San Francisco State University
Engineering 451

Laboratory #5 – FIR Filters I

5.1 Outline

The purpose of this lab is to investigate FIR filtering with windows and apply it to a couple of practical problems.

5.2 Background

In this lab we will create a number realizable FIR filters by multiplying the infinite impulse response of an ideal lowpass filter, $h_{ideal}[n]$, by a finite-length window function, $w[n]$.

Construction of window-based filters

The simplest case is that of the rectangular-window filter. Figure 5-1 shows how we create a rectangular-window filter of length $N = 21$.

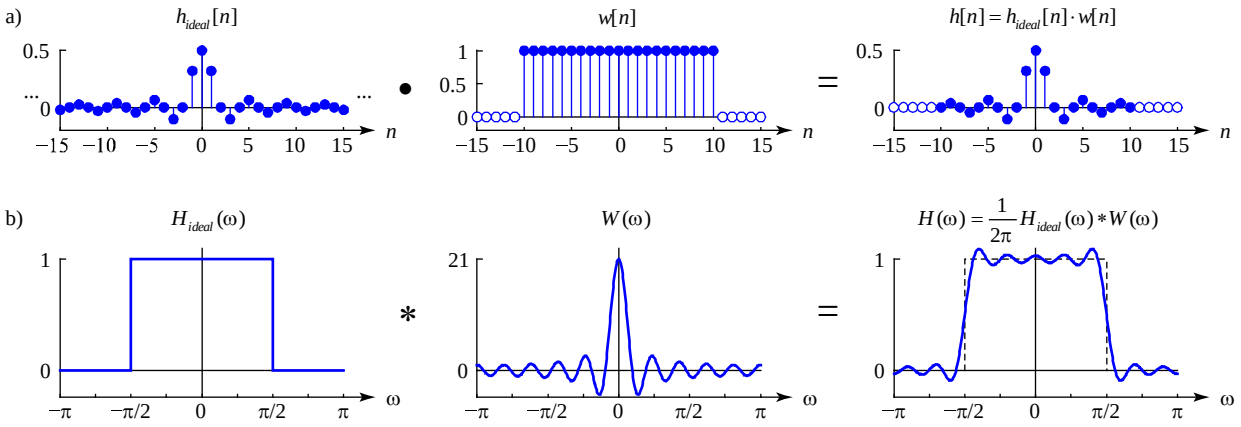


Figure 5-1: Windowing the impulse response of an ideal lowpass filter with a rectangular window

As shown in Figure 5-1a, The ideal impulse response,

$$h_{ideal}[n] = \frac{\omega_c}{\pi} \operatorname{sinc} \omega_c n$$

is multiplied by the finite-length rectangular window, $w[n] = \text{rect}_N[n]$, to produce a finite-length impulse response, $h[n] = h_{ideal}[n] \cdot w[n]$. As shown in Figure 5-1b, in the frequency domain, the frequency response of the filter, $H(\omega)$, is the convolution of $H_{ideal}(\omega)$, the frequency response of the ideal lowpass filter and $W(\omega)$, the frequency response of the rectangular window:

$$H(\omega) = \frac{1}{2\pi} H_{ideal}(\omega) * W(\omega) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_{ideal}(\xi) W(\omega - \xi) d\xi = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} W(\omega - \xi) d\xi ,$$

where

$$H_{ideal}(\omega) = \begin{cases} 1, & |\omega| < \omega_c \\ 0, & |\omega| > \omega_c \end{cases}$$

and for the rectangular window,

$$W(\omega) = \frac{\sin\frac{\omega N}{2}}{\sin\frac{\omega}{2}}.$$

Figures 5-2a and b show the frequency response of the filter on linear and log scales respectively, both on a scale of $0 \le \omega \le \pi$.

