Frequency-domain anti-aliasing filter

Sergio Frasca (April 2006)

Introduction

Points that should be clear:

- a f-d filter is a filter that is easier than a time domain filter to be synthesized, in case of "hard" requirements (0 phase, very steep frequency cut,…)
- it is a FIR filter, so it is intrinsically stable and it operates on the data exactly like a normal time domain filter (but normally has many more coefficients)
- it is much faster than the equivalent time domain filter
- the well known problems of the spectral estimation by periodograms don't affect the filter (except in the case when the spectral estimation is used for the synthesis of the filter (as in the case of whitening and Wiener adaptive filter synthesis).

A typical drawback of a f-d filter is that it cannot be applied in real time, or, more exactly, if it is used in real time it loses its velocity: these because it "filters" many samples at once, so it has to expect for a certain number of samples to be accumulated. To be used in real time, one should compute an FFT for each sample. In any case, if one wants 0-phase filters, these are strictly non-causal (and so not feasible for real time operation).

Another "drawback" is that these filters are often neglected in many books of signal analysis.

Now we discuss two topics, that arouse in the Virgo discussion of 4-4-2006:

- the filter error
- the response to particular "hard" signals

In the Appendices there are the characteristics of the frequency domain anti-aliasing filter compared with other proposed solution and the Matlab script used to produce the plots.

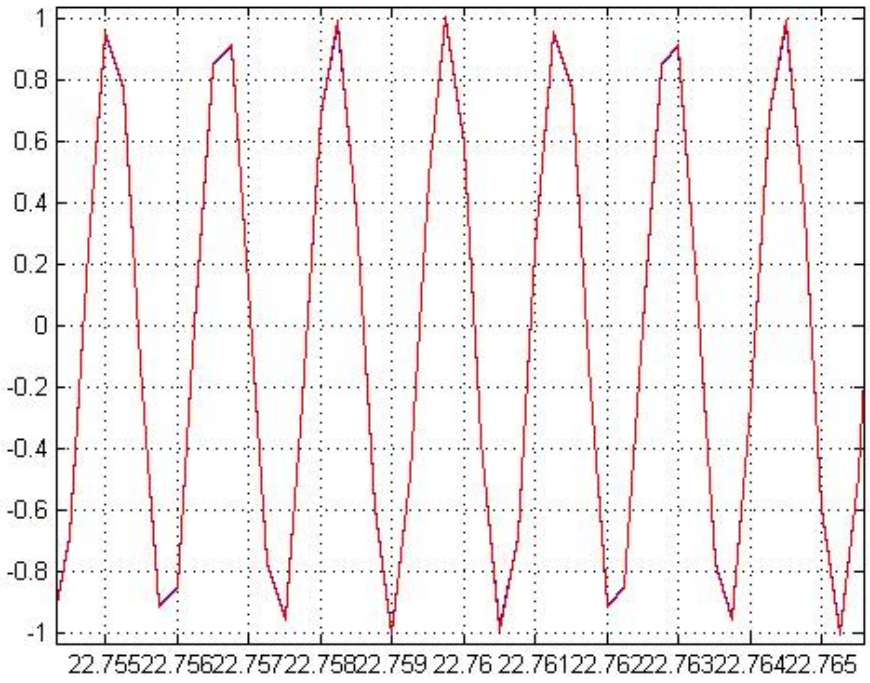
The filter error

I have synthesized a sampled sinusoidal signal (frequency 640 Hz, amplitude 1) at sampling frequencies of 20000 Hz (**A20000** – 1000000 samples) and 4000 Hz (**A4000** – 200000 samples).

From the A20000 I produced, by the frequency-domain decimation procedure **gd_decim** (inside Snag), the 4000 Hz signal **B4000** (200000 samples, but about 1000 on each extreme should be excluded). The FFT length used in **gd_decim** is 16384, so the A20000 data have been divided by the procedure in 123 interlaced pieces.

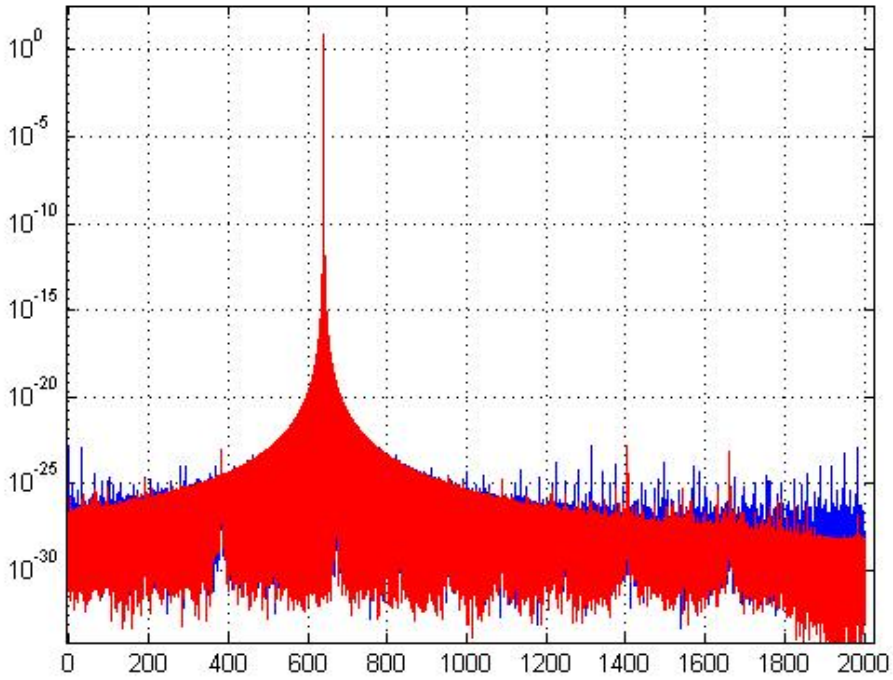
Then I just subtract the A4000 from B4000, obtaining the error signal **E4000**.

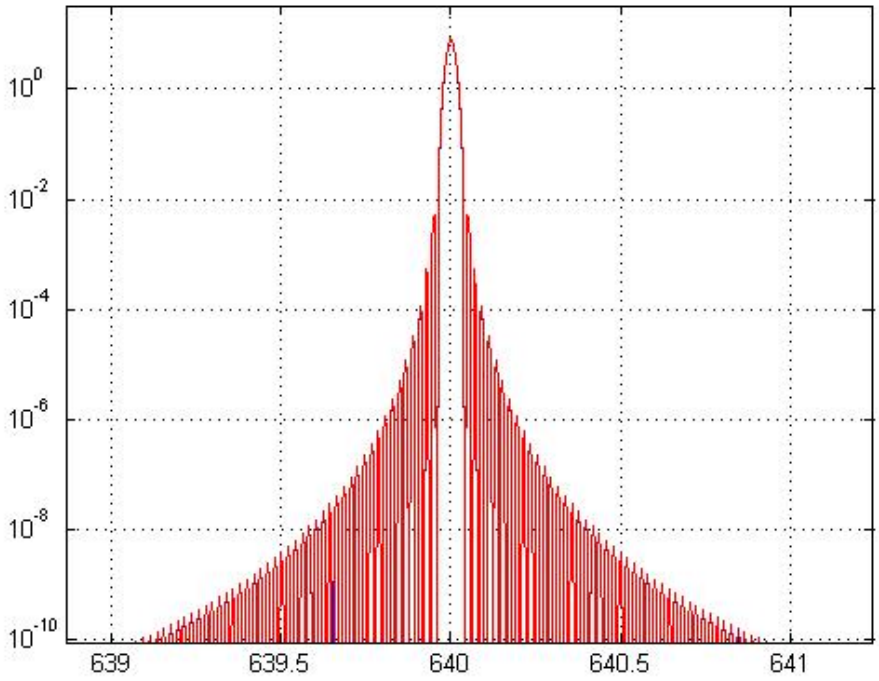
Here are, in the same plot, the A4000 (**blu**) and B4000 (**red**) signals:



(the two signals are practically perfectly superposed, so only the second appears).

