Homework #3

Ex1: Find the minimizer of a linearly constrained optimization by programming.

Ex2: Find the minimizer of a nonlinearly constrained optimization by programming. .

name: Ensheng Shi (石 恩 升)
student ID number: 4119105089

2019/ 11/ 13/

1 Problem description

There are two excises in this homework:

- **Ex1:** Find the minimizer of a linearly constrained optimization by programming.For briefness, We solve quadratic programming problem with linear constraints.

$$\begin{aligned} & min\frac{1}{2}x^T Hx + c^T x \\ st.\ & a_i^T x = b, i \in E = 1, 2, ...m_e \\ & a_i^T x \geq b, i \in I = m_e + 1, , ...m \end{aligned} \tag{1}$$

  where the constrains are linear,that is to say, the constrains satisfy the regularity conditions.Thus find KKT point is the good way to solve this problem.

- **EX2:** Find the minimizer of a non-linearly constrained optimization by programming.

$$\begin{aligned} & min\frac{1}{2}x^T Hx + c^T x \\ st.\ & h(x) = 0, i \in E = 1, 2, ...m_e \\ & c(x) \geq 0, i \in I = m_e + 1, , ...m \end{aligned} \tag{2}$$

  where the constrains are nonlinear, it is not always available to find KKT point to solve this problem. Hence, we should find other method to transfer the constrained problem to unconstrained problem.

2 Solution and Programming

In this section, we will solve above two problems * Find the solution of Quadratic programming with linear constraints (QPLC) with active set method (ASM). * Find the solution of a non-linearly constrained optimization with exterior point penalty function method also named exterior point penalty - sequence unconstrained minimization technique(EPP-SUMT).

2.1 QPLC with ASM

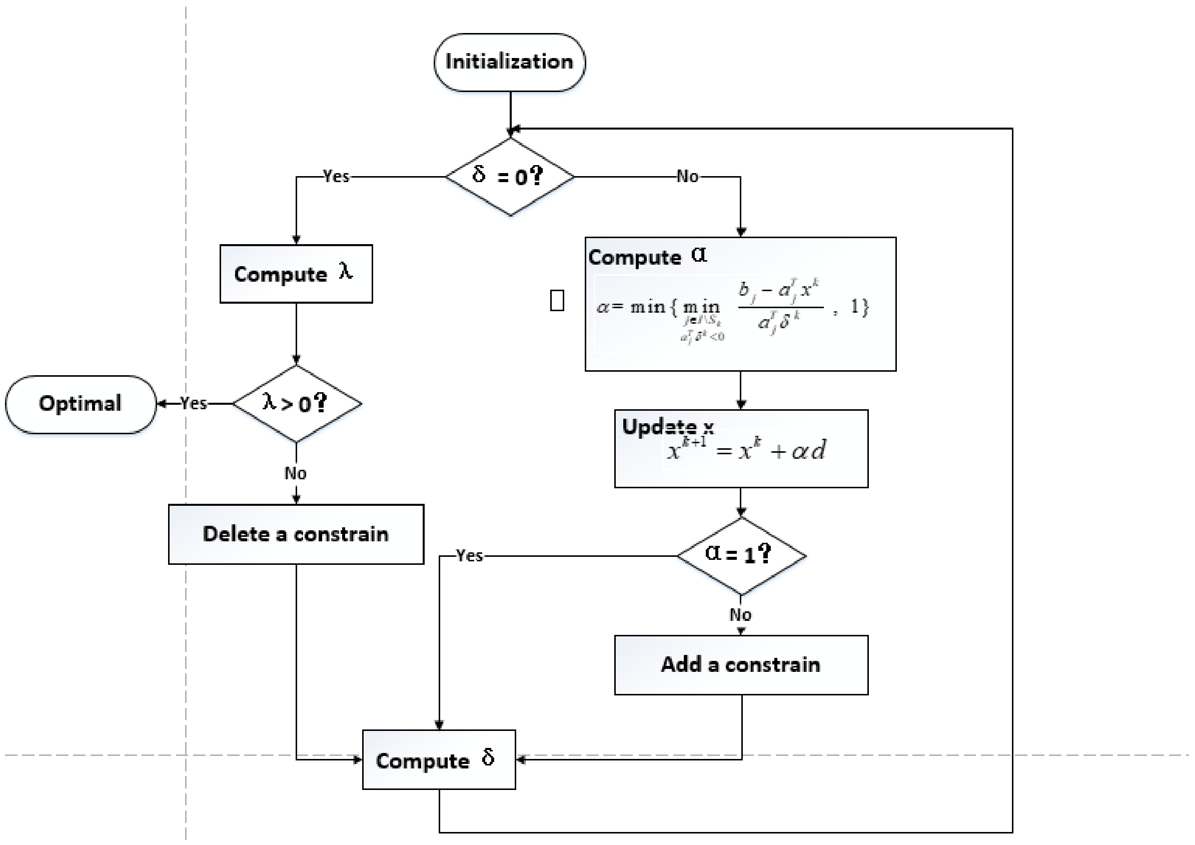
It is a common and effective method for active set method (ASE) to solve (QPLC).For ASE, finding the searching direction by reforming NOLC to NOLEC. In other word, only making use of active inequations to find the descent direction and using the other inequations to guarantee the sequence of iterations are feasible(find suitable step size).

The idea of ASM : Given x_k be a feasible point of QPLC. Solving

$$\{minf(x_K + d)s.t.(a_i)^T d = 0 \ i \in E \cup I_k\}$$

with lagrange multiplier method yields d_k or only considering equality constraints at the point x_k.

