Qiskit Document Tutorial勉強会
Optimization

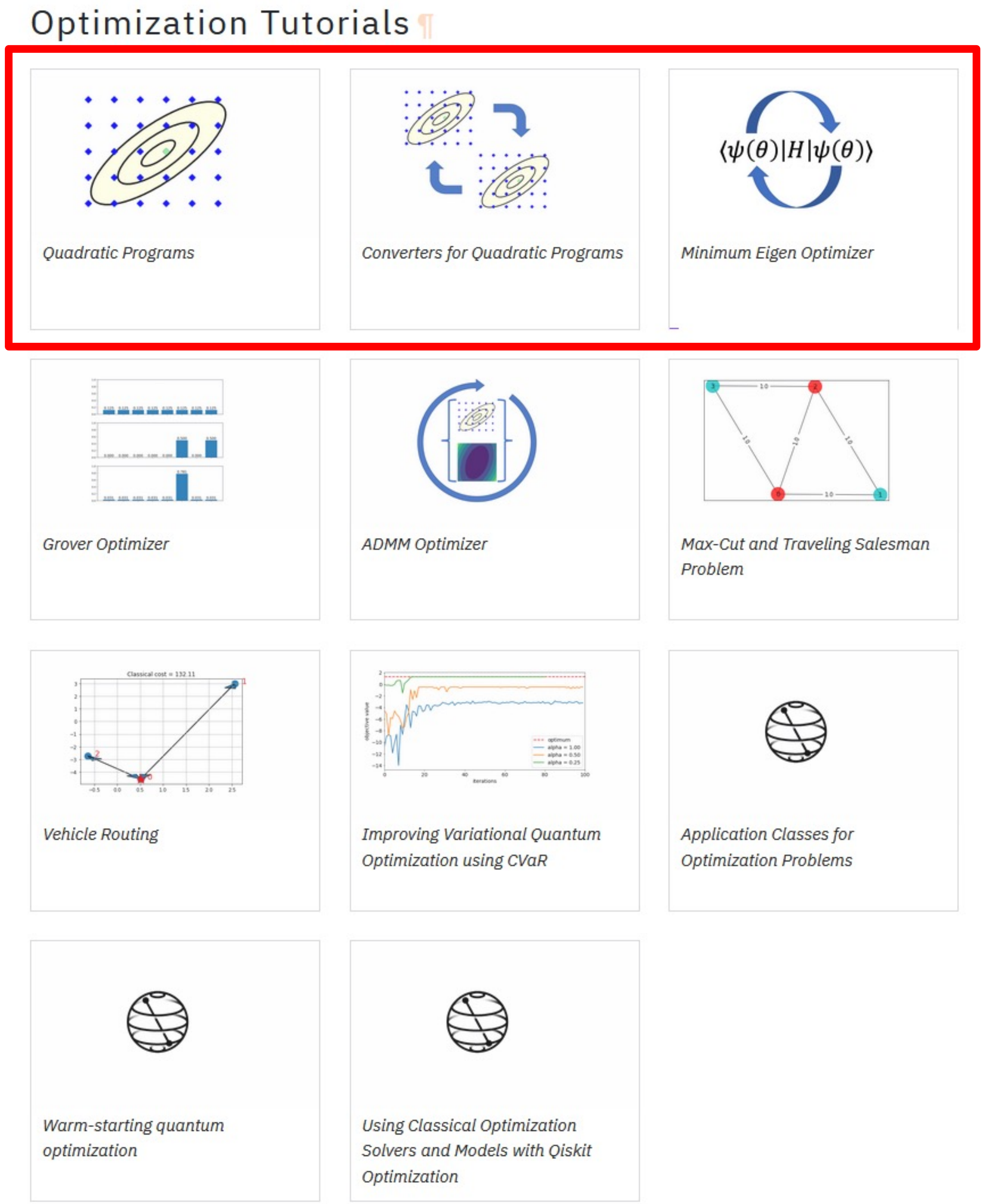
Shun Shirakawa

Agenda

1. Quadratic Programs
   2次計画問題

2. Converters for Quadratic Programs
   量子アルゴリズムで解ける形への変換

3. Minimum Eigen Optimizer
   最小固有値ソルバーによる求解

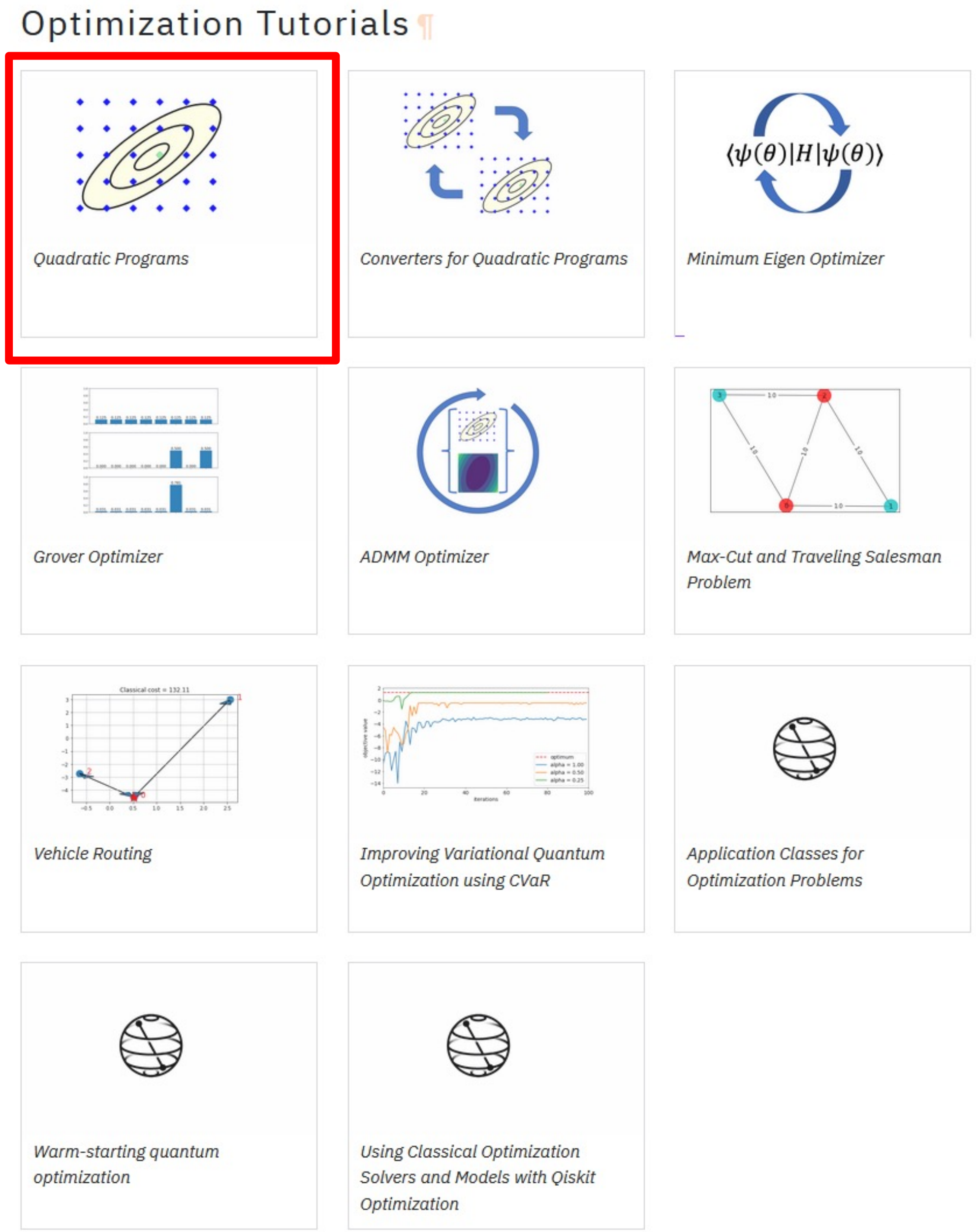


Agenda

1. Quadratic Programs
   2次計画問題

2. Converters for Quadratic Programs
   量子アルゴリズムで解ける形への変換

3. Minimum Eigen Optimizer
   最小固有値ソルバーによる求解



Quadratic Programs

2次計画問題

$$
\begin{aligned}
\text{minimize} \quad & x^\top Q_0 x + c^\top x \\
\text{subject to} \quad & Ax \le b \\
& x^\top Q_i x + a_i^\top x \le r_i, \quad 1, \dots, i, \dots, q \\
& l_i \le x_i \le u_i, \quad 1, \dots, i, \dots, n,
\end{aligned}
$$

