Programming Quantum Algorithms with Python and Qiskit

Quantum Computing

Bhattaraprot Bhabhatsatam, Ph.D.

Objectives

- **Write a Quantum Circuit in Python Using Qiskit**
  - Understand the structure and components of Qiskit Python code.
- **Implement Quantum Algorithms in Python**
  - Learn the modular approach to designing algorithms using Qiskit.
- **Simulate the Circuit Locally**
  - Utilize Qiskit simulators to verify circuit behavior.
- **Execute the Circuit on the IBM Quantum Cloud**
  - Send your code to IBM Quantum systems for real execution
- **Explore Real-World Examples**
  - **Maze Solver**: Solve mazes efficiently using quantum algorithms.
  - **Non-Convex Optimization**: Address challenging optimization problems with quantum techniques.

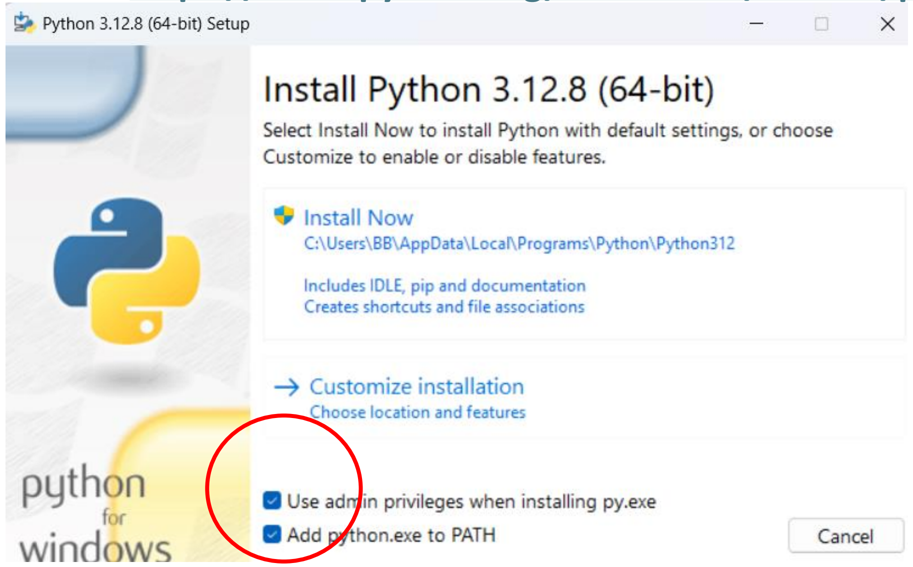
Lab 3-1 - Setup Environment

- Ensure your PC is ready for programming quantum circuits with Python and Qiskit.
- Install necessary software and libraries for local simulation and cloud access.

Lab 3-1 - Setup Environment

1. Install Python 3.12.8 (64-bit)

- https://www.python.org/downloads/release/python-3128/



Python 3.12.8 (64-bit) Setup

Install Python 3.12.8 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\BB\AppData\Local\Programs\Python\Python312

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

2.1 Open terminal as administrator

2.2 curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

2.3 python get-pip.py

3. Create virtual environment and activate

3.1 open terminal then cd c:\
3.2 mkdir qlabs
3.3 cd c:\qlabs
3.4 python -m venv qenv
3.5 cd qenv
3.6 Scripts\activate
3.7 download requirement file from www.bhattaraprot.com/requirements.txt save this file in c:\qlabs\qenv
3.8 pip install –r c:\qlabs\qenv\requirement.txt

4. Create src folder

4.1 cd c:\qlabs\qenv\
4.2 mkdir src
4.3 cd src
4.4 download sourcecode from www.bhattaraprot.com/source.zip
4.5 unzip source.zip

Skeleton of Qiskit Programming

- **Imports**: Necessary libraries.
- **Qubit Setup**: Initializing quantum and classical bits.
- **Quantum Gates**: Applying gates to manipulate qubits.
- **Circuit Visualization**: Drawing the circuit.
- **Measurement**: Measuring qubits into classical bits.

