A design method for very large FIR filters

G. Comoretto[1]

[1]INAF - Osservatorio Astrofisico di Arcetri

Arcetri Technical Report N° 2/2007

Abstract

Finite Impulse Response (FIR) filters are often used in digital signal processing. Commonly used methods for designing these filters in a general way fail to converge for filter sizes greater than a few hundred coefficients.

In this report we describe a method to compute large (up to several thousand elements) low-pass FIR filters, extending the frequency response of a smaller filter computed with the standard Remez method.

1 Introduction

One of the most commonly used elements in digital signal processing is a finite impulse response (FIR) filter. It is used for example to define the bandshape of digital receivers or in polyphase filterbanks and, with some post processing computations, as a prototype for Hilbert transform filters.

The filtering operation corresponds to a convolution in the time domain, with the filtered signal $y(t)$ related to the input signal $x(t)$ and to the filter coefficients $f(i)$ by the operation:

$$y(t) = \sum_{i=0}^{N-1} x(t - i\Delta t) f(i) \tag{1}$$

The filter response, in the frequency domain, is the Fourier transform of the filter coefficients (here $j = \sqrt{-1}$):

$$F(\nu) = \sum_{i=0}^{N-1} f(i) \exp\left(2\pi j \frac{i}{\Delta t}\right) \tag{2}$$

Note that $f(i)$ has discrete support (i.e. is nonzero only for integer values of i), while $F(\nu)$ is defined for any real ν, and is periodic with period $2\pi/\Delta t$. For real valued $f(i)$, $F(\nu)$ is hermitian, and needs only to be specified for positive frequencies $\nu \geq 0$. Usually the frequency ν is usually normalized to the period, and specified in the range $[-0.5 : 0.5]$.

A commonly used algorithm to compute $f(i)$ from a desired $F(i)$ is the Remez algorithm[1], and in particular its implementation due to Parks and McClellan[2]. Several implementations of this algorithm are available, e.g. in Matlab or as stand alone programs. We used a program written in C by Jake Janovetz, that allows to specify $F(\nu)$ over an arbitrary number of intervals in the range $[0 : 0.5]$, together with relative weights in each interval and the total number of filter coefficients. The resulting response is equiripple, with ripple amplitude inversely proportional to the weight of each interval. This algorithm works well for filters up to several hundreds of elements, but presents numerical instabilities for larger filters. In particular, when a very large number of oscillations is present, the algorithm converges to a false minimum of the solution, with all zero coefficients.

Many applications require a very large low-pass filter, of the order of several thousands of elements. The response is however often defined only in a small portion of the frequency range, at the lower end, and is identically zero everywhere else. For example a polyphase filterbank with 8 thousands of channels needs a low-pass filter with a cutoff normalized frequency of 1/16000, with at least 30000 elements. Even larger filterbanks are becoming common in large spectrometers.

A common technique to compute very large FIR filters consists in expressing $f(i)$ in a closed parametric form, that can be analytically Fourier transformed. Parameters are then adjusted in order to fit the desired transfer function $F(\nu)$. This approach is simple, fast, but has several limitations in the transfer functions that can be obtained. For example the truncated $\sin(x)/x$ gives large sidelobes just above the cutoff frequency, and the various cosine or polynomial tapering functions (Hanning, Hamming, etc) give a large tapering in the passband. It is difficult to obtaining a transfer function that is reasonably flat in the passband, and presents uniformly high rejection in the stopband.

In this report we describe a way to extend the Remez algorithm to very large FIR filters with very small cutoff frequency.

2 Algorithm description

Our problem is then to compute a FIR filter $f(i)$ with a very large number of coefficients and with a response that is identically zero for frequencies above $\nu_{max} \ll 1/2\Delta t$.

Our approach is to compute the filter coefficients for a filter $f_1(i)$ with a much higher time step $\Delta t_1 = k\Delta t$ with $k \gg 1$, providing that the period $1/\Delta t_1$ contains the passband and a sufficient portion of the stopband, $1/\Delta t_1 > \nu_{max}$. The response $F_1(\nu)$ is then extended numerically with an all zero response up to the actual limit frequency $1/\Delta t$, and transform back this response into a new filter. As f_1 requires a much lower number of taps, the Remez algorithm can be safely applied, and its advantages extended to

the desired filter f. This procedure is basically an efficient way to interpolate f_1 to a finer time resolution, and thus to a higher frequency.

