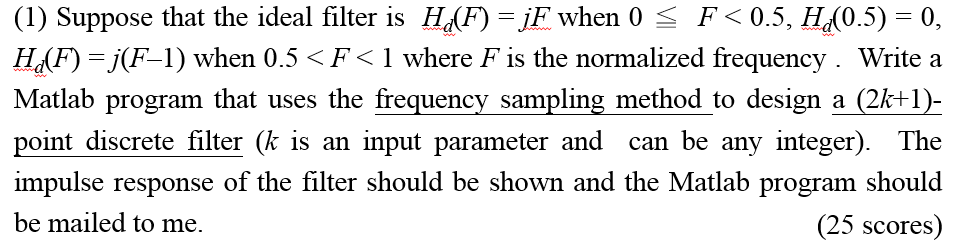
Advanced Digital Signal Processing Homework 2

**劉佳琛 R02943155**



The Matlab codes:

close all

clear all

k = 64;

N = 2 \* k + 1; % N = 21 for k = 10

m = 0: 1: N - 1;

Hd = (m/N - 1) \* j .\* (m/N > 0.5 & m/N < 1) + 0 .\* (m/N == 0.5) + m/N \* j .\* (m/N >= 0 & m/N < 0.5);

figure(1);

stem(m/N, imag(Hd));

title('the Hd sequence');

xlabel('m/N');

ylabel('Hd(m/N)');

r1 = zeros(1, N);

n = 0: 1: N - 1;

for m1 = 0: 1: N - 1

r1 = r1 + (1/N) \* (Hd(m1 + 1) \* exp(j \* 2 \* pi \* m1/N \* n));

end

r(1: (N - 1) /2) = r1((N + 1) / 2 + 1: N); %% r[n] = r1[n] for n = 0, 1, 2, ..., k

r((N + 1) / 2: N) = r1(1: (N + 1) /2); %% r[n - N] = r1[n] for n = k+1, k+2, ..., N-1

%stem((-k: 1: k), abs(r));

h = r;

figure(2);

subplot(2, 1, 1);

stem([0: length(h) - 1], real(h));

subplot(2, 1, 2);

stem([0: length(h) - 1], imag(h));

grid on;

figure(3);

stem([0: length(h) - 1], abs(h));

title('the h[n] sequence');

H = fft(h, 250);

figure(4);

plot(abs(H));

title('Amplitude of the spectrum of h[n]');

The impulse response h[n]:

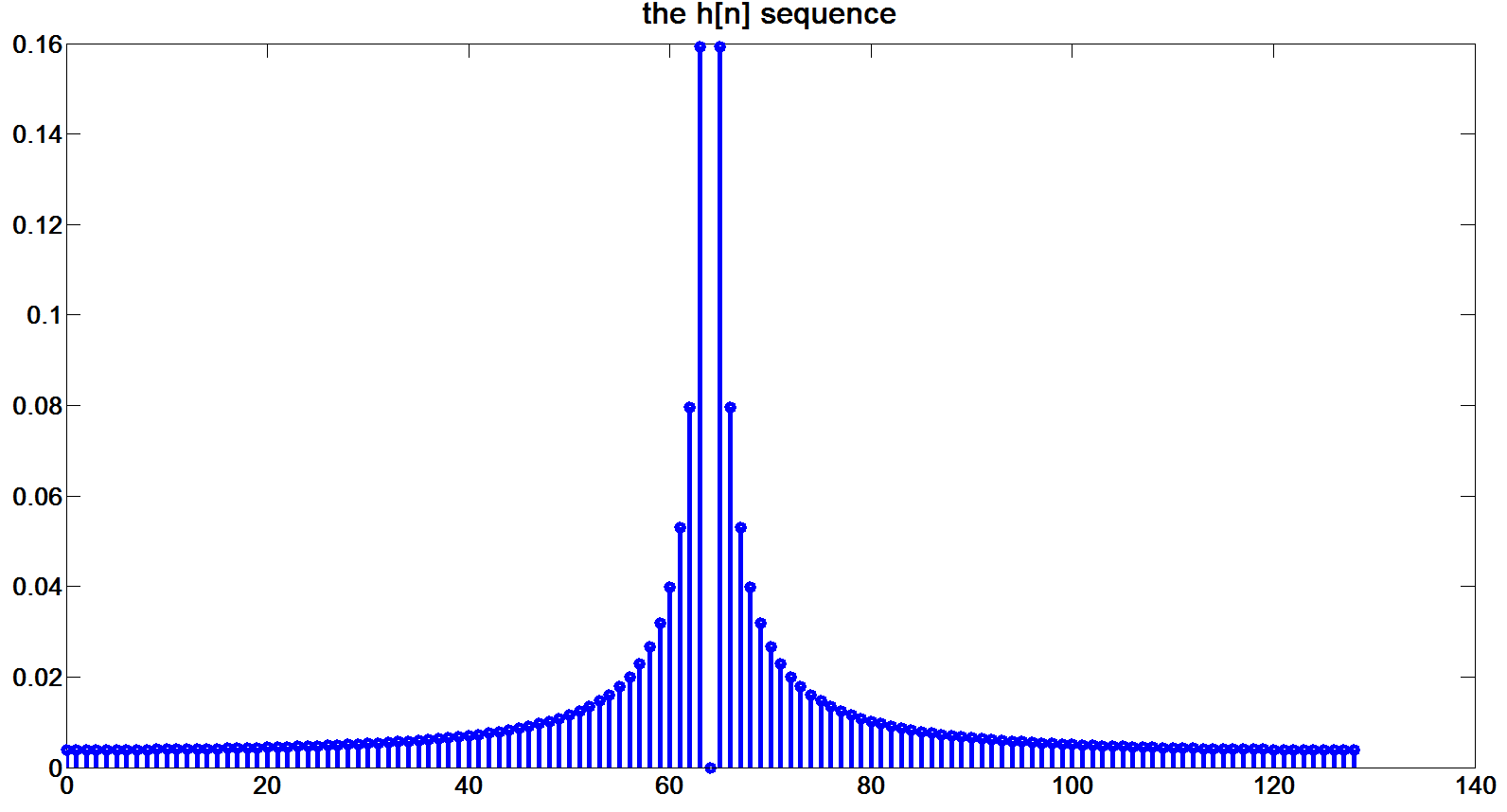


Fig 1. The Amplitude of the Impulse Response (|h[n]|)

