CSCI 4140 Quantum Optimization

Mark Green

Faculty of Science

Ontario Tech

Warning!!!

- We are using the Qiskit optimization framework, which is relatively new
- This framework is being updated in incompatible ways on a regular basis
- To follow along with this lecture you will need to update Qiskit

```
pip install --upgrade qiskit[visualization]
```

Introduction

- We’ve seen that optimization is an important application
- There are many classical optimization algorithms, and many ways to formulate optimization problems
- Most of these algorithms are exponential, so this is a good area for quantum algorithms
- Several quantum algorithms have been produced, some can run on existing quantum computers
- Have seen that D-Wave specializes in optimization

Outline

- Optimization problem formulation
- Qiskit optimization framework
- Formulating an optimization problem for Qiskit
- Outline of the key algorithms
- Implementation in Qiskit

Optimization Problems

- Don’t always need the best solution, a good approximation is often good enough
- If the result is better than what we currently have, then it is an improvement
- One of the few applications with this property, useful for some algorithms
- Optimization problem: want the maximum or minimum value of some function given a set of constraints
- This is quite general, need to cut it down a bit

Optimization Problems

- The particular kind of optimization problem that we will consider is called a constrained quadratic optimization problem:

$$\min_{x \in X} x^T A x + b^T x + c$$

subject to

$$x^T A_i x + b_i^T x + c_i \leq 0, \quad i = 1, \ldots, r,$$

where

$$X = \mathbb{R}^n \times \mathbb{Z}^m \times \{0, 1\}^k$$

$$A \in \mathbb{R}^{(n+m+k) \times (n+m+k)}$$

$$b \in \mathbb{R}^{(n+m+k)}$$

$$c \in \mathbb{R}$$

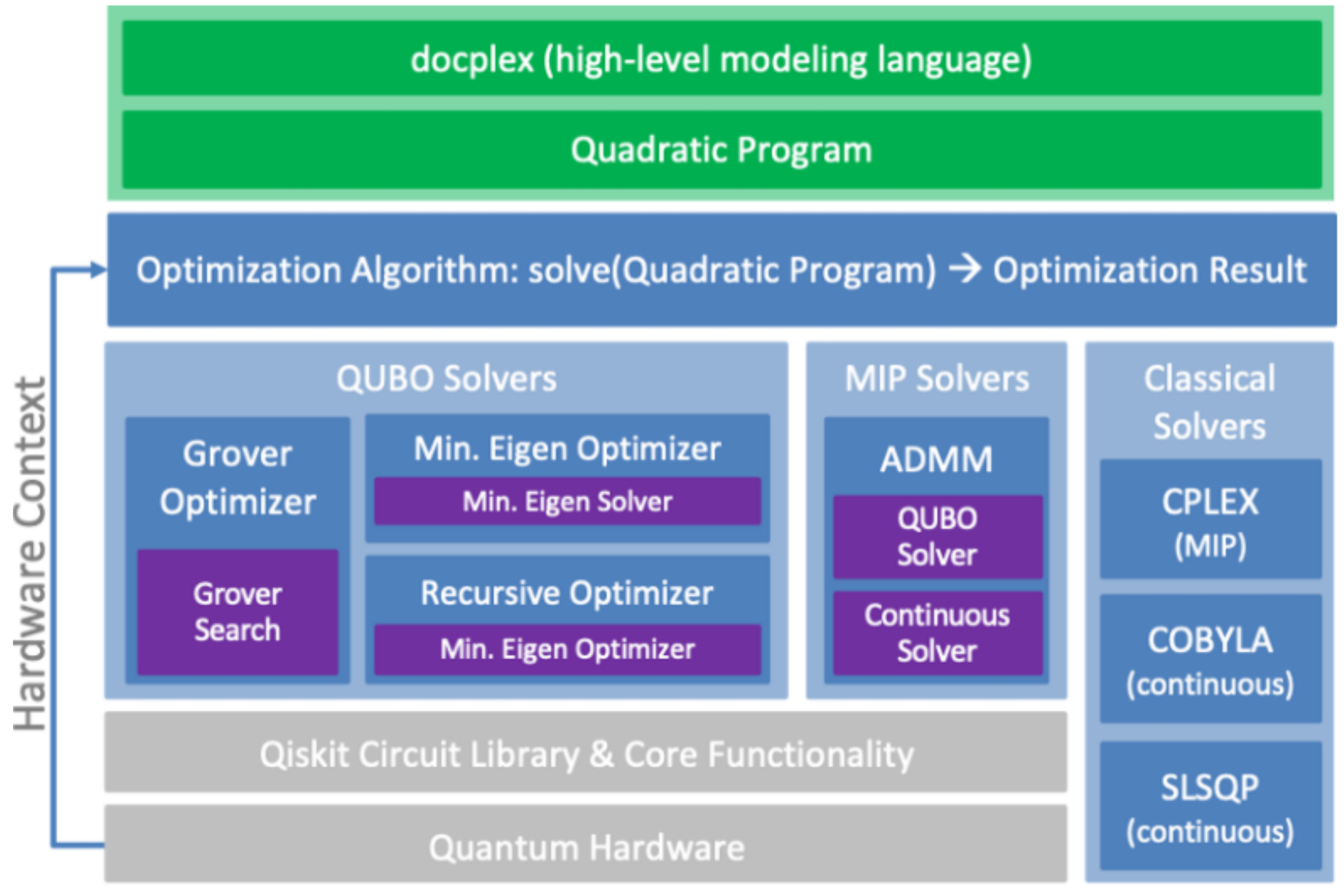
Optimization Problems

- This covers a wide range of interesting problems and is at the limit of what we can currently do with quantum computers
- We can have real, integer and binary variables
- We will start with binary variables, and work our way up to integers, we will not touch real values
- We have framed this as a minimization, we can do maximization in the same way, it’s just a sign reversal
- The constraints come from the problem we are trying to solve

Qiskit Optimization Framework

- Take more of a programming approach than a theory approach
- IBM provides an extensive framework for working with quadratic optimization problems, see next slide
- Starts with a high level description of the optimization problem in DOcplex
- This is then converted into a quadratic program
- This program can then be used with a wide range of quantum and classical optimization algorithms

Qiskit Optimization Framework

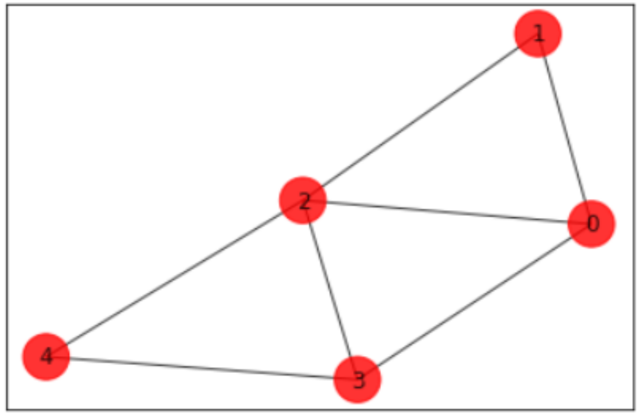


Problem Formulation

- Start by examining a particular problem and how it can be solved using Qiskit
- The problem is max-cut, divide a graph into two components such that there is a maximum number of edges between the components
- Each edge could have a weight, but we won’t be concerned with that
- This is a typical optimization problem on graphs
- We will start with a graph that has 5 nodes
- The following slide shows how we create the graph

Problem Formulation

```
# Create graph
G = nx.Graph()

# Add 5 nodes
n = 5
G.add_nodes_from(range(n))

# Add edges: tuple is (i,j,weight) where (i,j) is the edge
edges = [(0, 1, 1.0), (0, 2, 1.0), (0, 3, 1.0), (1, 2, 1.0), (2, 3, 1.0), (2, 4, 1.0), (3, 4, 1.0)]
G.add_weighted_edges_from(edges)

# Plot graph
plot_result(G, [0]*n)
```



Problem Formulation

- The variable edges is the list of edges in the graph, consists of the vertices the edge connects, plus a weight
- We now need to construct our optimization problem, which can be stated in the following way, called the objective:

$$\max_{x \in \{0,1\}^n} \sum_{(j,k) \in E} w_{jk}(x_j + x_k - 2x_j x_k)$$

- How does this work?
- We assign x_i the value 0 if its in the first subgraph, and 1 if its in the other subgraph

Problem Formulation

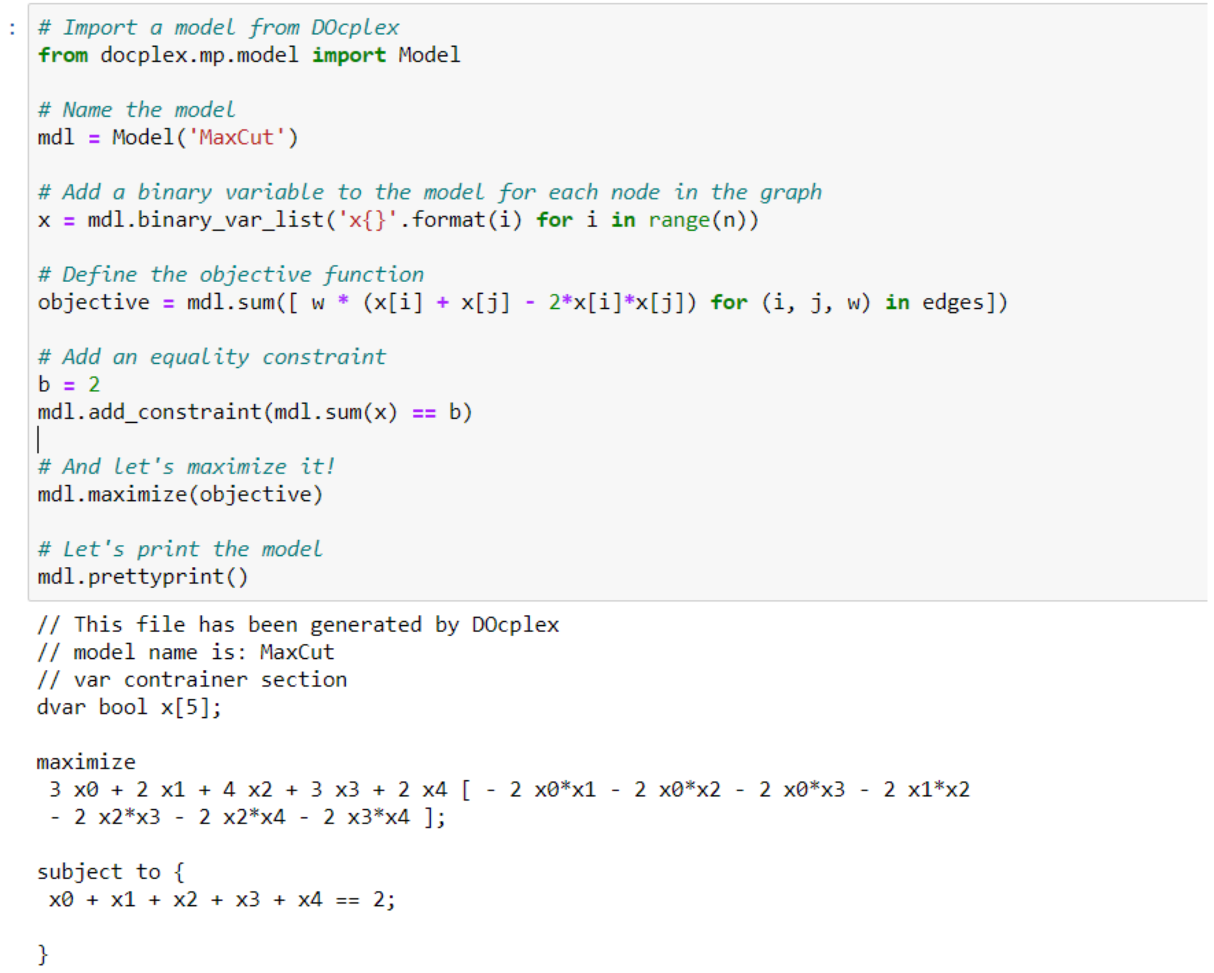
- If x_j and x_k are in the same subgraph the expression in the sum evaluates to zero, if they are in opposite sets the value is 1
- There are other ways of formulating this, but this approach works
- Now we add a constraint to the problem, the number of vertices in one of the subgraphs must be 2
- In general we can write this constraint as, and set b=2

$$\sum_{i=0}^{n-1} x_j = b$$

Problem Formulation

- Now we need to convert the problem to DOcplex
- We have 5 binary variables, one for each vertex in the graph
- We then add the objective function to the model
- Finally we add the constraint to the model
- Then we state that we want to maximize the objective, the other alternative is to minimize the objective
- The code for doing this is one the next slide

Problem Formulation



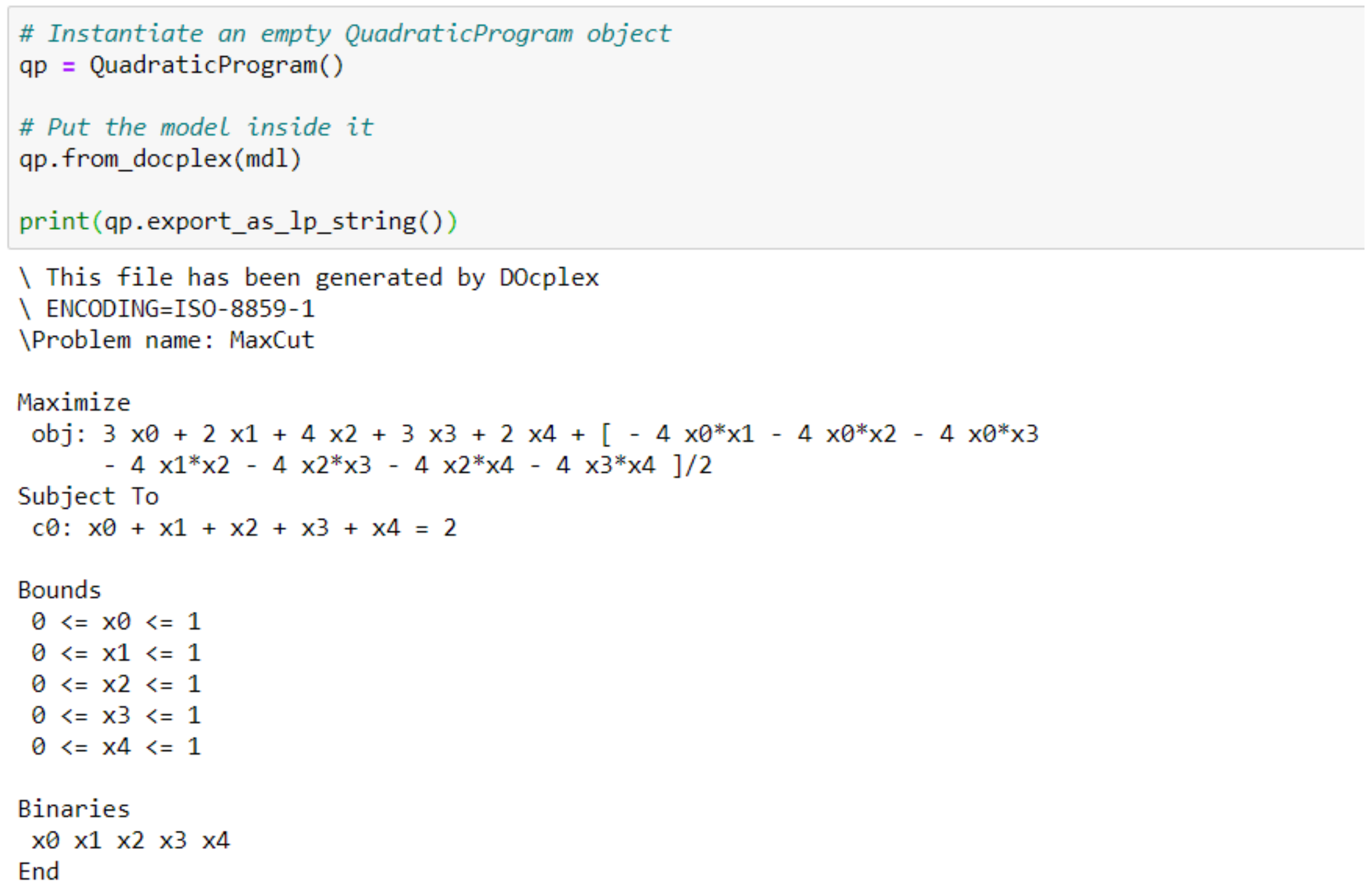
```python
# Import a model from DOcplex
from docplex.mp.model import Model

# Name the model
mdl = Model('MaxCut')

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

# Define the objective function
objective = mdl.sum([ w * (x[i] + x[j] - 2*x[i]*x[j]) for (i, j, w) in edges])

# Add an equality constraint
b = 2
mdl.add_constraint(mdl.sum(x) == b)

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

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

```
// This file has been generated by DOcplex
// model name is: MaxCut
// var contrainer section
dvar bool x[5];

maximize
 3 x0 + 2 x1 + 4 x2 + 3 x3 + 2 x4 [ - 2 x0*x1 - 2 x0*x2 - 2 x0*x3 - 2 x1*x2
 - 2 x2*x3 - 2 x2*x4 - 2 x3*x4 ];

subject to {
 x0 + x1 + x2 + x3 + x4 == 2;

