Coursework 1 Scientific Q1
Report – Joseph Liu 1935938

Program design and implementation:

For this course for this coursework, I was supposed to use Guide GUI (Graphical User Interface), but I have learned that will soon become obsolete and no longer working. Therefore, to make it future proof all the source codes are therefore in MLAPP file.

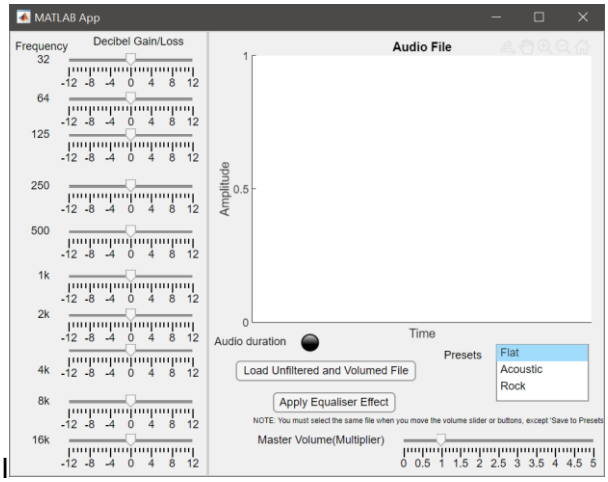


Figure 1: Starting GUI

I created a GUI because not only is for the coursework, but also gives a human readable interface for everyone to use intuitively. Other interfaces are less human readable and will confuse people who do not understand lower-level interface.

The coursework required the following: Graphical GUI to load and display audio file (in these formats: wav, mp3, aac, flac and m4a for assurance), equalising bands of 10 sliders of different frequency, Master volume, set and recall pre-sets and a novelty extension (audio length duration lamp).

Due to the awkward nature of the programme, the user must click either button to see any changes to the audio wave on the graph. Also, clicking either button will load an ‘Open file’ window, regardless of whether another file was already loaded; the user must load the same file as they desired earlier and do not pick any others by mistake.

First, I will discuss the Loading and Displaying the audio band and setting the Master volume. Having a ‘master volume’ allow the audio to louden/quieten at the user’s preferences, useful if the user wants the most/least attention to itself.

Adjust the volume multiplier as desired:

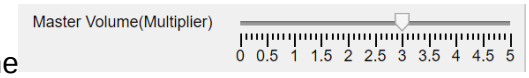


Figure 2: Changed Master Volume

Click on 'Load Unfiltered and Volumed File’. Then choose a desired sound file and see the amplified sound waves by the volume multiplier.

In Figure 3, you can choose files in wav, mp3, m4a, flac or aac. All the used audio files plus some bonus should be located the same directory as the MLAPP file.

Coursework 1 Scientific Q1
Report – Joseph Liu 1935938

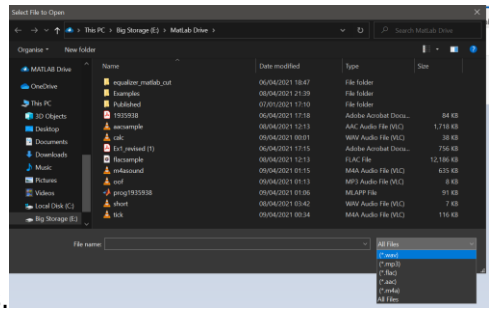


Figure 3: Choose audio file with possibilities.

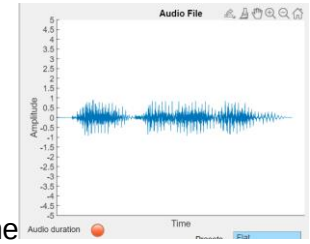


Figure 4: calc.wav file at multiplier 1 volume

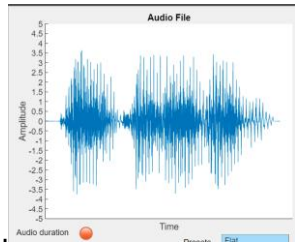


Figure 5: Same file at multiplier 4 volume.

