EE 364a Final

Yen-Yu Chang

March 15, 2020

1.

(a) We can formulate the problem as

$$\begin{array}{ll} \text{minimize} & ||x||_1 \\ \text{subject to} & y = ABx \end{array}$$

(b) We can formulate the problem as

$$\begin{array}{ll} \text{minimize} & ||x||_2 \\ \text{subject to} & y = ABx \end{array}$$

(c) We can formulate the problem as

$$\begin{array}{ll} \text{minimize} & ||x||_\infty \\ \text{subject to} & y = ABx \end{array}$$

(d) The following Python code solves the problems above:

```
import numpy as np
import cvxpy as cp

import matplotlib.pyplot as plt

from deconvolution_data import *

x_a = cp.Variable(n)
x_b = cp.Variable(n)
x_c = cp.Variable(n)

obj_a = cp.Minimize(cp.norm(x_a, 1))
obj_b = cp.Minimize(cp.norm(x_b, 2))
```

```
obj_c = cp.Minimize(cp.norm(x_c, 'inf'))

constraints_a = [Y == A @ B @ x_a]
constraints_b = [Y == A @ B @ x_b]
constraints_c = [Y == A @ B @ x_c]

prob_a = cp.Problem(obj_a, constraints_a)
prob_b = cp.Problem(obj_b, constraints_b)
prob_c = cp.Problem(obj_c, constraints_c)

prob_a.solve(solver = cp.SCS)
prob_b.solve(solver = cp.SCS)
prob_c.solve(solver = cp.SCS)

z_a = B @ x_a.value
z_b = B @ x_b.value
z_c = B @ x_c.value

plt.figure()
plt.imshow(np.reshape(z_a, (d, d)).T, cmap = 'gray', interpolation = 'nearest')
plt.savefig('1a.png')

plt.figure()
plt.imshow(np.reshape(z_b, (d, d)).T, cmap = 'gray', interpolation = 'nearest')
plt.savefig('1b.png')

plt.figure()
plt.imshow(np.reshape(z_c, (d, d)).T, cmap = 'gray', interpolation = 'nearest')
plt.savefig('1c.png')

plt.figure()
plt.imshow(np.reshape(Y, (d, d)).T, cmap = 'gray', interpolation = 'nearest')
plt.savefig('1y.png')
```

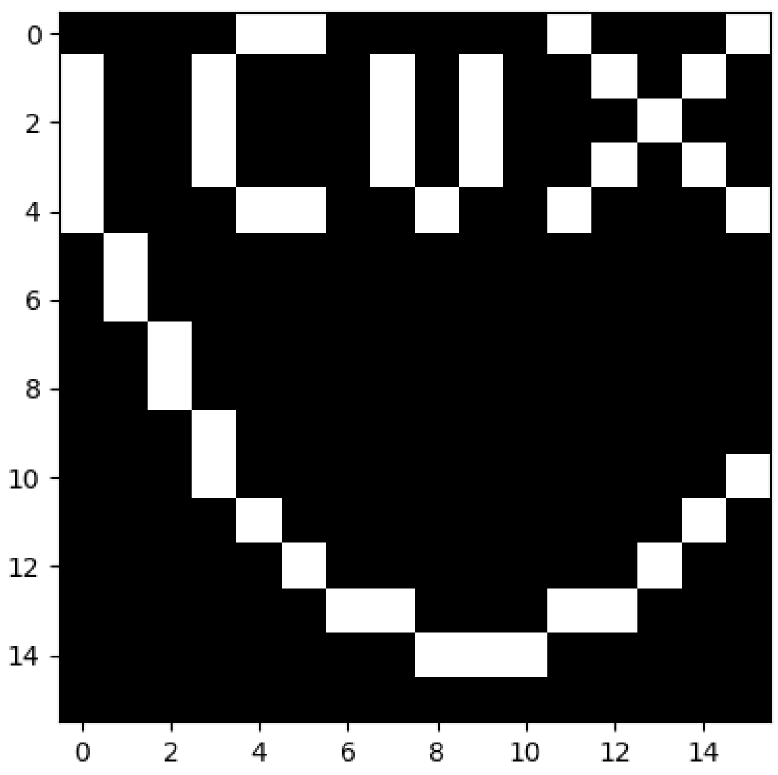


Figure 1: Figure (a)

For (a), the x^* is sparse, so the estimated image $z = Bx^*$ is also sparse. From the figure (a), we can see that the color for each pixel is only black or white, which means that z is sparse.

