Solving the Graph Isomorphism Problem with a Quantum Annealer

In [ ]:

```
import numpy as np
import math
import matplotlib.pyplot as plt
import time
import networkx as nx
```

初步准备

定义Pauli矩阵、恒同矩阵

In [ ]:

```
sigma_i = np.array([[1, 0], [0, 1]], dtype=complex)
sigma_x = np.array([[0, 1], [1, 0]])
sigma_y = np.array([[0, -1j], [1j, 0]], dtype=complex)
sigma_z = np.array([[1, 0], [0, -1]], dtype=complex)
I = np.identity(2)
```

定义两个待测图的邻接矩阵A1、A2

In [ ]:

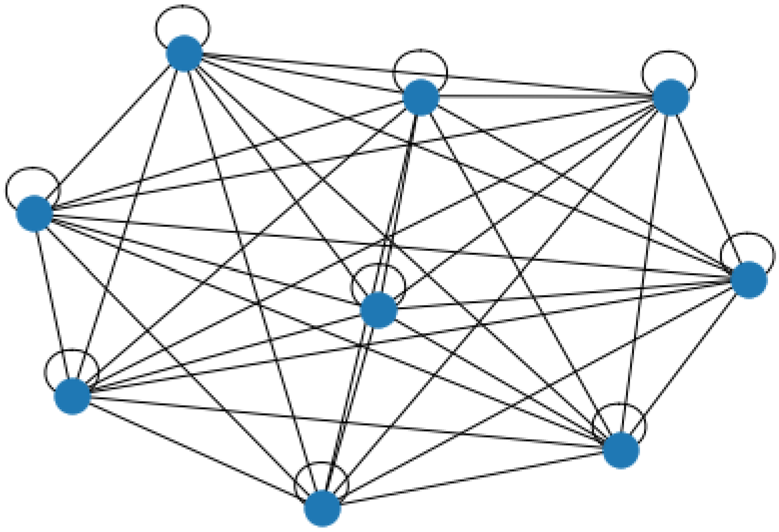
```
# 样例1

# 图1
# A1 = np.array([[0, 1, 0, 0, 1],
#                 [1, 0, 1, 0, 0],
#                 [0, 1, 0, 1, 0],
#                 [0, 0, 1, 0, 1],
#                 [1, 0, 0, 1, 0]], dtype = complex)

# 图2 (同构)
# A2 = np.array([[0, 0, 1, 1, 0],
#                 [0, 0, 0, 1, 1],
#                 [1, 0, 0, 0, 1],
#                 [1, 1, 0, 0, 0],
#                 [0, 1, 1, 0, 0]], dtype = complex)

# 图2 (不同构)
# A2 = np.array([[0, 0, 1, 0, 1],
#                 [0, 0, 0, 1, 1],
#                 [1, 0, 0, 0, 1],
#                 [1, 1, 0, 0, 0],
#                 [0, 1, 1, 0, 0]], dtype = complex)



# 样例2

A1 = np.array([[0, 1, 0, 0, 0, 0, 0, 0, 1],
               [1, 0, 0, 0, 0, 0, 0, 0, 1],
               [0, 0, 0, 1, 0, 0, 0, 0, 1],
               [0, 0, 1, 0, 0, 0, 0, 0, 1],
               [0, 0, 0, 0, 0, 1, 0, 0, 1],
               [0, 0, 0, 0, 1, 0, 0, 0, 1],
               [0, 0, 0, 0, 0, 0, 0, 1, 1],
               [0, 0, 0, 0, 0, 0, 1, 0, 1],
               [1, 1, 1, 1, 1, 1, 1, 1, 0]], dtype=complex)
```