}
```

Quadratic Problem

- We are solving quadratic problems, so we need to convert the model to a quadratic problem
- Qiskit provides a quadratic problem object, and it has a function that converts a DOcplex description into a quadratic problem
- This is shown on the next slide
- Notice that the quadratic part of the objective has been divided by 2, this is because each of the quadratic terms have been multiplied by 2
- It identifies all our variables as binaries and gives a range for them

Quadratic Problem

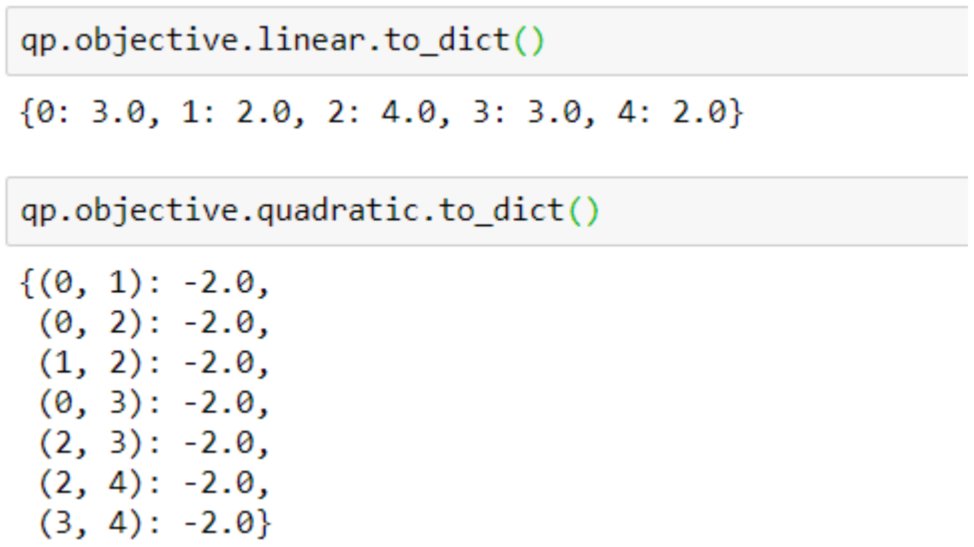


```python
# Instantiate an empty QuadraticProgram object
qp = QuadraticProgram()

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

print(qp.export_as_lp_string())
```

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

Maximize
 obj: 3 x0 + 2 x1 + 4 x2 + 3 x3 + 2 x4 + [ - 4 x0*x1 - 4 x0*x2 - 4 x0*x3
      - 4 x1*x2 - 4 x2*x3 - 4 x2*x4 - 4 x3*x4 ]/2
Subject To
 c0: x0 + x1 + x2 + x3 + x4 = 2

Bounds
 0 <= x0 <= 1
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1

Binaries
 x0 x1 x2 x3 x4
End
```

Quadratic Problem

- We can access different parts of the quadratic problem, such as the linear and quadratic coefficients



```
qp.objective.linear.to_dict()
```

```
{0: 3.0, 1: 2.0, 2: 4.0, 3: 3.0, 4: 2.0}
```

```
qp.objective.quadratic.to_dict()
```

```
{(0, 1): -2.0,
 (0, 2): -2.0,
 (1, 2): -2.0,
 (0, 3): -2.0,
 (2, 3): -2.0,
 (2, 4): -2.0,
 (3, 4): -2.0}
```

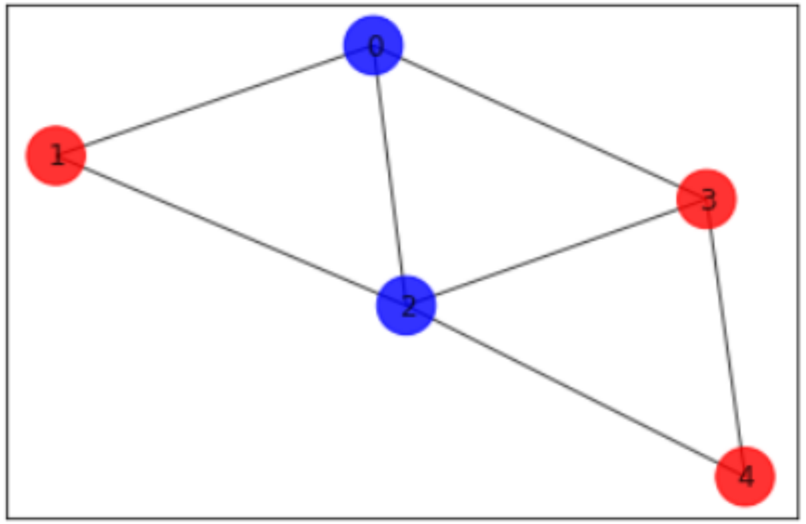
Classical Solution

- This is a toy problem so it’s easy to solve classically
- This gives us a correct solution that we can then compare to the results from the quantum algorithms
- We use a classical Eigen solver to solve this problem, we will see how eigenvalues fit into the picture later
- This produces one of the four solutions to the problem as shown on the next slide
- The two subgraphs are shown in different colours, and the optimal value is 5, you can count the edges

Classical Solution

```
solver = MinimumEigenOptimizer(NumPyMinimumEigensolver())
result = solver.solve(qp)
print(result)
plot_result(G, result.x)
```

```
x=[1.0,0.0,1.0,0.0,0.0], fval=5.0
```



Convert to QUBO

- The IBM framework has a number of solvers that operate on QUBO problems
- We've seen these before with D-Wave
- There is a problem here, recall what a QUBO looks like

$$\min_{x \in \{0,1\}^k} x^T A x + c$$

- All the variables are binary, not a problem for us
- But, there are no constraints, this is a problem

Convert to QUBO

- We have an equality constraint in our problem
- We can convert equality constraints to QUBO in the following way
- We have the following constraint

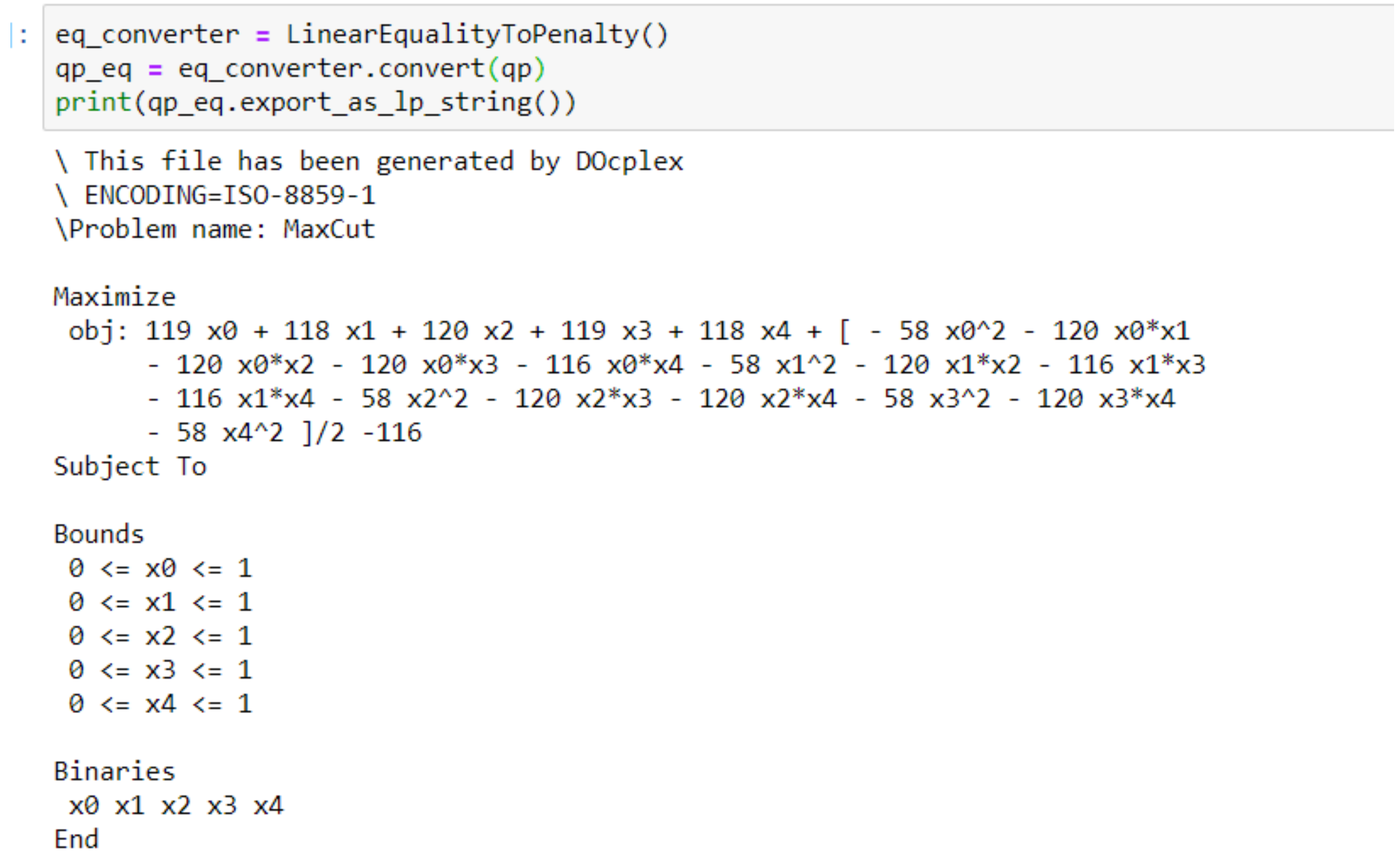
$$\sum_i a_i x_i = b$$

- It can be converted into, where M is a suitably large number

$$M(b - \sum_i a_i x_i)^2$$

Convert to QUBO

- This can be done in the following way

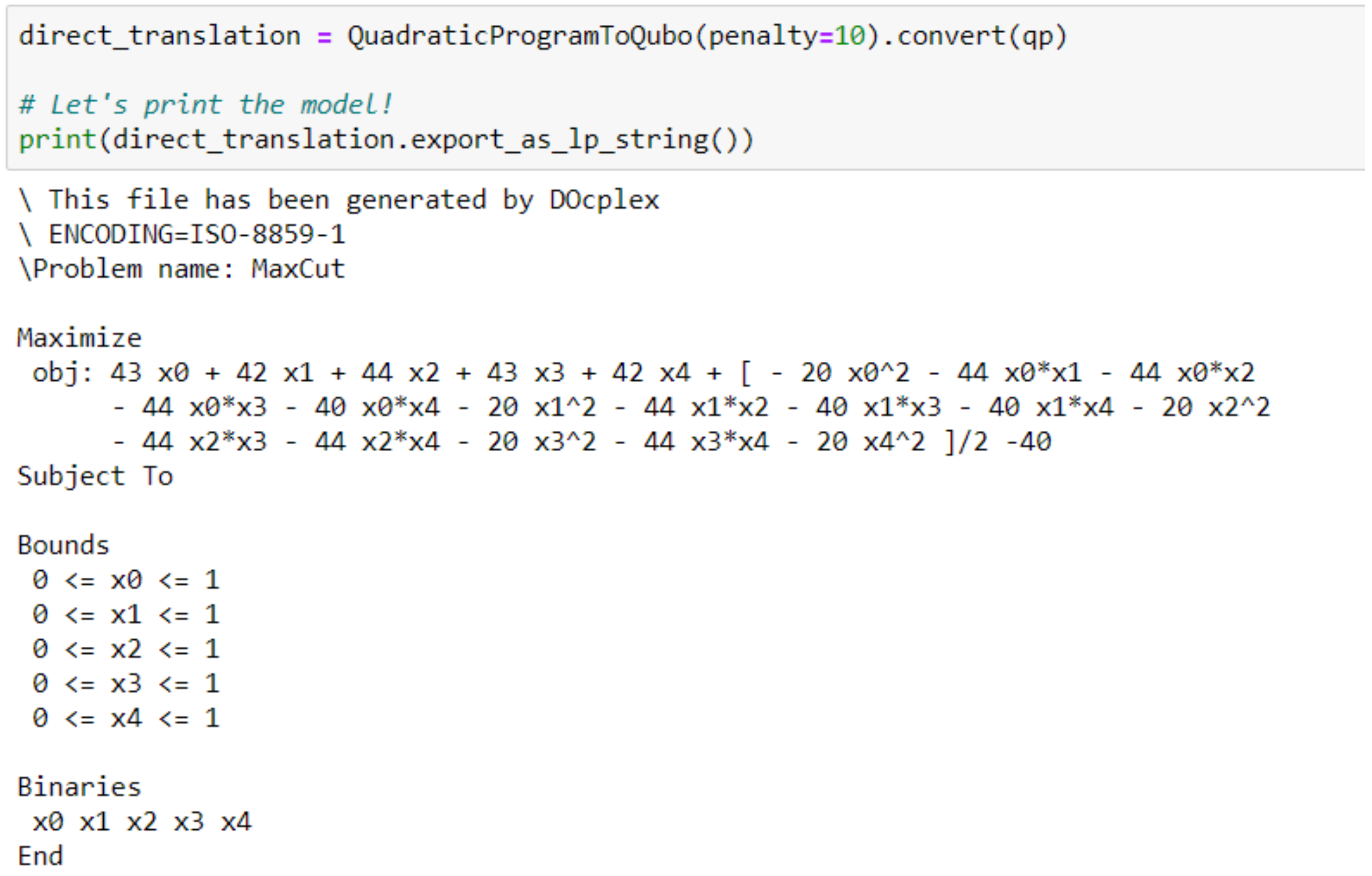


```
eq_converter = LinearEqualityToPenalty()
qp_eq = eq_converter.convert(qp)
print(qp_eq.export_as_lp_string())
```

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

Maximize
 obj: 119 x0 + 118 x1 + 120 x2 + 119 x3 + 118 x4 + [ - 58 x0^2 - 120 x0*x1
      - 120 x0*x2 - 120 x0*x3 - 116 x0*x4 - 58 x1^2 - 120 x1*x2 - 116 x1*x3
      - 116 x1*x4 - 58 x2^2 - 120 x2*x3 - 120 x2*x4 - 58 x3^2 - 120 x3*x4
      - 58 x4^2 ]/2 -116
Subject To

Bounds
 0 <= x0 <= 1
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1

Binaries
 x0 x1 x2 x3 x4
End
```

Convert to QUBO

- We can also do all the conversions at once



```python
direct_translation = QuadraticProgramToQubo(penalty=10).convert(qp)

# Let's print the model!
print(direct_translation.export_as_lp_string())
```

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

Maximize
 obj: 43 x0 + 42 x1 + 44 x2 + 43 x3 + 42 x4 + [ - 20 x0^2 - 44 x0*x1 - 44 x0*x2
      - 44 x0*x3 - 40 x0*x4 - 20 x1^2 - 44 x1*x2 - 40 x1*x3 - 40 x1*x4 - 20 x2^2
      - 44 x2*x3 - 44 x2*x4 - 20 x3^2 - 44 x3*x4 - 20 x4^2 ]/2 -40
Subject To

Bounds
 0 <= x0 <= 1
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1

Binaries
 x0 x1 x2 x3 x4
End
```

Convert to Ising Model

- A number of the optimization algorithms make use of Hamiltonians
- The easiest way to get this Hamiltonians is to convert our QUBO into an Ising model
- The Ising model is also binary, but it uses -1 and +1 instead of 0 and 1
- We will use z_i as our Ising variables, their relation to the x_i is given by

$$x_i = (1 - z_i)/2$$

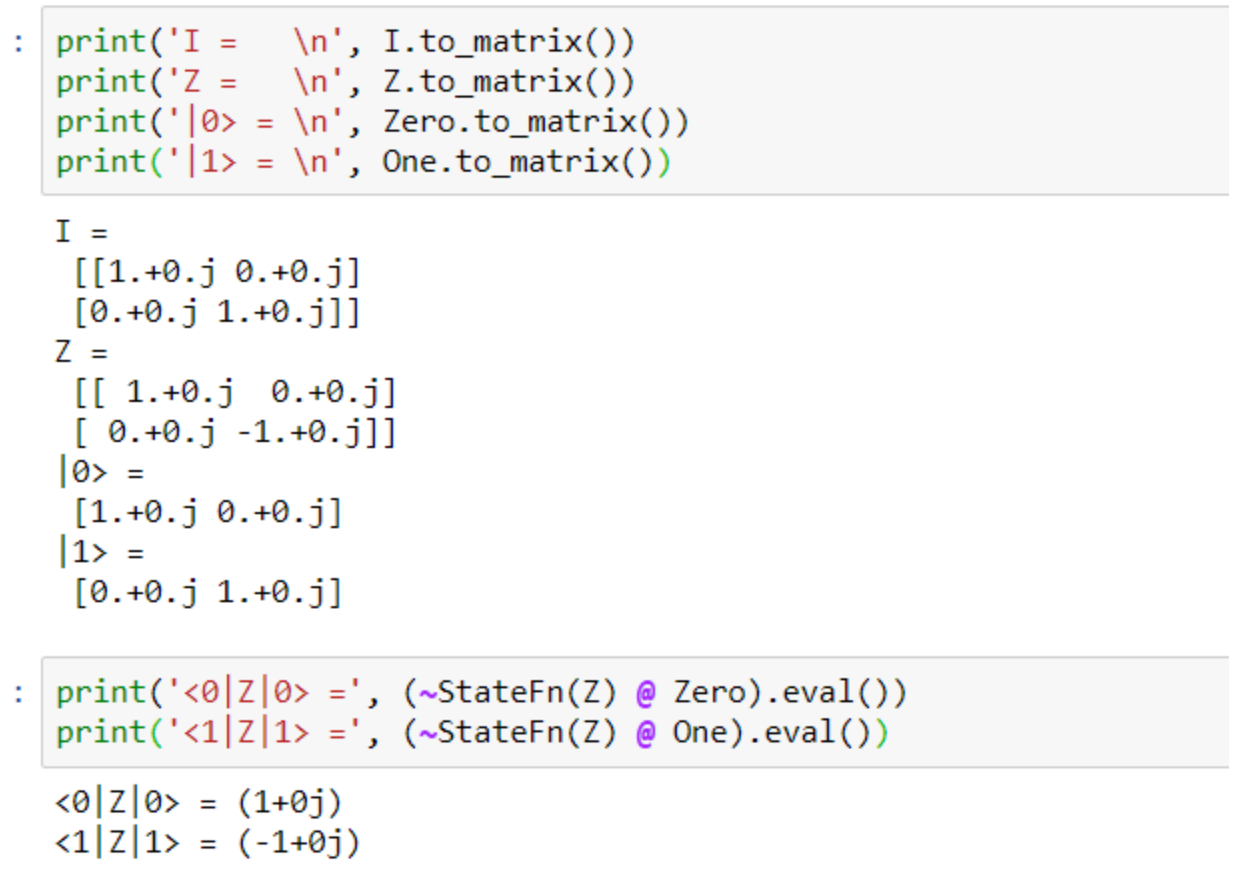
$$z_i z_j = \sigma_z^i \otimes \sigma_z^j$$

