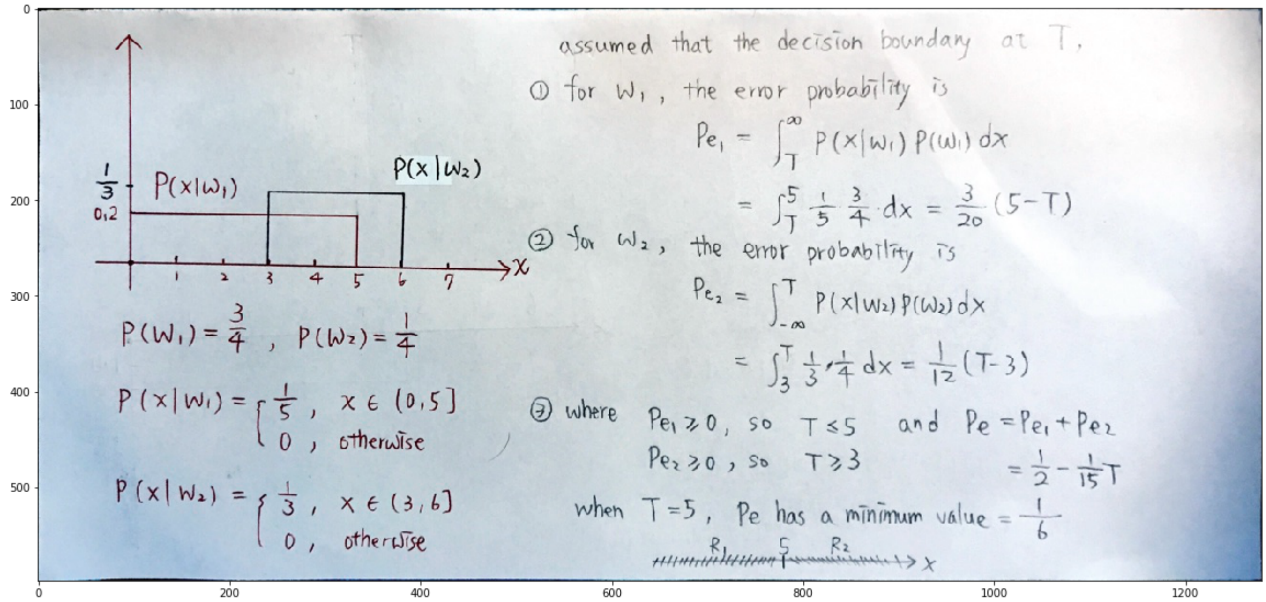
DLCV HW#1 - d05921027 張鈞閔

Problem 1

In [1]:
```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

prob1 = cv2.imread("prob1.jpg", 1)
plt.figure(figsize=(20,20))
plt.imshow(prob1)
```

Out[1]: <matplotlib.image.AxesImage at 0x11b621c50>



Problem 2

In [2]:
```
import os
import numpy as np
import pandas as pd

from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

data_dir = "data/"
```

In [3]:
```
ftrain = []
ytrain = []
for i in range(40):
    for j in range(6):
        ftrain.append(os.path.join(data_dir, str(i+1)+"_"+str(j+1)+".png"))
        ytrain.append(i)
print("number of training images:",len(ftrain))

original_shape = cv2.imread(ftrain[0],0).shape
print("original shape:",original_shape)

xtrain = []
for fn in ftrain:
    tmp = cv2.imread(fn,0)
    tmp = tmp.reshape(-1)
    xtrain.append(tmp)
xtrain = np.array(xtrain)
ytrain = np.array(ytrain)
print("training dataset for PCA, its shape =",xtrain.shape)
```

```
number of training images: 240
original shape: (56, 46)
training dataset for PCA, its shape = (240, 2576)
```