A number of problems have been encountered and analyzed

- The original filter is symmetric with respect to its center. For an even number of coefficients, the center falls at mid-point between two coefficients. This corresponds to a phase slope in the $F(nu)$, that is different for the original and the extended filter. The phase slope can be easily inserted in the frequency domain, before the inverse Fourier transform operation.
- The filter function $f(i)$ is identically zero when it is not defined, i.e. has a strong discontinuity at the edges of its definition range, The two coefficients in the $f(i)$ at the two extreme values of i are modified by the Remez algorithm in order to minimize any ill effect of this discontinuity, and cannot be derived by just interpolating the adjacent ones. A standard fitting program must be used to adjust the coefficients at the extremes of the filter.

Then the algorithm is the following

- Define the filter for a time step that is a multiple (usually a power of 2) of the original one: $\Delta t_1 = k\Delta t$
- Apply the Remez algorithm on these specifications, deriving a filter function $f_1(i)$, $i = 0 \dots (N_1 - 1)$. with a fraction of the number of coefficients equal to $N_1 = N/k$,
- Save the extreme coefficients $f_1(0)$ and $f_1(N_1 - 1)$, and substitute them with the adjacent value $f_1(1)$ and $f_1(N_1 - 2)$
- Fourier transform $f(i)$ in $F(k)$ and apply a phase slope to account for the actual center of the filter.
- Extend the $F(k)$ with zeros up to the desired filter length N and transform back to $f(i)$
- Substitute back the saved values in $f(0)$ and $f(N - 1)$. This provides a reasonable first guess for the fitting program in the next step.
- Perform an optimization fit of at least k coefficients at the edges of $f(i)$.

The fitting program reads the desired frequency response (usually the same used for the Remez algorithm), the interpolated filter coefficients, and adjusts an user defined number of coefficients at the edges of the filter to minimize an error function of the form

$$E = \int_0^{0.5} |(F(\nu) - E(\nu)) * W(\nu)|^p \, d\nu \tag{3}$$

where $F(\nu)$ and $E(\nu)$ are the desired and actual frequency responses, $W(\nu)$ is a weighting function, that is zero for the regions where the response is unconstrained, and the exponent p controls the uniformity of the error. A value of $p = 12$ has been found to produce a reasonably equiripple response.

3 Results

In this chapter we give a number of filters computed with this method.

3.1 Variable decimation FIR filter

The filter is part of a variable decimation baseband converter. It is used to select a portion of the input band, that is subsequently decimated in frequency.

The filter has a bandpass that is a fraction b of the output decimated sample rate that, according to the Nyquist theorem, must be $b < 0.5$, and should provide good rejections of any signal above frequency $1 - b$ to avoid aliasing after the decimation. If the decimation (input to output sample rate) is d, the filter must have a passband from 0 to b/d, and a stopband starting at $(1 - b)/d$. Using coefficient recirculation,

it is possible to increase the filter size proportionally to d. In the filter described here, d varies from 2 to 256, and the number of coefficients is always equal to $32d$. The fractional bandpass is $b = 0.44$, i.e. 88% of the ideal value. With these values, the Remez algorithm computes a filter with passband ripple of 0.0035 dB peak-peak, and a stopband attenuation of 82 dB, for any value of the decimation.

