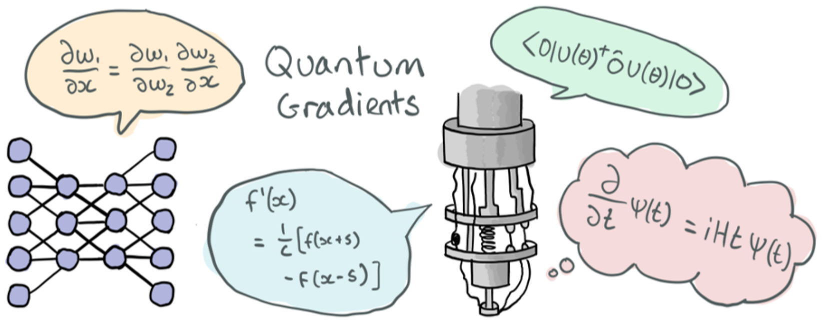

$\frac{\partial w_1}{\partial x} = \frac{\partial w_1}{\partial w_2}\frac{\partial w_2}{\partial x}$
Quantum Gradients
$\langle 0|U(\theta)^\dagger \hat{O} U(\theta)|0\rangle$
$f'(x) = \frac{1}{2}[f(x+s) - f(x-s)]$
$\frac{\partial}{\partial t}\psi(t) = iHt\,\psi(t)$

Quantum Gradients: Higher-Order Derivatives [200 points]

Version: 1

NOTE: Coding templates are provided for all challenge problems at this link. You are strongly encouraged to base your submission off the provided templates.

Overview: Quantum Gradients challenges

In this set of challenges, you'll explore various methods for computing gradients of variational quantum circuits.

Variational quantum circuits are quantum circuits that apply a parametrized unitary $U(\theta)$ to an initial state $|\psi\rangle$, and output a measurement statistic, such as the expectation value (average value) of a measurement operator $\hat{O}$. Crucially, the parameters (θ) of the circuit and the output measurement statistics are both real classical data, and the output is deterministic—given a specific parameter input θ, the expectation value will always be the same. Note that when running on quantum hardware, we estimate expectation values using a finite set of samples, so we might see some small fluctuations, but in the limit of infinite samples the expectation value is fixed. Quantum simulators allow us to estimate expectation values exactly (up to machine precision).

Variational quantum circuits are therefore differentiable functions, that map gate parameters to an expectation value. There are various ways we can differentiate a variational quantum circuit:

1. **Numerical differentiation:** We can treat the circuit as a mathematical black box, and vary the parameters an infinitesimal amount using the finite-difference method. Numerical differentiation is only an approximation and can be unstable, but it is one available technique for near-term quantum

hardware.

2. **The parameter-shift rule:** The parameter-shift rule provides a method of computing **exact** gradients of variational quantum circuits on near-term hardware, by shifting each parameter θ by a **constant**, **large** amount. Unlike the finite-difference method, it is not an approximation, but it only works for a subset of quantum gates.

3. **Backpropagation:** Backpropagation, or "reverse-mode" differentiation, powers machine-learning frameworks such as TensorFlow and PyTorch. During function execution, intermediate computations are stored, and later accumulated to produce the gradient. This introduces only a constant overhead to compute the gradient, making it more efficient than the other methods. This can easily be applied to quantum computing simulations, and there are even versions of backpropagation that are more efficient if the computation is unitary, like it is with quantum computing. However, backpropagation is not compatible with hardware, since we would need to 'peek' at the quantum state during execution.

Problem statement [200 points]

In this task, you will explore higher-order derivatives and compute the *Hessian* matrix.

Consider the parameter-shift rule,

$$\frac{\partial f}{\partial \theta_i} = \frac{f(\theta_i + s\hat{\mathbf{e}}_i) - f(\theta_i - s\hat{\mathbf{e}}_i)}{2\sin(s)},$$

where $f(\theta) = \langle\psi|\mathbf{U}^\dagger(\theta)\hat{\mathbf{O}}\mathbf{U}(\theta)|\psi\rangle$ is the output of the variational circuit, and $\hat{\mathbf{e}}_\mathbf{i}$ the ith unit vector.

If we want to compute the **second derivative** $\frac{\partial f}{\partial\theta_i\partial\theta_j}$, we can simply differentiate the parameter-shift rule again, and apply the product rule. This will lead to an expression with **four** circuit evaluations for the double derivative with respect to parameters θ_i and θ_j.

Thus, a circuit with p parameters will require $2p$ circuit evaluations to compute the gradient, and then another $4p^2$ evaluations to compute the Hessian matrix! However, if we are clever with our choice of the shift s, we can significantly reduce the number of evaluations required to compute both the gradient and Hessian, by more than half.

Given a variational quantum circuit QNode `circuit`, your task is to write snippets of code that:

- Compute the gradient of the circuit using the parameter-shift rule by hand—do not use PennyLane's built-in gradient methods!

- Compute the Hessian of the circuit using the parameter-shift rule, also by hand.

Further, your solution must compute these values using *exactly 51 quantum device evaluations.*

Guidance is provided in the problem template.

If you're stuck, you might find the following hint helpful:

- Can you re-use any of the evaluations made when computing the gradient to compute the Hessian?

Input

The problem template contains a function `gradient_200`, that accepts a real NumPy array of trainable parameters `weights` of size `(5,)`. This function must compute the gradient of the variational circuit, as well as the Hessian matrix.

Output

The return values should be a tuple, containing both:

- The gradient: a real NumPy array of size `(5,)`.
- The Hessian: a real NumPy array of size `(5, 5)`.

Acceptance Criteria

In order for your submission to be judged as "correct":

- The outputs generated by your solution when run with a given `.in` file must match those in the corresponding `.ans` file to within the `Tolerance` specified below. You solution will also output the differentiation method that was used, but this output is not included in the `.ans` file. As long as you used the parameter-shift method for your solution then this output should be correct.

- Your solution must take no longer than the `Time limit` specified below to produce its outputs.

You can test your solution by passing the `#.in` input data to your program as stdin and comparing the output to the corresponding `#.ans` file:

```
python3 quantum_gradients_200_template.py < 1.in
```

WARNING: Don't modify any of the code in the template outside of the `# QHACK #` markers, as this code is needed to test your solution. Do not add any print statements to your solution, as this will cause your submission to fail.

Specs
Tolerance: **0.05 (5%) + correct method**
Timelimit: **80 s**

Version History

Version 1: Initial document.