(SCHOOL OF ELECTRONICS AND COMMUNICATION ENGINNERING)

VIT® UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ■ CHENNAI
www.vit.ac.in

J component Final Report

ECE2006

DIGITAL SIGNAL PROCESSING

IIR BASED PARAMETRIC AUDIO EQUALIZER

SUBMITTED BY:

YATHARTH ASTHANA : 17BEC0502

PROJIT DEB : 17BEC0623

SPARSH ARYA : 17BEC0656

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our DSP teacher, Dr.Prakasam P, who gave us this golden opportunity to do this outstanding project on IIR Based Parametric Audio Equalizer, which also helped us in doing lot of research on this topic and as a result of which we got to learn lot of new things which helped us sharpen our concepts pertaining to the subject.

Secondly, we would like to thank our parents and friends who helped us a lot in completing this project in the limited span of time.

We are making this project, not only for the marks but also to develop skills and gain knowledge of the subject. Thank you again for all those who helped us in completing this project.

ABSTRACT

A tale technique for the adjustment of amplifiers and other sound frameworks utilizing IIR (vast drive reaction) parametric filters are introduced. The primary quality of the proposed channel plan strategy dwells in the way that the levelling structure is arranged from the earliest starting point as a chain of SOSs (second-request segments), where every so is a low-pass, high-pass, or pinnacle channel, characterized by its parameters. The calculation joins an immediate hunt technique with a heuristic parametric improvement process where limitations on the qualities could be forced so as to acquire down to earth executions. A psychoacoustic model dependent on the recognition of pinnacles and plunges in the recurrence reaction has been utilized to figure out which ones should be evened out, lessening the channel request without observable impact. The first processed areas of the planned channel are the ones that right the reaction all the more successfully, permitting versatile arrangements when equipment confinements exist or various degrees of amendment are required. The strategy has been approved with abstract testing and contrasted and different techniques. Its outcomes could be applied to inactive and multiway active loudspeakers.

