==

iQore Hybrid Execution Stack — Quantum Phase Estimation (QPE) Proof-of-Function

==

Overview:

This package contains a two-path simulation of the Quantum Phase Estimation (QPE) algorithm,

executed using iQore’s proprietary hybrid compute architecture. It demonstrates how

quantum-classical workflows can be validated on classical infrastructure with deterministic logic,

while optionally introducing synthetic quantum-like noise to simulate real-world uncertainty.

All benchmarks were derived from reproducible runs using the included codebases. Results are

documented in the accompanying technical brief and visual benchmark set.

Included Files:

• qpe_simulation_ideal.py

  - Executes QPE under ideal (noise-free) conditions

  - Provides full error metrics, bitstring mapping, and phase reconstruction accuracy

  - Ideal reference for clean execution validation

- qpe_simulation_noisy.py
  - Adds controlled Gaussian noise to simulate decoherence and phase drift
  - Benchmarks error growth, tolerance, and accuracy preservation across noise levels
  - Demonstrates classical error handling under hybrid conditions

- iQore - QPE - Proof of Function Brief - 2025.pdf
  - Technical summary including simulation tables, charts, and architecture explanation
  - Designed for internal use, partner review, and investor demonstration

- README.txt
  - This file

Usage Notes:

- Python 3.x required (NumPy, pandas, matplotlib)
- Ideal simulation can be executed without dependencies beyond base packages
- No simulator or quantum hardware required
- Designed to demonstrate execution viability under both clean and degraded conditions

----

Proprietary & Confidential Notice:

----

iQore_QPE_ideal.py:

```
import numpy as np

import pandas as pd

import time

import matplotlib.pyplot as plt


# ============================================================

# iQore Hybrid Execution – Quantum Phase Estimation (QPE)

# Noisy Simulation Version (Includes Decoherence Emulation)

# ============================================================


# --- Phase Encoding with Noise ---

def phase_to_bitstring_with_noise(phi, t, noise_level=0.0, flip_probability=0.0):
```

```
    """
    Convert a phase ϕ into a t-bit binary representation with simulated noise.

    - Gaussian noise emulates continuous phase decoherence.
    - Bit-flip noise simulates random bit errors (like qubit gate failures).
    """
    # Step 1: Apply Gaussian noise to the phase
    noisy_phi = phi + np.random.normal(0, noise_level)
    noisy_phi = min(max(noisy_phi, 0), 1)  # Clamp to valid [0,1] range

    # Step 2: Convert to binary string
    scaled = int(round(noisy_phi * (2 ** t)))
    bitstring = format(scaled % (2 ** t), f'0{t}b')

    # Step 3: Apply bit-flip noise probabilistically
    flipped_bitstring = ''.join(
        str(1 - int(bit)) if np.random.rand() < flip_probability else bit
        for bit in bitstring
    )

    # Step 4: Convert back to estimated phase
    estimated_phi = int(flipped_bitstring, 2) / (2 ** t)

    return flipped_bitstring, estimated_phi, noisy_phi

# --- Simulation Loop Across Inputs ---
```

```
def simulate_noisy_qpe(phi_values, precision_levels, noise_level=0.01,
flip_probability=0.01):
    """
    Run the noisy QPE simulation across multiple phase inputs and t-bit levels.

    Returns:
        A pandas DataFrame containing all simulation metrics.
    """
    results = []

    for phi in phi_values:
        for t in precision_levels:
            start_time = time.time()

            # Execute noisy QPE logic
            bitstring, estimated_phi, noisy_phi = phase_to_bitstring_with_noise(
                phi, t, noise_level, flip_probability
            )

            # Evaluate result
            error = abs(estimated_phi - phi)
            max_allowed_error = 1 / (2 ** (t + 1))
            passed = error <= max_allowed_error
            total_time = time.time() - start_time

            # Store results
```

```
        results.append({
            'True Phase (ϕ)': round(phi, 10),
            'Noisy Phase (ϕ')': round(noisy_phi, 10),
            'Precision Bits (t)': t,
            'Bitstring': bitstring,
            'Estimated Phase': round(estimated_phi, 10),
            'Absolute Error': round(error, 10),
            'Max Allowed Error': round(max_allowed_error, 10),
            'Passed': passed,
            'Execution Time (s)': round(total_time, 8)
        })

    return pd.DataFrame(results)

# --- Input Definitions ---
phi_test_cases = [
    0.625,        # Binary: 0.101
    0.5,          # Binary: 0.1
    0.375,        # Binary: 0.011
    0.125,        # Binary: 0.001
    0.6,          # Repeating binary
    float(np.pi / 7)  # Irrational phase
]

precision_bits = [3, 4, 5, 8, 10]
```

```
# --- Run Simulation + Benchmarking ---
if __name__ == "__main__":
  print("iQore Hybrid Execution – QPE Simulation with Noise\n")

  # Run simulation with defined noise parameters
  qpe_df = simulate_noisy_qpe(
    phi_test_cases,
    precision_bits,
    noise_level=0.01,       # Moderate Gaussian noise (phase distortion)
    flip_probability=0.01   # Low bit-flip noise (1%)
  )

  # Display results
  print(qpe_df.to_string(index=False))

  # --- Graph 1: Execution Time vs Bit Precision ---
  exec_time_grouped = qpe_df.groupby('Precision Bits (t)')['Execution Time (s)'].mean()
  plt.figure()
  plt.plot(exec_time_grouped.index, exec_time_grouped.values, marker='o')
  plt.title('Figure 1 – Total Execution Time (Noisy)')
  plt.xlabel('Precision Bits (t)')
  plt.ylabel('Avg Execution Time (s)')
  plt.grid(True)
  plt.tight_layout()
  plt.savefig("qpe_noisy_exec_time.png")
  plt.close()
```

```
    # --- Graph 2: Absolute Error vs Precision Bits ---
    error_grouped = qpe_df.groupby('Precision Bits (t)')['Absolute Error'].mean()
    plt.figure()
    plt.plot(error_grouped.index, error_grouped.values, marker='o', color='crimson')
    plt.title('Figure 3 – Error vs Precision Depth (With Noise)')
    plt.xlabel('Precision Bits (t)')
    plt.ylabel('Avg Absolute Error')
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("qpe_noisy_error_vs_precision.png")
    plt.close()
```