The algorithm flow graph is showed in figure1.



Initialization
δ = 0?
Yes
No
Compute λ
Compute α
$\alpha = \min\{\min_{j\in I\setminus S_k,\ a_j^T\delta^k<0} \frac{b_j - a_j^T x^k}{a_j^T\delta^k}, 1\}$
Optimal
Yes
λ > 0?
No
Update x
$x^{k+1} = x^k + \alpha d$
Delete a constrain
Yes
α = 1?
No
Add a constrain
Compute δ

Figure 1: Algorithm flow graph of ASM

where

$$\alpha = min\left(\underset{j \in I/S_k \ a_j^T(\delta)_k < 0}{min} \frac{b_j - a_j^T * x_k}{a_j^T(\delta)_k}, 1 \right) \delta = x^{k+1} - x^k \qquad (3)$$

It is easy to program according the control flow graph. In python it would be like this(only structure)

```
class Active_set(object):
    def __init__(self, H, c, A, b): ...
    def initial_set(self): ...
    def calculate_delta(self, x): ...
    def find_active_set(self): ...
    def calculate_alpha(self, x, d, activate_set_rows): ...

def Lagrange(H, c, A, b):...
```

2.2 NLC with EPP-SUMT

For non-linearly constrained optimization, we should transform the constrained optimization to a series of unconstrained optimization. This is the reason the method name is SUMT(sequence unconstrained minimization technique)

The unconstrained problem are formed by adding a term, called a penalty function, to objective function that consist of a penalty parameter multiplied by a measure of violation of the constraints. The measure of violation is positive number when the constraints are violated and is zero in the region where constraints are not violated. Denote

$$\overline{p}(x) = \sum_{i=1}^{m_e} |c_i(x)|^2 + \sum_{i=m_e+1}^{m} |min(0, c_i(x))|^2$$

The algorithm is showed in table1.

Algorithm 1 EPP—SUMT

step1 Initialize the x_0, penalty factor σ_1, amplification coefficient c and iteration times k = 0.
step2 Solve the unconstrained optimization

$$minP(x,\sigma) = f(x) + \sigma_k \overline{P}(x), \quad \sigma > 0$$

where $\overline{p}(x) = \sum_{i=1}^{m_e} |c_i(x)|^2 + \sum_{i=m_e+1}^{m} |min(0, c_i(x))|^2 yields x_k = x(\sigma_k)$
step3 If $\sigma_k \overline{P}(x) < \epsilon$, x_k is the approximate solution. Otherwise $\sigma_{k+1} = c\sigma_k, \quad k = k+1$ and goto step2.

In python it would be like this(only structure)

```
x = x0
while(True):
   sigma_k = c*sigma_k
   res = minimize(penalty_func(sigma_k), x, method='SLSQP', )
```

```
x = res.x
k = k + 1
data_x.append(x)
if penalty(sigma_k)(x) < epsilon:
    break
```

3 Results

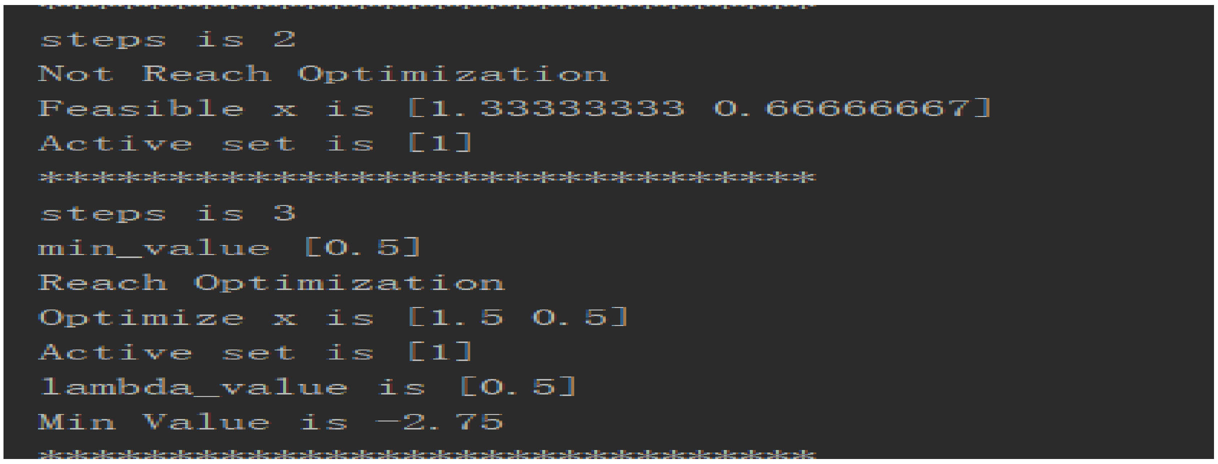
In this section , we will show some results and have a discussion with respect to the results of the methods. The 3.1 first displays the result of prolem in class and the discuss the cases of different initial points and high dimension.

