1 [Q1]

Since last assignment, we talked about proximal gradient descent. Then we talk about geometric view of contrained problems. After that, we talked about Lagrange problem and the Lagrange dual problem and how to relate the optimal $x^\star$ with $\lambda^\star$, $\upsilon^\star$. Besides, we dig deeper into KKT conditions and talked about strong duality. If $d^\star = p^\star$, strong duality holds. From there, we talked about indicator function and convex conjugate which are used to solve dual problem. Note that Fenchel duality is a more general way of Lagrangian duality.

2 [Q2]

done

3 [Q3]

3.1 (a)

$$\partial \|x\|_\infty = u$$

Define n as the index of the first maximum $|x_i|, \forall i$, if $|x_n| \geq |x_i|, \forall i$, we can have,

$$u_n = \text{sign}(x)$$

else, $u_n = 0$.
where

$$\text{sign}(x) = \begin{cases} 1, \text{ if } x \geq 0 \\ -1, \text{else} \end{cases}$$

3.2 (b)

From $u^T x = \|x\|_\infty$, we can conclude,

$$\langle y - x, u \rangle = u^T y - u^T x = u^T y - \|x\|_\infty \tag{1}$$

Since $\|u\|_1 = 1$, we have $|u_1| + |u_2| + \cdots + |u_n| = 1$.
Assume $\forall i, |y_j| \geq |y_i|$

$$u^T y = \sum_i^n u_i y_i \leq \sum_i^n |u_i||y_i| \leq \sum_i^n |u_i||y_j| \leq |y_j| \sum^n |u_i| = |y_j| = \|y\|_\infty \tag{2}$$

Hence, we can arrive at,

$$\langle y - x, u \rangle \leq \|y\|_\infty - \|x\|_\infty \tag{3}$$
$$f(x) + \langle y - x, u \rangle \leq \|y\|_\infty \tag{4}$$
$$f(x) + \langle y - x, u \rangle \leq f(y) \tag{5}$$

Therefore, u must be subgradient.

3.3 (c)

If u is a subgradient of $\|x\|_\infty$, then it satisfy,

$$\|y\|_\infty \geq \|x\|_\infty + u^T y - u^T x \tag{6}$$
$$\|y\|_\infty - u^T y \geq \|x\|_\infty - u^T x \tag{7}$$

Let $y = 0$, we can have,

$$0 \geq \|x\|_\infty - u^T x \tag{8}$$

Let $y = 2x$, we can have,

$$\|x\|_\infty - u^T x \geq 0 \tag{9}$$

Hence, we can arrive at,

$$\|x\|_\infty = u^T x \tag{10}$$

Besides, we can dig further as follows,

$$\|x\|_\infty = u^T x \tag{11}$$
$$= \sum_{i=1}^{n} u_i x_i \tag{12}$$
$$\leq \sum_{i=1}^{n} |u_i||x_i| \tag{13}$$
$$\leq \|x\|_\infty \|u\|_1 \tag{14}$$

Hence we can arrive at,

$$\|u\|_1 \geq 1 \tag{15}$$

Now, take $\|x\|_\infty = u^T x$ back to (6) we can get,

$$\|y\|_\infty \geq u^T y \tag{16}$$

Again, use the same trick as above, we can get,

$$u^T y \leq \|y\|_\infty \|u\|_1 \tag{17}$$
$$\leq \|y\|_\infty \tag{18}$$

we can get,

$$\|u\|_1 \leq 1 \tag{19}$$

Combine (15) with (19), we can get,

$$\|u\|_1 = 1 \tag{20}$$

Note (10) and (20) are exact two conditions needed to prove.

4 [Q4]

4.1 (a)

$$h_{B_2}(v) = \sup_{x \in B_2} \langle x, v \rangle \tag{21}$$

According to Cauchy-Schwartz inequality, we have,

$$\begin{aligned} \langle x, v \rangle &\leq \|x\|_2 \cdot \|v\|_2 && (22) \\ &\leq \|v\|_2 && (23) \end{aligned}$$

where inequality (23) holds because in set B_2, $\|x\|_2 \leq 1$
Hence,

$$h_{B_2}(v) = \|v\|_2$$

4.2 (b)

$$\begin{aligned} \|v\|_{2\star} &= \sup_{\|x\|_2 \leq 1} \langle x, v \rangle && (24) \\ &\leq \sup_{\|x\|_2 \leq 1} \|x\|_2 \|v\|_2 && (25) \\ &= \|v\|_2 && (26) \end{aligned}$$

where, inequality (25) comes from Cauchy-Schwartz inequality. And l_2 norm is a self-dual norm.

