Delft University of Technology

Digital Audio and Speech Processing
IN4182

Course Project

Author:
Edoardo Focante (5377544)

August 1, 2021

Part I
Single Microphone Speech-Enhancement

1 Overview

The purpose of this project is to develop a single-channel speech enhancement system for noise reduction. In order to do this, the system will operate trough different blocks. First of all, the signal will be divided into frames, so that in every frame the signal can be considered stationary. The signal frames are then taken with 50% overlap and multiplied with a Hanning window of proper length before further processing. After that, the **Y** matrix is constructed, by taking the FFT of every frame as one of its column.

After that, the system will estimate the PSD of the noise and the signal, and use those values to compute a gain function. After applying the gain to every $Y_k(l)$, the de-noised signal is reconstructed by applying an IFFT on each column of **Y** and summing through overlap and add. Due to the properties of the Hanning window, no de-windowing will be necessary.

The final product of the system will hopefully be a less noisy version of the input signal. The improvement will be evaluated via different metrics and through human listening.

2 Parameters Description

The data used for experiments is the one provided with the assignment description.

The signals used have been sampled at 16 KHz while the range of stationariety of speech can be taken between 20 and 30 ms. This means that the proper frame length should be taken between 320 and 480 samples. To help with the computation of the FFT, the frame length should also be picked as a power of 2. There is no power of 2 in the desidered interval, but the closest one is 512, which has been picked by stretching the restriction a bit.

Due to the fact that the absolute value of the FFT of a real signal is symmetric with respect to the central frequency bin, only one half of the FFT bins are used, in order to reduce computations. The whole Y matrix is then reconstructed in view of this property before applying the final IFFT and the overlap and add.

In order to estimate the signal power, a Bartlett periodogram is used. This helped reduce the variance of the classic periodogram trough a smoothing over $M = 32$ Frames.

In order to estimate the noise PSD, instead, a Minimum Statistics approach is used. The noise PSD estimate for a specific frame l is chosen as the minimum signal PSD over the the last D frames. To reduce the error related to the variance of the signal PSD an exponentially smoothed version of the Periodogram is used. The expression for this smoothing is

$$\hat{P}_{YY,k}(l) = \alpha\hat{P}_{YY,k}(l-1) + (1-\alpha)|Y_k(l)|^2$$

The value of alpha is chosen as suggested in [1] such that

$$\alpha = \frac{T_{SM}f_s/R - 1}{T_{SM}f_s/R + 1} \tag{1}$$

By choosing $T_{SM} = 0.2s$, $f_s = 16Khz$ and $R = \frac{L}{2} = 256$, expression 1 gives a value of α close to 0.85. Note that according to [1] the optimal α should be optimized and then updated in order to avoid unwanted effects such as the broadening of peaks in the speech signal. However, for the sake of simplicity, α will be treated as a constant. In addition to that, by taking the minimum over different frames, the Minimum Statistics estimate is generally biased, and hence has to be corrected accordingly. However, for simplicity, the self-written code will not apply a compensation and will hence work with a slightly biased estimate.

As already mentioned, to achieve our noise PSD estimate via minimum statistics, we will take the minimum signal PSD value over D frames. With smaller windows the noise estimation will be able to track variations in the noise level faster, but we cannot reduce it too much because we still have to ensure that the window includes a frame where there is no speech, so that the minimum is originated from a frame with noise only. Because of this reason, we would like to have a window between 1 to 2 seconds long, such that the presence of a frame without speech is ensured. Over different tests, smaller values of D such as 8 and 16 proved to be the best in terms of reduction of the mean squared error (MSE) between the clean and noisy signal. However, these values don't respect our window length constraint, and in fact slightly worsen the intelligibility. The final value of D is chosen equal to 100, so that the minimum is taken over a time equivalent of $100/2 \times 32ms = 1.6s$.