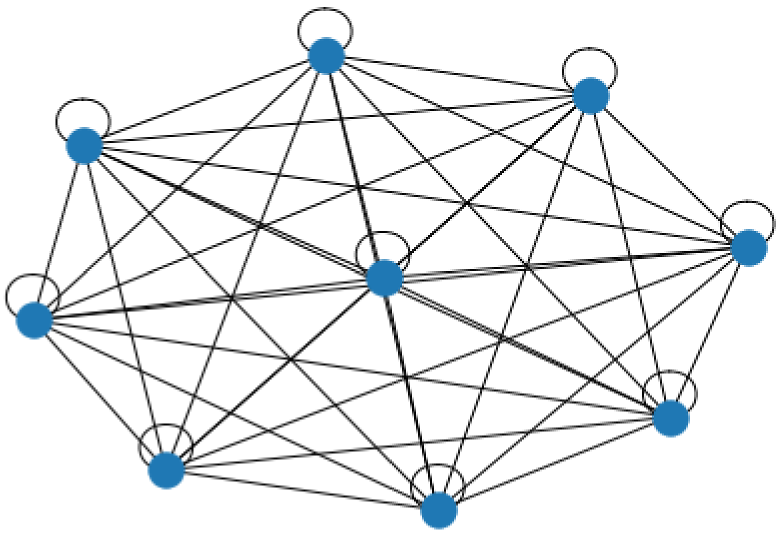
```python
A2 = np.array([[0, 1, 1, 1, 1, 0, 0, 0, 0],
               [1, 0, 1, 0, 0, 1, 1, 0, 0],
               [1, 1, 0, 0, 0, 0, 0, 1, 1],
               [1, 0, 0, 0, 1, 1, 0, 1, 0],
               [1, 0, 0, 1, 0, 0, 1, 0, 1],
               [0, 1, 0, 1, 0, 0, 1, 1, 0],
               [0, 1, 0, 0, 1, 1, 0, 0, 1],
               [0, 0, 1, 1, 0, 1, 0, 0, 1],
               [0, 0, 1, 0, 1, 0, 1, 1, 0]], dtype=complex)

# 样例3

# dim=16, degree=6, lambda=2, mu=2
# TOTAL NUMBER OF STRONGLY REGULAR GRAPHS = 2
# http://www.maths.gla.ac.uk/~es/SRGs/16-6-2-2

# A1 = np.array([
# [0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
# [1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
# [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
# [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
# [1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0],
# [1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0],
# [1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1],
# [0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0],
# [0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0],
# [0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1],
# [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0],
# [0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0],
# [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1],
# [0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1],
# [0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1],
# [0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0]
# ], dtype = complex)

# A2 = np.array([
# [0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
# [1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
# [1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0],
# [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0],
# [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0],
# [1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1],
# [1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1],
# [0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0],
# [0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1],
# [0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1],
# [0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1],
# [0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1],
# [0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0],
# [0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0],
# [0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0],
# [0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0]
# ], dtype = complex)
```

In [ ]:

```python
def generate_graph():
    G1, G2 = nx.Graph(), nx.Graph()
    for i in range(len(A1)):
        for j in range(len(A1)):
            G1.add_edge(i, j)
            G2.add_edge(i, j)
    nx.draw(G1)
    plt.show()
    nx.draw(G2)
```

```
    plt.show()

generate_graph()
```





定义后续运算所需的函数：计算Pauli-Spin的函数、进行向量单位化的函数

In [ ]:

```
def pauliZ_spin(k, N):   # 计算第k项为pauli-z，其他N-1项为I的张量积
    pauliZ_spin_matrix = sigma_z if k == 0 else I
    for i in range(N - 1):
        pauliZ_spin_matrix = np.kron(pauliZ_spin_matrix, sigma_z) if i + 1 == k \
            else np.kron(pauliZ_spin_matrix, I)
    return pauliZ_spin_matrix


# 向量归一化
def uniform(state):
    s = 0
    for i in state:
        s += abs(i) ** 2
    for i in range(len(state)):
        state[i] /= np.sqrt(s)
    return state
```