$\boldsymbol{Q_i}$: n×n行列
$\boldsymbol{A}$: m×n行列
$\boldsymbol{x}, \boldsymbol{c}$: n次元ベクトル
$\boldsymbol{b}$: m次元ベクトル
$\boldsymbol{x}$ の成分（変数）は、0-1／整数／実数

qiskitではQuadraticProgramクラスで表現

[1]:
```
from qiskit_optimization import QuadraticProgram
from qiskit_optimization.translators import from_docplex_mp
```

CPLEXモデルから構築する方法

まずCPLEXモデルとして構築し、それをQuadraticProgramに取り込む

CPLEX: IBMの数理最適化ソフトウェア製品
　線形計画法、混合整数計画法、2次計画法、制約プログラミングなどのソルバー
　pythonで扱うためのdocplexパッケージが提供されている

例）決定変数：$x \in \{0,1\}, y \in \mathbb{Z}\ (-1 \le y \le 5)$
　　目的関数：$x + 2y$（最小化）
　　制約条件：$x - y = 3, (x+y)(x-y) \le 1$

Minimize(Maximize)	目的関数
Subject To	制約条件
Bounds	変数の値域
Binaries	0-1変数
Generals	整数変数

[1]:
```
from qiskit_optimization import QuadraticProgram
from qiskit_optimization.translators import from_docplex_mp
```

[2]:
```
# Make a Docplex model
from docplex.mp.model import Model

mdl = Model('docplex model')
x = mdl.binary_var('x')
y = mdl.integer_var(lb=-1, ub=5, name='y')
mdl.minimize(x + 2 * y)
mdl.add_constraint(x - y == 3)
mdl.add_constraint((x + y) * (x - y) <= 1)
print(mdl.export_as_lp_string())
```

docplex APIを用いて
CPLEXモデルを構築

[3]:
```
# load from a Docplex model
mod = from_docplex_mp(mdl)
print(mod.export_as_lp_string())
```

CPLEXモデルから
QuadraticProgramを
構築する

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

Minimize
 obj: x + 2 y
Subject To
 c0: x - y = 3
 q0: [ x^2 - y^2 ] <= 1

Bounds
 0 <= x <= 1
 -1 <= y <= 5

Binaries
 x

Generals
 y
End
```

QuadraticProgramを直接構築する方法

モデルの定義

```
[4]: # make an empty problem
     mod = QuadraticProgram('my problem')
     print(mod.export_as_lp_string())
```

決定変数: binary_var / integer_var / continuous_var

例）$x \in \{0,1\}, y \in \mathbb{Z}\ (-1 \leq y \leq 5), z \in \mathbb{R}\ (-1 \leq z \leq 5)$

```
[5]: # Add variables
     mod.binary_var(name='x')
     mod.integer_var(name='y', lowerbound=-1, upperbound=5)
     mod.continuous_var(name='z', lowerbound=-1, upperbound=5)
     print(mod.export_as_lp_string())
```

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

Minimize
 obj: x + [ 4 x*y - 2 z^2 ]/2 + 3
Subject To

Bounds
 0 <= x <= 1
 -1 <= y <= 5
 -1 <= z <= 5

Binaries
 x

Generals
 y
End
```

Binaries（0-1）にも
Genarals（整数）にも出て
こないのが実数変数

name（変数名）
lowerbound（下限）
upperbound（上限）

変数名の先頭はE/e不可
（LPフォーマットの制限）

QuadraticProgramを直接構築する方法

目的関数: minimize / maximize

constant（定数項）
linear（一次項の係数）
quadratic（二次項の係数）

例）$2xy - z^2 + x + 3$（最小化）

各項の係数を辞書で与える方法

```
[6]: # Add objective function using dictionaries
mod.minimize(constant=3, linear={'x': 1}, quadratic={('x', 'y'): 2, ('z', 'z'): -1})
print(mod.export_as_lp_string())
```

定数項	3
一次項	x: 1
二次項	xy: 2, z^2: -1

各項の係数を行列で与える方法

```
[7]: # Add objective function using lists/arrays
mod.minimize(constant=3, linear=[1,0,0], quadratic=[[0,1,0],[1,0,0],[0,0,-1]])
print(mod.export_as_lp_string())
```

非対称行列で与えてもOK
[0,2,0],[0,0,0],[0,0,-1]

定数項 3

一次項 $\begin{matrix} 1 \\ 0 \\ 0 \end{matrix} \quad \leftarrow \quad \begin{matrix} x \\ y \\ z \end{matrix}$

二次項 $\begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{matrix} \quad \leftarrow \quad \begin{matrix} x^2 & xy & xz \\ yx & y^2 & yz \\ zx & zy & z^2 \end{matrix}$

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

Minimize
 obj: x + [ 4 x*y - 2 z^2 ]/2 + 3
Subject To

Bounds
 0 <= x <= 1
 -1 <= y <= 5
 -1 <= z <= 5

Binaries
 x

Generals
 y
End
```

2次部分の係数1/2は
LPフォーマットの規約

モデルの内容へのアクセス（目的関数）

```
[8]: print('constant:\t\t\t', mod.objective.constant)
print('linear dict:\t\t\t', mod.objective.linear.to_dict())
print('linear array:\t\t\t', mod.objective.linear.to_array())
print('linear array as sparse matrix:\n', mod.objective.linear.coefficients, '\n')
print('quadratic dict w/ index:\t', mod.objective.quadratic.to_dict())
print('quadratic dict w/ name:\t\t', mod.objective.quadratic.to_dict(use_name=True))
print('symmetric quadratic dict w/ name:\t', mod.objective.quadratic.to_dict(use_name=True,
symmetric=True))
print('quadratic matrix:\n', mod.objective.quadratic.to_array(),'\n')
print('symmetric quadratic matrix:\n', mod.objective.quadratic.to_array(symmetric=True),'\n')
print('quadratic matrix as sparse matrix:\n', mod.objective.quadratic.coefficients)
```

定数項
一次項（辞書形式）
一次項（配列形式）
一次項（疎な係数一覧）
二次項（辞書形式）
〃 ＋変数名
二次項（行列形式）
〃 ＋対称形式
〃 （疎な係数一覧）

```
constant:                          3
linear dict:                       {0: 1}
linear array:                      [1 0 0]
linear array as sparse matrix:
   (0, 0)       1

quadratic dict w/ index:           {(0, 1): 2, (2, 2): -1}
quadratic dict w/ name:            {('x', 'y'): 2, ('z', 'z'): -1}
symmetric quadratic dict w/ name:          {('y', 'x'): 1, ('x', 'y'): 1, ('z', 'z'): -1}
quadratic matrix:
 [[ 0  2  0]
 [ 0  0  0]
 [ 0  0 -1]]

symmetric quadratic matrix:
 [[ 0  1  0]
 [ 1  0  0]
 [ 0  0 -1]]

quadratic matrix as sparse matrix:
   (0, 1)       2
  (2, 2)        -1
```

QuadraticProgramを直接構築する方法（制約条件）

制約条件: linear_constraint / quadratic_constraint

linear（一次項の係数）
quadratic（二次項の係数）
sense（等号・不等号）
rhs（右辺値）

一次制約

[9]:
```python
# Add linear constraints
mod.linear_constraint(linear={'x': 1, 'y': 2}, sense='==', rhs=3, name='lin_eq')
mod.linear_constraint(linear={'x': 1, 'y': 2}, sense='<=', rhs=3, name='lin_leq')
mod.linear_constraint(linear={'x': 1, 'y': 2}, sense='>=', rhs=3, name='lin_geq')
print(mod.export_as_lp_string())
```

二次制約

[10]:
```python
# Add quadratic constraints
mod.quadratic_constraint(linear={'x': 1, 'y': 1}, quadratic={('x', 'x'): 1, ('y', 'z'): -1},
sense='==', rhs=1, name='quad_eq')
mod.quadratic_constraint(linear={'x': 1, 'y': 1}, quadratic={('x', 'x'): 1, ('y', 'z'): -1},
sense='<=', rhs=1, name='quad_leq')
mod.quadratic_constraint(linear={'x': 1, 'y': 1}, quadratic={('x', 'x'): 1, ('y', 'z'): -1},
sense='>=', rhs=1, name='quad_geq')
print(mod.export_as_lp_string())
```

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

Minimize
 obj: x + [ 4 x*y - 2 z^2 ]/2 + 3
Subject To
 lin_eq: x + 2 y = 3
 lin_leq: x + 2 y <= 3
 lin_geq: x + 2 y >= 3
 quad_eq: [ x^2 - y*z ] + x + y = 1
 quad_leq: [ x^2 - y*z ] + x + y <= 1
 quad_geq: [ x^2 - y*z ] + x + y >= 1

Bounds
 0 <= x <= 1
 -1 <= y <= 5
 -1 <= z <= 5

Binaries
 x

Generals
 y
End
```