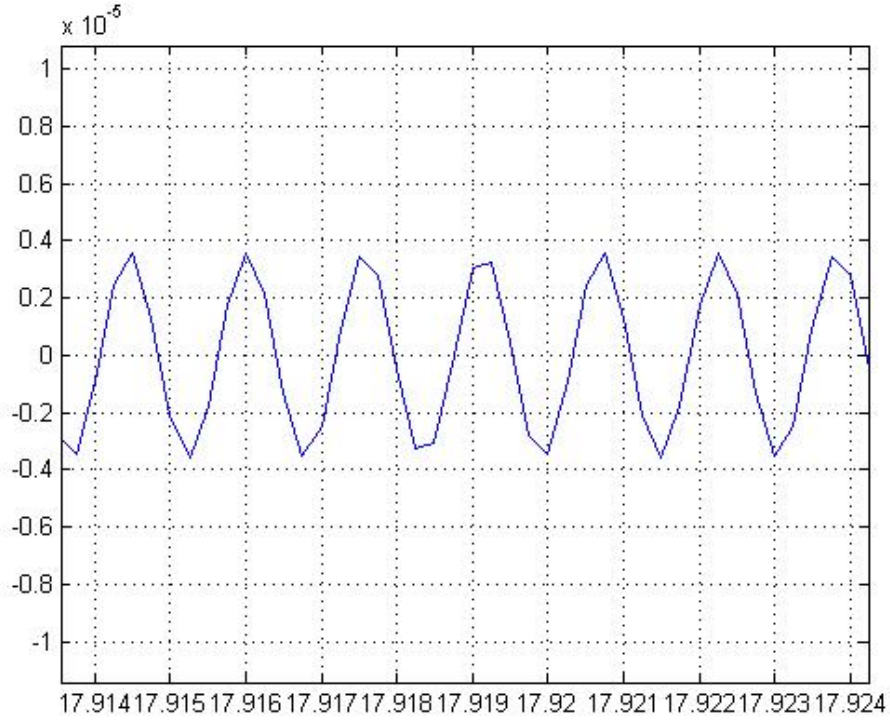
This is the spectrum



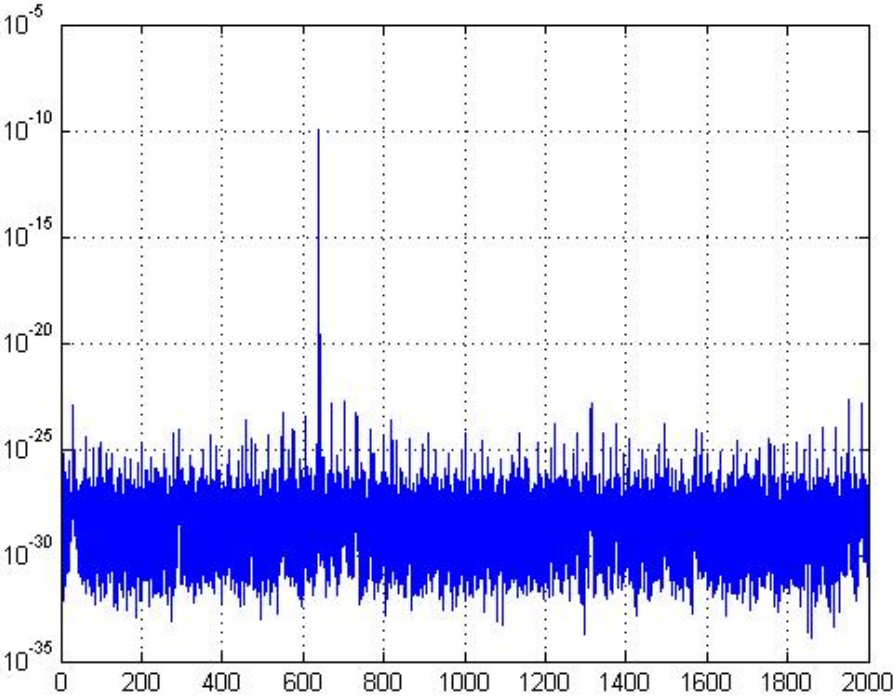


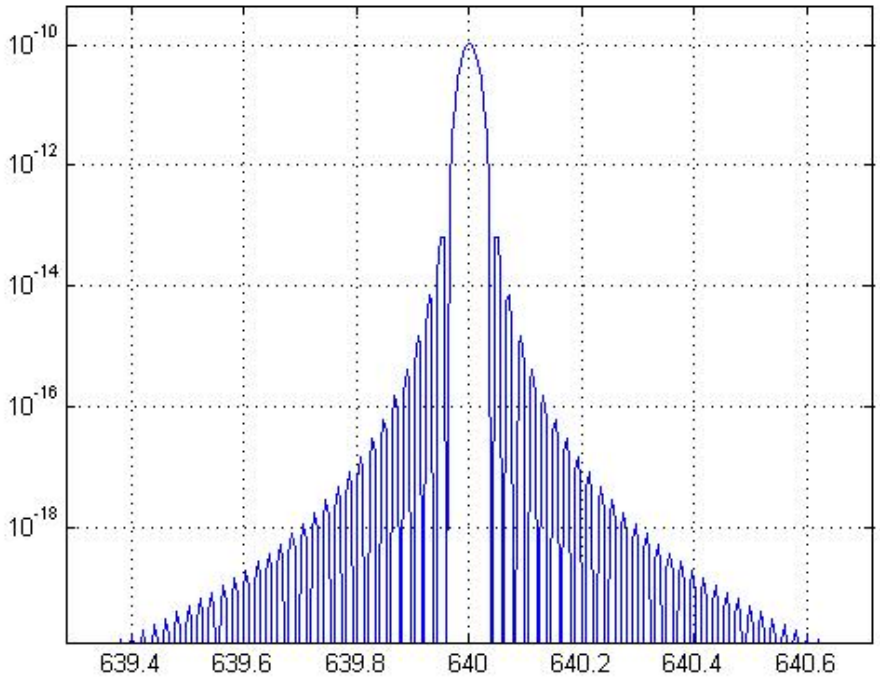
as it is clear the signals are almost identical. Note that the spectral spread is due to the spectral estimation, and is the same on both A4000 and B4000.

To appreciate the difference, here is the E4000 signal:



and the spectrum:

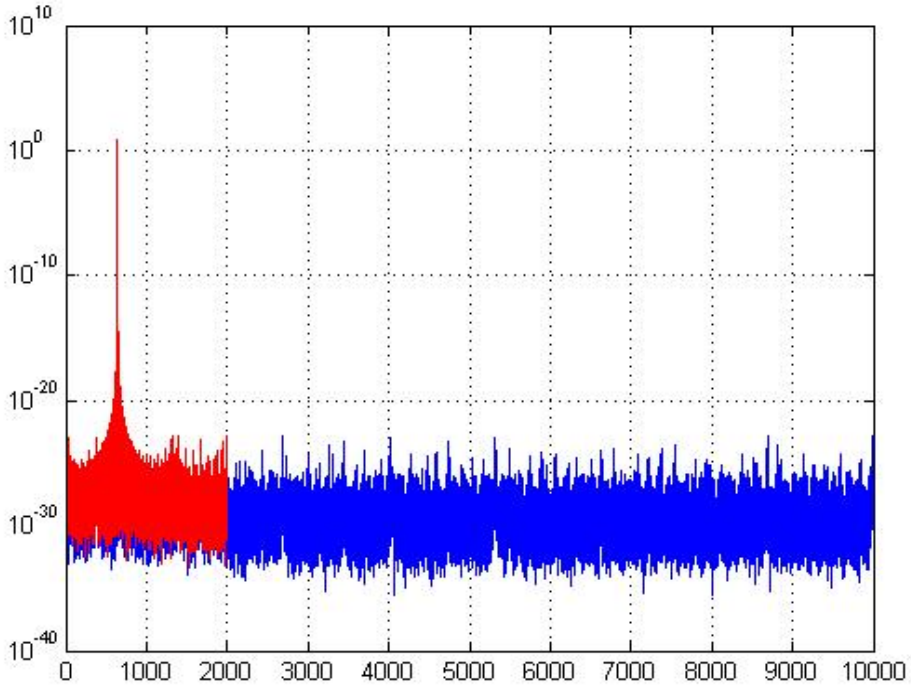




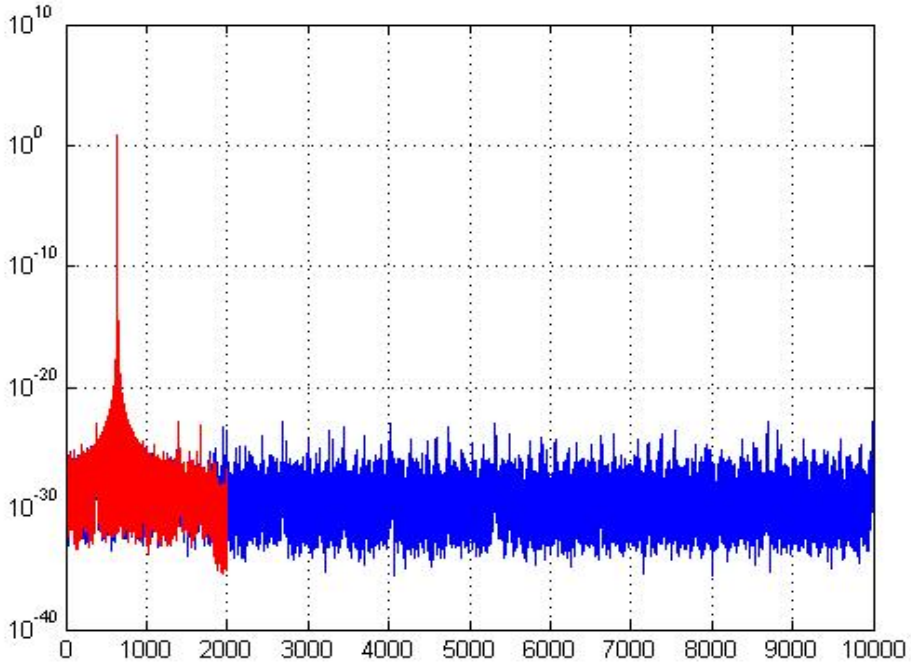
In practice we can see that the error signal is composed by two components:

- a sinusoid at the same frequency of the signal, due to the amplitude error of the filter (<0.00001)

- a wide band noise, due to the quantization and computation errors. Note that this effect is present both in the A4000 and in B4000 and it seems higher in A4000 (the "just sampled" sinusoid). This should be due to the fact that the quantization noise has a spectrum that is inversely proportional to the sampling frequency. Here are the spectra of A20000 and A4000:



for comparison, here is A20000 and B4000



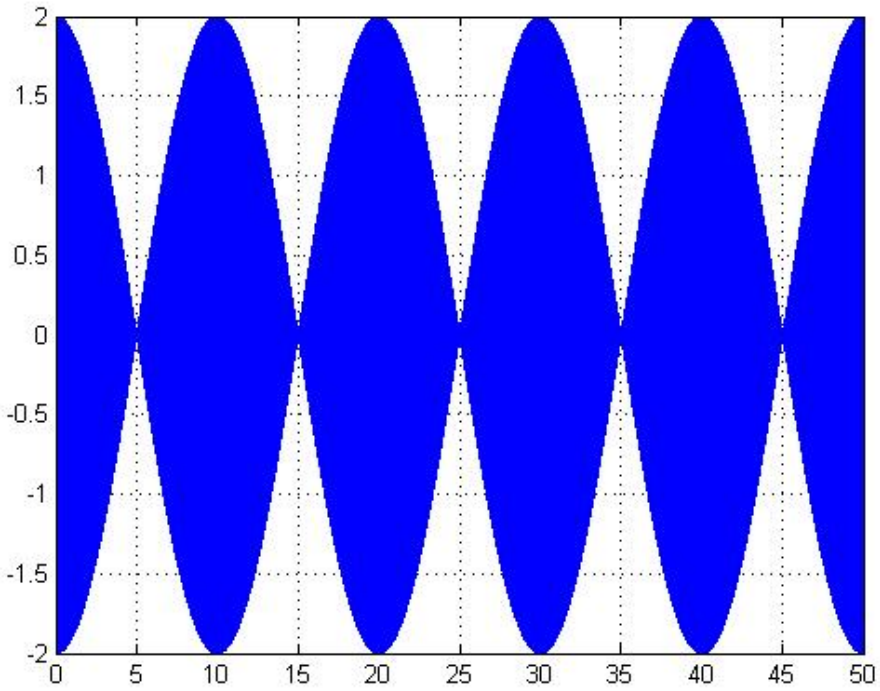
so, the reconstruction by gd_decim is, for this aspect, better than the directly sampled data at 4000 Hz.

