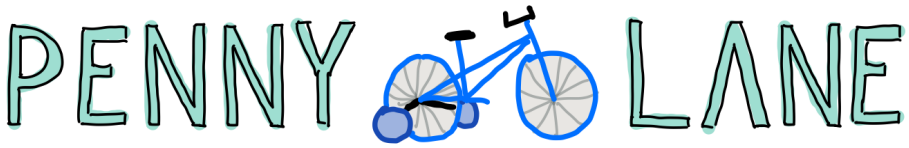

PennyLane 101: Bitflip Error Code [500 points]

Version: 1

PennyLane 101

The **PennyLane 101** challenges will introduce quantum computing concepts with PennyLane. Whether you're coming from an advanced quantum computing background, or you've never evaluated a quantum circuit before, these challenge questions will be a great start for you to learn how quantum computing works using PennyLane. Beyond these five questions in this category, there are well-developed demos and tutorials on the PennyLane website that are a good resource to fall back on if you are stuck. We also strongly recommend consulting the PennyLane documentation to see an exhaustive list of available gates and operations!

Problem statement [500 points]

In this challenge, you will foray into the world of error correction by creating an error-correcting code in PennyLane that will detect bit-flip errors. Today's quantum devices are noisy, and it is because of noise that we must develop codes that can correct any errors that are induced by it. Suppose that we wish to perfectly transmit a given quantum state through a quantum circuit. It could happen that during the transmission, the quantum state was altered, say, by a bit-flip error on one qubit/wire like in Figure 1.

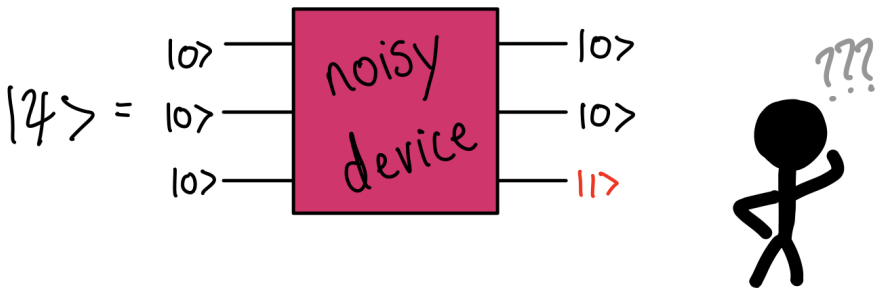

Figure 1: Bit-flip error

Luckily, there are ways to detect if such an error occurred. You must devise a protocol for doing so. Specifically, your code will do the following:

- Given an arbitrary one-qubit state, $|\psi\rangle = \alpha|0\rangle + \sqrt{1-\alpha^2}|1\rangle$, encode this state in a "logical" qubit state with three qubits.
- With $|\psi\rangle$, perform some pre- and post-processing around a given bit-flip error channel (`qml.BitFlip`) to probabilistically identify where (if at all!) a bit-flip error occurred in transmitting the state $|\psi\rangle$.

In PennyLane, there is built-in functionality to simulate noisy devices. We can model noisy devices through quantum channels. For this challenge, we will employ the `qml.BitFlip` error channel.

The provided template file `bitflip_error_template.py` contains a few functions. The `density_matrix` function creates the state $|\psi\rangle \otimes |00\rangle$ as a density matrix (i.e., $\hat{\rho} = |\psi\rangle\langle\psi|$), where the last two qubits/wires are ancillary (extra) qubits. The `circuit` function is a quantum function that processes the prepared state. The `error_wire` function interprets the output of the `circuit` function and returns a probability vector that reveals the error channel statistics. As input, you will be given the probability `p` for which a bit-flip error may occur, the parameter α that defines the input state, and the qubit/wire that is the (potential) victim of a bit-flip error.

Input

- `list`: A list containing the probability p for which a bit-flip error occurs, the parameter α that defines the input state $|\psi\rangle$, and the wire (integer) that may or may not be the victim of a bit-flip error.

Output

- `list(float)`: A list revealing the statistics of the bit-flip error channel. In transmitting the state $|\psi\rangle \otimes |00\rangle$ through the circuit, a bit-flip error could occur on wires 1, 2, or 3, or no wire at all. The output of your code will be a length-4 `np.ndarray` called `error_readout` that is interpreted as the probability that qubit i had a bit-flip error, where $i = 0$ corresponds to no error at all.

For example, `[0.28, 0.0, 0.72, 0.0]` means a 28% chance no bitflip error occurs, but if one does occur it occurs on qubit #2 with a 72% chance.

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 `0.0001` tolerance specified below. To clarify, your answer must satisfy

$$\text{tolerance} \geq \left| \frac{\text{your solution} - \text{correct answer}}{\text{correct answer}} \right|.$$

- Your solution must take no longer than the `60s` 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 {name_of_file}.py < 1.in
```

WARNING: Don't modify the code outside of the `# QHACK #` markers in the template file, 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: **0.0001**
Time limit: **60 s**

Version History

Version 1: Initial document.