After performing our Minimum Statistics estimation, we compute our Wiener gain as

$$G_k(l) = 1 - \frac{P_{NN,k}(l)}{P^B_{YY,k}(l)} \tag{2}$$

After applying the gain to $\mathbf{Y}$, we reconstruct the full matrix through its simmetry properties, and by applying IFFT and overlap and add we get our de-noised signal in the time domain.

3 Evaluation

To quantify the quality of the results the result we will use two different metrics. First we will compare the MSE before and after denoising. Since a better MSE doesn't necessarily correspond to an improvement in intelligibility, we will also use the short-time objective intelligibility (STOI) measure [2] [3]. This measure was developed with the purpose of measuring intelligibility without relying on human listeners. These type of metrics are called Objective Intelligibility Measures (OIM) and are very much needed to avoid and replace human hearing tests, which are expensive in terms of money and time. Without going too in detail, the STOI measure is based on calculating the correlation coefficient between the norm of the DFT-bins of the clean and noisy signal, averaged over different frames and different frequency octaves in the speech frequency range. Based on the results presented in [2] and [3], it can be said that this metric has a high correlation with speech intelligibility evaluated from human tests. Hence, it can be considered a reliable way to quantify intelligibility. The code for computing the STOI measure has not been self written but has been borrowed from the author of [2] and [3], C. Taal.

In addition to that, remember that our self-implemented version of the Minimum Statistics estimation doesn't take into account the bias and doesn't optimize α. In order to highlight the limitations of our code and to consolidate our results, we also decided to run a more accurate implementation of the Minimum Statistics, wrote by N. Jakovljević [4], as a reference.

4 Results

The final results will include the tests ran using two different speech samples and 4 different types of noise: Gaussian noise, Artificial non-stationary noise, Babble noise and Speech shaped noise. In addition to that, three SNR levels will be tested: -5dB, 0dB and 10DdB. Tables 1, 2 and 3 show the respective results for the different SNRs values. The tests have been carried out using both the speech signals provided, but since changing audio file led to very similar results, only the results achieved using the first speech sample are included in the report, to keep it more compact.

As we can see from all of the tables, the system is able to consistently reduce the MSE between the noisy and clean speech. Under a MSE perspective, the noise reduction works nicely for all the 4 noise types, but struggles a bit more with the non-stationary noise. This can be due to the fact that the noise power varies in the window of D frames we considered. Due to the fact that a minimum is taken, the Minimum Statistics approach is slow to track increases in noise, which could be making our estimate slightly inaccurate when compared to the cases where other types of noise are present.

By looking at the STOI metric, instead, we cannot see much improvement in the intelligibility of the de-noised signal. This is plausible, as the Wiener gain optimises the MSE, which is not necessarily correlated with the intelligibility.

A personal listening evaluation confirmed what the STOI metric conveys, as the voice in the processed signal is slightly more distinguishable from the noise but is also slightly corrupted by the de-noising process, which leads to barely noticeable improvements of the intelligibility only in the cases with the lowest SNR.

This result is not as surprising as it may seem, since many tests have already pointed out that single-microphone systems struggle in improving intelligibility, as can be read in [5] and [6].

Finally, to determine whether ignoring the bias and the optimization of α led to a worsened performance we repeated the same tests using the code provided in [4]. The results proven to be similar to the one reported in tables 1, 2 and 3 and have not been included for brevity. Overall, the usage of the more complex algorithm brought only barely significant and not consistent improvements, which were not relevant enough to justify implementing our own unbiased noise estimate.

In conclusion, the implemented system didn't prove to be too effective in increasing intelligibility, but was successful in reducing the mean squared error between the clean and noisy signal under all the considered noise scenarios.

