Exercise 1.

Exercise 2.1. Gradient descent of FoE.

Show that the gradient descent of E_θ^{FoE} is as ch:

$$u^{t+1} = u^t - \eta\left(\frac{1}{\sigma^2}(u^t - v) + \sum_{i=1}^{N} \bar{k}_i * \varphi_i'(k_i * u^t)\right),\ t = 0, \dots, T.$$

Recall that a gradient descent is an iterative minimization method with the following update equation:

$u^{t+1} = u^t - \eta \nabla E_\theta^{FoE}(u, v)$.

We have to compute ∇E_θ^{FoE}.

$$E_\theta^{FoE} = \frac{1}{2\sigma^2}\|u - v\|^2 + \sum_{x\in\Omega}\sum_{i=1}^{N} \varphi_i(k_i * u(x)) + C.$$

Lets start part by part.

$$\nabla\left(\frac{1}{2\sigma^2}\|u - v\|^2\right) = \frac{1}{\sigma^2}(u - v)$$

$$\nabla\left(\sum_{x\in\Omega}\sum_{i=1}^{N} \varphi_i(k_i * u(x))\right) = \sum_{x\in\Omega}\sum_{i=1}^{N} \nabla\varphi_i(k_i * u(x)) = \sum_{x\in\Omega}\sum_{i=1}^{N} \nabla(k_i * u(x)) \cdot \varphi_i'(k_i * u(x)) = \sum_{x\in\Omega}\sum_{i=1}^{N} \varphi_i'(k_i * u(x)) \begin{pmatrix} 0 \\ \vdots \\ k_i[x] \\ \vdots \\ k_i[1] \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Next, we will change summation order.

$$\sum_{i=1}^{N}\sum_{x\in\Omega} \begin{pmatrix} 0 \\ \vdots \\ k_i[x] \\ \vdots \\ k_i[1] \\ 0 \\ \vdots \\ 0 \end{pmatrix} \varphi_i'(k_i * u(x)) = \sum_{i=1}^{N} \bar{k}_i * \varphi_i'(k_i * u(x)).$$

$$\nabla C = 0 \Rightarrow \nabla E_\theta^{FoE} = \frac{1}{\sigma^2}(u - v) + \sum_{i=1}^{N} \bar{k}_i * \varphi_i'(k_i * u(x)) \Rightarrow$$

$$\Rightarrow u^{t+1} = u^t - \eta\left(\frac{1}{\sigma^2}(u - v) + \sum_{i=1}^{N} \bar{k}_i * \varphi_i'(k_i * u(x))\right)$$

Exercise 2.2. Maximum likelihood as density matching.

The Kullback-Leibler divergence (also called relative entropy) is a measure of how one probability distribution q differs from a second reference probability distribution p, given by:

$$KL(p(x)\|q(x)) = \int \log\left(\frac{p(x)}{q(x)}\right) p(x)\,dx$$

We assume that data distributed according to $p(x)$. Consider the limit of the log-likelihood when $m \to +\infty$:

$$\mathcal{L}_\infty(\theta) = E\{\log p_\theta(x)\} = \int \log p_\theta(x)\, p(x)\,dx = \lim_{m\to\infty} \frac{1}{m}\sum_{j=1}^{m} \log p_\theta(x_j).$$

Show that maximizing the expected log-likelihood $\mathcal{L}_\infty$ minimizes

$$KL(p(x)\|p_\theta(x)) = \int \log\left(\frac{p(x)}{p_\theta(x)}\right) p(x)\,dx$$

$$\max_\theta \mathcal{L}_\infty(\theta) = \max_\theta \int \log(p_\theta(x))\,p(x)\,dx = \min_\theta -\int \log(p_\theta(x))\,p(x)\,dx$$

Also, $\min_\theta KL(p(x)\|p_\theta(x)) = \min_\theta \int \log\left(\frac{p(x)}{p_\theta(x)}\right) p(x)\,dx = \min_\theta \int \log(p(x))\,p(x) - \log(p_\theta(x))\,p(x)\,dx$. The first part is independent of θ, so it is not involved in minimization, we have $\min_\theta -\int \log(p_\theta(x))\,p(x)\,dx + \int \log(p(x))\,p(x)\,dx \Rightarrow$ We have to solve the $\min_\theta -\int \log(p_\theta(x))\,p(x)\,dx$ which is the same as we got before, so maximizing $\mathcal{L}_\infty(\theta)$ minimizes $KL(p(x)\|p_\theta(x))$

Exercise 2.

