Digital Signal Processing 2/ Advanced Digital Signal Processing Lecture 11, Complex Signals and Filters, Hilbert Transform

Gerald Schuller, TU Ilmenau

Imagine we would like to know the precise, **instantaneous, amplitude of a sinusoid**. Just looking at the function this might not be so easy, we would have to determine the maximum or minimum, and depending on the sinusoidal frequency this might take some time, during which the amplitude also might have changed. But if we have a **complex exponential** with samples

$$x(n)=A\cdot e^{j\Omega n}=A\cdot\cos(\Omega n)+jA\cdot\sin(\Omega n)$$

(we know x(n) but not A)
we can easily determine the amplitude A only knowing one sample x(n) by taking the magnitude of this complex exponential,

$$A=\sqrt{\Re(x(n))^2+\Im(x(n))^2}=$$
$$=\sqrt{((A\cdot\cos(\Omega n))^2+(A\cdot\sin(\Omega n))^2)}$$

Observe that this computation of A is independent of the time index n, so it can be done **at every time instance**.
So if we not only have the sine or cosine function in itself, but both, we can easily compute the instantaneous amplitude in this

way. This means, instead of just having the sinusoidal function, we also have the **90 degrees phase shifted** version of it, to compute the amplitude. The problem is, if we only have one part (e.g. the real part) how do we get this 90 degrees phase shifted version? For example, look at the sine function. It already consists of 2 complex exponentials,

$$\sin(\Omega n) = \frac{1}{2j}\left(e^{j\Omega n} - e^{-j\Omega n}\right)$$

we just have one exponential too many. If we look at it in the Fourier Domain, we see that one exponential is at positive frequencies, and the other is at negative frequencies. If we could remove one of the 2 exponentials, for instance at the negative frequencies, we would have reached our goal of obtaining a complex exponential for amplitude computations. So what we need is **a filter**, which **attenuates the negative frequencies**, and leaves the **positive frequencies unchanged**!
So how do we obtain such a filter? First we formulate our requirement in the **frequency domain** (the DTFT domain):

$$H(\Omega) = \begin{Bmatrix} 1 & for\, \Omega > 0 \\ 0 & for\, \Omega < 0 \end{Bmatrix}$$

We could multiply our signal spectrum with this frequency domain formulation (which is often not practical), or in the time domain convolve with the **impulse response** of the resulting filter obtained with **inverse DTFT**,

$$h(n)=\frac{1}{2\pi}\int_0^{\pi} 1\cdot e^{j\Omega n}\,d\Omega=$$

for $n\neq 0$ this becomes

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

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

$$=\left\{\begin{matrix}\frac{j}{\pi n}\ for\,n\,odd\\ 0\ for\,n\,even\end{matrix}\right\}$$

For n=0 this inverse DTFT integral becomes

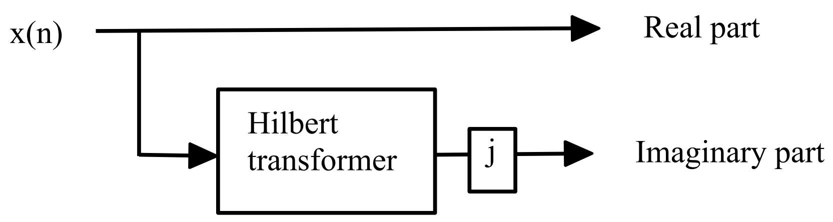
$$h(0)=\frac{1}{2\pi}\cdot\pi=\frac{1}{2}$$

Hence the resulting impulse response of this one-sided filter becomes

$$h(n)=\frac{1}{2}\cdot\delta(n)+\left\{\begin{matrix}\frac{j}{\pi n}\ for\,n\,odd\\ 0\ for\,n\,even\end{matrix}\right\}$$

This is now the resulting **impulse response** (time domain) of our **filter which passes all the positive frequencies and attenuates the negative frequencies.**

Here we can see that the first part with the **delta function** represents the **real part**, which is the signal itself (signal convolved with the delta impulse), except for a factor of 2. The second part represents the imaginary part of our one-sided signal (pos. frequencies). Multiplying both parts with this factor of 2 for simplicity, we obtain the following structure,



where the **Hilbert transformer** $h_H(n)$ is

$$h_H(n) = \begin{cases} \frac{2}{\pi n} \; for\, n\, odd \\ 0 \; for\, n\, even \end{cases}$$

(See also: Oppenheim, Schafer, “Discrete-Time Signal Processing”). The x(n) is our real valued signal. Observe that only this part, which creates the imaginary signal part, is the Hilbert Transformer. We can use it to construct a filter that suppresses the negative frequencies.

