Introduction to Quantum Computing Pennylane: gates and states

Chris Heunen

What is Pennylane?

- *Software framework* for *differentiable quantum programming*

- Python library
- Just-in-time compilation
- Not a programming language

- Similar to TensorFlow and PyTorch for classical computation.
- Focused on training variational quantum circuits
- Plugins for execution on various quantum hardware platforms
  (including IBM Qiskit, Amazon Braket, Google Cirq, IonQ, Rigetti, Qrack GPU simulation)

- Developed by Xanadu
- Open-source
- https://pennylane.ai/codebook/

Installing

```
python3 -m venv ~/.venv/pennylane
~/.venv/pennylane/bin/pip install pennylane
~/.venv/pennylane/bin/pip install matplotlib
```

Hello.py

```
import pennylane
print("hello")
```

VSCode

install extension Python Environment Manager

add python environment ~/.venv/pennylane

Jupyter

```
~/.venv/pennylane/bin/pip install ipykernel
```

- install VSCode extension Jupyter


```
import pennylane as pl
from pennylane import numpy as np
import matplotlib.pyplot as plt
```

Exercise 1: Normalise state

```
ket_0 = np.array([1, 0])
ket_1 = np.array([0, 1])

def normalise(alpha, beta):
      """Given complex amplitudes alpha and beta for the |0> and |1> states,
      return a vector (np.array[complex]) of size 2 for the normalised state"""

      # Compute vector psi [a,b] based on alpha and beta such that |a|^2+|b|^2=1
      psi =

      return psi

print(normalise(2,0))
```

Exercise 2: Inner product

```
def innerproduct(phi, psi):
      """Compute the (complex) inner product between two
      normalised states (np.array[complex]) phi and psi"""

      # Compute the inner product of phi and psi
      z =

      return z

print(f"<0|0> = {innerproduct(ket_0, ket_0)}")
print(f"<0|1> = {innerproduct(ket_0, ket_1)}")
print(f"<1|0> = {innerproduct(ket_1, ket_0)}")
print(f"<1|1> = {innerproduct(ket_1, ket_1)}")
```

Exercise 3: Measurement

```
def measure_state(psi, n):
    """Simulate n quantum measurements of state psi, returning n samples 0 or 1"""

    # Compute the measurement outcome probabilities
    # Return a list of sample measurement outcomes
    # Hint: use numpy.random.choice

psi = normalise(1,1j)
print(measure(psi, 10))
```

Exercise 4: Unitary gates

```
def apply_unitary(U, psi):
    """Apply a unitary operation U to state psi"""

    # Apply U to psi and return the result
    phi =
    return phi

U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)
psi = ket_0
print(apply_unitary(U, psi))
```

Exercise 5: Baby quantum simulator

- Initialize a qubit in state
- Apply the a provided unitary gate U
- Simulate measuring the output state 100 times

```
def quantum_simulator(U, psi):
   """Use previous exercises to sample the result of applying gate U to state psi 100 times"""

   statistics =
   return statistics

U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)
psi = ket_0
print(quantum_simulator(U, psi))
```