①

Let us suppose that the variance of M-dimensional projection $y_M = w_M^T x$ is maximized by $w = [u_1, u_2 .. u_M]$.

where, $u_1, u_2 .. u_M$ are orthonormal eigen vectors of sample covariance matrix (S) corresponding to M largest eigen values.

Therefore, we are considering that the results holds for projection spaces of dimensionality (M).

The M+1.th dimensional (subspace) space will be defined by

$u_1 u_2, u_3 .. u_M$ together with additional vector $U_{M+1}$.

We need to determine the value of the $U_{M+1}$, such that its linearly independent with all $u_1, u_2 .. u_M$. and its perpendicular to all the vectors of $u_1, u_2 .. u_M$ and it should be unit vector.

It means $u_{M+1}$ should be ortho-normal with $u_1, u_2 .. u_M$.

we can achieve the direction $U_{M+1}$ by Maximizing the $U_{M+1}^T S U_{M+1}$

so, we form an optimization problem with lagrange multiplier for adding constraints of orthonormality.

$$\underline{\text{Direction}(u_M)} = \max_{U_M} \; U_{M+1}^T S U_{M+1} + \lambda_{M+1}(1 - U_{M+1}^T U_{M+1}) + \sum_{i=1}^{M} \lambda_i (U_{M+1}^T U_i).$$

We are doing for $u_{M+1}$.

Therefore,

$$\frac{\partial}{\partial u_{M+1}}\left[ u_{M+1}^T S u_M + \lambda_{M+1}\left(1 - u_{M+1}^T u_{M+1}\right) + \sum_{i=1}^{M} \eta_i\, u_{M+1}^T u_i \right] = 0.$$

$$2Su_{M+1} + \lambda_{M+1}(-2u_{M+1}) + \sum_{i=1}^{M} \eta_i u_i = 0 \rightarrow (1)$$

In the above equation, $\eta_i$-terms, $\lambda_{M+1}$ need to evaluate.

If you multiply above equation (1) by $u_i$ where $i=1,\dots M$

$$u_1^T\left[2Su_{M+1} + \lambda_{M+1}(-2u_{M+1})\right] + \sum_{i=1}^{M} \eta_i^2 \left(u_1^T u_i\right) = 0 \Rightarrow \eta_1 = 0$$

$$u_2^T\left[2Su_{M+1} + \lambda_{M+1}(-2u_{M+1})\right] + \sum_{i=1}^{M} \eta_i \left(u_2^T u_i\right) = 0 \Rightarrow \eta_2 = 0$$

$$\vdots$$

$$u_M^T\left[2Su_{M+1} + \lambda_{M+1}(-2u_{M+1})\right] + \sum_{i=1}^{M} \eta_i \left(u_M^T u_i\right) = 0 \Rightarrow \eta_M = 0$$

Subsitute $\eta_1 = \eta_2 = \dots = \eta_M = 0$ in the equation (1),

$$2Su_{M+1} + \lambda_{M+1}(-2u_{M+1}) = 0.$$

$$Su_{M+1} = \lambda_{M+1} u_{M+1}$$

$$\boxed{u_{M+1}^T S u_{M+1} = \lambda_{M+1}}$$

So,

We need to select the eigenvalue, which is largest among those not previously selected.

Therefore,

It is proved that variance of $M+1$ - dimensional projection $y_{M+1} = w_{M+1}^T x$ is maximized by choosing $w_{M+1} = [w_M \; v_{M+1}]$.

(2)

$$\frac{d}{dA}(tr(AB)) = 2B - diag(B).$$

given, that A, B are symmetric matrices

trace(AB) :-

$$[AB]_{ij} = \sum_{k=1}^{M} [A]_{ik} [B]_{kj}$$

$$trace(AB) = \sum_{i=1}^{M} \left[ \sum_{k=1}^{M} [A]_{ik} [B]_{ki} \right]$$

since, we need to sum the digonal elements of AB.

For $\frac{d}{dA}(trace(AB))$ we need to calculate respect to element.

$\frac{d}{dA_{ij}}(trace(AB))$ : need to evaluate.

For this we need to consider two cases.

since, A, B are symmetric we need consider digonal case, non-diagonal case seperately.

For diagonal case:

$$\frac{d}{dA_{ii}}(trace(AB)) = \frac{d}{dA_{ii}} \left( \sum_{i=1}^{M} \sum_{k=1}^{M} A_{ik} B_{ki} \right)$$

$$= B_{ii}$$

$$\frac{d}{dA_{22}}(trace(AB)) = B_{22}$$

$$\vdots$$

$$\frac{d}{dA_{ii}}(trace(AB)) = B_{ii}$$

For non diagonal case:

$$\frac{d}{dA_{ij}}\left(\text{trace}(AB)\right) = \frac{d}{dA_{ij}}\left(\sum_{i=1}^{M}\sum_{K=1}^{M} A_{iK}B_{Ki}\right)$$

~~When $K=j$~~

$$\frac{d}{d(A_{12})}\left(\text{trace}(AB)\right) = A_{11}B_{11} + A_{12}B_{21} + \cdots A_{1M}B_{M1}.$$
$$+ A_{21}B_{12} + A_{22}B_{22} + \cdots A_{2M}B_{M2}.$$
$$\vdots \qquad\qquad \vdots$$
$$+ A_{M1}B_{1M} + A_{M2}B_{2M} + \cdots A_{MM}B_{MM}$$

$$= B_{21} + B_{12} = 2B_{21} = 2B_{12}.$$

$$\frac{d}{dA_{ij}}\left(\text{trace}(AB)\right) = 2B_{ji} = 2B_{ij}.$$

$$\frac{d}{dA}\left(\text{trace}(AB)\right) = \begin{bmatrix} B_{11} & 2B_{12} & 2B_{13} & \cdots & 2B_{1M} \\ 2B_{21} & B_{22} & 2B_{23} & \cdots & 2B_{2M} \\ 2B_{31} & 2B_{32} & B_{33} & \cdots & 2B_{3M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2B_{M1} & 2B_{M2} & 2B_{M3} & \cdots & B_{MM} \end{bmatrix}$$

$$= 2\begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1M} \\ B_{21} & B_{22} & \cdots & B_{2M} \\ \vdots & \vdots & & \vdots \\ B_{M1} & B_{M2} & \cdots & B_{MM} \end{bmatrix} - \begin{bmatrix} B_{11} & & & \\ & B_{22} & & \\ & & \ddots & \\ & & & B_{MM} \end{bmatrix}$$

$$= 2B - \text{diag}(B).$$

(2)

$$\frac{d}{dA} \log(|A|) = 2\bar{A}^{-1} - \mathrm{Diag}(\bar{A}^{-1}).$$

$$\frac{d}{dA} \log|A| = \frac{1}{|A|} \cdot \frac{d}{dA}(|A|).$$

since, $A \cdot \mathrm{Adj}A = (\det A) I$.

$$|A| = \det A = \sum_{K=1}^{M} A_{iK} [\mathrm{Adj}A]_{Ki}.$$

$$= \sum_{K=1}^{M} A_{iK} \cdot C_{iK}$$

$$\mathrm{Adj}A = [\text{Cofactor matrix}(A)]^T$$

$$\mathrm{Adj}A = C^T.$$

$$\det(A) = \sum_{K=1}^{M} A_{iK} C_{iK} = A_{11}C_{11} + A_{12}C_{21} + A_{13}C_{31} + \cdots A_{M1}C_{M1}$$

$\frac{d(\det(A))}{dA_{ij}}$ respect to digonal element :-

In the diagonal $i=j$

$$\frac{d}{dA_{ii}} (\det(A))$$ ~~need~~ calculate

$$\frac{d}{dA_{11}} (\det A) = \frac{d}{dA_{11}} \left( \sum_{K=1}^{M} A_{iK} C_{Ki} \right) = C_{11}$$

generalize for all $C_{ii}$.

$$\frac{d}{dA_{22}} (\det A) = C_{22}.$$

$$\vdots$$

$$\frac{d}{dA_{MM}} (\det A) = C_{MM}.$$

$\frac{d}{dA}(\det A)$ respect to Non diagonal element :-

we do the differentiation respect to one element $A_{12}$, later we generalize it.

$$\det A = C_{11}A_{11} + C_{12}A_{12} + C_{13}A_{13} + \dots C_{1M}A_{1M}$$

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \dots & A_{1M} \\ A_{21} & A_{22} & A_{23} \dots & A_{2M} \\ \vdots & & & \\ A_{M1} & A_{M2} & A_{M3} \dots & A_{MM} \end{bmatrix}$$

$$\frac{d}{dA_{12}}(\det A) = \frac{d}{dA_{12}}\left(C_{11}A_{11} + C_{12}A_{12} + C_{13}A_{13} + C_{14}A_{14} + \dots + C_{1M}A_{1M}\right)$$

$$= \frac{d}{dA_{12}}(C_{11}A_{11}) + \frac{d}{dA_{12}}(C_{12}A_{12}) + \frac{d}{dA_{12}}(C_{13}A_{13} + C_{14}A_{14} + \dots + C_{1M}A_{1M})$$

$$\frac{d}{dA_{12}}(C_{11}A_{11}) = \frac{d}{dA_{12}}(C_{11})A_{11} + C_{11}\frac{d}{dA_{12}}(A_{11})$$

$$C_{11} = \begin{vmatrix} A_{22} & A_{23} \dots & A_{2M} \\ A_{32} & A_{33} \dots & A_{3M} \\ & \vdots & \\ A_{M1} & A_{M2} \dots & A_{MM} \end{vmatrix} =$$

$C_{11}$ is not function of $A_{12}$

$$\frac{d}{dA_{12}}(C_{11}) = 0 \quad \text{and} \quad \frac{d}{dA_{12}}(A_{11}) = 0.$$