$z_i = \sigma_z^i$ where σ_z^i is the Pauli Z operator on the i^{th} qubit

Ising Model

- Before going further it’s worth taking a bit of time to explore how these operators work and how we can construct simple Hamiltonians using them
- Recall that we can represent our gates by matrices and our qubits by vectors
- The next slide shows these matrices
- It also show how our |0> and |1> vectors are converted to Ising values using expectations

Ising Model



```
: print('I =   \n', I.to_matrix())
  print('Z =   \n', Z.to_matrix())
  print('|0> = \n', Zero.to_matrix())
  print('|1> = \n', One.to_matrix())
```

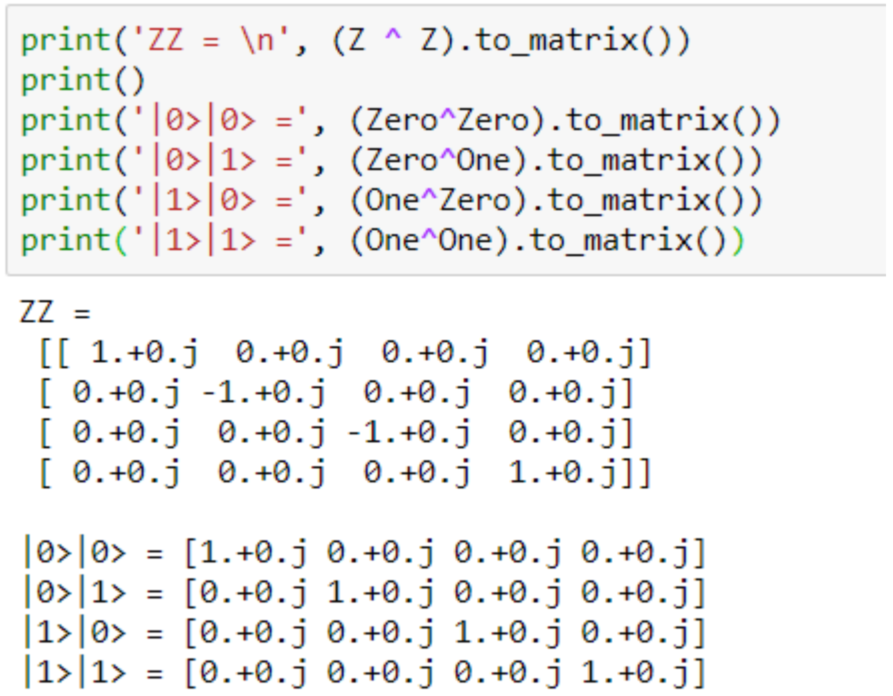
```
I =
 [[1.+0.j 0.+0.j]
 [0.+0.j 1.+0.j]]
Z =
 [[ 1.+0.j  0.+0.j]
 [ 0.+0.j -1.+0.j]]
|0> =
 [1.+0.j 0.+0.j]
|1> =
 [0.+0.j 1.+0.j]
```

```
: print('<0|Z|0> =', (~StateFn(Z) @ Zero).eval())
  print('<1|Z|1> =', (~StateFn(Z) @ One).eval())
```

```
<0|Z|0> = (1+0j)
<1|Z|1> = (-1+0j)
```

Ising Model

- This explains what happens with the Z operator in the Ising model, now let's turn to the ZZ operator

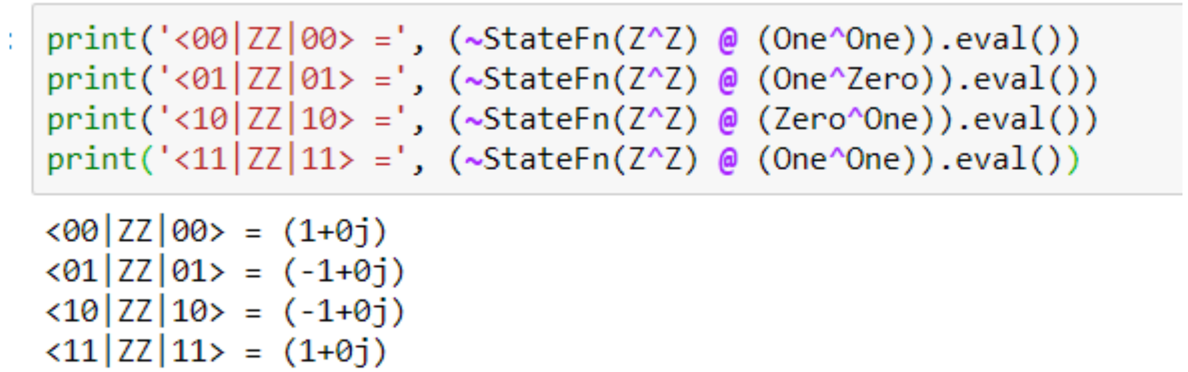


```python
print('ZZ = \n', (Z ^ Z).to_matrix())
print()
print('|0>|0> =', (Zero^Zero).to_matrix())
print('|0>|1> =', (Zero^One).to_matrix())
print('|1>|0> =', (One^Zero).to_matrix())
print('|1>|1> =', (One^One).to_matrix())
```

```
ZZ =
 [[ 1.+0.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j -1.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j -1.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j  1.+0.j]]

|0>|0> = [1.+0.j 0.+0.j 0.+0.j 0.+0.j]
|0>|1> = [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
|1>|0> = [0.+0.j 0.+0.j 1.+0.j 0.+0.j]
|1>|1> = [0.+0.j 0.+0.j 0.+0.j 1.+0.j]
```

Ising Model

- Now lets turn to the expectations



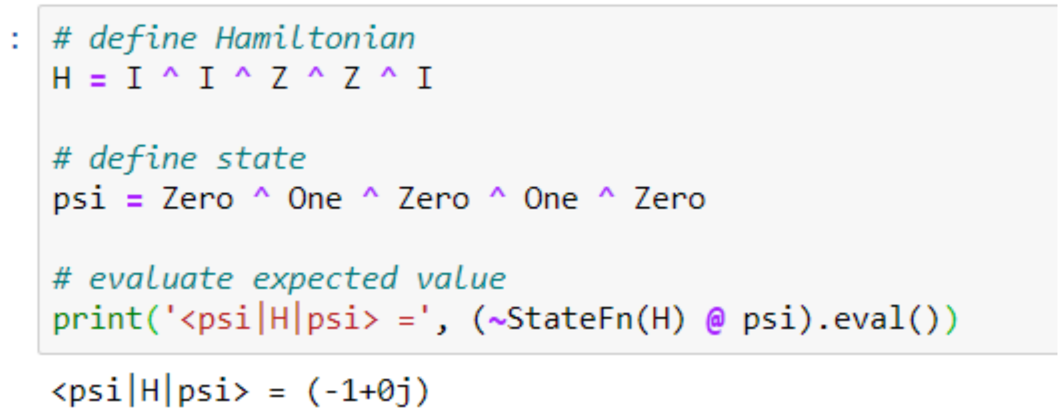
```
print('<00|ZZ|00> =', (~StateFn(Z^Z) @ (One^One)).eval())
print('<01|ZZ|01> =', (~StateFn(Z^Z) @ (One^Zero)).eval())
print('<10|ZZ|10> =', (~StateFn(Z^Z) @ (Zero^One)).eval())
print('<11|ZZ|11> =', (~StateFn(Z^Z) @ (One^One)).eval())
```

```
<00|ZZ|00> = (1+0j)
<01|ZZ|01> = (-1+0j)
<10|ZZ|10> = (-1+0j)
<11|ZZ|11> = (1+0j)
```

- Note that if both Ising bits are equal the result is +1, if they are not equal the result is -1
- This is the basis of the Ising model

Ising Model

- Now let’s build a toy Hamiltonian of 5 qubits, and a 5 qubit vector to test it on



```
# define Hamiltonian
H = I ^ I ^ Z ^ Z ^ I

# define state
psi = Zero ^ One ^ Zero ^ One ^ Zero

# evaluate expected value
print('<psi|H|psi> =', (~StateFn(H) @ psi).eval())
```

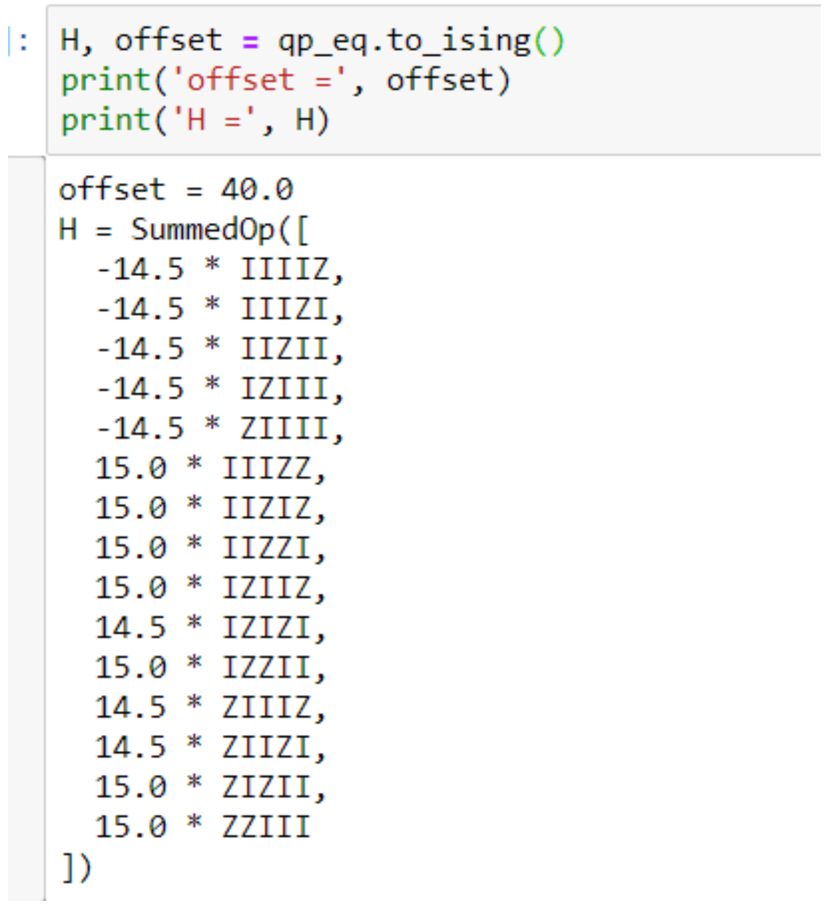
```
<psi|H|psi> = (-1+0j)
```

- So what’s going on here?
- In the Hamiltonian, Z is applied to the third and fourth qubits, but in psi these qubit have different values, resulting in a -1 value

Ising Model

- The to_ising() function can be used to convert a QUBO to a Ising model
- The function returns the Hamiltonian H and an offset value
- Due to the way the Hamiltonian is constructed we need to offset the value that we will get from our optimization algorithms

Ising Model



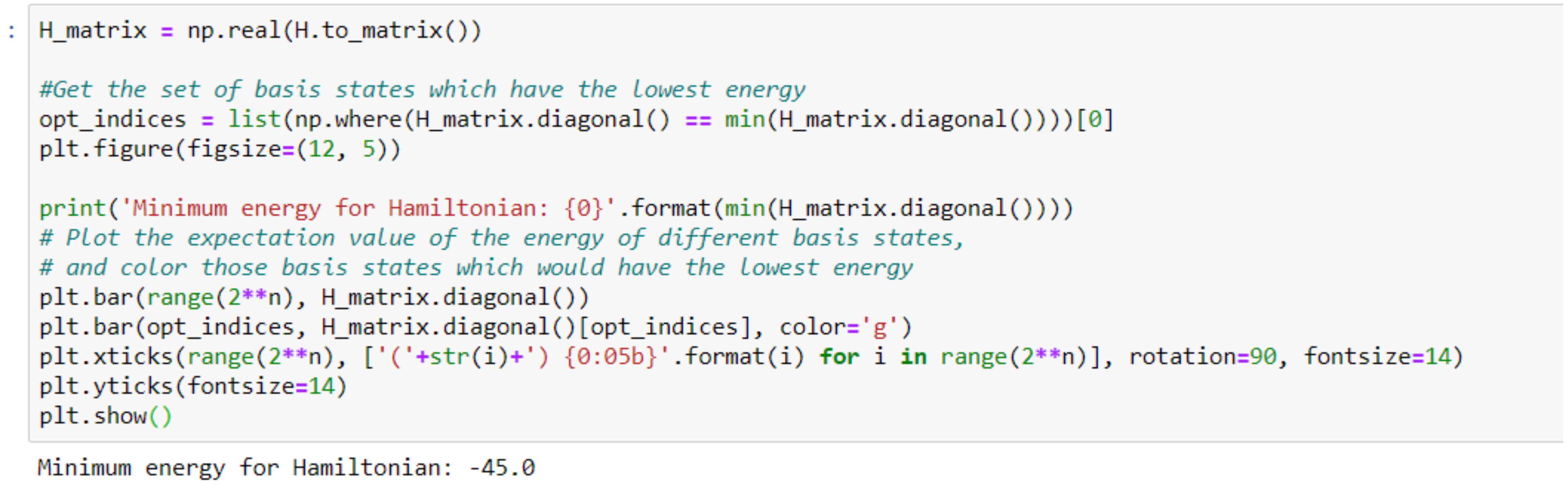
```
H, offset = qp_eq.to_ising()
print('offset =', offset)
print('H =', H)
```

```
offset = 40.0
H = SummedOp([
  -14.5 * IIIIZ,
  -14.5 * IIIZI,
  -14.5 * IIZII,
  -14.5 * IZIII,
  -14.5 * ZIIII,
  15.0 * IIIZZ,
  15.0 * IIZIZ,
  15.0 * IIZZI,
  15.0 * IZIIZ,
  14.5 * IZIZI,
  15.0 * IZZII,
  14.5 * ZIIIZ,
  14.5 * ZIIZI,
  15.0 * ZIZII,
  15.0 * ZZIII
])
```

Ising Model

- Before we examine some optimization algorithms we will briefly examine what the Ising model is telling us
- Start by converting the operators in the Hamiltonian to a real matrix so we can do some computations
- Next we find the states with the lowest energy
- We add the offset to this minimum, giving -5, we’ve converted a maximization to a minimization so the sign has been changed
- Finally we plot the energy levels for all the states, with the green bars indicating the states with the lowest energy

