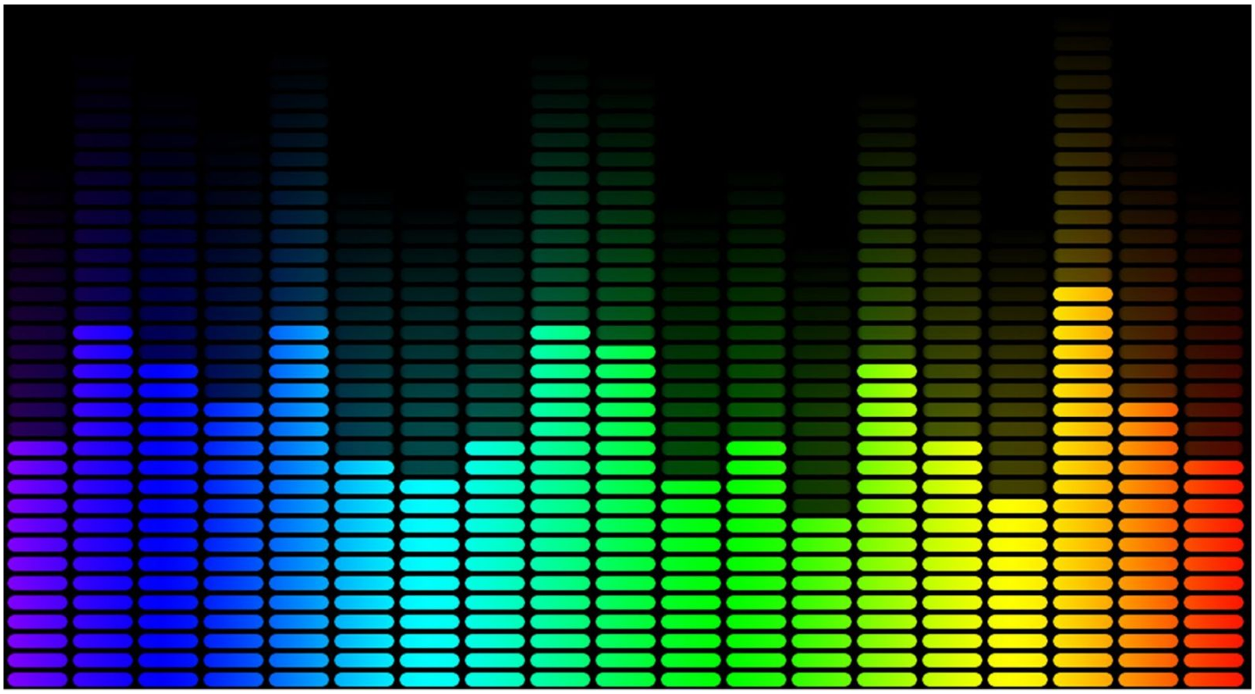
EE 459-01

Final Project- Fun Filtering Times (FFTs)