Implement the barrier method to solve QP

```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

def new_v(Q, p, v, t, diff_order_1, diff, t0):
    condition1 = t0*(np.dot(v.T, np.dot(Q, v)) + np.dot(p.T, v))
    for i in range(b.shape[0]):
      condition1 -= np.log(b[i]-np.dot(A[i], v))
    condition1 += alpha*t*np.dot(diff_order_1.T, diff)
    condition2 = t0*(np.dot((v + t*diff).T, np.dot(Q, v + t*diff)) + np.dot(p.T, v + t*dif
    condition2 -= np.log(b[i]-np.dot(A[i], v + t*diff))
    if condition2 <= condition1 or ((b-A.dot(v+t*diff))>0).all():
        return v + t*diff
    else:
        while condition2 > condition1 and ((b-A.dot(v+t*diff))>0).all():
          t *= beta
          condition1 = t0*(np.dot(v.T, np.dot(Q, v)) + np.dot(p.T, v))
          for i in range(b.shape[0]):
            condition1 -= np.log(b[i]-np.dot(A[i], v))
          condition1 += alpha*t*np.dot(diff_order_1.T, diff)
          condition2 = t0*(np.dot((v + t*diff).T, np.dot(Q, v + t*diff)) + np.dot(p.T, v +
          condition2 -= np.log(b[i]-np.dot(A[i], v + t*diff))
    return v + t*diff

def centering_step(Q, p, A, b, t, v0, eps):
    v_list = [v0.copy()]
    diff_order_1 = t*(2*np.dot(Q, v0) + p)
    diff_order_2 = 2*t*Q
    for i in range (b.shape[0]):
      diff_order_1 += A[i, np.newaxis].T/(b[i]-np.dot(A[i], v0))
      diff_order_2 += (np.outer(A[i, np.newaxis].T, A[i, np.newaxis].T)) / ((b[i] - np.dot
    diff = -1*np.dot(np.linalg.inv(diff_order_2), diff_order_1)
    acc = np.dot(diff_order_1.T, np.dot(np.linalg.inv(diff_order_2), diff_order_1))
    if acc/2 <= eps: return v_list
    v1 = v0.copy()
    while acc/2 >= eps:
      v1 = new_v(Q, p, v1, t=1, diff_order_1=diff_order_1, diff=diff, t0=t)
      diff_order_1 = t*(2*np.dot(Q, v1) + p)
      diff_order_2 = 2*t*Q
      for i in range (b.shape[0]):
        diff_order_1 += A[i, np.newaxis].T/(b[i]-np.dot(A[i], v1))
        diff_order_2 += (np.outer(A[i, np.newaxis].T, A[i, np.newaxis].T)) / ((b[i] - np.d
      diff = -np.dot(np.linalg.inv(diff_order_2), diff_order_1)
      acc = np.dot(diff_order_1.T, np.dot(np.linalg.inv(diff_order_2), diff_order_1))
      v_list.append(v1)
    return v_list


def barr_method(Q, p, A, b, v0, eps, mu):
    t = 1
    n = 0
    n_list = [0]
```