Therefore $\frac{d}{dA_{12}}(C_{11}A_{11}) = 0.$

$$\frac{d}{dA_{12}}(C_{12}A_{12}) = \frac{d}{dA_{12}}(C_{12})A_{12} + \frac{d}{dA_{12}}(A_{12})C_{12}$$

$$= C_{12} + A_{12}\frac{d}{dA_{12}}(C_{12})$$

$$C_{12} = \begin{vmatrix} A_{21} & A_{23} & A_{24} \cdots A_{2M} \\ A_{31} & A_{33} & A_{34} \cdots A_{3M} \\ \vdots & & \\ A_{M1} & A_{M3} & A_{M4} \cdots A_{MM} \end{vmatrix}$$

$$\frac{d(C_{12})}{dA_{12}} \frac{d(C_{12})}{dA_{21}} = \begin{vmatrix} A_{33} & A_{34} \cdots A_{3M} \\ A_{43} & A_{44} \cdots A_{4M} \\ \vdots & \\ A_{M3} & A_{M4} \cdots A_{MM} \end{vmatrix}$$

$$\frac{d}{dA_{12}}(C_{13}A_{13}) = A_{13}\frac{d}{dA_{12}}(C_{13})$$

$$C_{13} = \begin{vmatrix} A_{21} & A_{22} & A_{24} \cdots A_{2M} \\ A_{31} & A_{32} & A_{34} \cdots A_{3M} \\ \vdots & & \\ A_{M1} & A_{M2} & A_{M4} \cdots A_{MM} \end{vmatrix}$$

$$\frac{d}{dA_{12}}(C_{13}) = \frac{d}{dA_{21}}(C_{13}) = \begin{vmatrix} A_{32} & A_{34} \cdots A_{3M} \\ A_{42} & A_{44} \cdots A_{4M} \\ \vdots & \vdots \\ A_{M2} & A_{M4} \cdots A_{MM} \end{vmatrix}$$

$$\frac{d}{dA_{12}}(C_{14}A_{14}) = A_{14}\frac{d}{dA_{12}}(C_{14})$$

$$C_{14} = \begin{vmatrix} A_{21} & A_{22} & A_{23} & A_{25}\cdots & A_{2M} \\ A_{31} & A_{32} & A_{33} & A_{35}\cdots & A_{3M} \\ \vdots & & & & \\ A_{M1} & A_{M2} & A_{M3} & A_{M5}\cdots & A_{MM} \end{vmatrix}$$

$$\frac{d}{dA_{12}}(C_{14}) = \frac{d}{dA_{21}}(C_{14}) = \begin{vmatrix} A_{32} & A_{33} & A_{35}\cdots & A_{3M} \\ A_{42} & A_{43} & A_{45}\cdots & A_{4M} \\ \vdots & & & \\ A_{M2} & A_{M3} & A_{M5}\cdots & A_{MM} \end{vmatrix}$$

$$\frac{d}{dA_{12}}(C_{1M}A_{1M}) = A_{1M}\frac{d}{dA_{12}}(C_{1M})$$

$$C_{1M} = \begin{vmatrix} A_{21} & A_{22}\cdots & A_{2M-1} \\ A_{31} & A_{32}\cdots & A_{3,M-1} \\ \vdots & & \\ A_{M1} & A_{M2}\cdots & A_{M,M-1} \end{vmatrix}$$

$$\frac{d}{dA_{12}}(C_{1M}) = \frac{d}{dA_{21}}(C_{1M}) = \begin{vmatrix} A_{32} & A_{33}\cdots & A_{3,M-1} \\ A_{42} & A_{43}\cdots & A_{4,M-1} \\ \vdots & \vdots & \vdots \\ A_{M2} & A_{M3}\cdots & A_{M,M-1} \end{vmatrix}$$

$$\frac{d}{dA_{12}} \det(A) = \frac{d}{dA}(C_{11}A_{11}) + \frac{d}{dA_{12}}(C_{12}A_{12})$$

$$+ \frac{d}{dA_{12}}(C_{13}A_{13} + C_{14}A_{14} \cdots C_{1M}A_{1M})$$

$$= C_{12} + A_{12}\frac{d}{dA}(C_{12}) + A_{13}\frac{d}{dA_{13}}(C_{13}) + A_{14}\frac{d}{dA_{14}}(C_{14}) + \cdots$$

$$\cdots + A_{1M}\frac{d}{dA_{1M}}(C_{1M})$$

$$= C_{12} + A_{12}\begin{vmatrix} A_{33} & A_{34} \cdots & A_{3M} \\ A_{43} & A_{44} \cdots & A_{4M} \\ \vdots & & \\ A_{M3} & A_{M4} \cdots & A_{MM} \end{vmatrix} + A_{13}\begin{vmatrix} A_{32} & A_{34} \cdots & A_{3M} \\ A_{42} & A_{44} \cdots & A_{4M} \\ \vdots & & \\ A_{M2} & A_{M4} \cdots & A_{MM} \end{vmatrix}$$

$$+ A_{14}\begin{vmatrix} A_{32} & A_{33}\ A_{35} \cdots A_{3M} \\ A_{42} & A_{43}\ A_{45} \cdots A_{4M} \\ \vdots & \\ A_{M2} & A_{M3}\ A_{M5} \cdots A_{MM} \end{vmatrix} + \cdots$$

$$+ \cdots A_{1M}\begin{vmatrix} A_{32} & A_{33} \cdots & A_{3,M-1} \\ A_{42} & A_{43} \cdots & A_{4,M-1} \\ \vdots & & \\ A_{M,2} & A_{M3} \cdots & A_{MM-1} \end{vmatrix}$$

$$= C_{12} + \begin{vmatrix} A_{12} & A_{13} & A_{14} \cdots & A_{1M} \\ A_{32} & A_{33} & A_{34} \cdots & A_{3M} \\ A_{42} & A_{43} & A_{44} \cdots & A_{4M} \\ \vdots & & & \\ A_{M2} & A_{M3} & A_{M4} \cdots & A_{MM} \end{vmatrix} = C_{12} + C_{21} = 2C_{12}$$

$$\frac{d}{dA_{12}}(\det A) = 2C_{12}$$

$$\frac{d}{dA_{ij}}(\det A) = \begin{cases} 2C_{ij} & \text{for } i \neq j. \quad \text{for a non-diagonal element} \\ C_{ij} & \text{for } i = j \quad \text{for a diagonal element} \end{cases}$$

$$\frac{d}{dA}(\det A) = \begin{bmatrix} C_{11} & 2C_{12} & 2C_{13} \cdots & 2C_{1M} \\ 2C_{21} & C_{22} & 2C_{13} \cdots & 2C_{2M} \\ \vdots & & & \\ 2C_{M1} & 2C_{M2} & 2C_{M3} \cdots & C_{MM} \end{bmatrix}$$

$$= 2\begin{bmatrix} C_{11} & C_{12} & C_{13} \cdots & C_{1M} \\ C_{21} & C_{22} & C_{33} \cdots & C_{3M} \\ \vdots & & & \\ C_{M1} & C_{M2} & C_{M3} \cdots & C_{MM} \end{bmatrix} - \text{Diag}(C_{11}, C_{22} \cdots C_{MM})$$

$$= 2C - \text{Diag}(C)$$

Since A is symmetric.

Adj A and cofactor matrix (C) also symmetric

$$\text{Adj } A = C$$

$$\frac{d}{dA}(\det A) = 2\,\text{Adj}A - \text{Diag}((\text{Adj}A)_{11}, (\text{Adj}A)_{22}, \ldots, (\text{Adj}A)_{MM})$$

$$\frac{d}{dA}(\log(\det A)) = \frac{1}{\det A} \cdot \frac{d}{dA}(\det A)$$

$$= \frac{1}{\det A}\left(2\,\mathrm{Adj}A - \mathrm{Diag}(\mathrm{Adj}A)\right)$$

$$= 2\,\frac{\mathrm{Adj}A}{\det A} - \frac{\mathrm{Diag}(\mathrm{Adj}A)}{\det A}$$

$$= 2\,\frac{\mathrm{Adj}A}{\det A} - \mathrm{Diag}\left(\frac{\mathrm{Adj}A}{\det A}\right)$$

$$= 2\bar{A}^{T} - \mathrm{Diag}(\bar{A}^{T})$$

$$\frac{d}{dA}(\log(\det(A))) = 2\bar{A}^{T} - \mathrm{Diag}(\bar{A}^{T})$$

(3)

(a).

$$S_T^y = \frac{1}{N}\sum_{n=1}^{N} y_n . y_n^T$$

$$y_n = \Lambda^{1/2} W^T (x-\mu)$$

$x$ - $D \times 1$ image.

$\mu$ - $D \times 1$ matrix

$W \rightarrow$ PCA projection matrix. of $D \times d$.

Its consisting $d$ - eigenvectors. of $D$-dimensional of covariance matrix of sample data.

$$W \rightarrow \underbrace{[u_1\ u_2 \ldots u_d]}_{D \times d}$$

$\Lambda$ - $d \times d$ matrix containing $d$-largest eigenvalues of sample covariance matrix.

$$S_T^y = \frac{1}{N}\sum_{n=1}^{N} y_n y_n^T = \frac{1}{N}\sum \Lambda^{1/2} W^T (x-\mu)(x-\mu)^T W (\Lambda^{1/2})^T$$

$$= \Lambda^{1/2} W^T \left(\frac{1}{N}(x-\mu)(x-\mu)^T\right) W \Lambda^{1/2}$$

$$= \Lambda^{1/2} W^T S_x W \Lambda^{-1/2} \longrightarrow (1)$$

Since, • $W$ is with eigenvectors of sample covariance matrix,

$$W = [u_1\ u_2 \ldots u_d]$$

$$S_x W = [S_x u_1\ S_x u_2 \ldots S_x u_d]$$

$$= [\lambda_1 u_1\ \lambda_2 u_2 \ldots \lambda_d u_d]$$

$$W^T S_x W = \begin{bmatrix} u_1^T\lambda_1 u_1 & 0 & \cdots & 0 \\ 0 & u_2^T\lambda u_2 & \cdots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & u_d^T\lambda u_d \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{bmatrix} \rightarrow (2).$$

$$W^T S_x W = \Lambda.$$

Subsitute the equation ② in equation ①.

$$S_T^y = \Lambda^{-1/2} W^T S_x W \Lambda^{-1/2}$$

$$= \Lambda^{-1/2} \Lambda \Lambda^{-1/2}.$$

$$= I$$

$$\boxed{S_T^y = \frac{1}{N} \sum_{n=1}^{N} y_n y_n^T = I}$$

(b).

(b) We are assuming $S_W^y$ invertible..

$S_T^y$ : Total covariance matrix on whitened data $y$.

we already seen this,

$$S_T = I .$$

$S_B^y$ = Between class scatter.

$S_W^y$ = with in class scatter.

$$S_T^y = S_B^y + S_W^y \Rightarrow S_B^y = I - S_W^y.$$

From the fishers criterion, $\max_W \frac{W^T S_B^y W}{W^T S_W^y W}$ need to solve

$$\max_W \frac{W^T (I - S_W^y) W}{W^T S_W^y W}$$

$$\max_W \frac{W^T W}{W^T S_W^y W} - 1$$

$$J = \max_{w} \frac{w^T w}{w^T S_w^y w} \quad -1$$

$$= \max_{w} \left( \frac{w^T w}{w^T S_w^y w} \right)$$

$$J = \max_{w} w^T w \quad \text{s.t } w^T S_w^y w = 1$$

$$= \max_{w} w^T w \quad \text{s.t, } w^T S_w^y w = 1$$

$$J = \max_{w} w^T w + \lambda(1 - w^T S_w^y w)$$

$$\frac{dJ}{dw} = 0.$$

$$2w + \lambda(-2 S_w^y w) = 0$$

$$w = \lambda S_w^y w \implies w^T w = \lambda w^T S_w^y w \implies \lambda = \frac{w^T w}{(w^T S_w^y w)} \rightarrow (1)$$

$$\bar{S_w^1} w = \lambda w \rightarrow (2).$$

↓ it's eigen equation of $\bar{S_w^1}$.

