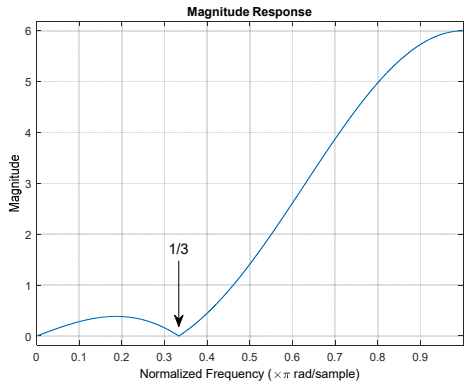
Multimedia Signal Processing 1st Module and Fundamentals of Multimedia Signal Processing

date: September 7th, 2023

Ex.1 (Pt.12)

A causal digital FIR filter with real coefficients, $H(\omega)$, has the following magnitude response:



[2 pts] Depict a possible zeros-poles plot.

[4 pts] Find a possible z-transform $H(z)$ of the filter and calculate its finite differences equation.

A signal $x(t)=10+5\cos\left(2\pi\cdot 5\cdot 10^3 t\right)+10\cos\left(2\pi\cdot 15\cdot 10^3 t\right)$ is sampled at 30kHz and then filtered with $H(z)$,

[6 pts] what will be the signal $y(n)=\ldots$ at the output of the filter?

Ex.2 (Pt.9)

A signal has a frequency band limited between 0 and 30kHz. It is sampled at 120 kHz and we need to resample it at 80 kHz.

[3 pts] Describe all the processing steps in order to obtain the resampled signal

[6 pts] Represent graphically the spectrum (with normalized frequencies) of the signal at each step of the resampling procedure.

CONTINUES ON THE BACK

Ex.3 (Pt.12) To be solved writing the MATLAB code on the paper.

1) [2 pt] We need to convert the sampling rate of a digital system from 1KHz to 600Hz. Build an FIR filter with 61 samples to enable the sampling rate conversion of the signals.
2) [2 pt] A sinusoidal signal x, with two frequency components and duration 0.5 seconds, enters the system. Build this signal such that, after the sampling rate conversion, only one component is kept and the other is filtered out. You choose the frequency components, motivating your choice.
3) [3 pt] Implement the interpolation part of the sampling rate conversion of x. In particular, to do so, consider each of the following possibilities (hint: be careful in selecting the correct signal to be filtered, which is not x…):
   - Use the function "conv" to filter the signal, defining the signal y.
   - Use the function "overlap_add.m" to filter the signal, defining the signal z.
   - Use the function "overlap_save.m" to filter the signal, defining the signal w.
   - The last two functions do not need to be written, take them for granted. They require as input parameters, in this order: the signal to be filtered, the FIR filter to use, the amount of signal samples to filter at a time (block-size); they return the filtered signal with the Overlap and Add and Overlap and Save techniques, respectively. You choose the parameters.
   - Plot the signals y, z and w in the same figure. How are they related? Which is the signals' length?
4) [2.5 pt] Instead of (3), implement the filtering process by computing the element-wise product between the signal and the filter in the DFT domain. Compute the DFT over the length of the input signal. Define the final signal as u.
   - Are there any differences between u and y, z and w? If yes, in which samples? Why?
   - If needed, select the minimum possible number of DFT samples to correct the result.
5) [2.5 pt] Select one of the signals y, z, w, u to complete the correct sampling rate conversion chain.
   - Compute the DFT of the signal x and that of the selected signal over 2048 samples.
   - Plot the absolute values of DFTs as a function of normalized frequencies.
   - Comment on the position of all the peaks.

Solutions

Ex.1

The filter frequency response presents two frequencies with zero magnitude in $\omega = 0$ and in $\omega = \frac{\pi}{3}$, furthermore its amplitude at the Nyquist frequency is 6.

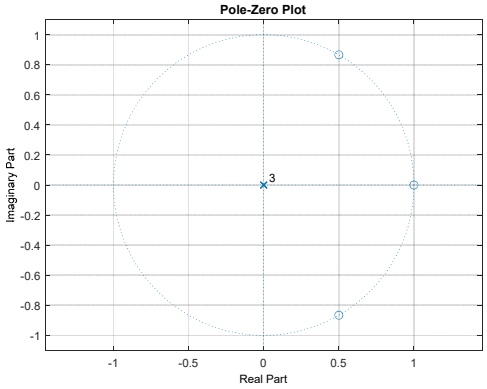
Since the filter has real coefficients, it will have just real or complex conjugate features, i.e. it will present a zero at $\omega = 0$ and a couple on conjugate zeros at $\omega = \pm\frac{\pi}{3}$. The filter will not present poles out of the origin. Since the filter removes completely the frequencies at $\omega = 0$ and at $\omega = \pm\frac{\pi}{3}$, the zeros will be on the unit circle of the pole-zero plot and, for the causality, three poles will be placed in the origin. A possible implementation could be:

$$H(z) = A \cdot \left(1 - z^{-1}\right)\left(1 - 2 \cdot \cos\left(\frac{\pi}{3}\right) z^{-1} + 1\right) = A\left(1 - z^{-1}\right)\left(1 - z^{-1} + z^{-2}\right) = A\left(1 - 2z^{-1} + 2z^{-2} - z^{-3}\right).$$

