最优化第二次作业

第二次作业题

课程名称：	最优化理论与算法 II
姓名：	郅啸淇
学院：	管理学院
专业：	大数据管理与应用
学号：	2184114639
指导老师：	Xiangyu Chang

2022 年 10 月 9 日

西安交通大學实验报告

专业：大数据管理与应用
姓名：郅啸淇
学号：2184114639
日期：2022 年 10 月 9 日
地点：寝室

课程名称：最优化理论与算法 II　　指导老师：Xiangyu Chang　　成绩：??
实验名称：第二次作业题　　实验类型：完成作业　　同组学生姓名：Nobody

一、 HW1

将原题中的限制条件 $\sum_{j=1}^{n} x_{ij} = a_i, \sum_{i=1}^{m} x_{ij} = b_j$
转化为 $I_{1*n}^T x_{ij} = a_i, I_{1*m}^T x_{ij} = b_j$
定义 $\mathbf{C} = \begin{bmatrix} C_{11} & \dots & C_{1n} \\ \dots & \dots & \dots \\ C_{m1} & \dots & C_{mn} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \dots & \dots & \dots \\ x_{m1} & \dots & x_{mn} \end{bmatrix}$
则原函数 $\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} C_{ij} = \mathbf{C}_{m*n}^T \mathbf{x}_{m*n} \mathbf{I}_{n*1}$
则原问题变为

$$\begin{aligned} \min_{x}\ & \mathbf{C}_{m*n}^T \mathbf{x}_{m*n} \mathbf{I}_{n*1} \\ s.t.\quad & x_{ij} \geq 0 \\ & I_{1*n}^T x_{ij} = a_i \\ & I_{1*m}^T x_{ij} = b_j \end{aligned} \tag{1}$$

二、 HW2

1. Lagrange Dual Problem and KKT Conditions

$L(x, \lambda, v) = C^T x + v^T (Ax - b) - \lambda x$
$g(\lambda, v) = \underset{x}{inf}\, L(x, \lambda, v) = -v^T b, if (C + A^T v - \lambda) = 0$
Lagrange Dual Problem

$$\begin{aligned} \max_{\lambda, v}\ & -\mathbf{v}^T b \\ s.t.\quad & C + A^T v \geq 0 \end{aligned} \tag{2}$$

$\bar{x} = diag(x), \bar{\lambda} = diag(\lambda)$
KKT Conditions $\begin{cases} C + A^T v - \lambda = 0 \\ Ax - b = 0 \\ x \geq 0 \\ \lambda \geq 0 \\ \bar{x}\bar{\lambda}\mathbf{1} = 0 \end{cases}$

2. Strong Duality

因为 $\lambda \geq 0$，所以 $L(x, \lambda, v)$ 为凸。取符合 KKT 条件的 (x^*, λ^*, v^*)，因为原问题为凸，则由定理得 (x^*, λ^*, v^*) 为原问题和对偶问题最优点并且 $d^* = p^*$，故强对偶成立。

三、 HW3

1. KKT Condditions

$L_\mu(x, \lambda) = C^T x - \mu \sum_i log x_i + v^T(Ax - b)$

$\frac{\partial L_\mu}{\partial x} = C - \mu(\sum \frac{1}{x} + v^T A) = 0$

Define:$\lambda_i = \frac{\mu}{x_i}, \mu = \lambda_i x_i$

$$\text{KKT Conditions} \begin{cases} C + A^T v - \lambda = 0 \\ Ax - b = 0 \\ \bar{x}\bar{\lambda}\mathbf{1} = m\bar{u}\mathbf{1} \end{cases}$$

2. Secant Equation

Define:$r_1 = C - \lambda + v^T A, r_2 = Ax - b, r_3 = \bar{x}\bar{\lambda}\mathbf{1} - \bar{\mu}\mathbf{1}$

$$F(x, \lambda, v) = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}$$

$$\text{Secant Equation: } F(x^t, \lambda^t, v^t) - \nabla F(x^t, \lambda^t, v^t)(\begin{pmatrix} x^t \\ \lambda^t \\ v^t \end{pmatrix} - \begin{pmatrix} x \\ \lambda \\ v \end{pmatrix}) = 0$$

$$\text{Explicit Form: } \begin{pmatrix} C - \lambda^t + v^T A \\ Ax^t - b \\ \bar{x}\bar{\lambda}\mathbf{1} - \bar{\mu}\mathbf{1} \end{pmatrix} - \begin{pmatrix} 0 & -1 & A \\ A & 0 & 0 \\ \bar{\lambda} & \bar{x} & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta \lambda \\ \Delta v \end{pmatrix} = 0$$

