Optimization Theory and Algorithm II **Homework 2 - 27/10/2021**

Homework 2

Lecturer: Xiangyu Chang *Scribe:* 吴博成

1 HW 1

1.1 问题重述

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \succeq 0 \end{aligned}$$

(i) 写出它的拉格朗日对偶问题。

(ii) 利用其 KKT 条件证明强对偶性成立。

1.2 问题求解

(i)

$$L(x, \lambda, v) = c^T x + v^T(Ax - b) - \lambda^T x$$

$$\begin{aligned} g(\lambda, v) &= \inf_x L(x, \lambda, v) \\ &= \inf_x \{(c + A^T v - \lambda)^T x - v^T b\} \end{aligned}$$

$$g(\lambda, v) = \begin{cases} -v^T b & , c + A^T v - \lambda = 0 \\ -\infty & , otherwise \end{cases}$$

对偶问题:

$$\begin{aligned} & \max_{\lambda, v} -v^T b \\ & s.t. \;\; c + A^T v - \lambda = 0 \\ & \qquad \lambda \succeq 0 \end{aligned}$$

等价于:

$$\begin{aligned} & \min_v v^T b \\ & s.t. \;\; c + A^T v \succeq 0 \end{aligned}$$

(ii) 已知 KKT 条件:

$$\begin{cases} c - \lambda^* + A^T v^* = 0 \\ Ax^* - b = 0 \\ \lambda_i^* x^* = 0 \\ x^* \succeq 0 \\ -\lambda^* \succeq 0 \end{cases}$$

λ^*, v^* 满足 KKT 条件，则有:

$$\begin{aligned} g(\lambda^*, v^*) &= \inf_x \{L(x, \lambda^*, v^*)\} \\ &= \inf_x \{(c + A^T v^* - \lambda^*)^T x - v^T b\} \\ &= c^T x^* + v^{*T}(Ax^* - b) - \lambda^* x^* \\ &= c^T x^* \\ &= f_0(x^*) \end{aligned}$$

故强对偶关系成立

2 HW 2

2.1 问题重述

考虑以下线性规划

$$
\begin{aligned}
&\min -5x_1 - x_2\\
&\text{s.t. } x_1 + x_2 \le 5\\
&\qquad 2x_1 + x_2/2 \le 8\\
&\qquad x_1 \ge 0, x_2 \ge 0
\end{aligned}
$$

(i) 添加松弛变量 x_3 和 x_4 以将此问题转换为标准形式。

(ii) 用单纯形法来解决这个问题。

(iii) 用内点法解决这一问题。

2.2 问题求解

(i) 通过引入松弛变量可以将问题转化为标准形式:

$$
\min_x -5x_1 - x_2 + 0x_3 + 0x_4
$$

$$
s.t. \begin{bmatrix} 1 & 1 & 1 & 0 \\ 2 & 1/2 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \end{bmatrix}
$$

