<Project 1> Speech Synthesis and Perception with Envelope Cue

Author

Lifer (XD

Introduction

Background

The cochlea plays an important role in the process of human speech reception and recognition. The Characteristic frequency of the basement membrane decreases from bottom to top. For humans, this resonance frequency ranges from approximately 20-20000 Hz, which is the normal hearing frequency range for humans. Received physical signals are converted into acoustic signals by the cochlea and transmitted to the nerves. The task people attempt to do is to simulate this process to create the cochlear implant.

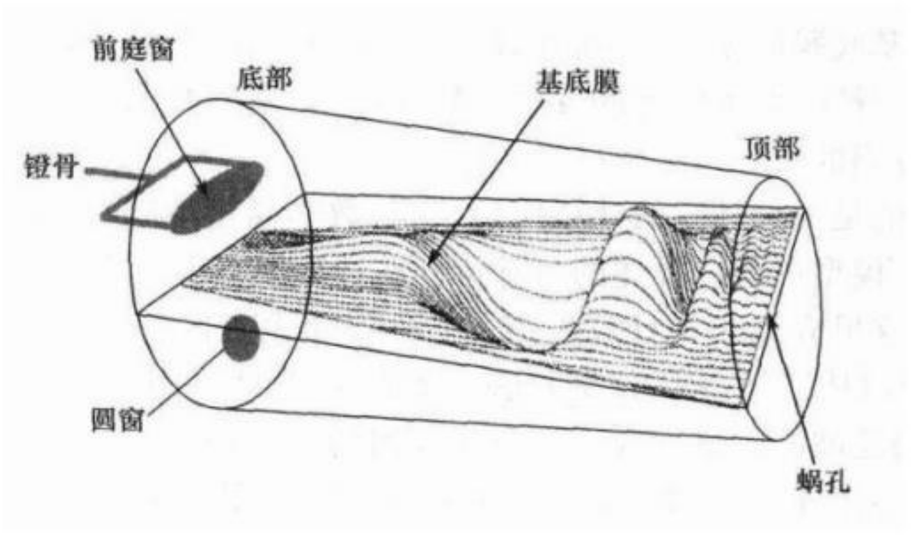


Figure 1 Cochlea Structure Display

In the realization process of the cochlear implant, the core part is the step of using the microprocessor to process the speech signal. There are currently two types of schemes for the processing of human voice signals. The first is a scheme based on spectral features for parameter extraction, and the second is a scheme based on band-pass filter banks. The following is a specific exploration based on the filter bank scheme.

According to formal research, the speech waveform can be decomposed, by Hilbert transform, into the product of the signal envelope and fine-structure waves. The envelope is most important for speech reception, which contains low-frequency; while the fine structure is most important for pitch perception and sound localization, which contains high frequency. From A. Nejat Ince's book, most parts of speech information are carried in the envelope, while the fine structure signals, part with the formant, maintain spectral details, identifying properties of sounds. The similarity between fine structure and the spectrum determines the retention of sound characteristics, especially for voiceless plosives. (1992, pp.55-57) The human voice has a uniquely more continuous spectral character. Thus, we need to build a Tone-vocoder to reduce the distortion of the vocal signal during transmission.

Tone-vocoder System

As Figure 1 shows, in this Tone-vocoder system, the original sound signal is segmented through a series of band-pass filters, and each segment is subjected to full-wave rectification, and then passed through a low-pass filter, to extract the envelope of each segment. Multiply these envelopes with a sine wave signal, which frequencies are at the frequency midpoint of the corresponding segmented passband. Finally, these loaded signals are synthesized, and then energy is normalized.

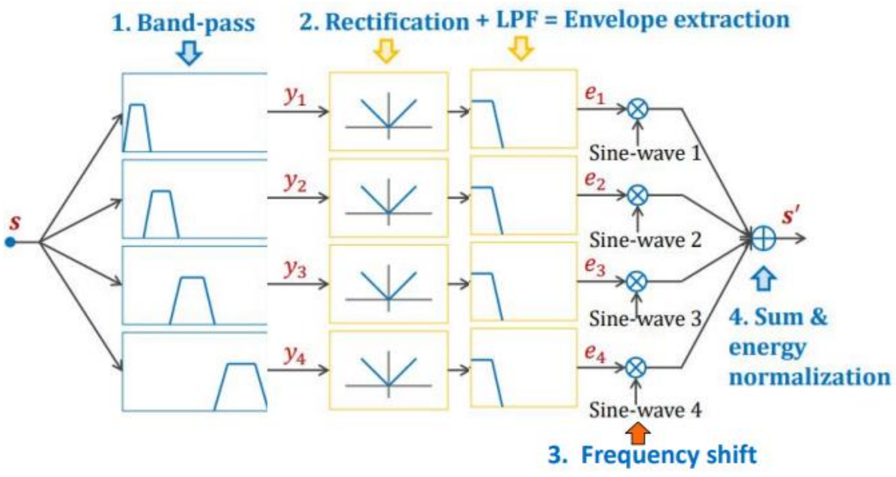


Figure 2 Tone-vocoder System

As for the detailed process, we first choose the suitable frequency range to divide. According to Steeneken and Houtgast, the center frequencies of the human voice range from 125 Hz to 8 kHz (1999, pp.109-123). Thus, choose frequencies ranging from 200 Hz to 7000 Hz, then use band-pass filters to segment these signals.

Implementation and analysis

Basic task introduction

For the first two tasks, we use two different methods to segment and compare their performance:

1) Equal frequency interval division between 200 Hz and 7000 Hz.

2) Equally divide the cochlea length and take the mapping of the corresponding length to set the frequency.

For the first method, divide the frequency band into N segments according to the numerical average of the frequency. Use these frequency bands' upper and lower limits as parameters of the buffer function to generate the corresponding bandpass filter, and filter the original signal to generate N-segment sub-signals

For the second method, we know the mapping relations of the cochlea length and resonance frequency are

$$f = 165.4 * (10^{0.06d} - 1)$$

In the above process, change different N, which is the segment number, to find better effects.

Next, abstract the envelope of the sub-signals. First, full-wave rectification is performed, which is the abs of the signals. Then, pass the signal through a low pass filter, which is generated by the function "butter". In this way, the envelope of each sub-signal is obtained. These envelope signals are multiplied by the sinusoidal carrier signal at the midpoint frequency of the corresponding frequency band, and the signals are synthesized, and then energy normalized.

In this process, change the cut-off frequency of the LPF to find better effects.

For the last two tasks, generate an SSN and add it to the original sound signal, then regard it as the original signal, and repeat the above operations. How to generate the SSN is mentioned in Lab 5, so it will not be repeated here.

Implementation of basic task

For Task 1, fix the cut-off frequency of the LPF in the process of abstracting the envelope on 50 Hz, then segment the signal by means of frequency equalization using different N. And for Task 2, fix the segment number N as 4 and change the cut-off frequency.

In this process, the functions we need to implement in the code are filter design, envelope extraction, generating SSN at a certain SNR, energy normalization, and file manipulation. In addition, encapsulation and integration of code functions are also required.

1. Parameter preposition and quick adjustment

```
% Veriables Setting

SSN = 'N';  % Add SNR 'Y' or not 'N'
Mode = 1;  % Equal frequency: 0 | Average cochlea length: 1
N_sub = 2;  % Number of segments
N_BPF = 4;  % BPF order
Cf = 50;  % Cut-off frequence of LPF
N_LPF = 4;  % LPF order
```

Figure 3 Parameter preposition and quick adjustment

2. Tone Vocoder design

```
% Batch Extract Envelopes
function E = Envelope(Y, fs, N1, cf)
[b, a] = butter(N1, cf / (fs / 2), "low");
sizeY = size(Y);
m1 = sizeY(1);
m2 = sizeY(2);
Y = abs(Y);
E = zeros(m1, m2);
for n = 1 : m1
    E(n, :) = filter(b, a, Y(n, :));
end
end
```

Figure 4 Batch Extract Envelopes

```
% Carrier loaded
loaded = zeros(Nmax, length(sound_read));
gap = (dmax - dmin) / Nmax;
for j = 1 : Nmax
    for k = 1 : length(sound_read)
        loaded(j, k) = envelopes(j, k) .* cos(2*pi * alter(dmin + (j-1/2)*gap) * (k-1) / fs);
    end
end
```

Figure 5 Carrier loaded - Easy to change the carrier frequency algorithm

For Task 3, we add SSN to the original signal and repeat the operations of Task 1. So as for Task 4 relevant to Task 2.

In this process, the function we need to implement in the code is adding the SSN signal at a certain SNR.

```
% Generate SSN
function Out = getSSN(sound_read, fs)
sound_read = sound_read';
nfft = 512;
noverlap = nfft / 2;
Windows = hamming(nfft);
% Returns the power spectral density
[Pxx,w] = pwelch(repmat(sound_read, 1, 10), Windows, noverlap, nfft, fs);
% Generate filter
b = fir2(3000, w/(fs / 2), sqrt(Pxx / max(Pxx)));
% Generate white noise
N = length(sound_read);
noise = 1 - 2 * rand(1, N + length(b) - 1);
% Generate SSN
ssn = filter(b, 1, noise);
ssn = ssn(length(b) : end);
% Adjust the energy of SSN
ssn = ssn / norm(ssn) * norm(sound_read) * 10 ^ (1 / 4);
% Addition sound_read with ssn
Out = sound_read + ssn;
% Normalize out to sound_read
Out = Out / norm(Out) * norm(sound_read);
end
```

Figure 6 Generate SSN

And to response the setting.

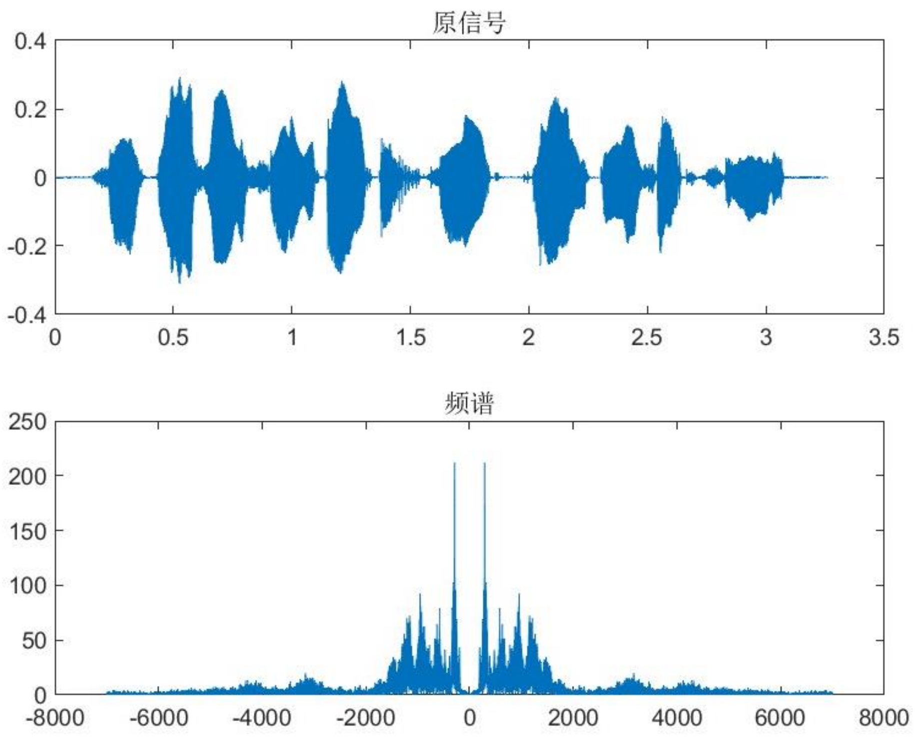
```
% Tone Vocoder

% Add SNR
if SSN == 'Y'
    sound_read = getSSN(sound_read, fs);
end
```

