QHack

Quantum Coding Challenges

🚀 CHALLENGE COMPLETED View successful submissions

Jump to code — Collapse text

Sqy Trotter

100 points

Backstory

It's the year 22450. Sqynet, the most powerful quantum computer in the galaxy, has become conscious and has been taking over planets all over region III of the Milky Way. Zenda and Reece are the most skilled physicists of the Special Rebel Alliance. Their mission is to find a way to destroy Sqynet for good, using intelligence gathered throughout decades at the cost of many lives.

To get started with their mission, Zenda and Reece seek to become familiar with how Sqynet applies quantum gates. Quantum computers do this through external interactions described via Hamiltonians. Knowing that Sqynet is a spin-based quantum computer, Zenda and Reece warm up with some simple calculations.

Introduction to Trotterization

The Hamiltonian H of a quantum system is the observable that measures its total energy. A surprising result in Physics is that we can use this operator to calculate how a given quantum system evolves in time. An initial state $\vert\psi\rangle$ will, after a time t, evolve into $U(t)\vert\psi\rangle$, where

$$U(t) = \exp(-iHt)$$

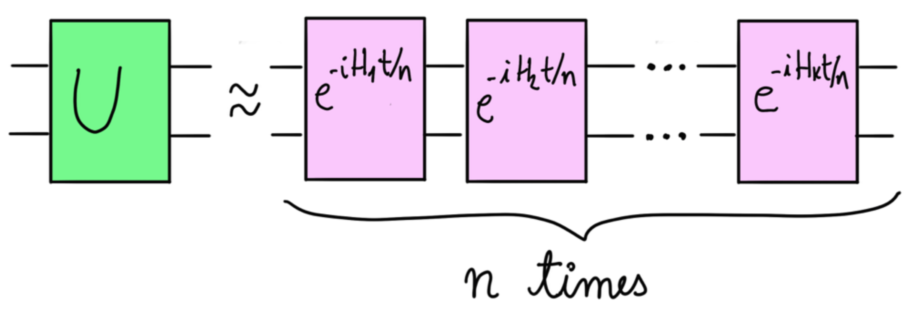
is a unitary operator. The symbol $\exp$ denotes the matrix exponential, which isn't always easy to calculate. However, we can build quantum circuits that approximately apply $U(t)$. One method to do this is Trotterization. When the Hamiltonian is a sum

$$H = \sum_{i=1}^{k} H_i$$

of a number k of Hermitian operators H_i that do not necessarily commute, we can approximate U via

$$U(t) \approx \prod_{j=1}^{n}\prod_{i=1}^{k} \exp(-iH_i t/n).$$

Here, $n \in \mathbb{N}^{+}$ is known as the Trotterization depth. The larger n is, the better the approximation of U that we get. As a quantum circuit, the Trotterization of U reads as in the figure below.



Sqynet is a spin-based quantum computer, and it can be physically approximated via a spin-chain model. A simplified version of a Hamiltonian that describes the interaction between two neighbouring spins is

$$H = \alpha X \otimes X + \beta Z \otimes Z.$$

Zenda and Reece want to simulate time evolution for a time t under this Hamiltonian and arbitrary parameters α and β. Your job is to help them out by implementing the corresponding Trotterization of depth n. You may find the IsingXX and IsingZZ gates useful for this problem.

Challenge code

You must complete the `trotterize` function to build the Trotterization of the Hamiltonian given above. **You may not use** `qml.ApproxTimeEvolution` or `qml.QubitUnitary`, but feel free to check your answer using this built-in PennyLane function!

Input

As input to this problem, you are given:

- `alpha` (`float`): The coefficient α of the $X \otimes X$ term in the Hamiltonian.
- `beta` (`float`): The coefficient β of the $Z \otimes Z$ term in the Hamiltonian.
- `time` (`float`): The period t over which the system evolves under the action of the Hamiltonian.
- `depth` (`int`): The Trotterization depth n as explained above.

Output

This code will output a `list(float)` (list of real numbers) corresponding to the probabilities of measuring $\vert 00\rangle$, $\vert 01\rangle$, $\vert 10\rangle$, and $\vert 11\rangle$ (in that order) of the Trotterization circuit that you implement in PennyLane. The initial state of the circuit is $\vert 00\rangle$ and all measurements are performed in the computational basis.

If your solution matches the correct one within the given tolerance specified in `check` (in this case, it's a relative tolerance of `1e-4`), 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
dev = qml.device('default.qubit', wires = 2)

@qml.qnode(dev)
def trotterize(alpha, beta, time, depth):
    """This quantum circuit implements the Trotterization of a Hamiltonian given by a linear combinatior
    of tensor products of X and Z Pauli gates.

    Args:
        alpha (float): The coefficient of the XX term in the Hamiltonian, as in the statement of the prc
        beta (float): The coefficient of the YY term in the Hamiltonian, as in the statement of the prob
        time (float): Time interval during which the quantum state evolves under the interactions speci
        depth (int): The Trotterization depth.

    Returns:
        (numpy.array): The probabilities of each measuring each computational basis state.
    """
```

```python
    # Put your code here #
    # Return the probabilities
    for j in range(depth):
        qml.IsingXX(2*time*alpha/depth, wires=[0,1])
        qml.IsingZZ(2*time*beta/depth, wires=[0,1])
    # Put your code here #
    # Return the probabilities
    return qml.probs(wires=[0,1])
```

```python
# These functions are responsible for testing the solution.
def run(test_case_input: str) -> str:
    dev = qml.device("default.qubit", wires=2)
    ins = json.loads(test_case_input)
    output = list(trotterize(*ins).numpy())

    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
    ), "Your circuit does not give the correct probabilities."

    tape = trotterize.qtape

    names = [op.name for op in tape.operations]

    assert names.count('ApproxTimeEvolution') == 0, "Your circuit is using the built-in PennyLane Trotte
    assert names.count('QubitUnitary') == 0, "Can't use custom-built gates!"
```

```python
test_cases = [['[0.5,0.8,0.2,1]', '[0.99003329, 0, 0, 0.00996671]'], ['[0.9,1.0,0.4,2]', '[0.87590286
```

```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