Noise Type	Noisy MSE	Processed MSE	Noisy STOI	Processed STOI
Gaussian Noise	9.2×10^{-3}	2.6×10^{-3}	0.544	0.556
Artificial Non-stationary Noise	9.2×10^{-3}	5.0×10^{-3}	0.507	0.497
Babble Noise	9.2×10^{-3}	4.1×10^{-3}	0.511	0.506
Speech-shaped Noise	9.2×10^{-3}	2.8×10^{-3}	0.498	0.503

Table 1: Evaluation of self-implemented noise reduction for SNR=-5 dB

Noise Type	Noisy MSE	Processed MSE	Noisy STOI	Processed STOI
Gaussian Noise	2.9×10^{-3}	9.4×10^{-4}	0.647	0.656
Artificial Non-stationary Noise	2.9×10^{-3}	1.6×10^{-3}	0.619	0.616
Babble Noise	2.9×10^{-3}	1.3×10^{-3}	0.610	0.609
Speech-shaped Noise	2.9×10^{-3}	1.1×10^{-3}	0.611	0.618

Table 2: Evaluation of self-implemented noise reduction for SNR=0 dB

Noise Type	Noisy MSE	Processed MSE	Noisy STOI	Processed STOI
Gaussian Noise	2.9×10^{-4}	1.4×10^{-4}	0.844	0.850
Artificial Non-stationary Noise	2.9×10^{-4}	1.9×10^{-4}	0.824	0.826
Babble Noise	2.9×10^{-4}	1.6×10^{-4}	0.783	0.785
Speech-shaped Noise	2.9×10^{-4}	1.6×10^{-4}	0.821	0.827

Table 3: Evaluation of self-implemented noise reduction for SNR=10 dB

Appendices

A MATLAB Code

```matlab
% DASP Single Channel
clear;
clear sound;
rng(1)
[x,fs]=audioread('clean_speech.wav');
[noise,fs]=audioread('babble_noise.wav');
fplot=0; %plot flag

SNR=10^(10/10); %linear SNR
%noise=randn(length(x),1); % Gaussian Noise
noise=noise(1:length(x));
comp=0;  %More complex algorithm flag (bias compensation and optimized \alpha)

SNRr=norm(x).^2/norm(noise).^2;
noise=noise*sqrt(SNRr/SNR);
% NP=norm(noise).^2
% SP=norm(x).^2
y=x+noise;
% sound(y,fs);
%% framing


K=512;  %512 samples = 32ms with fs=16Khz
overlap=0.5;
Y=zeros(K,floor(length(y)/K));
i=1;
for l=1:K*(1-overlap):length(y)-K
    Y(:,i)=(fft(y(l:l+K-1).*(hann(K)))).';
     i=i+1;
end
L=size(Y,2);
```

```
K=K/2+1;
Y=Y(1:K,:);

%% SIGNAL PSD ESTIMATE
Pyy=zeros(K,L);
Pyy=abs(Y).^2;
Pyyb=Pyy;
M=32; % bartlett average length 0.2ms @16Khz

%Bartlett averaging
    %Transient
for i=1:M
    Pyyb(:,i)=1/i*sum(Pyy(:,1:i),2);
end
for i=M:L %Bartlett averaging
Pyyb(:,i)=1/M*sum(Pyy(:,(i-M+1):i),2);
end

%Exponential smoothing
Pyyex=Pyy;
alfa=0.85;

if alfa~=1
for i=2:L
    Pyyex(:,i)=alfa*Pyyex(:,i-1)+(1-alfa)*Pyy(:,i);
end
end

if fplot
figure;
hold on
plot(Pyyex(100,:));
plot(Pyy(100,:));
legend('Pyy smoothed','Pyy')
end

%% Minimum Statistics (No bias correction)


    D=100; %100*32/2*ms =1600 ms
    Pnn=zeros(K,L);


    for i=1:D
    Pnn(:,i)=min(Pyyex(:,1:i),[],2);
    end
    for i=D:L
    Pnn(:,i)=min(Pyyex(:,i-D+1:i),[],2);
    end



%Plotting
if fplot
figure;
hold on;
plot(Pyyex(100,:));
plot(Pnn(100,:));
legend('Pyy','Pnn')
end