4.3 (c)

Let's assume $\forall i, |v_j| \geq |v_i|$

$$\begin{aligned} h_{B_1}(v) &= \sup_{x \in B_1} \langle x, v \rangle && (27) \\ &= \sup_{x \in B_1} \sum_i^n v_i x_i && (28) \\ &\leq \sup_{x \in B_1} \sum_i^n |v_i||x_i| && (29) \\ &\leq |v_j| \sup_{x \in B_1} \sum_i^n |x_i| && (30) \\ &= \|v\|_\infty && (31) \end{aligned}$$

where inequality (30) is because $\|x\|_1 \leq 1$.

4.4 (d)

Let's assume $\forall i, |v_j| \geq |v_i|$

$$\|v\|_{1\star} = \sup_{\|x\|_1 \leq 1} \langle x, v \rangle \tag{32}$$

$$= \sup_{\|x\|_1 \leq 1} \sum_i^n v_i x_i \tag{33}$$

$$= \sup_{\|x\|_1 \leq 1} \sum_i^n |v_i||x_i| \tag{34}$$

$$= |v_j| \sup_{\|x\|_1 \leq 1} \sum_i^n |x_i| \tag{35}$$

$$= \|v\|_\infty \tag{36}$$

4.5 (e)

$$h_{B_\infty}(v) = \sup_{\|x\|_\infty \leq 1} \langle x, v \rangle \tag{37}$$

$$= \sup_{\|x\|_\infty \leq 1} \sum_i^n v_i x_i \tag{38}$$

$$= \sup_{\|x\|_\infty \leq 1} \sum_i^n |v||x_i| \tag{39}$$

$$= \|x\|_\infty \sup_{\|x\|_\infty \leq 1} \sum_i^n |v_i| \tag{40}$$

$$= \sup_{\|x\|_\infty \leq 1} \sum_i^n |v_i| \tag{41}$$

$$= \|v\|_1 \tag{42}$$

4.6 (f)

$$\|v\|_{\infty\star} = \sup_{\|x\|_\infty \leq 1} \langle x, v \rangle \tag{43}$$

$$= \sup_{\|x\|_\infty \leq 1} \sum_i^n v_i x_i \tag{44}$$

$$= \sup_{\|x\|_\infty \leq 1} \sum_i^n |v||x_i| \tag{45}$$

$$= \|x\|_\infty \sup_{\|x\|_\infty \leq 1} \sum_i^n |v_i| \tag{46}$$

$$= \sup_{\|x\|_\infty \leq 1} \sum_i^n |v_i| \tag{47}$$

$$= \|v\|_1 \tag{48}$$

5 [Q5]

5.1 (a)

Since $v \in \partial f(x)$, we have,

$$f(y) \geq f(x) + \langle y - x, v \rangle \tag{49}$$
$$\langle y, v \rangle - f(y) \leq \langle x, v \rangle - f(x) \tag{50}$$

Take sup for y at both sides, we can get,

$$\sup_y \langle y, v \rangle - f(y) \leq \sup_y \langle x, v \rangle - f(x) \tag{51}$$
$$f^\star(v) \leq \langle x, v \rangle - f(x) \tag{52}$$
$$f^\star(v) + f(x) \leq \langle x, v \rangle \tag{53}$$

From Fenchel inequality, we can get,

$$f^\star(v) + f(x) \geq \langle x, v \rangle \tag{54}$$

Combined with (53) we can get,

$$f^\star(v) + f(x) = \langle x, v \rangle \tag{55}$$

Also, from Fenchel inequality, we can get

$$f^\star(y) + f(x) \geq \langle x, y \rangle \tag{56}$$
$$f^\star(y) \geq \langle x, v \rangle - f(x) + \langle y - v, x \rangle \tag{57}$$
$$f^\star(y) \geq f^\star(v) + \langle y - v, x \rangle \tag{58}$$

where, (58) holds because I take (55) into (57).

5.2 (b)

Let $g(x) = f^\star(x)$, since $x \in \partial g(v)$, according to conclusion from (a), we can have,

$$v \in \partial g^\star(x) \tag{59}$$

since we know $g^\star = f^{\star\star} = f$, we arrive at,

$$v \in \partial f(x) \tag{60}$$

6 [Q6]

6.1 (a)

Let's assume $d_1, d_2 \in$ set $D(x_0)$, we have,

$$d_1 : f(x_0 + t_1 d_1) \leq f(x_0), \quad \text{for some } t_1 \tag{61}$$
$$d_2 : f(x_0 + t + 2d_2) \leq f(x_0), \quad \text{for some } t_2 \tag{62}$$

Let's denote $x_1 = x_0 + t_1 d_1$ and $x_0 + t_2 d_2 = x_2$, by definition of convexity,

$$f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta) f(x_2), \theta \in [0,1] \tag{63}$$

Therefore,

$$f(x_0 + \theta t_1 d_1 + (1-\theta) t_2 d_2) \leq f(x_1) + (1-\theta) f(x_2) \tag{64}$$
$$\leq \theta f(x_0) + (1-\theta) f(x_0) \tag{65}$$
$$= f(x_0) \tag{66}$$

Although $\theta \in [0,1]$, θt_1 and $(1-\theta)t_2$ can be arbitrary number. Hence,

$$\theta t_1 d_1 + (1-\theta) t_2 d_2 \in \mathbf{D}, \quad \theta t_1, (1-\theta) t_2 \in \mathbf{R} \tag{67}$$

which indicates $\mathbf{D}$ is a convex cone.

