

阵列语音信号处理+人工智能语音信号处理

——author：李鸿基

阵列语音信号处理+人工智能语音信号处理

1 实验模型假设

1.1 协方差矩阵

1.2 阵列单元时间延迟补偿

2 DOA估计

2.1 多声源信号区分-music算法

2.2 宽带信号DOA估计

2.2.1 理论基础

2.2.2 宽带ISM算法代码实现

3 波束形成——基于空域滤波的语音增强

3.1 目标

本文对波束形成进行空域滤波语音增强验证的算法有：

1. 基于时延的直接波束形成

2. 最小方差无畸变响应 (MVDR)

3. 自适应波束形成算法中的广义旁瓣相消的波束形成算法 (GSC)

3.2 延迟求和波束形成 (time delay beamforming)

3.2.1 理论基础

3.2.2 Time delay beam forming的代码实现

3.2.3 子带相移波束形成器使用与效果方向图可视化

3.4 最小方差无失真响应(MVDR)

3.4.1 理论基础

3.4.2 MVDR代码实现

3.5 自适应波束形成算法

3.6 广义旁瓣相消器 (GSC) 的波束形成算法及其改进

3.6.1 广义旁瓣相消器在书中的理论基础

3.6.2 matlab代码进行GSC波束形成

3.8 实时声源定位和波束形成

4. 整体实验验证 (代码及结果展示python版本)

Steered – Response Power with Phase Transform

Multiple Signal Classification

Delay – and – Sum Beamforming

5. 语音信号分离

6. 语音识别算法模块

7. 自监督学习的语音信号特征提取

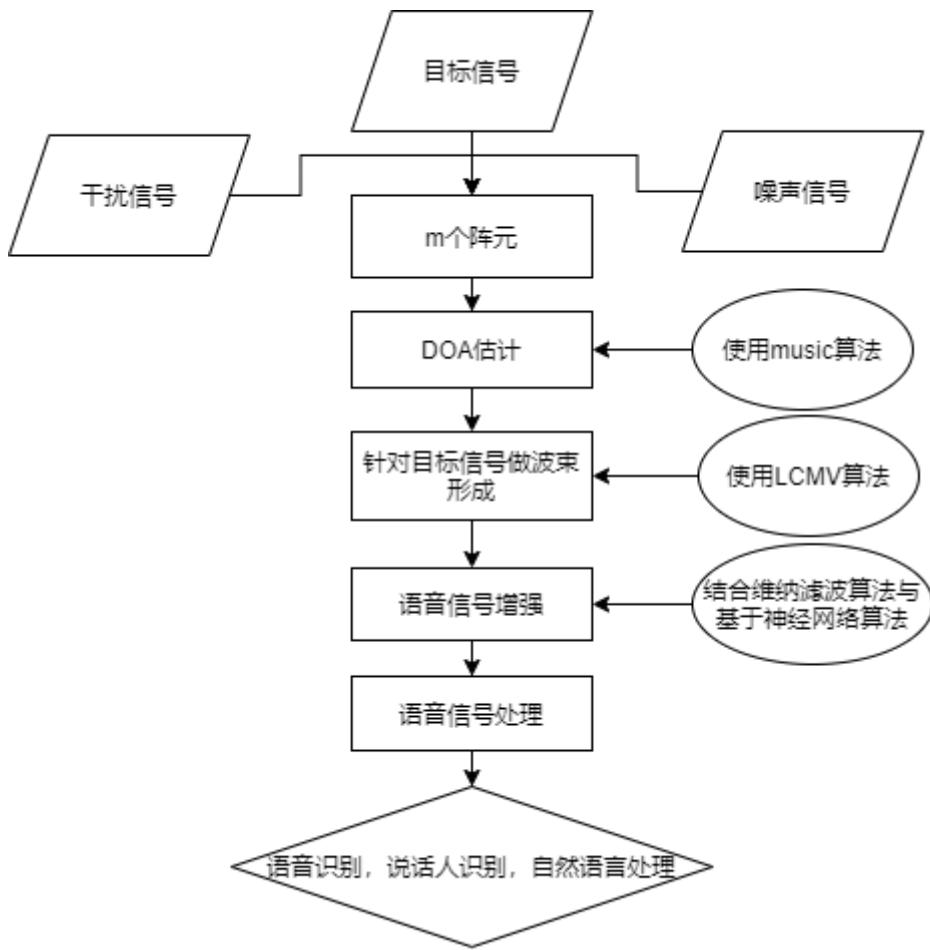
7.1 自监督学习的核心思想

7.3 CPC——语音方向自监督学习的奠基之作

7.4 wav2vec2.0

wav2vec2.0预训练特征提取模型结合CTC模型进行实时语音检测

#思考：是不是可以用Conditional的GAN进行beamforming的生成，直接跳过DOA。



麦克风阵列的优势在于，其可以在信号分辨率以内，利用DOA算法准确估计声源方向，针对已确定的声源方向，采用波束形成算法专注于“收听”该方向的语音信号并抑制周围干扰和噪声信号实现语音信号增强。我们基于阵列信号处理，首先根据m个阵元收集到的信号进行后续的信号处理。

麦克风阵列模型图如下所示：



1 实验模型假设

我们假设有如下模型：

$$x_m[n] = h_m[n] * s[n] + b_m[n]$$

其中， m 表示麦克风的阵元数量， n 表示信号采样点位， h_m 表示空间响应函数， $s[n]$ 表示信号源的语音信号， $b_m[n]$ 表示加性噪声信号， $x_m[n]$ 表示第 m 个麦克风在 n 时刻接收到的信号，以上模型通过STFT后可在频域同样表述为：

$$X_m(t, jw) = H_m(jw)S(t, jw) + B_m(t, jw)$$

或者以向量的形式：

$$\mathbf{X}(t, jw) = \mathbf{H}(jw)S(t, jw) + \mathbf{B}(t, jw)$$

其中 $\mathbf{X}(t, jw)$ 是一个M维的复数向量

在该假设模型中，我们假设 $h_m[n] = a_m[n] = \delta(n - \tau_m)$ ，我们同样将其写成频域的形式
 $H_m(jw) = A_m(jw) = e^{-jw\tau_m}$, 其中 τ_m 在方向向量中表示相对于参考阵元第 m 个阵元的时间延迟补偿项， $A_m(jw)$ 是一个M维的复数向量。

1.1 协方差矩阵

我们会利用到如下的协方差矩阵来进行波束形成：

$$\begin{aligned}\mathbf{R}_{XX}(jw) &= \frac{1}{T} \sum_{i=1}^T \mathbf{X}(t, jw)\mathbf{X}^H(t, jw) \\ \mathbf{R}_{SS}(jw) &= \frac{1}{T} \sum_{i=1}^T \mathbf{H}(jw)\mathbf{H}^H(jw)|S(t, jw)|^2 2 \\ \mathbf{R}_{NN}(jw) &= \frac{1}{T} \sum_{i=1}^T \mathbf{B}(t, jw)\mathbf{B}^H(t, jw)\end{aligned}$$

1.2 阵列单元时间延迟补偿

麦克风 1 和 m 之间的到达时间差可以使用具有相位变换的广义互相关($GCC - PHAT$) 来估计，其表达式如下：

$$\tau_m = argmax_{\tau} \int_{-\pi}^{+\pi} \frac{X_1(jw)X_m(jw)^*}{|X_1(jw)||X_m(jw)|} e^{jw\tau} dw$$

2 DOA估计

***我们需要另外明确的是一般除了基于时延估计以外的波束形成算法，大部分的算法在理论层面都是由窄带信号推导过来的，比如MUSIC，并不直接适用于语音信号这种宽带的信号，我们需要使用在频域分成子带的方式才可以使用这部分算法，比如MUSIC的变体subband-MUSIC。

$SRP - PHAT$ 在阵列周围的虚拟单位球面上扫描每个潜在的到达方向并计算相应的功率。对于每个DOA (由单位向量 \mathbf{u} 表示)，在 \mathbf{u} 的方向上有一个导向向量 $A(jw, u)$ 是一个针对方向 \mathbf{u} 的 M 维的复数向量：

$$E(\mathbf{u}) = \sum_{p=1}^M \sum_{q=p+1}^M \int_{-\pi}^{+\pi} \frac{X_p(j\omega)X_q(j\omega)^*}{|X_p(j\omega)||X_q(j\omega)|} A_p(j\omega, \mathbf{u})A_q(j\omega, \mathbf{u})^* d\omega$$

DOA由 $E(u)$ 的最大值来估计：

$$\mathbf{u}_{max} = argmax_{\mathbf{u}} E(\mathbf{u})$$

2.1 多声源信号区分-music算法

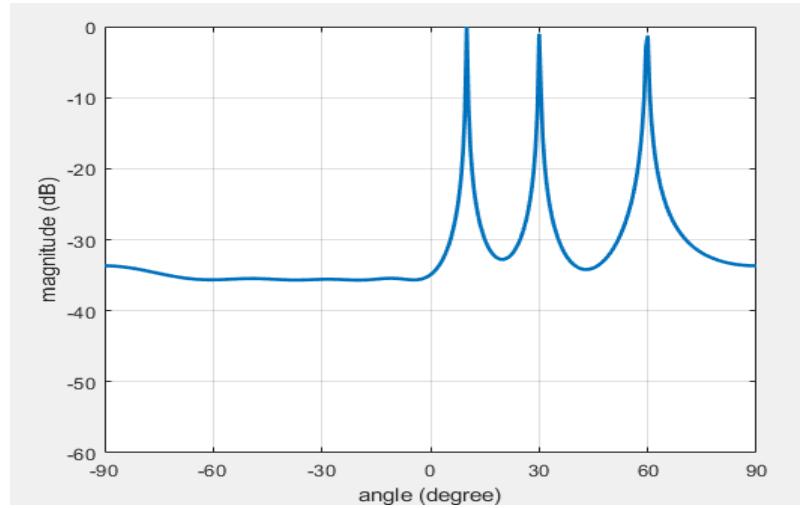
MUSIC 在阵列周围的虚拟单位球面上扫描每个潜在的到达方向并计算相应的功率。对于每个 DOA (由单位向量 \mathbf{u} 表示) , 每一个 \mathbf{u} 方向上都有一个导向向量 $\mathbf{A}(j\omega, \mathbf{u}) \in \mathbb{C}^{M \times 1}$ 。矩阵 $\mathbf{U}(j\omega) \in \mathbb{C}^{M \times S}$ 包括了 $\mathbf{R}_{XX}(j\omega)$ 经过特征值分解后最小的 S 个特征向量。功率谱函数如下所示:

$$E(\mathbf{u}) = \frac{\mathbf{A}(j\omega, \mathbf{u})^H \mathbf{A}(j\omega, \mathbf{u})}{\sqrt{\mathbf{A}(j\omega, \mathbf{u})^H \mathbf{U}(j\omega) \mathbf{U}(j\omega)^H \mathbf{A}(j\omega, \mathbf{u})}}$$

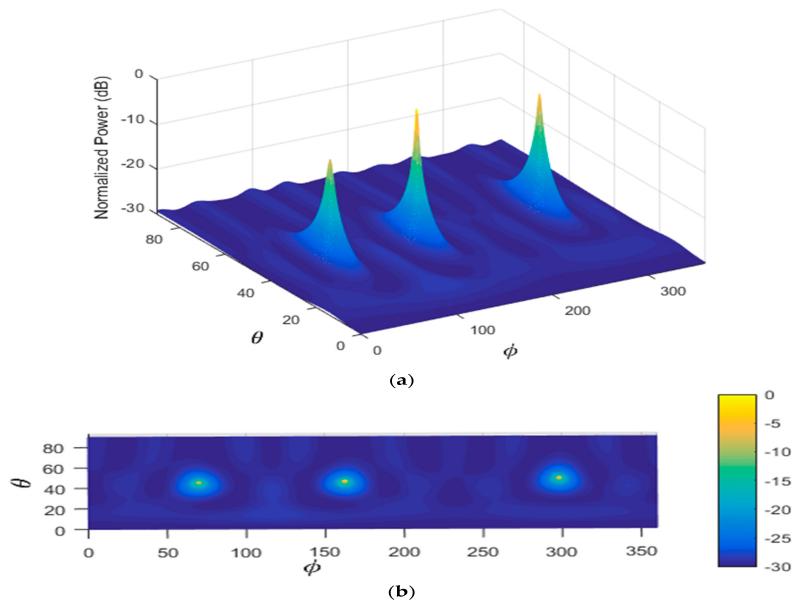
我们找到该功率谱最大值即为所需方向 \mathbf{u}

$$\mathbf{u}_{max} = argmax_{\mathbf{u}} E(\mathbf{u})$$

一维空间music谱估计:



二维空间角music算法DOA估计:



2.2 宽带信号DOA估计

2.2.1 理论基础

6.2.1 宽带信号的概念

宽带信号是指与其中心频率相比有很大带宽的信号。宽带与窄带是相对的，不满足窄带信号条件的信号可视为宽带信号，设信号带宽为 B ，时宽为 T ，中心频率为 f_0 ，首先给出窄信号的定义如下。

定义 6.1 $B \ll f_0$ ，即相对带宽 $\frac{B}{f_0} \ll 1$ ，一般窄带信号 $\frac{B}{f_0} < 0.1$ 。

定义 6.2 $\frac{2v}{c} \ll \frac{1}{TB}$ ，其中， v 是阵列与目标的相对径向运动速度， c 是信号在介质中的传播速度。

定义 6.3 $\frac{(M-1)d}{c} \ll \frac{1}{B}$ ，其中， M 是阵元数目， d 是阵元间距。

“定义 6.1”是对窄带信号的直观理解，同时也是窄带信号有效表示为其复解析形式的充要条件；很多文献均以该定义来区分信号是宽带信号还是窄带信号。“定义 6.2”是指在存在相对运动的系统中，在信号的持续时间 T 内，相对于信号的距离分辨率，目标若没有明显的位移，此时信号可视为窄带信号，否则信号就是宽带信号。“定义 6.3”是指在阵列信号处理中，如果信号带宽的倒数远远大于信号入射阵列孔径的最大传播时间，就称为窄带信号，否则为宽带信号。

从数学角度讲，复信号表示可以简化运算。一般窄带实信号 $f(t)$ 可用它的复解析形式

(预包络) $f_c(t)$ 来表示，即

$$\begin{aligned} f(t) &= \operatorname{Re}\{f_c(t)\} = \operatorname{Re}\{v(t)\exp(j[2\pi ft + \theta(t)])\} \\ &= \operatorname{Re}\{u(t)\exp(j2\pi ft)\} \end{aligned} \tag{6.2.1}$$

其中， $\operatorname{Re}\{\cdot\}$ 表示取其实部， $u(t) = v(t)\exp(j\theta(t))$ 为 $f(t)$ 的复包络， f 为载频。

6.2.2 阵列信号模型

阵列信号模型的假设条件如下所述。

- (1) 接收的目标信号为宽带信号，阵元位于信号源的远场，可近似认为接收到的信号为平面波；
- (2) 传播介质是无损的、线性的、非扩散性的、均匀而且各向同性的；
- (3) 阵元的几何尺寸远小于入射平面波的波长，而且阵元无指向性，可近似认为接收阵元是点元，空间增益为1；
- (4) 接收基阵的阵元间距远大于阵元尺寸，各阵元间的相互影响可以忽略不计；
- (5) 噪声为高斯噪声，且噪声和信号不相关。

虽然电磁波从点辐射源是以球面波向外传播的，但是只要离辐射源足够远（即远场），在接受的局部区域，球面波就可以近似为平面波。对传输介质的要求主要是为了使介质对传输信号的影响简化为与信号源和传感器阵列之间距离成比例的时间延迟。

由于信号到达各个阵元的时间差异，同一平面波在各阵元输出端的响应有不同的延迟时间。假设有 P 个宽带信号源分别从不同的方向辐射到 M 元宽带传感器阵列 ($P < M$)，则第 m 个传感器上接收的信号可表示为

$$x_m(t) = \sum_{p=1}^P s_p(t - \tau_m(\theta_p)) + n_m(t) \quad (6.2.2)$$

其中， $s_p(t)$ 是第 p 个信号源， $\theta_p(t)$ 是第 p 个信号源的到达方向， $\tau_m(\theta_p)$ 是第 p 个信号源到达第 m 个传感器相对于阵列参考阵元的时间延迟， $n_m(t)$ 是第 m 个阵元加性噪声。若是窄带信号，则可以用相移代替时延，式 (6.2.2) 可改写为

$$x_m(t) = \sum_{p=1}^P \exp(-j2\pi f \tau_m(\theta_p)) s_p(t) + n_m(t) = \sum_{p=1}^P a_m(\theta_p) s_p(t) + n_m(t) \quad (6.2.3)$$

令 $\mathbf{X} = [x_1(t), \dots, x_M(t)]^\top$, $\mathbf{N} = [n_1(t), \dots, n_M(t)]^\top$, $\mathbf{S} = [s_1(t), \dots, s_P(t)]^\top$, $\mathbf{A} = [\mathbf{a}(\theta_1), \dots, \mathbf{a}(\theta_p)]$ 。

采集多个快拍，于是式 (6.2.3) 的矩阵表示为

$$\mathbf{X} = \mathbf{AS} + \mathbf{N} \quad (6.2.4)$$

以上是阵列输出的窄带模型，是窄带阵列信号处理的基础。

由于宽带信号的方向向量与频率有关，因此在时域阵列接收数据无法表示为矩阵矢量表达式，故用频域模型来描述，取式 (6.2.2) 的傅里叶变换，可得

$$X_m(f_j) = \sum_{p=1}^P a_m(f_j, \theta_p) s_p(f_j) + N_m(f_j), j = 1, 2, \dots, J \quad (6.2.5)$$

令 $\mathbf{S}(f_j) = [s_1(f_j), \dots, s_p(f_j)]^\top$, $\mathbf{N}(f_j) = [N_1(f_j), \dots, N_M(f_j)]^\top$, $\mathbf{A}(f_j) = [\mathbf{a}(f_j, \theta_1), \dots, \mathbf{a}(f_j, \theta_p)]$, $\mathbf{a}(f_j, \theta_p) = [a_1(f_j, \theta_1), \dots, a_M(f_j, \theta_p)]^\top$ 。则有

$$\mathbf{X}(f_j) = \mathbf{A}(f_j) \mathbf{S}(f_j) + \mathbf{N}(f_j), j = 1, 2, \dots, J \quad (6.2.6)$$

上式即为阵列输出的宽带信号频域模型，可以看出其与窄带的时域模型很相似。

6.3 宽带信号源的 DOA 估计

6.3.1 非相干信号子空间 (ISM) 方法

非相干信号子空间方法 (ISM)^[1]是出现最早的宽带 DOA 估计算法，该方法首先将宽带信号在频域分解为 J 个窄带分量，然后在每个子带上直接进行窄带处理，即对每个子带的谱密度矩阵进行特征分解，根据信号子空间和噪声子空间的正交性构造空间谱，对所有子带的空间谱进行平均，最后得到宽带信号空间谱估计。为了估计各个窄带上的谱密度矩阵，需要把时域观测信号转换到频域。首先把观测时间 T_0 内采集的信号分成多段，再对每段信号进行 DFT 得到 J 组互不相关的窄带频域分量， K 为频域快拍，由此可以得到 K 个快拍，记为 $\mathbf{X}_k(f_j)$ ($k=1, 2, \dots, K$; $j=1, 2, \dots, J$)。ISM 算法的目的就是由这 K 个频域快拍估计多个目标的方位。

于是频率上的互谱密度为

$$\mathbf{R}_X(f_i) = \frac{1}{K} \sum_{k=1}^K \mathbf{X}_k(f_i) \mathbf{X}_k^H(f_i), \quad 1 \leq j \leq J \quad (6.3.1)$$

对 $\mathbf{R}_X(f_i)$ 进行特征值分解，有

$$\mathbf{R}_X(f_i) = \sum_{i=1}^M \lambda_i \boldsymbol{\mu}_i \boldsymbol{\mu}_i^H \quad (6.3.2)$$

其中，特征值 $\lambda_i > \sigma^2$ ($i=1, 2, \dots, P$) 对应的特征向量构成信号子空间 $\mathbf{U}_S = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_P]$ ，特征值 $\lambda_i \approx \sigma^2$ ($i=P+1, \dots, M$) 对应的特征向量构成噪声子空间 $\mathbf{U}_n(f_i) = [\mathbf{u}_{P+1}, \mathbf{u}_{P+2}, \dots, \mathbf{u}_M]$ ，则平均意义下 MUSIC 空间谱为

$$P(\theta) = \frac{1}{\frac{1}{J} \sum_{i=1}^J \left\| \mathbf{a}^H(f_i, \theta) \mathbf{U}_n(f_i) \right\|^2} \quad (6.3.3)$$

上面介绍的 ISM 算法将宽带信号在频域分解为 J 个窄带分量，直接对每一窄带利用 MUSIC 算法进行谱估计，但是只能解决非相干源的 DOA 估计问题。在现有的 ISM 算法基础上引入修正 MUSIC 算法，通过对接收数据阵进行去相关运算，使 ISM 算法对相干源的情况同样适用。

6.3.2 相干信号子空间（CSM）方法

相干信号子空间方法利用聚焦矩阵将不同频率的信号子空间映射到同一个参考频率上，然后将所有频率成分的信号功率谱密度矩阵进行平均。聚焦矩阵应满足以下聚焦变换

$$\mathbf{T}(f_j)\mathbf{A}(f_j) = \mathbf{A}(f_0), j = 1, \dots, J \quad (6.3.4)$$

其中， f_j 为带宽内任意频率， f_0 为参考频率，即聚焦频率。聚焦后的阵列输出为

$$\begin{aligned} \mathbf{T}(f_j)\mathbf{X}(f_j) &= \mathbf{T}(f_j)\mathbf{A}(f_j)\mathbf{S}(f_j) + \mathbf{T}(f_j)\mathbf{N}(f_j) \\ &= \mathbf{A}(f_0)\mathbf{S}(f_j) + \mathbf{T}(f_j)\mathbf{N}(f_j) \end{aligned} \quad (6.3.5)$$

由上式可知，聚焦变换后各频率点下的方向矩阵所包含的频率信息相等。因此，可对聚焦后阵列各频率点下的协方差矩阵求和平均得到

$$\begin{aligned} \mathbf{R}_y &= \frac{1}{J} \sum_{j=1}^J \mathbf{T}(f_j)\mathbf{X}(f_j)\mathbf{X}^H(f_j)\mathbf{T}^H(f_j) \\ &= \mathbf{A}(f_0) \left[\frac{1}{J} \sum_{j=1}^J \mathbf{P}_s(f_j) \right] \mathbf{A}^H(f_0) + \frac{1}{J} \sum_{j=1}^J \mathbf{T}(f_j)\mathbf{P}_n(f_j)\mathbf{T}^H(f_j) \end{aligned} \quad (6.3.6)$$

其中， $\mathbf{P}_s(f_j) = \mathbf{S}(f_j)\mathbf{S}^H(f_j)$, $\mathbf{P}_n(f_j) = \mathbf{N}(f_j)\mathbf{N}^H(f_j)$ 。定义 $\mathbf{R}_s = \sum_{j=1}^J \mathbf{P}_s(f_j)$, $\mathbf{R}_n = \sum_{j=1}^J \mathbf{T}(f_j)\mathbf{P}_n(f_j)\mathbf{T}^H(f_j)$ 。

对矩阵束 $(\mathbf{R}_y, \mathbf{R}_n)$ 进行广义特征分解得到特征值 λ_i (按降序排列) 和对应的特征向量 \mathbf{u}_i , $i = 1, 2, \dots, M$ 。定义 $\mathbf{U}_s = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_P]$ 和 $\mathbf{U}_n = [\mathbf{u}_{P+1}, \mathbf{u}_{P+2}, \dots, \mathbf{u}_M]$ 的列向量张成的空间分别为信号子空间和噪声子空间，则有 $\mathbf{A}^H(f_0)\mathbf{U}_n = 0$ 。由以上结论可以知道，信号子空间与噪声子空间相互正交，且包含了信号源数目以及到达角度的所有信息。因此，可以得到特征子空间类方法的空间谱

$$P(\theta) = \frac{1}{\|\mathbf{a}(f_0, \theta)\mathbf{U}_n\|^2} \quad (6.3.7)$$

子空间方法的步骤可以归纳为如下：(1) 阵列接收的数据分段进行 DFT；(2) 初步估计信号到达角度；(3) 构造聚焦矩阵 $\mathbf{T}(f_j)$ ；(4) 计算聚焦平均后 \mathbf{R}_y 和 \mathbf{R}_n ，形成相干信号子空间和相干噪声子空间；(5) 利用高分辨的方法得到信号的 DOA 估计。

