Lab 2: Linear and Block Convolution on Audio file.

Name: Himanshu Sharma Roll Number: 1610110149 Mail ID: hs583@snu.edu.in Instructor: Prof. Vijay Chakka

```
function a=main()
    % defining the 'main' namespace.

    % read the audio file.
    [x, fs] = audioread('audio.mp3');
    % take 5 second sample of the audio.
    v = x(1:5*fs);
    v = reshape(v, [], length(v));

    % define 'h1'.
    h1 = load('F:
\1610110149_LAB2_FRI_4TO6\1610110149_LAB2_FRI_4TO6\hpImpulseRes.mat');
    h1 = h1.h1;
    % define 'h2'.
    h2 = load('F:
\1610110149_LAB2_FRI_4TO6\1610110149_LAB2_FRI_4TO6\lpImpulseRes.mat');
    h2 = h2.h1;

    %
 -----------------------------------------------------------------
    % now use block convolution on h1 with block size of 512.
    %
 =================================================================
    y2 = block_convolve(v, h1, 512);
    y2 = reshape(y2, length(y2), []);

    % write the audio.
    audiowrite('F:
\1610110149_LAB2_FRI_4TO6\1610110149_LAB2_FRI_4TO6\hpf.wav', y2, fs);

    % define time array for block convolution output.
    T = 0:1/fs:length(y2)*(1/fs) - (1/fs);
    % define 5 sec time.
    t = 0:1/fs:5-(1/fs);
    % define discrete time for h1.
    th = 0:1:length(h1)-1;

    figure;
    % plot the audio signal.
    subplot(3,1,1);
    plot(t, v);
    title('Audio Signal of 5 sec');
    xlabel('Time (seconds)');
    ylabel('$$x(n)$$', 'interpreter', 'latex');
    grid on;
```

```
% plot the impulse resp h1.
subplot(3,1,2);
stem(th, h1);
xlabel('Discrete Time (n)');
ylabel('$$h_{1}(n)$$', 'interpreter', 'latex')
title('Impulse response of High Pass Filter');
grid on;

% plot the block convolution output.
subplot(3,1,3);
plot(T, y2);
xlabel('Time (seconds)');
ylabel('$$y(n)$$', 'interpreter', 'latex');
title('Block Convolution Output');
grid on;

%
-------------------------------------------------------------------
% now use block convolution on h2 with block size of 512.
%
===================================================================
y3 = block_convolve(v, h2, 512);
y3 = reshape(y3, length(y3), []);

% write the audio
audiowrite('F:
\1610110149_LAB2_FRI_4TO6\1610110149_LAB2_FRI_4TO6\lpf.wav', y3, fs);

% define time array for block convolution output.
T = 0:1/fs:length(y3)*(1/fs) - (1/fs);
% define 5 sec time.
t = 0:1/fs:5-(1/fs);
% define discrete time for h2.
th = 0:1:length(h2)-1;

figure;
% plot the audio signal.
subplot(3,1,1);
plot(t, v);
title('Audio Signal of 5 sec');
xlabel('Time (seconds)');
ylabel(' $$x(n)$$', 'interpreter', 'latex');
grid on;

% plot the impulse resp h2.
subplot(3,1,2);
stem(th, h2);
xlabel('Discrete Time (n)');
ylabel(' $$h_{2}(n)$$', 'interpreter', 'latex')
title('Impulse response of Low Pass Filter');
grid on;

% plot the block convolution output.
```