And to ensure that all other file format is supported: See Figure 6,7,8,9 for other file format support.

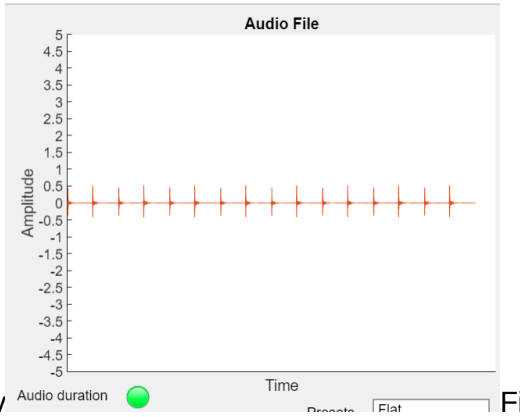


Figure 6: tiktok.wav

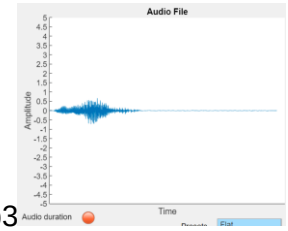


Figure 7: oof.mp3

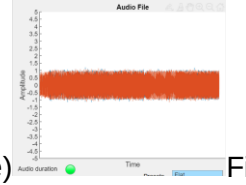


Figure 8: aacsample.aac (stereo file)

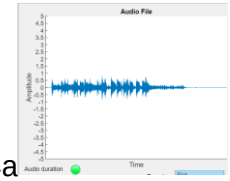


Figure 9: m4asound.m4a

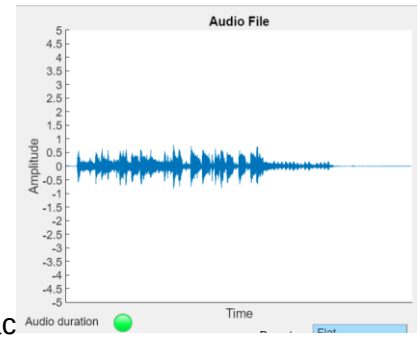


Figure 10: flacsample.flac

And changing the ‘master volume’ multiplier will amplify/quieten the waves.

Allowing multiple file support allows people to test the programme even if they do not have all the files in 5 different formats on the computer.

Next: Set and recall pre-sets, setting up the 10 equaliser sliders and applying the effects to the display file. Apart from applying the effect, hey were simple; there are 10 sliders, each categorised by their frequencies in hertz and values up to 12 decibels gained/lost (where values less than 0 lose decibels and vice versa). By default, all equaliser sliders are set to 0, but the user can customise them.

Also, you can click the pre-sets from either flat (all 0 decibel changes), acoustic or rock. By default, the pre-set is Flat, but just clicking on another pre-set will automatically change the equalising sliders to the matching pre-set. See figure 1 for ‘flat’ pre-set, figure 10 for ‘acoustic’ and figure 11 for ‘rock.’