(ii)(iii)

```python
import numpy as np
# 标准化后的矩阵输入初始化
class Simplex_Method(object):
    def __init__(self,A,b,c):
        self.A = A
        self.b = b
        self.c = -c #将min变为max
        self.tableau = np.array(np.concatenate((np.concatenate((A,b.T),axis=1),np.mat(np.append(self.c,0))),axis=0)) #单纯形表

    def solve(self):
        while self.can_be_improved():
            pivot_position = self.get_pivot_position()
            self.tableau = self.pivot_step(pivot_position)
        return self.get_solution()

    def can_be_improved(self): #基解可以被优化,终止条件
        self.z = self.tableau[-1,:-1]
```

```
        return any(x > 0 for x in self.z)

    def get_pivot_position(self): #找到更新位置
        column = next(i for i, x in enumerate(self.z) if x > 0)
        restrictions = []
        for eq in self.tableau[:-1,:]:
            el = eq[column]
            restrictions.append(np.inf if el <= 0 else eq[-1] / el)

        row = restrictions.index(min(restrictions))
        return row, column
    def pivot_step(self, pivot_position):# 更新单纯形表
        new_tableau = [[] for eq in self.tableau]
        i, j = pivot_position
        pivot_value = self.tableau[i][j]
        new_tableau[i] = np.array(self.tableau[i]) / pivot_value

        for eq_i, eq in enumerate(self.tableau):
            if eq_i != i:
                multiplier = np.array(new_tableau[i]) * self.tableau[eq_i][j]
                new_tableau[eq_i] = np.array(self.tableau[eq_i]) - multiplier
        return np.array(new_tableau)


    def is_basic(self,column):# 判断是否是基解
        return sum(column) == 1 and len([c for c in column if c == 0]) == len(column) - 1
    def get_solution(self):
        columns = np.array(self.tableau).T
        solutions = []
        for column in columns[:-1]:
            solution = 0
            if self.is_basic(column):
                one_index = column.tolist().index(1)
                solution = columns[-1][one_index]
            solutions.append(solution)
        return solutions


class Primal_Dual_Interior_Point_Method(object):
    def __init__(self,A,b,c,epsilon):
        self.A = A
        self.b = b
        self.c = c
        self.epsilon = epsilon
    def solve(self):
        (self.m,self.n) = (self.A.shape[0],self.A.shape[1])
```

```
        # 初始化x,l,s
        x = np.ones(shape=(self.n, ))
        l = np.ones(shape=(self.m, )) # l为lambda
        s = np.ones(shape=(self.n, ))
        k = 0
        while abs(np.dot(x, s)) > self.epsilon:
            k += 1
            sigma_k = 0.4 # 扰动KKT条件，sigma在（0,1）
            mu_k = np.dot(x, s) / self.n
            (delta_x,delta_l,delta_s) = self.solve_delta(x,l,s,sigma_k,mu_k)
            alpha = self.linesearch(x,s,delta_x,delta_l,delta_s) #线搜索寻找步长
            (x,l,s) = (x + alpha * delta_x,l + alpha * delta_l,s + alpha * delta_s)
        return x
    def solve_delta(self,x,l,s,sigma_k,mu_k):
        A_ = np.zeros(shape=(self.m + self.n + self.n, self.n + self.m + self.n))
        A_[0:self.m, 0:self.n] = np.copy(self.A)
        A_[self.m:self.m + self.n, self.n:self.n + self.m] = np.copy(self.A.T)
        A_[self.m:self.m + self.n, self.n + self.m:self.n + self.m + self.n] = np.eye(self.n)
        A_[self.m + self.n:self.m + self.n + self.n, 0:self.n] = np.copy(np.diag(s))
        A_[self.m + self.n:self.m + self.n + self.n, self.n + self.m:self.n + self.m + self.n] = np.copy(np.diag(
            x))

        r_ = np.zeros(shape=(self.n + self.m + self.n, ))
        r_[0:self.m] = np.copy(self.b - np.dot(self.A, x))
        r_[self.m:self.m + self.n] = np.copy(self.c - np.dot(self.A.T, l) - s)
        r_[self.m + self.n:self.m + self.n + self.n] = np.copy( sigma_k * mu_k * np.ones(shape=(self.n, )) -
            np.dot(np.dot(np.diag(x), np.diag(s)), np.ones(shape=(self.n, ))) )

        # solve for delta
        delta = np.linalg.solve(A_, r_)
        delta_x = delta[0:self.n]
        delta_l = delta[self.n:self.n + self.m]
        delta_s = delta[self.n + self.m:self.n + self.m + self.n]
        return (delta_x,delta_l,delta_s)

    def linesearch(self,x,s,delta_x,delta_l,delta_s):
        alpha_max = 1.0
        for i in range(self.n):
            if delta_x[i] < 0:
                alpha_max = min(alpha_max, -x[i]/delta_x[i])
            if delta_s[i] < 0:
                alpha_max = min(alpha_max, -s[i]/delta_s[i])
        eta_k = 0.99
        return min(1.0, eta_k * alpha_max)

A = np.array ([[1,1,1,0],[2,1/2,0,1]])
```

```python
b = np.array ([[5,8]])
c = np.array([[−5,−1,0,0]])
simplex = Simplex_Method(A,b,c)
print("单纯形法的解：",simplex.solve())
Primal_Dual = Primal_Dual_Interior_Point_Method(A,b,c,0.0001)
print("原始−对偶内点法的解：",Primal_Dual.solve())

P = np.array ([[2,−4],[0,4]])
q = np.array([−2,−6])
A = np.array([[1/2,1/2],[−1,2]])
b = np.array([1,2])
```

