```
import pickle
import matplotlib.pyplot as plt
import numpy as np

class HW3_Sol(object):

    def __init__(self):
        pass

    def load_data(self):
        self.x_train = pickle.load(open('x_train.p','rb'),
         encoding='latin1')
        self.y_train = pickle.load(open('y_train.p','rb'),
         encoding='latin1')
        self.x_test = pickle.load(open('x_test.p','rb'), encoding='latin1')
        self.y_test = pickle.load(open('y_test.p','rb'), encoding='latin1')

if __name__ == '__main__':

    hw3_sol = HW3_Sol()

    hw3_sol.load_data()

    # Your solution goes here
    #4(a)
    hw3_sol.x_train.astype(float)
    hw3_sol.y_train.astype(float)
    hw3_sol.x_test.astype(float)
    hw3_sol.y_test.astype(float)
    #Only choose one to plot each time
    #plot 0th image
    plt.contour(hw3_sol.x_train[0][0], hw3_sol.x_train[0][1],
     hw3_sol.x_train[0][2])
    #plot 10th image
    plt.contour(hw3_sol.x_train[10][0], hw3_sol.x_train[10][1],
     hw3_sol.x_train[10][2])
    #plot 20th image
    plt.contour(hw3_sol.x_train[20][0], hw3_sol.x_train[20][1],
     hw3_sol.x_train[20][2])
    print('0th image control vector:', hw3_sol.y_train[0])
    print('10th image control vector:', hw3_sol.y_train[10])
    print('20th image control vector:', hw3_sol.y_train[20])
    plt.show()

    #4(b)
    n = hw3_sol.x_train.shape[0]
    X = np.zeros((n,2700))
    for i in range(n):
        X[i] = hw3_sol.x_train[i].flatten()
    U = hw3_sol.y_train.reshape((n, 3))
    #PI = np.linalg.solve(X.T@X, X.T@U)

    #4(c)
    LAMBDA = [0.1, 1.0, 10.0, 100.0, 1000.0]
    errors = []
```

```
for i in range(5):
    PI = np.linalg.solve(X.T@X+LAMBDA[i]*np.identity(X.shape[1]), X.T@U)
    residual = X@PI - U
    error = np.zeros((n,1))
    for j in range(n):
        error[j] = residual[j][0]**2 + residual[j][1]**2 +
         residual[j][2]**2
    error = np.mean(error)
    errors += [error]
print('Average squared Euclidien distance for different lambda on
 training set:', errors)

#4(d)
scaled_errors = []
scaled_X = X*2/255-1
for i in range(5):
    PI_scaled =
     np.linalg.solve(scaled_X.T@scaled_X+LAMBDA
     [i]*np.identity(scaled_X.shape[1]), scaled_X.T@U)
    residual_scaled = scaled_X@PI_scaled - U
    scaled_error = np.zeros((n,1))
    for j in range(n):
        scaled_error[j] = residual_scaled[j][0]**2 +
         residual_scaled[j][1]**2 + residual_scaled[j][2]**2
    scaled_error = np.mean(scaled_error)
    scaled_errors += [scaled_error]
print('Average squared Euclidien distance for different lambda on
 training set after scale:', scaled_errors)

#4(e)
n_test = hw3_sol.x_test.shape[0]
X_test = np.zeros((n_test,2700))
for i in range(n_test):
    X_test[i] = hw3_sol.x_test[i].flatten()
U_test = hw3_sol.y_test.reshape((n_test, 3))
X_test_scaled = X_test*2/255-1
errors_test = []
scaled_errors_test = []
for i in range(5):
    PI =
     np.linalg.solve(X_test.T@X_test+LAMBDA[i]*np.identity(X_test.shape
     [1]), X_test.T@U_test)
    PI_scaled =
     np.linalg.solve(X_test_scaled.T@X_test_scaled+LAMBDA
     [i]*np.identity(X_test_scaled.shape[1]), X_test_scaled.T@U_test)
    residual_test = X_test@PI - U_test
    residual_scaled_test = X_test_scaled@PI_scaled - U_test
    error_test = np.zeros((n_test, 1))
    scaled_error_test = np.zeros((n_test, 1))
    for j in range(n_test):
        error_test[j] = residual_test[j][0]**2 + residual_test[j][1]**2
         + residual_test[j][2]**2
        scaled_error_test[j] = residual_scaled_test[j][0]**2 +
         residual_scaled_test[j][1]**2 + residual_scaled_test[j][2]**2
    error_test = np.mean(error_test)
```

```
    scaled_error_test = np.mean(scaled_error_test)
    errors_test += [error_test]
    scaled_errors_test += [scaled_error_test]
print('Average squared Euclidien distance for different lambda on test
 set:', errors_test)
print('Average squared Euclidien distance for different lambda on test
 set after scale:', scaled_errors_test)

#4(f)
u_train, s_train, v_train = np.linalg.svd(X.T@X +
 100*np.identity(X.shape[1]))
u_train_scaled, s_train_scaled, v_train_scaled =
 np.linalg.svd(scaled_X.T@scaled_X + 100*np.identity(scaled_X.shape[1]))
u_test, s_test, v_test = np.linalg.svd(X_test.T@X_test +
 100*np.identity(X_test.shape[1]))
u_test_scaled, s_test_scaled, v_test_scaled =
 np.linalg.svd(X_test_scaled.T@X_test_scaled +
 np.identity(X_test_scaled.shape[1]))
k_train = np.amax(s_train) / np.amin(s_train_scaled)
k_train_scaled = np.amax(s_train_scaled) / np.amin(s_train_scaled)
k_test = np.amax(s_test) / np.amin(s_test)
k_test_scaled = np.amax(s_test_scaled) / np.amin(s_test_scaled)
print('Condition number of training data without standardization:',
 k_train)
print('Condition number of training data with standardization:',
 k_train_scaled)
print('Condition number of test data without standardization:', k_test)
print('Condition number of test data with standardization:',
 k_test_scaled)
```