Double-click (or enter) to edit

Date : 2/9/25

```
                    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.

Algorithm - Deutsch-Jozsa for 2-qubits

1. Initialize qubits |00⟩|1⟩
2. Apply Hadamard to all 3 qubits
3. Apply the Oracle Uf : Use a controlled operation based on the function f(x)
4. Apply Hadamard gates to first 2 qubits
5. Measure first 2 qubits

```
 !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  # Import Aer from qiskit_aer
from qiskit.visualization import plot_histogram

# ==================== MATHEMATICAL MODEL ====================
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 DEFINITIONS ====================
oracle_types = [
    'constant_zero',
    'constant_one',
    'balanced_x0',
    'balanced_x1',
    'balanced_xor',
    'balanced_and'
]

def classical_truth_table(oracle_type):
    """Return classical truth table for verification"""
    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}

# ==================== PENNYLANE IMPLEMENTATION ===================
# Oracle functions for PennyLane
def constant_zero_oracle():
    # Does nothing; identity oracle
    pass

def constant_one_oracle():
    # Apply Z on ancilla to flip phase (equivalent to multiplying by -1)
```

```python
    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
}

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

def deutsch_jozsa_circuit(oracle_func):
    """Deutsch-Jozsa algorithm implementation"""
    # 1. Initialize |00⟩|1⟩
    qml.PauliX(wires=2)
    # 2. Apply Hadamard to all qubits
    for i in range(3):
        qml.Hadamard(wires=i)
    # 3. Apply oracle U_f
    oracle_func()
    # 4. Apply Hadamard to first 2 qubits
    qml.Hadamard(wires=0)
    qml.Hadamard(wires=1)
    # 5. Measure first 2 qubits probabilities
    return qml.probs(wires=[0, 1])

dj_qnode = qml.QNode(deutsch_jozsa_circuit, dev)

# ==================== QISKIT IMPLEMENTATION ====================
def create_dj_circuit_qiskit(oracle_type):
    """Create Deutsch-Jozsa circuit in Qiskit"""
    qc = QuantumCircuit(3, 2)
    # 1. Initialize |00⟩|1⟩
    qc.x(2)
    # 2. Apply Hadamard to all qubits
    qc.h([0, 1, 2])
    # 3. Apply oracle U_f
    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)
    # 4. Apply Hadamard to first 2 qubits
    qc.h([0, 1])
    # 5. Measure first 2 qubits
    qc.measure(0, 0)
    qc.measure(1, 1)
    return qc

def run_qiskit_circuit(oracle_type, shots=1000):
    """Run Qiskit circuit"""
    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()
```

```python
    result = job.result()
    counts = result.get_counts()
    return counts, qc

# ==================== SAMPLE INPUT/OUTPUT ====================
print("\n" + "="*50)
print("SAMPLE INPUT/OUTPUT FOR PENNYLANE AND QISKIT IMPLEMENTATIONS")
print("="*50)
print("Sample Input: Testing all 6 oracle types")
print("Expected Output: Constant oracles return |00⟩, balanced return other states")

results = []

for oracle_type in oracle_types:
    print(f"\nTesting {oracle_type}:")
    print(f"Classical truth table: {classical_truth_table(oracle_type)}")
    # PennyLane
    oracle_func = pennyLane_oracles[oracle_type]
    probs = dj_qnode(oracle_func)
    is_constant_pl = probs[0] > 0.9  # probability of |00> > 0.9

    # Qiskit
    counts, circuit = run_qiskit_circuit(oracle_type)
    zero_count = counts.get('00', 0)
    is_constant_qk = zero_count / 1000 > 0.9

    # Determine classical function type (constant or balanced)
    classical_values = list(classical_truth_table(oracle_type).values())
    is_constant_classical = all(v == classical_values[0] for v in classical_values)

    results.append({
        'oracle': oracle_type,
        'classical_type': 'Constant' if is_constant_classical 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})")

# ==================== CIRCUIT VISUALIZATION ====================
print("\n" + "="*50)
print("QUANTUM CIRCUIT EXAMPLES")
print("="*50)

# Show circuits for different oracle types
example_oracles = ['constant_zero', 'balanced_x0', 'balanced_and']
for oracle_type in example_oracles:
    print(f"\nCircuit for {oracle_type}:")
    # PennyLane circuit
    print("PennyLane:")
    oracle_func = pennyLane_oracles[oracle_type]
    print(qml.draw(dj_qnode)(oracle_func))
    # Qiskit circuit
    print("Qiskit:")
    qc = create_dj_circuit_qiskit(oracle_type)
    print(qc)