Particular signals

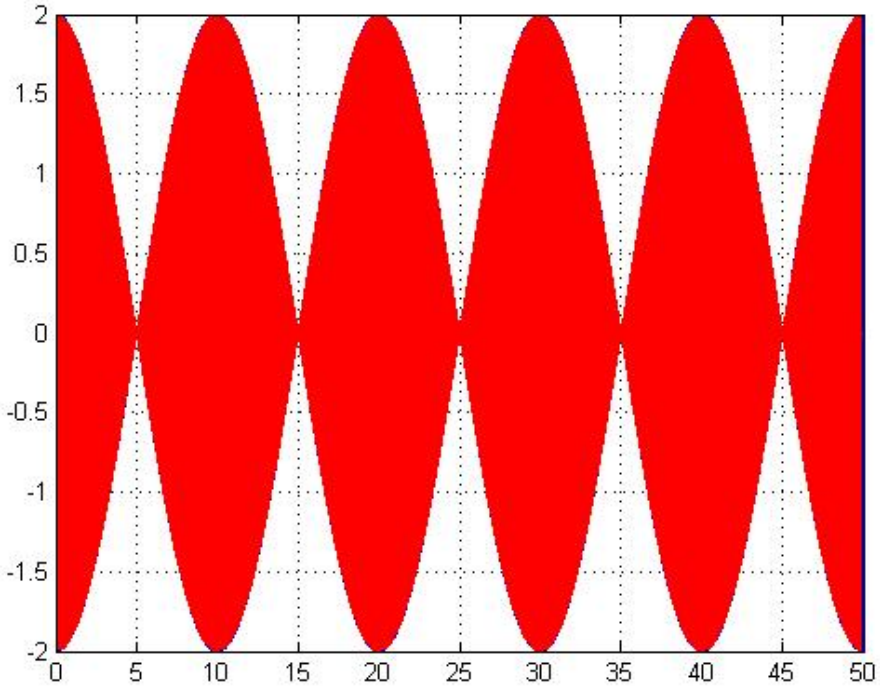
In order to understand the behaviour of the a-a filter, the analysis of the error should be enough, but for a better understanding, here is the analysis for particular cases.

Slow beat

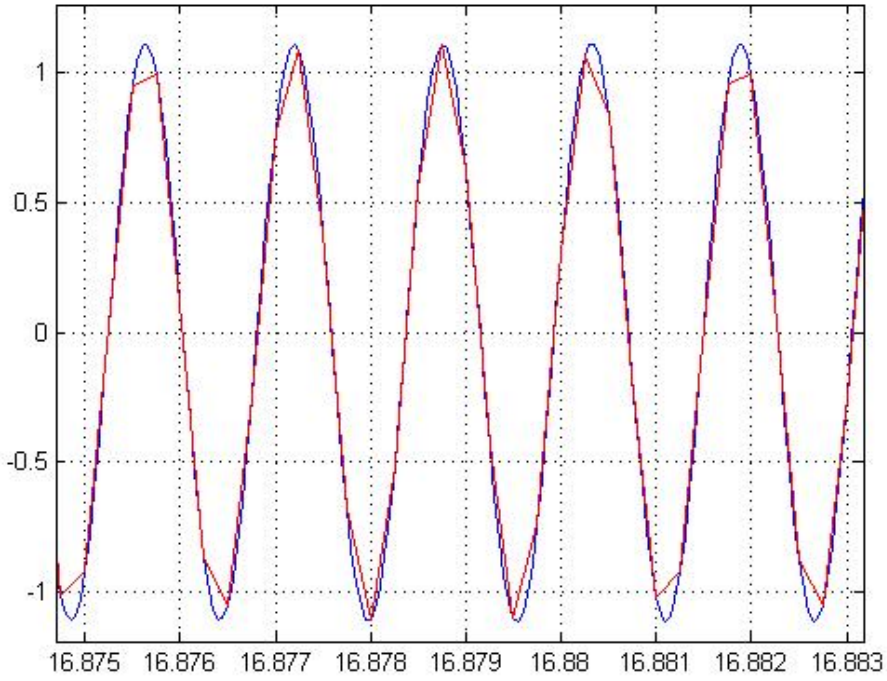
Consider a signal that is the beat of two frequencies, say 640 Hz and 640.1 Hz. This is the plot (sampled at 20 kHz):



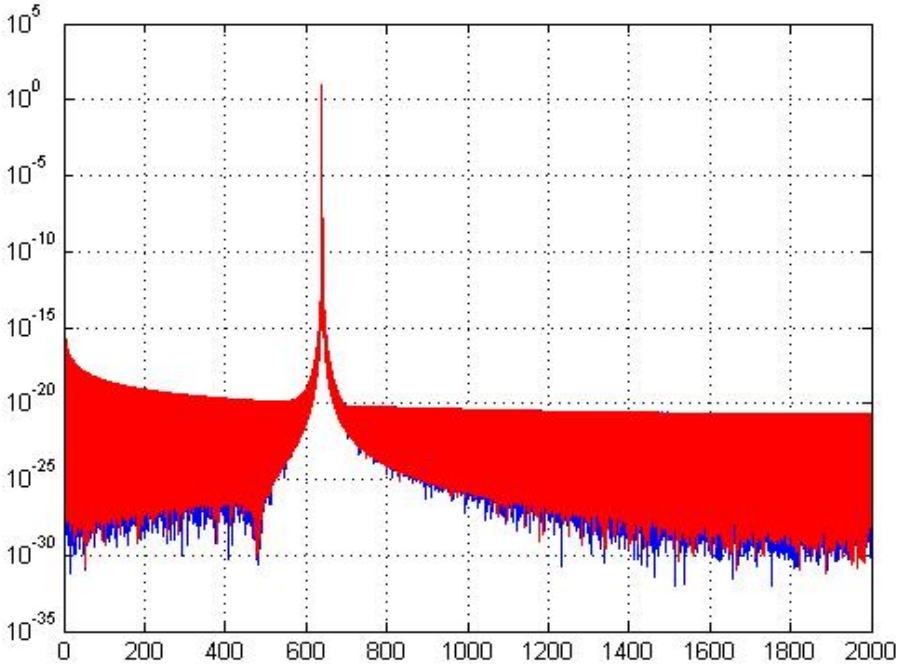
and this is the output of the gd_decim (at 4 kHz) (superposed to the former):

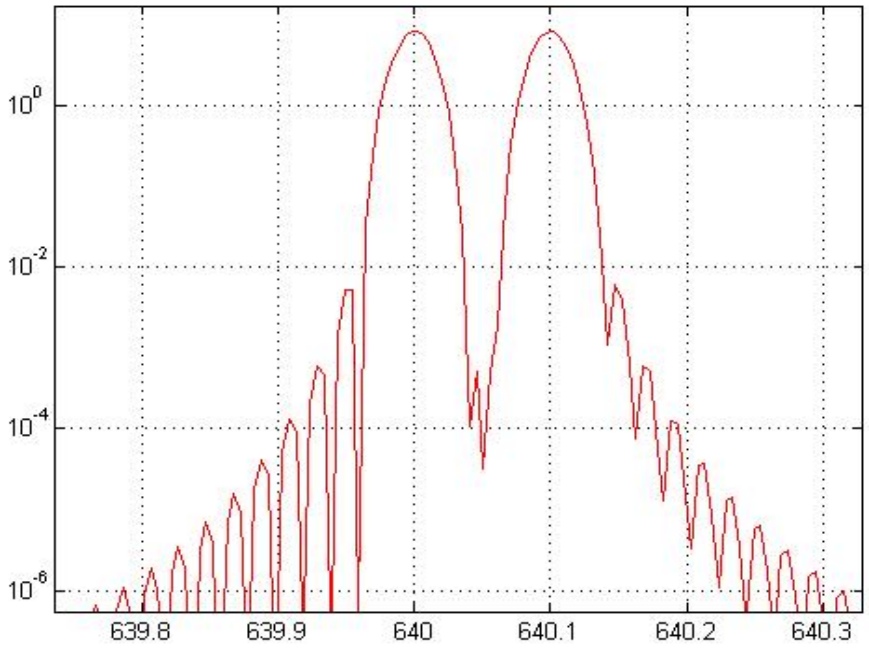


This is the zoom of both the signals:



The spectrum of the gd_decim signal and that of the theoretic one is



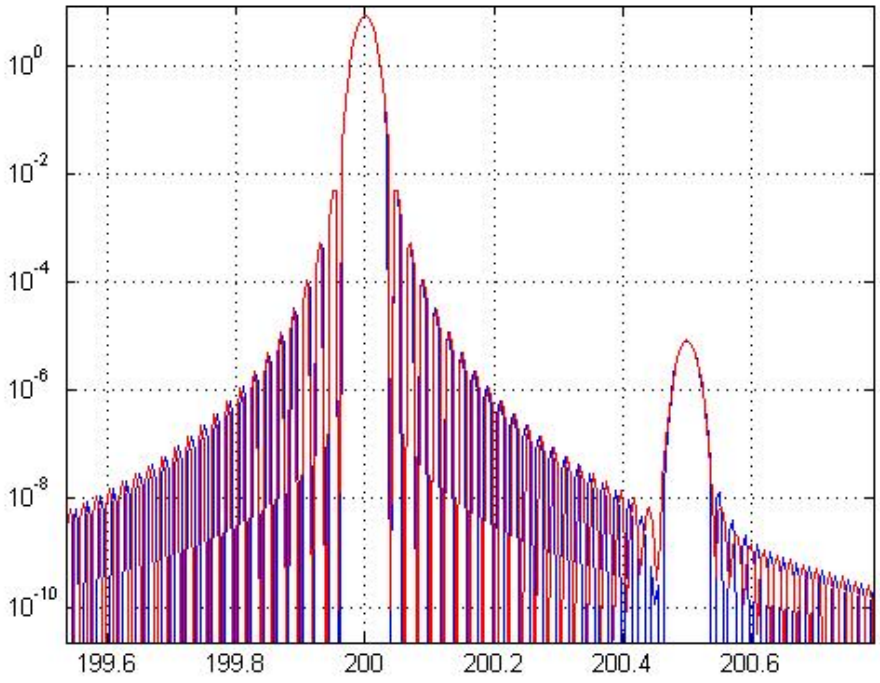


The superposition is perfect, so only the second signal appears.

Two near sinusoids with very different amplitudes

The two sinusoids are at 200 Hz and 200.5 Hz and the second has amplitude 0.001 of the other.

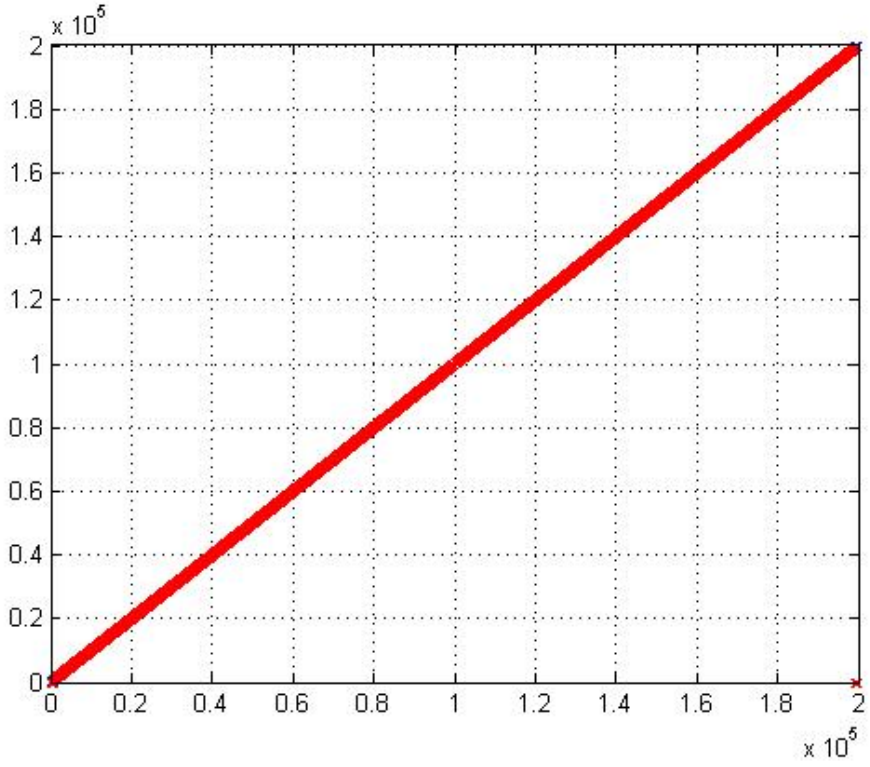
Here is the spectra of the signal at 20000 Hz (**blue**) and of the signal obtained by gd_decim at 4 kHz (**red**):



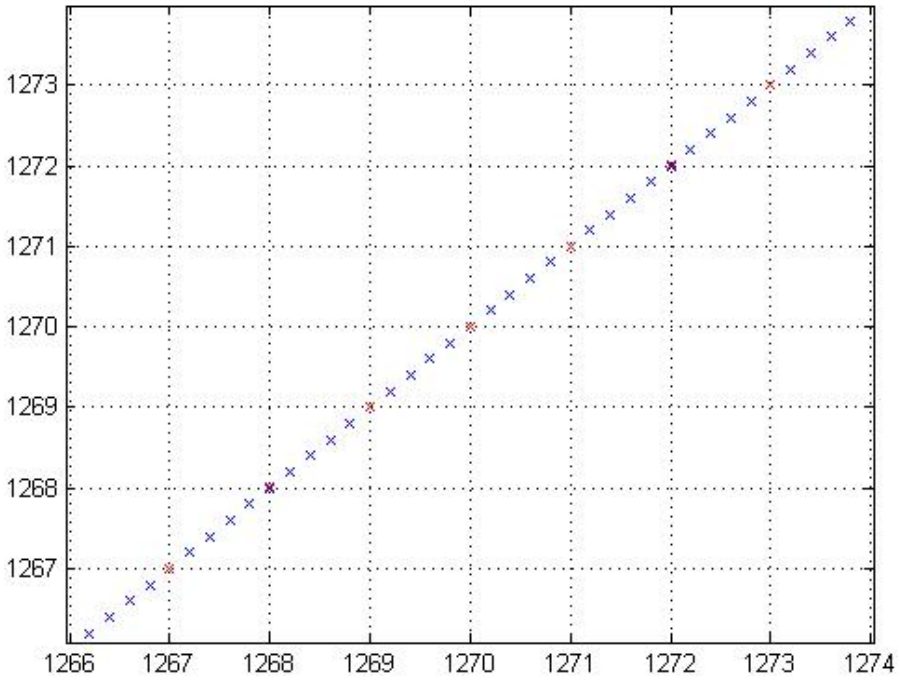
It should be clear that the limit is only due to the power spectrum estimation, not to the filter (the spectral estimation was done with a hanning window).

The ramp

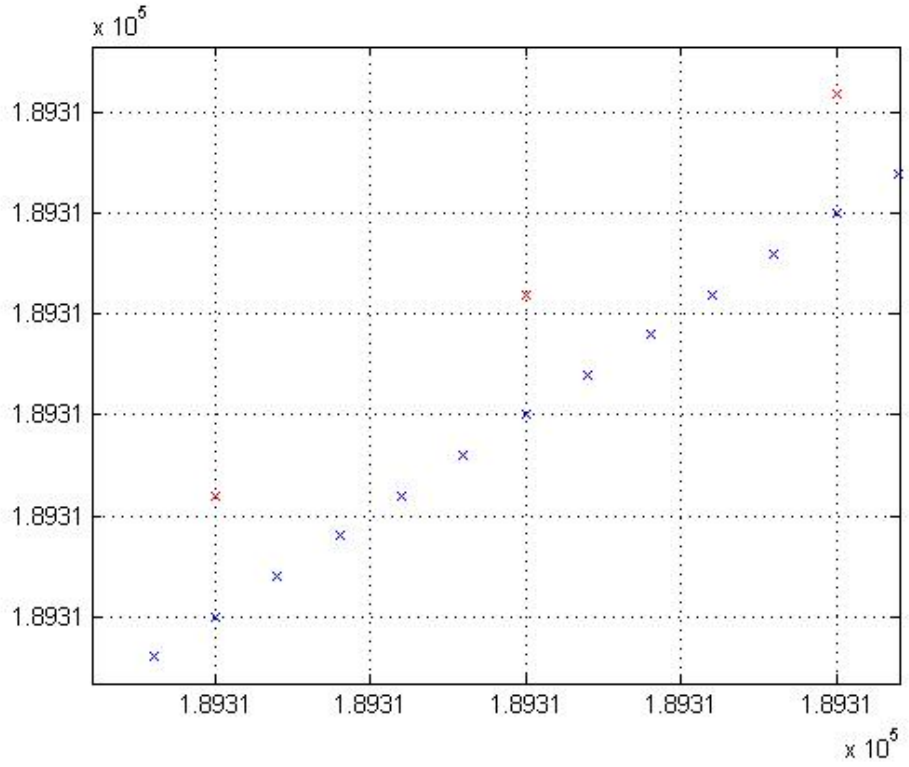
In order to check the continuity, consider a ramp signal sampled at 20000 kHz and apply our procedure:



(the thickness is due to the "x")



The tiny error on the filter amplitudes at 0 frequency gives a little error on the ramp samples for high times:



Even if this is not a practical problem for our gravitational signals, this point should be investigated.