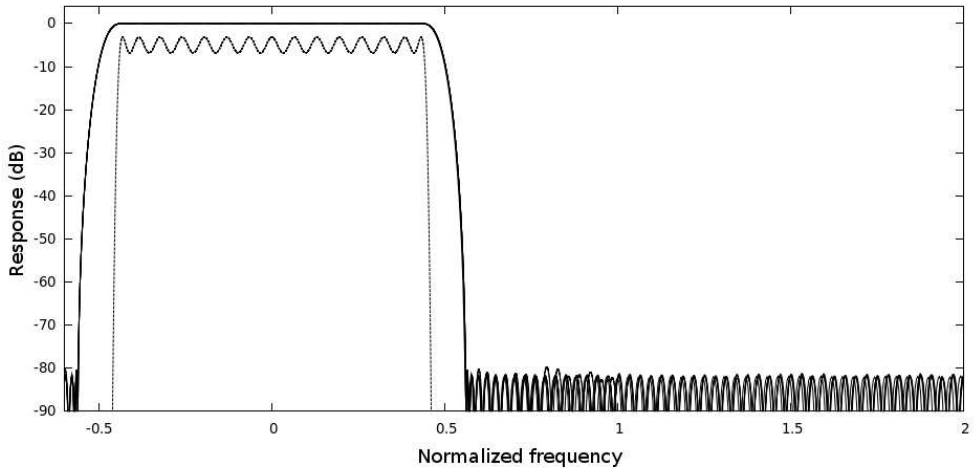

Figure 1: Response of a variable decimation FIR filter. Responses for decimations 2–256 have been shown, normalized by the output frequency and superimposed. Dashed curve is the passband, magnified by a factor 1000

The Remez algorithm converges up to a decimation factor of 8, with 256 coefficients in the filter. For higher decimations the method described in this report has been used, up th the largest filter with $d = 256$ and 8192 coefficients.

The response for all 8 filters, normalized to their output frequency, is shown in fig. 1. The details in the stopband are slightly different for different filters, but the overall attenuation, and the response in the passband and in the transition region are exactly superimposed.

Fitting 200 edge coefficients in the largest filter required about 2 hours of CPU time on a 64 bit, 2.4 GHz CPU.

3.2 16 channel polyphase filterbank

The filter is part of a 1/16 decimating polyphase filterbank. The attenuation has been chosen to be at least 100 dB, in any portion of the stopband that is aliased in the passband. The number of coefficients is 1024.

Relaxing the attenuation in those portions of the stopband that will be aliased to the filter transition region allows to increase filter performances in the relevant portions of the spectrum. This can be easily specified in the Remez algorithm, but not in a parametric, closed analytical form of the filter coefficients.

The Remez algorithm, again, fails to converge for more than 256 filter coefficients. A filter with 256 coefficients has been computed, and expanded by a factor of 4 in frequency. Up to 16 edge coefficients have been tuned by the optimization program, and the resulting filter has been shown in fig. 2. Coefficient fitting required 5 minutes of CPU time.

4 Program usage

A set of programs have been developed to perform the various steps of this method. They are listed here.

- `ruremez` Computes a filter giving a frequency specification. Output is a list of normalized tap coefficients

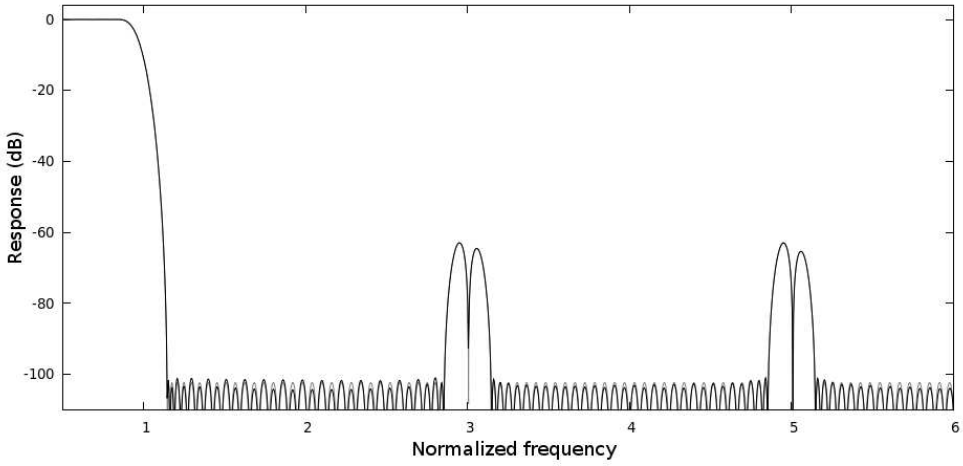

Figure 2: Response of a polyphase filterbank prototype filter. Gray: reduced size filter (1/8 the full size), black: full size extended filter

- `fft_ir` Computes the frequency response of a filter
- `truncate` Truncates the normalized coefficients of a filter to a given precision (number of bits). Used to prepare a table of integer coefficients to be used with integer arithmetic hardware
- `fft_extend` Extends a filter to a higher frequency range
- `minfir_data` Minimizes the edge coefficients of a filter, to fit a desired response. Very computing intensive, can be used only to twiddle a few filter coefficients.

