5-BAND EQUALIZER

FINAL WORK *FINAL VERSION*

Lucas Rodés
MAE Spring 2015

Table of Contents

1. INTRODUCTION

This project consists on a digital programmed mix table which includes an equalizer and some audio effects with the aim to manipulate and/or improve the sound quality of a certain signal. This signal could be e.g. either for music production or for communication transmission purposes.

1.1. Equalizer

As already mentioned in the title of this project, the equalizer will have 5 different frequency bands:

- Low
- Low-Mid
- Mid
- Mid-High
- High

This structure is used in many different mix turntables (production, live events). Nevertheless, good turntables might have more bands. DJ's turntables usually only have 3 different bands (i.e. low, mid, high), however *Allen&Health* – a well-known audio company – usually designs its DJ's turntables with 4 bands.

The first step is to choose the 5 different bands of the equalization and their respective cutoff frequencies, as well as their attenuation factor. Each band corresponds to a different filter. The 1st band is a low pass filter, the 5th is a high pass filter while the rest of the bands (2nd ,3rd and 4th) are band pass filters.

The design of the equalizer is shown below:

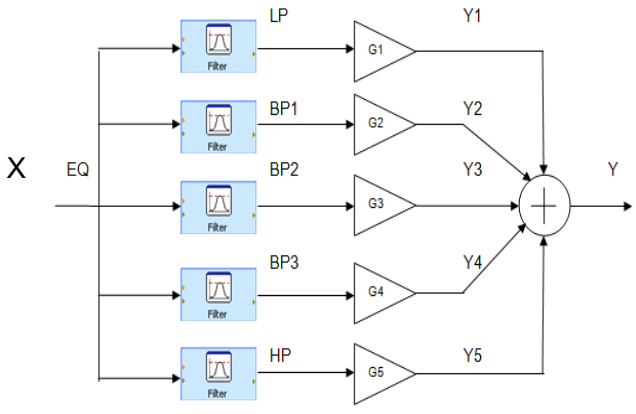

Figure 1 - Schematic of the equalizer

To make clear the picture shown above, we will explain what the different parameters are: The signal X is the input of the system, i.e. the signal to be equalized; $G1$, $G2$, $G3$, $G4$ and $G5$ are the gains of each band in dBs; $Y1$, $Y2$, $Y3$, $Y4$ and $Y5$ are the five different filtered signals for each band and Y is the sum of all the previous mentioned signals, i.e. the output of the system (equalization of the signal X).

To precise and go into depth in the main concepts of the design of the equalizer, below is shown a table containing the specifications of the five filters used in the equalization:

	Type	$f_{stop_1}(Hz)$	$f_{pass_1}(Hz)$	$f_{pass_2}(Hz)$	$f_{stop_2}(Hz)$	Atenuation $G\ (dB)$
1st	Low pass	400	450	-	-	60
2nd	Band pass	300	400	1500	1700	40
3rd	Band pass	1400	1500	6000	6100	40
4th	Band pass	5900	6000	10000	10100	60
5th	High pass	9900	10000	-	-	80

Figure 4.- Values of the 5 filters of the Equalizer

The criteria to design the filters is basically is stability, i.e. all of them have to be stable. To achieve this we have used the MATLAB tool called `fdatool`, which allows to easily designing a filter choosing its fundamental parameters; such as its stop and pass frequencies (two times if we are designing a band pass filter) and the attenuation for each band pass and the attenuation for the band stop (filtered band).

As already commented before, all the values shown in Figure 1 are rigorously selected in order to achieve the stability of the designed filter, which in fact is not easy to achieve for high attenuation values.

1.2. Audio effects

As a final extension of the project I have included six audio effects:

- White noise
- Reverb
- Delay
- Pitch
- Guitar Distortion
- Gater

All of them have been implemented using MATLAB software. They can be either applied to a recorded signal or to a loaded audio file. Note that if the audio file is too big the processing time will be higher.

In the following I will try to explain each of the effects and their mathematical background. The effects will be applied to the equalized signal previously denoted as Y.

White noise

This effect consists on adding white noise over the equalized signal Y:

$$Z_1[n] = Y[n] + n[n]$$

where $n \sim \mathcal{N}(0, \sigma^2)$ and σ^2 denotes the noise power. The user will be able to modify the value of σ^2 by increasing or decreasing the SNR value.

$$SNR = \frac{P}{\sigma^2}$$

where $P = \sum_{n=1}^{N} |Y[n]|^2$ and N is the size of the vector $Y[n]$.

Reverb

The reverb effect tries to imitate the room effect, i.e. the reflections of a room when playing some audio signal. This phenomenon is very remarkable in big buildings like for instance churches. We will apply this effect to the signal $Z_1[n]$ by using an impulse response $h_{rev}[n]$:

$$Z_2[n] = Z_1[n] * h_{rev}[n]$$

There are lots of different ways of applying reverb, by choosing different impulse responses. In my case I have chosen an exponential decreasing signal with 10 coefficients:

$$h_{rev}[n] = \sum_{k=0}^{9} \alpha^k \, \delta[n - kD]$$

where the value of α is selected by the user and $D = 3500$, which means a delay of $73ms$ if we are working with a sampling frequency of $f_s = 48kHz$. The criterion to select this value was by trying different values until I achieved the desired sound.

