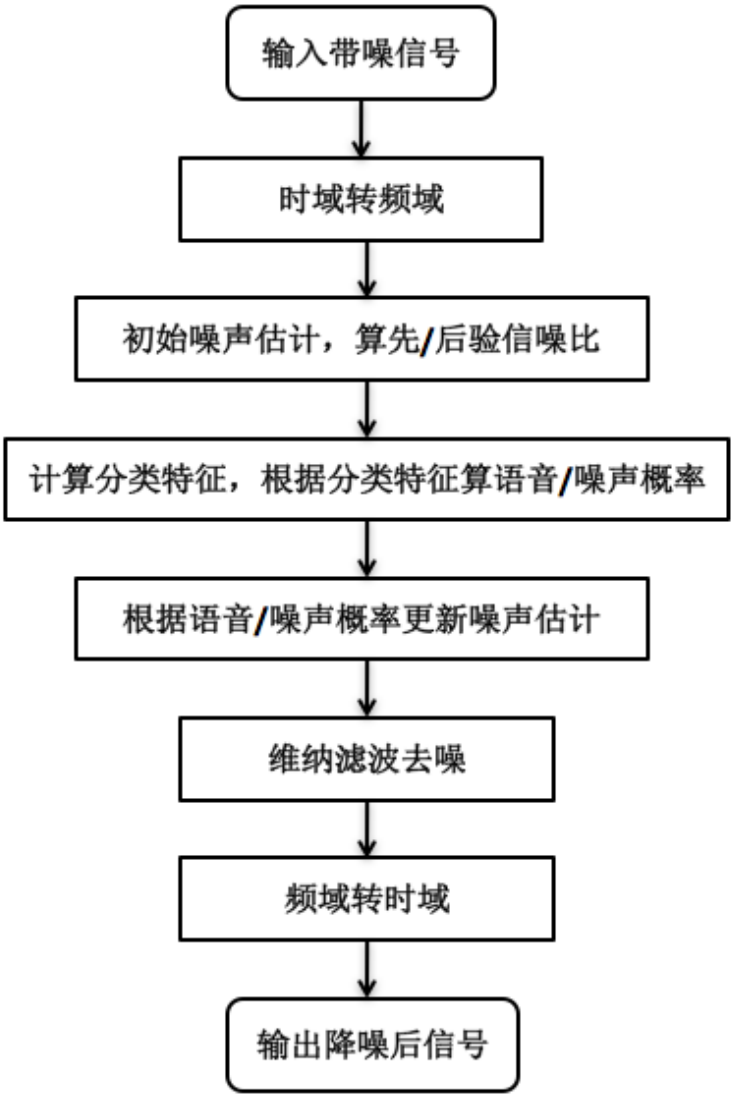
WebRTC单通道降噪

1.ANS基本处理流程

ANS的基本处理过程如下



主要分6步，具体如下：

（1） 把输入的带噪信号从时域转到频域，主要包括分帧、加窗和短时傅里叶变换(STFT)等

采用率为16kHz，10ms一帧，每帧160个采样点，加窗的目的是避免频谱泄漏。现在每帧160个点，需要补成256个点，取前一帧后96点，加上新的一帧160点，组成处理数据256点，便于做FFT。加窗后STFT的输出取129个点的值，由于这些值除了第0点和第N/2点（N=256，即第128点）点是实数外，其余点都是复数，且关于第N/2点共轭对称。

