Convex Optimization Exam, December 2020

— Clément Bonnet —

Exercise 1

Let (P) be the following program:

$$(P): \quad \underset{x \in \mathbb{R}^n}{\text{minimize}} \ \|x\|_\infty \quad \text{subject to } Ax = b$$

$$(P) \Leftrightarrow \underset{x,t}{\text{minimize}} \ t \quad \text{subject to} \begin{cases} \|x\|_\infty \leq t \\ Ax = b \end{cases}$$

$$(P) \Leftrightarrow \underset{x,t}{\text{minimize}} \ t \quad \text{subject to} \begin{cases} Ax = b \\ x \leq t 1_n \\ -x \leq t 1_n \end{cases}$$

This is a linear program.

Its Lagrangian function is:

$$\mathcal{L}(x, t, \lambda, \mu_1, \mu_2) = t + \lambda^T (Ax - b) + \mu_1^T (x - t 1_n) + \mu_2^T (-x - t 1_n)$$
$$= \left[1 - \mu_1^T 1_n - \mu_2^T 1_n\right] t + \left[A^T \lambda + \mu_1 - \mu_2\right]^T x - \lambda^T b$$

Let g be $g(\lambda, \mu_1, \mu_2) = \min_{x,t} \mathcal{L}(x, t, \lambda, \mu_1, \mu_2)$

$$= \begin{cases} -\lambda^T b & \text{if } 1 - \mu_1^T 1_n - \mu_2^T 1_n = 0 \\ & \text{and } A^T \lambda + \mu_1 - \mu_2 = 0 \\ -\infty & \text{otherwise} \end{cases}$$

Therefore, the dual of the linear program is the following:

$$(D): \quad \underset{\mu_1, \mu_2, \lambda}{\text{maximize}} \; -\lambda^T b$$
$$\text{subject to} \begin{cases} \mu_1^T 1_m + \mu_2^T 1_m = 1 \\ A^T \lambda + \mu_1 - \mu_2 = 0 \end{cases}$$

$$(D) \Leftrightarrow \underset{\lambda \in \mathbb{R}^m, \mu_1, \mu_2 \in \mathbb{R}^n_+}{\text{minimize}} \; \lambda^T b$$
$$\text{Subject to} \begin{cases} \mu_1^T 1_m + \mu_2^T 1_m = 1 \\ A^T \lambda + \mu_1 - \mu_2 = 0 \end{cases}$$

Exercise 2

Let D_{KL} be the Kullback-Leibler divergence.

$\forall u, v \in \mathbb{R}_{++}^n$, $D_{KL}(u,v) = \sum_{i=1}^{n} u_i \log\left(\frac{u_i}{v_i}\right)$

Let f be the negative entropy.

$\forall v \in \mathbb{R}_{++}^n$, $f(v) = \sum_{i=1}^{n} v_i \log v_i$

f is differentiable and $\nabla f(v) = \log(v) + 1_n$

where $\log(v) = (\log(v_1) \dots \log(v_n))^T$.

As provided by the hint,

$$f(u) - f(v) - \nabla f(v)^T(u-v)$$

$$= \sum u_i \log u_i - \sum v_i \log v_i - \underbrace{\sum u_i}_{=0} + \underbrace{\sum v_i}_{=0} - \sum u_i \log v_i + \sum v_i \log v$$

$$= \sum u_i \log\left(\frac{u_i}{v_i}\right) = D_{KL}(u,v)$$

Therefore, $D_{KL}(u,v) = f(u) - f(v) - \nabla f(v)^T(u-v)$.

f is twice differentiable and

$$\nabla^2 f(u) = \operatorname{diag}\left(\left(\frac{1}{u_i}\right)_{i \in [\![1,n]\!]}\right) \succ 0$$

Thus, f is strictly convex.

And, for all $(u,v) \in \mathbb{R}_{++}^n \times \mathbb{R}_{++}^n$, $f(u) \geq f(v) + \nabla f(v)^T(u-v)$

(first-order condition inequality), with equality when $u = v$.

