Komal Jindal
8 February 2021

Quantum Simulator

Gates available in simulator

Single qubit gates:

- "X" gate (Quantum bit flip gate)
- "H" gate (Hadamard gate)
- "I" gate (Identity)
- "Z" gate (Phase flip gate)
- "Y" gate
- "S" gate
- "T" gate
- "Sdag" gate (conjugate-transpose of "S" gate)
- "Tdag" gate (conjugate-transpose of "T" gate)

Code example:

```
qc = simulator.get_ground_state(3) #initialize state vector
circuit = [ { "gate": "X", "target": [0] },
            { "gate": "H", "target": [0] },
            { "gate": "I", "target": [1] },
            { "gate": "Z", "target": [1] },
            { "gate": "Y", "target": [1] },
            { "gate": "S", "target": [2] },
            { "gate": "T", "target": [1] }]
```

- "RX" gate: Rotation about x-axis by angle theta.

- "RY" gate: Rotation about y-axis by angle theta.

- "U1" gate (Unitary operator with 1 parameter) associated with following matrix:

$$U_1 = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}$$

Value of lambda must be passed as parameter. This is equivalent to rotation about z-axis by lambda.

- "U2" gate (Unitary operator with 2 parameters) associated with following matrix:

$$U_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i\lambda+i\phi} \end{bmatrix}$$

Value of lambda and phi must be passed as parameter.

- "U3" gate (Unitary operator with 3 parameters) associated with following matrix:

$$U_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+i\phi}\cos(\theta/2) \end{bmatrix}$$

Value of lambda, phi and theta must be passed as parameter.

Code example:

```
qc = simulator.get_ground_state(2)
circuit = [ {"gate":"RX","target":[0],"params":{"theta":pi/
2}},
{"gate":"U3","target":[0],"params":{"theta":pi/2,"phi":pi/
4,"lambda":pi/5}},
{"gate":"U1","target":[0],"params":{"lambda":pi}}]
```

Target should be 1-D list having only one element specifying target qubit in circuit, qubits indexing starting from 0.

Two qubit gates:

"CNOT" gate (Controlled not)

Code example:

```
qc = simulator.get_ground_state(2)
circuit = [ {"gate":"H","target":[0]},
            {"gate":"CNOT","target":[0,1]}]
```

Target should be 1-D list having only two elements. First one specifying control qubit and second as target qubit in circuit, qubits indexing starting from 0.

Functions in simulator

simulator.get_ground_state(num_qubits)

This function returns state vector for circuit having num_qubits qubits initialised to all zero state.

num_qubits should of type int and state vector returned is 1-D numpy array.

simulator.measure_state(state_vector)

This function measures state_vector passed and returns output binary number using weighted random function.

state_vector should be 1-D numpy array having amplitude for all possible binary states (length in powers of two).

simulator.get_counts(state_vector,num_shots)

This function measures state_vector, num_shots times using weighted random. And returns dictionary having binary string as key and number times that binary value measured as value.

num_shots should of type int and state vector as 1-D numpy array having amplitude for all possible binary states.

simulator.get_parameterized_gate(gate, params)

This function will return a 2-D numpy array (2-D matrix) corresponding to the gate given and params passed.

gate should be a string out of “U1”, “U2”,”U3”,”RX”,”RY” and params as dictionary having required parameters. Go to single qubit gates section for more information on params.

simulator.get_operator(total_qubits,gate,target_qubits,params)

This function will return quantum operator (matrix) corresponding to gate specified acting on target_qubits. Square matrix will be returned in form of numpy 2D array of dimensions (2^total_qubits) X (2^total_qubits).

total_qubits should be of int type. gate should be string type from any of gates given in chapter1. Target should be 1-D list. params is optional, required only for parameterised gates. params should be of dictionary type given all required parameters.

simulator.run_program(state_vector,circuit,global_params)

This function will read the circuit and apply transformations to state_vector. And, will return transformed state_vector.

global_parameters are optional, only need to be passed if parameters in “U1”,”U2”,”U3” gates are specified as string and need to be replaced by global parameters. As in case of optimisation algorithms.

state_vector must be 1-D numpy array. circuit is list of dictionaries, each dictionary having all required specifications for a gate. gates are applied in same order as they appear in circuit list. global_params must be a dictionary having all global parameters corresponding to all specified in general U gates.

Code example:

```
import simulator
import numpy as np
from math import pi

qc = simulator.get_ground_state(2)
circuit = [{"gate":"U3","target":[0],"params":
{"theta":"global1","phi":"global2","lambda":pi/5}},
          {"gate":"U1","target":[0],"params":{"lambda":pi}}]
result_vector = simulator.run_program(qc,circuit,
{"global1":pi/2,"global2":pi/4})
result = simulator.get_counts(result_vector,1000)
print(result)

unitary_gate = simulator.get_operator(1,"RX",[0],{"theta":
pi})
print(unitary_gate)
```

Software stack requirements

- Python version: 3.8.5
- Numpy version: 1.19.2
- Random module of python
- Math module of python