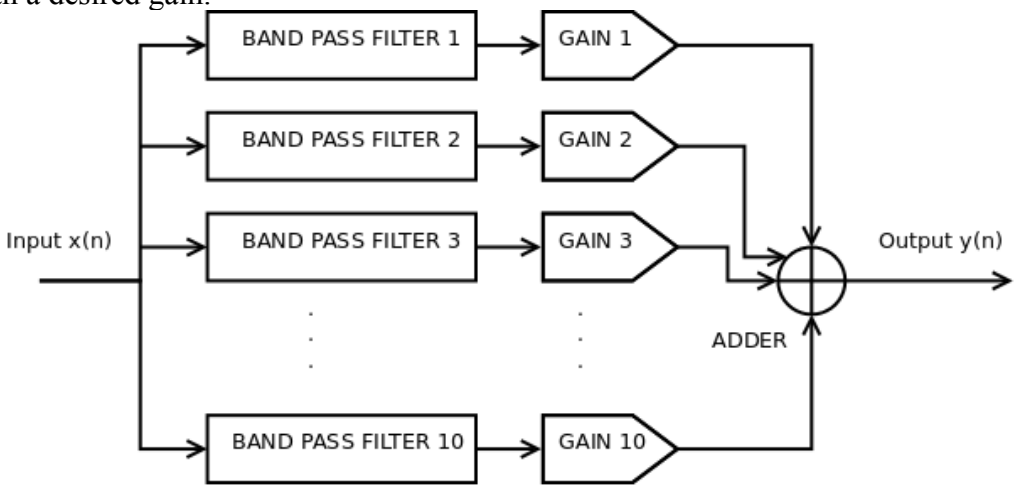
Program Assignment – IIR filter implementation

Most music players come with a graphic equalizer to adjust sound intensity at various frequencies. It can be used to enhance the listening experience or mimic different listening environments.

A music player with an equalizer.

Typical music files have frequency content below 20 KHz. Since the human ear is less sensitive to high frequencies, the equalizer bands are distributed non-uniformly giving finer frequency resolution at the lower end while combining large swathes of the spectrum together at higher frequencies. Equalization is performed by using filter banks to separate audio at different frequencies and then combining them together with a desired gain.



Equalization with filter banks.

The filter center frequencies are typically chosen to start at 31.25 Hz doubling at each band and ending at 16 KHz. The upper and lower cutoffs of each band can be obtained by

$$fc_{low} = f_{center} / \sqrt{2}$$

$$fc_{high} = f_{center} \cdot \sqrt{2}$$

An IIR filter is able to provide a sharp (closer to ideal) magnitude response with a much smaller filter order compared to FIR filters. This leads to computational simplicity which makes them attractive for real-time applications. They don't provide linear phase like FIR filters, and are more susceptible to round-off errors and becoming unstable. IIR filters are implemented as difference equations, unlike FIR filters which are applied using convolution.

You need to write 2 functions.
1) A function to calculate the DTFT amplitude response:

```
function [H w] = DTFT_iir_amp(b, a, R)
```

The arguments b and a are the coefficients of the transfer function H(z) written as:

$$H(z)=\frac{b_0 z^0+b_1 z^{-1}+...+b_M z^{-M}}{a_0 z^0+a_1 z^{-1}+...+a_N z^{-N}}$$

where the number of *a* coefficients is N+1 and the number of *b* coefficients is M+1. The filter order is N. Usually, M ≤ N. The DTFT can then be written as:

$$H(e^{j\omega})=\frac{b_0 e^{-j\omega 0}+b_1 e^{-j\omega 1}+...+b_M e^{-j\omega M}}{a_0 e^{-j\omega 0}+a_1 e^{-j\omega 1}+...+a_N e^{-j\omega N}}$$

From here onwards, we follow the same approach that was used to plot the DTFT of a sequence in program 2. We divide the interval between 0 and pi into R discrete points. So omega is represented by w(k) for k between 0 to R-1 where

$$w(k)=\frac{\pi k}{(R-1)}$$

The DTFT now becomes:

$$\left|H(e^{jw(k)})\right|=\left|H(k)\right|=\left|\frac{b_0 e^{-j\omega(k)0}+b_1 e^{-j\omega(k)1}+...+b_M e^{-j\omega(k)M}}{a_0 e^{-j\omega(k)0}+a_1 e^{-j\omega(k)1}+...+a_N e^{-j\omega(k)N}}\right|=\left|\frac{\sum_{m=0}^{M} b_m e^{-jw(k)m}}{\sum_{n=0}^{N} a_n e^{-jw(k)n}}\right|$$