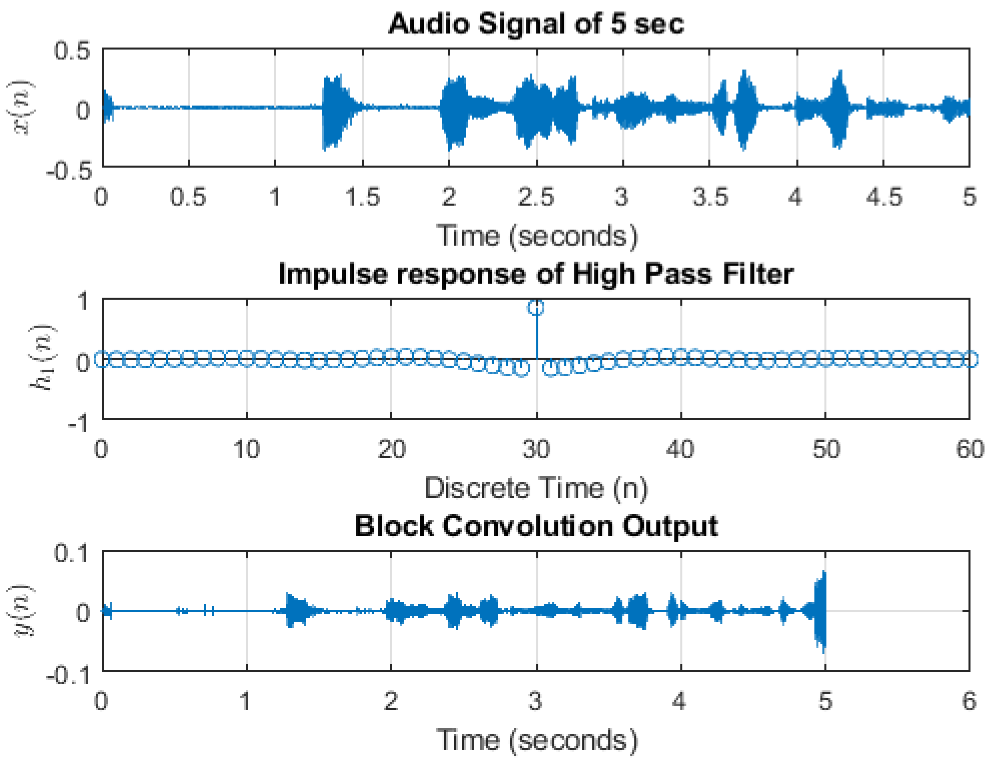
```
subplot(3,1,3);
plot(T, y3);
xlabel('Time (seconds)');
ylabel(' $$y(n)$$', 'interpreter', 'latex');
title('Block Convolution Output');
grid on;
%~~~~~~~~~~~~~~~~~~~~ DONE WITH AUDIO
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

%
-----------------------------------------------------------------
% Self taken example to demonstrate linear convolution made by the
% author.
%
=================================================================
% input signal is 'a'.
a = [1,2,3,4,5,6,7];
% impulse response is 'h'.
h = [1,0,-1];

% do the linear convolution by user defined function.
c = linear_convolution(a, h);

figure;

subplot(3,1,1);
stem(0:1:length(a)-1,a);
title('Input Signal');
xlabel('$$n$$', 'interpreter', 'latex');
ylabel('$$x(n)$$', 'interpreter', 'latex');
grid on;

subplot(3,1,2);
stem(0:1:length(h)-1, h);
title('Impulse Response');
xlabel('$$n$$ (integer value)', 'interpreter', 'latex');
ylabel('$$h(n)$$', 'interpreter', 'latex');
grid on;

subplot(3,1,3);
stem(0:1:length(c)-1, c);
title('Output Signal');
xlabel('$$n$$', 'interpreter', 'latex');
ylabel('$$y(n)$$', 'interpreter', 'latex');
grid on;

% ------------------------------------------------

function y = linear_convolution(x, h)

    % first add zeros to the array which has less length.
    if length(x) < length(h)
        zeros_needed = length(h) - length(x);
        x = [x zeros(1, zeros_needed)];
```

