Name: Okoye, Adunife Kizito
Student ID: 100611918
CSCI 4140U

Assignment 3

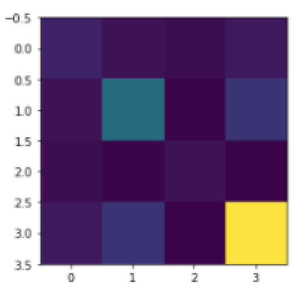
Steps

```
import qiskit.tools.jupyter

from qiskit import Aer
from qiskit.circuit.library import TwoLocal
from qiskit.aqua import QuantumInstance
from qiskit.finance.applications.ising import portfolio
from qiskit.optimization.applications.ising.common import sample_most_likely
from qiskit.finance.data_providers import RandomDataProvider
from qiskit.aqua.algorithms import VQE, QAOA, NumPyMinimumEigensolver
from qiskit.aqua.components.optimizers import COBYLA
import numpy as np
import matplotlib.pyplot as plt
import datetime
```

Version Information

Qiskit Software	Version
Qiskit	0.23.1
Terra	0.16.1
Aer	0.7.1
Ignis	0.5.1
Aqua	0.8.1
IBM Q Provider	0.11.1
System information	
Python	3.8.3 (default, Jul 2 2020, 11:26:31) [Clang 10.0.0]
OS	Darwin
CPUs	4
Memory (Gb)	16.0
	Sat Dec 05 23:12:52 2020 EST

```
plt.imshow(sigma, interpolation='nearest')
plt.show()
```



Start by constructing a QuadraticProblem for the optimization and data.

```
# Import a model from DOcplex
from docplex.mp.model import Model

# Name the model
ndl = Model('MinCut')

# Add a binary variable to the model for each node in the graph
x = ndl.binary_var_list('x{}'.format(i) for i in range(n))

# Define the objective function - more of a pseudo algorithm
q = 0.5
xt = np.transpose(x)
qxt = np.multiply(q,xt)
qxtE = np.multiply(qxt,sigma)
qxtEx = np.multiply(qxtE,x)
mt = np.transpose(mu)
mtx = np.multiply(mt,x)

object = np.subtract(qxtEx,mtx)

# Add an equality constraint
B = 2
ndl.add_constraint(ndl.sum(x) == B)

# And let's maximize it!
ndl.minimize(objective)

# Let's print the model
ndl.prettyprint()
```

Then you can basically follow the steps that we have used in the lecture. First, use a classical optimization solver to determine the correct answer. This will be your starting point. From there you can choose any of the quantum optimization techniques that we described in class. I found that Grover worked the best, but it is more work. You will need to experiment a bit with the parameters.