Imposing the value of 6 at the Nyquist means:

$$H(z = -1) = 6 \Rightarrow A(1 + 2 + 2 + 1) = 6 \Rightarrow A = 1.$$

Which gives the following pole-zero plot:



The signal sampled will be:

$$x[n] = 10 + 5\cos\left(2\pi \cdot \frac{5 \cdot 10^3}{30 \cdot 10^3} n\right) + 10\cos\left(2\pi \cdot \frac{15 \cdot 10^3}{30 \cdot 10^3} n\right) =$$

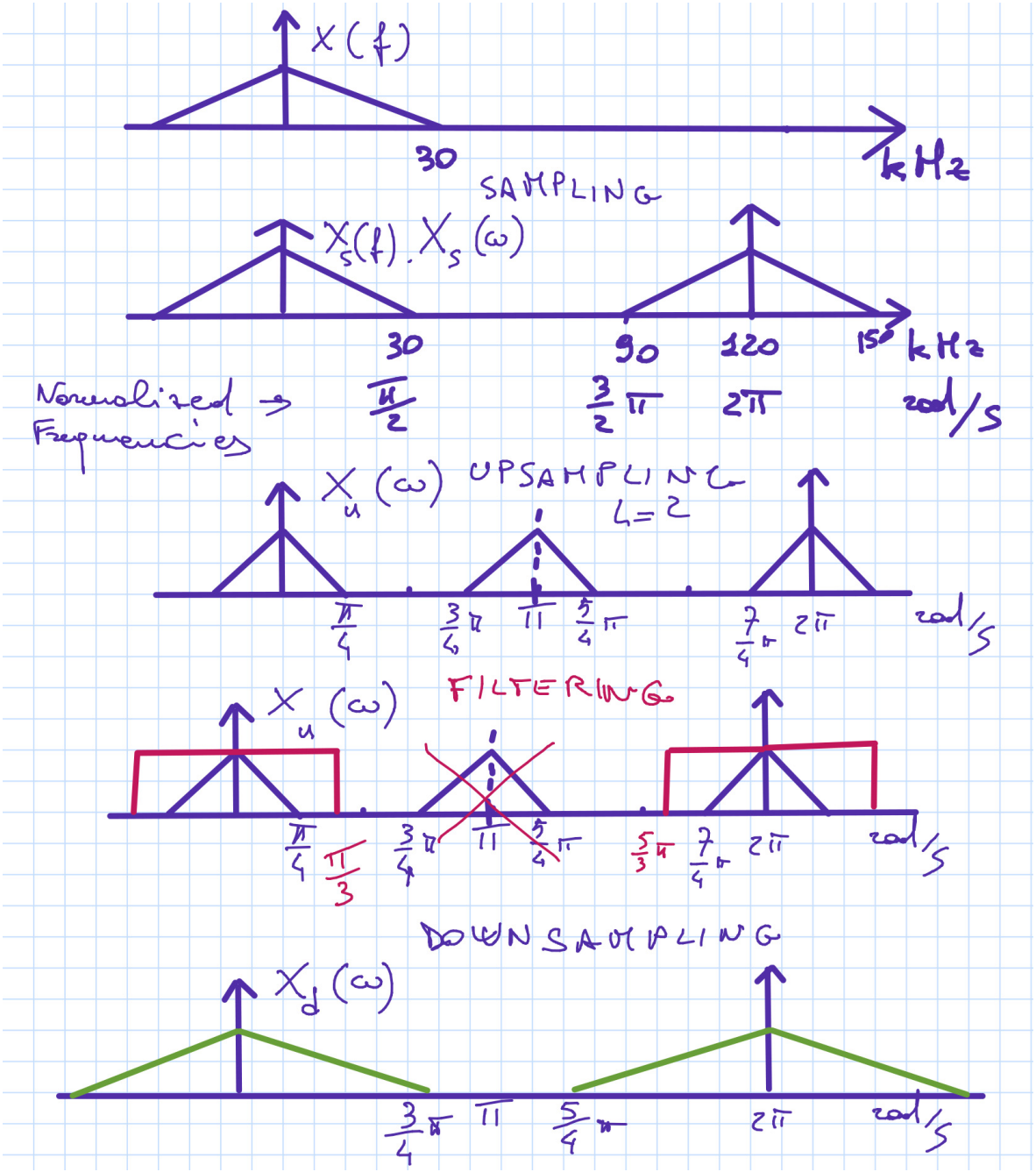
$$= 10 + 5\cos\left(\frac{\pi}{3} n\right) + 10\cos(\pi n)$$

