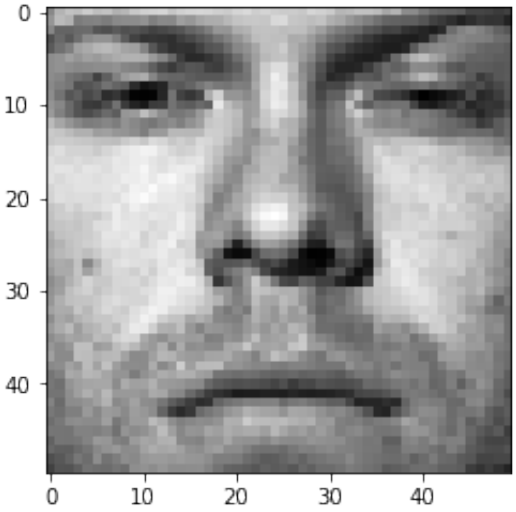
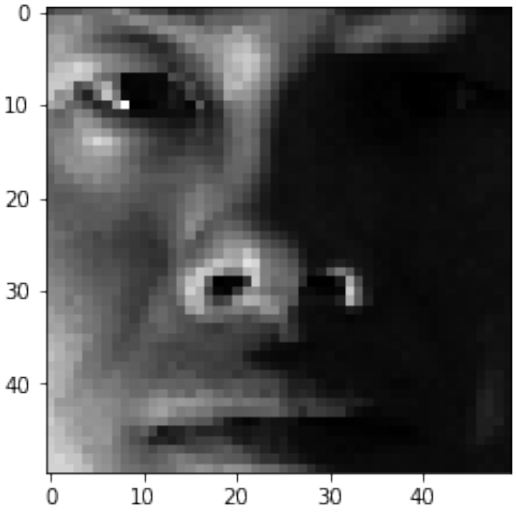
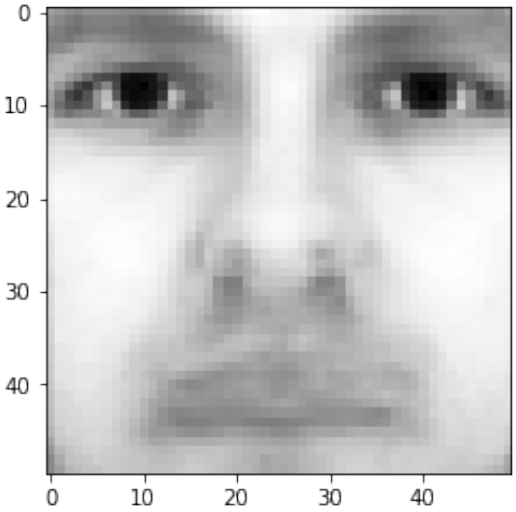
In [48]:
```
import numpy as np
from scipy import misc
from matplotlib import pylab as plt
import matplotlib.cm as cm
%matplotlib inline
train_labels, train_data = [], []
for line in open('./faces/train.txt'):
    im = misc.imread(line.strip().split()[0])
    train_data.append(im.reshape(2500,))
    train_labels.append(line.strip().split()[1])
train_data, train_labels = np.array(train_data, dtype=float), np.ar
ray(train_labels, dtype=int)

def showImg(data):
    plt.imshow(data, cmap = cm.Greys_r)
    plt.show()
showImg(train_data[10, :].reshape(50,50))
```



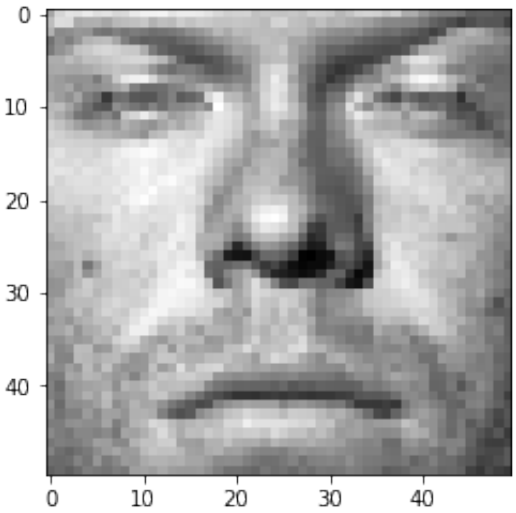
In [49]:
```
test_labels, test_data = [], []
for line in open('./faces/test.txt'):
    im = misc.imread(line.strip().split()[0])
    test_data.append(im.reshape(2500,))
    test_labels.append(line.strip().split()[1])
test_data, test_labels = np.array(test_data, dtype=float), np.array
(test_labels, dtype=int)
showImg(test_data[10, :].reshape(50,50))
```