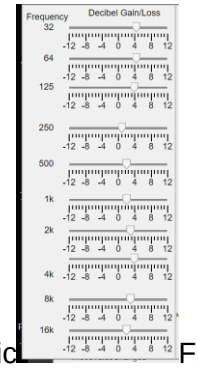


Figure 10: Acoustic

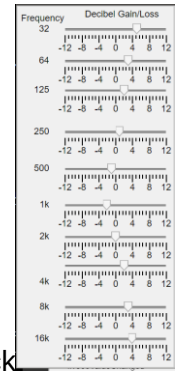
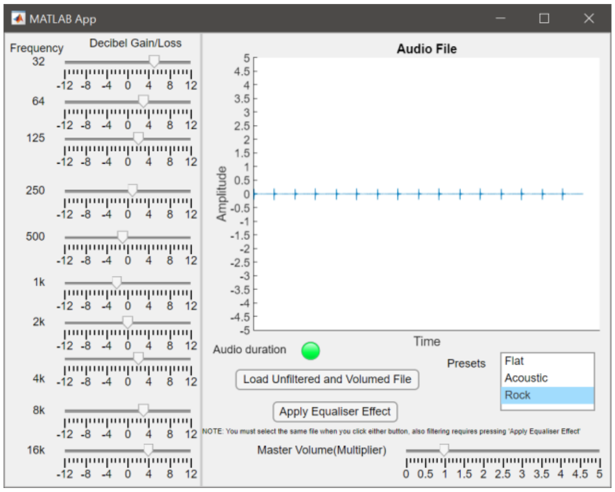


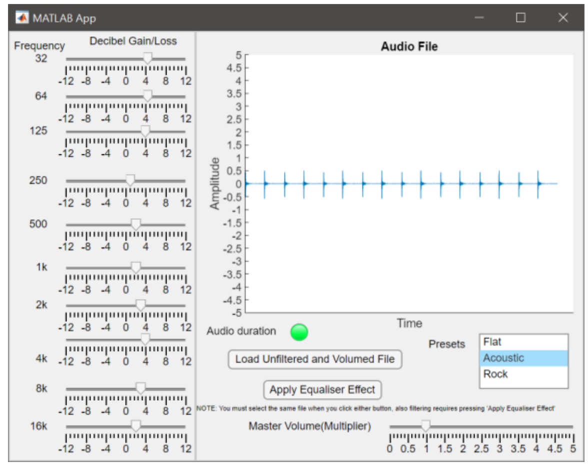
Figure 11: Rock

The equaliser sliders only take effect when the user clicks ‘Apply Equaliser Effect’ and chooses a file.

Figure 12 Below: tiktok.wav with Rock equaliser Figure 13 Below: same with Acoustic equaliser

Coursework 1 Scientific Q1
Report – Joseph Liu 1935938





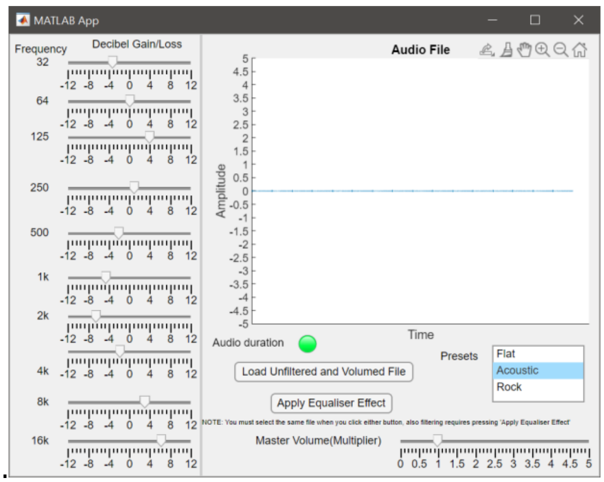
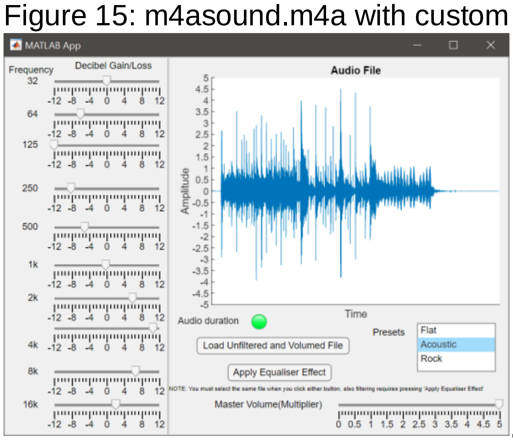


Figure 14: same with custom setting.

The user can also apply the 'master volume' amplifier:

Figure 15: m4asound.m4a with custom setting and volume multiplier 5



.