Lab 3-2 - Skeleton

Script sc3.py

```
# Import necessary Qiskit libraries
from qiskit import QuantumCircuit, transpile, assemble
from qiskit_aer import Aer, AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

# Step 1: Create a Quantum Circuit with 2 qubits and 2 classical bits
qc = QuantumCircuit(2, 2)

# Step 2: Apply a Hadamard gate to qubit 0 (superposition)
qc.h(0)

# Step 3: Apply a CNOT gate with qubit 0 as control and qubit 1 as target (entanglement)
qc.cx(0, 1)

# Step 4: Measure the qubits and store the results in classical bits
qc.measure([0, 1], [0, 1])

# Step 5:For execution
simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
sim_result = simulator.run(compiled_circuit).result()
counts = sim_result.get_counts()

print(qc)

qc.draw(output='mpl')
plt.show()

# Step 6: Get and display the result as a histogram
print("Result:", counts)
plot_histogram(counts)
plt.show()
```

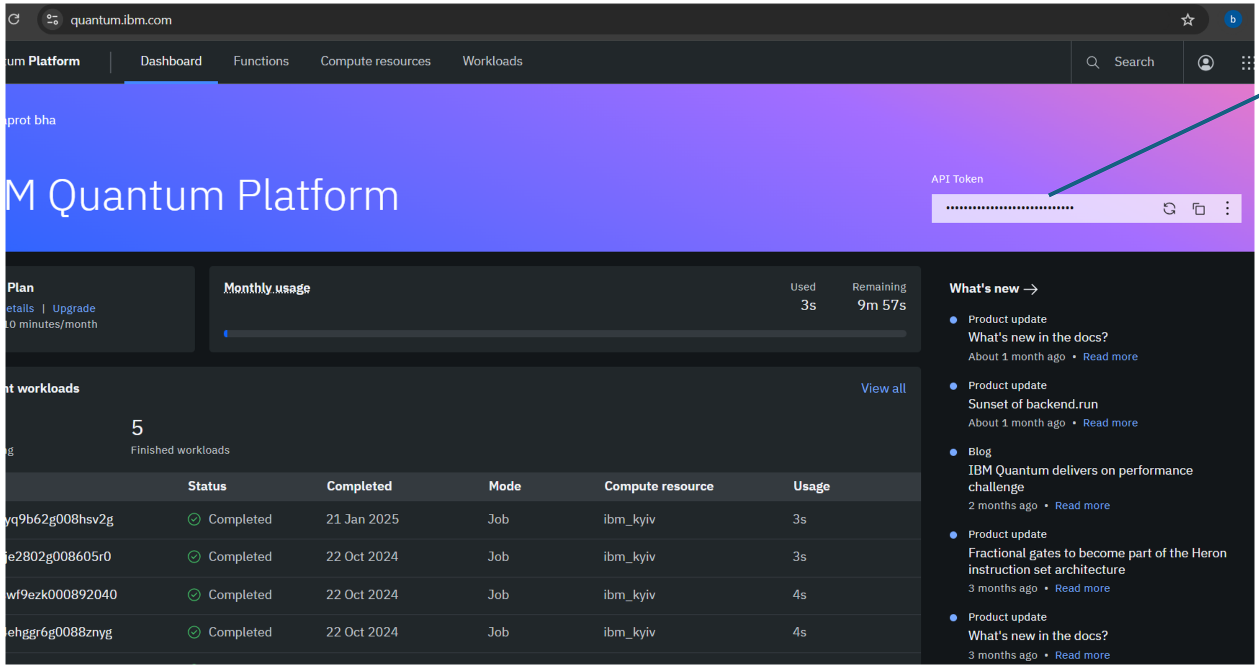
Steps to Execute on IBM Server

- After testing circuits on a local simulator, you can run them on IBM’s **real quantum hardware**.
- IBM provides access to real quantum computers through their **IBM Quantum Cloud**.
- Running on real hardware introduces practical quantum computing challenges, such as noise and limited qubit availability.

