Homework 6

Problem 1

(a)

Yes, if α is constant but sufficiently small, GD will converge to a minimizer of f.

(b)

No, even if the α is suffiently small, SGM still has no guarantee to converge to the minimizer of f.

(c)

Yes, there exists a α_k under certain conditions that makes SGM to converge to a minimizer of f.

Problem 2

1

According to the given equation of hinge loss, we define the term $\langle \mathbf{w}, \mathbf{x}_i \rangle + b$ in function $J(\mathbf{w}, b)$ as t_i, thus the given empirical risk can be rewritten as:

$$\begin{aligned} J(\mathbf{w}, b) &= \frac{1}{n} \sum_{i=1}^{n} (\max\{0, 1 - y_i t_i\}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \\ &= \sum_{i=1}^{n} \frac{1}{n} (\max\{0, 1 - y_i t_i\}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \\ &= \sum_{i=1}^{n} \left(\frac{1}{n} \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\} + \frac{\lambda}{2n} \|\mathbf{w}\|^2 \right). \end{aligned}$$

Therefore, $J_i(\mathbf{w}, b)$ can be interpreted as:

$$J_i(\mathbf{w}, b) = \frac{1}{n} \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\} + \frac{\lambda}{2n} \|\mathbf{w}\|^2.$$

2

Firstly, we should further rewrite J_i to be more readable

$$J_i(\mathbf{w}, b) = \frac{1}{n}\max\left\{0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\right\} + \frac{\lambda}{2n}\|\mathbf{w}\|^2,$$

subsequently, use θ to simplifiy J_i

$$\begin{aligned} J_i(\mathbf{w}, b) &= \frac{1}{n}\max\left\{0, 1 - y_i\theta^\top \tilde{\mathbf{x}}_i\right\} + \frac{\lambda}{2n}\left\|\begin{bmatrix} 0 & \\ & \mathbf{I} \end{bmatrix} \cdot \theta\right\|^2 \\ &= \frac{1}{n}\max\left\{0, 1 - y_i\theta^\top \tilde{\mathbf{x}}_i\right\} + \frac{\lambda}{2n}\theta^\top \begin{bmatrix} 0 & \\ & \mathbf{I} \end{bmatrix} \cdot \theta \end{aligned}$$

where $\tilde{\mathbf{x}_i} = [1, \mathbf{x_i}^\top]^\top$. We can define $\tilde{\mathbf{I}} = \begin{bmatrix} 0 & \\ & \mathbf{I} \end{bmatrix}$ for simplicity, then J_i can be further rewritten as

$$J_i = \frac{1}{n}\max\left\{0, 1 - y_i\theta^\top \tilde{\mathbf{x}}_i\right\} + \frac{\lambda}{2n}\theta^\top \tilde{\mathbf{I}}\theta$$

With the equation above, its gradient w.r.t. θ can be given as

$$\mathbf{u}_i = \begin{cases} \frac{1}{n}(-y_i\tilde{\mathbf{x}}_i + \lambda\tilde{\mathbf{I}}\theta), \text{ if } y_i\theta^\top \tilde{\mathbf{x}}_i < 1 \\ \frac{\lambda}{n}\tilde{\mathbf{I}}\theta, \text{ otherwise} \end{cases}$$

