Final Project COM-309: Quantum Information Processing

by Matthias Wyss (SCIPER 329884)
and Sofia Taouhid (SCIPER 339880)
Communication Systems, BA5
Group 15

Prof. Nicolas Macris

Contents

Fall Semester - 2023

1 Theoretical Background

1.1 Bell states

Let $|B_{00}\rangle_{AB} = \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big)$
We will prove that:

$$|B_{00}\rangle^{\otimes n} = \frac{1}{2^{\frac{n}{2}}} \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}$$

Notation: $|B_{00}\rangle^{\otimes n} = \underbrace{|B_{00}\rangle \otimes |B_{00}\rangle \otimes \cdots \otimes |B_{00}\rangle}_{n\,\text{times}}$

Let's prove it by induction:
Base case:
$n = 1$:

$$\begin{aligned}|B_{00}\rangle^{\otimes 1}_{AB} &= |B_{00}\rangle_{AB}\\ &= \frac{1}{2^{\frac{1}{2}}}\big(|00\rangle_{AB} + |11\rangle_{AB}\big)\end{aligned}$$

$n = 2$:

$$\begin{aligned}|B_{00}\rangle^{\otimes 2} &= |B_{00}\rangle_{A_1B_1} \otimes |B_{00}\rangle_{A_2B_2}\\ &= \frac{1}{2}\big(|00\rangle_{A_1B_1} + |11\rangle_{A_1B_1}\big) \otimes \big(|00\rangle_{A_2B_2} + |11\rangle_{A_2B_2}\big)\\ &= \frac{1}{2}\big(|00\rangle_{A_1A_2} \otimes |00\rangle_{B_1B_2} + |01\rangle_{A_1A_2} \otimes |01\rangle_{B_1B_2} + |10\rangle_{A_1A_2} \otimes |10\rangle_{B_1B_2} + |11\rangle_{A_1A_2} \otimes |11\rangle_{B_1B_2}\big)\end{aligned}$$

The property holds for $n = 1$ and $n = 2$.

Inductive step:
Suppose it holds for n, let's check for $n + 1$:

$$\begin{aligned}|B_{00}\rangle^{\otimes n+1} &= |B_{00}\rangle^{\otimes n} \otimes |B_{00}\rangle\\ &= \frac{1}{2^{\frac{n}{2}}} \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n} \otimes \frac{1}{2^{\frac{1}{2}}}\big(|00\rangle_{A_{n+1}B_{n+1}} + |11\rangle_{A_{n+1}B_{n+1}}\big)\\ &= \frac{1}{2^{\frac{n+1}{2}}}\left[\left(\sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\right) \otimes |00\rangle_{A_{n+1}B_{n+1}} + \left(\sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\right) \otimes |11\rangle_{A_{n+1}B_{n+1}}\right]\\ &= \frac{1}{2^{\frac{n+1}{2}}} \sum_{x\in\{0,1\}^n} |x0\rangle_{A_1\ldots A_{n+1}} \otimes |x0\rangle_{B_1\ldots B_{n+1}} + |x1\rangle_{A_1\ldots A_{n+1}} \otimes |x1\rangle_{B_1\ldots B_{n+1}}\\ &= \frac{1}{2^{\frac{n+1}{2}}} \sum_{x\in\{0,1\}^{n+1}} |x\rangle_{A_1\ldots A_{n+1}} \otimes |x\rangle_{B_1\ldots B_{n+1}}\end{aligned}$$

The last equality is true because to obtain all the qubits strings of length n+1, we take all those of length n and either add to them a 0 or a 1.

1.2 An interesting quantum circuit

1.2.1 Probability of measuring $|0\rangle$

a) The overall state after the first Hadamard gate is:

$$\begin{aligned}|\psi_1\rangle &= H|0\rangle \otimes |B_{00}\rangle^{\otimes n}\\ &= \frac{1}{2^{\frac{1}{2}}}\big(|0\rangle + |1\rangle\big) \otimes \frac{1}{2^{\frac{n}{2}}} \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\end{aligned}$$

b) The overall state after the controlled unitary is:

$$|\psi_2\rangle = \frac{1}{2^{\frac{n+1}{2}}}\big(|0\rangle \otimes \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\big) + \frac{1}{2^{\frac{n+1}{2}}}\big(|1\rangle \otimes U_{A_1\cdots A_n} \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\big)$$

$$= \frac{1}{2^{\frac{n+1}{2}}}\big(|0\rangle \otimes \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\big) + \frac{1}{2^{\frac{n+1}{2}}}\big(|1\rangle \otimes \sum_{x\in\{0,1\}^n} U|x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\big)$$

c) The overall state after the second Hadamard gate is:

$$|\psi_3\rangle = \frac{1}{2^{\frac{n+1}{2}}}\big(\frac{1}{2^{\frac{1}{2}}}\big(|0\rangle + |1\rangle\big) \otimes \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\big) + \frac{1}{2^{\frac{n+1}{2}}}\big(\frac{1}{2^{\frac{1}{2}}}\big(|0\rangle - |1\rangle\big) \otimes \sum_{x\in\{0,1\}^n} U|x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\big)$$

$$= \frac{1}{2^{\frac{n+2}{2}}}\big(|0\rangle \otimes \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n} + U|x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\big)$$

$$+ \frac{1}{2^{\frac{n+2}{2}}}\big(|1\rangle \otimes \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n} - U|x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\big)$$

d) The probability of getting $|0\rangle$ in the first qubit is:

$$\begin{aligned}
\Pr(0) &= \left(\frac{1}{2^{\frac{n+2}{2}}}\right)^2 \Bigg(\sum_{x\in\{0,1\}^n} \langle x|_{A_1\ldots A_n} \otimes \langle x|_{B_1\cdots B_n} + \langle x|_{A_1\ldots A_n} U^\dagger \otimes \langle x|_{B_1\cdots B_n}\Bigg) \\
&\cdot \Bigg(\sum_{x\in\{0,1\}^n} |x\rangle_{A_1\cdots A_n} \otimes |x\rangle_{B_1\cdots B_n} + U|x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\cdots B_n}\Bigg) \\
&= \frac{1}{2^{n+2}}\Bigg(\sum_{x\in\{0,1\}^n} \langle x|_{A_1\cdots A_n} \otimes \langle x|_{B_1\cdots B_n}\Bigg)\Bigg(\sum_{x\in\{0,1\}^n} |x\rangle_{A_1\cdots A_n} \otimes |x\rangle_{B_1\cdots B_n}\Bigg) \\
&+ \frac{1}{2^{n+2}}\Bigg(\sum_{x\in\{0,1\}^n} \langle x|_{A_1\cdots A_n} \otimes \langle x|_{B_1\cdots B_n}\Bigg)\Bigg(\sum_{x\in\{0,1\}^n} U|x\rangle_{A_1\cdots A_n} \otimes |x\rangle_{B_1\cdots B_n}\Bigg) \\
&+ \frac{1}{2^{n+2}}\Bigg(\sum_{x\in\{0,1\}^n} \langle x|_{A_1\cdots A_n} U^\dagger \otimes \langle x|_{B_1\cdots B_n}\Bigg)\Bigg(\sum_{x\in\{0,1\}^n} |x\rangle_{A_1\cdots A_n} \otimes |x\rangle_{B_1\cdots B_n}\Bigg) \\
&+ \frac{1}{2^{n+2}}\Bigg(\sum_{x\in\{0,1\}^n} \langle x|_{A_1\cdots A_n} U^\dagger \otimes \langle x|_{B_1\cdots B_n}\Bigg)\Bigg(\sum_{x\in\{0,1\}^n} U|x\rangle_{A_1\cdots A_n} \otimes |x\rangle_{B_1\cdots B_n}\Bigg) \\
&= \frac{1}{2^{n+2}} \cdot 2^n + \frac{1}{2^{n+2}} \cdot 2^n + \frac{1}{2^{n+2}} \cdot \mathrm{Tr}(U) + \frac{1}{2^{n+2}} \cdot \mathrm{Tr}(U^\dagger) \qquad (1) \\
&= \frac{1}{2} + \frac{1}{2^{n+2}}\big(\mathrm{Tr}(U) + \mathrm{Tr}(U^\dagger)\big) \\
&= \frac{1}{2} + \frac{1}{2^2}\frac{1}{2^n} 2\Re(\mathrm{Tr}(U)) \qquad (2) \\
&= \frac{1}{2} + \frac{1}{2}\frac{1}{2^n}\Re(\mathrm{Tr}(U))
\end{aligned}$$

We get (1) because in the first and the second term, we will have a nonzero element if and only if the x's are the same. Because when they are not the same, they are orthogonal to each other and the inner product gives 0. On the third and forth term, same thing but we have here the definition of the trace of a matrix that appears, which is:

$$\mathrm{Tr}(U) = \sum_{x\in\{0,1\}^n} \langle x|U|x\rangle$$

We get (2) because $\mathrm{Tr}(U^\dagger)$ is the complex conjugate of $\mathrm{Tr}(U)$. Therefore, if we add them up, the imaginary part cancels off and we get $2\Re(\mathrm{Tr}(U))$.

1.2.2 How to compute $\frac{1}{2^n}\Re(\mathrm{Tr}(U))$ using the circuit

As we've found an expression of the probability of measuring $|0\rangle$ in terms of $\frac{1}{2^n}\Re(\mathrm{Tr}(U))$, we can use it! How? Simply by simulating many times the experiment, noting each time the measurement is found. At the end, when we have enough simulations we compute [number of times we've measured $|0\rangle$]/[total number of measurements]. Say that it's equal to :

$$\Pr(0) = \frac{1}{2} + \frac{1}{2}\frac{1}{2^n}\Re(\mathrm{Tr}(U))$$

This is equivalent to :

$$\frac{1}{2^n}\Re(\mathrm{Tr}(U)) = 2\Pr(0) - 1$$

1.2.3 Check that $S = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$ is a valid quantum operation

We first compute $S^\dagger$. Then we check the identities $SS^\dagger = S'^\dagger S = \mathbf{I}$.

$$S^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

$$SS^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$S^\dagger S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

1.2.4 How to compute $\frac{1}{2^n}\Im(\mathrm{Tr}(U))$ using the new circuit

The new circuit is such that after the first Hadamard gate, and before the controlled-unitary, we apply S to the first qubit.
The overall state after the first Hadamard gate stays the same as for the first circuit:

$$\begin{aligned}|\psi_1\rangle &= H|0\rangle \otimes |B_{00}\rangle^{\otimes n} \\ &= \frac{1}{2^{\frac{1}{2}}}\big(|0\rangle + |1\rangle\big) \otimes \frac{1}{2^{\frac{n}{2}}} \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\end{aligned}$$

After going through S we have as overall state:

$$\begin{aligned}|\psi_1\rangle &= S\frac{1}{2^{\frac{1}{2}}}\big(|0\rangle + |1\rangle\big) \otimes \frac{1}{2^{\frac{n}{2}}} \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n} \\ &= \frac{1}{2^{\frac{1}{2}}}\big(|0\rangle - i|1\rangle\big) \otimes \frac{1}{2^{\frac{n}{2}}} \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\end{aligned}$$

The overall state after the controlled unitary is:

$$|\psi_2\rangle = \frac{1}{2^{\frac{n+1}{2}}}\left(|0\rangle \otimes \sum_{x\in\{0,1\}^n} |x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\right) - i\left(|1\rangle \otimes \sum_{x\in\{0,1\}^n} U_{A_1\cdots A_n}|x\rangle_{A_1\ldots A_n} \otimes |x\rangle_{B_1\ldots B_n}\right)$$