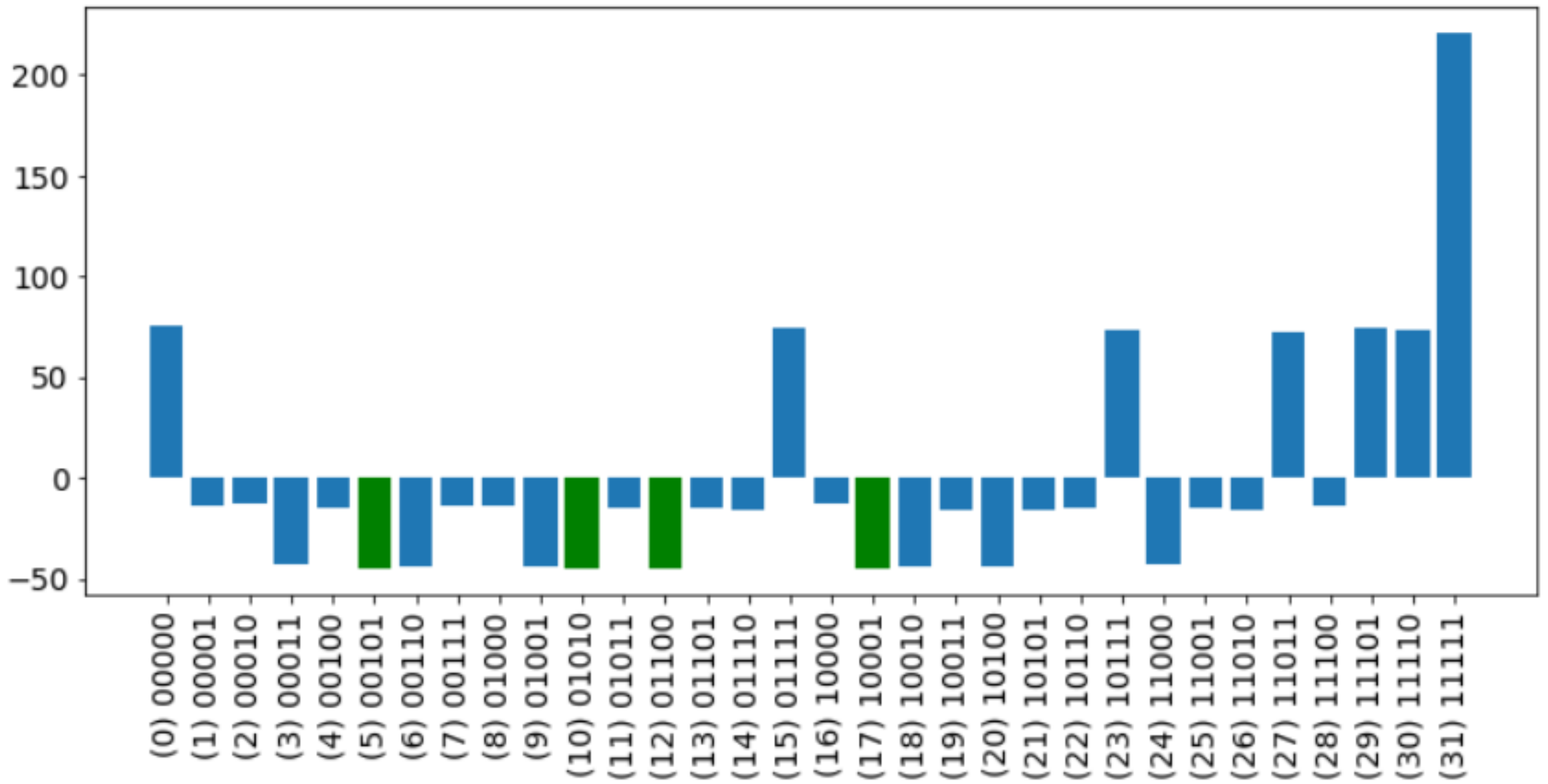
Ising Model



```
: H_matrix = np.real(H.to_matrix())

  #Get the set of basis states which have the lowest energy
  opt_indices = list(np.where(H_matrix.diagonal() == min(H_matrix.diagonal())))[0]
  plt.figure(figsize=(12, 5))

  print('Minimum energy for Hamiltonian: {0}'.format(min(H_matrix.diagonal())))
  # Plot the expectation value of the energy of different basis states,
  # and color those basis states which would have the lowest energy
  plt.bar(range(2**n), H_matrix.diagonal())
  plt.bar(opt_indices, H_matrix.diagonal()[opt_indices], color='g')
  plt.xticks(range(2**n), ['('+str(i)+') {0:05b}'.format(i) for i in range(2**n)], rotation=90, fontsize=14)
  plt.yticks(fontsize=14)
  plt.show()
```

```
Minimum energy for Hamiltonian: -45.0
```

Ising Model


200
150
100
50
0
−50
(0) 00000
(1) 00001
(2) 00010
(3) 00011
(4) 00100
(5) 00101
(6) 00110
(7) 00111
(8) 01000
(9) 01001
(10) 01010
(11) 01011
(12) 01100
(13) 01101
(14) 01110
(15) 01111
(16) 10000
(17) 10001
(18) 10010
(19) 10011
(20) 10100
(21) 10101
(22) 10110
(23) 10111
(24) 11000
(25) 11001
(26) 11010
(27) 11011
(28) 11100
(29) 11101
(30) 11110
(31) 11111

VQE Algorithm

- Convert the optimization problem into an eigenvalue/eigenvector problem
- Then use the variational quantum eigensolver to solve the problem
- Converts to problem into

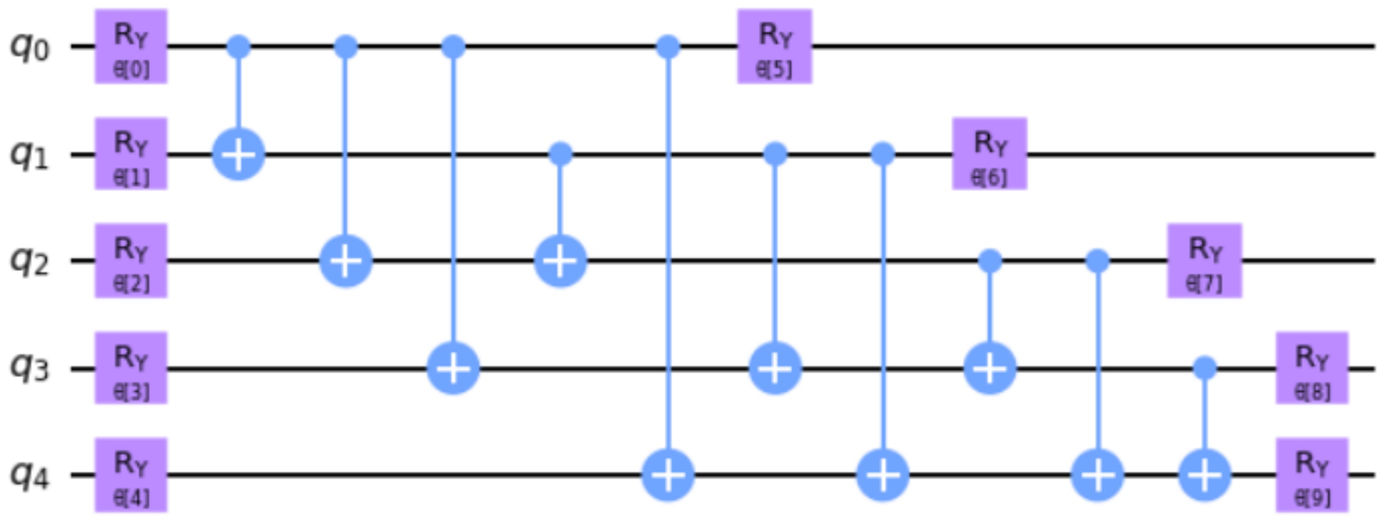
$$\min_{\theta} \langle \psi(\theta) | H | \psi(\theta) \rangle$$

- Combined quantum/classical algorithm, on the quantum side we compute the eigenvalues, the classical side computes θ values to produce the minimum

VQE Algorithm

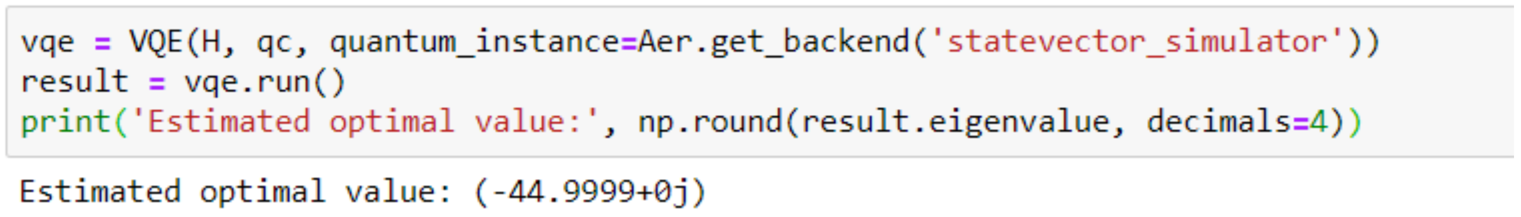
- The $\psi(\theta)$ is called the ansatz and θ is a vector of parameters, we can build a suitable ansatz in the following way

```
qc = RealAmplitudes(5, reps=1)
qc.draw(output='mpl')
```



VQE Algorithm

- We can take the ansatz, the Hamiltonian we've already computed and the backend and give it to the VQE solver in Qiskit

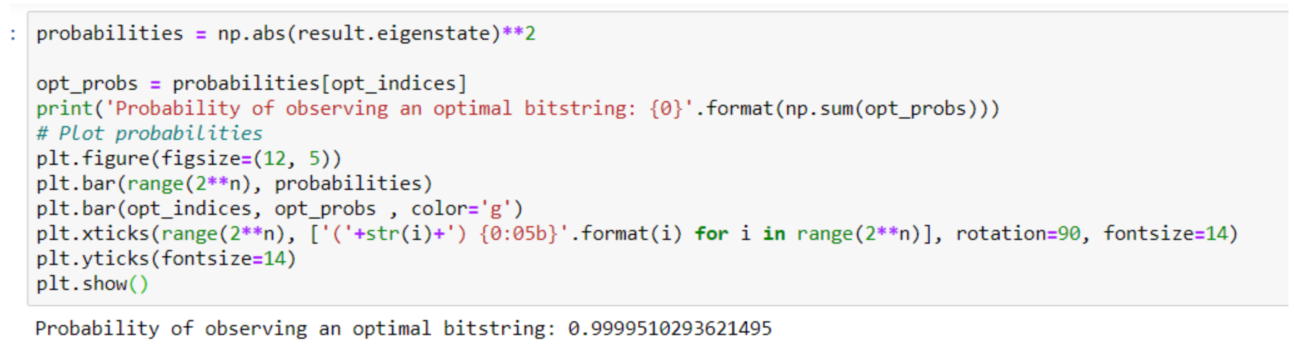


```
vqe = VQE(H, qc, quantum_instance=Aer.get_backend('statevector_simulator'))
result = vqe.run()
print('Estimated optimal value:', np.round(result.eigenvalue, decimals=4))
```

```
Estimated optimal value: (-44.9999+0j)
```

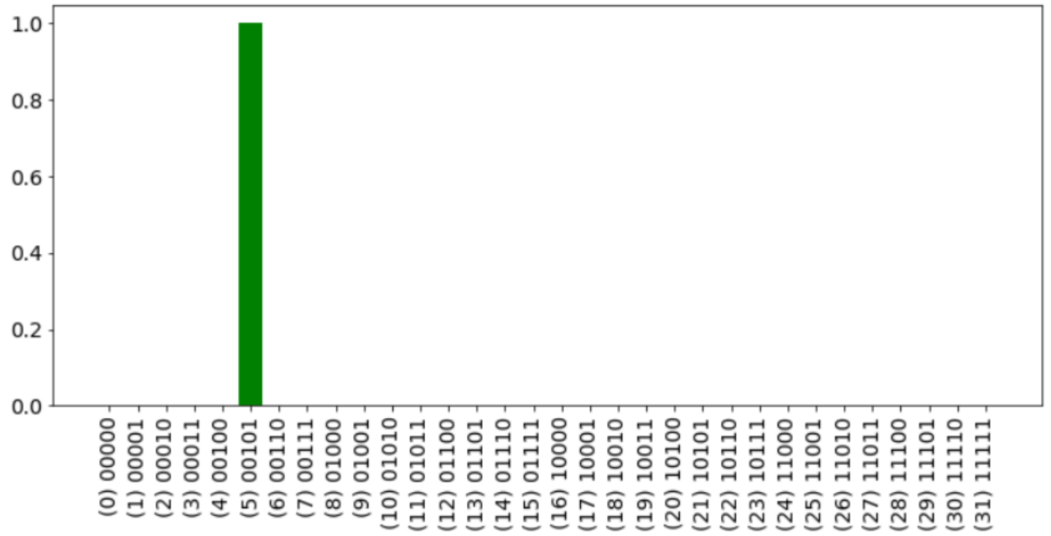
- The value that we get is very close to the exact value of -45 that we computed earlier
- The following slide shows one of the graph solutions, and we see that it has a very high probability

VQE Algorithm



```
probabilities = np.abs(result.eigenstate)**2

opt_probs = probabilities[opt_indices]
print('Probability of observing an optimal bitstring: {0}'.format(np.sum(opt_probs)))
# Plot probabilities
plt.figure(figsize=(12, 5))
plt.bar(range(2**n), probabilities)
plt.bar(opt_indices, opt_probs , color='g')
plt.xticks(range(2**n), ['('+str(i)+') {0:05b}'.format(i) for i in range(2**n)], rotation=90, fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

```
Probability of observing an optimal bitstring: 0.9999510293621495
```



Trotterized Annealing

- We’ve already seen quantum annealing with the D-Wave system
- Have two Hamiltonians, an initial Hamiltonian and a problem Hamilton
- The initial Hamiltonian is easy to construct, slowly evolve it towards the final Hamilton using an annealing schedule
- At this point we have a Hamiltonian for our problem, we just need to find the initial Hamiltonian

Trotterized Annealing

- Start with a simple 1 qubit example, here is our problem Hamiltonian

$$H_C = \sigma_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

- Our initial Hamiltonian will be σ_x
- H_c has a ground state of |1> and an optimal value of -1
- Our annealing process is given by

$$H_t = \frac{t}{T}\sigma_Z - (1 - \frac{t}{T})\sigma_X.$$

Trotterized Annealing

- The annealing process is approximated by

$$|\psi_{t+1}\rangle = e^{-iH_t\Delta t}|\psi_t\rangle$$

- Where Δt is a small time step
- The problem is that σ_x and σ_z don't commute making this difficult to compute
- We Trotterize to first apply $e^{-i\frac{t}{T}H_C\Delta t}$
- Followed by $e^{-i(1-\frac{t}{T})H_X\Delta t}$

Trotterized Annealing

- In terms of gates we have the following

$$e^{-i\frac{t}{T}H_C\Delta t} = e^{-i\sigma_Z\frac{\gamma_t}{2}} = R_Z(\gamma_t),$$

$$e^{-i(1-\frac{t}{T})H_X\Delta t} = e^{-i\sigma_X\frac{\beta_t}{2}} = R_X(\beta_t)$$

- Where

$$\gamma_t = 2t/T\Delta t$$

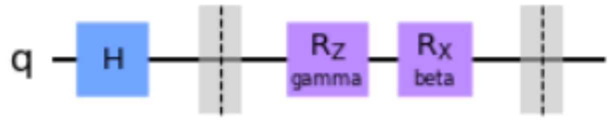
$$\beta_t = -2(1 - t/T)\Delta t.$$

- Note that γ_t and β_t define our annealing schedule

Trotterized Annealing

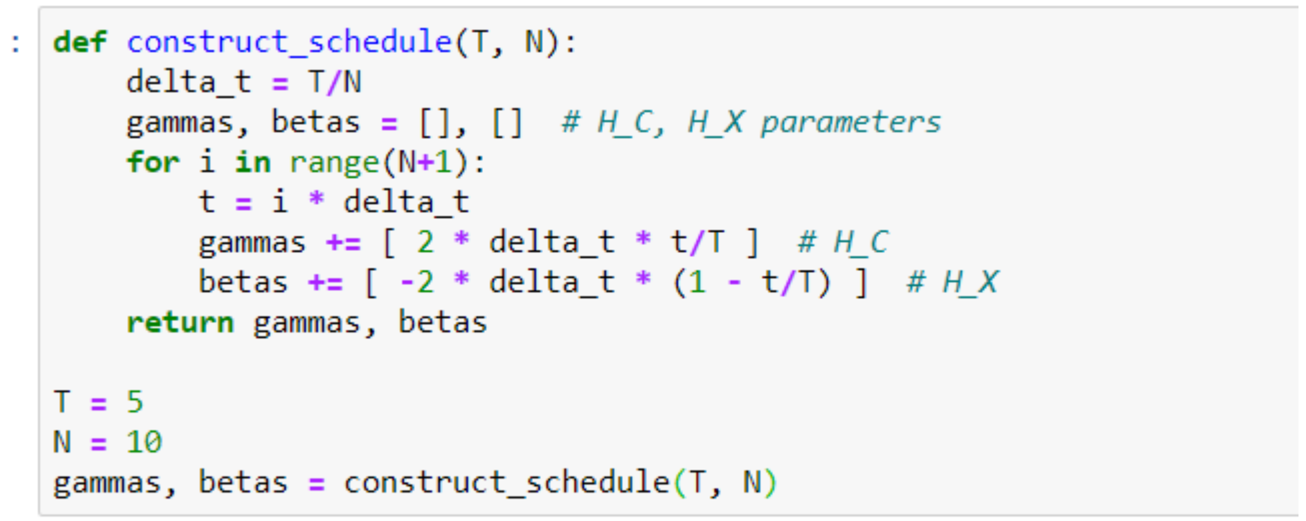
- Now that we’ve gone through the theory let’s examine how we would program this
- The circuit for an individual time step is

```python
from qiskit.circuit import Parameter
gamma, beta = Parameter('gamma'), Parameter('beta')