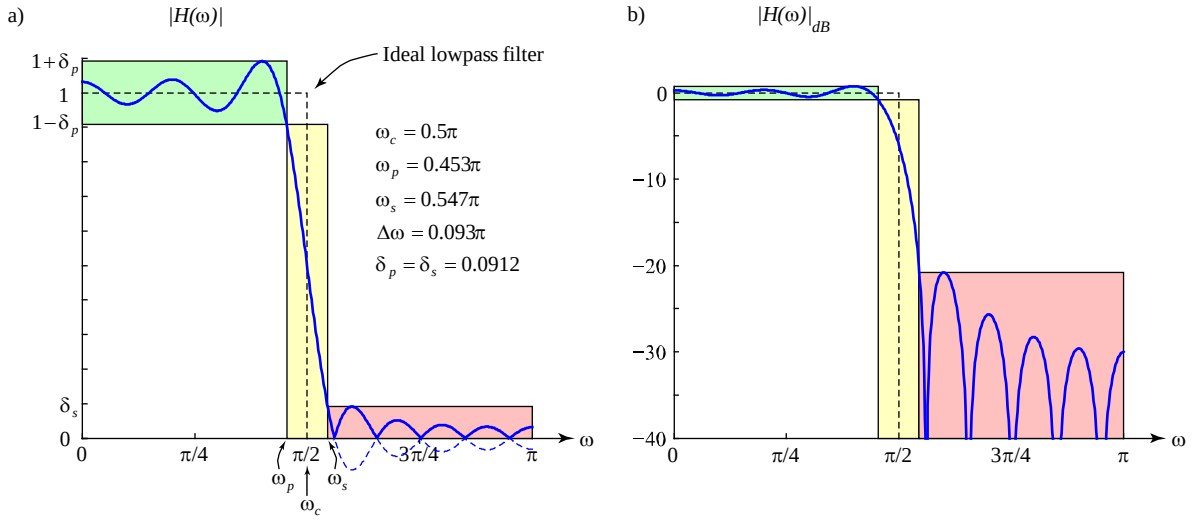


Figure 5-2: Frequency response of magnitude of an FIR filter on linear and log (dB) scales

The passband is shown in green, the stopband in red and the transition zone in yellow. In this lab, you will show that, for the rectangular window filter, the width of the transition zone is governed by the order of the filter, *N*, but that the minimum attenuation of the filter in the stopband, about -20 dB, is an inherent characteristic of the window function, $w[n] = \text{rect}_N[n]$, and doesn't change much as the order of the filter changes.

In this lab, you will be making three types of filters: rectangular-window, Hamming and Kaiser. All filters use the same $h_{ideal}[n]$. They differ only in the window function, $w[n]$, which is used to create the inpulse response, $h[n] = h_{ideal}[n] \cdot w[n]$.

Rectangular window. A (non-causal) causal rectangular-window filter of odd length is defined by

$$w[n] = \text{rect}_N[n] = 1, \qquad |n| < (N-1)/2.$$

Hamming window. The Hamming window is one of a family of raised cosine windows. It has the equation

$$w[n] = \left(0.54 + 0.46\cos\left(\frac{2\pi n}{N-1}n\right)\right)\text{rect}_N(n), \quad |n| \le (N-1)/2.$$

Kaiser window. The Kaiser window allows independent control over the width of the transition band and the magnitude of the pass-band and stop-band ripple. For the Kaiser window, $w[n]$ has the form of a modified Bessel function of the first kind of order zero, $I_o(x)$. The window has two parameters set by the designer, the ripple, δ, and the desired bandwidth of the transition zone, $\Delta\omega$. For this filter, both the pass-band and the stop-band ripple must be made the same, so we set $\delta \triangleq \min(\delta_p, \delta_s)$. For example, if we wanted a filter with a pass-band ripple of $\delta_p = 0.01$ and a stop-band ripple of $\delta_s = 0.0001$, then we would "overdesign" the filter to have $\delta = 0.0001$. Given values of δ and $\Delta\omega$, the Kaiser design equations allow us to compute the two important design parameters for the Kaiser filter: N and β. N is the length of the impulse response of the Kaiser filter. The smaller we wish to make $\Delta\omega$, larger N will be. β is a shape parameter that determines the ratio between the width of the main lobe and the minimum attenuation of the side-lobes.