The overall state after the second Hadamard gate is:

$$
\begin{aligned}
|\psi_3\rangle &= \frac{1}{2^{\frac{n+1}{2}}}\left(\frac{1}{2^{\frac{1}{2}}}\left(|0\rangle+|1\rangle\right)\otimes\sum_{x\in\{0,1\}^n}|x\rangle_{A_1\ldots A_n}\otimes|x\rangle_{B_1\ldots B_n}\right)-\frac{1}{2^{\frac{n+1}{2}}}i\left(\frac{1}{2^{\frac{1}{2}}}\left(\left(|0\rangle-|1\rangle\right)\right)\otimes\sum_{x\in\{0,1\}^n}U|x\rangle_{A_1\ldots A_n}\otimes|x\rangle_{B_1\ldots B_n}\right)\\
&= \frac{1}{2^{\frac{n+2}{2}}}\big(|0\rangle\otimes\sum_{x\in\{0,1\}^n}|x\rangle_{A_1\ldots A_n}\otimes|x\rangle_{B_1\ldots B_n}-iU|x\rangle_{A_1\ldots A_n}\otimes|x\rangle_{B_1\ldots B_n}\big)\\
&+ \frac{1}{2^{\frac{n+2}{2}}}\big(|1\rangle\otimes\sum_{x\in\{0,1\}^n}|x\rangle_{A_1\ldots A_n}\otimes|x\rangle_{B_1\ldots B_n}+iU|x\rangle_{A_1\ldots A_n}\otimes|x\rangle_{B_1\ldots B_n}\big)
\end{aligned}
$$

Let's now compute the probability to get $|0\rangle$ in the first qubit:

$$
\begin{aligned}
\Pr(0) &= \left(\frac{1}{2^{\frac{n+2}{2}}}\right)^2\Big(\sum_{x\in\{0,1\}^n}\langle x|_{A_1\ldots A_n}\otimes\langle x|_{B_1\cdots B_n}+i\langle x|_{A_1\ldots A_n}U^\dagger\otimes\langle x|_{B_1\cdots B_n}\Big)\\
&\cdot\Big(\sum_{x\in\{0,1\}^n}|x\rangle_{A_1\ldots A_n}\otimes|x\rangle_{B_1\cdots B_n}-iU|x\rangle_{A_1\ldots A_n}\otimes|x\rangle_{B_1\cdots B_n}\Big)\\
&= \frac{1}{2^{n+2}}\Big(\sum_{x\in\{0,1\}^n}\langle x|_{A_1\cdots A_n}\otimes\langle x|_{B_1\cdots B_n}\Big)\Big(\sum_{x\in\{0,1\}^n}|x\rangle_{A_1\cdots A_n}\otimes|x\rangle_{B_1\cdots B_n}\Big)\\
&+ \frac{1}{2^{n+2}}\Big(\sum_{x\in\{0,1\}^n}\langle x|_{A_1\cdots A_n}\otimes\langle x|_{B_1\cdots B_n}\Big)\Big(\sum_{x\in\{0,1\}^n}-iU|x\rangle_{A_1\cdots A_n}\otimes|x\rangle_{B_1\cdots B_n}\Big)\\
&+ \frac{1}{2^{n+2}}\Big(\sum_{x\in\{0,1\}^n}i\langle x|_{A_1\cdots A_n}U^\dagger\otimes\langle x|_{B_1\cdots B_n}\Big)\Big(\sum_{x\in\{0,1\}^n}|x\rangle_{A_1\cdots A_n}\otimes|x\rangle_{B_1\cdots B_n}\Big)\\
&+ \frac{1}{2^{n+2}}\Big(\sum_{x\in\{0,1\}^n}i\langle x|_{A_1\cdots A_n}U^\dagger\otimes\langle x|_{B_1\cdots B_n}\Big)\Big(\sum_{x\in\{0,1\}^n}-iU|x\rangle_{A_1\cdots A_n}\otimes|x\rangle_{B_1\cdots B_n}\Big)\\
&= \frac{1}{2^{n+2}}\cdot 2^n+\frac{1}{2^{n+2}}\cdot 2^n-\frac{i}{2^{n+2}}\cdot\mathrm{Tr}(U)+\frac{i}{2^{n+2}}\cdot\mathrm{Tr}(U^\dagger)\\
&= \frac{1}{2}-\frac{i}{2^{n+2}}\big(\mathrm{Tr}(U)-\mathrm{Tr}(U^\dagger)\big)\\
&= \frac{1}{2}+\frac{1}{2^2}\frac{1}{2^n}2\Im(\mathrm{Tr}(U))\\
&= \frac{1}{2}+\frac{1}{2}\frac{1}{2^n}\Im(\mathrm{Tr}(U))
\end{aligned}
$$

Now that we have found Pr(0) in terms of $\Im(\mathrm{Tr}(U))$, we can proceed as we said for $\Re(\mathrm{Tr}(U))$: simulate many times the experiment, noting each time the measurement found. At the end, when we have enough simulations we do compute [number of times we've measured $|0\rangle$]/[total number of measurements] say that it's equal to :

$$\Pr(0)=\frac{1}{2}+\frac{1}{2}\frac{1}{2^n}\Im(\mathrm{Tr}(U))$$

This is equivalent to :

$$\frac{1}{2^n}\Im(\mathrm{Tr}(U))=2\Pr(0)-1$$

1.2.5 How many operations does a naive classical trace computation need to do to compute the trace of a unitary action on n qubits ?

A classical trace computation do the addition of all the diagonal elements of the matrix. There are 2^n diagonal elements in the case of a unitary action on n qubits. So the computation will take 2^n-1 addition operation. Therefore, the naive computation of the trace of a unitary action on n qubits is $\mathcal{O}(2^n)$.

2 Practical Implementation

We import useful libraries:

```
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile
from qiskit.tools.jupyter import *
from qiskit.visualization import *
```

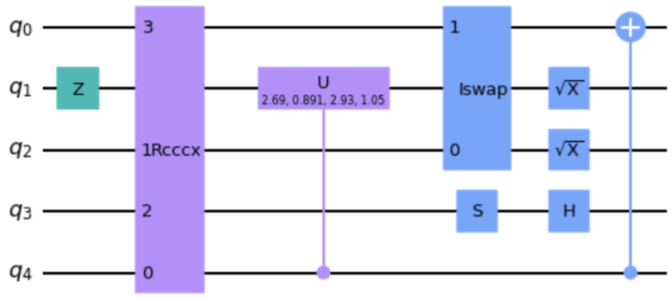
```
# Useful imports
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from qiskit import QuantumCircuit, execute, BasicAer, Aer, transpile
from qiskit.circuit import QuantumCircuit
from qiskit.circuit.library import PauliEvolutionGate
from qiskit.opflow import I, Z, X
from qiskit.circuit.random import random_circuit
from qiskit.quantum_info import Operator
```