# This circuit would be 1 time step
qc = QuantumCircuit(1)
qc.h(0)
qc.barrier()
qc.rz(gamma, 0)
qc.rx(beta, 0)
qc.barrier()
qc.draw(output='mpl')
```



Trotterized Annealing

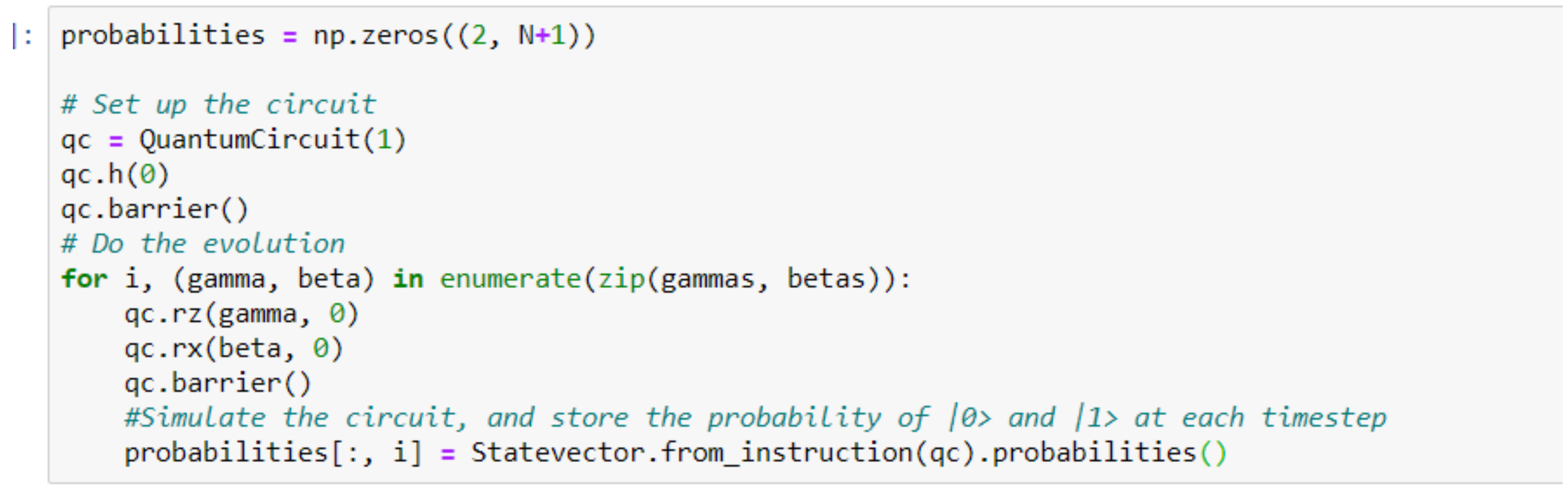
- We need to repeat this according to our schedule
- Start by computing the schedule



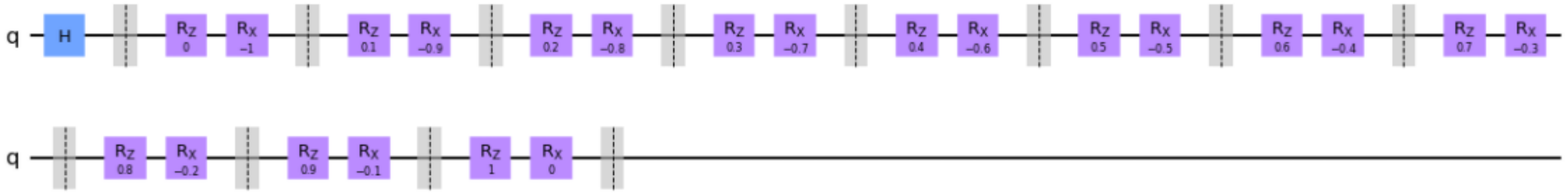
```
def construct_schedule(T, N):
    delta_t = T/N
    gammas, betas = [], []  # H_C, H_X parameters
    for i in range(N+1):
        t = i * delta_t
        gammas += [ 2 * delta_t * t/T ]  # H_C
        betas += [ -2 * delta_t * (1 - t/T) ]  # H_X
    return gammas, betas

T = 5
N = 10
gammas, betas = construct_schedule(T, N)
```

Trotterized Annealing

- Now let’s put together the complete algorithm



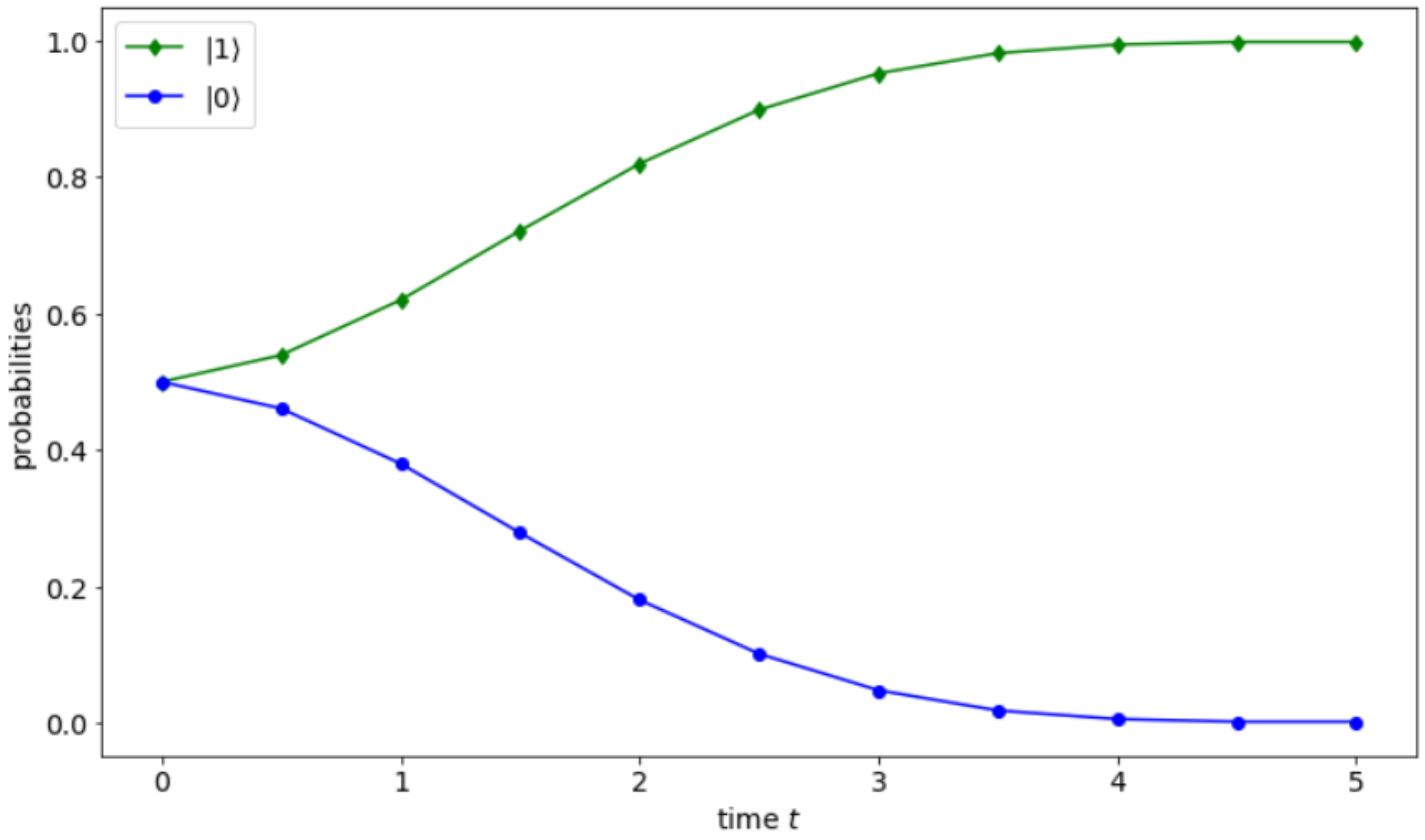
```
|: probabilities = np.zeros((2, N+1))

   # Set up the circuit
   qc = QuantumCircuit(1)
   qc.h(0)
   qc.barrier()
   # Do the evolution
   for i, (gamma, beta) in enumerate(zip(gammas, betas)):
       qc.rz(gamma, 0)
       qc.rx(beta, 0)
       qc.barrier()
       #Simulate the circuit, and store the probability of |0> and |1> at each timestep
       probabilities[:, i] = Statevector.from_instruction(qc).probabilities()
```



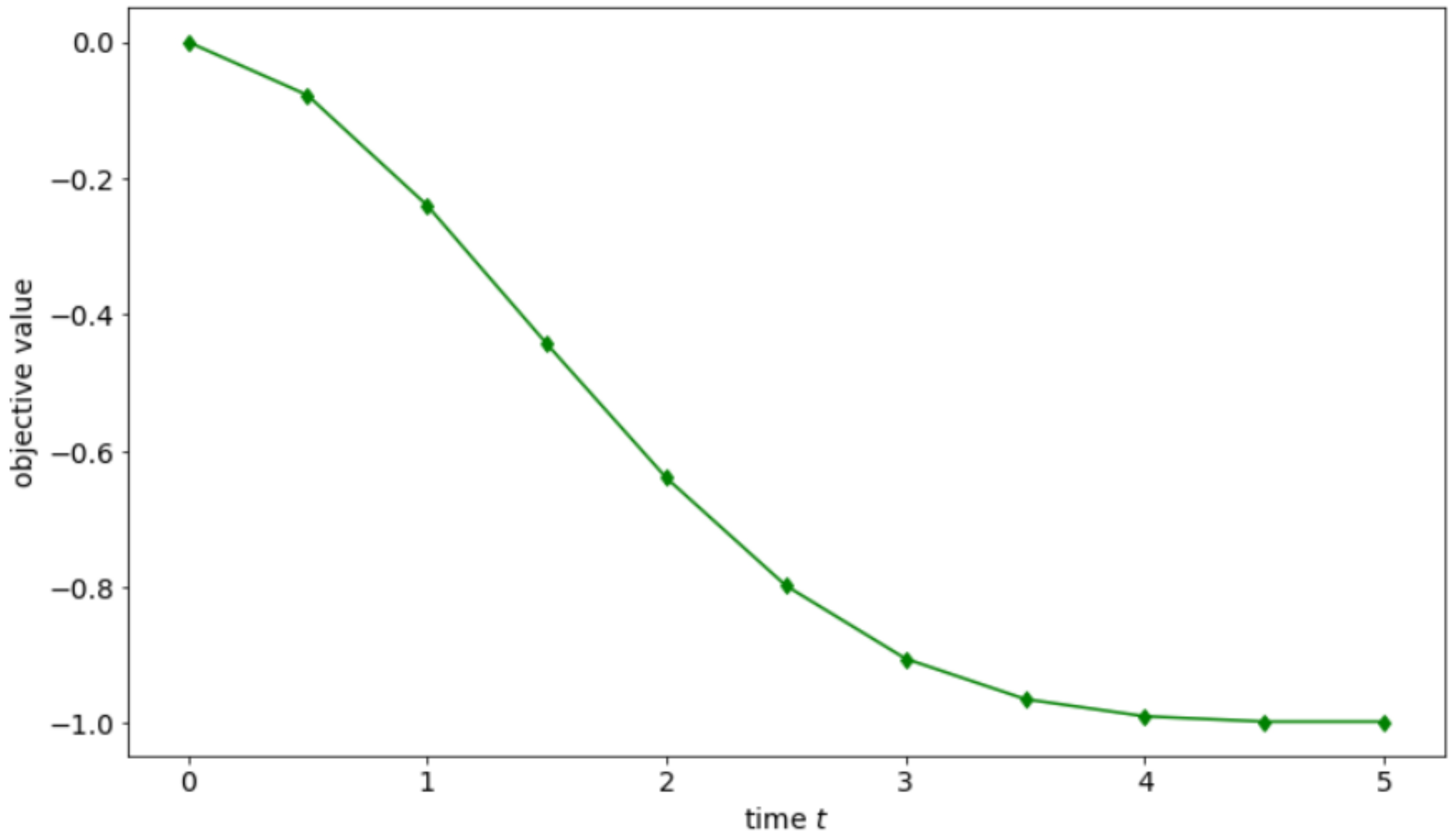
Trotterized Annealing

- Note that as we are building the complete circuit we are simulating it one time step at a time using the state vector simulator
- This gives us a step by step picture of how the annealing process evolves
- We have two possible results |0> and |1> the following slide show how the probabilities of these results changes over time
- The following slide shows how the value of the objective function varies over time

Trotterized Annealing



Trotterized Annealing



Recap

- The natural extension of this to multiple qubits leads to the QAOA algorithm, which we won’t discuss
- There are two problems with this algorithm, first the circuit grow quite large
- Second, it assume logical qubits
- The VQE approach is not as demanding and can potentially run on existing NISQ quantum computers

Grover Optimization

- Yes, we can apply Grover’s algorithm to optimization!
- But, it’s not straight forward
- Give an overview of the algorithm, and then use the Qiskit version to solve our sample problem
- Recall that Grovers algorithm is based on an oracle for a function f(x)
- This function takes 2^n binary variables and produces a single binary result
- This function has value 1 on the items we are searching for

Grover Optimization

- With the function f(x) we have the oracle

$$U_f : |x\rangle_n \rightarrow (-1)^{f(x)}|x\rangle_n$$

- Recall that this produces a negative value on the location we are searching for
- We apply the oracle to all 2^n values, the ones we are searching for become negative, lowers the average value
- Then apply diffuser which reflects about the average value, to increase amplitude of the ones we are looking for

Grover Optimization

- Our problem is slightly different
- With a QUBO we still have 2^n binary inputs, but the result of our function is an integer
- That is we have

$$f : \{0, \ldots, 2^n - 1\} \to \mathbb{Z},$$

- To use Grover’s algorithm we need a binary function for the oracle, we need some way of converting this function into a binary one

Grover Optimization

- This leads us to Grover adaptive search
- We are looking for the smallest value of f(x)
- Select some value x' and assume that f(x') is the smallest value
- We now use Grover's algorithm to search for all the values of x such that f(x)<f(x')
- We can turn this into an oracle in the following way

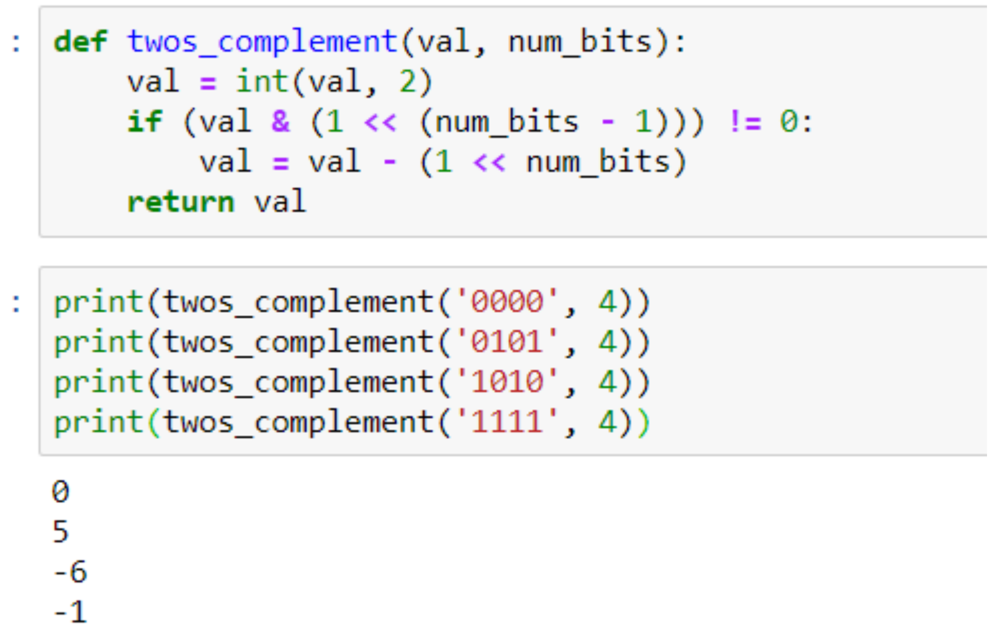
$$U_f(x') : |x\rangle_n \rightarrow (-1)^{f(x)<f(x')}|x\rangle_n$$

Grover Optimization

- After applying Grover’s algorithm we will have a value x that makes f(x) smaller
- We now use this value of x as x’ and repeat the process
- We repeat the process until we can’t find a smaller value of f(x)
- That’s the general idea
- Now we need to convert f(x) into a circuit that we can implement on a quantum computer
- This involves a few steps and some techniques we’ve seen before

Grover Optimization

- The first thing we need is 2’s complement arithmetic, going from a bit string to an integer is done as follows



```
: def twos_complement(val, num_bits):
      val = int(val, 2)
      if (val & (1 << (num_bits - 1))) != 0:
          val = val - (1 << num_bits)
      return val
```

```
: print(twos_complement('0000', 4))
  print(twos_complement('0101', 4))
  print(twos_complement('1010', 4))
  print(twos_complement('1111', 4))
```