モデルの内容へのアクセス（制約条件）

```
[11]: lin_geq = mod.get_linear_constraint('lin_geq')
      print('lin_geq:', lin_geq.linear.to_dict(use_name=True), lin_geq.sense, lin_geq.rhs)
      quad_geq = mod.get_quadratic_constraint('quad_geq')
      print('quad_geq:', quad_geq.linear.to_dict(use_name=True),
      quad_geq.quadratic.to_dict(use_name=True), quad_geq.sense, lin_geq.rhs)
```

一次制約の取得

二次制約の取得

```
lin_geq: {'x': 1.0, 'y': 2.0} ConstraintSense.GE 3
quad_geq: {'x': 1.0, 'y': 1.0} {('x', 'x'): 1.0, ('y', 'z'): -1.0} ConstraintSense.GE 3
```

制約条件の削除

```
[12]: # Remove constraints
      mod.remove_linear_constraint('lin_eq')
      mod.remove_quadratic_constraint('quad_leq')
      print(mod.export_as_lp_string())
```

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

Minimize
 obj: x + [ 4 x*y - 2 z^2 ]/2 + 3
Subject To
 lin_leq: x + 2 y <= 3
 lin_geq: x + 2 y >= 3
 quad_eq: [ x^2 - y*z ] + x + y = 1
 quad_geq: [ x^2 - y*z ] + x + y >= 1

Bounds
 0 <= x <= 1
 -1 <= y <= 5
 -1 <= z <= 5

Binaries
 x

Generals
 y
End
```

決定変数への代入

決定変数を定数または他の変数に置き換え

x に 0 を代入　　　　y に –z を代入

```
[13]: sub = mod.substitute_variables(constants={'x': 0}, variables={'y': ('z', -1)})
      print(sub.export_as_lp_string())
```

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

Minimize
 obj: x + [ 4 x*y - 2 z^2 ]/2 + 3
Subject To
 lin_leq: x + 2 y <= 3
 lin_geq: x + 2 y >= 3
 quad_eq: [ x^2 - y*z ] + x + y = 1
 quad_geq: [ x^2 - y*z ] + x + y >= 1

Bounds
 0 <= x <= 1
 -1 <= y <= 5
 -1 <= z <= 5

Binaries
 x

Generals
 y
End
```

$$x = 0$$
$$y = -z$$

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

Minimize
 obj: [ - 2 z^2 ]/2 + 3
Subject To
 lin_leq: - 2 z <= 3
 lin_geq: - 2 z >= 3
 quad_eq: [ z^2 ] - z = 1
 quad_geq: [ z^2 ] - z >= 1

Bounds
 -1 <= z <= 1
End
```

決定変数への代入

代入の結果、変数の値域制限に引っかかると実行不可能になる

```
[14]: sub = mod.substitute_variables(constants={'x': -1})
      print(sub.status)

Infeasible substitution for variable: x

QuadraticProgramStatus.INFEASIBLE
```

0 <= x <= 1 に違反

ひとつの変数に多段階の代入は行えない

```
[15]: from qiskit_optimization import QiskitOptimizationError
      try:
          sub = mod.substitute_variables(constants={'x': -1}, variables={'y': ('x', 1)})
      except QiskitOptimizationError as e:
          print('Error: {}'.format(e))