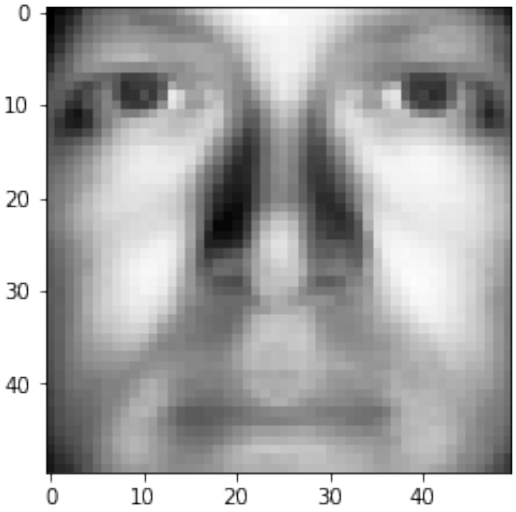
In [50]:
```
train_mean=train_data.mean(0)
test_mean=test_data.mean(0)
showImg(train_mean.reshape(50,50))
```

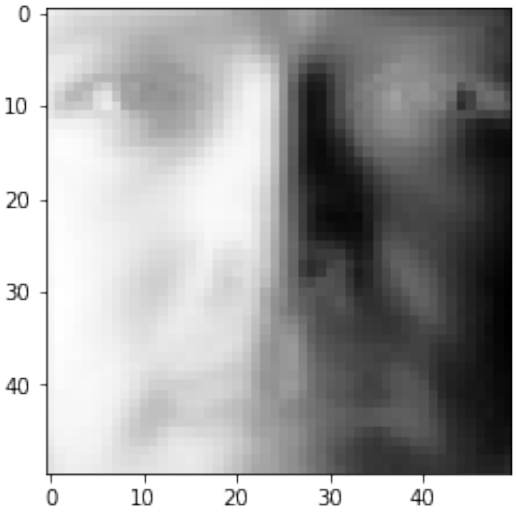


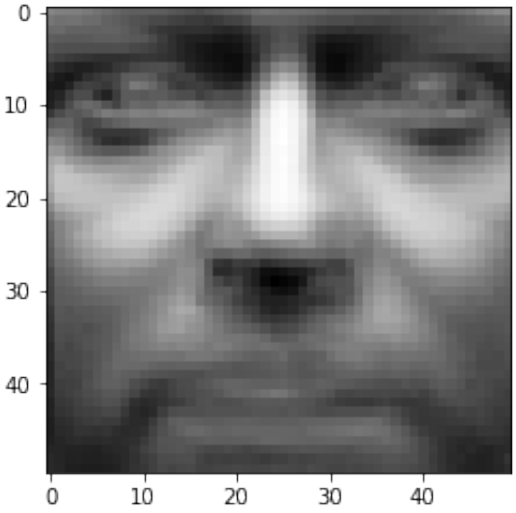
In [51]:
```
dist_train_data=np.array([xi-train_mean for xi in train_data])
dist_test_data=np.array([xi-train_mean for xi in test_data])
showImg(dist_train_data[10, :].reshape(50,50))
# new X, and X_test
train_data = dist_train_data
test_data = dist_test_data
```

