We want to solve the dual problem using the [CVXOPT](https://cvxopt.org/userguide/coneprog.html#quadratic-programming) library in python so we need to convert the dual problem to the standard form in the document:

Text

Description automatically generated

1. No slack variable:

The original dual problem is:

Text

Description automatically generatedText

Description automatically generated

Rewrite:

Text

Description automatically generated with low confidenceA picture containing text

Description automatically generated

So:

P = H

q = -1 vector

G =  diagonal matrix of -1s

h = 0 vector

A = y

B = 0 vector

def svm\_Slack\_dual(X,y):  
 #Initializing values and computing H. Note the 1. to force to float type  
 m,n = X.shape  
 y = y.reshape(-1,1) \* 1.  
 X\_dash = y \* X  
 H = np.dot(X\_dash , X\_dash.T) \* 1.  
  
 #Converting into cvxopt format  
 P = cvxopt\_matrix(H)  
 q = cvxopt\_matrix(-np.ones((m, 1)))  
 G = cvxopt\_matrix(-np.eye(m))  
 h = cvxopt\_matrix(np.zeros(m))  
 A = cvxopt\_matrix(y.reshape(1, -1))  
 b = cvxopt\_matrix(np.zeros(1))  
  
 #Setting solver parameters  
 cvxopt\_solvers.options['show\_progress'] = False  
 cvxopt\_solvers.options['abstol'] = 1e-10  
 cvxopt\_solvers.options['reltol'] = 1e-10  
 cvxopt\_solvers.options['feastol'] = 1e-10  
  
 sol = cvxopt\_solvers.qp(P, q, G, h, A, b)  
 alphas = np.array(sol['x'])  
 theta = ((y \* alphas).T @ X).reshape(-1,1)  
  
 threshold = 1e-5  
 S = (alphas > threshold).flatten()  
 theta0 = y[S] - np.dot(X[S], theta)  
 theta0 = theta0[0]  
  
 # print results  
 print('Alphas = ', alphas)  
 print('theta = ', theta.flatten())  
 print('theta0 = ', theta0)  
 return theta, theta0

Chart, line chart

Description automatically generated

1. For the situation with slack variable, the optimization problem is:

Text

Description automatically generated

Which could be rewrite as:

Text

Description automatically generated

We define G and h as follows to make it same as the standard form:

A picture containing diagram

Description automatically generated

X = np.array([[3,4],[1,4],[2,3],[6,-1],[7,-1],[5,-3],[2,4], [4,4]])  
y = np.array([-1, -1, -1, 1, 1, 1, 1, 1])  
  
x\_neg = X[0:3, ]  
x\_pos = X[3:X.shape[0], ]

def svm\_Slack\_dual(X, y, C=1):  
 #Initializing values and computing H. Note the 1. to force to float type  
 m,n = X.shape  
 y = y.reshape(-1, 1) \* 1.  
 X\_dash = y \* X  
 H = np.dot(X\_dash, X\_dash.T) \* 1.  
  
 #Converting into cvxopt format  
 P = cvxopt\_matrix(H)  
 q = cvxopt\_matrix(-np.ones((m, 1)))  
 G = cvxopt\_matrix(np.vstack((np.eye(m)\*-1, np.eye(m))))  
 h = cvxopt\_matrix(np.hstack((np.zeros(m), np.ones(m) \* C)))  
 A = cvxopt\_matrix(y.reshape(1, -1))  
 b = cvxopt\_matrix(np.zeros(1))  
  
 #Setting solver parameters  
 cvxopt\_solvers.options['show\_progress'] = False  
 cvxopt\_solvers.options['abstol'] = 1e-10  
 cvxopt\_solvers.options['reltol'] = 1e-10  
 cvxopt\_solvers.options['feastol'] = 1e-10  
  
 sol = cvxopt\_solvers.qp(P, q, G, h, A, b)  
 alphas = np.array(sol['x'])  
 theta = ((y \* alphas).T @ X).reshape(-1,1)  
  
 threshold = 1e-5  
 S = (alphas > threshold).flatten()  
 theta0 = y[S] - np.dot(X[S], theta)  
 theta0 = theta0[0]  
  
 # print results  
 print('Alphas = ', alphas)  
 print('theta = ', theta.flatten())  
 print('theta0 = ', theta0)  
 return theta, theta0

def plot\_decision\_boundary(X, y, theta0, theta):  
 x1 = np.array([min(X[:, 0]), max(X[:, 0])])  
 m = -theta[0] / theta[1]  
 c = -theta0 / theta[1]  
 x2 = m \* x1 + c  
  
 fig = plt.figure(figsize=(10, 8))  
 plt.scatter(x\_neg[:, 0], x\_neg[:, 1], marker='x', color='r', label='Negative -1')  
 plt.scatter(x\_pos[:, 0], x\_pos[:, 1], marker='o', color='b', label='Positive +1')  
 plt.xlabel("Feature 1")  
 plt.ylabel("Feature 2")  
 plt.title('SVM')  
 plt.plot(x1, x2, 'y-')  
  
  
results = svm\_Slack\_dual(X,y,C=1)  
beta = results[0]  
beta0 = results[1]  
plot\_decision\_boundary(X, y, beta0, beta)

Chart, scatter chart

Description automatically generated