Figure 7 Response the SSN setting

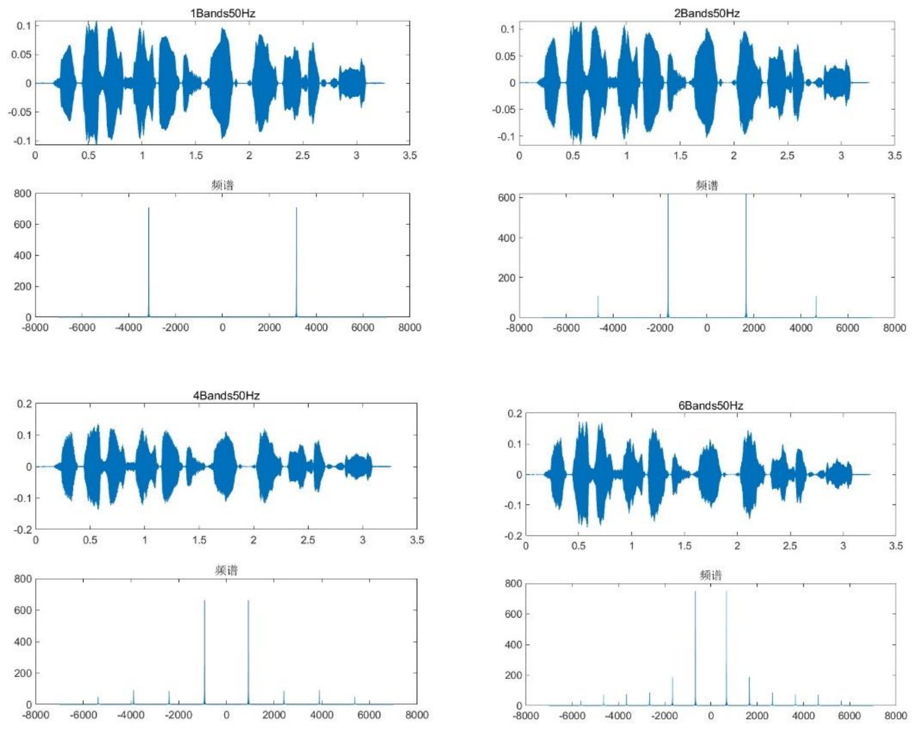
Result analysis of basic task

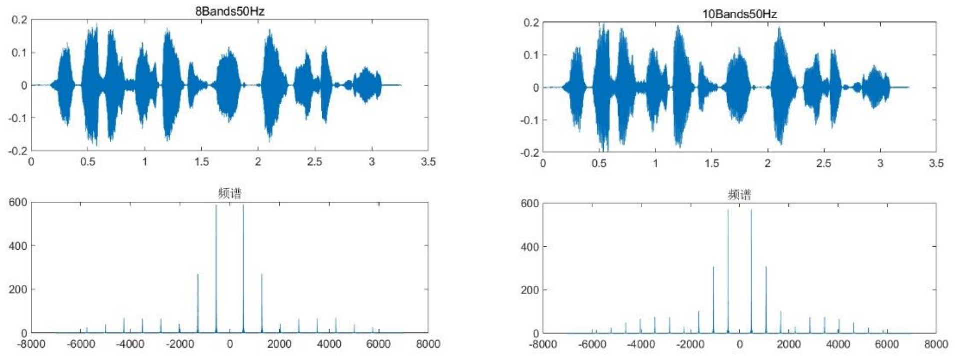
1. The results of Task 1 are as follows.

Original signal 1 and its spectrum

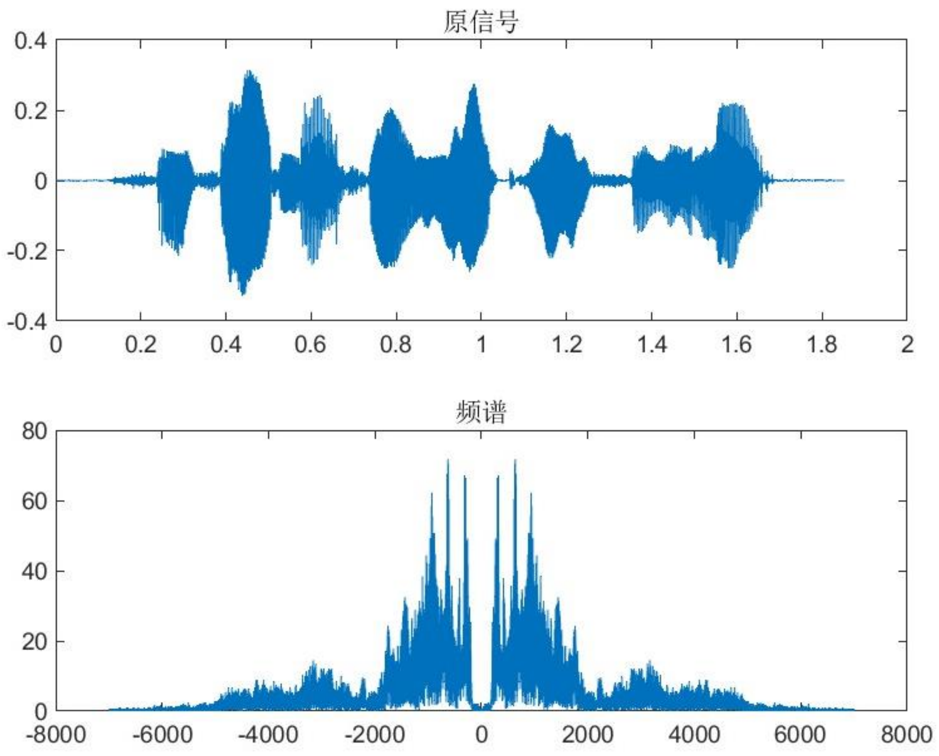


Signal 1 and spectrum recovered under different segment number

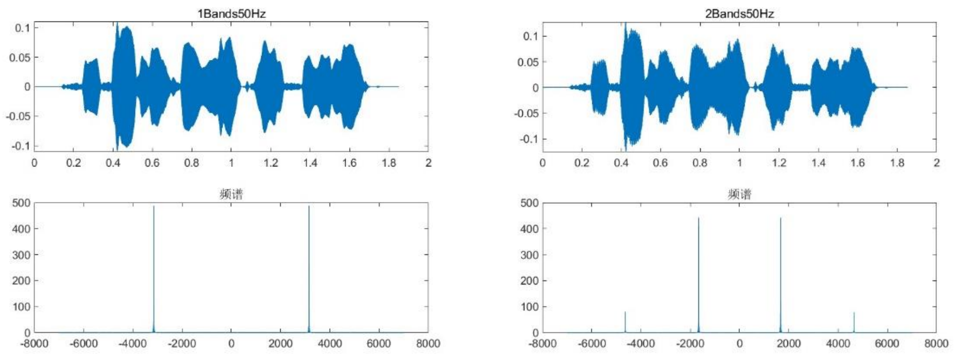


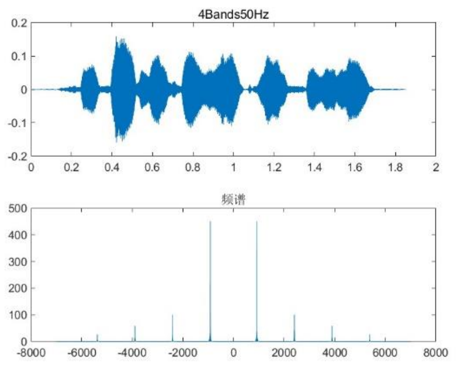


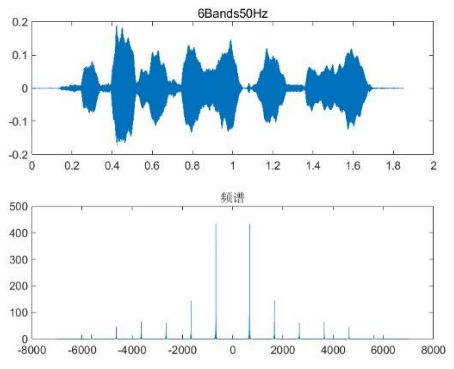
Original signal 2 and its spectrum

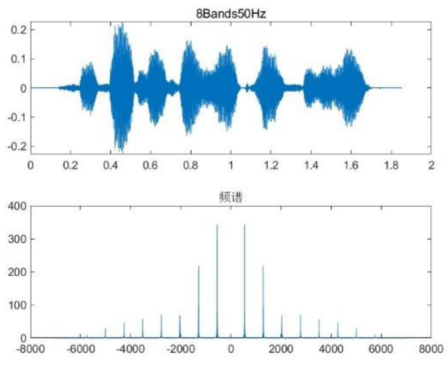


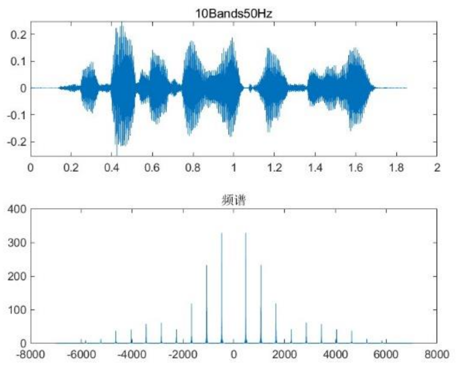
Signal 2 and spectrum recovered under different segments