建构哈密顿算子

建构满足绝热量子演化$H(G)=(1-s)H_0+sH_p(G)$的初态哈密顿算子H_0、问题哈密顿算子$H_p(G)$

In [ ]:
```
def generate_H0(N):  # 容易制备和求解本征态的哈密顿量H0
    H0 = np.zeros(2 ** N)
    for i in range(N):
        tmp = sigma_x
        for j in range(i):
            tmp = np.kron(I, tmp)
        for j in range(N-i-1):
            tmp = np.kron(tmp, I)
        # print("tmp=", tmp)
        H0 = H0 + tmp
    H0 /= 2
    print("H0 =", H0)
    return H0


def generate_HP(A):  # 目标哈密顿量（要求该哈密顿量的本征态）
    N =  len(A)
    HP = np.zeros(2 ** N)
    for i in range(N):
        for j in range(i+1, N):
            if A[i][j] == 1:
                tmp = sigma_z
                for k in range(i):
                    tmp = np.kron(I, tmp)
                for k in range(j - i - 1):
                    tmp = np.kron(tmp, I)
                tmp = np.kron(tmp, sigma_z)
                for k in range(N - j - 1):
                    tmp = np.kron(tmp, I)
                # print("tmp=", tmp)
                HP = HP + tmp
    print("HP =", HP)
    return HP
```

In [ ]:
```
H0 = generate_H0(len(A1))
```

```
H0 = [[0.  0.5 0.5 ... 0.  0.  0. ]
 [0.5 0.  0.  ... 0.  0.  0. ]
 [0.5 0.  0.  ... 0.  0.  0. ]
 ...
 [0.  0.  0.  ... 0.  0.  0.5]
 [0.  0.  0.  ... 0.  0.  0.5]
 [0.  0.  0.  ... 0.5 0.5 0. ]]
```

In [ ]:
```
H0_eigen = np.linalg.eig(H0)  # 耗时间，只计算一次
```

In [ ]:
```
HP_1 = generate_HP(A1)
```

```
HP = [[12.+0.j  0.+0.j  0.+0.j ...  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j -4.+0.j  0.+0.j ...  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  8.+0.j ...  0.+0.j  0.+0.j  0.+0.j]
 ...
 [ 0.+0.j  0.+0.j  0.+0.j ...  8.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j ...  0.+0.j -4.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j ...  0.+0.j  0.+0.j 12.+0.j]]
```

In [ ]:
```
HP_2 = generate_HP(A2)
```

```
HP = [[18.+0.j  0.+0.j  0.+0.j ...  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j 10.+0.j  0.+0.j ...  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j 10.+0.j ...  0.+0.j  0.+0.j  0.+0.j]
 ...
 [ 0.+0.j  0.+0.j  0.+0.j ... 10.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j ...  0.+0.j 10.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j ...  0.+0.j  0.+0.j 18.+0.j]]
```

执行绝热量子演化

令$s = \frac{t}{T}$，t从0开始每次循环增加$\frac{1}{steps}$，循环共执行$steps$次，以此模拟s由0逼近1的演化过程

In [ ]:

```
def annealing_solver(steps, H0, H1):
    print(f'\nenter solver, total {steps} steps')

    t = 0
    eg_vector1 = np.abs(H0_eigen[1][0])
    eg_value1 = H0_eigen[0][0]
    energy1 = [eg_value1]
    Ht = H0    # 初始哈密顿量
    h = 6.62607015e-34    # 普朗克常量

    def expectation_value(A, psi):  # 求A在状态psi下的期望值
        norm_psi = psi / np.linalg.norm(psi)  # 单位化psi
        expectation  = np.inner(np.conj(norm_psi).T, np.matmul(A, norm_psi) )  # Ca
        return expectation

    Q2 = []  # 无初始Q2
    def generate_Q2(psi):  # spin-glass order parameter
        # 计算 sigma { <pauli-z(i), pauli-z(j)> ^ 2 } (i != j)
        N = int(np.log2(len(H0)))
        sum = 0
        for i in range(N):
            for j in range(N):
                if i == j:
                    continue
                sum = sum + expectation_value((pauliZ_spin(i, N) * pauliZ_spin(j, N

        sum = sum / (N * (N - 1))
        return np.sqrt(sum)

    Mx = []  # 无初始Mx

    for s in range(steps):
        start_loop = time.time()

        t += 1 / steps
        eg_vector_tmp1 = uniform(np.dot(Ht, eg_vector1) * (-1j) * (math.pi * 2 / st
        Ht = (1 - t) * H0 + t * H1  # 哈密顿量

        eg_value1 = np.abs(eg_vector_tmp1[0]) * eg_value1 / np.abs(eg_vector1[0])
        spin_glass = generate_Q2(eg_vector_tmp1) # 计算Q2
        x_magnetization = 2 * expectation_value(H0, eg_vector_tmp1) # 计算Mx

        eg_vector1 = eg_vector_tmp1  # 更新状态phi
        uniform(eg_vector1)

        energy1.append(eg_value1)
        Q2.append(spin_glass)
        Mx.append(x_magnetization)

        print(f'step {s + 1} finished in {time.time() - start_loop} s')
```

```python
    # print(np.abs(uniform(eg_vector1)))
    # return energy1, uniform(eg_vector1)
    return energy1, Q2, Mx
```