From (1), (2),

Max eigenvalue of $\bar{S_w^1}$ is the solution for the maximum value of $\left( \frac{w^T w}{w^T S_w^y w} \right)$

Sq eigenvector of max eigen value is the solution (w).

Therefore,

LDA Projectionvector (W) is given by ~~x~~ eigenvector of max eigenvalue of $\bar{S_W}^{-1}$. thats why, it is eigenvector ~~of~~ min-eigenvalue ~~of~~ $S_W$.

The above proof was defined fundmentally.

We already know that,

for multi class fisher criterion,

The maximum eigen vectors of $\bar{S_W}^{-1} S_B$ is the transformation for (eigenvalues respective) LDA.

$$\bar{S_W}^{-1} S_B = \bar{S_W}^{-1} (S_T - S_W)$$

$$= \bar{S_W}^{-1} (I - S_W)$$

$$= \bar{S_W}^{-1} - I.$$

We can see that the maximum eigen value respective eigen vector of $\bar{S_W}^{-1} - I$ is the solution for LDA transformation.

$\bar{S_W}^{-1} - I$ and $\bar{S_W}^{-1}$ have same eigenvector.

and But eigen value ~~of maximum~~ is remains max only.

for $S_W$ eigen vectors are same as $\bar{S_W}^{-1}$,

But max eigenvalue ~~of~~ $S_W$ is min eigenvalue ~~of~~ $\bar{S_W}^{-1}$.

Therefore

First LDA Projection vector is eigenvector of $\bar{S_W}^{-1}$ with min magnitude of eigen value.

4. **Fischer faces** - Data is posted here
*http://leap.ee.iisc.ac.in/sriram/teaching/MLSP21/assignments/data/Data.tar.gz*
Copy and paste the above link in the browser with the underscore for MLSP_19

   15 subject faces with happy/sad emotion are provided in the data. Each image is of 100x100 matrix. Perform PCA on to reduce the dimension from 10000 to $K$ (using PCA for high dimensional data) and then perform LDA to one dimension. Plot the one dimension features for each image. Select the optimum threshold to classify the emotion and report the classification accuracy on the test data. What is the best choice of $K$ which gives the maximum separability ? (**Points** 25)

In [91]:
```
!pwd
```

```
/home/venkatesh/Documents/MLSP/Assignment_01
```

In [1]:
```
import numpy as np
import os
import matplotlib.pyplot as plt
import random
from PIL import Image
```

In [67]:
```
def load_image_data(data_path):
    image_file_paths=os.listdir(data_path)
    X=[]
    labels=[]

    print('image data loading from ',data_path)
    for i in image_file_paths:
        individual_image_path=data_path+"/"+i
        #print(i.split('.')[1])
        if(i.split('.')[1]=='happy'):
            labels.append(1)
        else:
            labels.append(0)
        individual_image_file = Image.open(individual_image_path)
        individual_image_file=np.array(individual_image_file,dtype='float64')
        individual_image_file=np.reshape(individual_image_file,-1)
        X.append(individual_image_file)

    # made into single numpy array of N*size_of_image
    X=np.array(X)
    labels=np.array(labels)

    print('image data loaded.')
    return X,labels

# Data Centering
def data_centering(X):
    X_mean=np.mean(X,axis=0)
    X_centered=X-X_mean
    return X_centered

# Higher dimensional PCA.
def higher_dimension_PCA(X):
    # Data Centering
    X_mean=np.mean(X,axis=0)
```

```
    X_centered=X-X_mean

    #checking the mean of the centered Data.
    #print(np.mean(X_centered,axis=0))
    #print(np.sum(np.mean(X_centered,axis=0)))

    # we are calculating the eigen values of the X@XT for calculating the eigen

    temp_outer_product=(1/N)*(X_centered@X_centered.T)
    temp_eigen_values,temp_eigen_vectors=np.linalg.eigh(temp_outer_product)

    idx = np.argsort(-temp_eigen_values)
    temp_eigen_values = temp_eigen_values[idx]
    temp_eigen_vectors = temp_eigen_vectors[:,idx]


    #print(temp_eigen_values.shape)
    #print(temp_eigen_vectors.shape)


    # For calculating the eigenvalues of the XT@X

    #print(temp_eigen_values)
    norms_of_eigen_vectors=np.sqrt(N*temp_eigen_values)
    eigen_vectors=X_centered.T@temp_eigen_vectors

    for i in range(eigen_vectors.shape[1]):
        eigen_vectors[:,i]=eigen_vectors[:,i]/norms_of_eigen_vectors[i]

    print('Eigen vectors shape',eigen_vectors.shape)

    return eigen_vectors

# Appling the LDA on projected_data
# LDA with fisher_descriminant analysis

def LDA(X):
    print('projected_data',X.shape)

    mean_1=np.zeros(K,)
    mean_0=np.zeros(K,)
    N_1=0
    N_0=0
    for i in range(N):
        if(labels[i]==1):
            mean_1=mean_1+X[i,:]
            N_1=N_1+1
        else:
            mean_0=mean_0+X[i,:]
            N_0=N_0+1

    mean_1=mean_1/N_1
    mean_0=mean_0/N_0

    SB=np.outer(mean_1-mean_0,(mean_1-mean_0))

    SW_1=np.zeros(K,)
    SW_0=np.zeros(K,)


    for i in range(N):
```

```
        if(labels[i]==1):
            SW_1=SW_1+np.outer(X[i,:]-mean_1,X[i,:]-mean_1)
        else:
            SW_0=SW_0+np.outer(X[i,:]-mean_0,X[i,:]-mean_0)

    SW=(1/N_0)*SW_0+(1/N_1)*SW_1
    print('SW',SW.shape)
    print('SB',SB.shape)

    v,w=np.linalg.eig(np.linalg.inv(SW)@SB)
    idx = np.argsort(-v)
    v = v[idx]
    w = w[:,idx]
    return v,w;

label_names=['sad','happy']
```

# Loading train image data

In [69]:

```
train_data_path='Data/emotion_classification/train'
X,labels=load_image_data(train_data_path)
N=X.shape[0]
D=X.shape[1]
print('\nThe dimension of the X (image):',D)
print('\nTotal number of samples (images) :',N)
```

```
image data loading from  Data/emotion_classification/train
image data loaded.

The dimension of the X (image): 10201

Total number of samples (images) : 20
```

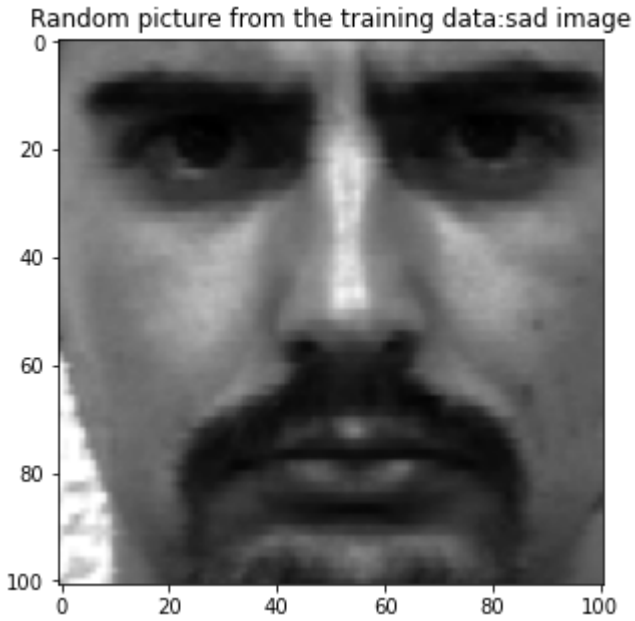
# Ploting a random picture from training data

In [89]:

```
#loading a random picture
random_picture_number=random.randint(0, 20)
#print(random_picture_number)
#print(X[random_picture_number,:].reshape(101,-1))
fig=plt.figure(figsize=(5,5))
plt.imshow(X[random_picture_number,:].reshape(101,-1),cmap='gray')
plt.title('Random picture from the training data:'+label_names[labels[random_pic
```

Out[89]:

```
Text(0.5, 1.0, 'Random picture from the training data:sad image')
```




# Evaluating the principle components of given trianing data

The number of the image samples is: $20$

The dimension of the vector is: $10201$

Therefore, the data matrix dimension is: $20 \times 10201$

The dimension of the image is much more greater than number of images.

Appling Higher-Dimensional PCA is suggestable.

In [64]:
```
# appling the Higer dimensional PCA on given training data X
# it return the all its eigen vector directions.
eigen_vectors=higher_dimension_PCA(X)
```

```
Eigen vectors shape (10201, 20)
```