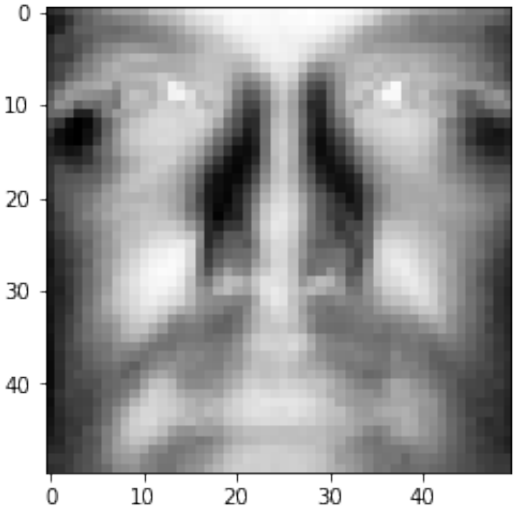


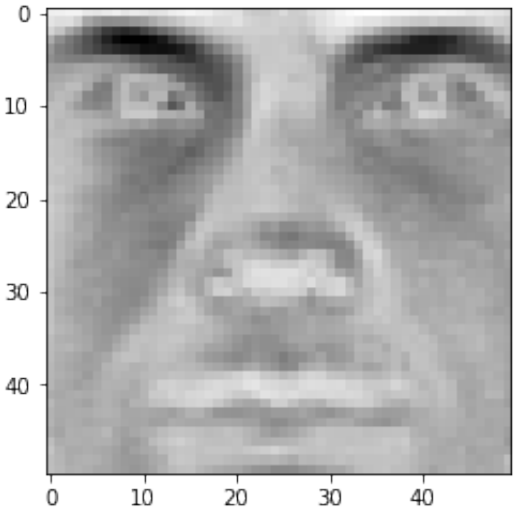
In [52]:
```
u,s,v=np.linalg.svd(train_data)
for eigenfaces in v[:10]:
    showImg(eigenfaces.reshape(50,50))
```