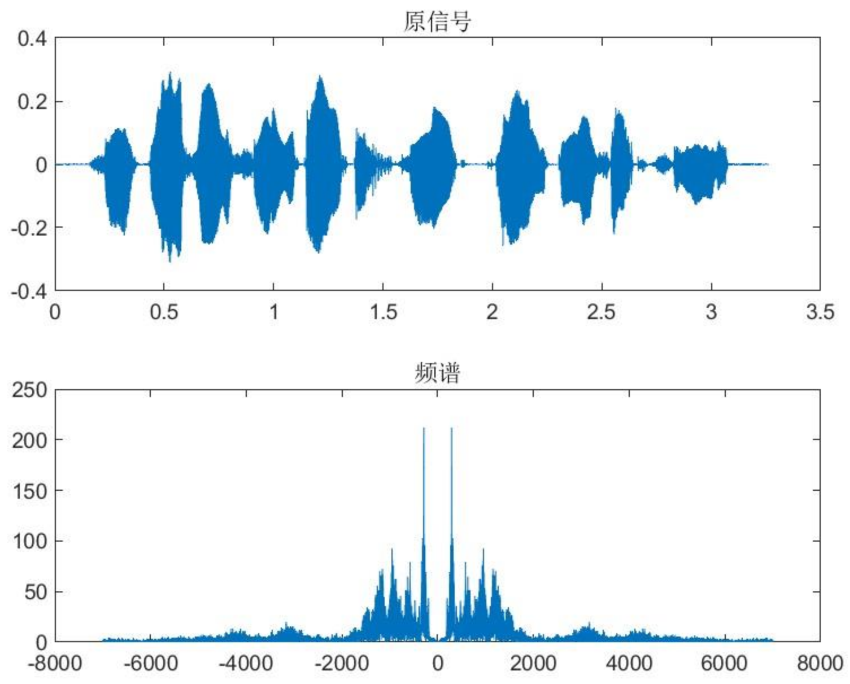




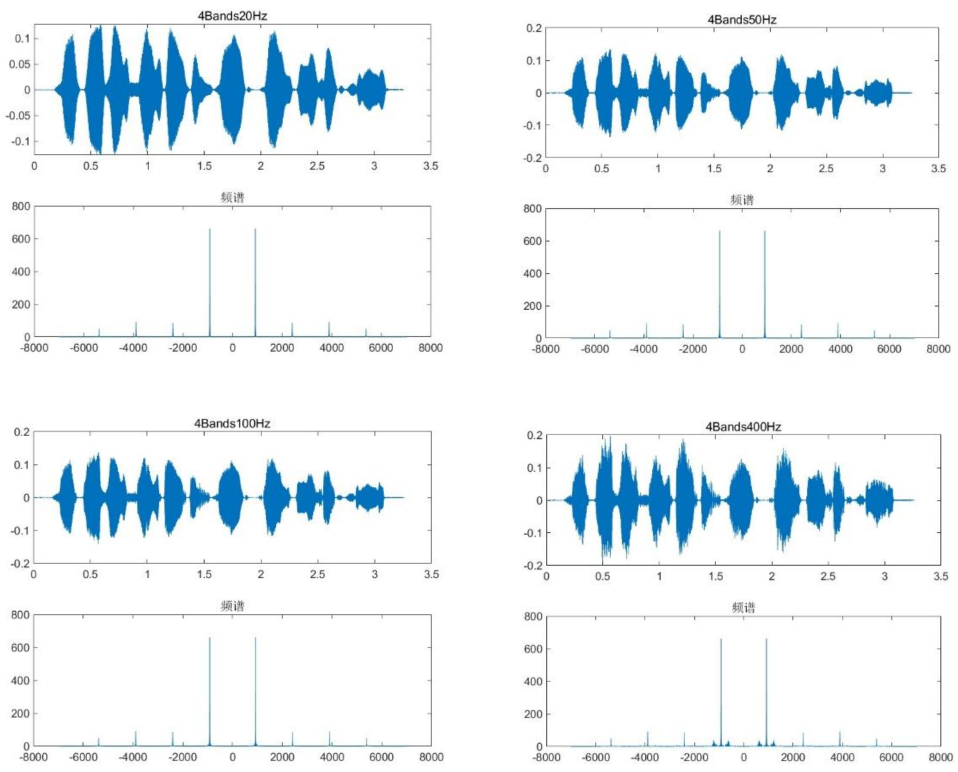
The greater the number of segments, the better the recovery.

2. The results of Task 2 are as follows.

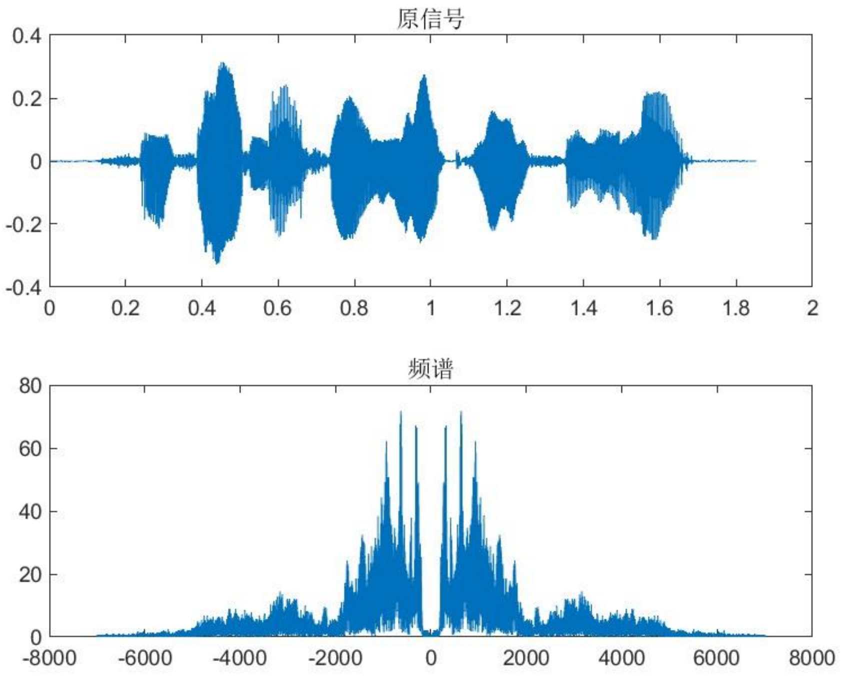
Original signal 1 and its spectrum



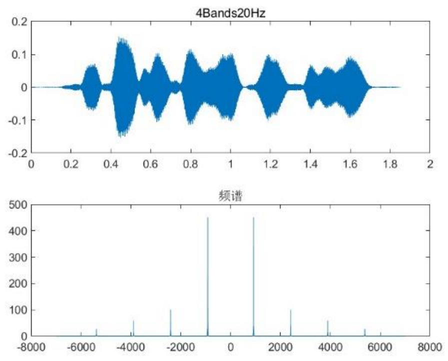
Signal 1 and spectrum recovered under different LPF cut-off frequencies.

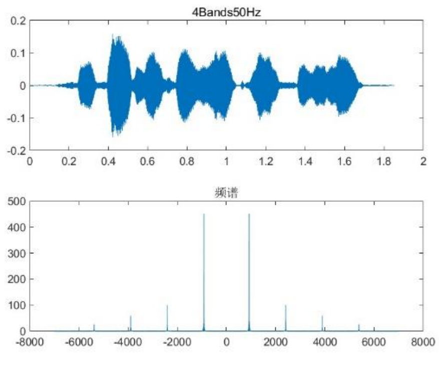


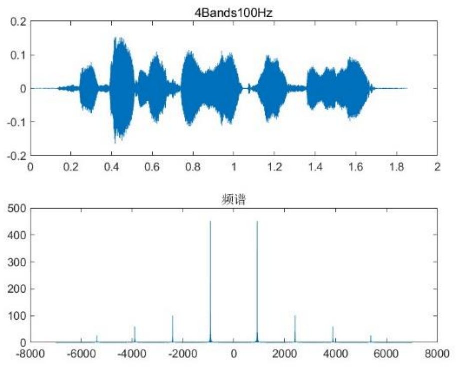
Original signal 2 and its spectrum

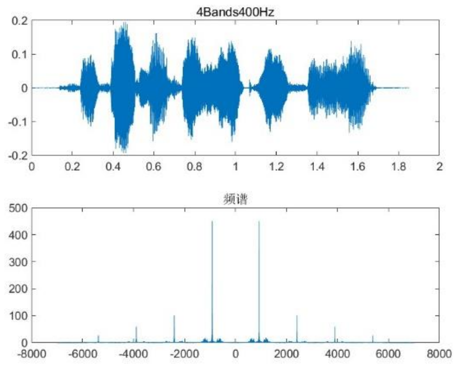


Signal 2 and spectrum recovered under different LPF cut-off frequencies.





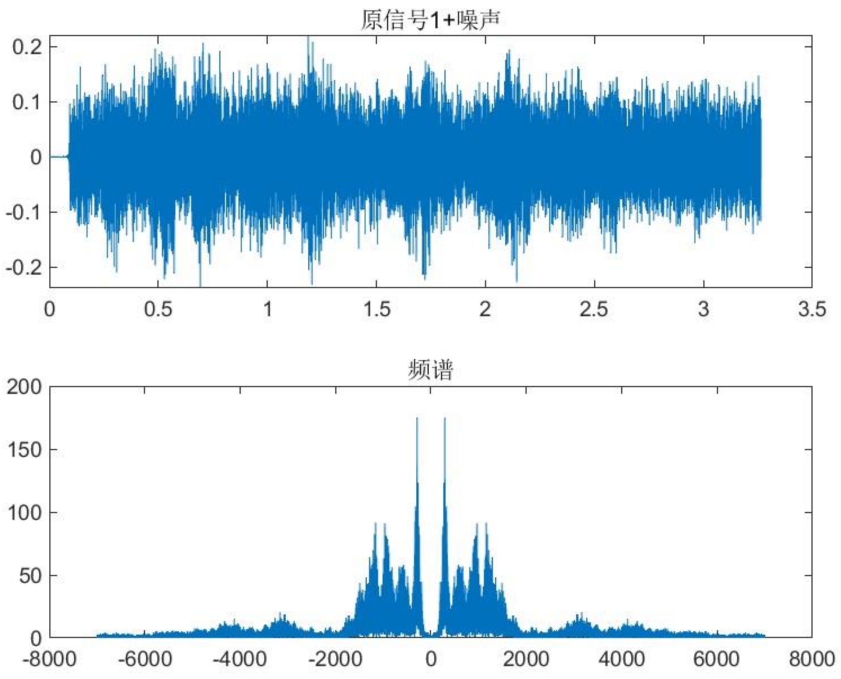




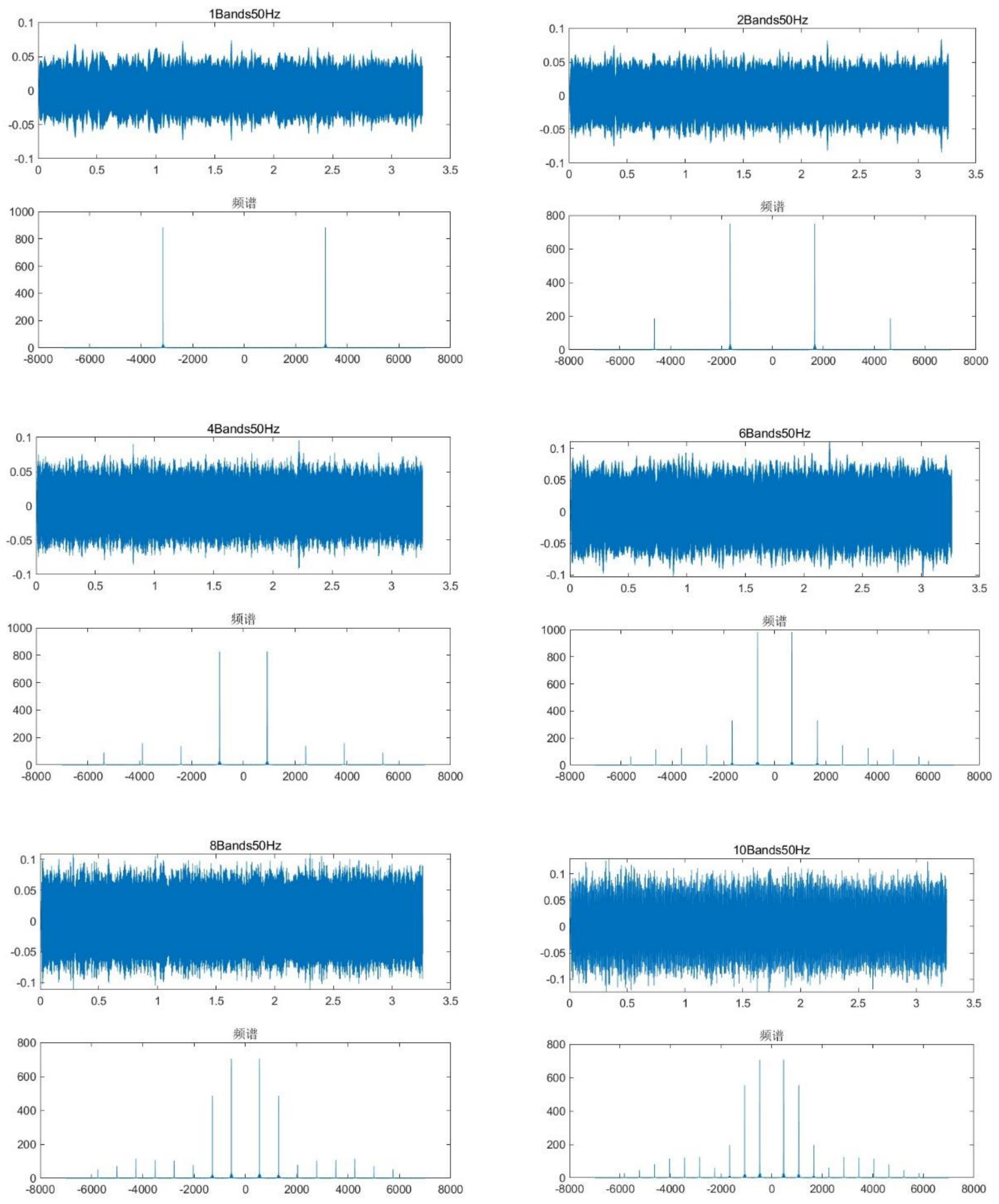
The higher the LPF cut-off frequency, the better the recovery.

