Lecture 6, Multirate Signal Processing, Windows, Sampling

How do we design Window functions to obtain higher stopband attenuation? There are 2 versions, for windows of **even** and **odd length**. We now analyse a few common window types.

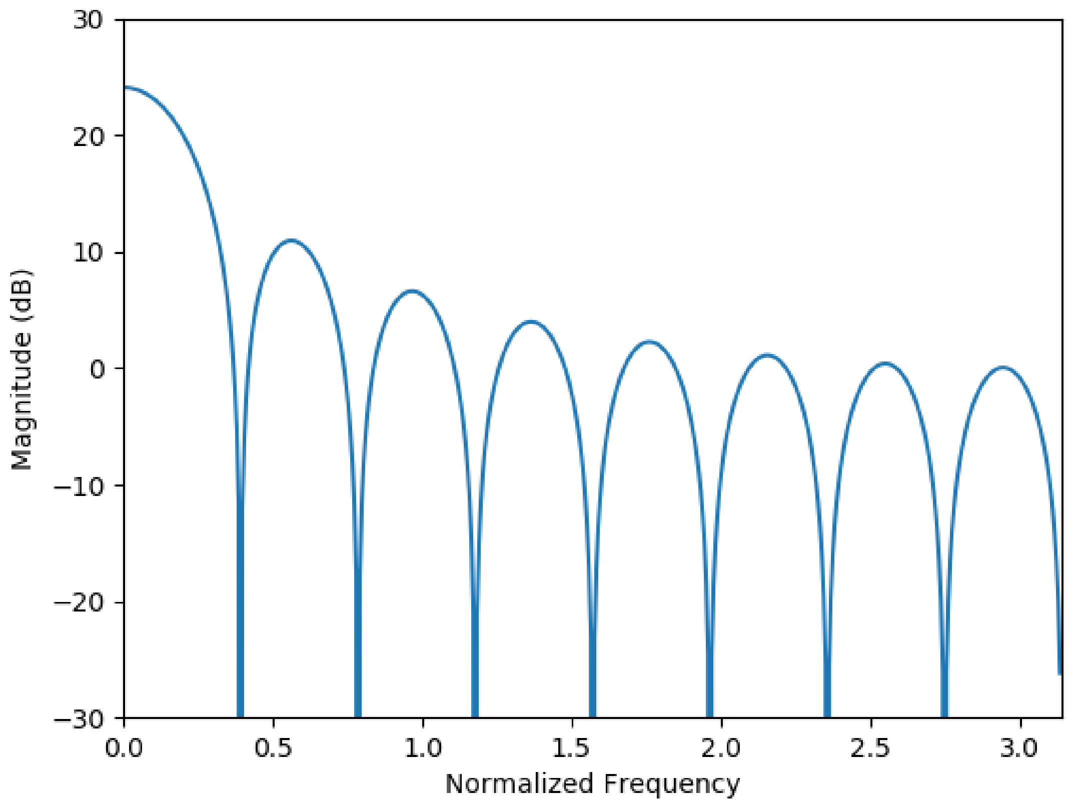
Lets start again with the **rectangular window**.

$$h(n)=1$$

for n=0,...,L-1.

In ipython we get its frequency response for L=16 :

```
ipython -pylab
h=ones(16);
import scipy.signal as signal
omega, H =signal.freqz(h)
plot(omega, 20*log10(abs(H)))
axis([0, 3.14, -30, 30])
xlabel('Normalized Frequency')
ylabel('Magnitude (dB)')
```



Observe: Its main lobe has a 3dB width of about $0.05\pi \approx 0.16$, the side lobe attenuation is about -15 to -25 dB.
We can design different windows, which de-emphasize a transition region from passband to stopband, and emphasize the stopband attenuation more than the passband attenuation.
This can be seen as minimizing a **weighted** squared error function, where the parts the we want to emphasize get a higher weight.

In Python we can formulate an error function with a weighted squared error as follows, with pb, tb: number of frequency samples in the passband and transitionband, respectively,

```
def errfunc(h):
   numfreqsamples=512
   #desired passband:
   pb=int(numfreqsamples/4.0)
   tb=int(numfreqsamples/8.0)
   w, H = signal.freqz(h,1,numfreqsamples)
   H_desired=np.concatenate((np.ones(pb),\
   np.zeros(numfreqsamples-pb)))
   weights = np.concatenate((np.ones(pb), \
   np.zeros(tb),
   1000*np.ones(numfreqsamples-pb-tb)))
   err = np.sum(np.abs(H-H_desired)*weights)
   return err
```

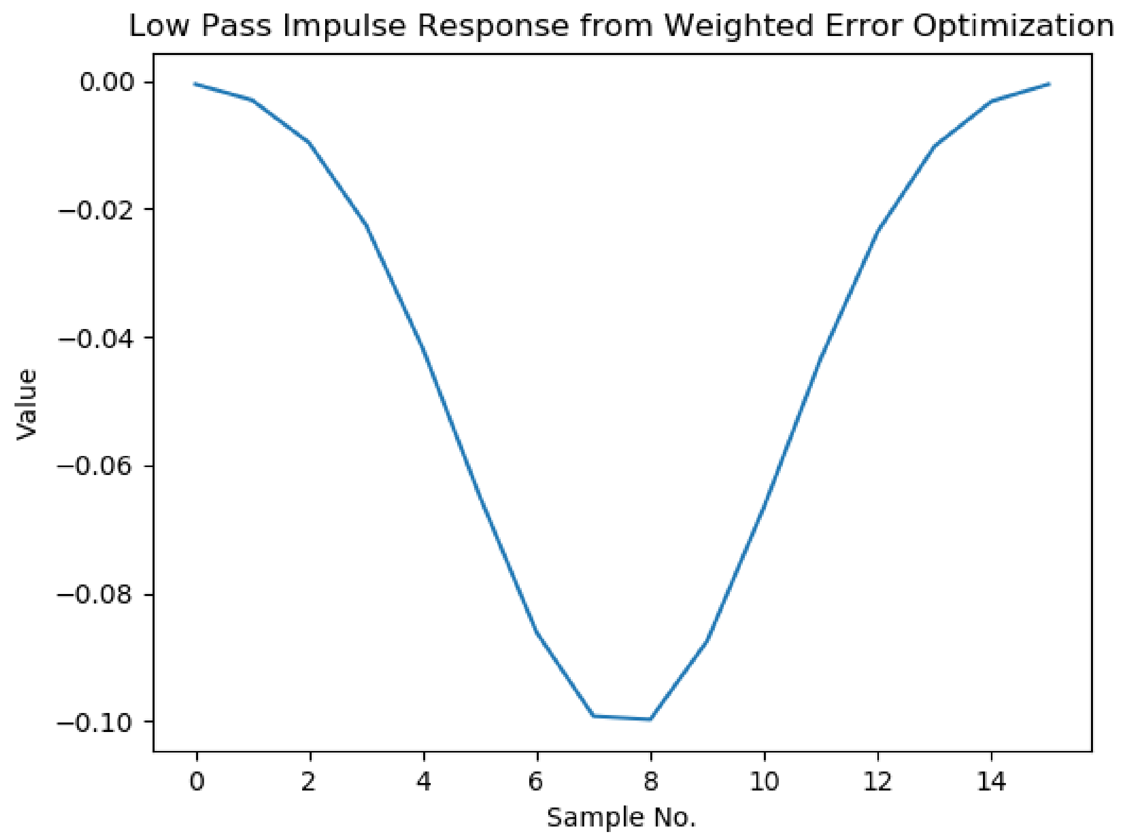
We can then apply optimization to obtain the window or filter samples which minimize this error, for instance using “scipy.optimize”. In this example a window or filter h (depending on how it is used) of length 16 samples or taps is obtained with

```
minout=opt.minimize(errfunc,np.random.rand(16))
h=minout.x
```