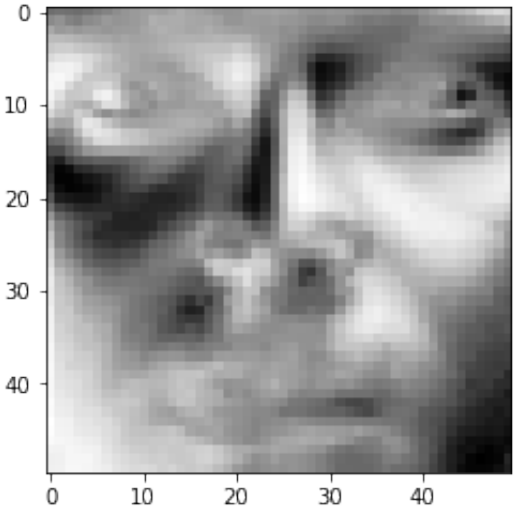


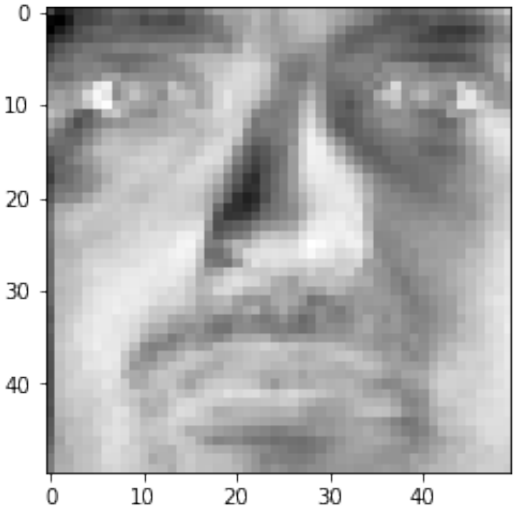

0
10
20
30
40
0
10
20
30
40

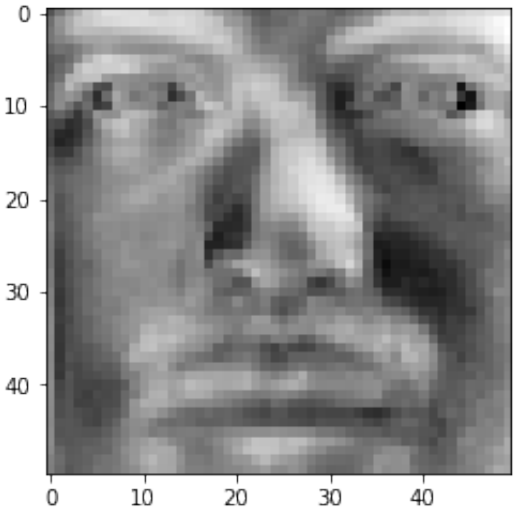

0
10
20
30
40
0
10
20
30
40

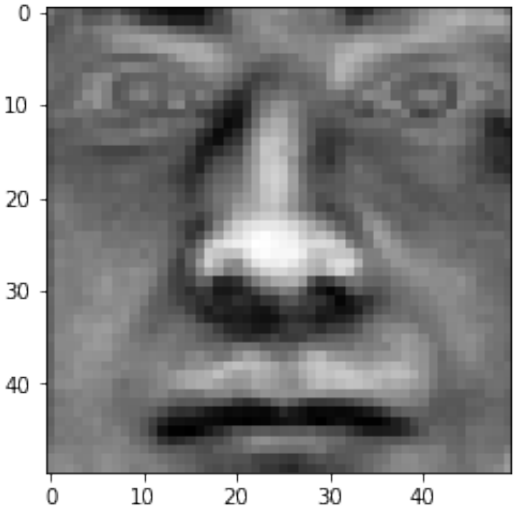

0
10
20
30
40
0
10
20
30
40

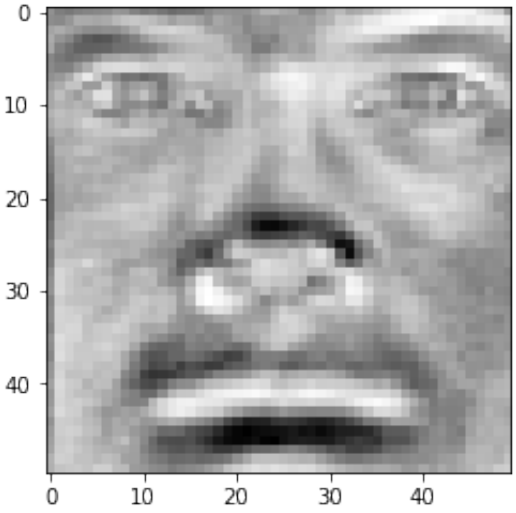

0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40

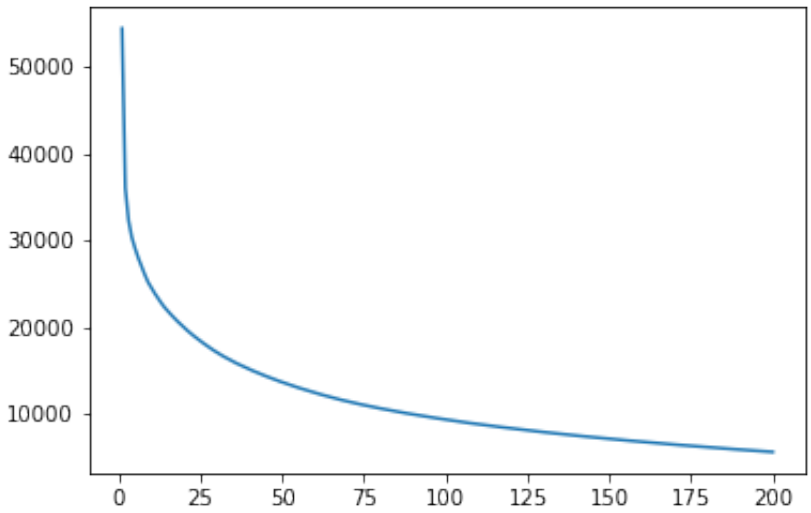

0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40

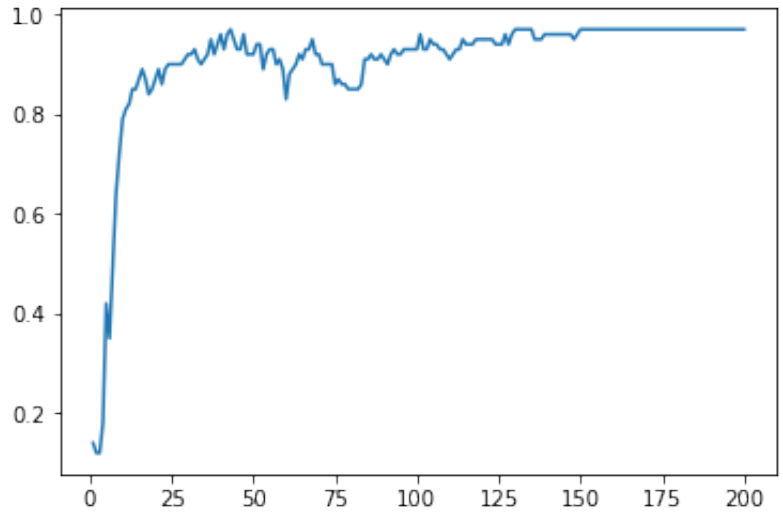
In [53]:
```
def rankRApprox(r):
    complete_s = np.zeros(train_data.shape,int)
    for i in range(len(s)):
        complete_s[i][i] = s[i]
    approx_data=np.dot(np.dot(u[:,:r],complete_s[:r,:r]),v[:r,:])
    return np.linalg.norm(train_data-approx_data)
ranks=[rankRApprox(i) for i in range(1,201)]
plt.plot(range(1,201),ranks)
plt.show()
```