3.1 The result of ASE

In the class, we solve a QP by ASM , so the example will be choose to verify my program.

$$\begin{aligned} min \quad & x_1^2 + x_2^2 - x_1 x_2 - 3x_1 \\ s.t \quad & -x_1 - x_2 \geq -2 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned} \tag{4}$$

The solution is $x = [3/2, 1/2]^T$. We random chose an active condition and set the corresponding initial set. Figure2.



```
steps is 2
Not Reach Optimization
Feasible x is [1.33333333 0.66666667]
Active set is [1]
****************************************
steps is 3
min_value [0.5]
Reach Optimization
Optimize x is [1.5 0.5]
Active set is [1]
lambda_value is [0.5]
Min Value is -2.75
****************************************
```

Figure 2: The result of example in class

The sequence of iterations is (0,0)->(5/3,1/3)->(3/2,1/2). All the points x_k are at the boundary of feasible area as showed in Figure 3

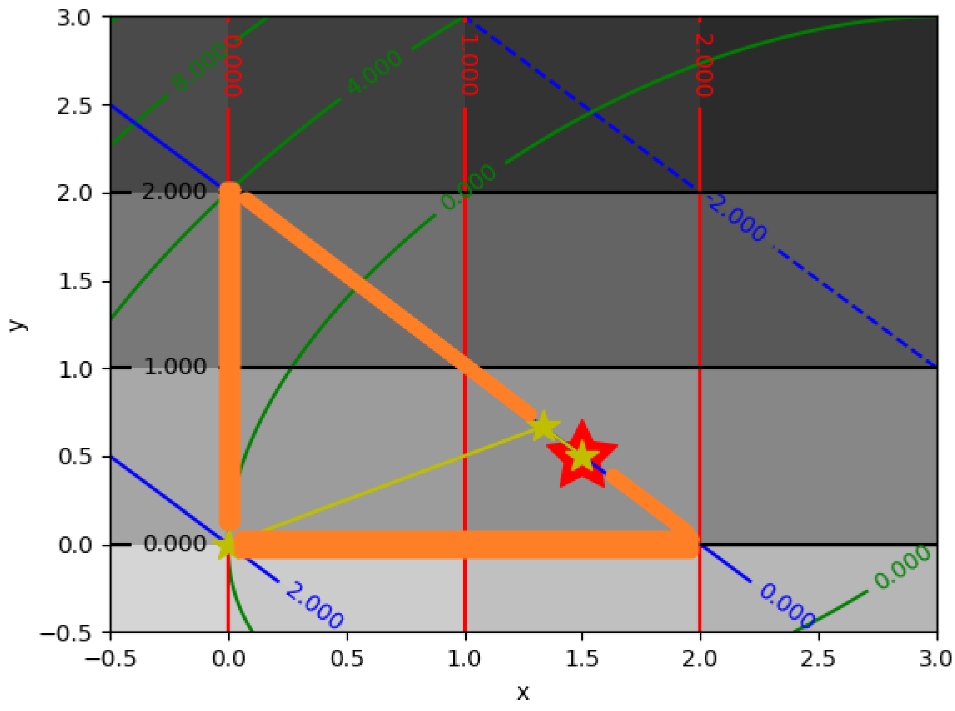
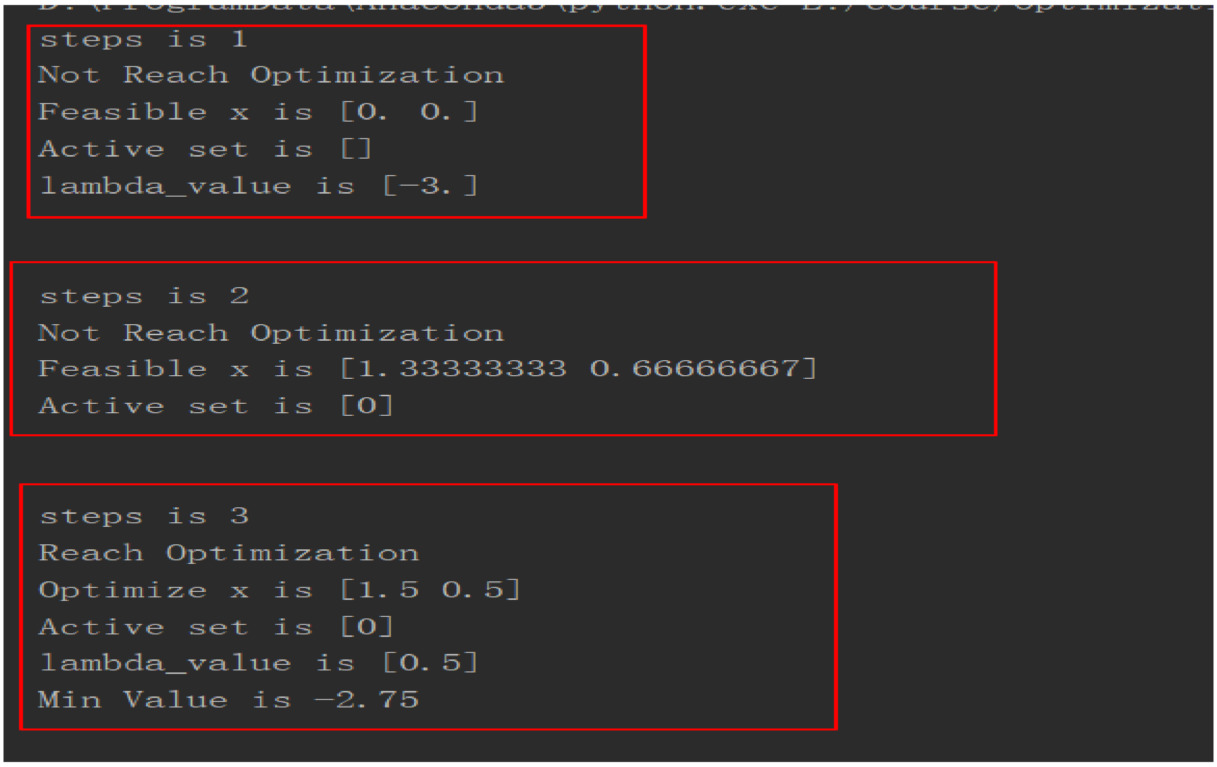