Matt Rochford & Addison Narter

Lab Bench #8

3/23/2018

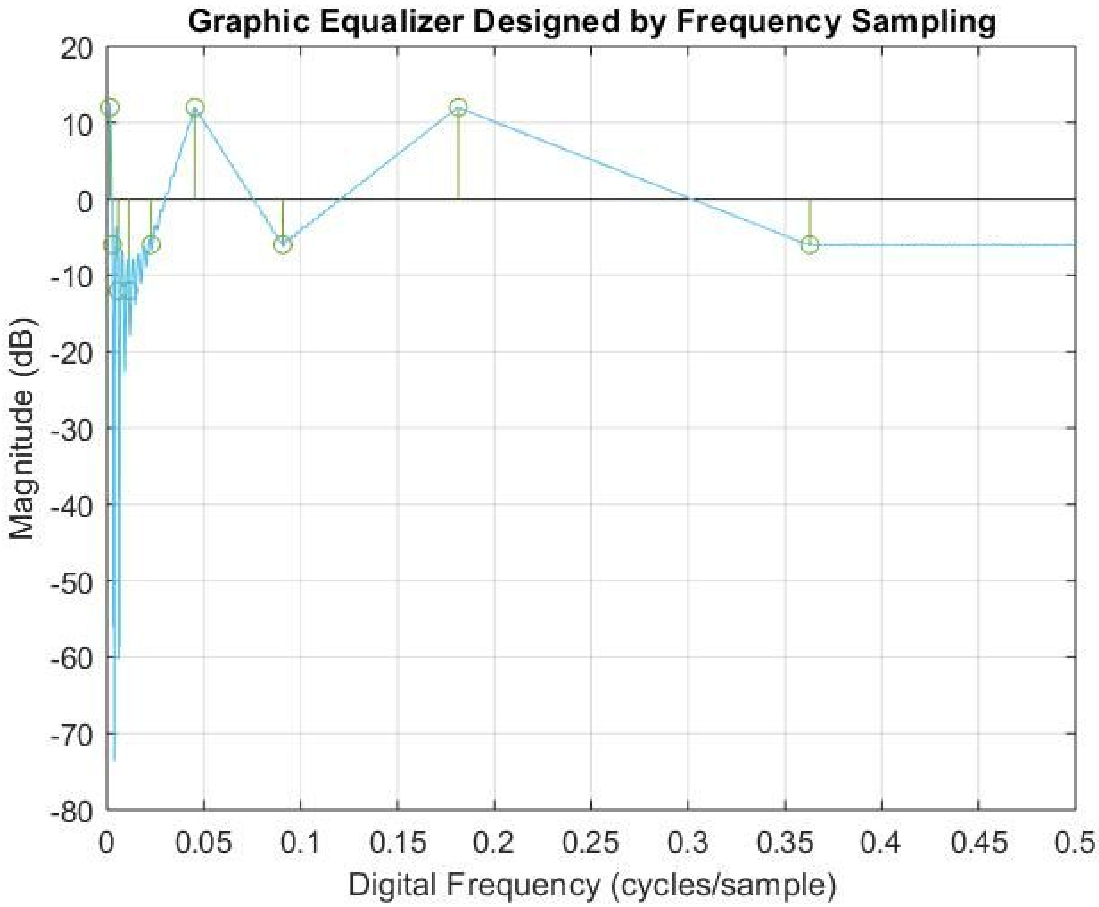


Figure 1: Frequency Response Plot for Test Settings

Equalizer Parameters

- Equalizer Settings (dB): [+12, -6, -12, -12, -6, +12, -6, +12, -6]
- Echo Delays (msec): [250 400 520 660 750 1220]
- Fractional Gains: [0.7 0.6 0.5 0.33 0.2 0.8]

Justification/Discussion

- Test setting were set in the lab for this trial. The echo settings clearly cause overlap and gain settings most likely need adjustments too in order to improve the sound at lower frequencies. The drums are far too loud and distorted.

