```
# 安裝 D-Wave 量子退火套件與 pyqubo
!pip install dwave-ocean-sdk
!pip install pyqubo
```

顯示隱藏的輸出內容

```
"""
引入 Pyqubo 中的 Binary 變數類別
更多資料：https://pyqubo.readthedocs.io/en/latest/refe
"""
from pyqubo import Binary
```

QUBO公式簡介

QUBO代表「二次無約束二元最佳化(Quadratic Unconstrained Binary Optimization)」[1]。QUBO公式是表達最佳化問題的的一種標準形式（canonical form），可以透過調整一組二元變數(binary variable)來最小化一個二次(quadratic)目標函數(objective function)。QUBO公式主要形式如下:

$$\min_{x\in\{0,1\}^n} f(x) = \sum_i Q_{ii}x_i + \sum_{i<j} Q_{ij}x_ix_j + c.$$

上式中x_i 為二元決策變數（其值為0 或 1），或是等義的，$x = (x_1, x_2, \ldots, x_n)^\top \in \{0,1\}^n$ 是二元決策變數向量，Q_{ii}是一次項係數，Q_{ij} 是二次項係數，而c是常數。

由於x_i的值為0 或 1，因此$x_i = x_i^2$，所以QUBO公式也可以寫成全部為二次項的形式:

$$\min_{x\in\{0,1\}^n} f(x) = \sum_i Q_{ii}x_i^2 + \sum_{i<j} Q_{ij}x_ix_j + c.$$

或是QUBO公式也可以寫成以下的矩陣形式:

$$\min_{x\in\{0,1\}^n} f(x) = x^\top Qx + c,$$

其中， $x = (x_1, x_2, \ldots, x_n)^\top \in \{0,1\}^n$ 是二元決策向量；Q 是一個大小為$n \times n$的上三角係數矩陣，也就是$Q_{ij} = 0$針對$i > j$，而c是常數項。

以下更詳細的展開，令

$$Q = \begin{bmatrix} Q_{11} & Q_{12} & \cdots & Q_{1n} \\ Q_{21} & Q_{22} & \cdots & Q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{n1} & Q_{n2} & \cdots & Q_{nn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},$$

則有

$$x^\top Q x \;=\; \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij} x_i x_j.$$

我們也可以將Q改為對稱矩陣（即 $Q_{ij} = Q_{ji}$），則需要常將原本的上三角矩陣係數平均分配一半到下三角矩陣係數中。具體定義為：

$$Q_{ij}^{\text{sym}} = \begin{cases} Q_{ii}, & i = j, \\ \frac{1}{2} Q_{ij}, & i \neq j, \end{cases}$$

此時可保證

$$x^\top Q^{\text{sym}} x \;=\; \sum_{i} Q_{ii} x_i \;+\; \sum_{i<j} Q_{ij} x_i x_j.$$

在實作時，常透過對稱化處理將 Q 視為對稱矩陣。

總的來說，QUBO公式具有「通用容器」特性：許多組合最佳化問題(Combinatorial Optimization Problems, COPs)，例如最大割問題(Max-Cut Problem, MCP)、旅行推銷員問題(Traveling Salesperson Problem, TSP)、圖著色問題(Graph Coloring Problem, GCP)、二次指派問題(Quadratic Assignment Problem, QAP)、0/1背包問題(0/1 Knapsack Problem, 0/1 KP)以及子集加總問題 (Subset Sum Problem, SSP)等，皆可以使用QUBO公式建模。稍後我們會說明，除了COP本身的最佳化目標之外，若COP帶有約束條件(constraint condition)，則約束條件會以懲罰項嵌入目標函數中。

易辛(Ising)模型簡介

易辛(Ising)模型[2]最初由德國物理學家Ernst Ising於1925年提出，作為一個研究鐵磁性（ferromagnetism）的理論模型，用來描述鐵磁材料中自旋系統的總能量（Hamiltonian）。伊辛模型使用自旋變數 $s_i \in \{-1, +1\}$，其能量常寫作

$$E(s) \;=\; \sum_i h_i s_i \;+\; \sum_{i<j} J_{ij} s_i s_j \;+\; \text{const},$$

其中 h_i 為外場（bias），J_{ij} 為耦合（coupler）。

QUBO公式與Ising模型應用

QUBO公式與Ising模型可以精確的互相轉換，說明如下:

令

$$x_i \;=\; \frac{1 + s_i}{2} \quad (\, x_i \in \{0, 1\},\, s_i \in \{-1, +1\}\,)$$

代入QUBO公式可得Ising參數:

$$J_{ij} = \frac{Q_{ij}}{4} \quad (i < j),$$

$$h_i = \frac{Q_{ii}}{2} \;+\; \frac{1}{4} \sum_{j \neq i} Q_{ij},$$

$$\text{const} = c \;+\; \frac{1}{2} \sum_i Q_{ii} \;+\; \frac{1}{4} \sum_{i<j} Q_{ij}.$$