Error: 'Cannot substitute by variable that gets substituted itself: y <- x 1'
```

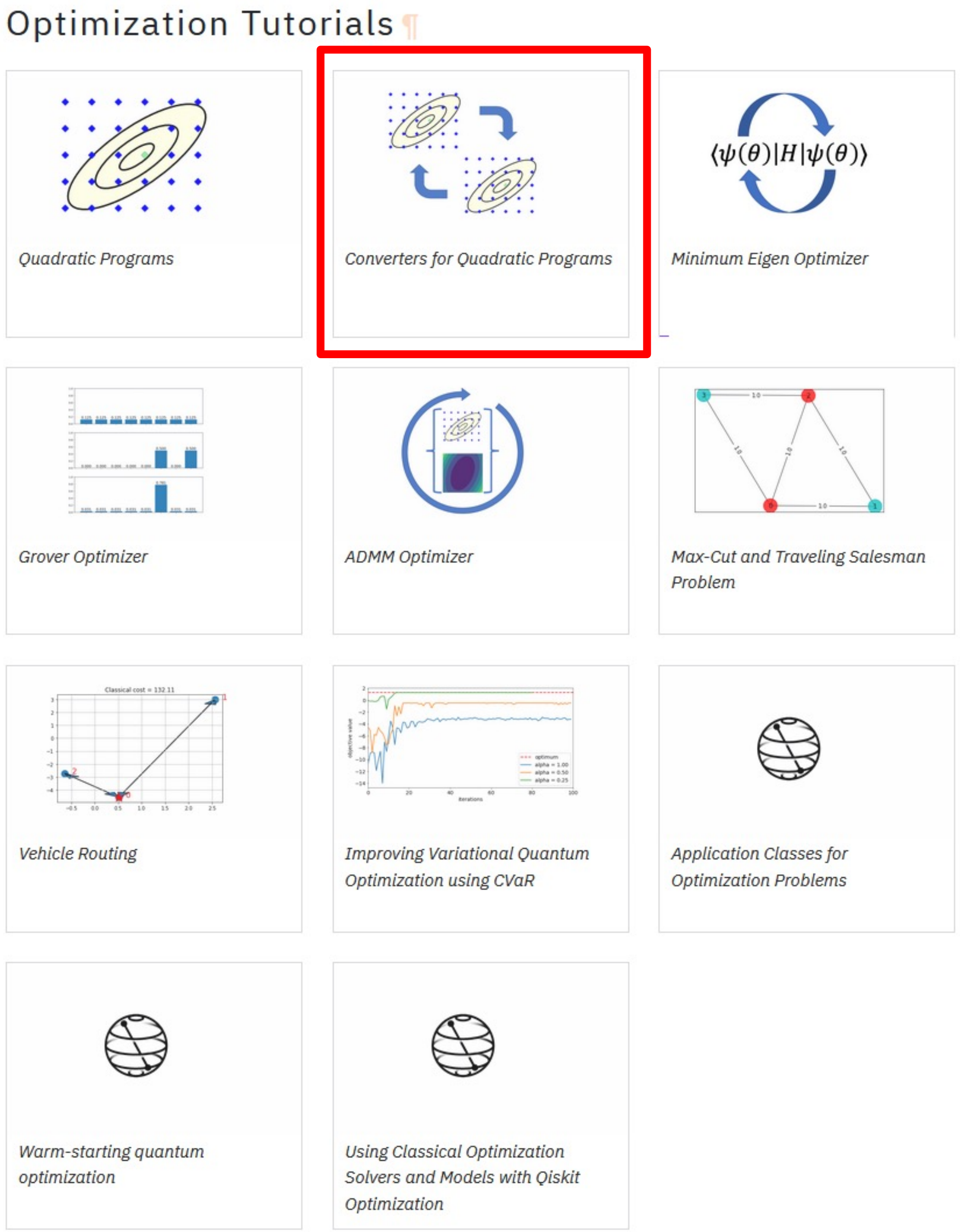
-1が代入されたxをさらに
yに代入するのはNG

Agenda

1. Quadratic Programs
   2次計画問題

2. Converters for Quadratic Programs
   量子アルゴリズムで解ける形への変換

3. Minimum Eigen Optimizer
   最小固有値ソルバーによる求解



Converters for Quadratic Programs

QuadraticProgram → QUBO → 解

Converters for Quadratic Programs
（この章）

2次計画問題を、QUBOの形に変換する
QUBO: Quadratic Unconstrained Binary Optimization

Quadratic	2次
Unconstrained	無制約
Binary	0-1変数
Optimization	

InequalityToEquality
　不等式制約→スラック変数付き等式制約
IntegerToBinary
　整数変数→係数付きバイナリ変数
LinearEqualityToPenalty
　等式制約→目的関数のペナルティ項
QuadraticProgramToQubo
　上の3つの変換処理のラッパー

Minimum Eigen Optimizer
（次の章）

QUBOの求解を、イジングモデルのハミルトニアンの基底状態を求めることに帰着させる
- バイナリ変数{0, 1}をスピン変数{-1, +1}に読み替え
- スピン変数は、パウリのZ行列で表現可能

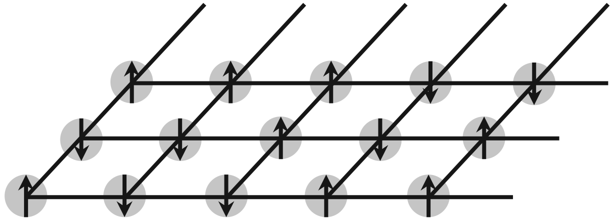
↓
最小固有値ソルバーで求解
- 基底ルーチンとして、VQE, QAOAなど

イジングモデル
- 磁性体を量子的な磁石スピンの集合体として表現したモデル
- 各スピンは上向きと下向きの重ね合わせ状態を取る
- スピンどうしの相互作用により安定な状態（基底状態）へ移行する

イジングモデルのハミルトニアン

$$H = \sum_i w_i Z_i + \sum_{i<j} w_{ij} Z_i Z_j$$

（Z_iはパウリZ演算子）



Converters for Quadratic Programs

QuadraticProgram → QUBO → 解

Converters for Quadratic Programs
（この章）

Minimum Eigen Optimizer
（次の章）

2次計画問題を、QUBOの形に変換する
QUBO: Quadratic Unconstrained Binary Optimization

Quadratic	2次
Unconstrained	無制約
Binary	0-1変数
Optimization	

InequalityToEquality
　不等式制約→スラック変数付き等式制約
IntegerToBinary
　整数変数→係数付きバイナリ変数
LinearEqualityToPenalty
　等式制約→目的関数のペナルティ項
QuadraticProgramToQubo
　上の3つの変換処理のラッパー

QUBOの求解を、イジングモデルのハミルトニアンの基底状態を求めることに帰着させる
- バイナリ変数{0, 1}をスピン変数{-1, +1}に読み替え
- スピン変数は、パウリのZ行列で表現可能

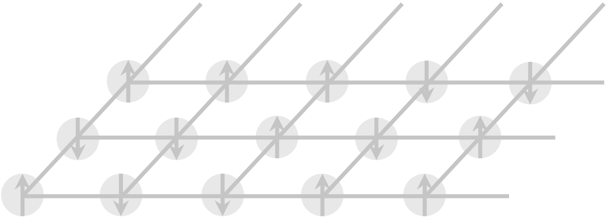
↓
最小固有値ソルバーで求解
- 基底ルーチンとして、VQE, QAOAなど

イジングモデル
- 磁性体を量子的な磁石スピンの集合体として表現したモデル
- 各スピンは上向きと下向きの重ね合わせ状態を取る
- スピンどうしの相互作用により安定な状態（基底状態）へ移行する

イジングモデルのハミルトニアン

$$H = \sum_i w_i Z_i + \sum_{i<j} w_{ij} Z_i Z_j$$

（Z_iはパウリZ演算子）



QuadraticProgramからQUBOへの変換

QuadraticProgramを量子アルゴリズムで解ける形にするために、QUBOへ変換する

2次整数計画問題

↓ InequalityToEquality
不等式制約→等式制約

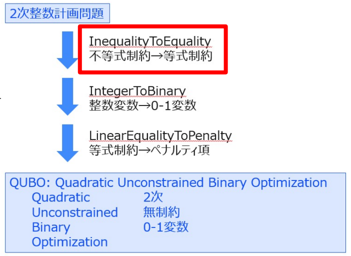
↓ IntegerToBinary
整数変数→0-1変数

↓ LinearEqualityToPenalty
等式制約→ペナルティ項

QUBO: Quadratic Unconstrained Binary Optimization

Quadratic	2次
Unconstrained	無制約
Binary	0-1変数
Optimization	

InequalityToEquality



不等式制約を、スラック変数付きの等式制約に変換

```
[3]: from qiskit_optimization.converters import InequalityToEquality
```

```
[4]: ineq2eq = InequalityToEquality()
     qp_eq = ineq2eq.convert(qp)
     print(qp_eq.export_as_lp_string())
```

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

Maximize
 obj: 2 x + y + z
Subject To
 xyz_leq: x + y + z <= 5.500000000000
 xyz_geq: x + y + z >= 2.500000000000

Bounds
 0 <= x <= 1
 0 <= y <= 1
       z <= 7

Binaries
 x y

Generals
 z
End
```

不等式制約

下界が省略されている場合は0（LPフォーマットの規約）

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

Maximize
 obj: 2 x + y + z
Subject To
 xyz_leq: x + y + z + xyz_leq@int_slack = 5
 xyz_geq: x + y + z - xyz_geq@int_slack = 3

Bounds
 0 <= x <= 1
 0 <= y <= 1
       z <= 7
       xyz_leq@int_slack <= 5
       xyz_geq@int_slack <= 6

Binaries
 x y

Generals
 z xyz_leq@int_slack xyz_geq@int_slack
End
```

スラック変数付きの**等式制約**
x, y, zはすべて整数なので右辺も整数化

スラック変数の値域はもとの変数の値域から自動定義

IntegerToBinary

2次整数計画問題
↓ InequalityToEquality 不等式制約→等式制約
↓ IntegerToBinary 整数変数→0-1変数
↓ LinearEqualityToPenalty 等式制約→ペナルティ項
QUBO: Quadratic Unconstrained Binary Optimization
Quadratic 2次
Unconstrained 無制約
Binary 0-1変数
Optimization

整数変数を、係数付きの0-1変数に変換

係数の決め方 S. Karimi, P. Ronagh, Practical Integer-to-Binary Mapping for Quantum Annealers （https://arxiv.org/pdf/1706.01945.pdf）の式(5)

整数変数 x

0-1変数 $y_i \ (0 \le i \le k, k = \lfloor \log N \rfloor)$

$$x \quad (0 \le x \le N) \Rightarrow \underbrace{2^0 y_0 + 2^1 y_1 + \cdots + 2^{k-1} y_{k-1}}_{} + \left(N - \sum_{i=0}^{k-1} 2^i\right) y_k$$

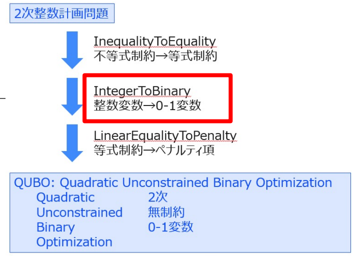
最後以外の係数は$2^0, 2^1, 2^2$ ...

最後の係数は残り

例：

$$x \quad (0 \le x \le 20) \Rightarrow 1y_0 + 2y_1 + 4y_2 + 8y_3 + 5y_4$$

IntegerToBinary



整数変数を、係数付きの0-1変数に変換

```
[6]: from qiskit_optimization.converters import IntegerToBinary
```

```
[7]: int2bin = IntegerToBinary()
     qp_eq_bin = int2bin.convert(qp_eq)
     print(qp_eq_bin.export_as_lp_string())
```

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

Maximize
 obj: 2 x + y + z
Subject To
 xyz_leq: x + y + z + xyz_leq@int_slack = 5
 xyz_geq: x + y + z - xyz_geq@int_slack = 3

Bounds
 0 <= x <= 1
 0 <= y <= 1
      z <= 7
      xyz_leq@int_slack <= 5
      xyz_geq@int_slack <= 6

Binaries
 x y

Generals
 z xyz_leq@int_slack xyz_geq@int_slack
End
```