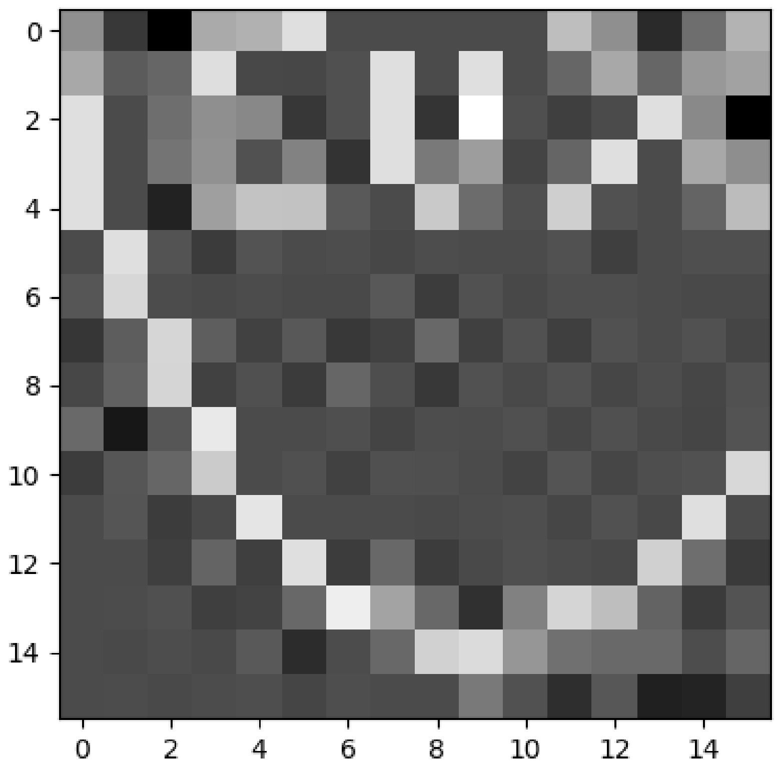


Figure 2: Figure (b)

For (b), the x^* is more continuous than (a), so the figure (b), which is the estimated image $z = Bx^*$, is more continuous than figure (a). There are more gray pixels in figure (b).

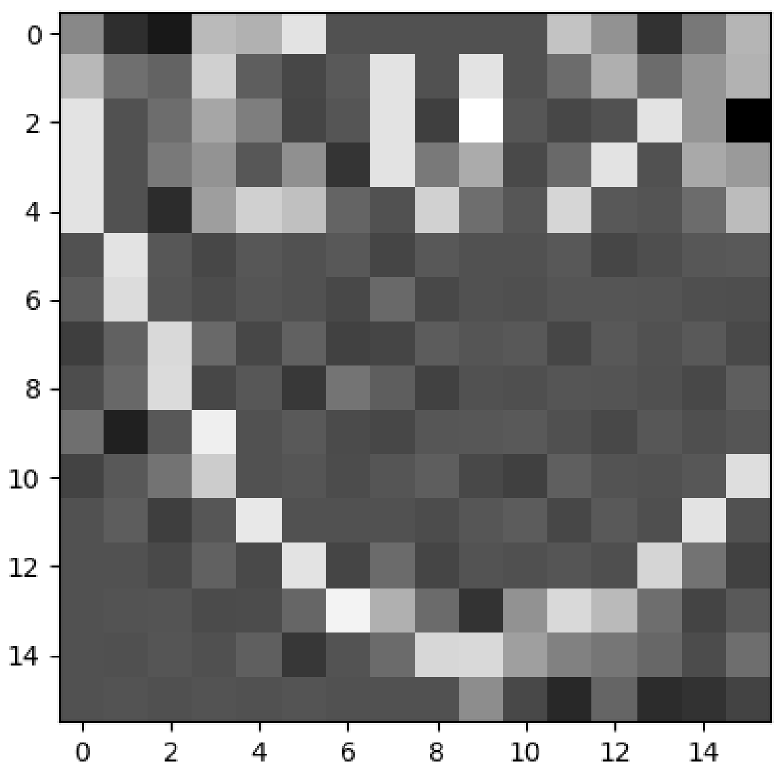


Figure 3: Figure (c)