Here's the procedure to compute β and N. First, the value of δ expressed in dB as the parameter $A = -20\log_{10}\delta$. Then the value of A is used to compute β:

$$\beta = \begin{cases} 0.1102(A-8.7), & A > 50 \\ 0.5842(A-21)^{0.4} + 0.07886(A-21), & 21 \le A \le 50 \\ 0, & A < 21 \end{cases}$$

We choose the filter order, N, to be

$$N = \left\lceil \frac{A-8}{2.285\Delta\omega} \right\rceil,$$

where the notation, $\lceil x \rceil$, means "largest integer not exceeding x", and equivalent to Matlab's `ceil` function. Given N and β we compute the Kaiser window as

$$w[n] = \frac{I_o\left(\beta\left(1-\left(\frac{2n}{N-1}\right)^2\right)^{\frac{1}{2}}\right)}{I_o(\beta)}, \qquad |n| \le \frac{N-1}{2}.$$

Finally, the Kaiser lowpass filter is

$$h[n] = h_{ideal}[n]w[n].$$

Highpass filters

Lowpass filters can also be used to design simple highpass, bandpass and bandstop filters. In this lab, we will need a highpass filter. To understand the procedure, consider an ideal lowpass filter with cutoff frequency, ω_c, defined by

$$H_{LP}(\omega) = \begin{cases} 1, & |\omega| < \omega_c \\ 0, & \omega_c < |\omega| < \pi \end{cases}$$

An ideal highpass filter with cutoff frequency, ω_c, is defined by

$$H_{HP}(\omega) = \begin{cases} 0, & |\omega| < \omega_c \\ 1, & \omega_c < |\omega| < \pi \end{cases}.$$

Hence,

$$H_{HP}(\omega) = 1 - H_{LP}(\omega),$$

which means that

$$h_{HP}[n] = \delta[n] - h_{LP}[n],$$

So, to create a highpass filter with cutoff frequency, ω_c, first create a lowpass prototype filter with the same cutoff frequency and then subtract its impulse response, $h_{LP}[n]$, from $\delta[n]$. Figure 5-3 shows the example of creating a highpass

filter of order, $N = 25$, with a cutoff of $\omega_c = 0.25\pi$ from the appropriate lowpass filter. Figure 5-3a shows the impulse responses and Figure 5-3b the frequency responses of the filter.

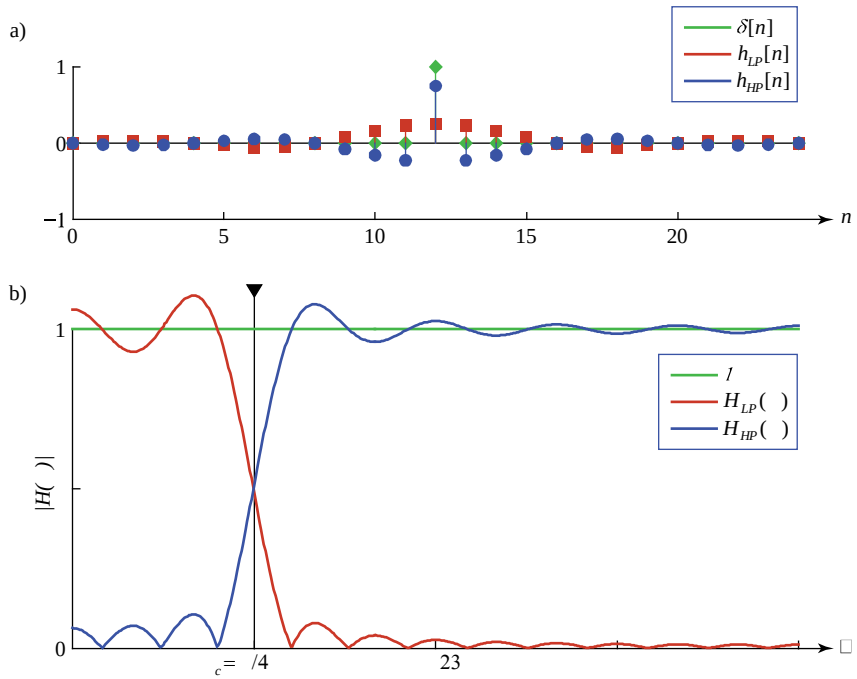


Figure 5-3: Lowpass to highpass transformation

DTMF tones

Inthis lab, we are going to use filters to operate on a sound file. The specific task is to separate the DTMF (Dual-tone Modulation Frequency) tones, that are used in old-school touch-tone telephones.

