The 36th Asilomar Conference on Signals, Systems, and Computers
Pacific Grove CA, USA, Nov. 3–6, 2002

A Specification Language for the Optimal Design of Exotic FIR Filters with Second-Order Cone Programs

Jeffrey O. Coleman — http://alum.mit.edu/www/jeffc
Dan P. Scholnik — *scholnik@nrl.navy.mil*
Josef J. Brandriss — *brandriss@radar.nrl.navy.mil*

Naval Research Laboratory, Radar Division
Signal Processing Theory & Methods Section
Washington, DC

Abstract

Application-tailored individual and joint FIR-filter designs of remarkable complexity are elegantly coded using our MATLAB *toolbox* Opt*, a research tool providing a DSP-oriented modeling language for driving ultra-efficient off-the-shelf numerical solvers of (linear and) second-order cone programs.* Opt *data types symbolically capture affine or (nonnegative definite) quadratic dependencies on optimization variables, which gain numeric values only later, when optimized. On those basic types it builds affine vector and complex-time-sequence types for specifying impulse-response structures in 1D or multi-D, with sample spacing either uniform or not. Dependencies can be manipulated symbolically with arithmetic and DSP operations including convolution, filter match, and Fourier transform. Linear and MS errors in frequency and time domains can be constructed, constrained, and optimized. MSE constructions include output powers of filter systems driven by symbolic random-process drive signals having user-specified PSDs.*

1: Introduction

For fourteen years we have designed all manner of FIR filters with our own special-purpose optimization languages. Experience with the best early one, which translated filter specifications in a flexible algebraic notation into linear programs (LPs) for solution [1], showed (1) the power of a flexible notation for constraint construction, (2) the importance of minimizing the learning curve between potential users and productive system use, and (3) the extraordinary importance in filter design of positive-definite quadratic constraints, which are not available in LPs.

Those lessons led in time to the Opt MATLAB toolbox for FIR-filter design presented here. Opt maps filter-design specifications in the form of linear and nonnegative-definite (NND) quadratic constraints into second-order cone programs (SOCPs) [2] for solution by any of several extraordinarily efficient, off-the-shelf numerical-optimization codes.

```
InitOpt ;
X = newOptSpace ;
```

This *SOCP Specification* defines impulse response h in terms of the variables in X and defines *constraints* and an *objective* in terms of h

```
soln = minimize( objective, constraints, X, 'sedumi' ) ;

% evaluate h at solution point soln
hopt = optimal( h, soln ) ;
```

Figure 1: In filter design in MATLAB/Opt, typically newOptSpace creates a set of variables to be optimized by minimize to minimize an *objective* subject to *constraints* specifying requirements on an impulse response (here h) constructed to reflect filter structure.

A MATLAB session for Opt filter design typically takes the form in Fig. 1. Once InitOpt has been called, once per MATLAB session, a "null SOCP" can be created at any time by newOptSpace, which returns a handle to a new set of optimization variables, an optimization space. A solution point in that space can be obtained at any time thereafter by passing its handle to minimize with an *objective*, a list of *constraints*, and a solver selection. Impulse-response variable h in Fig. 1 is an example of an Opt quantity that does not store numerical values but instead stores relationships to optimization variables. Numerical values are substituted into those relationships only when optimal evaluates the relationships at a solution point. This is the heart of Opt:

> *Opt permits meaningful algebraic operations on quantities that, because optimization has not yet taken place, have no numeric values.*

Work supported by the Office of Naval Research, Arlington VA, USA.

Elementary methods of *SOCP specification* are discussed in the next section. Section 3 then presents substantial examples that show the power of the approach.

The last argument to minimize selects a solver. Here we use 'sedumi' (fewcal.kub.nl/sturm/software/sedumi.html), a high-performance noncommercial (free) numeric solver for SOCPs. Other options include the remarkably fast 'mosek' (www.mosek.com) and 'loqosoclp', an SOCP/LP interface to the LOQO (www.princeton.edu/~rvdb), which accepts a callback function for plotting intermediate results.

2: *SOCP Specification*

The SOCP specification in Fig. 1 generally takes this form:

specification of an *impulse response*
specification of the *constraints*
specification of the *objective*

The *objective* is often simple and just defined in minimize's argument list. This is occasionally true of the *constraints* also. But these elements are key, wherever they are placed.