TABLE OF CONTENT

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

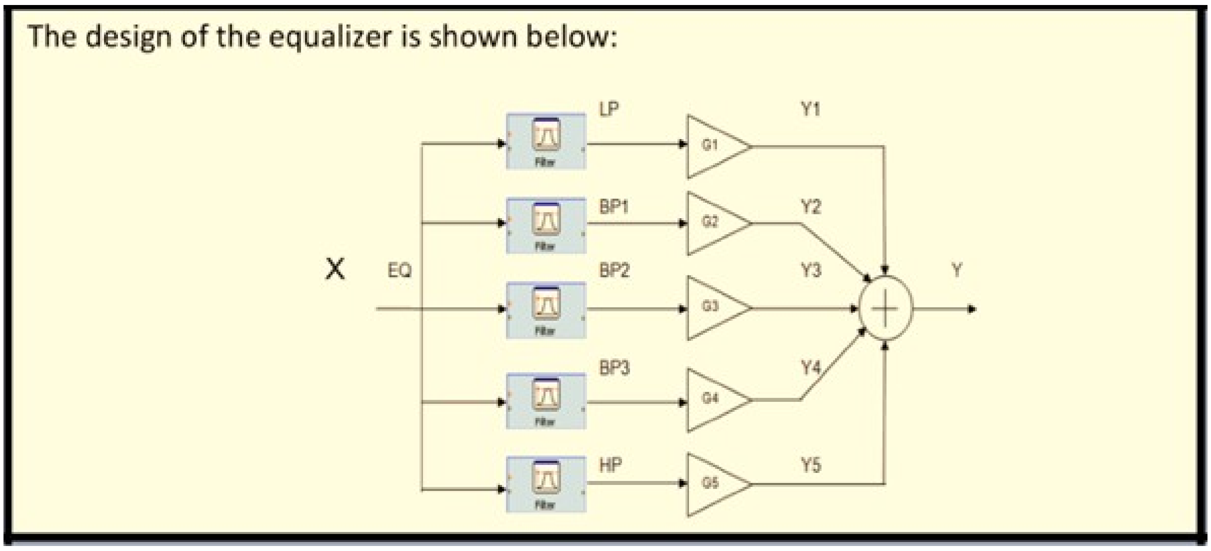
-Low-Mid

-Mid

-Mid-High

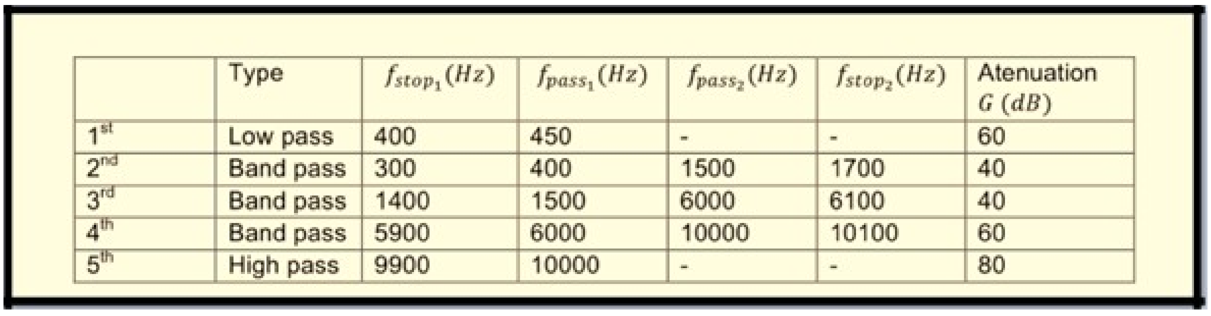
-High

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:

To make clear the picture shown above, we will explain what the different parameters are:

The signal 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 mainconcepts 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

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

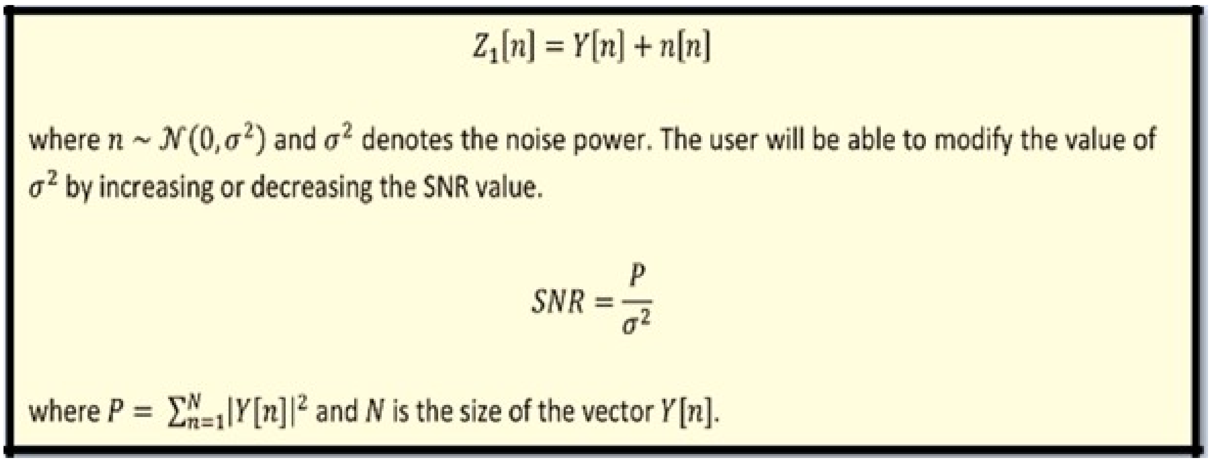
As a final extension of the project we have included six audio effects:

- 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.

1.3 WHITE NOISE

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



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

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 $Y[n]$.

1.4 REVERB

The reverb effect tries to imitate the room effect, i.e. the reflections of a room when playing someaudio signal. This phenomenon is very remarkable in big buildings like for instance church.

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

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:

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

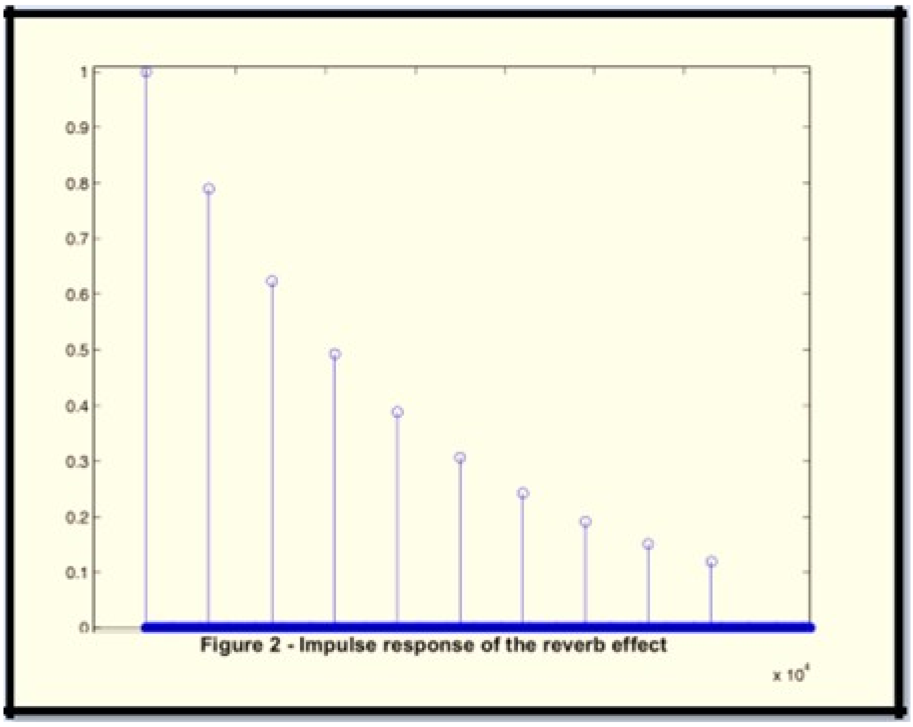
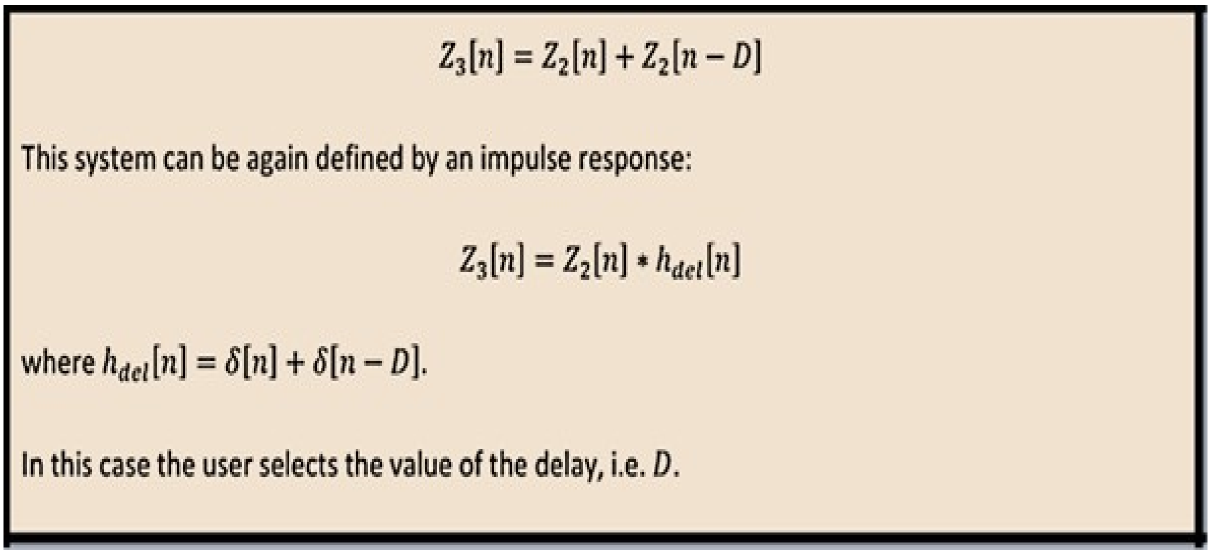


Figure 2 - Impulse response of the reverb effect

1.5 DELAY

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:

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

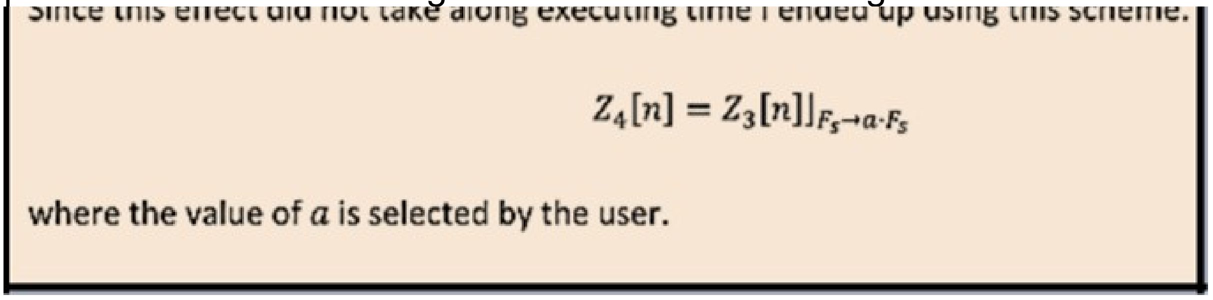
where $h_{del}[n] = \delta[n] + \delta[n-D]$.

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

1.6 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 we expected. Therefore we tried other algorithms, which involved working in the frequency domain. The results were really good, but the executing time was too high. Therefore, we finally decided to vary the sampling frequency. The results using this technique were fine, but they also shortened the signal length. For example, if we change the sampling frequency s→Fs/2 the pitch decreases but the signal lasts two times the original time.