Anyone who has used one of those way-old-fashioned wired touch tone telephones (how quaint! how 20th century) has heard (Dual-tone Modulation Frequency) DTMF tones. There are twelve keys on the normal phone arranged in a 4x3 matrix. When you press a key, the circuitry in the phone produces a sound that comprises the sum of two pure sine tones of equal amplitude and distinct frequencies. One of the tones – the *row tone* -- has a choice of four frequencies (697, 770, 852 and 941 HZ), each one indicating one of the rows of the keypad. The other tone – the *column tone* -- has a choice of three frequencies (1209, 1336 and 1477 Hz), each one indicating a column of the keypad. You'll notice that the frequencies of the row and column tones are monotonically increasing and also disjoint (that is, all the row tones are of lower frequency than the column tones). The sum of the two tones indicates a row and column in the keypad matrix, and hence identifies a unique key. For example, when you press the '8' key, you get the sum of sin waves at 852 (indicating the third row) and 1336 Hz (indicating the second column). Here is the touch-tone key pad with all the tone frequencies indicated

	1209	1339	1447
697	1	2	3
770	4	5	6
852	7	8	9
941	*	0	#

The task of the circuitry in the phone office is to decode the DTMF tones produced in the phone to determine which keys were pressed. This means we have to analyze the sum of two tones and split it into a column tone and a row tone in order to determine which key was pressed. That's what you are going to do in this lab.

5.3 Assignment

There are several parts to the assignment. First we will create a tool to plot the DTFT on a dB scale. Then we will use this tool to investigate the properties of window-based filters. Finally, we will use the filters to do some useful work.

Matlab function to display magnitude on a dB scale.

The first part of the assignment is to write a Matlab function to display the magnitude (only) of the Fourier transform of a given sequence on a dB scale.

```
function ph = magdb(h)
% MAGDB  Display magnitude of Fourier transform of h[n] on a dB scale.
%           ph = magdb(h)
```

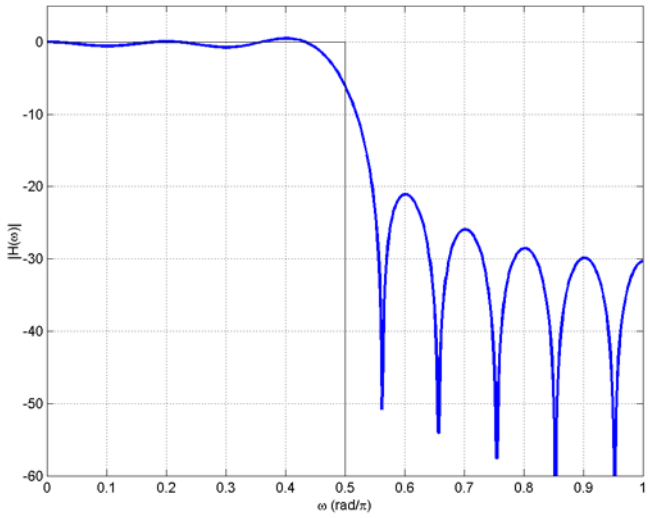
You did something similar in Lab #4, but here you will use the fast Fourier transform (the famous fft), specifically Matlab's `fft` function. For this lab, you don't have to know how the fft works. You only need recognize that

```
H = fft(h, N)
```

creates an output array, `H`, the same size as the input array and is equivalent to taking the DTFT of $h[n]$, sampled at N points, namely,

$$H[k] = H(\omega)\big|_{\omega=\frac{2\pi k}{N}}.$$

You get to choose N, and you obviously you want it to be larger than the length of $h[n]$. Something like 1024 points is a nice number. You'll plot it on a dB scale, so you can clearly see the stopband ripples.