The filter $H(z)$ will completely remove the first two components and amplify the Nyquist component:

$$y[n] = 6 \cdot 10\cos(\pi n) = 60\cos(\pi n)$$

Ex.2

Since the signal is resampled from 120 kHz to 80 kHz, it means that it has to be upsampled of an order of 2 (L=2), filtered with a low pass filter with a cut-off at $\omega_c = \frac{\pi}{3}$ and downsampled of an order of 3 (M=3).



Ex.3 (MATLAB CODE)

```
close all
clearvars
clc

%% 1. [2 pt]
```

```
% We need to convert the sampling rate of a digital system from 1KHz to
600Hz.
% Build an FIR filter with 61 samples to enable the sampling rate
conversion
% of the signals.

Fs = 1e3;
Fs_1 = 0.6e3;

% to find L and M --> rat(Fs_1/Fs) = 3/5 = L/M
L = 3;
M = 5;

% define the filter
cutoff = min([1/(2*L), 1/(2*M)]);
cutoff_filter = 2*cutoff;
N = 61;
% the order is N-1
h_multirate = L*fir1(N-1, cutoff_filter);

%% 2. [2 pt]

% A sinusoidal signal x, with two frequency components and duration 0.5
% seconds, enters the system. Build this signal such that, after the
% sampling rate conversion, only one component is kept and the other is
% filtered out. You choose the frequency components, motivating your
choice.

% If one frequency component is filtered out, it means that, after the
% upsampling, there is one frequency component > cutoff = 1/10.
% Therefore, we have to select a frequency component that, divided by L
% (due to the upsampling), is larger than 1/10 --> f1/L >= 1/10 --> f1
>=
% L/10 --> f1 >= 0.3.
% The remaining component should be f2 <= 0.3.
% for example, we can select these two contributions
f1 = 0.4;
f2 = 0.06;

% define the signal
time_axis = 0:1/Fs:.5;
x = cos(2*pi*f1*Fs*time_axis) + cos(2*pi*f2*Fs*time_axis);

%% 3. [3 pt]

% Implement the interpolation part of the sampling rate conversion of
x.
% In particular, to do so, consider each of the following possibilities
% (hint: be careful in selecting the correct signal to be filtered,
% which is not x):
% Use the function conv to filter the signal, defining the signal y.
% Use the function overlap_add.m, defining the signal z.
% Use the function overlap_save.m, defining the signal w.
% The last two functions require as input parameters, in this order:
% the signal to be filtered, the FIR filter to use, the amount of
signal
% samples to filter at a time (block-size); they return the filtered
signal
% with the Overlap and Add and Overlap and Save techniques,
respectively.
% You choose the parameters.
% Plot the signals y, z and w in the same figure. How are they related?
```

```
% Which is the signals' length?

% first, we need to upsample the signal x
x_up = zeros(1, length(x) * L);
x_up(1:L:end) = x;

% the signal to be filtered is x_up, not x

% conv
y = conv(x_up, h_multirate);

% the block-size should be larger than the filter length
% for instance, we can select:
B = length(h_multirate) + 10;
z = overlap_add(x_up, h_multirate, B);
w = overlap_save(x_up, h_multirate, B);

% plot
figure;
plot(y);
hold on;
plot(z, '--');
plot(w, '-.');
grid;
legend('y', 'z', 'w');

% the three signals are the same signal, as we implemented the linear
% convolution in three different ways.
% the signals' length is = length(x_up) + length(h_multirate) - 1

%% 4. [2.5 pt]

% Instead of (3), implement the filtering process by computing the
% element-wise product between the signal and the filter in the DFT
domain.
% Compute the DFT over the length of the input signal.
% Define the final signal as u.
% Are there any differences between u and y, z and w?
% If yes, in which samples? Why?
% If needed, select the minimum possible number of DFT samples to
correct
% the result.

X_up = fft(x_up);
H = fft(h_multirate, length(x_up));
u = ifft(X_up.*H);

% the signal u is different from y, z and w.
% first, the length corresponds to the length of x_up, which is lower
than
% the length of the other signals.
% second, there are cyclic convolution artifacts in the first
length(h_multirate) - 1
% samples.
% to visualize these artifacts (not required):
plot(u, 'o-');
grid;
legend('y', 'z', 'w', 'u');

% to obtain the same result, we need to select a number of samples for
the
% DFT equal to the length of the linear convolution
```

```
% length(x_up) + length(h_multirate) - 1
X_up = fft(x_up, length(x_up) + length(h_multirate) -1);