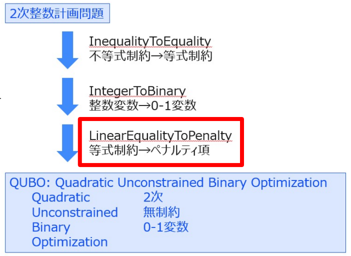
整数変数
0 <= z <= 7
0 <= a <= 5
0 <= b <= 6

```
Maximize
 obj: 2 x + y + z@0 + 2 z@1 + 4 z@2
Subject To
 xyz_leq: x + y + z@0 + 2 z@1 + 4 z@2 + xyz_leq@int_slack@0
          + 2 xyz_leq@int_slack@1 + 2 xyz_leq@int_slack@2 = 5
 xyz_geq: x + y + z@0 + 2 z@1 + 4 z@2 - xyz_geq@int_slack@0
          - 2 xyz_geq@int_slack@1 - 3 xyz_geq@int_slack@2 = 3

Bounds
 0 <= x <= 1
 0 <= y <= 1
 0 <= z@0 <= 1
 0 <= z@1 <= 1
 0 <= z@2 <= 1
 0 <= xyz_leq@int_slack@0 <= 1
 0 <= xyz_leq@int_slack@1 <= 1
 0 <= xyz_leq@int_slack@2 <= 1
 0 <= xyz_geq@int_slack@0 <= 1
 0 <= xyz_geq@int_slack@1 <= 1
 0 <= xyz_geq@int_slack@2 <= 1

Binaries
 x y z@0 z@1 z@2 xyz_leq@int_slack@0 xyz_leq@int_slack@1 xyz_leq@int_slack@2
 xyz_geq@int_slack@0 xyz_geq@int_slack@1 xyz_geq@int_slack@2
End
```

zの係数：{ 1, 2, 4 }
aの係数：{ 1, 2, 2 }
bの係数：{ 1, 2, 3 }

0-1変数
0 <= z0, z1, z2 <= 1
0 <= a0, a1, a2 <= 1
0 <= b0, b1, b2 <= 1

LinearEqualityToPenalty



線形制約を、目的関数の2次ペナルティ項に変換

線形制約

$$\sum_i a_i x_i = b$$

目的関数のペナルティ項

$$obj = \ldots \pm M\left(b - \sum_i a_i x_i\right)^2$$

Mはペナルティ係数（デフォルトは10^5）
最小化なら+、最大化なら−

これにより、線形制約のみのモデルは、無制約モデルになる

LinearEqualityToPenalty

2次整数計画問題

↓ InequalityToEquality
不等式制約→等式制約

↓ IntegerToBinary
整数変数→0-1変数

↓ LinearEqualityToPenalty
等式制約→ペナルティ項

QUBO: Quadratic Unconstrained Binary Optimization
Quadratic 2次
Unconstrained 無制約
Binary 0-1変数
Optimization

線形制約を、目的関数の2次ペナルティ項に変換

```
[9]: from qiskit_optimization.converters import LinearEqualityToPenalty
```

```
[10]: lineq2penalty = LinearEqualityToPenalty()
      qubo = lineq2penalty.convert(qp_eq_bin)
      print(qubo.export_as_lp_string())
```

線形制約

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

Maximize
 obj: 2 x + y + z@0 + 2 z@1 + 4 z@2
Subject To
 xyz_leq: x + y + z@0 + 2 z@1 + 4 z@2 + xyz_leq@int_slack@0
          + 2 xyz_leq@int_slack@1 + 2 xyz_leq@int_slack@2 = 5
 xyz_geq: x + y + z@0 + 2 z@1 + 4 z@2 - xyz_geq@int_slack@0
          - 2 xyz_geq@int_slack@1 - 3 xyz_geq@int_slack@2 = 3

Bounds
 0 <= x <= 1
 0 <= y <= 1
 0 <= z@0 <= 1
 0 <= z@1 <= 1
 0 <= z@2 <= 1
 0 <= xyz_leq@int_slack@0 <= 1
 0 <= xyz_leq@int_slack@1 <= 1
 0 <= xyz_leq@int_slack@2 <= 1
 0 <= xyz_geq@int_slack@0 <= 1
 0 <= xyz_geq@int_slack@1 <= 1
 0 <= xyz_geq@int_slack@2 <= 1

Binaries
 x y z@0 z@1 z@2 xyz_leq@int_slack@0 xyz_leq@int_slack@1 xyz_leq@int_slack@2
 xyz_geq@int_slack@0 xyz_geq@int_slack@1 xyz_geq@int_slack@2
End
```

目的関数の**ペナルティ項**

```
Maximize
 obj: 178 x + 177 y + 177 z@0 + 354 z@1 + 708 z@2 + 110 xyz_leq@int_slack@0
      + 220 xyz_leq@int_slack@1 + 220 xyz_leq@int_slack@2
      - 66 xyz_geq@int_slack@0 - 132 xyz_geq@int_slack@1
      - 198 xyz_geq@int_slack@2 + [ - 44 x^2 - 88 x*y - 88 x*z@0 - 176 x*z@1
      - 352 x*z@2 - 44 x*xyz_leq@int_slack@0 - 88 x*xyz_leq@int_slack@1
      - 88 x*xyz_leq@int_slack@2 + 44 x*xyz_geq@int_slack@0
      + 88 x*xyz_geq@int_slack@1 + 132 x*xyz_geq@int_slack@2 - 44 y^2 - 88 y*z@0
      - 176 y*z@1 - 352 y*z@2 - 44 y*xyz_leq@int_slack@0
      - 88 y*xyz_leq@int_slack@1 - 88 y*xyz_leq@int_slack@2
      + 44 y*xyz_geq@int_slack@0 + 88 y*xyz_geq@int_slack@1
      + 132 y*xyz_geq@int_slack@2 - 44 z@0^2 - 176 z@0*z@1 - 352 z@0*z@2
      - 44 z@0*xyz_leq@int_slack@0 - 88 z@0*xyz_leq@int_slack@1
      - 88 z@0*xyz_leq@int_slack@2 + 44 z@0*xyz_geq@int_slack@0
      + 88 z@0*xyz_geq@int_slack@1 + 132 z@0*xyz_geq@int_slack@2 - 176 z@1^2
      - 704 z@1*z@2 - 88 z@1*xyz_leq@int_slack@0 - 176 z@1*xyz_leq@int_slack@1
      - 176 z@1*xyz_leq@int_slack@2 + 88 z@1*xyz_geq@int_slack@0
      + 176 z@1*xyz_geq@int_slack@1 + 264 z@1*xyz_geq@int_slack@2 - 704 z@2^2
      - 176 z@2*xyz_leq@int_slack@0 - 352 z@2*xyz_leq@int_slack@1
      - 352 z@2*xyz_leq@int_slack@2 + 176 z@2*xyz_geq@int_slack@0
      + 352 z@2*xyz_geq@int_slack@1 + 528 z@2*xyz_geq@int_slack@2
      - 22 xyz_leq@int_slack@0^2 - 88 xyz_leq@int_slack@0*xyz_leq@int_slack@1
      - 88 xyz_leq@int_slack@0*xyz_leq@int_slack@2 - 88 xyz_leq@int_slack@1^2
      - 176 xyz_leq@int_slack@1*xyz_leq@int_slack@2 - 88 xyz_leq@int_slack@2^2
      - 22 xyz_geq@int_slack@0^2 - 88 xyz_geq@int_slack@0*xyz_geq@int_slack@1
      - 132 xyz_geq@int_slack@0*xyz_geq@int_slack@2 - 88 xyz_geq@int_slack@1^2
      - 264 xyz_geq@int_slack@1*xyz_geq@int_slack@2 - 198 xyz_geq@int_slack@2^2
      ]/2 -374
Subject To

Bounds
 0 <= x <= 1
 0 <= y <= 1
 0 <= z@0 <= 1
 0 <= z@1 <= 1
 0 <= z@2 <= 1
 0 <= xyz_leq@int_slack@0 <= 1
 0 <= xyz_leq@int_slack@1 <= 1
 0 <= xyz_leq@int_slack@2 <= 1
 0 <= xyz_geq@int_slack@0 <= 1
 0 <= xyz_geq@int_slack@1 <= 1
 0 <= xyz_geq@int_slack@2 <= 1