Lab 3-3 IBM Server

Script sc5.py

Script sc6.py


quantum.ibm.com

um Platform | Dashboard | Functions | Compute resources | Workloads | Search

prot bha

M Quantum Platform

API Token

Plan
etails | Upgrade
0 minutes/month

Monthly usage

Used 3s
Remaining 9m 57s

What's new →

Product update
What's new in the docs?
About 1 month ago • Read more

Product update
Sunset of backend.run
About 1 month ago • Read more

Blog
IBM Quantum delivers on performance challenge
2 months ago • Read more

Product update
Fractional gates to become part of the Heron instruction set architecture
3 months ago • Read more

Product update
What's new in the docs?
3 months ago • Read more

t workloads | View all

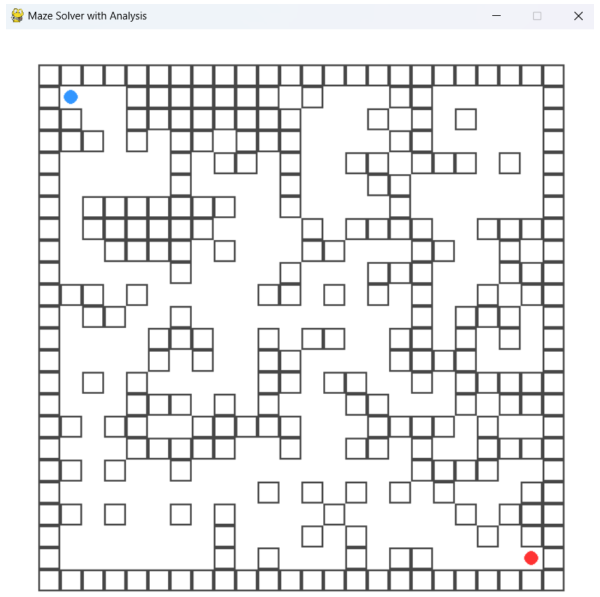
5
Finished workloads

	Status	Completed	Mode	Compute resource	Usage
yq9b62g008hsv2g	Completed	21 Jan 2025	Job	ibm_kyiv	3s
e2802g008605r0	Completed	22 Oct 2024	Job	ibm_kyiv	3s
wf9ezk000892040	Completed	22 Oct 2024	Job	ibm_kyiv	4s
ehggr6g0088znyg	Completed	22 Oct 2024	Job	ibm_kyiv	4s

```
from qiskit import QuantumCircuit, transpile, assemble
from qiskit_ibm_runtime import QiskitRuntimeService,
SamplerV2 as Sampler
from qiskit.visualization import plot_histogram

# Step 1: Initialize the Qiskit Runtime Service with the toker
service = QiskitRuntimeService(channel="ibm_quantum",
token="xxxxxxxxxx")
# Step 2: Create a simple quantum circuit
qc = QuantumCircuit(2, 2)
qc.measure([0, 1], [0, 1])  # Measure both qubits

backend = service.least_busy(operational=True,
simulator=False)

sampler = Sampler(backend)
job = sampler.run([qc])
print(f"job id: {job.job_id()}")
result = job.result()
print(result)
```