Remember, for many of these problems the first attempt doesn't always work out. You may need to run it several times and experiment with the parameters.

```
qp = QuadraticProgram()

# Put the model inside it
qp.from_docplex(ndl)

print(qp.export_as_lp_string())
```

```
\ This file has been generated by DOcplex
\ ENCODING=ISO-8859-1
\Problem name: MinCut

Minimize
 obj:
Subject To

Bounds
 0 <= x0 <= 1
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1

Binaries
 x0 x1 x2 x3
End
```

```
q = 0.5

budget = num_assets // 2
penalty = num_assets

qubitOp, offset = portfolio.get_operator(mu, sigma, q, budget, penalty)
def index_to_selection(i, num_assets):
    s = "{0:b}".format(i).rjust(num_assets)
    x = np.array([1 if s[i]=='1' else 0 for i in reversed(range(num_assets))])
    return x



def print_result(result):
    selection = sample_most_likely(result.eigenstate)
    value = portfolio.portfolio_value(selection, mu, sigma, q, budget, penalty)
    print('Optimal: selection {}, value {:.4f}'.format(selection, value))

    eigenvector = result.eigenstate if isinstance(result.eigenstate, np.ndarray) else result.eigenstate.to_matrix()
    probabilities = np.abs(eigenvector)**2
    i_sorted = reversed(np.argsort(probabilities))
    print('\n----------------- Full result ---------------------')
    print('selection\tvalue\t\tprobability')
    print('---------------------------------------------------')
    for i in i_sorted:
        x = index_to_selection(i, num_assets)
        value = portfolio.portfolio_value(x, mu, sigma, q, budget, penalty)
        probability = probabilities[i]
        print('%10s\t%.4f\t\t%.4f' %(x, value, probability))
```

```
solver = NumPyMinimumEigensolver(qubitOp)
result = solver.run()
print(result)
print_result(result)
```

```
{'eigenvalue': (-4.0088051883937315+0j), 'eigenstate': VectorStateFn(Statevector([0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+
0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j,
             0.+0.j, 0.+0.j],
            dims=(2, 2, 2, 2)), coeff=1.0, is_measurement=False)}
Optimal: selection [1 1 0 0], value 0.0000

----------------- Full result ---------------------
selection       value           probability
---------------------------------------------------
 [1 1 0 0]      0.0000          1.0000
 [1 1 1 1]      16.0179         0.0000
 [0 1 1 1]      4.0159          0.0000
 [1 0 1 1]      4.0193          0.0000
 [0 0 1 1]      0.0174          0.0000
 [1 1 0 1]      4.0150          0.0000
 [0 1 0 1]      0.0130          0.0000
 [1 0 0 1]      0.0164          0.0000
 [0 0 0 1]      4.0145          0.0000
 [1 1 1 0]      4.0030          0.0000
 [0 1 1 0]      0.0010          0.0000
 [1 0 1 0]      0.0048          0.0000
 [0 0 1 0]      4.0029          0.0000
 [0 1 0 0]      3.9981          0.0000
 [1 0 0 0]      4.0018          0.0000
 [0 0 0 0]      16.0000         0.0000
```

```python
backend = Aer.get_backend('statevector_simulator')
seed = 50

cobyla = COBYLA()
cobyla.set_options(maxiter=500)
ry = TwoLocal(qubitOp.num_qubits, 'ry', 'cz', reps=3, entanglement='full')
vqe = VQE(qubitOp, ry, cobyla)
vqe.random_seed = seed

quantum_instance = QuantumInstance(backend=backend, seed_simulator=seed, seed_transpiler=seed)

result = vqe.run(quantum_instance)

print_result(result)
```

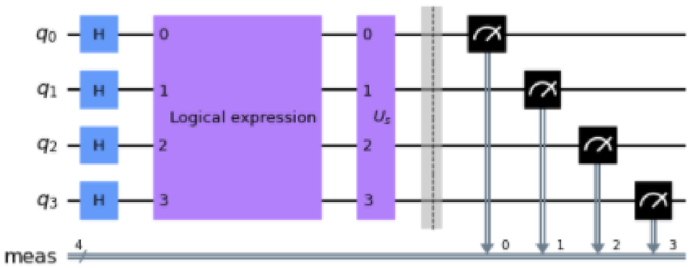
```
Optimal: selection [1. 0. 1. 0.], value 0.0048

----------------- Full result ---------------------
selection       value           probability
---------------------------------------------------
 [1 0 1 0]      0.0048          0.9013
 [0 1 1 0]      0.0010          0.0769
 [1 1 0 0]      0.0000          0.0138
 [0 0 1 1]      0.0174          0.0067
 [1 0 0 1]      0.0164          0.0009
 [0 1 0 0]      3.9981          0.0003
 [1 0 0 0]      4.0018          0.0000
 [0 0 0 1]      4.0145          0.0000
 [1 1 0 1]      4.0150          0.0000
 [0 1 1 1]      4.0159          0.0000
 [0 0 0 0]      16.0000         0.0000
 [1 1 1 0]      4.0030          0.0000
 [0 0 1 0]      4.0029          0.0000
 [0 1 0 1]      0.0130          0.0000
 [1 0 1 1]      4.0193          0.0000
 [1 1 1 1]      16.0179         0.0000
```

```python
backend = Aer.get_backend('statevector_simulator')
seed = 50

cobyla = COBYLA()
cobyla.set_options(maxiter=250)
qaoa = QAOA(qubitOp, cobyla, 3)

qaoa.random_seed = seed

quantum_instance = QuantumInstance(backend=backend, seed_simulator=seed, seed_transpiler=seed)

result = qaoa.run(quantum_instance)

print_result(result)
```