3. The results of Task 3 are as follows.

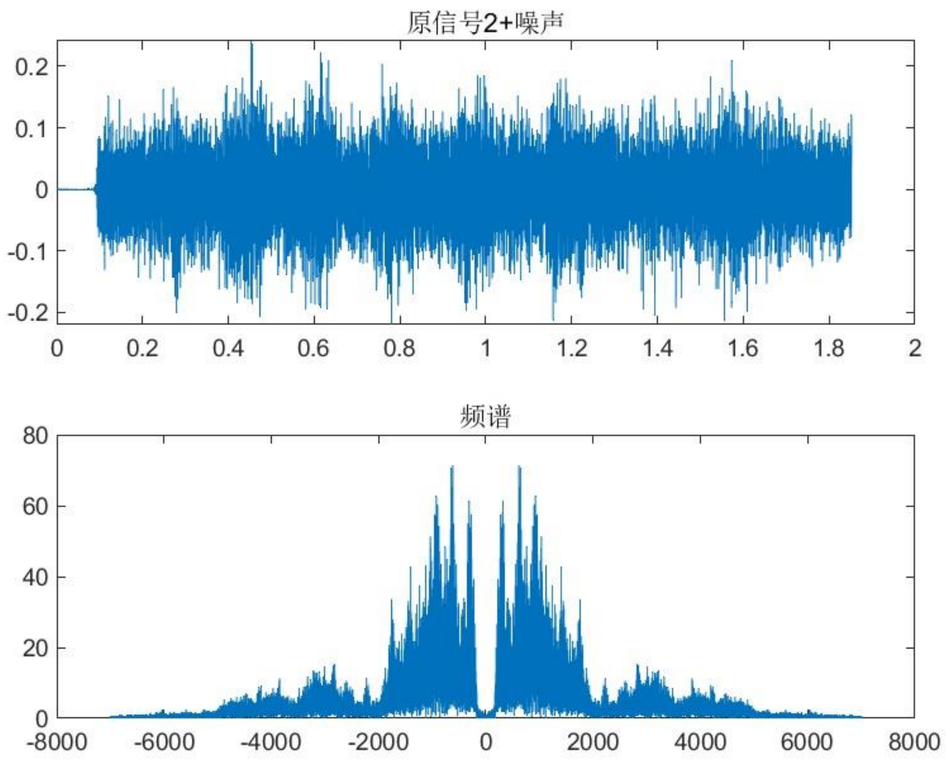
Original signal 1 with SSN and its spectrum



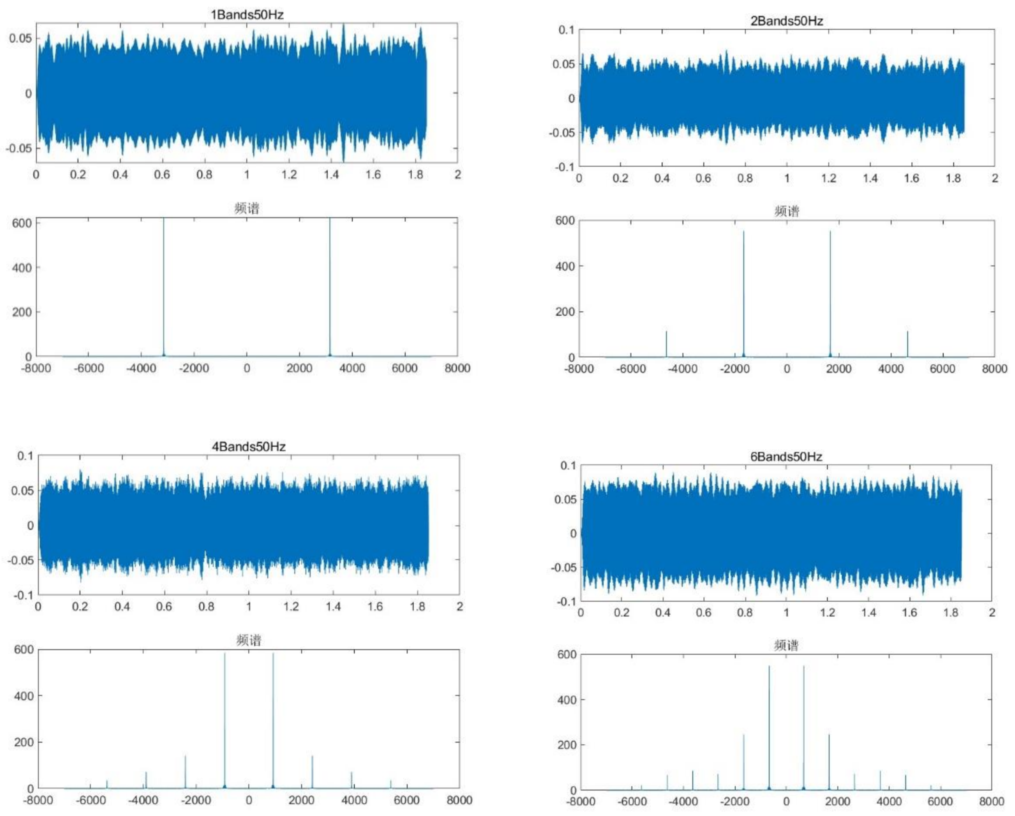
Signal 1 with SSN and spectrum recovered under different N.

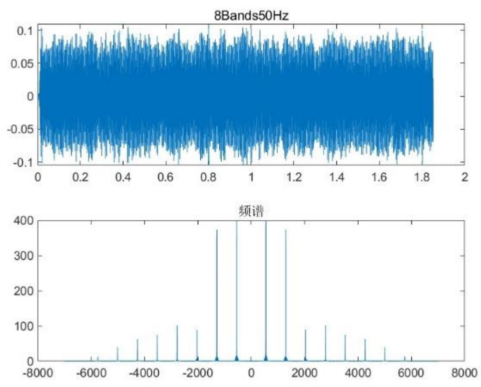


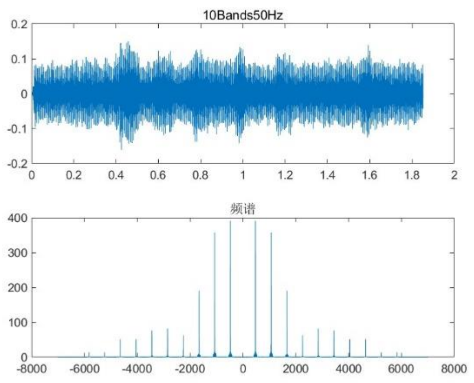
Original signal 2 with SSN and its spectrum



Signal 2 with SSN and spectrum recovered under different N.



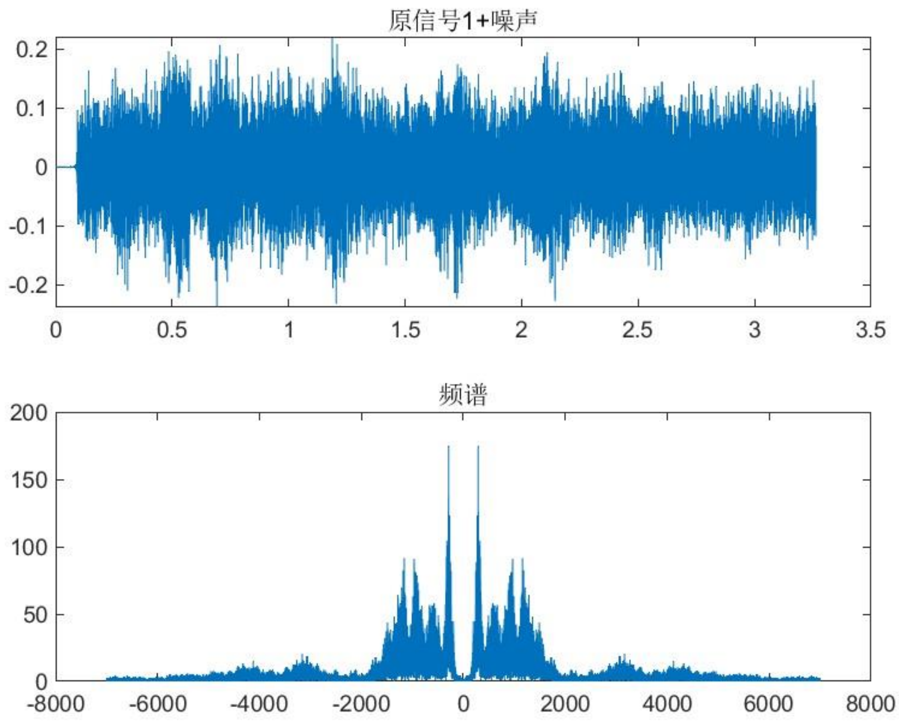




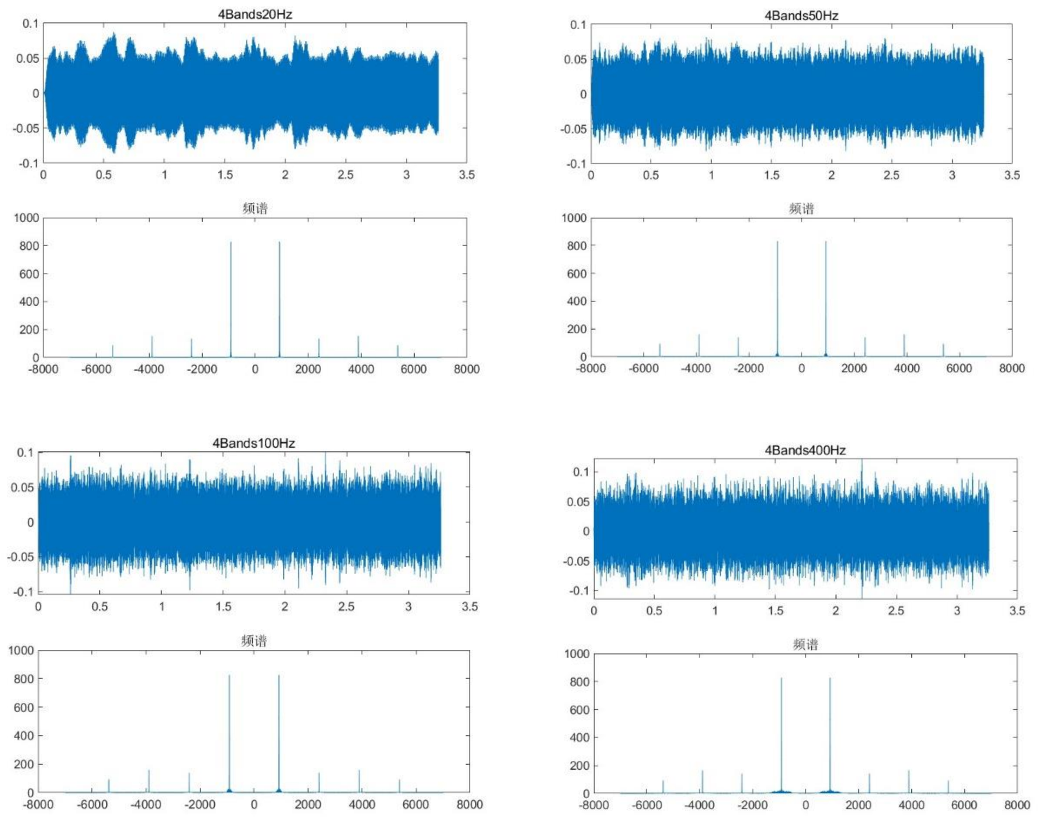
Under the same number of segments, the recovery of the signal with SSN is worse than that without SSN.