（2） 做初始噪声估计，基于估计出的噪声算先验信噪比和后验信噪比

（3） 计算分类特征，这些特征包括似然比检验(LRT)、频谱平坦度和频谱差异。根据这些特征确定语音/噪声概率，从而判定当前信号是语音还是噪声。

（4） 根据算出来的语音/噪声概率去更新噪声估计

（5） 重新计算先验后验SNR，根据先验SNR计算维纳滤波器H(k,m)，基于维纳滤波去噪。

（6） 把去噪后的信号从频域转换回时域，主要包括短时傅里叶逆变换(ISTFT)、加窗和重叠相加等。

2. 外部主函数

1. 预处理AnalyzeCore

计算分位数噪声估计、提取特征、语音/噪声最终功率

```
void WebRtcNs_AnalyzeCore(NoiseSuppressionC *self, const int16_t *speechFrame)
{
    //进160点，取前一帧后64点和本帧160，得256点的一帧处理数据self->analyzeBuf
    UpdateBuffer(speechFrame, self->blockLen, self->anaLen, self->analyzeBuf);
    //加窗 windata输出 实际上是矩阵相乘
    Windowing(self->window, self->analyzeBuf, self->anaLen, winData);
    //对windata做FFT，得到实部、虚部和magn信号
    FFT(self, winData, self->anaLen, self->magnLen, real, imag, magn);
    //进行分位数噪声估计
    NoiseEstimation(self, magn, noise);
    //计算前50帧，计算噪声模型估计，与分位数噪声估计结合，得到前50帧噪声估计
    if (self->blockInd < END_STARTUP_SHORT) {}
    //200帧内，做能量的归一化，存储到self->featureData[5]
    if (self->blockInd < END_STARTUP_LONG){}
    //计算先验信噪比和后验信噪比
    ComputeSnr(self, magn, noise, snrLocPrior, snrLocPost);
    //更新噪声参数--计算频谱平坦度、频谱模板差异特征。计算参数决策的直方图（特征的阈值和权重）
    FeatureUpdate(self, magn, updateParsFlag);
    //语言噪声概率计算，计算三个特征，得到概率加权得到最终概率，存储到self->speechProb
    SpeechNoiseProb(self, self->speechProb, snrLocPrior, snrLocPost);
    //更新噪声估计参数
    UpdateNoiseEstimate(self, magn, noise);
    //其他：记录下一帧噪声谱，更新帧数（前面）等操作
}
```

2. 降噪主程序ProcessCore

应用维纳滤波器滤除估计噪声量

```
void WebRtcNs_ProcessCore(NoiseSuppressionC *self, const int16_t *const *speechFrame,
 size_t num_bands, int16_t *const *outFrame)
{
    //进160点，取前一帧后64点和本帧160，得256点的一帧处理数据self->dataBuf
    UpdateBuffer(speechFrame[0], self->blockLen, self->anaLen, self->dataBuf);
    //加窗 windata输出 实际上是矩阵相乘
    Windowing(self->window, self->dataBuf, self->anaLen, winData);
    //处理能量为0的特殊情况（未使用），其中含有未使用的多子带处理（未使用）
    if (energy1 == 0.0){}
    //对windata做FFT，得到实部、虚部和magn信号
    FFT(self, winData, self->anaLen, self->magnLen, real, imag, magn);
    //计算先验信噪比SNR，根据先验SNR计算维纳滤波器的频率响应
    ComputeDdBasedWienerFilter(self, magn, theFilter);
    //维纳滤波器限制阈值处理(denoiseBound,1),前50帧特殊处理维纳滤波器参数，频域相乘，实际矩阵相乘
    for() {if (self->blockInd < END_STARTUP_SHORT){}}
    //记录下一帧噪声谱
    //对winData做IFFT反变换，得到输出信号
    IFFT(self, real, imag, self->magnLen, self->anaLen, winData);
```

```
    //200帧之后计算缩放因子gain，前200默认为1，计算模型提取标度fatctor--(4-51)
    if (self->gainmap == 1 && self->blockInd > END_STARTUP_LONG){}
    //加窗 windata输出 实际上是矩阵相乘，此时windata=x(n)
    Windowing(self->window, winData, self->anaLen, winData);
    //合成语音信号，进行缩放输入帧，输入256，叠加输出为160
    for()  {self->syntBuf[i] += factor * winData[i];}
    //输出处理后数据，输出叠加的160点（为前160点），实际上偏移为0（偏移未使用）
    for()  {fout[i - self->windShift] = self->syntBuf[i];}
    //叠接相加处理（将后面96点搬移到前面，用做叠接相加）
    UpdateBuffer(NULL, self->blockLen, self->anaLen, self->syntBuf);
    //输出到函数输出指针，同时进行限幅处理
    for()  {outFrame[0][i] = SPL_SAT(32767, fout[i], (-32768));}
    //计算多频带时域增益，测试代码未使用分子带（未使用）
    if (flagHB == 1)
}
```

3. 内部子函数

1. 初始噪声估计

1.分位数噪声估计

1. 参数设置

分位数噪声估计认为，即使是语音段，输入信号在某些频带分量上也可能没有信号能量，那么将某个频带上所有语音帧的能量做一个统计，设定一个分位数值，低于分位数值的认为是噪声，高于分位数值的认为是语音。

