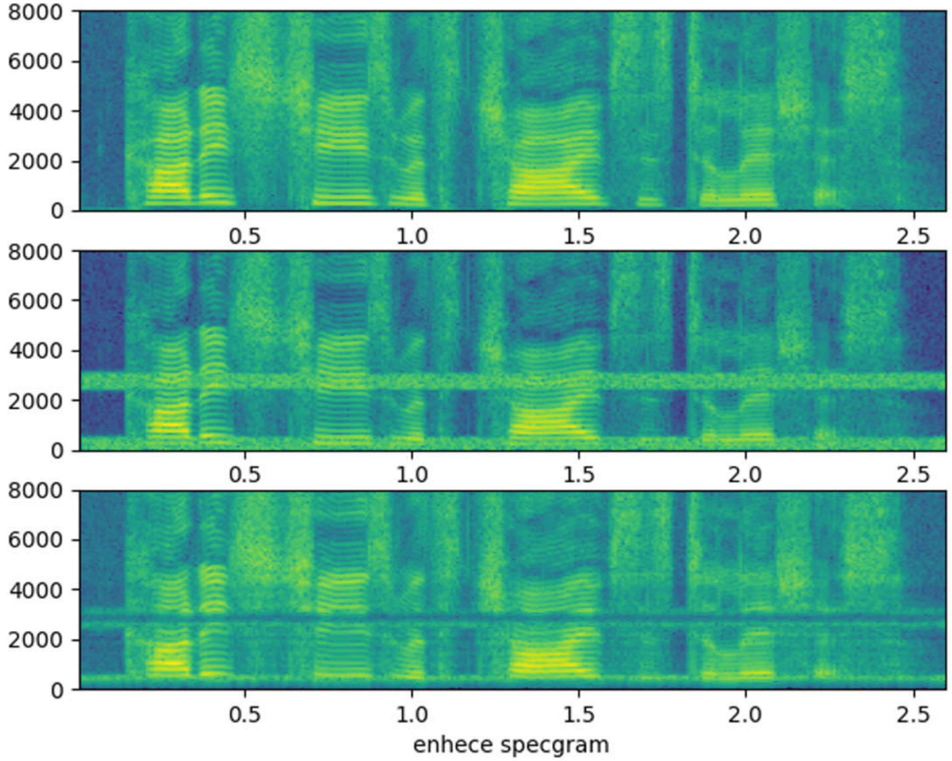
语音增强-维纳滤波

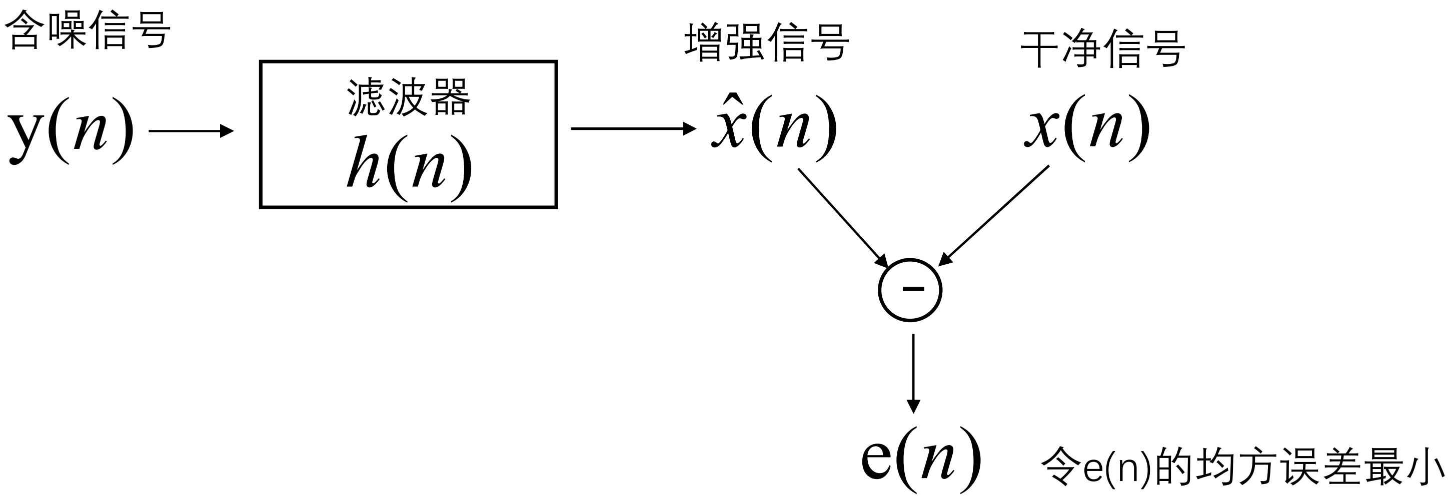
Speech Enhancement- Wiener Filter



于泓
鲁东大学
信息与电气工程学院

2021.6.28

维纳滤波（Wiener Filter）



设计h(n)可以使e(n)的均方误差最小

频率域分析

$$\hat{x}(n)=h(n)*y(n)$$

$$\hat{X}(\omega)=H(\omega)*Y(\omega)$$

假设，各个频段之间互不相关
可以针对每个频点k进行分析

$$\hat{X}(\omega_k)=H(\omega_k)*Y(\omega_k)$$

$$E(\omega_k)=X(\omega_k)-\hat{X}(\omega_k)=X(\omega_k)-H(\omega_k)*Y(\omega_k)$$

维纳滤波器

$$H(\omega_k)=\frac{P_{yx}^*(\omega_k)}{P_{yy}(\omega_k)}$$

根据最小均方误差准则

$$\mathrm{E}\left[\left|E(\omega_k)\right|^2\right]=\mathrm{E}\left\{[X(\omega_k)-H(\omega_k)Y(\omega_k)]^*[X(\omega_k)-H(\omega_k)Y(\omega_k)]\right\}$$

$$=\mathrm{E}[\left|X(\omega_k)\right|^2]$$

$$-H(\omega_k)\mathrm{E}[X^*(\omega_k)Y(\omega_k)]-H^*(\omega_k)\mathrm{E}[X(\omega_k)Y^*(\omega_k)]$$

$$+\left|H(\omega_k)\right|^2\mathrm{E}[\left|Y(\omega_k)\right|^2]$$

令 $P_{yy}(\omega_k)=E\left[\left|Y(\omega_k)\right|^2\right]$ $\quad P_{yx}(\omega_k)=E\left[Y(\omega_k)X(\omega_k)^*\right]$

$$\mathrm{J}=\mathrm{E}[\left|X(\omega_k)\right|^2]-H(\omega_k)P_{yx}(\omega_k)-H^*(\omega_k)P_{yx}^*(\omega_k)+\left|H(\omega_k)\right|^2P_{yy}(\omega_k)$$

