Lab 11 – FIR filter design using frequency sampling

11.1 Filter design using inverse DFT

In this problem we will design an odd length, linear phase, low-pass FIR filter. Write a function `type1_dft()` that computes a filter tap vector *h* by determining the ideal frequency response at *N* equally-spaced frequencies and taking the inverse DFT of this vector. Your function should take the inputs

- N, an odd positive integer (>=3)
- wc, a frequency cutoff in [*0, pi*)

and return the output h, a length-N vector of filter taps resulting from taking the inverse DFT. Remember that discrete-time filters must be *2*pi* -periodic, and that the phase should be linear with a slope related to the filter length N.

The resulting filter should be real and symmetric; you can get rid of small imaginary parts by setting h equal to real(h), but if the imaginary part is large you probably made an error. You can use the template given below.

```
function [w,h] = type1_dft(N,wc)
    % Create vector of equally-spaced frequencies
    w=...
    % Create ideal amplitude response of low-pass filter (remember,
    it should be symmetric about w = pi)
    Ad=...
    % Compute linear phase vector using correct slope
    phi = exp(-j*...);
    % Compute ideal frequency samples as product of Ad and phi
    H = ...
    % Compute filter taps via inverse DFT
    h = ifft(H);
    % Make result real to get rid of near-zero imaginary parts
    h = real(h);
    % Plot impulse response, magnitude response, and phase response
    over a finer frequency grid
    ...
end
```

(a) For the filter you design, plot (in same figure) the impulse response, magnitude response, and phase response for cut-off frequency $\omega_c = 0.4\,\pi$.
(b) How is the filter frequency response (i.e. DTFT) related to the ideal filter samples?
(c) How does the designed filter change as N is varied?
(d) Using the above method of frequency sampling, write another matlab function which does band-pass filtering with the pass band from $\omega_{c1} = 0.3\pi$ to $\omega_{c2} = 0.6\pi$.

11.2 Filter design with transition band

The overshoot of a digital linear-phase low-pass FIR filter can be lessened substantially if we allow non-binary amplitude samples near the cutoff frequency, also known as using a transition band.

Write a function `transitionband()` that takes the inputs

- N, the length of FIR low-pass filter (N should be odd)
- wc, the cutoff frequency of the filter in radians
- tbvals, two transition band values to be used on either side of the cutoff frequency

The function should return h, the taps of the designed filter.

```
function [h] = transitionband(N,wc,tbvals)
    % Create vector of N equally-spaced frequencies
    w = ...
    % Create ideal amplitude response of low-pass filter (remember,
    it should be symmetric about w = pi)
    Ad = ...
    % Determine which indices correspond to the samples just to the
    % left and the right of the cutoff frequency (if cutoff freq
    falls exactly
    % on a sample, use that sample and the one to the right).
    wleftind = ...
    wrightind = wleftind + 1;
    % Update amplitude response with given transition band values at
    these frequencies (remember the symmetry around pi);
    Ad(...) = tbvals(1);
    Ad(...) = tbvals(2);
    % Compute linear phase vector using correct slope
    phi = ...
    % Compute ideal frequency samples as product of Ad and phi
    H = Ad.*phi;
    % Compute filter taps via inverse DFT
    h = ifft(H);
    % Make result real to get rid of near-zero imaginary parts
    h = real(h);
    % Compute finely-spaced frequency response of designed filter
    [H,wfine] = freqz(h,1,1024);
end
```

Keeping the sampling locations the same and adjusting the transition band values, we can get a very smooth (i.e., low overshoot) filter, at the cost of a wider transition band. Note that much of the solution to Problem 11.1 can be re-used for this problem, but the amplitude response must be adjusted. Use the template given above for your filter design.

11.3 Filtering of signals

Generate a low frequency sinusoid (< $0.4\,\pi$) and a high frequency sinusoid (> $0.4\,\pi$) and add the two signals. Pass the combined signal through the FIR filters designed above (using `conv()` function in matlab) and plot the result in time domain and frequency domain. Do this for the FIR filters designed above.