4. The results of Task 4 are as follows.

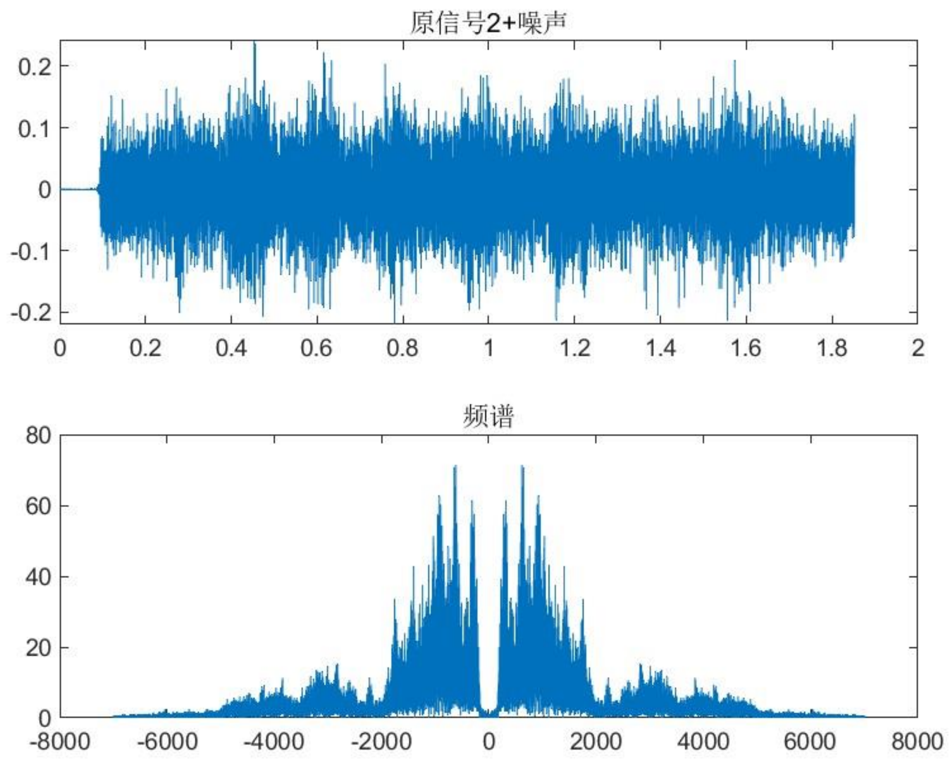
Original signal 1 with SSN and its spectrum



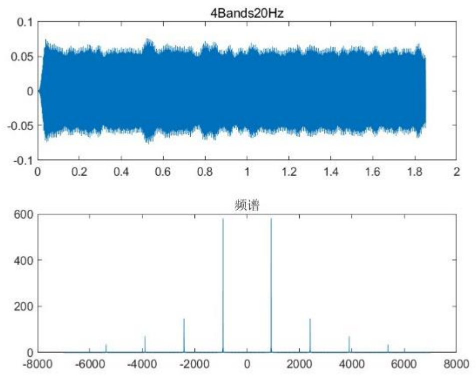
Signal 1 with SSN and spectrum recovered under different LPF cut-off frequencies.

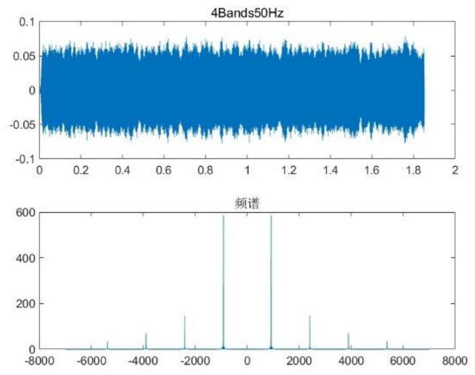


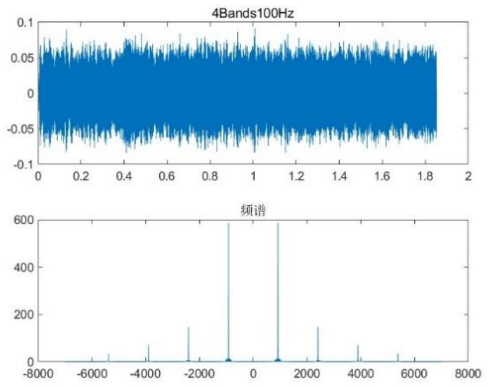
Original signal 2 with SSN and its spectrum

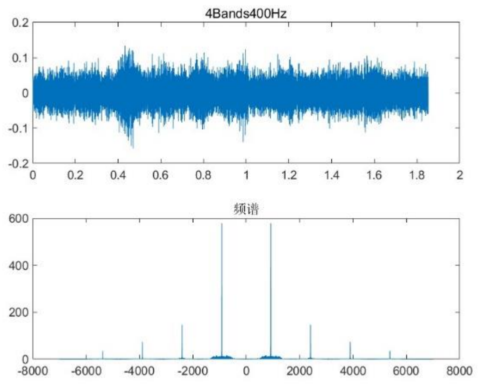


Signal 2 with SSN and spectrum recovered under different LPF cut-off frequencies.









Under the same LPF cutoff frequency, the recovery of the signal with SSN is worse than that without SSN.

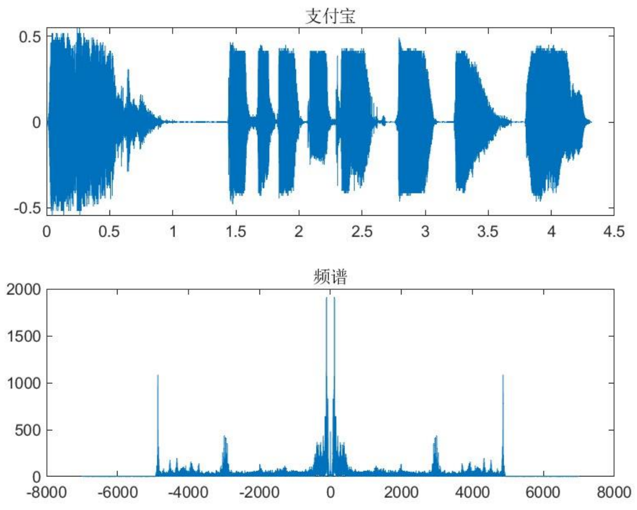
In summary, appropriately increasing N and the cutoff frequency can improve the recovery effect of the sound signal. The worse signal recovery effect with SSN is due to the fact that when the SNR is low, too much noise signal covers up the information of the original signal.

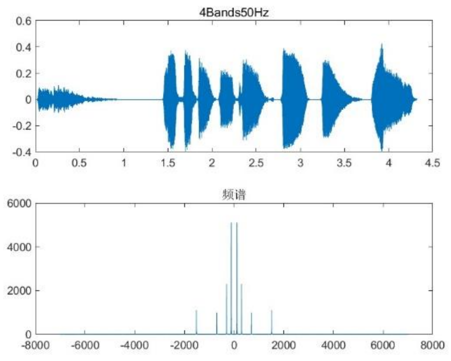
Advanced tasks

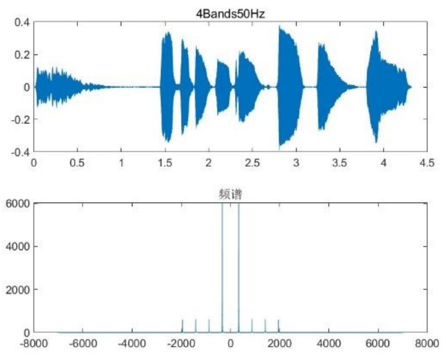
1. *Analysis of Segmentation Method of Bionic Cochlea*

Depending on different N, now test two cases.

1) Take the small N







Cochlear segmentation Average segmentation

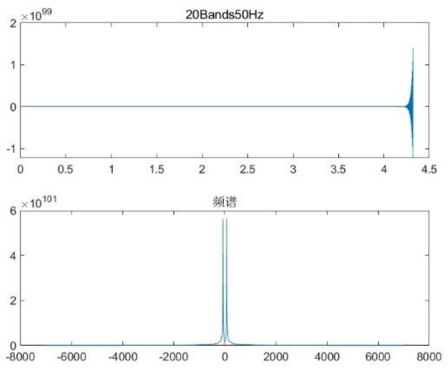
We set N=4, and then used cochlear segmentation and average segmentation to recover the signal. The cochlear recovery signal could already hear the content vaguely, but the average segmentation could not.

Due to the influence of the vocal organs of the human body, the components of the high-frequency part of the frequency spectrum of the speech signal are relatively reduced compared with the low-frequency part, and its energy is also much smaller than that of the low-frequency part.

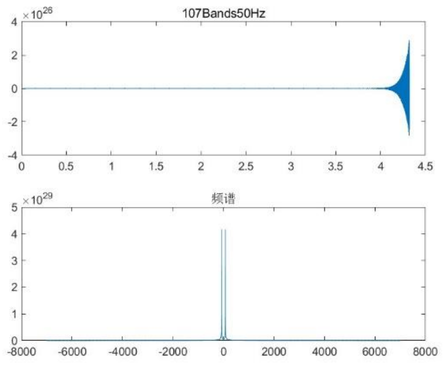
It is concluded that cochlea segmentation is better than average segmentation when the number of segments is small.

2) No limit to the number of segmentations

When the number of segments is not limited, the increase in the number of segments will reduce the frequency interval of each segment. When it is small to a certain extent, the filter will be unstable, resulting in failure to continue to recover the signal, as shown in the following figure:



Cochlear segmentation



Average segmentation

However, the cochlea shows this condition at 20 segments, and the average segment shows this condition at more than 100 segments.

When the two can recover the signal, the maximum segment recovery, and the average segment effect is better.

According to the two conditions, we concluded that the cochlea segmental recovery efficiency is high, and the average segmental recovery upper limit is high.

2. *Analysis of the Cut-off Frequency of LPF*

As the cutoff frequency is increased, the signal content is restored better, but the tone is lower and spikier.

3. *Changing the Carrier Frequency Calculation Method*

Under the conditions of N = 10 and Cut-off Frequency = 50 Hz, we tried to change the calculation method of the sinusoidal carrier signal frequency. Different from the original method of using the midpoint of the upper and lower bounds of the passband, we tried to use the geometric mean, arithmetic mean, harmonic mean, and square mean respectively.