iQore_QPE_noisy.py:

```
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt


#
==========================================================================
=====
# iQore Hybrid Execution – Quantum Phase Estimation (QPE) Simulation (No Noise)
#
==========================================================================
=====
```

```
# ----------------------------------------------
# Phase to Bitstring Conversion (Ideal Condition)
# ----------------------------------------------
def phase_to_bitstring(phi, t):
    """
    Converts a real-valued phase (phi) to its binary bitstring representation
    with t bits of precision. This simulates the quantum phase measurement
    process without any noise or uncertainty.

    Args:
        phi (float): The true phase value (between 0 and 1).
        t (int): Number of bits to use for precision.

    Returns:
        bitstring (str): Binary representation of the scaled phase.
        estimated_phi (float): Reconstructed phase from the bitstring.
    """
    scaled = int(round(phi * (2 ** t)))
    bitstring = format(scaled % (2 ** t), f'0{t}b')
    estimated_phi = scaled / (2 ** t)
    return bitstring, estimated_phi

# ----------------------------------------------
# QPE Simulation Loop for Multiple Inputs
# ----------------------------------------------
def simulate_qpe_test(phi_values, precision_levels):
```

```
"""
Simulates the execution of QPE for a range of phase values and precision levels.

Args:
    phi_values (list of float): True phase inputs to be tested.
    precision_levels (list of int): Bit-depth levels for estimation.

Returns:
    DataFrame: Contains metrics for each phase/precision pair including:
            - bitstring output
            - estimated phase
            - error magnitude
            - allowed tolerance
            - pass/fail status
            - execution time
"""
results = []

for phi in phi_values:
    for t in precision_levels:
        start_time = time.time()

        bitstring, estimated_phi = phase_to_bitstring(phi, t)
        error = abs(estimated_phi - phi)
        max_allowed_error = 1 / (2 ** (t + 1))
        passed = error <= max_allowed_error
```

```
        total_time = time.time() - start_time

        results.append({
            'True Phase (ϕ)': round(phi, 10),
            'Precision Bits (t)': t,
            'Expected Bitstring': bitstring,
            'Estimated Phase': round(estimated_phi, 10),
            'Absolute Error': round(error, 10),
            'Max Allowed Error': round(max_allowed_error, 10),
            'Passed': passed,
            'Execution Time (s)': round(total_time, 8)
        })

    return pd.DataFrame(results)

# ----------------------------------------------
# Input Configuration
# ----------------------------------------------
phi_test_cases = [
    0.625,        # Rational: Binary 0.101
    0.5,          # Rational: Binary 0.1
    0.375,        # Rational: Binary 0.011
    0.125,        # Rational: Binary 0.001
    0.6,          # Approximation: Binary repeating
    float(np.pi / 7) # Irrational input
```

```
]

precision_bits = [3, 4, 5, 8, 10]  # Levels of precision (bit-depth)

# ---------------------------------------------
# Graph Generation for Local Visualization
# ---------------------------------------------
def generate_graphs(df):
    """
    Plots and saves two charts for each phase input:
    - Estimated Phase vs. Precision
    - Error vs. Precision

    Args:
        df (DataFrame): Output of the simulation function.
    """
    for phi in df['True Phase (ϕ)'].unique():
        subset = df[df['True Phase (ϕ)'] == phi]

        # Plot 1: Estimated Phase vs Precision
        plt.figure()
        plt.plot(subset['Precision Bits (t)'], subset['Estimated Phase'], marker='o',
label='Estimated Phase')
        plt.axhline(y=phi, color='r', linestyle='--', label='True Phase')
        plt.title(f'Estimated Phase vs Precision for ϕ = {phi}')
        plt.xlabel('Precision Bits (t)')
```

```
    plt.ylabel('Phase')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(f"qpe_phase_estimate_phi_{str(phi).replace('.', '_')}.png")
    plt.close()

    # Plot 2: Error vs Precision
    plt.figure()
    plt.plot(subset['Precision Bits (t)'], subset['Absolute Error'], marker='x', color='orange',
label='Absolute Error')
    plt.axhline(y=subset['Max Allowed Error'].iloc[0], color='g', linestyle='--', label='Max
Allowed Error')
    plt.title(f'Error vs Precision for ϕ = {phi}')
    plt.xlabel('Precision Bits (t)')
    plt.ylabel('Error')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(f"qpe_error_phi_{str(phi).replace('.', '_')}.png")
    plt.close()

# ----------------------------------------------
# Main Execution Block
# ----------------------------------------------
if __name__ == "__main__":
  print("iQore Hybrid Execution – QPE Simulation Results\n")
```

```
df_results = simulate_qpe_test(phi_test_cases, precision_bits)

print(df_results.to_string(index=False))

generate_graphs(df_results)
```