Q1

Answer

(a)

根據我們所建立的regression model (詳細資訊如下所示)，我們可以評估模型下的B1約為4.561e-07 ；B2則為6.916e-07

(b)

When B1 + B2 = 1, we estimate B1 = 3.749765918236159 and B2= -2.749765918236159

In [113]:
```
# import packages needed
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
import statsmodels.api as sm
import scipy
```

In [111]:
```
# prepare the data
# create original dataframe
data_FRB = pd.DataFrame({
    'Year':[i for i in range(72, 87, 1)],
    'Capital':[1209188, 1330372, 1157371, 1070860, 1233475, 1355769, 1351667, 1326248, 1089545, 1111942,
              988165, 1069651, 1191677, 1246536, 1281262],
    'Labor':[1259142, 1371795, 1263084, 1118226, 1274345, 1369877, 1451595, 1328683, 1077207, 1056231, 947502,
            1057159, 1169442, 1195255, 1171664],
    'Value_Added':[11150.0, 12853.6, 10450.8, 9318.3, 12097.7, 12844.8, 13309.9, 13402.3, 8571.0, 8739.7,
                  8140.0, 10958.4, 10838.9, 10030.5, 10836.5]
})

# append 'log_Value_Added' as Value_added with log function applied
data_FRB.loc[:, 'log_Value_Added'] = np.log(data_FRB.loc[:, 'Value_Added'])

# divide data into X and y
X = data_FRB.iloc[:, 1:-2]
y = data_FRB.iloc[:, -1]
```

In [112]:
```
# train with multiple regression

# create a statsmodel
# create statsmodels model
X = sm.add_constant(X)
model = sm.regression.linear_model.OLS(y, X).fit()

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        log_Value_Added   R-squared:                       0.823
Model:                            OLS   Adj. R-squared:                  0.794
Method:                 Least Squares   F-statistic:                     27.91
Date:                Thu, 09 Apr 2020   Prob (F-statistic):           3.07e-05
Time:                        21:02:00   Log-Likelihood:                 19.449
No. Observations:                  15   AIC:                            -32.90
Df Residuals:                      12   BIC:                            -30.77
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          7.9019      0.208     37.941      0.000       7.448       8.356
Capital     4.561e-07   4.13e-07      1.106      0.291   -4.43e-07    1.35e-06
Labor       6.916e-07   3.41e-07      2.028      0.065   -5.16e-08    1.43e-06
==============================================================================
Omnibus:                        7.893   Durbin-Watson:                   2.002
Prob(Omnibus):                  0.019   Jarque-Bera (JB):                4.398
Skew:                           1.140   Prob(JB):                        0.111
Kurtosis:                       4.356   Cond. No.                     1.87e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.87e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
C:\Users\ricardo\Anaconda3\lib\site-packages\scipy\stats\stats.py:1535: UserWarning: kurtosistest only valid for n>=2
0 ... continuing anyway, n=15
  "anyway, n=%i" % int(n))
```

In [152]:
```
def sm_model(B1):
    return np.sum( ((X_1 * B1 + X_2 * B2) - y) ** 2 )

sm_model = sm_model(1)
```

In [165]:
```
y = data_FRB.iloc[:, -1].to_numpy()
X_1 = data_FRB.iloc[:, 1].to_numpy()
X_2 = data_FRB.iloc[:, 2].to_numpy()

def sm_model(B1):
    return np.sum( ((X_1 * B1 + X_2 * (1-B1)) - y) ** 2 )

# optimize the parameter with scipy
res = scipy.optimize.minimize(sm_model, x0=[0])
print('When B1 + B2 = 1, we estimate B1 = ', res.x[0], 'and B2=', (1- res.x[0]))
```

```
When B1 + B2 = 1, we estimate B1 =  3.749765918236159 and B2= -2.749765918236159
```

Q2

Answer

(a) 根據我的程式碼，Lambda在越小時，MSE同有下降的趨勢，並在Lambda=0.0時達到最小，MSE=1.8042175649346816e-08；使用stepwise regression得到MSE=0.014399127818860755為；相比較發現LASSO regression方法的預測效率較佳