Binaries
 x y z@0 z@1 z@2 xyz_leq@int_slack@0 xyz_leq@int_slack@1 xyz_leq@int_slack@2
 xyz_geq@int_slack@0 xyz_geq@int_slack@1 xyz_geq@int_slack@2
End
```

QuadraticProgramToQubo

QuadraticProgramを量子アルゴリズムで解ける形にするために、QUBOへ変換する

2次整数計画問題

InequalityToEquality
不等式制約→等式制約

IntegerToBinary
整数変数→0-1変数

LinearEqualityToPenalty
等式制約→ペナルティ項

ラッパーを提供
QuadraticProgramToQubo

QUBO: Quadratic Unconstrained Binary Optimization

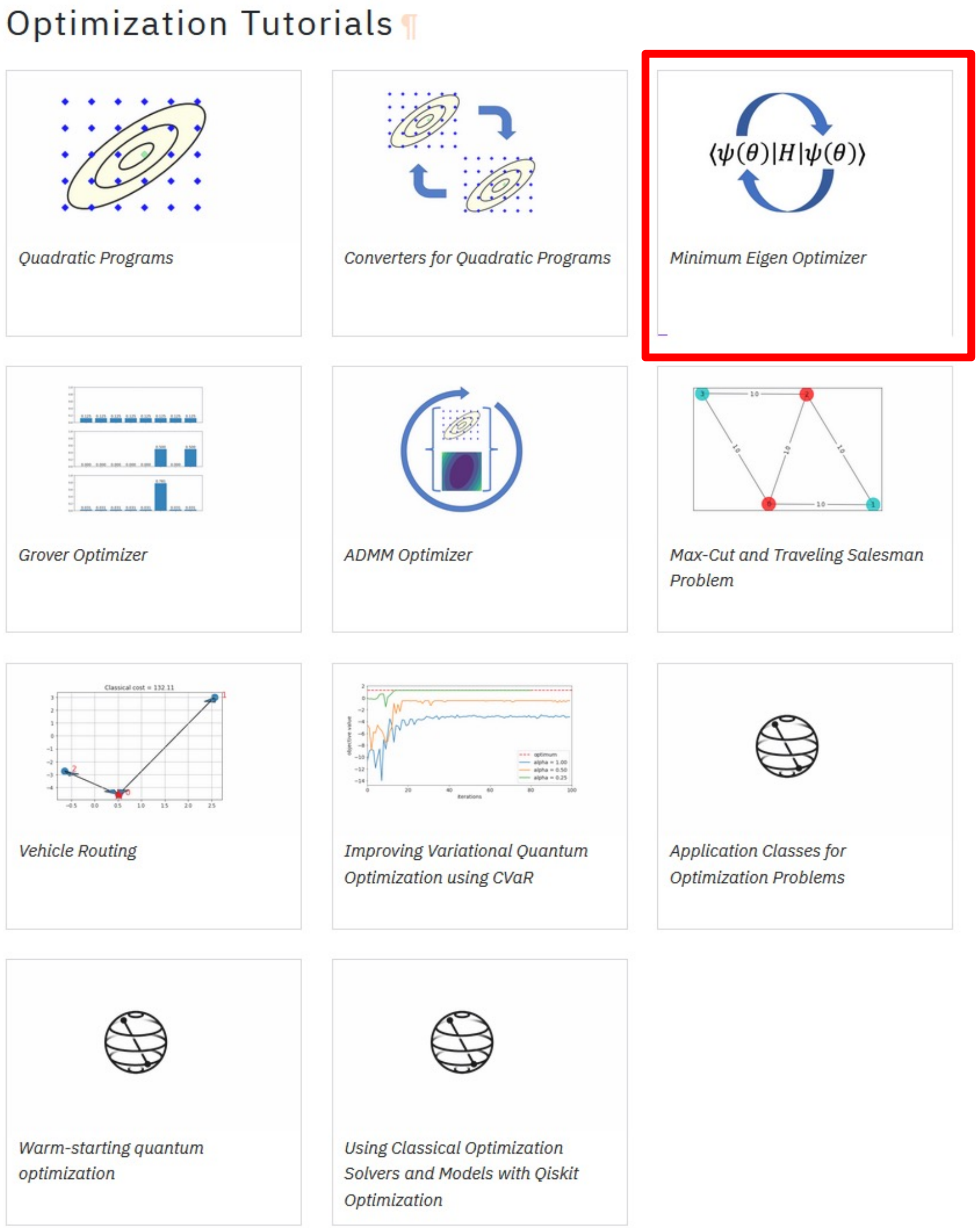
Quadratic	2次
Unconstrained	無制約
Binary	0-1変数
Optimization	

Agenda

1. Quadratic Programs
   2次計画問題

2. Converters for Quadratic Programs
   量子アルゴリズムで解ける形への変換

3. Minimum Eigen Optimizer
   最小固有値ソルバーによる求解



Converters for Quadratic Programs

QuadraticProgram → QUBO → 解

Converters for Quadratic Programs
（前の章）

Minimum Eigen Optimizer
（この章）

2次計画問題を、QUBOの形に変換する
QUBO: Quadratic Unconstrained Binary Optimization

Quadratic	2次
Unconstrained	無制約
Binary	0-1変数
Optimization	

InequalityToEquality
　不等式制約→スラック変数付き等式制約
IntegerToBinary
　整数変数→係数付きバイナリ変数
LinearEqualityToPenalty
　等式制約→目的関数のペナルティ項
QuadraticProgramToQubo
　上の3つの変換処理のラッパー

QUBOの求解を、イジングモデルのハミルトニアンの基底状態を求めることに帰着させる

- バイナリ変数{0, 1}をスピン変数{-1, +1}に読み替え
- スピン変数は、パウリのZ行列で表現可能

↓

最小固有値ソルバーで求解

- 基底ルーチンとして、VQE, QAOAなど

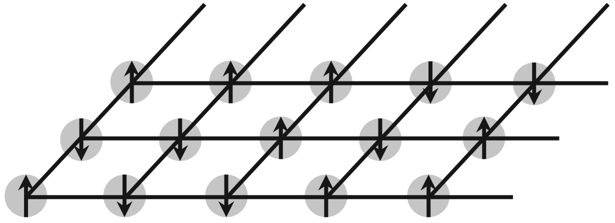
イジングモデル

- 磁性体を量子的な磁石スピンの集合体として表現したモデル
- 各スピンは上向きと下向きの重ね合わせ状態を取る
- スピンどうしの相互作用により安定な状態（基底状態）へ移行する

イジングモデルのハミルトニアン

$$H = \sum_i w_i Z_i + \sum_{i<j} w_{ij} Z_i Z_j$$

（Z_iはパウリZ演算子）



QUBOをMinimumEigenOptimizerで解く

解きたい問題をQUBOとして準備する

$$\text{minimize}(x - 2y + 3z + xy - xz + 2yz) \quad x, y, z \in \{0, 1\}$$

[1]:
```
from qiskit import BasicAer
from qiskit.utils import algorithm_globals, QuantumInstance
from qiskit.algorithms import QAOA, NumPyMinimumEigensolver
from qiskit_optimization.algorithms import MinimumEigenOptimizer,
RecursiveMinimumEigenOptimizer, SolutionSample, OptimizationResultStatus
from qiskit_optimization import QuadraticProgram
from qiskit.visualization import plot_histogram
from typing import List, Tuple
import numpy as np
```

[2]:
```
# create a QUBO
qubo = QuadraticProgram()
qubo.binary_var('x')
qubo.binary_var('y')
qubo.binary_var('z')
qubo.minimize(linear=[1,-2,3], quadratic={('x', 'y'): 1, ('x', 'z'): -1, ('y', 'z'): 2})
print(qubo.export_as_lp_string())
```

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

Minimize
 obj: x - 2 y + 3 z + [ 2 x*y - 2 x*z + 4 y*z ]/2
Subject To

Bounds
 0 <= x <= 1
 0 <= y <= 1
 0 <= z <= 1

Binaries
 x y z
End
```

QUBOをMinimumEigenOptimizerで解く

使用できるアルゴリズム：VQE, QAOA, NumPyMinimumEigensolver

古典的厳密求解

```
[5]: algorithm_globals.random_seed = 10598
     quantum_instance = QuantumInstance(BasicAer.get_backend('statevector_simulator'),
                                        seed_simulator=algorithm_globals.random_seed,
                                        seed_transpiler=algorithm_globals.random_seed)
     qaoa_mes = QAOA(quantum_instance=quantum_instance, initial_point=[0., 0.])
     exact_mes = NumPyMinimumEigensolver()
```