求导可得

$$\mathrm{J}=\mathrm{E}[\left|X(\omega_k)\right|^2]-H(\omega_k)P_{yx}(\omega_k)-H^*(\omega_k)P_{yx}^*(\omega_k)+\left|H(\omega_k)\right|^2P_{yy}(\omega_k)$$

$$\frac{\partial \mathrm{J}}{\partial H(\omega_k)}=H^*(\omega_k)P_{yy}(\omega_k)-P_{yx}(\omega_k)=\left[H(\omega_k)P_{yy}(\omega_k)-P_{yx}^*(\omega_k)\right]^*=0$$

应用于语音去噪

$$H(\omega_k)=\frac{P_{yx}^{*}(\omega_k)}{P_{yy}(\omega_k)}$$

$$y(n)=x(n)+\mathrm{n}(n)$$
$$Y(\omega_k)=X(\omega_k)+N(\omega_k)$$

$$H(\omega_k)=\frac{P_{yx}^{*}(\omega_k)}{P_{yy}(\omega_k)}$$

$$P_{yx}^{*}(\omega_k)=\mathrm{E}\left[X(\omega_k)\{X(\omega_k)+N(\omega_k)\}^{*}\right]$$
$$=\mathrm{E}\left[X(\omega_k)X^{*}(\omega_k)\right]+\mathrm{E}\left[X(\omega_k)N^{*}(\omega_k)\right]$$
$$=P_{xx}(\omega_k)$$

X与N互不相关，且N的期望为0

$$P_{yy}(\omega_k)=\mathrm{E}\left[\{X(\omega_k)+N(\omega_k)\}\{X(\omega_k)+N(\omega_k)\}^{*}\right]$$
$$=\mathrm{E}\left[X(\omega_k)X^{*}(\omega_k)\right]+\mathrm{E}\left[N(\omega_k)N^{*}(\omega_k)\right]$$
$$=P_{xx}(\omega_k)+P_{nn}(\omega_k)$$

$$H(\omega_k)=\frac{P_{yx}^{*}(\omega_k)}{P_{yy}(\omega_k)}=\frac{P_{xx}(\omega_k)}{P_{xx}(\omega_k)+P_{nn}(\omega_k)}$$