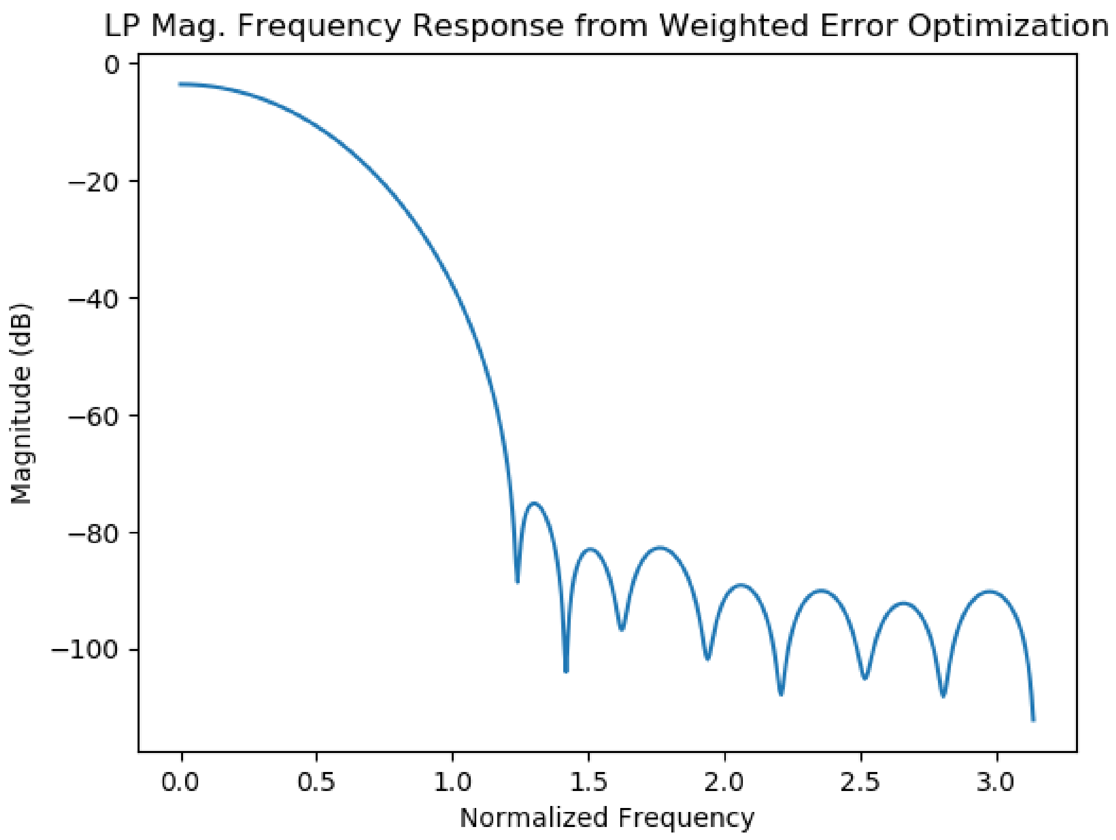
Let it run in a terminal shell with:

```
python errfunc.py
```

We get the samples or impulse response as



observe that the negative sign doesn't matter, because we only optimized for the magnitude. It magnitude of the frequency response is,



Observe the decently high stopband attenuation of about -80 dB!

Usually this optimization gives the best answer for most applications. But there are also more **“pre-fabricated” windows** for filter design, with different trade-offs of transition band width and stopband attenuation, for convenience.
The first is the **raised cosine window**, also known as **Hann- or Hanning Window**:

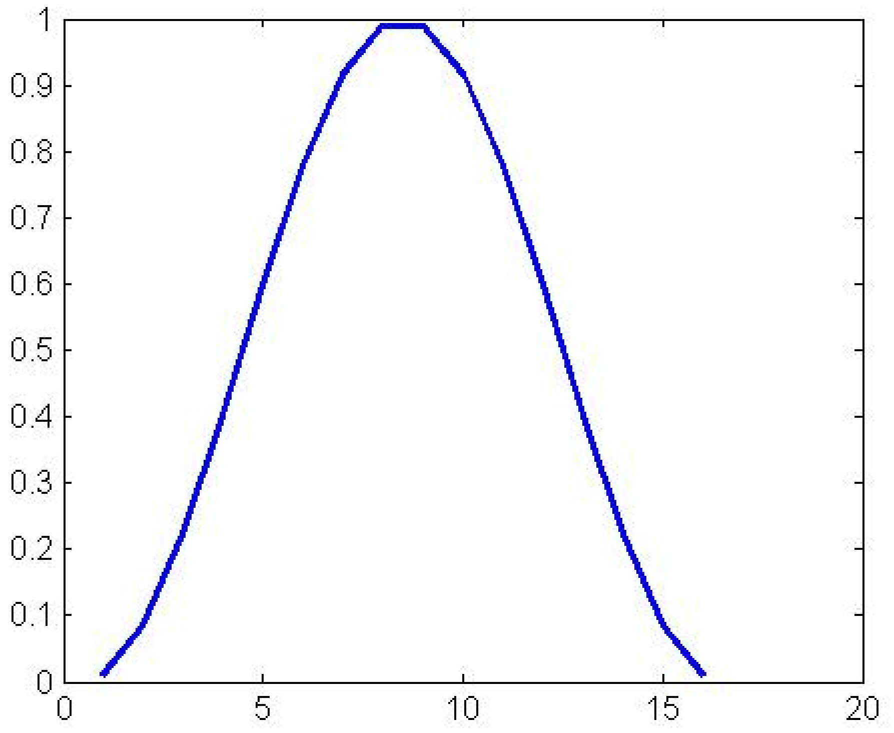
$$h(n)=0.5-0.5\cos\left(\frac{2\pi}{L}(n+0.5)\right) \quad ,$$

with n=0,…,L-1, for even window lengths.

iPython example of a raised cosine with even window length and L=16: We obtain its plot with with

```
ipython -pylab
h=0.5-0.5*cos(2*pi/16*(arange(16)+0.5))
plot(h)
```

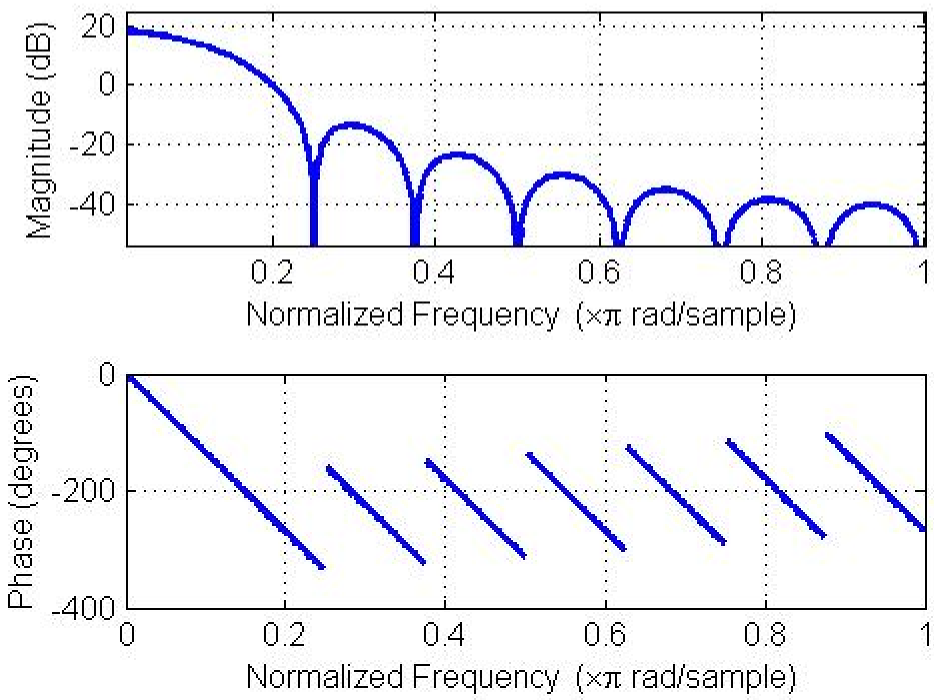
Observe that the center is between 2 samples!



Its frequency response is obtained with freqz. For convenience we define a function “freqz_plot.py” (in Moodle), which plots the magnitude and phase of the frequency response.

```
from freqz_plot import *
```

```
clf() #clear figure
freqz_plot(h)
```



Here we can see that we obtain a much higher attenuation for the first side lobe, at over -35 dB (measured from the maximum of the main lobe, the pass band), and far off in the stop band we get about -60 dB attenuation! But at the cost of a wider main lobe (its 3dB width is about 0.1π, twice as wide as for the rectangular window), which leads to a wider transition band.

