Calculating the Expectation Value of a Unitary Operator U w.r.t. a State Vector ψ with the Hadamard Test Circuit and Qiskit

Claudio Fontana,
University of Palermo
Palermo, Italy
claudio.fontana@unipa.it

```python
import qiskit as qis
import cmath
import numpy as np
from qiskit import BasicAer, execute
from qiskit.visualization import import plot_histogram


def hadamardTestCircuit(U,psi,imag:bool=False):
    cU=U.to_gate().control()
    #----
    qreg = qis.QuantumRegister(psi.num_qubits,name='psi')
    ancilla=qis.QuantumRegister(1,name='ancilla')
    creg = qis.ClassicalRegister(1,name='meas')
    circ = qis.QuantumCircuit(ancilla,qreg,creg)
    circ.append(psi,qreg)
    circ.h(ancilla)
    if imag:
        circ.sdg(ancilla)
    tmp=[ancilla]
    for q in qreg:
        tmp.append(q)
    circ.barrier()
    circ.append(cU,tmp)
    circ.barrier()
    circ.h(ancilla)
    circ.measure(ancilla,creg)
    return circ


#Example: operator pauli 'ZX'; psi=|0+>

psi = qis.QuantumCircuit(2,name='psi')
psi.h(0)
#----

U=qis.QuantumCircuit(psi.num_qubits,name='U')
U.x(0)
U.z(1)


#----
shots=10000
```
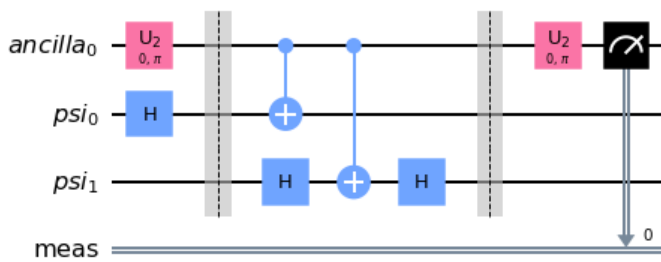
```
circ=hadamardTestCircuit(U,psi)
circ.decompose().draw()
```
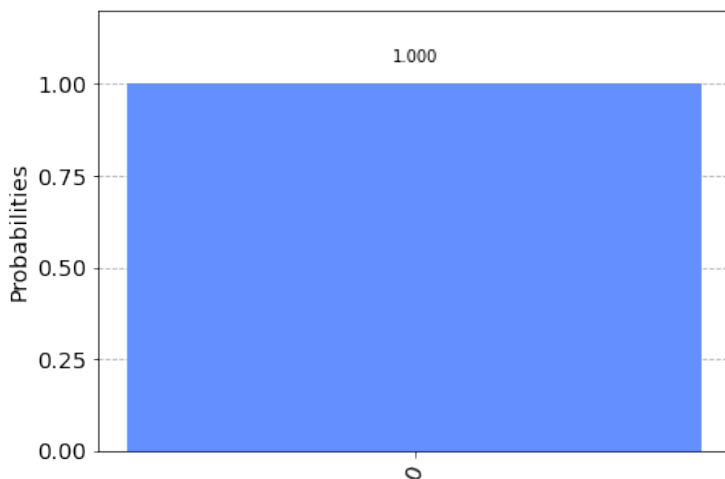


```python
simulator = BasicAer.get_backend('qasm_simulator')
counts = execute(circ, backend=simulator,shots=shots).result().get_counts()
print("Histogram Hadamard test for calculating Re(⟨ψ|U|ψ⟩)")
re=0.0
if '0' in counts:
    re=counts['0']
if '1' in counts:
    re-=counts['1']
re=re/shots
print("Re(⟨ψ|U|ψ⟩) =",re)
plot_histogram(counts)
```

Histogram Hadamard test for calculating Re(⟨ψ|U|ψ⟩)

Re(⟨ψ|U|ψ⟩) = 1.0


```
circ=hadamardTestCircuit(U,psi,True)
circ.draw()
```
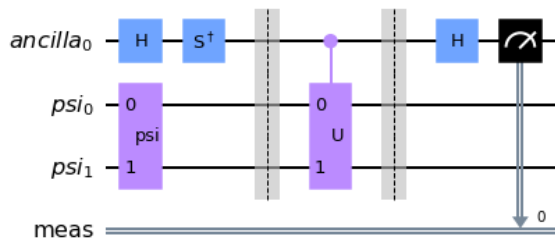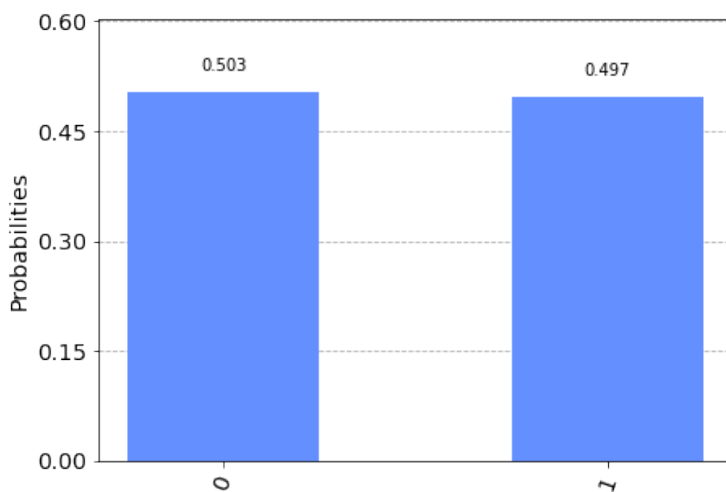
```python
simulator = BasicAer.get_backend('qasm_simulator')
counts = execute(circ, backend=simulator,shots=shots).result().get_counts()
print("Histogram Hadamard test for calculating Imag(⟨ψ|U|ψ⟩)")
im=0.0
if '0' in counts:
    im=counts['0']
if '1' in counts:
    im-=counts['1']
im=im/shots
print("Imag(⟨ψ|U|ψ⟩) =",im)
plot_histogram(counts)
```