2.2.2 宽带ISM算法代码实现

```

clear all ;close all;
M=12; %阵元数
N=200;%快拍数
ts=0.01;%时域采样间隔

f0=100; %入射信号中心频率
f1=80; %入射信号最低频率
f2=120;%入射信号最高频率

c=1500;
%声速
lambda= c/f0; %波长
d = lambda/2; %阵元间距
SNR = 15; %信噪比
b = pi/180;
theat1=30*b; %入射信号波束角1
theat2=0*b; %入射信号波束角2
n=ts:ts:N*ts;
theat = [theat1 theat2];

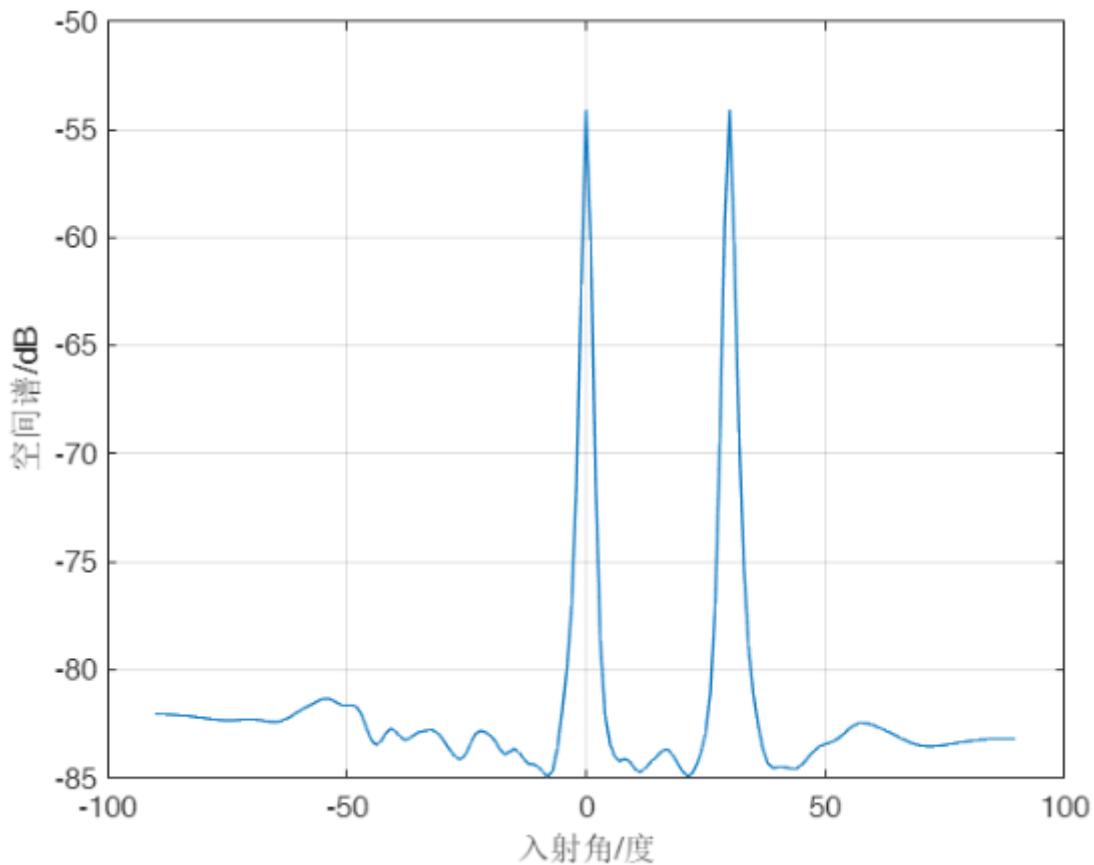
```

```

%%%%%%%%%%%%% produce signal
s1=chirp(n,80,1,120); %生成线性调频信号1
sa=fft(s1,2048); %进行FFT变换
figure, specgram(s1,256,1E3,256,250); %频谱图
s2=chirp(n+0.100,80,1,120); %生成线性调频信号2
sb=fft(s2,2048); %进行 FFT 变换
%%%%%%%%%%%%%ISM 算法
P=1:2;
a=zeros(M,2);
sump = zeros(1,181);
for i=1:N
    f=80+(i-1)*1.0;
    s=[sa(i) sb(i)]';
    for m=1:M
        a(m,P)=exp(-1j*2*pi*f*d/c*sin(theta(P))^(m-1));
    end
    R=a*(s*s')*a';
    [em,zm]=eig(R);
    [zm1,pos1]=max(zm);
    for l=1:2
        [zm2,pos2]=max(zm1);
        zm1(:,pos2)=[];
        em(:,pos2)=[];
    end
    k=1;
    for ii= -90:1:90
        arfa=sin(ii*b)*d/c;
        for iii=1:M
            tao(l,iii)=(iii-1)*arfa;
        end
        A=[exp(-1j*2*pi*f*tao)]';
        p(k)=A'*em*em'*A;
        k=k+1;
    end
    sump=sump+abs(p);
end
pmusic=1/33*sump;
pm=1./pmusic;
thetaesti=-90:1:90;
plot(thetaesti,20*log(abs(pm)));
xlabel('入射角/度');
ylabel('空间谱/dB'); grid on

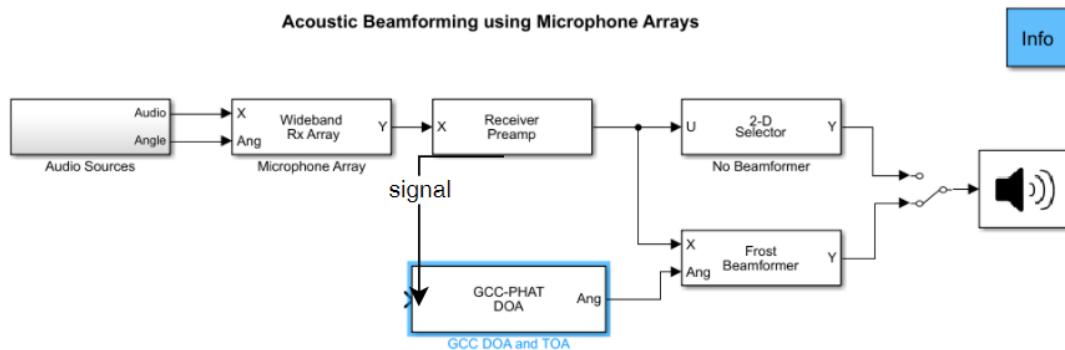
```

声源定位结果图



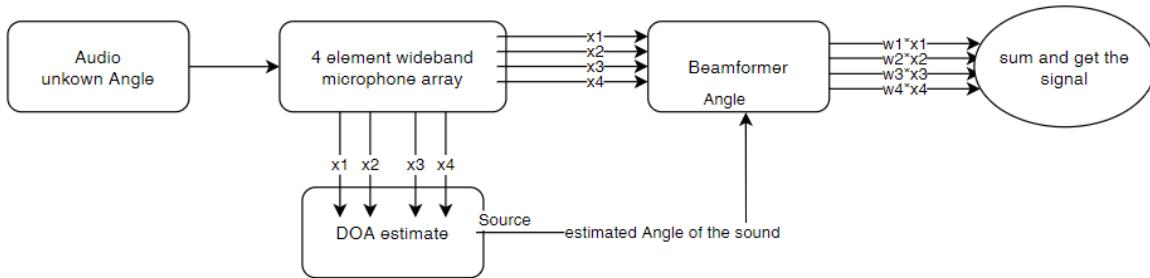
3 波束形成——基于空域滤波的语音增强

3.1 目标



需要在matlab simulink中实现上图。

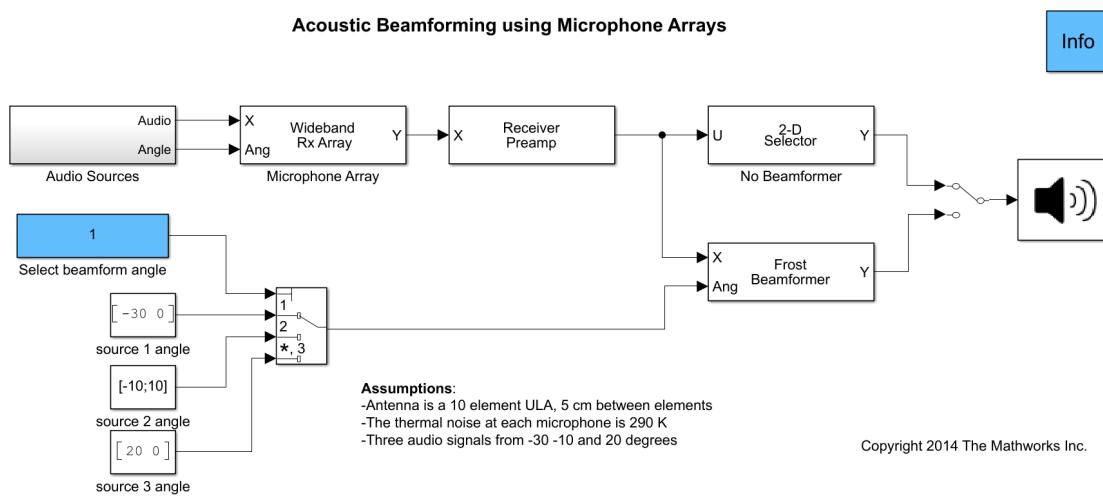
声源定位和波束形成的结合如下图所示，假设有一个四元的麦克风阵列，即有四个语音信号接收通道，接收到语音信号后，四个通道的信号会同时进入DOA估计模块和波束形成模块，DOA估计到声源对应的方位后，将得到的方位导向向量 \mathbf{A} 导入到beam former中作为指导使其生成每个信道对应的权重向量 \mathbf{w}



波束形成进行语音增强的就是给每个语音信道赋予相应的权值，使得最终相加后的语音信号可以实现空域滤波。我们在频域使用波束形成，所有的波束形成算法最后的形式如下，我们的目标就是要找到这个权重矩阵 \mathbf{W} ：

$$Y(j\omega) = \mathbf{W}^H(j\omega)\mathbf{X}(j\omega)$$

在matlab simulink中进行直接简单的波束形成过程如下图：



本文对波束形成进行空域滤波语音增强验证的算法有：

1. 基于时延的直接波束形成
2. 最小方差无畸变响应 (MVDR)
3. 自适应波束形成算法中的广义旁瓣相消的波束形成算法 (GSC)

***我们需要另外明确的是一般除了基于时延估计以外的波束形成算法，大部分的算法在理论层面都是由窄带信号推导过来的，比如MVDR，并不直接适用于语音信号这种宽带的信号，我们需要使用在频域分成子带的方式才可以使用这部分算法，比如MVDR的变体subband-MVDR。

matlab中给出了关于窄带信号和宽带信号的不同的波束形成算法模块，使用时需要注意

Narrowband Beamformers	
LCMV Beamformer	Narrowband linear constraint minimum variance (LCMV) beamformer
MVDR Beamformer	Narrowband MVDR (Capon) beamformer
Phase Shift Beamformer	Narrowband phase-shift beamformer
Wideband Beamformers	
Frost Beamformer	Frost beamformer
GSC Beamformer	Generalized sidelobe canceller
Subband MVDR Beamformer	Subband MVDR (Capon) beamformer
Subband Phase Shift Beamformer	Subband phase shift beamformer
Time Delay Beamformer	Time-delay beamformer
Time Delay LCMV Beamformer	Time delay LCMV beamformer

3.2 延迟求和波束形成 (time delay beamforming)

3.2.1 理论基础

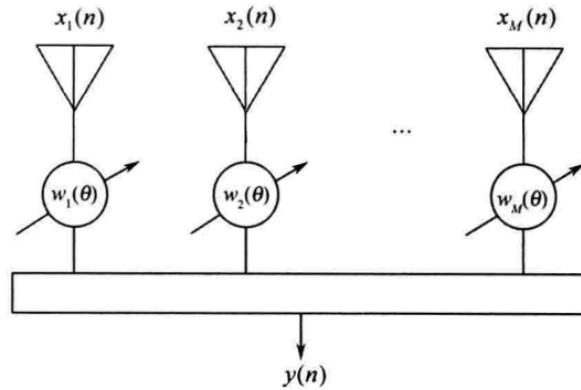


图 3.2.1 波束形成算法结构

这时阵列的输出可表示为

$$y(t) = \sum_{i=1}^M w_i^*(\theta)x_i(t) \quad (3.2.1)$$

如果采用向量来表示各阵元输出及加权系数，则有

$$\mathbf{x}(t) = [x_1(t) \ x_2(t) \ \cdots \ x_M(t)]^T, \mathbf{w}(\theta) = [w_1(\theta) \ w_2(\theta) \ \cdots \ w_M(\theta)]^T \quad (3.2.2)$$

那么，阵列的输出也可用向量表示

$$y(t) = \mathbf{w}^H(\theta)\mathbf{x}(t) \quad (3.2.3)$$

为了在某一方向 θ 上补偿各阵元之间的时延以形成一个主瓣，常规波束形成器在期望方向上的加权向量可以构成为

$$\mathbf{w}(\theta) = [1 \ e^{-j\omega\tau} \ \cdots \ e^{-j(M-1)\omega\tau}]^T \quad (3.2.4)$$

观察此加权向量，若空间只有一个来自方向 θ 的信号，其方向向量 $\mathbf{a}(\theta)$ 的表示形式跟此权向量一样。则有

$$y(t) = \mathbf{w}^H(\theta)\mathbf{x}(t) = \mathbf{a}^H(\theta)\mathbf{x}(t) \quad (3.2.5)$$

这时常规波束形成器的输出功率可以表示为

$$P_{CBF}(\theta) = E[y(t)^2] = \mathbf{w}^H(\theta)\mathbf{R}\mathbf{w}(\theta) = \mathbf{a}^H(\theta)\mathbf{R}\mathbf{a}(\theta) \quad (3.2.6)$$

其中，矩阵 \mathbf{R} 为阵列输出 $\mathbf{x}(t)$ 的协方差矩阵，即 $\mathbf{R} = E[\mathbf{x}(t)\mathbf{x}^H(t)]$ 。

下面来分析一下常规波束形成法的角分辨率问题。一般来说，当空间有两个同频信号投射到阵列，如果它们的空间方位角的间隔小于阵列主瓣波束宽度，这时不仅无法分辨它们而且还会严重影响系统的正常工作，即对于阵列远场中的两个点信号源，仅当它们之间的角度分离大于阵元间隔（或称阵列孔径）的倒数时，它们方可被分辨开，这就是瑞利准则。瑞利准则说明常规波束形成法固有的缺点就是角分辨率低，如果要设法提高角分辨率，就要增加阵元间隔或增加阵元个数。这在系统施工上是难以实现的。

3.2.2 Time delay beam forming的代码实现

延迟和求和波束形成器旨在对齐语音信号以产生建设性干扰，选择系数使得：

$$\mathbf{W}(j\omega) = \frac{1}{M} \mathbf{A}(j\omega)$$

式中 \mathbf{A} 为导向向量。

在下述例子中展示了如何用全向麦克风元件的麦克风阵列进行宽带时间延迟波束形成，该例子在matlab中需要利用 phase and array toolbox。

```
% 定义常数
c = 340;
t = linspace(0,1,50e3)';%时间
sig = chirp(t,0,1,1000);%创造一个简单的宽带信号

%% 用一个十个单元的直线麦克风阵列（ULA）收集语音信号。使用全向性的麦克风元件，采样频率为50kHz，单元间距小于一半的波长。语音信号以60度方位角和0度仰角入射到直线阵列上。在信号中加入随机噪声。
microphone = phased.OmnidirectionalMicrophoneElement(...  

    'FrequencyRange',[20 20e3]);
array = phased.ULA('Element',microphone,'NumElements',10,...  

    'ElementSpacing',0.01);
collector = phased.widebandCollector('Sensor',array,'SampleRate',5e4,...  

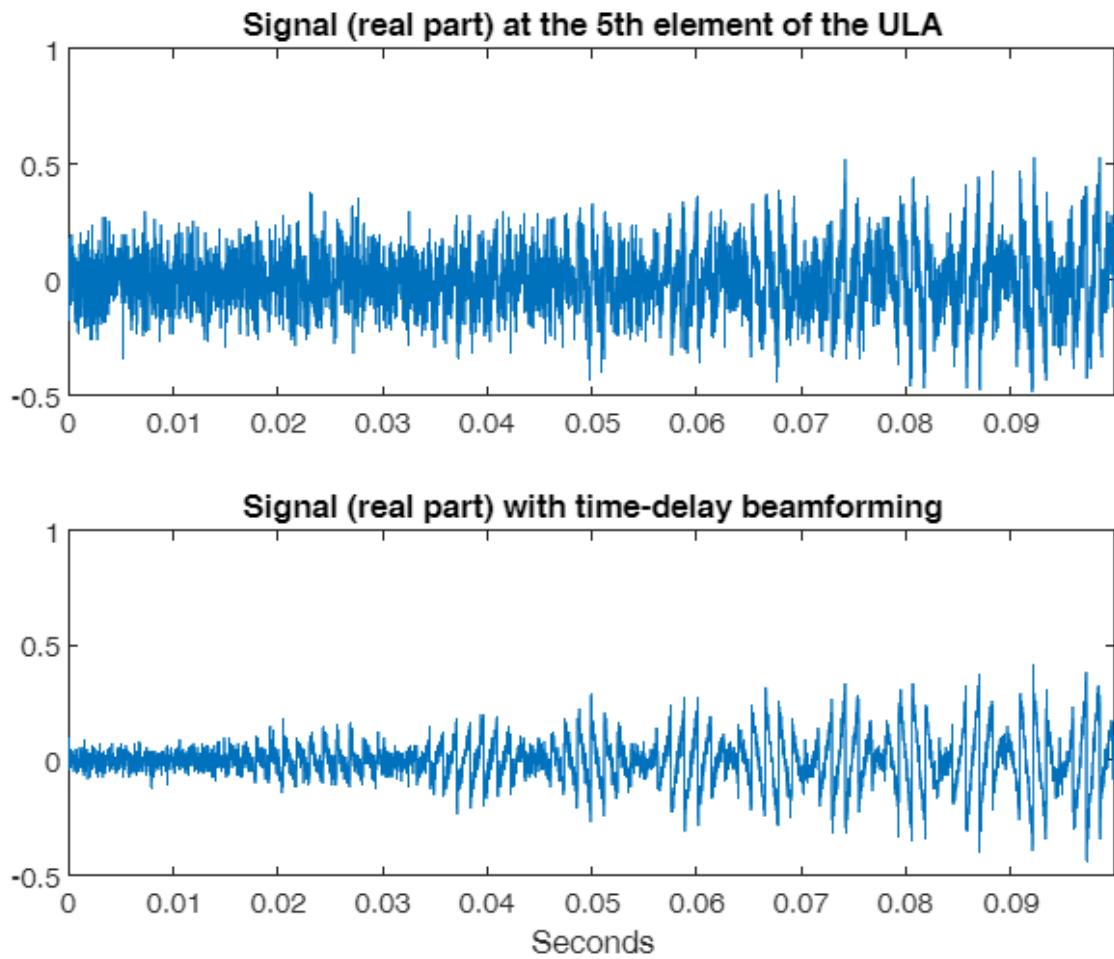
    'PropagationSpeed',c,'ModulatedInput',false);
sigang = [60;0];
rsig = collector(sig,sigang);
rsig = rsig + 0.1*randn(size(rsig));

%% 加载一个传统的时延波束形成器来提升语音信号的SNR
beamformer = phased.TimeDelayBeamformer('SensorArray',array,...  

    'SampleRate',5e4,'PropagationSpeed',c,'Direction',sigang);
y = beamformer(rsig);

subplot(2,1,1)
plot(t(1:5000),real(rsig(1:5e3,5)))
axis([0,t(5000),-0.5,1])
title('signal (real part) at the 5th element of the ULA')
subplot(2,1,2)
plot(t(1:5000),real(y(1:5e3)))
axis([0,t(5000),-0.5,1])
title('Signal (real part) with time-delay beamforming')
xlabel('Seconds')
```

结果图如下，可明显发现加入麦克风阵列的波束形成进行空域滤波后，有明显的语音增强效果。

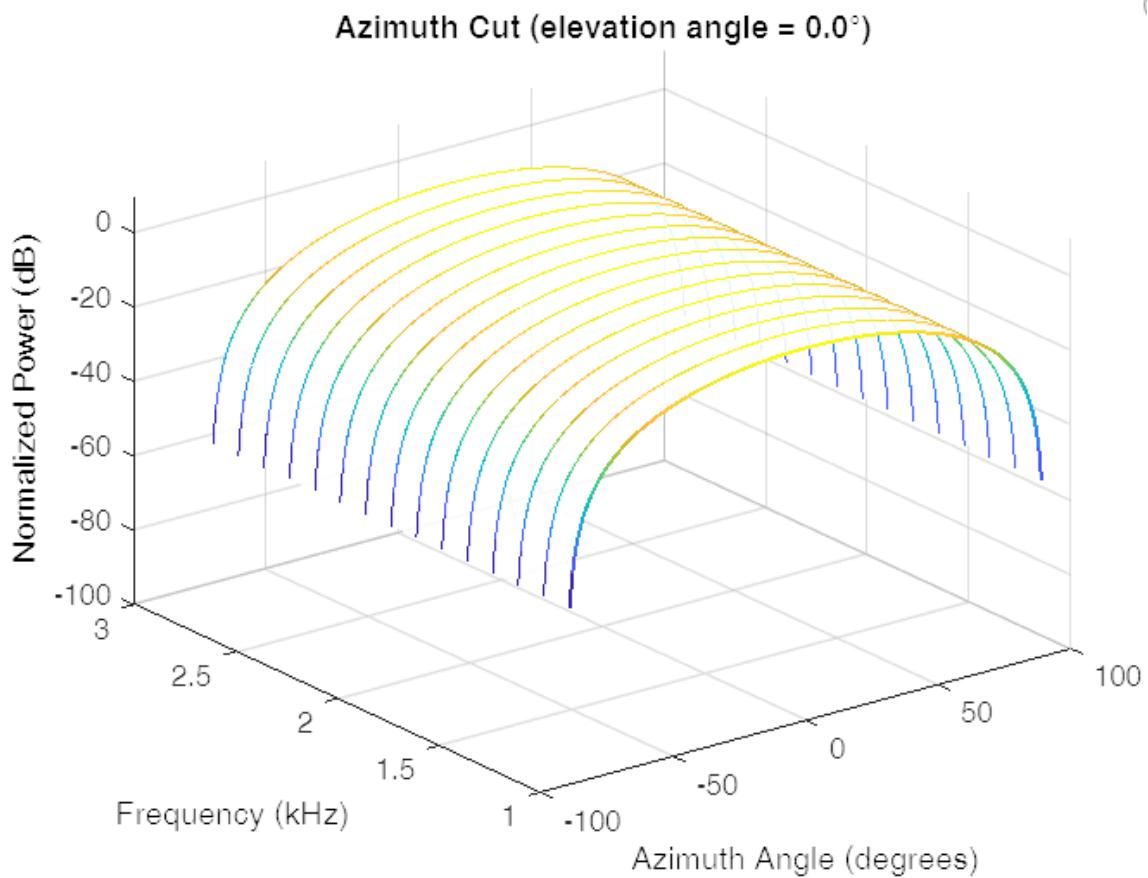


3.2.3 子带相移波束形成器使用与效果方向图可视化

以下例子展示如何可视化语音信号宽带波束形成器。（如前所述，子带方法就是为了解决宽带信号如语音信号本身存在的问题，需要分成子带再应用相移波束形成算法）

```
%% 常数定义
c = 340;
freq = [1000 2750];% 最小频率和最大频率
fc = 2000;
numels = 11;
microphone = phased.CosineAntennaElement('FrequencyRange',freq);%cosine型的麦克风
阵列
array = phased.ULA('NumElements',numels, ...
    'ElementSpacing',0.5*c/fc,'Element',microphone);

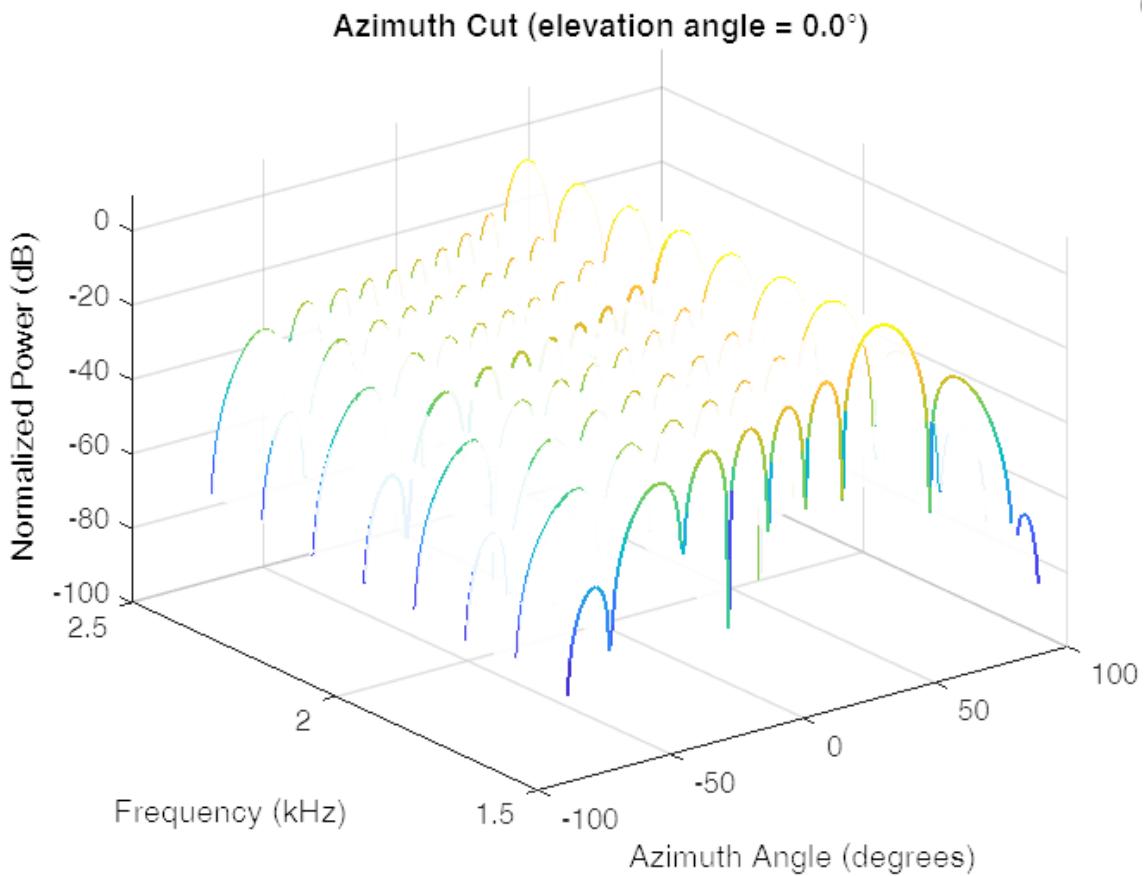
%% 画出一个麦克风单元的频率响应方向图
plotFreq = linspace(min(freq),max(freq),15);
pattern(microphone,plotFreq,[-180:180],0,'CoordinateSystem','rectangular',...
    'PlotStyle','waterfall','Type','powerdb')
```



```

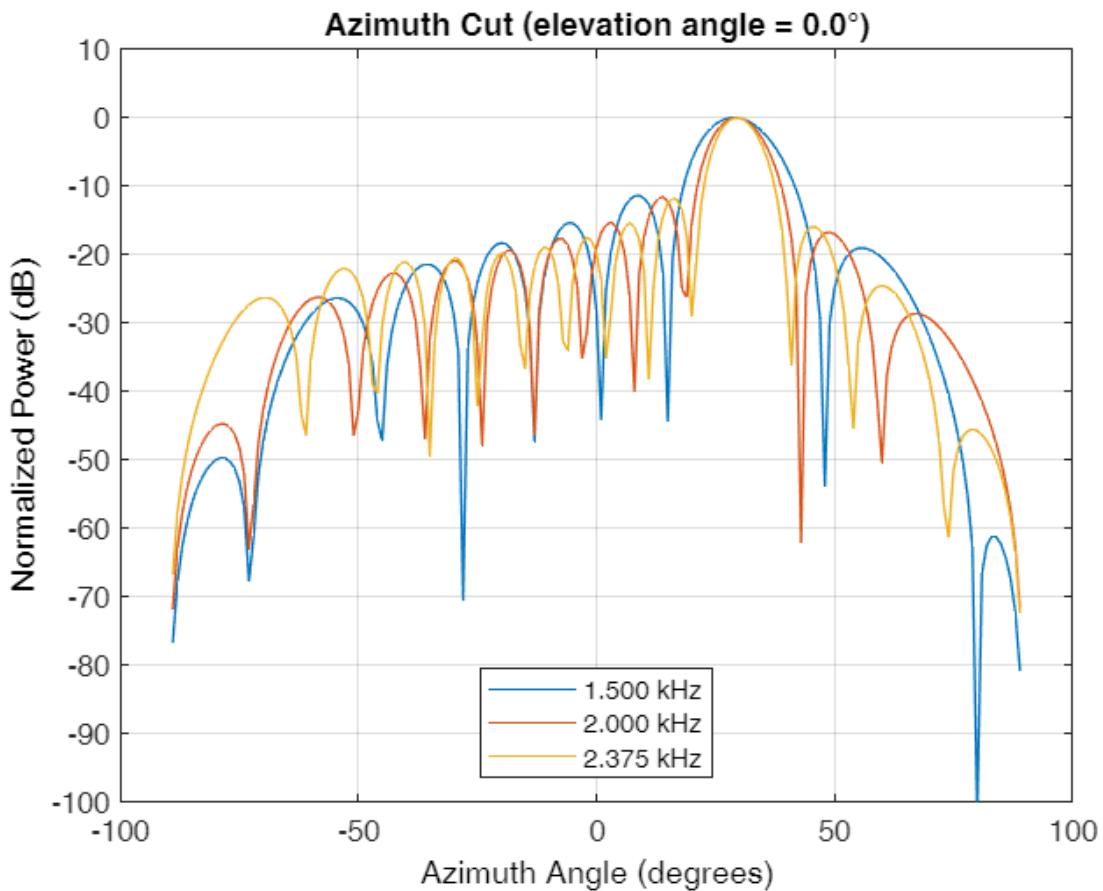
%% 给阵列加载一个子带波束形成器——此处利用了一个子带相移波束形成器
direction = [30;0];
numbands = 8;
beamformer = phased.SubbandPhaseShiftBeamformer('SensorArray',array, ...
    'Direction',direction, ...
    'OperatingFrequency',fc,'PropagationSpeed',c, ...
    'SampleRate',1e3, ...
    'WeightsOutputPort',true,'SubbandsOutputPort',true, ...
    'NumSubbands',numbands);
rx = ones(numbands,numels);
[y,w,centerfreqs] = beamformer(rx);
%% 画出在给每个单元信号通道进行波束形成赋权后，阵列整体的频率响应方向图
pattern(array,centerfreqs. ,
[-180:180],0,'weights',w,'CoordinateSystem','rectangular', ...
'PlotStyle','waterfall','Type','powerdb','PropagationSpeed',c)

