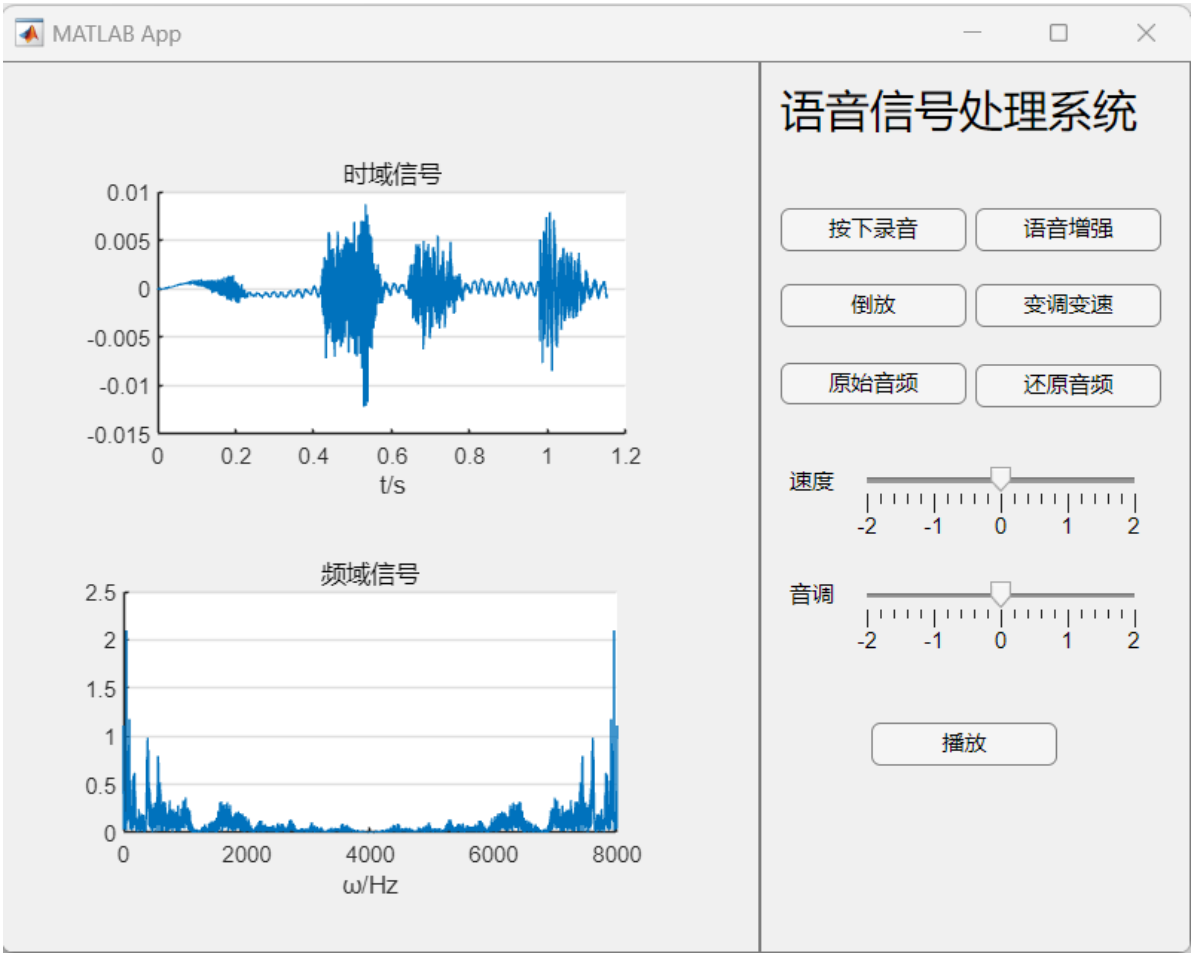
语音信号处理报告

---

1 概述

本次作业基于MATLAB R2023b - academic use中的App设计工具，实现了对语音信号的录制、播放、波形绘制、信号处理（语音增强、倒放、变速变调、变速不变调、变调不变速等）等功能，虽然语音重建之后的分辨率不高，但仍在人耳能分辨的范围内。

