In [ ]:

```
import scipy.io as sio
import pandas as pd
import numpy as np
```

In [ ]:

```
data_mat = sio.loadmat('exam_data/data-qp.mat')
# check keys
data_mat.keys()
```

Out[ ]:

```
dict_keys(['__header__', '__version__', '__globals__', 'A', 'C', 'b', 'd'])
```

In [ ]:

```
A = data_mat['A']
C = data_mat['C']
b = data_mat['b']
d = data_mat['d']
```

In [ ]:

```
def is_positive_semi_def(x):
    if np.all(np.linalg.eigvals(x) > 0):
        print('positive definite')
    elif np.all(np.linalg.eigvals(x) >= 0):
        print('positive semi definite')
    else:
        print('general matrix')
    # print('eigen values:\n',np.linalg.eigvals(x))
```

In [ ]:

```
alpha = 1/2
m = A.shape[0]
n = A.shape[1]
```

In [ ]:

```
# P
P_A = 2*(A.T @ A)
P_C = 2*(C.T @ C)
P = np.concatenate(( np.concatenate((P_A, np.zeros_like(P_A)), axis=1),np.concatenate((np.zeros_like(P_C),P_C), axis=1) ), axis=0)
P = np.concatenate((P, np.zeros((n+1,2*n))), axis=0)
P = np.concatenate((P, np.zeros((3*n+1,n+1))), axis=1)
P.shape
```

Out[ ]:

```
(31, 31)
```

In [ ]:

```
is_positive_semi_def(P)
```

```
positive semi definite
```

In [ ]:

```
# q
q = np.concatenate(( -2*(A.T @ b) ,-2*(C.T @ d) ),axis=0)
q = np.concatenate((q,np.zeros((n+1,1))), axis=0)
q = q.reshape(q.shape[0],)
q.shape
```

Out[ ]:

```
(31,)
```

In [ ]:

```
# G
G = np.concatenate((np.zeros((1,2*n)),np.ones((1,n+1))),axis=1)
G = np.concatenate((G,
        np.concatenate((np.zeros((n,n)), -1*np.identity(n),-1*np.identity(n), np.zeros((n,1))),axis=1)
        ),axis=0)
G = np.concatenate((G,
        np.concatenate((np.zeros((n,n)),  1*np.identity(n),-1*np.identity(n), np.zeros((n,1))),axis=1)
        ),axis=0)
G = np.concatenate((G,
        np.concatenate((-1*np.identity(n), np.zeros((n,n)), np.zeros((n,n)), -1*np.ones((n,1))),axis=1)
        ),axis=0)
G = np.concatenate((G,
        np.concatenate((1*np.identity(n), np.zeros((n,n)), np.zeros((n,n)), -1*np.ones((n,1))),axis=1)
        ),axis=0)
G.shape
```

Out[ ]:

```
(41, 31)
```

In [ ]:

```
# h
h = np.concatenate((np.array([alpha]).reshape(1,1),np.zeros((4*n,1))), axis=0)
h = h.reshape(h.shape[0],)
h.shape
```

Out[ ]:

```
(41,)
```

In [ ]:

```
# M
M = np.zeros((3*n+1,))
M.shape
```

Out[ ]:

```
(31,)
```

In [ ]:

```
# g
g = np.array([0])
```

In [ ]:

```
def f(x):
    return (1/2 * x.T @ P @ x) + (q.T @ x)
```

In [ ]:

```
from cvxopt import matrix, solvers
sol=solvers.qp(matrix(P), matrix(q), matrix(G), matrix(h))
w = np.array(sol['x'])
p_opt = float(f(w))
```

In [ ]:

```
# result
x = w[0:n]
z = w[n:2*n]
print('Optimal value:', p_opt)
print('{:5}{:15} {:15}'.format('','x', 'z'))
for i in range(n):
    print('{:>12.5E} {:15.5E}'.format(x[i][0], z[i][0]))
```

```
Optimal value: -8.500918794739846
     x               z
 1.21633E-01    -2.59405E-09
 9.14190E-02     1.60422E-10
 2.23835E-01    -5.09069E-09
 5.01055E-02     6.49523E-10
 2.23835E-01    -5.26103E-02
-2.23835E-01    -2.73284E-11
-1.90577E-02    -3.18965E-10
 1.55831E-01    -2.73922E-02
 6.52569E-02     1.13630E-10
-9.95057E-02    -1.96163E-01
```