(b) 如本題最下方程式碼所產生的圖所示。

In [1]:
```
# import packages needed
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
```

In [2]:
```
# prepare the data
ORL_data = pd.read_csv('./data/ORL_data.csv')
# divide into X and y
X = ORL_data.iloc[:, :-1]
y = ORL_data.iloc[:, -1].tolist()
```

In [6]:
```
# implement with Lasso regression (with sklearn package)

lambda_list = np.arange(0, 10.1, 0.1)
mse_list = []

for i, j in enumerate(lambda_list):
    reg_lasso = linear_model.Lasso(alpha=j)
    reg_lasso.fit(X, y)
    predict_list_original = reg_lasso.predict(X).tolist()
    mse_list.append(mean_squared_error(y, predict_list_original))

minimum_value = np.min(mse_list)
minimum_index_list = [i for i, j in enumerate(mse_list) if j == minimum_value]
# print(mse_list)
# print(minimum_index_list)
print('MSE is lowest when lambda=', lambda_list[minimum_index_list[0]],
      ', while MSE is ', mse_list[minimum_index_list[0]])
```

```
C:\Users\ricardo\AppData\Roaming\Python\Python37\site-packages\ipykernel_launcher.py:8: UserWarning: With alpha=0, th
is algorithm does not converge well. You are advised to use the LinearRegression estimator

C:\Users\ricardo\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: UserWarning: Coordinate
descent with no regularization may lead to unexpected results and is discouraged.
  positive)
C:\Users\ricardo\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: Obj
ective did not converge. You might want to increase the number of iterations. Duality gap: 0.025962355545366744, tole
rance: 0.004375
  positive)
```

```
MSE is lowest when lambda= 0.0 , while MSE is  1.8042175649346816e-08
```

In [7]:
```
# implement with Ridge regression (with sklearn package)
lambda_list = np.arange(0, 10.1, 0.1)
mse_list = []

for i, j in enumerate(lambda_list):
    reg_ridge = linear_model.Ridge(alpha=j)
    reg_ridge.fit(X, y)
    predict_list_original = reg_ridge.predict(X).tolist()
    mse_list.append(mean_squared_error(y, predict_list_original))

minimum_value = np.min(mse_list)
minimum_index_list = [i for i, j in enumerate(mse_list) if j == minimum_value]
# print(mse_list)
# print(minimum_index_list)
print('MSE is lowest when lambda=', lambda_list[minimum_index_list[0]],
      ', while MSE is ', mse_list[minimum_index_list[0]])
```

```
C:\Users\ricardo\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:188: LinAlgWarning: Ill-conditioned matri
x (rcond=6.20387e-19): result may not be accurate.
  overwrite_a=False)
```

```
MSE is lowest when lambda= 0.0 , while MSE is  3.033838863451482e-29
```

In [11]:
```
# To compare with stepwise regression, we select LASSO regression with lambda = 0, 0.001, and 1
# We use statsmodels package this time.
# Reference:
# Documentation of penalized (regularized) regression in statsmodels (Elastic Net)
# https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.fit_regularized.html

lambda_list = [0, 0.001, 1]
L1_wt = 1 # 1 when LASSO fit, 0 when ridge fit

# create statsmodels model
model = sm.regression.linear_model.OLS(y, X)

for i, j in enumerate(lambda_list):
    result_LASSO = model.fit_regularized(L1_wt=L1_wt, alpha=float(j))
    predict_list_original = result_LASSO.fittedvalues.tolist()
    mse_value = mean_squared_error(y, predict_list_original)
    print('MSE when lambda=', j,
      'is: ', mse_value)
```

```
MSE when lambda= 0 is:  0.03492422008163612
MSE when lambda= 0.001 is:  0.034903909423569794
MSE when lambda= 1 is:  0.06668417375325184
```