本程序的界面如下所示。



程序界面

本程序界面分为两部分。左边用于实时显示语音信号的时域波形和频域波形；右边是对语音信号的各种操作。

本程序的详细代码请参考提交的程序文件。

本程序的GUI完全由App designer自动生成，使用图形化的方式生成GUI，当然程序的界面是我自己设计的。

2 具体功能实现

本章介绍具体功能的实现代码。

2.0 几个变量和函数

首先介绍几个私有变量和函数（不包括回调函数）。

```
properties (Access = private)
        audioWave double     %原始的声音数组
        audioWaveBackUp double %原始音频的备份，前者用于处理
        time double           %储存时间信息
        timeSignal double    %储存时域上的信号
        omega double         %储存频率信息
        fqcSignal double      %储存频域上的信号
        FS = 44100           %采样频率
        recorder audiorecorder  %录音机
    end
```

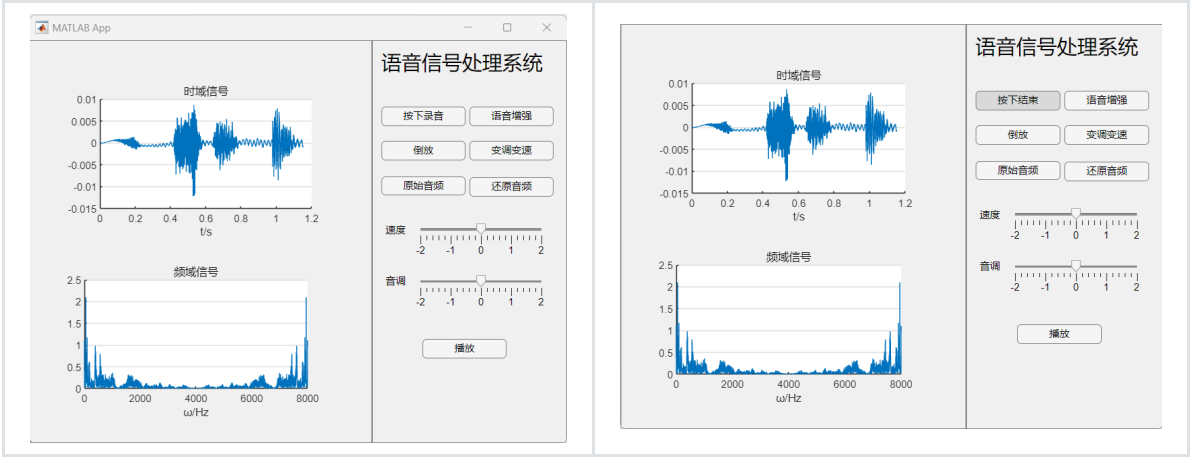
```
methods (Access = private)
        function startRecord(app)              %开始录音函数
        function store(app)                    %储存录音函数
        function drawMap(app)                  %画图函数
        function changeWave(app)               %创建时域和频域上的
%声音波形
        function enhanceSpeech(app)                %语音增强
        function speechproc(app,speech,mode,speed,pitch)
%语音重建和处理
        function PT=findpitch(~,s,WL)          %寻找基音周期
        function changeSpeed(app,speed)        %变速不变调
        function changePitch(app, pitch)       %变调不变速
        function invervse(app)                 %翻转音频
        function kidVoice(app)                 %变速变调，做着玩
end
```