# ==================== VISUALIZATION ====================
print("\n" + "="*50)
print("RESULTS VISUALIZATION")
print("="*50)

# Plot results
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.flatten()

for i, result in enumerate(results):
    states = ['00', '01', '10', '11']
    # PennyLane probabilities simplified: only |00> is significant output
    pl_probs = [result['pennyLane_p00'], 0, 0, 0]

    # Qiskit counts normalized
    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
```

```python
    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()

# ==================== CONCLUSION ====================
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 close to 1.0")
print("3. Balanced oracles return other states with probability close to 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, Simon)")
```

```
Collecting pennylane
  Downloading pennylane-0.43.0-py3-none-any.whl.metadata (11 kB)
Collecting qiskit
  Downloading qiskit-2.2.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (12 kB)
Collecting qiskit-aer
  Downloading qiskit_aer-0.17.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.3 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from pennylane) (1.16.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from pennylane) (3.5)
Collecting rustworkx>=0.14.0 (from pennylane)
  Downloading rustworkx-0.17.1-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (10 kB)
Requirement already satisfied: autograd in /usr/local/lib/python3.12/dist-packages (from pennylane) (1.8.0)
Collecting appdirs (from pennylane)
  Downloading appdirs-1.4.4-py2.py3-none-any.whl.metadata (9.0 kB)
Collecting autoray==0.8.0 (from pennylane)
  Downloading autoray-0.8.0-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: cachetools in /usr/local/lib/python3.12/dist-packages (from pennylane) (5.5.2)
Collecting pennylane-lightning>=0.43 (from pennylane)
  Downloading pennylane_lightning-0.43.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (11 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from pennylane) (2.32.4)
Requirement already satisfied: tomlkit in /usr/local/lib/python3.12/dist-packages (from pennylane) (0.13.3)
Requirement already satisfied: typing_extensions in /usr/local/lib/python3.12/dist-packages (from pennylane) (4.15.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from pennylane) (25.0)
Collecting diastatic-malt (from pennylane)
  Downloading diastatic_malt-2.15.2-py3-none-any.whl.metadata (2.6 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from pennylane) (2.0.2)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.12/dist-packages (from qiskit) (0.3.8)
Collecting stevedore>=3.0.0 (from qiskit)
  Downloading stevedore-5.5.0-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (5.9.5)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (2.9
Collecting scipy-openblas32>=0.3.26 (from pennylane-lightning>=0.43->pennylane)
  Downloading scipy_openblas32-0.3.30.0.6-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (57 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.1/57.1 kB 1.3 MB/s eta 0:00:00
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.0->qiskit
Requirement already satisfied: astunparse in /usr/local/lib/python3.12/dist-packages (from diastatic-malt->pennylane) (
Requirement already satisfied: gast in /usr/local/lib/python3.12/dist-packages (from diastatic-malt->pennylane) (0.6.0)
Requirement already satisfied: termcolor in /usr/local/lib/python3.12/dist-packages (from diastatic-malt->pennylane) (3
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->penn
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->pennylane) (3.11
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->pennylane)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->pennylane)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse->diastati
Downloading pennylane-0.43.0-py3-none-any.whl (5.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.3/5.3 MB 58.7 MB/s eta 0:00:00
Downloading autoray-0.8.0-py3-none-any.whl (934 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 934.3/934.3 kB 33.5 MB/s eta 0:00:00
Downloading qiskit-2.2.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (8.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.0/8.0 MB 68.5 MB/s eta 0:00:00
Downloading qiskit_aer-0.17.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 64.2 MB/s eta 0:00:00
Downloading pennylane_lightning-0.43.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (2.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 53.4 MB/s eta 0:00:00
Downloading rustworkx-0.17.1-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 49.3 MB/s eta 0:00:00
Downloading stevedore-5.5.0-py3-none-any.whl (49 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.5/49.5 kB 4.2 MB/s eta 0:00:00
Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Downloading diastatic_malt-2.15.2-py3-none-any.whl (167 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 167.9/167.9 kB 12.2 MB/s eta 0:00:00
Downloading scipy_openblas32-0.3.30.0.6-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (8.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.6/8.6 MB 54.2 MB/s eta 0:00:00
Installing collected packages: appdirs, stevedore, scipy-openblas32, rustworkx, autoray, qiskit, diastatic-malt, qiskit
Successfully installed appdirs-1.4.4 autoray-0.8.0 diastatic-malt-2.15.2 pennylane-0.43.0 pennylane-lightning-0.43.0 qi
/usr/local/lib/python3.12/dist-packages/pennylane/__init__.py:209: RuntimeWarning: PennyLane is not yet compatible with
  warnings.warn(
MATHEMATICAL MODEL
==================================================
For function f: {00, 01, 10, 11} → {0,1}:
- Constant: f(x) = 0 or 1 for all inputs
- Balanced: f(x) = 0 for half inputs, 1 for other half