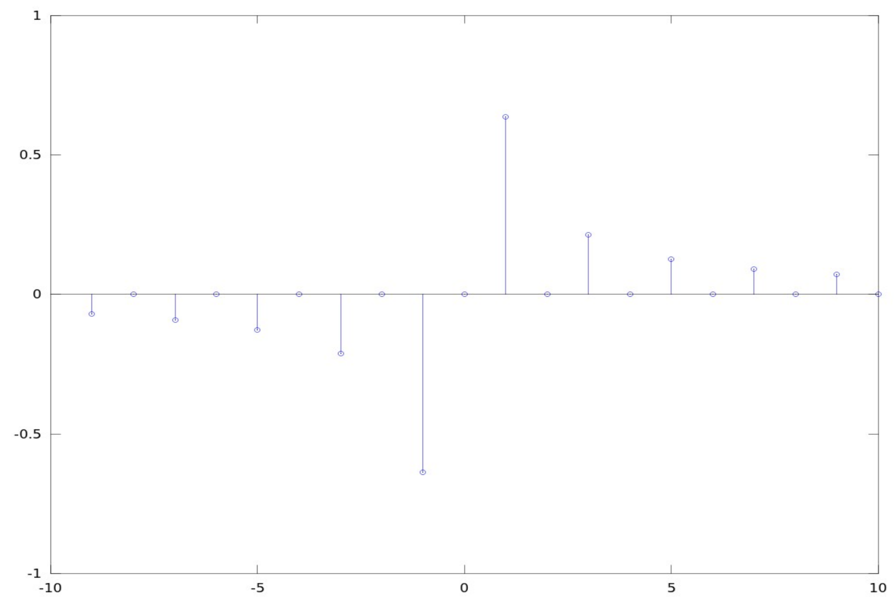
This means, we take our **original signal** (the sinusoid) and **define it as our real part**. Then we take the **Hilbert filtered signal**, filtered with the above $h_H(n)$, and define it as our **imaginary part**. Then both together have a **one-sided, positive only spectrum**! That also means, our Hilbert transform filter is our

90 degrees phase shifter that we where looking for!
You can also imagine, if the real part is a $\cos$ signal, then the Hilbert Transformer generates the $j \cdot \sin$ part.

Matlab/Octave example: Plot the Hilbert transformer for n=-10 ..10:

```
h=zeros(1,20);
n=-9:2:10;
h(n+10)=2./(pi*n);
stem(-9:10,h)
```