In [ ]:

```python
energy1, Q2_1, Mx_1 = annealing_solver(100, H0, HP_1)
```

```
enter solver, total 100 steps
step 1 finished in 0.8705539703369141 s
step 2 finished in 0.9085390567779541 s
step 3 finished in 0.8843410015106201 s
step 4 finished in 0.8775420188903809 s
step 5 finished in 0.886670126098633 s
step 6 finished in 0.8897819519042969 s
step 7 finished in 0.8881011009216309 s
step 8 finished in 0.8836019039154053 s
step 9 finished in 0.8786568641662598 s
step 10 finished in 0.8673663139343262 s
step 11 finished in 0.8812510967254639 s
step 12 finished in 0.8838012218475342 s
step 13 finished in 0.8633949756622314 s
step 14 finished in 0.865692138671875 s
step 15 finished in 0.919363260269165 s
step 16 finished in 0.8946897983551025 s
step 17 finished in 0.8383259773254395 s
step 18 finished in 0.8719892501831055 s
step 19 finished in 0.8754589557647705 s
step 20 finished in 0.9006109237670898 s
step 21 finished in 0.8770999908447266 s
step 22 finished in 0.8905353546142578 s
step 23 finished in 0.8406181335449219 s
step 24 finished in 0.8532941341400146 s
step 25 finished in 0.8468132019042969 s
step 26 finished in 0.8458149433135986 s
step 27 finished in 0.8887898921966553 s
step 28 finished in 0.8287410736083984 s
step 29 finished in 0.8574020862579346 s
step 30 finished in 0.8656761646270752 s
step 31 finished in 0.8636059761047363 s
step 32 finished in 0.8809852600097656 s
step 33 finished in 0.8614282608032227 s
step 34 finished in 0.8298001289367676 s
step 35 finished in 0.8370752334594727 s
step 36 finished in 0.8403308391571045 s
step 37 finished in 0.8390531539916992 s
step 38 finished in 0.8427019119262695 s
step 39 finished in 0.8455381393432617 s
step 40 finished in 0.8492181301116943 s
step 41 finished in 0.9555671215057373 s
step 42 finished in 1.053239107131958 s
step 43 finished in 0.8119220733642578 s
step 44 finished in 0.8712060451507568 s
step 45 finished in 0.8415710926055908 s
step 46 finished in 0.7988369464874268 s
step 47 finished in 0.8517131805419922 s
step 48 finished in 0.8136639595031738 s
step 49 finished in 0.8017451763153076 s
step 50 finished in 0.8388009071350098 s
step 51 finished in 0.8104119300842285 s
step 52 finished in 0.7946150302886963 s
step 53 finished in 0.7953610420227051 s
step 54 finished in 0.8121697902679443 s
step 55 finished in 0.8266730308532715 s
step 56 finished in 0.8309738636016846 s
```

```
step 57 finished in 0.8360593318939209 s
step 58 finished in 0.8145368099212646 s
step 59 finished in 0.7816638946533203 s
step 60 finished in 0.8132271766662598 s
step 61 finished in 0.8266448974609375 s
step 62 finished in 0.8183050155639648 s
step 63 finished in 0.8235399723052979 s
step 64 finished in 0.8387351036071777 s
step 65 finished in 0.8121547698974609 s
step 66 finished in 0.8044910430908203 s
step 67 finished in 0.8539469242095947 s
step 68 finished in 0.7763950824737549 s
step 69 finished in 0.7692270278930664 s
step 70 finished in 0.7977509498596191 s
step 71 finished in 0.8071770668029785 s
step 72 finished in 0.8336877822875977 s
step 73 finished in 0.834285020828247 s
step 74 finished in 0.8117129802703857 s
step 75 finished in 0.7936110496520996 s
step 76 finished in 0.9569211006164551 s
step 77 finished in 0.9592530727386475 s
step 78 finished in 0.8751258850097656 s
step 79 finished in 0.8364689350128174 s
step 80 finished in 0.842566967010498 s
step 81 finished in 0.8253979682922363 s
step 82 finished in 0.8414669036865234 s
step 83 finished in 0.84229207038794 s
step 84 finished in 0.8468620777130127 s
step 85 finished in 0.9073460102081299 s
step 86 finished in 0.9078421592712402 s
step 87 finished in 0.8376979827880859 s
step 88 finished in 0.8339128494262695 s
step 89 finished in 0.8351807594299316 s
step 90 finished in 0.8544652462005615 s
step 91 finished in 0.8392488956451416 s
step 92 finished in 0.9319028854370117 s
step 93 finished in 0.8790709972381592 s
step 94 finished in 0.8658621311187744 s
step 95 finished in 0.9425437450408936 s
step 96 finished in 0.9455227851867676 s
step 97 finished in 0.8178539276123047 s
step 98 finished in 1.0062298774719238 s
step 99 finished in 0.9200150966644287 s
step 100 finished in 0.869232177734375 s
```