In [14]:
```
# create stepwise regression function

def stepwise_selection(X, y, initial_list=[], threshold_in=0.01, threshold_out=0.05, verbose=True):
    """ Perform a forward-backward feature selection
    Reference: # https://www.twblogs.net/a/5c13a86fbd9eee5e40bb7431
    based on p-value from statsmodels.api.OLS
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite looping.
    """
    included = list(initial_list)

    while True:
        changed = False
        # forward step
        excluded = list(set(X.columns) - set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included + [new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed = True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

            # backward step
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included].values))).fit()
            print('R_square = ', model.rsquared)
            # use all coefs except intercept
            pvalues = model.pvalues.iloc[1:]
            worst_pval = pvalues.max()  # null if pvalues is empty
            if worst_pval > threshold_out:
                changed = True
                worst_feature = pvalues.idxmax()
                included.remove(included[worst_feature])
                if verbose:
                    print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
            if not changed:
                break
    return included
```

In [ ]:
```
# implement stepwise regression model
included_features = stepwise_selection(X, y)
print('resulting features:')
print(included_features)
print('Number of included feincluded_featurees is: ', len(included_features))
```

```
C:\Users\ricardo\AppData\Roaming\Python\Python37\site-packages\ipykernel_launcher.py:23: DeprecationWarning: The defa
ult dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to s
ilence this warning.
```

```
Add  pixel_1470                     with p-value 3.91841e-14
R_square =  0.13404309865692476
Add  pixel_2488                     with p-value 3.38413e-16
R_square =  0.2678723779962152
Add  pixel_2271                     with p-value 6.02768e-11
R_square =  0.34298316156352904
Add  pixel_1722                     with p-value 7.004e-08
R_square =  0.38964653438298524
Add  pixel_555                      with p-value 1.31884e-07
R_square =  0.4313340061910136
Add  pixel_1474                     with p-value 5.65748e-06
R_square =  0.46041103180240095
Add  pixel_1386                     with p-value 8.20976e-06
R_square =  0.48713763178062175
Add  pixel_1476                     with p-value 6.706e-07
R_square =  0.5185697594190422
Add  pixel_1051                     with p-value 6.99533e-06
R_square =  0.5428944164631441
Add  pixel_1561                     with p-value 1.92879e-05
R_square =  0.5638799029382524
Add  pixel_202                      with p-value 1.4955e-05
R_square =  0.5844727398029121
Add  pixel_133                      with p-value 1.23609e-05
R_square =  0.6045148272306424
Add  pixel_186                      with p-value 6.96418e-12
R_square =  0.6499422370848498
Add  pixel_1687                     with p-value 3.21537e-06
R_square =  0.6691369901454154
Add  pixel_319                      with p-value 3.01682e-06
R_square =  0.6874248986626645
Add  pixel_2393                     with p-value 7.13528e-05
R_square =  0.7000535030407766
Add  pixel_1984                     with p-value 3.73409e-06
R_square =  0.7164130615202478
Add  pixel_1325                     with p-value 6.84276e-05
R_square =  0.7279861021848284
Add  pixel_343                      with p-value 9.94106e-05
R_square =  0.7386297350875746
Add  pixel_325                      with p-value 0.000562402
R_square =  0.7467169625438768
Add  pixel_1730                     with p-value 0.000406945
R_square =  0.7549664661083192
Add  pixel_546                      with p-value 0.000298487
R_square =  0.7633328875127459
Add  pixel_380                      with p-value 4.57692e-05
R_square =  0.7735769523121608
Add  pixel_183                      with p-value 0.000308109
R_square =  0.7813138125011371
Add  pixel_95                       with p-value 3.63205e-05
R_square =  0.7910747573914318
Drop                             12 with p-value 0.606526
Add  pixel_1180                     with p-value 3.86642e-05
R_square =  0.8001947167154811
Add  pixel_2158                     with p-value 9.85121e-05
R_square =  0.8081656771085574
Add  pixel_2208                     with p-value 3.67526e-05
R_square =  0.8167620724289808
Add  pixel_2168                     with p-value 0.000133239
R_square =  0.8238401408803954
Add  pixel_1935                     with p-value 0.000485927
R_square =  0.8295471736784147
Add  pixel_2117                     with p-value 0.000222031
R_square =  0.8357387833553851
Add  pixel_430                      with p-value 0.000142728
R_square =  0.8420789131635588
Drop                             17 with p-value 0.0709408
Add  pixel_2516                     with p-value 0.000718295
R_square =  0.8455552264931512
Add  pixel_1260                     with p-value 0.00163652
R_square =  0.8496787374970376
Add  pixel_1809                     with p-value 0.00187537
R_square =  0.853602684916661
Add  pixel_1565                     with p-value 0.000980758
R_square =  0.8579015069074752
Add  pixel_845                      with p-value 0.00346772
R_square =  0.861202327963152
Add  pixel_1366                     with p-value 0.00615045
R_square =  0.8640466240370939
Add  pixel_414                      with p-value 0.00446665
R_square =  0.8670529819217331
Add  pixel_2121                     with p-value 0.00397132
R_square =  0.8700777953085174
Add  pixel_1936                     with p-value 0.00453467
R_square =  0.8729569119610687
```