```

```
%% Complete Mimimum Statistics

if comp==1
Pnn=noise_est_min_stat(Pyy,D);
end
%% GAIN CALCULATION
G=zeros(K,L);

G=1-(Pnn./Pyyb);
Yout=G.*Y;

Yout=[Yout; conj(flip(Yout(2:end-1,:)))]; %Reconstruct full Y through simmetry

%% IFFT
S=ifft(Yout);

%% Overlap and add
s=zeros(length(y),1);
i=1;
K=2*K-2;
for l=1:K*(1-overlap):length(y)-K
    s(l:l+K-1)=s(l:l+K-1)+S(:,i);
    i=i+1;
end

%% Output and Evaluation
clear sound;

% sound(s,fs);
MSEpre=mean((y - x).^2);
MSEpost=mean((s - x).^2);
STOIpre=stoi(x,y,fs);
STOIpost=stoi(x,s,fs);

function d = stoi(x, y, fs_signal)
%   d = stoi(x, y, fs_signal) returns the output of the short-time
%   objective intelligibility (STOI) measure described in [1, 2], where x
%   and y denote the clean and processed speech, respectively, with sample
%   rate fs_signal in Hz. The output d is expected to have a monotonic
%   relation with the subjective speech-intelligibility, where a higher d
%   denotes better intelligible speech. See [1, 2] for more details.
%
%   References:
%      [1] C.H.Taal, R.C.Hendriks, R.Heusdens, J.Jensen 'A Short-Time
%      Objective Intelligibility Measure for Time-Frequency Weighted Noisy
%      Speech', ICASSP 2010, Texas, Dallas.
%
    %      [2] C.H.Taal, R.C.Hendriks, R.Heusdens, J.Jensen 'An Algorithm for
    %      Intelligibility Prediction of Time-Frequency Weighted Noisy Speech',
    %      IEEE Transactions on Audio, Speech, and Language Processing, 2011.
%
%
% Copyright 2009: Delft University of Technology, Signal & Information
% Processing Lab. The software is free for non-commercial use. This program
% comes WITHOUT ANY WARRANTY.
%
%
%
% Updates:
% 2011-04-26 Using the more efficient 'taa_corr' instead of 'corr'