2.3 A random unitary and a quantum device

2.3.1 Random unitary generation

We generate a random unitary circuit over $n = 5$ qubits with depth $= 3$:

```
# Generate a random quantum circuit with 5 qubits and 3 layers
rdm_u = random_circuit(5, 3)

# Visualize the random circuit
rdm_u.draw(output='mpl')
```



2.3.2 Trace

We compute the trace of this unitary classically:

```
# Convert the random quantum circuit to an operator and extract its unitary matrix
operator = Operator(rdm_u)
unitary_matrix = operator.data

# Compute the trace of the unitary matrix
trace = np.trace(unitary_matrix)

# Print the trace and the real part of the trace
print(f"The trace is: {trace}")
print(f"The real part of the trace is: {np.real(trace)}")
```

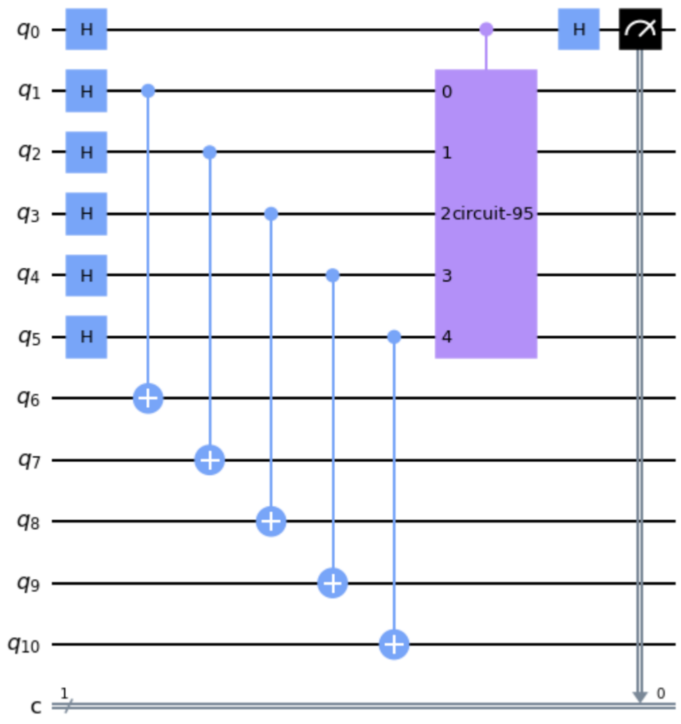
```
The trace is: (-0.5963444243101163-0.41281861726777347j)
The real part of the trace is: -0.5963444243101163
```

2.3.3 Trace estimation circuit

We create and run a trace estimation circuit on a quantum simulator. We compare the trace estimate from the quantum circuit and the classical trace computation:

```
# Number of qubits
n = 5

# Create a quantum circuit with 2n+1 qubits and 1 classical bit
circ = QuantumCircuit(2 * n + 1, 1)

# Apply Hadamard gate on the first qubit
circ.h(0)

# Construct n Bell states on qubits 1 to n
for i in range(1, n + 1):
    circ.h(i)
    circ.cx(i, i + n)

# Convert the random quantum circuit to a gate and apply it as a controlled gate
cu = rdm_u.to_gate().control(1)
circ.append(cu, [i for i in range(0, n + 1)])

# Apply Hadamard gate on the first qubit again
circ.h(0)

# Measure the first qubit and store the result in the classical bit
circ.measure(0, 0)
```

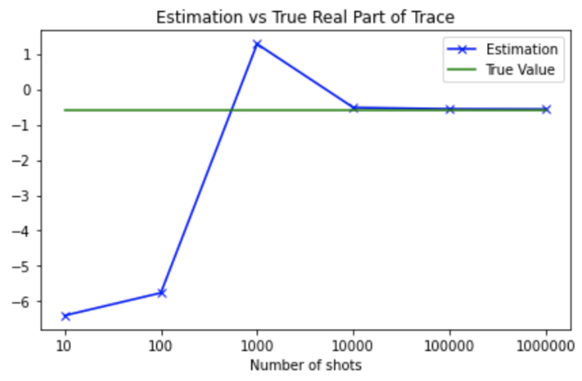
```
# Visualize the quantum circuit
circ.draw(output='mpl')
```