6.2 (b)

Since $\|x\|_2^2$ is differentiable everywhere,

$$\mathbf{D}(\boldsymbol{x}_0) = \{\mathbf{d} : \langle \mathbf{d}, \nabla f(x_0) \rangle < 0\} \tag{68}$$
$$= \{\mathbf{d} : \langle \mathbf{d}, 2x_0 \rangle < 0\} \tag{69}$$

6.3 (c)

From $\|x_0 + td|_1 \leq \|x_0\|_1$, we can get,

$$\mathbf{d} = \begin{cases} -\operatorname{sign}(x_n), & \text{if } n \in \Gamma \\ 0, & \text{if } n \notin \Gamma \end{cases} \tag{70}$$

where $\Gamma = \{n : x_n \neq 0\}$.

7 [Q7]

7.1 (a)

According to KKT condtion (K1), we have,

$$(x-2)(x-4) \leq 0 \tag{71}$$

the feasible set is $\{x : 2 \leq x \leq 4\}$

7.2 (b)

The Lagrangian for this problem is,

$$L(x, \lambda) = x^2 + 1 + \lambda(x - 2)(x - 4)$$

According to KKT conditions (K4), we have,

$$\frac{\partial L}{\partial x} = 2x + 2\lambda x - 6\lambda$$

Set it to 0, we can get x as follows,

$$x = \frac{3\lambda}{\lambda + 1}$$

Lagrange dual program is

$$\underset{\lambda}{\text{maximize}} \frac{-\lambda^3 + 8\lambda^2 + 10\lambda + 1}{(\lambda + 1)^2}, \quad \text{subject to } \lambda \geq 0$$

Apprently, this will be maximize when $\lambda = 2$, which, when substituted into the dual problem yields,

$$d(\lambda^\star) = 5$$

Hence, the optimal $x^\star$ is **2**, accordingly, the value of the objective function at the minimizer is **5**.

7.3 (c)

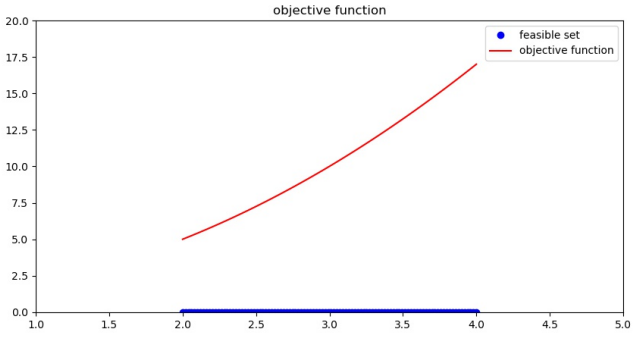

Figure 1: Plot of Objective Function

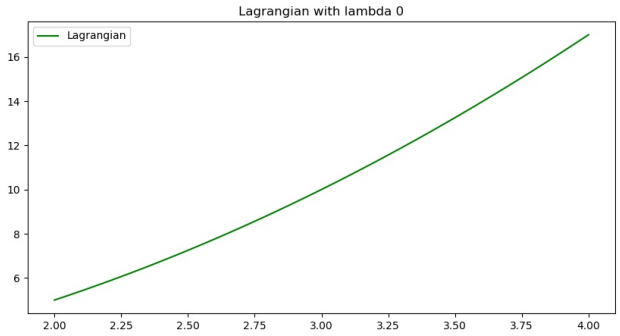

Figure 2: Plot of Lagrangian with $\lambda = 0$

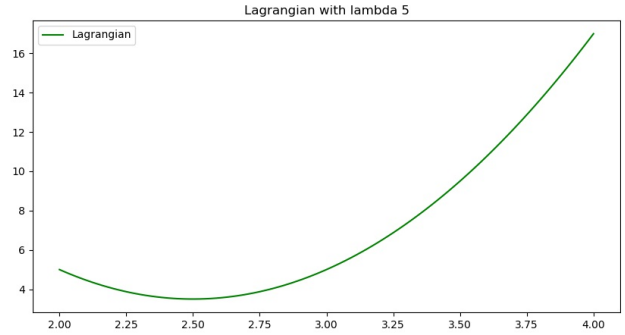

Figure 3: Plot of Lagrangian with $\lambda = 5$

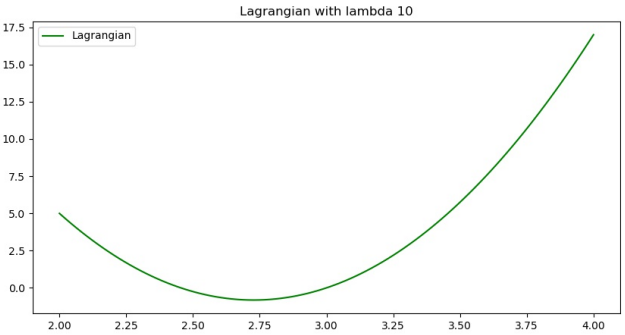

Figure 4: Plot of Lagrangian with $\lambda = 10$

7.4 (d)