For (c), each components in x^* has almost the same absolute value, which means that $|x_i^*| = \text{ Constant}, i = 1, \ldots, n$. The figure (c) also has gray pixels, but is not as clear nor smooth as the figure (b).

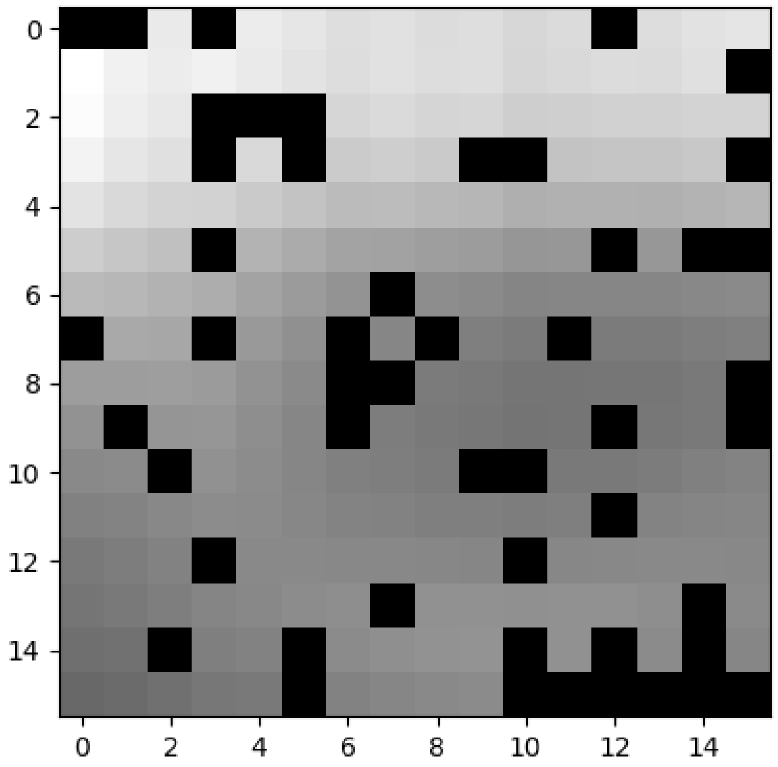


Figure 4: Sensed image y

2.

(a) If we want to check whether we can make $L_T = 0$ at time T, we can formulate the problem as:

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & L_{t+1} = L_t - P_t, \quad t = 1, \dots, T-1 \\ & c_{t+1} = c_t - P_t\mathbf{1} + P_t^T\mathbf{1}, \quad t = 1, \dots, T-1 \\ & P_t\mathbf{1} \preceq c_t, \quad L_1 = L_{1,\text{given}}, \quad c_1 = c_{1,\text{given}}, \quad L_T = 0 \end{array}$$

The convex problem above is a feasibility problem. If we want to find the minimum T for which there is a feasible sequence of Payments $P_1, \dots, P_{T-1}$ that results in $L_T = 0$, we can run a for loop from $T = 1$ to ∞ to check whether the problem is feasible (problem.status == 'optimal') for a given T. If the problem is feasible, which also means that problem.status == 'optimal', we can break the loop and report the minimum T.