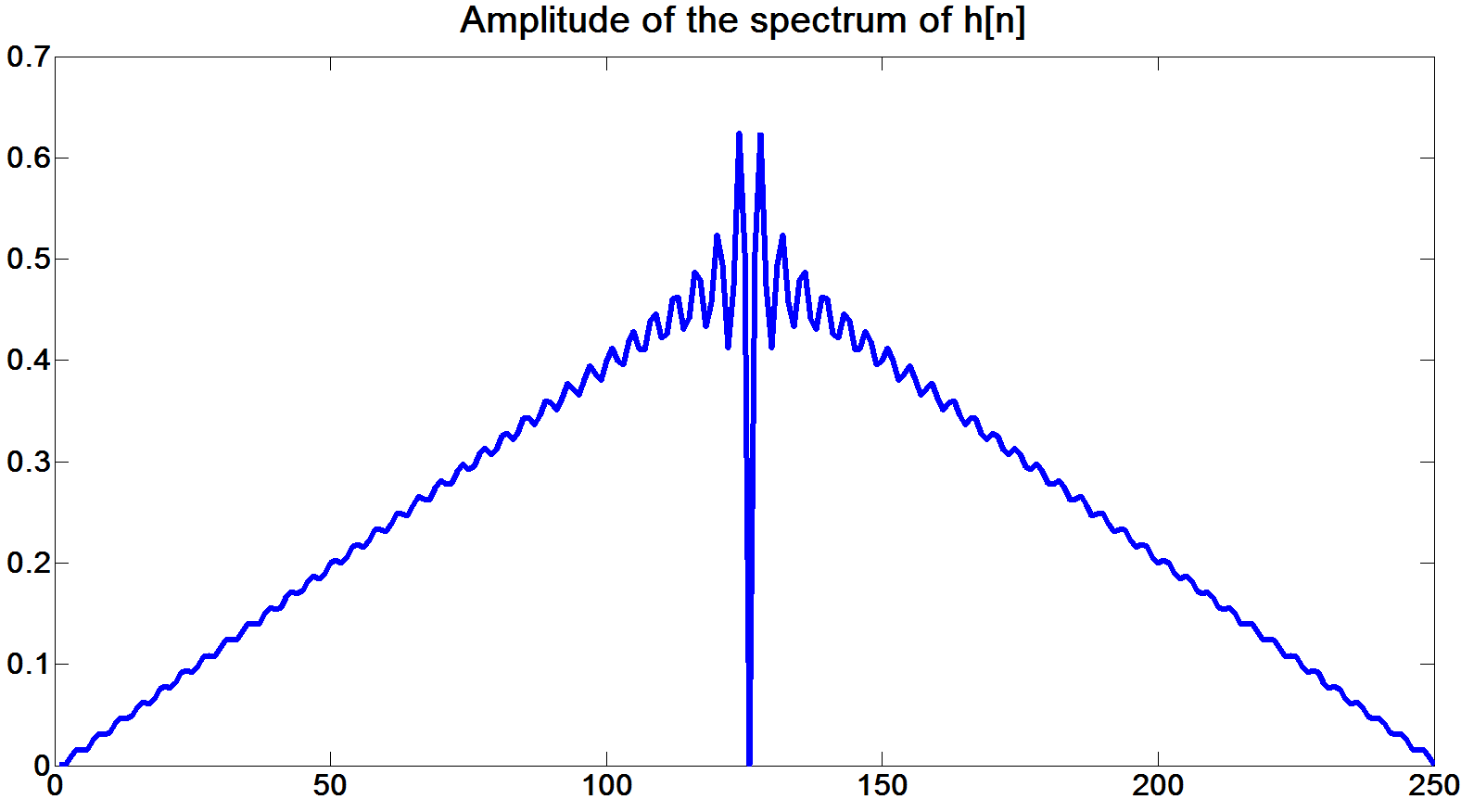
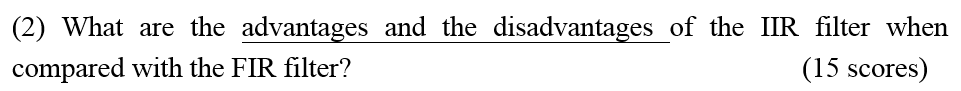


Fig 2. The Amplitude of the Transfer Function (|H(F)|)



**Advantages** of IIR filters:

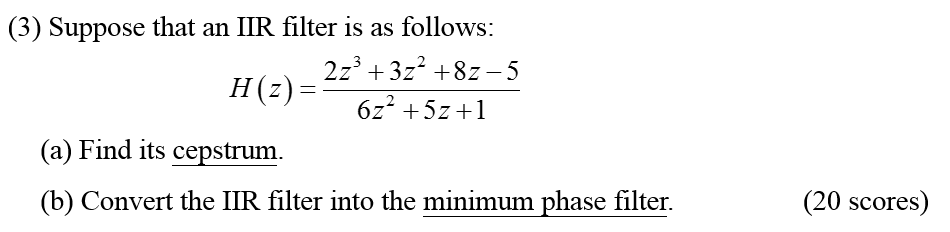
Easier to design & implement than FIR filters are, especially for conventional, theoretically optimized scenarios.

**Disadvantages** of IIR filters:

Impulse Responses of IIR filters are infinite, even when the input signals are finite in length.

For particular, unconventional scenarios (those that cannot be simply categorized as a low-pass, high-pass, band-stop or band-pass filter), IIR filters are harder to optimize in the design phase.

IIR filters are not 100% stable, as their poles might be located outside the unit circle (unlike those of the FIR filters).