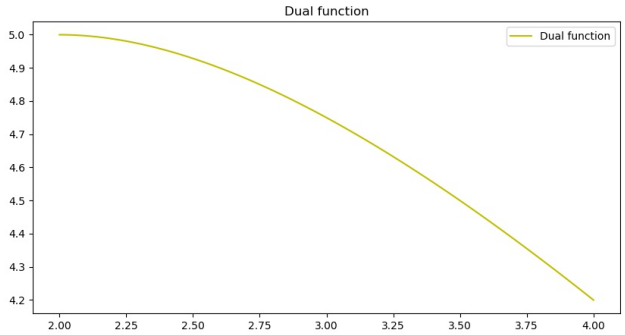

Figure 5: Plot of Dual function

7.5 (e)

The dual problem is

$$\underset{\lambda}{\text{maximize}} \frac{-\lambda^3 + 8\lambda^2 + 10\lambda + 1}{(\lambda + 1)^2}, \quad \text{subject to } \lambda \geq 0$$

And the maximizer $\lambda^\star = 2$, accordingly $d(\lambda^\star) = 5$.

Strong duality holds because the value of primal problem $p^\star$ is **5** and the value of dual problem $d^\star$ is **5**.

8 [Q8]

From (2), it is an unconstrained problem, take the derivative of x, we can have,

$$\nabla f(\boldsymbol{x}) + 2\alpha \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - b) = 0 \tag{72}$$
$$\nabla f(\tilde{\boldsymbol{x}}) + 2\alpha \boldsymbol{A}^T(\boldsymbol{A}\tilde{\boldsymbol{x}} - b) = 0 \tag{73}$$
$$\nabla f(\tilde{\boldsymbol{x}}) + \boldsymbol{A}\tilde{\boldsymbol{\upsilon}} = 0 \tag{74}$$

where $\tilde{\boldsymbol{x}}$ is the solution to (2), and $\tilde{\boldsymbol{\upsilon}} = 2\alpha(\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b})$.
From (1), it is a contrained problem, the Lagrangian is,

$$L(\boldsymbol{x}, \boldsymbol{\upsilon}) = f(\boldsymbol{x}) + \boldsymbol{\upsilon}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \tag{75}$$

Take derivative with respect to $\boldsymbol{x}$, we have,

$$\nabla f(\boldsymbol{x}) + \boldsymbol{A}\boldsymbol{\upsilon} = 0 \tag{76}$$

Note that a possible solution to (78) is exactly solution to (76) which is $\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{\upsilon}}$, and again assume the optimal value to make dual problem maximum is $\boldsymbol{\upsilon}^\star$, and $\boldsymbol{x}^\star$ is exact the value to make primal problem maximum, hence we can have,

$$d(\tilde{\boldsymbol{\upsilon}}) \leq d(\boldsymbol{\upsilon}^\star) \tag{77}$$
$$\leq f(\boldsymbol{x}^\star) \tag{78}$$

Hence, the lower bound of primal problem is

$$d(\tilde{\boldsymbol{\upsilon}}) \leq f(\boldsymbol{x}^\star) \tag{79}$$

9 [Q9]

9.1 (a)

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \upsilon) = \left[-\sum_{n=1}^{N} \log_2(\alpha_n + x_n) - \boldsymbol{\lambda}\boldsymbol{x} + \upsilon(\mathbf{1}^T\boldsymbol{x} - 1) \right]$$

The KKT conditions are as follows,

$$-x_n \leq 0, n = 1, \cdots, N \tag{80}$$
$$\mathbf{1}^T\boldsymbol{x} - 1 = 0 \tag{81}$$
$$\lambda_n \geq 0, n = 1, \cdots, N \tag{82}$$
$$-\lambda_n x_n = 0, n = 1, \cdots, N \tag{83}$$
$$-\frac{1}{\ln(2)(\alpha_n + x_n)} - \lambda_n + \upsilon = 0, n = 1, \cdots, N \tag{84}$$

From K3 and K4, we can get,

$$x_n^\star = \frac{\lambda^\star \alpha_n + \frac{1}{\ln 2}}{\upsilon^\star} - \alpha_n \tag{85}$$

9.2 (b)