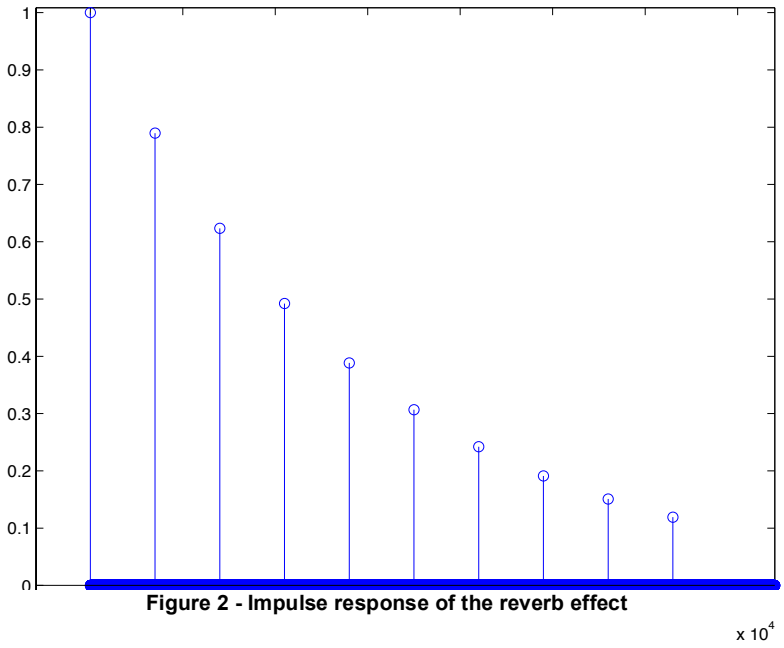

Figure 2 - Impulse response of the reverb effect

Delay

The delay effect was done really similar to the reverb effect. It basically consists on adding the input signal and a delayed version of itself.

$$Z_3[n] = Z_2[n] + Z_2[n - D]$$

This system can be again defined by an impulse response:

$$Z_3[n] = Z_2[n] * h_{del}[n]$$

where $h_{del}[n] = \delta[n] + \delta[n - D]$.

In this case the user selects the value of the delay, i.e. D.

Pitch

Increasing the pitch means increasing the musical note of the audio signal. This could be done by, for instance, multiplying the input signal with a sine wave at the desired central frequency. However, doing this the final result was not what I expected. Therefore I tried other algorithms, which involved working in the frequency domain. The results where really good, but the executing time was too high. Therefore, I finally decided to vary the sampling frequency. The results using this technique were fine, but they also shortened the signal length. For example, if I change the sampling frequency $F_s \rightarrow \frac{F_s}{2}$ the pitch decreases but the signal lasts two times the original time.

Since this effect did not take along executing time I ended up using this scheme.

$$Z_4[n] = Z_3[n]]_{F_s \to a \cdot F_s}$$

where the value of a is selected by the user.

Guitar Distortion

This effect consists on transforming the input signal in order to obtained a distorted version:

$$Z_5[n] = \frac{\left(1 + \frac{2b}{1-b}\right) \cdot Z_4[n]}{1 + \frac{2b}{1-b} \cdot Z_4[n]}$$

where the value of b was selected by the user. Note that $b \in [-0.99, 0.99]$.

The achieved distortion tries to imitate the sound of an overdriven guitar amplifier. The algorithm was found on www.musicdsp.org.

Gater

Finally, this effect consists on removing the signal intermittently. The following diagram tries to show the scheme:

Where *M* refers to both the number of samples that are conserved and discarded. We can model this behavior as:

$$Z_6[n] = Z_5[n] \cdot p[n]$$

where $p[n]$ is a rectangular pulse train with period M and duty cycle $DT = 50\%$.

2. SYSTEM GUI IMPLEMENTATION DOCUMENTATION

Since this program has a GUI, I will try to explain first the GUI design and later on I will explain into detail the code.

The first step is to call the software:

```
>> kadel_mt
```

The following window appears:

The following step is to press the “Initialize”.

The mix turntable appears, which is shown on the next picture.