定义 $\xi_k=\dfrac{P_{xx}(\omega_k)}{P_{nn}(\omega_k)}$ 为先验信噪比

$$H(\omega_k)=\frac{\xi_k}{\xi_k+1}$$

$$0<H(\omega_k)<1$$

物理意义 当信噪比大时，允许信号通过
当信噪比小时，抑制信号通过

几个变种

平方根维纳滤波

$$\hat{X}(\omega_k) = \sqrt{H(\omega_k)} * Y(\omega_k)$$

$$E\left|\hat{X}(\omega_k)\right|^2 = \left(\sqrt{H(\omega_k)}\right)^2 * E\left(\left|Y(\omega_k)\right|^2\right)$$

$$P_{\hat{x}\hat{x}}(\omega_k) = H(\omega_k)P_{yy}(\omega_k)$$

将下式带入 可得:

$$H(\omega_k) = \frac{P_{yx}^{*}(\omega_k)}{P_{yy}(\omega_k)} = \frac{P_{xx}(\omega_k)}{P_{xx}(\omega_k) + P_{nn}(\omega_k)}$$

$$P_{\hat{x}\hat{x}}(\omega_k) = P_{xx}(\omega_k)$$

保证增强后信号的能量谱与干净语音的能量谱相同

参数型维纳滤波器

$$H(\omega_k) = \left(\frac{P_{xx}(\omega_k)}{P_{xx}(\omega_k) + \alpha P_{nn}(\omega_k)}\right)^{\beta}$$