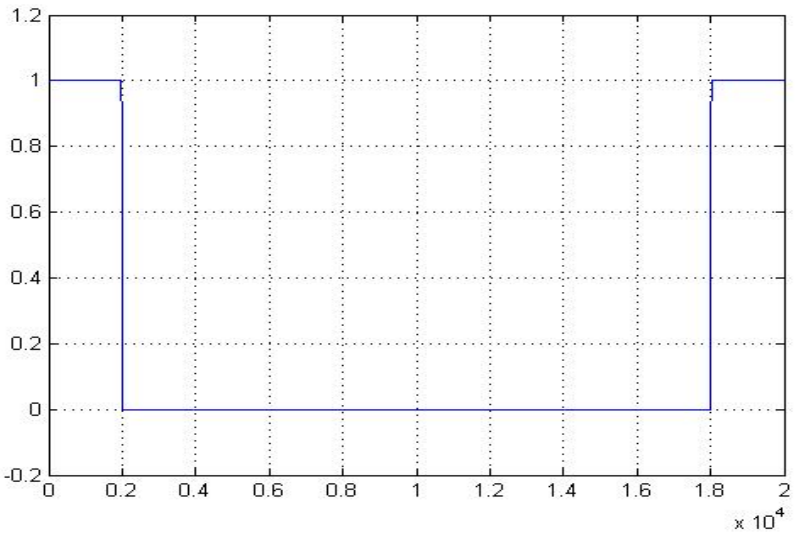
Appendix: filter responses

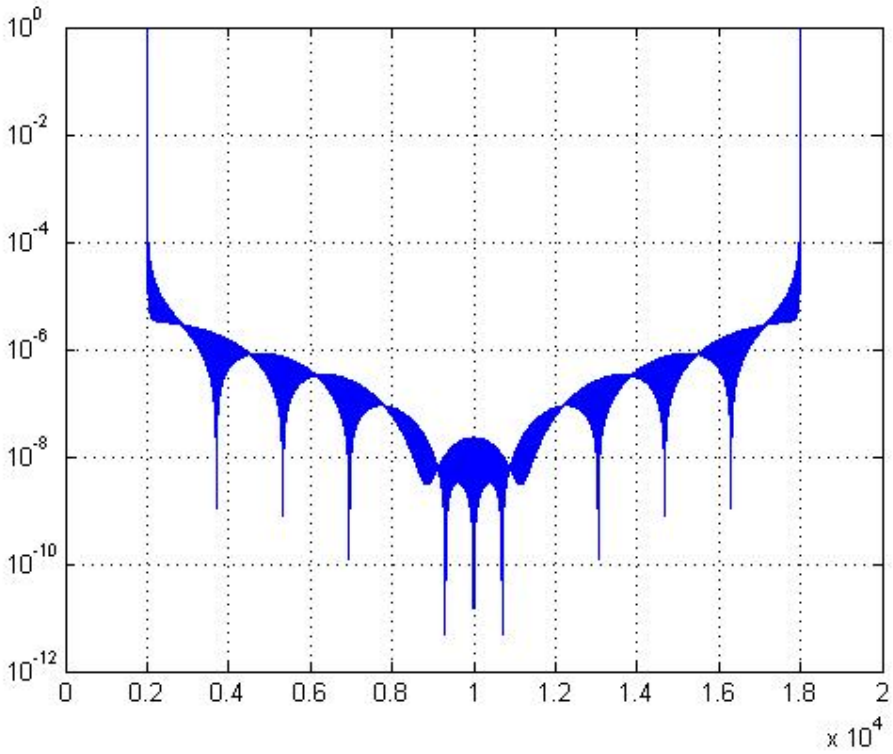
Decimation with frequency domain a-a filter

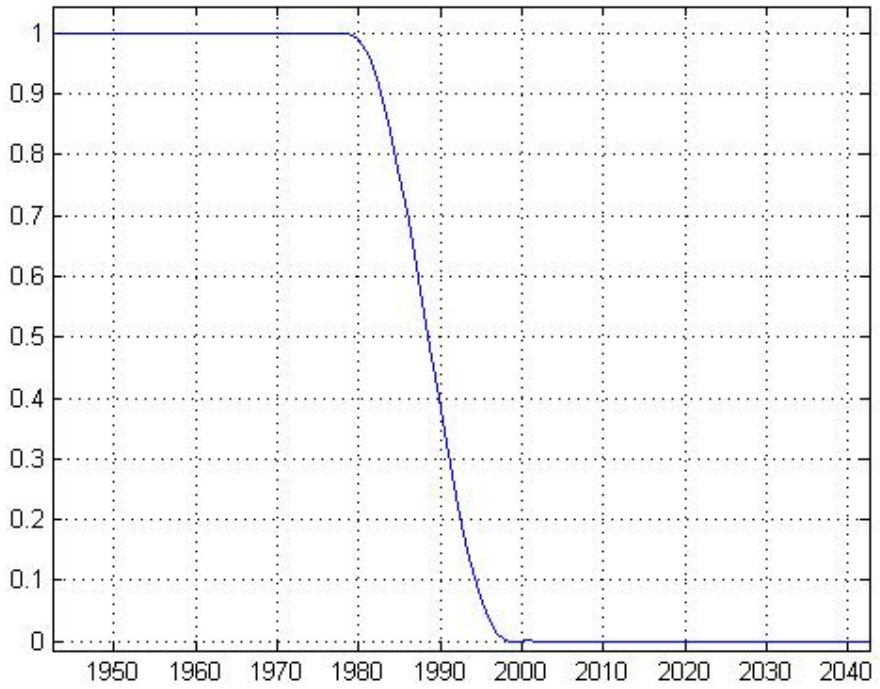
The Snag procedure, for GDs, is

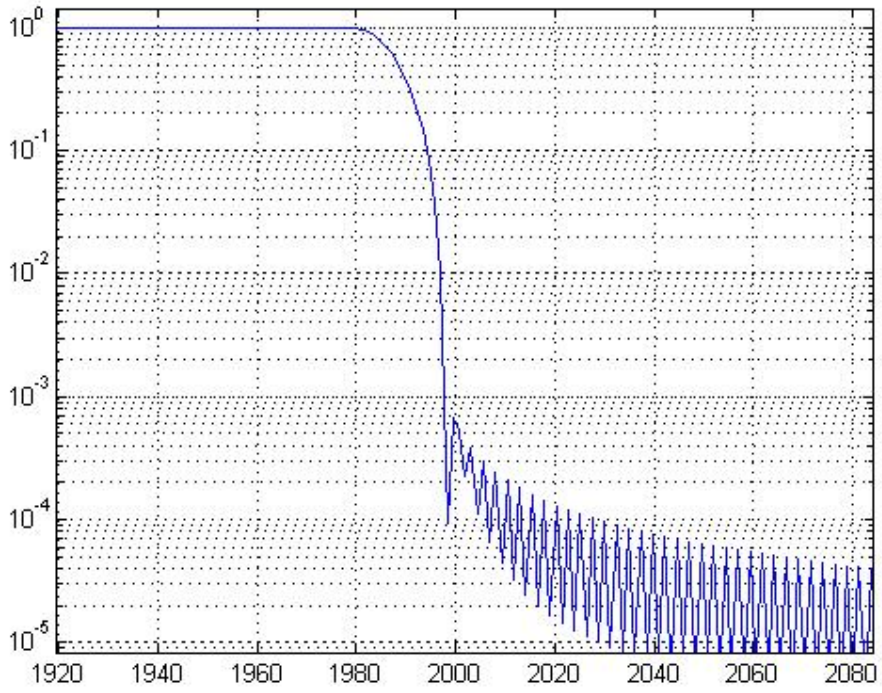
- **[gdout,frfilt]=gd_decim(gdin,decim,fftlen,icreal)**
  - **gdin** sampled data input GD or double array
  - **decim** decimation ratio; also non-integer (> 1; one every decim samples)
  - **fftlen** length of the fft (divisible by 4; 0 -> 16384)
  - i**creal** =1 impose real output
  - **gdout** output data

The frequency filter in output is given by (decim=5):

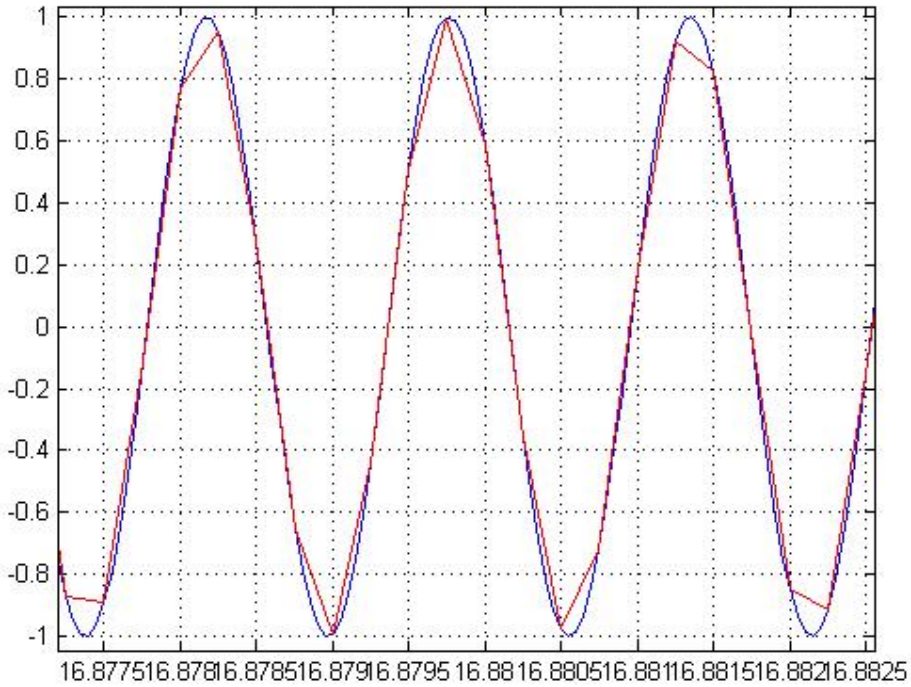



10^0
10^-2
10^-4
10^-6
10^-8
10^-10
10^-12
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
x 10^4