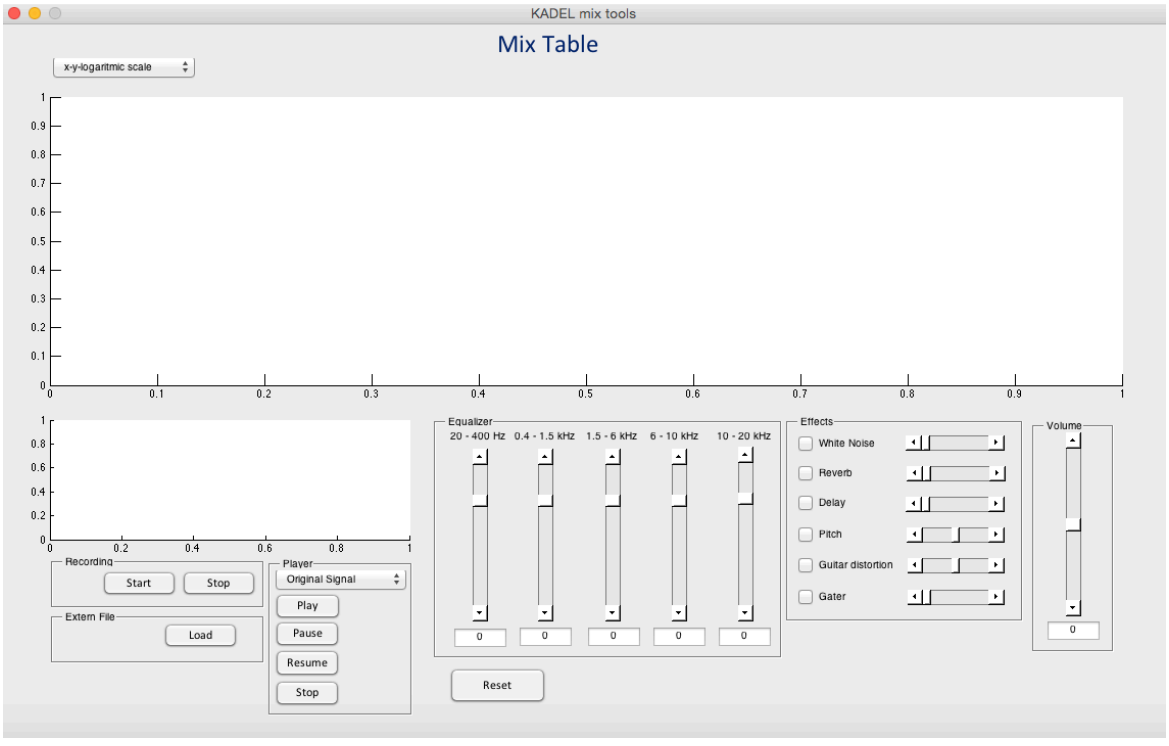

Figure 3 - Program Interface, Mix table including the equalizer and the effects pannel

On the following we explain what each field stands for.

- In the lower left corner we find the panel to **load the signal** that will be equalized. We have two options: Recording an audio signal pressing 'start' and 'stop' buttons or load an audio file from your library.

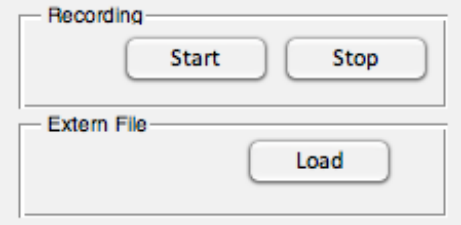

Figure 4 - Recording panel

- Next to the previous panel we find the **player panel**. In this panel we can play the current recorded signal or the already mixed one by choosing it from the menu above it. If the signal still has not been equalized, the same signal is played for both options

Figure 5 - Player Panel

Figure 6 - Popupmenu for selecting original/equalized signal

- Right next to the player panel we find the **equalizer panel**. It consists, as already explained, on 5 different bands. Each band has a maximum gain of 10 dB and a maximum attenuation of 20 dB.

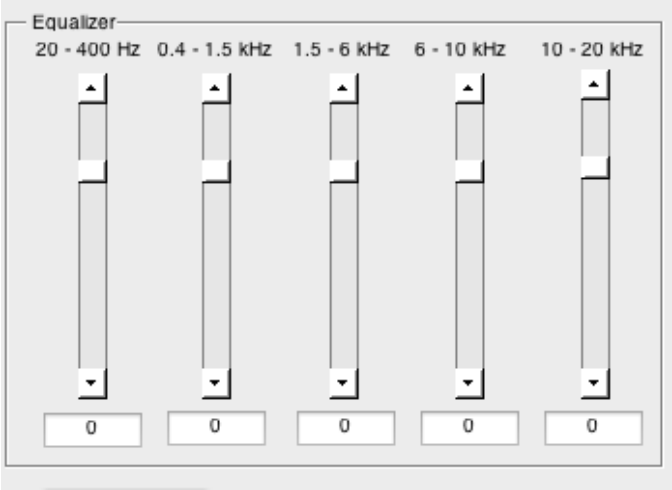

Figure 7 - Equalizer panel

- Next to it we find the **effect panel**. This panel contains the six effects previously mentioned and a slide bar for each of them to select their respective values. Note that there is also a check box for each effect. So, if the check box is not marked, the effect will not apply.

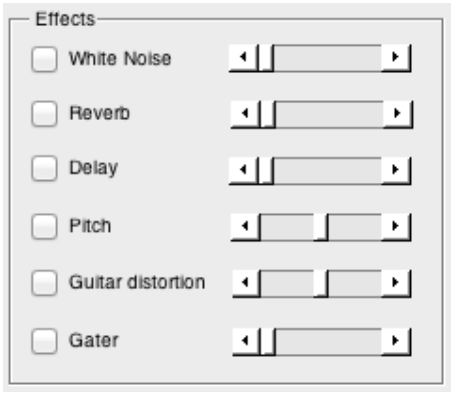

Figure 8 - Effect panel

- Finally there is the **volume slide**, which allows us to increase or decrease the volume of the mixed signal.