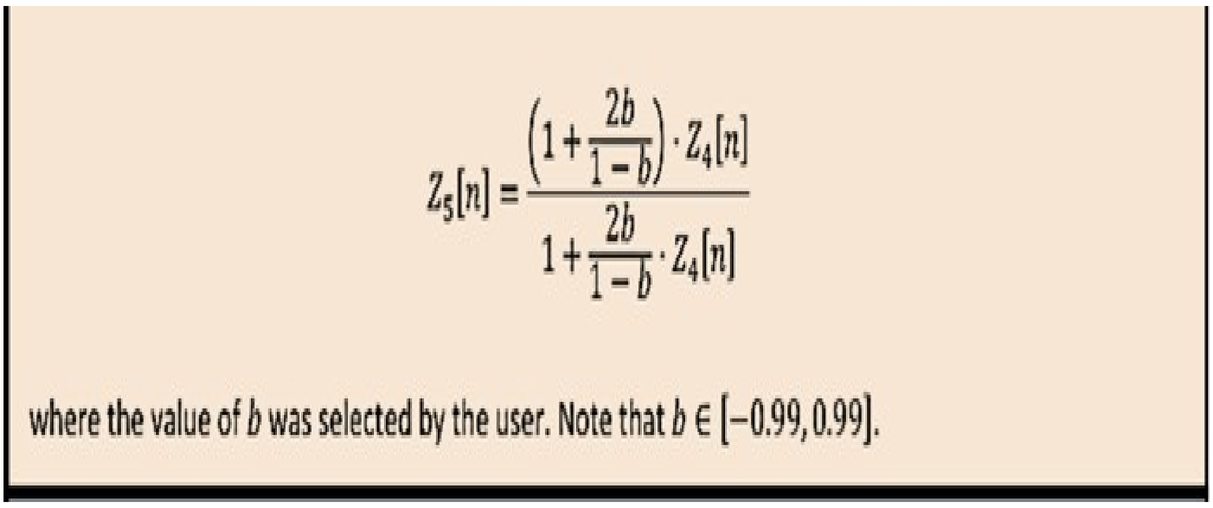


Since this effect did not take along executing time I ended up using this scheme.

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

where the value of a is selected by the user.

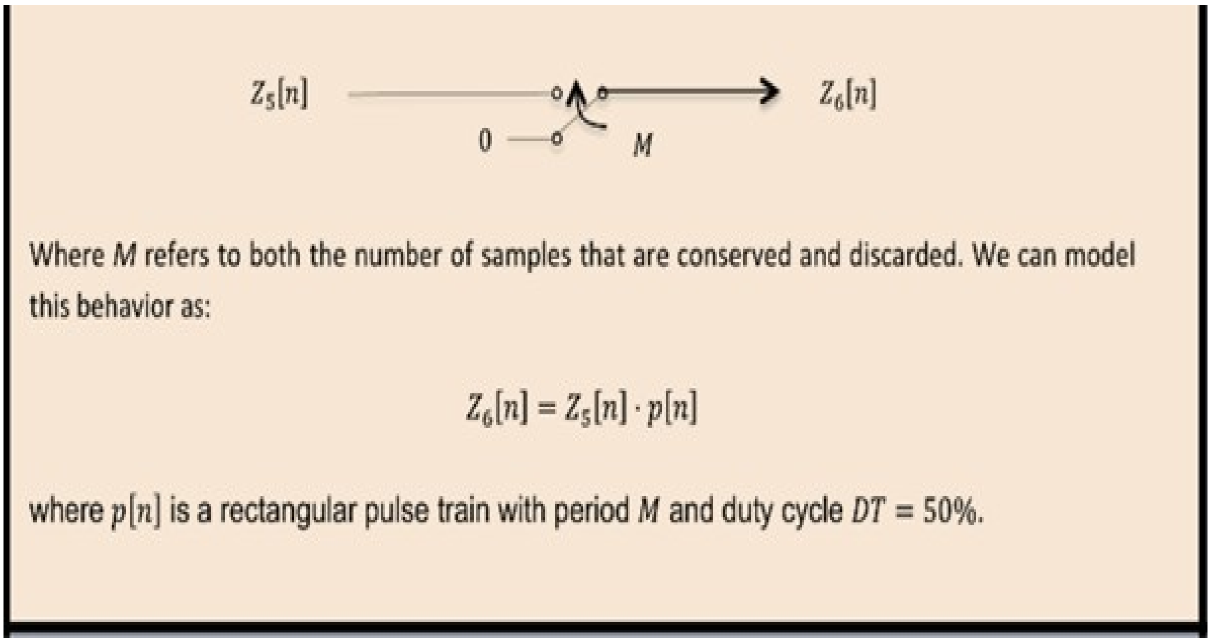
1.7 GUITAR DISTORTION

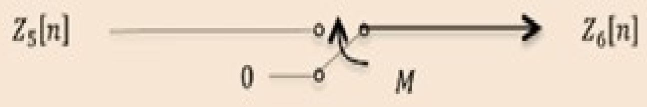


$$Z_5[n] = \frac{\left(1+\frac{2b}{1-b}\right)\cdot Z_4[n]}{1+\frac{2b}{1-b}\cdot Z_4[n]}$$