# Displaying the Eigen Faces:

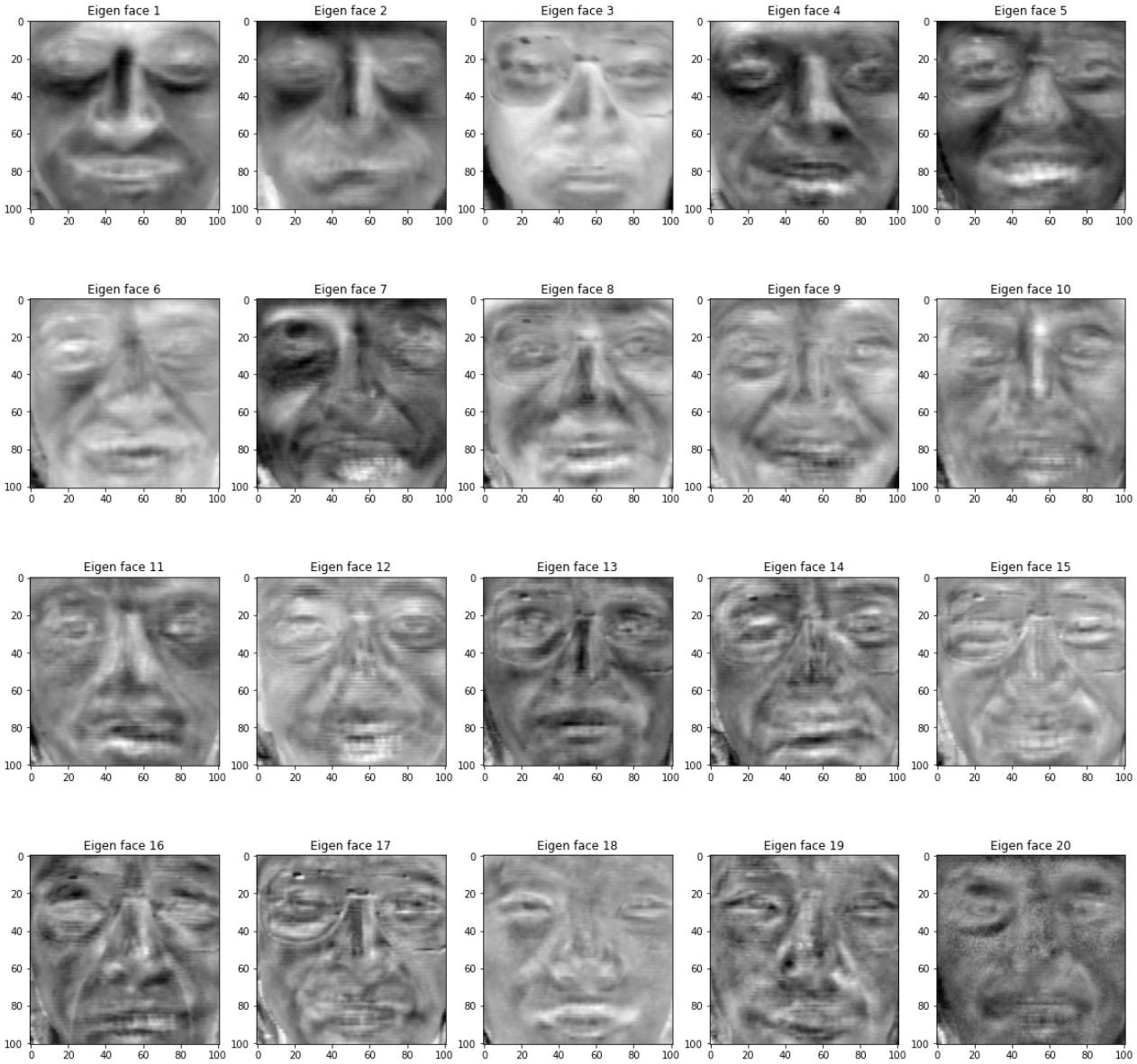
We are plotting every eigen vector by reshaping into image.

We can observe the face-features in the eigen vector.

We are plotting eigen vector of higher eigen values first. The next eigen vector is respective eigen vector of the imediate eigen value.

In [6]:
```
# plotting the eigenfaces
fig=plt.figure(figsize=(20,20))

for i in range(20):
    plot=fig.add_subplot(4,5,i+1)
```

```
    plot.set_title('Eigen face '+str(i+1))
    plt.imshow(eigen_vectors[:,i].reshape(101,101),cmap='gray')
```




# Projecting the data on K- dimensional space

We are projecting the image of 10201 dimensional vector on K-dimensional space using higher dimensional PCA Algorithm.

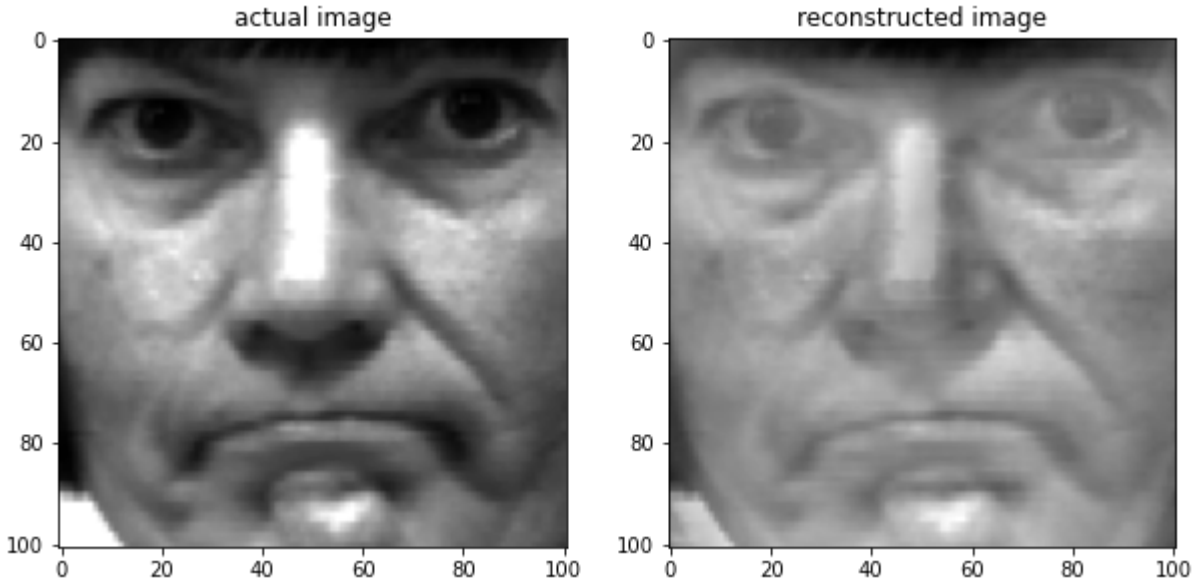
In [43]:

```
#Projecting on the K-dimensional space

K=10
X_centered=data_centering(X)
print('eigen_vectors',eigen_vectors.shape)
print('Data_shape',X_centered.shape)

projected_data=X_centered@eigen_vectors[:,0:K]
print(projected_data.shape)


fig=plt.figure(figsize=(10,10))
```

```
plot=fig.add_subplot(1,2,1)
plt.imshow(X[0,:].reshape(101,101),cmap='gray')
plot.set_title('actual image')

# reconstructing the image from projected image
recontructed_image_number=0
reconstruted_images=[]
reconstruted_image=np.zeros((10201,))

for i in range(K):
    reconstruted_image=reconstruted_image+projected_data[recontructed_image_numb
plot=fig.add_subplot(1,2,2)
plt.imshow(reconstruted_image.reshape(101,101),cmap='gray')
plot.set_title('reconstructed image')
```

```
eigen_vectors (10201, 20)
Data_shape (20, 10201)
(20, 10)
```

Out[43]:
```
Text(0.5, 1.0, 'reconstructed image')
```




# Appling LDA on Projected data:

We are using LDA algorithm for projecting on single dimensional plane.

We are using PCA projected data to projecting on single dimension plane.

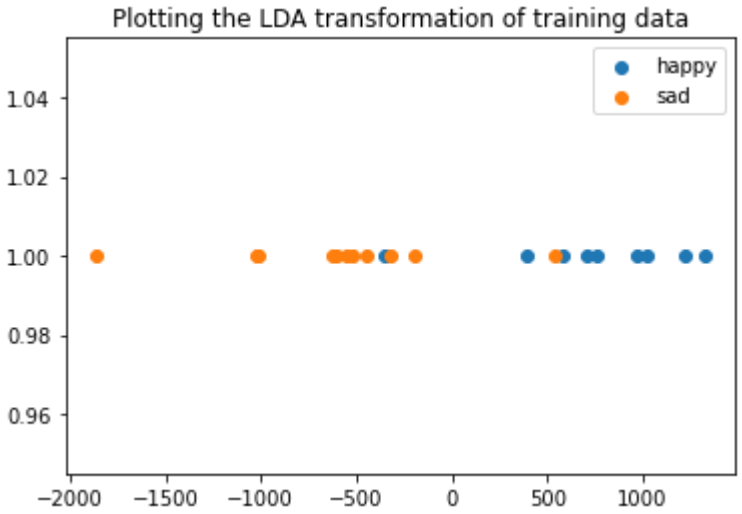
LDA uses the advantage of the supervised-ness of data to projecting.

In [123…

```
# Appling the LDA on projected_data

print('projected_data',projected_data.shape)
v,w=LDA(projected_data)

#-----------------------------------------------------------
# Projecting the data on single-plane where it seperates the data well.

largest_eigen_vector=w[:,0]
```

```
lda_classified_data=projected_data@largest_eigen_vector
print('lda_classified_data:',lda_classified_data.shape)

happy_indices=np.where(labels==1)
sad_indices=np.where(labels==0)

happy_data=lda_classified_data[happy_indices]
sad_data=lda_classified_data[sad_indices]

#print('happy_data',happy_data)
#print('sad_data',sad_data)
#print(np.ones_like(happy_data))
#print(np.ones_like(sad_data))

# Plotting LDA data
plt.scatter(happy_data,np.ones_like(happy_data))
plt.scatter(sad_data,np.ones_like(sad_data))
plt.legend(['happy','sad'])
plt.title('Plotting the LDA transformation of training data')
plt.show()
```

```
projected_data (20, 10)
projected_data (20, 10)
SW (10, 10)
SB (10, 10)
lda_classified_data: (20,)
```




# Loading the testing data:

In [45]:

```
# loading the test image data

test_data_path='Data/emotion_classification/test/'
test_X,test_labels=load_image_data(test_data_path)
```

```
(10, 10201)
```

# Using above LDA function for the testing data:

In [90]:

```
# Data Centering
test_X_centered=data_centering(test_X)
```

```
# Projecting the data on the K-space.
test_projected_data=test_X_centered@eigen_vectors[:,0:K]
print(test_projected_data.shape)

# we are taking that largest eigen vector and and projecting using that.
largest_eigen_vector=w[:,0]
test_lda_data=test_projected_data@largest_eigen_vector
print('test_lda_data',test_lda_data)

#appling the LDA
```

```
(10, 10)
test_lda_data [-741.72344715+0.j  217.40655616+0.j 1024.38642547+0.j  364.752539
84+0.j
 -982.97067882+0.j  616.5071799 +0.j  364.22994702+0.j -203.516159  +0.j
  146.66296121+0.j -805.73532462+0.j]
```

**We have used the threshold as zero.**

**If the LDA tranformation of projected data of image is greater than zero then it is happy face.**

**otherwise, it is sad face.**

In [48]:

```
# setting the threshold for the classification.
thresh_hold=0
```

We can see that, By using PCA and LDA we projected image of 10201-dimensional vector on single dimensional plane. We are also used the supervised-ness of the data using LDA.

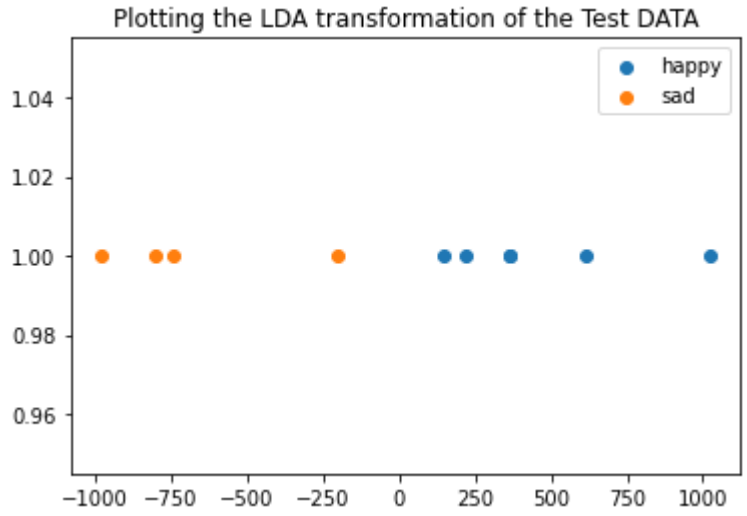
We can see the clear separation on the above plot.

blue dot represents the happy face.

orange dot represents the sad face.