$$H(\omega_k) = \left(\frac{\xi_k}{\xi_k + \alpha}\right)^{\beta}$$

如果能够提前获取一些先验知识
例如：噪声集中在那些频带等
可以灵活的设置参数

a越大对输出的抑制越强烈。

在噪声大的频带选取较大的a，在噪声小的频带选取较小的a

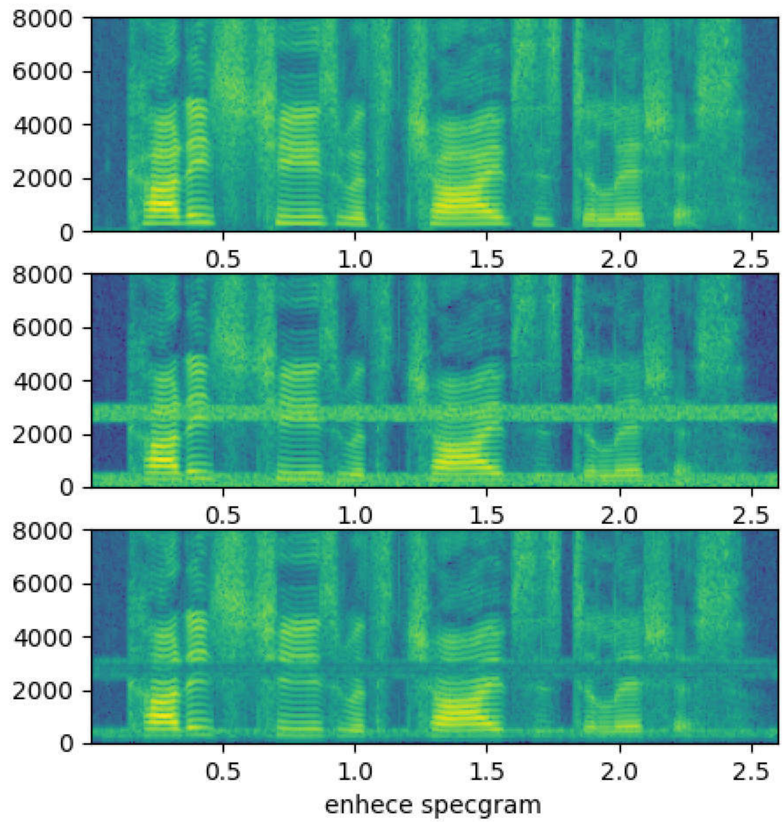
代码实现

情况1： 已知干净语音clean以及含噪语音noisy，求解滤波器H

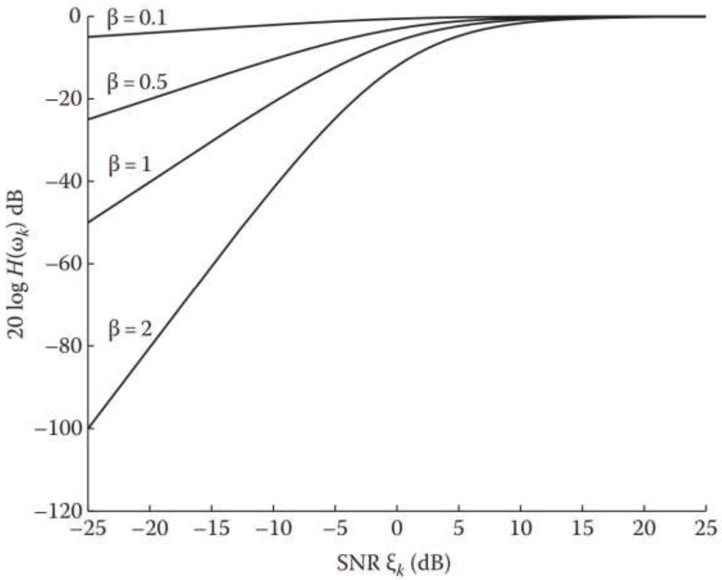
适用场景举例：在通信过程中，语音信号通过一个参数未知的噪声信道，在发送端加入一个约定好的已知信号（如，在有用信号的末尾）；在接收端，收取信号后，利用含噪的已知信号求解滤波器H，**并利用其对其余信号进行滤波去噪。**

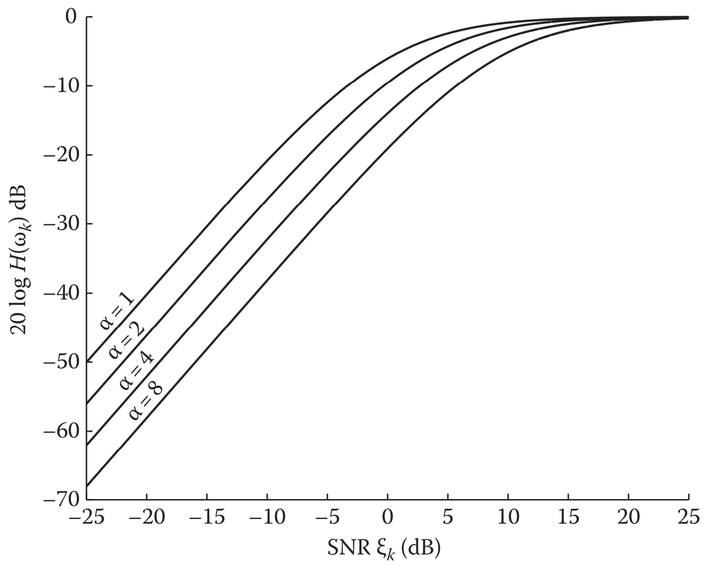
```
def wiener_filter(noisy,clean,noise,para):
    n_fft = para["n_fft"]
    hop_length = para["hop_length"]
    win_length = para["win_length"]
    alpha = para["alpha"]
    beta = para["beta"]

    S_noisy = librosa.stft(noisy,n_fft=n_fft, hop_length=hop_length, win_length=win_length)   #DxT
    S_noise = librosa.stft(noise,n_fft=n_fft, hop_length=hop_length, win_length=win_length)
    S_clean = librosa.stft(clean,n_fft=n_fft, hop_length=hop_length, win_length=win_length)

    Pxx = np.mean((np.abs(S_clean))**2,axis=1,keepdims=True) # Dx1
    Pnn = np.mean((np.abs(S_noise))**2,axis=1,keepdims=True)

    H = (Pxx/(Pxx+alpha*Pnn))**beta

    S_enhec = S_noisy*H

    enhenc = librosa.istft(S_enhec, hop_length=hop_length, win_length=win_length)
```

```python
if __name__ == "__main__":

    # 读取干净语音
    clean_wav_file = "sf1_cln.wav"
    clean,fs = librosa.load(clean_wav_file,sr=None)

    # 读取读取噪声语音
    noisy_wav_file = "sf1_n0L.wav"
    noisy,fs = librosa.load(noisy_wav_file,sr=None)

    # 获取噪声
    noise = noisy-clean

    # 设置模型参数
    para = {}
    para["n_fft"] = 256
    para["hop_length"] = 128
    para["win_length"] = 256
    para["alpha"] = 1
    para["beta"] = 5

    # 维纳滤波
    H,enhenc = wiener_filter(noisy,clean,noise,para)

    sf.write("enhce.wav",enhenc,fs)


    plt.subplot(3,1,1)
    plt.specgram(clean,NFFT=256,Fs=fs)
    plt.xlabel("clean specgram")
    plt.subplot(3,1,2)
    plt.specgram(noisy,NFFT=256,Fs=fs)
    plt.xlabel("noisy specgram")
    plt.subplot(3,1,3)
    plt.specgram(enhenc,NFFT=256,Fs=fs)
```