反向亦成立：若已知 Ising (h, J)，令 $s_i = 2x_i - 1$ 並整理，

$$Q_{ij} = 4J_{ij} \quad (i < j),$$

$$Q_{ii} = 2h_i \;-\; 2\sum_{j \neq i} J_{ij},$$

$$c = \text{const} \;+\; \sum_i h_i \;+\; \sum_{i<j} J_{ij}.$$

此對應確保QUBO公式能對接Ising模型，而Ising模型也能轉成QUBO公式。此形式正是量子退火機(quantum annealer)（如 D-Wave公司的Advantage [3]）、量子啟發退火機(quantum-inspired annealer)（如日本Fujitsu公司的Digital Annealer [4]與日本Hitachi公司的FPGA/CMOS退火機 [5]）、光學相干伊辛機(Coherent Ising Machine, CIM)(如史丹佛大學與日本NTT研究團隊提出的CIM [6])與許多古典啟發

式最佳化演算法(如模擬退火(Simulated Annealing, SA)演算法[7]以及禁忌搜尋(Tabu Search)演算法[8]）所處理的標的。

[1] Glover, F., Kochenberger, G., & Du, Y. (2018). A tutorial on formulating and using QUBO models. arXiv preprint arXiv:1811.11538.

[2] Ising, E. (1925). Beitrag zur theorie des ferromagnetismus. Zeitschrift für Physik, 31(1), 253-258.

[3] Johnson, M. W., Amin, M. H., Gildert, S., Lanting, T., Hamze, F., Dickson, N., ... & Rose, G. (2011). Quantum annealing with manufactured spins. Nature, 473(7346), 194-198.

[4] Aramon, M., Rosenberg, G., Valiante, E., Miyazawa, T., Tamura, H., & Katzgraber, H. G. (2019). Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. Frontiers in Physics, 7, 48.

[5] Inagaki, T., Haribara, Y., Igarashi, K., Sonobe, T., Tamate, S., Honjo, T., ... & Takesue, H. (2016). A coherent Ising machine for 2000-node optimization problems. Science, 354(6312), 603-606.

[6] Yamaoka, M., Yoshimura, C., Hayashi, M., Okuyama, T., Aoki, H., & Mizuno, H. (2015). A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing. IEEE Journal of Solid-State Circuits, 51(1), 303-309.

[7] Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. science, 220(4598), 671-680.

[8] Glover, F. (1989). Tabu search—part I. ORSA Journal on computing, 1(3), 190-206.

PyQUBO 的源起與歷史

起源背景

QUBO（Quadratic Unconstrained Binary Optimization，二次無約束二元最佳化）與 Ising 模型是各種**組合最佳化問題**的通用表述方式。隨著 D-Wave 量子退火機與各種伊辛機 (Ising machines) 的出現，研究者需要一個方便的程式工具，將現實問題快速轉換為 QUBO/Ising 形式。

為此，**Recruit Communications Co., Ltd.** 的研究團隊開發了 **PyQUBO** —— 一個以 Python 實現的 **DSL（Domain Specific Language）**，能夠以數學表達式方式建構 QUBO 模型，並自動輸出可供 D-Wave Ocean SDK 或其他 BQM（Binary Quadratic Model）相容框架使用的格式。

歷史沿革

- **2019 年**：在 *Journal of the Physical Society of Japan* 發表的文章中首次介紹了 PyQUBO 的設計理念【Tanahashi et al., 2019】。該論文討論了伊辛機的應用以及對應的軟體開發工具，其中就包含 PyQUBO 的初版構想。
- **2021 年**：PyQUBO 的完整介紹與功能發表於 *IEEE Transactions on Computers*【Zaman, Tanahashi & Tanaka, 2021】。同時在 arXiv 發表技術報告 (arXiv:2103.01708)，詳細介紹了 Constraint、Placeholder 與 Integer 類別的設計。
- **GitHub 開源**：專案由 Recruit Communications Co., Ltd. 在 GitHub 公開，維護者包含 **Kotaro Tanahashi** 與 **Shu Tanaka** 等人。其後與 C++ 庫 **cimod** 整合以提升效能。
- **後續更新**：PyQUBO 持續維護至今，支援 Python 3.13，並保持與 D-Wave Ocean 生態系整合。

作者與單位

- **主要作者**：
  - Mashiyat Zaman
  - Kotaro Tanahashi
  - Shu Tanaka
- **所屬單位**：Recruit Communications Co., Ltd., Japan

設計理念

PyQUBO 的設計目標是讓使用者以「數學表達式」描述最佳化問題，透過編譯器將其轉換為 QUBO 或 Ising 形式，並提供：

- **Constraint**：可將約束條件嵌入 QUBO。
- **Placeholder**：允許在不重新編譯的情況下調整懲罰參數。
- **Integer**：提供整數變數的編碼支援。