Here we see that we have negative indices. If

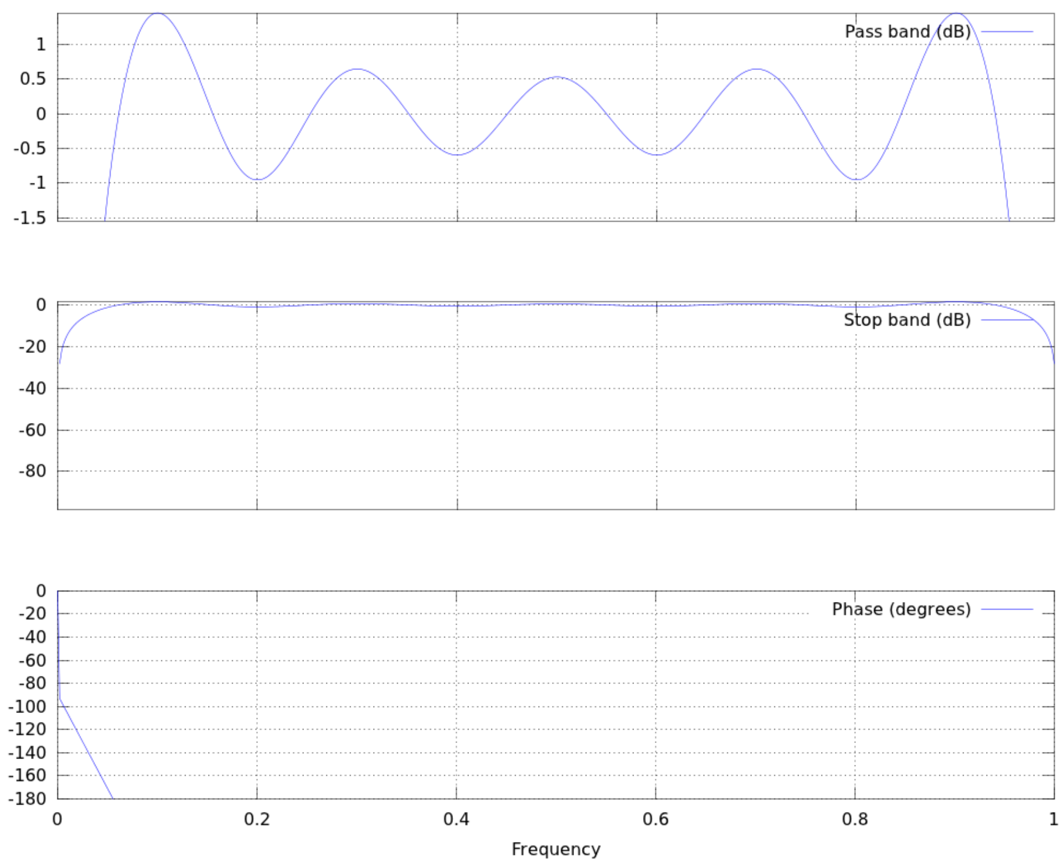


we want to obtain a causal system, we need to shift them to at or above zero using a suitable delay, hence the Hilbert transform involves some delay. Hence to make the imaginary and real part to fit to each other, we have to **delay the real part accordingly**.

This type of complex signal, with a 1-sided spectrum, is also called an “**analytic signal**”.

Let's take a look at the frequency response of our filter. Since we have included a delay, our phase should be our 90 degrees phase shift plus the linear phase from the delay. Hence our phase curve should hit the phase axis at frequency 0 at 90 degrees, for which we would like to zoom in to this part.

```
freqz(h);
axis([0 1 -180 0])
```



Here we can see that the phase curve indeed would hit the 90 degree mark at the phase axis, except that it goes to 0 before it. This is because a finite length Hilbert transformer does not work at frequency zero. This is also what we see at the magnitude plot. It only reaches about 0 dB attenuation at about frequency 0.08 and reaches higher attenuations at frequencies below about 0.05 and above about 0.95. Hence it is only a working Hilbert transformer within this frequency range.