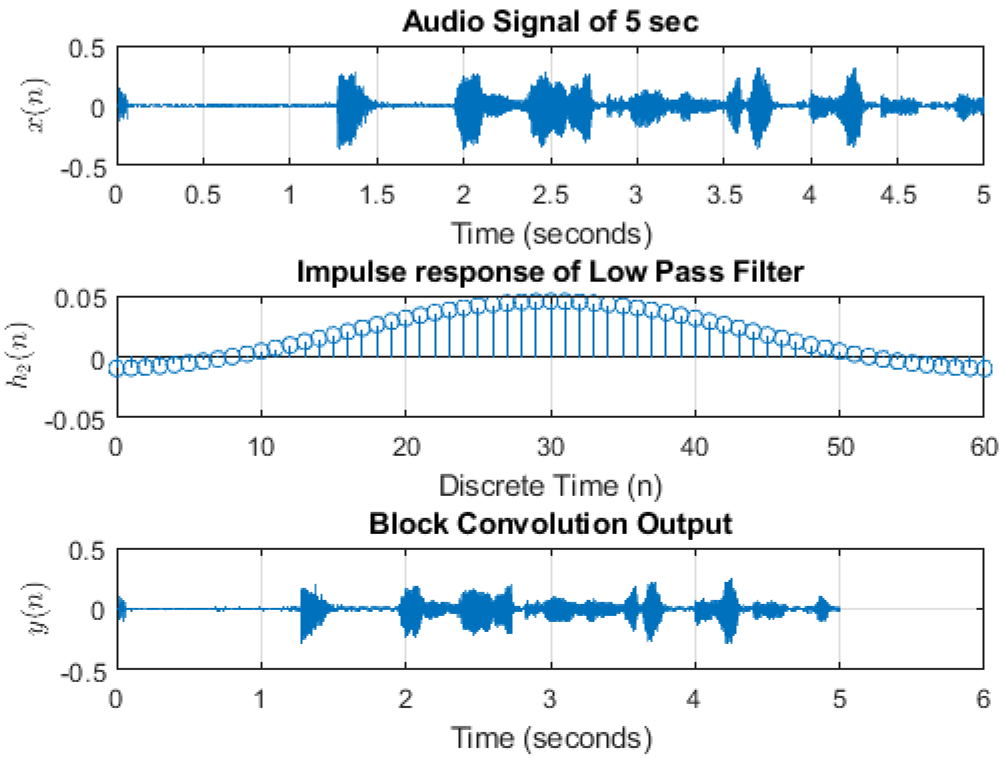
```
        else
            zeros_needed = length(x) - length(h);
            h = [h zeros(1, zeros_needed)];
        end;
        % define the output array 'y' as array of zeros.

        y = zeros(1, length(x)+length(h)-1);

        % reverse 'x' and 'h' so that multiplication can be done
iteratively.
        x = fliplr(x);
        h = fliplr(h);

        % now iterate in 'h' element by element and multiply each of
them to 'x'
        % to create a block.
        for i = 1:length(h)
            % define current block
            block = h(i)*x;

            % add zeros to front portion.
            block = [zeros(1, i-1) block];

            % now add zeros at the end of this block.
            % the number of zeros at the end must be the length(y) -
length(block)
            block = [block zeros(1, length(y)-length(block))];

            % add this block to 'y'
            y = y + block;
            %disp(fliplr(block));

        end;
         % since the result is reversed, reverse it again to get
correct answer.
         y = fliplr(y);
    end;

    % define block convolution function now.
    function y = block_convolve(x, h, block_size)
        % add zeros to 'x' and 'h' to make its length a multiple of
block_size.
        x = [x zeros(1, block_size - rem(length(x), block_size))];
        h = [h zeros(1, block_size - rem(length(h), block_size))];

        % initialise a blank output array 'y' of length(x) + length(h)
- 1
        y = zeros(1, length(x)+length(h)-1);

        % decide whose length is large, x or h.
        if length(x) > length(h)
            big = x;
            small = h;
        else
```

```
            big = h;
            small = x;
        end;

        % initialise a counter variable with 1.
        counter = 1;

        % now iterate for each block in the longer array, i.e., big.
        for i = 1:block_size:length(big)
            current_block = big(i:i+block_size-1);
            %disp('Current Block:');
            %disp(current_block);

            % now convolve this block with 'small'.
            out = linear_convolution(current_block, small);

            % add zeros in front of out.
            out = [zeros(1, counter-1) out];

            % add zeros at end of out. After this the length becomes
 equal to that of
            % length of 'y'.
            out = [out zeros(1, length(y) - length(out))];
            %cdisp('Output of current block');
            %disp(out);

            % add the final out block to 'y'.
            y = y + out;
            counter = counter + block_size;
        end;
    end;
end

% INFERENCE OF CONVOLUTED AUDIO.

% Using High Pass filter.
% ------------------------
% When a HPF is used on the 5 second audio, it allows high frequency
 content present in the audio to pass and
% the low frequency content is supressed. Therefore, its like
 sharpening of an audio file, just like equivalent of
% sharpening an image or detecting edges in image, since abrupt
 changes in the amplitude arises due to high frequency
% content, be it an image or an audio. So, the audio that is heared is
 sharp and crisp, somewhat high pitch.
%
% Using a Low Pass Filter.
% ------------------------
% The opposite happens in this case. The low pass filter allows low
 frequencies to pass and higher frequencies are
% blocked. Therefore, its equivalent to audio blurring, just like
 blurring and image, because the low frequency content
% returns all the information apart from abrupt changes. Hence, the
 audio that is heared now is of low pitch, not sharp and
```