使用するアルゴリズムの実体

```
[6]: qaoa = MinimumEigenOptimizer(qaoa_mes)   # using QAOA
     exact = MinimumEigenOptimizer(exact_mes)  # using the exact classical numpy minimum eigen
     solver
```

MinimumEigenOptimizerはI/F

```
[7]: exact_result = exact.solve(qubo)
     print(exact_result)

optimal function value: -2.0
optimal value: [0. 1. 0.]
status: SUCCESS
```

solveで求解

```
[8]: qaoa_result = qaoa.solve(qubo)
     print(qaoa_result)

optimal function value: -2.0
optimal value: [0. 1. 0.]
status: SUCCESS
```

ソルバーを変えても同じ答えが求まっている

QUBOとIsing operatorの相互変換

QUBO → Ising operator

```
[3]: op, offset = qubo.to_ising()
     print('offset: {}'.format(offset))
     print('operator:')
     print(op)
```

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

Minimize
 obj: x - 2 y + 3 z + [ 2 x*y - 2 x*z + 4 y*z ]/2
Subject To

Bounds
 0 <= x <= 1
 0 <= y <= 1
 0 <= z <= 1

Binaries
 x y z
End
```

```
offset: 1.5
operator:
-1.75 * ZII
+ 0.25 * IZI
+ 0.5 * ZZI
- 0.5 * IIZ
- 0.25 * ZIZ
+ 0.25 * IZZ
```

offset: 目的関数の定数項
operator: イジングオペレータ

Operatorとオフセット（目的関数の定数項）を返す

QUBOとIsing operatorの相互変換

QUBO ← Ising operator

```
[4]: qp=QuadraticProgram()
     qp.from_ising(op, offset, linear=True)
     print(qp.export_as_lp_string())
```

```
offset: 1.5
operator:
-1.75 * ZII
+ 0.25 * IZI
+ 0.5 * ZZI
- 0.5 * IIZ
- 0.25 * ZIZ
+ 0.25 * IZZ
```

→

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

Minimize
 obj: x0 - 2 x1 + 3 x2 + [ 2 x0*x1 - 2 x0*x2 + 4 x1*x2 ]/2
Subject To

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

Binaries
 x0 x1 x2
End
```

Ising Hamiltonian表現に基づかないアルゴリズム（GroverOptimizerなど）で解くことも可能

再帰的QAOA

解こうとする問題のサイズが大きくなると、QAOAの回路深さも大きくすることが必要になる
→短期的には量子デバイスで扱えないサイズになる
→回避策：再帰的(Recursive)QAOA

S. Bravyi, A. Kliesch, R. Koenig, E. Tang, Obstacles to State Preparation and Variational Optimization from Symmetry Protection, arXiv preprint arXiv:1910.08980 (2019).
https://www.youtube.com/watch?v=Nzi8H0m0y8k

Qiskitでは、RecursiveMinumEigenOptimizerクラスとして提供

- MinimumEigenOptimizerクラスを入力として取る
- 変数を1つずつ減らしながら再帰的に最適化を実行
- 変数の数が指定された閾値になったら、所定のソルバーで直接求解

```
[17]: rqaoa = RecursiveMinimumEigenOptimizer(qaoa, min_num_vars=1,
      min_num_vars_optimizer=exact)
```

min_num_vars
　変数をいくつまで減らしたら通常の求解に移行するか
min_num_vars_optimizer
　通常求解に使用するソルバー

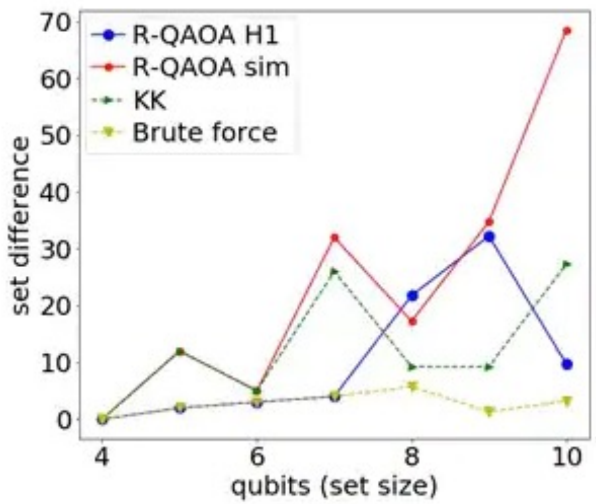
```
[18]: rqaoa_result = rqaoa.solve(qubo)
      print(rqaoa_result)
```

```
optimal function value: -2.0
optimal value: [0. 1. 0.]
status: SUCCESS
```

再帰的QAOAの応用例

R-QAOAの応用例：BMW

- 分割問題(number partitioning)にR-QAOAを適用
  - 整数の集合を、合計差ができるだけ小さくなるように2つの集合に分割
- 深さ1のR-QAOAで、古典アルゴリズム（Karmarkar-Karp法）に匹敵
  - 深さを増やせば、古典を凌駕すると期待



News / How BMW Can Maximize its Supply Chain Efficiency

How BMW Can Maximize Its Supply Chain Efficiency with Quantum

The luxury car company is getting the ultimate operational upgrade — powered by Honeywell's quantum computer.

https://www.honeywell.com/us/en/news/2021/01/exploring-supply-chain-solutions-with-quantum-computing

分割問題を解いてみる

分割問題 (number partitioning)

整数の集合を、合計差ができるだけ小さくなるように2つの集合に分割する（NP完全）

- { 3, 8, 5 } → { 3, 5 }, { 8 }
- { 1, 2, 4, 8 } → { 1, 2, 4 }, { 8 }

合計差ができるだけ小さくなるようにする
⇔ 合計ができるだけ同じ値になるようにする
⇔ （2つに分けた集合の一方の合計） – （もう一方の合計） をゼロに近くする

を目的関数とした最小化
と考えることができる

定式化

A, B: 整数の集合$\{ a_1, a_2, \dots, a_n \}$を分割した2つの集合

決定変数x_i $(i = 1, 2, \dots, n)$: $\begin{cases} x_i = 1 & (i\text{番目の整数が}A\text{に属する}) \\ x_i = -1 & (i\text{番目の整数が}B\text{に属する}) \end{cases}$

決定変数が+-1
目的関数が2次
制約条件がない

→イジングモデル

$$\text{minimize}(a_1 x_1 + a_2 x_2 + \cdots + a_n x_n)^2 \ \ (x_i \in \{1, -1\})$$

分割問題を解いてみる

例：5個の整数の場合

$a_i \in \mathbb{Z}$
$x_i \in \{ 1, -1 \}$

$$(a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5)^2$$

$$= {a_1}^2{x_1}^2 + {a_2}^2{x_2}^2 + {a_3}^2{x_3}^2 + {a_4}^2{x_4}^2 + {a_5}^2{x_5}^2 + 2a_1a_2x_1x_2 + 2a_1a_3x_1x_3 + \cdots + 2a_4a_5x_4x_5$$

$x_i = \pm 1$ なので、
${x_i}^2 = 1$

分割問題を解いてみる

例：5個の整数の場合

$a_i \in \mathbb{Z}$
$x_i \in \{\,1, -1\,\}$

$$(a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5)^2$$

$$= a_1{}^2x_1{}^2 + a_2{}^2x_2{}^2 + a_3{}^2x_3{}^2 + a_4{}^2x_4{}^2 + a_5{}^2x_5{}^2 + 2a_1a_2x_1x_2 + 2a_1a_3x_1x_3 + \cdots + 2a_4a_5x_4x_5$$

$x_i = \pm 1$ なので、
$x_i{}^2 = 1$

$$= \underbrace{(a_1{}^2 + a_2{}^2 + a_3{}^2 + a_4{}^2 + a_5{}^2)}_{\text{定数}} + \underbrace{X^T \begin{pmatrix} 0 & 2a_1a_2 & 2a_1a_3 & 2a_1a_4 & 2a_1a_5 \\ 0 & 0 & 2a_2a_3 & 2a_2a_4 & 2a_2a_5 \\ 0 & 0 & 0 & 2a_3a_4 & 2a_3a_5 \\ 0 & 0 & 0 & 0 & 2a_4a_5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} X}_{\text{イジングオペレータ}}$$