$$H(\omega_k) = \left(\frac{P_{xx}(\omega_k)}{P_{xx}(\omega_k) + \alpha P_{nn}(\omega_k)} \right)^{\beta}$$





α，β的不同对输出信号的抑制作用

情况2: 干净语音未知的情况下，可以利用谱减法先进行噪声估计，再进行滤波

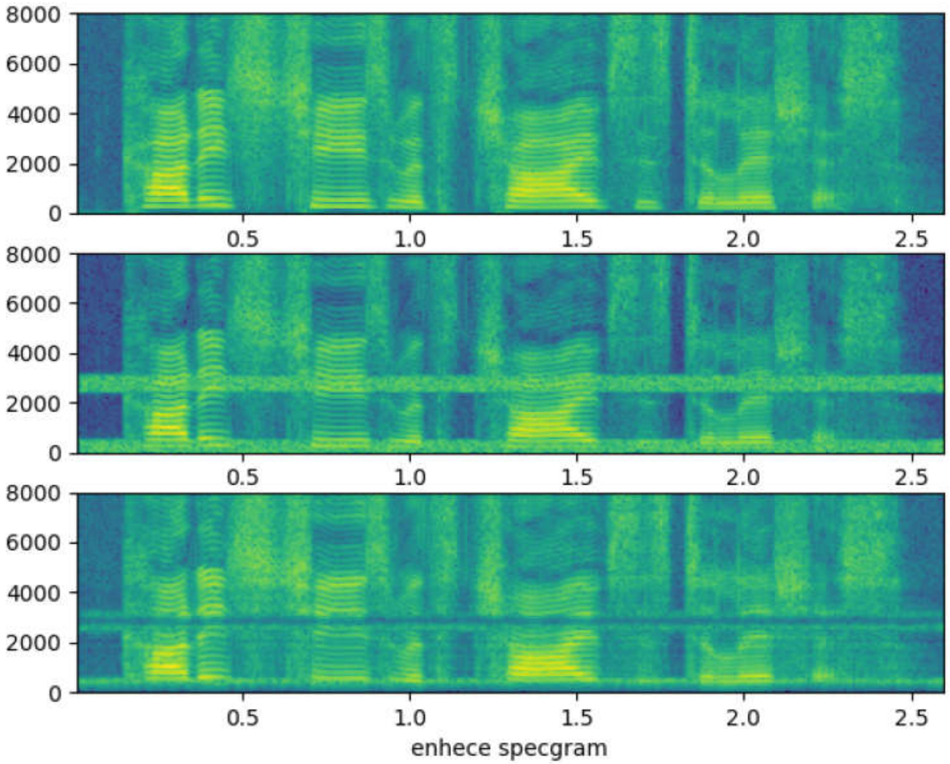
重写谱减法代码

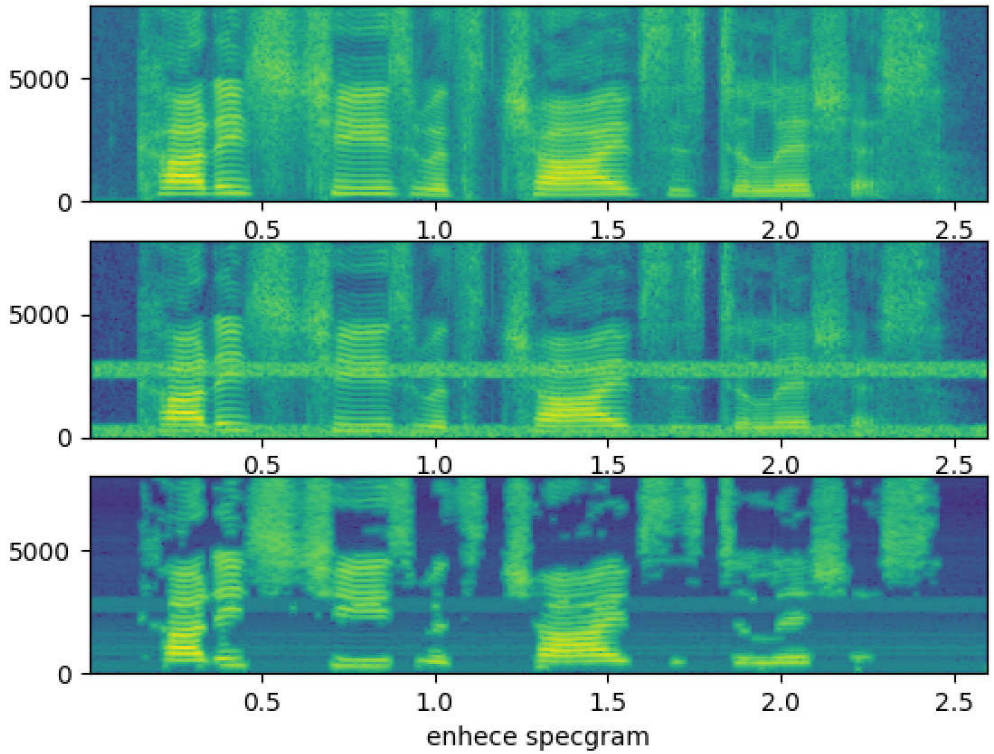
```
def sub_spec(noisy,noise,para):
    n_fft = para["n_fft"]
    hop_length = para["hop_length"]
    win_length = para["win_length"]

    # 计算noisy的频谱
    S_noisy = librosa.stft(noisy,n_fft=n_fft, hop_length=hop_length, win_length=win_length)
    D,T = np.shape(S_noisy)
    Mag_noisy= np.abs(S_noisy)
    Phase_nosiy= np.angle(S_noisy)
    Power_nosiy = Mag_noisy**2

    # 计算noise的频谱
    S_noise = librosa.stft(noise,n_fft=n_fft, hop_length=hop_length, win_length=win_length)
    Mag_nosie = np.mean(np.abs(S_noise),axis=1,keepdims=True)
    Power_nosie = Mag_nosie**2
    Power_nosie = np.tile(Power_nosie,[1,T])
```

```python
## 方法3 引入平滑
Mag_noisy_new = np.copy(Mag_noisy)
k = para["k"]
for t in range(k,T-k):
    Mag_noisy_new[:,t] = np.mean(Mag_noisy[:,t-k:t+k+1],axis=1)
Power_nosiy = Mag_noisy_new**2

# 超减法去噪
alpha = para["alpha"]
gamma = para["gamma"]

Power_enhenc = np.power(Power_nosiy,gamma) - alpha*np.power(Power_nosie,gamma)
Power_enhenc = np.power(Power_enhenc,1/gamma)

# 对于过小的值用 beta* Power_nosie 替代
beta = para["beta"]
mask = (Power_enhenc>=beta*Power_nosie)-0
Power_enhenc = mask*Power_enhenc + beta*(1-mask)*Power_nosie
Mag_enhenc = np.sqrt(Power_enhenc)
```

```python
Mag_enhenc_new  = np.copy(Mag_enhenc)
# 计算最大噪声残差
maxnr = np.max(np.abs(S_noisy[:,:31])-Mag_nosie,axis =1)

k = 1