2.1 录音与播放

2.1.1 录音与波形生成

实现了语音信号的录制（模拟信号转数字信号）。

在按下“按下录音”的状态按钮后开始录音，状态按钮文本变为“按下结束”，再次按下结束录音。



左边是没在录音，右边是正在录音。

回调函数代码如下：

```
% Value changed function: unOrRecordBotton
        function unOrRecordBottonValueChanged(app, event)
            value = app.unOrRecordBotton.Value;
            if value == 1
                app.unOrRecordBotton.Text = "按下结束";
                set(app.FrequencySlider, 'Value', 0);
                set(app.SpeedSlider, 'Value', 0);
                app.startRecord();
            end
            if value == 0
                app.unOrRecordBotton.Text = "按下录音";
                app.store();
                app.drawMap();
            end
        end
```

不按下时，Value=0，此时音频为空，故波形为空

第一次按下时，Value变为1，重置速度和音调条，调用startRecord函数

startRecord如下：

```
function startRecord(app)        %开始录音函数
            app.recorder = audiorecorder(app.FS,16,1);
            app.recorder.record();
end
```

初始化录音机并开始录音

第二次按下时，Value变为0，调用store和drawMap函数

store如下：

```
function store(app)            %储存录音函数
            app.recorder.pause();
            app.audioWave = app.recorder.getaudiodata();
            app.audioWaveBackUp = app.audioWave;
            app.changeWave();
end
```

通过停止录音，并将数字信号赋给app.audioWave，并留一个备份（作用后面会说），最后调用changeWave

changeWave如下：

```
function changeWave(app)         %创建时域和频域上的声音波形
            %时域
            fs = app.FS;
            app.time =
 linspace(0,length(app.audioWave)/fs,length(app.audioWave));
            app.timeSignal = app.audioWave;
            %频域
            fqcsignal = abs(fft(app.audioWave));
            app.fqcSignal = fqcsignal;
            app.omega = fs/length(app.audioWave)*
(0:length(app.audioWave)-1);
end
```

时域处理先计算录音时间，而后time和timeSignal两个数组的数据将是一一对应的

频域处理先对信号做快速傅里叶变换，再计算omega（一定小于采样频率），此后omega和fqcSignal一一对应

调用完store后调用drawMap函数，其代码如下：

```
function drawMap(app)            %画图函数
            plot(app.TimeMap,app.time,app.timeSignal);
            plot(app.FqcMap,app.omega,app.fqcSignal);
end
```

普通的画图函数，将波形赋给程序左边对应的图像区域

2.1.2 播放

实现对audioWave的播放，代码如下：

```
% Button pushed function: Display
        function DisplayButtonPushed(app, event)
            sound(app.audioWave,app.FS);
        end
```

普通的调用sound函数

2.2 语音增强

使用基于谱减法的语音增强功能，实测有一点效果。

谱减法的原理如下。

对于录制的语音信号，有

$$y(t) = s(t) + d(t) \tag{1}$$

其中$y(t), s(t), d(t)$分别是时域上的原始信号，语音信号和噪音信号

对式(1)做傅里叶变换，得到

$$Y(j\omega) = S(j\omega) + D(j\omega)$$

两边取模得到

$$|Y(j\omega)|^2 = |S(j\omega)|^2 + |D(j\omega)|^2 + 2\mathrm{Re}\{D^*(j\omega) \cdot S(j\omega)\}$$

由于噪音是随机的，故第三项在一段时间内对功率的贡献为零，得到功率的关系

$$|S(j\omega)|^2 = |Y(j\omega)|^2 - |D(j\omega)|^2$$

$$\Rightarrow |S(j\omega)| = \begin{cases} \sqrt{|Y(j\omega)|^2 - |D(j\omega)|^2} & |Y(j\omega)| > |D(j\omega)| \\ 0 \quad o.w. \end{cases} \tag{2}$$

