gatesandstates-answers

October 2, 2024

[ ]:
```
import pennylane as pl
from pennylane import numpy as np
```

1 Exercise 1: Normalise states

[ ]:
```
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 = np.empty(2).astype(complex)
    length = np.sqrt(alpha*np.conjugate(alpha) + beta*np.conjugate(beta))
    psi[0]=alpha/length
    psi[1]=beta/length

    return psi

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

```
[1.+0.j 0.+0.j]
```

2 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 = np.conjugate(phi[0])*psi[0] + np.conjugate(phi[1])*psi[1]

    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)}")
```

```
<0|0> = 1
<0|1> = 0
<1|0> = 0
<1|1> = 1
```

3 Exercise 3: Measurement

```
[ ]: def measure(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
         p = np.abs(psi[0])
         outcomes = [ np.random.choice([0,1], p=[p,1-p]) for i in range(n) ]
         return outcomes

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

```
[0, 0, 1, 0, 0, 1, 1, 0, 0, 0]
```

4 Exercise 4: Measurement

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

         # Apply U to psi and return the result
         phi = np.empty(2).astype(complex)
         phi[0] = U[0,0]*psi[0] + U[1,0]*psi[1]
         phi[1] = U[0,1]*psi[0] + U[1,1]*psi[1]
         # or simply: phi = np.dot(U, psi)
         return phi

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

```
[0.70710678+0.j 0.70710678+0.j]
```

5 Exercise 5: Baby quantum simulator

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

    statistics = measure(apply_unitary(U, psi), 100)
    return statistics

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

```
[1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0]
```

[ ]:
```
```