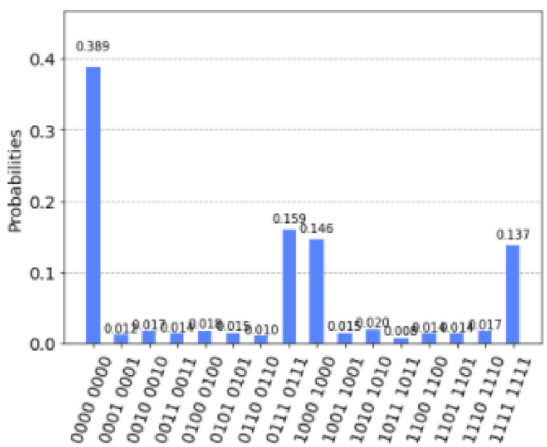
```
Optimal: selection [1. 1. 0. 0.], value 0.0000

----------------- Full result ---------------------
selection       value           probability
---------------------------------------------------
 [1 1 0 0]      0.0000          0.1668
 [0 1 1 0]      0.0010          0.1668
 [1 0 1 0]      0.0048          0.1667
 [0 1 0 1]      0.0130          0.1666
 [1 0 0 1]      0.0164          0.1666
 [0 0 1 1]      0.0174          0.1666
 [0 1 0 0]      3.9981          0.0000
 [1 0 0 0]      4.0018          0.0000
 [0 0 1 0]      4.0029          0.0000
 [0 0 0 1]      4.0145          0.0000
 [1 1 1 0]      4.0030          0.0000
 [1 1 1 1]      16.0179         0.0000
 [1 1 0 1]      4.0150          0.0000
 [0 1 1 1]      4.0159          0.0000
 [1 0 1 1]      4.0193          0.0000
 [0 0 0 0]      16.0000         0.0000
```

Grover with singular iteration

```
n = num_assets
grover_circuit = QuantumCircuit(n)
grover_circuit = initialize_s(grover_circuit, [0,1,2,3])
grover_circuit.append(oracle_gate, [0,1,2,3])
grover_circuit.append(diffuser(n), [0,1,2,3])
grover_circuit.measure_all()
grover_circuit.draw('mpl')
```

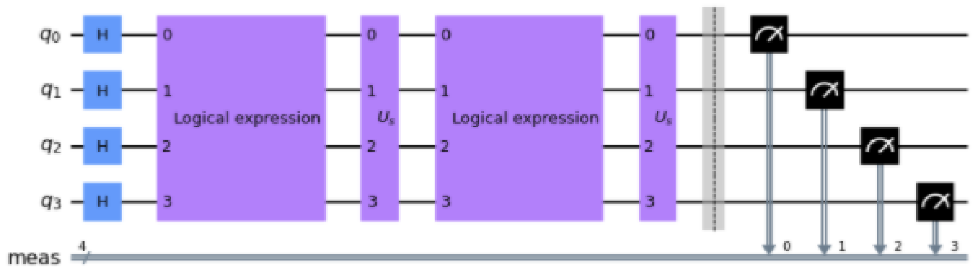



```
grover_circuit.measure_all()

qasm_simulator = Aer.get_backend('qasm_simulator')
shots = 1024
results = execute(grover_circuit, backend=qasm_simulator, shots=shots).result()
answer = results.get_counts()
plot_histogram(answer)
```