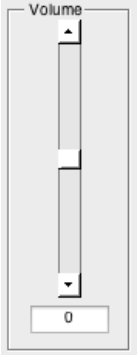

Figure 9 - Volume slide

To test the correct work of the equalizer we will load a music file.

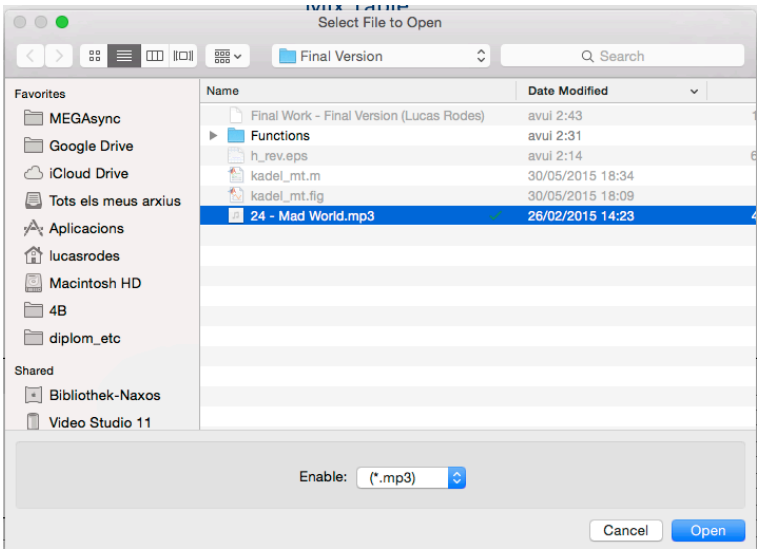

Figure 10 – Load window

After loading the music file, we are able to equalize the signal and play with the effects. An example of mix, involving equalization and effects, could be the following one:

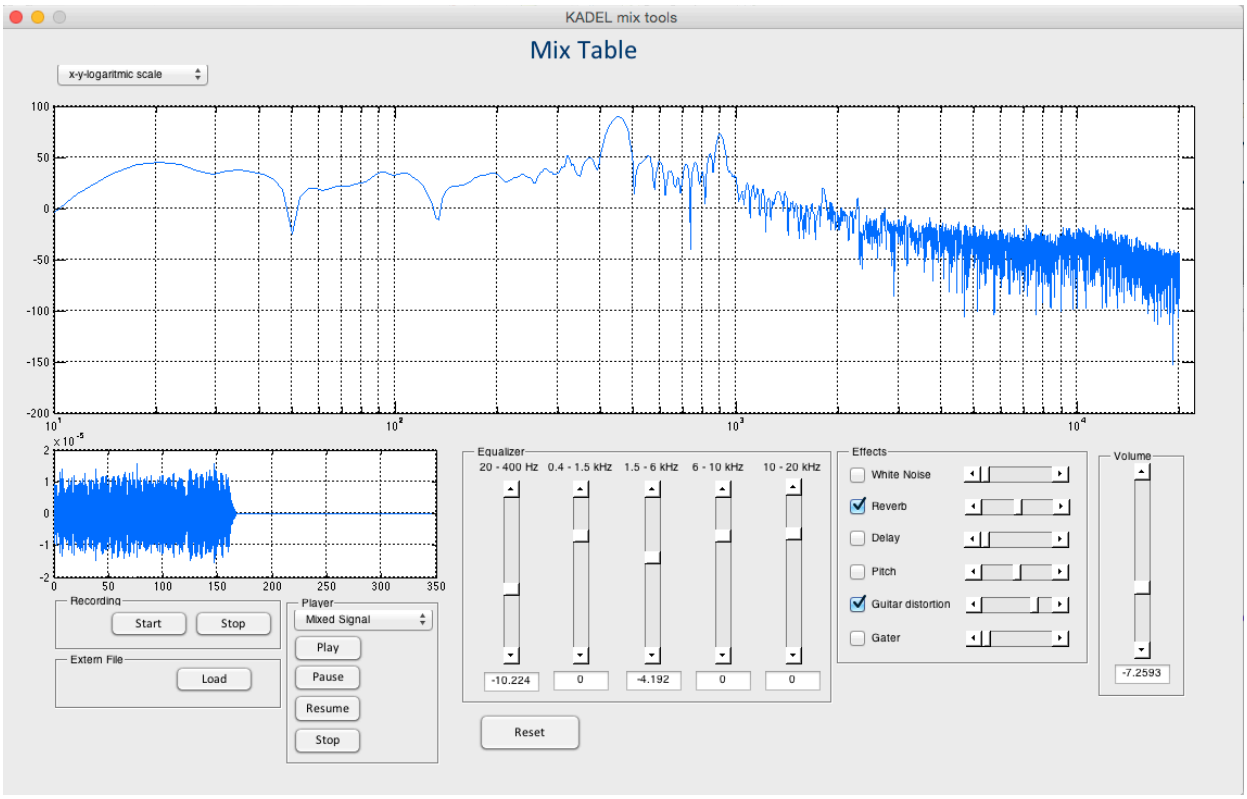

Figure 11 - Program interface with a mixed signal

In the upper site we can see the spectrum of the loaded signal. Note that the spectrum has been modified by the attenuation values of the five different filters and the effects.
In the lower left version we can see the plot of the mixed signal, since the “mixed signal” option is selected.