在相位方面，由于噪音的随机性以及语音信号强度应该要显著大于噪音（否则我的水平不够），得到

$$\angle S(j\omega) \approx \angle Y(j\omega) \tag{3}$$

由式(2)(3)，得到重建后的语音信号

$$S(j\omega) = |S(j\omega)|e^{j\cdot\angle S(j\omega)} \tag{4}$$

对式(4)做傅里叶反变换，得到增强后的语音

代码如下：

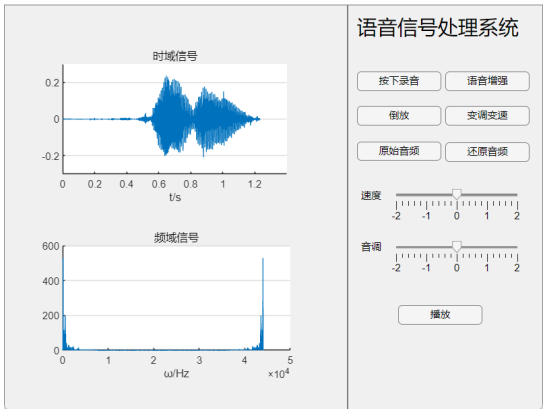
```
function enhanceSpeech(app)        %语音增强
            fs = app.FS;
            noiseAndSignal = app.audioWave;  %将原始信号作为带噪语音
            noise = noiseAndSignal(1:0.5*fs); %将前50ms的信号作为噪音
            fft_noiseAndSignal = fft(noiseAndSignal); %对带噪语音做fft
            phase_fft_noiseAndSignal = angle(fft_noiseAndSignal); %因为噪音相比语音
幅值小
```

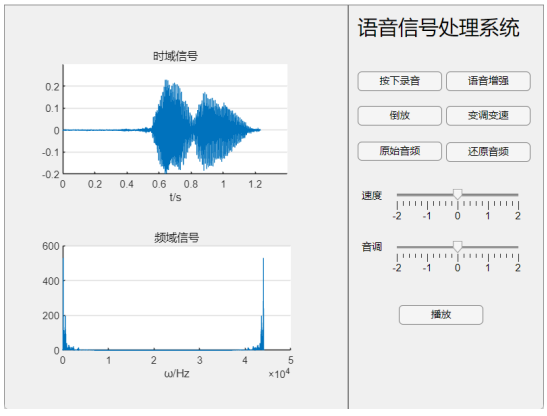
```
                                %对相位影响小，故认为相位不变
            fft_noise = fft(noise);                     %对噪音fft
            aveAmp = mean(abs(fft_noise));     %求噪音的平均幅度
            fft_signal_Amp = zeros(length(fft_noiseAndSignal),1);
            for i = 1:length(fft_noiseAndSignal)
                if abs(fft_noiseAndSignal(i))^2-aveAmp^2 > 0      %幅度不能小于0
                   fft_signal_Amp(i) = sqrt(abs(fft_noiseAndSignal(i))^2-
aveAmp^2);
%还原语音的幅度
                end
            end
            fft_signal = fft_signal_Amp .* exp(1i .* phase_fft_noiseAndSignal);%
还原相位
            signal = ifft(fft_signal);                                %还原信号
            app.audioWave = real(signal);

            app.changeWave();
            app.drawMap();
end
```

本程序使用按钮来执行上述程序，按钮的回调函数如下：

```
% Button pushed function: HenceSpeech
        function HenceSpeechButtonPushed(app, event)
            app.enhanceSpeech();
            app.HenceSpeech.Text = "增强成功";
            pause(1);            %显示一会
            app.HenceSpeech.Text = "语音增强";
        end
```