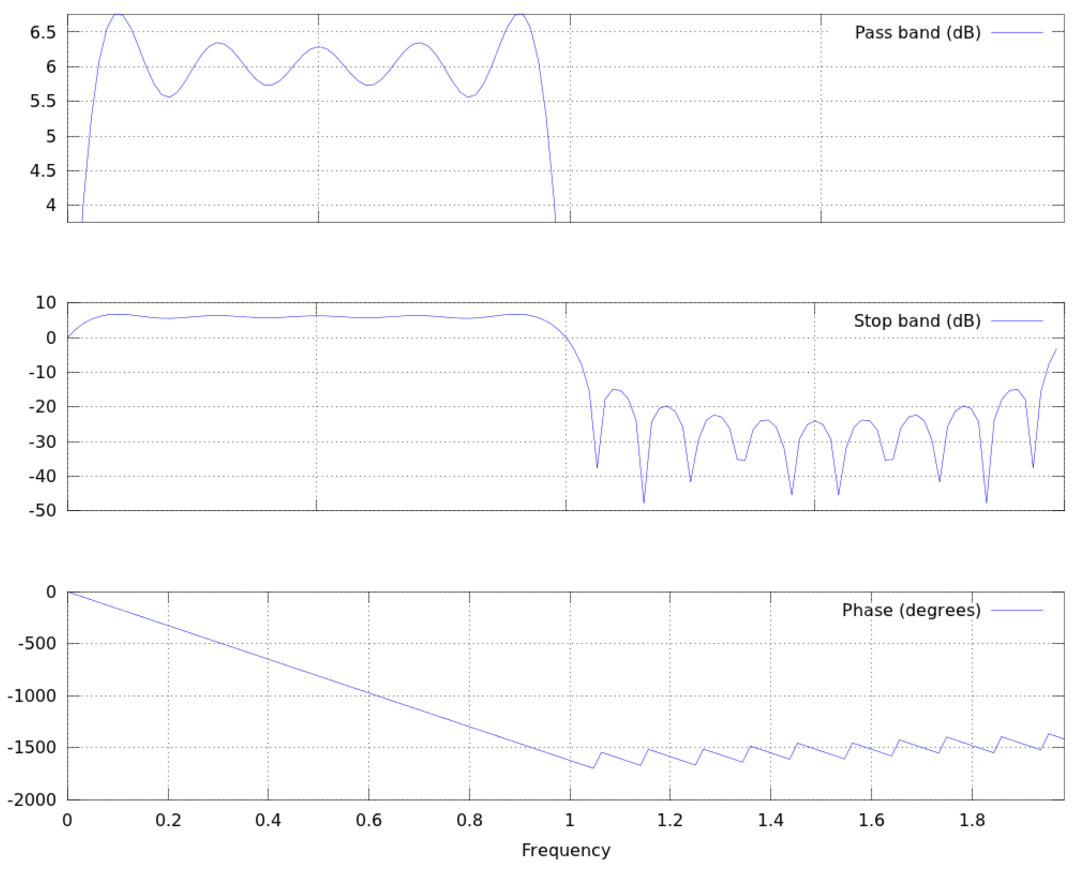
If we want to plot the frequency response of our **entire filter** (not just the Hilbert transformer part), which passes only the positive frequencies, we first need to construct our resulting complex filter, and then plot the frequency response on the **whole** frequency circle. First we need to create the correspondingly delayed unit impulse as the real part:

```
%construct a delayed impulse to implement the
delay for the real part:
delt=[zeros(1,9),1,zeros(1,10)]
%delt =
%0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
%0
%Then we need to add our imaginary part as our
Hilbert transform to obtain our complex filter:
h=zeros(1,20);
n=-9:2:10;
h(n+10)=2./(pi*n);

hone=delt+j*h
hone =
Columns 1 through 3:
0.00000 - 0.07074i 0.00000 + 0.00000i 0.00000 -
0.09095i
Columns 4 through 6:
0.00000 + 0.00000i 0.00000 - 0.12732i 0.00000 +
0.00000i
Columns 7 through 9:
0.00000 - 0.21221i 0.00000 + 0.00000i 0.00000 -
0.63662i
```

```
Columns 10 through 12:
1.00000 + 0.00000i 0.00000 + 0.63662i 0.00000 +
0.00000i
Columns 13 through 15:
0.00000 + 0.21221i 0.00000 + 0.00000i 0.00000 +
0.12732i
Columns 16 through 18:
0.00000 + 0.00000i 0.00000 + 0.09095i 0.00000 +
0.00000i
Columns 19 and 20:
0.00000 + 0.07074i 0.00000 + 0.00000i
>>> freqz(hone,1,128,'whole');
```

The resulting plot is:

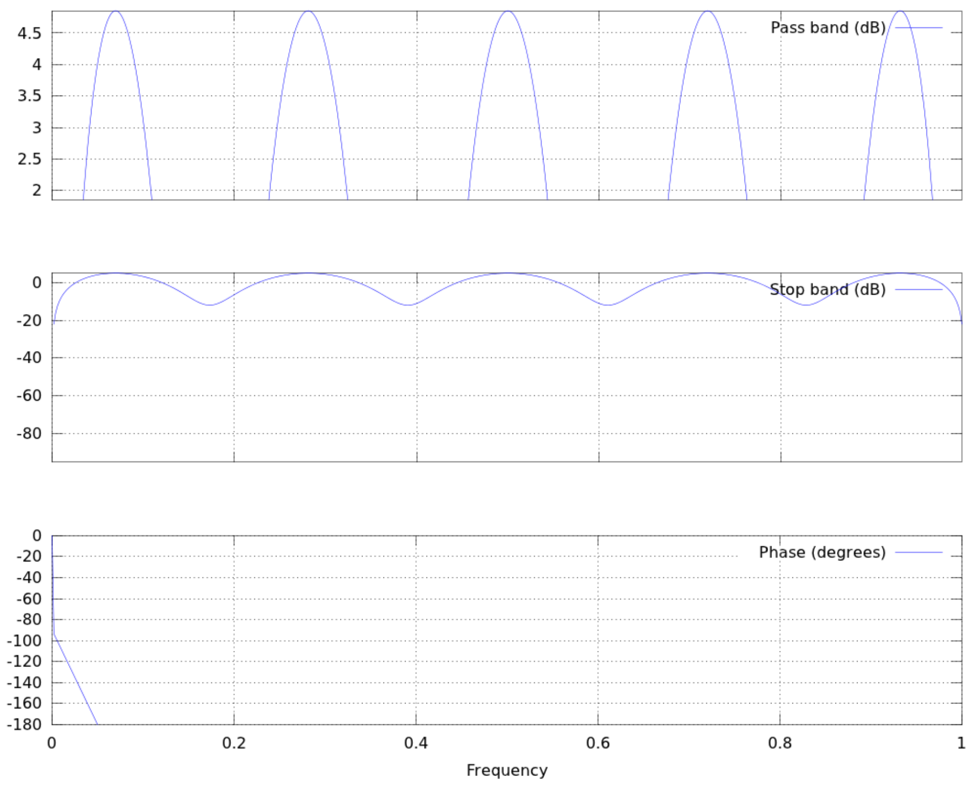


Here we can see that we have indeed a passband at the positive frequencies between 0 and 1. Observe that the passband is at about

6dB above 0dB, because we multiplied our filter by 2 to make it simpler.
The negative frequencies appear between 1 and 2 (or pi and 2pi) on the frequency axis, and we can see that we get about -30 dB attenuation there, which is not too much, but which we could increase it by making the filter longer. This also gives us a good indication of how well our filter is working!

The Octave/Matlab function “remez” also has an option for a Hilbert transform filter.

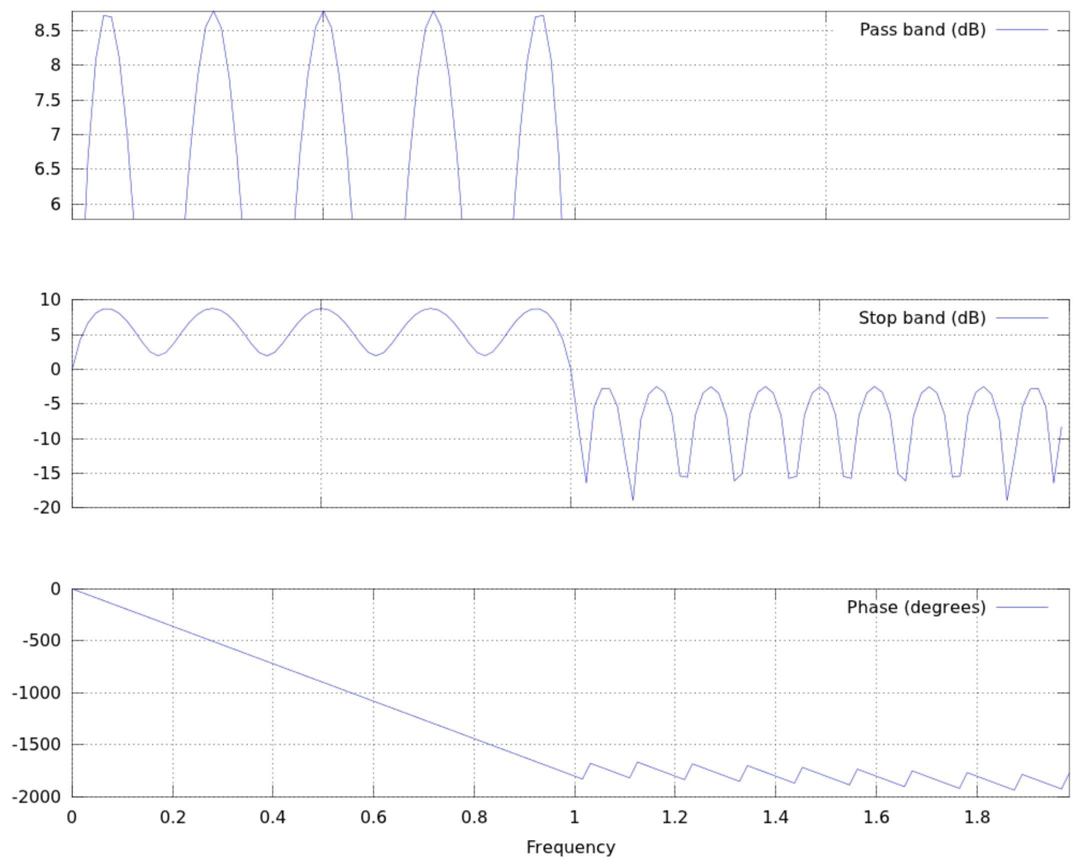
```
b=remez(20,[0  1],[1 1 ],'hilbert');
freqz(b);
axis([0 1 -180 0]);
```



