Handout Q: Four Ways to Filter a Signal

Problem: Evaluate four ways to filter an input signal. Run `waystofilt.m` on page 143 (Section 7.2.1) of Johnson, Sethares & Klein using

- *h*[*n*] that is a four-symbol raised cosine pulse with β = 0.75 (4 samples/symbol, i.e. 16 samples)
- *x*[*n*] that is an upsampled 8-PAM symbol amplitude signal with *d* = 1 and 4 samples/symbol and that is defined as the following 32-length vector (where each number is a sample value) as

$$\mathbf{x} = [-7\ 0\ 0\ 0\quad -5\ 0\ 0\ 0\quad -3\ 0\ 0\ 0\quad -1\ 0\ 0\ 0\quad 1\ 0\ 0\ 0\quad 3\ 0\ 0\ 0\quad 5\ 0\ 0\ 0\quad 7\ 0\ 0\ 0]$$

In the code provided by Johnson, Sethares & Klein, please replace `plot` with `stem` so that the discrete-time signals are plotted in discrete time instead of continuous time.

Please comment on the different outputs. Please state whether each method implements linear convolution or circular convolution or something else. Please see the online homework hints.

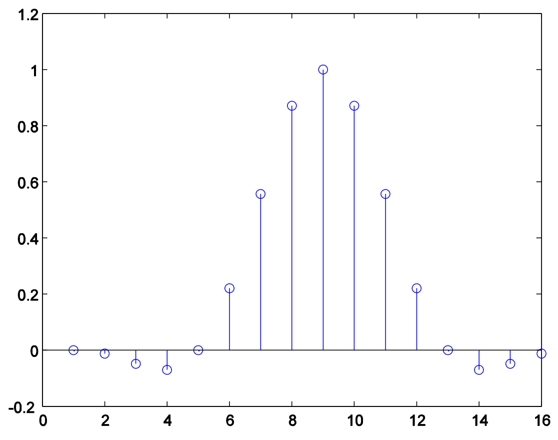
Hints: To compute the values of h, please use the "rcosine" command in Matlab and not the "SRRC" command. The length of h should be 16. The syntax of the "rcosine" command is

```
rcosine(Fd, Fs, TYPE_FLAG, beta)
```

The ratio Fs/Fd must be a positive integer. Since the the number of samples per symbol is 4, Fs/Fd must be 4. The rcosine function is defined in the Matlab communications toolbox.

Running the rcosine function with these parameters gives a pulse shape of 25 samples. We want to keep four symbol periods of the pulse shape. That is, we want to keep two symbol periods to the left of the maximum value, the symbol period containing the maximum value as the first sample, and the symbol period immediately following that:

```
rcosinelen25 = rcosine(1, 4, 'fir', 0.75);
h = rcosinelen25(5:20);
stem(h)
```



Some of the methods yield linear convolution, and some do not. With an input signal of 32 samples in length and a pulse shaping filter with an impulse response of 16 samples in length, linear convolution would produce a result that is 47 samples in length (i.e., 32 + 16 - 1).

For the FFT-based method, the length of the FFT determines the length of the filtered result. An FFT length of less than 47 would yield circular convolution, but it wouldn't be linear convolution. When the FFT length is long enough, the answer computed by circular convolution is the same as by linear convolution.

Consider when the filter is a block in a block diagram, as would be found in Simulink or LabVIEW. When executing, the filter block would take in one sample from the input and produce one sample on the output. How many times to execute the block? As many times as there are samples on the input. How many samples would be produced? As many times as the block would be executed.

In particular, pay attention to the use of the FFT to implement linear filtering. A similar trick is used in multicarrier communication systems, such as DSL, WiFi (IEEE 802.11a/g), WiMax (IEEE 802.16e-2005), next-generation cellular data transmission (LTE), terrestrial digital audio broadcast, and handheld and terrestrial digital video broadcast.

Solution: The filter is given by its impulse response $h[n]$ that has a length of L_h samples. The signal is given by $x[n]$ and it has a length of L_x samples. Both the impulse response and input signal are causal. In this problem, L_h is 16 samples and L_x is 32 samples.

The ***first way*** of filtering computes the output signal as the linear convolution of $x[n]$ and $h[n]$:

$$y_{linear}[n] = x[n] * h[n] = \sum_{m=0}^{L_h-1} h[m]\ x[n-m]$$

Linear convolution yields a signal of length L_x+L_h-1 = 47 samples.