3

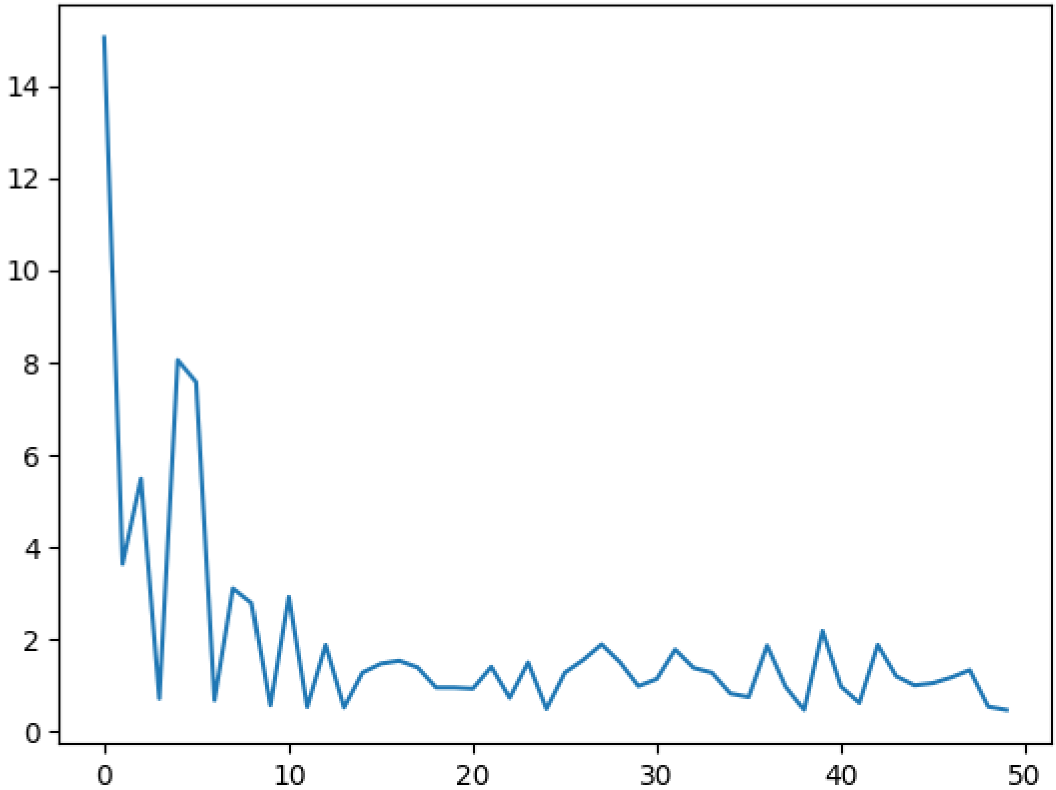
In [ ]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
```

In [ ]:

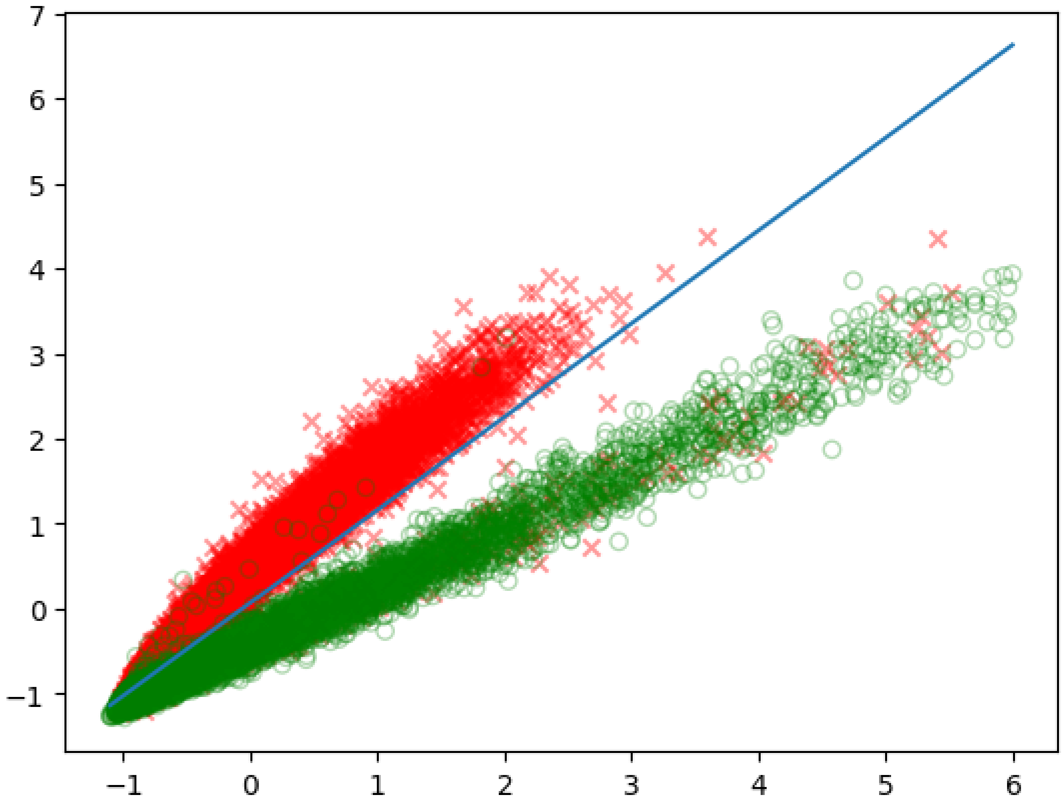
```
def computeSubgradient(x, y, theta, I_tilde):
  lamb = 0.001
  if y * np.dot(theta, x) < 1:
    u = -y * x + lamb * I_tilde @ theta
  else:
    u = lamb * I_tilde @ theta