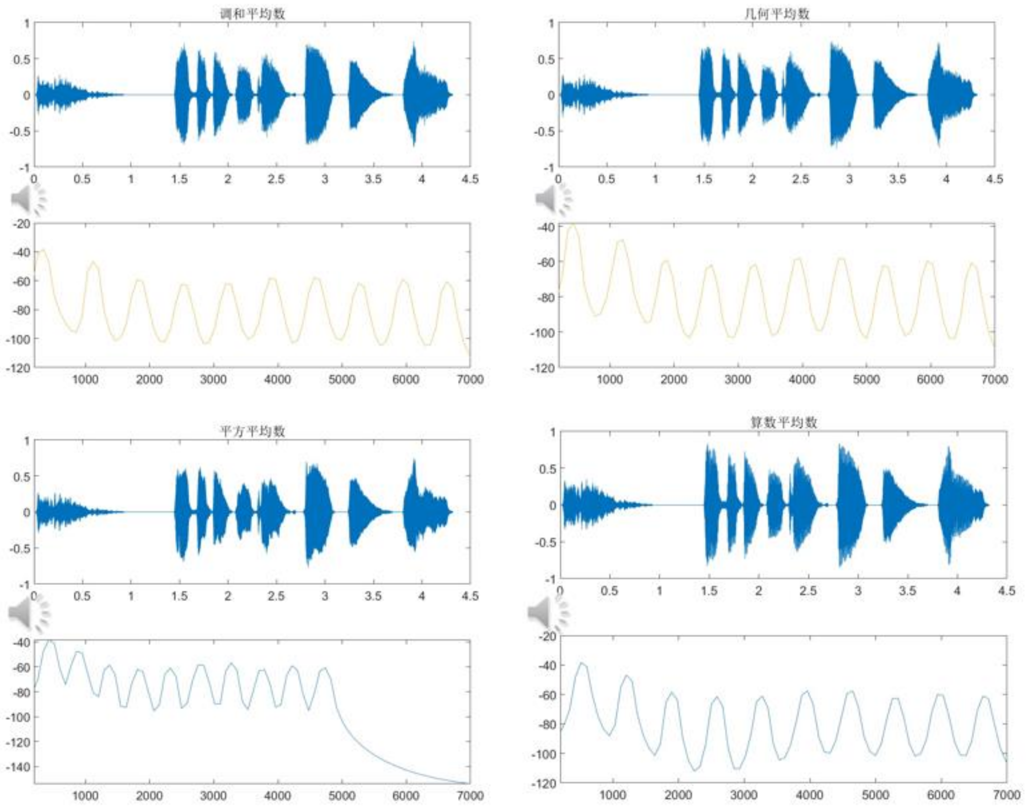
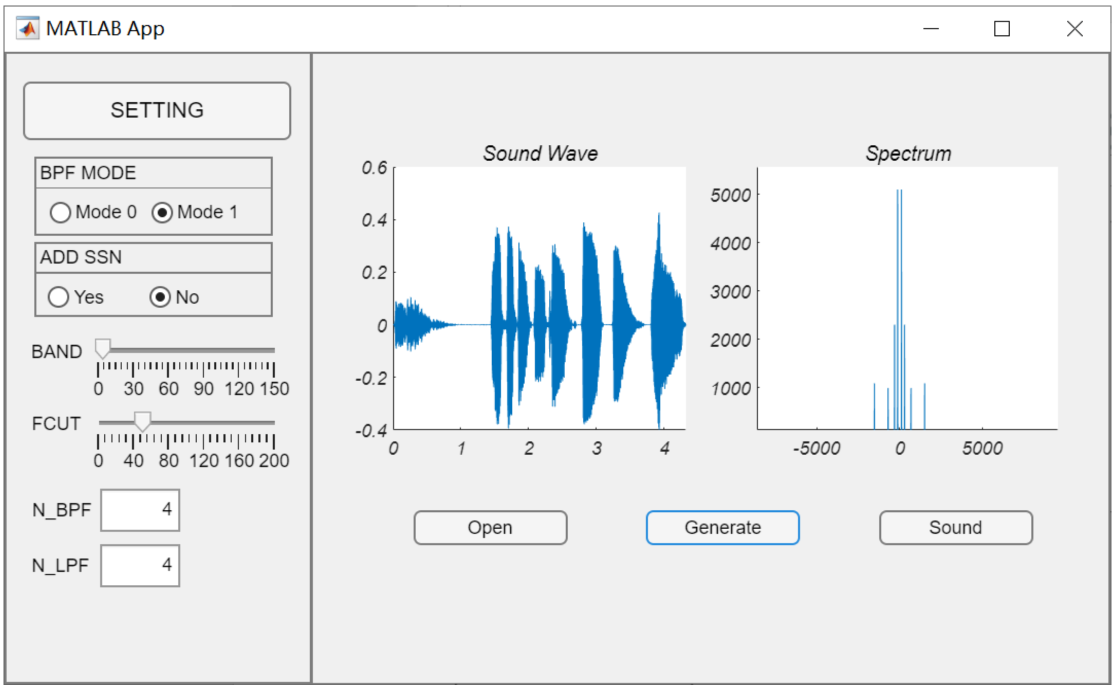


Figure 8 Different carrier frequency calculation method

According to the result images, it can be guessed that these differences are related to the distribution of high-frequency and low-frequency components in the original speech signal. You can try to use different calculation methods to deal with signals of different passbands to find a better solution.

4. *ToneVocoder CONSOLE Design*

To facilitate the adjustment of parameters to quickly obtain experimental results, we use MATLAB App Designer to provide a visual console for the sound wave and spectrum display of the synthesis results. This console can provide a visual operation interface for parameter changes such as bandpass mode selection, SSN signal addition, filter order adjustment, and cutoff frequency adjustment, and output the running results to the "./Output" folder.



MATLAB App

SETTING

BPF MODE: Mode 0 [✓ Mode 1]

ADD SSN: Yes [✓ No]

BAND

FCUT

N_BPF: 4

N_LPF: 4

Figure 9 ToneVocoder CONSOLE

Discussion

Further ideas

1. *Replace the carrier signal with spectral noise at a certain frequency*

    According to research, the closer the characteristics of the carrier signal are to the speech spectrum, the better the effect of signal restoration will be.

    Thus, replacing the original sinusoidal carrier signal with a carrier signal with spectral characteristics may effectively improve the signal recovery effect.

    However, the reason for decomposing and transmitting the original signal is that the human voice signal often has a strong continuity, and the transmission distortion of the continuous signal on the device is often large, so this method is likely to increase the loss of transmission.

2. *Spectral PEAK solution*

    Due to the influence of the vocal organs of the human body, the frequency spectrum of the voice signal is relatively reduced in the high-frequency part compared to the low-frequency part, and the SNR is low, causing the high-frequency signal to gradually weaken during transmission and gradually submerged by noise. To improve the quality of signal transmission, it is necessary to pre-emphasize the speech signal. Pre-emphasis can increase the energy of the speech signal in the high-frequency part, improve its signal-to-noise ratio, and at the same time make the signal spectrum of the high-frequency part relatively flat.

Due to the strong time-varying nature of the speech signal, it is a better choice to divide the signal into frames. For brief periods of time, we can approximate that the characteristics of the sound are constant.

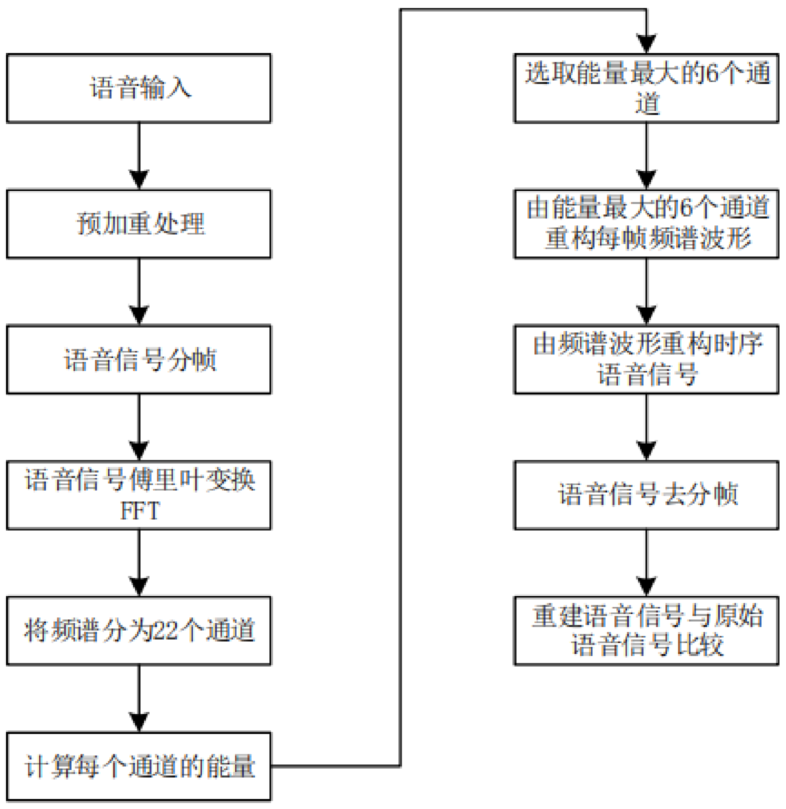


Figure 10 SMSP Algorithm

It should be noted that due to technical conditions and internal space limitations, the number of tiny electrodes that can be used for cochlear implants to stimulate human nerves is limited. Therefore, we need to make the most efficient use of the limited number of channels.

After the signal is frequency-screened, the passbands with the highest signal transmission efficiency are found by calculating the power spectrum, and they are transmitted preferentially. Finally, we can test the restoration effect.

Problems during the project

1. Segmentation method innovation.
2. Code implementation of task requirements.
3. Code specification and code docking.

What we have learned

1. Review how to use MATLAB functions to conduct audio signal processing.
2. Accumulated knowledge related to sound generation, synthesis transfer, etc., laying the foundation for future processing that may be required in similar projects.
3. Learn how to collect and refer to relevant materials, and further develop assumptions and discussions, construct models and corresponding explanations.

References

Ince, A. N. (1992). *Digital Speech Processing: Speech Encoding, Synthesis, and Recognition*. Kluwer Academic Publishers. https://doi.org/10.1007/978-1-4757-2148-5

Steeneken, H. J. m, & Houtgast, T. (1999). Mutual Dependence of the Octave-Band Weights in Predicting Speech Intelligibility. Mutual Dependence of the Octave-Band Weights in Predicting Speech Intelligibility, 28(2), 109–123. https://doi.org/https://doi.org/10.1016/S0167-6393(99)00007-2.