Using equalisation on audio files are useful, especially in audio production, where using them will make the sound file more prominent without being distorted by changing the 'Master volume.' They can also alter the file so they can remove high and/or low frequencies from the file. Removing high frequency from audio file will mean humans can distinguish between if they

have tinnitus or if the high frequency sound is coming from the audio file emitting from the speakers.

Algorithmic description of main elements:

Figure 16: LoadUnfilteredandVolumedFileButtonPushed

```
function LoadUnfilteredandVolumedFileButtonPushed(app, event)
    cla(app.UIAxes);
    app.PresetsListBox.Value="Flat";
    app.AudiodurationLamp.Color = 'black'; %Black default lamp
    chosenfile=uigetfile({'*.wav';'*.mp3';'*.flac';'*.aac';'*.m4a'});
    ylim(app.UIAxes, [-5 5]);
    %Referenced from Yared Andarge
    [y,Fs]=audioread(chosenfile);
    t=(0:size(y,1)-1/Fs); %Time Vector
    soundduration=length(y)./Fs; %file duration
    volumenow=app.MasterVolumeMultiplierSlider.Value;
    plot(app.UIAxes,t,y*volumenow);
    if soundduration <= 0.5
        app.AudiodurationLamp.Color='r';
    elseif soundduration <= 2
        app.AudiodurationLamp.Color=[1.00,0.41,0.16];
    else
        app.AudiodurationLamp.Color='g';
    end
end
```

This satisfies the following fundamentals of the programme: Novel Extension (Sound Duration Lamp), Load and Display file in the formats stated. Then, loads the graph to of y=-5 to 5, reads the opened file, finds the time vector and duration to give the length in seconds, plot the audio file and changes the lamp colour if the duration is under half a second, 2 seconds or neither.

Figure 17: Fixed Preset slider values when the user clicks on a preset box.

```
function PresetsListBoxValueChanged(app, event)
    if (app.PresetsListBox.Value=="Flat")
        app.three2.Value=0;
        app.six4.Value=0;
        app.one25.Value=0;
        app.two50.Value=0;
        app.five00.Value=0;
        app.onek.Value=0;
        app.twok.Value=0;
        app.fourk.Value=0;
        app.eightk.Value=0;
        app.sixteenk.Value=0;
    elseif (app.PresetsListBox.Value=="Acoustic")
        app.three2.Value=4.5;
        app.six4.Value=4.5;
        app.one25.Value=4;
        app.two50.Value=1;
        app.five00.Value=2;
        app.onek.Value=2;
        app.twok.Value=3;
        app.fourk.Value=4;
        app.eightk.Value=3;
        app.sixteenk.Value=2;
    elseif (app.PresetsListBox.Value=="Rock")
        app.three2.Value=5;
        app.six4.Value=3;
        app.one25.Value=2;
        app.two50.Value=1;
        app.five00.Value=-1;
        app.onek.Value=-2;
        app.twok.Value=0;
        app.fourk.Value=2;
        app.eightk.Value=3;
        app.sixteenk.Value=4;
    end
```