噪声估计分三个阶段，第一个阶段是前50帧，第二个阶段是51~200帧，第三个阶段是200帧以后的。50帧以后的只用分位数噪声估计法来估计噪声，而前50帧是分位数噪声估计法和噪声模型相结合，使噪声估计的更准确。NoiseEstimation(self, magn, noise)函数。

（1）分位数自然对数值：inst->lquantile，用lquantile表示，共有三组lquantile，数组大小为387（129*3）

（2）概率密度值：inst->density，用density表示，共有三组density，数组大小为387（129*3）

（3）控制计数值：inst->counter，三组不同的lquantile和density的更新由counter来控制，counter有三个整数值，每个值控制一组。counter数组的初始值基于将200一分为三，即为[66, 133, 200]。每处理完一帧counter值会加1，当值变为200时就会变为0。这样处理第二帧时counter值变为[67, 134, 0]，依此类推，也完成了0~200的遍历。

	频点0	频点1	...	频点127	频点128
第0组					
第1组					
第1组					

2. 公式计算

先定义变量：对于第i组第j个频点而言

$$delta = \begin{cases} \dfrac{40}{density[i*129+j]}, & density[i*129+j] > 1.0 \\ 40, & else \end{cases}$$

更新分位数：当频点对数谱lmagn[j] > lquantile[i*129 + j]时，表示lquantile偏小，需要增大，反之则需要减小。

$$lquantile[i*129+j] = \begin{cases} lquantile[1*129+j] + \dfrac{0.25*delta}{counter[i]+1}, Imagn[j] > lquantile[i*129+j] \\ lquantile[1*129+j] - \dfrac{0.75*delta}{counter[i]+1}, Imagn[j] < lquantile[i*129+j] \end{cases}$$

更新概率密度：当|lmagn[j] – lquantile[i*129+j]| < WIDTH(值为0.01)时，意味着当前的噪声估计比较准确，要更新概率密度。

$$density[i*129+j] = \frac{density[i*129+j]*counter[i] + 1/(2*WIDTH)}{counter[i]+1}$$

（1）当帧数小于200时，对最后一组（即第二组）的lquantile做自然指数运算，将其作为噪声估计值（noise[j]，每个频点一个值），可以看出每帧估出的噪声是不同的。——每点逐点更新

（2）当帧数大于等于200后，只有当counter数组里的值等于200时，才会将对应的组的lquantile做自然指数运算，将其作为噪声估计值。——每过66帧或者67帧噪声估计值才会更新。

```
self->quantile[i] = expf(self->lquantile[offset + i]);
noise[i] = self->quantile[i];
```

2. 噪声模型

前50帧利用分位数噪声估计法与噪声模型相结合来估计初始噪声。

（1）定义初始变量

定义四个初始表示变量：（不用前5个频点）

$$sumlogi = \sum_{i=5}^{128} ln(i), \qquad sumlogi2 = \sum_{i=5}^{128} (ln(i))^2;$$

$$sumlogmagn = \sum_{i=5}^{128} ln(magn[i]), \quad sumlogImagn = \sum_{i=5}^{128} [ln(i)*ln(magn[i])];$$

（2）表示白、粉红噪声参数

再利用上面定义的变量表示白噪声（white noise）和粉红噪声（pink noise）的参数：

$$whiteNoiseLevel = \sum_{i=0帧}^{当前帧} \frac{每帧幅度谱和}{129} * overdrive$$

$$pinkNoiseNumerator = \sum_{i=0帧}^{当前帧} max(\frac{sumlogi2*sumlogmagn - sumlogi*sumlogImagn}{sumlogi2*(129-5) - sumlogi*sumlogi}, 0)$$

$$pinkNoiseExp = \sum_{i=0帧}^{当前帧} min(max(\frac{sumlogi*sumlogmagn - (129-5)*sumlogImagn}{sumlogi2*(129-5) - sumlogi*sumlogi}, 0), 1)$$

$$parametricNum = (blockInd + 1.0) * e^{\frac{pinkNoiseNumerator}{blockInd+1.0}}$$

$$parametricExp = \frac{pinkNoiseExp}{blockInd + 1.0}$$

其中，overdrive是根据设置的降噪程度而得到的一个值（在初始化中设置）,blockInd表示当前帧的index。

（3）估计模型噪声

这样就可以利用白噪声和粉红噪声的参数来估计模型噪声。

$$parametricNoise = \begin{cases} whiteNoiseLevel, & pinkNoiseExp = 0 \\ \frac{parametricNum}{(usedBin)^{parametricExp}}, & 其他 \end{cases}$$