In [ ]:

```
energy2, Q2_2, Mx_2 = annealing_solver(100, H0, HP_2)
```

```
enter solver, total 100 steps
step 1 finished in 0.8821160793304443 s
step 2 finished in 0.841407060623169 s
step 3 finished in 0.842926025390625 s
step 4 finished in 0.819195032119751 s
step 5 finished in 0.8939609527587891 s
step 6 finished in 0.9377610683441162 s
step 7 finished in 0.885282039642334 s
step 8 finished in 0.8652451038360596 s
step 9 finished in 0.8876650333404541 s
step 10 finished in 0.902595043182373 s
step 11 finished in 0.8727309703826904 s
step 12 finished in 0.8436658382415771 s
step 13 finished in 0.866693019866943 s
step 14 finished in 0.8754768371582031 s
step 15 finished in 0.9223039150238037 s
step 16 finished in 0.9276480674743652 s
```

```
step 17 finished in 0.9081549644470215 s
step 18 finished in 0.9279427528381348 s
step 19 finished in 0.8420989513397217 s
step 20 finished in 0.8732619285583496 s
step 21 finished in 0.868110179901123 s
step 22 finished in 0.8823659420013428 s
step 23 finished in 0.8978550434112549 s
step 24 finished in 0.9044568538665771 s
step 25 finished in 0.8995468616485596 s
step 26 finished in 0.9214820861816406 s
step 27 finished in 0.8437619209289551 s
step 28 finished in 0.858860969543457 s
step 29 finished in 0.8906717300415039 s
step 30 finished in 0.9106490612030029 s
step 31 finished in 0.8649101257324219 s
step 32 finished in 0.8650541305541992 s
step 33 finished in 0.8767218589782715 s
step 34 finished in 0.8889949321746826 s
step 35 finished in 0.9034988880157471 s
step 36 finished in 0.9052779674530029 s
step 37 finished in 0.9096059799194336 s
step 38 finished in 0.8599488735198975 s
step 39 finished in 0.8675730228424072 s
step 40 finished in 0.9595048427581787 s
step 41 finished in 0.9040791988372803 s
step 42 finished in 0.871950626373291 s
step 43 finished in 0.868523120880127 s
step 44 finished in 0.8661408424377441 s
step 45 finished in 0.8860468864440918 s
step 46 finished in 0.9441862106323242 s
step 47 finished in 1.0933308601379395 s
step 48 finished in 0.9141020774841309 s
step 49 finished in 0.8763377666473389 s
step 50 finished in 0.8803279399871826 s
step 51 finished in 0.9737868309020996 s
step 52 finished in 0.8816847801208496 s
step 53 finished in 0.8491671085357666 s
step 54 finished in 0.8806290626525879 s
step 55 finished in 0.8729279041290283 s
step 56 finished in 0.9141318798065186 s
step 57 finished in 0.9583921432495117 s
step 58 finished in 0.879143238067627 s
step 59 finished in 0.8409750461578369 s
step 60 finished in 0.8792667388916016 s
step 61 finished in 0.9218428134918213 s
step 62 finished in 0.8736209869384766 s
step 63 finished in 0.8806381225585938 s
step 64 finished in 0.9073278903961182 s
step 65 finished in 0.9082279205322266 s
step 66 finished in 0.8423309326171875 s
step 67 finished in 0.8463737964630127 s
step 68 finished in 0.8287460803985596 s
step 69 finished in 0.8348312377929688 s
step 70 finished in 0.8343789577484131 s
step 71 finished in 0.8713040351867676 s
step 72 finished in 0.8681449890136719 s
step 73 finished in 0.8945419788360596 s
step 74 finished in 0.8713958263397217 s
step 75 finished in 0.8831908702850342 s
step 76 finished in 0.8891479969024658 s
step 77 finished in 0.8769607543945312 s
step 78 finished in 0.8602843284606934 s
step 79 finished in 0.8796720504760742 s
step 80 finished in 0.8730580806732178 s
step 81 finished in 0.9166469573974609 s
```

```
step 82 finished in 0.8701190948486328 s
step 83 finished in 0.8297250270843506 s
step 84 finished in 0.8411848545074463 s
step 85 finished in 0.894751787185669 s
step 86 finished in 0.8446459770202637 s
step 87 finished in 0.8442091941833496 s
step 88 finished in 0.8805129528045654 s
step 89 finished in 0.8475399017333984 s
step 90 finished in 0.8388500213623047 s
step 91 finished in 0.8822340965270996 s
step 92 finished in 0.9325339794158936 s
step 93 finished in 0.8375668525695801 s
step 94 finished in 0.8278779983520508 s
step 95 finished in 0.8394491672515869 s
step 96 finished in 0.8628652095794678 s
step 97 finished in 0.8266050815582275 s
step 98 finished in 0.8616769313812256 s
step 99 finished in 0.8622510433197021 s
step 100 finished in 0.862318754196167 s
```