Therefore, $D_{KL}(u,v) \geq 0$ with equality if and only if $u = v$.

Exercise 3

Let (P) be the following problem:

$$(P): \quad \underset{x \in \mathbb{R}^n}{\text{minimize}} \; c^T x \qquad \text{with } A \in S_n^+,\ A \succeq 0.$$

$$\text{subject to} \begin{cases} x^T (A - bb^T) x \leq 0 \\ b^T x \geqslant 0 \\ Dx = g \end{cases}$$

because $A \succeq 0$, there exists $V \in S_n$, $A = V^T V$.

Therefore, $(P) \Leftrightarrow$ minimize $c^T x$

$$\text{subject to} \begin{cases} x^T (V^T V - bb^T) x \leq 0 \\ b^T x \geqslant 0 \\ Dx = g \end{cases}$$

$\Leftrightarrow$ minimize $c^T x$

$$\text{subject to} \begin{cases} \|Vx\|_2^2 \leq (b^T x)^2 + b^T x \leq (b^T x)^2 \\ b^T x \geqslant 0 \\ Dx = g \end{cases}$$

$\Leftrightarrow$ $\underset{x \in \mathbb{R}^n}{\text{minimize}}$ $c^T x$

$$\text{subject to} \begin{cases} \|Vx\|_2 \leq b^T x \\ Dx = g \end{cases}$$

Therefore the problem (P) is a Second order Cone Program. It is thus a convex problem.

Exercise 4

$$(P): \underset{x \in D}{\text{minimize}} \; -\sum_{i=1}^{m} \log(b_i - a_i^T x), \qquad D = \{x \in \mathbb{R}^n \mid a_i^T x < b_i\}$$

$$(P) \Leftrightarrow \underset{x \in \mathbb{R}^n, c \in \mathbb{R}^m}{\text{minimize}} \; -\sum_{i=1}^{m} \log(c_i)$$

$$\text{subject to} \begin{cases} c_i = b_i - a_i^T x, \text{ for } i = 1 \dots m \\ c_i > 0, \text{ for } i = 1 \dots m \end{cases}$$

Its Lagrangian function is :

$$\mathcal{L}(x, c, \lambda, \mu) = -\sum_{i=1}^{m} \log(c_i) + \sum_{i=1}^{m} \lambda_i (c_i - b_i + a_i^T x) - \sum_{i=1}^{m} \mu_i c_i$$

$$= \left(\sum_{i=1}^{m} \lambda_i a_i\right)^T x + \sum_{i=1}^{m} (\lambda_i - \mu_i) c_i - \log(c_i) - \sum_{i=1}^{m} \lambda_i b_i$$

Let f be the function : $f(c_i) = (\lambda_i - \mu_i) c_i - \log c_i$ for $c_i > 0$.

f is differentiable and $f'(c_i) = \lambda_i - \mu_i - \frac{1}{c_i}$

f' reaches 0 if and only if $\lambda_i - \mu_i > 0$. Otherwise $\min_{c_i} f(c_i) = -\infty$.

The minimum of f is thus $f\left(\frac{1}{\lambda_i - \mu_i}\right) = 1 + \log(\lambda_i - \mu_i)$.

Therefore, one can compute the values of the following function: $g(\lambda, \mu) = \min_{x, c} \mathcal{L}(x, c, \lambda, \mu)$.

$$g(\lambda, \mu) = \begin{cases} \sum_{i=1}^{m} \log(\lambda_i - \mu_i) - \sum_{i=1}^{m} \lambda_i b_i + m & \text{if } \begin{cases} \sum_{i=1}^{m} \lambda_i a_i = 0 \\ \forall i = 1 \dots m, \; \lambda_i - \mu_i > 0 \end{cases} \\ -\infty \qquad \text{otherwise} \end{cases}$$

