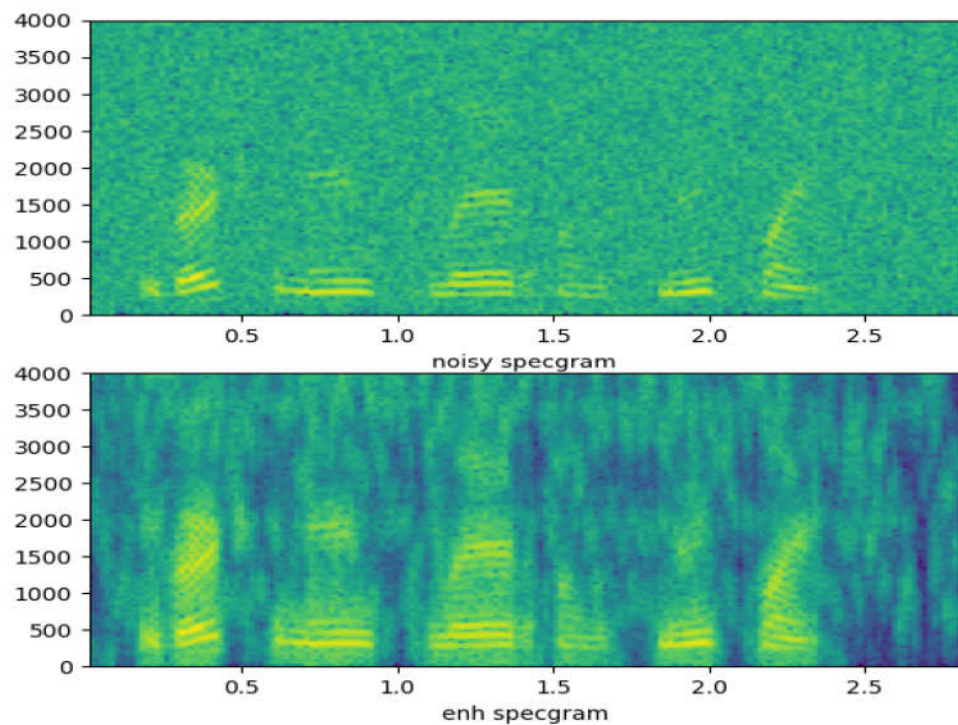


# 语音增强-子空间法

## Speech Enhancement- Subspace



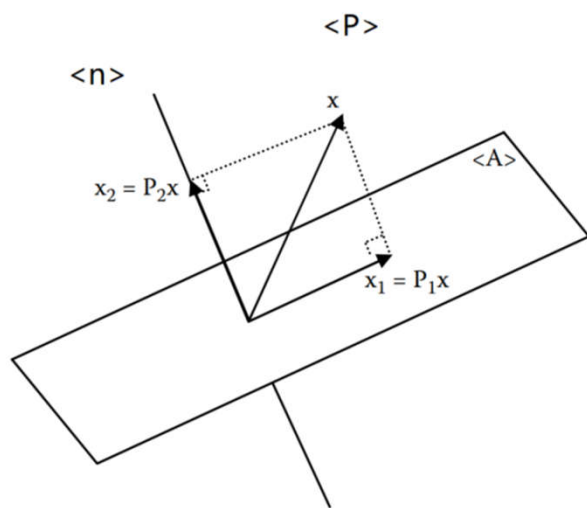
于泓

鲁东大学

信息与电气工程学院

2021.7.29

# 子空间 ( Subspace )



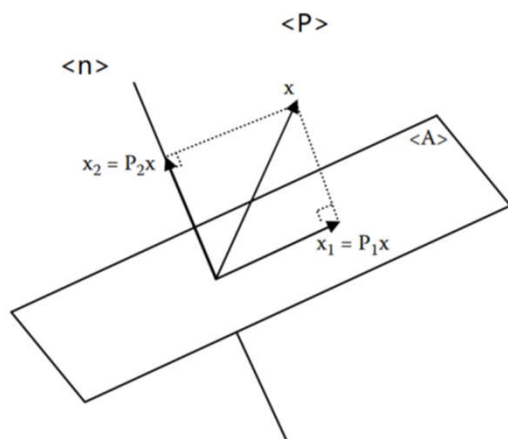
设计一个映射矩阵 $P$ ，能够将 $N$ 维向量（含噪信号）映射到2个相互正交的子空间  $\langle A \rangle$  和  $\langle n \rangle$   
即信号子空间和噪声子空间