for t in range(k,T-k):
    index = np.where(Mag_enhenc[:,t]<maxnr)[0]
    temp = np.min(Mag_enhenc[:,t-k:t+k+1],axis=1)
    Mag_enhenc_new[index,t] = temp[index]

# 对信号进行恢复
S_enhec = Mag_enhenc_new*np.exp(1j*Phase_nosiy)
enhenc = librosa.istft(S_enhec, hop_length=128, win_length=256)

return enhenc
```

```
if __name__ == "__main__":

    # 读取干净语音
    clean_wav_file = "sf1_cln.wav"
    clean,fs = librosa.load(clean_wav_file,sr=None)

    # 读取读取噪声语音
    noisy_wav_file = "sf1_n0L.wav"
    noisy,fs = librosa.load(noisy_wav_file,sr=None)


    # 设置谱减法模型参数
    para_sub_spec = {}
    para_sub_spec["n_fft"] = 256
    para_sub_spec["hop_length"] = 128
    para_sub_spec["win_length"] = 256
    para_sub_spec["alpha"] = 4
    para_sub_spec["beta"] = 0.0001
    para_sub_spec["gamma"] =1
    para_sub_spec["k"] =1

    # 利用谱减法估计噪声
    # 前5000点 大约30帧作为噪声
    est_clean = sub_spec(noisy,noisy[:5000],para_sub_spec)

    est_noise = noisy[:len(est_clean)]-est_clean

    # 设置维纳滤波模型参数
    para_wiener = {}
    para_wiener["n_fft"] = 256
    para_wiener["hop_length"] = 128
    para_wiener["win_length"] = 256
    para_wiener["alpha"] = 1
    para_wiener["beta"] =3

    # 维纳滤波
    H,enhenc = wiener_filter(noisy,est_clean,est_noise,para_wiener)

    sf.write("enhce_2.wav",enhenc,fs)


    plt.subplot(3,1,1)
    plt.specgram(clean,NFFT=256,Fs=fs)
    plt.xlabel("clean specgram")
    plt.subplot(3,1,2)
    plt.specgram(noisy,NFFT=256,Fs=fs)
    plt.xlabel("noisy specgram")
    plt.subplot(3,1,3)
    plt.specgram(enhenc,NFFT=256,Fs=fs)
    plt.xlabel("enhece specgram")
    plt.show()
```