Here's what you would get if you plotted the magnitude of a 21-pt FIR lowpass filter designed using a rectangular window to truncate the impulse response of an ideal LPF with a cutoff frequency of $\omega_c = 0.5\pi$. The Matlab command would be

```
magdb(fir1(20, 0.5, rectwin(21))
```

Note that I've also superimposed a grey line on the plot to indicate the frequency response of the ideal lowpass filter. Also, I used the `LineWidth` and `Color` options in the plot command to give me a thick, blue line. You will also want to add a way so you can change the color of your plot, so you can eventually display several plots on the same axis, each with a different color and line width. One way would be to add input arguments to the function call, for example

```
function magdb(h, color, linewidth)
```

But that's a pretty crummy idea. A better way would be to have the function return a handle to the plot, which you can then manipulate. Here's the outline of the function:

```
function ph = magdb(h)
% MAGDB  Display magnitude of Fourier transform of h[n] on a dB scale.
%           ph = magdb(h)

... bunch of code here resulting in array H,
which is the magnitude of Fourier transform of h[n] ...

ph = plot(...);

return
```

The `magdb` function returns a handle to the plot, `ph`, which allows you to manipulate the plot afterwards. For example, to change the line width to 3 and set the color to red, you could do the following:

```
>> ph = magdb(h);
>> set(ph, 'LineWidth', 3, 'Color', 'r');
```

When you write your function, please make sure that it doesn't call `figure`, `cla`, or `clf`.

Unnecessary Matlab Nerdism

If you call your function without a terminal semicolon and without assigning it to an output variable, it will always print the handle:

```
>>magdb(h)
>>ans =
     99.0009
```

Matlab allows a called function to sense both the number of input arguments and the number of output arguments with which the function was called, and also allows there to be a variable number of input or output arguments. We can use this functionality to make `magdb` only pass the handle to the base workspace if the function was called with an output argument. We use some special Matlab reserved key words, `varargout` and `nargout`. Here's how:

```
function varargout = magdb(h)
% MAGDB  Display magnitude of Fourier transform of h[n] on a dB scale.
%           varargout = magdb(h)

...

ph = plot(...);

if (nargout)
   varargout = {ph}; % note that these are cell brackets, not parens
```

```
end

return
```

For more information look at the Matlab help files for `varargout` and `nargout`.

Investigate filter behavior.

Having written your function to display the magnitude of the frequency response, you'll now use this to investigate the behavior of various FIR filters designed using truncation of the impulse response of the ideal LPF with windows of varying shapes. You will write functions that create sequences corresponding to FIR filters designed with three different windows: Rectangular, Hamming and Kaiser. In writing these functions, you may not use any of the commands that MATLAB provides in the signal toolbox. That means you can't use `fir1` or `window` or anything else. You may use things like `sinc` and `besseli` where appropriate. Of course, you can and should check that your functions perform the same as Matlab's equivalent filter design functions.

Rectangular window. Write a function `rectfilt` that creates an FIR filter of order, *N*, and cutoff frequency ω_c based on a rectangular window:

```
function h = rectfilt(N, wc)
% RECTFILT  Creates Rectangular filter.
%           h = rectfilt(N, wc)
%           returns filter of order N and cutoff wc
%           based on rectangular window. wc is in fractions
%           of pi.
```

Hamming window. Write a function `hammingfilt` that creates an FIR filter of order, *N*, and cutoff frequency ω_c based on a hamming window:

```
function h = hammingfilt(N, wc)
% HAMMINGFILT  Creates Hamming FIR filter.
%           h = hammingfilt(N, wc)
%           returns filter of order N and cutoff wc
%           based on hamming window. wc is in fractions
%           of pi.
```

Kaiser window. See my notes for the details of the design procedure for Kaiser windows. You will write two functions to design the Kaiser filter.

The first function, `kaiserparams`, will take as input the desired values of the width of the transition band, $\Delta\omega$, and the amplitude of the passband or stopband ripple, δ, and output the order of the Kaiser filter, *N*, and the Kaiser window shape parameter, β.

```
function [N, beta] = kaiserparams(deltaOmega, delta)
% KAISERPARAMS  Returns Kaiser design parameters N and beta.
%           [N, beta] = kaiserparams(deltaOmega, delta)
%           returns Kaiser design parameters N and beta,
%           given width of transition band, deltaOmega, and
%           amplitude of the stop/passband, delta.
%
%           Note: dOmega is given as a fraction of pi.
%           Test: [37, 5.6533] = kaiserparams(0.2, 0.001)
```