```

```
if length(x)~=length(y)
    error('x and y should have the same length');
end

% initialization
x           = x(:);                         % clean speech column vector
y           = y(:);                         % processed speech column vector

fs          = 10000;                        % sample rate of proposed intelligibility measure
N_frame         = 256;                          % window support
K           = 512;                          % FFT size
J           = 15;                           % Number of 1/3 octave bands
mn          = 150;                          % Center frequency of first 1/3 octave band in Hz.
H           = thirdoct(fs, K, J, mn);       % Get 1/3 octave band matrix
N           = 30;                           % Number of frames for intermediate intelligibility measure (Length analysis window)
Beta        = -15;                          % lower SDR-bound
dyn_range   = 40;                           % speech dynamic range

% resample signals if other samplerate is used than fs
if fs_signal ~= fs
    x   = resample(x, fs, fs_signal);
    y   = resample(y, fs, fs_signal);
end

% remove silent frames
[x y] = removeSilentFrames(x, y, dyn_range, N_frame, N_frame/2);

% apply 1/3 octave band TF-decomposition
x_hat           = stdft(x, N_frame, N_frame/2, K);      % apply short-time DFT to clean speech
y_hat           = stdft(y, N_frame, N_frame/2, K);      % apply short-time DFT to processed speech

x_hat       = x_hat(:, 1:(K/2+1)).';            % take clean single-sided spectrum
y_hat       = y_hat(:, 1:(K/2+1)).';            % take processed single-sided spectrum

X           = zeros(J, size(x_hat, 2));         % init memory for clean speech 1/3 octave band TF-representation
Y           = zeros(J, size(y_hat, 2));         % init memory for processed speech 1/3 octave band TF-representation

for i = 1:size(x_hat, 2)
    X(:, i)     = sqrt(H*abs(x_hat(:, i)).^2);      % apply 1/3 octave bands as described in Eq.(1) [1]
    Y(:, i)     = sqrt(H*abs(y_hat(:, i)).^2);
end

% loop al segments of length N and obtain intermediate intelligibility measure for all TF-regions
d_interm        = zeros(J, length(N:size(X, 2)));
   % init memory for intermediate intelligibility measure
c           = 10^(-Beta/20);                                                        % constant for clipping procedure

for m = N:size(X, 2)
    X_seg       = X(:, (m-N+1):m);
```

```
        % region with length N of clean TF-units for all j
    Y_seg       = Y(:, (m-N+1):m);      % region with length N of processed TF-units for all j
    alpha   = sqrt(sum(X_seg.^2, 2)./sum(Y_seg.^2, 2));                     % obtain scale factor for normalizing processed TF-region for all j
    aY_seg      = Y_seg.*repmat(alpha, [1 N]);      % obtain \alpha*Y_j(n) from Eq.(2) [1]
    for j = 1:J
        Y_prime                 = min(aY_seg(j, :), X_seg(j, :)+X_seg(j, :)*c); % apply clipping from Eq.(3)
        d_interm(j, m-N+1)  = taa_corr(X_seg(j, :).', Y_prime(:));             % obtain correlation coeffecient from Eq.(4) [1]
    end
end

d = mean(d_interm(:));                                                      % combine all intermediate intelligibility measures as in Eq.(4) [1]

%%
function  [A cf] = thirdoct(fs, N_fft, numBands, mn)
%   [A CF] = THIRDOCT(FS, N_FFT, NUMBANDS, MN) returns 1/3 octave band matrix
%   inputs:
%       FS:         samplerate
%       N_FFT:      FFT size
%       NUMBANDS:   number of bands
%       MN:         center frequency of first 1/3 octave band
%   outputs:
%       A:          octave band matrix
%       CF:         center frequencies