In [125…

```
# retrieving the indices of the happy and sad.
happy_indices=np.where(test_labels==1)
sad_indices=np.where(test_labels==0)

test_happy_data=test_lda_data[happy_indices]
test_sad_data=test_lda_data[sad_indices]
#print('test_happy_data',test_happy_data)
#print('test_sad_data',test_sad_data)

# Plotting the Testdata using LDA .

plt.scatter(test_happy_data,np.ones_like(test_happy_data))
plt.scatter(test_sad_data,np.ones_like(test_sad_data))
plt.legend(['happy','sad'])
plt.title('Plotting the LDA transformation of the Test DATA')
plt.show()
```




For k=10, we got the 100% accuracy on the test data. We can see the above plot sad points(orange) and happy(blue) points separated very well.

In [121…

```
# displaying actual vs predicted labels.

fig=plt.figure(figsize=(20,20))
fig.suptitle('Testdata plotting (actual vs predicted labels)', fontsize=20)

for i in range(len(test_labels)):
    plot=fig.add_subplot(2,5,i+1)
    plt.imshow(test_X[i,:].reshape(101,101),cmap='gray')
    if(test_lda_data[i]>thresh_hold):
        #print('actual lable:',label_names[test_labels[i]],'predicted lable:happ
        plot.set_title('actual lable:'+label_names[test_labels[i]]+'\npredicted
    else:
        #print('actual lable:',label_names[test_labels[i]],'predicted lable:sad'
        plot.set_title('actual lable:'+label_names[test_labels[i]]+'\npredicted
```

Testdata plotting (actual vs predicted labels)

5. **Speech spectrogram** - We have clean and noisy speech files here *http://leap.ee.iisc.ac.in/sriram/teaching/MLSP21/assignments/data/speech.zip*

   The files are in wav format sampled at $16kHz$. Write a function to compute the spectrogram of clean and noisy files (use 25 ms Hamming window with a shift of 10 ms, compute 256 point magnitude FFT and retain the first 128 dimensions in each window, apply log of the magnitude of the FFT.). For example, a speech file of 3s will have a spectrogram of size $128 \times 298$ (without any padding at the end, where 128 is the dimension of the feature vector and 298 is the number of frames).

   (a) Assume that each speech frame (128 dimensional vector) is independent of each other. From the clean files, compute the whitening transform. Apply the transform on the noisy speech features. Find the average of the absolute value of the non-diagonal entries of the sampled covariance matrix of the "whitenned" clean and noisy speech features. Comment on use of whitenning when the data distribution has changed.

   (b) Repeat the above procedure by reversing the roles of clean and noisy files.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import wave
import matplotlib.image as mpimg
import numpy.linalg as la
import os
import sys
import math
```

In [2]:
```
# It's a function for the reading the audio file.

def readWavFile(filename):
    audio  = wave.open(filename,'r')
    sample_frequecny=audio.getframerate()
    total_samples=audio.getnframes()

    signal = audio.readframes(-1)
    signal = np.frombuffer(signal,'int16')
    signal = np.asarray(signal,dtype='double')
    audio_time=total_samples/sample_frequecny
    print('sample frequency:',sample_frequecny)
    print('total samples',total_samples)
    print('total time of the audio:'+str(audio_time)+" seconds")
    audio.close()
    return signal, sample_frequecny,total_samples

# it's function for the plotting the audio.
def plotAudio(data,title='title'):
    fig, ax = plt.subplots()
    #ax.plot(t, s)
    ax.plot(data)
    ax.set(xlabel='samples', ylabel='Signal', title=title)
    ax.grid()
    fig.savefig(title+".png")
    plt.show()
```

```
# it is giving the window of the audio file.
def get_window(audio,window_size,hop_index,hop_length):
    lower=hop_index*hop_length
    upper=lower+window_size
    return audio[lower:upper]

# converting the sample_window to vector
def convert_sample_window_to_vector(window_sample):
    output=np.fft.fft(window_sample,256)
    return np.log(np.abs(output[0:128]))


def get_spectrogram_vector(audio_file_path,window_type,window_size,hop_length):

    audio1, sample_frequency, total_samples = readWavFile(audio_file_path)
    total_hops=math.floor((total_samples-window_size)/hop_length+1)
    window=np.ones(window_size)
    if(window_type=='hamming'):
        window=np.hamming(window_size)
        print('\nusing hamming window')

    total_hops=math.floor((total_samples-window_size)/hop_length+1)
    spectrogram_vector=[]
    for hop_index in range(total_hops):
        sample_window=get_window(audio1, window_size, hop_index, hop_length)
        sample_window=np.multiply(sample_window,window)
        feature_vector=convert_sample_window_to_vector(sample_window)
        spectrogram_vector.append(feature_vector)
    spectrogram_vector=np.array(spectrogram_vector)


    return spectrogram_vector

def whitening(X):
    X_mean=np.mean(X,axis=0)
    X_centred=X-X_mean
    #print(np.mean(X_centred))
    L,U=np.linalg.eigh((1/N)*(X_centred.T@X_centred))
    idx = np.argsort(-L)
    L = L[idx]
    U = U[:,idx]
    L_sqrt=np.diag(np.sqrt(1/L))
    transformed_data=X_centred@U
    Y=transformed_data@L_sqrt
    return Y,L,U

def plot_audio_spectrogram(audio_feature_vector,title):
    N=audio_feature_vector.shape[0]
    fig=plt.figure(figsize=(20,20))
    plt.imshow(clean_audio_feature_vector.T)
    plt.title(title)
    plt.xlabel('Hop index')
    plt.ylabel('Frequencies')
```

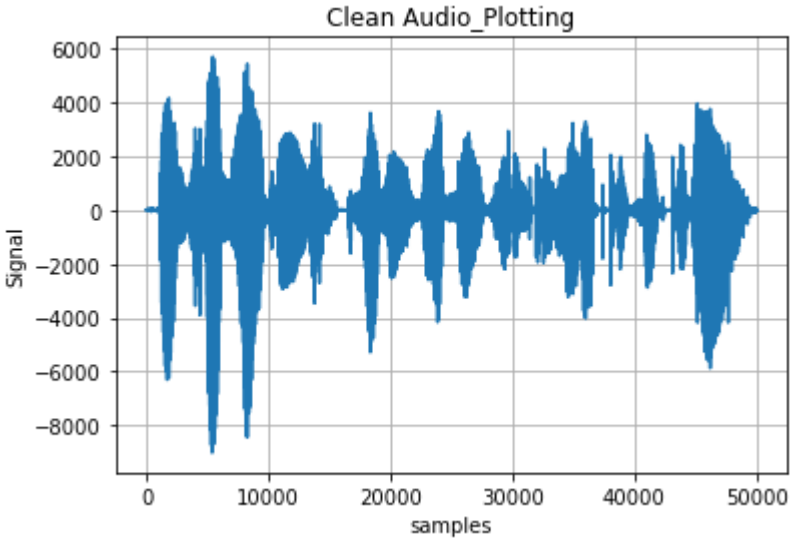
In [3]:

```
def cov_matrix(X):
    X_mean=np.mean(X,axis=0)
    X_centred=X-X_mean
    return ((1/N)*(X_centred.T@X_centred))
```

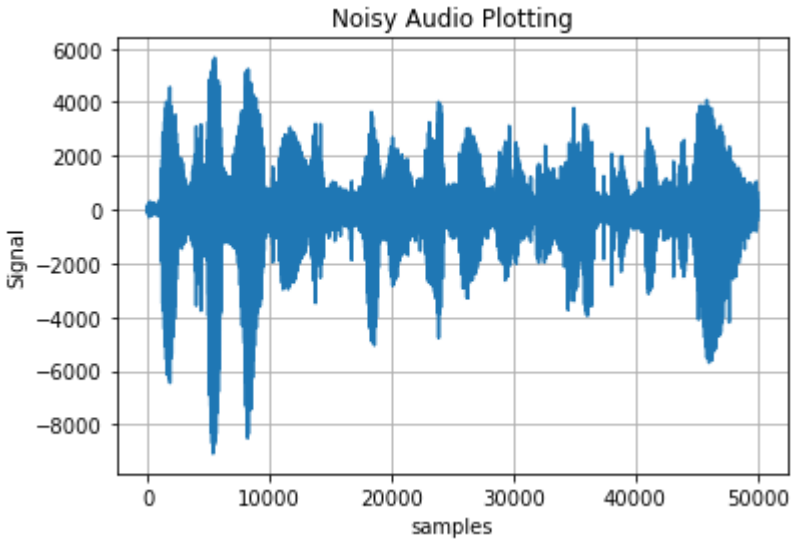
In [10]:
```
def avg_non_digonal_entry(cov_mat):
    diag_cov_mat=np.diag(cov_mat)
    diag_cov_mat=np.diag(diag_cov_mat)
    cov_mat_without_digonal=cov_mat-diag_cov_mat

    avg_non_digonal_entry=(1/(N))*(1/(N-1))*np.sum(np.abs(cov_mat_without_digona
    #print(np.diag(cov_mat_without_digonal).shape)
    #print(np.sum(cov_mat_without_digonal))
    #print('avg_non_digonal_entry',avg_non_digonal_entry)
    return avg_non_digonal_entry
```

In [6]:
```
# it's a 3.125 seconds audio file.
# it's frequecy is 16000 samples/second
# it's having 50,000 samples totally.
clean_audio_file_path='speech/speechFiles/clean.wav'

noisy_audio_file_path='speech/speechFiles/noisy.wav'

clean_audio, clean_audio_sample_frequency, clean_audio_total_samples = readWavFi
plotAudio(clean_audio,'Clean Audio_Plotting')

noisy_audio1, noisy_audio1_sample_frequency, noisy_audio1_total_samples = readWa
plotAudio(noisy_audio1,'Noisy Audio Plotting')
```

```
sample frequency: 16000
total samples 50000
total time of the audio:3.125 seconds
```