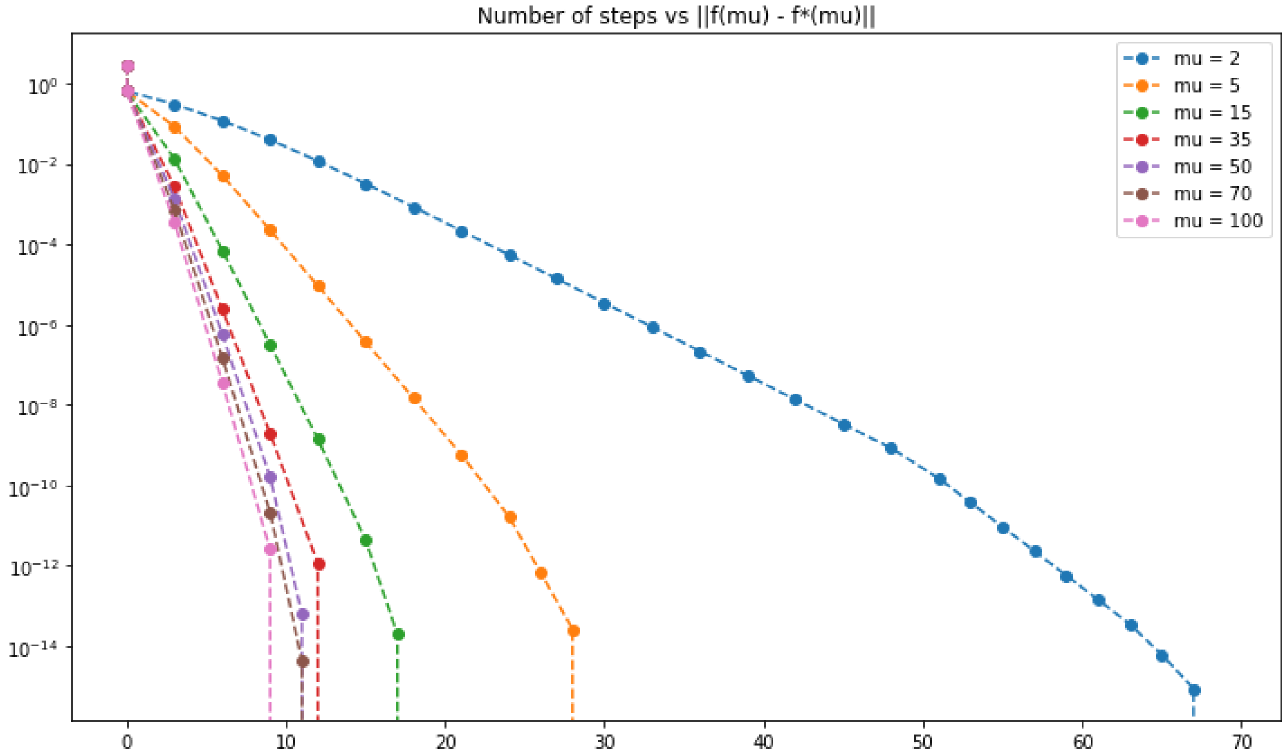
```
    v_list = [v0]
    f_list = [(np.dot(v0.T, np.dot(Q, v0)) + np.dot(p.T, v0))[0][0]]
    while True:
      v_list1 = centering_step(Q, p, A, b, t, v0, eps)
      v_center, n1 = v_list1[-1], len(v_list1)
      n_list.append(n)
      n += n1
      v_list.append(v_center)
      f_list.append((np.dot(v_center.T, np.dot(Q, v_center)) + np.dot(p.T, v_center))[0][0
      if b.shape[0]/t < eps:
          return v_list, n_list , f_list
      t = mu*t
```