f               = linspace(0, fs, N_fft+1);
f               = f(1:(N_fft/2+1));
k               = 0:(numBands-1);
cf              = 2.^(k/3)*mn;
fl              = sqrt((2.^(k/3)*mn).*2.^((k-1)/3)*mn);
fr              = sqrt((2.^(k/3)*mn).*2.^((k+1)/3)*mn);
A               = zeros(numBands, length(f));

for i = 1:(length(cf))
    [a b]                   = min((f-fl(i)).^2);
    fl(i)                   = f(b);
    fl_ii                   = b;

        [a b]                   = min((f-fr(i)).^2);
    fr(i)                   = f(b);
    fr_ii                   = b;
    A(i,fl_ii:(fr_ii-1))        = 1;
end

rnk         = sum(A, 2);
numBands        = find((rnk(2:end)>=rnk(1:(end-1))) & (rnk(2:end)~=0)~=0, 1, 'last' )+1;
A           = A(1:numBands, :);
cf          = cf(1:numBands);

%%
function x_stdft = stdft(x, N, K, N_fft)
%   X_STDFT = X_STDFT(X, N, K, N_FFT) returns the short-time
%       hanning-windowed dft of X with frame-size N, overlap K and DFT size
%   N_FFT. The columns and rows of X_STDFT denote the frame-index and
%   dft-bin index, respectively.
```

```
frames        = 1:K:(length(x)-N);
x_stdft       = zeros(length(frames), N_fft);

w             = hanning(N);
x             = x(:);

for i = 1:length(frames)
    ii                = frames(i):(frames(i)+N-1);
        x_stdft(i, :)    = fft(x(ii).*w, N_fft);
end

%%
function [x_sil y_sil] = removeSilentFrames(x, y, range, N, K)
%   [X_SIL Y_SIL] = REMOVESILENTFRAMES(X, Y, RANGE, N, K) X and Y
%   are segmented with frame-length N and overlap K, where the maximum energy
%   of all frames of X is determined, say X_MAX. X_SIL and Y_SIL are the
%   reconstructed signals, excluding the frames, where the energy of a frame
%   of X is smaller than X_MAX-RANGE

x       = x(:);
y       = y(:);

frames  = 1:K:(length(x)-N);
w       = hanning(N);
msk     = zeros(size(frames));

for j = 1:length(frames)
    jj       = frames(j):(frames(j)+N-1);
    msk(j)       = 20*log10(norm(x(jj).*w)./sqrt(N));
end

msk     = (msk-max(msk)+range)>0;
count   = 1;

x_sil   = zeros(size(x));
y_sil   = zeros(size(y));

for j = 1:length(frames)
    if msk(j)
        jj_i              = frames(j):(frames(j)+N-1);
        jj_o              = frames(count):(frames(count)+N-1);
        x_sil(jj_o)       = x_sil(jj_o) + x(jj_i).*w;
        y_sil(jj_o)       = y_sil(jj_o) + y(jj_i).*w;
        count             = count+1;
    end
end

x_sil = x_sil(1:jj_o(end));
y_sil = y_sil(1:jj_o(end));

%%
function rho = taa_corr(x, y)
%   RHO = TAA_CORR(X, Y) Returns correlation coeffecient between column
%   vectors x and y. Gives same results as 'corr' from statistics toolbox.
xn      = x-mean(x);
xn      = xn/sqrt(sum(xn.^2));
yn      = y-mean(y);
yn      = yn/sqrt(sum(yn.^2));
rho     = sum(xn.*yn);
```

```matlab

function [sigma_N_2, P, alpha] = noise_est_min_stat(abs_Y_2, param)
% Estimate noise spectrum using minimum statistics as proposed in [1]
%
% Inputs:
%   abs_Y_2 - noisy signal PSD (one column per frame)
%   param - parameters of the algorithm
%        + D - number of frames in analysing window for minimum search
%        + V - number of frames in analysing subwindow
%        + tshift - frame shift in seconds
%        + tdecay - decay time from signal peak to the noise level
%        + alpha_max - maximum value of smoothing factor
%        + alpha_min - minimum value of smoothing factor
%        + alpha_c_min - minimum value of alpha_c
%        + min_y_energy - minimum energy
%        + inv_q_eq_max - maximum value of Q_eq^-1
%        + beta_max / maximum value of beta