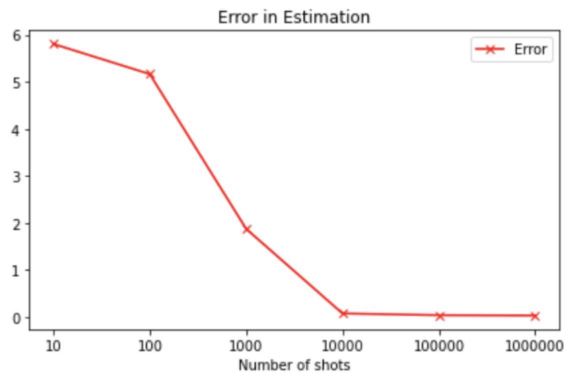


```python
# Number of qubits
n = 5

# Number of iterations
num_iterations = 6

# List of shot values (powers of 10)
shots_values = [10**i for i in range(1, num_iterations+1)]

# Lists to store results
estimated_real_parts = []
true_real_parts = []
errors = []

# Loop through different numbers of shots
for i in range(1, num_iterations+1):
    shots = 10**i

    # Run the simulation on the qasm simulator
    simulator = Aer.get_backend('qasm_simulator')
    transpiled_circ = transpile(circ, simulator)
    result = simulator.run(transpiled_circ, shots=shots).result()

    # Extract the counts and compute the probability of getting '0'
    counts = result.get_counts(transpiled_circ)
    num_times_zero = counts.get('0', 0)
    proba_0 = num_times_zero / shots

    # Estimate the real part of the trace
    estimated_trace_real_part = 2**(n+1) * proba_0 - 2**n
    estimated_real_parts.append(estimated_trace_real_part)

    # True real part of the trace
    true_real_part = np.real(trace)
    true_real_parts.append(true_real_part)

    # Compute the error
    error = abs(estimated_trace_real_part - true_real_part)
    errors.append(error)

    # Print results
    print(f"For a number of shots of: {shots}")
    print(f"The circuit estimates the real part of the trace: {estimated_trace_real_part}")
    print(f"The true real part of the trace is: {true_real_part}")
    print(f"We have an error of {error}\n")
```

```python
# Plot the results
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(estimated_real_parts, '-bx')
plt.plot(true_real_parts, '-g')
plt.legend(["Estimation", "True Value"])
plt.title("Estimation vs True Real Part of Trace")
plt.xticks(range(num_iterations), shots_values)
plt.xlabel("Number of shots")

plt.subplot(1, 2, 2)
plt.plot(errors, '-rx')
plt.legend(["Error"])
plt.title("Error in Estimation")
plt.xticks(range(num_iterations), shots_values)
plt.xlabel("Number of shots")

plt.tight_layout()
plt.show()
```

```
For a number of shots of: 10
The circuit estimates the real part of the trace: -6.3999999999999999
The true real part of the trace is: -0.5963444243101163
We have an error of 5.803655575689882

For a number of shots of: 100
The circuit estimates the real part of the trace: -5.760000000000002
The true real part of the trace is: -0.5963444243101163
We have an error of 5.163655575689885

For a number of shots of: 1000
The circuit estimates the real part of the trace: 1.2800000000000011
The true real part of the trace is: -0.5963444243101163
We have an error of 1.8763444243101173

For a number of shots of: 10000
The circuit estimates the real part of the trace: -0.5183999999999997
The true real part of the trace is: -0.5963444243101163
We have an error of 0.0779444243101165

For a number of shots of: 100000
The circuit estimates the real part of the trace: -0.5555200000000013
The true real part of the trace is: -0.5963444243101163
We have an error of 0.040824424310114904

For a number of shots of: 1000000
The circuit estimates the real part of the trace: -0.5624320000000012
The true real part of the trace is: -0.5963444243101163
We have an error of 0.0339124243101151
```





2.4 A mystery

2.4.1 Using the trace estimation circuit to estimate an angle θ

We can use Euler identity that states that for a matrix A :

$$e^{i\theta A} = \cos(\theta)I + i\sin(\theta)A$$

We can write our matrix U as:

$$U = e^{i\theta(X\otimes X)} = \cos(\theta)I_{4\times 4} + i\sin(\theta)(X\otimes X)$$

So using this identity, we can simulate the circuit many times and use the expressions of $\Re(\mathrm{Tr}(U))$ and $\Im(\mathrm{Tr}(U))$ in terms of the probabilities to find the angle θ.
Using Euler identity :

$$\mathrm{Tr}(U) = 4\cos(\theta) + i\sin(\theta)\mathrm{Tr}(X\otimes X)$$
$$\Rightarrow \begin{cases} \Re(\mathrm{Tr}(U)) = 4\cos(\theta) - \sin(\theta)\Im(\mathrm{Tr}(X\otimes X)) \\ \Im(\mathrm{Tr}(U)) = \sin(\theta)\Re(\mathrm{Tr}(X\otimes X)) \end{cases}$$

We compute $\mathrm{Tr}(X \otimes X)$:

$$X \otimes X = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\Rightarrow \mathrm{Tr}(X \otimes X) = 0$$

We found before (with the circuit without the S gate):

$$\frac{1}{2^n}\Re(\mathrm{Tr}(U)) = (2\,\mathrm{Pr}(0) - 1)$$
$$\Rightarrow \Re(\mathrm{Tr}(U)) = 2^n(2\,\mathrm{Pr}(0) - 1)$$

Thus we have:

$$\Re(\mathrm{Tr}(U)) = 4\cos(\theta) - \sin(\theta)\Im(\mathrm{Tr}(X \otimes X))$$
$$\Rightarrow 2^n(2\,\mathrm{Pr}(0) - 1) = 4\cos(\theta)$$
$$\Rightarrow \theta = \arccos\left(\frac{2^n(2\,\mathrm{Pr}(0) - 1)}{4}\right)$$

We generate the unitary $e^{i\theta(X \otimes X)}$ for $\theta = \frac{\pi}{3}$:

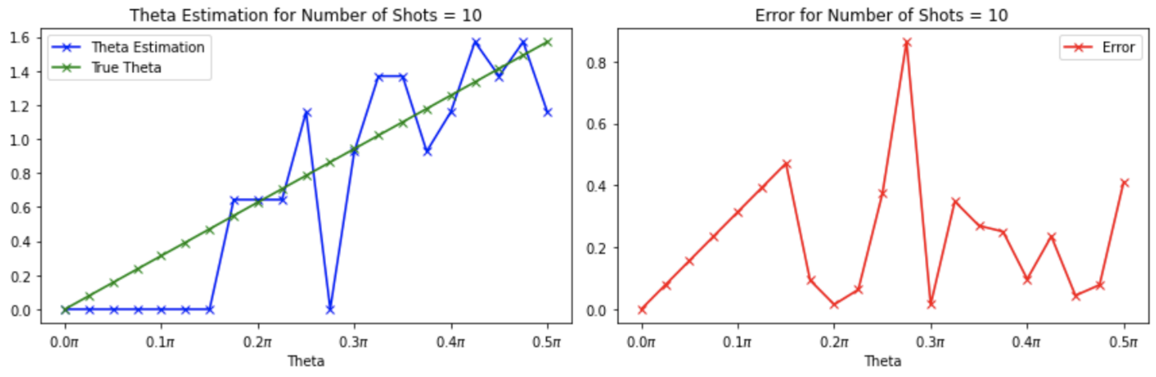
```
# Rotation angle
theta = np.pi/3

# PauliEvolutionGate with the parameterized rotation angle
U_mystery = PauliEvolutionGate(X^X, theta)

# Calculate the trace of the corresponding unitary operator
mystery_trace = np.trace(Operator(U_mystery).data)

# Print the trace and the real part of the trace
print(f"The trace is: {mystery_trace}")
print(f"The real part of the trace is: {np.real(mystery_trace)}")
```