We could have,

$$x_n \left(\upsilon - \frac{1}{\ln(2)(\alpha_n + x_n)} \right) = 0 \tag{86}$$

$$\upsilon \geq \frac{1}{\ln(2)(\alpha_n + x_n)} \tag{87}$$

If $\upsilon < \frac{1}{\ln(2)(\alpha_n + x_n)}$, to satisfy (77), $x_n > 0$, hence,

$$x_n = \frac{1}{\ln(2)\upsilon} - \alpha_n \tag{88}$$

If $\upsilon \geq \frac{1}{\ln(2)\alpha_n} > \frac{1}{\ln(2)(\alpha_n + x_n)}$,

$$x_n = 0 \tag{89}$$

Hence,

$$x_n^\star = \max \left\{ 0, -\alpha_n + \frac{1}{\ln(2)\upsilon^\star} \right\} \tag{90}$$

$$\lambda_n^\star = \max \left\{ 0, \upsilon^\star - \frac{1}{\ln(2)\alpha_n} \right\} \tag{91}$$

$$\sum_n \max \left\{ 0, -\alpha_n + \frac{1}{\ln(2)\upsilon^\star} \right\} = 1 \tag{92}$$

9.3 (c)

Given $\upsilon^\star$, we can use,

$$\sum_n \max \left\{ 0, -\alpha_n + \frac{1}{\ln(2)\upsilon^\star} \right\} = 1 \tag{93}$$

Then we can get $x_n^\star$ and $\lambda_n^\star$ using equations from (b).

9.4 (d)

$$\sum_n \max \left\{ 0, -\alpha_n + \frac{1}{\ln(2)\upsilon^\star} \right\} = 1 \tag{94}$$

Here is the algorithm to solve this problem,

Algorithm 1: Algorithm to find $\upsilon^\star$

Data: Set $\mathbf{D} = \{\alpha_i\}_{i=1}^N$
Result: $\upsilon^\star$

$\upsilon_1 = \frac{N}{\ln(2)(1+\sum_{i=1}^N \alpha_i)}$;
Sort $\mathbf{D}$ in ascending order;
i=1;
while $\upsilon_i \leq \frac{1}{\ln(2)\cdot\alpha_i}$ **do**
    $\alpha_i = 0$;
    $N = N - 1$;
    $i = i + 1$;
    $\upsilon_i = \frac{N}{\ln(2)(1+\sum_i \alpha_i)}$;
end

Note that in step 8, $\sum_i \alpha_i$ means you should add all elements in set **D** while $N = N - 1$.

Briefly sepeaking, in the intial state, I assume all the patch needs to be filled with water, and then iterates, in the iteration, I always remove the patch which has the highest ground level α_i until i can found the patch which satisfy $\sum_n -\alpha_n + \frac{1}{\ln(2)v^\star} = 1$.

9.5 (e)

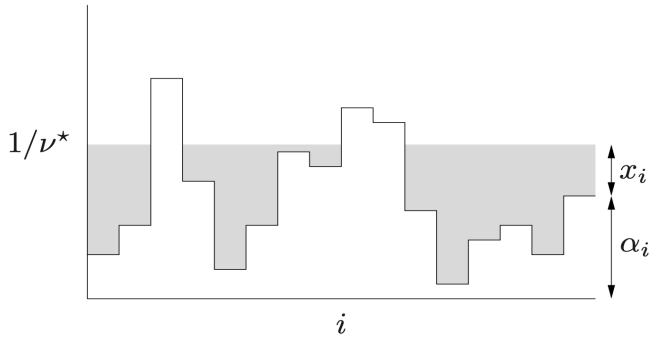

Figure 6: Relation between $\alpha_n, x_n^\star$ and $v^\star$

We think of α_i as the ground level above patch i, and then flood the region with water to a depth of $\frac{1}{v}$, as illustrated in Figure(6), The total amount of water used is then $\sum_i^n \max\left\{0, -\alpha_i + \frac{1}{\ln(2)v^\star}\right\}$. We then increase the flood level until we have used a total amount of water equal to one. The depth of water above patch i is then the optimal value $x^\star$

10 [Q10]

10.1 (a)

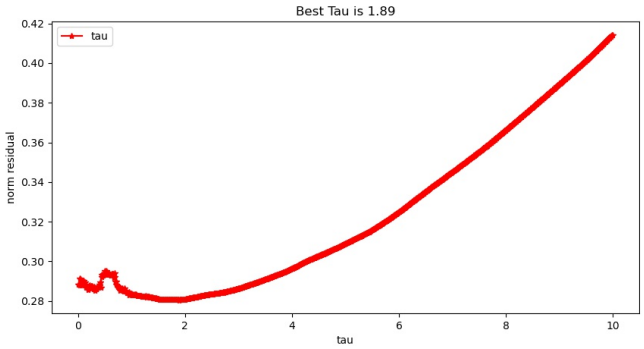

Figure 7: Best τ

Here I set τ from 0.01 to 10, and calculate the norm residual. I found the best τ is **1.89** as is indicated in Figure(7).