```



上图画出每个子带在中心频率处的波束形成方向图，其他的基于子带分解的波束形成方法也可以这样可视化。

```
%%三个不同频率下的麦克分阵列方向图对比图
centerfreqs = fftshift(centerfreqs);
w = fftshift(w,2);
idx = [1,5,8];
pattern(array,centerfreqs(idx).',
[-180:180],0,'weights',w(:,idx),'CoordinateSystem','rectangular',...
'PlotStyle','overlay','Type','powerdb','PropagationSpeed',c)
Legend('Location','South')
```



3.4 最小方差无失真响应(MVDR)

3.4.1 理论基础

3.2.2 波束形成的最佳权向量

上述“导向”作用是通过调整加权系数完成的，阵列对各阵元的接收信号向量 $\mathbf{x}(n)$ 在各阵元上分量的加权和。令权向量为 $\mathbf{w} = [w_1, \dots, w_M]^T$ ，则输出可表示为

$$y(n) = \mathbf{w}^H \mathbf{x}(n) = \sum_{m=1}^M w_m^* x_m(n) \quad (3.2.7)$$

对不同的权向量，上式对来自不同方向的电波便有不同的响应，从而形成不同方向的空间波束。一般用移相器进行加权处理，即只调整信号相位，不改变信号幅度，因为信号在任一瞬间各阵元上的幅度是相同的。不难看出，若空间只有一个来自方向 θ_k 的电波，其方向向量为 $\mathbf{a}(\theta_k)$ ，则当权向量 \mathbf{w} 取作 $\mathbf{a}(\theta_k)$ 时，输出 $y(n) = \mathbf{a}(\theta_k)^H \mathbf{a}(\theta_k) = M$ 最大，实现导相定位作用。这时，各路的加权信号为相干叠加，称这一结果为空域匹配滤波。

匹配滤波在白噪声背景下是最佳的，如果存在干扰信号就要另外进行考虑。下面考虑更复杂情况下的波束形成。假设空间远场有一个感兴趣的信号 $d(t)$ （或称期望信号，其波达方向为 θ_d ）和 J 个不感兴趣的信号 $i_j(t), j = 1, \dots, J$ （或称干扰信号，其波达方向为 θ_{ij} ）。令每个阵元上的加性白噪声为 $n_k(t)$ ，它们都具有相同的方差 σ^2 。在这些假设条件下，第 k 个阵元上的接收信号可以表示为

$$x_k(t) = a_k(\theta_d) d(t) + \sum_{j=1}^J a_k(\theta_{ij}) i_j(t) + n_k(t) \quad (3.2.8)$$

式(3.2.8)中等式右边的三项分别表示信号、干扰和噪声。若用矩阵形式表示，则有

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_M(t) \end{bmatrix} = [\mathbf{a}(\theta_d), \mathbf{a}(\theta_{i_1}), \dots, \mathbf{a}(\theta_{i_J})] \begin{bmatrix} d(t) \\ i_1(t) \\ \vdots \\ i_J(t) \end{bmatrix} + \begin{bmatrix} n_1(t) \\ n_2(t) \\ \vdots \\ n_M(t) \end{bmatrix} \quad (3.2.9)$$

或简记作

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t) = \mathbf{a}(\theta_d)d(t) + \sum_{j=1}^J \mathbf{a}(\theta_{i_j})i_j(t) + \mathbf{n}(t) \quad (3.2.10)$$

式中， $\mathbf{a}(\theta_k) = [a_1(\theta_k), \dots, a_M(\theta_k)]^\top$ ，表示来自波达方向 θ_k ($k = d, i_1, i_2, \dots$)的发射信源的方向向量。 N 个快拍的波束形成器输出 $y(t) = \mathbf{w}^\text{H} \mathbf{x}(t)$ ($t = 1, \dots, N$)的平均功率为

$$\begin{aligned} P(w) &= \frac{1}{N} \sum_{t=1}^N |y(t)|^2 = \frac{1}{N} \sum_{t=1}^N |\mathbf{w}^\text{H} \mathbf{x}(t)|^2 \\ &= |\mathbf{w}^\text{H} \mathbf{a}(\theta_d)|^2 \frac{1}{N} \sum_{t=1}^N |d(t)|^2 + \sum_{j=1}^J \left[\frac{1}{N} \sum_{t=1}^N |i_j(t)|^2 \right] |\mathbf{w}^\text{H} \mathbf{a}(\theta_{i_j})|^2 + \frac{1}{N} \|\mathbf{w}\|^2 \sum_{t=1}^N \|\mathbf{n}(t)\|^2 \end{aligned} \quad (3.2.11)$$

这里忽略了不同用户之间的相互作用项，即交叉项 $i_j(t)i_k^*(t)$ 。当 $N \rightarrow \infty$ 时，式(3.2.11)可写为

$$P(w) = \mathbb{E}[|y(t)|^2] = \mathbf{w}^\text{H} \mathbb{E}[\mathbf{x}(t)\mathbf{x}^\text{H}(t)]\mathbf{w} = \mathbf{w}^\text{H} \mathbf{R} \mathbf{w} \quad (3.2.12)$$

式中， $\mathbf{R} = \mathbb{E}[\mathbf{x}(t)\mathbf{x}^\text{H}(t)]$ 为阵列输出的协方差矩阵。

另一方面，当 $N \rightarrow \infty$ 时，式(3.2.11)可表示为

$$P(w) = \mathbb{E}[|d(t)|^2] |\mathbf{w}^\text{H} \mathbf{a}(\theta_d)|^2 + \sum_{j=1}^J \mathbb{E}[|i_j(t)|^2] |\mathbf{w}^\text{H} \mathbf{a}(\theta_{i_j})|^2 + \sigma_n^2 \|\mathbf{w}\|^2 \quad (3.2.13)$$

在获得上式的过程中，使用了各加性噪声具有相同的方差 σ_n^2 这一假设。

为了保证来自方向 θ_d 期望信号的正确接收，并完全抑制其他 J 个干扰，很容易根据式(3.2.13)得到关于权向量的约束条件

$$\mathbf{w}^\text{H} \mathbf{a}(\theta_d) = 1, \quad \mathbf{w}^\text{H} \mathbf{a}(\theta_{i_j}) = 0 \quad (3.2.14)$$

约束条件式(3.2.14)称为波束“置零条件”，因为它强迫接收阵列波束方向图的“零点”指向所有 J 个干扰信号。在以上两个约束条件下，式(3.2.13)简化为 $P(w) = \mathbb{E}[|d(t)|^2] + \sigma_n^2 \|\mathbf{w}\|^2$ 。

从提高信噪比的角度来看，以上的干扰置零并不是最佳的。这是因为虽然选定的权值可使干扰输出为零，但可能使噪声输出加大。因此，抑制干扰和噪声应一同考虑。这样一来，波束形成器最佳权向量的确定可以叙述为，在约束条件式(3.2.14)的约束下，求满足式(3.2.15)的权向量 \mathbf{w} ，有

$$\min_w \mathbb{E}[|y(t)|^2] = \min_w \{\mathbf{w}^\text{H} \hat{\mathbf{R}} \mathbf{w}\} \quad (3.2.15)$$

这个问题很容易用Lagrange乘子法求解。令目标函数为

$$L(\mathbf{w}) = \mathbf{w}^H \hat{\mathbf{R}} \mathbf{w} + \lambda [\mathbf{w}^H \mathbf{a}(\theta_d) - 1] \quad (3.2.16)$$

根据线性代数的有关知识，函数 $f(\mathbf{w})$ 对复向量 $\mathbf{w} = [w_0, w_1, \dots, w_{M-1}]^T$ ($w_i = a_i + jb_i$) 的偏导数定义为

$$\frac{\partial}{\partial \mathbf{w}} f(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial a_0} f(\mathbf{w}) \\ \vdots \\ \frac{\partial}{\partial a_{M-1}} f(\mathbf{w}) \end{bmatrix} + j \begin{bmatrix} \frac{\partial}{\partial b_0} f(\mathbf{w}) \\ \vdots \\ \frac{\partial}{\partial b_{M-1}} f(\mathbf{w}) \end{bmatrix} \quad (3.2.17)$$

利用这一定义，可以得到

$$\frac{\partial(\mathbf{w}^H \mathbf{A} \mathbf{w})}{\partial \mathbf{w}} = 2 \mathbf{A} \mathbf{w}, \quad \frac{\partial(\mathbf{w}^H \mathbf{c})}{\partial \mathbf{w}} = \mathbf{c} \quad (3.2.18)$$

由式 (3.2.16) 和式 (3.2.18) 易知， $\partial L(\mathbf{w}) / \partial \mathbf{w} = 0$ 的结果为 $2 \mathbf{R} \mathbf{w} + \lambda \mathbf{a}(\theta_d) = 0$ ，得到接收来自方向 θ_d 的期望信号的波束形成器的最佳权向量为

$$\mathbf{w}_{\text{opt}} = \mu \mathbf{R}^{-1} \mathbf{a}(\theta_d) \quad (3.2.19a)$$

式中， μ 为一比例常数； θ_d 是期望信号的波达方向。这样，就可以决定 $J+1$ 个发射信号的波束形成的最佳权向量。此时，波束形成器将只接收来自方向 θ_d 的信号，并抑制所有来自其他波达方向的信号。

注意到约束条件 $\mathbf{w}^H \mathbf{a}(\theta_d) = 1$ 也可等价写作 $\mathbf{a}^H(\theta_d) \mathbf{w} = 1$ ，则式 (3.2.19a) 两边同乘 $\mathbf{a}^H(\theta_d)$ ，并与等价的约束条件比较，可得式 (3.2.19a) 中的常数 μ 应满足

$$\mu = \frac{1}{\mathbf{a}^H(\theta_d) \mathbf{R}^{-1} \mathbf{a}(\theta_d)} \quad (3.2.19b)$$

3.4.2 MVDR代码实现

MVDR方法有如下的导向向量形成器，其具有抑制干扰信号的能力：

$$\mathbf{W}(j\omega) = \frac{\mathbf{R}_{XX}^{-1}(j\omega) \mathbf{A}(j\omega)}{\mathbf{A}^H(j\omega) \mathbf{R}_{XX}^{-1}(j\omega) \mathbf{A}(j\omega)}$$

针对宽带的语言信号，我们使用MVDR的变体——子带的MVDR波束形成方法。

对于子带MVDR有如下说明：The Subband MVDR Beamformer block performs minimum variance distortionless response (MVDR) beamforming on wideband signals. Signals are decomposed into frequency subbands and narrowband MVDR beamforming is performed in each band. The resulting subband signals are summed to form the output signal. MVDR beamforming preserves signal power in a given direction while suppressing interference and noise from other directions. The MVDR beamformer is also called the Capon beamformer.

下述例子将子带MVDR波束形成应用到一个十一个单元的直线麦克风阵列上，对到达的简单语音信号进行波束形成，以优化从0度方位角和0度仰角到达的线性调频语音信号的增益。该语音信号的带宽为2.0 kHz。此外，还有单位振幅为2.250 kHz的干扰正弦波从方位角28度和仰角0度到达。我们将会显示MVDR波束成形器如何消除干扰信号，显示在2.250kHz附近的几个频率的阵列模式。在这里音速设置为1500米/秒（根据不同的需要可以修改）。

```
%> 模拟到达语音信号和噪声
array = phased.ULA('NumElements', 11, 'ElementSpacing');
fs = 2000;
carrierFreq = 2000;
t = (0:1/fs:2)';
sig = chirp(t, 0, 2, fs/2);
```

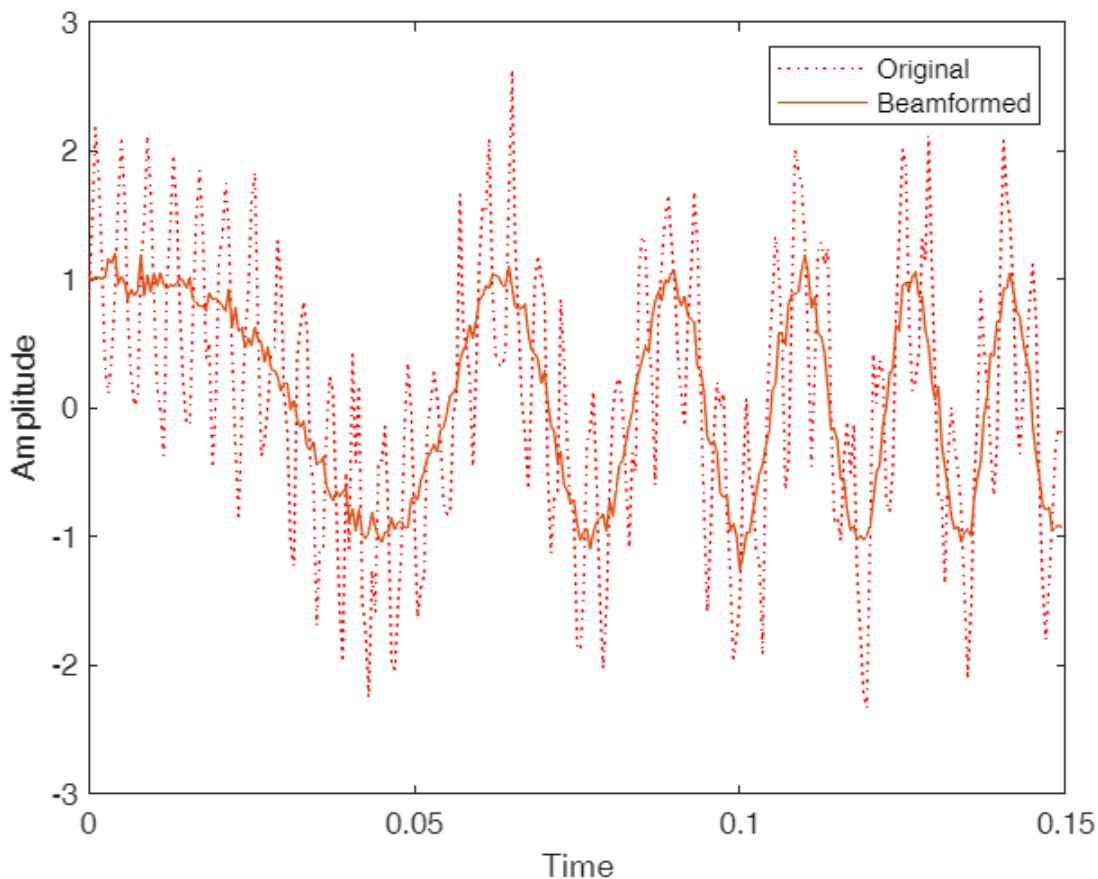
```

c = 1500;
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c, ...
    'SampleRate',fs,'ModulatedInput',true, ...
    'CarrierFrequency',carrierFreq);
incidentAngle = [0;0];
sig1 = collector(sig,incidentAngle);
noise = 0.3*(randn(size(sig1)) + 1j*randn(size(sig1)));

%% 结合语音信号和噪声信号
fint = 250;
sigint = sin(2*pi*fint*t);
interfangle = [28;0];
sigint1 = collector(sigint,interfangle);
rx = sig1 + sigint1 + noise;

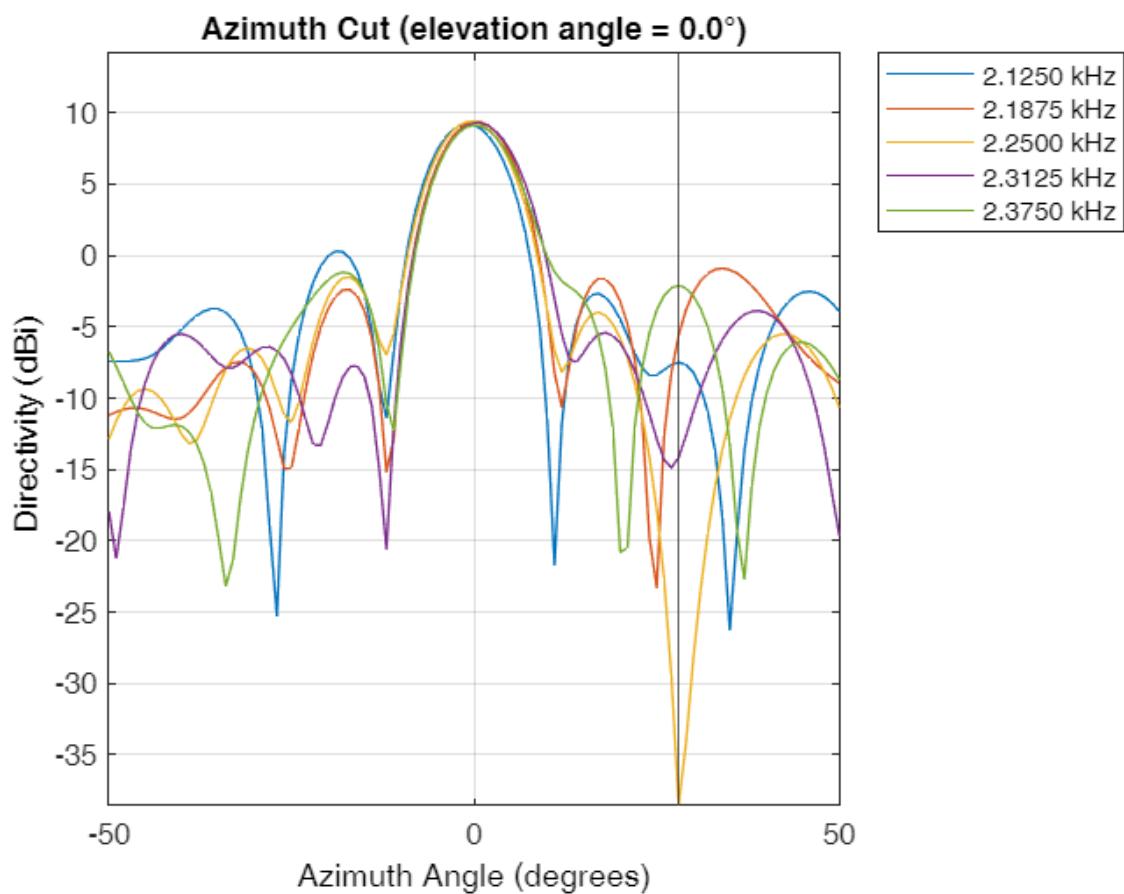
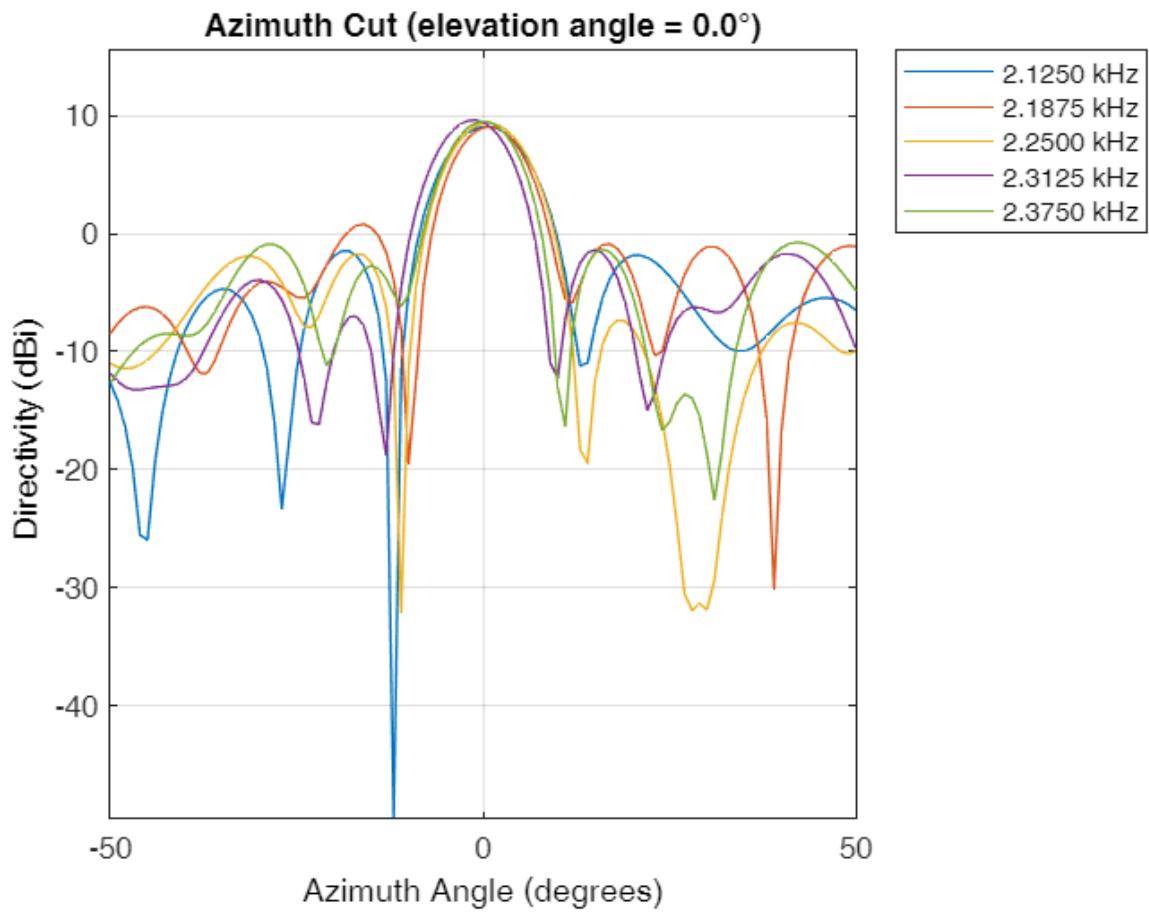
%% 加载MVDR到不同阵列信号通道上
beamformer = phased.SubbandMVDRBeamformer('SensorArray',array, ...
    'Direction',incidentAngle,'OperatingFrequency',carrierFreq, ...
    'PropagationSpeed',c,'SampleRate',fs,'TrainingInputPort',true, ...
    'NumSubbands',64, ...
    'SubbandsOutputPort',true,'WeightsOutputPort',true);
[y,w,subbandfreq] = beamformer(rx,sigint1 + noise);
tidx = [1:300];
plot(t(tidx),real(rx(tidx,6)), 'r:', t(tidx),real(y(tidx)))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')

```



上图可明显看出波束形成后的可将干扰信号消除。

不同频率阵列响应对比：



我们可以很清楚的看到当 $f = 2.25\text{kHz}$ 时，方向图在28度（黑线标出）MVDR能够自适应的将这一干扰消除。这就是波束形成算法的魅力，不需要知道干扰的具体位置，就可以将其有效抑制。

3.5 自适应波束形成算法

自适应波束形成基础理论部分：

传统自适应波束形成的结构如图 3.3.1 所示，波束形成的权重通过自适应信号处理获得。假定阵元 m 的输出为连续基带（即复包络）信号 $x_m(t)$ ，经过 A/D 转换后，变成离散基带信号 $x_m(k)$ ，其中 $m=0,1,\cdots,M-1$ ，并以阵元 0 为参考点。另外，假定共有 Q 个信源存在， $w_q(k)$ 表示在时刻 k 对第 q 信号解调所加的权向量，其中 $q=1,\cdots,Q$ 。权向量用某种准则确定，使解调出来的第 q 个信号的质量在某种意义上最优。