Histogram Hadamard test for calculating Imag(⟨ψ|U|ψ⟩)

Imag(⟨ψ|U|ψ⟩) = 0.0058


```python
print("Expectation value: ⟨ψ|U|ψ⟩ =",complex(re,im))
```

Expectation value: ⟨ψ|U|ψ⟩ = (1+0.0058j)

Verification: $(Z \otimes X) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix}$; $|\psi\rangle = |0+\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$; as a result:

Claudio Fontana 27/07/2022

$$\langle\psi|ZX|\psi\rangle = \frac{1}{2}\begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}\cdot\begin{bmatrix}0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 0 & -1\\ 0 & 0 & -1 & 0\end{bmatrix}\cdot\begin{bmatrix}1\\ 1\\ 0\\ 0\end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}\cdot\begin{bmatrix}1\\ 1\\ 0\\ 0\end{bmatrix}$$

$$= 1$$

Q.E.D.

Comparison with the method proposed by Ryan LaRose

```
#see: https://github.com/rmlarose/QuIC-
Seminar/blob/master/fall2019/06vqas/variational_algorithms.ipynb
#For Qiskit's ordering convention see
#https://github.com/Qiskit/qiskit-terra/issues/1148
#https://quantumcomputing.stackexchange.com/questions/15830/qiskit-order-of-
multiplication-and-tensor-product
import qiskit as qis
import numpy as np


"""Helper function to evaluate the expectation of any valid Pauli string."""
def expectation_circuit(circuit: qis.QuantumCircuit, pauli_string: str) -
> qis.QuantumCircuit:
    """Returns a circuit to compute expectation of the Pauli string in the
    state prepared by the input circuit.

    Args:
        circuit: Prepares the state |\psi> from |0>.
        pauli_string: String (tensor product) of Paulis to evaluate
                      an expectation of. The length of pauli_string
                      must be equal to the total number of qubits in
                      the circuit. (Use identities for no operator!)
    """
    temp = circuit.copy()

    if len(circuit.qregs) != 1:
        raise ValueError("Circuit should have only one quantum register.")
    if len(circuit.cregs) != 1:
        print("# cregs =", len(circuit.cregs))
        raise ValueError("Circuit should have only one classical register.")
```

Claudio Fontana 27/07/2022

```python
    qreg = circuit.qregs[0]
    creg = circuit.cregs[0]
    nqubits = len(qreg)
    pauli_string = pauli_string.upper().strip()

    if len(pauli_string) != nqubits:
        raise ValueError(
            f"Circuit has {nqubits} qubits but pauli_string has {len(pauli_string)} operators."
        )

    for (qubit, pauli) in enumerate(pauli_string[::-1]): #reverse string for little-endian qiskit's ordering
        if pauli == "I":
            continue
        elif pauli == "X":
            temp.h(qreg[qubit])
            temp.measure(qreg[qubit], creg[qubit])
        elif pauli == "Y":
            temp.s(qreg[qubit])
            temp.h(qreg[qubit])
            temp.measure(qreg[qubit], creg[qubit])
        elif pauli == "Z":
            temp.measure(qreg[qubit], creg[qubit])
        else:
            raise ValueError(f"{pauli} is an invalid Pauli string key. Should be I, X, Y, or Z.")

    return temp


"""Function to execute the circuit and do the postprocessing."""
def run_and_process(circuit: qis.QuantumCircuit, shots: int = 10000) -> float:
    """Runs an 'expectation circuit' and returns the expectation value of the
    measured Pauli string.

    Args:
        circuit: Circuit to execute.
        shots: Number of circuit executions.
    """

    # Execute the circuit
    backend = qis.BasicAer.get_backend("qasm_simulator")
    job = qis.execute(circuit, backend, shots=shots)
    counts = job.result().get_counts()

    # Do the postprocessing
    val = 0.0
    for bitstring, count in counts.items():
        sign = (-1) ** bitstring.count("0")
```

Claudio Fontana 27/07/2022

```
        val += sign * count

    return val / shots

qreg = qis.QuantumRegister(2)
creg = qis.ClassicalRegister(2)
psi = qis.QuantumCircuit(2)

#----
psi.h(0)
#----
circ = qis.QuantumCircuit(qreg, creg)
circ.append(psi,qreg)
circ.barrier()
pauli = 'ZX'

exptCirc=expectation_circuit(circ, pauli)
print("Expectation value:",run_and_process(exptCirc,10000))
exptCirc.decompose().draw()
```
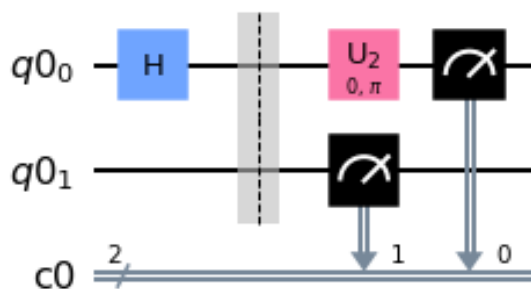
```
Expectation value: 1.0
```


Conclusion:

The results are consistent.

See also:

[QuIC-Seminar/variational_algorithms.ipynb at master · rmlarose/QuIC-Seminar · GitHub](#)

[Variational Quantum Linear Solver (qiskit.org)](#);

[Variational Quantum Linear Solver — PennyLane;](#)

[Hadamard test (quantum computation) - Wikipedia](#)

Claudio Fontana 27/07/2022