10.2 (b)

Note that I set initial vector as np.ones(N) in the following problems.

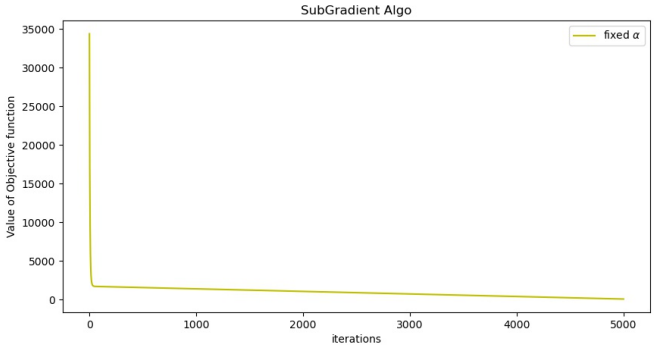

Figure 8: Subgradient Descent, fixed α

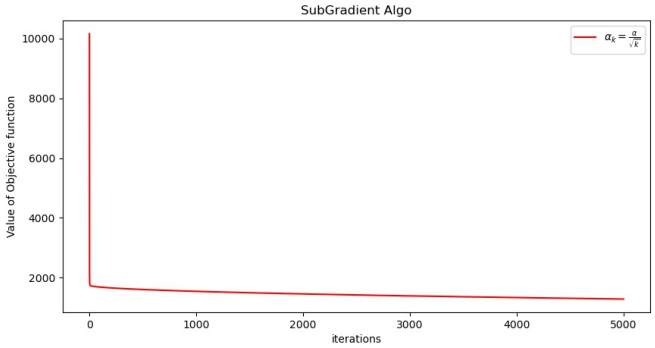

Figure 9: Subgradient Descent, $\frac{\alpha}{\sqrt{k}}$

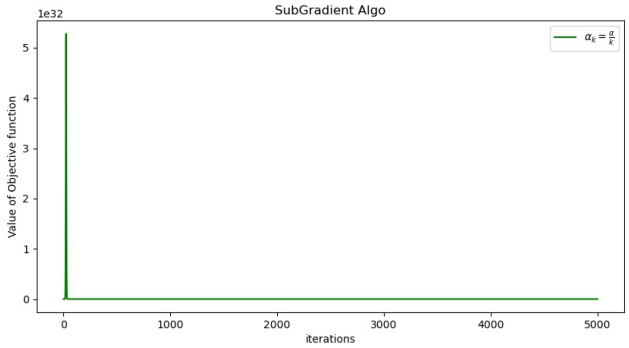

Figure 10: Subgradient Descent, $\frac{\alpha}{k}$

Here, If set α_k as fixed, it's not guaranteed that it can converge. But here it converges as is shown in Figure(8). Here I set α as 0.0001. And If we set $\alpha_k = \frac{\alpha}{\sqrt{k}}$ and $\alpha_k = \frac{\alpha}{k}$, it converges as shown in Figure(9) and Figure(10). For $\frac{\alpha}{\sqrt{k}}$, I set α as 0.001, and for $\frac{\alpha}{k}$, I set α as 0.03. Apprently, for all the cases, it needs more than 5000 iterations to converge at a better result.

10.3 (c)

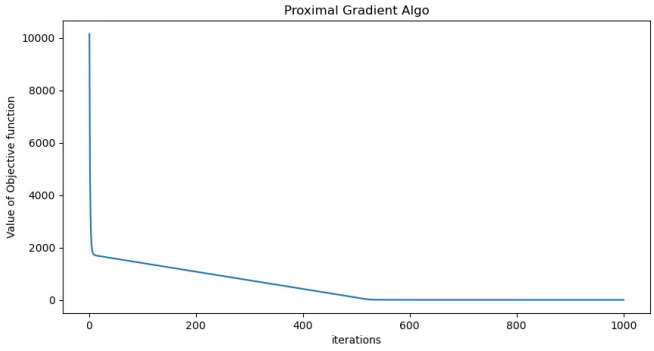

Figure 11: Proximal Gradient Descent

10.4 (d)

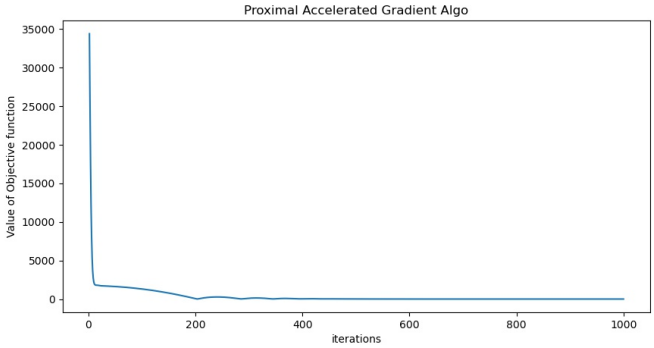

Figure 12: Proximal Gradient Descent with acceleration

11 Appendix

All the source code can be found from my personal Github.