The dual problem is thus : (D) : $\underset{\lambda, \mu}{\text{maximize}} \; g(\lambda, \mu)$

$$(\mathcal{D}) \iff \begin{array}{l|l} \underset{\lambda \in \mathbb{R}^m,\, \mu \in \mathbb{R}_+^m}{\text{maximize}} & \sum_{i=1}^{m} \log(\lambda_i - \mu_i) - \sum \lambda_i b_i + m \\ \text{subject to} & \sum \lambda_i a_i = 0 \\ & \lambda - \mu > 0 \end{array}$$

Exercise 5

Let us consider the problem :

(1) : minimize $f_0(x)$ subject to $Ax = b$ with $A \in \mathbb{R}^{m \times n}$, $\operatorname{rank} A = m$, f_0 convex and differentiable

Quadratic penalty method :

$$\phi(x) = f_0(x) + \alpha \|Ax - b\|_2^2 \quad \text{with } \alpha > 0.$$

Let us suppose $\hat{x}$ is a minimizer of ϕ and let's find a dual feasible point for (1).

The Lagrangian of (1) is : $\mathcal{L}(x, \lambda) = f_0(x) + \lambda^T (Ax - b)$

$$\mathcal{L}(x, \lambda) = f_0(x) + (A^T \lambda)^T x - \lambda^T b$$

Let g be : $g(\lambda) = \min_x \mathcal{L}(x, \lambda)$.

Then the dual problem of (1) is the following :

$$(D) : \underset{\lambda \in \mathbb{R}^m}{\text{maximize}}\ g(\lambda)$$

A feasible point of (D) is $g(\lambda_0)$ with $\lambda_0 \in \mathbb{R}^m$ and $g(\lambda_0) = \min_{x \in \mathbb{R}^n} f_0(x) + (A^T \lambda_0)^T x - \lambda_0^T b$

ϕ is convex and differentiable since f_0 is so.

$$\forall x \in \mathbb{R}^n,\ \phi(\hat{x}) \geq \phi(x) + \nabla \phi(x)^T (\hat{x} - x)$$

with $\nabla \phi(x) = \nabla f_0(x) + 2\alpha A^T A x - 2\alpha A^T b$

8

$$g(\lambda) = \min_x f_0(x) + (A^T\lambda)^T x - \lambda^T b$$
$$= \min_x \mathcal{L}(x, \lambda)$$
$$= \mathcal{L}(\bar{x}, \lambda) \quad \text{for } \bar{x} \in \operatorname{argmin}_x \mathcal{L}(x, \lambda)$$
$$\nabla_x \mathcal{L}(\bar{x}, \lambda) = \boxed{0 = \nabla f_0(\bar{x}) + A^T\lambda_0}$$

Also, since $\tilde{x}$ minimizes ϕ,

$$\nabla\phi(\tilde{x}) = \boxed{0 = \nabla f_0(\tilde{x}) + 2\alpha A^T A\tilde{x} - 2\alpha A^T b}$$

for $\bar{x} = \tilde{x}$,

$$-A^T\lambda_0 = -2\alpha A^T A\tilde{x} + 2\alpha A^T b$$

since rank $A = m$, $\boxed{\lambda_0 = 2\alpha(A\tilde{x} - b)}$

λ_0 is a dual feasible point for (1).

Let p be the optimal value of (1),

$$p = \max_\lambda g(\lambda) \geq g(\lambda_0)$$

$$g(\lambda_0) = \min_x f_0(x) + 2\alpha(A\tilde{x} - b)^T(Ax - b)$$

Exercise 6

Let us consider the following classification problem:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|w\|_2^2 + C1^T z && \text{in the variables } w \in \mathbb{R}^m, z \in \mathbb{R}^m \\ &\text{subject to} && \begin{cases} y_i (w^T x_i) \geq 1 - z_i, & i = 1 \dots m \\ z \geq 0 \end{cases} \end{aligned}$$