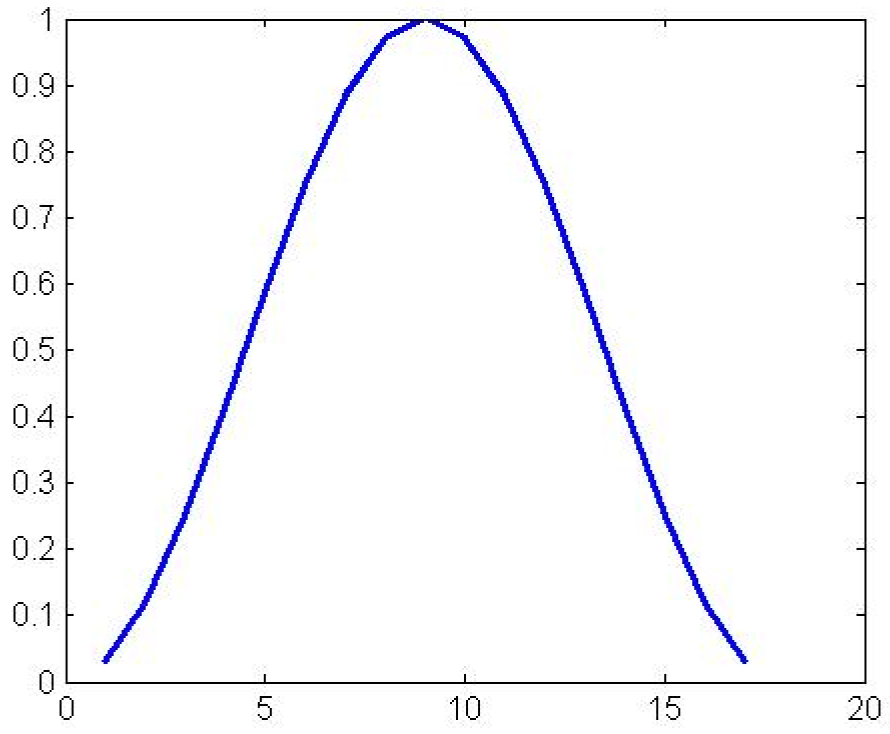
This shows a **general trade-off**: We can **trade transition width** for **stop-band attenuation**.

For odd window lengths we get:

$$h(n)=0.5-0.5\cos\left(\frac{2\pi}{L+1}(n+1)\right) ,$$

with n=0,…,L-1.

Example for L=17 (number of samples of the impulse response, equal to the coefficients, since this represents an FIR filter):
*h=0.5-0.5*cos(2*pi/18*(arange(17)+1))*
plot(h)
Observe that here the center is right on one sample!



For the **Sine window** we get

$$h(n)=\sin\left(\frac{\pi}{L}(n+0.5)\right) \quad ,$$

with n=0,…,L-1 for even window lengths L, or

$$h(n)=\sin\left(\frac{\pi}{L+1}(n+1)\right) \quad ,$$

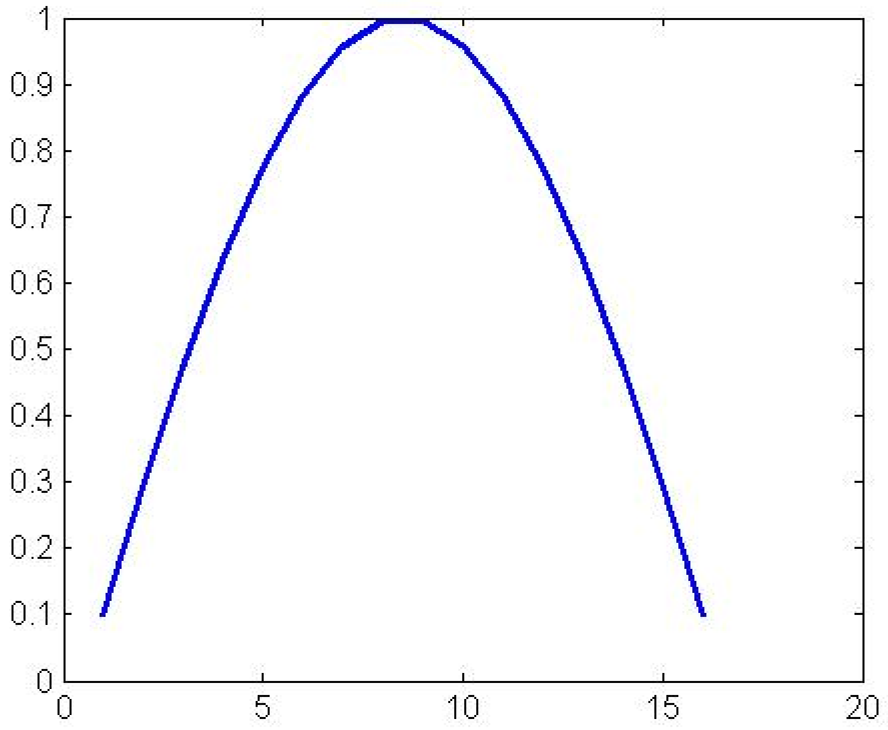
with n=0,…,L-1 for odd window length L.

Example for L=16:

s=sin(pi/16(arange(16)+0.5))*

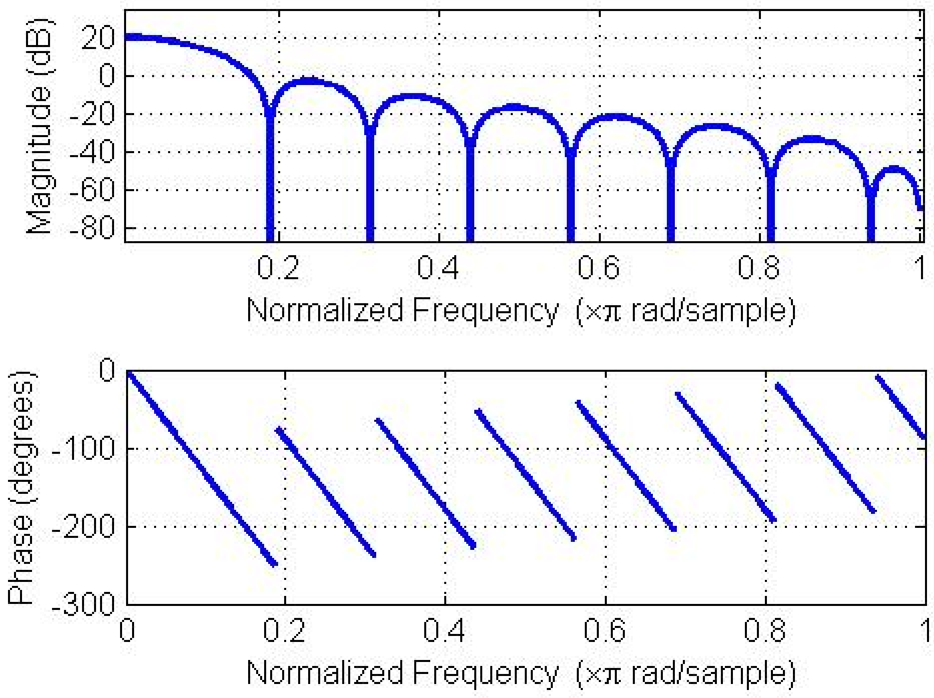
clf()

plot(s)



Its frequency response is

```
clf()
freqz_plot(s)
```



Here we can see that the **main lobe is somewhat narrower** than for the raised cosine window, with a 3dB width of about 0.04π, but the first side lobe has only about -20 dB attenuation. But the further side lobes increase in attenuation. This **attenuation** is more than for the rectangular window, but **less than for the raised cosine window**. On the other hand, its transition bandwidth is less than for the raised cosine window.

Observe that this always results in positive values for the window functions, and that they are perfectly symmetric (for odd lengths there is a sample right at the center, for even length windows the center is right between 2 center samples).

Well known is also the so-called **Kaiser Window:**

$$h(n)=0.5\, I_0\left[\beta\sqrt{1-(\frac{2n}{L})^2}\right]/I_0(\beta)\quad \text{, for}\quad |n|\leq L/2$$