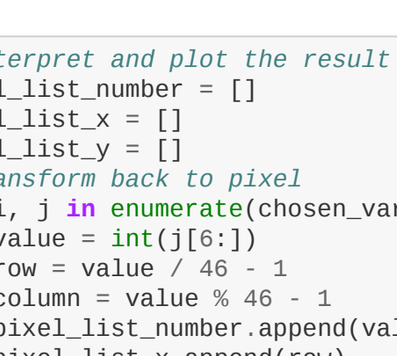
In [9]:
```
# create new_dataset with feature selected
included_features = ['pixel_1470', 'pixel_2488', 'pixel_2271', 'pixel_1722', 'pixel_555', 'pixel_1474', \
 'pixel_1386', 'pixel_1476', 'pixel_1051', 'pixel_1561', 'pixel_202', 'pixel_133',  \
 'pixel_1687', 'pixel_319', 'pixel_2393', 'pixel_1984', 'pixel_1325', 'pixel_325', 'pixel_1730', \
 'pixel_546', 'pixel_380', 'pixel_183', 'pixel_95', 'pixel_1180', 'pixel_2158', 'pixel_2208', 'pixel_2168', \
 'pixel_1935', 'pixel_2117', 'pixel_430', 'pixel_2516', 'pixel_1260 ', 'pixel_1809', 'pixel_1565', 'pixel_845', \
 'pixel_1366', 'pixel_414', 'pixel_2121', 'pixel_1936']


# prepare the data
X = ORL_data[ORL_data.columns & included_features]
# included_featuresde into X and y
y = ORL_data.iloc[:, -1].tolist()
# calculate mse with features selected by stepwise regression
reg = linear_model.LinearRegression()
reg.fit(X, y)
predict_list = reg.predict(X)
mse_value = mean_squared_error(y, predict_list)
print('MSE=', mse_value)
```

```
MSE= 0.014399127818860755
```

In [13]:
```
# Implement LASSO model when setting lambda=1
result_LASSO = model.fit_regularized(L1_wt=1, alpha=1)
predict_list_original = result_LASSO.fittedvalues.tolist()
mse_value = mean_squared_error(y, predict_list_original)
print('MSE when lambda=', j, 'is: ', mse_value)

# interpret the result of LASSO
param_data_LASSO = pd.DataFrame({
    'Pixel':result_LASSO.params.index.tolist(),
    'Params':result_LASSO.params.values.tolist()
})

param_data_LASSO = param_data_LASSO.sort_values(by=['Params'], ascending=False)
chosen_vars_LASSO = param_data_LASSO.iloc[0:54, 0].values.tolist()
```

```
MSE when lambda= 1 is:  0.06668417375325184
```

In [14]:
```
# Interpret and plot the result
pixel_list_number = []
pixel_list_x = []
pixel_list_y = []
# transform back to pixel
for i, j in enumerate(chosen_vars_LASSO):
    value = int(j[6:])
    row = value / 46 - 1
    column = value % 46 - 1
    pixel_list_number.append(value)
    pixel_list_x.append(row)
    pixel_list_y.append(column)
    file = "./data/ORL_Faces_Classified/0/1_1.png"
    sample = cv2.imread(file, cv2.IMREAD_GRAYSCALE)
    for i, j in enumerate(pixel_list_x):
        x_value = int(j - 1)
        y_value = int(pixel_list_y[i] - 1)
        sample[x_value, y_value] = 0
# cv2.imwrite( "./sample.png", sample)
imgplot = plt.imshow(sample, cmap='gray', vmin=0, vmax=255)
plt.show()
```