```
sample frequency: 16000
total samples 50000
total time of the audio:3.125 seconds
```




Sample frequncy of the audio file is:$16KHz$.

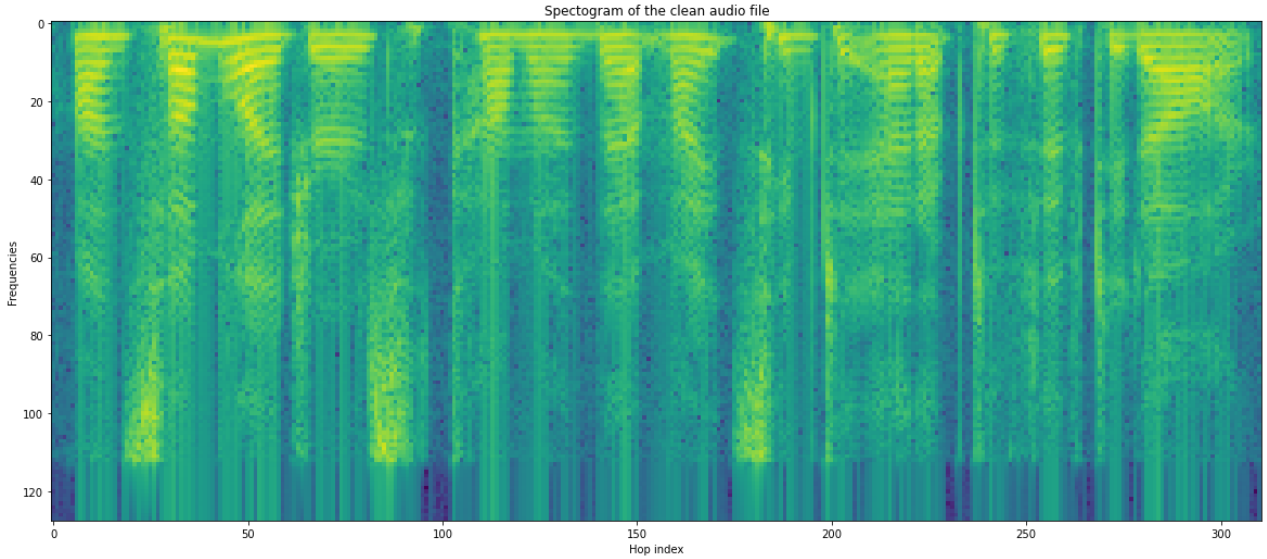
Winow size is:25 ms=25*16=400 samples.

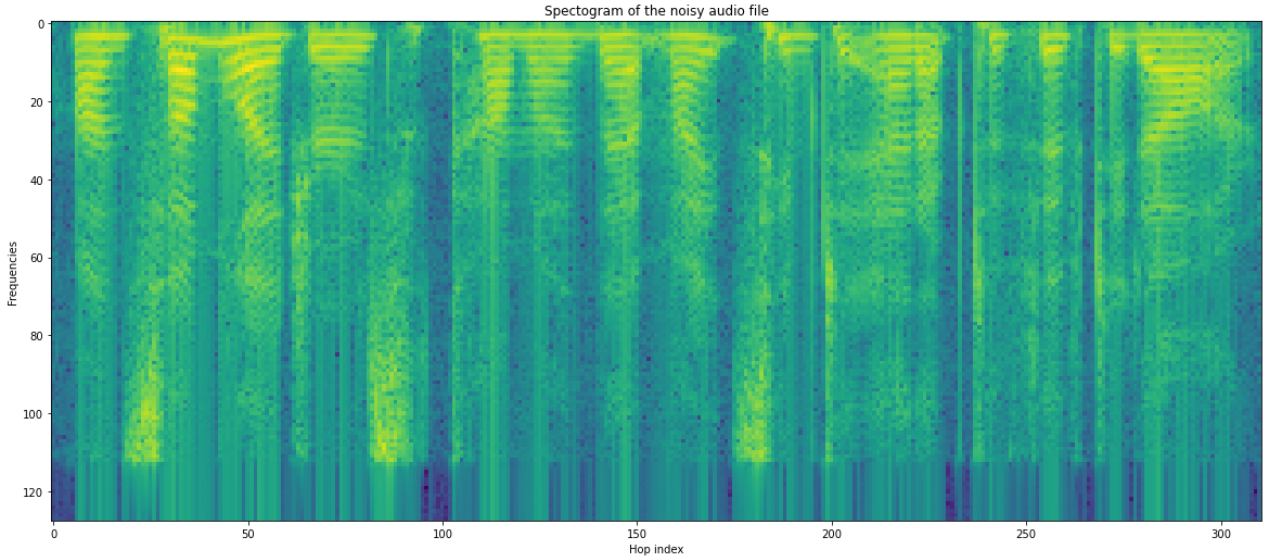
shift(Hop length) is:10 ms=10*16=160 samples.

In [7]:
```
window_type='hamming'
window_size=400
hop_length=160
hop_index=10
```

In [11]:
```
clean_audio_feature_vector=get_spectrogram_vector(clean_audio_file_path,
                                                  window_type,
                                                  window_size,
                                                  hop_length)
N=clean_audio_feature_vector.shape[0]
plot_audio_spectrogram(clean_audio_feature_vector,'Spectogram of the clean audio

noisy_audio_feature_vector=get_spectrogram_vector(noisy_audio_file_path,
                                                  window_type,
                                                  window_size,
                                                  hop_length)
N=noisy_audio_feature_vector.shape[0]
plot_audio_spectrogram(noisy_audio_feature_vector,'Spectogram of the noisy audio
```

```
sample frequency: 16000
total samples 50000
total time of the audio:3.125 seconds

using hamming window
sample frequency: 16000
total samples 50000
total time of the audio:3.125 seconds

using hamming window
```

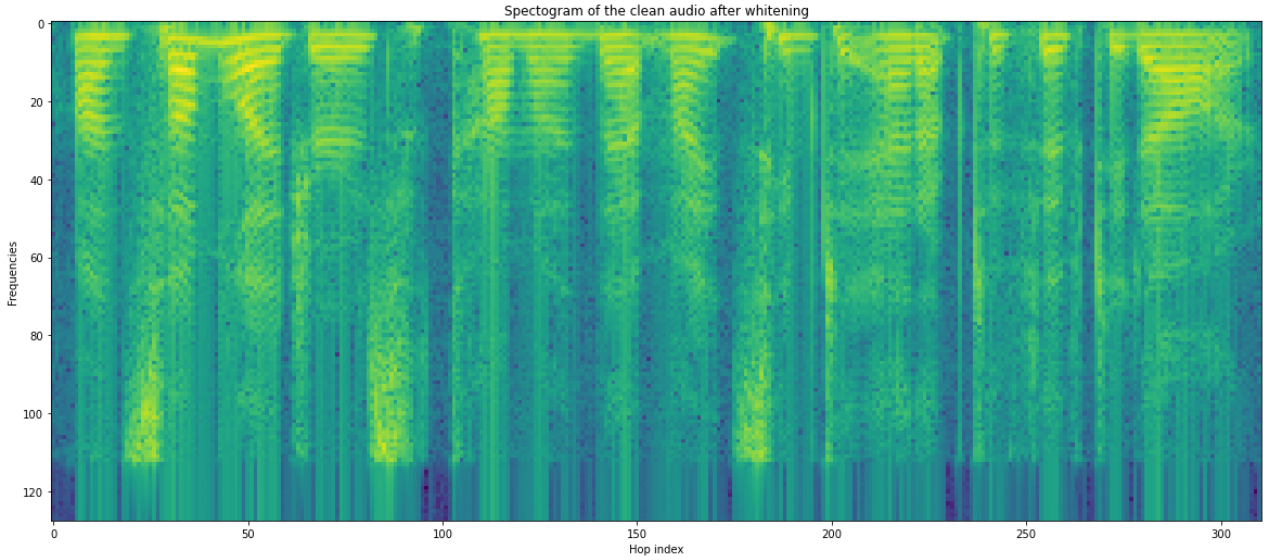



# Whitening tranformation of the Clean audio file

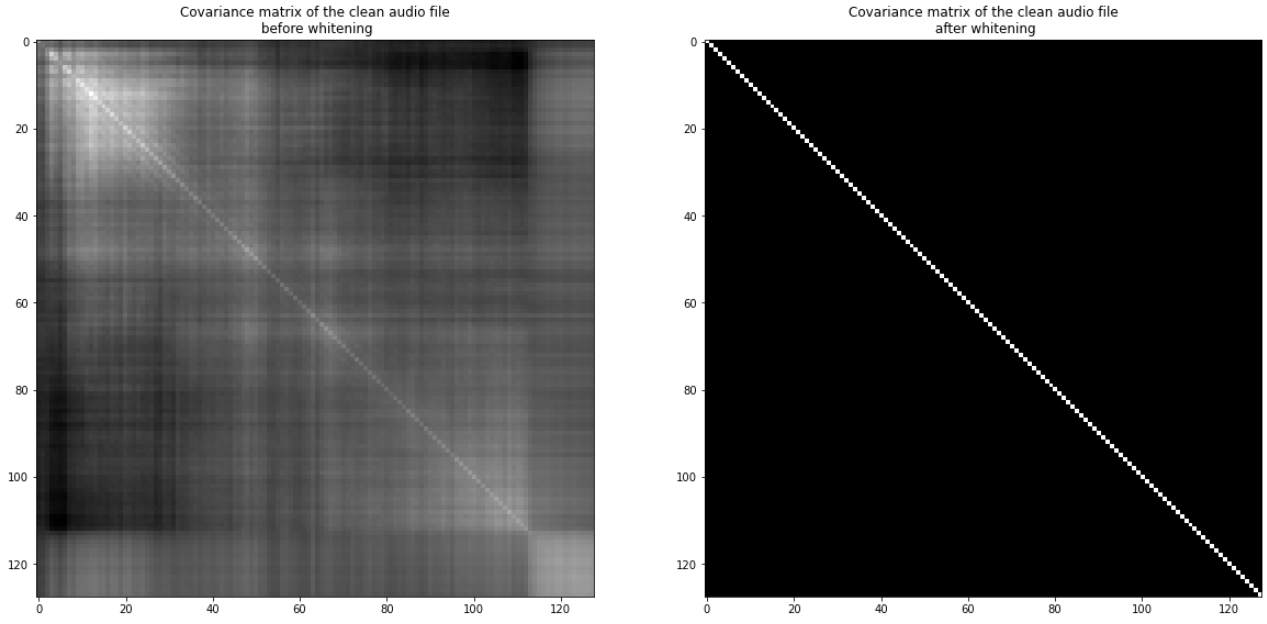
In [18]:
```
clean_audio_Y,clean_audio_L,clean_audio_U=whitening(clean_audio_feature_vector)
clean_audio_cov_mat=(1/N)*clean_audio_Y.T@clean_audio_Y
print('avg_non_digonal_entry of clean audio',avg_non_digonal_entry(clean_audio_c
plot_audio_spectrogram(clean_audio_Y,'Spectogram of the clean audio after whiter
```