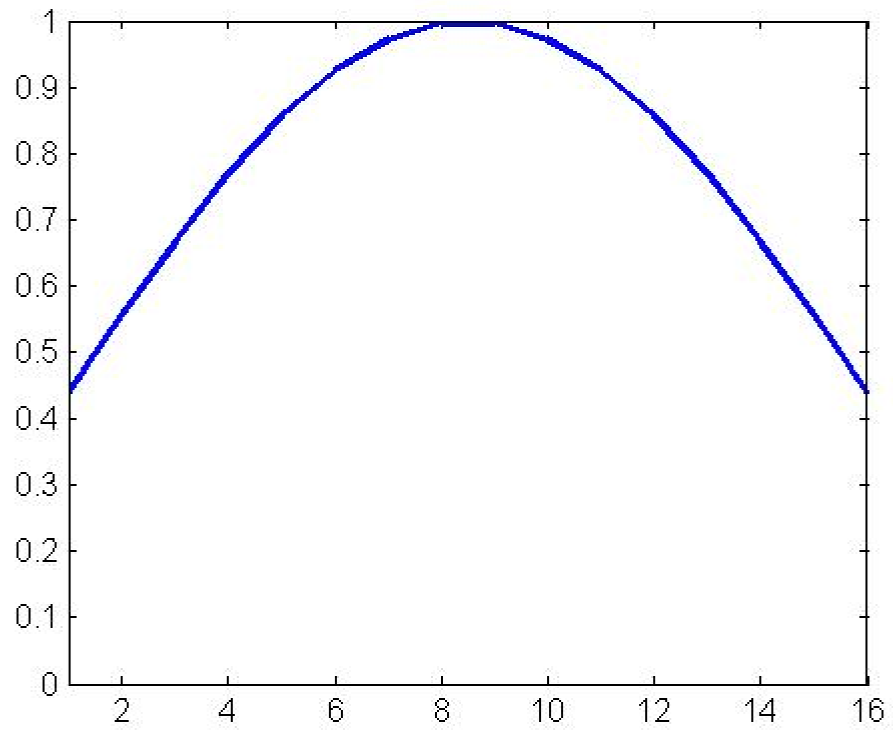
(a non-causal representation) where the Bessel function is used,

$$I_0(x)=1+\sum_{k=1}^{\infty}\left[\frac{(0.5\text{x})^2 k}{k!}\right]^2$$

where in practical designs often the first 20 terms are used (Strang, Nguyen, “Wavelets and Filter Banks”). The parameter β is used to **trade off** the **transition bandwidth of the filter and its stopband attenuation**.

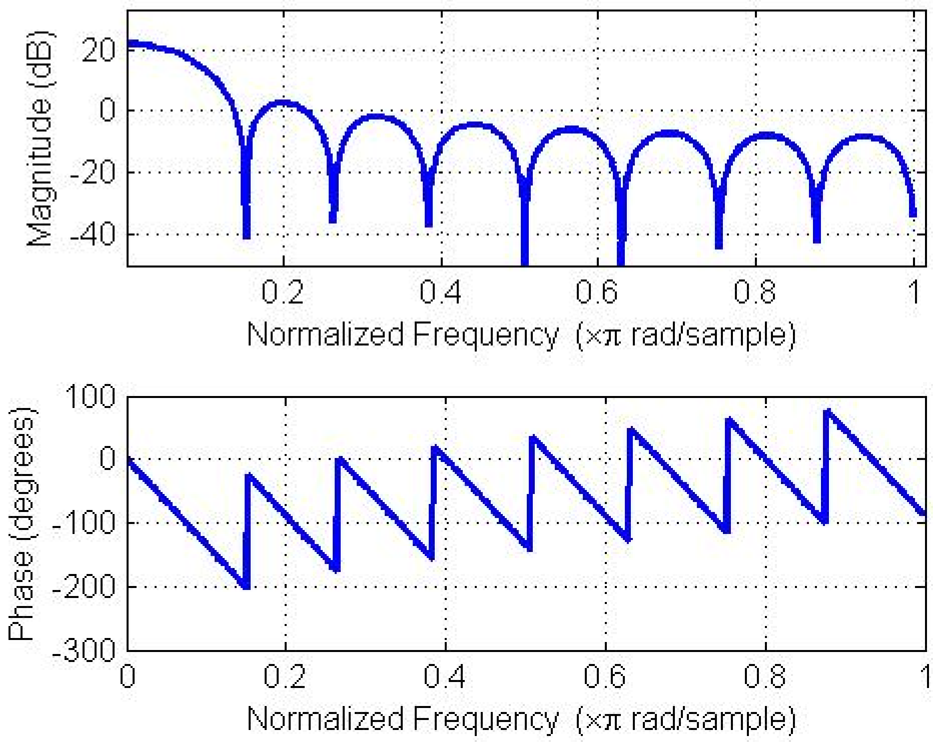
Example for L=16 and $\beta=2$, with iPython:

hk=kaiser(16,2)
clf()
plot(hk)



We obtain its frequency response with

```
clf()
freqz_plot(hk)
```



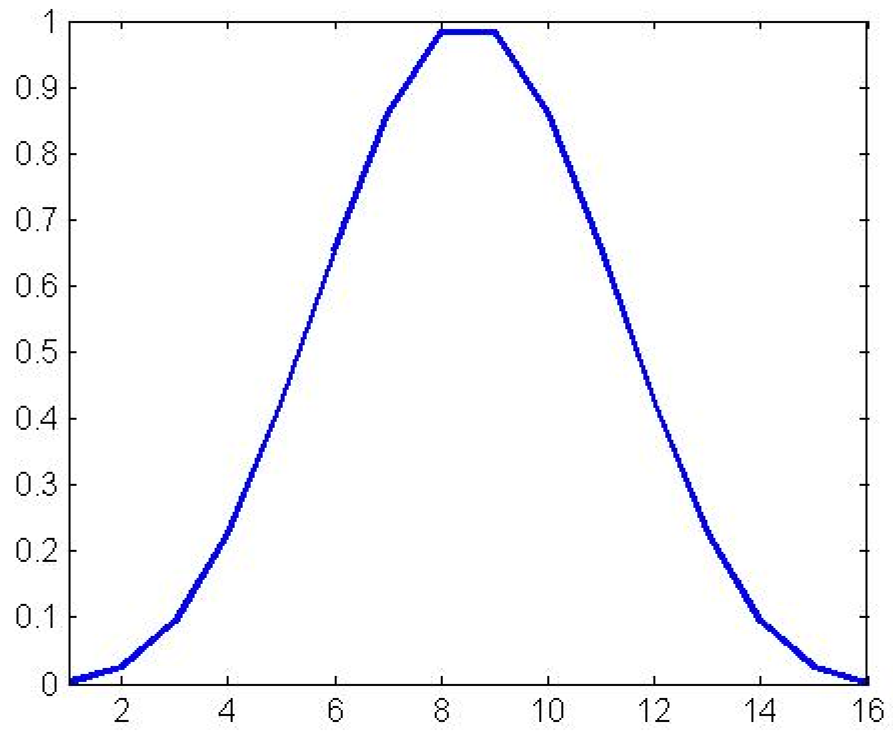
Here we can see that the main lobe (pass band and transition band), for this β , is even narrower than for the sine window, the first side lobe also has about -20 dB attenuation, but the further side lobes don't have much increasing attenuation.

For $\beta=8$ we obtain:

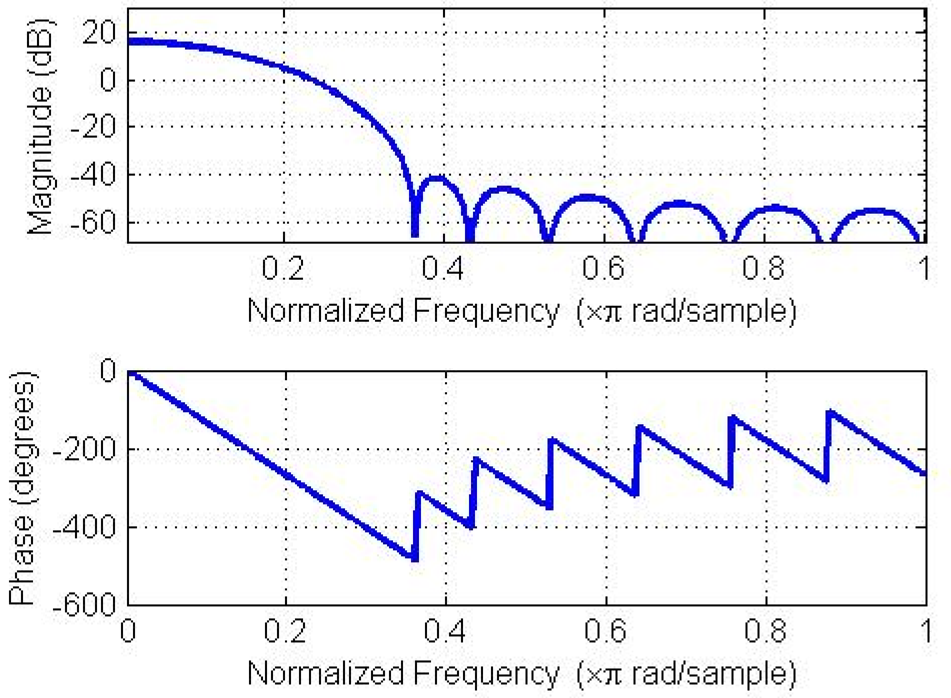
hk=kaiser(16,8)

clf()

plot(hk)



and for the frequency response we get
freqz_plot(hk)



Here we can now see the other extreme, with a very wide main lobe (pass and transition band), but a first side lobe with already about -60 dB attenuation, which would fulfill our requirements!