Q3

Answer

(a)

如下方程式碼所示

(b)

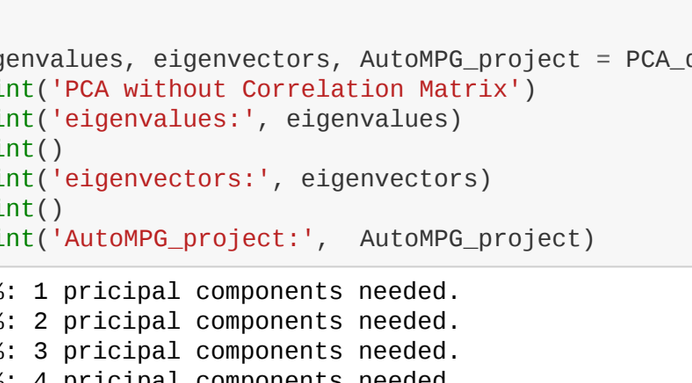
我們在做PCA的時候，是找向量來maximize variance，但我們的X只有做center沒有做標準化，所以會被各個variable的scale影響，所以是 scale variant 的，其受 scale 影響的情況並可從下方程式馬所output的圖形得知。

In [30]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

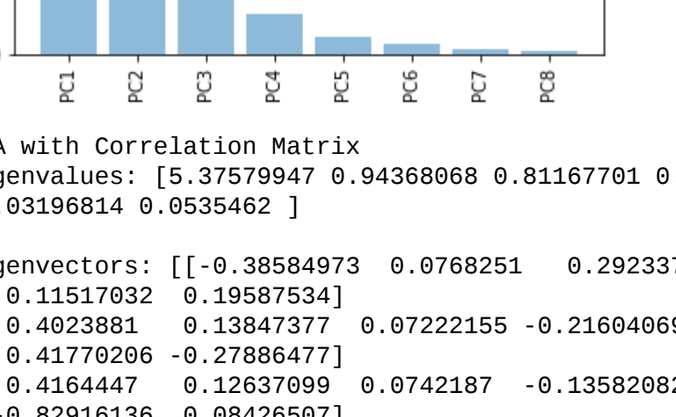
In [105]:
```
def PCA_decomposition(X, isCorrMX):
    # Reference: https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/
    # calculate mean of the array
    X_mean = np.mean(X.T, axis=1)
    # center the values
    X_center = X - X_mean
    # check if use Correlation Matrix
    if isCorrMX == True:
        X_covariance = np.corrcoef(X_center.T)
    else:
        X_covariance = np.cov(X_center.T)
    # calculate eigenvalues and eigenvectors
    # https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.eig.html
    eigenvalues, eigenvectors = np.linalg.eig(X_covariance)
    X_project = -(eigenvectors.T.dot(X_center.T).T)

    # calculate explainable ratio of each pricipal components
    explainable_ratio_list = []
    for i in range(eigenvalues.shape[0]):
        eigenvalues_totals = np.sum(eigenvalues)
        ratio = eigenvalues[i] / eigenvalues_totals
        explainable_ratio_list.append(ratio)

    # calculate how many principal components needed to explain 50, 60, 70, 80, 90% of total variance
    total_explainable_ratio = 0
    total_explainable_ratio_list = []
    culmulative_explainable_ratio_list = []
    components_needed_list = []
    for i, j in enumerate(explainable_ratio_list):
        total_explainable_ratio += j
        total_explainable_ratio_list.append(total_explainable_ratio)
        culmulative_explainable_ratio_list.append(np.sum(explainable_ratio_list[(i):]))

    total_explainable_ratio = 0
    for i, j in enumerate(explainable_ratio_list):
        total_explainable_ratio += j
        if total_explainable_ratio >= 0.9:
            if len(components_needed_list) < 5:
                components_needed_list.append(i + 1)
                print('90%:', str(i + 1), 'pricipal components needed.')
            break
        elif total_explainable_ratio >= 0.8:
            if len(components_needed_list) < 4:
                components_needed_list.append(i + 1)
                print('80%:', str(i + 1), 'pricipal components needed.')
        elif total_explainable_ratio >= 0.7:
            if len(components_needed_list) < 3:
                components_needed_list.append(i + 1)
                print('70%:', str(i + 1), 'pricipal components needed.')
        elif total_explainable_ratio >= 0.6:
            if len(components_needed_list) < 2:
                components_needed_list.append(i + 1)
                print('60%:', str(i + 1), 'pricipal components needed.')
        elif total_explainable_ratio >= 0.5:
            if len(components_needed_list) < 1:
                components_needed_list.append(i + 1)
                print('50%:', str(i + 1), 'pricipal components needed.')

    # Plot the scree plot
    if len(explainable_ratio_list) <= 15:
        x_list = ['PC' + str(i) for i in range(1, len(explainable_ratio_list) + 1)]
        plt.plot(total_explainable_ratio_list)
        plt.bar(x_list, culmulative_explainable_ratio_list, align='center', alpha=0.5)
    else:
        x_list = ['PC' + str(i) for i in range(1, 16)]
        plt.plot(total_explainable_ratio_list[:15])
        plt.bar(x_list, culmulative_explainable_ratio_list[:15], align='center', alpha=0.5)
    plt.xticks(rotation=90)
    plt.show()

    return eigenvalues, eigenvectors, X_project
```