In [55]:
```
def rDimensionalFeatureMatrix(r):
    F = np.dot(train_data,np.transpose(v[:r,:]))
    F_test = np.dot(test_data,np.transpose(v[:r,:]))
    return F, F_test
```

In [56]:
```
from sklearn import linear_model

def evaluateLRWithFeatureMatrix(r):
    F, F_test = rDimensionalFeatureMatrix(r)
    trainedModel = linear_model.LogisticRegression(multi_class='ovr
')
    score = trainedModel.fit(F,train_labels).score(F_test,test_labe
ls)
    return score

print ("Classification accuracy for r = 10 is: ", evaluateLRWithFea
tureMatrix(10))

scores=[evaluateLRWithFeatureMatrix(i) for i in range(1,201)]
plt.plot(range(1,201),scores)
plt.show()
```

```
Classification accuracy for r = 10 is:  0.79
```



2. b)
How many samples (dishes) are there in the training set?
Sample number: 39774
How many categories (types of cuisine)?
Category number: 20
How many unique ingredients are there?
Ingredient number: 6714

In [39]:
```
import json
import numpy as np

with open('train.json', 'r') as f:
     train_raw_data = json.load(f)
with open('test.json', 'r') as f:
     test_raw_data = json.load(f)
cuisines=set([i['cuisine'] for i in data])
ingredients=set()
for i in [i['ingredients'] for i in data]:
    ingredients |= set(i)
print("Sample number:", len(data))
print("Category number:",len(cuisines))
print("Ingredient number:",len(ingredients))
```

```
Sample number: 39774
Category number: 20
Ingredient number: 6714
```

In [36]:
```
ingredients=list(ingredients)
train_data,train_labels,test_data,test_ids=[],[],[],[]
for sample in train_raw_data:
    sampleArr=[0]*len(ingredients)
    for i in sample['ingredients']:
        sampleArr[ingredients.index(i)]=1
    train_data.append(sampleArr)
    train_labels.append(sample['cuisine'])
for sample in test_raw_data:
    sampleArr=[0]*len(ingredients)
    for i in sample['ingredients']:
        if i in ingredients: sampleArr[ingredients.index(i)]=1
    test_data.append(sampleArr)
    test_ids.append(sample['id'])
```

e)

Gaussian prior 3 fold cross validation:

In [32]:
```
from sklearn import datasets,cross_validation,linear_model
from sklearn.naive_bayes import GaussianNB,BernoulliNB

for train, test in (cross_validation.KFold(len(train_data), n_folds
=3)):
    cv_train_data=np.array(train_data)[train]
    cv_train_labels=np.array(train_labels)[train]
    cv_test_data=np.array(train_data)[test]
    cv_test_labels=np.array(train_labels)[test]
    gnb = GaussianNB().fit(cv_train_data, cv_train_labels)
    y_pred = gnb.predict(cv_test_data)
    print("Number of mislabeled points out of a total %d points : %
d" % (cv_test_data.shape[0],(cv_test_labels != y_pred).sum()))
```

```
Number of mislabeled points out of a total 13258 points : 8233
Number of mislabeled points out of a total 13258 points : 8181
Number of mislabeled points out of a total 13258 points : 8252
```

Bernoulli prior 3 fold cross validation:

In [33]:
```
for train, test in (cross_validation.KFold(len(train_data), n_folds
=3)):
    cv_train_data=np.array(train_data)[train]
    cv_train_labels=np.array(train_labels)[train]
    cv_test_data=np.array(train_data)[test]
    cv_test_labels=np.array(train_labels)[test]
    bnb = BernoulliNB().fit(cv_train_data, cv_train_labels)
    y_pred = bnb.predict(cv_test_data)
    print("Number of mislabeled points out of a total %d points : %
d" % (cv_test_data.shape[0],(cv_test_labels != y_pred).sum()))
```

```
Number of mislabeled points out of a total 13258 points : 4187
Number of mislabeled points out of a total 13258 points : 4249
Number of mislabeled points out of a total 13258 points : 4151
```

Gaussian

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution.
Another common technique for handling continuous values is to use binning to discretize the feature values, to obtain a new set of Bernoulli-distributed features; some literature in fact suggests that this is necessary to apply naive Bayes, but it is not, and the discretization may throw away discriminative information.
In this case, the cuesine type is discrete, and when apply gaussian distrbution on such a discrete variable, we are essentilaly leaving out some discriminative information. So the classifier works not so good.

Bernoulli ¶

In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs.
Bernoulli event model is especially popular for classifying short texts.
It has the benefit of explicitly modelling the absence of terms.
However, in this case, the missing ingredients of a cuisine are not considerably important with respect to training the classifier of cuisine.
Overall, bernoulli naive bayes classifier works quite well in this case

In [34]:
```
for train, test in (cross_validation.KFold(len(train_data), n_folds
=3)):
    cv_train_data=np.array(train_data)[train]
    cv_train_labels=np.array(train_labels)[train]
    cv_test_data=np.array(train_data)[test]
    cv_test_labels=np.array(train_labels)[test]
    lr = linear_model.LogisticRegression(multi_class='ovr')
    lr.fit(cv_train_data, cv_train_labels)
    y_pred = lr.predict(cv_test_data)
    print("Number of mislabeled points out of a total %d points : %
d" % (cv_test_data.shape[0],(cv_test_labels != y_pred).sum()))
```

```
Number of mislabeled points out of a total 13258 points : 2972
Number of mislabeled points out of a total 13258 points : 3021
Number of mislabeled points out of a total 13258 points : 2934
```

In [38]:
```
import csv
lr = linear_model.LogisticRegression(multi_class='ovr')
lr.fit(train_data, train_labels)
y_pred = lr.predict(test_data)
with open('submission.csv', 'w', newline='') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=',')
    spamwriter.writerow(['id','cuisine'])
    for i,label in enumerate(y_pred):
        spamwriter.writerow([test_ids[i],label])
```

1. Exercise 4.1

Proof. By Lagrange multipliers, we have the following:

$$L(a) = a^T B a - \lambda(a^T W a - 1)$$

When $\nabla L(a) = 2a^T B^T - 2\lambda a^T W^T = 0$, $L(a)$ is maximized.

$$\Rightarrow Ba = \lambda W a .$$

to maximize $a^T B a$ for $a^T W a = 1$,

$$a^T B a = a^T \lambda W a = \lambda a^T W a = \lambda$$

that is, we can maximize $a^T B a$ by maximizing λ,
then we have converted solving the generalized eigenvalues problem
to a standard eigenvalue problem.

2. Exercise 4.2

a) $\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$ with decision rule $G(x) = \arg\max_k \delta_k(x)$.

substitute the two classes for $k=1$ and $k=2$, we have the following:

$$\delta_1(x) < \delta_2(x)$$

$$x^T \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{2} \hat{\mu}_2^T \hat{\Sigma}^{-1} \hat{\mu}_2 - \frac{1}{2} \hat{\mu}_1^T \hat{\Sigma}^{-1} \hat{\mu}_1 + \log \frac{N}{N} - \log \frac{N_2}{N}$$

b) Let U_i be the n element vector with j-th element 1 if the j-th observation is class i, 0 otherwise.
Then target vector $Y = t_1 U_1 + t_2 U_2$, where t_i are target labels, and we have $\mathbb{1} = U_1 + U_2$.
Our estimates μ_1, μ_2 are $X^T U_i = N_i \hat{\mu}_i$ and that $X^T Y = t_1 N_1 \hat{\mu}_1 + t_2 N_2 \mu_2$