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$$

イジングモデルのハミルトニアン

$$H = \sum_i w_i Z_i + \sum_{i<j} w_{ij} Z_i Z_j$$

（Z_iはパウリZ演算子）

分割問題を解いてみる

$$\underbrace{({a_1}^2+{a_2}^2+{a_3}^2+{a_4}^2+{a_5}^2)}_{\text{定数}}+\underbrace{X^T\begin{pmatrix}0 & 2a_1a_2 & 2a_1a_3 & 2a_1a_4 & 2a_1a_5\\ 0 & 0 & 2a_2a_3 & 2a_2a_4 & 2a_2a_5\\ 0 & 0 & 0 & 2a_3a_4 & 2a_3a_5\\ 0 & 0 & 0 & 0 & 2a_4a_5\\ 0 & 0 & 0 & 0 & 0\end{pmatrix}X}_{\text{イジングオペレータ}}$$

コード：

```
from qiskit import BasicAer
from qiskit_optimization import QuadraticProgram
from qiskit.opflow import I, Z
from qiskit.utils import QuantumInstance
from qiskit.algorithms import QAOA
from qiskit_optimization.algorithms import MinimumEigenOptimizer

def solve_number_partitioning(a1, a2, a3, a4, a5):
    op = 2 * a1 * a2 * (I ^ I ^ I ^ Z ^ Z) + \
         2 * a1 * a3 * (I ^ I ^ Z ^ I ^ Z) + \
         2 * a1 * a4 * (I ^ Z ^ I ^ I ^ Z) + \
         2 * a1 * a5 * (Z ^ I ^ I ^ I ^ Z) + \
         2 * a2 * a3 * (I ^ I ^ Z ^ Z ^ I) + \
         2 * a2 * a4 * (I ^ Z ^ I ^ Z ^ I) + \
         2 * a2 * a5 * (Z ^ I ^ I ^ Z ^ I) + \
         2 * a3 * a4 * (I ^ Z ^ Z ^ I ^ I) + \
         2 * a3 * a5 * (Z ^ I ^ Z ^ I ^ I) + \
         2 * a4 * a5 * (Z ^ Z ^ I ^ I ^ I)
    offset = a1 * a1 + a2 * a2 + a3 * a3 + a4 * a4 + a5 * a5
    print(op)

    qp = QuadraticProgram()
    qp.from_ising(op, offset)
    print(qp)

    quantum_instance = QuantumInstance(BasicAer.get_backend('statevector_simulator'))
    qaoa_mes = QAOA(quantum_instance=quantum_instance, initial_point=[0., 0.])
    qaoa = MinimumEigenOptimizer(qaoa_mes)    # using QAOA
    qaoa_result = qaoa.solve(qp)
    print(qaoa_result)

solve_number_partitioning(10, 3, 7, 8, 2)
solve_number_partitioning(10, 15, 20, 25, 30)
```

右から
x_1, x_2, x_3, x_4, x_5
に対応

[10, 3, 7, 8, 2] のとき

```
60.0 * IIIZZ
+ 140.0 * IIZIZ
+ 160.0 * IZIIZ
+ 40.0 * ZIIIZ
+ 42.0 * IIZZI
+ 48.0 * IZIZI
+ 12.0 * ZIIZI
+ 112.0 * IZZII
+ 28.0 * ZIZII
+ 32.0 * ZZIII
```

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

Minimize
 obj: [ - 1600 x0^2 + 480 x0*x1 + 1120 x0*x2 + 1280 x0*x3 + 320 x0*x4 - 648 x1^2
      + 336 x1*x2 + 384 x1*x3 + 96 x1*x4 - 1288 x2^2 + 896 x2*x3 + 224 x2*x4
      - 1408 x3^2 + 256 x3*x4 - 448 x4^2 ]/2 + 900
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
```

```
optimal function value: 0.0
optimal value: [0. 0. 1. 1. 0.]
status: SUCCESS
```

[0, 0, 1, 1, 0] → (10, 3, 7, 8, 2)
[1, 1, 0, 1, 0] → (10, 15, 20, 25, 30)

分割問題を解いてみる

Qiskitには分割問題用のクラスが用意されている

NumberPartition ¶

CLASS **NumberPartition**(*number_set*) [SOURCE] ¶

Bases: `qiskit_optimization.applications.optimization_application.OptimizationApplication`

Optimization application for the “number partition” [1] problem.

References

[1]: “Partition problem”, https://en.wikipedia.org/wiki/Partition_problem

Parameters

number_set (`List`[`int`]) – A list of integers

分割したい整数の集合

Methods

`interpret`(result)	Interpret a result as a list of subsets
`to_quadratic_program`()	Convert a number partitioning problem instance into a `QuadraticProgram`

結果の変数値を
分割集合に翻訳

2次計画問題への
変換

分割問題を解いてみる

コード：

```python
from qiskit import BasicAer
from qiskit_optimization.applications import NumberPartition
from qiskit.utils import QuantumInstance
from qiskit.algorithms import QAOA
from qiskit_optimization.algorithms import MinimumEigenOptimizer
from qiskit_optimization.converters import QuadraticProgramToQubo

def solve_number_partitioning(values):
    pb = NumberPartition(values)
    qp = pb.to_quadratic_program()
    print(qp)

    conv = QuadraticProgramToQubo()
    qubo = conv.convert(qp)
    op, offset = qubo.to_ising()
    print('offset: {}'.format(offset))
    print('operator:')
    print(op)
    print()

    quantum_instance = QuantumInstance(BasicAer.get_backend('statevector_simulator'))
    qaoa_mes = QAOA(quantum_instance=quantum_instance, initial_point=[0., 0.])
    qaoa = MinimumEigenOptimizer(qaoa_mes)    # using QAOA
    qaoa_result = qaoa.solve(qubo)
    print(qaoa_result)
    print()

    res = pb.interpret(qaoa_result)
    print(res)
    print()

solve_number_partitioning([10, 3, 7, 8, 2])
solve_number_partitioning([10, 15, 20, 25, 30])
```

[10, 3, 7, 8, 2] のとき

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

Minimize
 obj:
Subject To
 c0: - 20 x_0 - 6 x_1 - 14 x_2 - 16 x_3 - 4 x_4 = -30

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1
 0 <= x_4 <= 1

Binaries
 x_0 x_1 x_2 x_3 x_4
End
```

```
offset: 226.0
operator:
32.0 * ZZIII
+ 28.0 * ZIZII
+ 112.0 * IZZII
+ 12.0 * ZIIZI
+ 48.0 * IZIZI
+ 42.0 * IIZZI
+ 40.0 * ZIIIZ
+ 160.0 * IZIIZ
+ 140.0 * IIZIZ
+ 60.0 * IIIZZ
```

```
optimal function value: 0.0
optimal value: [0. 0. 1. 1. 0.]
status: SUCCESS
```

```
[[10, 3, 2], [7, 8]]
```

[[10, 3, 2], [7, 8]]

[[20, 30], [10, 15, 25]]

分割問題を解いてみる

qiskit_optimization.applicationsモジュールでは、NumberPartition以外にも個別問題用のクラスを提供

`Clique`	クリーク問題
`ExactCover`	集合分割問題
`GraphPartition`	グラフ分割問題
`Knapsack`	ナップサック問題
`Maxcut`	最大カット問題
`NumberPartition`	整数分割問題
`SetPacking`	集合パッキング問題
`StableSet`	安定集合問題
`Tsp`	巡回セールスマン問題
`VehicleRouting`	配車計画問題
`VertexCover`	頂点被覆問題

Optimizationまとめ

- 組み合わせ最適化問題は、QUBOに変換可能
  - Qiskitで自動変換機能を提供

- QUBOの解はイジングハミルトニアンの基底状態と等価
  - QAOA, VQE, 古典ソルバーなどの求解アルゴリズムを提供
  - 再帰的QAOAにも対応
  - 最小固有値ソルバー以外のアルゴリズムも提供

- Qiskitでは、典型的な個別問題クラスも提供
  - NumberPartition, Maxcut, Tsp, etc

ご清聴ありがとうございました。

Shun Shirakawa