Figure 18: function for equalising the audio file made by me and the people referenced:

```
function ApplyEqualiserEffectButtonPushed(app, event)
    cla(app.UIAxes);
    volumenow=app.MasterVolumeMultiplierSlider.Value;
    app.AudiodurationLamp.Color = 'black'; %Black default lamp
    chosenfile=uigetfile({'*.wav';'*.mp3';'*.flac';'*.aac';'*.m4a'});
    ylim(app.UIAxes, [-5 5]);

    %Referenced from Spyros Lontos, Arvita Agus Kurniasari and M Holters
    [y,Fs]=audioread(chosenfile);
    t=(0:size(y,1)-1/Fs); %Time Vector
    soundduration=length(y)./Fs; %file duration
    %Lowpass filter
    filter32=lowshelving(app,y,(2*32)/Fs,app.three2.Value);
    %Peak Filter
    filter64=peakfilt(app,filter32,(2*64)/Fs,(2*32)/Fs,app.six4.Value);
    filter125=peakfilt(app,filter64,(2*125)/Fs,(2*64)/Fs,app.one25.Value);
    filter250=peakfilt(app,filter125,(2*250)/Fs,(2*125)/Fs,app.two50.Value);
    filter500=peakfilt(app,filter250,(2*500)/Fs,(2*250)/Fs,app.five00.Value);
    filter1k=peakfilt(app,filter500,(2*1000)/Fs,(2*500)/Fs,app.onek.Value);
    filter2k=peakfilt(app,filter1k,(2*2000)/Fs,(2*1000)/Fs,app.twok.Value);
    filter4k=peakfilt(app,filter2k,(2*4000)/Fs,(2*2000)/Fs,app.fourk.Value);
    filter8k=peakfilt(app,filter4k,(2*8000)/Fs,(2*4000)/Fs,app.eightk.Value);
    %Highpass Filter
    filter16k=highshelving(app,filter8k,(2*16000)/Fs,app.sixteenk.Value);
    %Add effect
    audioequaliser=filter16k;

    yEqualised=volumenow*audioequaliser;
    plot(app.UIAxes,t,yEqualised);
    if soundduration <= 0.5
        app.AudiodurationLamp.Color='r';
    elseif soundduration <= 2
        app.AudiodurationLamp.Color=[1.00,0.41,0.16];
    else
        app.AudiodurationLamp.Color='g';
    end
end
```

Figure 19: M. Holters' code for low and high frequency shelving and peak filter:

```
        function ls = lowshelving(~,x, Wc, G)
% Author: M. Holters
% Applies a low-frequency shelving filter to the input signal x.

%-----------------------------------------------------------------------
% All source code provided until 'End of reference' is provided without any warranties as published in
% DAFX book 2nd edition, copyright Wiley & Sons 2011, available at
% http://www.dafx.de. It may be used for educational purposes and not
% for commercial applications without further permission.
%-----------------------------------------------------------------------

        V0 = 10^(G/20); H0 = V0 - 1;
        if G >= 0
          c = (tan(pi*Wc/2)-1) / (tan(pi*Wc/2)+1);     % boost
        else
          c = (tan(pi*Wc/2)-V0) / (tan(pi*Wc/2)+V0);   % cut
        end
        xh = 0;
        for n=1:length(x)
          xh_new = x(n) - c*xh;
          ap_y = c * xh_new + xh;
          xh = xh_new;
          ls(n) = 0.5 * H0 * (x(n) + ap_y) + x(n);  % change to minus for HS
        end
        end

        function pf = peakfilt(~,x,Wc,Wb,G)
        % Author: M. Holters
        % Applies a peak filter to the input signal x.
            V0 = 10^(G/20); H0 = V0 - 1;
            if G >= 0
                c = (tan(pi*Wb/2)-1) / (tan(pi*Wb/2)+1);     % boost
            else
                c = (tan(pi*Wb/2)-V0) / (tan(pi*Wb/2)+V0);   % cut
            end
            d = -cos(pi*Wc);
            xh = [0, 0];
            for n=1:length(x)
                xh_new = x(n) - d*(1-c)*xh(1) + c*xh(2);
                ap_y = -c * xh_new + d*(1-c)*xh(1) + xh(2);
                xh = [xh_new, xh(1)];
                pf(n) = 0.5 * H0 * (x(n) - ap_y) + x(n);
            end
        end
        function hs = highshelving(~,x, Wc, G)
            % Author: M. Holters
            % Applies a low-frequency shelving filter to the input signal x.
            V0 = 10^(G/20); H0 = V0 - 1;
            if G >= 0
              c = (tan(pi*Wc/2)-1) / (tan(pi*Wc/2)+1);     % boost
            else
              c = (tan(pi*Wc/2)-V0) / (tan(pi*Wc/2)+V0);   % cut
            end
            xh = 0;
            for n=1:length(x)
              xh_new = x(n) - c*xh;
              ap_y = c * xh_new + xh;
              xh = xh_new;
            hs(n) = 0.5 * H0 * (x(n) + ap_y) - x(n);  % change to plus for LS
            end
        end
% End of reference
    end
```