$$RSS = \sum_{i=1}^{N} (y_i - \beta_0 - \beta^T x)^2 = (Y - \beta_0 \mathbb{1} - X\beta)^T (Y - \beta_0 \mathbb{1} - X\beta)$$

$$\frac{dRSS}{\beta_0} = 2N\beta_0 - 2\mathbb{1}^T (Y - X\beta) = 0 \Rightarrow \beta_0 = \frac{1}{N} \mathbb{1}^T (Y - X\beta)$$

$$\frac{dRSS}{\beta} = 2X^T X\beta - 2X^T Y + 2\beta_0 X^T \mathbb{1} = 0$$

$$\Rightarrow X^T X\beta + \frac{1}{N} \mathbb{1}^T (Y - X\beta) X^T \mathbb{1} = X^T Y$$

$$\Rightarrow (X^T X - \frac{1}{N} X^T \mathbb{1} \mathbb{1}^T X)\beta = X^T Y - \frac{1}{N} X^T \mathbb{1} \mathbb{1}^T Y$$

The RHS can be written as

$$X^T Y - \frac{1}{N} X^T \mathbb{1}\mathbb{1}^T Y = t_1 N_1 \hat{\mu}_1 + t_2 N_2 \hat{\mu}_2 - \frac{1}{N}(N_1\hat{\mu}_1 + N_2\hat{\mu}_2)(t_1 N_1 + t_2 N_2) = \frac{N_1 N_2}{N}(t_1 - t_2)(\hat{\mu}_1 - \hat{\mu}_2)$$

since we have $X^T V_i = N\hat{\mu}_i$, $\mathbb{1} = V_1 + V_2$

We have: $X^T X = (N-2)\hat{\Sigma} + N_1\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\mu_2^T$.

$$\Rightarrow X^T X - \frac{1}{N} X^T \mathbb{1}\mathbb{1} X = (N-2)\hat{\Sigma} + \frac{N_1 N_2}{N}\hat{\Sigma}_B$$

putting this together, we obtain the required result,

$$\left((N-2)\hat{\Sigma} + \frac{N_1 N_2}{N}\hat{\Sigma}_B\right)\hat{\beta} = \frac{N_1 N_2}{N}(t_1 - t_2)(\hat{\mu}_1 - \hat{\mu}_2)$$

$t_1 = -N/N_1$, $t_2 = N/N_2$

$$\Rightarrow \left((N-2)\hat{\Sigma} + \frac{N_1 N_2}{N}\Sigma_B\right)\hat{\beta} = \frac{N_1 N_2}{N}\left(-\frac{N}{N_1} - \frac{N}{N_2}\right)(\hat{\mu}_1 - \hat{\mu}_2) = \frac{N_1 N_2}{N}\left(\frac{N(N_1+N_2)}{N_1 N_2}\right)(\hat{\mu}_2 - \hat{\mu}_1)$$

$$= N(\hat{\mu}_2 - \hat{\mu}_1)$$

c) As we have $\hat{\Sigma}_B = \frac{N_1 N_2}{N^2}(\hat{\mu}_2 - \hat{\mu}_1)(\mu_2 - \mu_1)^T$.

so $\hat{\Sigma}_B \cdot \hat{\beta} = \frac{N_1 N_2}{N^2}(\hat{\mu}_2 - \hat{\mu}_1)(\mu_2 - \mu_1)^T \cdot \hat{\beta}$, let $\lambda = \frac{N_1 N_2}{N^2}(\mu_2 - \mu_1)^T\hat{\beta}$

$$= \lambda \cdot (\hat{\mu}_2 - \hat{\mu}_1)$$

Therefore, $\hat{\beta} \propto \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$

d) t_1, t_2 were arbitary and distinct, the result follows.