%        + noise_slope_max_table - for comparison of Q^-1
%        + a_v - factor used in B_c calcluation
%
% Outputs:
%  sigma_N_2 - estimated noise PSD
%  P - smoothed periodogram
%  alpha - optimal smoothing factor

% [1] Martin, R.
%     "Noise power spectral density estimation based on optimal smoothing
%      and minimum statistics"
%      IEEE Transactions on Speech and Audio Processing, 2001, 9, 504-512

% author:  jakovnik@gmail.com
% date: 2016/08/23

if or(nargin < 1, nargin > 2)
  error ('invalid number of input arguments');
end
%% Set default values
if nargin == 1
  param = [];
end
D = 96;
if isfield(param,'D')
  D = param.D;
end
V = 12;
if isfield(param,'V')
  V = param.V;
end
tshift = 16e-3;
if isfield(param,'tshift')
  tshift = param.tshift;
end
tdecay = 64e-3;
if isfield(param,'tdecay')
  tdecay = param.tdecay;
end
alpha_max = 0.96;
if isfield(param,'alpha_max')
  alpha_max = param.alpha_max;
end
alpha_min = 0.30;
```

```
if isfield(param,'alpha_min')
  alpha_min = param.alpha_min;
end
alpha_c_min = 0.7;
if isfield(param,'alpha_c_min')
  alpha_c_min = param.alpha_c_min;
end
min_y_energy = 1e-9;
if isfield(param,'min_y_energy')
  min_y_energy = param.min_y_energy;
end
inv_q_eq_max = 1/2;
if isfield(param,'inv_q_eq_max')
  inv_q_eq_max = param.inv_q_eq_max;
end
beta_max = 0.8;
if isfield(param,'beta_max')
  beta_max = param.beta_max;
end
noise_slope_max_table = [0.03 8;
                         0.05 4;
                         0.06 2];
if isfield(param,'noise_slope_max_table')
  noise_slope_max_table = param.noise_slope_max_table;
end
a_v = 2.12;
if isfield(param,'a_v')
  a_v = param.a_v;
end
MAX_VALUE = 1e9;

U = round(D/V);
M_D = mean_of_the_min_table3(D);
M_V = mean_of_the_min_table3(V);
[L,n_frames] = size(abs_Y_2);
snr_exp = -tshift/tdecay;

% initialize
sigma_N_2 = zeros(L,n_frames);
P = zeros(L,n_frames);
alpha = zeros(L,n_frames);
% calculation
buff_idx = 1;
subwc = V;
P_prev = abs_Y_2(:,1);
alpha_c_prev = 1; %set initial value (it is suppose noise at begin)
sigma_N_2_prev = abs_Y_2(:,1);
sigma_N_2(:,1) = abs_Y_2(:,1);
mean_P = abs_Y_2(:,1);
mean_P_2 = mean_P.^2;
actmin_buff = MAX_VALUE*ones(L,U);
actmin = MAX_VALUE*ones(L,1);
actmin_sub = MAX_VALUE*ones(L,1);
lmin_flag = zeros(L,1);
for lambda = 1:n_frames
  % compute the smoothing parameter alpha
  y_energy = sum(abs_Y_2(:,lambda));
  p_energy = sum(P_prev);
  if y_energy == 0
    y_energy = min_y_energy;
  end
```

```
alpha_c = 1/(1 + (p_energy/y_energy - 1)^2);                    %eq.  9
alpha_c = 0.7*alpha_c_prev + 0.3*max(alpha_c,alpha_c_min);        %eq. 10
alpha_c_prev = alpha_c;
alpha_opt = alpha_max * alpha_c ./...
  (1 + (P_prev./sigma_N_2_prev - 1).^2);                          %eq. 11
snr = p_energy/sum(sigma_N_2_prev);
alpha_opt = max(alpha_opt,min(alpha_min,snr^snr_exp));             %eq.+12
alpha(:,lambda) = alpha_opt;
% compute smoothed power P(\lambda,k)
P(:,lambda) = alpha_opt .* P_prev + (1 - alpha_opt)...
                 .*abs_Y_2(:,lambda);                              %eq.  4
% compute bias correction
beta = min(alpha_opt.^2,beta_max);
mean_P = beta.*mean_P + (1-beta).*P(:,lambda);                     %eq. 20
mean_P_2 = beta.*mean_P_2 + (1-beta).*P(:,lambda).^2;              %eq. 21
var_P = mean_P_2 - mean_P.^2;                                      %eq. 22
inv_Q_eq = var_P./(2*sigma_N_2_prev.^2);                           %eq. 23
%inv_Q_eq = max(min(inv_Q_eq,INV_Q_EQ_MAX),INV_Q_EQ_MIN/lambda);
inv_Q_eq = max(min(inv_Q_eq,inv_q_eq_max),1/20/lambda);
tilda_Qeq_D = (1./inv_Q_eq - 2*M_D)/(1-M_D);                       %eq. 16
tilda_Qeq_V = (1./inv_Q_eq - 2*M_V)/(1-M_V);                       %eq. 16
