Jordan-Wigner encoding

December 8, 2020

0.1 Jordan-Wigner encoding

0.1.1 1 Qubit

Jordan-Wigner enoding is a method for fermionic creation and annihilation operations. Let's start from one-qubit operations. Q is an annihilation operator, and Q+ is a creation operator. |0> is a vacancy state and |1> is a occupied state. In the first part, we will apply the operators Q and Q+ on |0> and |1> states and demonstrate the results.

[1]:
```
# Import the libraries
import numpy as np
```

[2]:
```
# Define the vacancy and the occupied states
vacancy = np.array([1,0])
occupied = np.array([0,1])

# Q = |0><1| = (X+iY)/2
Q = (np.array([[0,1],[1,0]]) + 1j*np.array([[0,-1j],[1j,0]]))/2

# Q|0>
annilation_zero = np.dot(Q,vacancy)
# Q|1>
annilation_one = np.dot(Q,occupied)

#print the output
print("annihilation operation on vancy state :")
print(annilation_zero, "which means the state disappeared!")
print("annihilation operation on occupied state :")
print(annilation_one, ", meaning |0> -> |1>")

# Q_dagger = |0><1| = (X-iY)/2
Q_dagger = (np.array([[0,1],[1,0]]) - 1j*np.array([[0,-1j],[1j,0]]))/2

# Q|0>
creation_dagger_zero = np.dot(Q_dagger,vacancy)
# Q|1>
creation_dagger_one = np.dot(Q_dagger,occupied)
```

```
#print the output
print("creation operation on vancy state :")
print(creation_dagger_zero, ", meaning |0> -> |1>")
print("annihilation operation on occupied state :")
print(creation_dagger_one, "which means the state disappeared!")
```

```
annihilation operation on vancy state :
[0.+0.j 0.+0.j] which means the state disappeared!
annihilation operation on occupied state :
[1.+0.j 0.+0.j] , meaning |0> -> |1>
creation operation on vancy state :
[0.+0.j 1.+0.j] , meaning |0> -> |1>
annihilation operation on occupied state :
[0.+0.j 0.+0.j] which means the state disappeared!
```

0.2 2 Qubits

Jordan-Wigner enoding is a method for fermionic creation and annihilation operations. In this section a two-fermion system represented by two qubits will be presented.
There will be creation operations and annihilation operations applied on q1 or q2.

|q1q2> will be initialized as either |00>, |01>, |10> or |11> and after the operations applied on the qubits, the results are shown as follow.

[3]:
```
##|q1q2> states

#create a 4X1 vector |00>
qubits00 = np.array([1,0,0,0])
#create a 4X1 vector |01>
qubits01 = np.array([0,1,0,0])
#create a 4X1 vector |10>
qubits10 = np.array([0,0,1,0])
#create a 4X1 vector |11>
qubits11 = np.array([0,0,0,1])


##Operator definition

# Pauli z
z = np.array([[1,0],[0,-1]])
#create a 4X4 Q(annihilation) matrix on q1
Q1_four = np.kron(Q,np.eye(2)) # annihilation on q1
#create a 4X4 Q(annihilation) matrix on q2
Q2_four = np.kron(z,Q) # annihilation on q2

#create a 4X4 Q+(creation) matrix on q1
Q1_dagger_four = np.kron(Q_dagger,np.eye(2)) # creation on q1
```

```python
#create a 4X4 Q+(creation) matrix on q2
Q2_dagger_four = np.kron(z,Q_dagger) # creation on q2


##Annihilation

# annihilation on q1 for |10>
annih_q1_qubits10 = np.dot(Q1_four, qubits10)
print("annihilation on q1 for |10>:", annih_q1_qubits10)

# annihilation on q1 for |11>
annih_q1_qubits01 = np.dot(Q1_four, qubits11)
print("annihilation on q1 for |11>:", annih_q1_qubits01)

# annihilation on q2 for |01>
annih_q2_qubits01 = np.dot(Q2_four, qubits01)
print("annihilation on q2 for |01>:", annih_q2_qubits01)

# annihilation on q2 for |11>
annih_q2_qubits11 = np.dot(Q2_four, qubits11)
print("annihilation on q2 for |11>:", annih_q2_qubits11)

print("--------------------------------------------------------------")

##Creation

# creation on q1 for |00>
create_q1_qubits00 = np.dot(Q1_dagger_four, qubits00)
print("creation on q1 for |00>:", create_q1_qubits00)

# creation on q1 for |01>
create_q1_qubits01 = np.dot(Q1_dagger_four, qubits01)
print("creation on q1 for |01>:", create_q1_qubits01)

# creation on q2 for |00>
create_q2_qubits00 = np.dot(Q2_dagger_four, qubits00)
print("creation on q2 for |00>:", create_q2_qubits00)

# creation on q2 for |10>
create_q2_qubits10 = np.dot(Q2_dagger_four, qubits10)
print("creation on q2 for |10>:", create_q2_qubits10)
```

```
annihilation on q1 for |10>: [1.+0.j 0.+0.j 0.+0.j 0.+0.j]
annihilation on q1 for |11>: [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
annihilation on q2 for |01>: [1.+0.j 0.+0.j 0.+0.j 0.+0.j]
annihilation on q2 for |11>: [ 0.+0.j  0.+0.j -1.+0.j  0.+0.j]
--------------------------------------------------------------
```

```
creation on q1 for |00>: [0.+0.j 0.+0.j 1.+0.j 0.+0.j]
creation on q1 for |01>: [0.+0.j 0.+0.j 0.+0.j 1.+0.j]
creation on q2 for |00>: [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
creation on q2 for |10>: [ 0.+0.j  0.+0.j  0.+0.j -1.+0.j]
```

0.3 Trivial Hamiltonian for 2 qubits

$H = \epsilon \sum_{i=1}^{2} a_i^\dagger a_i$, where ϵ is epsilon

[4]:
```
## Hamiltonian
epsilon = 1
H = epsilon* (np.dot(Q1_dagger_four, Q1_four) + np.dot(Q2_dagger_four, Q2_four))

# Hamiltonian on |11>
H_11 = np.dot(H, qubits11)
print(H_11, ": Hamiltonian on |11>")
```

```
[0.+0.j 0.+0.j 0.+0.j 2.+0.j] : Hamiltonian on |11>
```