在按下按钮后波形变化，说明增强成功。





左边是处理前，右边是处理后，显然右边的毛刺更少。

2.3 变速不变调及变调不变速

基于LPC语音重建的变速不变调和变调不变速。

2.3.1 LPC语音重建和合成

LPC的基本思想是，一个语音取样的现在值可以用若干个语音取样过去值的线性加权组合来逼近，有以下公式

$$s(n) = \sum_{k=1}^{N} a_k s(n-k) + Ge(n) \qquad (5)$$

其中$s(n)$是当前时刻的语音，$s(n-k)$是过去时刻的语音，$Ge(n)$是当前时刻的激励（或者说误差）。

$$Ge(n) = s(n) - \sum_{k=1}^{N} a_k s(n-k) \qquad (6)$$

这意味着只需要知道系数a_k，就可以通过式(5)计算出激励$e(n)$，通过$e(n)$和上面的逆过程，保持a_k不变，就能重建出语音。

求解a_k用到了matlab自带的lpc函数。该函数能自动进行LPC方程求解，利用自相关法计算a_k。

代码如下：

```
function restructSpeech(app,speech)
            %定义常数
            FL = app.FS / 100;       %帧长(10ms)
            WL = 3 * FL;             %窗长
            P = 800;                 %预测系数个数(拉满了)
            s = speech;              %赋值
            L = length(s);
            FN = floor(L/FL) - 2;    %计算帧数
            last_syn = 0;            %记录计算到的帧数
            %预测和重建滤波器
            exc = zeros(L,1);        %预测误差e（激励信号）
            zi_pre = zeros(P,1);     %预测滤波器的状态
            s_rec = zeros(L,1);      %重建的语音
            zi_rec = zeros(P,1);     %重建语音滤波器状态
            %合成滤波器
            exc_syn = zeros(L,1);    %合成的激励信号
            s_syn = zeros(L,1);      %合成的语音
            zi_syn = zeros(P,1);     %合成滤波器的状态

            hw = hamming(WL);        %汉明窗
            for n = 3:FN
                %计算预测系数
                s_w = s(n*FL-WL+1:n*FL).*hw;%汉明窗加权后的语音
                %A是预测系数，E被用于计算合成激励的能量
                [A, E] = lpc(s_w,P);      %用LPC计算P个预测系数
                s_f = s((n-1)*FL+1:n*FL);    %处理这帧语言

                %计算激励
                %使用filter函数s_f计算激励，注意保持滤波器状态

                [exc((n-1)*FL+1:n*FL),zi_pre] =
                filter([0 -A(2:end)],1,s_f,zi_pre);
                %上面求出的激励就是这帧的激励
                %由于lpc()默认A(1)=1,故去除第一项
                %根据公式(6)，在计算exc时aK是减去的，故-A


                %重建语言
                %使用filter函数和exc重建语言，注意保持滤波器状态
                [s_rec((n-1)*FL+1:n*FL),zi_rec] =
                filter(1,A,exc((n-1)*FL+1:n*FL),zi_rec);
                        %上面重建的语言就是这帧的重建语言
```

```
                        %根据公式(5)
                        %与exc不同的是滤波器的分子分母也要相反

                    %不需要掌握的东西，抄的老师给的资料
                    s_Pitch = exc(n*FL-WL+1:n*FL);
                    PT = app.findpitch(s_Pitch,WL);%计算基音周期PT
                    G = sqrt(E*PT);            %计算合成激励的能量G

                    index = (1:n*FL-last_syn);
                    exc_syn1 = zeros(length(index),1);
                    exc_syn1(mod(index,PT)==0) = G;%用周期信号作为激励
                    exc_syn((n-1)*FL+1:n*FL) =
                    exc_syn1((n-1)*FL-last_syn+1:n*FL-last_syn);
                                        %计算得到的合成激励
                    [s_syn((n-1)*FL+1:n*FL),zi_syn] =
                    filter(1,A,exc_syn((n-1)*FL+1:n*FL),zi_syn); %得到合成语音
                    last_syn = last_syn+PT*floor((n*FL-last_syn)/PT);
                                        %计算本次算到的帧数，通过能容纳多少个基音周期

                end
                app.audioWave = s_syn;
                app.changeWave();
                app.drawMap();
            end
```

详细见注释。这是调试程序时用的函数，提交的程序中也许被我注释，也许被我删除。

findpitch函数用于寻找基音周期，如下：

```
function PT=findpitch(~,s,WL)
            [B,A] = butter(5,700/4000);
            s = filter(B,A,s);
            R = zeros(WL-1,1);
            for k = 1:WL-1
                R(k) = s(WL:end)' * s(WL-k:end-k);
            end
            [~,T] = max(R);
            PT = T;
end
```