The second function, `kaiserfilt`, creates an FIR filter of cutoff frequency ω_c based on a Kaiser window with a given N and β.

```
function h = kaiserfilt(N, wc, beta)
% KAISERFILT  Creates FIR Kaiser filter.
%           h = kaiserfilt(N, wc, beta) creates Kaiser filter
%           h, of order N, cutoff wc
%           and shape parameter beta. wc is in fractions of pi
```

Matlab has several functions that you can use to design FIR filters using a variety of windows. I will be checking your filter design routines against Matlab's `fir1` routine. This has syntax

```
h = fir1(n - 1, wc, window(n));
```

where `n` is the order of the filter, `wc` is the cutoff frequency and `window` denotes the window type. For example, to create a 11th order rectangular window with a cutoff of 0.2π, we would call

```
>> n = 11;
>> wc = 0.2;
>> h = fir1(n-1, wc, rectwin(n));
```

To create a filter based on a Hamming window, use

```
>> h = fir1(n-1, wc, hamming(n));
```

For a Kaiser window, use

```
>> h = fir1(n-1, wc, kaiser(n, beta));
```

Note the typical Matlab bogosity that the first argument is n-1 but the argument to the window function is n. Go figure.

IMPORTANT NOTE:
By default, Matlab's `fir1` function normalizes the impulse response of the lowpass filter so that the magnitude of the filter's response at. $\omega = 0$ .is 0 dB; that is, $H(0) = 1$. From the definition of the DTFT, this means that $h[n]$ must be normalized such that

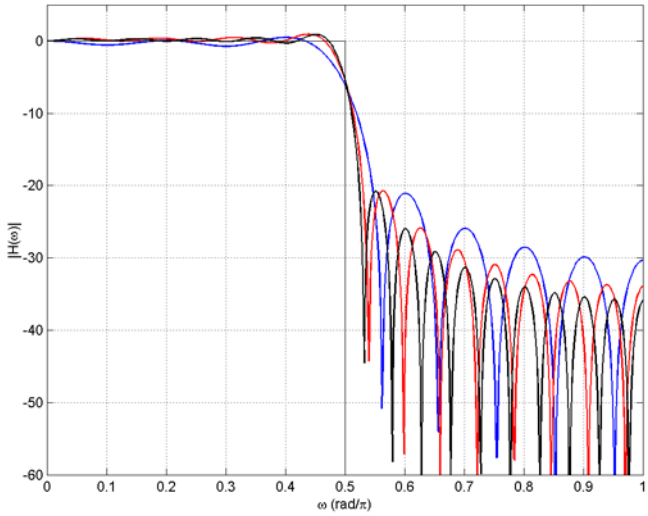
$$H(0) = H(\omega)\big|_{\omega=0} = \left(\sum_{n=-\infty}^{\infty} h[n] e^{-j\omega n} \right)\Bigg|_{\omega=0} = \sum_{n=-\infty}^{\infty} h[n] = 1.$$

You should do the same thing so your filter's response will match Matlab's. Also, when I test your filters, I will only try to create filters of odd order (i.e. N odd).

Comparative behavior of window filters

Having written these functions, we will now investigate their behavior.

Rectangular-window filters
Effect of filter order. Plot the frequency response of a rectangular window designed with the same value of cutoff frequency, $\omega_c = 0.5\pi$, but with values of $N = 21$ (blue trace), 31 (red) and 41 (black). You should get a picture like this:



If you have to manually change the vertical scale, you can use Matlab's `ylim` command. To change the horizontal scale, you can use Matlab's `xlim` command. To change both horizontal and vertical scales together, you can use Matlab's `axis` command. Note that you can change the axes limits even after the plot has been made. Just click on the figure of interest and enter the `ylim`, `xlim` or `axis` command.

These rectangular-window filters don't do a very good job of filtering. What do you conclude about the effect of increasing the filter order on the stopband attenuation (i.e. magnitude)?

Effect of ω_c. Produce a plot of the frequency response of a rectangular window designed with the same value of $N = 31$, but cutoff frequency, $\omega_c = 0.2\pi$ (blue), 0.4π (red) and 0.6π (black). Does the stopband magnitude or the width of the transition band depend significantly on ω_c ? Why or why not?

Hamming-window filters.

Effect of filter order. Repeat the same experiment as in the preceding section. What do you conclude about the ability to change the stopband attenuation by changing filter order?