```
avg_non_digonal_entry of clean audio 8.720124449301144e-16
```




In [19]:
```
fig=plt.figure(figsize=(20,20))

plot=fig.add_subplot(1,2,1)
plt.imshow(cov_matrix(clean_audio_feature_vector),cmap='gray')
plot.set_title('Covariance matrix of the clean audio file\n before whitening')
plot=fig.add_subplot(1,2,2)
plt.imshow(clean_audio_cov_mat,cmap='gray')
plot.set_title('Covariance matrix of the clean audio file\n after whitening',)
```

Out[19]: Text(0.5, 1.0, 'Covariance matrix of the clean audio file\n after whitening')




We can see the differences between the above two co-variance matrix plottings.

After whitening, the co-variance matrix of clean audio is identity matrix and all the non-digonal entries are almost zero.

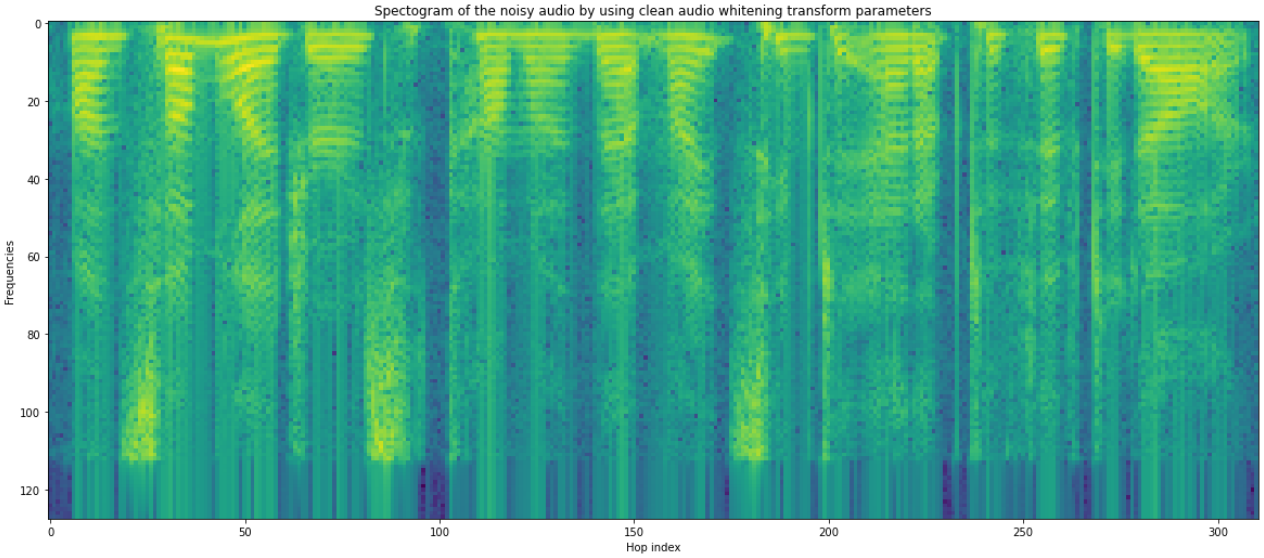
The average absoulte value of the non digonal entry for clean audio is: 8.8.720124449301144e-16

# we use the whitening tranformation of the clean

# audio file to whitening the noisy audio

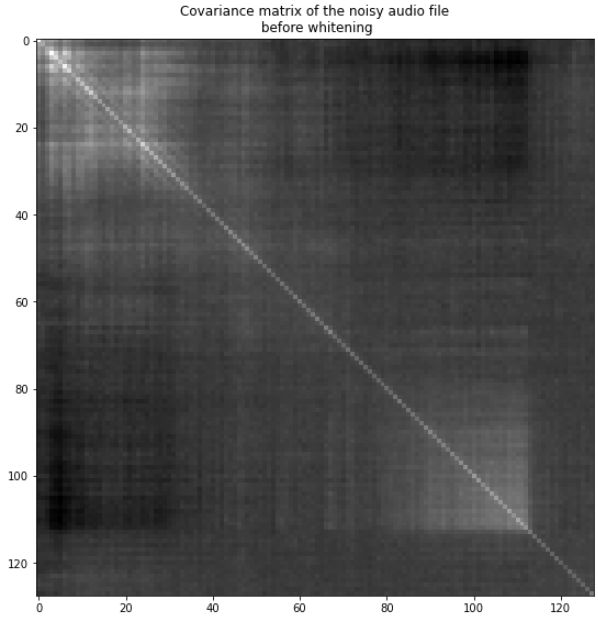
In [20]:
```
# appling that transform on the noise image

# data centering on noisy data.
noisy_audio_centred=noisy_audio_feature_vector-np.mean(noisy_audio_feature_vecto
clean_audio_L_sqrt=np.diag(np.sqrt(1/clean_audio_L))
transformed_data=noisy_audio_centred@clean_audio_U
noisy_audio_Y=transformed_data@clean_audio_L_sqrt
title='Spectogram of the noisy audio by using clean audio whitening transform pa

plot_audio_spectrogram(clean_audio_Y,title)
noisy_audio_cov_mat=(1/N)*noisy_audio_Y.T@noisy_audio_Y

print("\nwe use the whitening tranformation of the clean audio file to whitening
print('avg_non_digonal_entry of noisy audio',avg_non_digonal_entry(noisy_audio_c
```

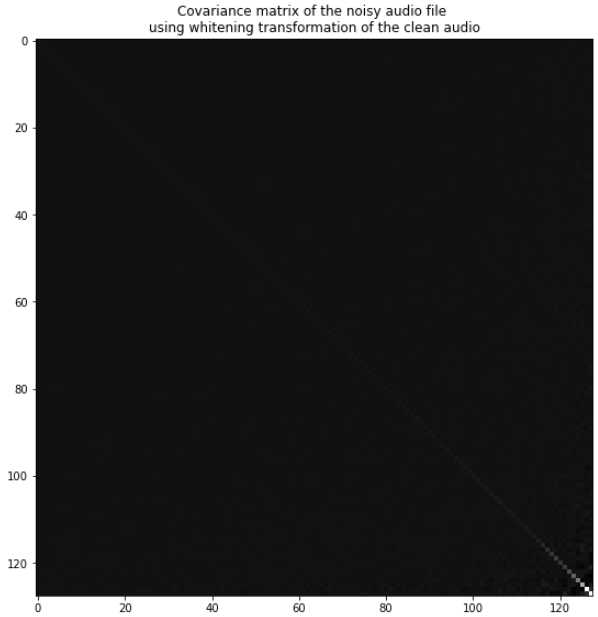
```
we use the whitening tranformation of the clean audio file to whitening the nois
y audio

avg_non_digonal_entry of noisy audio 0.025737912817858855
```




In [21]:
```
fig=plt.figure(figsize=(20,20))
plot=fig.add_subplot(1,2,1)
plt.imshow(cov_matrix(noisy_audio_feature_vector),cmap='gray')
plot.set_title('Covariance matrix of the noisy audio file\n before whitening')
plot=fig.add_subplot(1,2,2)
plt.imshow(noisy_audio_cov_mat,cmap='gray')
plot.set_title( 'Covariance matrix of the noisy audio file \nusing whitening tra
```

Out[21]:
```
Text(0.5, 1.0, 'Covariance matrix of the noisy audio file \nusing whitening tran
sformation of the clean audio')
```




The above two plots represent a noisy audio file's co-variance matrix and its whitening transformation(from clean audio whitening transformation).

In the second plot, the diagonal is not visible clearly like the previous.

The average absolute value of the non-digonal entry for noisy audio is 0.025737912817858855.

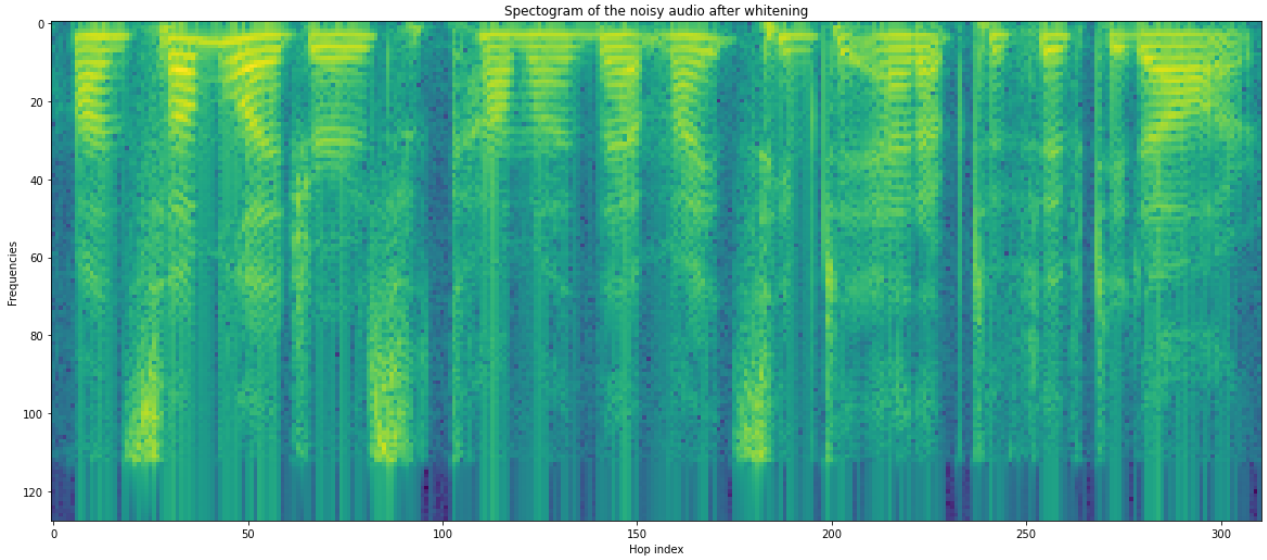
It is relatively higher than the previous case.

Since we are using the whitening-transformation of the clean audio, noise audio is not whitened properly.

# Whitening tranformation of the noisy audio file

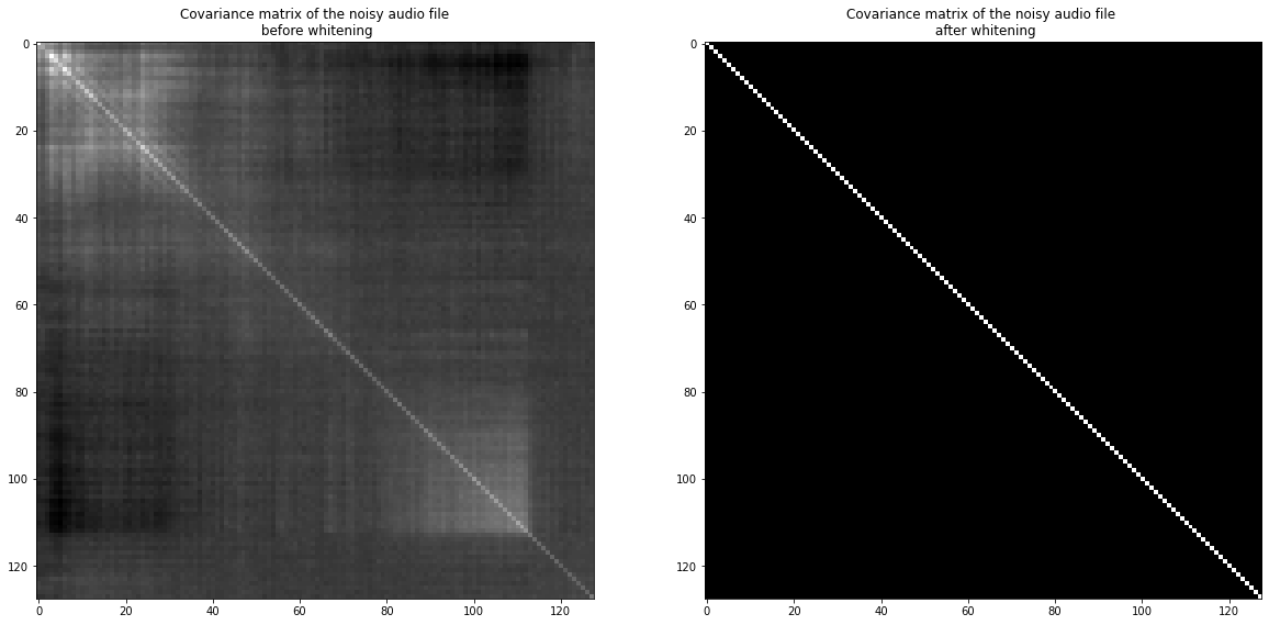
In [22]:
```
noisy_audio_Y,noisy_audio_L,noisy_audio_U=whitening(noisy_audio_feature_vector)
noisy_audio_cov_mat=(1/N)*noisy_audio_Y.T@noisy_audio_Y
print('avg non digonal_entry of noisy_audio cov_matrix',avg_non_digonal_entry(nc