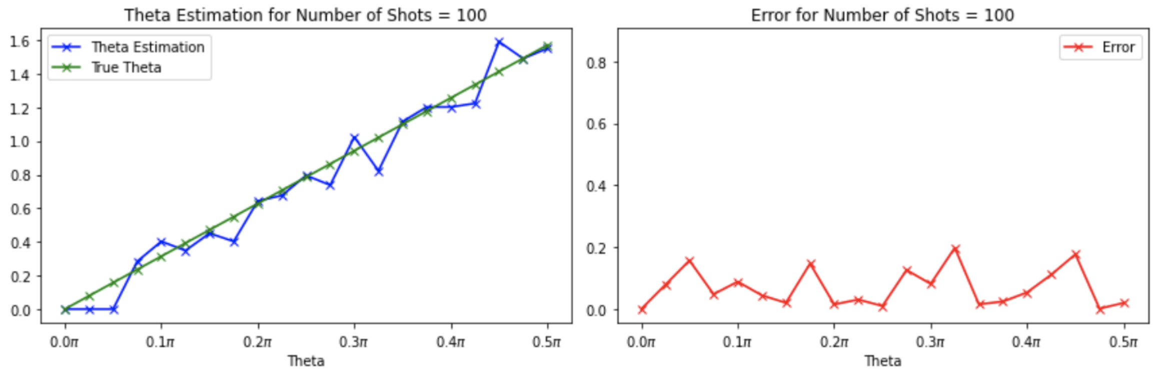
```
The trace is: (2.0000000000000004+0j)
The real part of the trace is: 2.0000000000000004
```

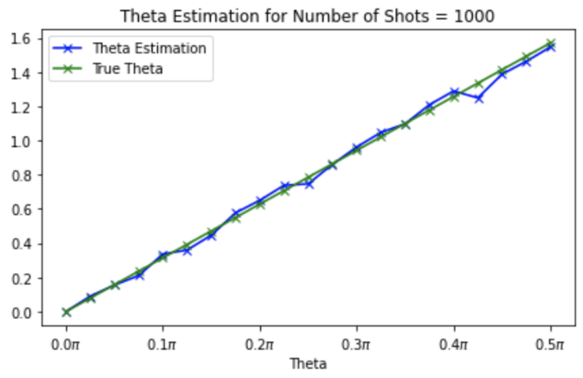
2.4.2 Trace estimation circuit for different values of θ

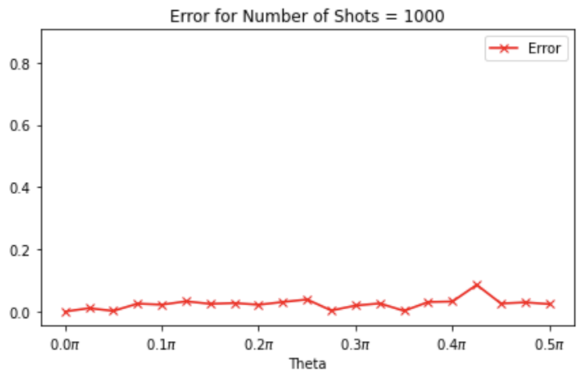
We generate different values of θ, run the trace estimation circuit for each value and plot the results:

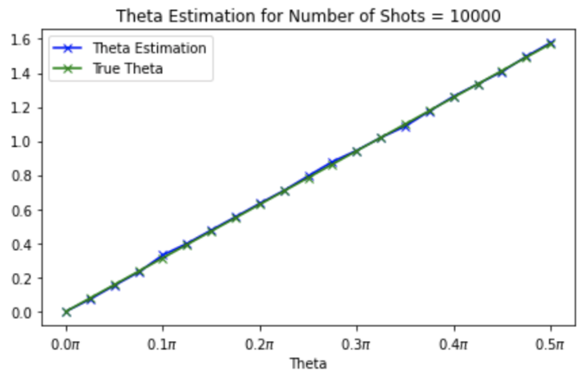
```
# Number of values for theta
num_values = 20

# Number of qubits
n = 2

# Number of iterations
num_iterations = 6

# Variable for y-axis limits in the error plots
y_limits = None

# Loop through different numbers of shots
for i in range(1, num_iterations + 1):
    shots = 10 ** i

    # Lists to store results
    thetas = []
    estimated_thetas = []
    errors = []

    # Loop through different values of theta
    for j in range(0, num_values + 1):
        theta = j * (np.pi / (2 * num_values))
        thetas.append(theta)

        # Define the PauliEvolutionGate with the parameterized theta
        U_mystery = PauliEvolutionGate(X ^ X, theta)

        # Construct the quantum circuit for the parameterized gate
        mystery_circ = QuantumCircuit(2 * n + 1, 1)
        mystery_circ.h(0)
        for k in range(1, n + 1):
            mystery_circ.h(k)
            mystery_circ.cx(k, k + n)
        cu = U_mystery.control(1)
        mystery_circ.append(cu, [0, 1, 2])
        mystery_circ.h(0)
        mystery_circ.measure(0, 0)

        # Run the quantum simulation
        simulator = Aer.get_backend('qasm_simulator')
        mystery_transpiled_circ = transpile(mystery_circ, simulator)
        result = simulator.run(mystery_transpiled_circ, shots=shots).result()
        # Analyze the result and estimate theta
        counts = result.get_counts(mystery_transpiled_circ)
        proba_0 = counts.get('0', 0) / shots
        estimated_theta = np.arccos((2 ** (n + 1) * proba_0 - 2 ** n) / 4)
        estimated_thetas.append(estimated_theta)

        # Compute the error between the true and estimated parameter
        errors.append(abs(estimated_theta - theta))

    # Plot the results for the current number of shots
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(thetas, estimated_thetas, '-bx')
    plt.plot(thetas, thetas, '-gx')
    plt.legend(["Theta Estimation", "True Theta"])
    plt.title(f"Theta Estimation for Number of Shots = {shots}")
    plt.xlabel("Theta")
    plt.xticks(np.linspace(0, np.pi / 2, 6), labels=[f'{i/10}$\pi$' for i in range(6)])

    plt.subplot(1, 2, 2)
    plt.plot(thetas, errors, '-rx')
    plt.legend(["Error"])
    plt.title(f"Error for Number of Shots = {shots}")
    plt.xlabel("Theta")
    plt.xticks(np.linspace(0, np.pi / 2, 6), labels=[f'{i/10}$\pi$' for i in range(6)])

    if y_limits is None:
        y_limits = plt.gca().get_ylim()
    else:
        plt.gca().set_ylim(y_limits)

    plt.tight_layout()
    plt.show()
```