四、 HW4

1. Standard Form

$$\begin{aligned} \min_{x_1, x_2, x_3, x_4} \quad & -5x_1 - x_2 + 0x_3 + 0x_4 \\ s.t. \quad & x_1 + x_2 + x_3 = 5 \\ & 2x_1 + \tfrac{1}{2}x_2 + x_4 = 8 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned} \tag{3}$$

2. Implement of Simplex method

```python
import numpy as np
from math import inf

```

```python

def Termination(target):
    for x in target:
        if x < 0:
            return False
    return True


def index(table):
    min = table[0]
    index2 = 0
    for i, t in enumerate(table):
        if table[i] < min:
            min = table[i]
            index2 = i
    return index2


c = np.array([-5, -1])
b = np.array([[5], [8]])
Info = np.array([[1, 2], [2,1/2]]) # 代表限制条件
basic = np.eye(Info.shape[0]) # 产生基变量
s = np.append(Info, basic, axis=1)
A = np.append(s, b, axis=1)
c1 = np.append(c, np.zeros(Info.shape[0]))
print(c1)
print(A)
Cost = np.array(np.zeros(A.shape[1]))
Cost[0:c.shape[0]] = c
BV = np.array(range(c.shape[0], A.shape[1] - 1))
temp = np.array([Cost[y] for y in BV]).reshape(1, 2)
target = Cost - np.dot(temp, A)

print(target)
target_zero = [x < 0 for x in target[0][:target[0].shape[0] - 1]]

target_zero = [x for x in target[0][:target[0].shape[0] - 1]]
while 1:
    if Termination(target[0]) == True:
        break
    ans = np.zeros(A.shape[0])
    for i in range(0, A.shape[0]):
        ans[i] = inf
    enter = index(target[0][:target[0].shape[0] - 1])
    for i in range(0, A.shape[0]):
        if A[i][enter] > 0:
            ans[i] = A[i][A.shape[1] - 1] / A[i][enter]
        else:
            ans[i] = inf
    leave = BV[index(ans)]
```

```python
    A[index(ans)] = A[index(ans)] / A[index(ans)][enter]
    for i in range(0, A.shape[0]):
        if i != index(ans):
            A[i] = A[i][:] - A[i][enter] * A[index(ans)]
        else:
            A[i] = A[index(ans)]
    BV[index(ans)] = enter
    res = 0
    print("basis feasible vairable:")
    for i in range(0, A.shape[0]):
        print("x", BV[i], ":")
        res += c1[BV[i]] * A[i][A.shape[1] - 1]
        print(A[i][A.shape[1] - 1])
    print("result:")
    print(res)
    print(A)
    print("target[0], before:")
    print(target[0])
    temp = np.array([Cost[y] for y in BV]).reshape(1, 2)
    target[0] = target[0] - np.dot(temp, A)
    print(target[0])
```