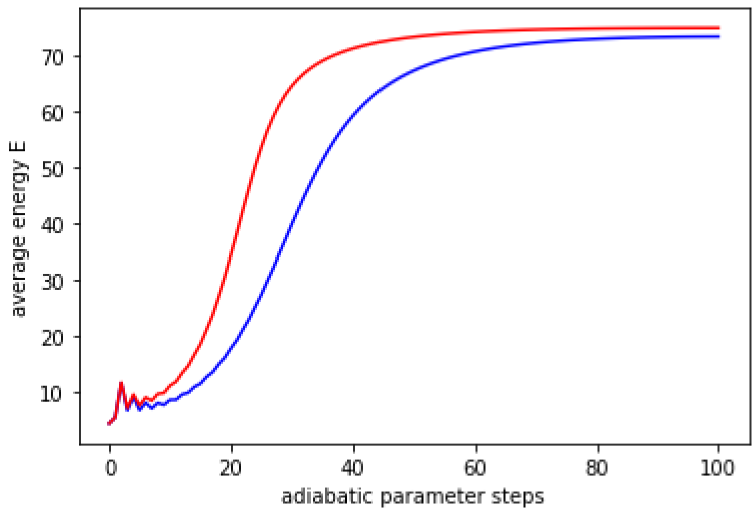
作图展示结果

若energy曲线与Q2曲线均重合，则说明二图同构；反之则二图不同构

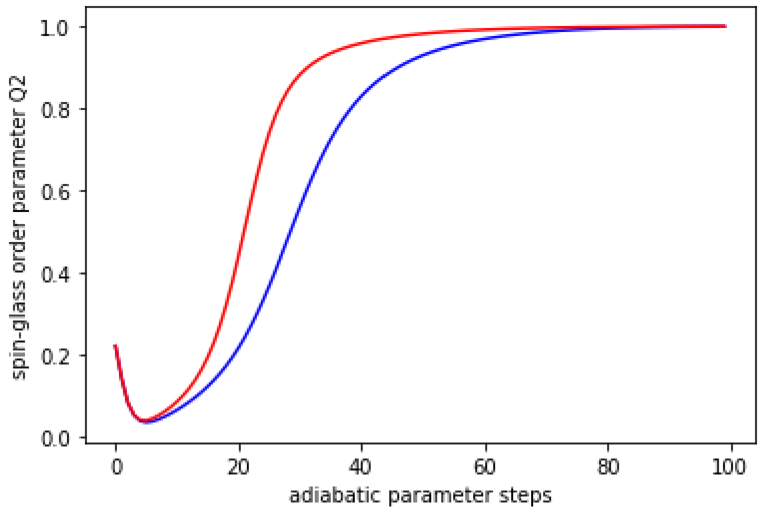
In [ ]:

```
plt.figure()
plt.plot(energy1, c = 'blue')
plt.plot(energy2, c = 'red')
plt.xlabel('adiabatic parameter steps')
plt.ylabel('average energy E')
plt.show()
print('energy1:', energy1[len(energy1) - 1])
print('energy2:', energy2[len(energy2) - 1])

plt.plot(Q2_1, c = 'blue')
plt.plot(Q2_2, c = 'red')
plt.xlabel('adiabatic parameter steps')
plt.ylabel('spin-glass order parameter Q2')
plt.show()
print('Q2_1:', Q2_1[len(Q2_1) - 1])
print('Q2_2:', Q2_2[len(Q2_2) - 1])

plt.plot(Mx_1, c = 'blue')