The ***second way*** is to use the filter command in Matlab/Mathscript. The filter command produces one output sample for each input sample. This is a common behavior for a filter block in a block diagram simulation framework, e.g. Simulink or LabVIEW. When executing, the filter block would take in one sample from the input and produce one sample on the output. The scheduler will execute the block as many times as there are samples on the input. So, the length of the filtered signal would be L_x = 32 samples. To obtain an output of length of L_x+L_h-1 samples, one would append L_h-1 zeros to $x[n]$.

The ***third way*** is compute the output by using a Fourier-domain approach. For linear convolution, the discrete-time Fourier transform of the linear convolution of $x[n]$ and $h[n]$ is simply the product of their individual discrete-time Fourier transforms. The product could then be inverse transformed to find the filtered signal in the discrete-time domain. That approach, however, is difficult to automate using only numeric calculations. An alternative is to use the Fast Fourier Transform (FFT).

The FFT of the *circular convolution* of $x[n]$ and $h[n]$ is the product of their individual FFTs. In circular convolution, the signals $x[n]$ and $h[n]$ are considered to be periodic with period N. One period of N samples of the circular convolution is defined as

$$y_{circular}[n] = x[n] \otimes_N h[n] = \sum_{m=0}^{N-1} h[((m))_N]\; x[((n-m))_N]$$

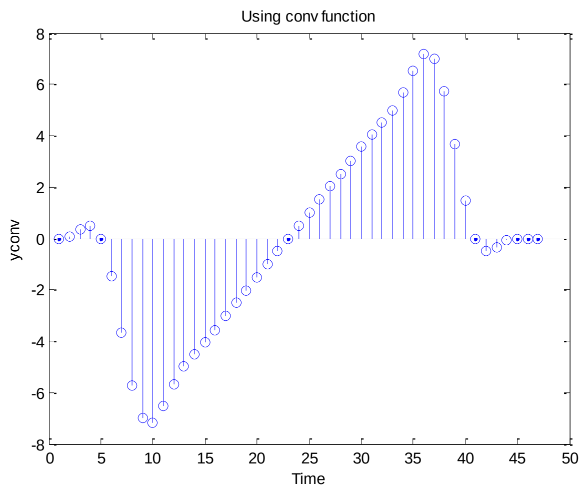
where $((\bullet))_N$ means that the argument is taken modulo N. We will henceforth refer to the circular convolution between periodic signals of length N as circular convolution of length N. On a programmable digital signal processor, we would use the modulo addressing mode to accelerate the computation of circular convolution.

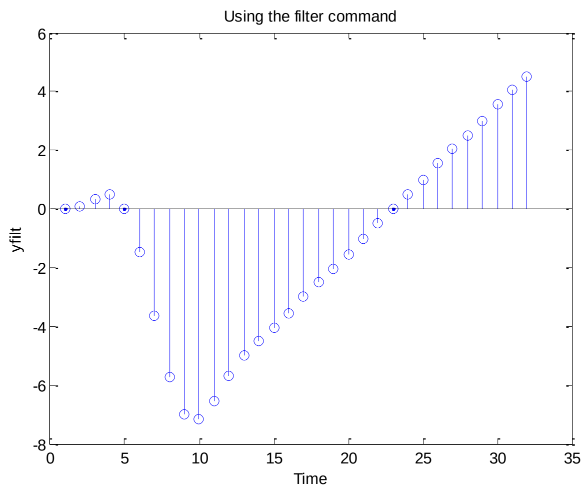
Circular convolution of two finite-length sequences $x[n]$ and $h[n]$ is equivalent to linear convolution of those sequences by padding (appending) L_x-1 zeros to $h[n]$ and L_h-1 zeros to $x[n]$ so that both of them are of the same length and using a circular convolution length of L_x+L_h-1 samples. This is the approach used in the FFT-based method in this problem.