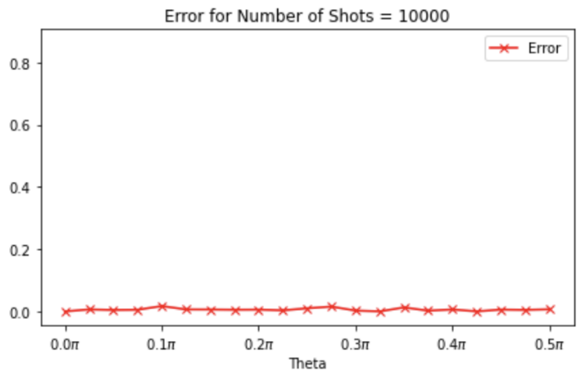


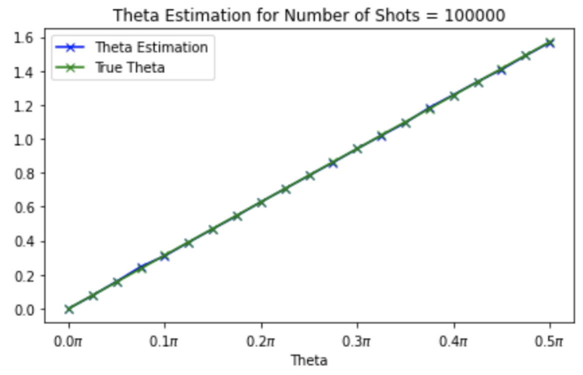


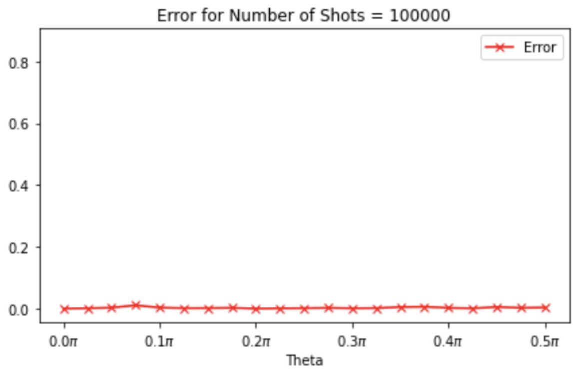

Theta Estimation for Number of Shots = 1000
Theta Estimation
True Theta
Theta

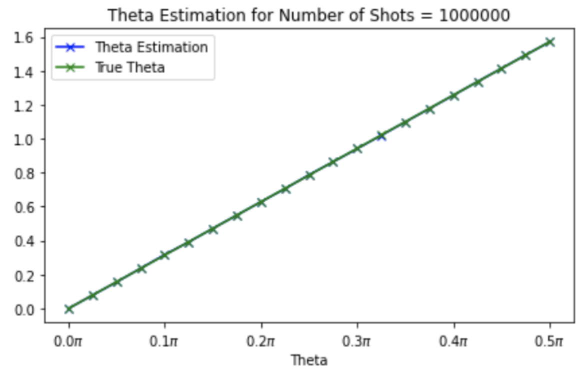

Error for Number of Shots = 1000
Error
Theta

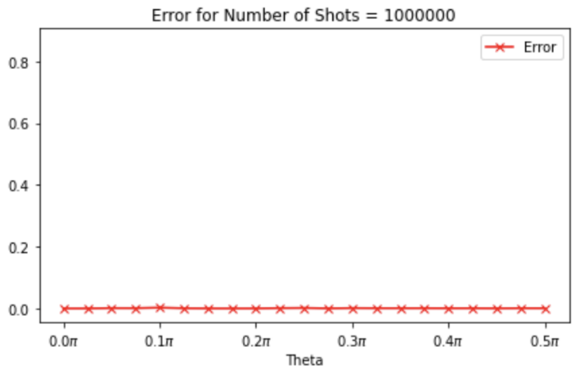

Theta Estimation for Number of Shots = 10000
Theta Estimation
True Theta
Theta


Error for Number of Shots = 10000
Error
Theta


Theta Estimation for Number of Shots = 100000
Theta Estimation
True Theta
Theta


Error for Number of Shots = 100000
Error
Theta


Theta Estimation for Number of Shots = 1000000
Theta Estimation
True Theta
Theta


Error for Number of Shots = 1000000
Error
Theta

2.5 Confidence intervals

2.5.1 Expression of $\bar{X}$ and S_k

We know that $\bar{X} := \frac{1}{k}\sum_{i=1}^{k} X_i$.
We also know that X_i is 1 if the measurement is $|0\rangle$ and 0 otherwise. Here we run the circuit k times and there are k_0 measurements that give $|0\rangle$, so k_0 nonzero terms (more precisely, each equal to 1) in the sum.
Therefore:

$$\bar{X} = \frac{1}{k}\underbrace{(1+\cdots+1)}_{k_0 \text{ times}} = \frac{k_0}{k} = \hat{p}$$

Let's try to find the expression of S_k now.
We know that $S_k^2 := \frac{1}{k}\sum_{i=1}^{k}(X_i - \bar{X})^2$
Let's use it :

$$\begin{aligned} S_k^2 &= \frac{1}{k}\sum_{i=1}^{k}(X_i - \bar{X})^2 \\ &= \frac{k-k_0}{k}\hat{p}^2 + \frac{k_0}{k}(1-\hat{p})^2 \\ &= \hat{p}^2 - \hat{p}^3 + \hat{p} - 2\hat{p}^2 + \hat{p}^3 \\ &= -\hat{p}^2 + \hat{p} \\ &= \hat{p}(1-\hat{p}) \end{aligned}$$

We thus end up with $S_k = \sqrt{\hat{p}(1-\hat{p})}$.