The SOCP *objective* and *constraints* in Fig. 1 are generally functions of the *impulse response* to be optimized, so we first discuss the incorporation of a filter's fundamental structure into the definition of an *impulse response* in Opt.

2.1: Specification of an *impulse response*

Here Opt variables become *impulse response*s that relate to the optimization variables in set X as shown in Figure 2.

Opt supports a finite-length-sequence data type incorporating both values and times of samples. Opt code

```
w = optSequence( vect ) ;
```

associates times $0 \ldots$ (length(vect) $-$ 1) with the elements of MATLAB vector vect to make w in effect a *fixed* (nonoptimizable) impulse response. Alternate form

```
u = optSequence( N, X ) ;
```

creates a sequence with samples $0 \ldots \mathrm{N}-1$ associated with real optimization variables in set X. Function optSequence selects *virgin* optimization variables, those not previously in such relationships with user variables. Here u is the *optimizable* impulse response of a real nonlinear-phase filter. Such sequences also represent uniformly spaced impulse trains in continuous time; other optSequence forms place impulses nonuniformly and in multiple dimensions.

Overloaded MATLAB operators allow construction of structured impulse responses. Prime yields a time-reversed conjugate (match) and simplifies imposition of linear phase:

```
u = optSequence( N, X ) ;
v = u + u' ;
```

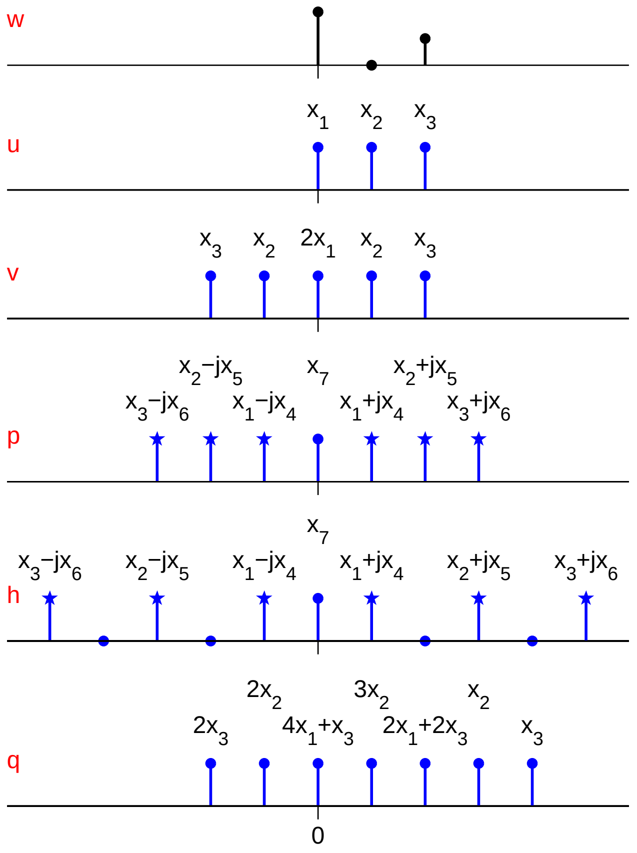


Figure 2: These outputs of MATLAB/Opt commands plot(w, 'label'); plot(u, 'label'); ... **relate example Opt sequences from Section 2.1 to real optimization variables** $x_1, x_2, \ldots \in$ X**. Here** vect = [2 0 1] **and** N = 3**.**

Here v is a real impulse response with linear phase and support on $-(\mathrm{N}-1), \ldots, \mathrm{N}-1$. The creation instead of a complex linear-phase impulse response is straightforward using Opt delay operator "|" and simple arithmetic operations:

```
a = optSequence( N, X ) ;
b = optSequence( N, X ) ;
r = ( a + j * b ) | 1 ;
p = r + optSequence( 1, X ) + r' ;
```

Here p depends on $2N+1$ distinct optimization variables, as each is virgin when optSequence enters it into relationship. The center sample is real, as required, by construction; canceling imaginary parts in conjugate summing instead would have left a nonvirgin optimization variable bereft of relationship, a *widow*, a situation best avoided.

Opt's time-axis scale-up operator "./" here makes a complex linear-phase impulse response be halfband also:

```
u = optSequence( N, X ) ;
v = optSequence( N, X ) ;
c = optSequence( 1, X ) ;
r = ( ( u + j * v ) ./ 2 ) | 1 ;
h = r' + c + r ;
```

(Unsurprisingly, operator “ . \ ” would perform decimation.)