plot_audio_spectrogram(noisy_audio_Y,'Spectogram of the noisy audio after whiter
# appling that transform on the noise image
```

```
avg non digonal_entry of noisy_audio cov_matrix 9.255733703258624e-17
```




In [23]:
```
fig=plt.figure(figsize=(20,20))
plot=fig.add_subplot(1,2,1)
plt.imshow(cov_matrix(noisy_audio_feature_vector),cmap='gray')
plot.set_title('Covariance matrix of the noisy audio file\n before whitening')
plot=fig.add_subplot(1,2,2)
plt.imshow(noisy_audio_cov_mat,cmap='gray')
plot.set_title( 'Covariance matrix of the noisy audio file \n after whitening')
```

Out[23]: Text(0.5, 1.0, 'Covariance matrix of the noisy audio file \n after whitening')




The above two plots represent the covariance matrices of the noisy audio and its whitening transformation.

We can see the differences between the above two covariance matrix plottings.

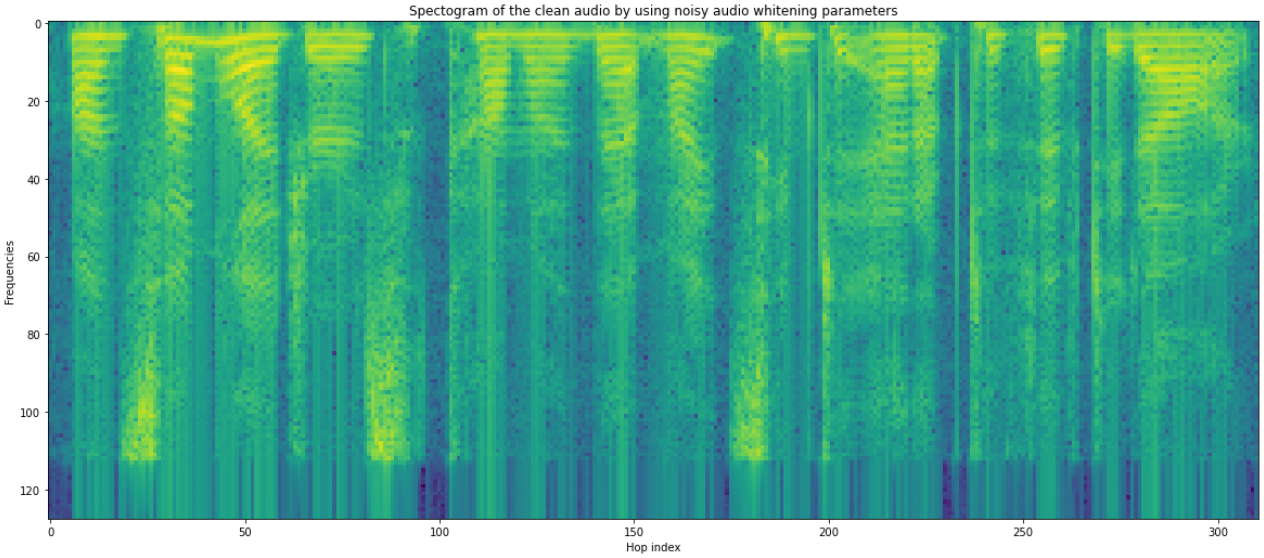
After whitening, the covariance matrix of noisy audio is the identity matrix, and all the non-diagonal entries are almost zero.

The average absolute value of the non-diagonal entry for noisy audio is: 9.255733703258624e-17

# we use the whitening tranformation of the noisy audio file to whitening the clean audio

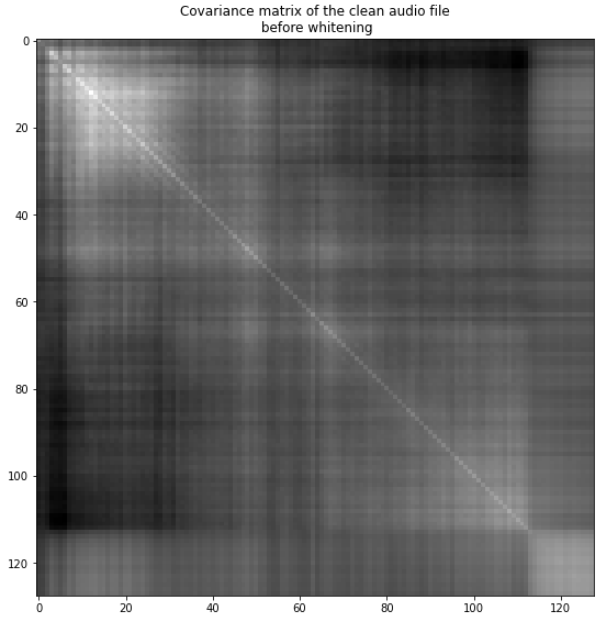
In [25]:
```
# data centering on noisy data.
clean_audio_centred=clean_audio_feature_vector-np.mean(clean_audio_feature_vecto
noisy_audio_L_sqrt=np.diag(np.sqrt(1/noisy_audio_L))
transformed_data=clean_audio_centred@noisy_audio_U
clean_audio_Y=transformed_data@noisy_audio_L_sqrt
clean_audio_cov_mat=(1/N)*clean_audio_Y.T@clean_audio_Y
plot_audio_spectrogram(clean_audio_Y,'Spectogram of the clean audio by using noi
print("\nwe use the whitening tranformation of the noisy audio file to whitening
print('avg_non_digonal_entry of clean audio',avg_non_digonal_entry(clean_audio_c
```

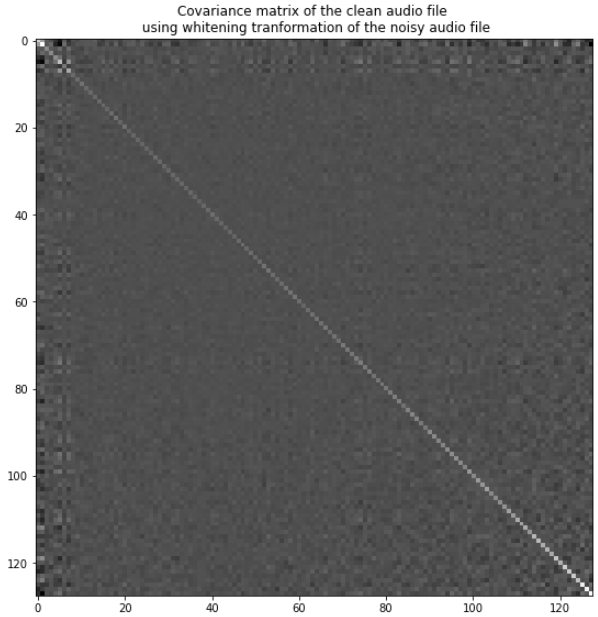
```
we use the whitening tranformation of the noisy audio file to whitening clean au
dio

avg_non_digonal_entry of clean audio 0.020631940128433213
```




In [27]:
```
fig=plt.figure(figsize=(20,20))
plot=fig.add_subplot(1,2,1)
plt.imshow(cov_matrix(clean_audio_feature_vector),cmap='gray')
plot.set_title('Covariance matrix of the clean audio file\n before whitening')
plot=fig.add_subplot(1,2,2)
plt.imshow(clean_audio_cov_mat,cmap='gray')
plot.set_title( 'Covariance matrix of the clean audio file \n using whitening tr
```

Out[27]:
```
Text(0.5, 1.0, 'Covariance matrix of the clean audio file \n using whitening tra
nformation of the noisy audio file')
```




The above two plots represent the clean audio file's covariance matrix and whitening transformation of it (from noisy audio whitening-transformation).

In the second plot, the diagonal elements are not visible as in the previous plot, and non-diagonal's are also not zero (we can observe some black and white spots).

The average absolute value of the non-diagonal entry for noisy audio is 0.020631940128433213.

It is relatively higher than the previous case.

Since we are using the whitening-transformation of the noisy audio, clean audio is not appropriately whitened.

</font>

***Comment on use of whitenning when the data distribution has changed.***

If you apply a whitening transformation on the same data (on which we are computed the whitening transformation), it performs very well. It means after whitening transformation co-variance matrix will become a proper identity matrix.

But, when using the same whitening transformation on some other data, it is not performing well.

It is neither good at un-correlating the non-diagonal parameters nor making diagonal parameters to unit variance because the transformation parameters(L, U) may be related to the data distribution.

Therefore, when using the same transformation on some other data, it may not work correctly.

We can conclude that whitening transformation of some data distribution may not be the proper choice for some other distribution.