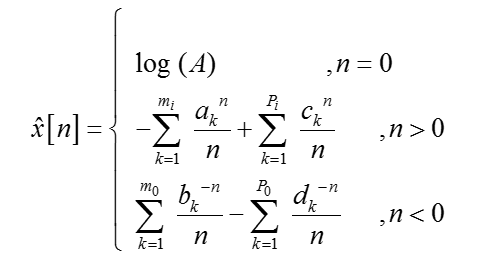


(a)

=> A = 10/6 = 5/3; a1 = 1/2; b1 = (-1 - 2i)/5; b2 = (-1 + 2i)/5; c1 = -1/3; c2 = -1/2;

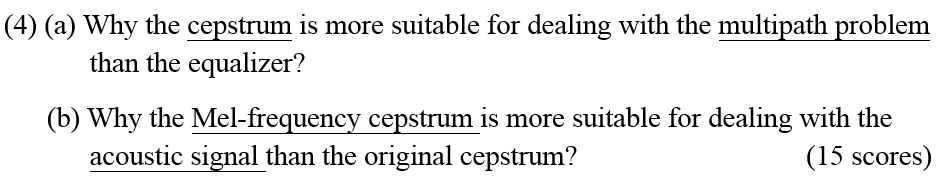
time delay = z-1.

Since



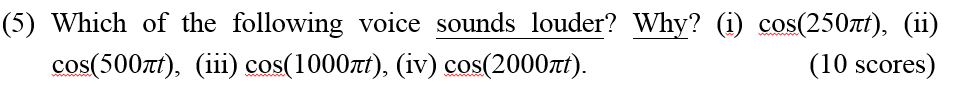
(b)

Obviously the zeros -1+2i and -1-2i are outside the unit circle, therefore to transform the filter represented by H(z), we need to rebuild the transfer function as below:



(a) Using the cepstrum, one is able to deal the delay of ONLY ONE path when calculating the “interfering effect” other paths have on the path at hand. In this way, delays of other paths do not need to be calculated when the interest is to filter the multi-path effect on the path under discussion.

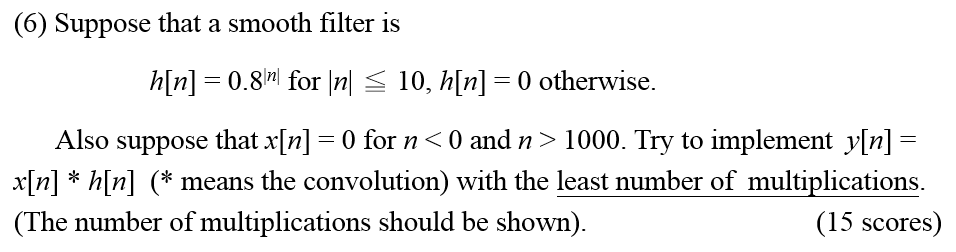
(b) The Mel-freq. cepstrum has a wider mask on a higher frequency, which is suitable for the acoustic “habit” of the human ear. For the Mel-frequency cepstrum (according to Wikipedia), the frequency bands are equally spaced on the mel scale (a log-based scale), which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.



cos(2000πt) > cos(1000πt) > cos(500πt) > cos(250πt)

Why:

Since the amplitudes are all the same (A = 1), the higher the frequency is, the easier it is for the sound to be received by the human ear as long as they are within the range of the human hearing (approximately 20Hz~20kHz).



For the known sequence of h[n], we have its Z-transform H(z) as:

* y1[n] = 0.8y[n - 1] + x[n] – 0.811x[n – 11];

y2[n] = 0.8y[n + 1] + x[n] – 0.810x[n + 10].

Originally h[n] has values for [-10, 10]; x[n] has values for [0, 1000], so y[n] would have the maximum “valued” interval as [-10, 1010].

However, due to the limitations imposed by the right-hand part of the two equations as well as the [-10, 1010] boundary, y1[n] and y2[n] only have values for [0, 1010] and [-10, 999], respectively.

0.8y[n - 1]: has values for [1, 1010], so 1010 multiplications for this part;

0.811x[n – 11]: has values of [11, 1010], so 1000 multiplications for this part;

0.8y[n + 1]: has values for [-9, 999], so 1009 multiplications for this part;

0.810x[n + 10]: has values for [-10, 990], so 1001 multiplications for this part.

Total # of multiplications necessary: 1010 + 1000 + 1009 + 1001 = 4020.