Program Assignment – FIR filter design

In a 3-way speaker system the audio is separated into low-frequency, mid-frequency, and high-frequency components, which are used to drive a woofer, a mid-range speaker, and tweeters, respectively. Most systems use a single wide band amplifier at the output and separate the frequency components using analog crossover filters built from discrete components.

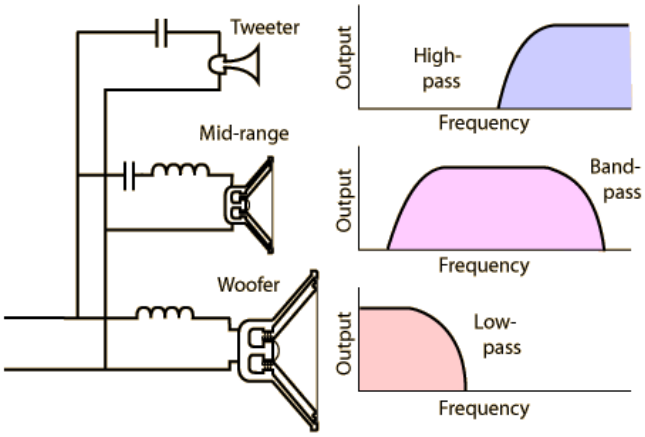


Illustration 1: Analog cross over filter.
(http://hyperphysics.phy-astr.gsu.edu/hbase/audio/cross.html#c4)

Such systems are inexpensive, but can have certain drawbacks:

1. The filters may not cut out at the precise frequency at which the adjacent filter begins. This leads to both the speakers trying to drive the overlapping frequencies leading to uneven output.
2. It is hard to have precise control over the phase of analog filters. This may lead to phase distortion and out of phase results from different speakers.

We present a high fidelity music system where the frequency components will be separated by digital filtering inside a DSP processor.

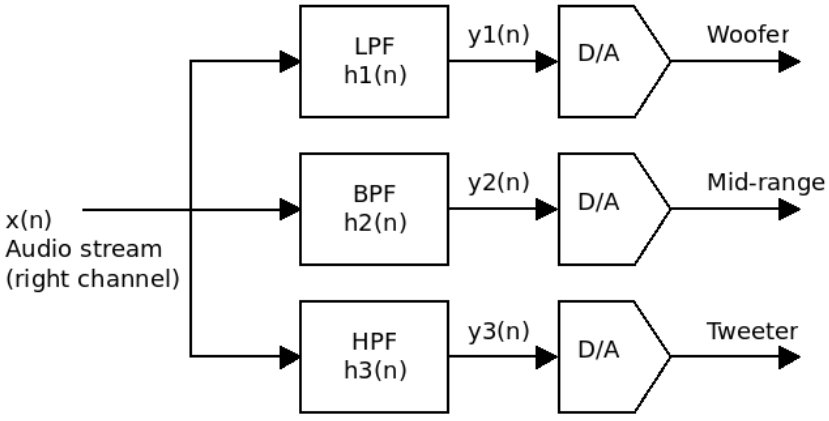


Illustration 2: Block diagram of the DSP output filtering stage.

Instead of using analog crossover filters, this high end product uses programmable digital filters and dedicated A/D converters and amplifiers to drive the sub-woofer, mid-range speaker, and the tweeter. Your goal is to design FIR filters which will separate audio for these three speakers. We will have precise control over the filter cutoffs without having to mess with the hardware. FIR filters will also give us precise control over the phase and eliminate phase distortion.