其中当频点id小于5时，usedBin = 5, 其他情况下usedBin = 频点id。

（4）结合噪声估计

最后根据分位数估计噪声noise和模型估计噪声parametric_noise得到最终的估计噪声了。对于每个频点j，有表达式

$$noise[j] = \frac{noise[j] * blockInd + \frac{parametricNoise * (50 - blockInd)}{blockInd + 1}}{50}$$

2.信噪比计算ComputeSnr

后验SNR指观测到的带噪信号Y和噪声功率相比之值

$$\sigma_k(m) = \frac{|Y_k(m)|^2}{|N_k(m)|^2} = \frac{Y_k(m)}{N_k(m)}$$

先验SNR是与噪声功率相关的纯净信号功率的期望值

$$\rho_k(m) = \frac{|S_k(m)|^2}{|N_k(m)|^2} = \frac{S_k(m)}{N_k(m)}$$

Y是输入含噪声的频谱，N是噪声频谱，S是指输入的纯净信号。先用判决引导法（DD），得到如下式子1，再推导得

$$\rho(k,m) = \gamma_d * \rho(k,m-1) + (1-\gamma_d) * max[\sigma_k(m) - 1.0]$$

$$\rho(k,m-1) = \frac{S(k,m-1)}{N(k,m-1)} = \frac{H(k,m-1)Y(k,m-1)}{N(k,m-1)}$$

$$\rho(k,m) = \gamma_d * H(k,m-1) * \frac{Y(m-1)}{N(m-1)} + (1-\gamma_d) * max[\sigma_k(m) - 1.0]$$

先验SNR采用上一帧估计的后验SNR（加号前）和瞬态SNR的平滑值（加号后）。

3. 特征提取SpeechNoiseProb

```
static void SpeechNoiseProb(NoiseSuppressionC *self,float
*probSpeechFinal,const float *snrLocPrior,const float *snrLocPost) {
    //计算平滑对数LRT特征
    ...
    self->featureData[3] = logLrtTimeAvgKsum;
    //LRT特征过小，取w大一些即2倍
    //平滑处理 双曲正切 映射：M(z) = 0.5*(tanh(w1z1) + 0.5) ,z1 = 特征 - 阈值参数
    indicator0 = 0.5f*(tanhf(widthPrior*(logLrtTimeAvgKsum - threshPrior0)) +
1.f);
    //更新频谱平坦度
    tmpFloat1 = self->featureData[0];
    //平滑处理 双曲正切 映射：M(z) = 0.5*(tanh(w2z2) + 0.5) ,z2 = 特征 - 阈值参数
    indicator1 = 0.5f*(tanhf((float)sgnMap*widthPrior*(threshPrior1-
tmpFloat1))+1.f);
```

```
    //更新频谱模板差异特征
    tmpFloat1 = self->featureData[4];
    //频谱模板差异特征过小，取w大一些即2倍
    //平滑处理 双曲正切 映射：M(z) = 0.5*(tanh(w3z3) + 0.5) ,z3 = 特征 - 阈值参数
    indicator2 = 0.5f * (tanhf(widthPrior * (tmpFloat1 - threshPrior2)) + 1.f);
    //综合所有特征，加权求和
    indPrior = weightIndPrior0 * indicator0 + w1 * indicator1 + w2 * indicator2;
    //平滑处理，计算之前的语音概率，做限幅处理
    self->priorSpeechProb += PRIOR_UPDATE * (indPrior - self->priorSpeechProb);
    //结合先验模型和LR因子计算最终语音概率，把对数域概率转换为正常的概率
    for (i = 0; i < self->magnLen; i++) {}
}
```

4. 特征公式计算

1.特征条件下的语音概率

（1）基于平均LR的平滑对数LRT特征估计

$$\Delta_k = \frac{exp\{\rho_k(m)\sigma_k(m)\}}{1+\rho_k(m)}$$

$$log(\Delta_k(m)) = \gamma log(\Delta_k(m-1)) + (1-\gamma)log(\Delta_k(m))$$

$$log(\Delta_k(m)) += (1-\gamma)log(\Delta_k(m)) + \gamma\{(1+\sigma_k)\frac{2\rho_k}{1+2\rho_k} - log(1+2\rho_k)\}$$