Its Lagrangian function is:

$$\mathcal{L}(w, z, \mu, \alpha) = \frac{1}{2} w^T w + C 1^T z - \mu^T z + \sum_{i=1}^{m} \alpha_i \left(1 - z_i - y_i (w^T x_i)\right)$$

$$= \frac{1}{2} w^T w - \left(\sum_{i=1}^{m} \alpha_i y_i x_i\right)^T w + \left(C1 - \mu - \alpha\right)^T z + \alpha^T 1$$

$$g(\mu, \alpha) = \min_{w, z} \mathcal{L}(w, z, \mu, \alpha)$$

$$= \begin{cases} -\infty & \text{if } \mu + \alpha \neq C1 \\ -\frac{1}{2}\left(\sum_{i=1}^{m} \alpha_i y_i x_i\right)^T \left(\sum_{i=1}^{m} \alpha_i y_i x_i\right) + \alpha^T 1 & \text{if } \mu + \alpha = C1. \end{cases}$$

Therefore, the dual problem is the following:

$$\begin{aligned} (D): \underset{\mu \in \mathbb{R}^m, \alpha \in \mathbb{R}^m}{\text{maximize}} \quad & \alpha^T 1 - \frac{1}{2}\left\|\sum_{i=1}^{m} \alpha_i y_i x_i\right\|_2^2 \\ \text{subject to} \quad & \begin{cases} \mu + \alpha = C1 \\ \mu \geq 0 \\ \alpha \geq 0 \end{cases} \end{aligned}$$

$$(D) \Leftrightarrow \begin{aligned} \underset{\alpha \in \mathbb{R}^m}{\text{maximize}} \quad & 1^T \alpha - \frac{1}{2}\left\|\sum_{i=1}^{m} \alpha_i y_i x_i\right\|_2^2 \\ \text{subject to} \quad & 0 \leq \alpha \leq C1 \end{aligned}$$

Once the optimal value α^* of the dual problem is found, one can use the KKT conditions to access the optimal value w^* of the primal.

KKT: $\frac{\partial \mathcal{L}}{\partial w}(w^*, z^*, \alpha^*, \mu^*) = 0 = w^* - \sum_{i=1}^{m} \alpha_i^* y_i x_i$

Therefore, $\boxed{w^* = \sum_{i=1}^{m} \alpha_i^* y_i x_i}$

Convex Optimization Exam

December 1, 2020

1 Convex Optimization Exam

1.1 Author: Clément Bonnet

1.2 Exercise 6

Let (P) be a primal problem and (D) its dual.

$$
\begin{aligned}
(P): \quad & \underset{w\in\mathbb{R}^n, z\in\mathbb{R}^m}{\text{minimize}} && \frac{1}{2}\|w\|_2^2 + C1^T z \\
& \text{subject to} && y_i(w^T x_i) \geq 1 - z_i, \forall i = 1 \cdots n \\
& && z \succeq 0 \\
(D): \quad & \underset{\alpha\in\mathbb{R}^m}{\text{minimize}} && f(\alpha) = \frac{1}{2}\Big\| \sum_{i=1}^{m} \alpha_i y_i x_i \Big\|_2^2 - 1^T\alpha \\
& \text{subject to} && 0 \preceq \alpha \preceq C1
\end{aligned}
$$

After including the log-barrier for the dual problem, the function f_t to minimize now becomes:

$$
f_t(\alpha) = \frac{t}{2}\Big\| \sum_{i=1}^{m} \alpha_i y_i x_i \Big\|_2^2 - t1^T\alpha - \sum_{i=1}^{m} \log(\alpha_i(C - \alpha_i))
$$

One can derive its gradient ∇f_t:

$$
\nabla f_t(\alpha)_i = ty_i\Big(\sum_{k=1}^{m} \alpha_k y_k x_k\Big)^T x_i - t - \frac{1}{\alpha_i} - \frac{1}{C - \alpha_i}
$$

Finally, the hessian matrix follows from differentiating the gradient.

$$
\nabla^2 f_t(v)_{i,j} = \begin{cases} ty_i y_j x_j^T x_i, & \text{if } i \neq j \\ ty_i^2 x_i^T x_i + \frac{1}{\alpha_i^2} - \frac{1}{(C-\alpha_i)^2}, & \text{if } i = j \end{cases}
$$