Note that the spectrum of the signal can be plotted in multiple ways, as the following screenshot shows.

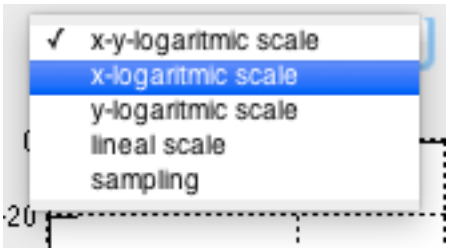

Figure 12 - Options to plot the spectrum of the signal

2. SYSTEM CODE IMPLEMENTATION DOCUMENTATION

Since there are a lot of handles to consider, we will discuss only one for each group. On the following I will try to explain the functionality of each handle. Furthermore I will also attach the code of the callback of each handle.

It is important to say that there are some generic functions that will be explained at the end:

- `mix`
- `reset`
- `plotspectrum`

All of them are called from different callbacks.

Furthermore, `handle.metricdata` is used in order to share variables between callbacks and, in case it is needed in the future, with other GUIs.

First of all is important to say that some variables have been created in the opening function, i.e. `EQUALIZER1_OpeningFcn(…)`.

```
function EQUALIZER1_OpeningFcn(hObject, eventdata, handles, varargin)
      handles.output = hObject;
      guidata(hObject, handles);

      warning ('off','all');
      w1 = warndlg('Initializing...');
      reset(hObject,handles);
      Fs=48e3; nB=16; nC=1; % Define basic recording parameters
      recObj = audiorecorder(Fs,nB,nC); % Define the recording object

      % share with other GUIs
      handles.metricdata.recObj = recObj;
      guidata(hObject,handles);
      close(w1);
```

This way, we achieve more efficiency in the executing time.

- **Text Edits**

  - *Equalization*

    These are the text edit boxes located under the 6 sliders (5 bands and volume) of the program interface. The user can introduce the attenuation factor for each band using them. The slider position will be updated automatically.

```
function textBAND1_Callback(hObject, eventdata, handles)
        set(handles.sliderBAND1,'Value',str2double(get(handles.text
        BAND1,'string')));
        mix(hObject,handles);
```

    Note that there is also a text edit box for the volume.

- **Sliders**

  - *Equalization*

    These have the same function as the text edits explained above. The values of the text edits are updated also automatically. The user can use both options.

```
function sliderBAND1_Callback(hObject, eventdata, handles)
        s1 = (get(handles.sliderBAND1,'Value'));
        set(handles.textBAND1,'String',num2str(s1));
        try
                mix(hObject,handles);
        catch err
        end
```

  - *Effects*

    The effects only have the slider bars as representatives. Therefore their values are selected by moving the slide. If the user wants to enable the effect the associated checkbox has to be marked.

- **Buttons**

  - *Recording*

    One option is to record yourself on the microphone and use the recorded signal. In order to do it, there are two buttons to start and stop the recording. Both buttons have a Callback that creates an audiorecorder object, which is shared between functions. Function `record` and `stop` are used to accomplish it. Note that a sampling rate is needed, in this case $f_s = 48\ kHz$.

```
function buttonSTARTREC_Callback(hObject, eventdata, handles)
        record(handles.metricdata.recObj); % record voice

        w_s = warndlg('Recording...'); % warning dialog
        % share recordedSignal with other GUIs
        handles.metricdata.w_s = w_s;
        guidata(hObject,handles)
```

As already said before, the signal is stored in the `audiorecorder` object called `recObj`. This variable is shared with other callbacks using `handles.metricdata`.

Note that once the signal has been stopped from recording, it is automatically plotted both in the upper axes (spectrum) and the lower left axes (time domain signal). To plot the spectrum function `plotspectrum` is used.

An `audioplayer` variable is defined associated to the recorded signal. At the end the signal and the created `audioplayer` variable is stored using `handles.metricdata`, so that the signal can be easily managed by the play panel.

```
function buttonSTOPREC_Callback(hObject, eventdata, handles)
    try
        stop(handles.metricdata.recObj); % stop recording
        reset(hObject,handles);
        recordedSignal = (getaudiodata(handles.metricdata.recObj))'; % obtain vector from recorded
            file
        Fs = 48e3; % Sampling rate

        % share recordedSignal with other GUIs
        handles.metricdata.recordedSignal = recordedSignal;
        guidata(hObject,handles)
        recordedSignal = handles.metricdata.recordedSignal;

        % share Fs with other GUIs
        handles.metricdata.Fs = Fs;
        guidata(hObject,handles);
        Fs = handles.metricdata.Fs;

        close(handles.metricdata.w_s);
        % plot recorded signal
        axes(handles.axes3);
        plot((1:length(recordedSignal))/Fs,recordedSignal/Fs);
        grid on;

        mode = get(handles.popmenuSPECTRUM,'Value'); % Obtain mode of plotting the spectrum of the
            signal
        plotspectrum(recordedSignal,handles.axes2,mode,Fs); % Plot the spectrum of the signal

        % share recorded signal with other GUIs
        handles.metricdata.signal = recordedSignal;
        handles.metricdata.mixedSignal = recordedSignal; % initially equal to the input signal
        guidata(hObject,handles);
        recordedSignal = handles.metricdata.signal;

        player_sig = audioplayer(recordedSignal, Fs); % Define audioplayer for the recorded signal

        % share player with other GUIs
        handles.metricdata.player_sig = player_sig;
        handles.metricdata.player_eq = player_sig; % Set player of mixed signal equal original signal
        guidata(hObject,handles);

    catch err
    end
```