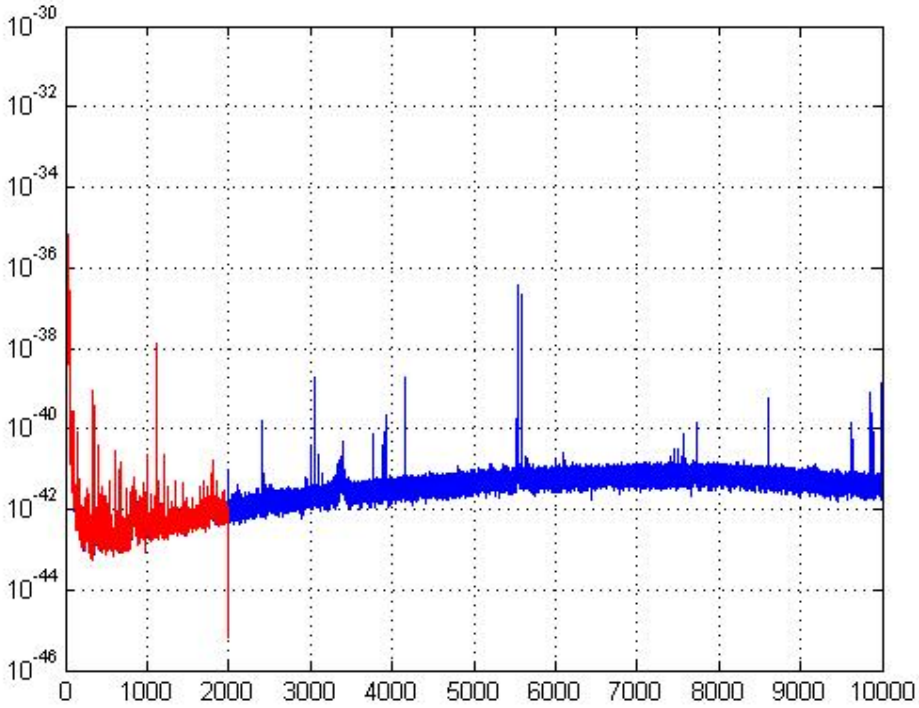

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1950
1960
1970
1980
1990
2000
2010
2020
2030
2040

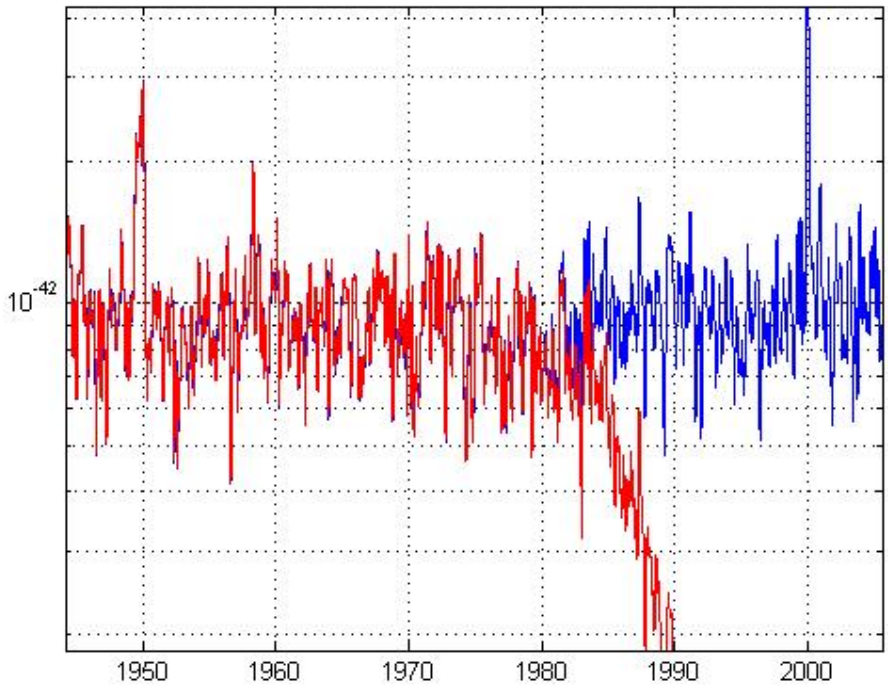


Here is the output to a 630 Hz sinusoid (blue the 20 kHz sampling, red the 4 kHz sampling); note the 0 phase and perfect match:

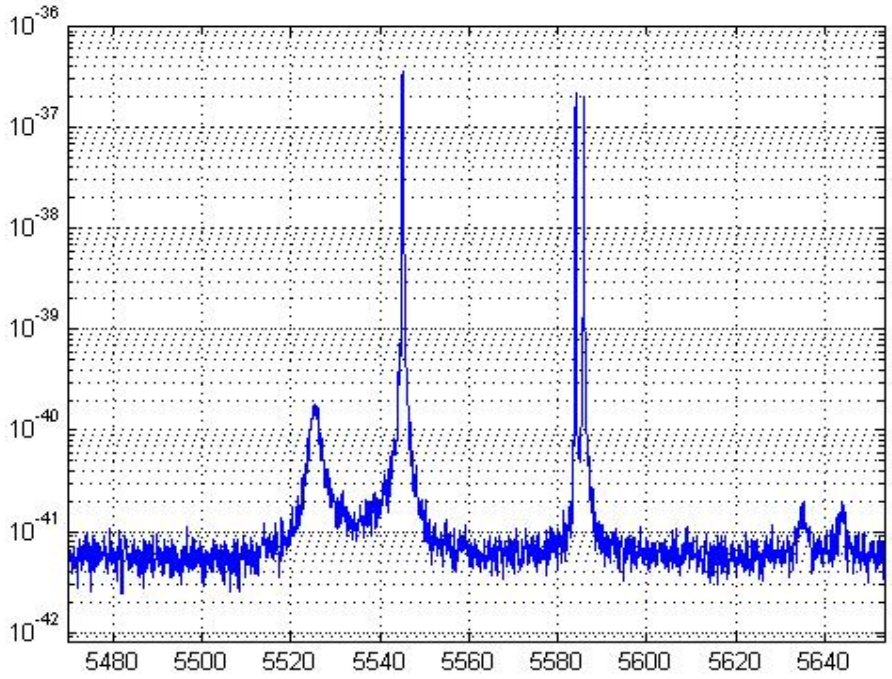


With the Virgo data:

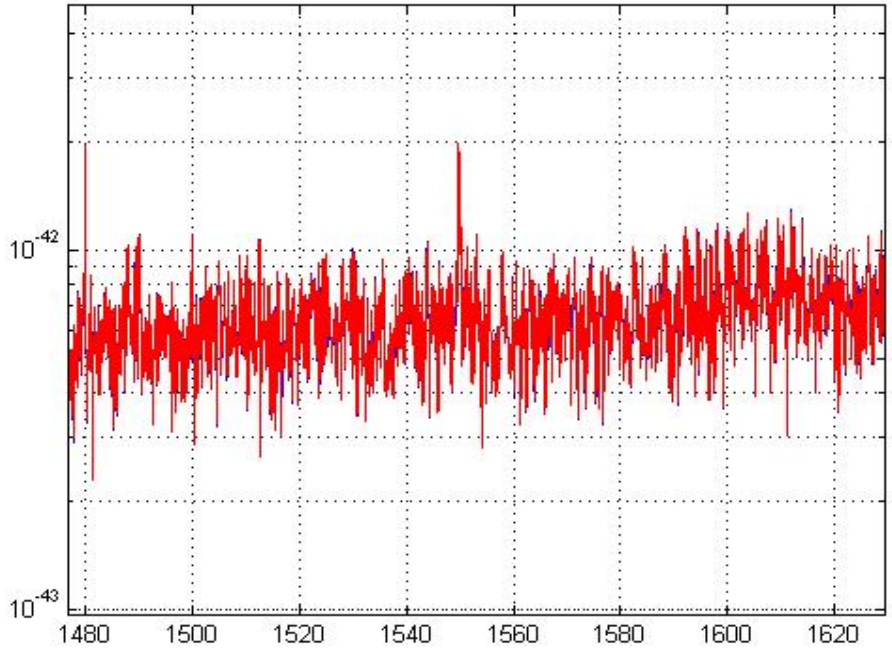




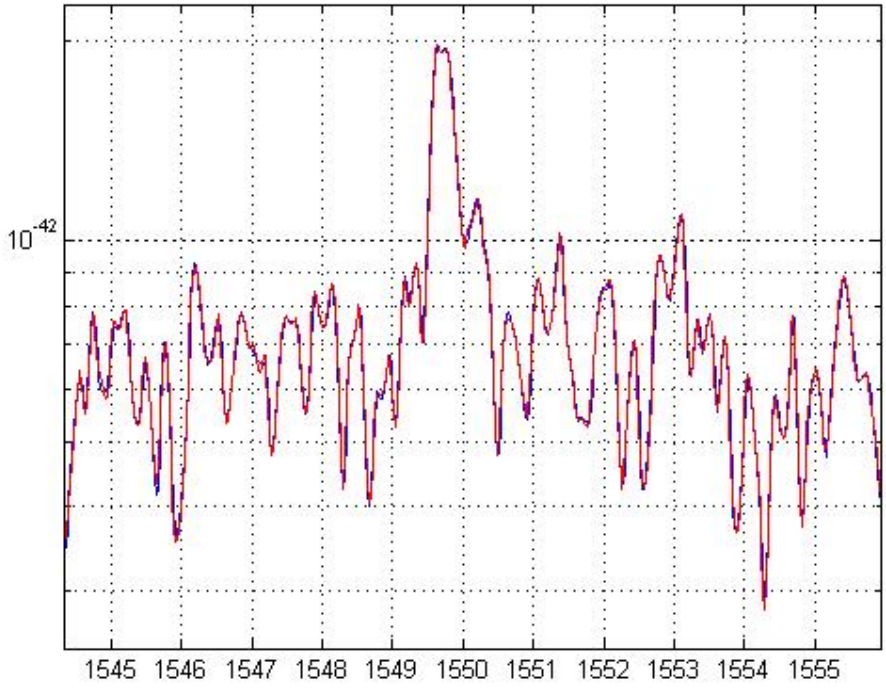
The big line:



The aliased region:



The peak at 1550 is not an aliased peak, as it is seen with a zoom of the superponed spectra:



Butterworth low-pass

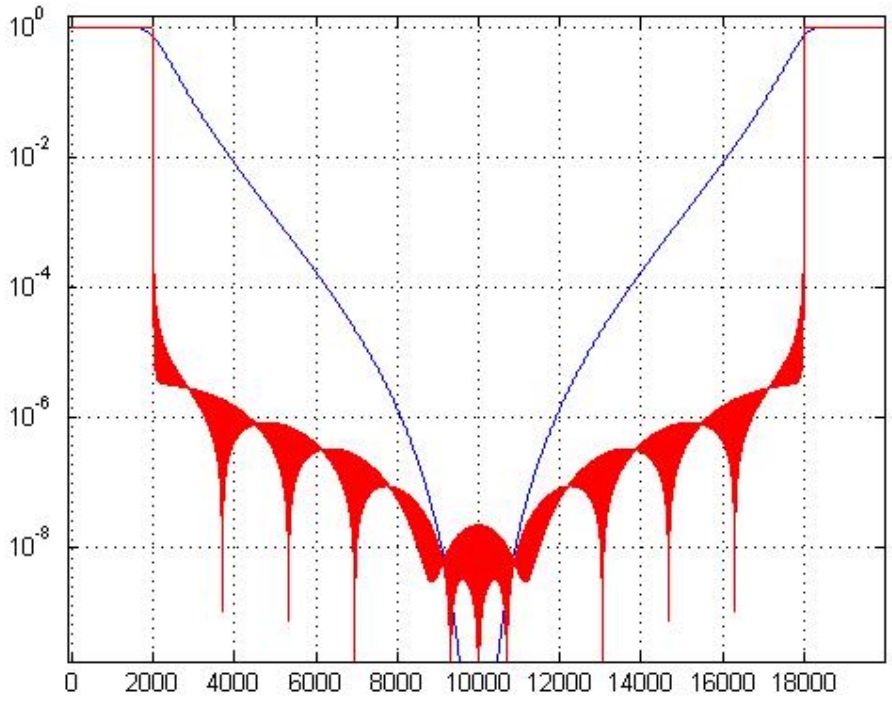
A 6-order low-pass Butterworth filter with cut at 0.2 (normalized frequency) is given by Matlab by:

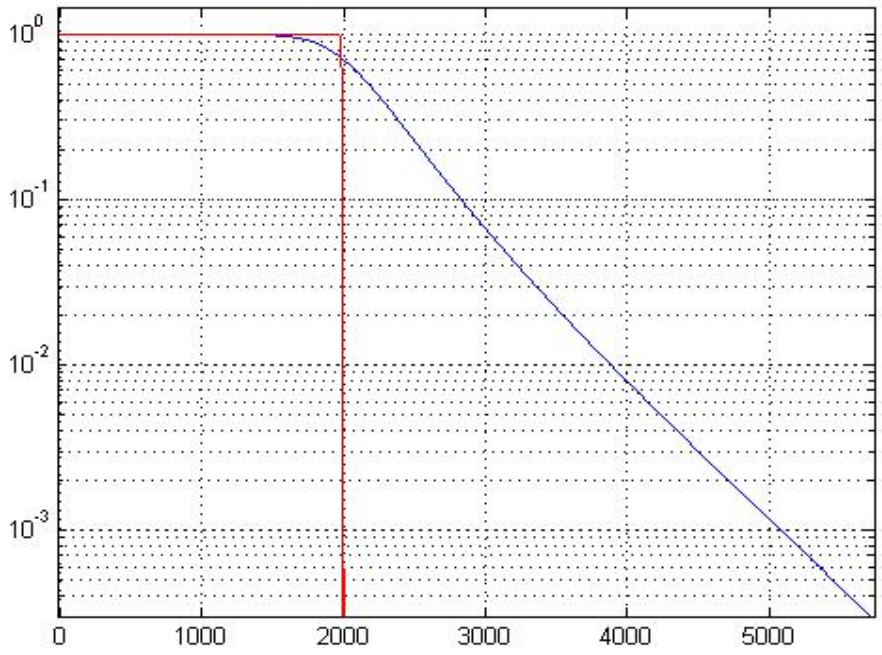
[b,a]=butter(6,0.2)

b = [0.0003 0.0020 0.0051 0.0068 0.0051 0.0020 0.0003]

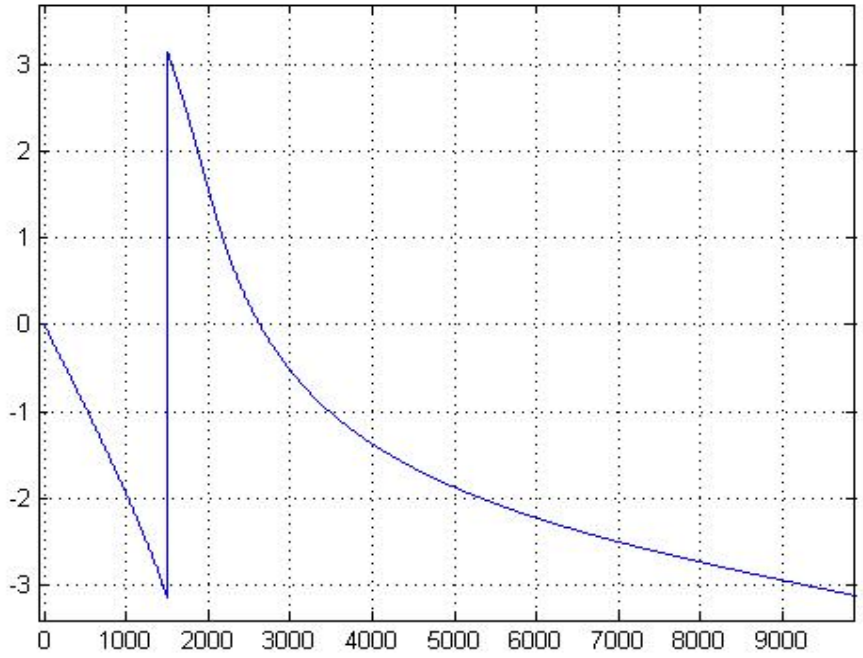
a = [1.0000 -3.5794 5.6587 -4.9654 2.5295 -0.7053 0.0838]

Here is the transfer function (together with the frequency domain a-a filter)





The frequency domain a-a filter is 0-phase, the phase of the Butterworth filter is



Use of the decimate Matlab procedure

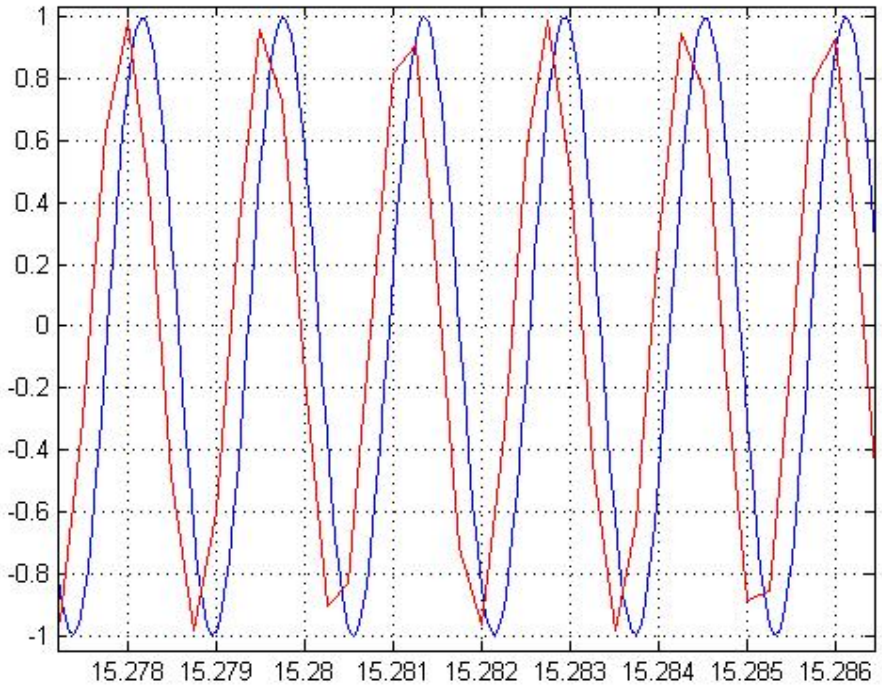
Let us see the effect of the Matlab (Signal Processing toolbox) **decimate** procedure, as proposed by F.F. In the default way, with a call like

>> y5=decimate(y1,5);