Figure 3: The iteration sequence. the solution of each unconstrained optimization is represented as a five-pointed star. The biggest five-pointed star is the minimizer. The triangular region is the feasible area. we can observe that all the points x_k are at the boundary of feasible area. The other line is contour line

3.1.1 Different initial points

In my implementation of ASM , an active condition is random selected ,so corresponding initial points will be different. The above initial point is $x_0 = [0, 0]$. When we set initial point $x_0 = [1.5, 0]$,active set is empty and it also reached the optimal solution. Note that we chose different constraints to determine different initial points x_0, so all the initial points are located in the boundary of feasible area. However we revised the program to allow us to set initial point and active set. For my perspective , it is convenient to set initial point and active set automatically.



```
steps is 1
Not Reach Optimization
Feasible x is [0.  0.]
Active set is []
lambda_value is [-3.]

steps is 2
Not Reach Optimization
Feasible x is [1.33333333 0.66666667]
Active set is [0]

steps is 3
Reach Optimization
Optimize x is [1.5 0.5]
Active set is [0]
lambda_value is [0.5]
Min Value is -2.75
```

Figure 4: The result of example in class

3.1.2 High dimension

Take another example.When the dimension is higher, the ASM can still find the solution. Figure 5 presents the result.

The results prove ASE method is workable for QPLC and the sequence of iterations is in the boundary of feasible area.

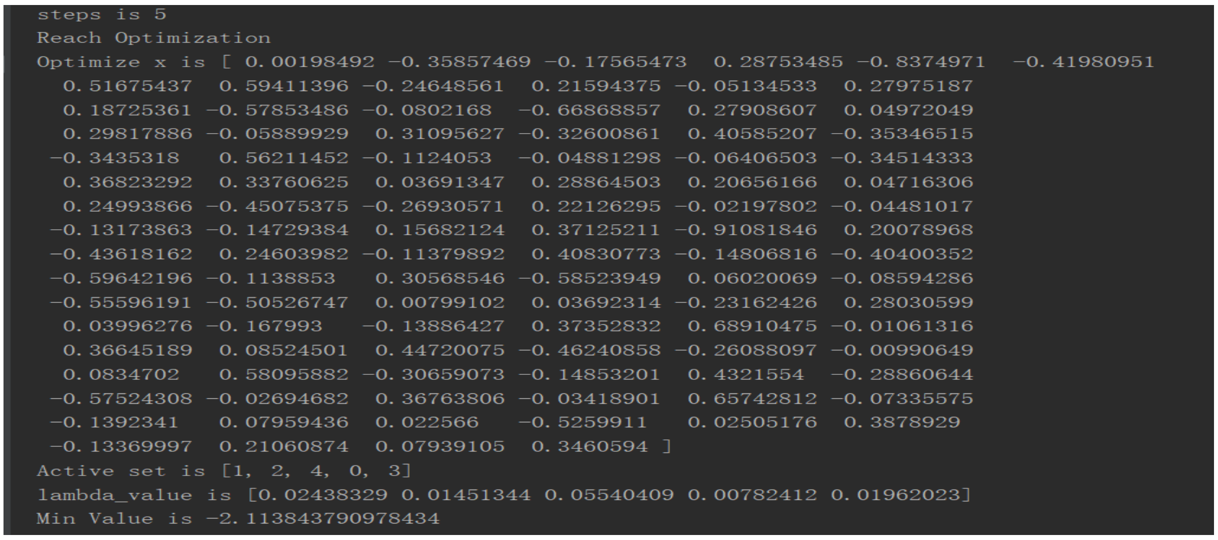
3.2 The result of EPP-SUMT

In above section, we have given the algrithm of EPP-SUMT. Here, take a simple example.

$$minf(x) = x_1^2 + x_2^2 s.t x_1 - 1 \geq 0$$

Obviously, the solution is $x^* = [1,0]^T$. We set $x_0 = [0.0, 2.0]_T, \epsilon = 10E-5, \sigma_0 = 1, c = 10$. The result is showed in figure 6

we can observe that all the point is out of feasible area except the solution.



```
steps is 5
Reach Optimization
Optimize x is [ 0.00198492 -0.35857469 -0.17565473  0.28753485 -0.8374971  -0.41980951
  0.51675437  0.59411396 -0.24648561  0.21594375 -0.05134533  0.27975187
  0.18725361 -0.57853486 -0.0802168  -0.66868857  0.27908607  0.04972049
  0.29817886 -0.05889929  0.31095627 -0.32600861  0.40585207 -0.35346515
 -0.3435318   0.56211452 -0.1124053  -0.04881298 -0.06406503 -0.34514333
  0.36823292  0.33760625  0.03691347  0.28864503  0.20656166  0.04716306
  0.24993866 -0.45075375 -0.26930571  0.22126295 -0.02197802 -0.04481017
 -0.13173863 -0.14729384  0.15682124  0.37125211 -0.91081846  0.20078968
 -0.43618162  0.24603982 -0.11379892  0.40830773 -0.14806816 -0.40400352
 -0.59642196 -0.1138853   0.30568546 -0.58523949  0.06020069 -0.08594286
 -0.55596191 -0.50526747  0.00799102  0.03692314 -0.23162426  0.28030599
  0.03996276 -0.167993   -0.13886427  0.37352832  0.68910475 -0.01061316
  0.36645189  0.08524501  0.44720075 -0.46240858 -0.26088097 -0.00990649
  0.0834702   0.58095882 -0.30659073 -0.14853201  0.4321554  -0.28860644
 -0.57524308 -0.02694682  0.36763806 -0.03418901  0.65742812 -0.07335575
 -0.1392341   0.07959436  0.022566   -0.5259911   0.02505176  0.3878929
 -0.13369997  0.21060874  0.07939105  0.3460594 ]
Active set is [1,  2,  4,  0,  3]
lambda_value is [0.02438329 0.01451344 0.05540409 0.00782412 0.01962023]
Min Value is -2.113843790978434
```