plt.plot(Mx_2, c = 'red')
plt.xlabel('adiabatic parameter steps')
plt.ylabel('x-magnetization Mx')
plt.show()
print('Mx_1:', Q2_1[len(Mx_1) - 1])
print('Mx_2:', Q2_2[len(Mx_2) - 1])
```

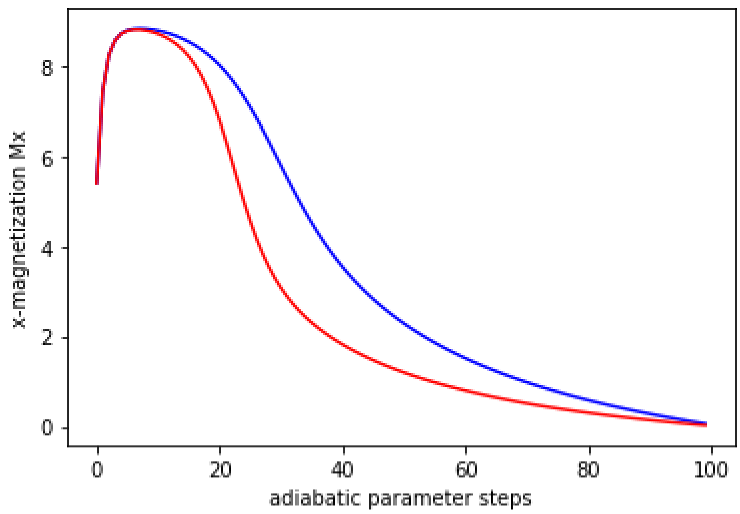
```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/matpl
otlib/cbook/__init__.py:1333: ComplexWarning: Casting complex values to real discar
ds the imaginary part
  return np.asarray(x, float)
```




```
energy1: (73.3418437052696+0j)
energy2: (74.87244359373983+0j)
```




```
Q2_1: (0.9999443651063884+0j)
Q2_2: (0.9999914942323809+0j)
```




```
Mx_1: (0.9999443651063884+0j)
Mx_2: (0.9999914942323809+0j)
```