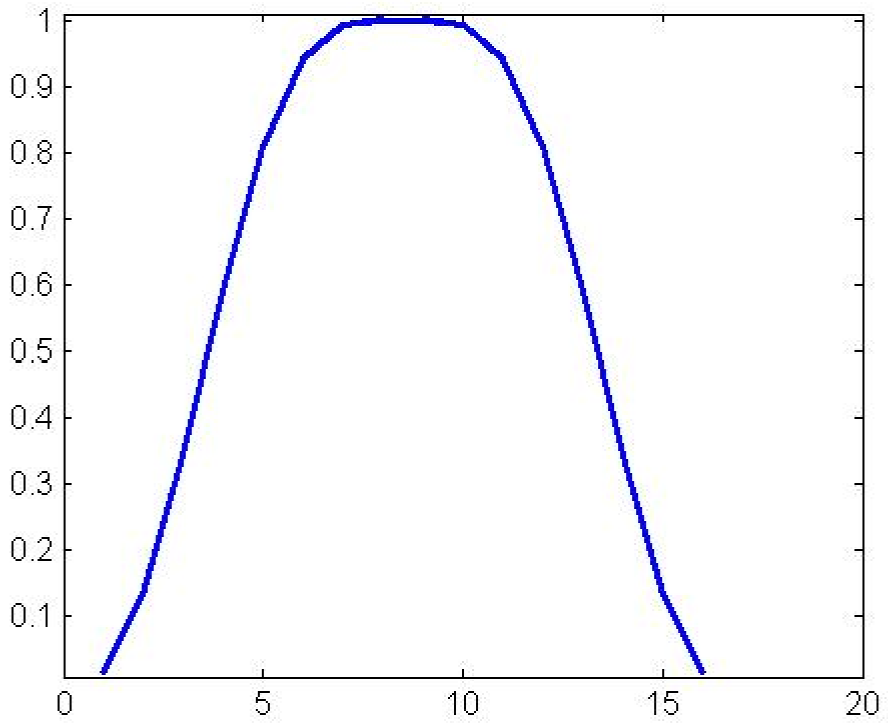
An interesting window is also

$$h(n)=\sin\left(\frac{\pi}{2}\sin\left(\frac{\pi}{L}(n+0.5)\right)^2\right) \quad \text{, n=0,…,L-1}$$

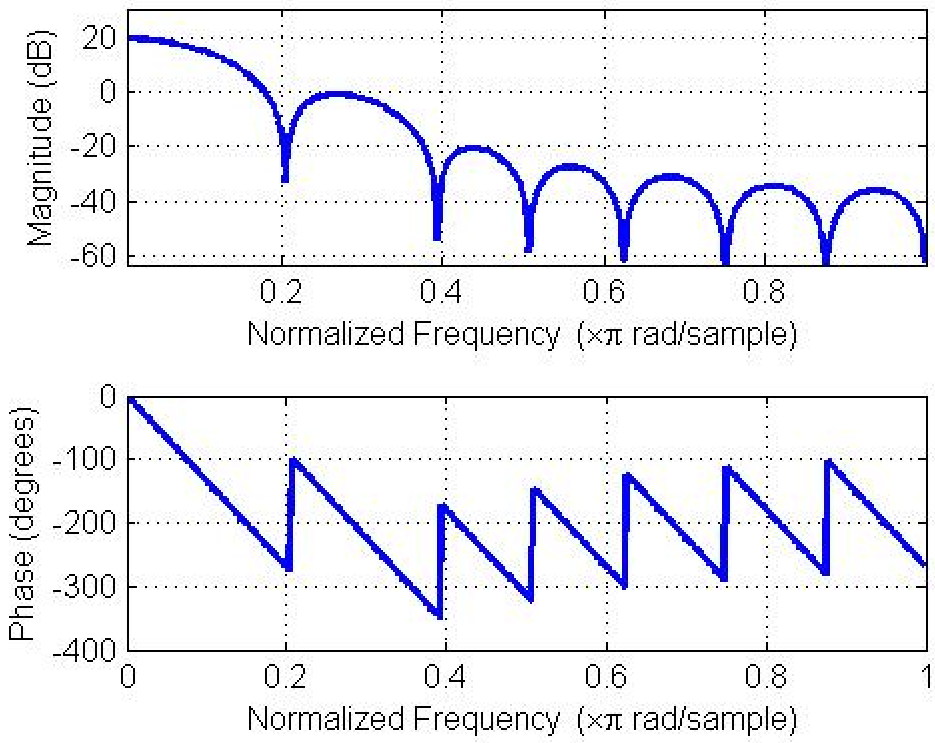
(from:http://xiph.org/vorbis/doc/Vorbis_I_spec.pdf).

For L=16 we get:

```
sq=sin(pi/2*sin(pi/16*(arange(16)+0.5)).^2)
clf()
plot(sq)
```



Its frequency response is:
freqz(sq)



Here we can see that the main lobe is wider than for the sine window, the first side lobe also has about -20 dB stop band attenuation, but the further side lobes increase their attenuation more than for the sine window.

It results in a good stopband attenuation, but is lacking the additional parameter we saw with the Kaiser window.

The MPEG AAC audio coder also uses an optimized window function, the so-called Kaiser-Bessel Derived (**KBD**) window, which results from **numerical optimization**.

Conclusion: Filter Design with the Window Method

We start with the **ideal filter** (sinc), which is infinitely long. To make it causal and to obtain a desired trade-off between the

transition band width and the stopband attenuation, we multiply it with a finite length window. This window is either obtained by optimization, or is chosen from one of the “pre-fabtricated” ones. Longer filters also lead to narrower transition bands.
The resulting frequency response after multiplying the ideal impulse response (sinc) with the window function is then the convolution of the ideal frequency response and the window frequency response. The resulting passband width is the **ideal passband width plus the passband width of the window**. The resulting **stopband starts** at the stopband frequency of the ideal frequency response **(cutoff frequency) plus the frequency of the start of the stopband of the window function**.
To obtain a given passband or stopband, this has to be taken into account, and the cutoff frequency has to be modified accordingly.

Example:
We saw that the **Kaiser Window** at least fulfils the **requirement** for the attenuation of our downsampling application example. How do we get the correct start of the **stopband** (for attenuating the aliasing for downsampling sufficiently) using our filter design method?
Our stop band should start at 0.5 for a downsampling factor of N=2. Looking at the Kaiser window with $\beta=8$ we see that we get -60 dB at a normalized frequency of about 0.36. Hence our ideal filter needs to have the **end of its pass band** at 0.5-0.36=0.14 (remember, here 1 was the Nyquist

frequency), hence $\omega_s = \omega_c = 0.14 \cdot \pi$. Observe: here we need the multiplication with pi since for our formula pi is the Nyquist frequency.
Now we just need to plug this into our formula for the ideal filter (the sinc function), with L=16,

$$h(n) = \frac{\sin(\omega_c(n-7.5))}{\omega_c(n-7.5)} \text{ for } n=0,\ldots,15 \ ,$$

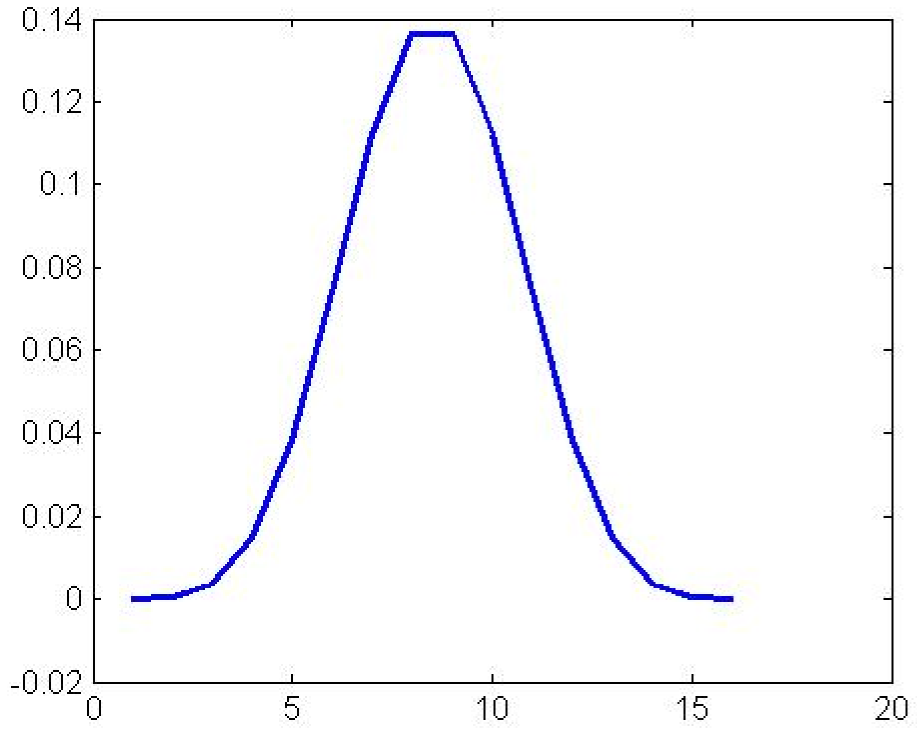
or, with a different normalization to let the passband start at 0 dB:

$$h(n) = \frac{\sin(\omega_c(n-7.5))}{\pi(n-7.5)}$$

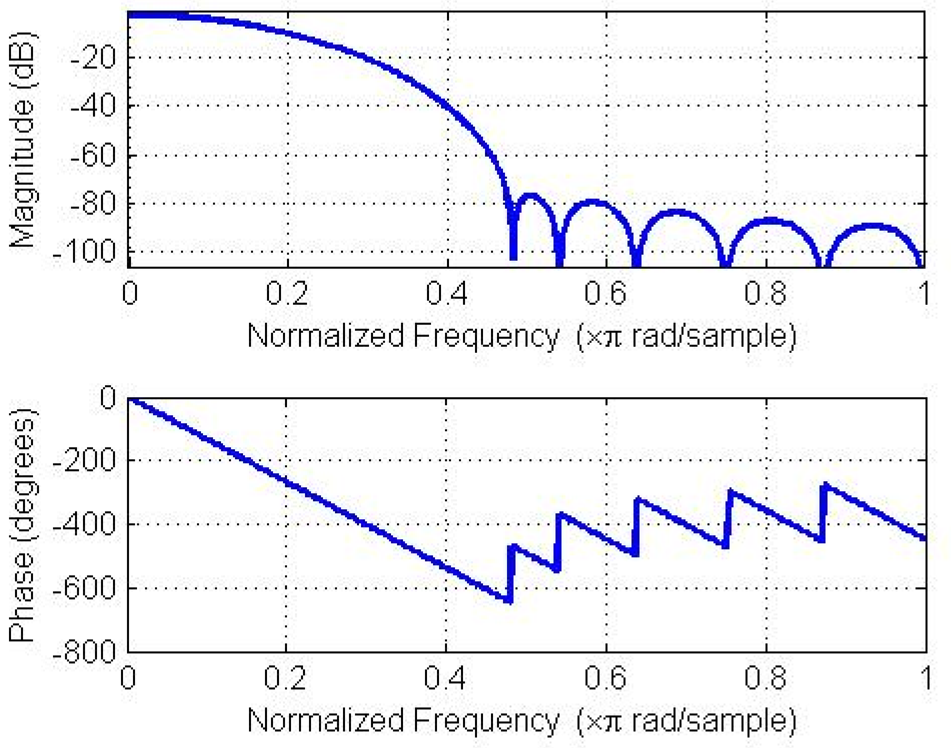
(with $\omega_c = 0.14 \cdot \pi$) and multiply it with our Kaiser window.

In iPython:

```
n=arange(16);
h=sin(0.14*pi*(n-7.5))/(pi*(n-7.5));
hk=kaiser(16,8);
hfilt=hk*h;
plot(hfilt)
```

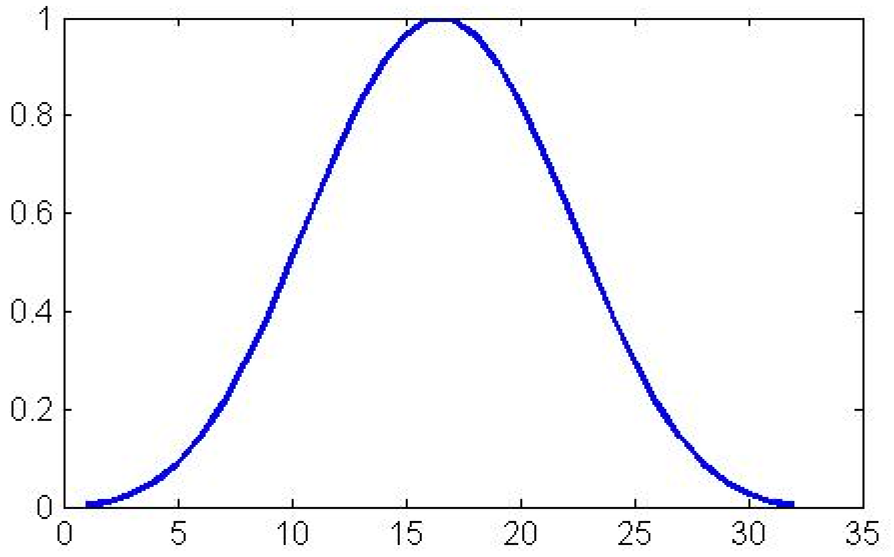


its frequency response:
`freqz_plot(hfilt)`

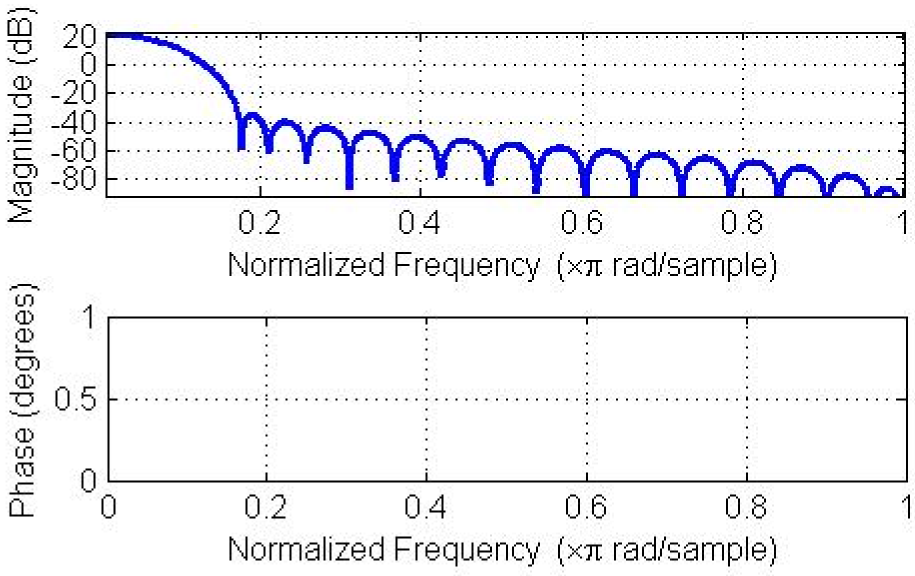


Here we see that at normalized frequency 0.5 it has indeed enough attenuation, at about -80 dB! But the pass band (up to about -6dB, upper plot) is only up to about 0.15, which is usually not enough! We didn't really specify it, but for practical purposes this would usually not work.

So how can we improve the **pass band** now? Since we already tried different compromises for the width of the transition band and the stop band attenuation, we can now try increasing the filter length. Try L=32 (instead of 16). The following is the **Kaiser window** for $\beta=8$ with length **L=32**:



Its frequency response is
```
freqz_plot(hk)
```



Observe: The Kaiser window would already be our final filter, if our ideal impulse response would consist of an infinite sequence of ones. This is the case if our ideal filter is only a delta pulse at frequency zero, hence an infinitely small low pass filter.

Observe that the main lobe of this length 32 window (up to about 0.17) is about half as wide as the main lobe at length 16! In this way, we half the transition width of our resulting filter. Here we can say that our pass band ends at normalized frequency 0.17. Hence we need to have our ideal filter with a stop band starting at 0.5-0.17=0.33, resulting in the formula for the ideal impulse response (as above for the length 16 filter), with $\omega_c = 0.33 \cdot \pi$, and a delay of 31/2=15.5:

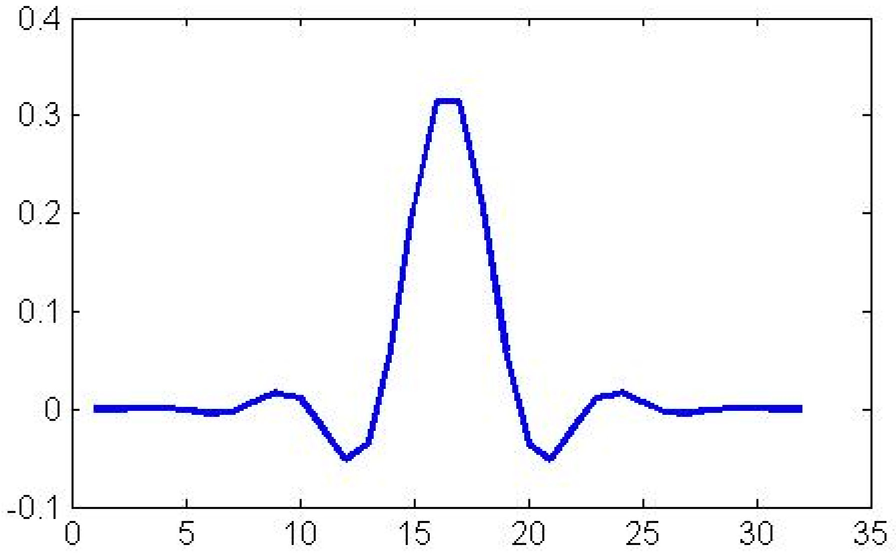
$$h(n) = \frac{\sin(\omega_c (n - 15.5))}{\pi (n - 15.5)} \text{ for } n = 0, \ldots, 31 \,,$$

and multiply it with our Kaiser window.