输出结果：

```
单纯形法的解： [4.0, 0, 1.0, 0]
原始−对偶内点法的解： [3.99997933e+00 6.88991803e−05 9.99951770e−01 6.88940209e−06]
```

3 HW 3

3.1 问题重述

首先，回顾课堂讲稿中的支持向量机。然后考虑下面的松弛线性支持向量机模型：

$$\min_{\mathbf{w},\mathbf{b},\xi} \frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{n} \xi_i$$
$$\text{s.t. } y_i\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + \mathbf{b}\right) \geq 1 - \xi_i$$
$$\xi_i \geq 0, i = 1, \ldots, n$$

其中 C 是一个常数。请写出它的拉格朗日对偶问题。

3.2 问题求解

Lagrange 函数

$$L\left(w, b, \xi, \alpha\right) = \frac{\|w\|^2}{2} + c\sum_i \xi_i - \sum_i \alpha\left[y_i\left(\langle w, x_i \rangle + b\right) - \left(1 - \xi_i\right)\right]$$

KKT 条件:

$$\begin{cases} \nabla_w L = w - \sum_i \alpha_i y_i x_i = 0 \\ \nabla_b L = -\sum_i \alpha_i y_i = 0 \\ \nabla_\xi L = C - \alpha = 0 \\ \alpha_i \geq 0 \\ y_i\left(\langle w, x_i \rangle + b\right) \geq 1 - \xi_i \\ \alpha_i [y_i\left(\langle w, x_i \rangle + b\right) - \left(1 - \xi_i\right)] = 0 \end{cases}$$

将 $w^* = \sum_i \alpha_i y_i x_i$，$C = \alpha$ 代入 Lagrange 函数，解得其拉格朗日对偶问题是：

$$\max_\alpha - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \left\langle x_i, x_j \right\rangle + \sum_i \alpha_i$$
$$s.t. \sum_i \alpha_i y_i = 0; \alpha_i \geqslant 0$$

4 HW 4

4.1 问题重述

使用内点法求解：

$$\min x_1^2+2x_2^2-2x_1-6x_2-2x_1x_2$$
$$\text{s.t. } x_1/2+x_2/2\le 1, -x_1+2x_2\le 2, x_1\ge 0, x_2\ge 0$$

4.2 问题求解

首先将问题化为标准形式：

$$\min_x 1/2x^TAx+bx$$
$$s.t. Cx\le d$$

之后引入损失函数 log：

$$\min_x 1/2x^TAx+bx-\mu\sum_i logs_i$$
$$s.t. Cx+s=d$$

将原问题带入得：

$$\min_x 1/2x^T\begin{bmatrix}2 & -4\\ 0 & 4\end{bmatrix}x+\begin{bmatrix}-2 & -6\end{bmatrix}x-\mu\sum_i logs_i$$
$$s.t.\begin{bmatrix}1/2 & 1/2\\ -1 & 2\end{bmatrix}x+s=\begin{bmatrix}1\\2\end{bmatrix}$$