Here are some design specifications for the DSP system:

1. Sampling rate of 40000 Hz.
2. Linear phase for low distortion.
3. LPF frequency filter cutoffs: DC to 800Hz
4. BPF frequency filter cutoffs: 500Hz to 8000Hz
5. HPF frequency filter cutoffs: 5000Hz to Nyquist frequency

You need to write 4 functions. Three of them are almost identical to each other.
1) A function to design an FIR low pass filter:

```
function h = fir_lpf(Nh, w_c)
```

This function implements the FIR LPF equation:

$$h(n+1)=\frac{\sin(\omega_c(n-N_d))}{\pi(n-N_d)} \quad if\ n\neq Nd \qquad 1(a)$$

$$h(n+1)=\frac{\omega_c}{\pi} \qquad if\ n=Nd \qquad 1(b)$$

Note: The blue +1 is for MATLAB's indexing, to make our arrays start at 1.

Here is what goes on inside this function:

- Preallocate memory for h = zeros(1,Nh)
- Calculate Nd = (Nh-1)/2
- Start a loop for n going from 0 to Nh-1
  - if n is not equal (~=) to Nd, calculate h(n+1) using equation (1a)
  - else, calculate h(n+1) using (b).
- End the loop over n

2) A function to design an FIR band pass filter:

```
function h = fir_bpf(Nh, w_c1, w_c2)
```

This function implements the FIR BPF equation:

$$h(n+1)=\frac{\sin(\omega_{c2}(n-N_d))-\sin(\omega_{c1}(n-N_d))}{\pi(n-N_d)} \quad if\ n\neq Nd \qquad 2(a)$$

$$h(n+1)=\frac{\omega_{c2}-\omega_{c1}}{\pi} \qquad if\ n=Nd \qquad 2(b)$$

Here is what goes on inside this function:

- Preallocate memory for h = zeros(1,Nh)
- Calculate Nd = (Nh-1)/2
- Start a loop for n going from 0 to Nh-1
  - if n is not equal to Nd, calculate h(n+1) using equation (2a)
  - else, calculate h(n+1) using (2b).
- End the loop over n

3) A function to design an FIR high pass filter:

```
function h = fir_hpf(Nh, omega_c)
```

This function implements the FIR LPF equation:

$$h(n+1)=\frac{-\sin(\omega_c(n-N_d))}{\pi(n-N_d)} \quad if\ n\neq Nd \qquad 3(a)$$

$$h(n+1)=1-\frac{\omega_c}{\pi} \qquad if\ n=Nd \qquad 3(b)$$

Here is what goes on inside this function:

- Preallocate memory for h = zeros(Nh,1)
- Calculate Nd = (Nh-1)/2
- Start a loop for n going from 0 to Nh-1
  - if n is not equal to Nd, calculate h(n) using equation (3a)
  - else, calculate h(n) using (3b).
- End the loop over n

4) A function to calculate the DTFT amplitude response:

```
function [X w] = DTFT_amp(x, R)
```

Recall that the DTFT equation is:

$$H(e^{j\omega})=\sum_{-\infty}^{\infty} h(n)e^{-j\omega n} \qquad -\pi\leq\omega\leq\pi$$

Let's modify it to use it with our FIR filters and sequences. We know our filters start at time 0 and have Nh elements, thus n varies only from 0 to Nh-1.

$$H(e^{j\omega})=\sum_{n=0}^{Nh-1} h(n)e^{-j\omega n} \qquad -\pi\leq\omega\leq\pi$$

The next problem we face is, that omega is a continuous variable between -pi and pi. We will choose to neglect the negative frequencies and focus on the frequencies from 0 to pi. How can we realize the continuous variable omega? 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:

$$H(e^{jw(k)})=H(k)=\sum_{n=0}^{Nh-1} h(n)e^{-jw(k)n} \qquad 0\leq k\leq R-1$$

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

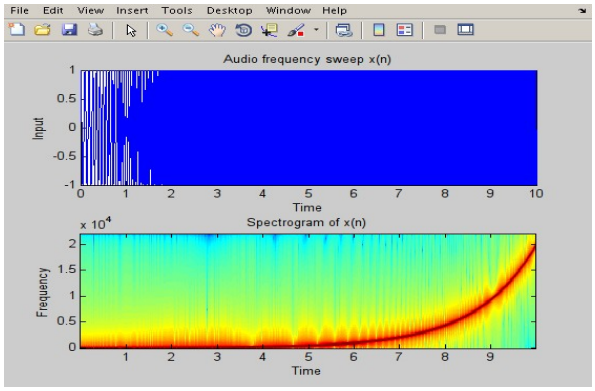
- Find the length of the sequence Nh = length(h)
- Allocate memory for the DTFT as H = zeros(R,1)
- Start an outer loop over k going from 0 to R-1
  - Calculate w(k+1) = pi*k/(R-1)
  - Set sum = 0
  - Start a loop over n going from 0 to Nh-1
    - Increment sum as: sum = sum + h(n+1)*e^(-j * w(k+1) * n)
  - End the loop over n
  - Set H(k+1) = abs(sum), 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.

Testing the filters

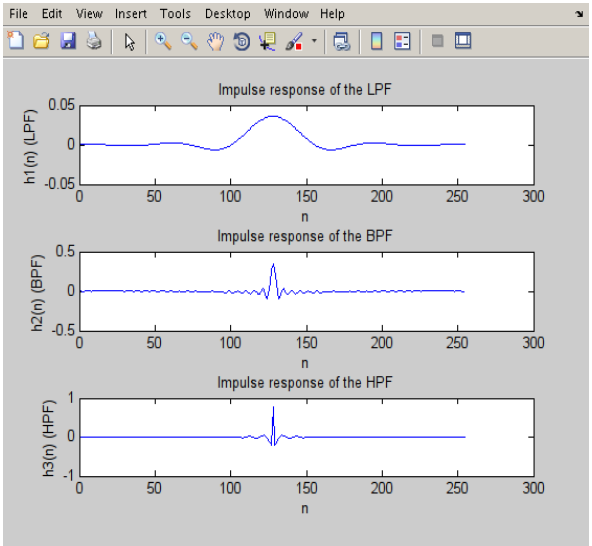
You are provided a file “main.m” that implements the steps below. This script is capable of running on its own using MATLAB's built in functions. You are required to provide the four functions described previously which will be used instead of MATLAB's built-ins.

1. **Input sequence**: We use a frequency sweep signal which will have frequencies from 10Hz to 20KHz. The spectrogram of the signal shows the frequency content of the signal at different

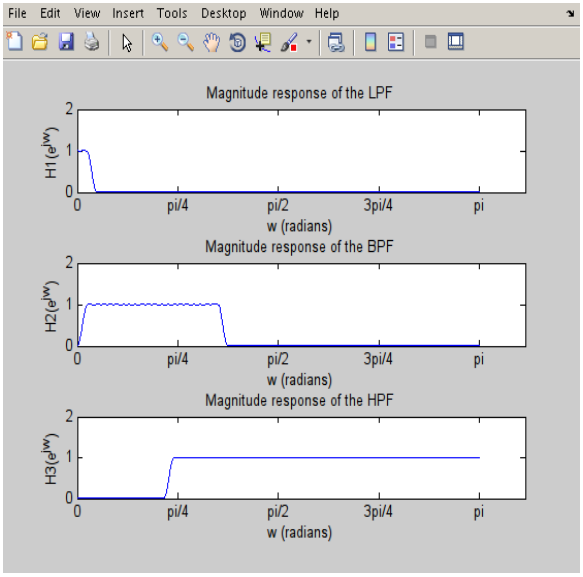


   times. The frequencies in red are of the highest magnitude. We have deliberately chosen a logarithmic sweep as we cannot distinguish clearly between high frequencies, i.e. one can easily tell the difference between 100Hz and 200Hz waves, but barely tell apart a 15KHz wave from a 16KHz wave.

