HW5

February 23, 2023

[ ]:
```
import random
import numpy as np
from qutip import *
from qutip.measurement import measure

from qiskit.visualization import array_to_latex
```

[ ]:
```
H = ket("H") # horizontal state
V = ket("V") # vertical state
D = 1/np.sqrt(2) * (H + V) # Diagonal state (+45)
A = 1/np.sqrt(2) * (H - V) # Anti-Diagonal state (-45)

# Helper function to convert a basis into a string representation
def basis_to_str(B):
    if B == H:
        return "H"
    elif B == V:
        return "V"
    elif B == D:
        return "D"
    elif B == A:
        return "A"
    else:
        return "UNKNOWN"
```

[ ]:
```
class DecoyProtocol():
    def __init__(self, num_photons, frequency):
        self.num_photons = num_photons
        self.frequency = frequency
```

[ ]:
```
def GetCorrectBits(N_p, bob_measurements, bob_basis, alice_basis,␣
 ↪decoy_protocol=None):
    bob_measured_bits = []
    num_correct = 0
    num_incorrect = 0
    num_decoys = 0
    for i in range(1, N_p+1):
```

```
        if decoy_protocol != None and i % decoy_protocol.frequency == 0 and␣
↪bob_measurements[i][0] == 1 and bob_measurements[i][1] > 0:
            num_decoys += 1
            continue

        inner_product = (bob_basis[i].dag() * alice_basis[i])[0]
        if bob_measurements[i][0] == 1:
            bit = 1 if bob_basis[i] == V else 0

            print("Bob got click for photon {:02d} with basis {}. Sent bit was␣
↪{}.".format(i, basis_to_str(bob_basis[i]), bit))
            # Check if the bases are orthogonal. If so, then the measurement is␣
↪incorrect
            print("Inner product of Bob's basis and Alice's basis: {}".
↪format(inner_product))
            if inner_product > 0:
                bob_measured_bits.append(bit)
                num_correct += 1
                print("This measurement is correct.\n")
            else:
                num_incorrect += 1
                print("This measurement is incorrect.\n")


    print(f"\nBob recieved {num_correct} correct bits and {num_incorrect}␣
↪incorrect bits.")
    print(f"Error rate: {np.abs(num_correct - num_incorrect) / (num_correct +␣
↪num_incorrect)}")
    print(f"Bob's Efficiency: {((num_correct + num_incorrect) / N_p * 10)})") #␣
↪10% of the photons are lost
    if decoy_protocol != None:
        print(f"Number of decoys: {num_decoys}")
        print(f"Decoy Efficiency: {num_decoys / (N_p / decoy_protocol.
↪frequency)}")
    return bob_measured_bits
```

[ ]:
```
def GetRandomBasis(N_p, basis_states):
    basis = {}
    for i in range(1, N_p+1):
        basis[i] = random.choice(basis_states)
    return basis
```

[ ]:
```
# Function modeling Alice sending a message to Bob with N_p number of photons
def AliceMessage(N_p, basis, p2, decoy_protocol=None):
    sent_message = {}
    for i in range(1, N_p+1):
```

```
        num_photons_emitted = 0

        # Send decoy protocol photons at the given frequency
        if decoy_protocol != None and i % decoy_protocol.frequency == 0:
            num_photons_emitted = decoy_protocol.num_photons
        else:
            num_photons_emitted = 2 if random.uniform(0,1) <= p2 else 1
        sent_message[i] = (basis[i], num_photons_emitted)
    return sent_message
```

[ ]:
```
def BobReceiveMessage(N_p, message, measurement_basis, efficiency,␣
↪decoy_protocol=None):
    measurements = {}
    for i in range(1, N_p+1):
        # First, check if we recieve the photon at all
        if message[i][0] == None or message[i][1] <= 0:
            measurements[i] = (0,0)
            continue

        # Check protocol
        if decoy_protocol != None and i % decoy_protocol.frequency == 0:

            num_decoy_photons = message[i][1] * efficiency
            # Check that our decoy protocol photon count passes the efficiency␣
↪threshold
            if(num_decoy_photons> 0):
                measurements[i] = (1, num_decoy_photons)
            else:
                measurements[i] = (0, 0)
            continue

        # Only recieve photons with a given efficiency
        if random.uniform(0,1) <= efficiency:
            # Create a density matrix out of our basix
            cur_basis_op = ket2dm(measurement_basis[i])

            # Measure the projected eigen value/state with the given basis
            eigen_val, eigen_state = measure(message[i][0], cur_basis_op)
            measurements[i] = (round(eigen_val), message[i][1])
        else:
            # If we don't recieve the photon, then we measure 0
            measurements[i] = (0, 0)

    return measurements
```