Quantum State Evolution:
1. |ψ₀⟩ = |00⟩|1⟩
2. |ψ₁⟩ = H⊗³|ψ₀⟩ = ½Σ|x⟩(|0⟩-|1⟩)/√2
3. |ψ₂⟩ = U_f|ψ₁⟩ = ½Σ(-1)^f(x)|x⟩(|0⟩-|1⟩)/√2
4. |ψ₃⟩ = H⊗²|ψ₂⟩
5. Measure: if |00⟩ → constant, else → balanced

==================================================
SAMPLE INPUT/OUTPUT FOR PENNYLANE AND QISKIT IMPLEMENTATIONS
==================================================
Sample Input: Testing all 6 oracle types
Expected Output: Constant oracles return |00⟩, balanced return other states

Testing constant_zero:
Classical truth table: {'00': 0, '01': 0, '10': 0, '11': 0}
/usr/local/lib/python3.12/dist-packages/pennylane/devices/device_api.py:193: PennyLaneDeprecationWarning: Setting shots
```

```
/usr/local/lib/python3.11/dist-packages/pennylane/devices/device_api.py:193: PennyLaneDeprecationWarning: Setting shots
  warnings.warn(
PennyLane: Constant (P(|00⟩) = 1.0000)
Qiskit: Constant (Counts: {'00': 1000})

Testing constant_one:
Classical truth table: {'00': 1, '01': 1, '10': 1, '11': 1}
PennyLane: Constant (P(|00⟩) = 1.0000)
Qiskit: Constant (Counts: {'00': 1000})

Testing balanced_x0:
Classical truth table: {'00': 0, '01': 0, '10': 1, '11': 1}
PennyLane: Balanced (P(|00⟩) = 0.0000)
Qiskit: Balanced (Counts: {'01': 1000})

Testing balanced_x1:
Classical truth table: {'00': 0, '01': 1, '10': 0, '11': 1}
PennyLane: Balanced (P(|00⟩) = 0.0000)
Qiskit: Balanced (Counts: {'10': 1000})

Testing balanced_xor:
Classical truth table: {'00': 0, '01': 1, '10': 1, '11': 0}
PennyLane: Balanced (P(|00⟩) = 0.0000)
Qiskit: Balanced (Counts: {'11': 1000})

Testing balanced_and:
Classical truth table: {'00': 0, '01': 0, '10': 0, '11': 1}
PennyLane: Balanced (P(|00⟩) = 0.2750)
Qiskit: Balanced (Counts: {'00': 258, '10': 235, '11': 245, '01': 262})

==================================================
QUANTUM CIRCUIT EXAMPLES
==================================================

Circuit for constant_zero:
PennyLane:
0: ──H──H─┤ ╭Probs
1: ──H──H─┤ ╰Probs
2: ──X──H─┤
Qiskit:
     ┌───┐┌───┐┌─┐
q_0: ┤ H ├┤ H ├┤M├───
     ├───┤├───┤└╥┘┌─┐
q_1: ┤ H ├┤ H ├─╫─┤M├
     ├───┤├───┤ ║ └╥┘
q_2: ┤ X ├┤ H ├─╫──╫─
     └───┘└───┘ ║  ║
c: 2/═══════════╩══╩═
                0  1

Circuit for balanced_x0:
PennyLane:
0: ──H────╭●──H─┤ ╭Probs
1: ──H────│───H─┤ ╰Probs
2: ──X──H─╰X────┤
Qiskit:
     ┌───┐          ┌───┐┌─┐
q_0: ┤ H ├───────■──┤ H ├┤M├
     ├───┤┌───┐  │  └┬─┬┘└╥┘
q_1: ┤ H ├┤ H ├──┼───┤M├──╫─
     ├───┤├───┤┌─┴─┐ └╥┘  ║
q_2: ┤ X ├┤ H ├┤ X ├──╫───╫─
     └───┘└───┘└───┘  ║   ║
c: 2/═════════════════╩═══╩═
```