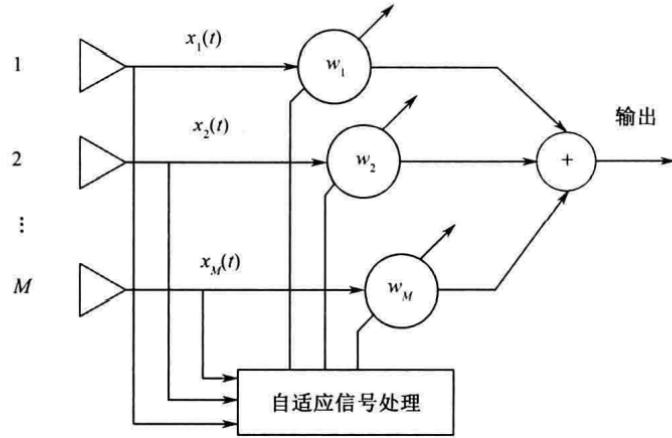


图 3.3.1 自适应波束形成的结构

在最佳波束形成中，权向量通过代价函数的最小化确定。在典型情况下，这种代价函数越小，阵列输出信号的质量也越好，因此当代价函数最小时，自适应阵列输出信号的质量最好。

代价函数有两种最常用的形式，它们均是通信系统中广泛使用的最著名方法——最小均方误差（MMSE）方法和最小二乘（LS）方法。

1. MMSE 方法

MMSE 准则是波形估计、信号检测和系统参数辨识等信号处理中广泛使用的一种优化准则。顾名思义，MMSE 准则就是使估计误差 $y(k) - d_q(k)$ 的均方值最小化，即代价函数取

$$J(\mathbf{w}_q) = \mathbb{E}[\|\mathbf{w}_q^H \mathbf{x}(k) - d_q(k)\|^2] \quad (3.3.1)$$

式中 $\mathbf{x}(k) = [x_0(k), x_1(k), \dots, x_{M-1}(k)]^T$ 。代价函数为第 q 个信号的阵列输出与该信号在时刻 k 的期望形式之间的平方误差的数学期望值。上式可以展开成

$$J(\mathbf{w}_q) = \mathbf{w}_q^H \mathbb{E}[\mathbf{x}(k) \mathbf{x}^H(k)] \mathbf{w}_q - \mathbb{E}[d_q(k) \mathbf{x}^H(k)] \mathbf{w}_q - \mathbf{w}_q^H \mathbb{E}[\mathbf{x}(k) d_q^*(k)] + \mathbb{E}[d_q(k) d_q^*(k)]$$

由上式可以求得

$$\frac{\partial}{\partial \mathbf{w}_q} J(\mathbf{w}_q) = 2 \mathbb{E}[\mathbf{x}(k) \mathbf{x}^H(k)] \mathbf{w}_q - 2 \mathbb{E}[\mathbf{x}(k) d_q^*(k)] = 2 \mathbf{R}_x \mathbf{w}_q - 2 \mathbf{r}_{xd} \quad (3.3.2)$$

式中 \mathbf{R}_x 是数据向量 $\mathbf{x}(k)$ 的自相关矩阵，即

$$\mathbf{R}_x = \mathbb{E}[\mathbf{x}(k) \mathbf{x}^H(k)] \quad (3.3.3)$$

而 \mathbf{r}_{xd} 是数据向量 $\mathbf{x}(k)$ 与期望信号 $d_q(k)$ 的互相关向量，即

$$\mathbf{r}_{xd} = \mathbb{E}[\mathbf{x}(k) d_q^*(k)] \quad (3.3.4)$$

令 $\frac{\partial}{\partial \mathbf{w}_q} J(\mathbf{w}_q) = 0$ ，则得

$$\mathbf{w}_q = \mathbf{R}_x^{-1} \mathbf{r}_{xd} \quad (3.3.5)$$

这就是 MMSE 意义下的最佳阵列权向量，它是 Wiener 滤波理论中最佳滤波器的标准形式。

2. LS 方法

在 MMSE 方法中，代价函数定义为阵列输出与第 q 个用户期望响应之间误差平方的总体平均（均方差），实际数据向量总是有限长的，如果直接定义代价函数为其误差平方，则得到 LS 方法。

假定有 N 个快拍的数据向量 $\mathbf{x}(k)$, $k=1, \dots, N$, 定义代价函数

$$J(\mathbf{w}_q) = \left| \sum_{k=1}^N [\mathbf{w}_q^H(k) \mathbf{x}(k) - d_q(k)] \right|^2 \quad (3.3.6)$$

则求出其梯度为

$$\nabla J(\mathbf{w}_q) = \frac{\partial}{\partial \mathbf{w}_q} J(\mathbf{w}_q) = 2 \sum_{m=1}^N \sum_{n=1}^N \mathbf{x}(m) \mathbf{x}^H(n) \mathbf{w}_q - 2 \sum_{m=1}^N \sum_{n=1}^N \mathbf{x}(m) d_q^*(n) \quad (3.3.7)$$

令梯度等于零，易得

$$\mathbf{w}_q = (\mathbf{X}^H \mathbf{X})^{-1} \mathbf{X}^H \mathbf{d}_q \quad (3.3.8)$$

这就是最小二乘意义下针对第 q 个用户的波束形成器的最佳权向量，式中 \mathbf{X} 和 \mathbf{d}_q 分别是数据向量和期望信号向量。其值为

$$\begin{aligned} \mathbf{X} &= [\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(N)] \\ \mathbf{d}_q &= [d_q(1), d_q(2), \dots, d_q(N)]^T \end{aligned} \quad (3.3.9)$$

上面介绍的 MMSE 方法和 LS 方法的核心问题是，在对第 q 个用户进行波束形成时，需要在接收端使用该信源的期望响应。为了提供这一期望响应，就必须周期性发送对发射机和接收机二者皆为已知的训练序列。训练序列占用了通信系统宝贵的频谱资源，这是

MMSE 方法和 LS 方法均存在的主要缺陷。一种可以代替训练序列的方法是采用决策指向更新对期望响应进行学习。在决策指向更新中，期望信号样本的估计根据阵列输出和信号解调器的输出重构。由于期望信号是在接收端产生的，不需要发射数据的知识，因此不需要训练序列。

笔者思考：自适应波束形成中的 MMSE 和 LS 方法都需要与用户达成一种标准信号校准的形式，使得该算法可以利用标准信号训练出对应的波束形成权重向量。后续的解决方法是在接收信号端进行校准信号的放置。

3.3.2 权向量更新的自适应算法

上面介绍的自适应阵列的最佳权向量的确定需要求解方程，一般来说，并不希望直接求解方程，其理由如下：①由于移动用户环境是时变的，所以权向量的解必须能及时更新；②由于估计最佳解需要的数据是含噪声的，所以希望使用一种更新技术，可利用已求出的权向量求平滑最佳响应的估计，以减小噪声的影响。因此，希望使用自适应算法周期更新权向量。

自适应算法既可采用迭代模式，也可采用分块模式。所谓迭代模式，就是在每个迭代步骤， n 时刻的权向量加上一校正量后，即组成 $n+1$ 时刻的权向量，用它逼近最佳权向量。在分块模式中，权向量不是每个时刻都更新，而是每隔一定时间周期才更新；由于一定时间周期对应于一数据块而不是一数据点，所以这种更新又称分块更新。

为了使阵列系统能自适应工作，就必须将上节介绍的方法归结为自适应算法。这里以MMSE方法为例，说明如何把它变成一种自适应算法。

考虑随机梯度算法，其更新权向量的一般公式为

$$\mathbf{w}_q(k+1) = \mathbf{w}_q(k) - \frac{1}{2} \mu \nabla \quad (3.3.10)$$

式中， $\nabla = \frac{\partial}{\partial \mathbf{w}_q(k)} J(\mathbf{w}_q(k))$ ， μ 称为收敛因子，它控制自适应算法的收敛速度，则

$$\nabla = \mathbf{R}_x \mathbf{w}_q(k) - \mathbf{r}_{xd} = \mathbb{E}[\mathbf{x}(k) \mathbf{x}^H(k)] \mathbf{w}_q(k) - \mathbb{E}[\mathbf{x}(k) d_q^*(k)] \quad (3.3.11)$$

上式中的数学期望用各自的瞬时值代替，即得 k 时刻的梯度估计值如下

$$\hat{\nabla}(k) = \mathbf{x}(k) [\mathbf{x}^H(k) \mathbf{w}_q(k) - d_q^*(k)] = \mathbf{x}(k) f(k) \quad (3.3.12)$$

式中， $f(k) = \mathbf{x}^H(k) \mathbf{w}_q(k) - d_q^*(k)$ ，代表阵列输出与第 q 个用户期望响应 $d_q(k)$ 之间的瞬时误差。容易证明，梯度估计 $\hat{\nabla}(k)$ 是真实梯度 ∇ 的无偏估计。

将式(3.3.12)代入式(3.3.10)，即得到熟悉的LMS自适应算法为

$$\mathbf{w}_q(k+1) = \mathbf{w}_q(k) - \mu \mathbf{x}(k) f(k) \quad (3.3.13)$$

MMSE方法可以用LMS算法实现，而LS方法的自适应算法为递推最小二乘(RLS)算法。表3.3.1列出了自适应阵列系统权向量更新的三种自适应算法，它们是LMS算法、RLS算法和Bussgang算法。从表中可看出，自适应算法LMS和RLS需要使用训练序列，但Bussgang算法不需要训练序列。除了Bussgang算法外，还有一些自适应算法也不需要训练序列。这些不需要训练序列的方法习惯统称为盲自适应算法。

注意，在Bussgang算法中， $g(y(k))$ 是一个非线性的估计子，它对解调器输出的信号 $y(k)$

作用，并用 $g(y(k))$ 代替期望信号 $d(k)$ ，然后产生误差函数 $e(k)=d(k)-y(k)$ 。

表 3.3.1 三种自适应波束形成算法的比较

算法	最小均方(LMS)算法	递推最小二乘(RLS)算法	Bussgang算法
初始化	$\hat{\mathbf{w}}_0 = 0$	$\hat{\mathbf{w}}_0 = 0; \mathbf{P}_0 = \delta^{-1} \mathbf{I}$	$\hat{\mathbf{w}}_0 = [1, 0, \dots, 0]^T$
更新公式	$y(k) = \hat{\mathbf{w}}^H(k) \mathbf{x}(k)$ $e(k) = d(k) - y(k)$ $\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \mu \mathbf{x}(k) e^*(k)$	$\mathbf{v}(k) = \mathbf{P}(k-1) \mathbf{x}(k)$ $\mathbf{u}(k) = \frac{\lambda^{-1} \mathbf{v}(k)}{1 + \lambda^{-1} \mathbf{x}^H(k) \mathbf{v}(k)}$ $\alpha(k) = d(k) - \hat{\mathbf{w}}^H(k-1) \mathbf{x}(k)$ $\hat{\mathbf{w}}(k) = \hat{\mathbf{w}}(k-1) + \mathbf{u}(k) \alpha^*(k)$ $\mathbf{P}(k) = \lambda^{-1} [\mathbf{I} - \mathbf{u}(k) \mathbf{x}^H(k)] \mathbf{P}(k-1)$	$y(k) = \hat{\mathbf{w}}^H(k) \mathbf{x}(k)$ $e(k) = g(y(k)) - y(k)$ $\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \mu \mathbf{x}(k) e^*(k)$
收敛因子	步长参数 μ $0 < \mu < \text{tr}(\mathbf{R})$	遗忘因子 λ $0 < \lambda < 1$	步长参数 μ

***下面是将自适应波束形成算法在频域或小波变换域的实现，注意，matlab中子带的语音处理自适应波束的算法是基于频域的。

3.3.3 基于变换域的自适应波束形成算法

LMS 的优点是结构简单、算法复杂度低、易于实现、稳定性高；缺点主要是收敛速度较慢，因而其应用也受到一定的限制。分析表明，影响 LMS 自适应波束形成器收敛速度的主要因素是输入信号的最大、最小特征值之比，该值越小收敛就越快^[33]。为了提高收敛速度和计算性能，人们开始研究变换域的自适应滤波方法。文献[34~36]研究了频域的波束形成技术；文献[37]研究了基于余弦变换的波束形成技术；文献[39]改进了频域自适应波束形成算法；文献[40]提出了小波域自适应波束形成算法。

基于频域 LMS 的自适应算法结构如图 3.3.2 所示，该算法先对输入信号进行 FFT 变换，再通过 LMS 算法在频域上进行波束形成。根据前面分析知道：通过对阵列天线接收到的信号 $\mathbf{x}(n)$ 进行 FFT，经过 FFT 后的 $\mathbf{r}(n)$ ，自相关性下降，呈带状分布，这样 LMS 算法收敛速度就很快。当存在相干信源，假设它们 DOA 不同，相干信源在时域相干，但在频域是不相干的，所以基于频域 LMS 的自适应波束形成算法对相干信源具有鲁棒性。

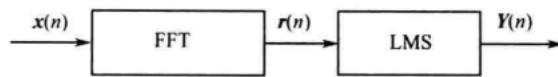


图 3.3.2 基于频域 LMS 的自适应算法结构

FFT 变换后的矩阵 \mathbf{R}_{rr} 的最大、最小特征值之比小于 \mathbf{R}_{xx} 的最大、最小特征值之比。所以，自适应波束形成算法的收敛速度得到了提高。基于频域 LMS 的自适应波束形成算法先对输入信号进行频域变换，然后用 LMS 算法来实现在频域自适应波束形成。与最小均方（LMS）自适应波束形成算法相比，增加 FFT 的额外的计算量。但频域变换采用快速算法，计算量不大。设阵列中传感器数量为 M ，LMS 算法每迭代一次的复数加法次数为 $2M$ ，复数乘法次数约为 $2M+1$ 。FFT 中复数加法次数为 $M\log_2 M$ ，复数乘法复杂度为 $M/2 \times \log_2 M$ 。当

$M=32$ 时, FFT 只相当于数次 LMS 迭代。而且 FFT 已经有现成硬件, 实现容易。经 FFT 变换后信号自相关性下降, 之后的 LMS 算法收敛速度大大提高。总体而言, 基于频域 LMS 的自适应波束形成算法的计算量比 LMS 自适应波束形成算法的计算量下降了很多。

文献[39]研究的降维频域自适应波束形成算法结构如图 3.3.3 所示, 该算法先对接收信号进行 FFT, 然后再带通滤波, 最后通过 LMS 算法实现了频域的自适应波束形成。

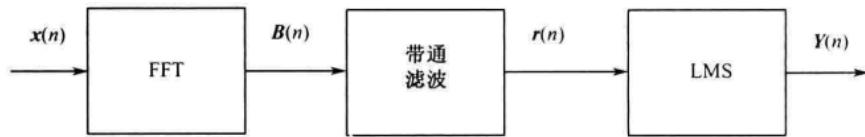


图 3.3.3 降维的频域自适应波束形成

文献[40]提出了小波域的自适应波束形成算法。小波域的波束形成算法结构如图 3.3.4 所示, 先多分辨率分解, 再进行 LMS 算法。根据前面分析, 不同的 DOA 对应于不同的空间分辨率, 通过对阵列天线接收到的信号 $x(n)$ 进行多分辨率分解, 经过小波变换后的 $r(n)$ 是稀疏矩阵, 所以 LMS 算法收敛速度就很快。

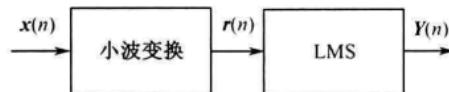


图 3.3.4 小波域的波束形成算法

3.6 广义旁瓣相消器 (GSC) 的波束形成算法及其改进

3.6.1 广义旁瓣相消器在书中的理论基础

线性约束最小方差 (LCMV) 准则是最常用的自适应波束形成方法。广义旁瓣相消器 (GSC) 是 LCMV 一种等效的实现结构, GSC 结构将自适应波束形成的约束优化问题转换为无约束的优化问题, 分为自适应和非自适应两个支路, 分别称为辅助支路和主支路, 要求期望信号只能从非自适应的主支路通过, 而自适应的辅助支路中仅含有干扰和噪声分量。在高信噪比的情况下, 将有一部分期望信号泄漏到辅助支路中, 因而出现了信号相消现象。文献[23]提出了信号子空间投影的 GSC 改进算法, 来提高 GSC 稳健性, 但在低信噪比下易发生波束形成畸变。本文将提出一种改进的广义旁瓣相消器 (GSC) 的波束形成方法, 即基于特征结构的 GSC 算法 (ES-GSC)。该算法不仅克服了传统 GSC 算法在高信噪比下波束形成效果变差的缺点, 而且克服了文献[23]提出的改进 GSC 算法在低信噪比下性能差的缺点。

3.4.1 广义旁瓣相消器 (GSC) 算法

线性约束最小方差 (LCMV) 准则可表示为

$$\mathbf{w} = \arg \min_{\mathbf{w}} \mathbf{w}^H \mathbf{R} \mathbf{w} \quad \text{s.t.} \quad \mathbf{C}^H \mathbf{w} = \mathbf{f} \quad (3.4.1)$$

其中, \mathbf{R} 为接收信号的自相关矩阵, \mathbf{C} 为 $M \times (J+1)$ 维约束矩阵, \mathbf{f} 为 $(J+1)$ 维约束向量, M 为阵列中天线数, J 为干扰信号的个数。上式的最优解为

$$\mathbf{w} = \mathbf{R}^{-1} \mathbf{C} (\mathbf{C}^H \mathbf{R}^{-1} \mathbf{C})^{-1} \mathbf{f} \quad (3.4.2)$$

如图 3.4.1 所示, 在与 LCMV 等效的广义旁瓣相消器结构中, 权向量被分解为自适应权和非自适应权两部分, 其中非自适应部分位于约束子空间中, 而自适应部分正交于约束子空间, 系统的权向量可表示为

$$\mathbf{w} = \mathbf{w}_q - \mathbf{B} \mathbf{w}_a \quad (3.4.3)$$

其中,

$$\mathbf{w}_q = (\mathbf{C} \mathbf{C}^H)^{-1} \mathbf{C} \mathbf{f} \quad (3.4.4)$$

$$\mathbf{w}_a = (\mathbf{B}^H \mathbf{R} \mathbf{B})^{-1} \mathbf{B}^H \mathbf{R} \mathbf{w}_q \quad (3.4.5)$$

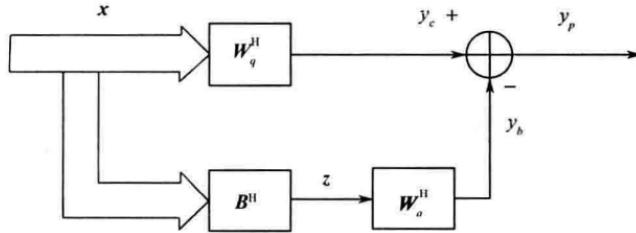


图 3.4.1 广义旁瓣对消器结构

\mathbf{B} 为 $M \times (M - J - 1)$ 维阻塞矩阵, $\mathbf{B}^H \mathbf{C} = 0$, \mathbf{B} 的作用就是将期望信号阻塞掉而使之不进入辅助支路, 组成 \mathbf{B} 的列向量位于约束子空间的正交互补空间中, 令 $y_c = \mathbf{w}_q^H \mathbf{x}$, $z = \mathbf{B}^H \mathbf{x}$, 则自适应权向量又可表示为 $\mathbf{w}_a = \mathbf{R}_z^{-1} \mathbf{p}_z$, \mathbf{w}_a 是使上下支路均方误差最小化的维纳解, 其中 $\mathbf{R}_z = \mathbf{B}^H \mathbf{R} \mathbf{B}$ 是 z 的协方差矩阵, $\mathbf{p}_z = \mathbf{B}^H \mathbf{R} \mathbf{w}_q$ 是 z 和 y_c 的互相关向量, 若 z 中含有很少的期望信号时, GSC 仍能正常工作, 但若 z 所含的期望信号超过一定程度时, 将会引起严重的期望信号相消现象。

3.4.2 GSC 的改进算法

文献[23]提出了信号子空间投影的 GSC 改进算法(IGSC), 以提高 GSC 的稳健性。GSC 的阻塞矩阵 \mathbf{B} 一般由约束子空间可正交互补空间的一个基构成, 从而有 $\mathbf{B}^H \mathbf{C} = 0$ 。为了便于说明, 假设 $\mathbf{C} = \mathbf{a}(\theta_0)$, 即期望信号方向矢量。在高信噪比的情况下, 阻塞矩阵 \mathbf{B} 不能全部阻塞期望信号, 将有一部分期望信号泄漏到辅助支路中, 出现了信号相消现象。为了减少泄漏到 GSC 辅助支路中期望信号的能量, 对阻塞矩阵 \mathbf{B} 加以改进。对阵列协方差矩阵 \mathbf{R} 进行特征分解, 得到期望信号和干扰信号的子空间 \mathbf{U} , 统称为信号子空间, 其中 $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{J+1}$ 是 \mathbf{R} 的 $J+1$ 个大特征值对应的特征矢量, 把 $\mathbf{a}(\theta_0)$ 向信号子空间投影, 得到

$$\mathbf{a}_p(\theta_0) = \mathbf{U} \mathbf{U}^H \mathbf{a}(\theta_0) \quad (3.4.6)$$

用 $\mathbf{a}_p(\theta_0)$ 的正交补生成的阻塞矩阵 \mathbf{B}_p 比直接用 $\mathbf{a}(\theta_0)$ 的正交补生成的阻塞矩阵 \mathbf{B} 有更好的阻塞能力。波束形成的权向量可表示为

$$\mathbf{w} = \mathbf{w}_q - \mathbf{B}_p \mathbf{w}'_a \quad (3.4.7)$$

其中, \mathbf{w}_q 如公式 (3.4.4) 所示, \mathbf{w}'_a 表示为

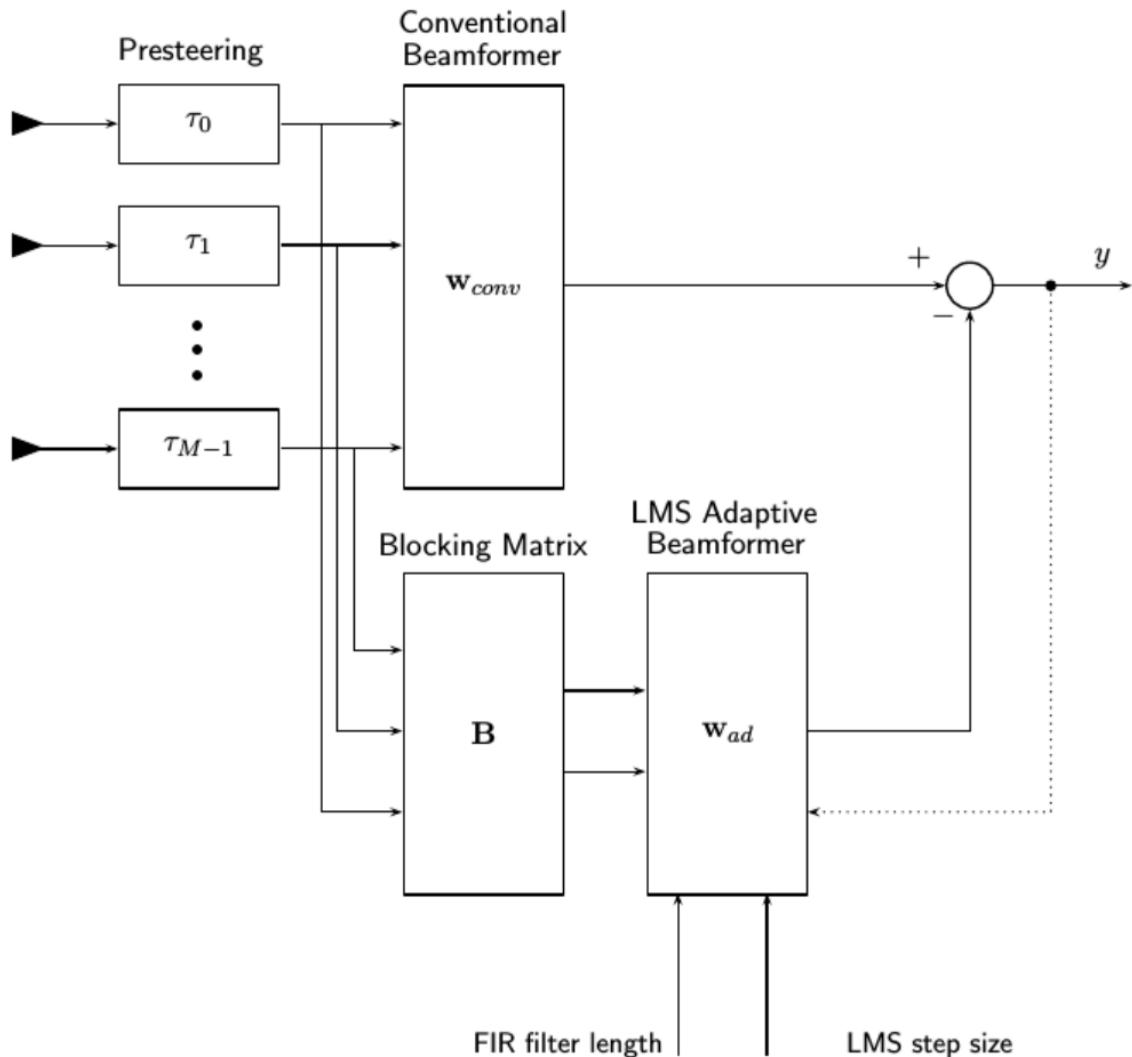
$$\mathbf{w}'_a = (\mathbf{B}_p^H \mathbf{R} \mathbf{B}_p)^{-1} \mathbf{B}_p^H \mathbf{R} \mathbf{w}_q \quad (3.4.8)$$

笔者思考如下：

广义侧波消除器（GSC）是线性约束最小方差（LCMV）波束成形器的一个有效实现。LCMV波束成形器使阵列的输出功率最小，同时保留一个或多个指定方向的功率。这种类型的波束成形器被称为受限波束成形器。你可以为受限波束成形器计算精确的权重，但当元素的数量很大时，计算的成本很高。这种计算需要反演一个大的空间协方差矩阵。GSC公式将自适应约束优化LCMV问题转换为自适应无约束问题，从而简化了实施。

在GSC算法中，传入的传感器数据被分成两条信号路径，如框图所示。上面的路径是一个传统的波束成形器。下层路径是一个自适应无约束波束成形器，其目的是最小化GSC输出功率。GSC算法由以下步骤组成。