Lab 3-4 Maze Solver

python quantum_maze_solver.py --size=24 --complexity=1.5 --show-analysis

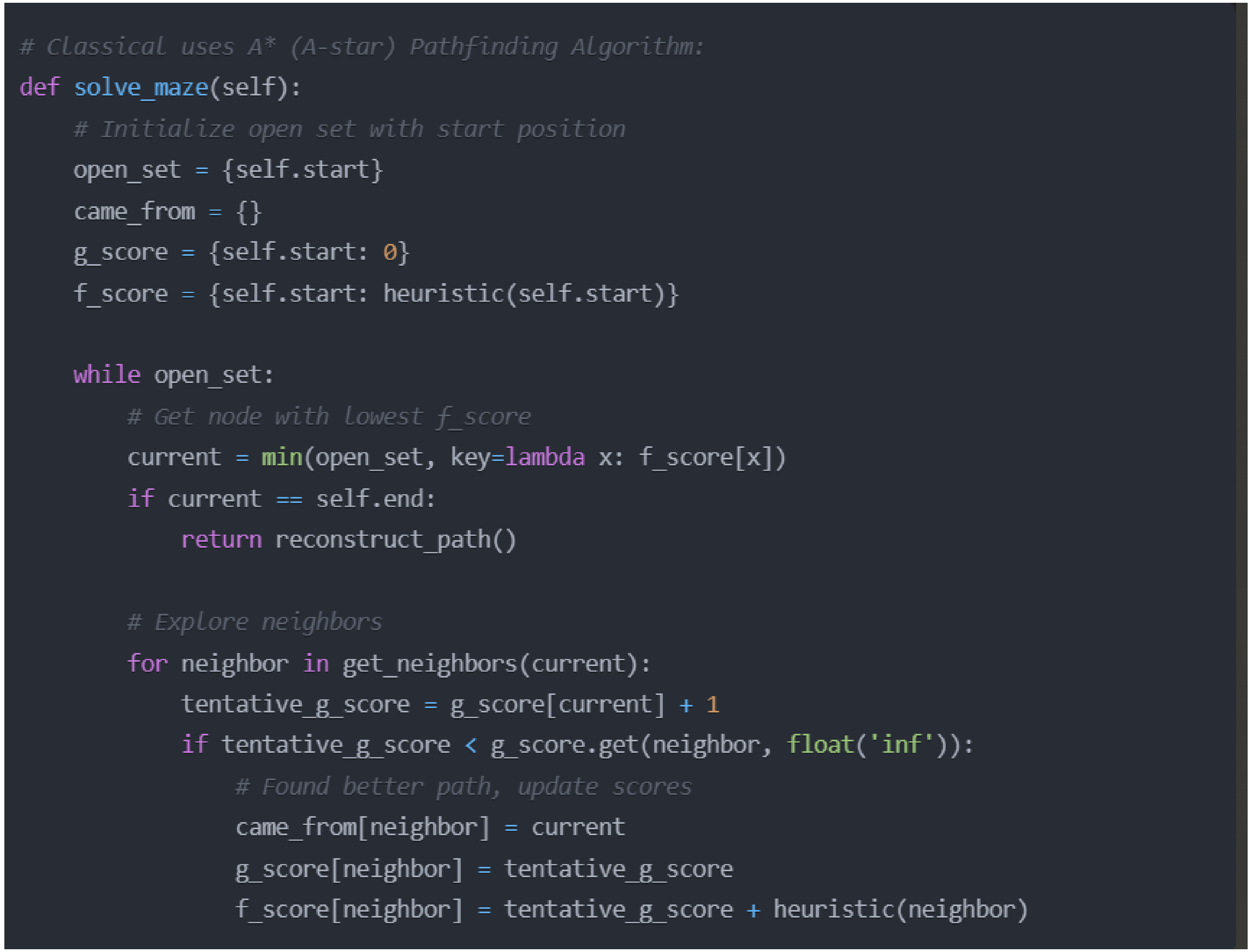


A. Introduction
  - Basic maze setup (size=24, complexity=1.5 recommended)
  - Understanding quantum vs classical paths

B. Hands-on Components
  - Press 'C': Run classical solution
  - Press 'Q': Run quantum solution
  - Press 'R': Reset maze