This filter also fullfils the 90 degree phase shift requirement, but seems to have much stronger ripples in the magnitude of the frequency response!
But observe the equi-ripple behaviour in the passband, which is what we expect from remez.

Let's look at the whole frequency circle again, in Octave:

```
%Delay for the real part:
delt=[zeros(1,10),1,zeros(1,10)]
delt =
Columns 1 through 20:
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0
Column 21:
0
```
honeremez=delt+j*b'
```
honeremez =
Columns 1 through 3:
0.00000 - 0.00000i 0.00000 - 0.41954i 0.00000 +
0.00000i
Columns 4 through 6:
0.00000 - 0.09991i 0.00000 - 0.00000i 0.00000 -
0.13309i
Columns 7 through 9:
0.00000 - 0.00000i 0.00000 - 0.21540i 0.00000 -
0.00000i
Columns 10 through 12:
0.00000 - 0.63701i 1.00000 + 0.00000i 0.00000 +
0.63701i
Columns 13 through 15:
0.00000 + 0.00000i 0.00000 + 0.21540i 0.00000 +
0.00000i
Columns 16 through 18:
0.00000 + 0.13309i 0.00000 + 0.00000i 0.00000 +
0.09991i
Columns 19 through 21:
0.00000 - 0.00000i 0.00000 + 0.41954i 0.00000 +
0.00000i
```
>>> freqz(honeremez,1,128,'whole');