e). $\hat{\beta}_0 = \frac{1}{N}\mathbb{1}^T(Y - X\hat{\beta})$

$$= \frac{1}{N}(t_1 N_1 + t_2 N_2) - \frac{1}{N}\mathbb{1}^T X\hat{\beta}_1$$

$$= -\frac{1}{N}(N_1\hat{\mu}_1^T + N_2\hat{\mu}_2^T)\hat{\beta}$$

then $\hat{f}(x) = \frac{1}{N}(Nx^T - N_1\mu_1^T - N_2\hat{\mu}_2^T)\hat{\beta}$

$$= \frac{1}{N}(Nx^T - N_1\hat{\mu}_1^T - N_2\hat{\mu}_2^T)\lambda\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$$

for $\lambda \in R$. the classification rule is $\hat{f}(x) > 0$, that is

$$Nx^T\lambda\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > (N_1\hat{\mu}_1^T + N_2\hat{\mu}_2^T)\lambda\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$$

$$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{N}(N_1\mu_1^T + N_2\mu_2^T)\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1).$$

it is different from LDA decision rule unless $N_1 = N_2$

In [1]:
```
import numpy as np
M = [[1,0,3],[3,7,2],[2,-2,8],[0,-1,1],[5,8,7]]
M = np.array(M, dtype = float)

# a)
print ("a)")
MTM = np.dot(np.transpose(M),M)
MMT = np.dot(M,np.transpose(M))

print ("MTM:\n",MTM)
print ("MMT:\n",MMT)
print ("\n")

eigvalueA, eigvectorA = np.linalg.eig(MTM)
eigvalueB, eigvectorB = np.linalg.eig(MMT)

# b)
print ("b)")
print ("eigvalueA:\n",eigvalueA)
print ("eigvalueB:\n",eigvalueB)
print ("\n")

# c)
print ("c)")
print ("eigvectorA:\n",eigvectorA)
print ("eigvectorB:\n",eigvectorB)
print ("\n")

# d)
print ("d)")
u,s,v = np.linalg.svd(M)
s =np.diag(s)
print ("U:\n",u)
print ("S:\n",s)
print ("V:\n",v)
print ("\n")

# e)
print ("e)")
def rankRApprox(matrix,r):
    u,s,v = np.linalg.svd(matrix)
    s =np.diag(s)
    approx_data=np.dot(np.dot(u[:,:r],s[:r,:r]),v[:r,:])
    return (np.linalg.norm(matrix - approx_data))

print ("Rank 1 approximation:\n",rankRApprox(M,1))
# ranks=[rankRApprox(i) for i in range(1,201)]
# plt.plot(range(1,201),ranks)
# plt.show()
```

```
a)
```

```
MTM:
 [[  39.   57.   60.]
 [  57.  118.   53.]
 [  60.   53.  127.]]
MMT:
 [[  10.    9.   26.    3.   26.]
 [   9.   62.    8.   -5.   85.]
 [  26.    8.   72.   10.   50.]
 [   3.   -5.   10.    2.   -1.]
 [  26.   85.   50.   -1.  138.]]


b)
eigvalueA:
 [  2.14670489e+02  -1.50990331e-14   6.93295108e+01]
eigvalueB:
 [  2.14670489e+02  -5.99520433e-15   6.93295108e+01   1.23720729e
-14
  -7.24621456e-16]


c)
eigvectorA:
 [[ 0.42615127  0.90453403 -0.01460404]
 [ 0.61500884 -0.30151134 -0.72859799]
 [ 0.66344497 -0.30151134  0.68478587]]
eigvectorB:
 [[-0.16492942 -0.95539856  0.24497323 -0.06464508 -0.10671808]
 [-0.47164732 -0.03481209 -0.45330644  0.75010839 -0.04684757]
 [-0.33647055  0.27076072  0.82943965  0.3284142  -0.17364364]
 [-0.00330585  0.04409532  0.16974659  0.04591676  0.97063121]
 [-0.79820031  0.10366268 -0.13310656 -0.5685017   0.11890961]]


d)
U:
 [[-0.16492942 -0.24497323  0.94690858  0.02817617 -0.12392196]
 [-0.47164732  0.45330644 -0.06559072  0.10010213 -0.74681959]
 [-0.33647055 -0.82943965 -0.30585475 -0.15523825 -0.28490595]
 [-0.00330585 -0.16974659 -0.0689086   0.98238169  0.03678296]
 [-0.79820031  0.13310656 -0.02768538 -0.00360121  0.58683853]]
S:
 [[  1.46516378e+01   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   8.32643446e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   1.68573193e-15]]
V:
 [[-0.42615127 -0.61500884 -0.66344497]
 [ 0.01460404  0.72859799 -0.68478587]
 [-0.90453403  0.30151134  0.30151134]]


e)
Rank 1 approximation:
```