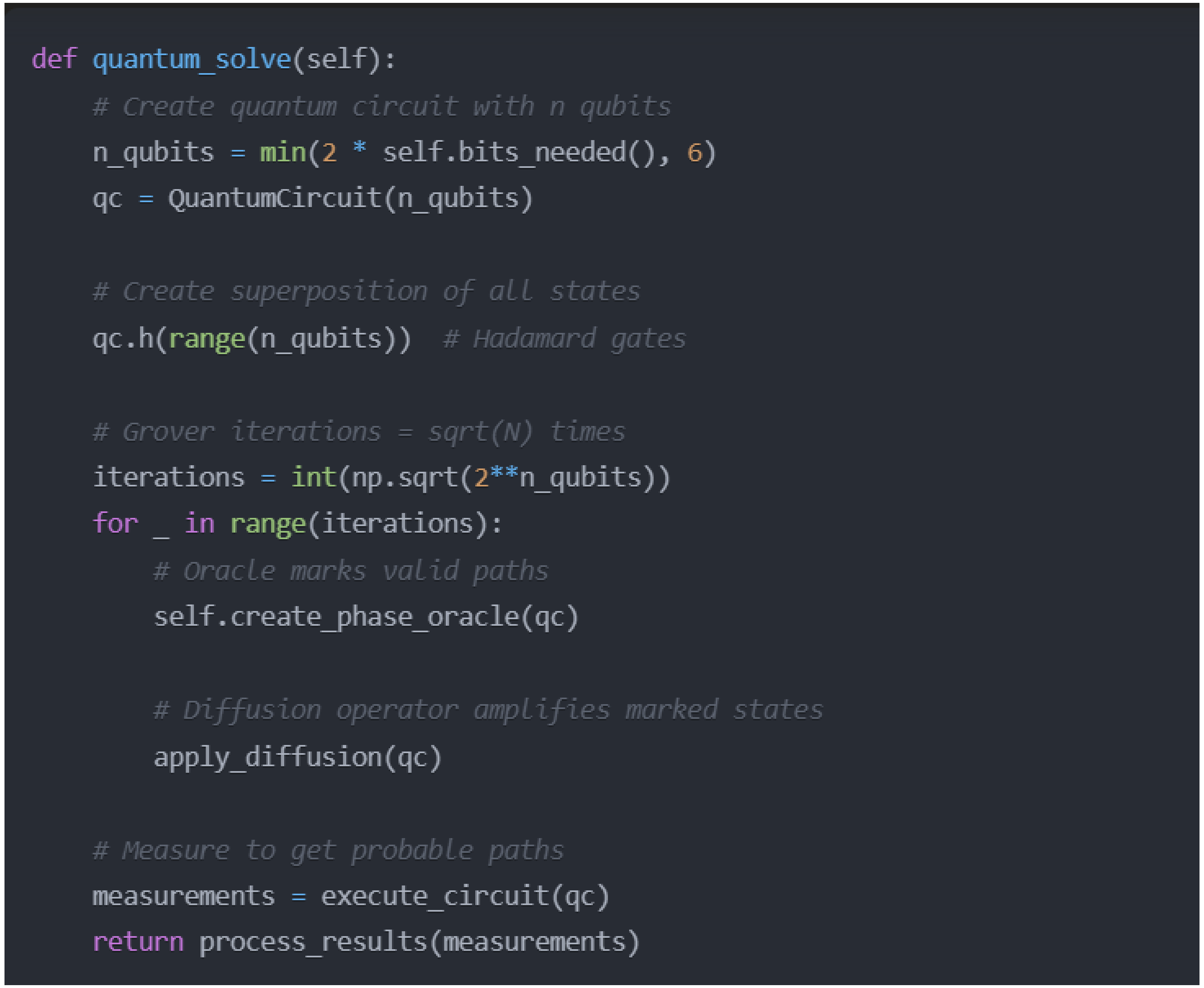
C. Analysis
  - Time comparison
  - Path efficiency
  - State exploration
  - Success rates