Figure 5: Solution of QP with ASM , dimension = 100

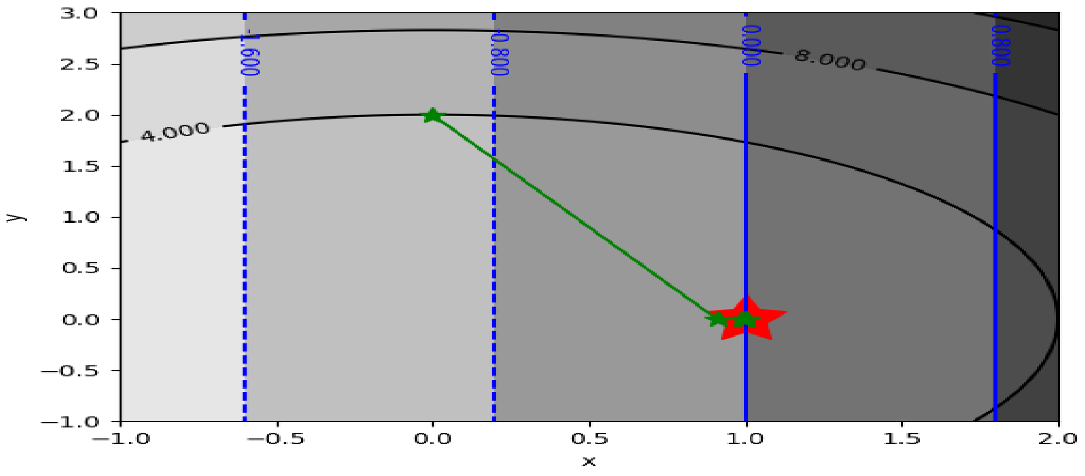


Figure 6: Solution of EPP-SUMT. Feasible area is $x_1 \geq 1$. Three little five-pointed stars are iteration points and the biggest star is minimizer .we can observe that all the point is out of feasible area except the solution

3.2.1 Different initial points

When the initial points is set in the feasible , the result is showed in figure 7. We can concluded that the EPP-SUMT replaces a constrained optimization problem a series of unconstrained problems whose solutions converge to the solution from the out of the feasible area.

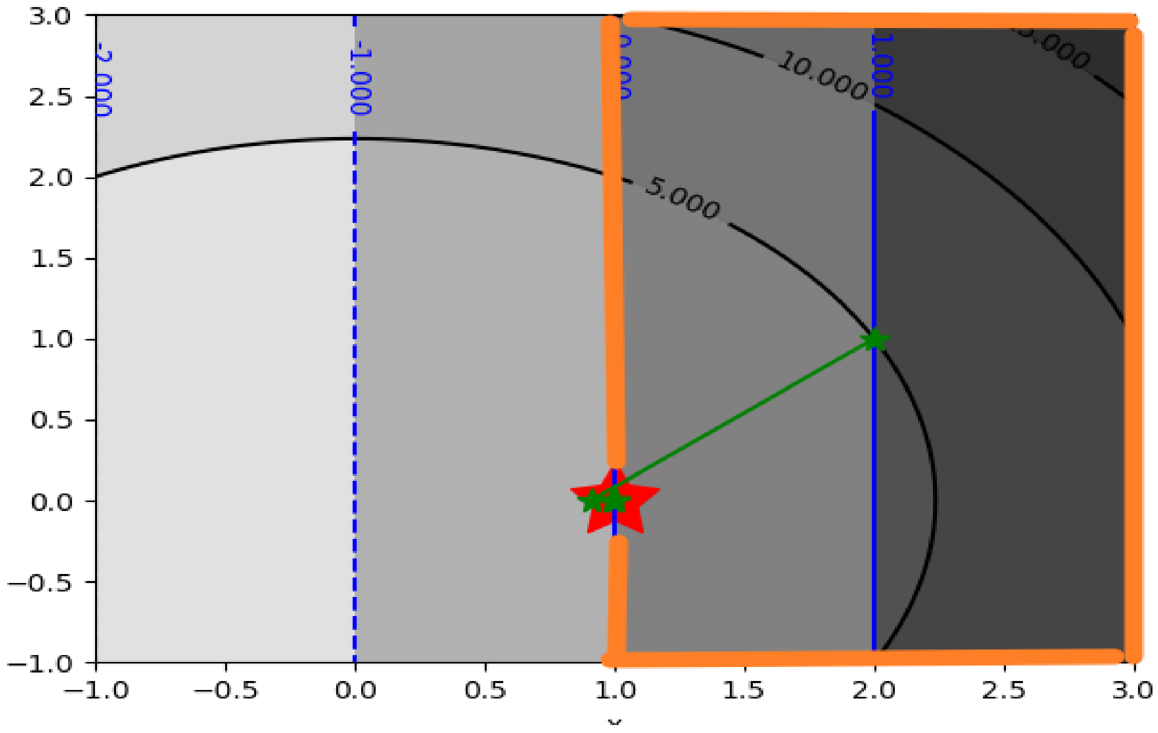


Figure 7: Solution of EPP-SUMT, initial area is in the feasible area, which is displayed by coarse. In the first iteration, the point is out of feasible area .Eventually the solution reach the boundary of feasible area.