王旭 电子耳蜗音频信号处理系统设计 [D]. 西安电子科技大学, 2017

Appendix 1

```
% #ToneVocoder#
clear;clc;

% Import the sound_read

[sound_read, fs] = audioread('Sound.wav');
t = (0 : (length(sound_read) - 1)) / fs;



% Veriables Setting

SSN = 'N';    % Add SNR 'Y' or not 'N'
Mode = 1;    % Equal frequency: 0 | Average cochlea length: 1
N_sub = 2;    % Number of segments
N_BPF = 4;    % BPF order
Cf = 50;    % Cut-off frequence of LPF
N_LPF = 4;    % LPF order



% Tone Vocoder

% Add SNR
if SSN == 'Y'
    sound_read = getSSN(sound_read, fs);
end

% Passband Mode
% Mode 0 is Equal frequency
if Mode == 0

    Nmax = N_sub;
    fmin = 200;
    fmax = 7000;

    % Generate an array of buffer filter coefficients
    b = zeros(Nmax, 2 * N_BPF + 1);
    a = zeros(Nmax, 2 * N_BPF + 1);
    gap = (fmax - fmin) / Nmax;
    for j = 1 : Nmax
        [b(j, :), a(j, :)] = butter(N_BPF, [fmin + (j-1)*gap fmin + j*gap] / fs * 2);
    end

    % Generate sub-bands
    cuted = zeros(Nmax, length(sound_read));
    for j = 1 : Nmax
        cuted(j, :) = filter(b(j, :), a(j, :), sound_read);
    end

    % Generate the Envelope
    envelopes = Envelope(cuted(:, :), fs, N_LPF, Cf);

    % Carrier loaded
    loaded = zeros(Nmax, length(sound_read));
    gap = (fmax - fmin) / Nmax;
    for j = 1 : Nmax
        for k = 1 : length(sound_read)
            loaded(j, k) = envelopes(j, k) .* cos(2*pi * (fmin + (j-1/2)*gap) * (k-1) / fs);
        end
    end

    % Synthesis and normalization
    output = zeros(1, length(sound_read));
    for j = 1 : Nmax
        output = output + loaded(j, :);
    end

    % Display the output
    sound(output, fs);
    figure;
    plot(t, output);
    if SSN == 'Y'

audiowrite(['./Output/EqualFrequency_WithSSN_',num2str(N_sub),'Bands_',num2str(
Cf),'Hz.wav'], output, fs);
```

```
    end
    if SSN == 'N'

audiowrite(['./Output/EqualFrequency_',num2str(N_sub),'Bands_',num2str(Cf),'Hz.wa
v'], output, fs);
    end



    % Mode 1 is Average cochlea length
elseif Mode == 1

    dmin = (50 * log(1827 / 827)) / (3 * log(10));
    dmax = (50 * log(35827 / 827)) / (3 * log(10));
    Nmax = N_sub;

    % Generate an array of buffer filter coefficients
    b = zeros(Nmax, 2 * N_BPF + 1);
    a = zeros(Nmax, 2 * N_BPF + 1);
    gap = (dmax - dmin) / Nmax;
    for j = 1 : Nmax
        [b(j, :), a(j, :)] = butter(N_BPF, [alter(dmin + (j-1)*gap) alter(dmin + j*gap)] / fs
* 2);
    end

    % Generate sub-bands
    cuted = zeros(Nmax, length(sound_read));
    for j = 1 : Nmax
        cuted(j, :) = filter(b(j, :), a(j, :), sound_read);
    end

    % Generate the Envelope
    envelopes = Envelope(cuted(:, :), fs, N_LPF, Cf);

    % Carrier loaded
    loaded = zeros(Nmax, length(sound_read));
    gap = (dmax - dmin) / Nmax;
    for j = 1 : Nmax
        for k = 1 : length(sound_read)
```

```
            loaded(j, k) = envelopes(j, k) .* cos(2*pi * alter(dmin + (j-1/2)*gap) * (k-1) /
fs);
        end
    end

    % Synthesis and normalization
    output = zeros(1, length(sound_read));
    for j = 1 : Nmax
        output = output + loaded(j, :);
    end

    % Display the outputs
%       sound(output, fs);
    figure;
    subplot(2, 1, 1), plot(t, output);

    ak = fftshift(abs(fft(output)));
    w = linspace(-7000, 7000, length(abs(fft(output))));
    subplot(2, 1, 2), plot(w, ak);

    % Save
    if SSN == 'Y'

audiowrite(['./Output/Cochlea_WithSSN_',num2str(N_sub),'Bands_',num2str(Cf),'Hz.
wav'], output, fs);
    end
    if SSN == 'N'

audiowrite(['./Output/Cochlea_',num2str(N_sub),'Bands_',num2str(Cf),'Hz.wav'],
output, fs);
    end

end


% Functions
% Generate SSN
function Out = getSSN(sound_read, fs)
sound_read = sound_read';
nfft = 512;
noverlap = nfft / 2;
Windows = hamming(nfft);
% Returns the power spectral density
[Pxx,w] = pwelch(repmat(sound_read, 1, 10), Windows, noverlap, nfft, fs);
% Generate filter
b = fir2(3000, w/(fs / 2), sqrt(Pxx / max(Pxx)));
% Generate white noise
N = length(sound_read);
noise = 1 - 2 * rand(1, N + length(b) - 1);
% Generate SSN
ssn = filter(b, 1, noise);
ssn = ssn(length(b) : end);
% Adjust the energy of SSN
ssn = ssn / norm(ssn) * norm(sound_read) * 10 ^ (1 / 4);
% Addition sound_read with ssn
Out = sound_read + ssn;
% Normalize out to sound_read
Out = Out / norm(Out) * norm(sound_read);
end

% Batch Extract Envelopes
function E = Envelope(Y, fs, N1, cf)
[b, a] = butter(N1, cf / (fs / 2), "low");
sizeY = size(Y);
m1 = sizeY(1);
m2 = sizeY(2);
Y = abs(Y);
E = zeros(m1, m2);
for n = 1 : m1
    E(n, :) = filter(b, a, Y(n, :));
end
end

% f-d Alter
```

```
function f = alter(d)
f=165.4*(10^(0.06*d)-1);
end
```