8000
6000
4000
2000
0
0.5
1.0
1.5
2.0
2.5
enhece specgram


5000
0
0.5
1.0
1.5
2.0
2.5
enhece specgram

和谱减法的比较

情况2:如果已知噪声的某些分布特性，可以通过人为向干净语音加噪声的方法获取一组训练数据，利用这些数据训练维纳滤波器H进行滤波

获取颜色噪声

```
# 获得颜色噪声
# N 噪声样本点数目
# fs 采样率
# f_L,f_H  噪声所在频带

def gen_color_noise(N,order_filter,fs,f_L,f_H):

    noise = np.random.randn(N)
    m_firwin = firwin(order_filter, [2*f_L/fs, 2*f_H/fs], pass_zero="bandpass")
    color_noise = lfilter(m_firwin, 1.0, noise)
    return color_noise
```

添加某信噪比噪声

```
def add_nosie(clean,noise,snr):
    # 干净信号与噪声信号不相等，退出
    if not len(clean) == len(noise):
        print("")
        return False

    # 干净信号的能量
    p_clean = np.sum(np.abs(clean)**2)

    # 噪声信号的能量
    p_noise = np.sum(np.abs(noise)**2)

    # 计算缩放因子
    scale =  np.sqrt( (p_clean/p_noise) * np.power(10,-snr/10) )

    # 获得加性噪声
    noisy = clean + scale * noise

    return noisy
```