2. **Filter design**: The given code uses MATLAB's built in filter design capabilities to design the three filters h1, h2, and h3. Once you finish writing the fir_LPF, fir_BPF, and fir_HPF functions and save them in the same folder as your main program, it will automatically pick up the new functions and start using them instead. It will let you know if it was able to find your functions or not at the end. The program plots the impulse response of the three filters in the second figure window. The impulse response generated by your fir_LPF, fir_BPF, and fir_HPF functions should be very similar to what is obtained by using MATLAB's functions.

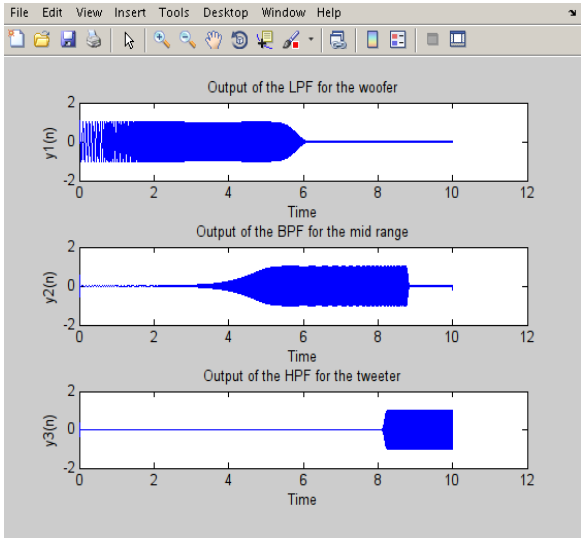


3. **Visualizing the DTFT $H(e^{jw})$ of the filters**: The given code uses MATLAB's fft() function to visualize the magnitude response of the three filters. Once you implement the DTFT_amp function and place the source code file in the same folder as the main program, it should automatically start using it for the magnitude response plots. The magnitude response calculated by your function should look similar to what is obtained using MATLAB's fft.



4. **Applying the filter to the signal**: The main program convolves the three filters with the input sequence x(n). It uses MATLAB's built in convolution function to do this.

5. **The filtered output**: The filters will only let through the frequencies they were designed for. The outputs from the three filters will look like:



6. **Playback**: There is a piece of commented code at the end which plays back the three filtered audio segments. You may uncomment it to hear what the output is like, but it may not be pleasant to your ears.

7. **Ensuring your functions were found**: Make sure you see the following message after the program finishes running:

```
Function 1 written by you was found.
Function 2 written by you was found.
Function 3 written by you was found.
Function 4 written by you was found.
```

If you get a message that a particular function was not found, it means you have not implemented it yet, or that you saved your function in a folder other than the one containing main.m file you are running. Correct the situation and try again.

Optional: Testing on an audio file

MATLAB is capable of reading uncompressed PCM audio files with .wav extension. Make the following changes in Part 1 of the code:

Delete the following code:	Replace with the following:

`% Sampling frequency of the audio.`
`Fs = 40000;`
 	
`% Start and end frequencies of the frequency sweep signal.`
`f_start = 10;`
`f_end = 20000;`
 	
`% 'chirp' is a MATLAB function that produces a signal with a frequency`
`% sweep between a start and end frequency.`
`t = 0:1/Fs:10;  % 10 seconds @ Fs sample rate`
`x = chirp(t, f_start, 10, f_end, 'logarithmic');`
	`% Read a WAV file. The file should be in the same folder as`
	`% this script. The wavread function also returns the`
	`% sampling rate of the file.`
	`[x Fs] = wavread('Friday.wav');`
	`% Convert to single channel by averaging the left and`
	`% right channels.`
	`x = mean(x,2);`
	`% Generate a time axis for the plots`
	`t = 0:1/Fs:(length(x)-1)/Fs;`

Uncomment the playback portion at the end of the file to hear the results. MATLAB cannot read MP3 files. You can convert MP3 files to WAV using software like Audacity, but keep the audio clip size small or MATLAB would freeze.

Rohit Rawat
08/11/2013