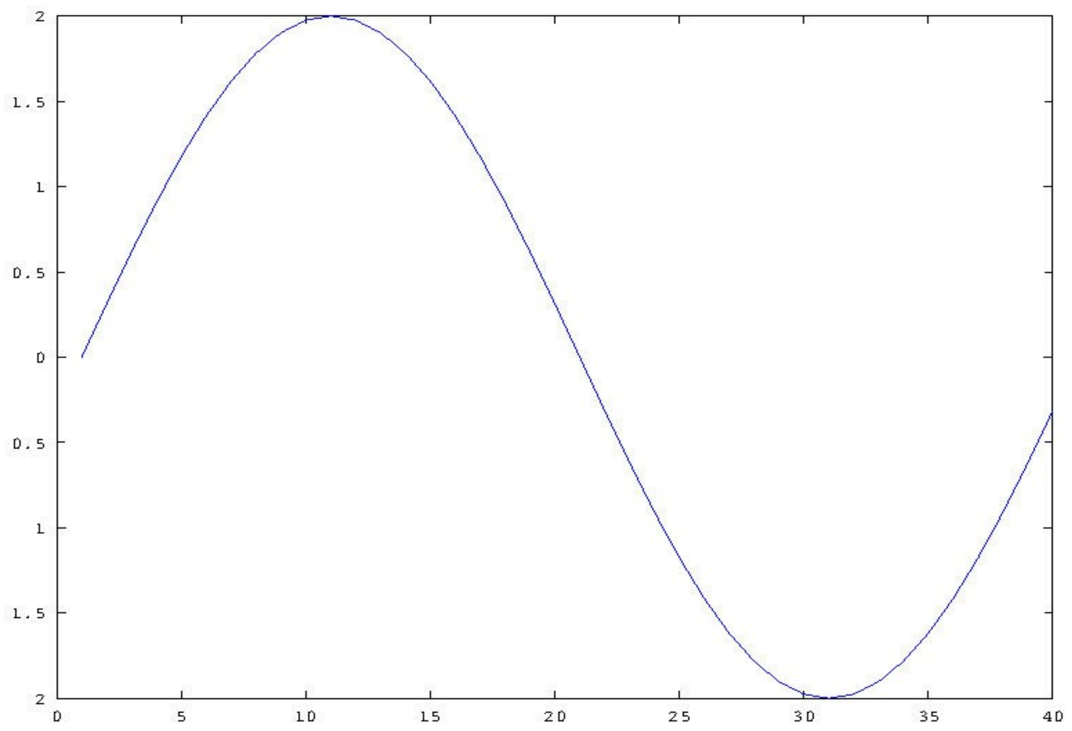


Here we can see that we have **less attenuation** as before (only about -10 dB in the stopband compared to the pass band!), more ripples in the passband, but we also have a **narrower transition band** at the high and low frequency end of the spectrum, compared to the previous filter! Also again observe the equi-ripple behaviour in the pass band and also the stop band. Probably we obtain more practical filters, if we change the corner frequecies to slightly above 0 and slightly below 1.

Example for the Measurement of the Amplitude:

We can now test our application example of measuring the Amplitude of a sinusoid with our Hilbert transform design. We saw that the lower end for the passband of our design is at normalized frequency of about 0.05. Hence we test a sinusoid of that frequency,

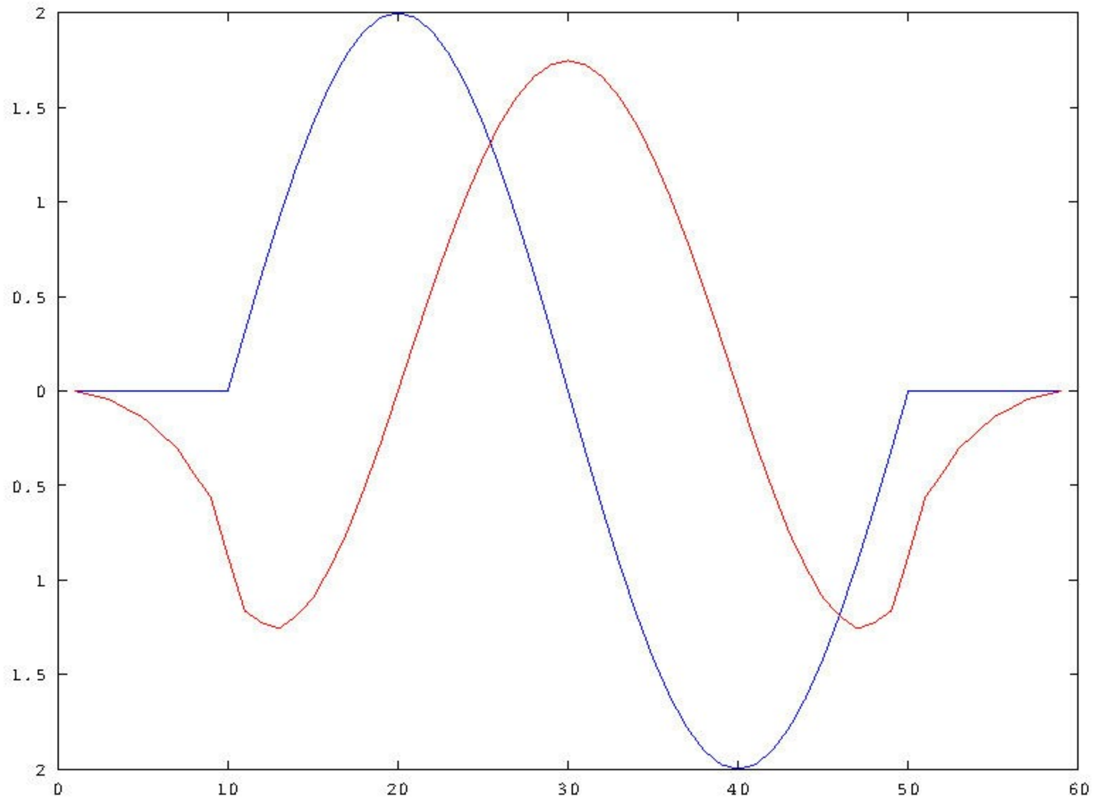
```
x=2*sin(pi*0.05*(0:39));
plot(x)
```