（2）频谱平坦度特征

$$F_2 = \frac{\prod_k |Y_k|^{1/N}}{\frac{1}{N}\sum_k |Y_k|}$$

（3）频谱模板差异特征

$$F_3 = \frac{varMagn - \frac{covMagnPause^2}{varPause}}{feature[5]}$$

magn 是信号幅值谱估计值，信号加噪声。

M(F)为映射函数，宜用非线性函数，如人工智能(AI)中常用做激活函数的S函数(sigmoid)和双曲正切(tanh)等，因为它们都把函数的取值范围压在了(0, 1)或者(-1, 1)范围内。tanh的取值范围是（-1, 1），所以M(F)的取值范围是（0，1）。

$$q(k,m) = \beta * q(k,m-1) + (1-\beta)[\tau_1 M(F1) + \tau_2 M(F2) + \tau_3 M(F3)]$$

2.带噪语音和特征条件下的语音概率

令H1(k, m)表示第m帧的第k个频点上是语音状态，表示H0(k, m)第m帧的第k个频点上是噪声状态，Y(k,m)表示第m帧的第k个频点上的幅度谱，{F}表示特征集合。为方便书写，简写为H1、H0、Y和F。P(H1 | Y F)表示在带噪语音和特征条件下是语音的概率，Δ(k,m)为似然比，q表示特征条件下得语音的概率，q=P(H1 | F)。

$$P(H_1|YF) = \frac{\Delta(k,m) * q}{\Delta(k,m) * q + 1 - q}$$

```
    gainPrior = (1.f - self->priorSpeechProb) / (self->priorSpeechProb +
0.0001f);
    for (i = 0; i < self->magnLen; i++) {
        invLrt = expf(-self->logLrtTimeAvg[i]);
        invLrt = gainPrior * invLrt;
        probSpeechFinal[i] = 1.f / (1.f + invLrt);
    }
```

5.噪声估计更新

在带噪语音和特征条件下语音和噪声的概率求出来后就可以去更新噪声的估计了（因为先前的估计是初始估计，不太准）

$$N(k,m) = \gamma * N(k,m-1) + (1-\gamma) * (P(H_1|YF) * N(k,m-1) + P(H_0|YF) * Y(k,m))$$

其中N(k, m)为本帧将要估计出来的噪声，N(k, m-1)为上帧已估计出来的更新过的噪声，Y(k,m-1)为本帧带噪的语音，γ为平滑系数，P(H1|YF)为是语音的概率，P(H0|YF)为是噪声的概率。

6. 降噪部分

1.维纳滤波原理

输入信号y(n)经过一个滤波器后产生一个输出信号x(n)，希望x(n)尽量逼近期望信号d(n)。这可以通过计算估计误差e(n)并使其最小化来实现，能够最小化这个估计误差的最优滤波器叫做维纳滤波器。

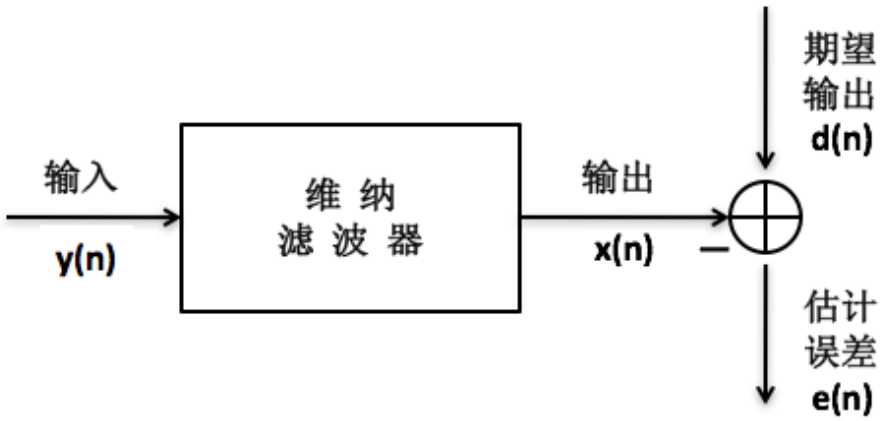


图 1

$$x(n) = h(n) * y(n)$$
$$X(w) = H(w) * Y(w)$$
$$E(w_k) = D(w_k) - X(w_k) = D(w_k) - H(w_k) * Y(w_k)$$