Once the optimal value $\alpha^\star$ of the dual problem is found, one can recover the optimal solution $w^\star$ of the primal using the KKT conditions. At the optimum,

$$\nabla_w L(w, z, \mu, \alpha) = 0 = w^\star - \sum_{i=1}^{m} \alpha_i^\star y_i x_i$$

$$\implies \boxed{w^\star = \sum_{i=1}^{m} \alpha_i^\star y_i x_i}$$

[1]:
```
# Import statements

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

[2]:
```
# Auxiliary functions

def in_domain(a, c):
    """
    Check whether a is in the domain of function f_t. So that log can be␣
 ↪defined.
    """
    return (a < c).all() & (a > 0).all()

def backtrack(x, y, c, t, a, eps, alpha, beta, step_dir, grad):
    """
    Backtracking line search.
    Return the right step_size for next update.
    """
    f_t = lambda a: t/2 * np.linalg.norm(((a.reshape(-1,1))*(y.
 ↪reshape(-1,1))*x).sum(axis=0))**2 - t*a.sum() - np.log(a*(c-a)).sum()
    a_back = a.copy()
    step_size = 1
    backtrack_crit = False
    while not backtrack_crit:
        step_size *= beta
        if in_domain(a_back + step_size*step_dir, c):
            backtrack_crit = f_t(a_back + step_size*step_dir) < f_t(a_back) +␣
 ↪alpha*step_size*grad.T@step_dir
    return step_size
```

[3]:
```
# Barrier method implementation

def centering_step(x, y, c, t, a, eps, alpha, beta, verbose=False):
    """
    Centering step.
    """
```

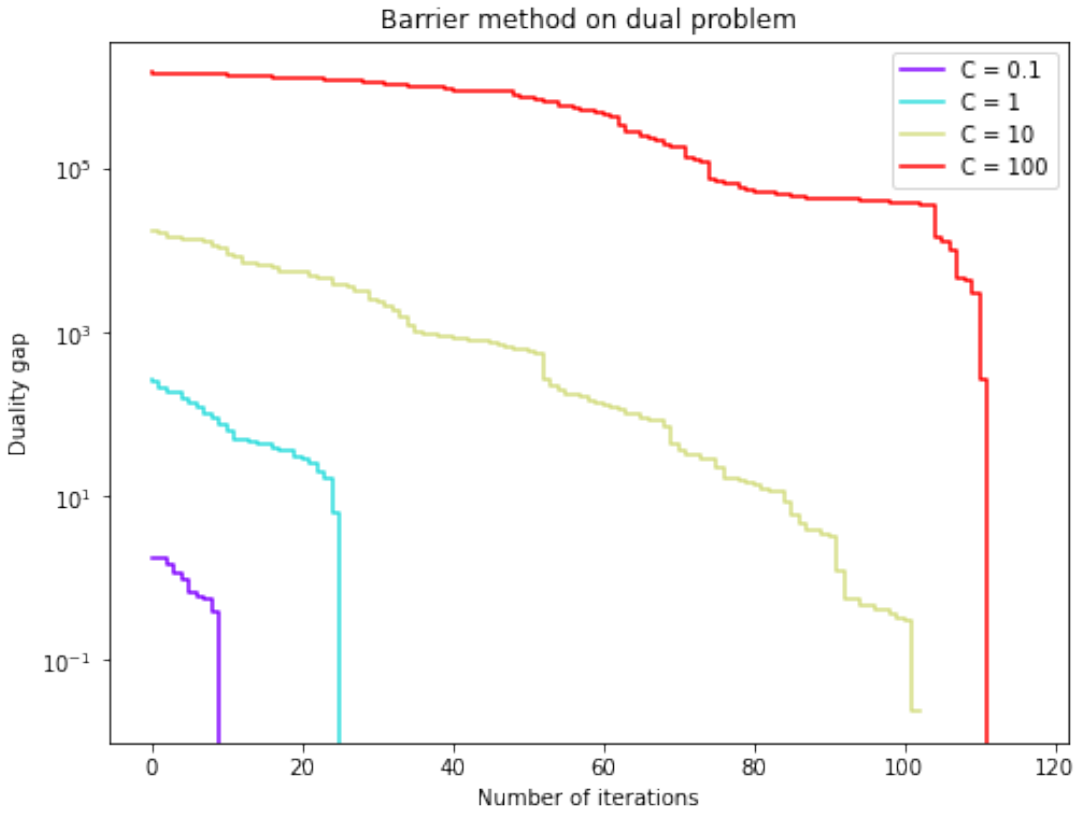
```
    # Side effect on a
    m = len(y)
    seq = [a.copy()]
    center_crit = False
    while not center_crit:
        # Compute gradient and hessian matrix
        grad = np.zeros(m)
        for i in range(m):
            grad[i] = t*y[i]*((a.reshape(-1,1))*(y.reshape(-1,1))*x).
 ↪sum(axis=0)@x[i,:] -t -1/a[i] -1/(c - a[i])
        hess = np.zeros((m,m))
        for i in range(m):
            for j in range(m):
                hess[i,j] = t*y[i]*y[j]*x[i,:]@x[j,:]
        hess = hess + np.diag(1/a**2 - 1/(c-a)**2)
        # Newton step
        step_dir = np.linalg.solve(hess, -grad)
        # Newton decrement
        squared_decr = -grad.T @ step_dir
        center_crit = squared_decr/2 <= eps
        if verbose: print("Squared decrement:", squared_decr)
        if center_crit: break
        # Backtracking line search
        step_size = backtrack(x, y, c, t, a, eps, alpha, beta, step_dir, grad)
        a += step_size*step_dir
        seq.append(a.copy())
    return seq