11.1 Q7

```
import numpy as np
from matplotlib import pyplot as plt

def func(x):
    return x**2 + 1

def contraint(x):
    return (x-2) * (x-4)

def lagrange(x, lam):
    return func(x) + lam * contraint(x)

def dual(x):
    numerator = -x**3 + 8 * x**2 + 10 * x + 1
    denumerator = (x + 1)**2
    return numerator / denumerator
```

```
def plotObj(x, y):
    plt.figure(figsize=(10, 5))
    plt.plot(x, np.zeros(len(x)), 'bo', label='feasible set')
    plt.plot(x, y, 'r', label='objective function')
    plt.title('objective function')
    plt.xlim([1.0, 5.0])
    plt.ylim([-0.01, 20])
    plt.legend()
    plt.savefig('./hw05/7c.jpg')

def plotLagrange(x, y, lam):
    plt.figure(figsize=(10, 5))
    plt.plot(x, y, 'g', label='Lagrangian')
    plt.title('Lagrangian with lambda {}'.format(lam))
    plt.legend()
    plt.savefig('./hw05/7c-lam{}.jpg'.format(lam))

def plotDual(x, y):
    plt.figure(figsize=(10, 5))
    plt.plot(x, y, 'y', label='Dual function')
    plt.title('Dual function')
    plt.legend()
    plt.savefig('./hw05/7d.jpg')


if __name__ == '__main__':
    x = np.linspace(2,4,100)
    y = func(x)
    plotObj(x, y)
    for lam in [0, 5, 10]:
        L = lagrange(x, lam)
        plotLagrange(x, L, lam)
    d = dual(x)
    plotDual(x, d)
```