```
0
5
-6
-1
```

Grover Optimization

- Now we need to encode an integer k, in a form where we can do addition on a quantum computer
- Here's how we can do it

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} e^{2\pi i \frac{xk}{2^n}} |x\rangle_n = \bigotimes_{i=0}^{n-1} R_Z\left(2\pi \frac{2^i}{2^n} k\right) |+\rangle_n =: U_k |+\rangle_n .$$

- The left side looks just like our Fourier transform
- The next expression shows how we can implement it with gates
- If we apply U_k to |+> we have a representation of k

Grover Optimization

- To get back our original value we just need to apply the inverse transform

$$QFT^{-1}U_k|+\rangle_n = |k\rangle_n.$$

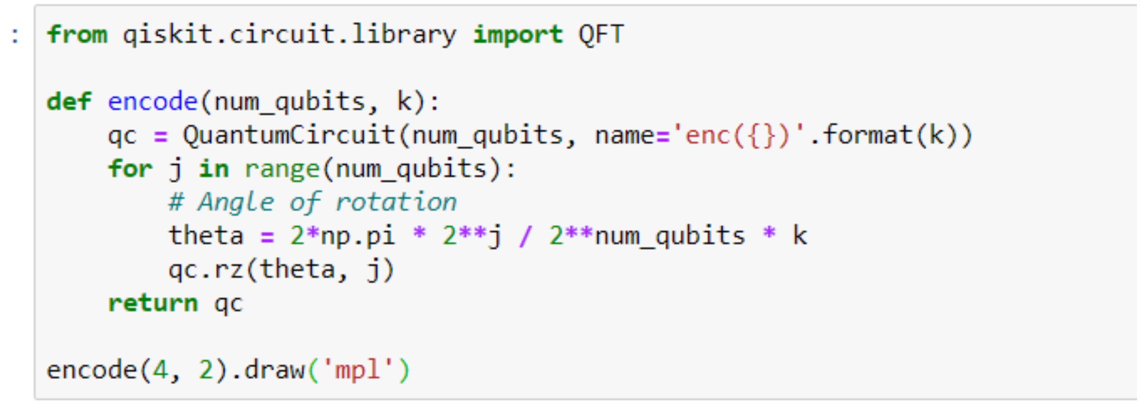
- But there is a bit of a trick we can play here

$$QFT^{-1}U_{k_2}U_{k_1}|+\rangle_n = |k_1 + k_2\rangle_n.$$

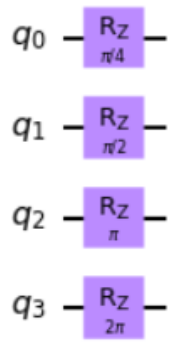
- We now have addition

Grover Optimization

- Now let’s start implementing this idea
- Start with the R_Z gates

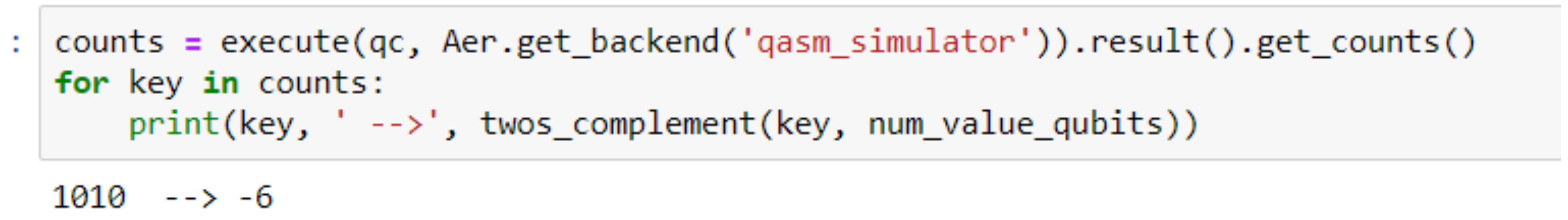


```
from qiskit.circuit.library import QFT

def encode(num_qubits, k):
    qc = QuantumCircuit(num_qubits, name='enc({})'.format(k))
    for j in range(num_qubits):
        # Angle of rotation
        theta = 2*np.pi * 2**j / 2**num_qubits * k
        qc.rz(theta, j)
    return qc

encode(4, 2).draw('mpl')
```



Grover Optimization

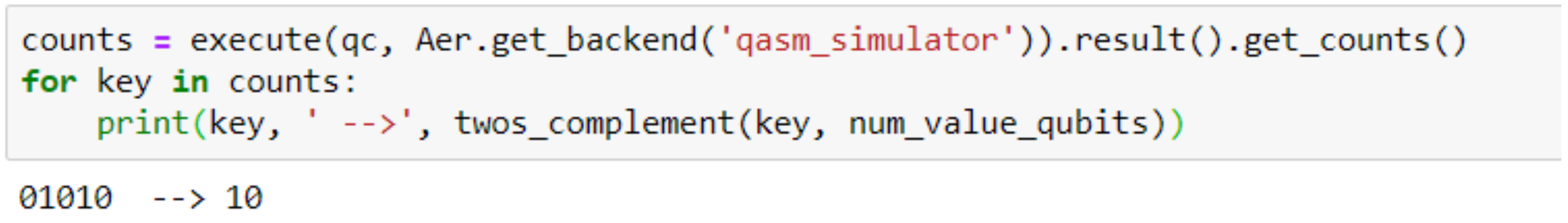
- Let’s put this all together, the code and circuit are on the next slide
- We encode 10 in 4 bits, this gives us



```
counts = execute(qc, Aer.get_backend('qasm_simulator')).result().get_counts()
for key in counts:
    print(key, ' -->', twos_complement(key, num_value_qubits))
```

```
1010  --> -6
```

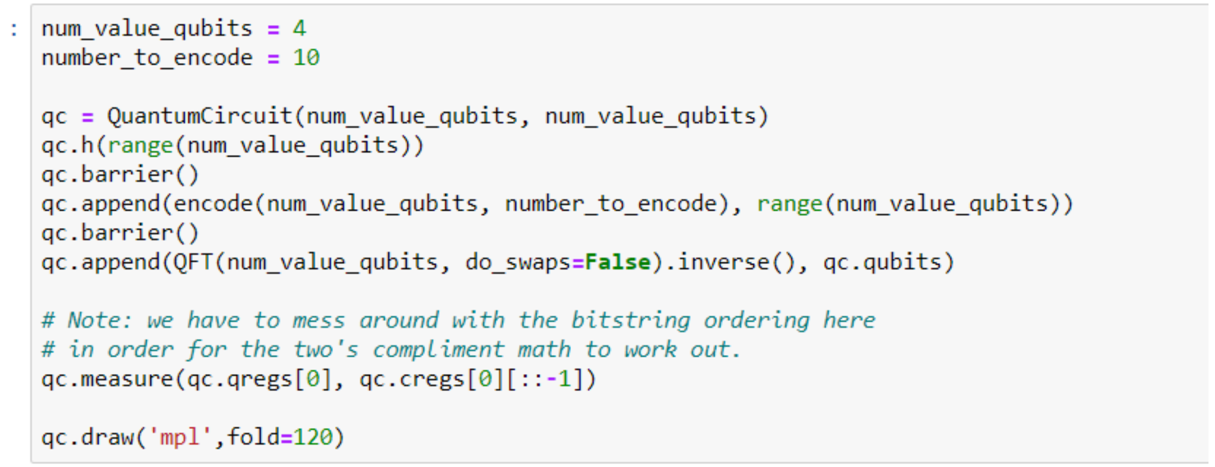
- What happened? I can’t encode 10 in 4 bits, add an extra bit



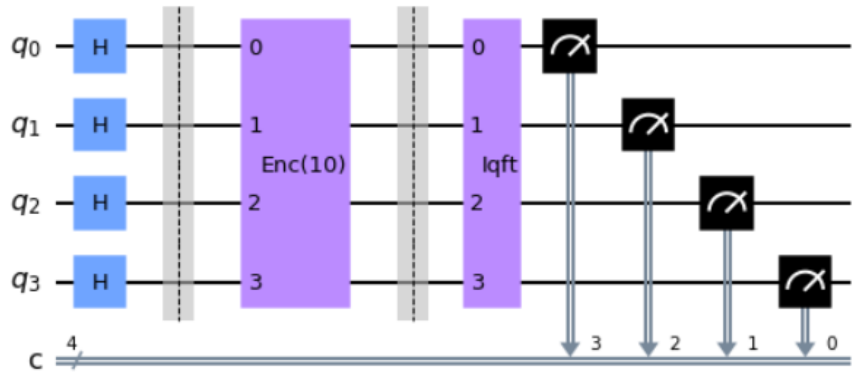
```
counts = execute(qc, Aer.get_backend('qasm_simulator')).result().get_counts()
for key in counts:
    print(key, ' -->', twos_complement(key, num_value_qubits))
```

```
01010  --> 10
```

Grover Optimization

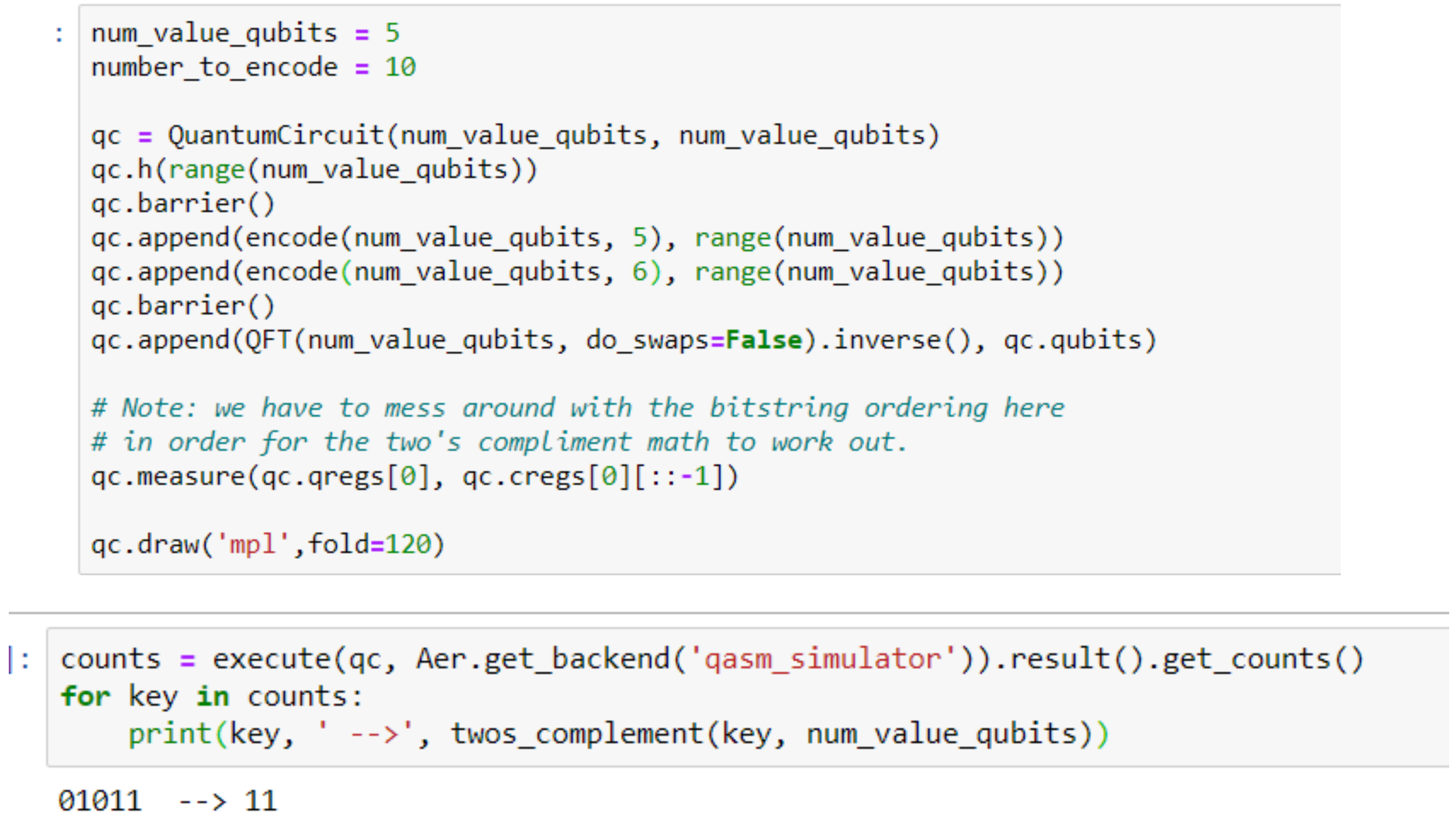


```
num_value_qubits = 4
number_to_encode = 10

qc = QuantumCircuit(num_value_qubits, num_value_qubits)
qc.h(range(num_value_qubits))
qc.barrier()
qc.append(encode(num_value_qubits, number_to_encode), range(num_value_qubits))
qc.barrier()
qc.append(QFT(num_value_qubits, do_swaps=False).inverse(), qc.qubits)

# Note: we have to mess around with the bitstring ordering here
# in order for the two's compliment math to work out.
qc.measure(qc.qregs[0], qc.cregs[0][::-1])

qc.draw('mpl',fold=120)
```



Grover Optimization

- Stick with 5 bits and try addition

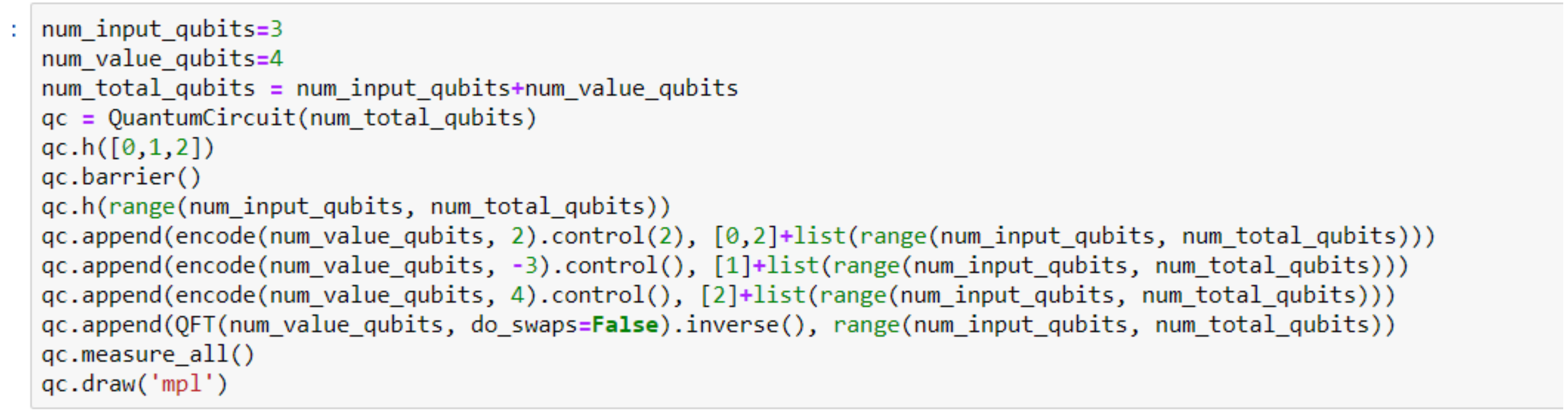


```
:  num_value_qubits = 5
   number_to_encode = 10

   qc = QuantumCircuit(num_value_qubits, num_value_qubits)
   qc.h(range(num_value_qubits))
   qc.barrier()
   qc.append(encode(num_value_qubits, 5), range(num_value_qubits))
   qc.append(encode(num_value_qubits, 6), range(num_value_qubits))
   qc.barrier()
   qc.append(QFT(num_value_qubits, do_swaps=False).inverse(), qc.qubits)

   # Note: we have to mess around with the bitstring ordering here
   # in order for the two's compliment math to work out.
   qc.measure(qc.qregs[0], qc.cregs[0][::-1])

   qc.draw('mpl',fold=120)
```

```
]: counts = execute(qc, Aer.get_backend('qasm_simulator')).result().get_counts()
   for key in counts:
       print(key, ' -->', twos_complement(key, num_value_qubits))