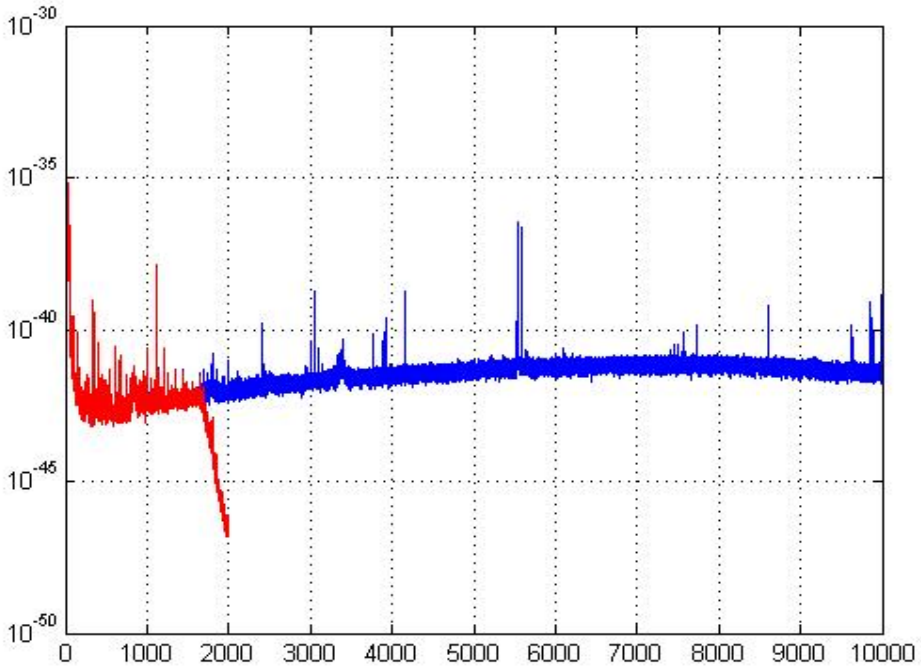
they say that

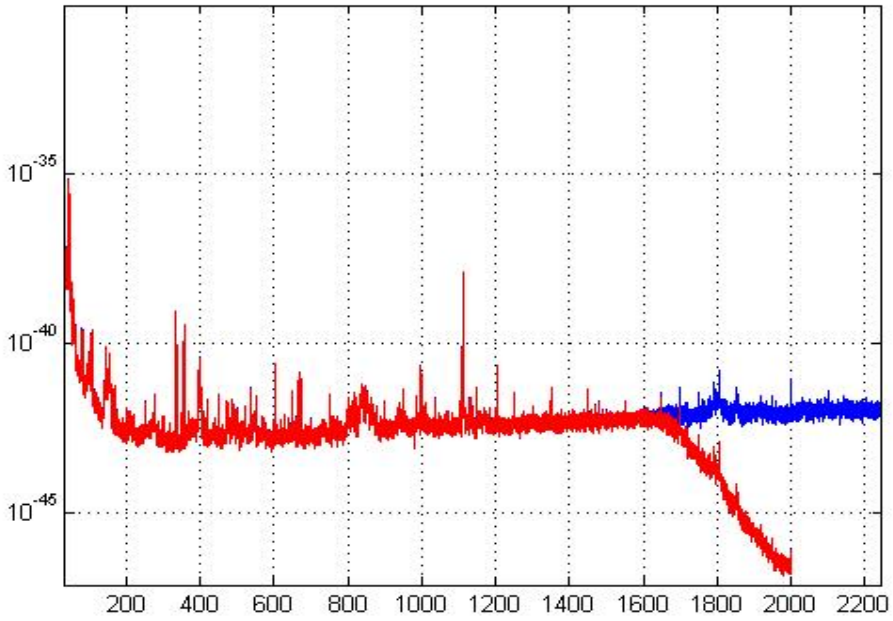
"By default, decimate employs an eighth-order lowpass Chebyshev Type I filter. It filters the input sequence in both the forward and reverse directions to remove all phase distortion, effectively doubling the filter order."

But in practice it seems not 0 phase; in fact this is the effect on a 630 Hz sinusoid:



a phase shift of roughly 40 degrees. If we apply it to the Virgo data, we have:



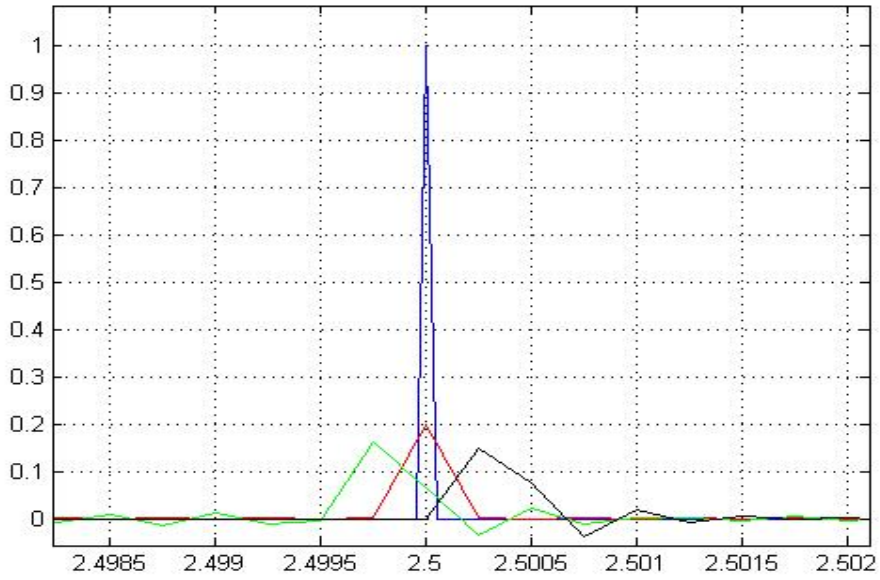


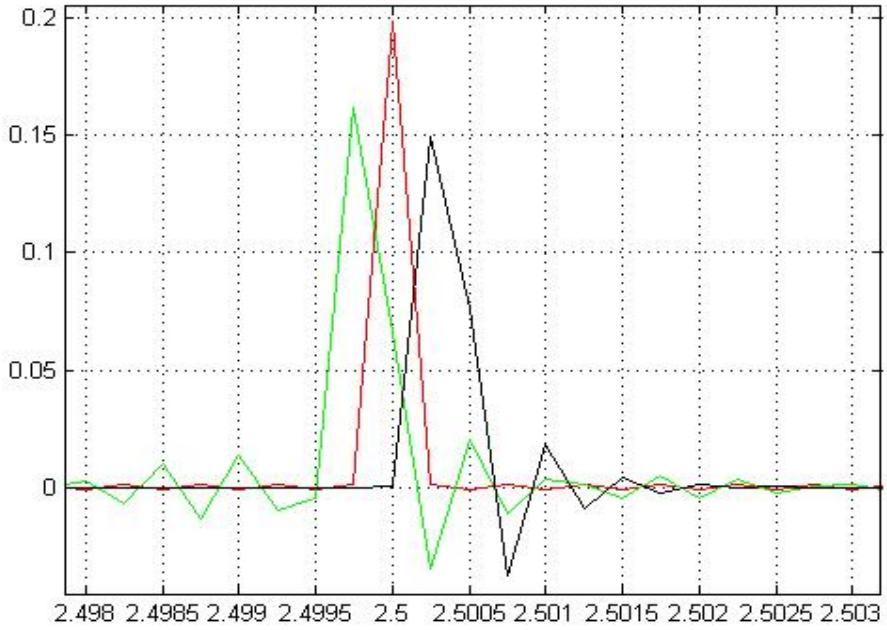
In the case of a stream of data, obviously the bi-directional filtering cannot be performed.
In any case about 500 Hz are lost.

Pulse response

Let us see the pulse response of the three procedures

- Butterworth 6th order (black)
- Matlab - Cebyshev 8th order bi-directional (green)
- Frequency filter (red)





Note the non-symmetric response of the Matlab decimation (that means no 0-phase).

Computer cost (computed with Matlab)

	parameters	time (s)	time (s)
		(for 2^22 samples)	(for 1 s of data)
Butterworth 6th	12	0.98	0.0047
decimate	16	2.89	0.0138
freq.dom. a-a	8132	4.00	0.0191

Appendix: the Matlab script

To run the script, Snag must installed.

```
% aa_analysis

nspet=1;
lfft=16384;
freq=640;
% freq=200;

% Benoit

g1=gd_sin('freq',freq,'len',1000000,'dt',0.00005);
s1=gd_pows(g1,'pieces',nspet,'resolution',4,'window',2,'short'
);
[g5,frfilt]=gd_decim(g1,5,lfft,1);

g5a=gd_sin('freq',freq,'len',200000,'dt',0.00025);

d5=g5-g5a;
dd5=y_gd(d5);
dd5=dd5(1601:198400);
dd5=gd(dd5);
dd5=edit_gd(dd5,'dx',0.00025,'ini',0.00025*1600);
sd5=gd_pows(dd5,'pieces',nspet,'resolution',4,'window',2,'shor
t');

gg5=y_gd(g5);
gg5=gg5(1601:198400);
gg5=gd(gg5);
gg5=edit_gd(gg5,'dx',0.00025,'ini',0.00025*1600);
ss5=gd_pows(gg5,'pieces',nspet,'resolution',4,'window',2,'shor
t');

gg5a=y_gd(g5a);
gg5a=gg5a(1601:198400);
gg5a=gd(gg5a);
gg5a=edit_gd(gg5a,'dx',0.00025,'ini',0.00025*1600);
ss5a=gd_pows(gg5a,'pieces',nspet,'resolution',4,'window',2,'sh
ort');

% Passuelo

freps=0.1;
% freps=0.5;
amp=1;
% amp=0.001;

b1=gd_sin('freq',freq+freps,'len',1000000,'dt',0.00005);
b1=amp*b1+g1;
sb1=gd_pows(b1,'pieces',nspet,'resolution',4,'window',2,'short
');
[b5,frfilt]=gd_decim(b1,5,lfft,1);
```

```
b5a=gd_sin('freq',freq+freps,'len',200000,'dt',0.00025);
b5a=b5a+g5a;

bb5=y_gd(b5);
bb5=bb5(1601:198400);
bb5=gd(bb5);
bb5=edit_gd(bb5,'dx',0.00025,'ini',0.00025*1600);
sb5=gd_pows(bb5,'pieces',1,'resolution',4,'window',2,'short');

bb5a=y_gd(b5a);
bb5a=bb5a(1601:198400);
bb5a=gd(bb5a);
bb5a=edit_gd(bb5a,'dx',0.00025,'ini',0.00025*1600);
sb5a=gd_pows(bb5a,'pieces',1,'resolution',4,'window',2,'short'
);

% ramp

r1=0:0.2:199999;
r1=gd(r1);
r1=edit_gd(r1,'dx',0.2);

[r5,frfilt]=gd_decim(r1,5,lfft,1);
r5=gd(r5);

r5a=0:199999;
r5a=gd(r5a);
```