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

1.8 GATER





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.)<u>METHODOLOGY</u>

A parametric equalizer gives a finite arrangement of channels that clients can tune. Each channel is in actuality a second request infinite impulse response(IIR) filter in which the coefficients are determined as indicated by given parameters.The exactness and strength of IIR channels rely upon their topology. A decent topology deals with constraining aggregator flood, and limiting mistake input in the structure.In a parametric equalizer, five sorts of channels are ordinarily required: shelving (low-rack and high-rack),peaking and band-pass (low-pass and high-pass) filters. They are described in terms of gain (for cresting and racking filters), focal frequency (for topping filters),mid-point recurrence (for racking filters) or cut-off frequency (for band-pass filters), and quality factor.

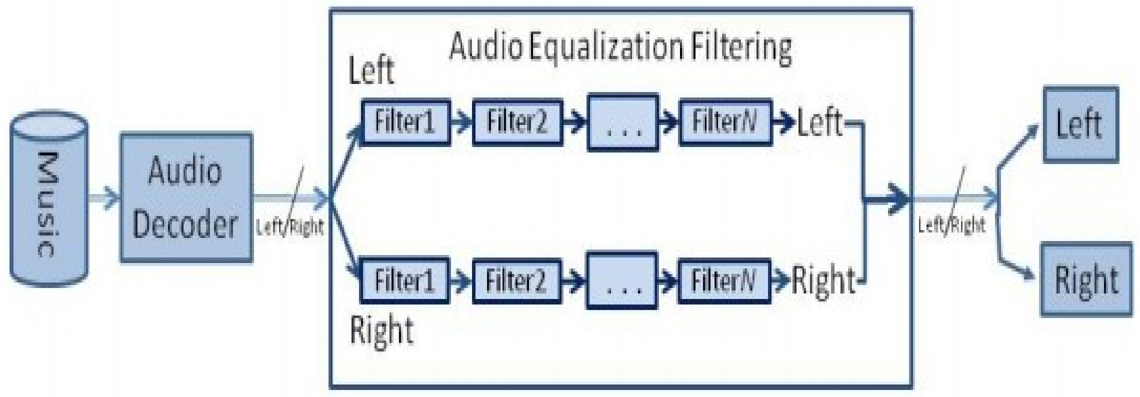
Graphic equalizer filters modify the unearthly substance of music by separating left and right sound system flags through a bank of equal channels, adding the outcomes to make the yield left and right signals. Each channel passes some portion of the sign's range. Included together again in the wake of sifting the left/right signals are reproduced with altered ghastly substance. Expanding the increase of a channel will underline the sign in that channel's recurrence band. Diminishing the increase of a channel will de-underline the sign in that channel's recurrence band.

The Audio Decoder now takes crude parallel music and translates it into a flood of 16 or 24 piece whole numbers, with a couple of such numbers speaking to a solitary left/right example of music. Normally these examples are played at 44,100 or 48,000 example for each second. Signs from the left or right channel are taken care of into a bank of equal band filters. The yield of each channel is increased by a client flexible increase and afterward added together to make a left or right yield signal, which is then sent by means of a DAC to speakers.

Each channel yield is utilized to refresh the sign quality of each channel's yield. It is this information that is shown on the Graphic Equalizer screen.

Filters for each band are intended to cover over the sign's range. The sign’s range can be separated into similarly estimated band or into groups that expansion by a factor of two with each higher recurrence band.

Looking at the sifting structure of a parametric equalizer uncovers why the sign quality presentation is absent. As observed underneath, parametric evening out is cultivated by a chain of consecutive channels, with each channel having 0 dB (solidarity) increase outside of determined frequency range.



3.)**<u>RESULTS AND DISCUSSIONS</u>**

Contrasted with FIR filters, IIR filters model size reaction all the more effectively with less computational unpredictability, however they should be structured cautiously to ensure soundness. The most significant phase of planning an IIR equalizer is amplifier displaying. The steadiness of the amplifier's IIR model must be ensured, and the request for the IIR model ought to be appropriately picked to diminish computational intricacy.

As a matter of first importance, the request for the amplifier's IIR model can be handily structured. Second, both the base stage part and the abundance stage segment can be balanced. Third, the solidness of the IIR model determined by this technique is ensured. Besides, since there isn't any versatile hunt strategies remembered for the calculation, combination issues can be stayed away from. At last, the blunder between the amplifier's reaction and its model can be effortlessly controlled, with the goal that distinctive channel requests can be picked in various applications dependent on the resistance of the mistake.The structure of a multi-band equalizer starts by getting the information sign and addition esteems for each channel.

Hence the sound quality of a certain signal can be improved.