代码求解：

```
import numpy as np
from numpy.lib.twodim_base import diag
class Interior_Point_Method_For_QP(object):
    def __init__(self,A,b,C,d,epsilon):
        self.A = A
        self.b = b
        self.C = C
        self.d = d
        self.epsilon = epsilon
    def solve(self):
        (self.m,self.n) = (self.C.shape[0],self.C.shape[1])
        # 初始化x,l,s
        x = np.ones(shape=(self.n, ))*10
        s = np.ones(shape=(self.m, )) # l为lambda
        v = np.ones(shape=(self.m, ))
```

```python
        mu_k = np.dot(x, s)*0.1 / self.n
        # 辅助函数F_mu_t
        F = np.concatenate((np.dot(self.C,x)+s-self.d, np.dot(np.dot(diag(v),diag(s))-mu_k,np.ones(shape=self.m,)), np.dot(np.dot(self.A.T,self.A),x)-np.dot(self.A.T,self.b)+np.dot(self.C.T,v)))
        print(F)
        k = 0
        while np.linalg.norm(F, ord=2) > self.epsilon and k <= 1000:
            k += 1
            mu_k = np.dot(x, s)*0.1 / self.n
            (delta_x,delta_s,delta_v) = self.solve_delta(x,s,v,F,mu_k)
            alpha = self.linesearch(x,s,delta_x,delta_s,delta_v) #线搜索寻找步长
            (x,s,v) = (x + alpha * delta_x,s + alpha * delta_s,v + alpha * delta_v)
            print(x,s,v)
        return x
    def solve_delta(self,x,s,v,F,mu_k):
        A_ = np.zeros(shape=(self.m + self.m + self.n, self.m + self.m + self.n))
        A_[0:self.m, 0:self.m] = np.eye(self.m,self.m)
        A_[0:self.m, self.m+self.n:self.m + self.m + self.n] = np.copy(self.C)
        A_[self.m:self.m + self.m, self.m:self.m + self.m] = diag(s)
        A_[self.m:self.m + self.m, 0:self.m] = diag(v)
        A_[self.m + self.m:self.m + self.m + self.n, self.m:self.m + self.m] = np.copy(self.C.T)
        A_[self.m + self.m:self.m + self.m + self.n, self.m + self.m:self.m + self.m + self.n] = np.dot(self.A.T, self.A)

        r_ = -F
        # solve for delta
        delta = np.linalg.solve(A_, r_)
        delta_s = delta[0:self.m]
        delta_v = delta[self.m:self.m + self.m]
        delta_x = delta[self.m + self.m:self.m + self.m + self.n]
        return (delta_x,delta_s,delta_v)

    def linesearch(self,x,s,delta_x,delta_s,delta_v):
        alpha_max = 1.0
        for i in range(self.n):
            if delta_x[i] < 0:
                alpha_max = min(alpha_max, -x[i]/delta_x[i])
            if delta_s[i] < 0:
                alpha_max = min(alpha_max, -s[i]/delta_s[i])
        eta_k = 0.99
        return min(1.0, eta_k * alpha_max)

A = np.array([[2,-4],[0,4]])
b = np.array([-2,-6])
C = np.array([[1/2,1/2],[-1,2]])
d = np.array([1,2])
```

```python
solver = Interior_Point_Method_For_QP(A,b,C,d,epsilon = 0.001)
print(solver.solve())
```

结果为:

```
[-0.4 1.2]
```

5 HW 5

5.1 问题重述

从点 $\mathbf{x}_0$ 到超平面 $\{\mathrm{x} \mid A\mathrm{x}=\mathbf{b}\}$ 的最短距离问题可表示为二次规划

$$\min_{\mathrm{x}} \frac{1}{2}\left\|\mathbf{x}_0-\mathrm{x}\right\|^2$$
$$\text{s.t. } A\mathbf{x}=\mathbf{b}$$

求解下述问题。

(i) 证明最优 Lagrange 乘子：

$$\boldsymbol{\nu}^*=\left(AA^{\top}\right)^{-1}\left(A\mathbf{x}_0-\mathbf{b}\right)$$

(ii) 证明解是：

$$\mathbf{x}^*=\mathbf{x}_0-A^{\top}\left(AA^{\top}\right)^{-1}\left(A\mathbf{x}_0-\mathbf{b}\right)$$

5.2 问题求解

（i）

$$L\left(x,\nu\right)=\frac{1}{2}\left\|x_0-x\right\|^2+\nu^T\left(Ax-b\right)$$

拉格朗日对偶问题:

$$g(v)=\inf_x L(x,v)$$
$$\frac{\partial L}{\partial x}=-(x_0-x)+A^Tv=0$$

故：

$$x=x_0-A^Tv$$

代入上式中可得知：

$$g(v)=-\frac{\|A^Tv\|^2}{2}+v^TAx_0-v^Tb$$

对 v 求导，并且令结果为 0，可以解得：

$$\nu^*=(AA^T)^{-1}(Ax_0-b)$$

（ii）已知 KKT 条件：

$$\begin{cases} x^*-x_0+A^Tv=0 \\ Ax=b \end{cases}$$

将第一问的结果：

$$\nu^*=(AA^T)^{-1}(Ax_0-b)$$

代入 $x^* - x_0 + A^T v = 0$ 式中，可以解得：

$$x^* = x_0 - A^T(AA^T)^{-1}(Ax_0 - b)$$