🚀 **CHALLENGE COMPLETED** View successful submissions

Jump to code — Collapse text

6. Hamiltonian Sandwich

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 #6 — Hamiltonians

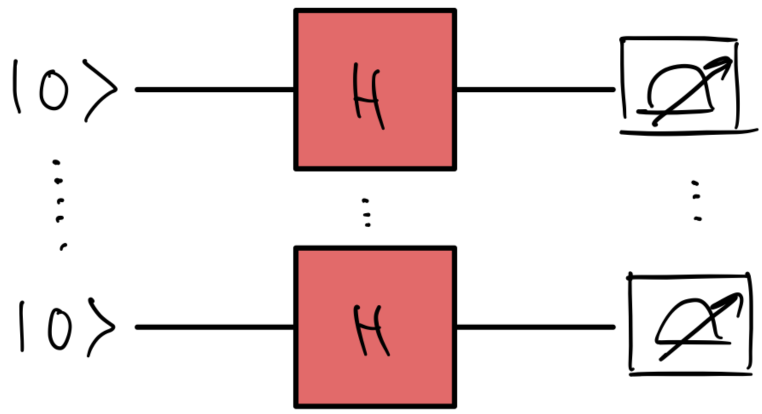
The Hamiltonian is the energy observable for a quantum system, and a quintessential component in many quantum algorithms. How do we implement Hamiltonians in PennyLane? You'll be tested on this in this challenge.

You will be tasked with creating the Hamiltonian

$$H = \frac{1}{3}\sum_{i<j} X_i X_j - \sum_{i=0}^{n-1} Z_i,$$

where n is the number of qubits, X_i and Z_i are the familiar Pauli X and Z operators, respectively, and $\sum_{i<j}$ denotes a sum over all pairs (e.g. for $n = 3$, the pairs are $(i,j) = (0,1), (0,2), (1,2)$). Note that we're indexing from 0!

In this challenge, you need to create the following quantum circuit simulation that returns the expectation value of this Hamiltonian.



To be clear, each wire represents n qubits, and $|0\rangle$ really means $|0\rangle^{\otimes n}$, i.e. the $|0\rangle$ state for each of these n qubits. Also, be mindful that the H gates represent the Hadamard gate, not the Hamiltonian (which is not unitary, in general)!

Challenge code

In the code below, you must complete two functions:

- `hamiltonian`: responsible for creating the Hamiltonian in question for a general number of qubits (`num_wires`). **You must complete this function.**
- `expectation_value`: simulates the circuit in question and returns the expectation value of the Hamiltonian in question. **You must complete**

this function by creating a QNode within this function that returns the expectation value of the Hamiltonian.

Here are some helpful resources and hints:

- The $X_i X_j$ term, mathematically, denotes a *tensor product* between the two Pauli-X operators. Here are some ways you can perform this in PennyLane:
  - use the `@` operator to take the tensor product between operators;
  - use `qml.prod`.
- `qml.Hamiltonian`
- Operator arithmetic

Input

As input to this problem, you are given the number of qubits n, `num_wires` (`int`).

Output

This code must output the expectation value of the Hamiltonian (`float`).

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
```

```
def hamiltonian(num_wires):
    """A function for creating the Hamiltonian in question for a
    number of qubits.

    Args:
        num_wires (int): The number of qubits.

    Returns:
        (qml.Hamiltonian): A PennyLane Hamiltonian.
    """

    # Put your solution here #
    return

def expectation_value(num_wires):
    """Simulates the circuit in question and returns the expecta
    Hamiltonian in question.

    Args:
        num_wires (int): The number of qubits.

    Returns:
        (float): The expectation value of the Hamiltonian.
    """

    # Put your solution here #

    # Define a device using qml.device
    dev =

    @qml.qnode(dev)
    def circuit(num_wires):
        """A quantum circuit with Hadamard gates on every qubit
        the expectation value of the Hamiltonian in question.
        """

        # Put Hadamard gates here #

        # Then return the expectation value of the Hamiltonian u
        return
```

```python
    return circuit(num_wires)

```

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

    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

```

```python
test_cases = [['8', '9.33333']]
```

```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: '{expe

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

Copy all

Submit

Open Notebook

Reset