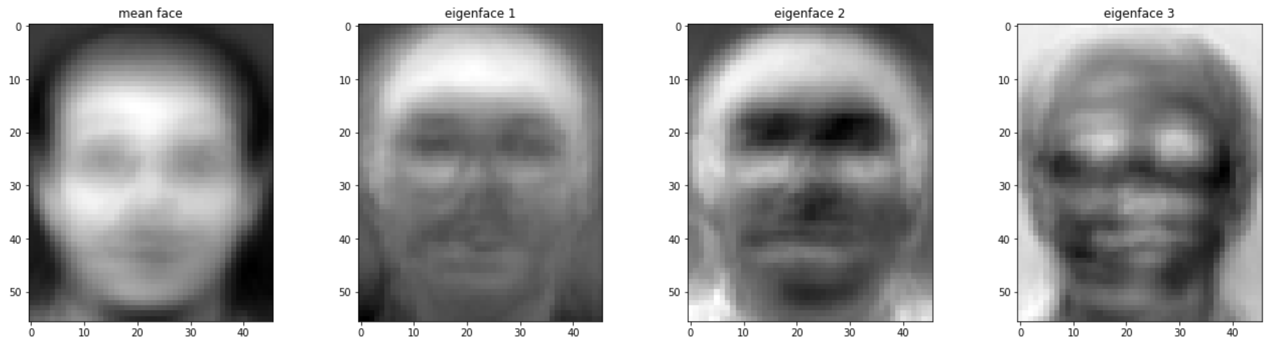
(a) mean face and the first three eigenfaces

In [4]:

```python
mean_face = np.mean(xtrain,axis=0)
mean_face = np.reshape(mean_face,newshape=original_shape)

pca = PCA(n_components=min(xtrain.shape)-1)
e = pca.fit(xtrain-mean_face.reshape(-1))
pc1 = np.reshape(e.components_[0],newshape=original_shape)
pc2 = np.reshape(e.components_[1],newshape=original_shape)
pc3 = np.reshape(e.components_[2],newshape=original_shape)

plt.figure(figsize=(20,16))
plt.subplot(141)
plt.imshow(mean_face, cmap='gray')
plt.title("mean face")
plt.subplot(142)
plt.imshow(pc1,cmap='gray')
plt.title("eigenface 1")
plt.subplot(143)
plt.imshow(pc2,cmap='gray')
plt.title("eigenface 2")
plt.subplot(144)
plt.imshow(pc3,cmap='gray')
plt.title("eigenface 3")
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,wspace=0.35)
plt.show()
plt.close()
```



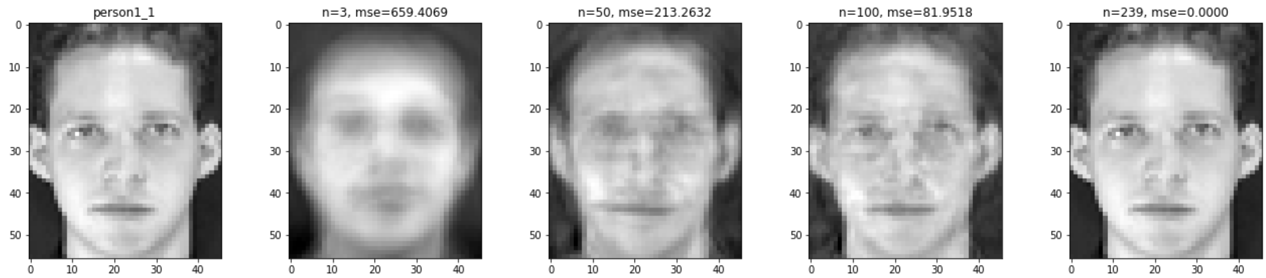
(b) reconstruction

In [5]:

```python
target = cv2.imread("data/1_1.png",0)
plt.figure(figsize=(20,16))
plt.subplot(1,5,1)
plt.title("person1_1")
plt.imshow(target, cmap="gray")

target = np.reshape(target,newshape=(1,-1))
e_target = e.transform(target - mean_face.reshape(-1))

n = [3,50,100,239]
for k in range(len(n)):
    tmp = np.dot(e_target[0,:n[k]], e.components_[:n[k]]) + mean_face.reshape(-1)
    mse = np.mean((tmp - target)**2)
    tmp = np.reshape(tmp, newshape=original_shape)
    plt.subplot(1,5,k+2)
    plt.title("n=%s, mse=%.4f" % (n[k], mse))
    plt.imshow(tmp, cmap='gray')
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.1, right=0.95, hspace=0.25,wspace=0.35)
plt.show()
plt.close()
```