First, the programme collects the file opened, then run Holters' lowshelving once, then peakfilt 8 times and highshelving once. Each frequency will be minimised or enhanced depnding on the slider values to each respective frequency. Each filter value is hopped over to the next function until filter16k, which by then the audio file has been enhanced in all the frequencies then amplified by the volume, and graphed out. The Lamp colour changes colour to the length of the file, which is surprisingly, never altered.

Sound Duration Lamp (Novel extension):

In the GUI, there is a lamp which changes colour, depending on the duration of the audio file imported.

The lamp changes colour to either these: Green, Orange or Red, depending on the duration.

At the start, when no file has been loaded, the lamp will be black, to indicate nothing was loaded in the audio file axes graph. The lamp also turns black when the user is on the 'Select File' window'

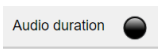


Figure 20: Black Audio Duration Lamp.

For all audio files less than half a second, the lamp turns red. Between half and two seconds, the lamp turns orange. Any longer, the lamp turns green. The novel inclusion of the duration lamps tells the user to import a long duration file so that the equalisation effect is more noticeable than extremely short audio files.

Clicking on ‘Load Volumed file and audio duration’ will open figure 2:

Clicking on calc.wav will load the audio file and give a signal result onto the audio file axes, between –2 and 2. The audio duration lamp will turn orange, because the audio length is between 500 and 2000 milliseconds.

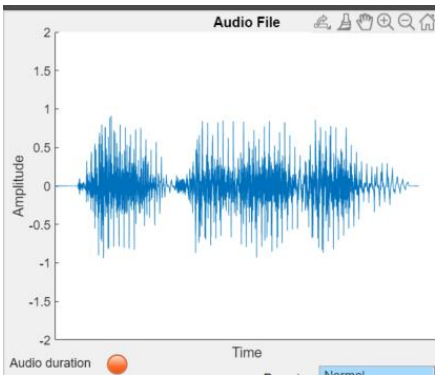


Figure 21: Orange Lamp from decoding calc.wav

If I instead launch a file longer than 2 seconds, i.e. James Arthur – Driver's License.mp3 (for the purpose of lamp demonstration), the lamp turns green. And clap.mp3 of length less than half a second, turns the lamp red.

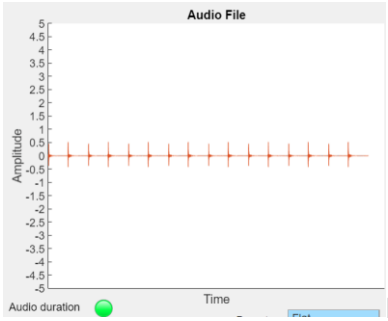


Figure 22: tiktok.wav and green lamp

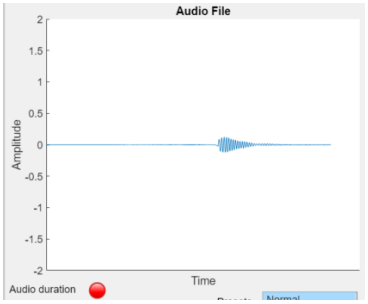


Figure 23: Short.wav file for red lamp

By having a good length audio file, the user will hear an enormous difference when equalising the sound file. If the sound file is too short, the effects could be unnoticeable.

The changing colour lamp is one of the easiest novel extensions I can made in MATLAB, whilst its usefulness is only to make people aware of the length of the file, and if it is too short to be accepted. Because the graph has no x-axis label to show how long the audio file, the lamp is a very good indicator to show the duration. The only thing I needed was if statements to change the colour. All the lamp code is entirely mine and does not use any third-party code.