A convolution operator allows certain cascades to be defined. If two filters have impulse responses p and w then

```
q = p .* w ;
```

makes q the impulse response of their cascade. There is an important restriction, however. Every sample in an Opt sequence must depend only affinely (linearly plus a constant) on optimization variables. To avoid disallowed quadratic dependencies, convolution and multiplication must have at least one “fixed” or nonoptimizable argument, one that, like the fixed sequence w defined earlier, is not dependent on optimization variables. These optimizable-nonoptimizable cascades are used when designing a filter to meet specifications on a cascade of which it is a member.

The MATLAB data structures of Opt sequence variables w . . . q above, as well as those for nonuniformly spaced and multidimensional impulse trains, actually each contain: (1) a specification of the times on which the sequence is supported and (2) coefficients specifying the affine relationship of each sample or impulse area to the optimization variables. Details for the one-dimensional discrete-time case are presented [3] and summarized [4] elsewhere (using the slightly different notation of earlier software).

2.2: Construction of Error Measures

2.2.1: Frequency-Domain Grids Ref. [5] discusses the use of second-order cone constraints, one for each frequency in a grid stretched across a band of interest, to construct measures of frequency-domain errors in the Chebychev or minimax or L_∞ sense, all equivalent in this context, in the L_2 or MSE (mean squared error) sense, and in the L_1 (mean absolute error) sense. Those techniques begin here:

```
 f = linspace( f1, f2, ( f2 − f1 ) ∗ 20 ∗ length( h ) ) ;
H = fourier( h, f ) ;
```

If h is an impulse response, H is the corresponding frequency response “evaluated” on the MATLAB vector f of frequencies. Evaluation is with respect to frequency only; if h is optimizable then H is also—its samples remain indeterminant because they depend on impulse-response coefficients that as yet have no numeric values. Here MATLAB’s linspace function creates a vector with elements stretching between its first two arguments with length specified by the third, here set high enough for typical FIR design work. Factor 20 might be as low as 5 for quick experimentation or as high as 50 or more when the Fourier samples are to be used for precise approximation of an L_1 norm [5].

If h is linear-phase so that H is real by construction,

```
C = { −10 ^ (−55/20) < real( H ), . . .
                              real( H ) < 10 ^ (−55/20) } ;
```

sets C to a list containing two linear constraints for each frequency sample to bound H between $-\epsilon$ and ϵ, where $20 \log_{10} \epsilon = -55$ dB. Taking the real part eliminates computational noise in the zero imaginary components in order that “<” see real arguments as required. If h is instead nonlinear-phase, so that H is genuinely complex,

```
d = optVar( X ) ;
C = abs( H ) < d ;
```

sets C to a list of second-order cone constraints, one for each sample in H, that upper bound the magnitude response by d, tied here to a virgin optimization variable. A constant bound would certainly be valid, but here variable d can be passed as the *objective* to minimize in order to minimize the peak complex magnitude of the samples in H. (This form using abs(·) would also work for the linear-phase case, but a degenerate second-order cone would be used at each frequency instead of a linear-constraint pair as before.)

The forms just presented typically specify stopbands but apply to passbands as well if, when C is defined, H is replaced with (1 − H) to specify a desired complex passband gain of unity. Or suppose some known filter is represented by MATLAB vector G containing its complex frequency response on a grid f of passband frequencies, and suppose fixed filter G is to be cascaded with the filter H to be optimized. To minimize the MS passband error between this cascade and a delay of T samples (possibly nonintegral):

```
 H   = fourier( h, f ) ;
 d   = optVar( X ) ;
 C   = sum( abs( H ∗ G − exp( − j ∗ 2 ∗ pi ∗ f ∗ T ) ) .^ 2 ) . . .
         < d .^ 2 ;
soln = minimize( d, { C, OtherConstraints }, X, 'sedumi' ) ;
```

Before, “<” related expressions affine in the optimization variables of set X, but here the left side is quadratic in X (possibly with linear and constant terms) and NND (*non-negative, definite*ly), making C an NND quadratic constraint.