In [170]:
```
# implement PCA with AutoMPG dataset
AutoMPG = pd.read_csv('./data/AutoMPG.csv')

eigenvalues, eigenvectors, AutoMPG_project = PCA_decomposition(AutoMPG, True)
print('PCA with Correlation Matrix')
print('eigenvalues:', eigenvalues)
print()
print('eigenvectors:', eigenvectors)
print()
print('AutoMPG_project:',  AutoMPG_project)



eigenvalues, eigenvectors, AutoMPG_project = PCA_decomposition(AutoMPG, False)
print('PCA without Correlation Matrix')
print('eigenvalues:', eigenvalues)
print()
print('eigenvectors:', eigenvectors)
print()
print('AutoMPG_project:',  AutoMPG_project)
```

```
60%: 1 pricipal components needed.
70%: 2 pricipal components needed.
80%: 3 pricipal components needed.
90%: 4 pricipal components needed.
```




```
PCA with Correlation Matrix
eigenvalues: [5.37579947 0.94368068 0.81167701 0.48615382 0.18286556 0.11430911
 0.03196814 0.0535462 ]

eigenvectors: [[-0.38584973  0.0768251   0.29233712  0.09995932 -0.74033405  0.38734891
   0.11517032  0.19587534]
 [ 0.4023881   0.13847377  0.07222155 -0.21604069 -0.48258829 -0.53093056
   0.41770206 -0.27886477]
 [ 0.4164447   0.12637099  0.0742187  -0.13582082 -0.30329986 -0.00696426
  -0.82916136  0.08426507]
 [ 0.4018397  -0.11139899  0.23608904 -0.11972353  0.08431898  0.66672875
   0.13477375 -0.53501633]
 [ 0.40157692  0.21103513 -0.00094958 -0.32248018  0.1312946   0.23577482
   0.30996269  0.72201169]
 [-0.2647346   0.41672193 -0.63954802 -0.49281516 -0.09771744  0.20294159
  -0.03519452 -0.22890465]
 [-0.21387799  0.6905824   0.58696339 -0.10603199  0.30146799 -0.11007417
  -0.05431804 -0.12503449]
 [-0.2778673  -0.50144442  0.30745409 -0.74327015  0.04736686 -0.12085714
  -0.07951279  0.03453075]]

AutoMPG_project: [[-272.83046976 -117.33197165  -11.96660362 ... -138.12273605
   -74.54082255 -376.28573339]
 [-381.32241659 -158.44667041  -23.19016013 ... -205.12681707
  -102.05824478 -497.50945919]
 [-258.40565926 -101.72696191  -18.20890927 ... -135.14507474
   -47.07325409 -317.6444326 ]
 ...
 [ 309.15514682  146.39744394    1.76117171 ...  173.37431414
   165.41119263  485.63717067]