(b) The minimum $T = 5$. The following Python code solves the problem.

```
# basic data for multi-period liability clearing problem

import numpy as np
import cvxpy as cp

n = 7
L1 = np.array(
    [[0., 90, 0, 3, 79, 0, 0],
     [57, 0, 69, 37, 0, 94, 56],
     [79, 53, 0, 0, 0, 0, 0],
     [0, 0, 1, 0, 0, 73, 20],
     [0, 0, 42, 0, 0, 0, 90],
     [0, 34, 0, 0, 13, 0, 0],
     [38, 0, 0, 94, 85, 22, 0]]
)
c1 = np.array([10., 146., 30., 10., 10., 10., 83.])

np.testing.assert_array_less(L1 @ np.ones(n) - L1.T @ np.ones(n), c1)

for T in range(1, 100):
  P = []
  L = []
  c = []

  for i in range(T):
```

```
  P.append(cp.Variable((n, n)))

for i in range(T + 1):
  L.append(cp.Variable((n, n,)))

for i in range(T + 1):
  c.append(cp.Variable(n))

constraints = [L[0] == L1, c[0] == c1]

for t in range(T):
  constraints += [L[t + 1] == L[t] - P[t]]
  constraints += [c[t + 1] == c[t] - P[t] @ np.ones(n) + P[t].T @ np.ones(n)]
  constraints += [P[t] @ np.ones(n) <= c[t]]

for t in range(T + 1):
  constraints += [c[t] - L[t] @ np.ones(n) + L[t].T @ np.ones(n) >= 0]

constraints += [L[-1] == 0]

obj = cp.Minimize(0)
prob = cp.Problem(obj, constraints)
prob.solve(solver = cp.SCS)
if prob.status != 'infeasible':
  print("Minimum T is: {}".format(T + 1))
  break
```

3.

(a) The CDF of Y is

$$F_Y(y) = P(Y \le y) = P(\sqrt{X} \le y) = P(X \le y^2) = F_X(y^2).$$

Then we have the density function of Y as

$$p_Y(y) = \frac{dF_Y(y)}{dy} = p_X(y^2)\frac{dx}{dy}.$$

Since $y = \sqrt{x}$, we have $\frac{dx}{dy} = 2y$. Hence $p_Y(y) = 2yp_X(y^2)$ Then we get

$$\nabla_y p_Y(y) = 4y^2 p'_X(y^2) + 2p_X(y^2)$$

$$\nabla_y^2 p_Y(y) = 8y^3 p''_X(y^2) + 12yp'_X(y^2)$$

$$p_Y(y) \cdot \nabla_y^2 p_Y(y) = 16y^4 p''_X(y^2) \cdot p_X(y^2) + 24y^2 p'_X(y^2) \cdot p_X(y^2)$$

$$(\nabla_y p_Y(y))^2 = 16y^4 \left(p'_X(y^2)\right)^2 + 16y^2 p'_X(y^2) p_X(y^2) + 4\left(p_X(y^2)\right)^2$$

And we calculate $(\nabla_y p_Y(y))^2 - p_Y(y) \cdot \nabla_y^2 p_Y(y)$ as:

$$(\nabla_y p_Y(y))^2 - p_Y(y) \cdot \nabla_y^2 p_Y(y) = 16[\left(p'_X(y^2)\right)^2 - p''_X(y^2) \cdot p_X(y^2)] - 8y^2 p'_X(y^2) \cdot p_X(y^2) + 4\left(p_X(y^2)\right)^2$$

Since p_X is log-concave and decreasing, we have

$$[\left(p'_X(y^2)\right)^2 - p''_X(y^2) \cdot p_X(y^2)] \ge 0$$

and $p'_X(y^2) < 0$. Therefore, $(\nabla_y p_Y(y))^2 - p_Y(y) \cdot \nabla_y^2 p_Y(y) \ge 0$, which means that Y has a log-concave density.

(b) Z is not log-concave. There is a counter example as follow: Let

$$p_X(x) = \begin{cases} -x + \sqrt{2}, & 0 \le x \le \sqrt{2} \\ 0, & \text{otherwise} \end{cases}$$