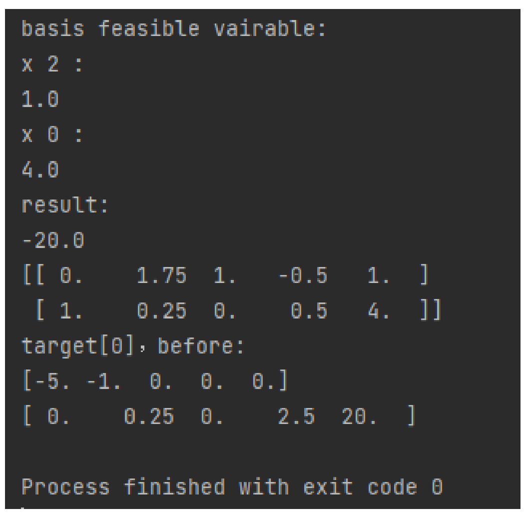

图 1: Result of Simplex Method

3. Implement of Interior Point method

```python
import numpy as np


def Interior_Point(c, A, b):
    x = np.ones((A.shape[1], 1))
    v = np.ones((b.shape[0], 1))
```

```python
    lam = np.ones((x.shape[0], 1))
    one = np.ones((x.shape[0], 1))
    mu = 1
    n = A.shape[1]
    x_ = np.diag(x.flatten())
    lam_ = np.diag(lam.flatten())
    r1 = np.matmul(A, x) - b
    r2 = np.matmul(np.matmul(x_, lam_), one) - mu * one
    r3 = np.matmul(A.T, v) + c - lam
    r = np.vstack((r1, r2, r3))
    F = r
    n1 = np.linalg.norm(r1)
    n2 = np.linalg.norm(r2)
    n3 = np.linalg.norm(r3)
    zero11 = np.zeros((A.shape[0], x.shape[0]))
    zero12 = np.zeros((A.shape[0], A.shape[0]))
    zero22 = np.zeros((x.shape[0], A.shape[0]))
    zero33 = np.zeros((A.shape[1], A.shape[1]))
    one31 = np.eye(A.shape[1])
    tol = 1e-8
    t = 1
    alpha = 0.5
    while max(n1, n2, n3) > tol:
        nablaF = np.vstack((np.hstack((zero11, zero12, A))
                            , np.hstack((x_, zero22, lam_))
                            , np.hstack((-one31, A.T, zero33))))
        delta = np.linalg.solve(nablaF, -r)
        delta_lam = delta[0:lam.shape[0], :]
        delta_v = delta[lam.shape[0]:lam.shape[0] + v.shape[0], :]
        delta_x = delta[lam.shape[0] + v.shape[0]:, :]
        alpha = Alpha(c, A, b, lam, v, x, alpha, delta_lam, delta_v, delta_x)
        lam = lam + alpha * delta_lam
        v = v + alpha * delta_v
        x = x + alpha * delta_x
        x_ = np.diag(x.flatten())
        lam_ = np.diag(lam.flatten())
        mu = (0.1 / n) * np.dot(lam.flatten(), x.flatten())
        r1 = np.matmul(A, x) - b
        r2 = np.matmul(np.matmul(x_, lam_), one) - mu * one
        r3 = np.matmul(A.T, v) + c - lam
        r = np.vstack((r1, r2, r3))
        F = r
        n1 = np.linalg.norm(r1)
        n2 = np.linalg.norm(r2)
        n3 = np.linalg.norm(r3)
        t = t + 1
        z = (c.T @ x).flatten()[0]
    print("x:", x.flatten())
    print("v:", v.flatten())
    print("lambda:", lam.flatten())
```

```python
    print("mu:", mu)
    print("alpha:", alpha)
    print('最优值:', z)


def Alpha(c, A, b, lam, v, x, alpha, delta_lam, delta_v, delta_x):
    alpha_x = []
    alpha_lam = []
    for i in range(x.shape[0]):
        if delta_x.flatten()[i] < 0:
            alpha_x.append(x.flatten()[i] / -delta_x.flatten()[i])
        if delta_lam.flatten()[i] < 0:
            alpha_lam.append(lam.flatten()[i] / -delta_lam.flatten()[i])
    if len(alpha_x) == 0 and len(alpha_lam) == 0:
        return alpha
    else:
        alpha_x.append(np.inf)
        alpha_lam.append(np.inf)
        alpha_x = np.array(alpha_x)
        alpha_lam = np.array(alpha_lam)
        alpha_max = min(np.min(alpha_x), np.min(alpha_lam))
        alpha_k = min(1, 0.99 * alpha_max)
    return alpha_k


c = np.array([-5, -1, 0, 0]).reshape(-1, 1)
A = np.array([[1, 1, 1, 0], [2, 0.5, 0, 1]])
b = np.array([5, 8]).reshape(-1, 1)
Interior_Point(c, A, b)
```

```
x: [3.99999999e+00 1.81223587e-08 9.99999987e-01 1.81223363e-09]
v: [4.53057973e-09 2.50000000e+00]
lambda: [1.13264596e-09 2.50000004e-01 4.53057973e-09 2.50000000e+00]
mu: 4.530584332237974e-10
alpha: 1
最优值: -19.999999990938825
```