In iPython:

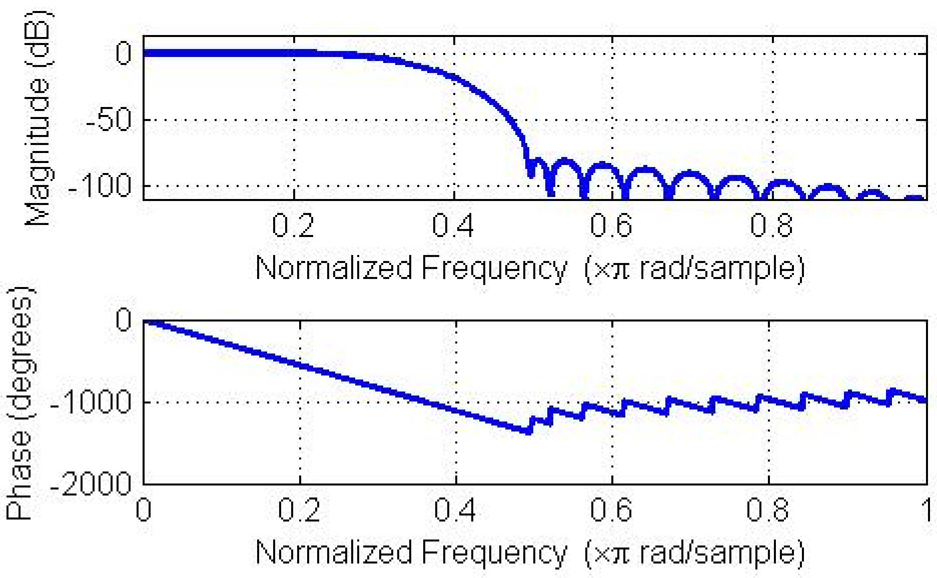
```
n=arange(32);
#ideal impulse response:
h=sin(0.33*pi*(n-15.5))/(pi*(n-15.5));
#Kaiser window:
hk=kaiser(32,8);
#multiply ideal filter and Kaiser window:
hfilt=hk*h;
plot(hfilt)
```

Resulting in the following impulse response:



Its frequency response is:

```
freqz_plot(hfilt)
```



Here we can now see that our stop band, which starts at 0.5, has indeed still enough attenuation, at about -80 dB, and if we take 3dB as the limit for our **pass band**, it goes up to normalized frequency **0.3**.
Going back to our downsampling example, where we downsample from 44.1 kHz to 22.05 kHz sampling rate, the normalized frequency 0.5 corresponds to 11 kHz, and the upper limit of our pass band is 0.3 or 6.6 kHz. This now looks like a usable filter for our application!
This also show why the **usable** frequencies in a time-discrete representation is always **clearly lower** than the **Nyquist** frequency (we need filters, which have **transition bands**).

How to obtain a high pass or band pass, Modulation

1st Approach (Ideal Filter Design):
We again design an ideal filter, and then window it.
For instance if we want to obtain a high pass, we can start to design an **ideal high pass filter**, using our inverse DTFT, which gives us a doubly infinite impulse response, from $-\infty$ to $+\infty$, and then window this ideal impulse response to obtain an FIR filter. For the ideal high pass, we can define the desired frequency response $H_d(\Omega)$ as 1 at the high frequencies, above a cutoff frequency Ω_c, and 0 at the low frequencies,

$$H_d(\Omega)=\left\{\begin{matrix} 1\, for\,|\Omega|>\Omega_c \\ 0\, else \end{matrix}\right\}$$

If we want to have a real valued impulse response, we need to make the frequency response such that its values at negative frequencies are **conjugate complex** of the values at positive frequencies. The easiest way to do it here is to have the negative frequencies identical to the positive frequencies,

$$H_d(\Omega)=\begin{cases} 1 & \text{for} \quad -\pi<\Omega<-\Omega_c \\ 0 & \text{for} \quad -\Omega_c\le\Omega\le+\Omega_c \\ 1 & \text{for} \quad \Omega_c<\Omega<\pi \end{cases}$$

Now we can apply the inverse DTFT, find an analytical solution, just like with the low pass filter, to obtain the ideal impulse response, and then **multiply it with a window** to obtain an FIR filter.

2nd Approach (Modulation):
Shifting our (ideal) pass band to the **desired position in frequency**. In this way, we can turn a low pass into a band pass or a high pass, depending on where we shift our pass band (in this way we can turn a new problem into our known problem, the design of a lowpass filter). How do we shift our pass band in the frequency domain? We convolve it in the frequency domain with a Dirac impulse at the desired center frequency Ω_0 ,

$$H_d(\Omega)*\delta(\Omega-\Omega_0)=H_d(\Omega-\Omega_0)$$

If we want to have real valued impulse responses, we need to preserve the symmetry between positive and negative frequencies, by also shifting the frequency response by the same amount to negative frequencies,

$$H_d(\Omega)*(\delta(\Omega-\Omega_0)+\delta(\Omega+\Omega_0))=H_d(\Omega-\Omega_0)+H_d(\Omega+\Omega_0)$$

How does this change our (ideal) impulse response? To answer, we take the inverse DTFT. The convolution in the frequency domain becomes a multiplication in the time domain. Now we just need the inverse DTFT of $(\delta(\Omega-\Omega_0)+\delta(\Omega+\Omega_0))$, or the inverse DTFT of $\delta(\Omega-\Omega_0)$. To obtain it we can simply use our formula for the inverse DTFT,

$$\frac{1}{2\pi}\int_{-\pi}^{\pi}\delta(\Omega-\Omega_0)\cdot e^{j\Omega\cdot n}\,d\Omega=\frac{1}{2\pi}e^{j\cdot\Omega_0\cdot n}$$

Remember, the integration of a function multiplied with a Dirac impulse is the function value at the position of the Dirac (here: Ω_0), $\int_{-\infty}^{\infty} \delta(\Omega-\Omega_0) H(\Omega) d\Omega = H(\Omega_0)$

Now we can get the inverse DTFT of $(\delta(\Omega-\Omega_0)+\delta(\Omega+\Omega_0))$, which is now

$$\frac{1}{2\pi}\cdot\left(e^{j\cdot\Omega_0\cdot n}+e^{-j\cdot\Omega_0\cdot n}\right)=\frac{1}{\pi}\cdot\cos(\Omega_0\cdot n)$$

This is the function, which we need to multiply with our ideal low pass filter, to obtain an ideal filter where the pass band is centered around Ω_0 . We call this cos function a "**modulation** function", and the multiplication with this function a "**modulation**"
Observe that we can also introduce a phase shift p into this modulation function, for instance turning the cos function into a sine function. This would still work, because in the frequency domain this is a multiplication with another complex exponential from the phase term in the time-domain, just like from a time lag.

Python Example:
This principle can also be applied directly to audio signals. The following example takes the signal from a microphone and modulates (multiplies) it with a 500 Hz sine function:

```
python pyrecplay_modulationblock.py
```

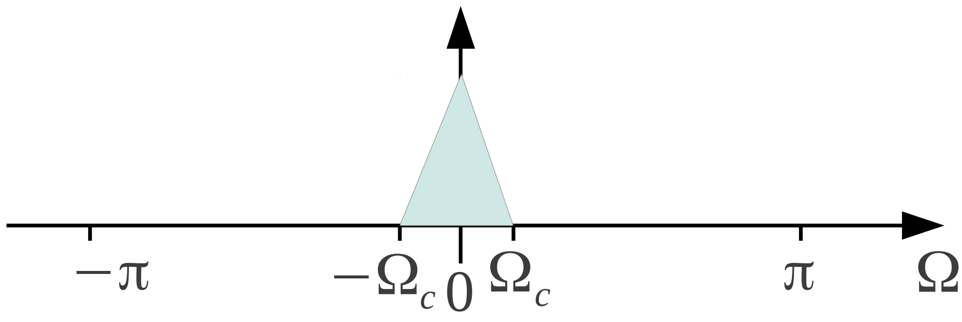
Observe: Sounds and voice sound higher pitched, like a Mickey-Mouse voice, as a result of the frequency shift!