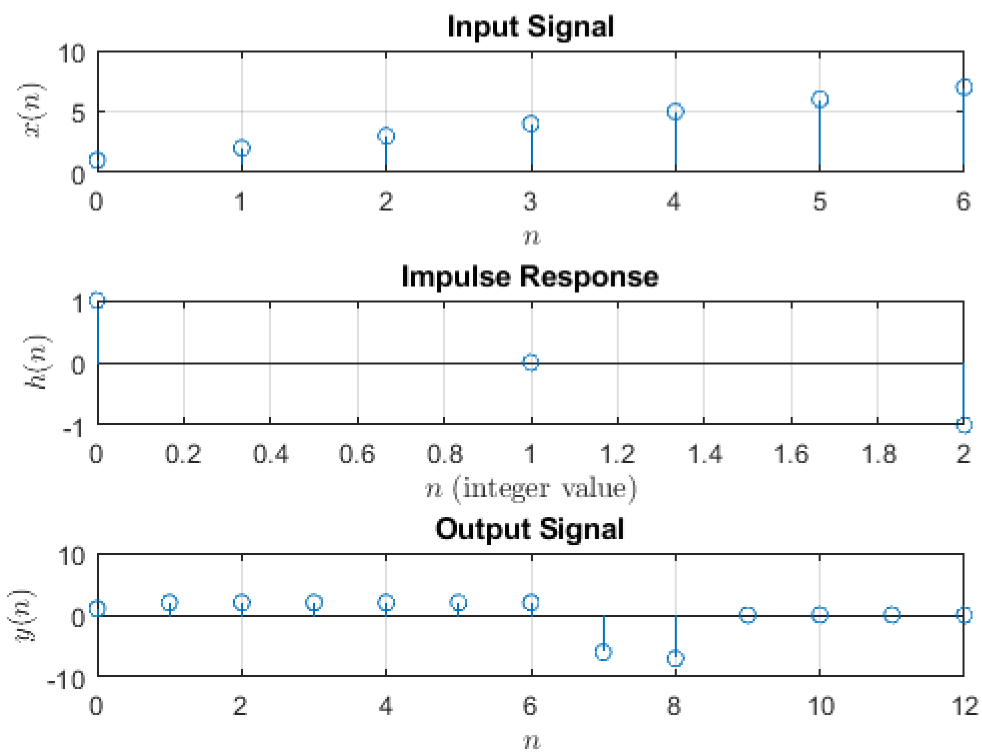
```
% also its not crisp.
```

```
ans =

     1     2     3     4     5     6     7
```





Audio Signal of 5 sec
$x(n)$
Time (seconds)
Impulse response of Low Pass Filter
$h_2(n)$
Discrete Time (n)
Block Convolution Output
$y(n)$
Time (seconds)



Input Signal
$x(n)$
n
Impulse Response
$h(n)$
n (integer value)
Output Signal
$y(n)$
n

Published with MATLAB® R2017b