图 2: Result of Interior Method

五、 HW5

```python
import numpy as np


def Interior_Point(H, c, A, b):
    x = np.ones((A.shape[1], 1))
    v = np.ones((b.shape[0], 1))
```

```
    lam = np.ones((x.shape[0], 1))
    one = np.ones((x.shape[0], 1))
    mu = 1
    n = A.shape[1]
    x_ = np.diag(x.flatten())
    lam_ = np.diag(lam.flatten())
    r1 = np.matmul(A, x) - b
    r2 = np.matmul(np.matmul(x_, lam_), one) - mu * one
    r3 = np.matmul(A.T, v) + np.matmul(H, x) + c - lam
    r = np.vstack((r1, r2, r3))
    F = r
    n1 = np.linalg.norm(r1)
    n2 = np.linalg.norm(r2)
    n3 = np.linalg.norm(r3)
    zero11 = np.zeros((A.shape[0], x.shape[0]))
    zero12 = np.zeros((A.shape[0], A.shape[0]))
    zero22 = np.zeros((x.shape[0], A.shape[0]))
    zero33 = np.zeros((A.shape[1], A.shape[1]))
    one31 = np.eye(A.shape[1])
    tol = 1e-8
    t = 0
    alpha = 1
    while max(n1, n2, n3) > tol:
        nablaF = np.vstack((np.hstack((zero11, zero12, A))
                            , np.hstack((x_, zero22, lam_))
                            , np.hstack((-one31, A.T, H))))
        delta = np.linalg.solve(nablaF, -r)
        delta_lam = delta[0:lam.shape[0], :]
        delta_v = delta[lam.shape[0]:lam.shape[0] + v.shape[0], :]
        delta_x = delta[lam.shape[0] + v.shape[0]:, :]
        alpha = Alpha(c, A, b, lam, v, x, alpha, delta_lam, delta_v, delta_x)
        lam = lam + alpha * delta_lam
        v = v + alpha * delta_v
        x = x + alpha * delta_x
        x_ = np.diag(x.flatten())
        lam_ = np.diag(lam.flatten())
        mu = (0.1 / n) * np.dot(lam.flatten(), x.flatten())
        r1 = np.matmul(A, x) - b
        r2 = np.matmul(np.matmul(x_, lam_), one) - mu * one
        r3 = np.matmul(A.T, v) + np.matmul(H, x) + c - lam
        r = np.vstack((r1, r2, r3))
        F = r
        n1 = np.linalg.norm(r1)
        n2 = np.linalg.norm(r2)
        n3 = np.linalg.norm(r3)
        t = t + 1
        z = (c.T @ x).flatten()[0]
    print("x:", x.flatten())
    print("v:", v.flatten())
    print("lambda:", lam.flatten())
```

```python
    print("mu:", mu)
    print("alpha:", alpha)
    print('最优值:', z)


def Alpha(c, A, b, lam, v, x, alpha, delta_lam, delta_v, delta_x):
    alpha_x = []
    alpha_lam = []
    for i in range(x.shape[0]):
        if delta_x.flatten()[i] < 0:
            alpha_x.append(x.flatten()[i] / -delta_x.flatten()[i])
        if delta_lam.flatten()[i] < 0:
            alpha_lam.append(lam.flatten()[i] / -delta_lam.flatten()[i])
    if len(alpha_x) == 0 and len(alpha_lam) == 0:
        return alpha
    else:
        alpha_x.append(np.inf)
        alpha_lam.append(np.inf)
        alpha_x = np.array(alpha_x)
        alpha_lam = np.array(alpha_lam)
        alpha_max = min(np.min(alpha_x), np.min(alpha_lam))
        alpha_k = min(1, 0.99 * alpha_max)
    return alpha_k


H = np.array([[2, -2, 0, 0], [-2, 4, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]])
c = np.array([-2, -6, 0, 0]).reshape(-1, 1)
A = np.array([[0.5, 0.5, 1, 0], [-1, 2, 0, 1]])
b = np.array([1, 2]).reshape(-1, 1)
Interior_Point(H, c, A, b)
```

```
x: [0.8000000342, 1.20000000234, 2.234e-09, 0.400000056]
v: [5.60000000e+00 1.48303345e-09]
lambda: [7.41516535e-10 4.94344322e-10 5.60000000e+00 1.48303345e-09]
mu: 5.932132530307578e-11
alpha: 1
最优值: -7.19999452456
```

图 3: Result of Interior Method