[ ]:
```
def Eavesdrop(N_p, message, measurement_basis):
    eve_message = {}
    measurements = {}
    for i in range(1, N_p+1):
        # Create a density matrix out of our basix
        cur_basis_op = ket2dm(measurement_basis[i])

        # Measure the projected eigen value/state with the given basis
        eigen_val, eigen_state = measure(message[i][0], cur_basis_op)
        measurements[i] = round(eigen_val)

        photon_num = message[i][1]

        # Case where eve splits photons
        if photon_num > 1:
            eve_message[i] = (message[i][0], photon_num - 1)
        else:
            # Got a click. We for sure know the state
            if measurements[i] == 1:
                if measurement_basis[i] == V:
                    eve_message[i] = (D, photon_num)
                else:
                    eve_message[i] = (H, photon_num)
            # Don't have a click, so we stay silent
            elif measurements[i] == 0:
                eve_message[i] = (None, 0)

    return eve_message
```

0.1 1 B.i

Case where p2 = 0.2

[ ]:
```
N_p = 1000
p_2 = 0.2
efficiency = 0.1
alice_basis = GetRandomBasis(N_p, [H,D])
alice_message = AliceMessage(N_p, alice_basis, p_2)

eve_basis = GetRandomBasis(N_p, [V,A])
eve_message = Eavesdrop(N_p, alice_message, eve_basis)

bob_basis = GetRandomBasis(N_p, [V,A])
bob_measurements = BobReceiveMessage(N_p, eve_message, bob_basis, efficiency)
correct_bits = GetCorrectBits(N_p, bob_measurements, bob_basis, alice_basis)
```

```
Bob got click for photon 127 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
```

```
This measurement is correct.

Bob got click for photon 261 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 268 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 356 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 386 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 512 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 683 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 722 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 847 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 888 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.


Bob recieved 10 correct bits and 0 incorrect bits.
Error rate: 1.0
Bob's Efficiency: 0.1
```

In this case, Eve has a probability of splitting greater than the efficiency. So she's able to sufficiently split the quantum state and intercept the message without messing with the efficiency of the message.

0.2 1 B.ii

Case where p2=0.05

[ ]:
```
N_p = 1000
p_2 = 0.05
efficiency = 0.1
alice_basis = GetRandomBasis(N_p, [H,D])
alice_message = AliceMessage(N_p, alice_basis, p_2)

eve_basis = GetRandomBasis(N_p, [V,A])
eve_message = Eavesdrop(N_p, alice_message, eve_basis)

bob_basis = GetRandomBasis(N_p, [V,A])
bob_measurements = BobReceiveMessage(N_p, eve_message, bob_basis, efficiency)
correct_bits = GetCorrectBits(N_p, bob_measurements, bob_basis, alice_basis)
```

```
Bob got click for photon 205 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 254 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 430 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 470 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 491 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 744 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 976 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.


Bob recieved 7 correct bits and 0 incorrect bits.
Error rate: 1.0
```

```
Bob's Efficiency: 0.07
```

In this case, Eve does a worse job beause there's a higher probability that Alice is only sending a single photon. Therefore Eve can't split and has to remain silent more frequently. Thus, harming the efficiency of the message.

0.3 2

[ ]:
```
N_p = 1000
p_2 = 0.05
efficiency = 0.1
decoy_protocol = DecoyProtocol(10, 3)
alice_basis = GetRandomBasis(N_p, [H,D])
alice_message = AliceMessage(N_p, alice_basis, p_2, decoy_protocol)

eve_basis = GetRandomBasis(N_p, [V,A])
eve_message = Eavesdrop(N_p, alice_message, eve_basis)

bob_basis = GetRandomBasis(N_p, [V,A])
bob_measurements = BobReceiveMessage(N_p, alice_message, bob_basis, efficiency,␣
 ↪decoy_protocol)
correct_bits = GetCorrectBits(N_p, bob_measurements, bob_basis, alice_basis,␣
 ↪decoy_protocol)
```

```
Bob got click for photon 28 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 50 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 79 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 143 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 215 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 223 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.
```

```
Bob got click for photon 361 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 374 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 565 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 686 with basis V. Sent bit was 1.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 694 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.

Bob got click for photon 998 with basis A. Sent bit was 0.
Inner product of Bob's basis and Alice's basis: [[0.70710678+0.j]]
This measurement is correct.


Bob recieved 12 correct bits and 0 incorrect bits.
Error rate: 1.0
Bob's Efficiency: 0.12)
Number of decoys: 333
Decoy Efficiency: 0.9990000000000001
```

Now, we've added in a decoy protocol to send out 10 photon pulses every 3 iterations. This will allow bob to compare efficiencies between the normal incoming message photons and the decoy - allowing him to filter out wether or not there was a spy messing with his efficiency