Something we could have done is notice that each of the X_i's follows a Bernoulli distribution of probability of success (getting $|0\rangle$) $\hat{p}$. So their sum follows a Binomial distribution of k times an experiment of probability of success $\hat{p}$. Thus, the mean of the sum is k$\hat{p}$ and the mean of $\frac{1}{k}\sum_{i=1}^{k} X_i$ would be $\frac{1}{k}k\hat{p} = \hat{p}$. For the variance, it would be the variance of Bernoulli(k,$\hat{p}$) multiplied by $\frac{1}{k}$ which is : $S_k^2 = \frac{1}{k}k\hat{p}(1-\hat{p}) \Rightarrow S_k = \hat{p}(1-\hat{p})$.

2.5.2 Find a CI for $\Re(\mathrm{Tr}(U))$ at level $1-\alpha$ and the $\hat{p}$ that gives the widest interval

The Confidence Interval (CI) for $E[X_i]$ is $\hat{p} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}}$.
As we know that $E[X_i] = \Pr(0) = \hat{p} = \frac{1}{2} + \frac{1}{2}\frac{1}{2^n}\Re(\mathrm{Tr}(U))$, we can replace $E[X_i]$ by this to find a CI for $\Re(\mathrm{Tr}(U))$:

$$\begin{aligned} \hat{p} - 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} &\leq \frac{1}{2} + \frac{1}{2}\frac{1}{2^n}\Re(\mathrm{Tr}(U)) \leq \hat{p} + 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} \\ \Leftrightarrow \hat{p} - 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} - \frac{1}{2} &\leq \frac{1}{2}\frac{1}{2^n}\Re(\mathrm{Tr}(U)) \leq \hat{p} + 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} - \frac{1}{2} \\ \Leftrightarrow 2^{n+1}\left(\hat{p} - 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} - \frac{1}{2}\right) &\leq \Re(\mathrm{Tr}(U)) \leq 2^{n+1}\left(\hat{p} + 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} - \frac{1}{2}\right) \end{aligned}$$

We end up with the following confidence interval for $\Re(\mathrm{Tr}(U))$:

$$2^{n+1}\left(\hat{p} - \frac{1}{2} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}}\right)$$

Now, we want to find the $\hat{p}$ that gives us the widest interval. A way to do that is to minimize the function corresponding to the size of the interval:

$$\begin{aligned} f(\hat{p}) &= 2^{n+1}\left(\hat{p} - \frac{1}{2} + 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}}\right) - 2^{n+1}\left(\hat{p} - \frac{1}{2} - 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{k}}\right) \\ &= 2^{n+1}\left(3.92\sqrt{\frac{\hat{p}(1-\hat{p})}{k}}\right) \end{aligned}$$

Let's compute the derivative of f in terms of $\hat{p}$ to find the $\hat{p}$ that minimizes the function:

$$\frac{df}{d\hat{p}} = 1.96\frac{1-2\hat{p}}{\sqrt{k\hat{p}(1-\hat{p})}} = 0 \Leftrightarrow \hat{p} = \frac{1}{2}$$

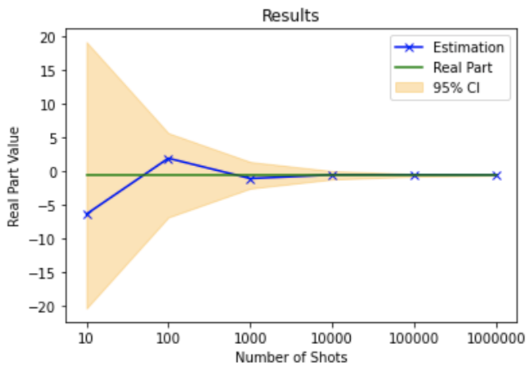
$\hat{p} = \frac{1}{2}$ gives the widest interval.

2.5.3 Trace estimation circuit with different number of shots and with CI at 95%

For different values of shots num_shots, we run the trace estimation circuit num_shots times and compute the previously derived single value estimate of $\Re(\mathrm{Tr}(U))$ and the CI we got previously at level 95%.
We plot the real value of $\Re(\mathrm{Tr}(U))$, the single-value estimate of it, and the CI against num_shots:

```python
# Number of qubits
n = 5

# Number of iterations
num_iterations = 6

# Lists to store results
estimated_real_parts = []
real_parts = []
CI_1_list = []
CI_2_list = []

p = 1/2
z_0975 = 1.96

# Loop through different numbers of shots
for i in range(1, num_iterations+1):
    num_shots = 10**i

    # Simulate quantum circuit
    simulator = Aer.get_backend('qasm_simulator')
    transpiled_circ = transpile(circ, simulator)
    result = simulator.run(transpiled_circ, shots=num_shots).result()

    # Calculate probability of measuring '0'
    counts = result.get_counts(transpiled_circ)
    num_times_zero = counts.get('0', 0)
    proba_0 = num_times_zero / num_shots

    # Estimate real part of the trace
    estimated_trace_real_part = 2**(n + 1) * proba_0 - 2**n
    estimated_real_parts.append(estimated_trace_real_part)
    real_parts.append(np.real(trace))

    # Calculate confidence intervals
    CI_1 = 2**(n + 1) * (p - 0.5 + z_0975 * np.sqrt((p * (1 - p)) / num_shots)) + np.real(trace)
    CI_1_list.append(CI_1)

    CI_2 = 2**(n + 1) * (p - 0.5 - z_0975 * np.sqrt((p * (1 - p)) / num_shots)) + np.real(trace)
    CI_2_list.append(CI_2)
```

```
shots_values = [10**i for i in range(1, num_iterations+1)]

# Plot the results
plt.figure()
plt.plot(estimated_real_parts, '-bx')
plt.plot(real_parts, '-g')
plt.fill_between([i for i in range(0, num_iterations)], CI_1_list, CI_2_list, color='orange', alpha=0.3)
plt.legend(["Estimation", "Real Part", "95% CI"])
plt.title("Results")
plt.xlabel("Number of Shots")
plt.ylabel("Real Part Value")
plt.xticks([i for i in range(0, num_iterations)], shots_values)
plt.show()
```


Results
Estimation
Real Part
95% CI
Real Part Value
Number of Shots