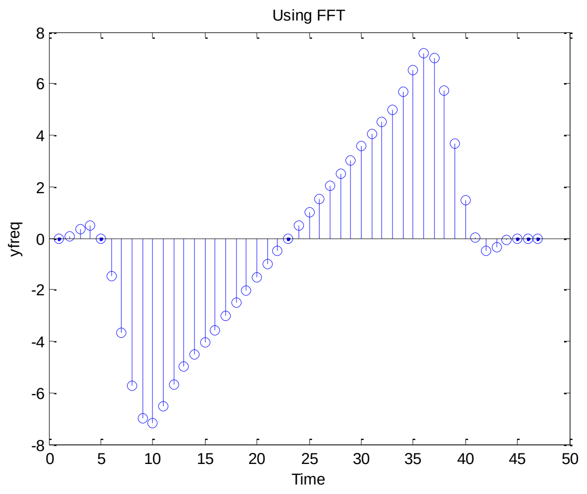
The FFT-based method to compute the linear convolution uses an FFT length of N of L_x+L_h-1. First, the FFT of length N of the zero-padded $x[n]$ is computed to give $X[k]$, and the FFT of length N of the zero-padded $h[n]$ is computed to give $H[k]$. Second, the product $Y_{circular}[k] = X[k]\, H[k]$ for $k = 0,\ldots, N$-1 is computed. Then, the inverse FFT of length N of $Y_{circular}[k]$ is computed to find $y_{circular}[n]$. This ***third way*** results in an output signal of L_x+L_h-1 = 47 samples.

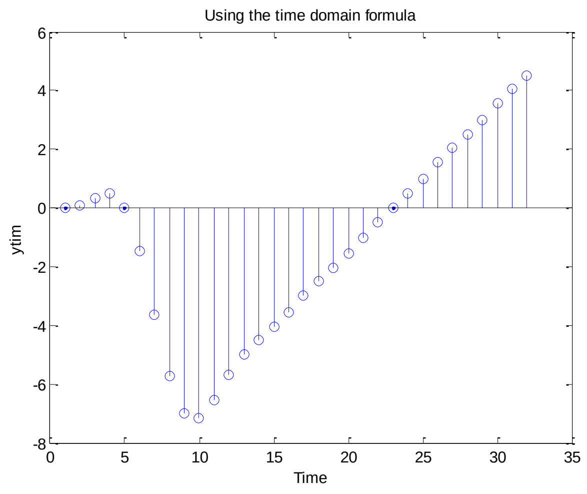
The ***fourth way*** to filter a signal uses a time-domain formula. It is an alternate implementation of the same approach used by the filter command. Hence, this approach gives an output of length L_x = 32 samples.

```
% waystofilt.m "conv" vs. "filter" vs. "freq domain" vs. "time domain"
over=4; % 4 samples/symbol
r=0.75; % roll-off
rcosinelen25 = rcosine(1, 4, 'fir', 0.75);
h = rcosinelen25(5:20);
x= [-7 0 0 0  -5 0 0 0  -3 0 0 0  -1 0 0 0  1 0 0 0  3 0 0 0  5 0 0 0  7 0 0 0];
yconv=conv(h,x) ;                       % (a) convolve x[n] * h[n]
n=1:length(yconv);stem(n,yconv)
xlabel('Time');ylabel('yconv');title('Using conv function'); figure
yfilt=filter(h,1,x) ;                   % (b) filter x[n] with h[n]
n=1:length(yfilt);stem(n,yfilt)
xlabel('Time');ylabel('yfilt');title('Using the filter command'); figure
N=length(h)+length(x)-1;                % pad length for FFT
ffth=fft([h zeros(1,N-length(h))]);     % FFT of impulse response = H[k]
fftx=fft([x, zeros(1,N-length(x))]);    % FFT of input = X[k]
ffty=ffth .* fftx;                        % product of H[k] and X[k]
yfreq=real(ifft(ffty));                 % (c)IFFT of product gives y[n]
                                        % it's complex due to roundoff
n=1:length(yfreq); stem(n,yfreq)
xlabel('Time');ylabel('yfreq');title('Using FFT'); figure
z=[zeros(1,length(h)-1),x];             % initial state in filter = 0
for k=1:length(x)                       % (d) time domain method
  ytim(k)=fliplr(h)*z(k:k+length(h)-1)';        % iterates once for each x[k]
end                                     % to directly calculate y[k]
n=1:length(ytim); stem(n,ytim)
xlabel('Time');ylabel('ytim');title('Using the time domain formula');
%end of function
```


Using conv function
yconv
Time
8
6
4
2
0
-2
-4
-6
-8
0
5
10
15
20
25
30
35
40
45
50


Using the filter command
yfilt
Time
6
4
2
0
-2
-4
-6
-8
0
5
10
15
20
25
30
35


Using FFT
yfreq
Time
8
6
4
2
0
-2
-4
-6
-8
0
5
10
15
20
25
30
35
40
45
50


Using the time domain formula
ytim
Time
6
4
2
0
-2
-4
-6
-8
0
5
10
15
20
25
30
35