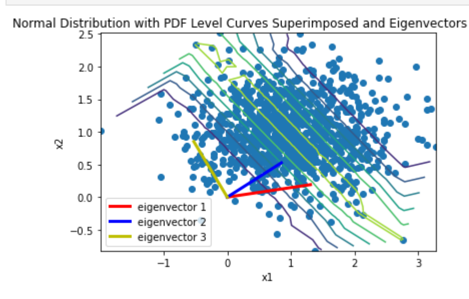
```
import numpy as np
[illegible]
import matplotlib.pyplot as plt
[illegible]
from scipy.io import loadmat
[illegible]
```

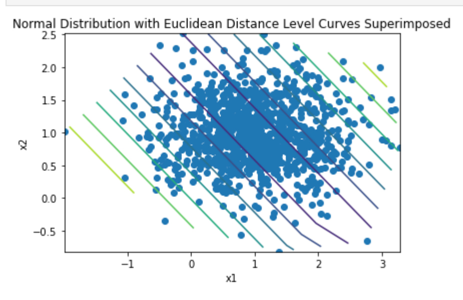
```
#2a
[illegible]
```

Normal Distribution with PDF Level Curves Superimposed and Eigenvectors Graphed



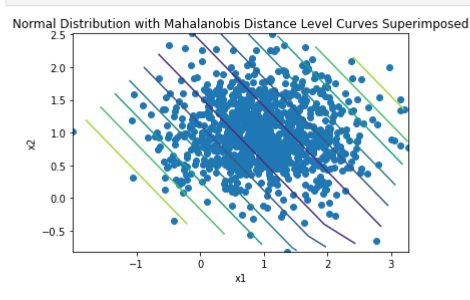
In [ ]:
```
#2c

#calculate Euclidean distance
[illegible]

plt.title('Normal Distribution with Euclidean Distance Level Curves Superimposed')
[illegible]

plt.show()
```

Normal Distribution with Euclidean Distance Level Curves Superimposed



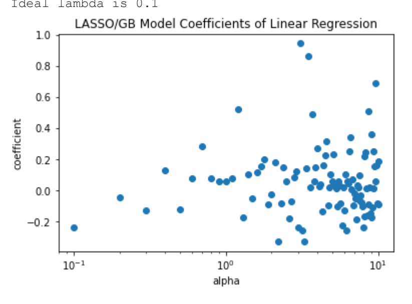
In [ ]:
```
#2d
#calculate Mahalanobis distance
[illegible]

plt.title('Normal Distribution with Mahalanobis Distance Level Curves Superimposed')
[illegible]

plt.show()
```

Normal Distribution with Mahalanobis Distance Level Curves Superimposed

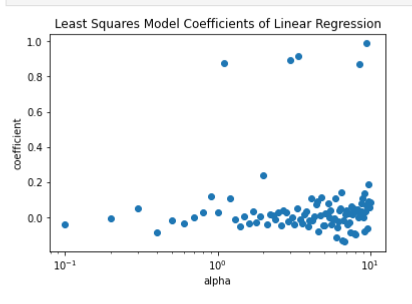


In [ ]:
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from scipy.io import loadmat
import pandas as pd

#3a

#Grad Descent Algo

#Loss function
def loss(y,ypred):
    loss=(y-ypred)**2
    return(loss)

[illegible]

#load in samples
[illegible]

[illegible]
ax.set_xscale('log')

plt.title('LASSO/GB Model Coefficients of Linear Regression')
plt.xlabel('alpha')
plt.ylabel('coefficient')

plt.axis('tight')
plt.show()
```

Ideal lambda is 0.1



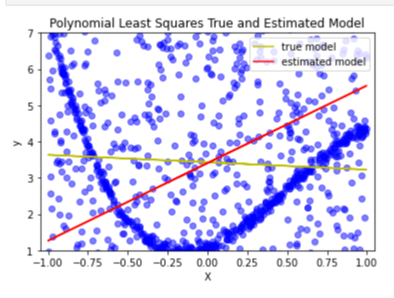
In [28]:
```
from sklearn import linear_model

#3b

#Train linear model
alphaList_ls = [l*0.1 for l in range(1,101)]

t = linear_model.LinearRegression()
x_delete_train = np.delete(x_train, 0, 1) #reshape/delete 1. column so we can match train and test sets

model = t.fit(x_delete_train, y_train)

#model coefficients
coefficients_ls = model.coef_


#plot coefficients

ax = plt.gca()

ax.scatter(alphaList_ls, coefficients_ls)
ax.set_xscale('log')

plt.title('Least Squares Model Coefficients of Linear Regression')
plt.xlabel('alpha')
plt.ylabel('coefficient')

plt.axis('tight')
plt.show()
```



In [29]:
```
#3c

#Lasso

#MSE on test data
#best y predcitions from LASSO
Y_pred_Lasso = lassoModel.predict(x_test).reshape(-1,1)
Y_true_Lasso = y_test