Comparison with Rectangular filter. Plot the frequency response of a Hamming window filter (blue) and a Rectangular window filter (red) both of order, $N = 21$, and $\omega_c = 0.5\pi$. Compare the frequency response of these filters; specifically, which has the sharper (smaller) transition band and which has the larger stopband attenuation?

Kaiser filter.

Effect of filter order. Repeat the same experiment as in the preceding section with a Kaiser filter with $\omega_c = 0.5\pi$, $\beta = 5$ and N of 21 (blue trace), 31 (red) and 41 (black).

Effect of β. Produce plots of the filter's response for $\omega_c = 0.5\pi$, and $N = 21$ and β of 2 (blue trace), 4 (red) and 6 (black).

Comment on the effects of changing N and β. Do these parameters change $\Delta\omega$ and δ independently? Do you care as long as you have design formulae that allow you to get the values of $\Delta\omega$ and δ you desire?

Highpass filters.

Once you know how to design a lowpass filter, designing a highpass filter is (relatively) easy.

Please design and plot the Kaiser highpass filter with a $\omega_c = 0.4\pi$, $\Delta\omega = 0.2\pi$ and $\delta = 0.001$. You may want to play with filtering some sounds or music to hear the effects of your filters. All you have to do is convolve the input sound with your filter's impulse response using Matlab's conv .

Phone tones

Finally, we'll design a filter to do a particular task. The task is to separate DTMF (Dual-tone Modulation Frequency) tones.

In the final project in this course, we will decode a random sequence of digits from a .wav file recorded from tones keyed in by a phone. However, in this part of our current lab assignment, we'll just introduce the problem and play with trying to separate the tones using – you guessed it – FIR filters designed using windows. This isn't necessarily the best approach to this problem.

The only thing we'll do in this lab is to use a lowpass and highpass filter to separate the row tones from the column tones. Because the frequencies of the row and column tones are disjoint (that is, all the row tones are of lower frequency than the column tones), this is an easy task. The only thing you have to think about is what you want the parameters of your Kaiser filter to be, specifically, ω_c, $\Delta\omega$ and δ. If you look closely at the matrix of tones, you'll see that the '*' key is the most challenging case. If you get the separation working for this key, then it will work for them all. (Why is this?).

In this part, you will be designing a discrete-time filter with a cutoff frequency ω_c to filter analog data that has been sampled at a rate f_s samples/sec. Hence, this filter will be acting like an effective analog filter with a cutoff frequency of f_c. In order to design the discrete-time filter, you need to know the relation between f_c to ω_c:

$$\omega_c = 2\pi \frac{f_c}{f_s}.$$

Write a function separate that inputs a sound file of DTMF tones, `s`, and outputs two sound files, `sr` and `sc`, which correspond to the row and column tones. The design goal is to achieve at least 40 dB of separation between row and column tones. That is, at any time the column tone should contain no more 1% of the energy of a row tone, and vice versa. You'll probably want to derive two Kaiser filters, one highpass and one lowpass. They may have quite a high order, but that's O.K. The only thing you have to watch out for is that the order of the Kaiser filters should be odd. So, if you're designing a Kaiser highpass filter to match specified values of $\Delta\omega$ and δ, and the value you get from the Kaiser design formulae is $N = 21.2$, you need to round this up to 23 instead of 22.

Here's an outline of the task:

```
function [sr, sc] = separate(s, fs)
% SEPARATE  Separates row and column tone for a DTMF tones
%           [sr, sc] = separate(s, fs)
%           produces row and column sound files sr and sc
%           for a sound file, s, of DTMF tones that have been
%           digitized at sample rate, fs.

%% Part 1 Make filters
%  Create lowpass FIR Kaiser filter of appropriate order using
%  your kaiserparams and kaiserfilt functions

%  Create highpass FIR Kaiser filter of appropriate order

%  Perform filtering

return
```

When you've finished, download the .wav file of phone tones and listen to the sound files you've generated to see if you've succeeded.

Download the following files: lab5.m, test_lab5a.p, test_lab5b.p and test_lab5c.p and put them in the same directory as all your functions. Then publish lab5. Submit the output along with a printout of your functions

---