- *Loading*

  Another option is to load an audio file using the load button. The file can be of multiple formats (including .wav, .mp3 etc.). To load the signal `uigetfile` function is used. To extract the audio content from the file `audioread` is used. Note that this function returns both the signal vector and its sampling frequency. The rest of the code is very similar to the callback of the stop button.

```
function buttonLOAD_Callback(hObject, eventdata, handles)
        try
            % Load file
            filename = '';
            [filename,pathname] = uigetfile({'*.mp3';'*.wav';'*.ogg';'*.flac';'*.au';'*.m4a';'*.mp4'});
            if isequal(filename,'')
                w_l = warndlg('Loading...');
            end
            [loadedSignal,Fs] = audioread(strcat(pathname,filename));

            loadedSignal = loadedSignal(:,1);
            reset(hObject,handles)

            % share loadedSignal with other GUIs
            handles.metricdata.loadedSignal = loadedSignal;
            guidata(hObject,handles);
            loadedSignal = handles.metricdata.loadedSignal;

            % share Fs with other GUIs
            handles.metricdata.Fs = Fs;
            guidata(hObject,handles);
            Fs = handles.metricdata.Fs;

            try
                close(w_l);
            catch err
            end

            axes(handles.axes3);
            plot((1:length(loadedSignal))/Fs,loadedSignal/Fs);
            grid on;

            mode = get(handles.popmenuSPECTRUM,'Value'); % Obtain mode of plotting the spectrum of the signal
            plotspectrum(loadedSignal(:,1),handles.axes2,mode,Fs); % Plot the spectrum of the signal

            % share loaded signal with other GUIs
            handles.metricdata.signal = loadedSignal;
            handles.metricdata.mixedSignal = loadedSignal; % initially equal to the input signal
            guidata(hObject,handles);
            loadedSignal = handles.metricdata.signal;

            % Define and share player object
            player_sig = audioplayer(loadedSignal, Fs); % Define audioplayer for the recorded signal
            handles.metricdata.player_sig = player_sig;
            handles.metricdata.player_eq = player_sig; % Set player of mixed signal equal original signal
            guidata(hObject,handles);

        catch err
        end
```

- *Playing*

  In this section we find the Play, Pause, Resume and Stop buttons. All of them share a very similar code structure.
  Depending on the selected value in the pop up menu above the panel they will manage either the original signal or the equalized signal.

```
function buttonSTOP_Callback(hObject, eventdata, handles)
        w_b = warndlg('Wait...');
        original_or_mixed = get(handles.popmenuFILESELEC,'Value');
        switch original_or_mixed
            case 1
                play(handles.metricdata.player_sig);
                close(w_b);
            otherwise
                play(handles.metricdata.player_eq);
                close(w_b);
        end
```

```
function buttonPLAY_Callback(hObject, eventdata, handles)
        original_or_mixed = get(handles.popmenuFILESELEC,'Value');
            switch original_or_mixed
                case 1
                     stop(handles.metricdata.player_sig);
               otherwise
                     stop(handles.metricdata.player_eq);
         end
```

```
function buttonPAUSE_Callback(hObject, eventdata, handles)
        original_or_equalized = get(handles.popmenuFILESELEC, 'Value');
        switch original_or_equalized
            case 1
                pause(handles.metricdata.player_sig);
            otherwise
                pause(handles.metricdata.player_eq);
        end
```

```
function buttonRESUME_Callback(hObject, eventdata, handles)
        original_or_equalized = get(handles.popmenuFILESELEC, 'Value');
        switch original_or_equalized
            case 1
                resume(handles.metricdata.player_sig);
            otherwise
                resume(handles.metricdata.player_eq);
        end
```

- *Reset*

  This button resets the whole interface. To do it, it calls the function `reset`, which will be explained later.

```
function buttonRESET_Callback(hObject, eventdata, handles)
        reset(hObject,handles);
```

- Axes

- *Spectrum*

  The axes of the spectrum is called axes2. Note the multiple different options of plot (popupmenu above the axes).

- *Time domain*

  This axes are called axes3, and can plot two different signals: the original loaded/recorded signal or the equalized signal.