MSE_Lasso = np.square(np.subtract(Y_true_Lasso, Y_pred_Lasso)).mean()

#Least Squares

#MSE on test data
#best y predictions from LS
intercept = model.intercept_
slope = model.coef_

for i in range(len(x_test)): #y = mx+b
    Y_pred_LS = intercept + slope * x_test[i]
Y_true_LS = y_test

MSE_LS = np.square(np.subtract(Y_true_LS, Y_pred_LS)).mean()


print(f"The mean square error on test set (true vs predicted output) for LASSO is {MSE_Lasso} compared to {MSE_
```

```
The mean square error on test set (true vs predicted output) for LASSO is 6.252351768583674 compared to 8.08601
7848092585 for least squares ... hence as LASSO produces a lower MSE, it is a model with better fit
```

In [30]:
```
#4a

#using 1000 samples

#load in samples
hw4 = loadmat('HW1_4.mat')
hw4.keys() #check data

xdict = hw4['X']
ydict = hw4['Y']
alpha = hw4['alpha']

#convert dict to np.array
xlist = list(xdict)
ylist = list(ydict)
X = np.array(xlist)
ytrue = np.array(ylist)

#change to 1D arrays to plot
X_1D = X[:,0]
ytrue_1D = ytrue[:,0]

e = np.random.normal(loc = 0, scale = 0.01) #gaussian noise

#added gausian noise
ytrue_1D = ytrue_1D + e
degree = 3

#plot samples
plt.scatter(X_1D, ytrue_1D, color = 'b', alpha = 0.5)

#plot true model
slope, intercept, r_value, p_value, std_err = stats.linregress(X[:,0], ytrue[:,0])
#mtrue = linregress(x, ytrue) #to find slope mtrue[0] and intercept mtrue[1] of true
plt.plot(X_1D, slope*X_1D + intercept, '-y', label = 'true model') #plotting y = mx+b

#estimated model with poly least squares (polyfit)

polyfit = np.polyfit(X_1D, ytrue_1D, degree)
yestimate = -polyfit[0]*X**3 + polyfit[1]*X**2 + polyfit[2]*X + polyfit[3]
mest = stats.linregress(X[:,0], yestimate[:,0])

#plot estimated model
plt.plot(X_1D, mest[0]*X_1D + mest[1], '-r', label = 'estimated model')

plt.title('Polynomial Least Squares True and Estimated Model')
plt.xlabel('X')
plt.ylabel('y')

plt.axis([-1.05, 1.05, 1, 7])
plt.legend()
plt.show()
```



In [33]:
```
#4b

#combine X and ytrue

data = np.column_stack([X_1D,ytrue_1D])

#fit line using all data
model = LineModelND()
model.estimate(data)

#implement RANSAC with fitted line data to get model_r
model_r, inliers = ransac(data, LineModelND, min_samples = 2, residual_threshold = 1, max_trials = 1000)
outliers = inliers == False #separate out the outliers from inliers

#generate coordinates
ytrue= model.predict_y(X_1D)
yestimate = model_r.predict_y(X_1D)


#finding consensus set
randset = []
con = [] #consensus set
inliers_count = 0

loop = 5


for i in range(loop): #tried my best on this consensus set implementation
    randset.append(data[rand.randint(0, 1000)])

    if randset[i][1] - ytrue[i]  <= 0.3: #threshold
        con.append(randset[i])
        if data[inliers][i][0] == randset[i][0] and data[inliers][i][1]== randset[i][1]:
            inliers_count+=1
    if len(con) > 0:
        if inliers_count/len(con) * 100 > 40: #inliers exceed 40%
            break
    else:
        continue


    loop+=1



#plotting consensus set
for i in range(len(con)):
    con_set = plt.plot(con[i][0], con[i][1], '.y', alpha=1, markersize = 30) #consensus set


plt.plot(data[inliers, 0], data[inliers, 1], '.r', alpha=0.6, label = 'Inliers') #inliers
plt.plot(data[outliers, 0], data[outliers, 1], '.b', alpha=0.6, label = 'Outliers')#outliers

plt.plot(X_1D, ytrue, '-y', label='true model')
plt.plot(X_1D, yestimate, '-r', label='estimated model')

plt.title('RANSAC True and Estimated Model with Consensus Set (Yellow Circles)')
plt.xlabel('X')
plt.ylabel('y')

plt.axis([-1.05, 1.05, 1, 7])
plt.legend()
plt.show()
```