In Conclusion:
We can shift our ideal filter in the frequency domain by multiplying the ideal (or already windowed) impulse response with a modulation function

$$\frac{1}{\pi}\cdot\cos\left(\Omega_0\cdot(n-p)\right)$$

where p is some phase delay.

This is illustrated in the following pictures. First a low pass original spectrum,



Next the spectrum after our modulation with frequency Ω_0

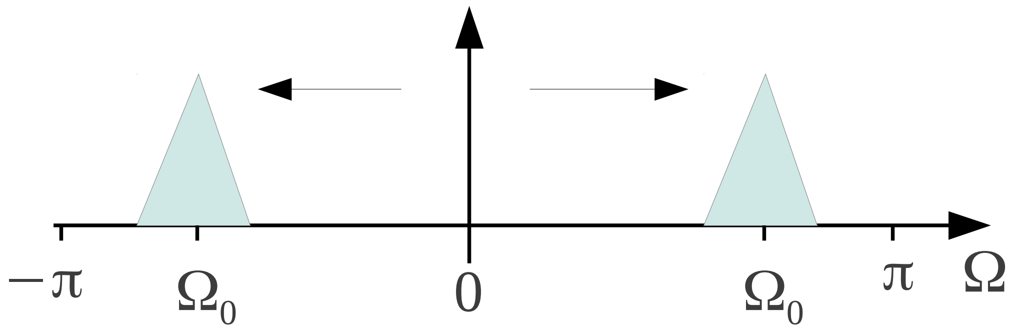


Figure 1.

Example:

To obtain a high pass, we need to shift the pass band from 0 to π , hence we get $\Omega_0=\pi$, and choose p=0. The modulation function is then

$$\frac{1}{\pi}\cdot\cos\left(\pi\cdot n\right)$$

which is simply a sequence of +/-1!

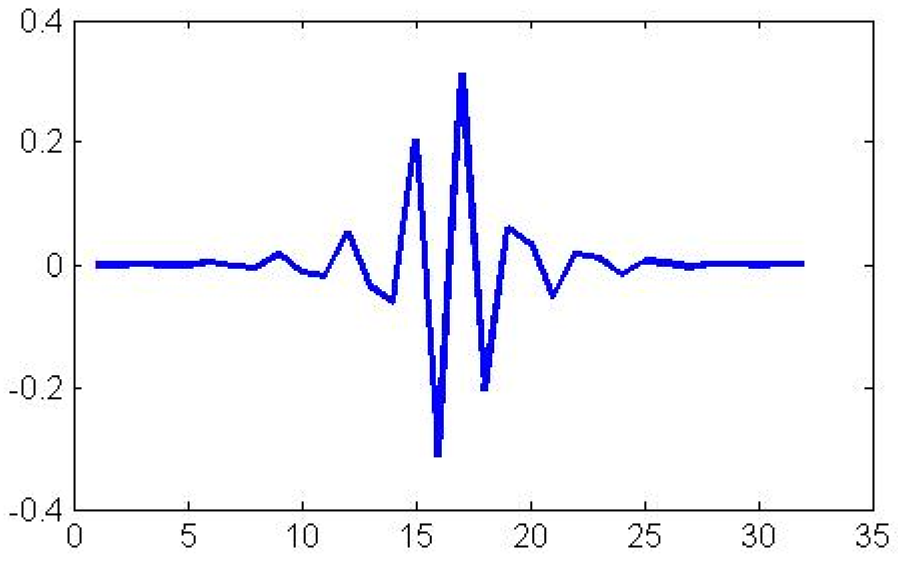
For instance for our low pass:

ipython -pylab

```
n=arange(32);
#ideal impulse response:
h=sin(0.33*pi*(n-15.5))/(pi*(n-15.5));
#Kaiser window:
hk=kaiser(32,8);
#multiply ideal filter and Kaiser window:
hfilt=hk*h;
```

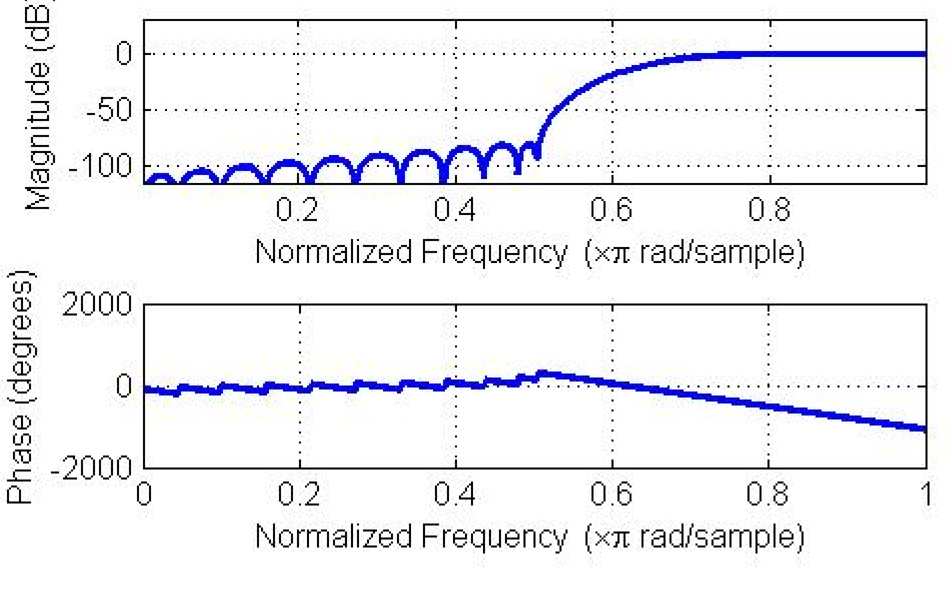
We can make a **high pass** out of it using modulation

```
hp=hfilt*cos(pi *arange(32))
plot(hp)
```



Here we can see the effect of the modulation with the +/-1 sequence.

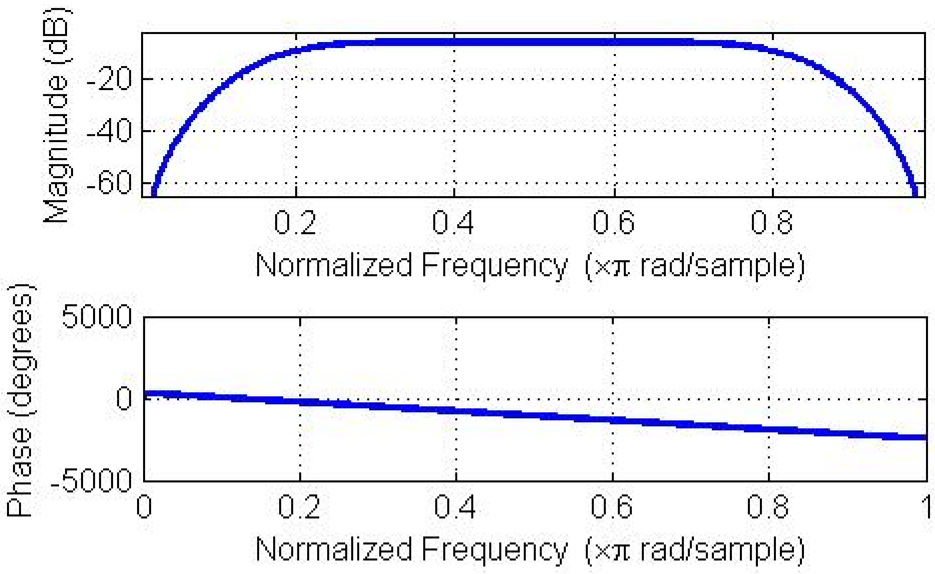
The resulting frequency response is
```
from freqz_plot import *
freqz_plot(hp)
```



and we see that we indeed obtained a high pass. It is basically looks mirrored around the center. It really is shifted, but what we see as the high pass part was the negative frequency part of our low pass.

We can also obtain a **band pass** with center frequency pi/2 with

```
hbp=hfilt.*cos(pi/2 *arange(32))
freqz_plot(hbp)
```



and we see that we indeed get a band pass!

Observe that we obtain a pass band in this band pass case which is twice as wide as in the case of a low pass or a high pass, because here the negative frequencies of the low or high pass case appear as the other half of the pass band, as could be seen in Figure 1.