4.1 Program runremez

The program runs the Remez algorithm. Input parameters are given as a textual file, on the standard input, and the filter coefficients are printed on the standard output.

Input lines beginning with `#` at the beginning of the file are considered comments and ignored. Parameters are given one per line.

- Number of filter coefficients. The number must be positive, greater than 11. The program considers both odd and even values. The maximum filter size for the current version is 8192, but any value greater than 200 may cause numerical problems
- Number of bands. Must be between 2 and 17
- Filter type. The program can compute a low-pass filter (type=1), a differentiator (type=2) or an Hilbert filter (type=3).
- Grid density. Is the number of points in the internal frequency grid for each filter coefficient. The specified value is currently ignored, and a fixed value of 16 is used instead
- For each desired frequency range the start and stop frequency, the desired amplitude and phase (ignored here), and the amplitude weight, must be specified, one line per frequency range.

The program prints a comment line with the number of iterations used, followed by the filter coefficients, as floating point values, one per line.

Example use: `runremez < filter.spec > filter.ir` File `filter.spec` contains:

```
# Example filter: 64 taps, 1/2 band low pass filter
# Usable bandpass: 88% of Nyquist band
#
  64
   2
   1
  16
0.0  0.22  1.0 1.0
0.28 0.5   0.0 25.
```

4.2 Program fft_ir

This program computes the response (the Fourier transform) of a filter specified as a list of real filter coefficients. It reads the coefficients from its standard input (so it can be used in a pipe with `runremez`), and produces on standard output a file with 5 columns. Columns represent the normalized phase (from 0 to 0.5), the response (amplitude and phase), its norm and its phase (in radians).

An optional parameter `-l length` specifies the FFT length. Default length is 8192 points, and must be a power of 2. $l/2+1$ lines are written on output, as the response is hermitian.

Example use: `fft_ir -l 65536 < filter.ir > filter.fft`

4.3 Program truncate

This program is used to truncate filter coefficients to a given number of bits.

Program use: `truncate [options] <input.ir > output.ir`

Program operates reading from standard input and writing to standard output. For example it can be used in a pipe together with `runremez`.

Parameters are:

- `-l <length>`: specifies filter length. Default is 2^16, must be specified for larger filters.
- `-b <bits>`: specifies the bits used in the coefficient representation mantissa. As usually filter coefficients are signed, the value specified must be one less than the number of bits in the signed value.
- `-m <max>`: Maximum value expected. If not specified, uses maximum coefficient in the input filter. Must be used when different filters must be quantized together, with the same scale.
- `-i` Use integer output coefficients. If not specified, output coefficients have the same normalization of the input ones. i.e. the response computed by `fft_ir` will have the same amplitude scale. If specified, the coefficients will be specified as integers, in the range $\pm 2^b - 1$.

4.4 Program minfir_data

The program accepts a filter definition file, a file with a set of filter coefficients, and adjusts the extreme coefficients in order to optimize the filter response.

Program use: `minfir_data <parameter file> <input> <output>`

Note that this program requires the files to be given as parameters, and not as pipes connected to standard input/output.

The parameter file has the same structure and format of that used for the program `runremez`, but instead of the filter type, in the third line, the number of coefficents to optimize at each end is specified. Frequencies in the filter response are expressed in terms of the final filter sampling rate, so if this file is derived from the one used to compute the reduced filter $f_1(i)$, all frequencies must be divided by k.

The program prints periodically the error function defined in eq 3. Every 10000 iterations of the optimization process, the output file is updated with the current version of the optimized filter, so if the optimization process converges very slowly, but does not stop, it is possible to inspect the partial result and stop the optimization if this is reasonable.

References

[1] W. Fraser "A Survey of Methods of Computing Minimax and Near-Minimax Polynomial Approximations for Functions of a single Independent Variable". **J. ACM** 12: *295* (1965)

[2] Parks, McClellan "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase". **IEEE Trans. on Circuit Theory**, 19(2), *189* (1972)

[3] Wikipedia entry of the Parks-McClellan algorithm
`http://en.wikipedia.org/wiki/Parks-McClellan_filter_design_algorithm`

[4] G. Comoretto, A. Russo, G. Tuccari, A. Baudry, P. Camino, B. Quertier "Uniboard Digital Receiver Design Document"; Arcetri Tech. Rep. n. 5/2011

Contents