   01011  --> 11
```

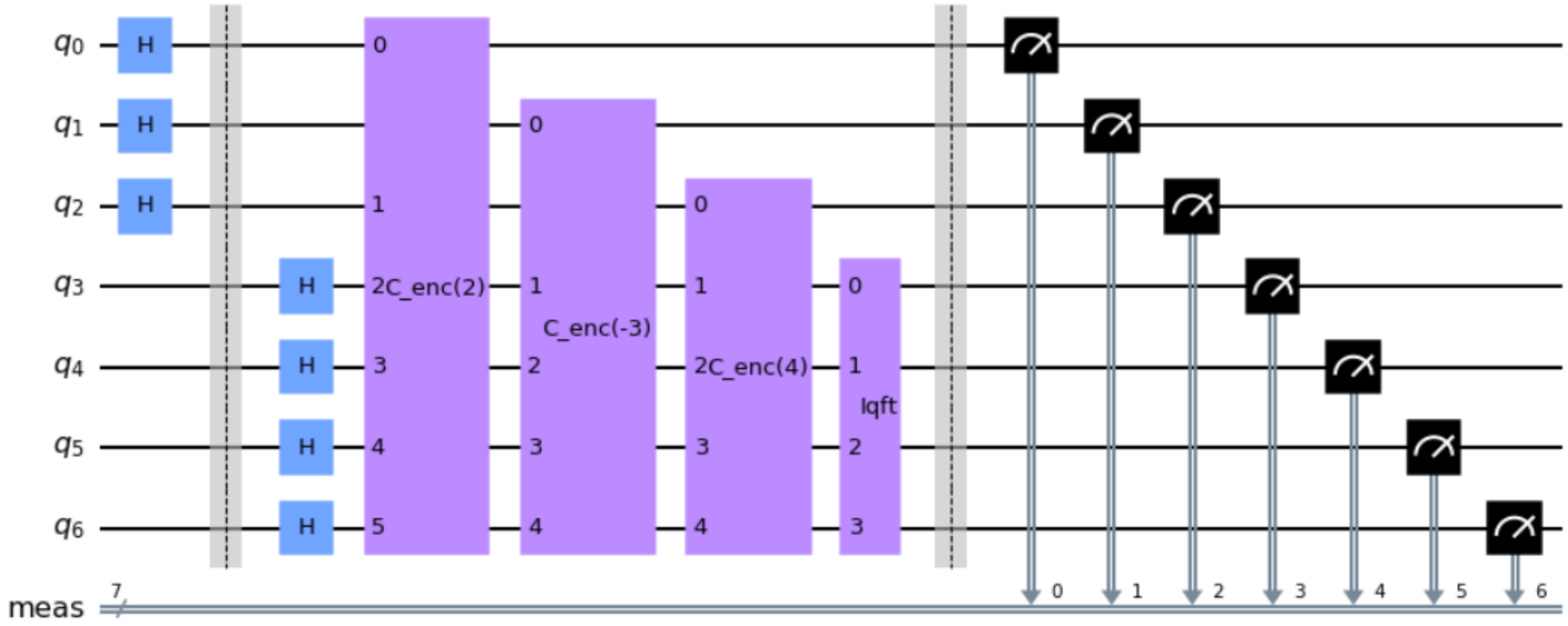
Grover Optimization

- Now we get to how we can evaluate f(x)
- Let’s make our circuit a bit more complicated



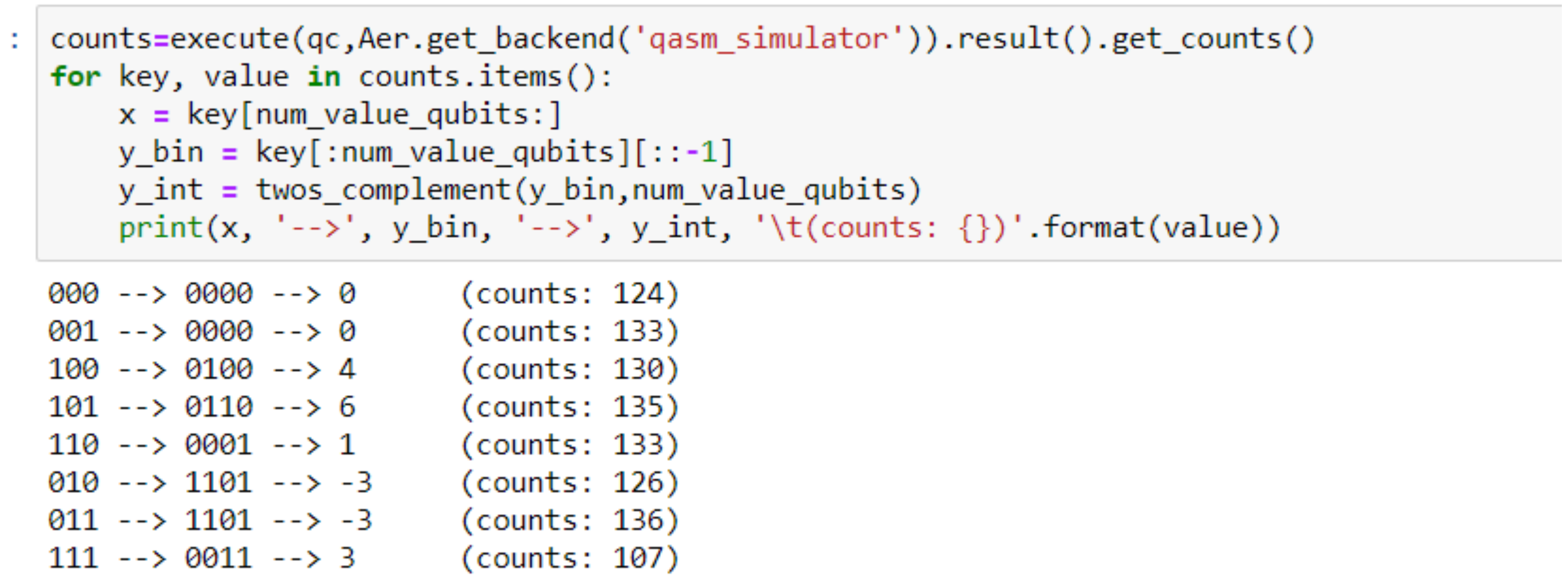
```
num_input_qubits=3
num_value_qubits=4
num_total_qubits = num_input_qubits+num_value_qubits
qc = QuantumCircuit(num_total_qubits)
qc.h([0,1,2])
qc.barrier()
qc.h(range(num_input_qubits, num_total_qubits))
qc.append(encode(num_value_qubits, 2).control(2), [0,2]+list(range(num_input_qubits, num_total_qubits)))
qc.append(encode(num_value_qubits, -3).control(), [1]+list(range(num_input_qubits, num_total_qubits)))
qc.append(encode(num_value_qubits, 4).control(), [2]+list(range(num_input_qubits, num_total_qubits)))
qc.append(QFT(num_value_qubits, do_swaps=False).inverse(), range(num_input_qubits, num_total_qubits))
qc.measure_all()
qc.draw('mpl')
```

Grover Optimization



Grover Optimization

- Note that we are now putting controls on our circuit, we can then run some code to evaluate this



```
counts=execute(qc,Aer.get_backend('qasm_simulator')).result().get_counts()
for key, value in counts.items():
    x = key[num_value_qubits:]
    y_bin = key[:num_value_qubits][::-1]
    y_int = twos_complement(y_bin,num_value_qubits)
    print(x, '-->', y_bin, '-->', y_int, '\t(counts: {})'.format(value))
```

```
000 --> 0000 --> 0      (counts: 124)
001 --> 0000 --> 0      (counts: 133)
100 --> 0100 --> 4      (counts: 130)
101 --> 0110 --> 6      (counts: 135)
110 --> 0001 --> 1      (counts: 133)
010 --> 1101 --> -3     (counts: 126)
011 --> 1101 --> -3     (counts: 136)
111 --> 0011 --> 3      (counts: 107)
```

Grover Optimization

- Look carefully at the values and how we are controlling the encodings
- We are evaluating

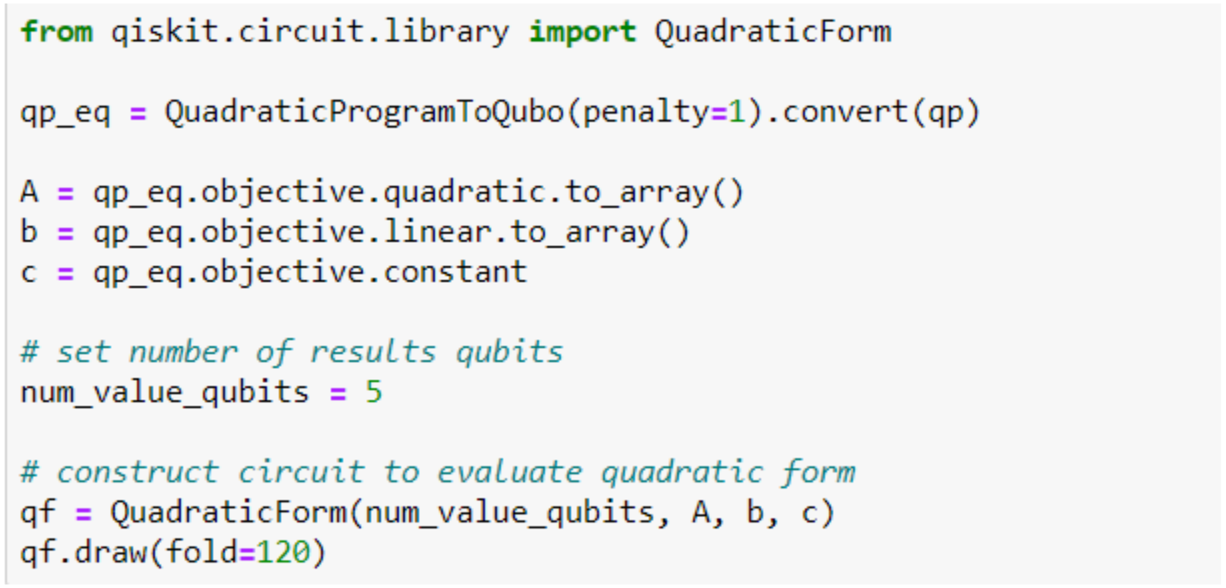
$$Q(x) = 2x_0x_2 - 3x_1 + x_2$$

- Note that this is a QUBO
- We now have a way of evaluating QUBOs, which are our objective function
- We now have a good part of our oracle!

Grover Optimization

- Note what’s going on with construction the QUBO, the quadratic terms are double controlled, the linear terms are single controlled, for a constant there would be no control input
- For each optimization we could build the circuit ourselves, but this is a lot of work
- Instead we can use the QuadraticForm object in Qiskit to do this for us, see next slide
- Note, I had to set the penalty for linear constraints to 1 in order to make the rest of this example work

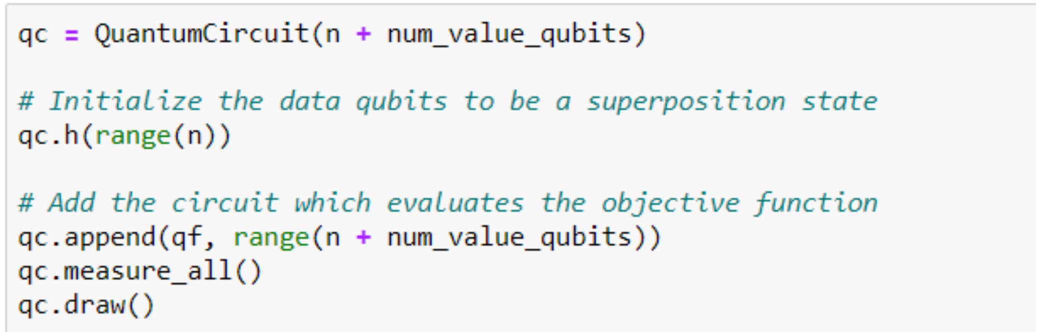
Grover Optimization



```
from qiskit.circuit.library import QuadraticForm

qp_eq = QuadraticProgramToQubo(penalty=1).convert(qp)

A = qp_eq.objective.quadratic.to_array()
b = qp_eq.objective.linear.to_array()
c = qp_eq.objective.constant

# set number of results qubits
num_value_qubits = 5

# construct circuit to evaluate quadratic form
qf = QuadraticForm(num_value_qubits, A, b, c)
qf.draw(fold=120)
```

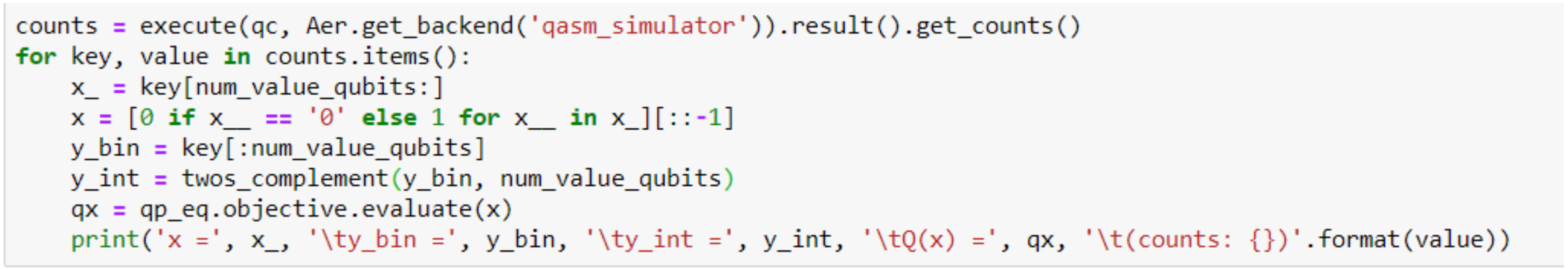
Grover Optimization

- Next build a circuit based on this



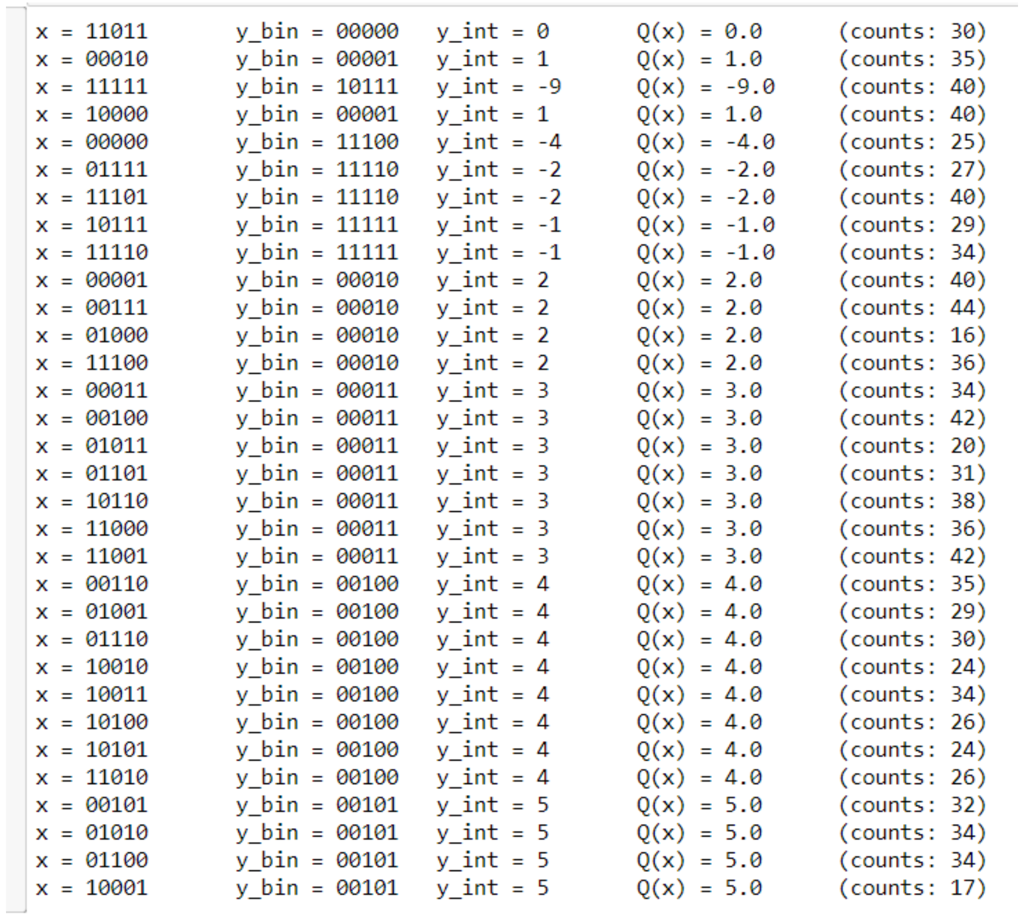
```
qc = QuantumCircuit(n + num_value_qubits)

# Initialize the data qubits to be a superposition state
qc.h(range(n))

# Add the circuit which evaluates the objective function
qc.append(qf, range(n + num_value_qubits))
qc.measure_all()
qc.draw()
```

- And evaluate the objective