其中 $P$  可以用 $N$ 个列向量来表示  $\{P_1, \dots, P_N\}$

信号子空间表示为  $P_1 = \{P_1, P_2, \dots, P_M\}$

噪声子空间表示为  $P_2 = \{P_{M+1}, \dots, P_N\}$

正交性:  $P_i P_i = 1$   
 $P_i P_j = 0$



P域的正变换与反变换

$$\mathbf{y}^p = \mathbf{P}\mathbf{y}$$

$$\mathbf{P} \in (\mathbf{N}, \mathbf{N}) \quad \mathbf{y} \in (\mathbf{N}, 1)$$

$$\mathbf{y} = \mathbf{P}^{-1}\mathbf{y}^p = \mathbf{P}^T\mathbf{y}^p \text{ (如果正交)}$$

$$\mathbf{P} \in (\mathbf{N}, \mathbf{N}) \quad \mathbf{y} \in (\mathbf{N}, 1)$$

### 去噪过程:

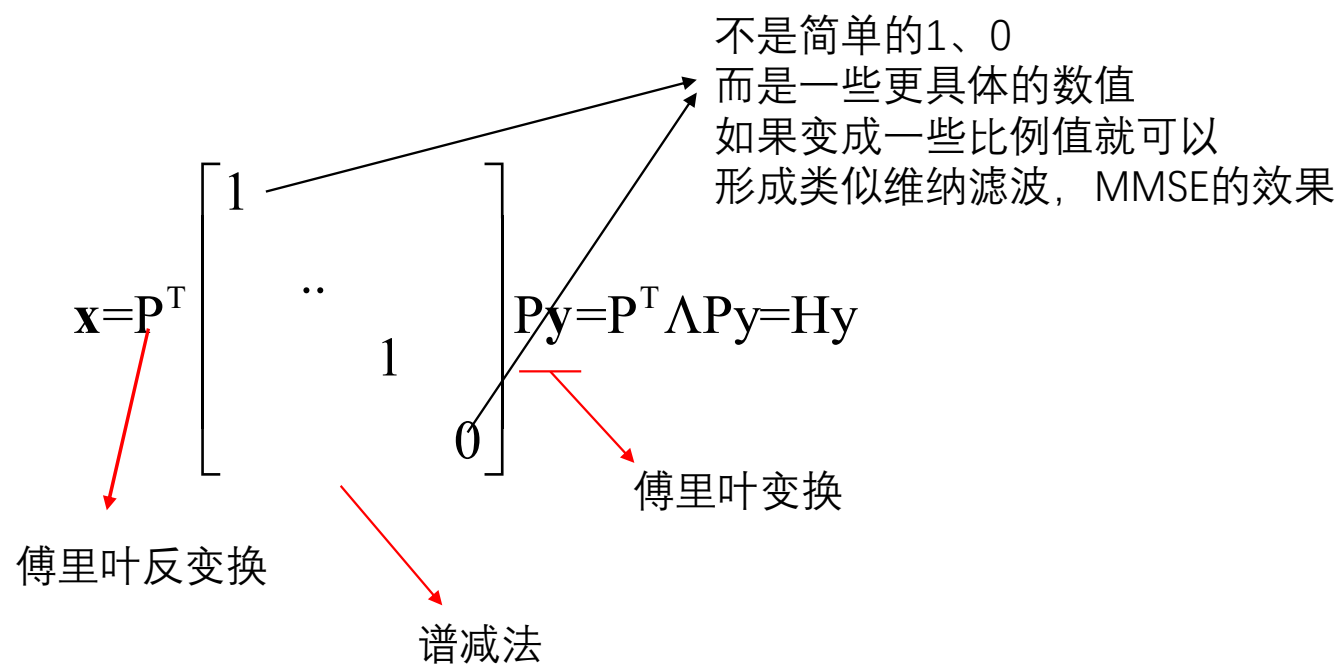
噪声信号  $y = x + n$ ，在时域上  $x$  和  $n$  本来很难分开，但是经过  $P$  的映射，将信号映射到  $P$  域后， $y^p$ ，落在子空间  $\langle A \rangle$  上的部分就是信号即  $y^p$  的前  $M$  个维度就是干净信号。落在空间  $\langle n \rangle$  上的部分即后  $N-M$  个维度即噪声部分。将噪声部分清零，再进行  $P$  域的反变换，就得到增强后的信号。