  return u

def subgradientMethod(X_tilde, y, max_itera):
  # Initialize parameters
  n, dim = X_tilde.shape
  alpha = lambda x: 100 / x

  I_tilde = np.identity(X_tilde.shape[1])
  I_tilde[0, 0] = 0

  J_values = np.zeros(max_itera)

  for j in range(1, max_itera+1):
    a = alpha(j)
    theta = np.zeros(dim)
    u = np.zeros(dim)
    for i in range(0, n):
      # get new theta
      u += computeSubgradient(X_tilde[i, :], y[i], theta, I_tilde)
/ n

      theta -= a * u

    ts = X_tilde @ theta
    hinges = np.maximum(1 - ts * y, 0)
    J_values[j-1] = hinges.mean() + 0.001 / 2 * np.linalg.norm(I_t
ilde * theta) ** 2

  plt.plot(np.arange(max_itera), J_values)
  plt.show()

  plt.scatter(X_tilde[y == 1, 1], X_tilde[y == 1, 2], marker="x",
c ="red", alpha = 0.4)
  plt.scatter(X_tilde[y == -1, 1], X_tilde[y == -1, 2], marker
="o", facecolor = "none", edgecolors ="green", alpha = 0.3)

  hyerplane = lambda x: -x * theta[1] / theta[2] - theta[0] / thet
a[2]

  begin = hyerplane(np.min(X_tilde[:, 1]))
  end = hyerplane(np.max(X_tilde[:, 1]))

  plt.plot([np.min(X_tilde[:, 1]), np.max(X_tilde[:, 1])], [begin,
end])
  plt.show()
  return J_values, theta
```

In [ ]:

```
# Load data

X = np.loadtxt("nuclear/nuclear_x.csv", delimiter = ",")
y = np.loadtxt("nuclear/nuclear_y.csv", delimiter = ",")

SS = StandardScaler()
X = SS.fit_transform(X)

X_tilde = np.c_[np.ones(X.shape[0]), X]