Then we have

$$\nabla_x p_X(x) = \begin{cases} -1, & 0 \le x \le \sqrt{2} \\ 0, \text{otherwise} \end{cases}$$

$$\nabla_x^2 p_X(x) = 0$$

$$\int_0^\infty p_X(x)dx = \int_0^{\sqrt{2}} (-x + \sqrt{2})dx = 1$$

We see that p_X is log-concave since $(\nabla_x p_X(x))^2 \geq \nabla_x^2 p_X(x) \cdot p_X(x)$, decreasing, and the area is 1. The CDF of Z is

$$F_Z(z) = P(Z \leq z) = P(X^2 \leq z) = P(X \leq \sqrt{z}) = F_X(\sqrt{z}) = \int_0^{\sqrt{z}} (-x+\sqrt{2})dx = \frac{-z}{2}+\sqrt{2}z^{\frac{1}{2}}$$

So,

$$p_Z(z) = \frac{dF_Z(z)}{dz} = \frac{-1}{2} + \frac{\sqrt{2}}{2} z^{\frac{-1}{2}}$$

$$\nabla_z p_Z(z) = \frac{-\sqrt{2}}{4} z^{\frac{-3}{2}}$$

$$\nabla_z^2 p_Z(z) = \frac{3\sqrt{2}}{8} z^{\frac{-5}{2}}$$

Let $z = 0.5$ and calculate $p_Z(z) \cdot \nabla_z^2 p_Z(z)$ and $(\nabla_z p_Z(z))^2$ as

$$p_Z(0.5) \cdot \nabla_z^2 p_Z(0.5) = \frac{3}{8} 0.5^{-3} - \frac{3\sqrt{2}}{16} 0.5^{-2.5} = 1.5$$

$$(\nabla_z p_Z(0.5))^2 = \frac{1}{8} 0.5^{-3} = 1$$

We see that $(\nabla_z p_Z(0.5))^2 \not\geq p_Z(0.5) \cdot \nabla_z^2 p_z(0.5)$. Therefore, Z is not log-concave.

4.

(a)

$$\text{minimize} \quad \tfrac{\lambda}{2}\sum_{i=1}^{m} \|x - \mathbf{prox}_{f_i}^{\lambda}(x)\|_2^2 + \sum_{i=1}^{m} f_i(\mathbf{prox}_{f_i}^{\lambda}(x))$$

We introduce new variables by the definition of proximal mapping

$$z_i = \mathbf{prox}_{f_i}^{\lambda}(x), \quad i = 1, \ldots, m$$

$$f_i(z_i) + \frac{\lambda}{2}\|z_i - x\|_2^2 \le t_i, \quad i = 1, \ldots, m$$

Then we can express the problem as

$$\begin{array}{ll} \text{minimize} & \tfrac{\lambda}{2}\sum_{i=1}^{m} \|x - z_i\|_2^2 + \sum_{i=1}^{m} f_i(z_i) \\ \text{subject to} & f_i(z_i) + \tfrac{\lambda}{2}\|z_i - x\|_2^2 \le t_i, \quad i = 1, \ldots, m \end{array}$$

Since $\|\cdot\|_2^2$ is convex and $x - z_i$ is affine, $\sum_{i=1}^{m} \|x - z_i\|_2^2$ is convex. Since f_i are convex functions, the objective function $\frac{\lambda}{2}\sum_{i=1}^{m} \|x - z_i\|_2^2 + \sum_{i=1}^{m} f_i(z_i)$ is convex. And the constraints $f_i(z_i) + \frac{\lambda}{2}\|z_i - x\|_2^2 \le t_i$ are convex for $i = 1, \ldots m$. Thus, it is a convex optimization problem.

(b) Let

$$g_i(y) = f_i(y) + \frac{\lambda}{2}\|y - x\|_2^2, \quad i = 1, \ldots, m$$

$g_i(y)$ has minimum when

$$\nabla_y g_i(y) = \nabla_y (f_i(y) + \frac{\lambda}{2}\|y - x\|_2^2) = 0.$$

And we know that

$$f_i(y) = \frac{1}{2}(a_i^T y - b_i)^2$$

Thus

$$\nabla_y (f_i(y) + \frac{\lambda}{2}\|y - x\|_2^2) = (a_i^T y - b_i)a_i + \lambda(y - x)$$

which implies that

$$y = (a_i a_i^T + \lambda I)^{-1}(\lambda x + b_i a_i).$$

Therefore,

$$\mathbf{prox}_{f_i}^{\lambda}(x) = (a_i a_i^T + \lambda I)^{-1}(\lambda x + b_i a_i)$$