$$\mathbf{x} = \mathbf{P}^T \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{P}\mathbf{y} = \mathbf{P}^T \Lambda \mathbf{P}\mathbf{y} = \mathbf{H}\mathbf{y}$$

P域反变换
P域映射

保留信号子空间  
去除噪声子空间

## 与谱减法很类似



子空间法的本质就是寻找一个H能够将含噪信号y映射成干净信号x

$$y=x+n$$

$$\hat{x}=Hy$$

$$\varepsilon=\hat{x}-x=(H-I) \cdot x+H \cdot n=\varepsilon_x+\varepsilon_n$$

语音损失
噪声残留

定义:

$$\overline{\varepsilon_x^2}=E[\varepsilon_x^T \varepsilon_x]=tr\left(E[\varepsilon_x \varepsilon_x^T]\right) \longrightarrow \text{语音损失能量}$$

$$\overline{\varepsilon_n^2}=E[\varepsilon_n^T \varepsilon_n]=tr\left(E[\varepsilon_n \varepsilon_n^T]\right) \longrightarrow \text{噪声残留能量}$$

求解H使得:

$$\min_H \overline{\varepsilon_x^2}$$

同时满足  $\frac{1}{N} \overline{\varepsilon_n^2} < \delta^2$

特征维度
允许噪声冗余

$$\overline{\varepsilon_x^2}=tr\left(HR_x H-HR_x -R_x H+R_x\right)$$

$$\overline{\varepsilon_n^2}=tr\left(HR_n H^T\right)$$

n的协方差矩阵
x的协方差矩阵

### 利用拉格朗日乘子法进行求解

$$L(H, \mu) = \text{tr}(H R_x H^T - H R_x - R_x H^T + R_x) + \mu \text{tr}(H R_n H^T - N \delta^2)$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{H}} = \mathbf{0}$$

$$R_x H^T - R_x + \mu R_n H^T = 0$$

$$H=R_x(R_x+\mu R_n)^{-1}$$

对 $R_x$ 进行特征值分解

$R_x = U \Delta_x U^T$

实对称

正交阵

特征值  
对角阵

最终获得：

$$H_{opt} = U \Delta_{\mathbf{x}} (\Delta_{\mathbf{x}} + \mu U^T R_{\mathbf{n}} U)^{-1} U^T$$

不是对角阵

在做高斯白噪声假设的前提下:

即  $R_n = \sigma_n^2 I$        $U^T R_n U$  是对角线矩阵

设法将  $U^T R_n U$  对角线化

寻找  $\Lambda_x V$  使得:

$$V^T R_x V = \Lambda_x$$

$$V^T R_n V = I$$

$\Lambda_x V$  分别是

$$\Sigma = R_n^{-1} R_x$$

的特征值与特征向量

此时:

$$H_{opt} = V^{-T} \Lambda_x (\Lambda_x + \mu I)^{-1} V^T$$

$$G = \Lambda_x (\Lambda_x + \mu I)^{-1} \quad \text{对角阵}$$

$$g_{kk} = \begin{cases} \frac{\lambda_x^{(k)}}{\lambda_x^{(k)} + \mu}, & k = 1, 2, \dots, M \\ 0, & k = M + 1, \dots, \end{cases}$$

M的选取?