 [ 186.08782859   75.30797004    9.34206647 ...   98.03570443
    51.80210414  247.8975295 ]
 [ 147.85462174   55.28848696    8.93567461 ...   72.97178584
    21.0029035   180.92813672]]
90%: 1 pricipal components needed.
```




```
PCA without Correlation Matrix
eigenvalues: [7.32159308e+05 1.51405465e+03 2.61318016e+02 1.23093098e+01
 2.90140740e+00 1.88860286e+00 3.57475600e-01 2.57690315e-01]

eigenvectors: [[-3.22887929e-03 -7.39571693e-03 -1.68367301e-02  3.58097463e-01
   1.37761080e-01  9.14674563e-01 -1.07540868e-01  6.51528870e-02]
 [ 1.79261370e-03  1.33241105e-02 -7.29020365e-03 -2.55793640e-03
  -1.87354959e-02 -1.58567937e-02  3.90300201e-01  9.20229787e-01]
 [ 1.14340720e-01  9.45741882e-01 -3.03364700e-01  9.48215207e-03
   1.03606815e-02 -1.56514533e-03  6.34075260e-04 -1.63781474e-02]
 [ 3.89668646e-02  2.98262863e-01  9.48521534e-01  4.74430424e-02
   8.52720884e-02 -1.29726089e-02 -8.28994320e-03  8.28033286e-03]
 [ 9.92668234e-01 -1.20773235e-01 -2.48833009e-03 -5.94826172e-04
  -2.71266838e-03  3.10360543e-03 -2.33657118e-04 -1.09049120e-04]
 [-1.35283053e-03 -3.48265951e-02 -7.69835369e-02 -5.00497666e-02
   9.86437038e-01 -1.30450550e-01  1.00203344e-02  1.33435475e-02]
 [-1.33685177e-03 -2.38692393e-02 -4.30407932e-02  9.31043932e-01
  -4.59157658e-03 -3.59349460e-01  3.51338495e-02 -1.85917655e-02]
 [-5.51536058e-04 -3.24365681e-03  1.24486303e-02  8.41269228e-03
   1.43429869e-02  1.29573354e-01  9.13617393e-01 -3.84800879e-01]]

AutoMPG_project: [[-5.36449619e+02 -5.08358444e+01  1.07053844e+01 ... -1.50243791e+00
  -2.00383876e-01 -7.71177669e-01]
 [-7.30349183e+02 -7.91426100e+01 -9.03776726e+00 ... -4.71265361e-01
  -2.49066403e-02 -2.46702756e-01]
 [-4.70986617e+02 -7.54516690e+01 -5.17422460e+00 ... -1.14517452e+00
  -4.74281894e-02 -7.50696497e-01]
 ...
 [ 6.85187239e+02 -2.00931891e+01 -2.93030701e-01 ...  1.38148524e-01
   1.02831913e+00  1.82533974e-01]
 [ 3.59520648e+02  3.56706345e+01  1.23051772e+00 ...  1.49371911e+00
   8.20525580e-01  3.16047114e-02]
 [ 2.65219821e+02  4.72323729e+01 -1.59888239e+00 ...  1.69806277e-01
   9.97862955e-01 -9.33253015e-02]]
```