and we reformulate $h(x)$ as

$$\begin{aligned} h(x) &= \frac{\lambda}{2}\sum_{i=1}^{m} \|x - \mathbf{prox}_{f_i}^{\lambda}(x)\|_2^2 + \sum_{i=1}^{m} f_i(\mathbf{prox}_{f_i}^{\lambda}(x)) \\ &= \frac{\lambda}{2}\sum_{i=1}^{m} \|x - (a_i a_i^T + \lambda I)^{-1}(\lambda x + b_i a_i)\|_2^2 + \sum_{i=1}^{m} \left(a_i^T (a_i a_i^T + \lambda I)^{-1}(\lambda x + b_i a_i) - b_i\right)^2 \end{aligned}$$

Therefore, the original optimization problem becomes:

$$\text{minimize} \quad \tfrac{\lambda}{2}\textstyle\sum_{i=1}^{m} \|x - (a_i a_i^T + \lambda I)^{-1}(\lambda x + b_i a_i)\|_2^2 + \sum_{i=1}^{m} \left(a_i^T (a_i a_i^T + \lambda I)^{-1}(\lambda x + b_i a_i) - b_i\right)^2$$

which is a quardratic program with the single variable x.

5.

(a) Assume that there is a consistent score, the problem can be formulated as a feasibility problem. Hence, we can find s by solving the following problem:

$$\begin{array}{ll}\text{minimize} & 0 \\ \text{subject to} & s_{i^j_{idx}} \geq s_{i^j_{idx+1}} + \epsilon,\ j = 1, \ldots, m,\ idx = 1, \ldots, k-1,\ i \in \{1, \ldots, n\}\end{array}$$

Since we can not solve convex problems with strict inequalities, we have to reformulate $s_{i^j_{idx}} > s_{i^j_{idx+1}}$ as $s_{i^j_{idx}} \geq s_{i^j_{idx+1}} + \epsilon$, where ϵ is a parameter. We can just set $\epsilon = 0.001$.

(b) The ordering is 23, 1, 13, 9, 3, 5, 24, 17, 21, 12, 16, 11, 4, 19, 20, 8, 2, 10, 7, 14, 6, 25, 18, 22, 15. The following Python code solves the problem:

```
import numpy as np
import cvxpy as cp

from ranked_lists_data import *

s = cp.Variable(n)

obj = cp.Minimize(0)
constraints = []

for i in range(Sigma.shape[1]):
  for x in range(0, Sigma.shape[0] - 1):
    constraints += [s[Sigma[x, i] - 1] >= s[Sigma[x + 1, i] - 1] + 1e-3]

prob = cp.Problem(obj, constraints)
prob.solve(solver = cp.SCS)
print("The ordering is: {}".format(np.argsort(-s.value) + 1))
```

(c) We can define a new score function ϕ for each ranking σ^j as:

$$\phi(\sigma^j) = \max\left\{(s_{i^j_2} - s_{i^j_1})_+, \ldots, (s_{i^j_k} - s_{i^j_{k-1}})_+\right\}$$

where $(u)_+ = \max(0, u)$. $\phi(\sigma^j) = 0$ if σ^j is consistent with the score list s, but $\phi(\sigma^j) > 0$ when σ^j is inconsistent with the score list. We want to satisfy ranking lists as many as possible, so we have to minimize the total new score function, which can be formulated as the following convex optimization problem:

$$\begin{array}{ll}\text{minimize} & \sum_{j=1}^m \phi(\sigma^j) \\ \text{subject to} & \phi(\sigma^j) = \max\left\{(s_{i_2^j} - s_{i_1^j})_+, \ldots, (s_{i_k^j} - s_{i_{k-1}^j})_+\right\} \\ & \qquad\qquad \text{for } i \in \{1, \ldots, m\}, \quad j = 1, \ldots, m\end{array}$$