2.2.2: Random Processes as Drive Signals DSP design often requires error measures of the form

$$\mathrm{MSE} = \int |H(f) - D(f)|^2\, W(f)\, df, \qquad (1)$$

where $H(f)$ is to approximate, with error weighting $W(f)$, some desired function $D(f)$. If $\int W(f)\, df = 1$ with $W(f)$ taking only values $\{0, \alpha\}$ for some α, then this MSE is the mean squared error (MSE) between $H(f)$ and $D(f)$ in the support of $W(f)$. But if $W(f)$ is the power spectral density (PSD) of a zero-mean random process driving filters $H(f)$ and $D(f)$, the integral is also the average power in the output-error signal, the difference between the filter outputs. An MSE specified as an error power can be derived by Opt automatically and to machine precision.

Here is a zero-mean, discrete-time Opt random process:

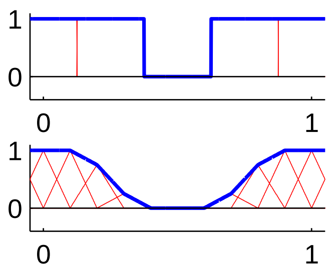


Figure 3: Opt random-process spectra. Above: the 'Box' **basis, one-period weights** [1 1 0 1]**. Below: linear interpolation between one-period weights** [4 4 3 1 0 0 0 1 3 4] /4 **using the** 'Triangle' **basis.**

```
s = Process( 'Box', 0, [ 1 1 0 1 ] ) ;
```

The three arguments to Process jointly specify the PSD of s. Call them Φ, Δ, and $\mathbf{w}$. Here *mother function* $\Phi(f)$ is a 'Box', frequency offset $\Delta = 0$, and weight vector $\mathbf{w} = $[1 1 0 1] with length (call it) $M = 4$, giving s the PSD at the top of Fig. 3. The significance of $\mathbf{w}$ may well be apparent, but formally Opt constructs the required periodic PSD in steps:

1. Weight vector $\mathbf{w}$ is extended periodicially to create a doubly infinite sequence $w_k \triangleq \mathbf{w}_{1+(k \bmod M)}$,
2. the infinite weight sequence is applied to integer shifts of the supplied mother function $\Phi(f)$, creating a function $\sum_k w_k \Phi(f-k)$ with period M,
3. this function is offset upward in frequency by Δ, yielding $\sum_k w_k \Phi(f - \Delta - k)$, and finally
4. scaling the offset function down in frequency by M sets the period in normalized frequency f to unity:
$$\mathrm{PSD} = \sum_k w_k \Phi(Mf - \Delta - k).$$

The mother 'Box' function has unit width, so the offset, scaled-down copies touch without overlap to create an ideal brick-wall spectrum. To instead create a spectrum interpolated linearly between specified samples, use the symmetric-triangle mother function supported on $[-1, 1]$:

```
r = Process( 'Triangle', 0, [ 1 1 3/4 1/4 0 0 0 1/4 3/4 1 ] ) ;
```

The PSD of r in Fig. 3 approximates a 60% raised-cosine spectrum at an oversampling rate of two. A longer weight vector would improve the approximation.

Opt random processes can be added, subtracted, scaled, and convolved with impulse responses and can have their average powers obtained. Here pwr(s) and pwr(r) return 0.75 and 0.5 respectively, and if both h and d are fixed,

```
p = pwr( r .∗ ( h − d ) ) ;
```

sets p to the numeric MSE in (1) with $W(f)$ the PSD of r. There are no simulations or numerical integrals; calculation to machine precision is possible because available mother functions exist internal to Opt as Fourier pairs, and the required integration is actually an inverse Fourier transform.