将小于0  
的特征值  
置0

=0 谱减  
=1 维纳

由于 $R_x$ 不方便直接计算

假设:  $R_y = R_x + R_n$

$$\Sigma = R_n^{-1} R_x = R_n^{-1} (R_y - R_n) = R_n^{-1} R_y - I$$

$\mu$  的估计

$$\mu = \mu_0 - (\text{SNR}_{\text{dB}})/s$$

与信噪比  
成反比

$$\text{SNR} = \frac{\text{tr}(V^T R_x V)}{\text{tr}(V^T R_n V)} = \frac{\sum_{k=1}^M \lambda_x^{(k)}}{N}.$$

具体流程:

(1) 信号分帧

(2) 估计 $R_y$ 和 $R_n$

(3) 计算:

$$\Sigma = R_n^{-1} R_y - I.$$

(4) 对  $\Sigma$  进行特征值分解  
计算特征值和特征向量  $\Lambda_x$   $V$

(5) 计算 SNR 估计  $\mu$

(6) 计算对角线矩阵 $G$

$$g_{kk} = \begin{cases} \frac{\lambda_x^{(k)}}{\lambda_x^{(k)} + \mu}, & k = 1, 2, \dots, M \\ 0, & k = M+1, \dots, K \end{cases}$$

(7) 计算  $H = V^{-T} G V^T$

(8) 进行增强  $x = Hy$

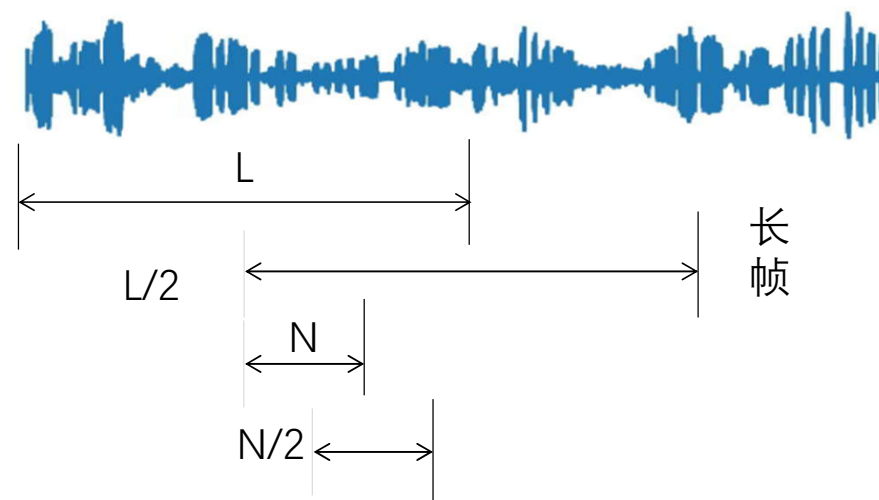


## $R_y$ 与 $R_n$ 的估计

整个算法的好坏主要取决于协方差矩阵的估计  
在语音增强的算法中，分成的**时域帧，不易过长**  
一般取值在30-40个样本点。

在进行 $R_y$ 与 $R_n$ 的估计时，为了能够准确估计往往需要截取较长的信号。

因此在实际的操作中，需要先截取**长帧**（ $L$ ）进行 $R_y$ 的估计（ $N \times N$ ,  $N \ll L$ ），并推算 $H$   
再将**长帧**，分成**短帧**进行增强。



时域语音信号的协方差矩阵应当是Toeplitz(托普利兹)矩阵

每条对角线上的元素相同

a: 当前点  
与当前  
点的相关性

b: 当前点与邻近  
点的相关性

$$\begin{bmatrix}
 a & b & c & d & e \\
 b & a & b & c & d \\
 c & b & a & b & c \\
 d & c & b & a & b \\
 e & d & c & b & a
 \end{bmatrix}$$

c 表示当前点和间隔点的相关性

## $R_y$ 与 $R_n$ 的估计

### (1) 一般求解方法

$$\mathbf{y}^L \bullet \mathbf{y}^{LT} = \begin{bmatrix} y_1 y_1 & y_1 y_2 & y_1 y_3 & & & & \\ & y_2 y_2 & y_2 y_3 & y_2 y_4 & & & \\ & & y_3 y_3 & y_3 y_4 & y_3 y_5 & & \\ & & & y_4 y_4 & y_4 y_5 & y_4 y_6 & \\ & & & & y_5 y_5 & y_5 y_6 & \dots \\ & & & & & y_6 y_6 & \dots & y_{L-2} y_L \\ & & & & & & \dots & y_{L-1} y_L \\ & & & & & & & y_L y_L \end{bmatrix}$$