(d) The ordering is 23, 1, 13, 9, 3, 5, 24, 17, 21, 12, 19, 11, 4, 16, 20, 8, 2, 10, 7, 14, 6, 25, 18, 22, 15. The number of inconsistent lists is 3. The following Python code solves the problem:

```
import numpy as np
import cvxpy as cp

from ranked_lists_inconsistent_data import *

obj = 0
s = cp.Variable(n)
s_matrix = cp.Variable((k, m))

for i in range(m):
  obj += cp.pos(cp.max(s_matrix[1:, i] - s_matrix[:-1, i]))
obj = cp.Minimize(obj)
constraints = []
for i in range(k):
  for j in range(m):
    constraints += [s_matrix[i, j] == s[Sigma[i, j] - 1]]

prob = cp.Problem(obj, constraints)
prob.solve(solver = cp.ECOS)
print("The ordering is {}".format(np.argsort(-s.value) + 1))
print(s.value)

idx2pos = {}
for i in range(len(s.value)):
  idx2pos[np.argsort(-s.value)[i] + 1] = i

total = 0
for i in range(m):
  for j in range(k - 1):
    if idx2pos[Sigma[j + 1, i]] < idx2pos[Sigma[j, i]]:
      total += 1
      break

print("The number of inconsistent lists is: {}".format(total))
```

6.

(a)

$$f_m(x) = \frac{1}{m}\sum_{i=1}^{m} d_C(x, w_i) = \frac{1}{m}\sum_{i=1}^{m} \frac{\sup_{y\in C} |w_i^T(y-x)|}{||w_i||_2} = \frac{1}{m}\sum_{i=1}^{m} \frac{|\sigma_C(w_i) - w_i^T x|}{\|w_i\|_2}$$

We can formulate the problem of minimizing f_m as:

$$\text{minimize} \quad \frac{1}{m}\sum_{i=1}^{m} \frac{|\sigma_C(w_i)-w_i^T x|}{\|w_i\|_2}$$

The support function $\sigma_C(w)$ is convex, $w_i^T x$ is affine, so $\sum_{i=1}^m |\sigma_C(w_i) - w_i^T x|$ is convex. It is a convex optimization problem.

(b) Yes, $x^* = 0$ is a minimizer of f. Since C is convex and symmetric, we have that $x^* = 0 \in C$. Suppose now $x^* = \Delta$ and $\Delta \neq 0$. Since C is symmetric, we have $-\Delta \in C$. By triangle inequality, we have

$$|w_i^T(y-\Delta)| \le |w_i^T y| + |w_i^T(-\Delta)|$$
$$|w_i^T(y+\Delta)| \le |w_i^T y| + |w_i^T(+\Delta)|$$

So Δ must to be 0 such that $x^* = \Delta = 0$ satisfies the equalities of the above inequalities to achieve the minimum. Therefore, $x^* = 0$ is a minimizer of f.

(c) The problem can be expressed as:

$$\begin{array}{ll} \text{minimize} & \frac{1}{m}\sum_{i=1}^{m} \frac{\sup_{y\in C} |w_i^T(y-x)|}{\|w_i\|_2} \\ \text{subject to} & Ay \preceq b \\ & Ax \preceq b \end{array}$$

We introduce new variables

$$t_i = \frac{\sup_{y\in C} |w_i^T(y-x)|}{\|w_i\|_2}, \quad i = 1, \dots, m$$

And we know that $t_i \ge 0$ for $i = 1, \dots, m$. Then the problem can be reformulated as

$$\begin{array}{ll} \text{minimize} & \frac{1}{m}\sum_{i=1}^{m} t_i \\ \text{subject to} & Ax \preceq b \\ & Ay \preceq b \\ & t_i \ge 0, \quad i = 1, \dots, m \\ & w_i^T(y-x) \le t_i\|w_i\|_2, \quad i = 1, \dots, m \\ & w_i^T(y-x) \ge -t_i\|w_i\|_2, \quad i = 1, \dots, m \end{array}$$

It is a linear program since its objective function and all constraints are affine.