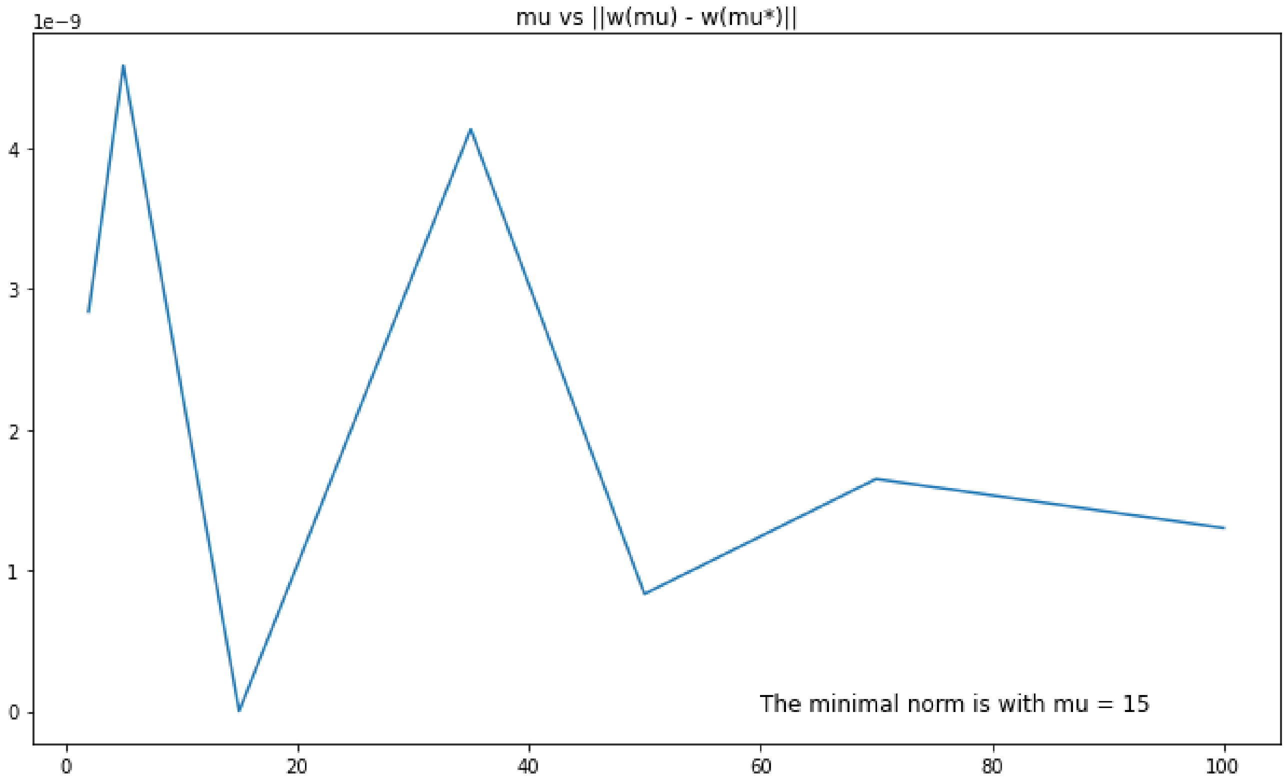
Exercise 3.

Test your function on randomly generated matrices X and observations y with λ = 10. Plot precision criterion and gap f(v_t)−f* in semilog scale (using the best value found for f as a surrogate for f*). Repeat for different values of the barrier method parameter μ = 2, 15, 50, 100, ... and check the impact on w. What would be an appropriate choice for μ?

```
alpha = 0.05
beta = 0.1
n = np.random.randint(low = 10, high=15)
d = np.random.randint(low = 15, high=20)
eps = 10e-7
lambda1 = 10
X = np.random.rand(n,d)
Q = 0.5*np.eye(n)
p = -np.random.rand(n,1)
A = np.vstack((X.T,-X.T))
b = lambda1*np.ones((2*d,1))
v0 = np.zeros((n,1))
mu_list = [2, 5, 15, 35, 50, 70, 100]
w_list = []
f1_list = []
plt.figure(figsize = (12, 7))
for mu in mu_list:
    result = barr_method(Q, p, A, b, v0, eps, mu)
    v = result[0][-1]
    v_list = result[0]
    n_list = result[1]
    f_list = result[2]
    w_list.append(np.linalg.lstsq(X,-v+p)[0])
    f1_list.append(f_list[-1])
    plt.plot(n_list, f_list - f_list[-1],  linestyle='--', marker='o', label='mu = {}'.for
plt.title("Number of steps vs ||f(mu) - f*(mu)||")
plt.semilogy()
plt.legend()
plt.show()
plt.figure(figsize = (12, 7))
w_norm = []
for w in w_list:
```

```
  w_norm.append(np.linalg.norm(w-w_list[np.argmin(f1_list)]))
plt.plot(mu_list, w_norm)
plt.text(x = 60, y = 0, s = "The minimal norm is with mu = {}".format(mu_list[np.argmin(np
plt.title("mu vs ||w(mu) - w(mu*)||")
plt.show()
```





The best mu that minimizes ||w(mu) - w(mu*)|| is mu = 15 and with mu = 100 we have fastest, but the norm is bigger than 0, so the best combination of minimazation of norm and speed is mu = 15.