def barr_method(x, y, c, a0, eps=1e-3, mu=10, t0=1, alpha=0.1, beta=0.5,␣
 ↪verbose=False):
    """
    Barrier method to solve the dual problem.
    Returns a sequence of values until the optimum.
    """
    t = t0
    m = len(y)
    seq = []
    a = a0.copy()
    while m/t >= eps:
        t *= mu
        seq.extend(centering_step(x, y, c, t, a, eps, alpha, beta, verbose))
    return seq
```

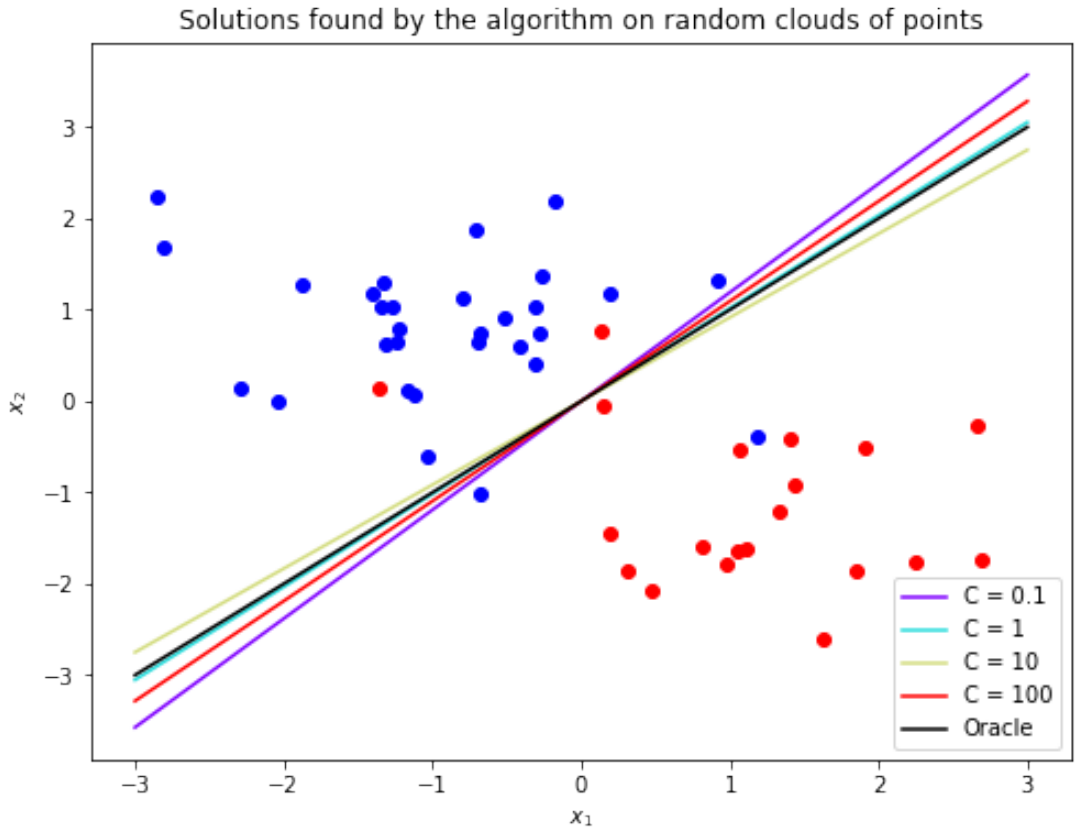
[4]:
```
# Plotting functions