取斜对角线元素取均值，取N条对角线

这里默认  $y_1$  到  $y_L$  对协方差矩阵的贡献都是相同的

### (2) 多尺度窗口法

利用一组窗函数对  $y^{(L)}$  进行不同方式的加权

$$\mathbf{y}_w^L \bullet \mathbf{y}_w^{LT}$$

$$\mathbf{y}_w^L = [w_1 \otimes y^L, w_2 \otimes y^L, \dots, w_C \otimes y^L]$$

C个窗函数对  $y^L$  进行加权拼接后生成  $y_w^L$  ( $N \times C$ )

$$w_{ci} = \sqrt{\frac{2}{L+1}} \sin\left(\frac{\pi c}{L+1} i\right)$$

$$c = 1, \dots, C$$

$$i = 1, \dots, L$$

## 具体代码

```
import librosa
from librosa.util import frame
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt
from scipy.linalg import toeplitz
```

# 读取原始

```
noisy_wav_file = "in_SNR5_sp01.wav"
noisy_speech, fs = librosa.load(noisy_wav_file, sr=None)
```

```
subframe_dur= 4 #子帧的时长为4ms
len_subframe= int(np.floor( fs* subframe_dur/ 1000))
```

```
P= len_subframe # 协方差矩阵的大小 P*P
```

```
frame_dur= 32 # 帧的时长
len_frame = int(np.floor(frame_dur* fs/ 1000))
```

```
len_step_frame= int(len_frame/ 2) #帧移
```

# 帧 和 子帧的窗函数

```
window_frame = np.hamming(len_frame)
window_subframe= np.hamming(len_subframe)
```

分别  
设置帧长

以及子帧  
的长度

## 进行Rn的估计

L=16 # 多窗口法进行Rx估计时窗口的数据

```
# 进行噪声协方差矩阵的估计
# 假设前120ms是噪声
noise_dur = 120
N_noise=int(np.floor( noise_dur* fs/ 1000))
noise= noisy_speech[:N_noise]

# 获取一组窗函数
tapers= sine_taper( L, N_noise)
# 进行噪声的协方差矩阵估计
Rn = estimate_R(noise,P,tapers)
iRn = np.linalg.inv(Rn)
```

```
# 构建多窗口函数
def sine_taper(L, N):
    tapers= np.zeros( [N, L]);
    index = np.array([i+1 for i in range(N)])
    for i in range(L):
        tapers[:,i] = np.sqrt(2/(N+1))* np.sin(np.pi * (i+1)*index/(N+1))

    return tapers
```

```
# x 输入信号长度为N
# p 估计的协方差矩阵的大小 p<<N
# W 一组窗口函数 大小为 N x L
def estimate_R(x,p,W):

    N,L = np.shape(W)
    x_rep = np.tile(x,[L,1]) # L x N
    x_rep = x_rep.T          # N x L

    # 对信号进行加窗处理
    x_w= W* x_rep

    # 矩阵相乘
    R1= np.dot(x_w, x_w.T)

    r = np.zeros(p)
    for i in range(p):
        r[i] = np.sum(np.diag(R1,k=i))

    R_est = toeplitz(r)
    return R_est
```

$$w_{ci} = \sqrt{\frac{2}{L+1}} \sin\left(\frac{\pi c}{L+1} i\right)$$

```

# 对噪声信号进行分帧
noisy_frames = frame(noisy_speech, len_frame, len_step_frame,axis = 0 )
# 获取帧数
N_frame = noisy_frames.shape[0]

# 获取用来进行Ry估计的tapers
tapers_noisy = sine_taper( L, len_frame)

# 存储增强后的语音帧
enh_frames = np.zeros(np.shape(noisy_frames))