根据最小均方误差准则

$$J = E[|E(w_k)|^2] = E\{[D(w_k) - H(w_k) * Y(w_k)]^*[D(w_k) - H(w_k) * Y(w_k)]\}$$
$$= E[|D(w_k)|^2] - H(w_k)E[D^*(w_k)Y(w_k)] - H^*(w_k)E[D(w_k)Y^*(w_k)] + |H(w_k)|^2E[|Y(w_k)|^2]$$

令

$$P_{yy}(w_k) = E[|Y(w_k)|^2] \qquad P_{yd}(w_k) = E[Y(w_k)D(w_k)^*]$$

化简得

$$J = E[|D(w_k)|^2] - H(w_k)P_{yd}(w_k) - H^*(w_k)P_{yd}^*(w_k) + |H(w_k)|^2 P_{yy}(w_k)$$

求导得（H(wk)共轭和H(wk)是两个函数，求导不用复合求导）

$$\frac{\partial J}{\partial H(w_k)} = H^*(w_k)P_{yy}(w_k) - P_{yd}(w_k) = [H(w_k)P_{yy}(w_k) - P_{yd}^*(w_k)]^* = 0$$

$$H(w_k) = \frac{P_{yd}^*(w_k)}{P_{yy}(w_k)}$$

如果要把维纳滤波用到语音降噪上，图1中的y(n)就是带噪语音信号，x(n)就是纯净语音信号。假设n(n)表示噪声信号，如果只考虑加性噪声，则带噪语音信号、纯净语音信号和噪声信号的关系如下：y(n) = x(n) + n(n)，做傅里叶变换后的表达式如下：

$$Y(w_k) = X(w_k) + N(w_k)$$

假设噪声与语音不相关且具有零均值，则

$$P_{yd}(w_k) = E[Y(w_k)X(w_k)^*] = E[(X(w_k) + N(w_k)) * X(w_k)^*]$$
$$= E[(X(w_k) * X(w_k)^* + N(w_k) * X(w_k)^*] = E[(X(w_k) * X(w_k)^*] = P_{xx}(w_k)$$

$$P_{yy}(w_k) = E[Y(w_k)Y(w_k)^*] = E[(X(w_k) + N(w_k)) * (X^*(w_k) + N^*(w_k))]$$
$$= E[(X(w_k) * X(w_k)^* + X(w_k) * N(w_k)^* + N(w_k) * X(w_k)^* + N(w_k) * N(w_k)^*]$$
$$= E[(X(w_k) * X(w_k)^* + (N(w_k) * N(w_k)^*] = P_{xx}(w_k) + P_{nn}(w_k)$$

其中，Pxx表示纯净语音的自功率谱，Pnn表示噪声的自功率谱，由此可得

$$H(w_k) = \frac{P_{xx}(w_k)}{P_{xx}(w_k) + P_{nn}(w_k)}$$

$$\rho(w_k) = \frac{P_{xx}(w_k)}{P_{nn}(w_k)}$$

p(wk)为频点为wk时的先验信噪比（prior SNR），表示纯净语音和噪声的功率比值，后验信噪比（post SNR）表示带噪语音和噪声的功率比值，得到维纳滤波器的通用的表示形式如下，webRTC里的ANS就是基于这个表达式做语音降噪的。

$$H(w_k) = \frac{\rho(w_k)}{\rho(w_k) + 1}$$

2.基于维纳滤波降噪

因为估计出来的噪声更新了，应该是噪声估计的更准了，有必要重新算一下先验信噪比和后验信噪比。

公式如ComputerSnr一样。具体软件实现时对对H(k, m)做了一定的改进：

$$H(k, m) = \frac{\rho(k, m)}{\beta + \rho(k, m)}$$

即用β替代1。β是根据设定的降噪程度来取值的，设定的降噪程度越厉害，β取值越大，在代码中对应self->overdrive。同时对H(k, m)做一定的防越界处理，最大值是1（即不降噪），最小值也是根据设定的降噪程度来取值的，比如取0.5。算出的H(k, m)保存在数组self->smooth里。

得到H(k, m)后，降噪后的语音就可以利用表达式 S(k, m) = H(k, m)Y(k,m)求出来了。

7.信号重建

1.重建表达式

```
self->syntBuf[i] += factor * winData[i];
```

2.重建IFFT和加窗

对上面输出的S(k, m) 做IFFT反变换，代码里为winData数组，得到输出信号winData。为256点数据，但S(k, m) 为129点数据，所以后面96点为默认0值。