2.3.2 变速不变调

由2.3.1我们已经合成（不是重建）了原始语言，虽然分辨率比较低但是可以接受。

变速不变调语音的合成无非就是在合成时保持s_Pitch，PT和a_k，并且加长每一帧持续的时间。

本程序在该功能上使用滑块来控制速度，回调函数如下：

```
% Value changed function: SpeedSlider
        function SpeedSliderValueChanged(app, event)
            value = -app.SpeedSlider.Value;
            speed = 2 ^ value;
            app.changeSpeed(speed);
            app.changeWave();
            app.drawMap();
        end
```

changeSpeed函数如下：

```
        function changeSpeed(app,speed)
            app.speechproc(app.audioWave,1,speed,1);
        end
```

speechproc函数其实和restructSpeech异曲同工，放在后面介绍。

2.3.3 变调不变速

利用2.3.2得到的变速不变调语音合成代码，可以很方便的合成可认读的变调不变速语音。

本程序在该功能上使用滑块来控制音调（频率），回调函数如下：

```
% Value changed function: FrequencySlider
        function FrequencySliderValueChanged(app, event)
            value = -app.FrequencySlider.Value;
            pitch = 2 ^ value;
            app.changePitch(pitch);
            app.changeWave();
            app.omega = app.omega * pitch;  %为了画频谱
            app.drawMap();
        end
```

changePitch函数如下：

```
function changePitch(app, pitch)
            app.speechproc(app.audioWave,2,1,pitch);
            app.speechproc(app.audioWave,1,1/pitch,1);
end
```

第一次调用speechproc函数，使语音变速变调；第二次调用该函数，使语音变速不变调。效果相当于变调不变速。

2.3.4 speechproc函数

这是本程序实现变速不变调和变调不变速的关键函数，其代码如下：

```
function speechproc(app,speech,mode,speed,pitch)
            %定义常数
            FL = app.FS / 100;       %帧长
            WL = 3 * FL;             %窗长
            P = 800;                 %预测系数个数
            s = speech;              %赋值
            L = length(s);
            FN = floor(L/FL) - 2;    %计算帧数

            %预测和重建滤波器
            exc = zeros(L,1);        %预测误差e（激励信号）
            zi_pre = zeros(P,1);     %预测滤波器的状态

            %变调不变速滤波器
            s_syn_f = zeros(L,1);    %合成语音

            %变速不变调滤波器
            exc_syn_s = zeros(ceil(speed*L),1); %变速激励
```

```
            s_syn_s = zeros(ceil(speed*L),1);    %合成语音
            last_syn_s = 0;                      %记录位置
            zi_syn_s = zeros(P,1);               %合成滤波器

            hw = hamming(WL);        %汉明窗

            %依次处理每帧语言
            if mode == 1    %变速不变调
                for n = 3:FN
                    %计算预测系数
                    s_w = s(n*FL-WL+1:n*FL) .* hw;
                    %汉明窗加权后的语音
                    %A是预测系数，E被用于计算合成激励的能量
                    [A, E] = lpc(s_w,P);  %用LPC计算P个预测系数
                    s_f = s((n-1)*FL+1:n*FL);    %处理这帧语言
                    %计算激励
                    %使用filter函数s_f计算激励，注意保持滤波器状态

                    [exc((n-1)*FL+1:n*FL),zi_pre] =
                    filter([0 -A(2:end)],1,s_f,zi_pre);
                    %上面求出的激励就是这帧的激励

                    %下面的代码只有在得到exc后才能计算正确
                    s_Pitch = exc(n*FL-WL+1:n*FL);
                    PT = app.findpitch(s_Pitch,WL);
                    %计算基音周期PT
                    G = sqrt(E*PT);        %计算合成激励的能量G



                    %不改变基音周期和预测系数，将合成激励的长度变为原先的speed倍
                    %作为filter的输入得到新的合成语音，达到变速不变调
                    FL_s = ceil(FL*speed);
                    index_s = (1:n*FL_s-last_syn_s)';
                    exc_syn1_s = zeros(length(index_s),1);
                    exc_syn1_s(mod(index_s,PT)==0) = G;
                    %用周期脉冲信号作为激励，保持PT不变
                    exc_syn_s((n-1)*FL_s+1:n*FL_s) =
                    exc_syn1_s((n-1)*FL_s-last_syn_s+1:n*FL_s-last_syn_s);
                    %计算得到的加长合成激励
                    [s_syn_s((n-1)*FL_s+1:n*FL_s),zi_syn_s] =
                    filter(1,A,exc_syn_s((n-1)*FL_s+1:n*FL_s),zi_syn_s);
                    %计算得到的加长合成语音
                    last_syn_s = last_syn_s+PT*floor((n*FL_s-last_syn_s)/PT);  %
更新位置

                end
            else                            %变调不变速
                %改变采样频率，得到变速变调的语音，再用变速不变调的算法还原速度
                fs = ceil(app.FS*pitch);
                s_syn_f = resample(app.audioWave,fs,app.FS);
            end

            if mode == 1
                app.audioWave = s_syn_s / max(s_syn_s);
            else
                app.audioWave = s_syn_f / max(s_syn_f);
            end
```

```
            app.changeWave();
            app.drawMap();
        end
```