(c) kNN in projected spaces

In [6]:

```
ptrain = e.transform(xtrain-mean_face.reshape(-1))
ytrain = np.array(ytrain)

params = {'n_neighbors':[1,3,5]}
kNN = KNeighborsClassifier()
clf = GridSearchCV(kNN, params,cv=3)

n = [3, 50, 159]
res = dict()
for k in n:
    clf.fit(ptrain[:,:k], ytrain)
    res['n='+str(k)] = np.array(clf.cv_results_['mean_test_score'])
res = pd.DataFrame.from_dict(res,orient='index')
res.columns = ['k=1','k=3','k=5']
print(res)
```

```
           k=1       k=3       k=5
n=3    0.708333  0.587500  0.487500
n=50   0.929167  0.875000  0.775000
n=159  0.925000  0.870833  0.745833
```

Best choice: k=1, n=50

In [7]:

```
k, n = 1, 50
```

In [8]:

```
# get testing filenames and labels
ftest = []
ytest = []
for i in range(40):
    for j in range(6,10):
        ftest.append(os.path.join(data_dir, str(i+1)+"_"+str(j+1)+".png"))
        ytest.append(i)
print("number of test images:",len(ftest))

# read testing images
xtest = []
for fn in ftest:
    tmp = cv2.imread(fn,0)
    tmp = tmp.reshape(-1)
    xtest.append(tmp)
xtest = np.array(xtest)
ytest = np.array(ytest)
print("testing dataset, its shape =",xtest.shape)

# Project images onto the principal components
ptest = e.transform(xtest-mean_face.reshape(-1))
print("projected testing dataset, its shape=", ptest.shape)
```

```
number of test images: 160
testing dataset, its shape = (160, 2576)
projected testing dataset, its shape= (160, 239)
```

In [9]:

```
# kNN model with optimized hyper-parameter (k,n)
bestkNN = KNeighborsClassifier(n_neighbors=k)
bestkNN.fit(ptrain[:,:n], ytrain)
ypred = bestkNN.predict(ptest[:,:50])
print("overall accuracy:",accuracy_score(y_pred=ypred, y_true=ytest))
```

```
overall accuracy: 0.9625
```

bonus

(1) A is a $d \times d$ sysmetric matrix, so A is diagonalizable and here we denote the d diagonal elements as $\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_d$ where λ_1 is the largest value of elements

(2) The corresponding eigenvectors of the d distinct eigenvalues are $\overrightarrow{x_1},\ \overrightarrow{x_2},\ \ldots,\ \overrightarrow{x_d}$, and can be viewed as a basis of a d-dimension space

(3) Based on (2), without loss of generality, initialize a d-dimension vector, $\overrightarrow{u_0}$, can be represented as $\overrightarrow{u_0} = c_1 \cdot \overrightarrow{x_1} + c_2 \cdot \overrightarrow{x_2} + \ldots + c_d \cdot \overrightarrow{x_d}$

(4) Based on (3) and (4),

$\overrightarrow{u_1} = A \cdot \overrightarrow{u_0} = c_1 \cdot \lambda_1 \cdot \overrightarrow{x_1} + c_2 \cdot \lambda_2 \cdot \overrightarrow{x_2} + \ldots + c_d \cdot \lambda_d \cdot \overrightarrow{x_d}$

$\overrightarrow{u_2} = A^2 \cdot \overrightarrow{u_0} = c_1 \cdot \lambda_1^2 \cdot \overrightarrow{x_1} + c_2 \cdot \lambda_2^2 \cdot \overrightarrow{x_2} + \ldots + c_d \cdot \lambda_d^2 \cdot \overrightarrow{x_d}$

$\overrightarrow{u_k} = A^k \cdot \overrightarrow{u_0} = c_1 \cdot \lambda_1^k \cdot \overrightarrow{x_1} + c_2 \cdot \lambda_2^k \cdot \overrightarrow{x_2} + \ldots + c_d \cdot \lambda_d^k \cdot \overrightarrow{x_d}$

=> $\overrightarrow{u_k} = \lambda_1^k [c_1 \cdot \overrightarrow{x_1} + c_2 \cdot \frac{\lambda_2^k}{\lambda_1^k} \cdot \overrightarrow{x_2} + \ldots + c_d \cdot \frac{\lambda_d^k}{\lambda_1^k} \cdot \overrightarrow{x_d}]$ => $\overrightarrow{u_k} \approx \lambda_1^k \cdot c_1 \cdot \overrightarrow{x_1}$ for large k

(5) Normalize $\overrightarrow{u_k}$ to a unit vector by $\frac{\overrightarrow{u_k}}{|\overrightarrow{u_k}|}$, and finally we obtain a unit vector which is in the same direction of the first eigenvector.