B_min = 1 + 2*(D-1)./tilda_Qeq_D;                                  %eq. 17
B_min_sub = 1 + 2*(V-1)./tilda_Qeq_V;                              %eq. 17
% compute inv Q_eq
inv_Q_eq = mean(inv_Q_eq);
% minimum search
B_c = 1 + a_v*sqrt(inv_Q_eq);
P_x_B_min_x_B_c = P(:,lambda).*B_min*B_c;
k_mode = P_x_B_min_x_B_c < actmin;
actmin(k_mode) = P_x_B_min_x_B_c(k_mode);
actmin_sub(k_mode) = P(k_mode,lambda).*B_min_sub(k_mode)*B_c;
if subwc == V
  lmin_flag(k_mode) = 0;
  actmin_buff(:,buff_idx) = actmin;
  if buff_idx == U
    buff_idx = 1;
  else
    buff_idx = buff_idx + 1;
  end
  P_min_u = min(actmin_buff,[],2);
  p = find(inv_Q_eq < noise_slope_max_table(:,1));
  if isempty(p)
    noise_slope_max = 1.2;
  else
    noise_slope_max = noise_slope_max_table(p(1),2);
  end
  p = and(and(lmin_flag, actmin_sub < noise_slope_max.*P_min_u),...
          actmin_sub > P_min_u);
  if any(p)
    P_min_u(p) = actmin_sub(p);
    actmin(p) = actmin_sub(p);
  end
  lmin_flag = zeros(L,1);
  subwc = 1;
  actmin(:) = MAX_VALUE;
  actmin_sub(:) = MAX_VALUE;
  sigma_N_2(:,lambda) = P_min_u;
else
  if subwc > 1
    lmin_flag(k_mode) = 1;
    sigma_N_2(:,lambda) = min(actmin_sub,P_min_u);
```

```
            P_min_u = sigma_N_2(:,lambda);
          else
            sigma_N_2(:,lambda) = min(actmin_sub,P_min_u);
          end
          subwc = subwc + 1;
        end
        P_prev = P(:,lambda);
        sigma_N_2_prev = sigma_N_2(:,lambda);
    end
    end

    function [M_D, H_D] = mean_of_the_min_table3(D)
    DMH = [   1 0.000 0.000;    2 0.260 0.150;    5 0.480 0.480;    8 0.580 0.780; ...
             10 0.610 0.980;   15 0.668 1.550;   20 0.705 2.000;   30 0.762 2.300; ...
             40 0.800 2.520;   60 0.841 2.900;   80 0.865 3.250; 120 0.890 4.000; ...
            140 0.900 4.100; 160 0.910 4.100];

      p = find(DMH(:,1) == D);
      if ~isempty(p)
        M_D = DMH(p,2);
        H_D = DMH(p,3);
      else
        p = find(DMH(:,1) < D);
        if isempty(p)
          error('invalid value for: D');
        end
        p = p(end);
        if p > size(DMH,1)-1;
          error('invalid value for: D');
        end
        delta_p_rat = (D - DMH(p,1))/(DMH(p+1,1)-DMH(p,1));
        M_D = DMH(p,2) + (DMH(p+1,2)-DMH(p,2))*delta_p_rat;
        H_D = DMH(p,3) + (DMH(p+1,3)-DMH(p,3))*delta_p_rat;
      end
    end
```

References

[1] R. Martin. Noise power spectral density estimation based on optimal smoothing and minimum statistics. *IEEE Transactions on Speech and Audio Processing*, 9(5):504–512, 2001.

[2] Cees H. Taal, Richard C. Hendriks, Richard Heusdens, and Jesper Jensen. A short-time objective intelligibility measure for time-frequency weighted noisy speech. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4214–4217, 2010.

[3] Cees H. Taal, Richard C. Hendriks, Richard Heusdens, and Jesper Jensen. An algorithm for intelligibility prediction of time–frequency weighted noisy speech. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(7):2125–2136, 2011.

[4] Niksa Jakovljevic, Dragisa Miskovic, and Željen Trpovski. Evaluation of noise estimation algorithms based on minimum statistics and signal to noise ratio. pages 1–4, 11 2016.

[5] Yi Hu and Philipos C. Loizou. A comparative intelligibility study of speech enhancement algorithms. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 4, pages IV–561–IV–564, 2007.

[6] C. Ludvigsen, C. Elberling, and G. Keidser. Evaluation of a noise reduction method–comparison between observed scores and scores predicted from sti. *Scandinavian audiology. Supplementum*, 38:50–5, 1993.