主要功能与实现方式和2.3.1的restructSpeech类似。输入参数mode表示当前使用的模式，mode=1表示变速不变调，mode=2表示变调又变速。

在处理变速不变调时，使用FL_s = ceil(FL*speed)将每一帧变为原先的speed倍，后续处理和合成语音相同。

在处理变调不变速时，通过调用两次该函数，达到简单处理结果，分辨率不高就是了。

2.4 附加小功能

增加了倒放，变速变调，播放原始音频和还原音频的小功能，挺好玩的。

2.4.1 倒放

按钮的回调函数如下：

```
% Button pushed function: inverse
        function inverseButtonPushed(app, event)
            app.invervse();
            app.inverse.Text = "倒放成功";
            pause(1);
            app.inverse.Text = "倒放";
        end
```

inverves函数如下：

```
function invervse(app)
            app.audioWave = flip(app.audioWave);
            app.changeWave();
            app.drawMap();
end
```

直接用flip翻转语音数组就好，用于古神语解密。

2.4.2 变速变调

按钮的回调函数如下：

```
% Button pushed function: kidsVoice
        function kidsVoiceButtonPushed(app, event)
            app.kidsVoice.Text = "小孩版";
            app.kidVoice();
            pause(1);
            app.kidsVoice.Text = "变速变调";
        end
```

kidVoice函数如下：

```
function kidVoice(app)
            sound(app.audioWave, 2*app.FS);
end
```

直接用2倍的采样频率播放语音，用于纪念故人，播放see you again。

2.4.3 播放原始音频

按钮的回调函数如下：

```
% Button pushed function: playOrigin
        function playOriginButtonPushed(app, event)
                    app.playOrigin.Text = "正在播放";
                    sound(app.audioWaveBackUp, app.FS);
                    app.playOrigin.Text = "原始音频";
        end
```

直接播放原始音频，方便我对比调试语音的处理效果

2.4.4 还原音频

按钮的回调函数如下：

```
% Button pushed function: undoAll
        function undoAllButtonPushed(app, event)
            app.undoAll.Text = "正在还原";
            app.audioWave = app.audioWaveBackUp;
            set(app.SpeedSlider, 'Value', 0);
            set(app.FrequencySlider, 'Value', 0);
            app.changeWave();
            app.drawMap();
            pause(0.5);
            app.undoAll.Text = "还原音频";
        end
```

将事先备份的原始音频赋给语音数组，并将所有设置归零。这个功能在我调试程序时大放异彩，主要是重建后的语音分辨率太低，要是不还原的话一直重建根本听不清。