subgradientMethod(X_tilde, y, 50)
```



0
2
4
6
8
10
12
14
0
10
20
30
40
50



−1
0
1
2
3
4
5
6
7
−1
0
1
2
3
4
5
6

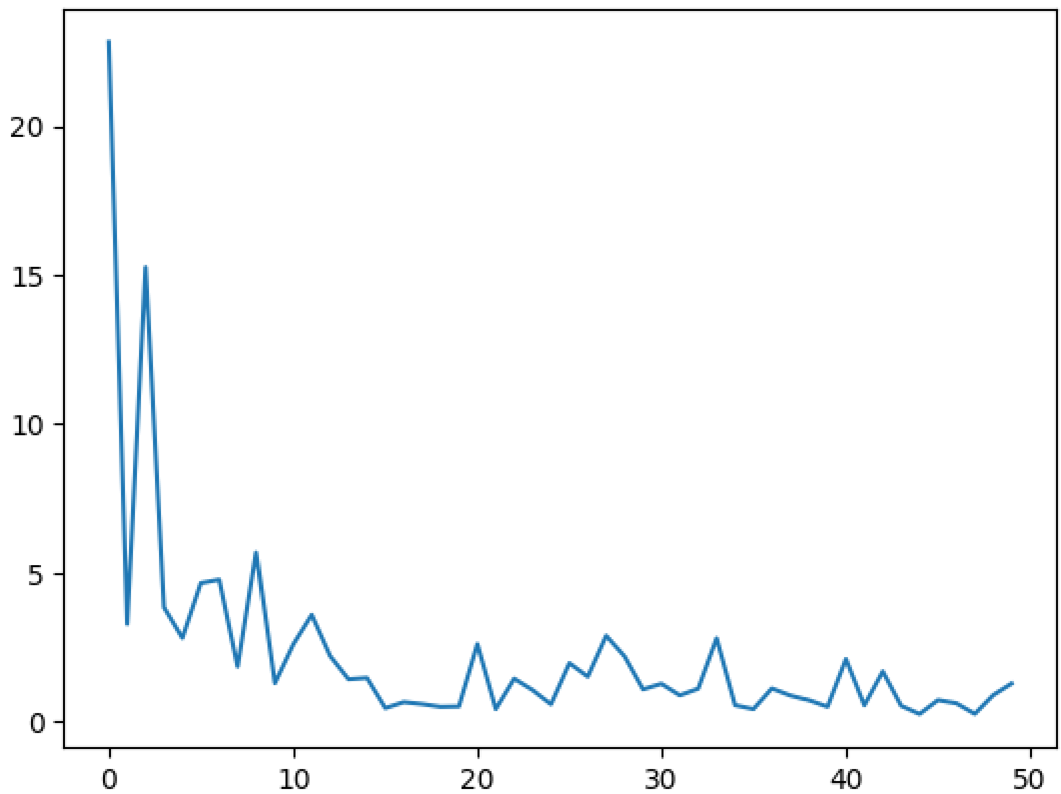
```
Out[ ]: (array([15.05098959,  3.6180429 ,  5.47297593,  0.69266158,  8.039
        58479,
                 7.56029971,  0.65628558,  3.08753336,  2.77041819,  0.545
        9478 ,
                 2.90351307,  0.51039883,  1.85894139,  0.49967506,  1.257
        00048,
                 1.4509005 ,  1.51489118,  1.36942965,  0.933362  ,  0.931
        55659,
                 0.90937535,  1.382746  ,  0.70848012,  1.47747331,  0.468
        74347,
                 1.257107  ,  1.53306105,  1.87106271,  1.48229553,  0.963
        13542,
                 1.1187471 ,  1.75878564,  1.36016706,  1.25588804,  0.803
        29498,
                 0.72720429,  1.84625381,  0.95392217,  0.44765199,  2.163
        13149,
                 0.95682193,  0.59924837,  1.86026521,  1.17911384,  0.979
        95145,
                 1.02922016,  1.15336156,  1.31212879,  0.51829045,  0.445
        48371]),
         array([ -0.8404    , -13.25914646,  12.11200179]))
```

4.

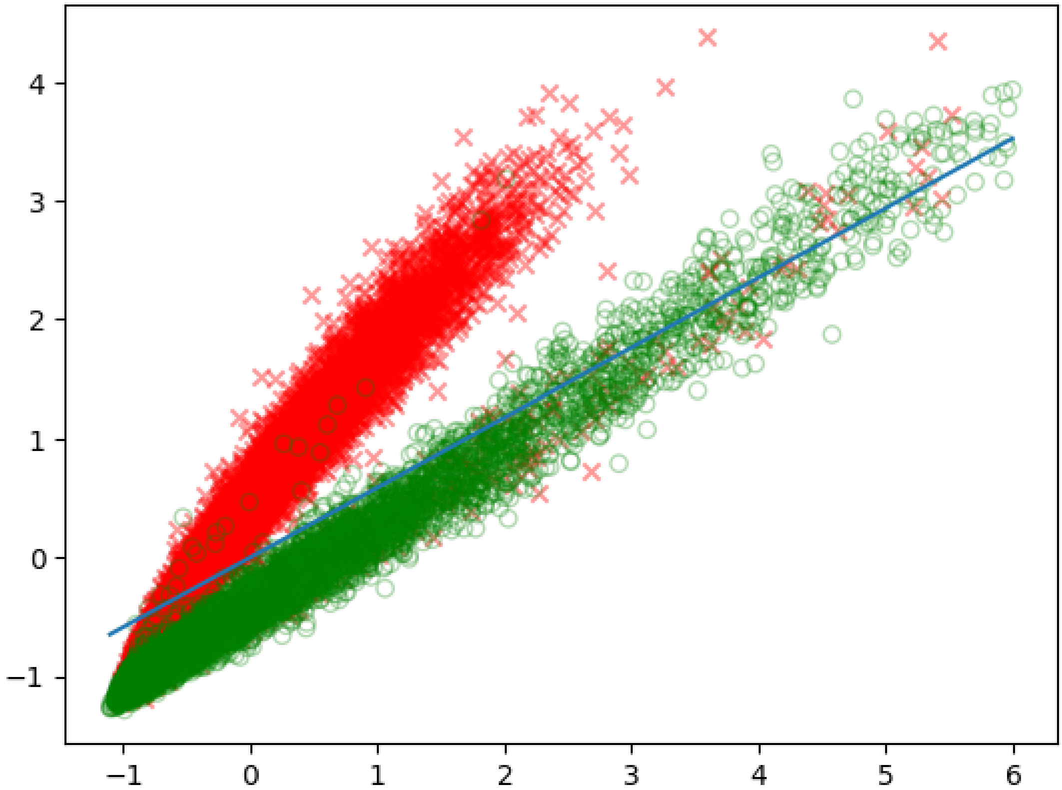
We can basically use the same code just with a littel ajustment

In [ ]:

```
def stochasticSubgradientMethod(X_tilde, y, max_itera):
  # Initialize parameters
  n, dim = X_tilde.shape
  alpha = lambda x: 100 / x

  I_tilde = np.identity(X_tilde.shape[1])
  I_tilde[0, 0] = 0

  J_values = np.zeros(max_itera)

  for j in range(1, max_itera+1):
    a = alpha(j)
    theta = np.zeros(dim)
    u = np.zeros(dim)
    rand_array = np.random.permutation(n)
    for i in range(0, n):
      # get new theta
      rand_index = rand_array[i]
      u += computeSubgradient(X_tilde[rand_index, :], y[rand_inde
x], theta, I_tilde) / n

      theta -= a * u

    ts = X_tilde @ theta
    hinges = np.maximum(1 - ts * y, 0)
    J_values[j-1] = hinges.mean() + 0.001 / 2 * np.linalg.norm(I_t
ilde * theta) ** 2

  plt.plot(np.arange(max_itera), J_values)
  plt.show()

  plt.scatter(X_tilde[y == 1, 1], X_tilde[y == 1, 2], marker="x",
c ="red", alpha = 0.4)
  plt.scatter(X_tilde[y == -1, 1], X_tilde[y == -1, 2], marker
="o", facecolor = "none", edgecolors ="green", alpha = 0.3)

  hyerplane = lambda x: -x * theta[1] / theta[2] - theta[0] / thet
a[2]

  begin = hyerplane(np.min(X_tilde[:, 1]))
  end = hyerplane(np.max(X_tilde[:, 1]))

  plt.plot([np.min(X_tilde[:, 1]), np.max(X_tilde[:, 1])], [begin,
end])
  return J_values, theta

stochasticSubgradientMethod(X_tilde, y, 50)
```




```
Out[ ]: (array([22.82362422,  3.27781129, 15.25096146,  3.83667577,  2.807
        34846,
                 4.65318401,  4.76814831,  1.84552666,  5.67810148,  1.287
        81668,
                 2.59523057,  3.58692953,  2.20083148,  1.42060213,  1.465
        80991,
                 0.4494465 ,  0.65256208,  0.58660752,  0.49422353,  0.506
        8582 ,
                 2.60661447,  0.4165309 ,  1.44183498,  1.05708743,  0.578
        04881,
                 1.96225195,  1.5067882 ,  2.88374416,  2.1879576 ,  1.080
        87398,
                 1.26871614,  0.87984576,  1.10367946,  2.79993408,  0.548
        5905 ,
                 0.41737387,  1.11170578,  0.8755751 ,  0.71501795,  0.496
        23101,
                 2.09794503,  0.54420843,  1.68298497,  0.52781969,  0.251
        97609,
                 0.7203079 ,  0.6132065 ,  0.2561923 ,  0.88666322,  1.277
        01102]),
         array([ 5.00000000e-04, -1.38967327e+01,  2.36348274e+01]))
```



5

The convergence speed of stochastic subgradient method is visibely faster than the subgradient method