H = fft(h_multirate, length(x_up) + length(h_multirate) -1);
u = ifft(X_up.*H);

% now the result is the same as y, z, w (not required):
plot(u, 'x--');
grid;
legend('y', 'z', 'w', 'u', 'u (correct)');

%% 5. [2.5 pt]

% Select one of the signals y, z, w, u to complete the correct sampling
rate
% conversion chain.
% Compute the DFT of the signal x and that of the selected signal over
2048
% samples.
% Plot the absolute values of DFTs as a function of normalized
frequencies.
% Comment on the position of all the peaks.

% all the signals are equal; for example, we can select y.
x_down = y(1:M:end);

% dft
n_fft = 2048;
X_updown = fft(x_down, n_fft);
X = fft(x, n_fft);
norm_freq_axis = 0:1/n_fft:1 - 1/n_fft;

figure;
plot(norm_freq_axis, abs(X));
hold on;
grid;
stem(norm_freq_axis, abs(X_updown));
% in X, we find two peaks in f1 and f2 and the symmetric ones.
% in X_updown, we find only one peak centered in f2/L*M and the
symmetric
% one. The f1-component has been filtered out by the filter.

%% functions (not required)

function [filtered_signal] = overlap_add(signal, filter, L)

Lconv = L + length(filter) - 1;

% fft of the filter over Lconv samples
filter_f = fft(filter, Lconv);

% auxiliary variable
x_aux = signal;

% index for the blocks
b = 1;

while true

    % select the signal block
    x_block = x_aux(1:L);

    % fft of the signal block over Lconv samples
```

```
    X_block = fft(x_block, Lconv);

    % result of the cyclic conv over Lconv samples
    y_block = ifft(X_block .* filter_f);

    if b == 1
        y_oa = y_block;
    else
        % if the block is not the first one, add L zeros at the end of
        % the signal y_oa to contain the new samples related to the
block
        y_oa = padarray(y_oa, [0, L], 'post');

        % put the result in the right position
        y_oa(1 + (b-1)*L:end) = y_oa(1 + (b-1)*L:end) + y_block;
    end

    % update blocks
    b = b + 1;

    % delete the already processed block
    x_aux = x_aux(L+1:end);

    % if the number of remaining samples is less than the block size,
stop
    if length(x_aux) < L
        break
    end

end

% consider the last block of the signal
% (operations to be done are the same as in the while loop)
x_block = x_aux;
X_block = fft(x_block, Lconv);
y_block = ifft(X_block .* filter_f);
y_oa = padarray(y_oa, [0, L], 'post');
y_oa(1 + (b-1)*L:end) = y_oa(1 + (b-1)*L:end) + y_block;

% delete the final zeroes
filtered_signal = y_oa(1:length(signal) + length(filter) -1);

end

function [filtered_signal] = overlap_save(signal, filter, L)

% number of wrong samples due to the cyclic conv = overlap size.
overlap = length(filter) - 1;

% initialize the output signal (this will be the concatenation of the
% results related to each block, without overlap)
y_os = [];

% fft of the filter over L samples
filter_f = fft(filter, L);

% auxiliary variable
x_aux = signal;

% add P - 1 zeros at the beginning
x_aux = padarray(x_aux, [0, overlap], 'pre');
```

```
% index for the blocks
b = 1;

while true

    % select the signal block
    x_block = x_aux(1:L);

    % fft of the signal block over L samples
    X_block = fft(x_block, L);

    % result of the cyclic conv over L samples
    y_block = ifft(X_block .* filter_f);

    % delete the first P - 1 samples
    y_block = y_block(overlap + 1:end);

    % concatenate the result
    y_os = [y_os, y_block];

    % update blocks
    b = b + 1;

    % delete the already processed block, but take the next one with an
    % overlap = P - 1 samples
    x_aux = x_aux(L + 1 - overlap:end);

    % if the number of remaining samples is less than the block size,
stop
    if length(x_aux) < L
        break
    end

end

% consider the last blocks of the signal (operations to be done are the
same
% as in the while loop)

x_block = x_aux;
X_block = fft(x_block, L);
y_block = ifft(X_block .* filter_f);
y_block = y_block(overlap + 1:end);
y_os = [y_os, y_block];
% for managing the last blocks
n_last_blocks = ceil(length(x_aux) / (L + 1 - overlap));
for last_block_idx = 1:n_last_blocks
    x_block = x_aux(1 + (L - overlap)*last_block_idx:end);
    X_block = fft(x_block, L);
    y_block = ifft(X_block .* filter_f);
    y_block = y_block(overlap + 1:end);
    y_os = [y_os, y_block];
end

% delete the final zeroes
filtered_signal = y_os(1:length(signal) + length(filter) -1);

end
```