信号重建叠加时一般要求能量或者幅值不变，能量是幅值的平方。那些重叠的点（假设幅值为m）在上一帧中加窗时做了类余弦操作，加窗后幅值变成了m*cosθ，在当前帧中加窗时做了类正弦操作，加窗后幅值变成了m*sinθ，能量和为m2*cos2θ + m2*sin2θ, 正好等于m2(原信号的能量)，这说明只要把重叠部分相加就可以保证语音信号的连贯。这也是再做一次加窗操作并重叠相加的原因。

3.能量缩放因子

需要注意的是还有一个能量缩放因子factor。它在前200帧默认为1，后续帧按如下逻辑关系得到。能量缩放可用于帮助重建语音帧，且重建方式可增加经抑制后的语音的能量。由于噪声抑制可能会降低语音信号水平，因此需要对语音段适当放大。

$$gain = \sqrt{\frac{降噪前能量}{降噪后能量 + 1}}$$

$$factor1 = \begin{cases} 1.0, & gain <= BLIM(0.5) \\ 1.0 + 1.3 * (gain - BLIM), & gain * 1.0 + 1.3 * (gain - BLIM) <= 1 \\ 1.0/gain, & gain * 1.0 + 1.3 * (gain - BLIM) > 1 \end{cases}$$

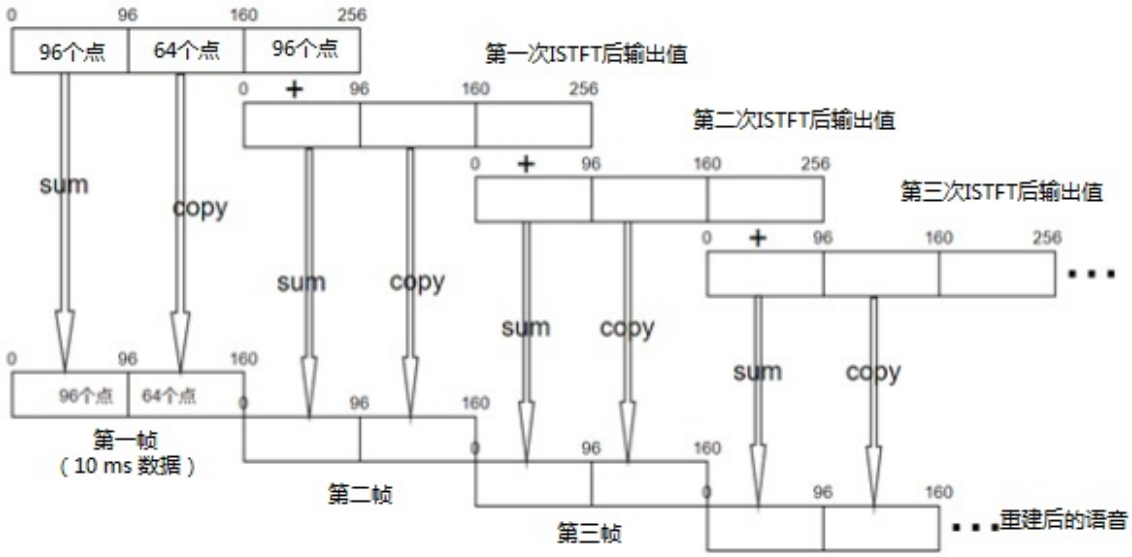
$$factor2 = \begin{cases} 1.0, & gain >= BLIM(0.5) \\ 1.0 - 0.3 * (BLIM - max(gain, self->denoiseBound)), & gain < BLIM \end{cases}$$

$$factor = self->priorSpeechProb * factor1 + (1.0 - self->priorSpeechProb) * factor2$$

self->priorSpeechProb表示当前一帧的语音概率。

4.叠接相加重建长信号

信号重建采用叠接相加，前一帧的后96点和当前帧前96点相加，输出前160点作为输出。



4.总结

对核心降噪部分简单总结下，先利用分位数噪声估计法得到噪声的初始估计并基于这个估计出来的噪声算后验信噪比和先验信噪比，然后基于先后验信噪比算似然比以及在带噪语音和特征条件下得到语音和噪声的概率，再利用得到的语音和噪声的概率去更新噪声的估计，从而得到更准确的噪声估计，最后基于更新后的噪声估计重新算后验信噪比和先验信噪比，根据基于先验信噪比的维纳滤波表达式去得到降噪后的语音。(https://www.cnblogs.com/talkaudiodev/p/15466095.html)