Now we can filter it with our filter which passes only positive frequencies “hone”,

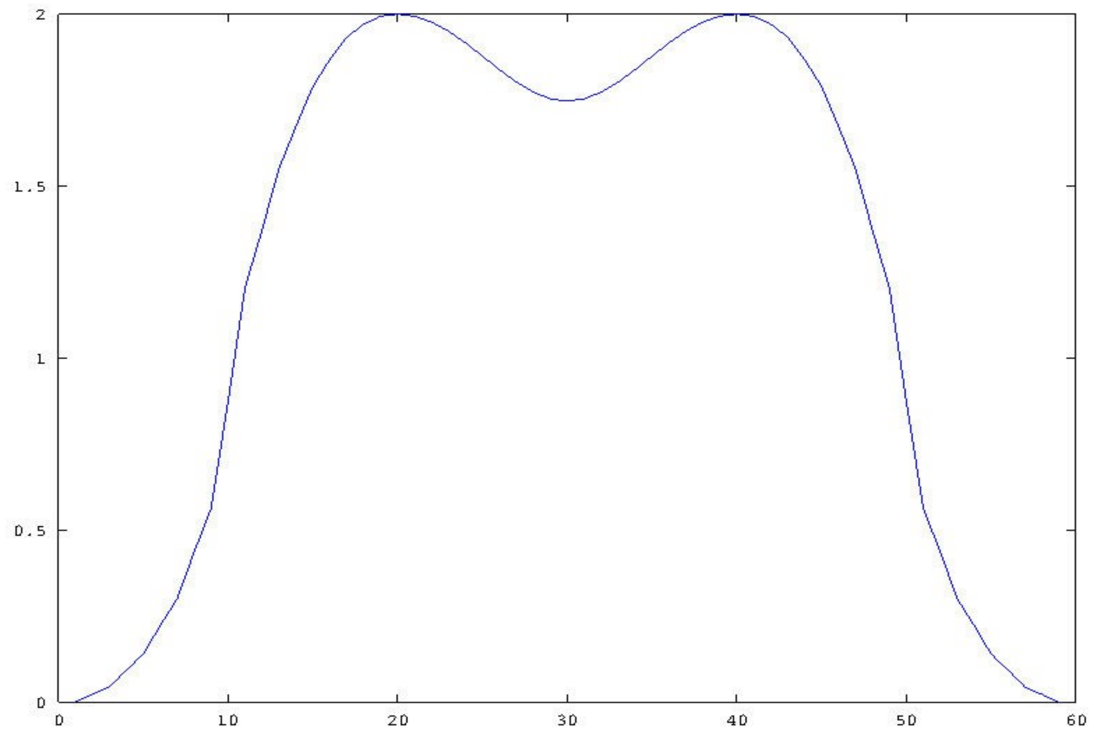
```
xhone=conv(x,hone);
plot(real(xhone))
hold
plot(imag(xhone),'r')
```

Here we can see that we get indeed a 90 degree phase shifted version, the red curve, about between sample 15 and 45.



Now we can compute the magnitude of this complex signal “xhone” to obtain the amplitude of our sinusoidal signal,

```
hold off
plot(abs(xhone))
```



We see that between about sample 15 and 45 we obtain the **amplitude** of our sinusoidal signal with about **10% accuracy**, which roughly corresponds to the -20dB attenuation (corresponding to an attenuation factor of 0.1) that our filter “hone” provides. This also hints at the fact that we can improve the magnitude estimation by having a filter with a **higher attenuation** at negative frequencies.