This is something we can do on a computer. The function DTFT_iir_amp will be implemented as:

- Find M and N from lengths of *b* and *a* as M = length(b)-1, N = length(a)-1.
- Pre-allocate memory for the DTFT
- Start an outer loop over k going from 0 to R-1
  - Calculate w(k+1) = pi*k/(R-1)
  - Set NUM = 0
  - Start a loop over m going from 0 to M
    - NUM = NUM + b(m+1)*exp(-j * w(k+1) * m)
  - End the loop over m
  - Set DEN = 0
  - Start a loop over n going from 0 to N
    - DEN = DEN + ??? (complete this the same way as in the previous loop, but using 'a' and 'n' instead)
  - End the loop over n
  - Set H(k+1) = abs(NUM/DEN), since we are interested in the *magnitude response*.
- End the loop over k

The blue +1s are to make the array indices start at 1 to work with MATLAB.

2) A function to apply the difference equation to a sequence.

```
function y = iir_filter(b, a, x)
```

The arguments are the coefficients *a* and *b* of the difference equation, and the input sequence x. The

difference equation for the 4th order filter that we obtain from the H(z) above is:

$$y(n)=\frac{1}{a_0}[b_0x(n)+b_1x(n-1)+b_2x(n-2)+b_3x(n-3)+b_4x(n-4)$$
$$-a_1y(n-1)-a_2y(n-2)-a_3y(n-3)-a_4y(n-4)]$$

Here is a sample implementation that is incomplete and has bugs in it. Here we have assumed the filter order is N=M=4 and both *a* and *b* have exactly 5 elements each. You will need to fix this code and get it to work for the N=4 case.

```
function y = iir_filter(b, a, x)
% Implements a 4th order difference equation.

Nx = length(x);
y = zeros(Nx,1);

for n=0:Nx-1
    y(n +1) = (b(0 +1)*x(n +1) + b(1 +1)*x(n-1 +1) ...
        + b(2 +1)*x(n-2 +1) + b(3 +1)*x(n-3 +1) + b(4 +1)*x(n-4 +1) ...
        - a(1 +1)*y(n-1 +1) - ??? - ??? - ??? ...
        ) / a(0 +1);
end
```

Note the blue +1s are to shift the indices by 1 for MATLAB. They have been deliberately written that way for clarity. To get this code to run, you will need to fix the loop and pre-compute a few values outside of the loop, and complete the missing portions indicated by ??? marks.

Optional: You can also write a version of iir_filter that can work for any filter order (i.e. where N and M could be anything). Such an implementation will have a loop over n, containing 2 nested loops to traverse the *a* and *b* coefficients. Since the order is not known in advance, it is not possible to pre-compute any values of y, but you will have to use if statements within your loops to take care of 0 and negative indices being generated. You are trying to implement the following:

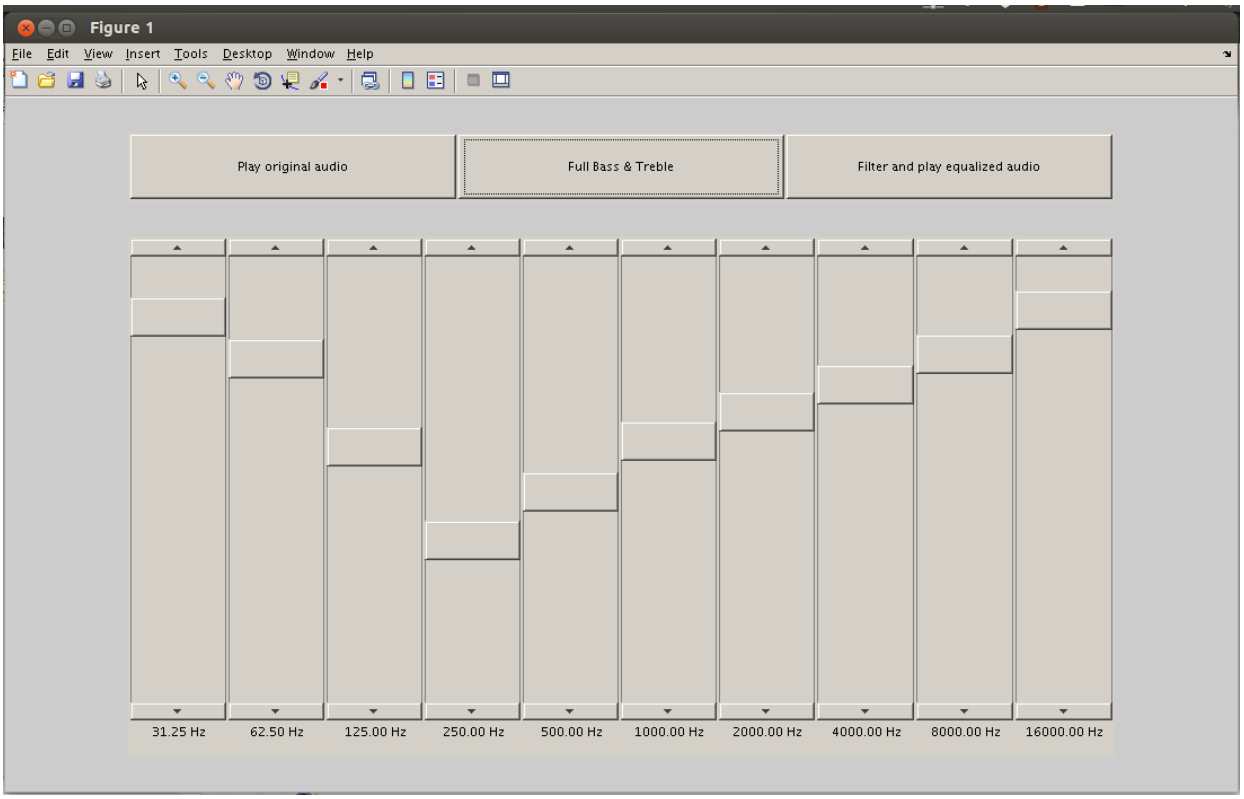
$$M=length(b)-1, N=length(a)-1$$
$$y(n)=\sum_{r=0}^{M} b_r x(n-r)-\sum_{k=0}^{N} b_k y(n-k)$$

Your MATLAB loops would look like:

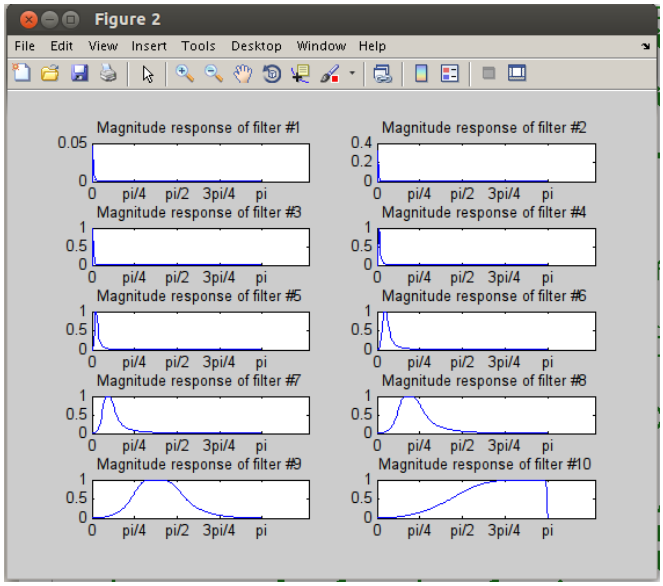
```
…
y = zeros(Nx,1);
…
for n=?
    for r=?
        if(?)
            ?
        end
    end
    for k=?
        if(?)
            ?
        end
    end
end
```

Running the program:

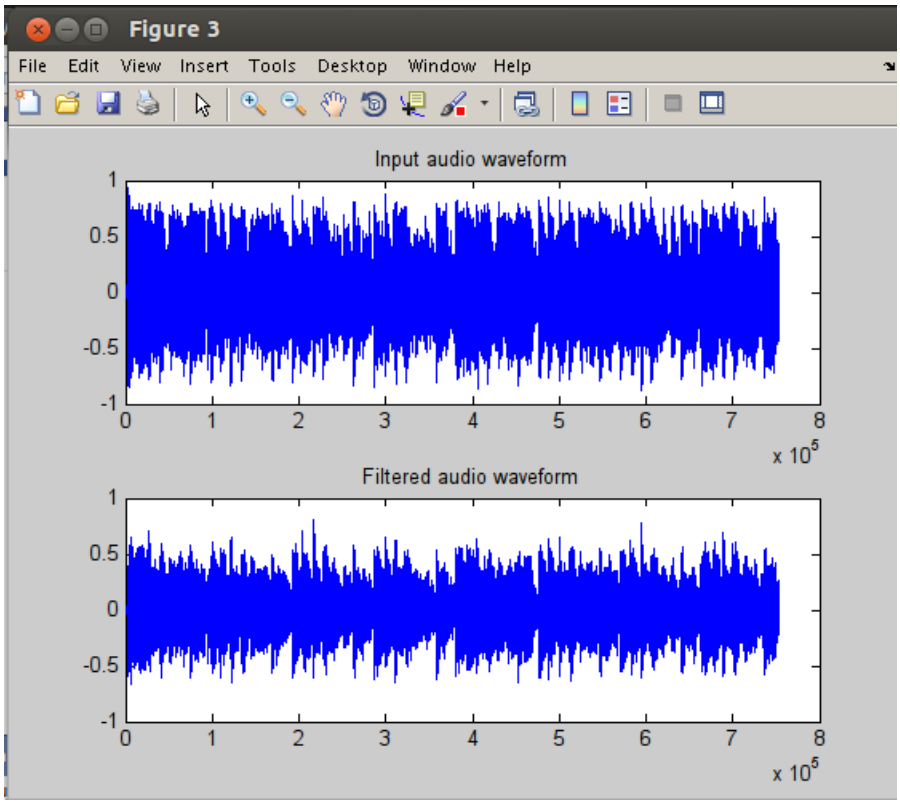
Run the file equalizer_gui.m to launch the GUI. The interface is pretty self explanatory. You can either choose one of the presets for the equalizer, or manually set the sliders. Pressing the Filter and Play button will start the processing and play back the filtered audio. Changes to the equalizer settings are ignored once the file has started playing.



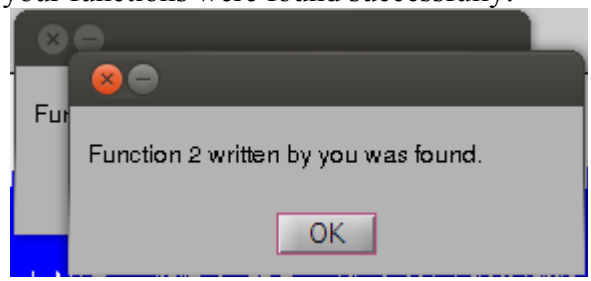
The magnitude response of the ten band pass filters is shown in another window. If you write the DTFT_iir_amp function, then it is used to produce these plots instead of MATLAB's built in functions.



The filtered waveform is shown next to the original waveform for a comparison. You may be able to zoom in and identify the low frequency portions of the audio clip. High frequencies look like noise and are harder to identify visually. It is a good idea to ensure that the amplitude of the output waveform stays within +/-1. Anything beyond that will be clipped off during playback or when storing to a file. Clipping can be avoided by using a 'pre-amp' gain to reduce the input amplitude.



You should see a message if your functions were found successfully:



Optional Work:

Run Program 2 again with a filter order Nh = 129 (around line 80). Observe the sharpness of the slope of |H(e^jw)| for the FIR filters, and compare that with the slope of |H(e^jw)| of the IIR filter of order 4 that are produced by this program.

Estimate the number of multiplies required to compute y(n) with an FIR filter with Nh = 129 for one value of n, and the number of multiplies required to compute one value of y(n) with an IIR filter of order 4.

Rohit Rawat
08/12/2013