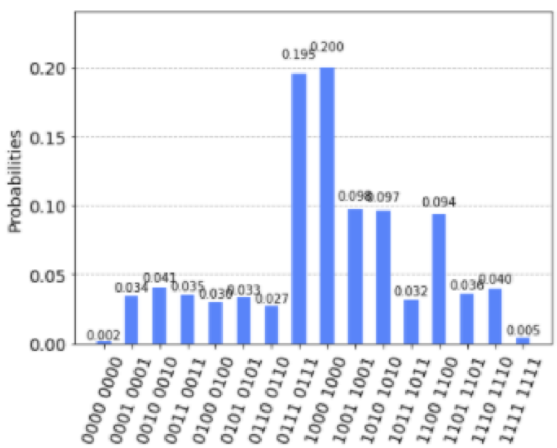


Grover with double iteration

```
n = num_assets
grover_circuit = QuantumCircuit(n)
grover_circuit = initialize_s(grover_circuit, [0,1,2,3])
grover_circuit.append(oracle_gate, [0,1,2,3])
grover_circuit.append(diffuser(n), [0,1,2,3])
grover_circuit.append(oracle_gate, [0,1,2,3])
grover_circuit.append(diffuser(n), [0,1,2,3])
grover_circuit.measure_all()
grover_circuit.draw('mpl')
```

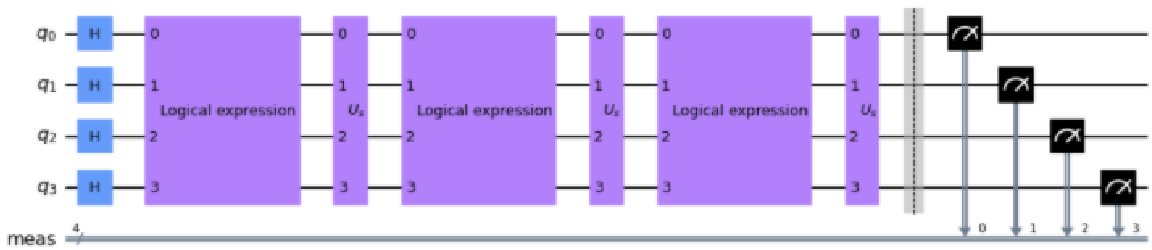



```
grover_circuit.measure_all()

qasm_simulator = Aer.get_backend('qasm_simulator')
shots = 1024
results = execute(grover_circuit, backend=qasm_simulator, shots=shots).result()
answer = results.get_counts()
plot_histogram(answer)
```

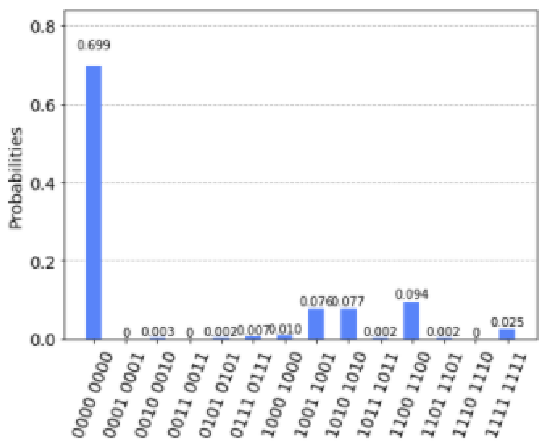


Grover with triple iteration

```
n = num_assets
grover_circuit = QuantumCircuit(n)
grover_circuit = initialize_s(grover_circuit, [0,1,2,3])
grover_circuit.append(oracle_gate, [0,1,2,3])
grover_circuit.append(diffuser(n), [0,1,2,3])
grover_circuit.append(oracle_gate, [0,1,2,3])
grover_circuit.append(diffuser(n), [0,1,2,3])
grover_circuit.append(oracle_gate, [0,1,2,3])
grover_circuit.append(diffuser(n), [0,1,2,3])
grover_circuit.measure_all()
grover_circuit.draw('mpl')
```




```
grover_circuit.measure_all()

qasm_simulator = Aer.get_backend('qasm_simulator')
shots = 1024
results = execute(grover_circuit, backend=qasm_simulator, shots=shots).result()
answer = results.get_counts()
plot_histogram(answer)
```




```
%qiskit_version_table
```

Version Information

Qiskit Software	Version
Qiskit	0.23.1
Terra	0.16.1
Aer	0.7.1
Ignis	0.5.1
Aqua	0.8.1
IBM Q Provider	0.11.1
System information	
Python	3.8.3 (default, Jul 2 2020, 11:26:31) [Clang 10.0.0]
OS	Darwin
CPUs	4
Memory (Gb)	16.0
	Sat Dec 05 23:39:34 2020 EST