11.2 Q10

```
import numpy as np
import cvxpy as cp
import random
import os
from matplotlib import pyplot as plt

def Q10_a():
    res = []
    for tau in np.arange(0.01, 10, 0.01):
        x = cp.Variable(N)
        cost = 0.5 * cp.sum_squares(y - A @ x) + tau * cp.norm(x, 1)
        prob = cp.Problem(cp.Minimize(cost))
        prob.solve()
        res.append(cp.norm(x.value - x0, p=2).value)
    optimal_tau = (res.index(min(res)) + 1) /100
    return res, optimal_tau

def funcValue(tau, A, x, y):
    var = y - A @ x
    return 0.5 * np.linalg.norm(x=var, ord=2) ** 2 + tau * np.linalg.norm
                                        (x=x, ord=1)
```

```
def subGradient(alpha, tau, A, x, y, maxiter, delta, flag):
    k = 1
    x = np.mat(x).T #(N ,1)
    y = np.mat(y).T #(M, 1)
    A = np.mat(A) #(M, N)
    value = []
    iteration = []
    while (k < maxiter) and (np.linalg.norm(x - np.mat(x0)) > delta):
        dk = A.T @ (A @ x - y) + tau * np.sign(x) #(N, 1)
        if flag == 'fixed':
            x = x - alpha * dk # (N, 1)
        elif flag == 'k':
            x = x - alpha * dk / k
        else:
            x = x - alpha * dk / np.sqrt(k) # (N, 1)

        value.append(funcValue(tau, A, x, y))
        iteration.append(k)
        k += 1
        print(x.flatten())
    return x, value, iteration

def Proximal(alpha, tau, A, x, y, maxiter, delta):
    def T(x):
        for i in range(len(x)):
            if x[i] >= tau * alpha:
                x[i] = x[i] - tau * alpha
            elif abs(x[i]) <= tau * alpha:
                x[i] = 0
            elif x[i] <= - tau * alpha:
                x[i] = x[i] + tau * alpha
            else:
                return -1
        return x
    k = 0
    x = np.mat(x).T #(N ,1)
    y = np.mat(y).T #(M, 1)
    A = np.mat(A) #(M, N)
    fValue = []
    iteration = []
    while (k < maxiter) and (np.linalg.norm(x - np.mat(x0)) > delta):
        x = T(x + alpha * A.T @ (y - A @ x))
        k += 1
        print(x.flatten())
        fValue.append(funcValue(tau, A, x, y))
        iteration.append(k)
    return x, fValue, iteration


def accrProximal(alpha, tau, A, x, y, maxiter, delta):
    def Tec(x):
        for i in range(len(x)):
            if x[i] >= tau * alpha:
                x[i] = x[i] - tau * alpha
            elif abs(x[i]) <= tau * alpha:
                x[i] = 0
            elif x[i] <= - tau * alpha:
                x[i] = x[i] + tau * alpha
            else:
                return -1
```

```
            return x
        k = 1
        pk = np.mat(np.zeros(N)).T #(N, 1)

        x = [np.mat(x).T] #(N ,1)
        y = np.mat(y).T #(M, 1)
        A = np.mat(A) #(M, N)

        gd = A.T @ (A @ (x[-1] + pk) - y) #(N, 1)
        fValue = []
        iteration = []
        while (k < maxiter) and (np.linalg.norm(x[-1] - np.mat(x0)) > delta):
            x.append(Tec(x[-1] + pk - alpha * gd))
            k += 1
            pk = ((k - 1) / (k + 2)) * (x[-1] - x[-2])
            gd = A.T @ (A @ (x[-1] + pk) - y)

            print(x[-1].flatten())
            fValue.append(funcValue(tau, A, x[-1], y))
            iteration.append(k)
        return x[-1], fValue, iteration

if __name__ == '__main__':
    np.random.seed(2021) # Set random seed so results are repeatable #
                                                Set parameters
    M = 100
    N = 1000
    S = 10
    # Define A and y
    A = np.random.randn(M, N) # (M, N)
    ind0 = np.random.choice(N, S, 0)
    x0 = np.zeros(N)
    x0[ind0] = np.random.rand(S) # (N, )
    y = A@x0 + .25*np.random.randn(M) # (M, )
    savepath = 'C:/gexueren/Desktop/hw05/hw05/hw05'

    ######################### Q1 ########################
    res, optimal_tau = Q10_a()
    plt.figure(figsize=(10,5))
    plt.plot(np.arange(0.01,10, 0.01), res, 'r-*', label='tau')
    plt.title('Best Tau is {}'.format(optimal_tau))
    plt.xlabel('tau')
    plt.ylabel('norm residual')
    plt.legend()
    plt.savefig(os.path.join(savepath, 'Q10_1.jpg'))

    ######################### Q2 ########################
    x = np.ones(N) # initial value (N, )
    _, fValue_0, iteration_0 = subGradient(alpha=0.001, tau=optimal_tau,
                                           A=A, x=x, y=y, maxiter=5000,
                                           delta=1e-3, flag='sqrt(k)')
    plt.figure(figsize=(10, 5))
    plt.plot(iteration_0, fValue_0, color='r', label=r'$\alpha_{k}= \frac
                                           {\alpha}{\sqrt{k}}$')
    plt.title('SubGradient Algo')
    plt.xlabel('iterations')
    plt.ylabel('Value of Objective function')
    plt.legend()
    plt.savefig(os.path.join(savepath, 'Q10_2_1.jpg'))

    x = np.ones(N) # initial value (N, )
```

```
    _, fValue_1, iteration_1 = subGradient(alpha=0.03, tau=optimal_tau, A
                                           =A, x=x, y=y, maxiter=5000,
                                           delta=1e-3, flag='k')
    plt.figure(figsize=(10, 5))
    plt.plot(iteration_1, fValue_1, color='g', label=r'$\alpha_{k}= \frac
                                           {\alpha}{k}$')
    plt.title('SubGradient Algo')
    plt.xlabel('iterations')
    plt.ylabel('Value of Objective function')
    plt.legend()
    plt.savefig(os.path.join(savepath, 'Q10_2_2.jpg'))

    x = np.ones(N) # initial value (N, )
    _, fValue_2, iteration_2 = subGradient(alpha=0.03, tau=optimal_tau, A
                                           =A, x=x, y=y, maxiter=5000,
                                           delta=1e-3, flag='fixed')
    plt.figure(figsize=(10, 5))
    plt.plot(iteration_2, fValue_2, color='y', label=r'fixed $\alpha$')
    plt.title('SubGradient Algo')
    plt.xlabel('iterations')
    plt.ylabel('Value of Objective function')
    plt.legend()
    plt.savefig(os.path.join(savepath, 'Q10_2_3.jpg'))

    ######################## Q3 ########################
    x = np.ones(N) # initial value (N, )
    _, fValue, iteration = Proximal(alpha=0.001, tau=optimal_tau, A=A, x=
                                           x, y=y, maxiter=1000, delta=1e-3
                                           )
    plt.figure(figsize=(10, 5))
    plt.plot(iteration, fValue)
    plt.title('Proximal Gradient Algo')
    plt.xlabel('iterations')
    plt.ylabel('Value of Objective function')
    plt.savefig(os.path.join(savepath, 'Q10_3.jpg'))

    ######################## Q4 ########################
    x = np.ones(N) # initial value (N, )
    _, fValue, iteration = accrProximal(alpha=0.0001, tau=optimal_tau, A=
                                           A, x=x, y=y, maxiter=1000, delta
                                           =1e-3)
    plt.figure(figsize=(10, 5))
    plt.plot(iteration, fValue)
    plt.title('Proximal Accelerated Gradient Algo')
    plt.xlabel('iterations')
    plt.ylabel('Value of Objective function')
    plt.savefig(os.path.join(savepath, 'Q10_4.jpg'))
```