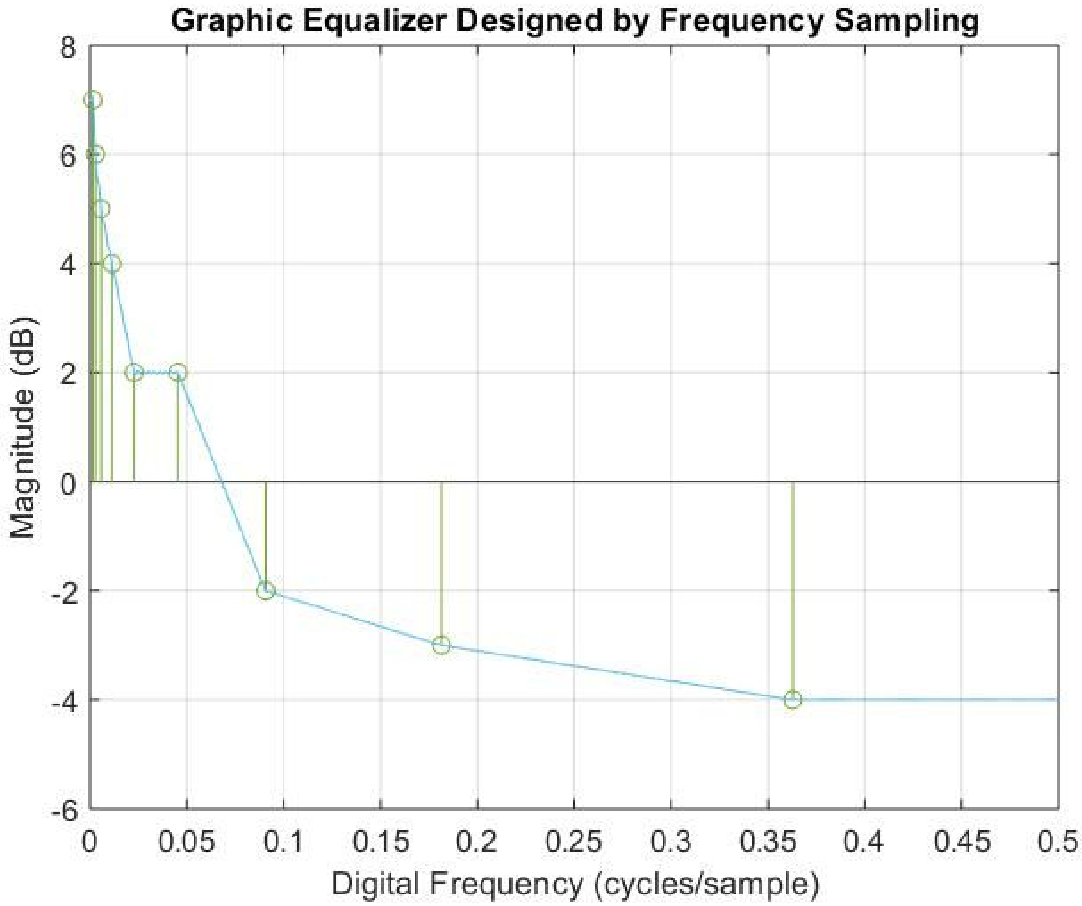


Figure 2: Frequency Response Plot for Improved Settings

Equalizer Parameters

- Equalizer Settings (dB): [7 6 5 4 2 2 -2 -3 -4]
- Echo Delays (msec): [100]
- Fractional Gains: [0.7 0.6 0.5 0.33 0.2 0.8]

Justification/Discussion

- To improve the original test file the first change that had to be made was lowering the number of echo delays. Changing the echo to one 100 millisecond delay. This allowed for a bit of echo but didn't cause overlap like the test file. Then we edditted the gain to account for distortion in low frequency noise from the drumline. Adjusting these two settings a few times ended up being enough to improve the Zarathustra file to its musical glory.

Most Interesting Lab

- This lab was one of our favorites. I wish it wasn't finals week so we could've spent more time on it. It has a ton of real life applications as well and we may even play around with some of our own audio files. Aside from this lab, the IIR implementation lab was a very interesting. One of our group members is a BMED student and the applications of butterworth filters in BMED signalling was very interesting to him.

Least Interesting Lab

- Hunt for red october and the first lab were the least helpful. The first lab was worth doing but we think it might be a better homework assignment or supplemental information. Hunter for red october was mildly useful, but a lot of the lab seemed like busy work.

Design Implementation Documentation:

Script Files

```
%% FFTCONV
function [ yn ] = fftconv( xn, hn )
%   Fast linear time-domain convolution of two finite sequences,
%   xn and hn, to be fast the sequences are zero padded to a power of two
%   at or above the length of the sequence.  Convolution performed by
%   transforming the sequences into the frequency domain and
%   multiplying the two.  yn is then obtained by inverse
%   transforming back into time domain.  Plots of each sequence in both
%   domains is put in figure 1.

lenx = length(xn); %find length of x-time domain sequence
lenh = length(hn); %find length of h-time domain sequence
len = lenh + lenx - 1; %length of zero pad length
fastlen = 2.^nextpow2(len); %faster length for fft function

% zero-padding and taking the fft of xn
xind = [0 : (lenx-1)];
XF = fft(xn,fastlen);
Fx = length(XF);
Fdx = [0 : (Fx - 1)] ./Fx;

% zero-padding and taking the fft of hn
hind = [0 : (lenh-1)];
HF = fft(hn,fastlen);
Fh = length(HF);
Fdh = [0 : (Fh - 1)] ./Fh;

% multiplying in the Fourier domain to convolve in the time domain
YF = XF .*HF;

%taking the inverse transform of YF to convert it into the time domain
ynfft = ifft(YF);
zero_array = zeros(1,len);
yn(1:len) = ynfft(1:len);
leny = length(yn);
yind = [0 : (leny - 1)];
Fy = length(YF);
Fdy = [0 : (Fy - 1)] ./ Fy;

if (Fx < 1000) & (Fh < 1000) & (Fy < 1000) %make sure computer won't crash
figure (1) %plot everything into one figure

subplot(3, 2, 1) %time domain x
stem(xind, xn, '.')
title(' x[n] Sequence ')
xlabel(' Sample n ')
ylabel('Amplitude')
grid on

subplot(3, 2, 2) %frequency domain x
plot(Fdx, abs(XF))
title(' X[k] Spectrum ')
xlabel('Digital Frequency - cyc/sample')
ylabel('Magnitude Response')
grid on

subplot(3, 2, 3) %time domain h
```

```
stem(hind, hn, '.')
title(' h[n] Sequence ')
xlabel(' Sample n ')
ylabel('Amplitude')
grid on

subplot(3, 2, 4) %frequency domain h
plot(Fdh, abs(HF))
title(' H[k] Spectrum ')
xlabel('Digital Frequency - cyc/sample')
ylabel('Magnitude Response')
grid on

subplot(3, 2, 5) %time domain y
stem(yind, yn, '.')
title(' y[n] Sequence ')
xlabel(' Sample n ')
ylabel('Amplitude')
grid on

subplot(3, 2, 6) %frequency domain y
plot(Fdy, abs(YF))
title(' Y[k] Spectrum ')
xlabel('Digital Frequency - cyc/sample')
ylabel('Magnitude Response')
grid on
else
    %fprintf('Sequence too long to display');
end

end

%% Equilizer Coefficients
function [hn] = equalizer_coefficients(dB_gain)
% Function returns Bk values when given dB gain
% Bk is array of filter coefficients
% dB_gain is dB gain values for each frequency band

if length(dB_gain) ~= 9
   error('Error: Input array must have 9 values')
end

% Design an Equalizer with:
f_s = 44100;         % 44.1kHz sampling frequency
f_c(1) = 62.5;       % 1st band center
for i =1:8           % Build array of band centers
    f_c(i+1) = 2*f_c(i);
end
for i =1:9           % Calculate digital cutoffs
    F_c(i) = f_c(i)/f_s;
end

% Calculate Filter Length (M)
M = 1/F_c(1);       %find minimum spacing needed
M = ceil(M);        %round up to next integer
if (mod(M,2) == 0)  %if M is even make odd
    M = M + 1;
end

% Find slopes between band center (in dB/sample)
```

```
for i = 1:8
    slope(i) = (dB_gain(i+1) - dB_gain(i))/(2^(i-1));
end

% Build Desired magnitue response array
for i = 1:floor(M/2)                  % Build first half of array
    if (i <= 1);                      % 0~62.5Hz
        Hd_mag_db(i) = dB_gain(1);
    elseif ((i > 1) & (i <= 2));      % 62.5~125Hz
        Hd_mag_db(i) = Hd_mag_db(i-1)+slope(1);
    elseif ((i > 2) & (i <= 4));      % 125~250Hz
        Hd_mag_db(i) = Hd_mag_db(i-1)+slope(2);
    elseif ((i > 4) & (i <= 8));      % 250~500Hz
        Hd_mag_db(i) = Hd_mag_db(i-1)+slope(3);
    elseif ((i > 8) & (i <= 16));     % 500~1000Hz
        Hd_mag_db(i) = Hd_mag_db(i-1)+slope(4);
    elseif ((i > 16) & (i <= 32));    % 1~2kHz
        Hd_mag_db(i) = Hd_mag_db(i-1)+slope(5);
    elseif ((i > 32) & (i <= 64));    % 2~4kHz
        Hd_mag_db(i) = Hd_mag_db(i-1)+slope(6);
    elseif ((i > 64) & (i <= 128));   % 4~8kHz
        Hd_mag_db(i) = Hd_mag_db(i-1)+slope(7);
    elseif ((i > 128) & (i <= 256)); % 8~16kHz
        Hd_mag_db(i) = Hd_mag_db(i-1)+slope(8);
    elseif (i > 256);                 % 16-22kHz
        Hd_mag_db(i) = dB_gain(9);
    end
end

Hd_flip = fliplr(Hd_mag_db(1:end));         % Create mirrored response
Hd_array = [dB_gain(1) Hd_mag_db Hd_flip];  % Concatenate response with mirror
% dB gain(1) is needed for response value at F=0

Hd_mag = db2mag(Hd_array);   % Convert dB to linear

for k = 1:M; % create index vector
    F(k) = (k-1)/M;                         % find digital freq values
    Hd_phase(k) = -pi*(k-1)*(M-1)/M;        % create linear phase array
    Hd(k) = Hd_mag(k)*exp(j*Hd_phase(k));  % build desired freq respone
end

hn = real(ifft(Hd));                 % find difference equation coefficients
[h,w] = freqz(hn,1,f_s);             % calculate H(F)


%make plots (not required just for visual verification)
figure(1)
stem(F_c,dB_gain)
hold on
grid on
plot(w/(2*pi),mag2db(abs(h)))
xlabel('Digital Frequency (cycles/sample)')
ylabel('Magnitude (dB)')
title('Graphic Equalizer Designed by Frequency Sampling')

end
```