4 Conclusion and Acquirement

When implement ASM, we should solve two sub problem.

- Find the solution of linear equation constrains using Lagrange multiplier method.
- Understand what KKT point and KKT conditions are and how to use them.

So we have a profound understanding of Lagrange multiplier method and KKT. They are useful and powerful. After learning optimization method, I think i should know them. Next, when we encounter QPLC, ASM is a note bad menthod to handle it. When implement EPP, we should

- **the idea of penalty method**A penalty method replaces a constrained optimization problem by a series of unconstrained problems whose solutions ideally converge to the solution of the original constrained problem.

- **how to transform constrained optimization to unconstrained optimizatio**The unconstrained problems are formed by adding a term, called a penalty function, to the objective function that consists of a penalty parameter multiplied by a measure of violation of the constraints.

After finishing exercise 2, we know a new thought to solve the constrained optimization.

Appendix

4.1 ASE.py

```
#!/usr/bin/env python
#!-*-coding:utf-8 -*-
'''
@version: python3.7
@author: ‘enshi ‘
@license: Apache Licence
@contact: *******@qq.com
@site:
@software: PyCharm
@file: ASM.py
@time: 11/10/2019 5:00 PM
'''

# ASE
# min 1/2x.THx+c.Tx
# s.t. Ax>=b
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.axisartist as axisartist
from scipy.stats import ortho_group
from matplotlib import pyplot as plt
def drawContour(data_x,data_y,xstar):
   x_arange = np.linspace(-0.5, 3.0, 256)
   y_arange = np.linspace(-0.5, 3.0, 256)
   X, Y = np.meshgrid(x_arange, y_arange)
   Z1 = X ** 2 + Y ** 2-X*Y-3*X
   Z2= -X - Y + 2
   Z3 = X
   Z4 = Y
```

```
    plt.xlabel('x')
    plt.ylabel('y')
    nm = 3
    plt.contourf(X, Y, Z1, nm, alpha=0.75, cmap='gray_r')
    plt.contourf(X, Y, Z2, nm, alpha=0.75, cmap='gray_r')
    plt.contourf(X, Y, Z3, nm, alpha=0.75, cmap='gray_r')
    plt.contourf(X, Y, Z4, nm, alpha=0.75, cmap='gray_r')
    C1 = plt.contour(X, Y, Z1, nm, colors='green')
    C2 = plt.contour(X, Y, Z2, nm, colors='blue')
    C3 = plt.contour(X, Y, Z3, nm, colors='red')
    C4 = plt.contour(X, Y, Z4, nm, colors='black')
    plt.clabel(C1, inline=1, fontsize=10)
    plt.clabel(C2, inline=1, fontsize=10)
    plt.clabel(C3, inline=1, fontsize=10)
    plt.clabel(C4, inline=1, fontsize=10)
    plt.plot(data_x, data_y, marker = "*",c = "y",ms ="15")
    plt.scatter(xstar[0], xstar[1], marker=(5, 1), c="r", s=1000)
    #plt.plot(1,0,"*", ms = 10)
    plt.show()

class Active_set(object):
    def __init__(self, H, c, A, b):
        self.H = H
        self.c = c
        self.A = A
        self.b = b
        self.epsilon = 1e-6

    def initial_set(self):
        # select initial active set
        activae_set_rows = [0]


        idx = activae_set_rows[0]
        index = np.where(self.A[idx] != 0)[0][0]
        value = self.A[idx][index]#系数
        feasible_x = np.zeros(len(self.A[idx]))
        #print("b shape",b.shape)
        feasible_x[index] = self.b[idx][0]/float(value)

        feasible_x = feasible_x.reshape(-1, 1)

        return activae_set_rows, feasible_x

    def calculate_delta(self, x):
        # cacluate derivation in x point
        return np.matmul(self.H, x) + self.c

    def find_active_set(self):
        activate_set_rows, feasible_x = self.initial_set()
        steps = 0
```

```
print("steps is {}".format(steps))
print("initial x is {}".format(feasible_x.flatten()))
print("Active set is {}".format(activate_set_rows))
print(30*"*")
data_x = [feasible_x.flatten()]
#print("data_x", data_x)
while True:
   steps += 1
   print("steps is {}".format(steps))

   # new c
   partial_x = self.calculate_delta(feasible_x)
   actual_A, actual_b, actual_b1 = find_new_data(self.A, self.b,
        activate_set_rows)
   # 利用Lagrange求得等式约束的解
   delta = Lagrange(self.H, partial_x, actual_A, actual_b)
   # print(20*"#")
   # print("delta",delta)
   # input()
   solution = delta[0: self.H.shape[1]]

   if np.sum(np.abs(solution)) < self.epsilon:
      # if np.all(solution == 0):
      # judeg all of   >= 0
      #outcome = Lagrange(self.H, self.c, actual_A, actual_b1)
      outcome = Lagrange(self.H, self.c, actual_A, actual_b1)
      lambda_value = outcome[self.H.shape[1]:].flatten()
      min_value = lambda_value.min()
      if min_value >= 0:
         print("Reach Optimization")
         print("Optimize x is {}".format(feasible_x.flatten()))
         print("Active set is {}".format(activate_set_rows))
         print("lambda_value is {}".format(lambda_value))
         print(30*"*")

         break
      else:
         index = np.argmin(lambda_value)
         activate_set_rows.pop(index)
         #
         feasible_x = feasible_x
         print("Not Reach Optimization")
         print("Feasible x is {}".format(feasible_x.flatten()))
         print("Active set is {}".format(activate_set_rows))
         print("lambda_value is {}".format(lambda_value))
         #print("Min Value is
              {}".format(0.5*np.matmul(np.matmul(feasible_x.T, self.H),
              feasible_x)[0][0] + (self.c.T@feasible_x)[0][0]))
         print(30*"*")
   else:
      in_row, alpha_k = self.calculate_alpha(feasible_x, solution.reshape(-1,
                 1), activate_set_rows)
            if in_row == -1:
                alpha = 1
            else:
                alpha = min(1, alpha_k)

            feasible_x += alpha * solution
            data_x.append(feasible_x.flatten())
            if alpha != 1:
                activate_set_rows.append(in_row)
                print("Not Reach Optimization")
                print("Feasible x is {}".format(feasible_x.flatten()))
                print("Active set is {}".format(activate_set_rows))
                print(30*"*")

                continue
            else:
                #outcome = Lagrange(self.H, self.c, actual_A, actual_b1)
                outcome = Lagrange(self.H, self.c, actual_A, actual_b1)
                lambda_value = outcome[self.H.shape[1]:].flatten()
                print("min_value",lambda_value)
                min_value = lambda_value.min()
                if min_value >= 0:
                    print("Reach Optimization")
                    print("Optimize x is {}".format(feasible_x.flatten()))
                    print("Active set is {}".format(activate_set_rows))
                    print("lambda_value is {}".format(lambda_value))
                    print("Min Value is
                          {}".format(0.5*np.matmul(np.matmul(feasible_x.T, self.H),
                          feasible_x)[0][0] + (self.c.T@feasible_x)[0][0]))
                    print(30*"*")

                    break
                else:
                    index = np.argmin(lambda_value)
                    activate_set_rows.pop(index)
                    print("Not Reach Optimization")
                    print("Feasible x is {}".format(feasible_x))
                    print("Active set is {}".format(activate_set_rows))
                    print("lambda_value is {}".format(lambda_value))
                    print(30*"*")

        #print("data_x", data_x)
    #print("data_x,before", data_x)
    data_x = np.array(data_x)
    #print("data_x", data_x)
    drawContour(data_x[:, 0], data_x[:, 1], feasible_x)


def calculate_alpha(self, x, d, activate_set_rows):
        min_alpha = 0
        inrow = -1
        for i in range(self.A.shape[0]):
            if i in activate_set_rows:
                continue
            else:
                b_i = self.b[i][0]
                a_i = self.A[i].reshape(-1, 1)
                #print("a_i ",a_i,"d ",d)
                low_number = np.matmul(a_i.T, d)
                if low_number >= 0:
                    continue
                else:
                    new_alpha = (b_i - np.matmul(a_i.T, x)[0][0])/float(low_number)
                    if inrow == -1:
                        inrow = i
                        min_alpha = new_alpha
                    # elif new_alpha < min_alpha and new_alpha != 0:
                    elif new_alpha < min_alpha:
                        min_alpha = new_alpha
                        inrow = i
                    else:
                        continue
        return inrow, min_alpha


def Lagrange(H, c, A, b):
    # lagrange eqaution
    up_layer = np.concatenate((H, -A.T), axis=1)
    zero_0 = np.zeros([A.shape[0], A.shape[0]])
    low_layer = np.concatenate((A, zero_0), axis=1)
    lagrange_matrix = np.concatenate((up_layer, low_layer), axis=0)
    #e,v = np.linalg.eig(lagrange_matrix)

    #print("e",e)
    actual_b = np.concatenate((-c, b), axis=0)
    '''
    lagrange_matrix_inverse = np.linalg.inv(lagrange_matrix)
    return np.matmul(lagrange_matrix_inverse, actual_b)
    '''
    return np.linalg.solve(lagrange_matrix.T @ lagrange_matrix, lagrange_matrix.T
         @actual_b)



def find_new_data(A, b, activate_set_rows):
    # activate_set_rows is empty
    actual_A = A[activate_set_rows]
    actual_b = np.zeros_like(b[activate_set_rows])
    return actual_A, actual_b, b[activate_set_rows]
```

```python
def create_H_c(dimension,matrix_type):
    eigvals = 10*np.abs(np.random.random((dimension)))
    A = np.eye(dimension)
    m = abs(np.random.randint(10))
    #print(m)
    for i in range(dimension):
        A[i][i] = eigvals[i]
    seed = 1
    U= np.float32(ortho_group.rvs(dim=dimension, random_state=seed))
    b = np.random.rand(dimension, 1)
    A = U.transpose().dot(A).dot(U)
    if matrix_type == "convex":
        return A,b
    if matrix_type == "consistently convex":
        return A+m*np.eye(dimension),b,m
    if matrix_type == "bounded convex":
        return

if __name__ == "__main__":
    '''
    H = np.array([[2, 0], [0, 2]])
    c = np.array([-2, -5]).reshape(-1, 1)
    A = np.array([[1, -2], [-1, -2], [-1, 2], [1, 0], [0, 1]])
    b = np.array([-2, -6, -2, 0, 0]).reshape(-1, 1)
   '''
    H = np.array([[2, -1], [-1, 2]])
    c = np.array([-3, 0]).reshape(-1, 1)
    A = np.array([[-1, -1], [1, 0], [0, 1]])
    b = np.array([-2, 0, 0]).reshape(-1, 1)

    H = np.array([[2, -1], [-1, 2]])
    c = np.array([-3, 0]).reshape(-1, 1)
    A = np.array([[1, 0],[-1, -1], [0, 1]])
    b = np.array([0, -2, 0]).reshape(-1, 1)

    '''
    H = np.array([[2, 0], [0, 2]])
    c = np.array([-2, -5]).reshape(-1, 1)
    A = np.array([[1, -2], [-1, -2], [-1, -2], [1, 0], [0, 1]])
    b = np.array([-2, -6, -2, 0, 0]).reshape(-1, 1)

    dimension = 100
    matrix_type = "convex"
    np.random.seed(0)
    H,c = create_H_c(dimension,matrix_type)
    c = c.reshape(-1,1)
    A = np.random.randint(0,10,size = [5,dimension])
    b= np.random.randint(0,10,size = [dimension,1]).reshape(-1, 1)

    H = np.array([[2, 0], [0, 2]])
    c = np.array([-1, -1]).reshape(-1, 1)
    A = np.array([[-1, -1], [1, 0], [0, 1]])
    b = np.array([-2, 0, 0]).reshape(-1, 1)
'''
    test = Active_set(H, c, A, b)

    # if A.shape[1] < 3:
    #     for i in range(3):
    #
    #         plt.plot([b[i]/A[i][0],0] , [0,b[i]/A[i][1]] , color='r')

    plt.show()
    test.find_active_set()
```