def print_duality_gap(seq, x, y, c=None, color=None):
```

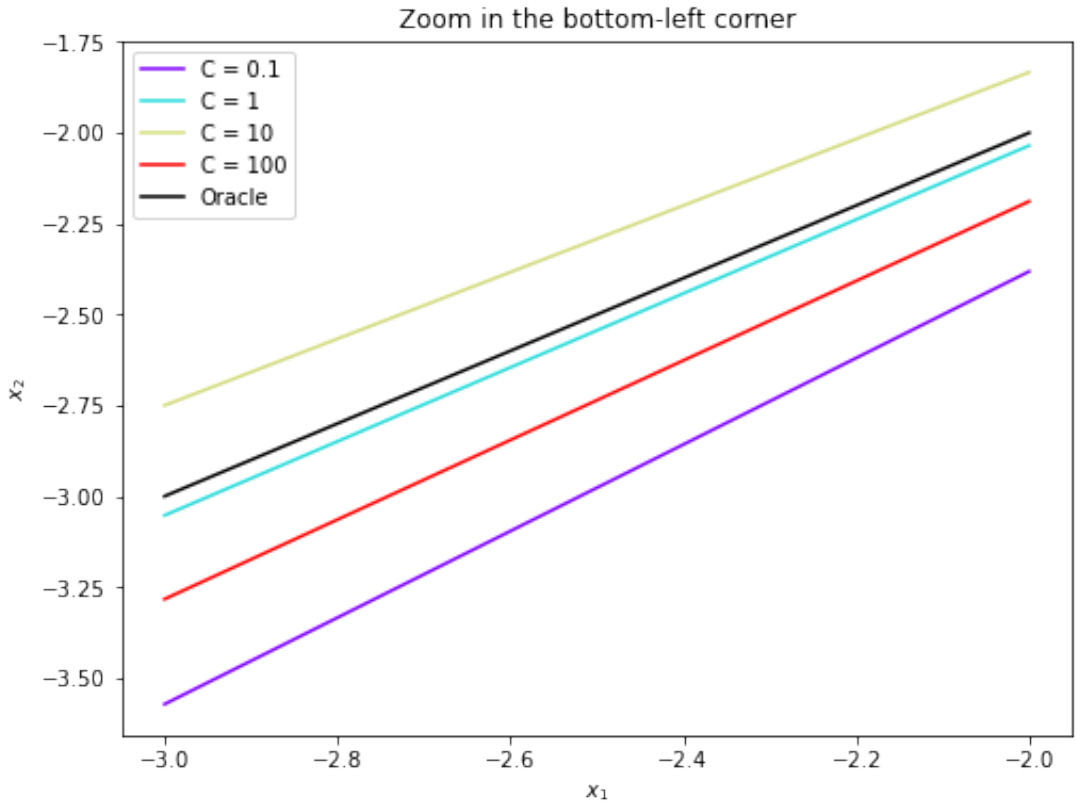
```
    f = lambda a: 1/2 * np.linalg.norm(((a.reshape(-1,1))*(y.reshape(-1,1))*x).
↪sum(axis=0))**2 - a.sum()
    f_seq_arr = np.array([f(seq_i) for seq_i in seq])
    i_star, f_star = np.argmin(f_seq_arr), f_seq_arr.min()
    a_star = np.array(seq)[i_star]
    w_star = ((a_star.reshape(-1,1))*(y.reshape(-1,1))*x).sum(axis=0)
    to_plot = [abs(f_seq_arr_i - f_star) for i, f_seq_arr_i in␣
↪enumerate(f_seq_arr) if i != i_star]
    plt.step(range(len(to_plot)), to_plot, label="C = {}".format(c),␣
↪color=color);
    plt.yscale("log");
    plt.xlabel("Number of iterations");
    plt.ylabel("Duality gap");
    plt.legend();
    plt.title("Barrier method on dual problem");
    return a_star, w_star