1. 在GSC算法中，传入的传感器数据被分成两条信号路径，如框图所示。上面的路径是一个传统的波束成形器。下层路径是一个自适应无约束波束成形器，其目的是最小化GSC输出功率。GSC算法由以下步骤组成。
 2. 通过对传入的信号进行时移，对元素传感器数据进行预处理。预先对所有传感器元件信号进行时间调整。时间偏移取决于信号的到达角度。
 3. 将预置的信号通过上层路径传入具有固定权重的传统波束成形器 w_{conv} 。
 4. 阻塞矩阵与信号正交，并将信号从下层路径移除。
 5. 通过一组FIR滤波器对下层路径的信号进行过滤。FilterLength属性设置了滤波器的长度。滤波器的系数是自适应滤波器的权重 w_{ad} 。
 6. 计算上部和下部信号路径之间的差值。这个差值就是波束成形的GSC输出。
 7. 将波束成形的输出反馈给滤波器。滤波器使用最小均方（LMS）算法适应其权重。实际的自适应LMS步长等于LMSStepSize属性的值除以总信号功率。



3.6.2 matlab代码进行GSC波束形成

我们在空气中建立一个有十一个单元的直线麦克风阵列。一个简单的宽带语音信号在方位角-50度和仰角0度方向入射到阵列上。将GSC波束形成的信号与Frost波束形成的信号进行比较。信号的传播速度为340米/秒，采样率为8千赫。

创建传声器和阵列系统对象。阵列元素的间距为二分之一波长。设置信号频率为奈奎斯特频率的二分之一。需要指出的是函数chirp可以创造出近似于语音信号的宽带信号，便于实验进行。

```

c = 340.0;
fs = 8.0e3;
fc = fs/2;
lam = c/fc;
transducer = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[20
20000]);
array =
phased.ULA('Element',transducer,'NumElements',11,'ElementSpacing',lam/2);

%%模拟一个500Hz带宽的语音信号
t = 0:1/fs:.5;
signal = chirp(t,0,0.5,500);

%% 让语音信号入射到麦克风阵列，加入环境高斯白噪声
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c, ...
    'SampleRate',fs,'ModulatedInput',false,'NumSubbands',512);
incidentAngle = [-50;0];

```

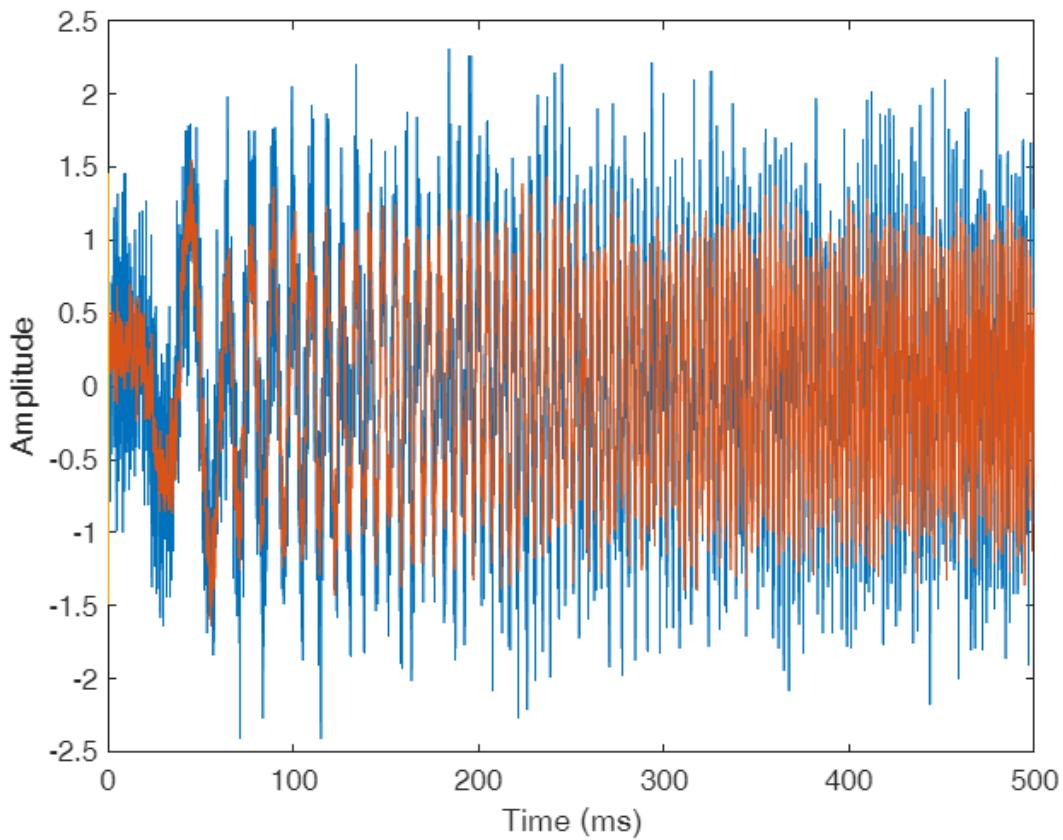
```

signal = collector(signal.',incidentAngle);
noise = 0.5*randn(size(signal));
recsignal = signal + noise;

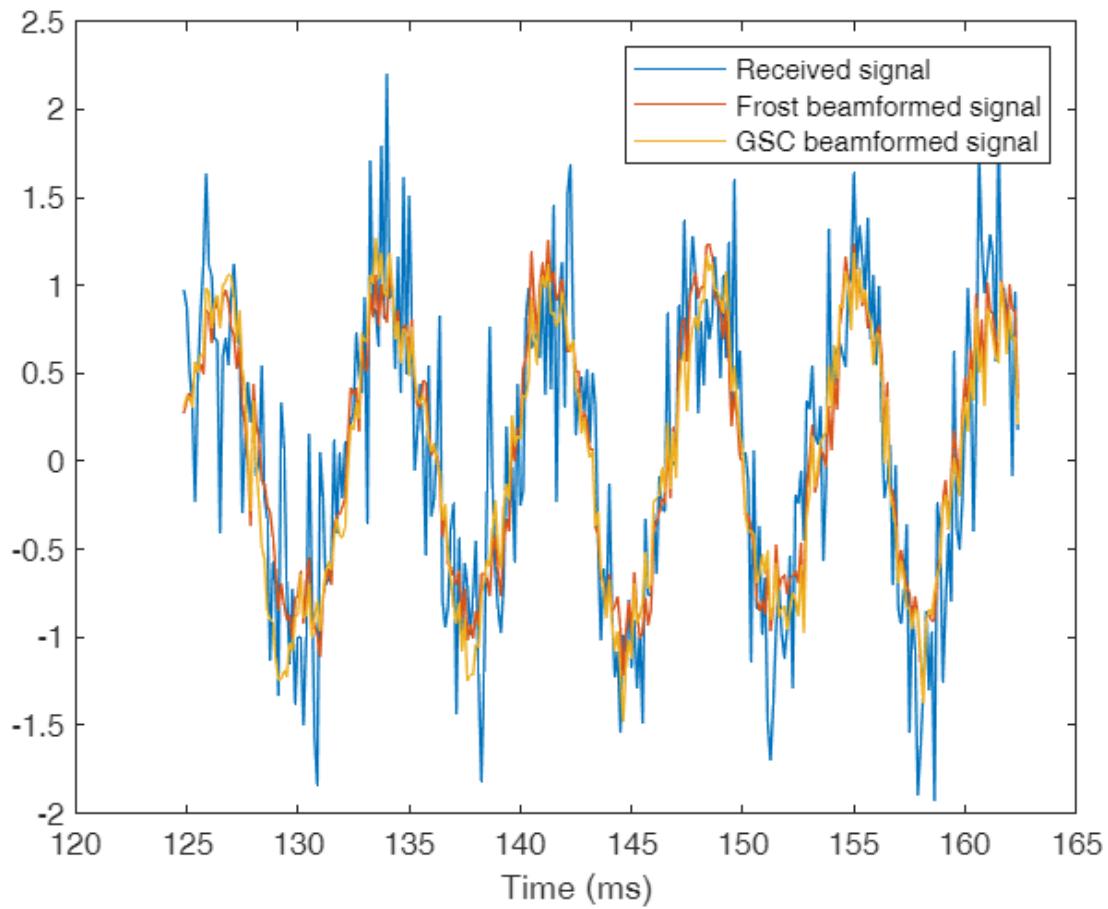
%% 加载一个frost beamforming 针对给定期望信号入射角度
frostbeamformer = phased.FrostBeamformer('SensorArray',array,'PropagationSpeed',
...
c,'SampleRate',fs,'Direction',incidentAngle,'FilterLength',15);
yfrost = frostbeamformer(recsignal);

%% 执行GSC波束成形，并绘制波束成形器输出与Frost波束成形器输出的对比图。同时绘制到达阵列中间元素的非波束成形信号。
gscbeamformer = phased.GSCBeamformer('SensorArray',array, ...
'PropagationSpeed',c,'SampleRate',fs,'Direction',incidentAngle, ...
'FilterLength',15);
ygsc = gscbeamformer(recsignal);
plot(t*1000,recsignal(:,6),t*1000,yfrost,t,ygsc)
xlabel('Time (ms)')
ylabel('Amplitude')

```



放大仔细看



我们也可以将另外一个非入射角度传入GSC beamformer中，试试对其他方向beamfrom会发生什么

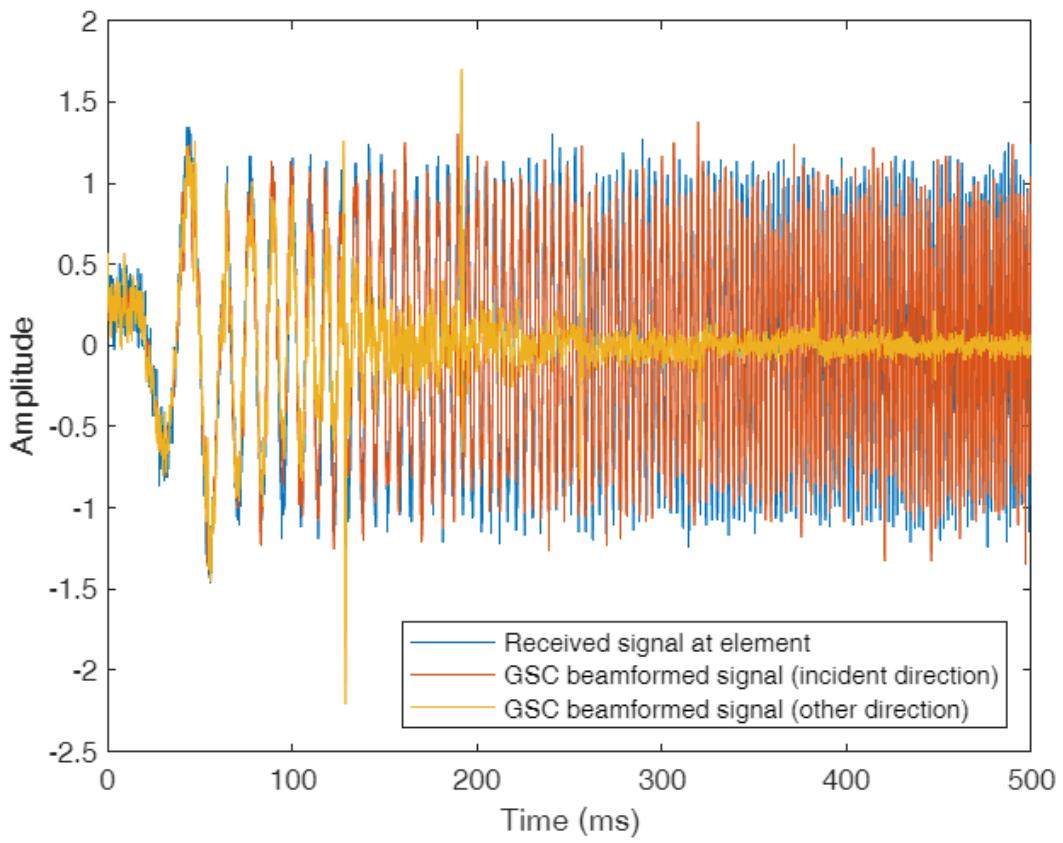
```
c = 340.0;
fs = 8.0e3;
fc = fs/2;
lam = c/fc;
transducer = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[20
20000]);
array =
phased.ULA('Element',transducer,'NumElements',11,'ElementSpacing',lam/2);

t = 0:1/fs:0.5;
signal = chirp(t,0,0.5,500);

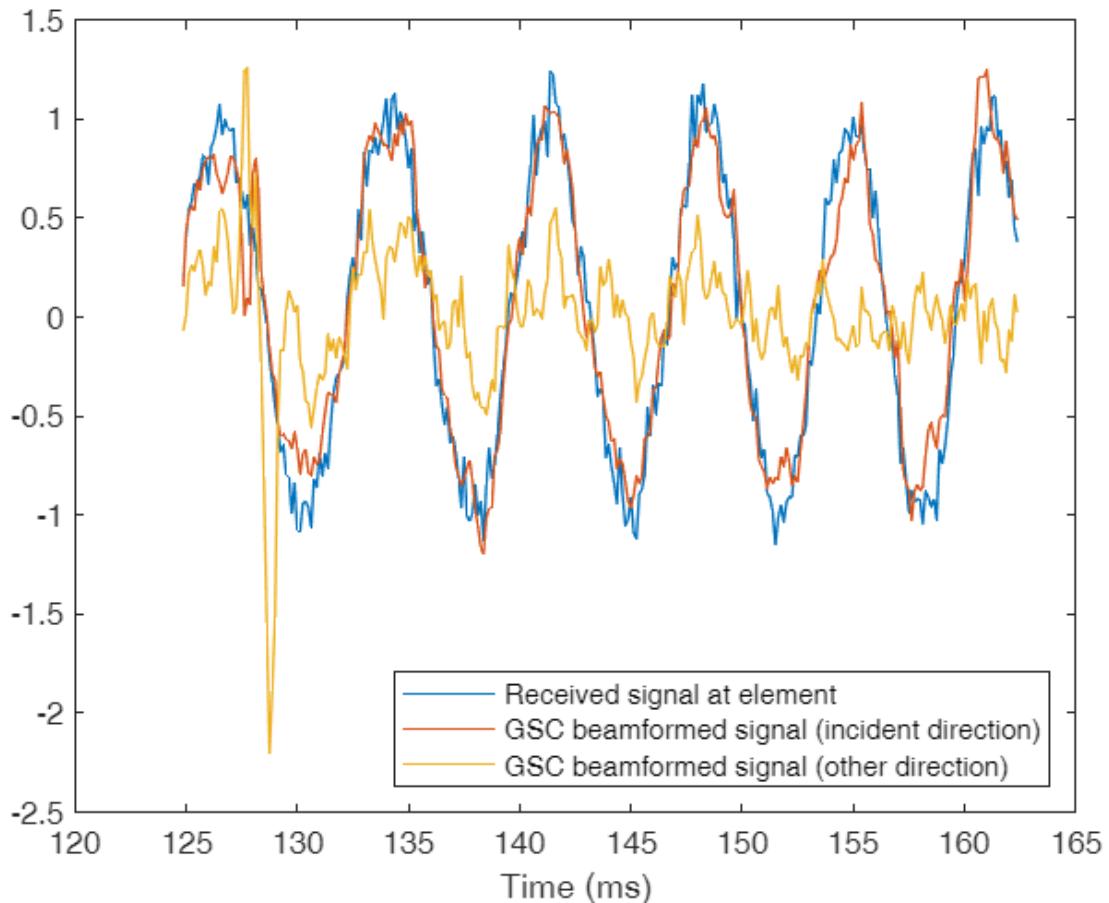
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c, ...
    'SampleRate',fs,'ModulatedInput',false,'NumSubbands',512);
incidentAngle = [-50;0];
signal = collector(signal.','');
noise = 0.1*randn(size(signal));
recsignal = signal + noise;

gscbeamformer = phased.GSCBeamformer('SensorArray',array, ...
    'PropagationSpeed',c,'SampleRate',fs,'DirectionSource','Input port', ...
    'FilterLength',5);
ygscl = gscbeamformer(recsignal,incidentAngle);
ygsco = gscbeamformer(recsignal,[20;30]);
plot(t*1000,recsignal(:,6),t*1000,ygscl,t*1000,ygsco)
xlabel('Time (ms)')
ylabel('Amplitude')
```

```
legend('Received signal at element','GSC beamformed signal (incident direction)', ...
'GSC beamformed signal (other direction)','Location','southeast')
```



放大看看：



3.8 实时声源定位和波束形成

我们主要将基于宽带信号的music声源定位算法与基于frost-beam former的自适应波束形成算法相结合从而满足实时定位收音。算法核心在于对收得语音信号进行分帧处理后，对每一帧的语音信号做doa加beam forming，使其可进行实时运行，便于与后端的语音识别模块进行组合，实现恶劣背景噪声环境下的语音识别系统。代码如下：

```
%Select and Configure the Source of Audio Samples
clear all;close all;clc;
sourceChoice = 'recorded';
%Set the duration of live processing. Set how many samples per channel to acquire
%and process each iteration.
endTime = 20;
audioFrameLength = 3200;

%Create the source.
switch sourceChoice
    case 'live'
        fs = 16000;
        audioInput = audioDeviceReader( ...
            'Device','Microphone Array (Microsoft Kinect USB Audio)', ...
            'SampleRate',fs, ...
            'NumChannels',4, ...
            'OutputDataType','double', ...
            'SamplesPerFrame',audioFrameLength);
    case 'recorded'
        % This audio file holds a 20-second recording of 4 raw audio
        % channels acquired with a Microsoft Kinect(TM) for windows(R) in
        % the presence of a noisy source moving in front of the array
        % roughly from -40 to about +40 degrees and then back to the
        % initial position.
        audioFileName = 'AudioArray-16-16-4channels-20secs.wav';
        audioInput = dsp.AudioFileReader( ...
            'OutputDataType','double', ...
            'Filename',audioFileName, ...
            'PlayCount',inf, ...
            'SamplesPerFrame',audioFrameLength);
        fs = audioInput.SampleRate;
    end

[x,fs] = audioread('AudioArray-16-16-4channels-20secs.wav');
sound(x(:,2),fs);

%Define Array Geometry
player = audioDeviceWriter('SampleRate',fs);% define the audio player
micPositions = [-0.088, 0.042, 0.078, 0.11];

freq_test = [200 500 800 1000 1500 2000];
microphone = ...
    phased.OmnidirectionalMicrophoneElement('FrequencyRange',[20 4000]);
figure(1);pattern(microphone,freq_test,
[-180:180],0,'CoordinateSystem','polar','Type','powerdb',...
'Normalize',true);
```

```

ypos = [0 0 0 0]; zpos = [0 0 0 0];
array = phased.ConformalArray('Element',microphone, ...
    'ElementPosition',[micPositions; ypos; zpos]);
figure(2);viewArray(array)
figure(3);pattern(array,freq_test,
[-180:180],0,'CoordinateSystem','polar','Type','powerdb',...
    'Normalize',true,'PropagationSpeed',340.0);
plotFreq = linspace(min(freq_test),max(freq_test),15);
figure(4);pattern(microphone,plotFreq,
[-180:180],0,'CoordinateSystem','rectangular',...
    'PlotStyle','waterfall','Type','powerdb')
figure(5);pattern(array,plotFreq,
[-180:180],0,'CoordinateSystem','rectangular',...
    'PlotStyle','waterfall','Type','powerdb','PropagationSpeed',340);

%Form Microphone Pairs
%The algorithm used in this example works with pairs of microphones
independently. It then combines the individual DOA estimates %to provide a single
live DOA output. The more pairs available, the more robust (yet computationally
expensive) DOA estimation. %The maximum number of pairs available can be computed
as nchoosek(length(micPositions),2). In this case, the 3 pairs with the %largest
inter-microphone distances are selected. The larger the inter-microphone distance
the more sensitive the DOA estimate. %Each column of the following matrix
describes a choice of microphone pair within the array. All values must be
integers between %1 and length(micPositions).
micPairs = [1 4; 1 3; 1 2];
numPairs = size(micPairs, 1);

%Initialize DOA visualization
%Create an instance of the helper plotting object DOADisplay. This displays the
estimated DOA live with an arrow on a polar plot.
DOAPointer = DOADisplay();

%Create and Configure the Algorithmic Building Blocks
%Use a helper object to rearrange the input samples according to how the
microphone pairs are selected.
bufferLength = 64;
preprocessor = PairArrayPreprocessor( ...
    'MicPositions',micPositions, ...
    'MicPairs',micPairs, ...
    'BufferLength',bufferLength);
micSeparations = getPairSeparations(preprocessor);

%The main algorithmic building block of this example is a cross-correlator. That
is used in conjunction with an interpolator to %ensure a finer DOA resolution. In
this simple case it is sufficient to use the same two objects across the
different pairs %available. In general, however, different channels may need to
independently save their internal states and hence to be handled %by separate
objects.
interpFactor = 8;
b = interpFactor * fir1((2*interpFactor*8-1),1/interpFactor);
groupDelay = median(grpdelay(b));
interpolator =
dsp.FIRInterpolator('InterpolationFactor',interpFactor,'Numerator',b);

```

```

%Acquire and Process Signals in a Loop
%For each iteration of the following while loop: read audioFrameLength samples
for each audio channel, process the data to %estimate a DOA value and display the
result on a bespoke arrow-based polar visualization.

close all
tic
for idx = 1:(endTime*fs/audioFrameLength)
    temp_idx = [((idx-1)*audioFrameLength+1):(idx*audioFrameLength)];
    cyclestart = toc;
    % Read a multichannel frame from the audio source
    % The returned array is of size AudioFrameLength x size(micPositions,2)
    multichannelAudioFrame = audioInput();

    % Rearrange the acquired sample in 4-D array of size
    % bufferLength x numBuffers x 2 x numPairs where 2 is the number of
    % channels per microphone pair
    bufferedFrame = preprocess(multichannelAudioFrame);

    % First, estimate the DOA for each pair, independently

    % Initialize arrays used across available pairs
    numBuffers = size(bufferedFrame, 2);
    delays = zeros(1,numPairs);
    anglesInRadians = zeros(1,numPairs);
    xcDense = zeros((2*bufferLength-1)*interpFactor, numPairs);

    % Loop through available pairs
    for kPair = 1:numPairs
        % Estimate inter-microphone delay for each 2-channel buffer
        delayVector = zeros(numBuffers, 1);
        for kBuffer = 1:numBuffers
            % Cross-correlate pair channels to get a coarse
            % crosscorrelation
            xcCoarse = xcorr( ...
                bufferedFrame(:,kBuffer,1,kPair), ...
                bufferedFrame(:,kBuffer,2,kPair));

            % Interpolate to increase spatial resolution
            xcDense = interpolator(flipud(xcCoarse));

            % Extract position of maximum, equal to delay in sample time
            % units, including the group delay of the interpolation filter
            [~,idxloc] = max(xcDense);
            delayVector(kBuffer) = ...
                (idxloc - groupDelay)/interpFactor - bufferLength;
        end

        % Combine DOA estimation across pairs by selecting the median value
        delays(kPair) = median(delayVector);

        % Convert delay into angle using the microsoft pair spatial
        % separations provided
        anglesInRadians(kPair) = HelperDelayToAngle(delays(kPair), fs, ...
            micSeparations(kPair));
    end
end

```

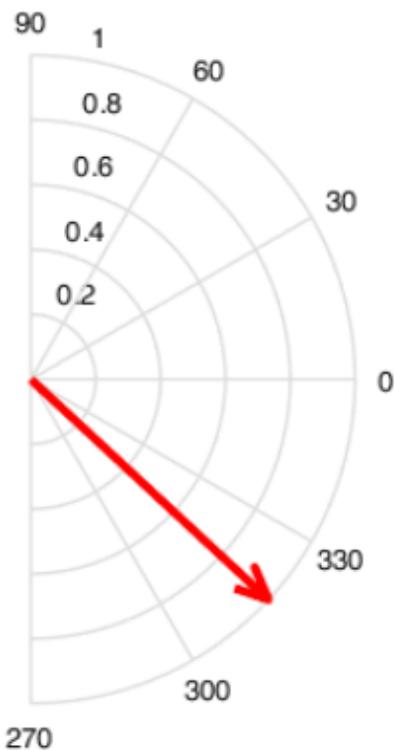
```

% Combine DOA estimation across pairs by keeping only the median value
DOAInRadians = median(anglesInRadians);

% Arrow display
DOAPointer(DOAInRadians)
% time delay beamforming
angle = rad2deg(DOAInRadians);
c = 340; f0 = 1000;d = c/f0/4;M = 15;
theta=-90:0.1:90;theta0=angle;
for i=1:M
    a(i,:)=exp(1j*2*pi*(i-1)*d*sind(theta));
    w(1,i)=exp(1j*2*pi*(i-1)*d*sind(theta0));
end
BP=conj(w)*a/M;
figure(6);polarplot(theta*pi/180,real(BP));title('wideband beamformer');
%player(temp);
% Delay cycle execution artificially if using recorded data
if(strcmp(sourceChoice,'recorded'))
    pause(audioFrameLength/fs - toc + cyclestart)
end
end

