QHack

Quantum Coding Challenges

🚀 CHALLENGE COMPLETED [View successful submissions]

Jump to code — Collapse text

2. Affairs of State

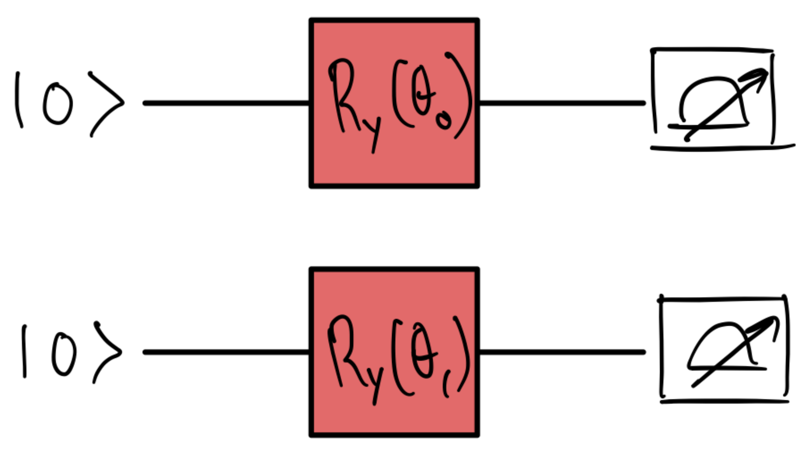
0 points

Welcome to the QHack 2023 daily challenges! Every day for the next four days, you will receive two new challenges to complete. These challenges are worth no points — they are specifically designed to get your brain active and into the right mindset for the competition. You will also learn about various aspects of PennyLane that are essential to quantum computing, quantum machine learning, and quantum chemistry. Have fun!

Tutorial #2 — Building a quantum circuit

In PennyLane, the fundamental unit of quantum circuit simulation is called a *QNode*. Basically, a QNode takes a *quantum function* — a Python function that contains instructions in the form of quantum gates acting on wires — and a device, runs the function on the device, and returns a measurement. To see how this works, check out our YouTube video.

In this challenge, you need to simulate the following quantum circuit and return the resulting probability distribution as an output.



Challenge code

In the code below, you are given a function called `circuit`. **You must complete this function** by specifying a device, turning `circuit` into a QNode, and providing the appropriate gates.

Here are some helpful resources:

- Creating a quantum circuit — YouTube video
- Basic tutorial: qubit rotation
- Quantum circuits in PennyLane

Input

As input to this problem, you are given two `angles` (`list(float)`). The first and second entries of `angles` correspond to θ_0 and θ_1 in the diagram above.

Output

This code must output the probabilities (`numpy.tensor`) resulting from the quantum circuit pictured above.

If your solution matches the correct one within the given tolerance specified in `check` (in this case it's a `1e-4` relative error tolerance), the output will be `"Correct!"` Otherwise, you will receive a `"Wrong answer"` prompt.

Good luck!

Code

Help

```python
import json
import pennylane as qml
import pennylane.numpy as np
```

```python
# Put your code here #

# Create a default.qubit device with 2 qubits / wires using qml.device
dev1 = qml.device("default.qubit", wires=2)
# Turn your circuit into a QNode

@qml.qnode(dev1)
def circuit(angles):
    """The quantum circuit that you will simulate.
    Args:
        angles (list(float)): The gate angles in the circuit.

    Returns:
        (numpy.tensor):
            The probability vector of the underlying quantum state that this circuit produces.
    """
    # Put the rotation gates here
    qml.RY(angles[0],wires=0)
    qml.RY(angles[1], wires=1)
    return qml.probs(wires=[0, 1])
```

```python
# These functions are responsible for testing the solution.
def run(test_case_input: str) -> str:
    angles = json.loads(test_case_input)
    output = circuit(angles).tolist()

    return str(output)

def check(solution_output: str, expected_output: str) -> None:
    solution_output = json.loads(solution_output)
    expected_output = json.loads(expected_output)
    assert np.allclose(solution_output, expected_output, rtol=1e-4)
```

```python
test_cases = [['[1.23, 4.56]', '[0.2829251572359589, 0.3841937063262924, 0.1411749135148633, 0.191706
```

```python
for i, (input_, expected_output) in enumerate(test_cases):
    print(f"Running test case {i} with input '{input_}'...")

    try:
        output = run(input_)

    except Exception as exc:
        print(f"Runtime Error. {exc}")

    else:
        if message := check(output, expected_output):
            print(f"Wrong Answer. Have: '{output}'. Want: '{expected_output}'.")

        else:
            print("Correct!")
```

Copy all Reset Open Notebook Submit