def plot_all(c_list, m=10, n=2, eps=1e-3, random_seed=0, plot_points=True,␣
↪zoom=True):
    np.random.seed(random_seed)
    means = np.array([[-1, 1], [1, -1]])
    sigmas = np.array([[[1, 0],[0, 1]], [[1, 0],[0, 1]]])
    x = np.zeros((m,n))
    y = np.zeros(m)
    for i in range(m):
        k = 0 if np.random.rand() > 0.5 else 1
        x[i,:] = np.random.multivariate_normal(means[k], sigmas[k])
        y[i] = 2*k - 1
    colors = matplotlib.cm.rainbow(np.linspace(0, 1, len(c_list)))
    w_stars = np.zeros((len(c_list), n))
    a_stars = np.zeros((len(c_list), m))
    plt.figure(figsize=(8,6));
    for i, c in enumerate(c_list):
        a0 = c * np.random.rand(m)
        seq = barr_method(x, y, c, a0, eps=eps)
        a_star, w_star = print_duality_gap(seq, x, y, c, colors[i])
        a_stars[i] = a_star
        w_stars[i] = w_star
    if plot_points:
        plt.figure(figsize=(8,6));
        plt.scatter(x[:,0][y==-1], x[:,1][y==-1], color="blue");
        plt.scatter(x[:,0][y==1], x[:,1][y==1], color="red");
        plt.xlabel("$x_1$");
        plt.ylabel("$x_2$");
        plt.title("Solutions found by the algorithm on random clouds of␣
↪points");
        x_w = np.linspace(-3, 3, 100)
```

```
        for i, w_star in enumerate(w_stars):
            y_w = -w_star[0]/w_star[1] * x_w
            plt.plot(x_w, y_w, label="C = {}".format(c_list[i]),␣
↪color=colors[i]);
        y_w = x_w
        plt.plot(x_w, y_w, label="Oracle", color="black");
        plt.legend();
        if zoom:
            plt.figure(figsize=(8,6));
            plt.title("Zoom in the bottom-left corner");
            plt.xlabel("$x_1$");
            plt.ylabel("$x_2$");
            x_w = np.linspace(-3, -2, 10);
            for i, w_star in enumerate(w_stars):
                y_w = -w_star[0]/w_star[1] * x_w
                plt.plot(x_w, y_w, label="C = {}".format(c_list[i]),␣
↪color=colors[i]);
            y_w = x_w
            plt.plot(x_w, y_w, label="Oracle", color="black");
            plt.legend();
    return x, y, a_stars, w_stars
```

[5]:
```
# Experiment with c = 0.1, 1, 10, 100 | m = 50

c_list = [0.1, 1, 10, 100]
x, y, a_stars, w_stars = plot_all(c_list, m=50, n=2, eps=1e-7, random_seed=1)
```


Barrier method on dual problem
C = 0.1
C = 1
C = 10
C = 100
Duality gap
10^5
10^3
10^1
10^{-1}
0
20
40
60
80
100
120
Number of iterations


Solutions found by the algorithm on random clouds of points
C = 0.1
C = 1
C = 10
C = 100
Oracle
x_1
x_2



$C = 1$ leads to the best fit, although they all fit the data quite well.