4.2 EPP-SUMT.py

```
#!/usr/bin/env python
#!-*-coding:utf-8 -*-
'''
@version: python3.7
@author:  ‘enshi ‘
@license: Apache Licence
@contact: *******@qq.com
@site:
@software: PyCharm
@file: EPP-SUMT.py
@time: 11/12/2019 4:12 PM
'''

from scipy.optimize import minimize
import numpy as np
from matplotlib import pyplot as plt
def drawContour(data_x,data_y,xstar):
    x_arange = np.linspace(-1.0, 3.0, 256)
    y_arange = np.linspace(-1.0, 3.0, 256)
    X, Y = np.meshgrid(x_arange, y_arange)
    Z1 = X ** 2 + Y ** 2
    Z2= X - 1
    #Z2 = Y - X ** 2
    #Z3 = X + Y
    plt.xlabel('x')
    plt.ylabel('y')
    nm = 3
    plt.contourf(X, Y, Z1, nm, alpha=0.75, cmap='gray_r')
    plt.contourf(X, Y, Z2, nm, alpha=0.75, cmap='gray_r')
    #plt.contourf(X, Y, Z3, nm, alpha=0.75, cmap='rainbow')
    C1 = plt.contour(X, Y, Z1, nm, colors='black')
    C2 = plt.contour(X, Y, Z2, nm, colors='blue')
    #C3 = plt.contour(X, Y, Z3, 8, colors='red')
    plt.clabel(C1, inline=1, fontsize=10)
    plt.clabel(C2, inline=1, fontsize=10)
    #plt.clabel(C3, inline=1, fontsize=10)
    plt.plot(data_x, data_y, marker = "*",c = "g",ms ="10")
    plt.scatter(xstar[0], xstar[1], marker=(5, 1), c="r", s=1000)
    #plt.plot(1,0,"*", ms = 10)
    plt.show()
#p1,epsilon, c,
def func ():
    fun = lambda x: x[0] ** 2 + x[1] ** 2
    return fun
def penalty(sigma_k):
    pfun = lambda x: sigma_k*(min((x[0]-1 ),0))**2
    return pfun
# step2
def penalty_func(sigma_k):
    pfun = lambda x: x[0] ** 2 + x[1] ** 2 + sigma_k*(min((x[0]-1 ),0))**2
    return pfun
if __name__ == "__main__":
    ##step1 初始化x0,惩罚因子 xigema 1, k =1
    epsilon = 10E-5
    c = 10
    x0 = np.array((2.0, 1.0))
    sigma_k = 1
    k = 0
    #res = minimize(penalty_func(sigma_k) , x0, method='SLSQP', )
    #####
    print("x = ",x0,'\t',"success :" + "True",'\t',"fucntion value",func()(x0))
    #judege  p (x)  < epsilon , xkis solution, stop, otersize  k+1 = c k , k = k+1
         gotostep2
    x = x0
    data_x = [x]
    print("penalty(x)",penalty(sigma_k)(x))
    while(True):
        sigma_k = c*sigma_k
        res = minimize(penalty_func(sigma_k), x, method='SLSQP', )
        x = res.x
        print("Iteration time : ",k)
        print("x = ", res.x, '\t', "success", res.success, '\t',
             "function value", res.fun - penalty(sigma_k)(x), "\t", "penalty value :",
                  penalty(sigma_k)(x))
        k = k + 1
        data_x.append(x)
        if penalty(sigma_k)(x) < epsilon:
           break
    data_x = np.array(data_x)
    drawContour(data_x[:, 0], data_x[:, 1], x)
```