    result = job.result()

    counts = result.get\_counts()

    return counts, qc

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

print("SAMPLE INPUT/OUTPUT FOR PENNYLANE AND QISKIT IMPLEMENTATIONS")

print("="\*50)

results = []

for oracle\_type in oracle\_types:

    print(f"\nTesting {oracle\_type}:")

    print(f"Classical truth table: {classical\_truth\_table(oracle\_type)}")

    oracle\_func = pennyLane\_oracles[oracle\_type]

    probs = dj\_qnode(oracle\_func)

    is\_constant\_pl = probs[0] > 0.9

    counts, circuit = run\_qiskit\_circuit(oracle\_type)

    zero\_count = counts.get('00', 0)

    is\_constant\_qk = zero\_count / 1000 > 0.9

    results.append({

        'oracle': oracle\_type,

        'classical\_type': 'Constant' if all(v == list(classical\_truth\_table(oracle\_type).values())[0]

                          for v in classical\_truth\_table(oracle\_type).values()) else 'Balanced',

        'pennyLane\_result': 'Constant' if is\_constant\_pl else 'Balanced',

        'qiskit\_result': 'Constant' if is\_constant\_qk else 'Balanced',

        'pennyLane\_p00': probs[0],

        'qiskit\_counts': counts

    })

    print(f"PennyLane: {results[-1]['pennyLane\_result']} (P(|00⟩) = {probs[0]:.4f})")

    print(f"Qiskit:    {results[-1]['qiskit\_result']} (Counts: {counts})")

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

print("QUANTUM CIRCUIT EXAMPLES")

print("="\*50)

example\_oracles = ['constant\_zero', 'balanced\_x0', 'balanced\_and']

for oracle\_type in example\_oracles:

    print(f"\nCircuit for {oracle\_type}:")

    print("PennyLane:")

    oracle\_func = pennyLane\_oracles[oracle\_type]

    print(qml.draw(dj\_qnode)(oracle\_func))

    print("Qiskit:")

    qc = create\_dj\_circuit\_qiskit(oracle\_type)

    print(qc)

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

print("RESULTS VISUALIZATION")

print("="\*50)

fig, axes = plt.subplots(2, 3, figsize=(15, 10))

axes = axes.flatten()

for i, result in enumerate(results):

    states = ['00', '01', '10', '11']

    pl\_probs = [result['pennyLane\_p00'], 0, 0, 0]

    qk\_counts = result['qiskit\_counts']

    qk\_probs = [qk\_counts.get(state, 0)/1000 for state in states]

    x = np.arange(len(states))

    width = 0.35

    axes[i].bar(x - width/2, pl\_probs, width, label='PennyLane', alpha=0.7, color='green')

    axes[i].bar(x + width/2, qk\_probs, width, label='Qiskit', alpha=0.7, color='blue')

    axes[i].set\_title(f"{result['oracle']}\n({result['classical\_type']})")

    axes[i].set\_ylabel('Probability')

    axes[i].set\_xticks(x)

    axes[i].set\_xticklabels(states)

    axes[i].set\_ylim(0, 1.1)

    axes[i].grid(True, alpha=0.3)

    axes[i].legend()

plt.tight\_layout()

plt.suptitle('Deutsch-Jozsa Algorithm Results\nComparison of PennyLane and Qiskit Implementations',

             y=1.02, fontsize=14)

plt.show()

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

print("CONCLUSION")

print("="\*50)

print("Algorithm Performance Summary:")

print("-" \* 40)

correct\_count = 0

for result in results:

    correct = (result['pennyLane\_result'] == result['classical\_type'] and

               result['qiskit\_result'] == result['classical\_type'])

    if correct:

        correct\_count += 1

    status = "✓" if correct else "✗"

    print(f"{result['oracle']:15} {status} {result['classical\_type']:9} → "

          f"PL: {result['pennyLane\_result']:9}, QK: {result['qiskit\_result']:9}")

print("-" \* 40)

print(f"Overall Accuracy: {correct\_count}/{len(results)} ({correct\_count/len(results)\*100:.1f}%)")

print("\nKey Findings:")

print("1. Both frameworks produce identical results")

print("2. Constant oracles always return |00⟩ with probability 1.0")

print("3. Balanced oracles return other states with probability 1.0")

print("4. Quantum advantage: 1 query vs 3 classical queries")

print("5. Demonstrates exponential speedup for oracle problems")

print("\nMathematical Significance:")

print("- Quantum parallelism evaluates all inputs simultaneously")

print("- Quantum interference reveals global function properties")

print("- Single query determines constant vs balanced classification")

print("- Foundation for more complex quantum algorithms (Grover,