- Checkboxes

When the software detects that the user has activate or deactivate the checkbox of a certain effect, this one is applied (i.e. the `mix` function is called).

```
function check_gater_Callback(hObject, eventdata, handles)
      mix(hObject,handles);
```

- Pop Up menu

- *Spectrum mode*

  With this popup menu the user can choose the modality of the plot of the signal spectrum. Note that depending on the selected value in the popup menu of the time domain (explained below) either the original or the equalized signal's spectrum will be plotted.

```
function popmenuSPECTRUM_Callback(hObject, eventdata, handles)
        original_or_mixed = get(handles.popmenuFILESELEC, 'Value');
        w = warndlg('Wait...');

        switch original_or_mixed
             case 1 % Original signal
                 signal = handles.metricdata.signal;
                 mode = get(handles.popmenuSPECTRUM,'Value');
                 plotspectrum(signal,handles.axes2,mode,Fs);
             otherwise % mixed signal
                 mixedSignal = handles.metricdata.mixedSignal;
                 mode = get(handles.popmenuSPECTRUM, 'Value');
                 plotspectrum(mixedSignal,handles.axes2,mode,Fs);
        end

         close(w);
```

- *Time domain*

  This popup menu selects which signal is being plotted in both axes.

```
function popmenuFILESELEC_Callback(hObject, eventdata, handles)
        w = warndlg('Wait...');
        original_or_mixed = get(handles.popmenuFILESELEC, 'Value');
        Fs = 48e3;

         try
             switch original_or_mixed
                 case 1
                    signal = handles.metricdata.signal;
                    axes(handles.axes3);
                    plot((1:length(signal))/Fs,signal/Fs); grid on;
                    mode = get(handles.popmenuSPECTRUM, 'Value');
                    plotspectrum(signal,handles.axes2,mode,Fs);
                 otherwise
                    mixedSignal = handles.metricdata.mixedSignal;
                    axes(handles.axes3);
                    plot((1:length(mixedSignal))/Fs,mixedSignal/Fs); grid on;
                    mode = get(handles.popmenuSPECTRUM, 'Value');
                    plotspectrum(mixedSignal,handles.axes2,mode,Fs);
             end
        catch err
        end
        close(w);
```

- Auxiliar functions:

- *Mix*

  This is the core of the software. This function follows the structure presented in Figure 1.

  When the signal has been equalized it is stored again using `handles.metricdata`. Furthermore an associated `audioplayer` object is also created and shared using `handles.metricdata`.

```
function mix(hObject,handles)

        w_m = warndlg('Mixing signal...');
        % Start with the Equalization %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Obtain gains
        g(1) = 10^(str2double(get(handles.textBAND1, 'string'))/20);
        g(2) = 10^(str2double(get(handles.textBAND2, 'string'))/20);
        g(3) = 10^(str2double(get(handles.textBAND3, 'string'))/20);
        g(4) = 10^(str2double(get(handles.textBAND4, 'string'))/20);
        g(5) = 10^(str2double(get(handles.textBAND5, 'string'))/20);
        g(6) = 10^(str2double(get(handles.textMAIN, 'string'))/20);

        % Sampling freq
        Fs=handles.metricdata.Fs;

        %stop frequencies
        % Band 1:
        fc(1)= 400;
        % Band 2:
        fc(2)=1500;
        % Band 3:
        fc(3)=6000;
        % Band 4:
        fc(4)=10000;
        % Band 5:
        fc(5)=20000;

       signal = handles.metricdata.signal;


          % BAND FILTERS
          % Band 1 Filter:
          [b1, a1] = tf(design(fdesign.lowpass(fc(1),450, 1, 60, Fs), 'ellip', 'MatchExactly',
       'both'));
          y1 = g(1)*filter(b1,a1,signal);
          % Band 2 Filter:
          [b2, a2] = tf(design(fdesign.bandpass(300, fc(1), fc(2),1700, 40, 1, 40, Fs 'ellip',
       'MatchExactly', 'both'));
          y2 = g(2)*filter(b2,a2,signal);
          % Band 3 Filter:
          [b3, a3] = tf(design(fdesign.bandpass(1400, fc(2), fc(3),6100, 40, 1, 40, Fs 'ellip',
       'MatchExactly', 'both''));
          y3 = g(3)*filter(b3,a3,signal);
          % Band 4 Filter:
          [b4, a4] = tf(design(fdesign.bandpass(5900, fc(3), fc(4), 10100, 60, 1, 60, Fs),
       'ellip', 'MatchExactly', 'both'));
          y4 = g(4)*filter(b4,a4,signal);
          % Band 5 Filter:
          [b5, a5] = tf(design(fdesign.highpass(9900, fc(4), 80, 1, Fs), 'ellip', 'MatchExactly',
       'both'));
          y5 = g(5)*filter(b5,a5,signal);

          % Equalized signal
          mixedSignal = g(6)*(y1+y2+y3+y4+y5);

       % Add the Effects %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


          % NOISE
          if get(handles.check_whitenoise,'Value')
              snr = 50 - get(handles.slider_white,'Value'); % get effect value
              disp(num2str(snr));
              mixedSignal = awgn(mixedSignal,snr,'measured');
          end
```

```
% REVERB
  if get(handles.check_reverb,'Value')

      a = get(handles.slider_reverb,'Value');  % get effect value

      % Define Reverb filter
      D = 3500; % delay
      h_rev = zeros(size(mixedSignal));
      for i=0:9
          h_rev(1+i*D) = a^i;
      end

      % Convolution
      mixedSignal = conv(mixedSignal,h_rev);
  end

  % DELAY
  if get(handles.check_delay,'Value')
      D = round(get(handles.slider_delay,'Value'));  % get effect value
      if D == 0
          D = 1;
      end
      h_del = zeros(size(mixedSignal));
      h_del(1) =1;
      h_del(D) = 1;

      % Convolution
      mixedSignal = conv(mixedSignal,h_del);
  end

  % PITCH
  if get(handles.check_pitch,'Value')
      a = sqrt(2)^(get(handles.slider_pitch,'Value'));  % get effect value
      Fs = a*Fs;
  end

% DISTORTION
  if get(handles.check_distortion,'Value')
      b = get(handles.slider_distortion,'Value');  % get effect value
      k = 2*b/(1-b);
      mixedSignal = (1+k)*(mixedSignal)./(1+k*abs(mixedSignal));
  end


  % GATER
  if get(handles.check_gater,'Value')
      k = get(handles.slider_gater,'Value');

      if k ~=0

          mystep = 0.01*2^(round((1-k)*5))*Fs;

          cont = 1;
          while((length(mixedSignal)-cont)>=mystep)

              mixedSignal(cont:cont+mystep) = 0;
              cont = cont + 2*mystep;
          end
      end

  end

 % Update spectrum plot
  mode = get(handles.popmenuSPECTRUM,'Value'); % Obtain mode of plotting the spectrum of
the signal
  plotspectrum(mixedSignal,handles.axes2,mode,Fs);

 % Update time domain plot
  mode = get(handles.popmenuFILESELEC,'Value'); % Obtain mode of plotting the time domain
of the signal
  axes(handles.axes3);
```

```
if mode == 1
    plot((1:length(signal))/Fs,signal/Fs); grid on;
else
    plot((1:length(mixedSignal))/Fs,mixedSignal/Fs); grid on;
end

% share mixed signal with other GUIs
handles.metricdata.mixedSignal = mixedSignal ;
guidata(hObject,handles);
mixedSignal=handles.metricdata.mixedSignal;

% Define and share player object
player_eq = audioplayer(mixedSignal, Fs); % Define audioplayer for the recorded signal
handles.metricdata.player_eq = player_eq;
guidata(hObject,handles);

close(w_m);
```