Lab Tasks
1. Compare different maze sizes (4x4 vs 8x8)
2. Analyze success rates with varying complexity
3. Study quantum path vs classical path differences
4. Understand qubit limitations and encoding



```python
# Classical uses A* (A-star) Pathfinding Algorithm:
def solve_maze(self):
    # Initialize open set with start position
    open_set = {self.start}
    came_from = {}
    g_score = {self.start: 0}
    f_score = {self.start: heuristic(self.start)}

    while open_set:
        # Get node with lowest f_score
        current = min(open_set, key=lambda x: f_score[x])
        if current == self.end:
            return reconstruct_path()

        # Explore neighbors
        for neighbor in get_neighbors(current):
            tentative_g_score = g_score[current] + 1
            if tentative_g_score < g_score.get(neighbor, float('inf')):
                # Found better path, update scores
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic(neighbor)
```

•Uses A* algorithm with Manhattan distance heuristic
•Guaranteed to find optimal path
•Explores paths sequentially
•Time complexity: O(V + E) where V = vertices, E = edges

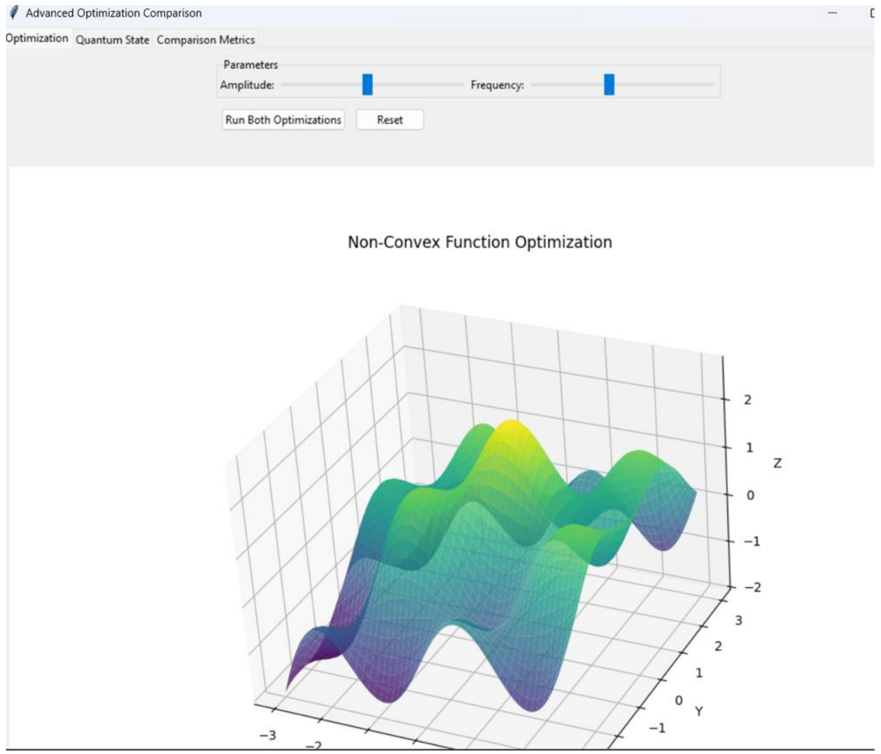


```python
def quantum_solve(self):
    # Create quantum circuit with n qubits
    n_qubits = min(2 * self.bits_needed(), 6)
    qc = QuantumCircuit(n_qubits)

    # Create superposition of all states
    qc.h(range(n_qubits))  # Hadamard gates

    # Grover iterations = sqrt(N) times
    iterations = int(np.sqrt(2**n_qubits))
    for _ in range(iterations):
        # Oracle marks valid paths
        self.create_phase_oracle(qc)

        # Diffusion operator amplifies marked states
        apply_diffusion(qc)

    # Measure to get probable paths
    measurements = execute_circuit(qc)
    return process_results(measurements)
```

- Uses Grover's algorithm
- Creates superposition of all possible paths
- Oracle marks valid moves
- Explores paths in parallel
- Theoretical speedup: $O(\sqrt{N})$ vs $O(N)$ classical
- Quantum is faster but probabilistic

Lab 3-5 Non-Convex

python non-convex.py



1.What is Non-convex Optimization?