SOURCE CODE:

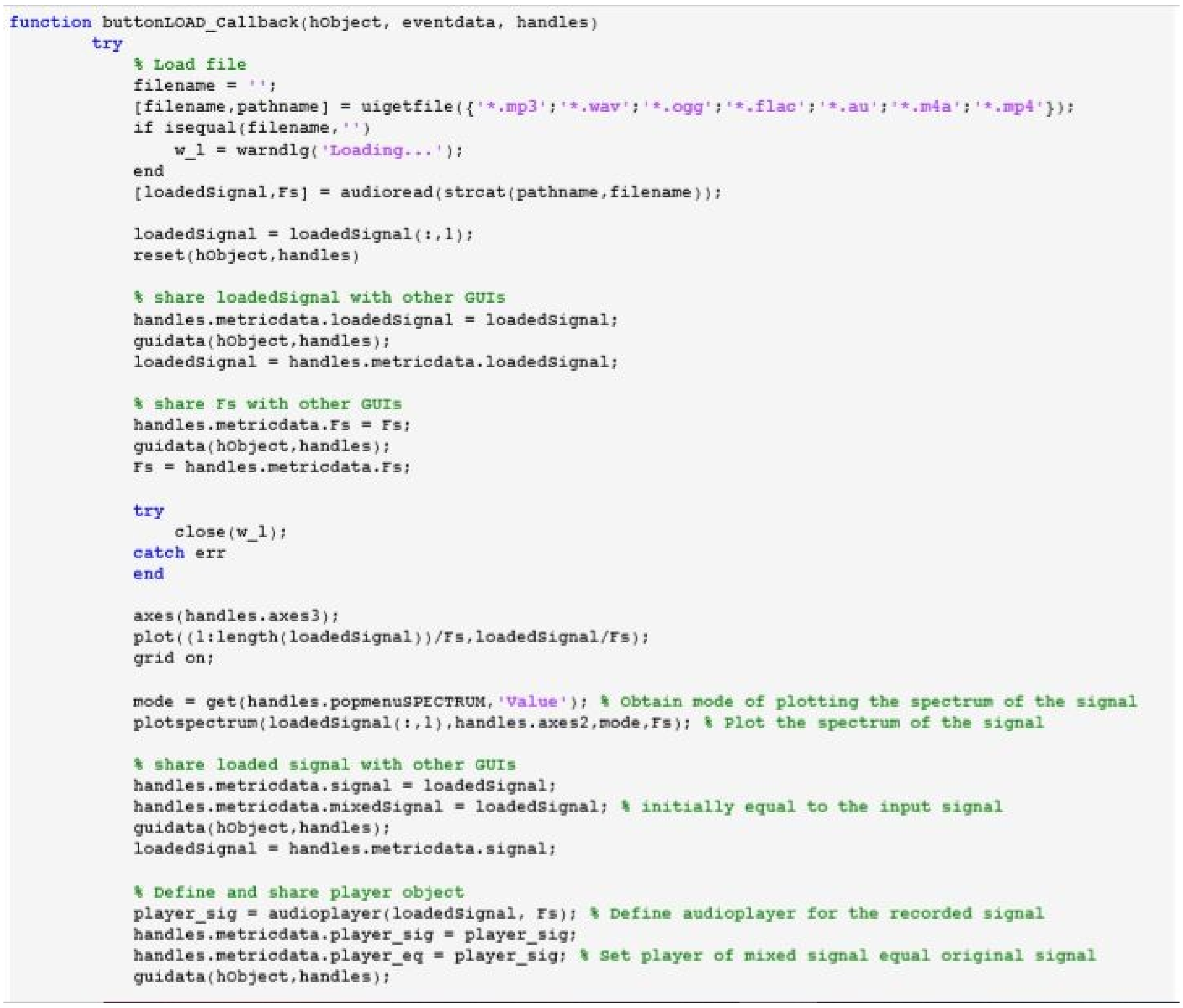
```
function EQUALIZER1_OpeningFcn(hObject, eventdata, handles, varargin)
      handles.output = hObject;
      guidata(hObject, handles);

      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);
```

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

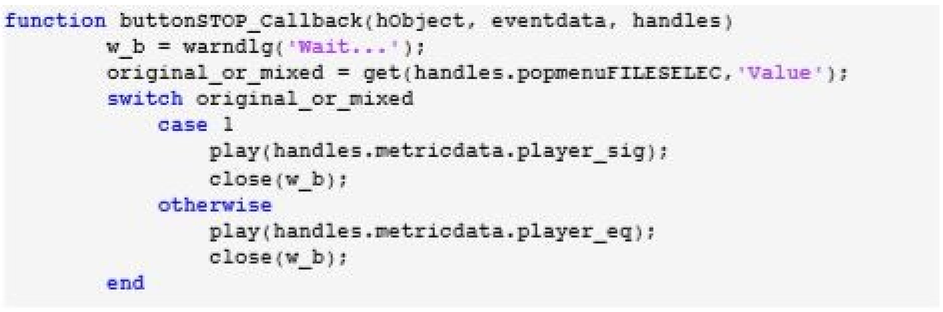
```
function sliderBAND1_Callback(hObject, eventdata, handles)
        s1 = (get(handles.sliderBAND1,'Value'));
        set(handles.textBAND1,'String',num2str(s1));
        try
                mix(hObject,handles);
        catch err
        end
```

```
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)
```