In this function we observe the different implementations of the effects.

For example, the factor for the pitch is computed as $\sqrt{2}^{n}$, where n is the value introduced by the user. Since this number can be a negative value, the pitch will be decreased.

- *Reset*

  This function resets all the values. This means all the text edit boxes become empty, the axes are cleared and the sliders are brought to their original position.

```
function reset(hObject,handles)
        set(handles.sliderBAND1, 'Value',0);
        set(handles.sliderBAND2, 'Value',0);
        set(handles.sliderBAND3, 'Value',0);
        set(handles.sliderBAND4, 'Value',0);
        set(handles.sliderBAND5, 'Value',0);
        set(handles.sliderMAIN, 'Value',0);
        set(handles.slider_white, 'Value',0);
        set(handles.slider_reverb, 'Value',0);
        set(handles.slider_delay, 'Value',0);
        set(handles.slider_pitch, 'Value',0);
        set(handles.slider_distortion, 'Value',0);
        set(handles.slider_gater, 'Value',0);
        set(handles.textBAND1, 'String',0);
        set(handles.textBAND2, 'String',0);
        set(handles.textBAND3, 'String',0);
        set(handles.textBAND4, 'String',0);
        set(handles.textBAND5, 'String',0);
        set(handles.textMAIN, 'String',0);
        cla(handles.axes2,'reset');
        cla(handles.axes3, 'reset');
        set(handles.check_whitenoise, 'Value',0);
        set(handles.check_reverb, 'Value',0);
        set(handles.check_delay, 'Value',0);
        set(handles.check_pitch, 'Value',0);
        set(handles.check_distortion, 'Value',0);
        set(handles.check_gater, 'Value',0);

        set(handles.popmenuFILESELEC,'Value',1);
        try
                stop(handles.metricdata.player_sig);
                stop(handles.metricdata.player_eq);
        catch err
        end
```

- *Plotspectrum*

  This function plots the spectrum of the original/equalized signal in the selected mode (popup menu *Spectrum mode*).

```
function plotspectrum(input,t,mode,Fs)
          N = 2^14; k=0:N-1; f= (Fs/N).*k;X = fft(input,N);
          axes(t);
          switch mode
              case 1
                  semilogx(f(1:20000/(Fs/N)),20*log(abs(X(1:20000/(Fs/N)))));  xlim([10 22000]);
                  grid on;
              case 2
                  semilogx(f(1:20000/(Fs/N)),(abs(X(1:20000/(Fs/N)))));
                  grid on;
              case 3
                  plot(f(1:20000/(Fs/N)),20*log(abs(X(1:20000/(Fs/N)))));
                  grid on;
              case 4
                  plot(f(1:20000/(Fs/N)),(abs(X(1:20000/(Fs/N)))));
                  grid on;
              otherwise
                  stem(f(1:20000/(Fs/N)),(abs(X(1:20000/(Fs/N)))));
                  grid on;
          end
```