•A non-convex function has multiple local minima/maxima (peaks and valleys)

•Unlike convex functions, finding the global minimum is challenging

•Real-world examples include:

- Neural network training
- Molecular structure optimization
- Portfolio optimization
- Circuit design

2.How to Use the Tool:

A. Parameters Control:

•Amplitude slider: Controls the height/depth of peaks and valleys

•Frequency slider: Controls how many peaks/valleys appear

•The 3D surface shows the function landscape:

- X and Y axes: Input variables
- Z axis (height): Function value
- Colors: Represent height (darker = lower values)

B. Optimization Methods:

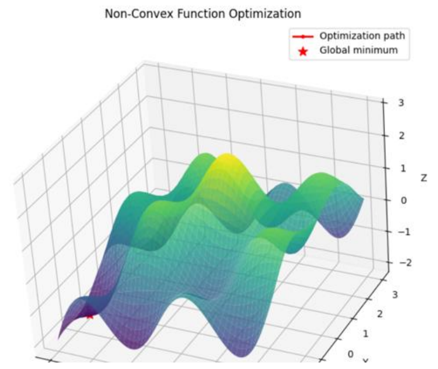
•Classical Method (Blue Path):

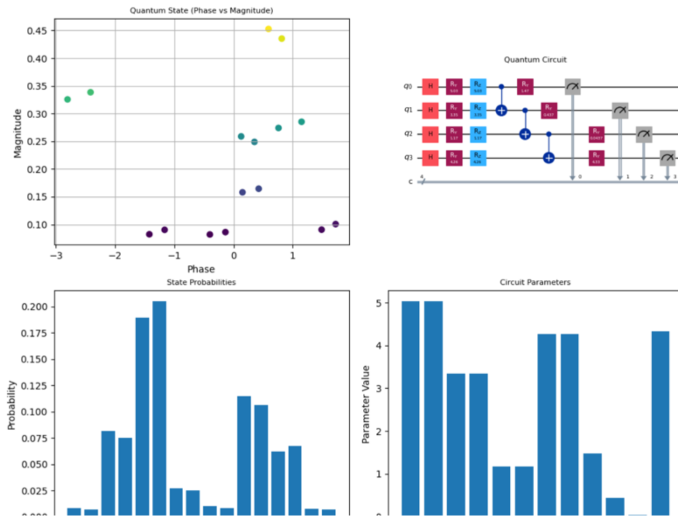
- Uses Differential Evolution algorithm
- Explores multiple points simultaneously
- Generally more stable and reliable
- Shown in blue gradient path

•Quantum Method (Red Path):

- Uses quantum circuit to explore landscape
- Can potentially find solutions faster
- May escape local minima differently
- Shown in red gradient path

Both optimizations completed





3.How to Read Results:

A. Visual Interpretation:

•Surface Plot:

- Peaks = high values
- Valleys = low values
- Color gradient shows height
- Contours on bottom show level curves

B. Optimization Paths:

•Blue path: Classical optimization route

- Darker blue = later iterations
- Blue star = classical best point found

•Red path: Quantum optimization route

- Darker red = later iterations
- Red star = quantum best point found

C. Performance Metrics (in Metrics tab):

•Iterations: Number of steps taken

•Time: Computation time

•Path Length: Total distance traveled

•Final Value: Best function value found

Assignment

- Maze
  - Compare classical vs quantum solutions
    - Record path lengths
    - Compare execution times

- Non-Convex Optimization
  - Characterize different types of non-convex functions
  - Study symmetry and patterns - Analyze relationship between parameters and difficulty