Practical Implementation: Quantum Logarithmic Vectors in Python

Key Concepts to Simulate:

- Position vectors $\vec{v}=(x,y,z)$\vec{v} = (x, y, z)v=(x,y,z) in classical space.
- Quantum amplitude $\psi(\vec{v})=A\cdot\exp(-i\theta)$\psi(\vec{v}) = A \cdot \exp(-i\theta)ψ(v)=A · exp(−iθ), where $A=\log_{\text{base 2}}(1/d)$A = \log_{\text{base 2}}(1/d)A=log base 2(1/d).
- A 4D tetrahedron space with rotations and dimensional contributions.

Python Code:

```
import numpy as np
import math
import matplotlib.pyplot as plt

# Define points A and B in classical 3D space
A = np.array([3, 1, 0])  # Adding a 3rd dimension for 3D compatibility
B = np.array([5, 4, 0])

# Calculate vector AB
vector_AB = B - A
magnitude_AB = np.linalg.norm(vector_AB)

# Quantum amplitude using a logarithmic factor
def quantum_amplitude(vector, base=2):
    d = np.linalg.norm(vector)
    A = math.log(1/d, base)  # Logarithmic scaling
    theta = np.pi / 4        # Example phase (radians)
    psi = A * np.exp(-1j * theta)
    return psi

amplitude_AB = quantum_amplitude(vector_AB)

# Print results
print("Classical Vector AB:", vector_AB)
print("Magnitude of AB:", magnitude_AB)
print("Quantum Amplitude of AB:", amplitude_AB)

# Visualizing the vector in 3D space
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.quiver(0, 0, 0, vector_AB[0], vector_AB[1], vector_AB[2],
color='b', label="Vector AB")
ax.scatter(A[0], A[1], A[2], color='r', label="Point A")
ax.scatter(B[0], B[1], B[2], color='g', label="Point B")
ax.legend()
plt.show()
```

Explanation:

- **Quantum amplitude**: Uses the logarithmic factor scaled by the inverse distance and adds a phase factor.
- **Visualization**: The vector $\vec{AB}$\vec{AB}AB is displayed in 3D space, showing classical and quantum interpretations.

---

2. Practical Implementation: Quantum Logarithmic Dynamics in MATLAB

Key Concepts:

- Position vectors in a 4D tetrahedron (x,y,z,wx, y, z, wx,y,z,w).
- Logarithmic scaling for quantum contributions.
- Rotation of a 4D vector to visualize dimensional transitions.

MATLAB Code:

matlab

Copiar código

```
% Define points A and B in 4D space
A = [3, 1, 0, 0];
B = [5, 4, 0, 0];

% Calculate vector AB
vector_AB = B - A;
magnitude_AB = norm(vector_AB);

% Quantum amplitude using a logarithmic factor
base = 2;
d = norm(vector_AB);
A_log = log2(1/d);  % Logarithmic scaling
```

```
theta = pi/4;       % Example phase
psi = A_log * exp(-1i * theta);

% Display results
disp('Classical Vector AB:');
disp(vector_AB);
disp('Magnitude of AB:');
disp(magnitude_AB);
disp('Quantum Amplitude of AB:');
disp(psi);

% Visualization of vector in 3D space (projection)
figure;
quiver3(0, 0, 0, vector_AB(1), vector_AB(2), vector_AB(3), 'b',
'LineWidth', 2);
hold on;
scatter3(A(1), A(2), A(3), 'r', 'filled');
scatter3(B(1), B(2), B(3), 'g', 'filled');
title('Vector AB in 3D Space (Projection)');
xlabel('X-axis'); ylabel('Y-axis'); zlabel('Z-axis');
legend('Vector AB', 'Point A', 'Point B');
grid on;
hold off;
```

Explanation:

- **4D Rotations**: Though the visualization is in 3D (a projection), the underlying computations can handle 4D rotations using appropriate linear transformations.
- **Quantum amplitude**: Similar to Python, it incorporates logarithmic scaling and a phase term for quantum effects.

Applications of These Implementations

1. **Quantum-Classical Hybrid Modeling**: Simulate transitions between classical paths and quantum behaviors, useful in aerospace navigation.
2. **Advanced Sensor Algorithms**: Use quantum amplitudes to optimize sensor systems operating in higher dimensions (e.g., 4D tetrahedron spaces).

3. **Quantum Simulation for Multidimensional Geometry**: Explore quantum-inspired models for 3D and 4D spaces, aiding in holographic and ultrasonic technologies.