list

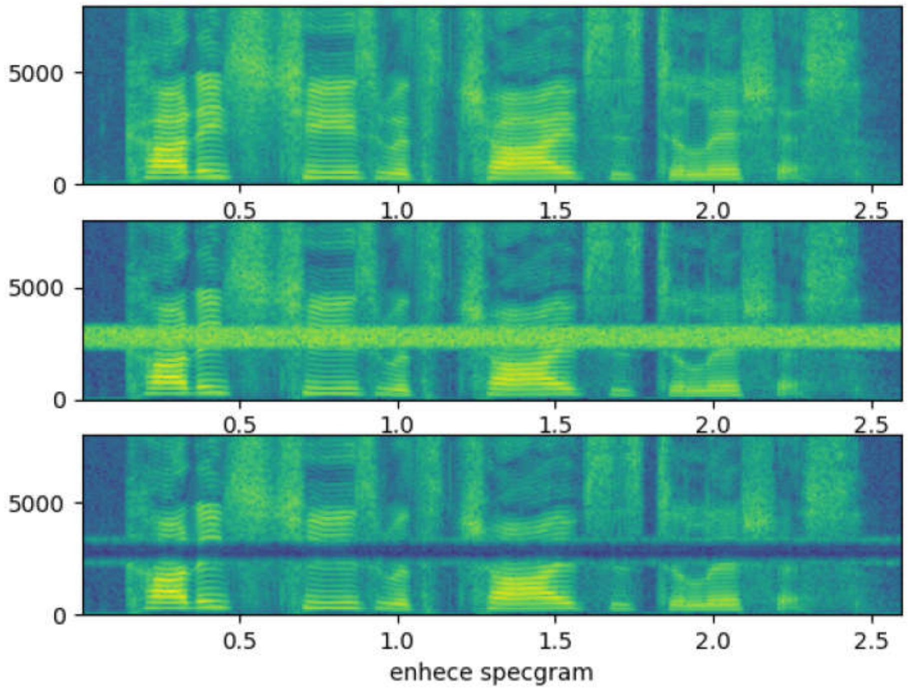
训练滤波器

```
def train_wiener_filter(cleans,noises,para):
    n_fft = para["n_fft"]
    hop_length = para["hop_length"]
    win_length = para["win_length"]
    alpha = para["alpha"]
    beta = para["beta"]
    Pxxs = []
    Pnns =[]
    for clean,noise in zip(cleans,noises):
        S_clean = librosa.stft(clean,n_fft=n_fft, hop_length=hop_length, win_length=win_length)
        S_noise = librosa.stft(noise,n_fft=n_fft, hop_length=hop_length, win_length=win_length)
        Pxx = np.mean((np.abs(S_clean))**2,axis=1,keepdims=True) # Dx1
        Pnn = np.mean((np.abs(S_noise))**2,axis=1,keepdims=True)
        Pxxs.append(Pxx)
        Pnns.append(Pnn)

    train_Pxx = np.mean(np.concatenate(Pxxs,axis=1),axis=1,keepdims=True)
    train_Pnn = np.mean(np.concatenate(Pnns,axis=1),axis=1,keepdims=True)

    H = (train_Pxx/(train_Pxx+alpha*train_Pnn))**beta

    return H
```

```python
if __name__ == "__main__":

    files = ["sf2_cln.wav","sf3_cln.wav","sm1_cln.wav","sm2_cln.wav","sm3_cln.wav"]
    cleans =[]
    noises= []

    for file in files:
        print(file)
        # 读取干净语音
        clean,fs = librosa.load(file,sr=None)
        # 生成噪声
        noise = gen_color_noise(len(clean),128,fs,2400,3200)
        # 添加噪声
        noisy = add_nosie(clean,noise,5)
        cleans.append(clean)
        noises.append(noisy-clean)

    # 设置维纳滤波模型参数
    para_wiener = {}
    para_wiener["n_fft"] = 256
    para_wiener["hop_length"] = 128
    para_wiener["win_length"] = 256
    para_wiener["alpha"] = 1
    para_wiener["beta"] =3

    # 训练维纳滤波器
    H= train_wiener_filter(cleans,noises,para_wiener)
```

训练滤波器

```python
# 测试语音
clean_wav_file = "sf1_cln.wav"
test_clean,fs = librosa.load(clean_wav_file,sr=None)
test_noise = gen_color_noise(len(test_clean),128,fs,2400,3200)
test_noisy = add_nosie(test_clean,test_noise,5)
sf.write("test_noisy.wav",test_noisy,fs)

# 利用训练的滤波器进行滤波
S_test_noisy = librosa.stft(test_noisy,
                            n_fft=para_wiener["n_fft"],
                            hop_length=para_wiener["hop_length"],
                            win_length=para_wiener["win_length"])
S_test_enhec = S_test_noisy*H
test_enhenc = librosa.istft(S_test_enhec,
                            hop_length=para_wiener["hop_length"],
                            win_length=para_wiener["win_length"])

sf.write("enhce_3.wav",test_enhenc,fs)


plt.subplot(3,1,1)
plt.specgram(test_clean,NFFT=256,Fs=fs)
plt.xlabel("clean specgram")
plt.subplot(3,1,2)
plt.specgram(test_noisy,NFFT=256,Fs=fs)
plt.xlabel("noisy specgram")
plt.subplot(3,1,3)
plt.specgram(test_enhenc,NFFT=256,Fs=fs)
plt.xlabel("enhece specgram")
plt.show()
```



noisy enhance

局限和改进：

（1） $H(\omega_k)=\dfrac{P_{yx}^{\ *}(\omega_k)}{P_{yy}(\omega_k)}=\dfrac{P_{xx}(\omega_k)}{P_{xx}(\omega_k)+P_{nn}(\omega_k)}$ 时域滤波，物理不可实现

（2）噪声谱固定

（3） 滤波器参数固定