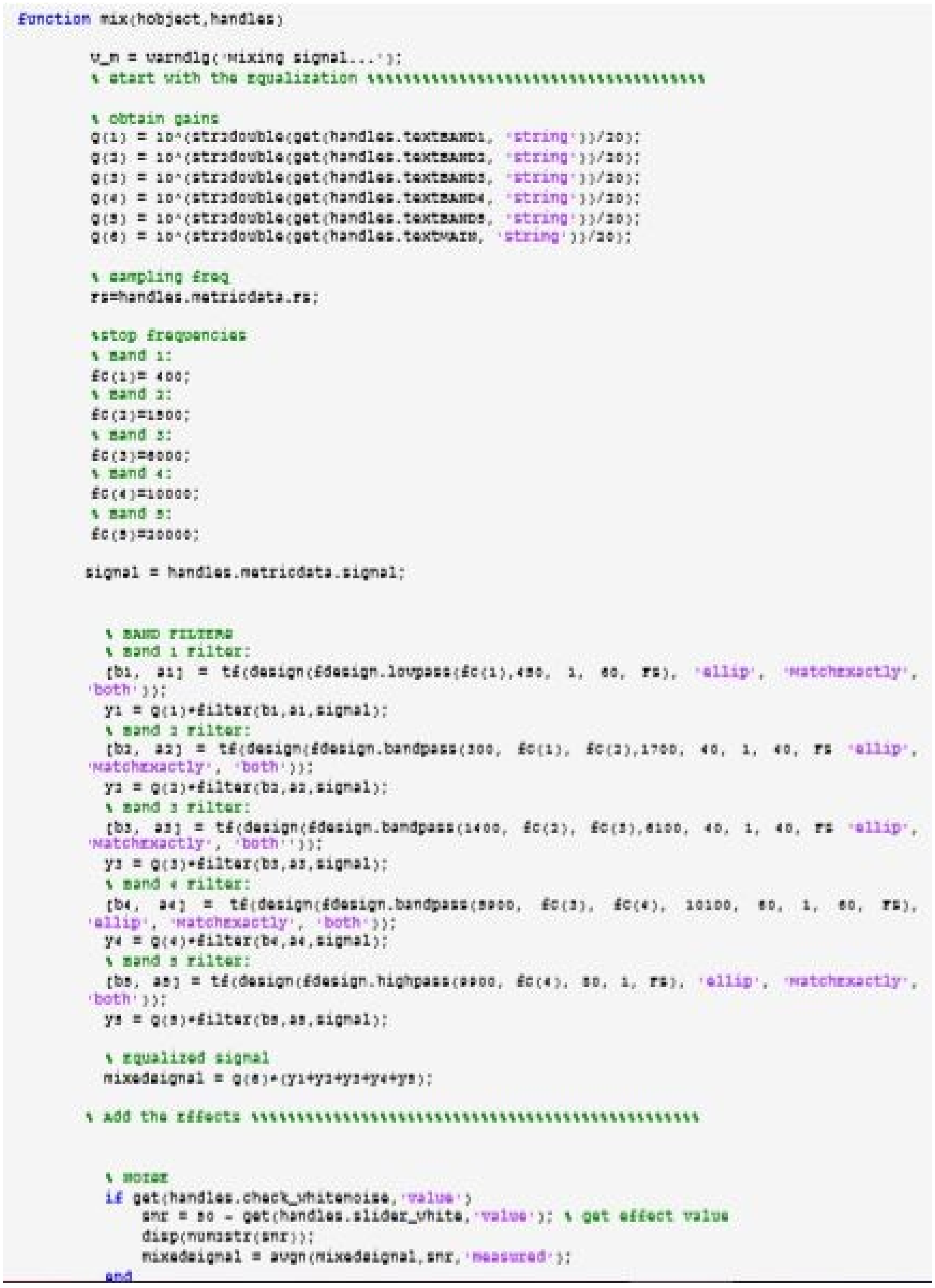
```
function buttonSTOPREC_Callback(hObject, eventdata, handles)
    try
        stop(handles.metricdata.recObj); % stop recording
        reset(hObject,handles);
        recordedSignal  =  (getaudiodata(handles.metricdata.recObj))';  %  obtain  vector  from  recorded
              file
        Fs =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



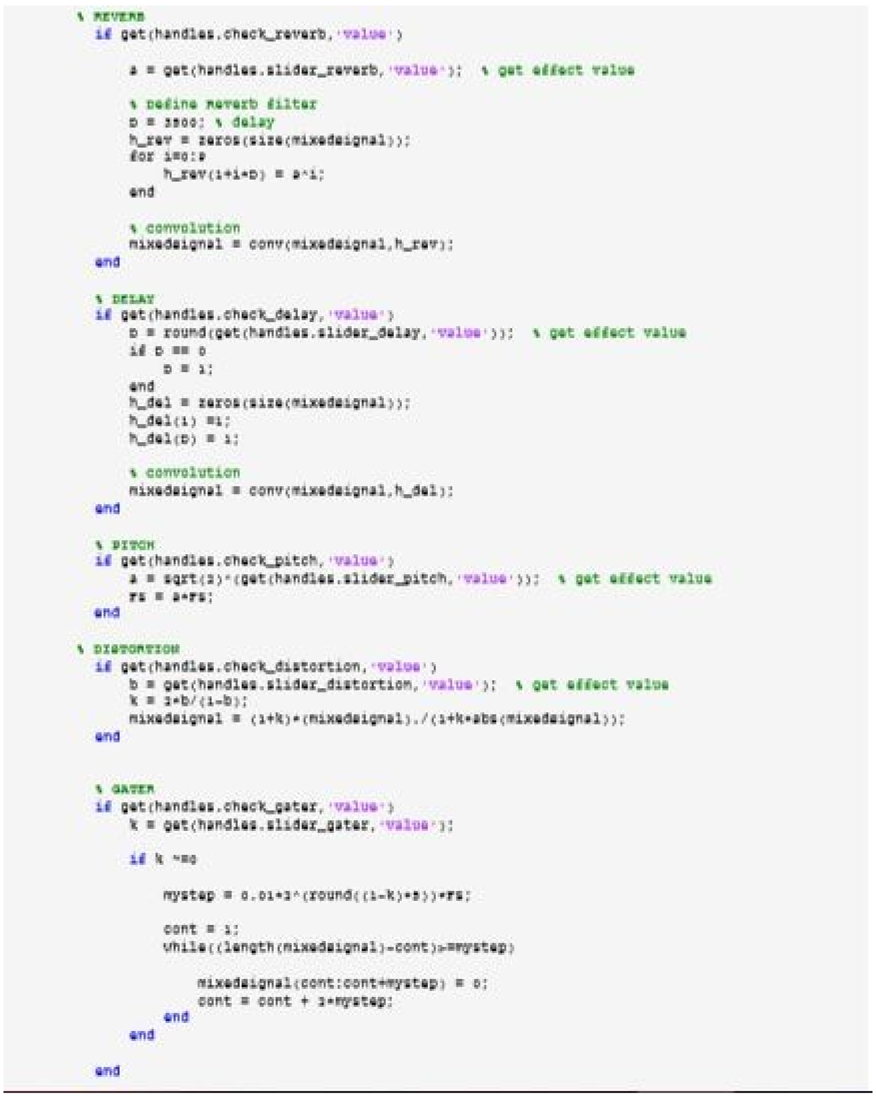
```
function buttonLOAD_Callback(hObject, eventdata, handles)
        try
            % Load file
            filename = '';
            [filename,pathname] = uigetfile({'*.mp3';'*.wav';'*.ogg';'*.flac';'*.au';'*.m4a';'*.mp4'});
            if isequal(filename,'')
                w_l = warndlg('Loading...');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



```
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 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)=1600;
        % 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),  50,  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 = 2500; % delay
      h_rev = zeros(size(mixedSignal));
      for i=0:5
          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)*8))*Fs;

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

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

  end
```

```
    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);
```

```
            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
```

```
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
```

REFERENCES

1.) https://asa.scitation.org

2.) http://citeseerx.ist.psu.edu

3.) http://azadproject.ir/wp-content/uploads/2014/07/Accurate-Discretization-of-Analog-Audio-Filters-with-Application-to-Parametric-Equalizer-Design.pdf

4.)https://www.eecs.qmul.ac.uk/~josh/documents/2011/Reiss-2011-TASLP-ParametricEqualisers.pdf

5.) http://www.aes.org/e-lib/browse.cfm?elib=13893

6.) http://www.thatcorp.com/datashts/AES13-041_Control_of_DSP_Parametric_EQs.pdf

7.) https://patents.google.com/patent/US8363853

8.)http://cv.cs.nthu.edu.tw/upload/courses/14/uploads/CS3570_Chapter5_part2.pdf

9.) ftp://ftp.esat.kuleuven.be/pub/SISTA/ida/reports/18-04.pdf

10.) http://egr.uri.edu/wp-uploads/asee2016/99-446-1-RV.pdf