Q4

Answer

(a)

50%: 2 pricipal components needed.

60%: 3 pricipal components needed.

70%: 6 pricipal components needed.

80%: 15 pricipal components needed.

90%: 47 pricipal components needed.

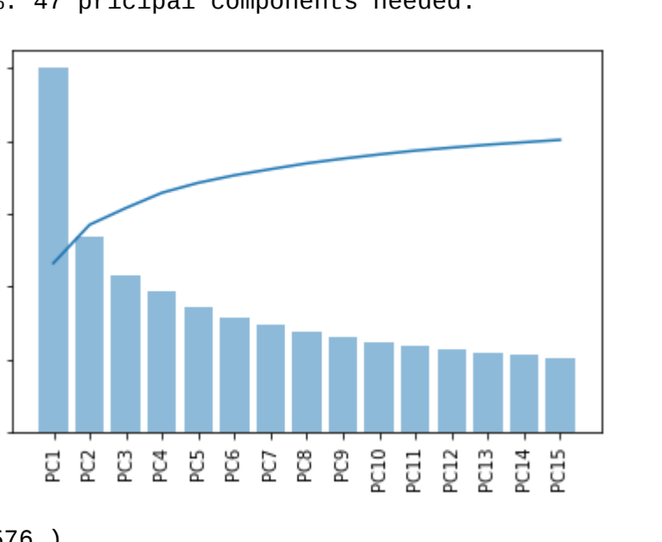
(b)

如下圖程式碼output所示

In [100]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

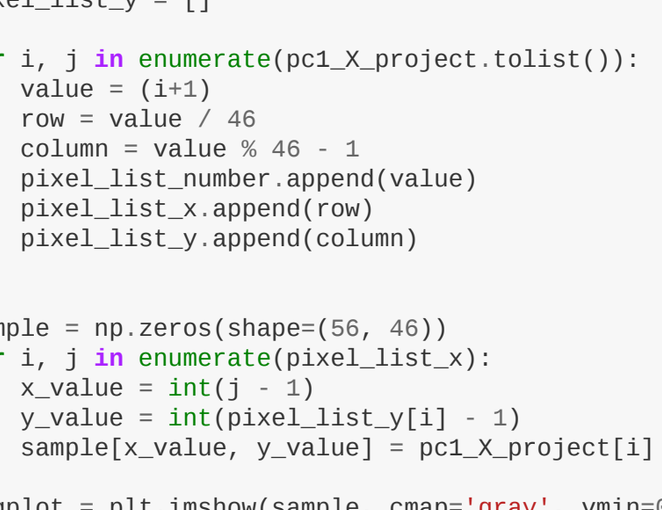
In [107]:
```
ORL_data = pd.read_csv('./data/ORL_data.csv').iloc[:,:-1]

ORL_data_array = np.transpose(ORL_data.to_numpy())
# print(ORL_data_array.shape)
eigenvalues, eigenvectors, X_project = PCA_decomposition(ORL_data_array, False)
# print(eigenvalues.shape)
```

```
50%: 2 pricipal components needed.
60%: 3 pricipal components needed.
70%: 6 pricipal components needed.
80%: 15 pricipal components needed.
90%: 47 pricipal components needed.
```



In [108]:
```
ORL_data = pd.read_csv('./data/ORL_data.csv').iloc[:,:-1]

ORL_data_array = np.transpose(ORL_data.to_numpy())
# print(ORL_data_array.shape)
eigenvalues, eigenvectors, X_project = PCA_decomposition(ORL_data_array, False)
# print(eigenvalues.shape)
# print(X_project.shape)

pc1_X_project = X_project[:, 0]
print(pc1_X_project.shape)
# print(pc1_X_project[0])
min_pc1 = np.min(pc1_X_project)
range_pc1 = np.max(pc1_X_project) - np.min(pc1_X_project)

for i, j in enumerate(pc1_X_project):
    pc1_X_project[i] = 255 * ((j - min_pc1) / range_pc1)

# print(pc1_X_project)
# print(pc1_X_project.shape)
```

```
50%: 2 pricipal components needed.
60%: 3 pricipal components needed.
70%: 6 pricipal components needed.
80%: 15 pricipal components needed.
90%: 47 pricipal components needed.
```




```
(2576,)
```

In [109]:
```
pixel_list_number = []
pixel_list_x = []
pixel_list_y = []

for i, j in enumerate(pc1_X_project.tolist()):
    value = (i+1)
    row = value / 46
    column = value % 46 - 1
    pixel_list_number.append(value)
    pixel_list_x.append(row)
    pixel_list_y.append(column)


sample = np.zeros(shape=(56, 46))
for i, j in enumerate(pixel_list_x):
    x_value = int(j - 1)
    y_value = int(pixel_list_y[i] - 1)
    sample[x_value, y_value] = pc1_X_project[i]

imgplot = plt.imshow(sample, cmap='gray', vmin=0, vmax=255)
plt.show()
```