```

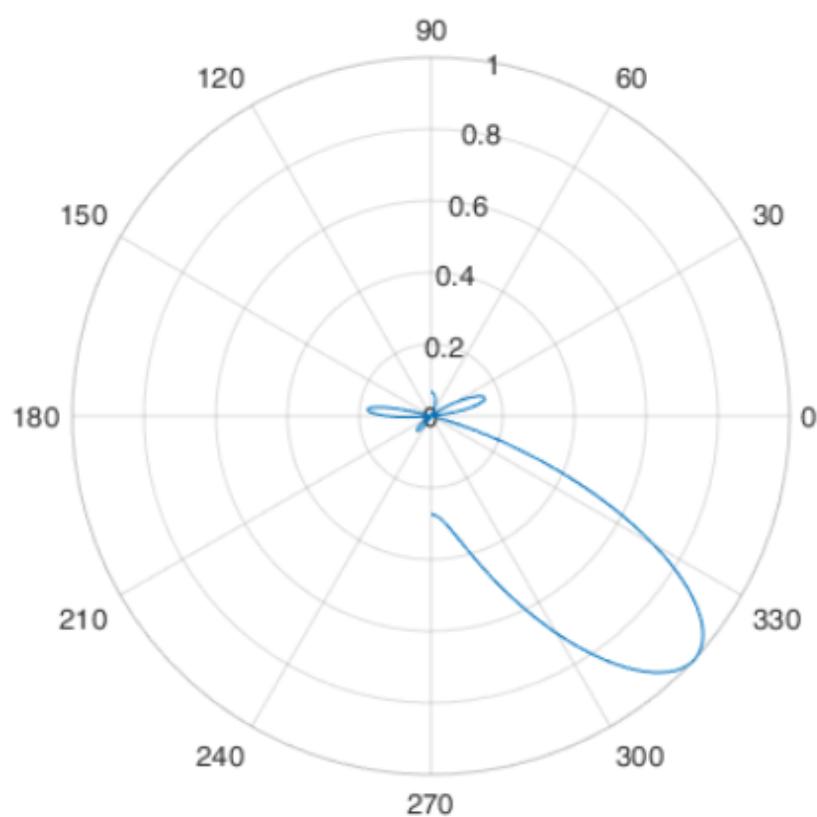
最终形成如下的声源定位和波束形成的可视化图形。



1.0x ▾

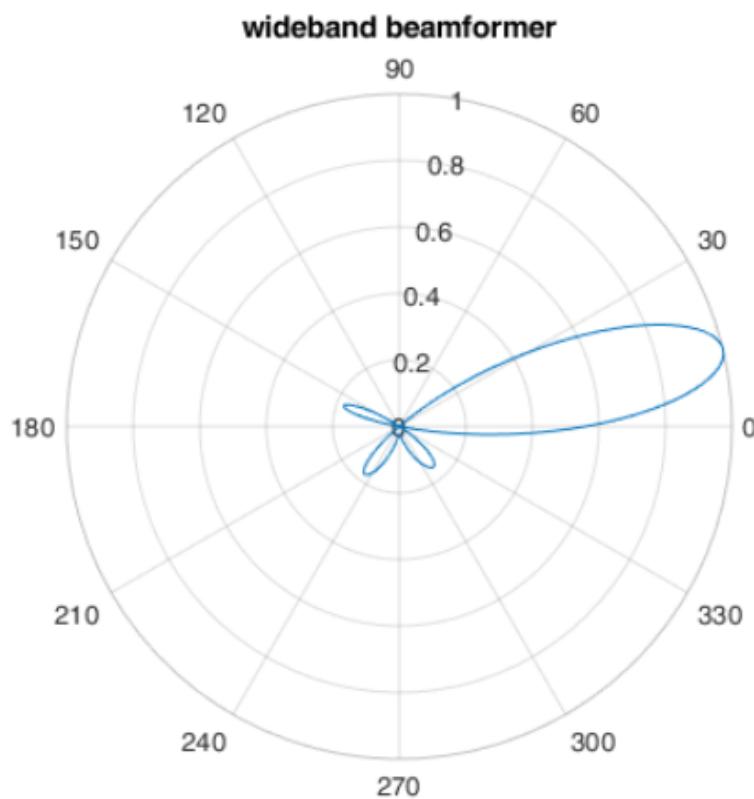
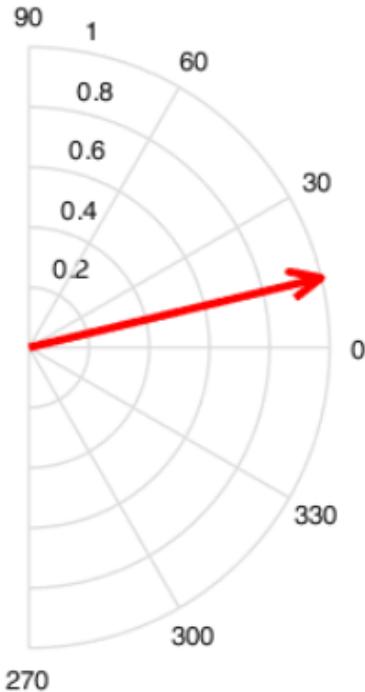


wideband beamformer



1.0x ▾





4. 整体实验验证（代码及结果展示python版本）

我们将加载通过模拟 4 麦克风阵列在空气中的传播获得的语言信号。我们还将加载漫反射噪声（在所有方向）和定向噪声（可以建模为空间中的点源）。这里的目地是将混响语音与噪声混合以生成噪声混合，并测试波束成形方法以增强语音。

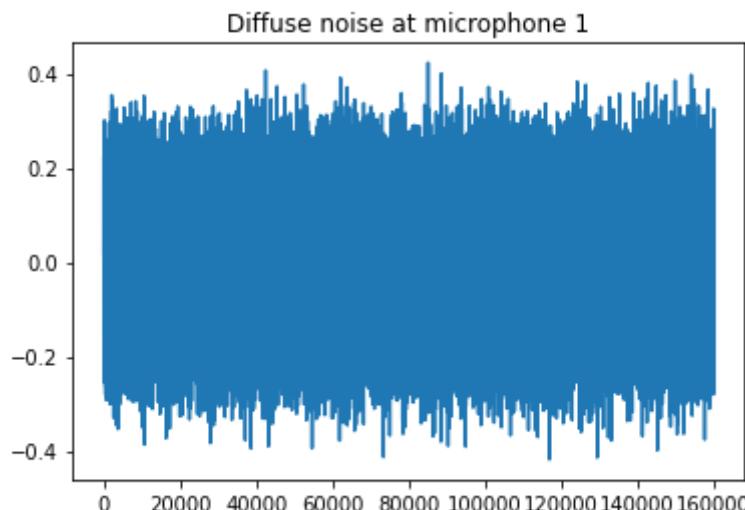
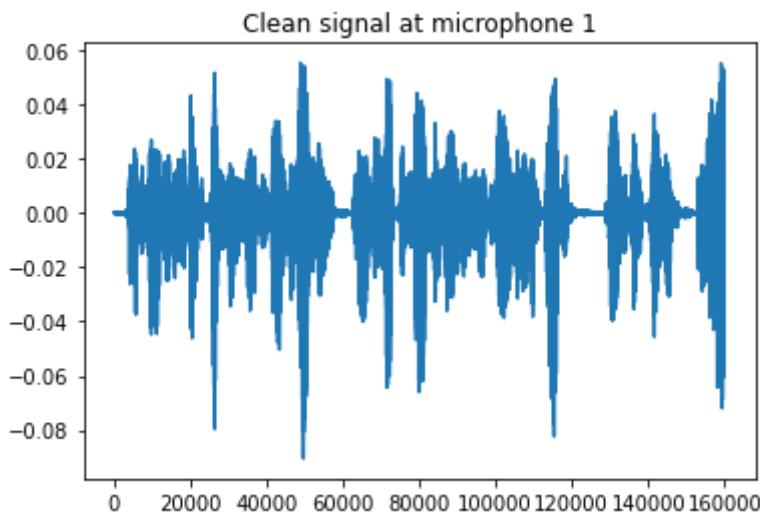
```

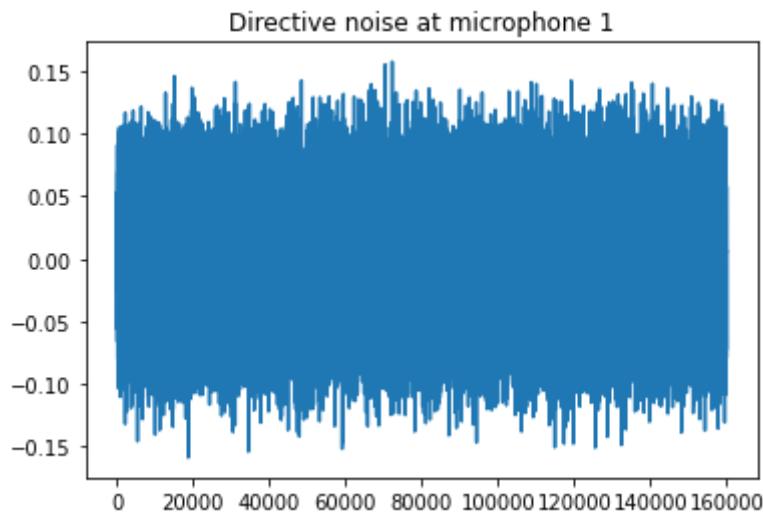
import matplotlib.pyplot as plt
from speechbrain.dataio.dataio import read_audio

xs_speech = read_audio('speech_-0.82918_0.55279_-0.082918.flac') # [time,
channels]
xs_speech = xs_speech.unsqueeze(0) # [batch, time, channels]
xs_noise_diff = read_audio('noise_diffuse.flac') # [time, channels]
xs_noise_diff = xs_noise_diff.unsqueeze(0) # [batch, time, channels]
xs_noise_loc = read_audio('noise_0.70225_-0.70225_0.11704.flac') # [time,
channels]
xs_noise_loc = xs_noise_loc.unsqueeze(0) # [batch, time, channels]
fs = 16000 # sampling rate

plt.figure(1)
plt.title('Clean signal at microphone 1')
plt.plot(xs_speech.squeeze()[:,0])
plt.figure(2)
plt.title('Diffuse noise at microphone 1')
plt.plot(xs_noise_diff.squeeze()[:,0])
plt.figure(3)
plt.title('Directive noise at microphone 1')
plt.plot(xs_noise_loc.squeeze(0)[:,0])
plt.show()
xs_noise_diff.shape

```





可以收听该声音

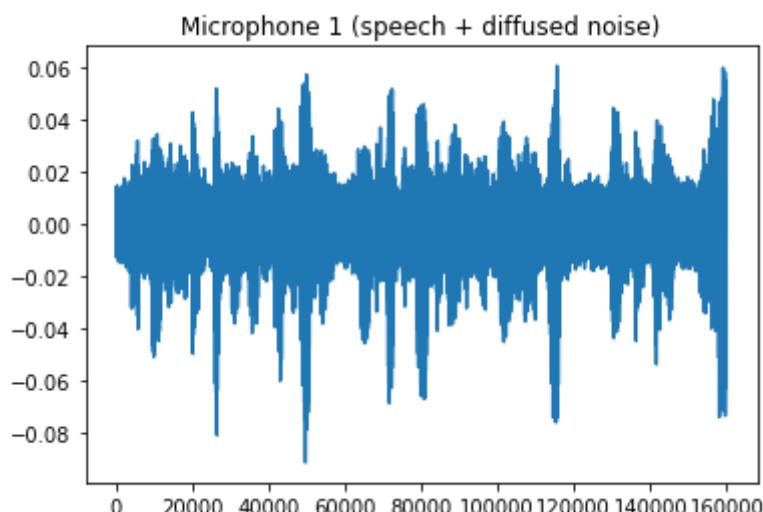
```
from IPython.display import Audio
Audio(xs_speech.squeeze()[:,0],rate=fs)
```

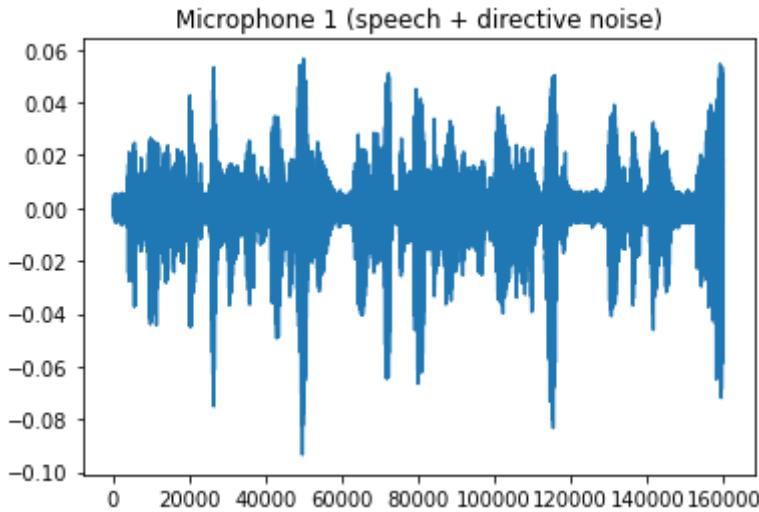
将原始语音与噪声进行混合

```
ss = xs_speech
nn_diff = 0.05 * xs_noise_diff
nn_loc = 0.05 * xs_noise_loc
xs_diffused_noise = ss + nn_diff
xs_localized_noise = ss + nn_loc
```

观察添加噪声后的语音信号

```
plt.figure(1)
plt.title('Microphone 1 (speech + diffused noise)')
plt.plot(xs_diffused_noise.squeeze()[:,0])
plt.figure(2)
plt.title('Microphone 1 (speech + directive noise)')
plt.plot(xs_localized_noise.squeeze()[:,0])
plt.show()
```





Steered – ResponsePowerwithPhaseTransform

STFT 将在频域中转换信号，然后协方差将计算每个频率区间的协方差矩阵。 *SRP – PHAT* 模块将返回到达方向。我们需要提供麦克风阵列的几何形状，在本例中是一个圆形阵列，四个麦克风均匀分布，直径为 $0.1m$ 。系统估计每个 *STFT* 帧的 DOA。在此示例中，我们使用来自方向 $x=-0.82918$ 、 $y=0.55279$ 和 $z=-0.082918$ 的声源。我们从结果中看到方向是相当准确的（由于球体离散化，有细微的差异）。另请注意，由于所有麦克风都位于 xy 平面上，因此系统无法区分正 z 轴和负 z 轴。

```

from speechbrain.dataio.dataio import read_audio
from speechbrain.processing.features import STFT
from speechbrain.processing.multi_mic import Covariance
from speechbrain.processing.multi_mic import SrpPhat

import torch

mics = torch.zeros((4,3), dtype=torch.float)
mics[0,:] = torch.FloatTensor([-0.05, -0.05, +0.00])
mics[1,:] = torch.FloatTensor([-0.05, +0.05, +0.00])
mics[2,:] = torch.FloatTensor([+0.05, +0.05, +0.00])
mics[3,:] = torch.FloatTensor([+0.05, +0.05, +0.00])

stft = STFT(sample_rate=fs)
cov = Covariance()
srpphat = SrpPhat(mics=mics)

Xs = stft(xs_diffused_noise)
XXs = cov(Xs)
doas = srpphat(XXs)

print(doas)

```

```

output:
tensor([[[[-0.8284,  0.5570,  0.0588],
           [-0.8284,  0.5570,  0.0588],
           [-0.8284,  0.5570,  0.0588],
           ...,
           [-0.8284,  0.5570,  0.0588],
           [-0.8284,  0.5570,  0.0588],
           [-0.8284,  0.5570,  0.0588]]])

```

Multiple Signal Classification

同理，针对 *MUSIC* 算法，我们采用与上述相同的方式进行 DOA 估计。

```

from speechbrain.dataio.dataio import read_audio
from speechbrain.processing.features import STFT
from speechbrain.processing.multi_mic import Covariance
from speechbrain.processing.multi_mic import Music

import torch

mics = torch.zeros((4,3), dtype=torch.float)
mics[0,:] = torch.FloatTensor([-0.05, -0.05, +0.00])
mics[1,:] = torch.FloatTensor([-0.05, +0.05, +0.00])
mics[2,:] = torch.FloatTensor([+0.05, +0.05, +0.00])
mics[3,:] = torch.FloatTensor([+0.05, +0.05, +0.00])

stft = STFT(sample_rate=fs)
cov = Covariance()
music = Music(mics=mics)

xs = stft(xs_diffused_noise)
xxs = cov(xs)
doas = music(xxs)

print(doas)

```

```

tensor([[[[-0.8271,  0.5576, -0.0702],
           [-0.8271,  0.5576, -0.0702],
           [-0.8271,  0.5576, -0.0702],
           ...,
           [-0.8271,  0.5576, -0.0702],
           [-0.8271,  0.5576, -0.0702],
           [-0.8271,  0.5576, -0.0702]]])

```

Delay-and-Sum Beamforming

STFT 将在频域中转换信号，然后协方差将计算每个频率区间的协方差矩阵。*GCC-PHAT* 模块将估计每个麦克风之间的到达时间差 (*TDOA*)，并使用此 *TDOA* 进行延迟和求和。

针对被漫反射噪声破坏的语音信号

```

from speechbrain.processing.features import STFT, ISTFT
from speechbrain.processing.multi_mic import Covariance

```

```

from speechbrain.processing.multi_mic import GccPhat
from speechbrain.processing.multi_mic import DelaySum

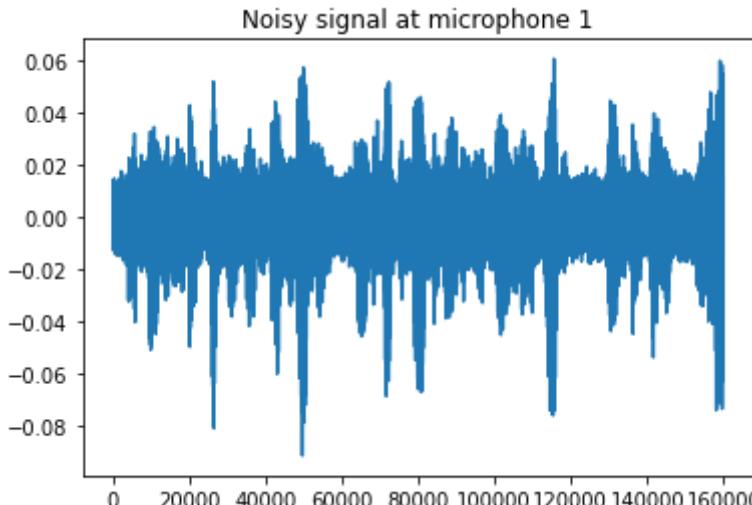
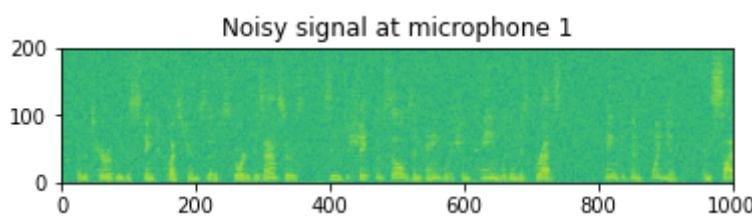
import matplotlib.pyplot as plt
import torch

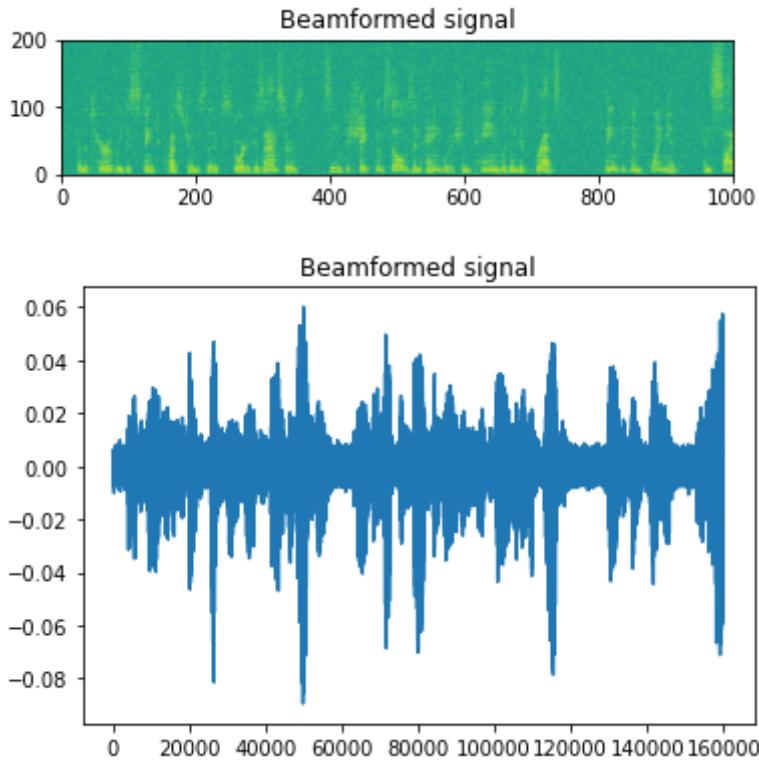
stft = STFT(sample_rate=fs)
cov = Covariance()
gccphat = GccPhat()
delaysum = DelaySum()
istft = ISTFT(sample_rate=fs)

Xs = stft(xs_diffused_noise)
XXs = cov(Xs)
tdoas = gccphat(XXs)
Ys_ds = delaysum(Xs, tdoas)
ys_ds = istft(Ys_ds)

plt.figure(1)
plt.title('Noisy signal at microphone 1')
plt.imshow(torch.transpose(torch.log(Xs[0,:,:,:,0]**2 + Xs[0,:,:,:,1]**2), 1, 0), origin="lower")
plt.figure(2)
plt.title('Noisy signal at microphone 1')
plt.plot(xs_diffused_noise.squeeze()[:,0])
plt.figure(3)
plt.title('Beamformed signal')
plt.imshow(torch.transpose(torch.log(Ys_ds[0,:,:,:,0]**2 + Ys_ds[0,:,:,:,1]**2), 1, 0), origin="lower")
plt.figure(4)
plt.title('Beamformed signal')
plt.plot(ys_ds.squeeze())
plt.show()

```





5.语音信号分离

在源分离中，目标是能够从观察到的混合信号中分离出源，该混合信号由多个源的叠加组成。我们用一个例子来证明这一点。

```

import numpy as np
import matplotlib.pyplot as plt

T = 1000
t = np.arange(0, T)
fs = 3000
f0 = 10

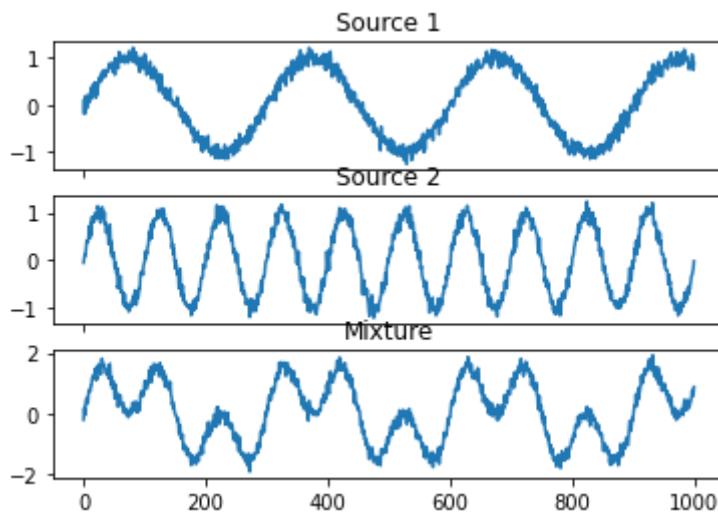
source1 = np.sin(2*np.pi*(f0/fs)*t) + 0.1*np.random.randn(T)
source2 = np.sin(2*np.pi*(3*f0/fs)*t)+ 0.1*np.random.randn(T)
mixture = source1 + source2

plt.subplot(311)
plt.plot(source1)
plt.title('Source 1')
plt.xticks(np.arange(0, 100, T), '')

plt.subplot(312)
plt.plot(source2)
plt.title('Source 2')
plt.xticks(np.arange(0, 100, T), '')

plt.subplot(313)
plt.plot(mixture)
plt.title('Mixture')
plt.show()

```



目标是从混合信号中获取源 1 和源 2。在我们的例子中，Source 1 是频率为 f_0 的噪声正弦曲线，Source 2 是频率为 $3 \cdot f_0$ 的噪声正弦曲线。

现在我们考虑一个更一般的情况，其中，源 1 是随机频率小于 $f_{threshold}$ 的正弦波，源 2 是频率大于 $f_{threshold}$ 的正弦波。让我们首先使用 *Speechbrain* 构建数据集和数据加载器。然后，我们将构建一个能够成功分离来源的模型。

```

import torch
import torch.utils.data as data_utils
import librosa.display as lrd

N = 100
f_th = 200
fs = 8000

T = 10000
t = torch.arange(0, T).unsqueeze(0)
f1 = torch.randint(5, f_th, (N, 1))
f2 = torch.randint(f_th, 400, (N, 1))
batch_size = 10

source1 = torch.sin(2*np.pi*(f1/fs)*t)
source2 = torch.sin(2*np.pi*(f2/fs)*t)
mixture = source1 + source2
N_train = 90
train_dataset = data_utils.TensorDataset(source1[:N_train], source2[:N_train],
                                         mixture[:N_train])
test_dataset = data_utils.TensorDataset(source1[N_train:], source2[N_train:],
                                         mixture[N_train:])

train_loader = data_utils.DataLoader(train_dataset, batch_size=batch_size)
test_loader = data_utils.DataLoader(test_dataset, batch_size=batch_size)

# now let's visualize the frequency spectra for the dataset
fft_size = 200

plt.figure(figsize=[20, 10], dpi=50)

plt.subplot(131)
mix_gt = mixture[N_train]

```

```

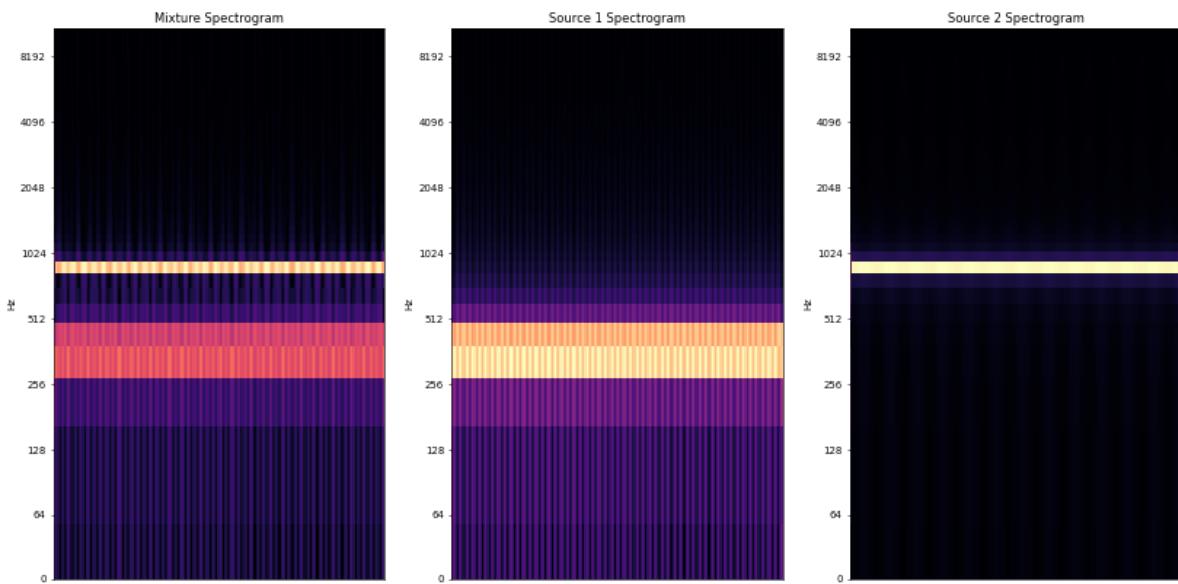
mix_spec = torch.sqrt((torch.stft(mix_gt, n_fft=fft_size)**2).sum(-1))
lrd.specshow(mix_spec.numpy(), y_axis='log')
plt.title('Mixture Spectrogram')

plt.subplot(132)
source1_gt = source1[N_train]
source1_spec = torch.sqrt((torch.stft(source1_gt, n_fft=fft_size)**2).sum(-1))
lrd.specshow(source1_spec.numpy(), y_axis='log')
plt.title('Source 1 Spectrogram')

plt.subplot(133)
source2_gt = source2[N_train]
source2_spec = torch.sqrt((torch.stft(source2_gt, n_fft=fft_size)**2).sum(-1))
lrd.specshow(source2_spec.numpy(), y_axis='log')
plt.title('Source 2 Spectrogram')

plt.show()