```

```

vad_thre= 1.2 # 是否进行Rn 更新的阈值
mu_vad= 0.98 # Rn 更新参数

```

VAD判定

$$\Sigma = R_n^{-1} R_y - I$$

特征值分解

```

# 逐帧处理
for n in range(N_frame):

```

```

    # 读取一帧数据
    noisy = noisy_frames[n]

    # 进行协方差矩阵 Ry 估计
    Ry = estimate_R(noisy,P,tapers_noisy)

```

R<sub>y</sub>的估计

```

    # 利用 VAD 进行 Rn 的更新
    vad_ratio= Ry[0,0]/ Rn[0,0]
    if vad_ratio<= vad_thre:
        Rn= mu_vad* Rn+ (1- mu_vad)* Ry
        iRn= np.linalg.inv( Rn)

```

R<sub>n</sub>的估计

```

    # 计算iRnRx
    In= np.eye(P)
    iRnRx= np.dot(iRn, Ry)- In

    # 获取 特征向量 V与对角线特征值矩阵 D
    d, V = np.linalg.eig(iRnRx)
    iV= np.linalg.inv(V)

```

```
# 进行信噪比估计 将特征值小于0 的部分置
# 用来估计 delta_X
d[d<0]=0
dRx = d
```

```
# 根据信噪比计算 mu
SNR = np.sum(dRx)/P
SNR_db = 10 * np.log10( SNR+ 1e-10)
if SNR_db >= 20:
    mu = mu_toplus
elif SNR_db < 20 and SNR_db >= -5 :
    mu = mu0- SNR_db * mu_slope
else:
    mu = mu_tominus
```

```
# 计算增益系数
gain_vals= dRx/( dRx+mu)
G= np.diag( gain_vals)
```

```
# 获得映射矩阵
H = np.dot(np.dot(iV.T,G),V.T)
```

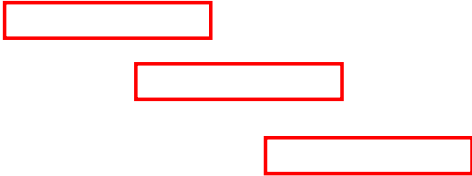
计算  $H = V^{-T}GV^T$

$$g_{kk} = \begin{cases} \frac{\lambda_x^{(k)}}{\lambda_x^{(k)} + \mu}, & k = 1, 2, \dots, M \\ 0, & k = M + 1, \dots, K \end{cases}$$

```
mu_max=5 # mu的最大值
mu_toplus= 1 # mu的最小值
# 估计mu 时使用的参数
mu_tominus= mu_max;
mu_slope= (mu_tominus- mu_toplus )/ 25;
mu0= mu_toplus+ 20* mu_slope;
```

$$\mu = \begin{cases} \mu_0 - (\text{SNR}_{\text{dB}})/s, & -5 < \text{SNR}_{\text{dB}} < 20 \\ 1 & \text{SNR}_{\text{dB}} \geq 20 \\ 5 & \text{SNR}_{\text{dB}} \leq -5 \end{cases}$$

```
# 对frame 进行分帧
sub_frames = frame(noisy, len_subframe, int(len_subframe/2),axis = 0) # N * d
# 逐帧 sub_frames 进行增强
enh_sub_frames = np.dot(sub_frames,H.T) # N x d
# 加窗
enh_sub_frames = enh_sub_frames * window_subframe
# 信号恢复
enh_signal = frame2singal(enh_sub_frames)
enh_frames[n] = enh_signal
```



```
# 信号重构
def frame2singal(frames):
    N,d = np.shape(frames)

    half_frame = int(d/2)
    overlap = np.zeros(half_frame)
    len_singal = d + (N-1)*(half_frame)
    start = 0
    singal = np.zeros(len_singal)
    for i in range(N):
        temp = frames[i]
        singal[start:start+half_frame] = temp[:half_frame] + overlap
        overlap = temp[half_frame:]
        start = start + half_frame
    singal[start:] = overlap
    return singal
```