這些功能使得 PyQUBO 特別適合於實驗性建模與快速原型開發。

參考文獻

- Tanahashi, K., Takayanagi, S., Motohashi, T., & Tanaka, S. (2019). *Application of Ising Machines and a Software Development for Ising Machines*. Journal of the Physical Society of Japan, 88(6), 061010. https://doi.org/10.7566/JPSJ.88.061010
- Zaman, M., Tanahashi, K., & Tanaka, S. (2021). *PyQUBO: Python Library for Mapping Combinatorial Optimization Problems to QUBO Form*. IEEE Transactions on Computers, 70(8), 1201–1213. https://doi.org/10.1109/TC.2021.3065091
- GitHub Repository: https://github.com/recruit-communications/pyqubo
- PyQUBO Documentation: https://pyqubo.readthedocs.io/

Subset Sum Problem 簡述

Subset Sum Problem 是一個經典的 NP 完全(NP-Complete, NPC)問題。給定一組整數和一個目標值 target，目標是在這些整數中找出一個子集，使其和等於 target。

問題轉換

假設我們有一組整數 $A_1, A_2, \ldots, A_n$ 和一個目標值 target，我們希望選擇這組整數的某個子集，使其和為 target。我們可以將此問題轉換為 QUBO 的形式，具體步驟如下：

定義二元變數：

將每個整數 A_i 對應到一個二元變數 x_i，若 $x_i = 1$，表示選擇 A_i 作為子集的一部分；若 $x_i = 0$，表示不選擇該數。

構建 QUBO 目標函數：

目標是使選擇的整數和 target 盡可能接近，即希望滿足：

$\Sigma_i^n A_i x_i = target$

將其轉換為一個 QUBO 目標函數，可以表示為對此和與目標值 target 之間的差平方最小化：

$(\Sigma_i^n A_i x_i - target)^2$

展開並簡化：

將上述目標函數展開後，我們得到一個二次項與一次項組成的函數：

其中常數項對優化過程無影響，可以忽略。因此我們最小化以下目標函數：

轉換成 QUBO 矩陣： 根據展開的式子，我們可以將 QUBO 問題表示為矩陣 Q 的形式，並將其輸入到量子計算機或經典優化算法中進行求解。

```
# 定義子集合問題資料
A = [1, 2, 3, 4] # 定義集合元素
n = len(A)
target = 5 # 定義子集和的目標

```

```
"""
宣告一個變數 H，H 表示 Hamiltonian 是描述系統能量
首先，我們需要初始化 H 變數為 0 即可。
"""
H = 0

"""
```

```
因為我們有 n 個元素，所以定義 n 個二元變數，xi 變
e.g. x = [Binary(x0), Binary(x1), Binary(x2), Bi
Binary 是宣告二元變數變數，讓程式知道這是二元變數，P
Binary("x") 是指建構一個 Label 為 "x" 的 Binary
"""
x = [Binary('x'+str(i)) for i in range(n)]


```

```
"""
定義目標函式，
首先，我們將計算 A[i]*x[i] 總和
e.g. H = A[0]*x[0]+A[1]*x[1]+A[2]*x[2]+A[3]*x[3]

接著我們將這個總和減去目標後平方。
e.g. H = ((A[0]*x[0]+A[1]*x[1]+A[2]*x[2]+A[3]*x[3])-

量子退火的目標是要找到若干組 x 變數的值（0 或 1）可
"""
for i in range(len(A)): # 總和所有的 A[i] * x[i
    H += A[i]*x[i]

H -= target # (總和 - 目標)
H = H*H # (總和 - 目標)^2

```

```
"""
最後，我們要將目標函式編譯成 QUBO 形式。
H.compile() 將 H 編譯成 Pyqubo 的 Model 物件，這
[Model 文件](https://pyqubo.readthedocs.io/en/latest/

model.to_qubo() 會回傳一個 tuple 為 (QUBO矩陣, 能量
[to_qubo() 文件](https://pyqubo.readthedocs.io/en/lat
註：offset 是 pyqubo 轉換為 qubo 的過程中所產生的常
"""
model = H.compile()
Q, offset = model.to_qubo() # 將目標函數轉換成 QU
print(Q) # 印出 QUBO 矩陣
```

```
{('x2', 'x0'): 6.0, ('x3', 'x2'): 24.0, ('x3', 'x3'): -24.0, ('x2',
```

```
from dwave.samplers import SimulatedAnnealingSample
sampler = SimulatedAnnealingSampler() # 宣告 Sampl
sampleset = sampler.sample_qubo(Q, num_reads=1000)
best_sample = sampleset.first.sample # 將能量最低的
print("best solution: ", best_sample)
print("Hamiltonian: ", sampleset.first.energy+offset)
```

```
best solution: {'x0': np.int8(0), 'x1': np.int8(1), 'x2': np.int8(
Hamiltonian: 0.0
```

```
# 檢查解答是否符合要求
total = 0
for i in range(len(A)):
        if best_sample[f'x{i}'] == 1:
            total += A[i]

print(total == target)
```

```
True
```