```



现在我们创建了数据集，我们现在可以专注于构建一个能够从混合信号中恢复原始源的模型。为此，我们将使用 *Speechbrain*。让我们首先安装 *Speechbrain*。

```

%%capture
!pip install speechbrain

```

```

import speechbrain as sb
import torch.nn as nn

# define the model
class simpleseparator(nn.Module):
    def __init__(self, fft_size, hidden_size, num_sources=2):
        super(simpleseparator, self).__init__()
        self.masking = nn.LSTM(input_size=fft_size//2 + 1, hidden_size=hidden_size,
batch_first=True, bidirectional=True)
        self.output_layer = nn.Linear(in_features=hidden_size*2,
out_features=num_sources*(fft_size//2 + 1))
        self.fft_size=fft_size
        self.num_sources = num_sources

```

```

def forward(self, inp):
    # batch x freq x time x realim
    y = torch.stft(inp, n_fft=self.fft_size)

    # batch x freq x time
    mag = torch.sqrt((y ** 2).sum(-1))
    phase = torch.atan2(y[:, :, :, 1], y[:, :, :, 0])

    # batch x time x freq
    mag = mag.permute(0, 2, 1)

    # batch x time x feature
    rnn_out = self.masking(mag)[0]

    # batch x time x (nfft*num_sources)
    lin_out = self.output_layer(rnn_out)

    # batch x time x nfft x num_sources
    lin_out = nn.functional.relu(lin_out.reshape(lin_out.size(0),
                                                lin_out.size(1), -1, self.num_sources))

    # reconstruct in time domain
    sources = []
    all_masks = []
    for n in range(self.num_sources):
        sourcehat_mask = (lin_out[:, :, :, n])
        all_masks.append(sourcehat_mask)

    # multiply with mask and magnitude
    sourcehat_dft = (sourcehat_mask * mag).permute(0, 2, 1) * torch.exp(1j * phase)

    # reconstruct in time domain with istft
    sourcehat = torch.istft(sourcehat_dft, n_fft=self.fft_size)
    sources.append(sourcehat)

    return sources, all_masks, mag

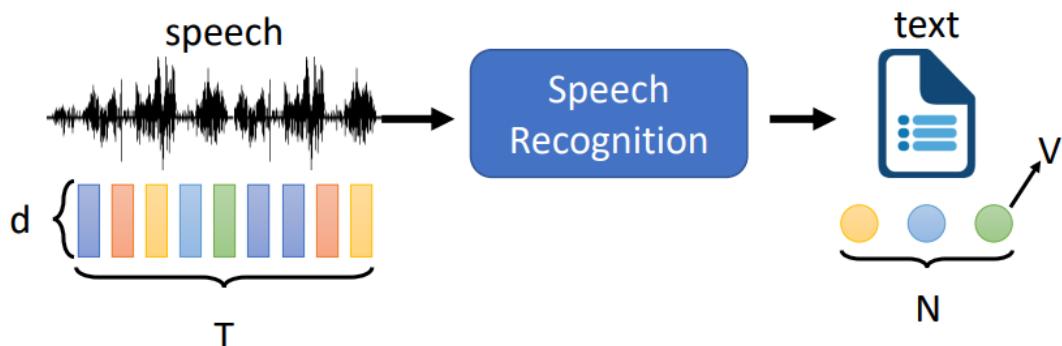
# test_forwardpass
model = simpleseparator(fft_size=fft_size, hidden_size=300)
est_sources, _, _ = model.forward(mixture[:5])

```

6.语音识别算法模块

语音识别可以看成seq2seq的过程(输入序列输出序列)，如下图，除了模型之外，确定输出的token(即输出的最小单位，如分类问题输出类别，翻译输出字符等)，语音识别常用的token包括Phoneme(类似音标)、Grapheme(类似中文里的字、英文里字母)、Word(词)、甚至有用byte的(19年ICASSP，这样就可以做语言无关的语音识别系统了)，具体优缺点见视频，此外最近的一些工作把语音识别和翻译、词性识别、语义识别等做在一起

Speech Recognition



Speech: a sequence of vector (length T, dimension d)

Text: a sequence of token (length N, V different tokens)

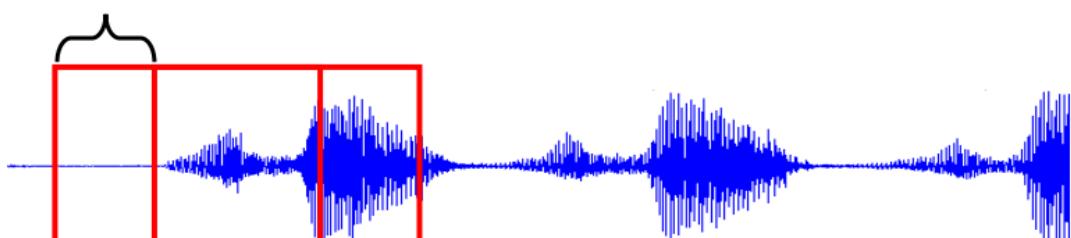
Usually $T > N$

输出是token组成的序列，而输入特征序列种类也有很多，如滤波器组输出特征、MFCC特征，或者直接用向量作为特征(向量长度为时间T和采样率的乘积)，这几个向量关系如下图

Acoustic Feature

length T, dimension d

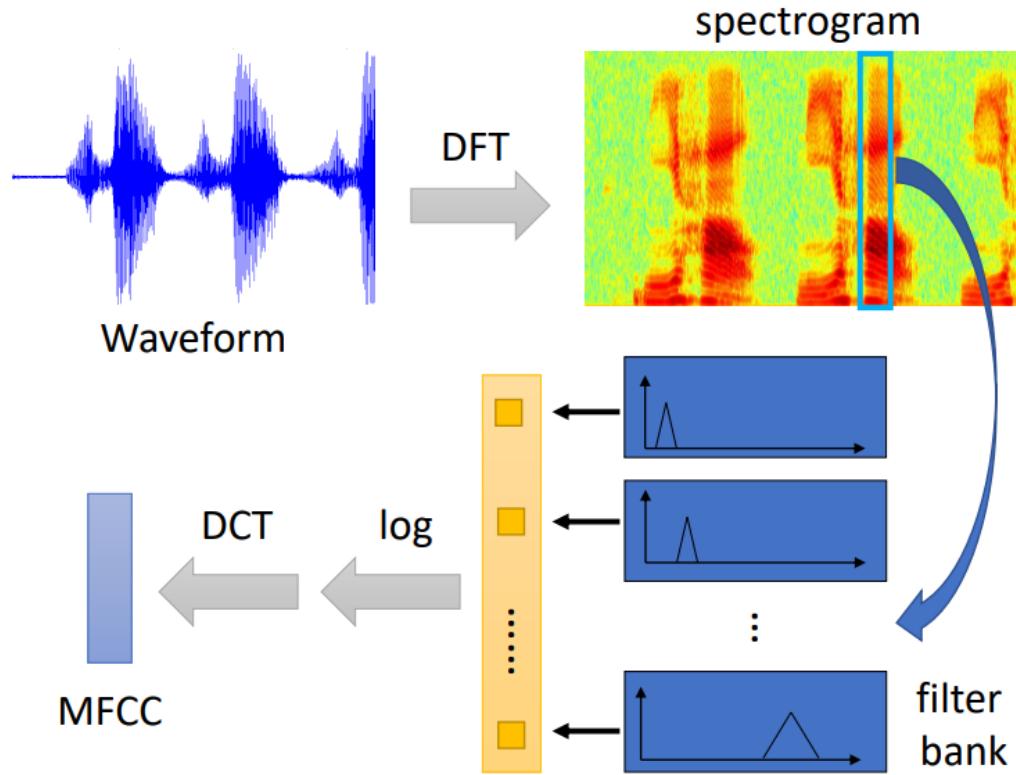
10ms 1s → 100 frames



數位語音處理 第七章
Speech Signal and Front-end Processing
[http://ocw.aca.ntu.edu.tw/ntu-ocw/ocw/cou/104S204/7](http://ocw.aca.ntu.edu.tw/ntu-ocw/ocw/cou/104S204/)

frame 400 sample points (16KHz)
 39-dim MFCC
 80-dim filter bank output

Acoustic Feature



语音识别有如下集中有监督的训练模型

Models to be introduced

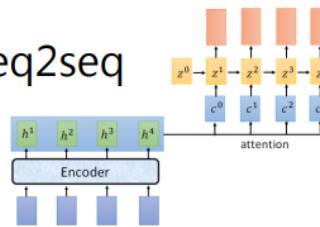
- Listen, Attend, and Spell (LAS) [Chorowski. et al., NIPS'15]
- Connectionist Temporal Classification (CTC)
[Graves, et al., ICML'06]
- RNN Transducer (RNN-T) [Graves, ICML workshop'12]
- Neural Transducer [Jaitly, et al., NIPS'16]
- Monotonic Chunkwise Attention (MoChA)
[Chiu, et al., ICLR'18]

总体来说，LAS模型可看成经典的seq2seq模型，由编码器、注意力机制、解码器组成，常用的编码器包括（RNN、LSTM、CNN、TCN（由因果卷积和空洞卷积、残差连接组成的卷积模型）、self-attention等），注意力机制包括点乘、加法Additive Attention等，解码器输出常常使用beam search算法，具体见下文，LAS缺点是不够实时（因为要做整个输入序列的attention），MoChA通过神经网络来学习移动

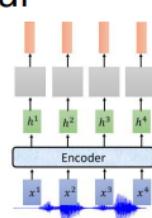
的窗口大小，此外基于CTC思想的模型如CTC、RNN-T还有一个问题是排列问题，假设解码器(时间)为6，而目标token序列长度仅为2，就需要重复输出token或者输出空，因此同一种token序列会对应很多种CTC输出排列方式，这个问题会在后文给解决方案

Summary

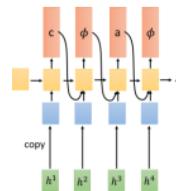
LAS: 就是 seq2seq



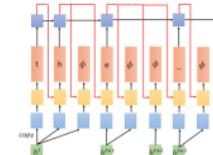
CTC: decoder 是 linear classifier 的 seq2seq



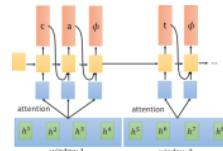
RNA: 輸入一個東西就要輸出一個東西的 seq2seq



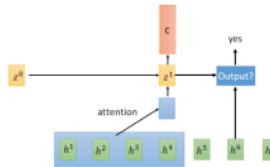
RNN-T: 輸入一個東西可以輸出多個東西的 seq2seq



Neural Transducer: 每次輸入一個 window 的 RNN-T

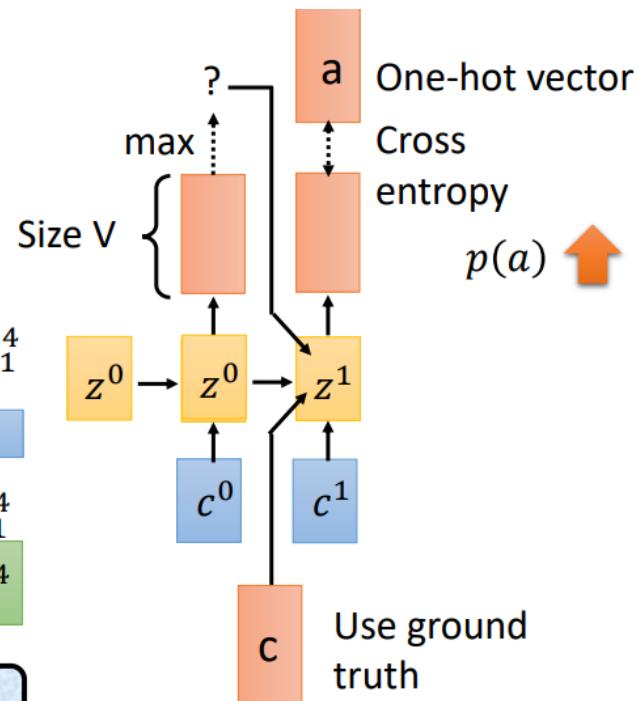
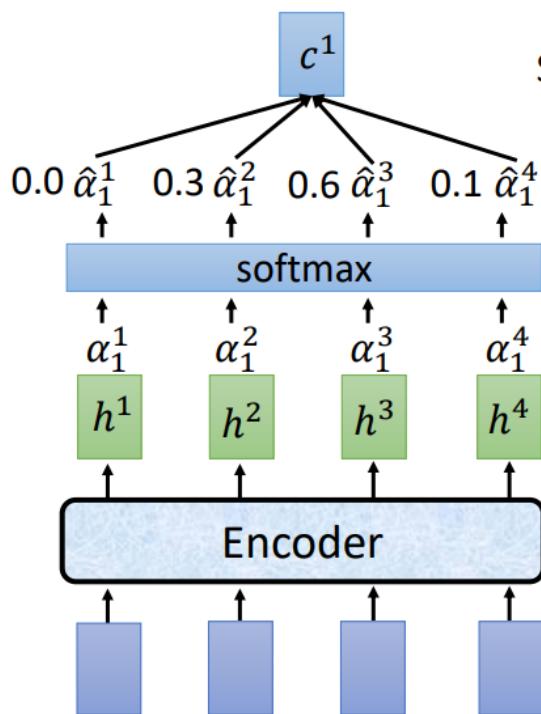


MoCha: window 移動伸縮自如的 Neural Transducer

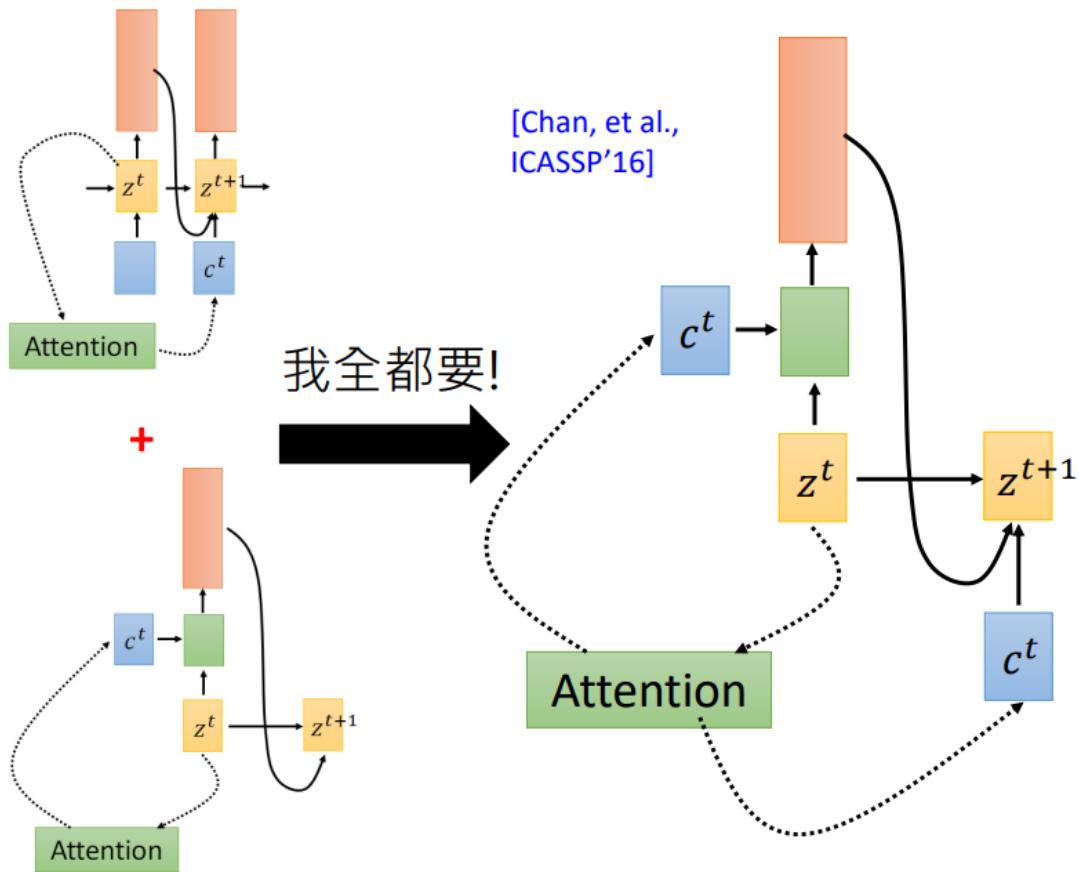


在训练的时候，往往采用Teacher Forcing的方式，即并不是使用前一步的输出作为下一步输入，而是使用前一步的正确答案输出作为下一步的输入

Training

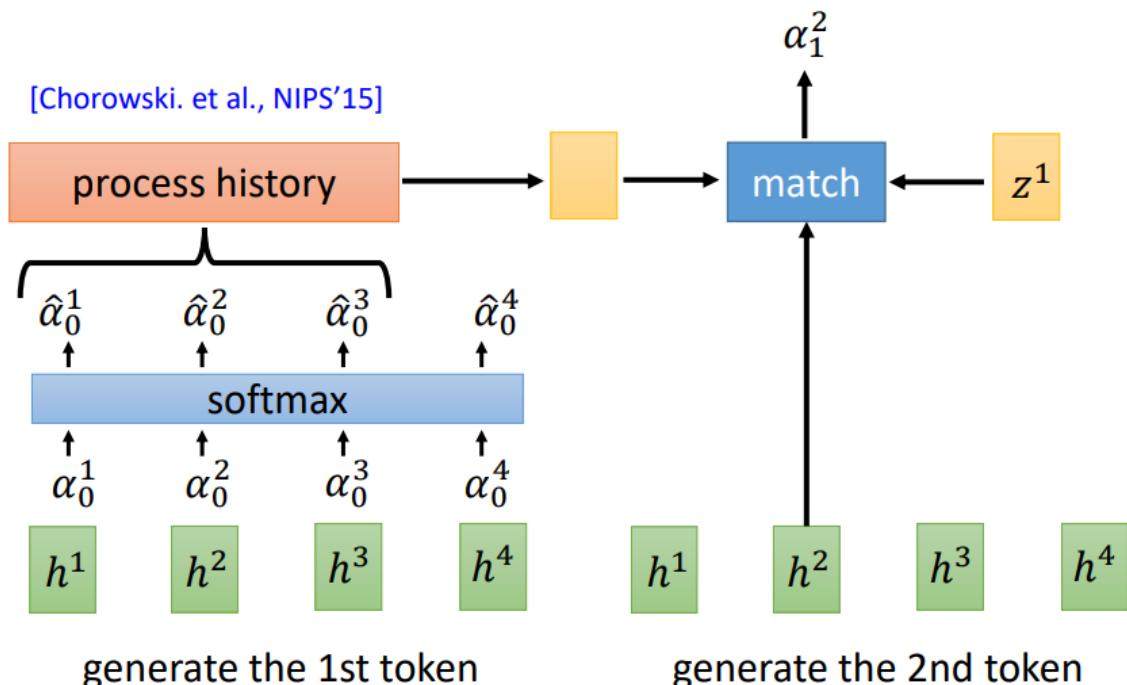


Teacher Forcing



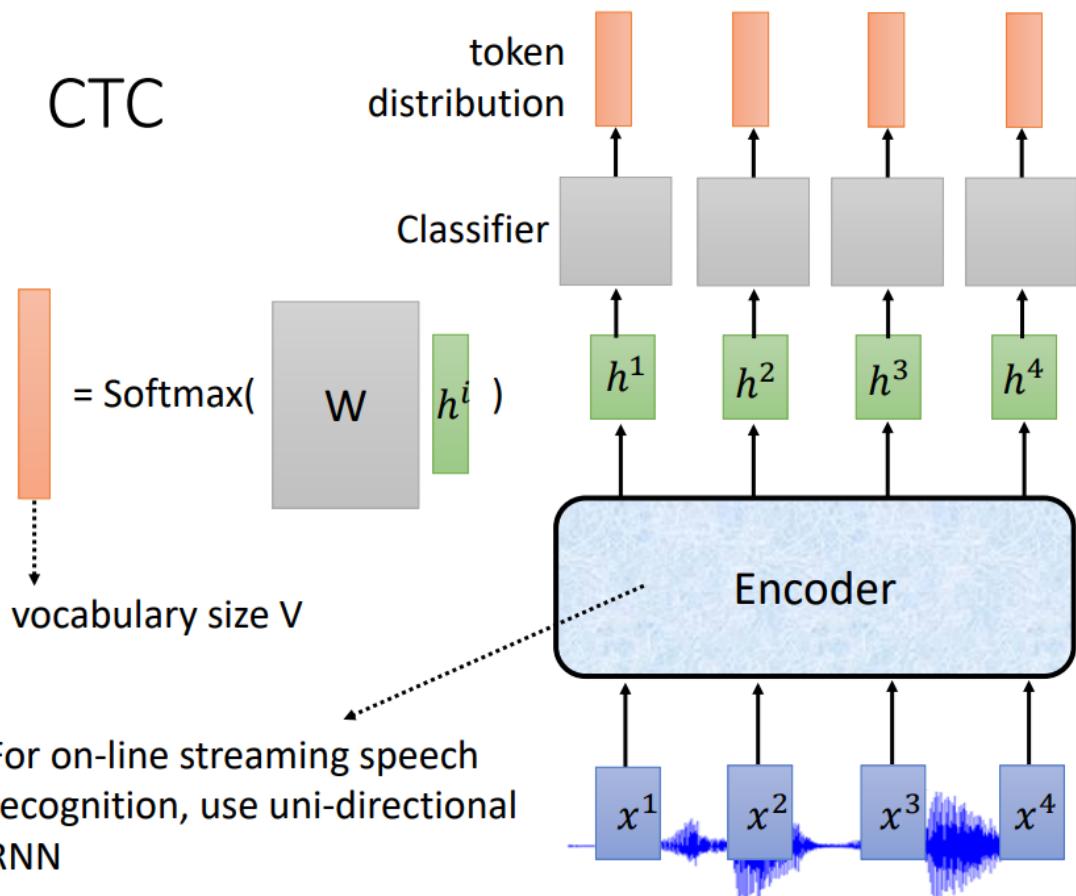
由于语音识别任务输入和输出高度一致特性(不像翻译存在倒装等情况), 又提出了Location-aware attention

Location-aware attention



CTC模型可以看成编码器和线性分类器作为解码器, 优点是实时 (编码器需要使用单向RNN、LSTM) , 可以做online的系统, 但是没有attention来结合语境信息 (这会成为我们后续的做实时识别的关键)

CTC



RNA在CTC基础上把前一时间点的输出送入下一编码器，考虑了局部语境信息

RNA

Recurrent Neural Aligner

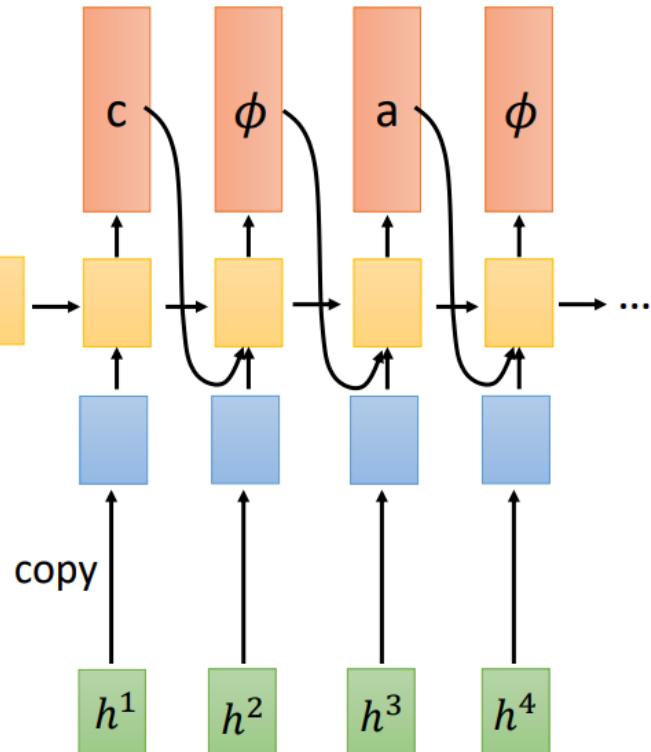
[Sak, et al., INTERSPEECH'17]