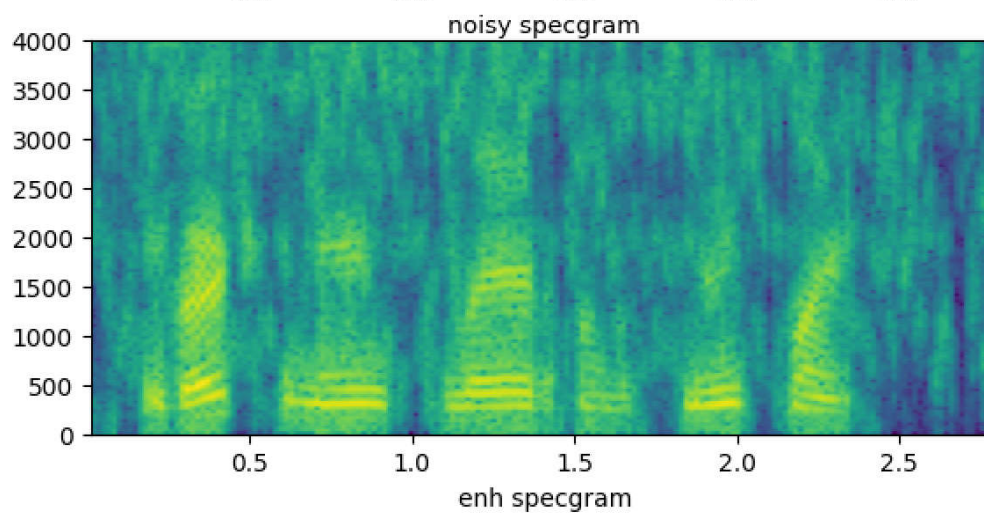
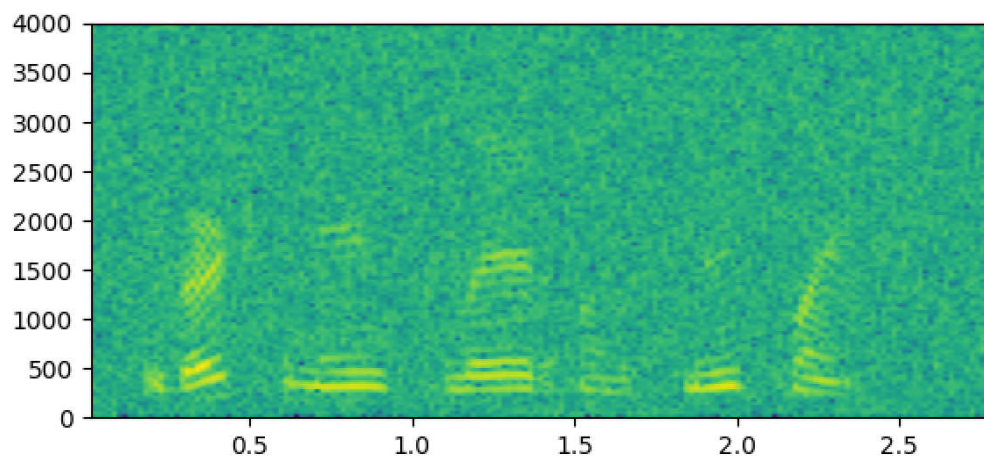
```
# 对增强帧进行加窗
enh_frames = enh_frames*window_frame
enh_wav = frame2singal(enh_frames)

sf.write("enhce_klt.wav",enh_wav,fs)

plt.subplot(2,1,1)
plt.specgram(noisy_speech,NFFT=256,Fs=fs)
plt.xlabel("noisy specgram")
plt.subplot(2,1,2)
plt.specgram(enh_wav,NFFT=256,Fs=fs)
plt.xlabel("enh specgram")
plt.show()
```

## 参考文献

- [1] Hu, Y. and Loizou, P. (2003). A generalized subspace approach for enhancing speech corrupted by colored noise. IEEE Trans. on Speech and Audio Processing, 11, 334-341.



子空间法