```
counts = execute(qc, Aer.get_backend('qasm_simulator')).result().get_counts()
for key, value in counts.items():
    x_ = key[num_value_qubits:]
    x = [0 if x__ == '0' else 1 for x__ in x_][::-1]
    y_bin = key[:num_value_qubits]
    y_int = twos_complement(y_bin, num_value_qubits)
    qx = qp_eq.objective.evaluate(x)
    print('x =', x_, '\ty_bin =', y_bin, '\ty_int =', y_int, '\tQ(x) =', qx, '\t(counts: {})'.format(value))
```

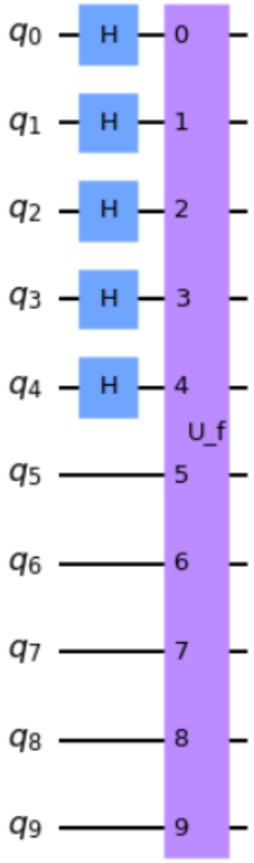
Grover Optimization



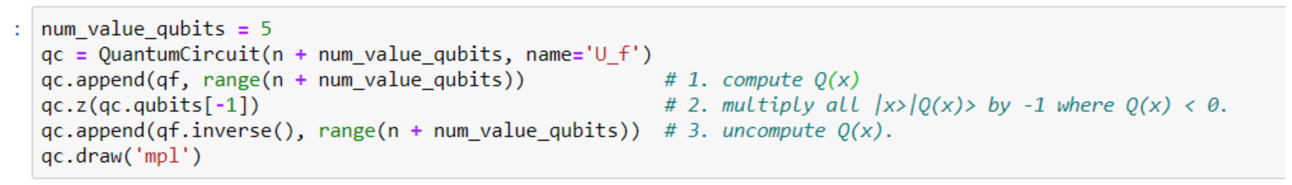
```
x = 11011      y_bin = 00000   y_int = 0      Q(x) = 0.0      (counts: 30)
x = 00010      y_bin = 00001   y_int = 1      Q(x) = 1.0      (counts: 35)
x = 11111      y_bin = 10111   y_int = -9     Q(x) = -9.0     (counts: 40)
x = 10000      y_bin = 00001   y_int = 1      Q(x) = 1.0      (counts: 40)
x = 00000      y_bin = 11100   y_int = -4     Q(x) = -4.0     (counts: 25)
x = 01111      y_bin = 11110   y_int = -2     Q(x) = -2.0     (counts: 27)
x = 11101      y_bin = 11110   y_int = -2     Q(x) = -2.0     (counts: 40)
x = 10111      y_bin = 11111   y_int = -1     Q(x) = -1.0     (counts: 29)
x = 11110      y_bin = 11111   y_int = -1     Q(x) = -1.0     (counts: 34)
x = 00001      y_bin = 00010   y_int = 2      Q(x) = 2.0      (counts: 40)
x = 00111      y_bin = 00010   y_int = 2      Q(x) = 2.0      (counts: 44)
x = 01000      y_bin = 00010   y_int = 2      Q(x) = 2.0      (counts: 16)
x = 11100      y_bin = 00010   y_int = 2      Q(x) = 2.0      (counts: 36)
x = 00011      y_bin = 00011   y_int = 3      Q(x) = 3.0      (counts: 34)
x = 00100      y_bin = 00011   y_int = 3      Q(x) = 3.0      (counts: 42)
x = 01011      y_bin = 00011   y_int = 3      Q(x) = 3.0      (counts: 20)
x = 01101      y_bin = 00011   y_int = 3      Q(x) = 3.0      (counts: 31)
x = 10110      y_bin = 00011   y_int = 3      Q(x) = 3.0      (counts: 38)
x = 11000      y_bin = 00011   y_int = 3      Q(x) = 3.0      (counts: 36)
x = 11001      y_bin = 00011   y_int = 3      Q(x) = 3.0      (counts: 42)
x = 00110      y_bin = 00100   y_int = 4      Q(x) = 4.0      (counts: 35)
x = 01001      y_bin = 00100   y_int = 4      Q(x) = 4.0      (counts: 29)
x = 01110      y_bin = 00100   y_int = 4      Q(x) = 4.0      (counts: 30)
x = 10010      y_bin = 00100   y_int = 4      Q(x) = 4.0      (counts: 24)
x = 10011      y_bin = 00100   y_int = 4      Q(x) = 4.0      (counts: 34)
x = 10100      y_bin = 00100   y_int = 4      Q(x) = 4.0      (counts: 26)
x = 10101      y_bin = 00100   y_int = 4      Q(x) = 4.0      (counts: 24)
x = 11010      y_bin = 00100   y_int = 4      Q(x) = 4.0      (counts: 26)
x = 00101      y_bin = 00101   y_int = 5      Q(x) = 5.0      (counts: 32)
x = 01010      y_bin = 00101   y_int = 5      Q(x) = 5.0      (counts: 34)
x = 01100      y_bin = 00101   y_int = 5      Q(x) = 5.0      (counts: 34)
x = 10001      y_bin = 00101   y_int = 5      Q(x) = 5.0      (counts: 17)
```

Grover Optimization

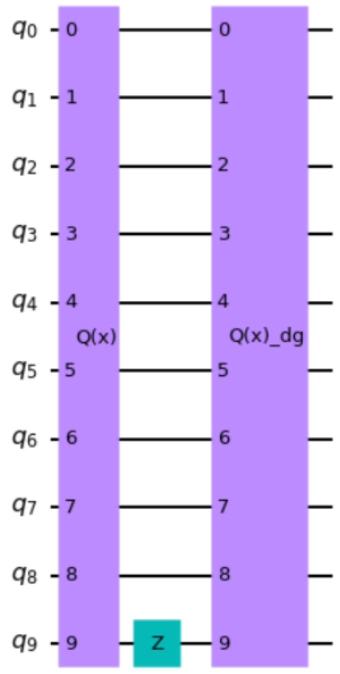
- The y_int value is the value computed by the circuit and Q(x) is the result of classically evaluating the objective, they agree
- The next slide shows how we build the oracle
- Recall that the top bit is the sign bit, the Z gate does the multiply by -1 when the objective is less than zero
- To complete the circuit add some H gates



Grover Optimization

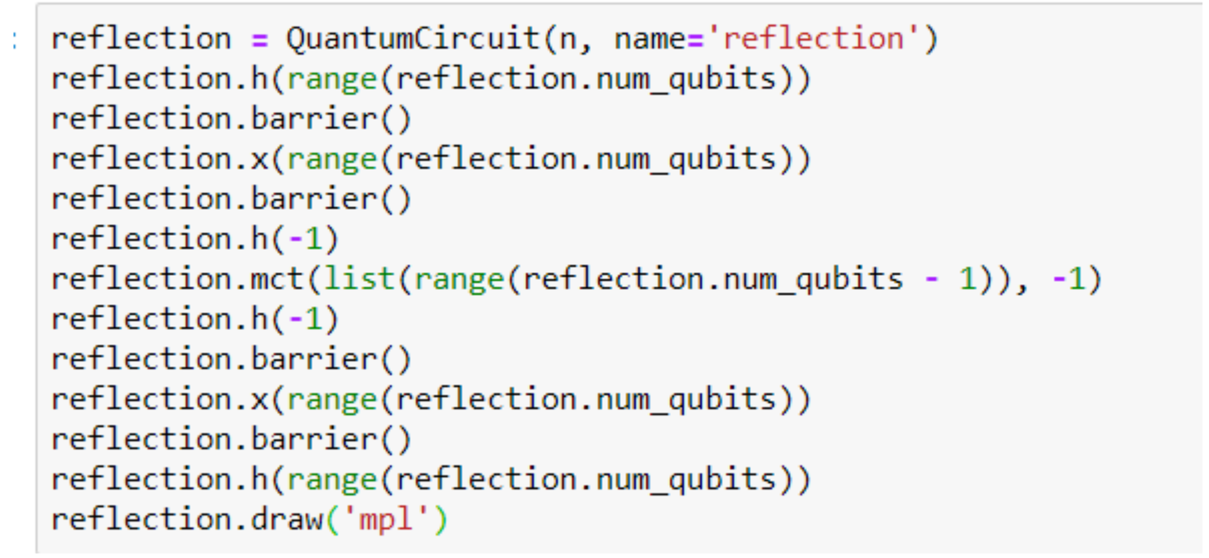


```
num_value_qubits = 5
qc = QuantumCircuit(n + num_value_qubits, name='U_f')
qc.append(qf, range(n + num_value_qubits))             # 1. compute Q(x)
qc.z(qc.qubits[-1])                                    # 2. multiply all |x>|Q(x)> by -1 where Q(x) < 0.
qc.append(qf.inverse(), range(n + num_value_qubits))  # 3. uncompute Q(x).
qc.draw('mpl')
```



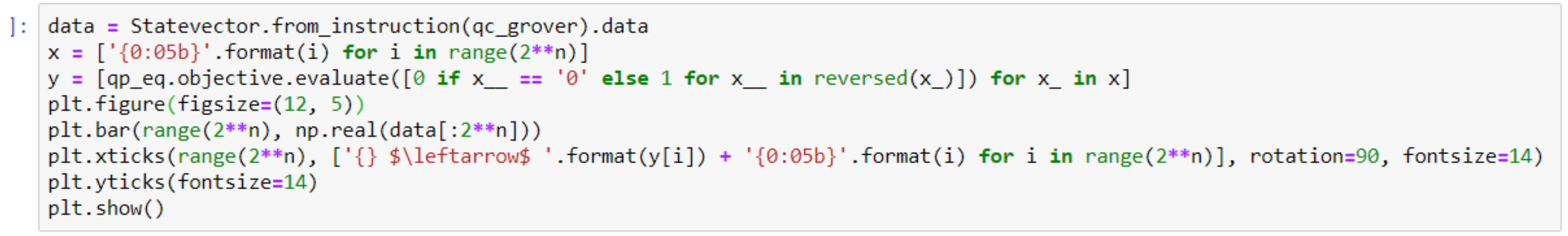
Grover Optimization

- The next slide shows the first step of the algorithm, this doesn’t agree with what we have seen before, since we are now doing a minimization and not a maximization
- Now we add a diffuser

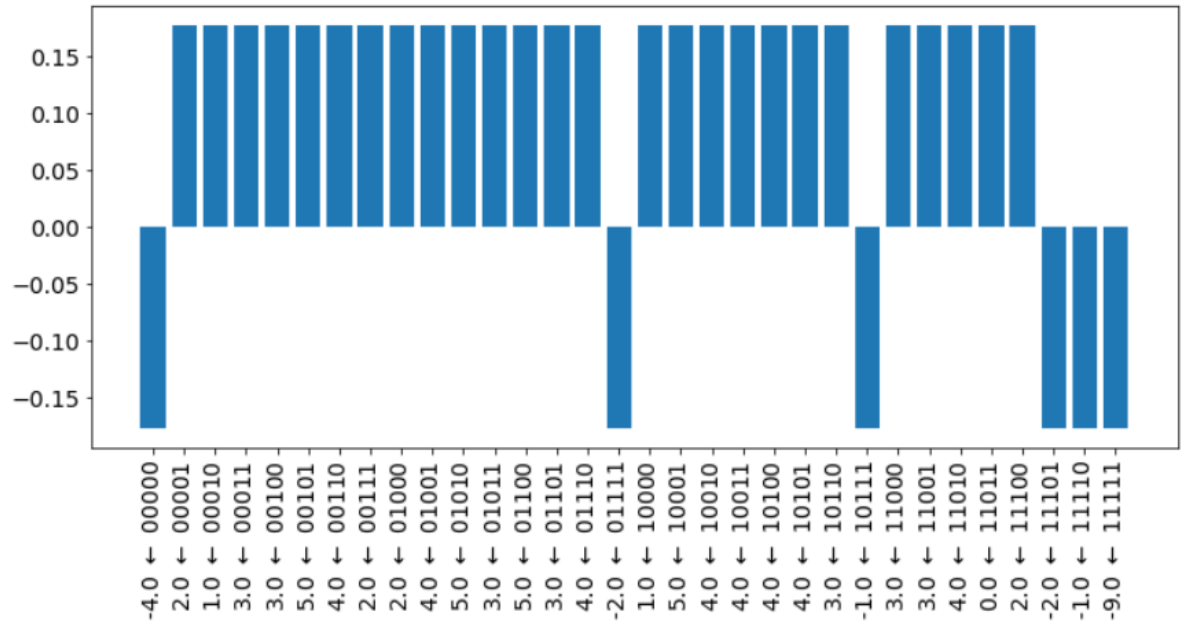


```
reflection = QuantumCircuit(n, name='reflection')
reflection.h(range(reflection.num_qubits))
reflection.barrier()
reflection.x(range(reflection.num_qubits))
reflection.barrier()
reflection.h(-1)
reflection.mct(list(range(reflection.num_qubits - 1)), -1)
reflection.h(-1)
reflection.barrier()
reflection.x(range(reflection.num_qubits))
reflection.barrier()
reflection.h(range(reflection.num_qubits))
reflection.draw('mpl')
```

Grover Optimization

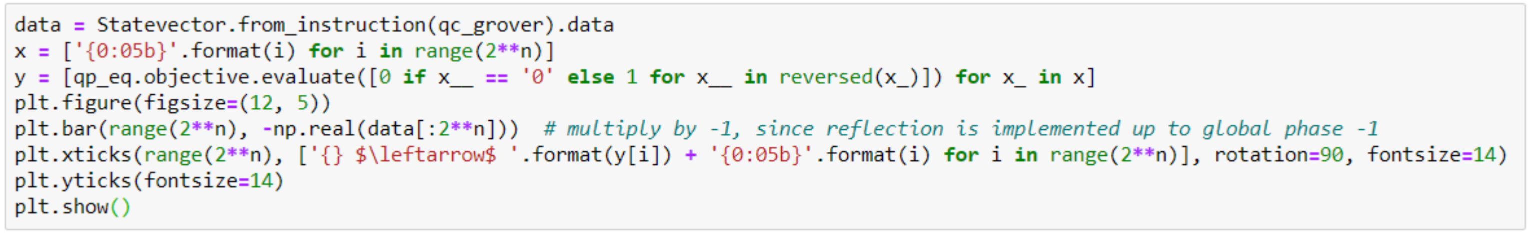


```
]: data = Statevector.from_instruction(qc_grover).data
   x = ['{0:05b}'.format(i) for i in range(2**n)]
   y = [qp_eq.objective.evaluate([0 if x__ == '0' else 1 for x__ in reversed(x_)]) for x_ in x]
   plt.figure(figsize=(12, 5))
   plt.bar(range(2**n), np.real(data[:2**n]))
   plt.xticks(range(2**n), ['{} $\leftarrow$ '.format(y[i]) + '{0:05b}'.format(i) for i in range(2**n)], rotation=90, fontsize=14)
   plt.yticks(fontsize=14)
   plt.show()
```

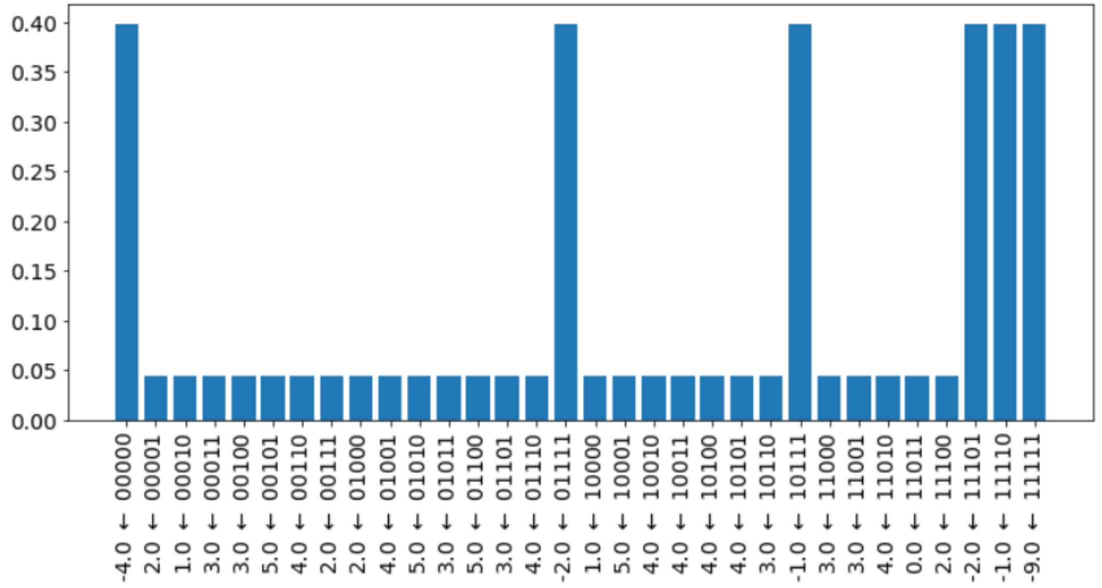


Grover Optimization

- With the diffuser we get the following



```
data = Statevector.from_instruction(qc_grover).data
x = ['{0:05b}'.format(i) for i in range(2**n)]
y = [qp_eq.objective.evaluate([0 if x__ == '0' else 1 for x__ in reversed(x_)]) for x_ in x]
plt.figure(figsize=(12, 5))
plt.bar(range(2**n), -np.real(data[:2**n]))  # multiply by -1, since reflection is implemented up to global phase -1
plt.xticks(range(2**n), ['{} $\leftarrow$ '.format(y[i]) + '{0:05b}'.format(i) for i in range(2**n)], rotation=90, fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



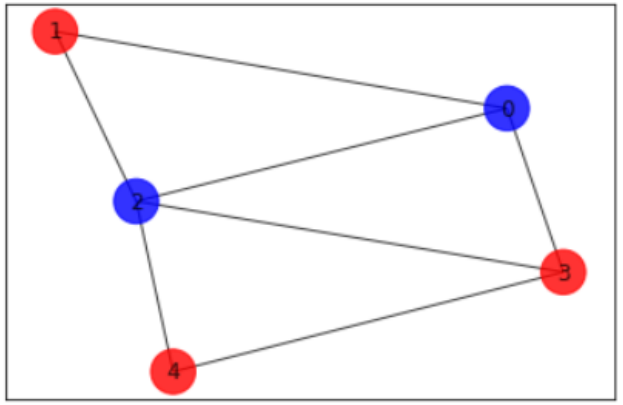
Grover Optimization

- In order to get a proper maximization we would need to multiply the objective by -1
- This looks like a lot of work, but Qiskit provides a Grover optimizer for us
- We just need to give it a quadratic problem, and it will construct the appropriate objective and run it
- This is shown on the next slide

Grover Optimization

```
: from qiskit.optimization.algorithms import GroverOptimizer

  # set up Grover Optimizer
  grover = GroverOptimizer(num_value_qubits=5, penalty=1, quantum_instance=Aer.get_backend('statevector_simulator'))

  # solver problem
  result = grover.solve(qp)

  print(result)
  plot_result(G, result.x)
```

```
optimal function value: 5.0
optimal value: [1 0 1 0 0]
status: SUCCESS
```



Grover Optimization

- This all looks easy when I present it in a lecture, but that’s not the case, there was a lot of hacking around required
- The penalty of 1 is not the default, with the default value it doesn’t work, I had to spend some time finding a value that would work
- The result on the previous slide is not what happens the first time
- I had to run the code multiple times to get the optimal result
- Other times I got a close result or an error
- Clearly this is not like a classical computer!

More on Quadratic Programs

- So far we have started with a DOcplex and converted this to a QuadraticForm
- This works well for some applications, but not for others
- If we already have the matrix of quadratic coefficients and the vector of linear coefficients, why bother converting them to a DOcplex just to have them converted back to matrices and vectors?
- We can construct our quadratic programs in a different way to make this easier

More on Quadratic Programs

- Start as usual by creating a quadratic program object
- Then in the call to maximize or minimize pass in the coefficients for the quadratic form
- Example:

```
mod.minimize(constant=3, linear=[1,0,0], quadratic=[[0,1,0],[1,0,0],[0,0,-1]])
```

- The linear term is a Python 1D array and the quadratic term is a Python 2D array

More on Quadratic Programs

- You can also add constraints to the program by calling the linear_constraint function
- Example:

```
mod.linear_constraint(linear={'x': 1, 'y': 2}, sense='==', rhs=3, name='lin_eq’)
```

- The linear parameter is a dictionary, the index is the variable name and the value is the coefficient
- The rhs parameter is the value of the constraint
- The sense parameter is the comparison to be used

Summary

- Have examined several approaches to optimization on quantum computers
- Gone through the process of converting an optimization problem into a program for a quantum computer
- Most interesting problems are too large to run on existing quantum computers
- Getting the correct answer is not always a straight forward process