CTC Decoder:
take one vector as input,
output one token

RNA adds dependency

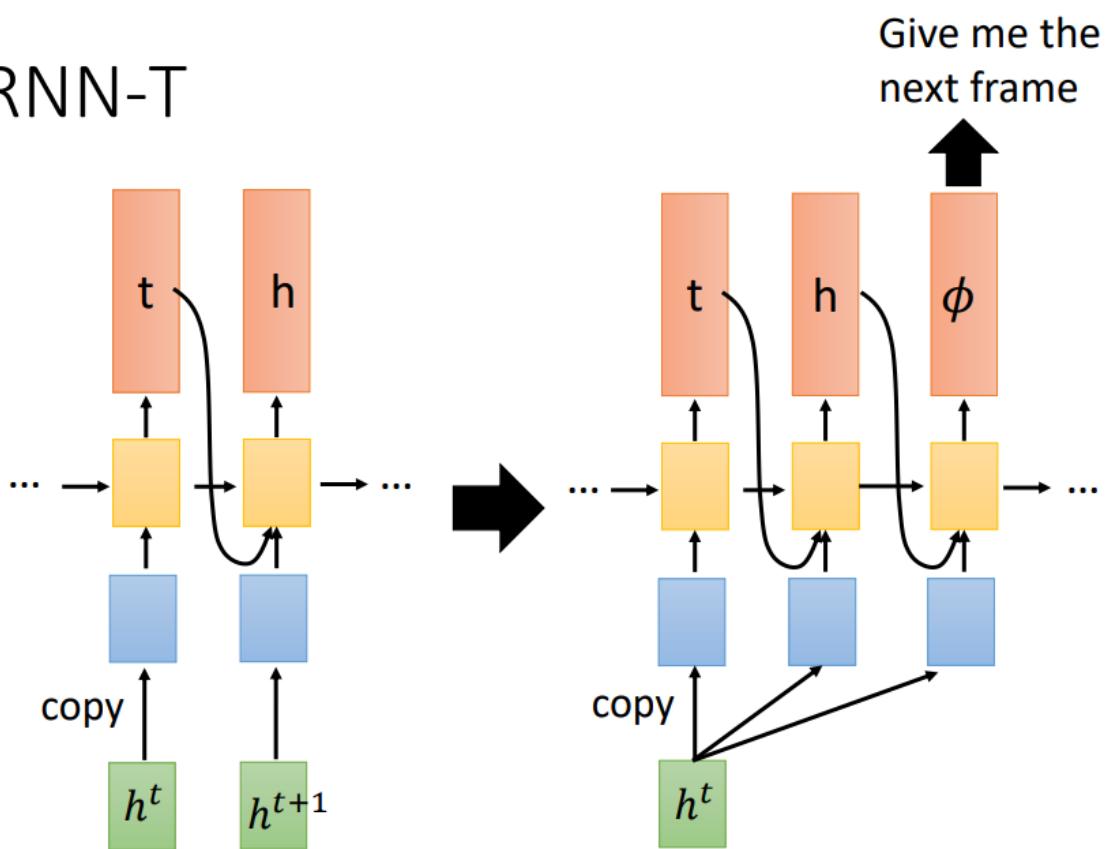
Can one vector map to
multiple tokens?

for example, "th"



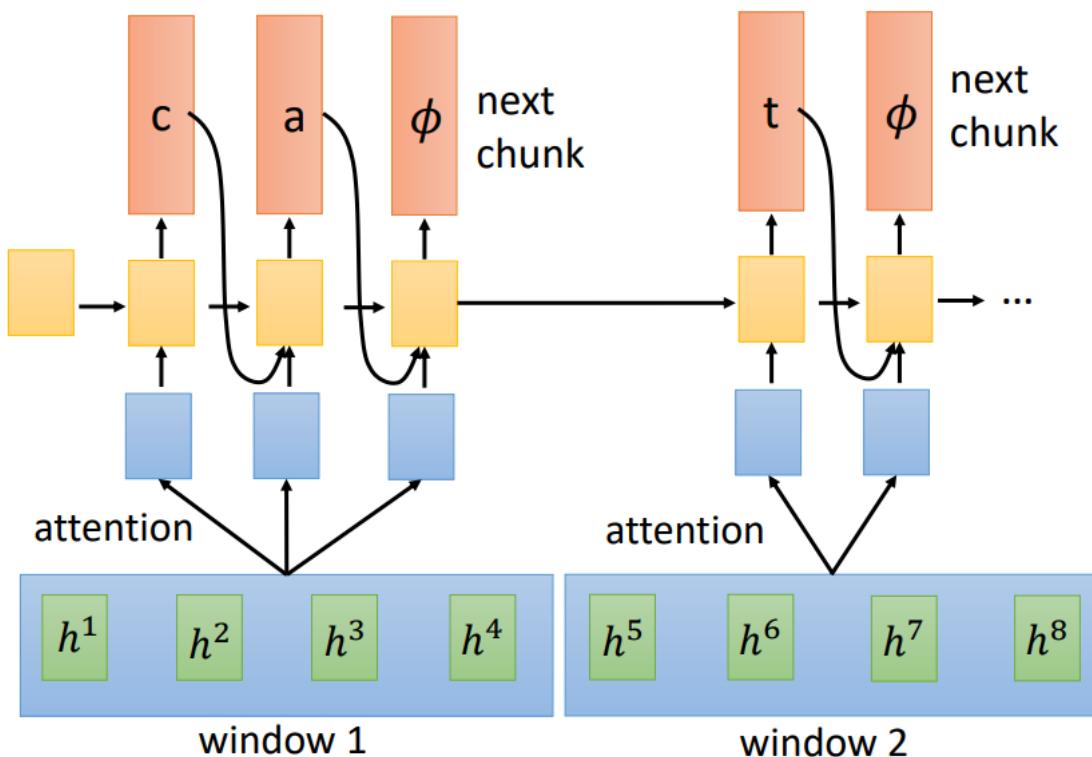
RNN-T在RNA基础上考虑了一种特殊情况，即一个输入可能产生多个token

RNN-T



Neural Transducer在RNN-T基础上一次读多个输入作为窗口，并在窗口上做注意力加权，相当于结合LAS中的attention和RNN-T一个输入可能多个输出的思想，并且局部加权可以保证实时性受影响较小，Neural Transducer窗口大小和移动方式是固定的

Neural Transducer



7. 自监督学习的语音信号特征提取

我们希望可以利用大量无标签的语音信号提取出优于MFCC、Fbank等传统用于进行语音识别的特征，所有有了自监督提取语音特征的方法。

7.1 自监督学习的核心思想

自监督训练，就是指*训练模型在一堆没有标签的样本中，通过对抗学习、聚类等方法，提取出一些高效、鲁棒、通用的特征*，然后通过少量有监督的数据微调模型，让有表征能力的“通用特征”进一步升级成具有区分性的“专属特征”。自监督训练的目的就是使用大量没有标签的数据和少量带标签的数据训练模型，以实现降低标注成本、降低迁移成本的目的。

什么是有区分性的专属特征？

专属特征，就是指可以把不同类别在高维空间分开的特征，其可以分别代表不同类别的特性，以便于更好的分类。举个例子，当我们做二分类的时候，一类样本是树木，一类样本是狗。那么网络最后得到的特征可能是：有没有耳朵、有没有四条腿、有没有树干、有没有叶子、是不是绿色等等特征。这些特征可以很好的把狗和树区分开来。所以它是这项分类任务需要的专属特征。

什么是通用的特征？

自监督训练最开始是使用无标签数据来训练模型以获得一些特征，但这儿是没有任何的监督信息，那么获得的特征跟任何的监督目的无关，也就没办法区分是狗还是树了。

“那么自监督训练得到的特征，期望其具有什么样的特性呢？”

一个好的encoder，应该是encoder得到的潜在向量（latent vector）被decoder解码后，得到新的matrix的重构误差尽可能的小。也就是说，自监督训练得到的特征应该具备目标的一些通用特征，这些特征类似乐高积木，可以通过它们很好的重构出树木、动物、车辆等等目标（但没办法用来区分树木和狗）。

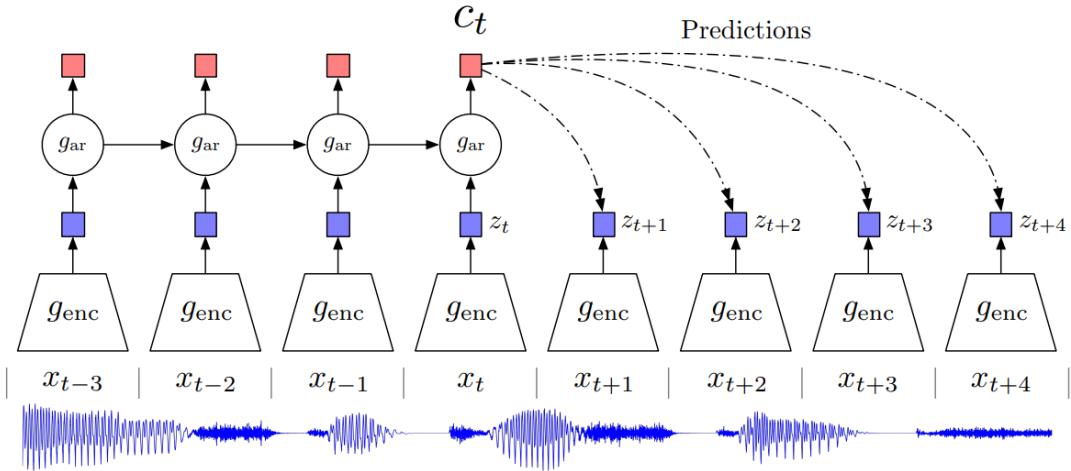
所以自监督训练的思路就是：

1. 先通过无标签的数据，训练encoder，获得一些高效、鲁棒、通用的特征表示，他们不对应任何具体的下游任务，仅仅提取一些自认为可靠的特征出来。
2. 然后再通过带标签的监督数据微调模型，将前面得到的“乐高积木”进一步组合成为“树枝”、“耳朵”、“四肢”等有区分性的专属特征。
3. 最后通过全连接层实现语音识别、声音事件检测、说话人识别、语音唤醒等等下游任务。

在无监督训练阶段，训练过程是不跟任何下游任务进行绑定的，所以它训练得到的特征可以服务于各种下游任务，这就为什么它的迁移成本更低。

7.3 CPC——语音方向自监督学习的奠基之作

CPC是谷歌2019年发布的一篇论文《**Representation Learning with Contrastive Predictive Coding**》。其提出使用对抗学习的手段，学习音频的潜在特征，并通过历史特征经过预测网络来预测未来几帧的潜在特征。如果预测结果跟真实特征非常接近，就可以说明网络得到的特征可以很好的重构出期望的特征，其重构误差很小。CPC网络结构如下所示，其由一个编码器（encoder）和一个预测模型（autoregressive mode）自回归器构成。



算法基本结构：

1. 输入，使用一维音频信号作为输入，而非FFT或者Fbank特征
2. 编码器enc，用于对一维信号进行编码,提取特征，一般是一维卷积。
3. 潜在特征z(t)，输入音频通过encoder卷积之后得到的输出。
4. 预测模型ar，将t时刻及之前的特征输入预测模型中，预测接下来几帧特征的值，一般为LSTM/RNN。
5. 上下文特征c(t)，预测模型的输出，也是最终用于微调下游任务的特征。

损失函数：

$$\mathcal{L}_N = - \mathbb{E}_X \left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)} \right]$$

$$f_k(x_{t+k}, c_t) = \exp \left(z_{t+k}^T W_k c_t \right),$$

$f_k(x)$ 是 $z(t+k)$ 和 $c(t)$ 的相似性度量函数，可以是函数形式、可以是内积、也可以是余弦距离。 $z(t+k)$ 是 t 时刻起，未来第 k 帧的潜在特征，每一个 k 都对应了一个 $f_k(x)$ 。 $x_j \in X_n$ ($n=1,2,3\dots N$) 参与loss计算的这个 N 个样本中，有1个正样本 $z(t+k)$ ，和 $N-1$ 个负样本，其中负样本是随机从其他时刻采样的值。整个损失函数的目的是使 $z(t+k)$ 跟 $[W(k)c(t)]$ 的相似度尽量的高，跟其他负样本的相似度尽量的低，这样loss才能尽可能的小。

算法思想：

由于语音具有短时平稳性，其在较短周期内是具有一定规律的(可预测)。CPC期望通过训练模型，使 t 时刻得到的预测值 $c(t)$ 经过linear层映射后，跟encoder得到的潜在特征 $z(t+1)$ 、 $z(t+2)$ 、 $z(t+3)$ 、 $z(t+4)$ 尽量的接近。即， $c(t)$ 通过一些变换后，可以很好的用来重构未来的特征 $z(t+k)$ 。

算法流程：

1. 在原始信号上面取一些时间窗口frames，然后使用一个具有编码能力的函数 (CNN)，获得 $Z(t)$ ，尺寸为 $[B \times T \times D]$ ，其中 B 是batch_size， T 是时间帧，由一维卷积的kernel size和stride决定， D 是特征维度，由卷积的channel数决定。

2. 用t时刻及之前的Z，放到预测模型（LSTM/self-attention），获得预测值c(t)，尺寸为[BxD]。
3. 利用c(t)通过linear层，预测接下来几帧的结果 $z'(t+1), z'(t+2), \dots, z'(t+k) = W(t+k)c(t)$ 。
4. 求取 $z'(t+k)$ 和 $z(t+k)$ 的相似度，代入损失函数训练己可。

算法微调：

当训练好encoder部分后，针对不同的下游任务，只需要在c(t)后面串接一个小型的微调网络（一般就是一个linear层），然后通过cross entropy或者CTC即可实现下游任务的训练。

几点说明：

1. 结合第一节所说的树木、乐高积木的例子，这里面 $z(t)$ 就是里面的树木/狗等目标， $c(t)$ 是乐高积木。乐高积木(c_t)可以通过linear操作 $z'=W*c$ ，去很好的重构出树木(z_t)，就实现了自监督训练的目的。
2. 在下游任务微调的过程中，将乐高积木 c_t 通过一个或几个linear层，就可以组合成有区分性的专属特征，然后通过交叉熵或者CTC进行训练，实现语音识别、说话人识别等下游任务。
3. CPC的效果并不如监督训练的效果好，但它引入了一个很好的自监督训练思想，才有了后面的里程碑似成果——wav2vec2.0。

7.4 wav2vec2.0

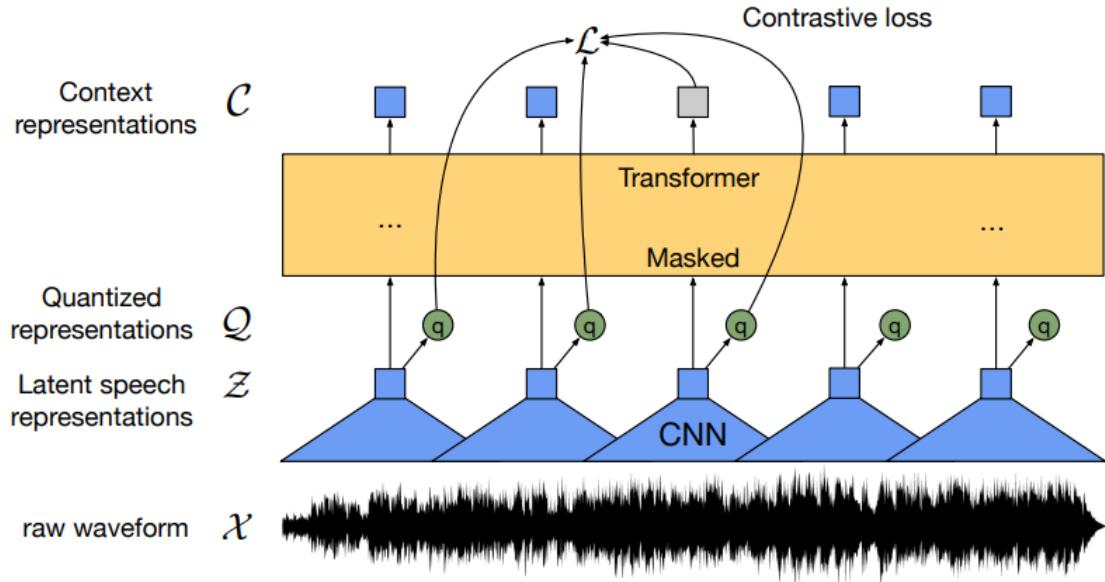
wav2vec 2.0是facebook AI实验室2020年发表的一篇论文。其在librispeech数据集上，只用了100个小时的数据就达到了原来sota的效果，并且仅使用10分钟的数据也可以达到很好的算法性能。

Table 3: TIMIT phoneme recognition accuracy in terms of phoneme error rate (PER).

	dev PER	test PER
CNN + TD-filterbanks [59]	15.6	18.0
PASE+ [47]	-	17.2
Li-GRU + fMLLR [46]	-	14.9
wav2vec [49]	12.9	14.7
vq-wav2vec [5]	9.6	11.6
This work (no LM)		
LARGE (LS-960)	7.4	8.3

wav2vec 2.0基本结构

从网络结构来看，wav2vec 2.0和CPC是非常相似的，都是由编码器和自回归网络构成，输入也都是一维的音频信号。区别就是于wav2vec 2.0使用了**transformer**代替了RNN，同时还引入了一个**乘积量化**的操作（图中 $z \rightarrow q$ 的操作）。接下来我们具体说明一下，这两个操作是什么，以及它为什么让wav2vec 2.0的效果有如此大的提升。



引入transformer结构

加入transformer结构很容易理解，现在不管是CV、ASR还是NLP，网络里面不带个transformer都不好意思发文章了。那transformer为啥这么牛逼呢，我觉得是因为以下两点：

1、从原来RNN的信息单向传递机制，升级成了self-attention的全局注意力机制。让模型 $t+k$ 时刻的输出不仅仅依赖于历史的信息，而是依赖于全局的信息。聪明的小伙伴可以就会问了，如果依赖全局信息，岂不是可以作弊，直接拿到 $z(t+k)$ 的信息了吗？后面会说明。

2、让任意两个时刻特征的距离变成了一个常量，RNN中的 t_0 时刻跟 t_{10} 时刻的信息如果要交互，必须经过 $t_1 \sim t_9$ ，才能传递过去，信息会随着传递距离增加而衰减。transformer中token2token的特性，克服了这个缺点，得整个预测模型对信息的捕获能力更强，所求特征的表征能力也就更强了。

针对transformer利用全局信息参与计算，可能会直接获得 $z(t+k)$ 信息的问题。文章的处理方法是随机加入一些mask，被mask的特征是不参与transformer算法过程的，而正样本/负样本均是从这些mask里面挑选。这样就避免了参与loss计算的样本同时又参与了 ct 的预测。

引入乘积量化结构

乘积量化操作 (product quantization) 个人认为是一个很巧妙的trick，详细内容可以根据该关键词搜索一下相关文章，这里就不详细展开了。文章里面还用到了gumble softmax的操作，其原因就是PQ量化后，特征空间变成了离散的。因为可导必连续，变量离散化了也就不可导了，没办法反向传播，所以使用了gumble softmax来近似，让loss可导。感兴趣的同学可以看看相关资料，这里也不展开了。这里只是简单介绍一下乘积量化的原理和优点。

乘积量化，是指笛卡尔积 (Cartesian product)，意思是指把原来的向量空间分解为若干个低维向量空间的笛卡尔积，并对分解得到的低维向量空间分别做量化 (quantization)。这样每个向量就能由多个低维空间的量化code组合表示。这里的量化不是将float量化成int，而是把连续空间量化成有限空间。

1、乘积量化的原理

说人话，就是

- 把原来连续的特征空间假设是 d 维，拆分成 G 个子空间 (codebook)，每个子空间维度是 d/G 。
- 然后分别在每个子空间里面聚类 (K-means)，一共获得 V 个中心和其中心特征。
- 每个类别的特征用其中心特征代替。

结果就是，原来 d 维的连续空间 (有无限种特征表达形式)，坍缩成了有限离散的空间 $[G \times V]$ ，其可能的特征种类数就只有 $G \times V$ 个。

2、乘积量化巧妙在哪儿

乘积量化操作通过将无限的特征表达空间坍缩成有限的离散空间，让特征的鲁棒性更强，不会受少量扰动的影响（只要还在某一类里面，特征都由中心特征来代替）。这个聚类过程也是一个特征提取的过程，让特征的表征能力更强了。

损失函数

wav2vec 2.0的损失函数由两部分构成，对抗性损失 \mathcal{L}_m 和多样性损失 \mathcal{L}_d 。

\mathcal{L}_m 的形式和CPC是相似的，区别在于使用余弦距离 sim 代替原来的linear映射层，同时用乘积量化的结果 qt 代替原来 zt （表征能力更强嘛）。

\mathcal{L}_d 是新引入的多样性损失，其目的是监督乘积量化中的聚类过程，期望每个中心点尽量的远。其中 G 是codebook数量， V 是聚类中心的数量， p 是某个特征在某个 (g, v) 子空间的概率值，其具体表达式就是gumble softmax的表达式。

$$\mathcal{L} = \mathcal{L}_m + \alpha \mathcal{L}_d$$

$$\mathcal{L}_m = -\log \frac{\exp(sim(\mathbf{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathbf{Q}_t} \exp(sim(\mathbf{c}_t, \tilde{\mathbf{q}})/\kappa)}$$

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^G -H(\bar{p}_g) = \frac{1}{GV} \sum_{g=1}^G \sum_{v=1}^V \bar{p}_{g,v} \log \bar{p}_{g,v}$$

$$p_{g,v} = \frac{\exp(l_{g,v} + n_v)/\tau}{\sum_{k=1}^V \exp(l_{g,k} + n_k)/\tau},$$

几点说明：

- 1、一维卷积获得的结果 z ，一方面经过mask后直接送入到了transformer中，另一方面通过乘积量化的操作获得 q ，参与后面的损失函数。
- 2、其中 ct , qt 均是来源于mask的部分，非mask的部分是用于预测mask部分的。所以 \mathcal{L}_m 中的样本组成是：transformer在第 t 个mask中心的输出 ct ——预测值，通过乘积量化在第 t 个mask中得到的聚类中心值 qt ——正样本（理论上完成聚类后，mask里面的特征因为相似，其都属于同一个类，所以就一个中心），从其他mask里面随机抽取的乘积量化结果 q ——负样本。
- 3、算法微调是在transformer后面接了一个linear层进行微调。

wav2vec2.0预训练特征提取模型结合CTC模型进行实时语音检测

trick：利用python中的pyaudio库进行对电脑麦克风的调用，使其可以实时收录声音，并将语音加载至语音识别模型中实现实时的语音识别。其中利用了huggingface中的预训练wav2vec2.0模型。

```
import pyaudio
import webbrtcvad
```

```

from wav2vec2_inference import Wave2Vec2Inference
import numpy as np
import threading
import copy
import time
from sys import exit
import contextvars
from queue import Queue


class Livewav2Vec2():
    exit_event = threading.Event()
    def __init__(self, model_name, device_name="default"):
        self.model_name = model_name
        self.device_name = device_name

    def stop(self):
        """stop the asr process"""
        Livewav2Vec2.exit_event.set()
        self.asr_input_queue.put("close")
        print("asr stopped")

    def start(self):
        """start the asr process"""
        self.asr_output_queue = Queue()
        self.asr_input_queue = Queue()
        self.asr_process = threading.Thread(target=Livewav2Vec2.asr_process,
                                             args=(self.model_name, self.asr_input_queue, self.asr_output_queue,))
        self.asr_process.start()
        time.sleep(5) # start vad after asr model is loaded
        self.vad_process = threading.Thread(target=Livewav2Vec2.vad_process,
                                             args=(self.device_name, self.asr_input_queue,))
        self.vad_process.start()

    def vad_process(device_name, asr_input_queue):
        vad = webrtcvad.Vad()
        vad.set_mode(1)

        audio = pyaudio.PyAudio()
        FORMAT = pyaudio.paInt16
        CHANNELS = 1
        RATE = 16000
        # A frame must be either 10, 20, or 30 ms in duration for webrtcvad
        FRAME_DURATION = 30
        CHUNK = int(RATE * FRAME_DURATION / 1000)
        RECORD_SECONDS = 50

        microphones = Livewav2Vec2.list_microphones(audio)
        selected_input_device_id = Livewav2Vec2.get_input_device_id(
            device_name, microphones)

        stream = audio.open(input_device_index=selected_input_device_id,
                            format=FORMAT,

```

```

        channels=CHANNELS,
        rate=RATE,
        input=True,
        frames_per_buffer=CHUNK)

frames = b''
while True:
    if LiveWav2Vec2.exit_event.is_set():
        break
    frame = stream.read(CHUNK)
    is_speech = vad.is_speech(frame, RATE)
    if is_speech:
        frames += frame
    else:
        if len(frames) > 1:
            asr_input_queue.put(frames)
        frames = b''
stream.stop_stream()
stream.close()
audio.terminate()

def asr_process(model_name, in_queue, output_queue):
    wave2vec_asr = Wave2Vec2Inference(model_name)

    print("\nlistening to your voice\n")
    while True:
        audio_frames = in_queue.get()
        if audio_frames == "close":
            break

        float64_buffer = np.frombuffer(
            audio_frames, dtype=np.int16) / 32767
        start = time.perf_counter()
        text = wave2vec_asr.buffer_to_text(float64_buffer).lower()
        inference_time = time.perf_counter() - start
        sample_length = len(float64_buffer) / 16000 # length in sec
        if text != "":
            output_queue.put([text, sample_length, inference_time])

def get_input_device_id(device_name, microphones):
    for device in microphones:
        if device_name in device[1]:
            return device[0]

def list_microphones(pyaudio_instance):
    info = pyaudio_instance.get_host_api_info_by_index(0)
    numdevices = info.get('deviceCount')

    result = []
    for i in range(0, numdevices):
        if (pyaudio_instance.get_device_info_by_host_api_device_index(i).get('maxInputChannels')) > 0:
            name =
pyaudio_instance.get_device_info_by_host_api_device_index(

```

```

        0, i).get('name')
    result += [[i, name]]
return result

def get_last_text(self):
    """returns the text, sample length and inference time in seconds."""
    return self.asr_output_queue.get()

if __name__ == "__main__":
    print("Live ASR")

asr = Livewav2Vec2("facebook/wav2vec2-large-960h-lv60-self")

asr.start()

try:
    while True:
        text,sample_length,inference_time = asr.get_last_text()

        print(f'{sample_length:.3f}s\t{inference_time:.3f}s\t{text}')

except KeyboardInterrupt:
    asr.stop()
    exit()

```

最终实现如下图所示的实时语音识别结果输出

```

listening to your voice

17.490s 6.548s  this is a test towards speech recognition
22.770s 8.966s  this is a deep learning newronawork for speech recognition

```

```

listening to your voice

4.230s 1.552s  sp allas
7.890s 2.974s  this is a test for english speech recognition
9.900s 3.793s  deep learning has received the spotlight due to its capacity to self humanlike tasks
2.190s 0.862s  through heeratical burney
2.580s 0.951s  theoretical learning
7.350s 2.754s  a survey on test generation using generative universal networks
9.090s 3.372s  this work presents a farther review concerning recent studies and test generation
1.170s 0.502s  evance
3.060s 1.138s  using generative idiversal networks
1.470s 0.600s  continuous
3.120s 1.184s  sentence
2.220s 0.855s  words
8.610s 3.193s  one way to realize that such a task is not beli
3.690s 1.362s  difference from traditional machine learning ones
5.550s 2.088s  tosget

```