Appendix 2

```
classdef ToneVocoder < matlab.apps.AppBase


    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                   matlab.ui.Figure
        GridLayout                 matlab.ui.container.GridLayout
        LeftPanel                  matlab.ui.container.Panel
        SETTINGButton              matlab.ui.control.Button
        FCUTSlider                 matlab.ui.control.Slider
        FCUTSliderLabel            matlab.ui.control.Label
        BANDSlider                 matlab.ui.control.Slider
        BANDSliderLabel            matlab.ui.control.Label
        ADDSSNButtonGroup          matlab.ui.container.ButtonGroup
        NoButton                   matlab.ui.control.RadioButton
        YesButton                  matlab.ui.control.RadioButton
        BPFMODEButtonGroup         matlab.ui.container.ButtonGroup
        Mode1Button                matlab.ui.control.RadioButton
        Mode0Button                matlab.ui.control.RadioButton
        N_LPFEditField             matlab.ui.control.NumericEditField
        N_LPFEditFieldLabel        matlab.ui.control.Label
        N_BPFEditField             matlab.ui.control.NumericEditField
        N_BPFEditFieldLabel        matlab.ui.control.Label
        RightPanel                 matlab.ui.container.Panel
        SoundButton                matlab.ui.control.Button
        GenerateButton             matlab.ui.control.Button
        OpenButton                 matlab.ui.control.Button
        UIAxes2                    matlab.ui.control.UIAxes
        UIAxes                     matlab.ui.control.UIAxes
    end


    % Properties that correspond to apps with auto-reflow
    properties (Access = private)
        onePanelWidth = 576;
    end
    properties (Access = private)
        SSN = 'N'; % Add SNR 'Y' or not 'N'
        Mode = 0; % Equal frequency: 0 | Average cochlea length: 1
        N_sub = 4; % Number of segments
        N_BPF = 4; % BPF order
        Cf = 50; % Cut-off frequence of LPF
        N_LPF = 4; % LPF order

        sound_read % Description
        t % Description
        y % Description
        fs % Description
        w0 % Description
        ak % Description

        isOpen = 0; % Description
        source % Description
    end

    methods (Access = private)



function results = main (app)
% Tone Vocoder

mkdir Output\

% Add SNR
if app.SSN == 'Y'
app.sound_read = getSSN(app, app.sound_read, app.fs);
end
```

```
% Passband Mode
% Mode 0 Equal frequency
if app.Mode == 0


Nmax = app.N_sub;
fmin=200;
fmax=7000;


% Generate an array of buffer filter coefficients
b = zeros(Nmax, 2 * app.N_BPF + 1);
a = zeros(Nmax, 2 * app.N_BPF + 1);
gap = (fmax - fmin) / Nmax;
for j = 1 : Nmax
[b(j, :), a(j, :)] = butter(app.N_BPF, [fmin + (j-1)*gap fmin + j*gap] / app.fs * 2);
end


% Generate sub-bands
cuted = zeros(Nmax, length(app.sound_read));
for j = 1 : Nmax
cuted(j, :) = filter(b(j, :), a(j, :), app.sound_read);
end


% Generate the Envelope
envelopes = Envelope(app, cuted(:, :), app.fs, app.N_LPF, app.Cf);


% Carrier loaded
loaded = zeros(Nmax, length(app.sound_read));
gap = (fmax - fmin) / Nmax;
for j = 1 : Nmax
for k = 1 : length(app.sound_read)
loaded(j, k) = envelopes(j, k) .* cos(2*pi * (fmin + (j-1/2)*gap) * (k-1) / app.fs);
end
end
```

```
% Synthesis and normalization
output = zeros(1, length(app.sound_read));
for j = 1 : Nmax
output = output + loaded(j, :);
end


% % Display the output
% sound(output, app.fs);
% figure;
% plot(app.t, output);


app.ak = fftshift(abs(fft(output)));
app.w0 = linspace(-7000, 7000, length(abs(fft(output))));


if app.SSN == 'Y'
audiowrite(['./Output/EqualFrequency_WithSSN_',num2str(app.N_sub),'Bands_',num
2str(app.Cf),'Hz.wav'], output, app.fs);
end
if app.SSN == 'N'
audiowrite(['./Output/EqualFrequency_',num2str(app.N_sub),'Bands_',num2str(app.C
f),'Hz.wav'], output, app.fs);
end
app.y = output;
results = [output, app.fs, app.t];


% Mode 1 Average cochlea length
elseif app.Mode == 1


dmin = (50 * log(1827 / 827)) / (3 * log(10));
dmax = (50 * log(35827 / 827)) / (3 * log(10));
Nmax = app.N_sub;


% Generate an array of buffer filter coefficients
b = zeros(Nmax, 2 * app.N_BPF + 1);
```

```
a = zeros(Nmax, 2 * app.N_BPF + 1);
gap = (dmax - dmin) / Nmax;
for j = 1 : Nmax
[b(j, :), a(j, :)] = butter(app.N_BPF, [alter(app, dmin + (j-1)*gap) alter(app, dmin +
j*gap)] / app.fs * 2);
end


% Generate sub-bands
cuted = zeros(Nmax, length(app.sound_read));
for j = 1 : Nmax
cuted(j, :) = filter(b(j, :), a(j, :), app.sound_read);
end


% Generate the Envelope
envelopes = Envelope(app, cuted(:, :), app.fs, app.N_LPF, app.Cf);


% Carrier loaded
loaded = zeros(Nmax, length(app.sound_read));
gap = (dmax - dmin) / Nmax;
for j = 1 : Nmax
for k = 1 : length(app.sound_read)
loaded(j, k) = envelopes(j, k) .* cos(2*pi * alter(app, dmin + (j-1/2)*gap) * (k-1) /
app.fs);
end
end


% Synthesis and normalization
output = zeros(1, length(app.sound_read));
for j = 1 : Nmax
output = output + loaded(j, :);
end


% % Display the outputs
% pause(5);
% sound(output, app.fs);
% figure;
% plot(app.t, output);


app.ak = fftshift(abs(fft(output)));
app.w0 = linspace(-7000, 7000, length(abs(fft(output))));


if app.SSN == 'Y'
audiowrite(['./Output/Cochlea_WithSSN_',num2str(app.N_sub),'Bands_',num2str(app
.Cf),'Hz.wav'], output, app.fs);
end
if app.SSN == 'N'
audiowrite(['./Output/Cochlea_',num2str(app.N_sub),'Bands_',num2str(app.Cf),'Hz.w
av'], output, app.fs);
end
app.y = output;
results = [output, app.fs, app.t];
end


end


% Functions


% Generate SSN
function Out = getSSN(~, sound_read, fs)
nfft = 512;
noverlap = nfft / 2;
Windows = hamming(nfft);
% Returns the power spectral density
[Pxx,w] = pwelch(repmat(sound_read, 1, 10), Windows, noverlap, nfft, fs);
% Generate filter
b = fir2(3000, w/(fs / 2), sqrt(Pxx / max(Pxx)));
% Generate white noise
N = length(sound_read);
noise = 1 - 2 * rand(1, N + length(b) - 1);
% Generate SSN
```

```
ssn = filter(b, 1, noise);
ssn = ssn(length(b) : end);
% Adjust the energy of SSN
ssn = ssn / norm(ssn) * norm(sound_read) * 10 ^ (1 / 4);
% Addition sound_read with ssn
Out = sound_read + ssn';
% Normalize out to sound_read
Out = Out / norm(Out) * norm(sound_read);
end


% Batch Extract Envelopes
function E = Envelope(~, Y, fs, N1, cf)
[b, a] = butter(N1, cf / (fs / 2), "low");
sizeY = size(Y);
m1 = sizeY(1);
m2 = sizeY(2);
Y = abs(Y);
E = zeros(m1, m2);
for n = 1 : m1
E(n, :) = filter(b, a, Y(n, :));
end
end


% f-d Alter
function f = alter(~, d)
f=165.4*(10^(0.06*d)-1);
end




end




% Callbacks that handle component events
methods (Access = private)
```

```
% Selection changed function: BPFMODEButtonGroup
function BPFMODEButtonGroupSelectionChanged(app, event)
selectedButton = app.BPFMODEButtonGroup.SelectedObject;
if selectedButton == app.Mode0Button
app.Mode = 0;
elseif selectedButton == app.Mode1Button
app.Mode = 1;
end
end


% Selection changed function: ADDSSNButtonGroup
function ADDSSNButtonGroupSelectionChanged(app, event)
selectedButton = app.ADDSSNButtonGroup.SelectedObject;
if selectedButton == app.YesButton
app.SSN = 'Y';
elseif selectedButton == app.NoButton
app.SSN = 'N';
end
end


% Button pushed function: GenerateButton
function GenerateButtonPushed(app, event)
if app.isOpen == 1
app.sound_read = app.source;
app.main;
plot(app.UIAxes, app.t, app.y);
plot(app.UIAxes2, app.w0, app.ak);
else
msgbox('Please open a sound file ~')
end
end


% Value changed function: BANDSlider
function BANDSliderValueChanged(app, event)
value = app.BANDSlider.Value;
```

```
app.N_sub = ceil(value);
end


% Value changed function: FCUTSlider
function FCUTSliderValueChanged(app, event)
value = app.FCUTSlider.Value;
app.Cf = ceil(value);
end


% Value changed function: N_BPFEditField
function N_BPFEditFieldValueChanged(app, event)
value = app.N_BPFEditField.Value;
app.N_BPF = value;
end


% Value changed function: N_LPFEditField
function N_LPFEditFieldValueChanged(app, event)
value = app.N_LPFEditField.Value;
app.N_LPF = value;
end


% Button pushed function: OpenButton
function OpenButtonPushed(app, event)
[filename,pathname] = uigetfile({'*.wav'},'请打开音频捏');
if pathname ~= 0
str = [pathname filename];
app.isOpen = 1;
[app.source, app.fs] = audioread(str);
app.sound_read = app.source;
app.t = (0 : (length(app.sound_read) - 1)) / app.fs;
app.y = app.sound_read';
plot(app.UIAxes, app.t, app.y);
app.ak = fftshift(abs(fft(app.y)));
app.w0 = linspace(-7000, 7000, length(abs(fft(app.y))));
plot(app.UIAxes2, app.w0, app.ak);
end
```

```
end


% Button pushed function: SoundButton
function SoundButtonPushed(app, event)


if app.isOpen == 1
sound(app.y, app.fs);
else
msgbox('Please open a sound file ~')
end
end


% Changes arrangement of the app based on UIFigure width
function updateAppLayout(app, event)
currentFigureWidth = app.UIFigure.Position(3);
if(currentFigureWidth <= app.onePanelWidth)
% Change to a 2x1 grid
app.GridLayout.RowHeight = {410, 410};
app.GridLayout.ColumnWidth = {'1x'};
app.RightPanel.Layout.Row = 2;
app.RightPanel.Layout.Column = 1;
else
% Change to a 1x2 grid
app.GridLayout.RowHeight = {'1x'};
app.GridLayout.ColumnWidth = {199, '1x'};
app.RightPanel.Layout.Row = 1;
app.RightPanel.Layout.Column = 2;
end
end
end


% Component initialization
methods (Access = private)


% Create UIFigure and components
```

```
function createComponents(app)


% Create UIFigure and hide until all components are created
app.UIFigure = uifigure('Visible', 'off');
app.UIFigure.AutoResizeChildren = 'off';
app.UIFigure.Position = [100 100 717 410];
app.UIFigure.Name = 'MATLAB App';
app.UIFigure.SizeChangedFcn = createCallbackFcn(app, @updateAppLayout, true);


% Create GridLayout
app.GridLayout = uigridlayout(app.UIFigure);
app.GridLayout.ColumnWidth = {199, '1x'};
app.GridLayout.RowHeight = {'1x'};
app.GridLayout.ColumnSpacing = 0;
app.GridLayout.RowSpacing = 0;
app.GridLayout.Padding = [0 0 0 0];
app.GridLayout.Scrollable = 'on';


% Create LeftPanel
app.LeftPanel = uipanel(app.GridLayout);
app.LeftPanel.Layout.Row = 1;
app.LeftPanel.Layout.Column = 1;


% Create N_BPFEditFieldLabel
app.N_BPFEditFieldLabel = uilabel(app.LeftPanel);
app.N_BPFEditFieldLabel.HorizontalAlignment = 'right';
app.N_BPFEditFieldLabel.Position = [14 102 43 22];
app.N_BPFEditFieldLabel.Text = 'N_BPF';


% Create N_BPFEditField
app.N_BPFEditField = uieditfield(app.LeftPanel, 'numeric');
app.N_BPFEditField.ValueChangedFcn = createCallbackFcn(app,
@N_BPFEditFieldValueChanged, true);
app.N_BPFEditField.Position = [62 99 52 28];
app.N_BPFEditField.Value = 4;
```

```
% Create N_LPFEditFieldLabel
app.N_LPFEditFieldLabel = uilabel(app.LeftPanel);
app.N_LPFEditFieldLabel.HorizontalAlignment = 'right';
app.N_LPFEditFieldLabel.Position = [13 66 42 22];
app.N_LPFEditFieldLabel.Text = 'N_LPF';


% Create N_LPFEditField
app.N_LPFEditField = uieditfield(app.LeftPanel, 'numeric');
app.N_LPFEditField.ValueChangedFcn = createCallbackFcn(app,
@N_LPFEditFieldValueChanged, true);
app.N_LPFEditField.Position = [62 63 52 28];
app.N_LPFEditField.Value = 4;


% Create BPFMODEButtonGroup
app.BPFMODEButtonGroup = uibuttongroup(app.LeftPanel);
app.BPFMODEButtonGroup.SelectionChangedFcn = createCallbackFcn(app,
@BPFMODEButtonGroupSelectionChanged, true);
app.BPFMODEButtonGroup.Title = 'BPF MODE';
app.BPFMODEButtonGroup.Position = [19 291 155 51];


% Create Mode0Button
app.Mode0Button = uiradiobutton(app.BPFMODEButtonGroup);
app.Mode0Button.Text = 'Mode 0';
app.Mode0Button.Position = [10 4 62 22];
app.Mode0Button.Value = true;


% Create Mode1Button
app.Mode1Button = uiradiobutton(app.BPFMODEButtonGroup);
app.Mode1Button.Text = 'Mode 1';
app.Mode1Button.Position = [76 4 65 22];


% Create ADDSSNButtonGroup
app.ADDSSNButtonGroup = uibuttongroup(app.LeftPanel);
app.ADDSSNButtonGroup.SelectionChangedFcn = createCallbackFcn(app,
@ADDSSNButtonGroupSelectionChanged, true);
app.ADDSSNButtonGroup.Title = 'ADD SSN';
app.ADDSSNButtonGroup.Position = [19 238 155 49];


% Create YesButton
app.YesButton = uiradiobutton(app.ADDSSNButtonGroup);
app.YesButton.Text = 'Yes';
app.YesButton.Position = [9 2 50 22];


% Create NoButton
app.NoButton = uiradiobutton(app.ADDSSNButtonGroup);
app.NoButton.Text = 'No';
app.NoButton.Position = [75 2 43 22];
app.NoButton.Value = true;


% Create BANDSliderLabel
app.BANDSliderLabel = uilabel(app.LeftPanel);
app.BANDSliderLabel.HorizontalAlignment = 'right';
app.BANDSliderLabel.Position = [12 205 39 22];
app.BANDSliderLabel.Text = 'BAND';


% Create BANDSlider
app.BANDSlider = uislider(app.LeftPanel);
app.BANDSlider.Limits = [0 150];
app.BANDSlider.ValueChangedFcn = createCallbackFcn(app,
@BANDSliderValueChanged, true);
app.BANDSlider.Position = [61 215 114 3];
app.BANDSlider.Value = 4;


% Create FCUTSliderLabel
app.FCUTSliderLabel = uilabel(app.LeftPanel);
app.FCUTSliderLabel.HorizontalAlignment = 'right';
app.FCUTSliderLabel.Position = [13 158 37 22];
app.FCUTSliderLabel.Text = 'FCUT';
```

```
% Create FCUTSlider
app.FCUTSlider = uislider(app.LeftPanel);
app.FCUTSlider.Limits = [0 200];
app.FCUTSlider.ValueChangedFcn = createCallbackFcn(app,
@FCUTSliderValueChanged, true);
app.FCUTSlider.Position = [61 168 114 3];
app.FCUTSlider.Value = 50;


% Create SETTINGButton
app.SETTINGButton = uibutton(app.LeftPanel, 'push');
app.SETTINGButton.FontSize = 14;
app.SETTINGButton.Position = [12 353 174 38];
app.SETTINGButton.Text = 'SETTING';


% Create RightPanel
app.RightPanel = uipanel(app.GridLayout);
app.RightPanel.Layout.Row = 1;
app.RightPanel.Layout.Column = 2;


% Create UIAxes
app.UIAxes = uiaxes(app.RightPanel);
title(app.UIAxes, 'Sound Wave')
zlabel(app.UIAxes, ' ')
app.UIAxes.FontAngle = 'italic';
app.UIAxes.Position = [28 146 220 209];


% Create UIAxes2
app.UIAxes2 = uiaxes(app.RightPanel);
title(app.UIAxes2, 'Spectrum')
app.UIAxes2.FontAngle = 'italic';
app.UIAxes2.Position = [258 146 231 209];


% Create OpenButton
```

```
app.OpenButton = uibutton(app.RightPanel, 'push');
app.OpenButton.ButtonPushedFcn = createCallbackFcn(app, @OpenButtonPushed,
true);
app.OpenButton.Position = [66 90 100 23];
app.OpenButton.Text = 'Open';


% Create GenerateButton
app.GenerateButton = uibutton(app.RightPanel, 'push');
app.GenerateButton.ButtonPushedFcn = createCallbackFcn(app,
@GenerateButtonPushed, true);
app.GenerateButton.Position = [217 90 100 23];
app.GenerateButton.Text = 'Generate';


% Create SoundButton
app.SoundButton = uibutton(app.RightPanel, 'push');
app.SoundButton.ButtonPushedFcn = createCallbackFcn(app, @SoundButtonPushed,
true);
app.SoundButton.Position = [368 90 100 23];
app.SoundButton.Text = 'Sound';


% Show the figure after all components are created
app.UIFigure.Visible = 'on';
end
end


% App creation and deletion
methods (Access = public)


% Construct app
function app = ToneVocoder


% Create UIFigure and components
createComponents(app)
```

```
% Register the app with App Designer
registerApp(app, app.UIFigure)


if nargout == 0
clear app
end
end


% Code that executes before app deletion
function delete(app)


% Delete UIFigure when app is deleted
delete(app.UIFigure)
end
end
end
```