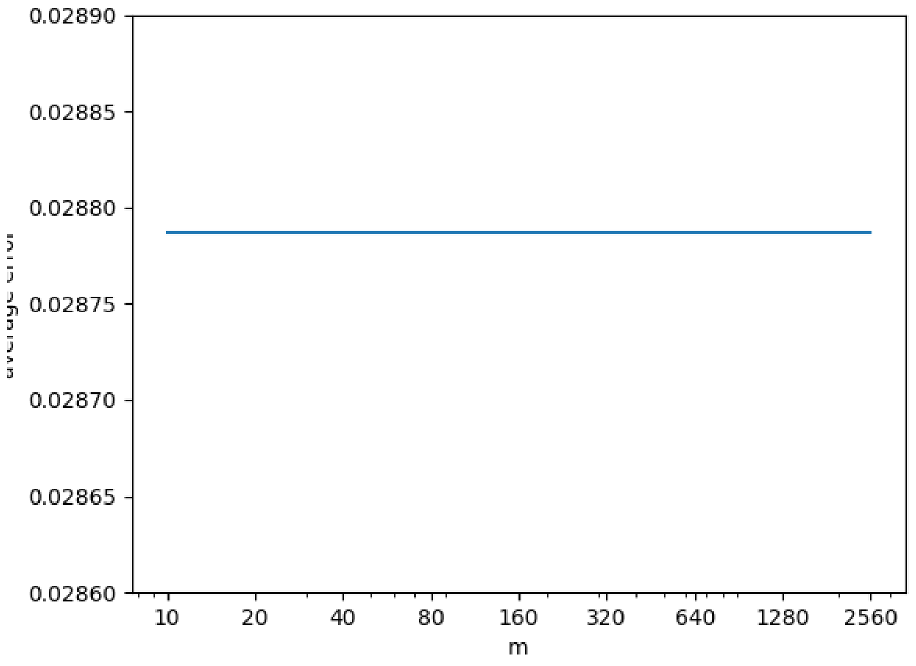
(d) The linear program becomes as :

$$\begin{array}{ll} \text{minimize} & \frac{1}{m}\sum_{i=1}^{m} t_i \\ \text{subject to} & x \succeq 0 \\ & y \succeq 0 \\ & \mathbf{1}^T x \leq 1 \\ & \mathbf{1}^T y \leq 1 \\ & t_i \geq 0, \quad i = 1, \ldots, m \\ & w_i^T(y - x) \leq t_i \|w_i\|_2, \quad i = 1, \ldots, m \\ & w_i^T(y - x) \geq -t_i \|w_i\|_2, \quad i = 1, \ldots, m \\ & \|w_i\| = 1, \quad i = 1, \ldots, m \end{array}$$

The following Python code solves the problem:

```
import numpy as np
import cvxpy as cp
import matplotlib.pyplot as plt

m_list = [10, 20, 40, 80, 160, 320, 640, 1280, 2560]
n = 10
error_list = []

iterations = 20

for m in m_list:
  error = 0
  for iteration in range(iterations):
    w = np.random.uniform(-1, 1, (n, m))
    w = w / np.linalg.norm(w, axis = 0)

    x = cp.Variable(n)
    y = cp.Variable(n)
    t = cp.Variable(m)

    obj = cp.Minimize(cp.sum(t) / m)
    constraints = [t >= 0,\
  x >= 0,\
  y >= 0,\
  np.ones(n).T @ x <= 1, np.ones(n).T @ y <= 1]
    for i in range(m):
      constraints += [w[:, i].T @ (y - x) <= t[i]]
      constraints += [w[:, i].T @ (y - x) >= -t[i]]

    prob = cp.Problem(obj, constraints)
    prob.solve(solver = cp.SCS)
    error += (np.linalg.norm(x.value - (np.ones(n) / n)))
  error /= iterations
  error_list.append(error)

plt.figure()
plt.plot(np.array(m_list), np.array(error_list))
plt.xlabel('m')
plt.ylabel('average error')
plt.ylim((0.0286, 0.0289))
plt.xscale('log')
plt.xticks(m_list, m_list)
plt.savefig('6.png')
plt.show()
```

The plot of average errors versus the sample size m is as follow. We set the x-axis as log scale.


average error
m