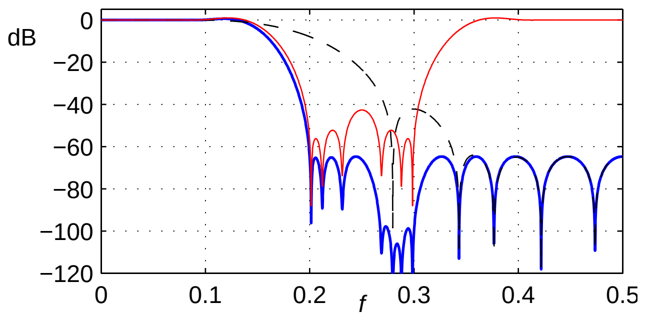
An h optimizable over a set X of optimization variables is more interesting. As (1) is quadratic in h, which is affine

```
InitOpt ;
Xg = newOptSpace ;
Xh = newOptSpace ;

Ng = 8 ; Nh = 15 ;

% interpolation filter g
u = optSequence( Ng, Xg ) ;
g = u ' + u ;

% shaping filter h
h =  optSequence( Nh, Xh ) . / 2 ;

% cascade should look like delay to this passband signal
sig = Process( 'Box', 0, [ 1 0 0 0 0 ] ) ;
T = 6 ; delay = optSequence( [1] ) | T ;    % desired delay

% grids for stopband
df = 1 / ( 40 ∗ ( Ng + Nh ) ) ;
f = 0.2 : df : 0.5 ;    % of cascade
fh = 0.2 : df : 0.3 ;    % of shaping filter

epsg = optVar( Xg ) ;
epsh = optVar( Xh ) ;
gopt = optSequence( [1] ) ;
for n = 1 : 4    % enough for convergence
  % optimize shaping filter h
  MSE = pwr( sig . ∗ gopt . ∗ h − sig . ∗ delay ) / pwr( sig ) ;
  gain = fourier( gopt, fh ) ∗ fourier( h, fh ) ;
  optXh  = minimize( epsh, . . .
       { abs( gain ) < epsh, MSE < 1e−5 }, Xh, 'sedumi' ) ;
  hopt = optimal( h, optXh ) ;

  % optimize interpolation filter g
  MSE = pwr(  sig . ∗ g . ∗ hopt − sig . ∗ delay. . .
            ) / pwr( sig ) ;
  gain = fourier( g, f ) ∗ fourier( hopt, f ) ;
  optXg  = minimize( epsg, . . .
       { abs( gain ) < epsg, MSE < 1e−5 }, Xg, 'sedumi' ) ;
  gopt = optimal( g, optXg ) ;
end ;
```

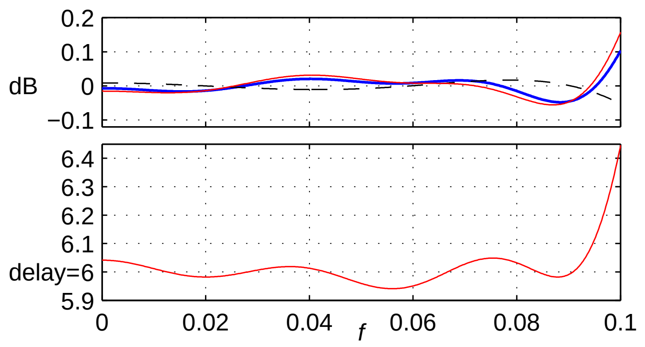
Figure 4: The interpolation and shaping filters of an IFIR cascade are given linear and nonlinear phase respectively and optimized alternately to minimize complex errors in the cascade's stopband and passband in peak and MS senses respectively, the latter with respect to a desired pure delay. Plots can be made afterward from matlab vectors double(gopt) **and** double(hopt)**.**

in X, the MSE will contain terms quadratic and linear in X. PSDs are nonnegative, as then are both the integrand in (1) and the MSE. This then specifies an NND quadratic constraint to upper bound an MSE by an optimizable bound b:

```
b  = optVar( X ) ;
d  = optSequence( DesiredResponse ) ;
r  = Process( 'Box', 0, W ) ;
e  = r . ∗  ( h  −  d ) ;
C  = pwr( e ) / pwr( r )  <  b  ;
```



(a) Overall magnitude responses.



(b) Passband magnitudes, cascade group delay.

Figure 5: The Opt code in Fig. 4 designs shaping (light) and interpolation (dashed) filters so that specifications are met on their cascade (heavy), an IFIR filter.

After bound b is minimized, “<” will hold with equality. Normalizing the power out of the difference filter by the numerical power into it ensures that as long as the PSD of r is constant on its support, double(optimal(b, soln)) will be the actual MS approximation error across that support band.

3: Concluding Examples

The Fig. 4 code alternately optimizes each component of an interpolated FIR (IFIR) filter, iterating towards Fig. 5. Figures 6 and 7 present (without code) a 2D example, a third-band filter: shifting two copies of its frequency response so that their passbands land in disjoint subsets of the original interstices results in the three responses summing to unity. Elsewhere we have presented data-communication filters, [6], polyphase filters for periodically nonuniform bandpass sampling and reconstruction [7], eigenfilters [4] (which Opt also supports), and a wideband antenna array with complex FIR filters on each element optimized jointly with power-efficiency (transmit) or SNR (receive) constraints [8].

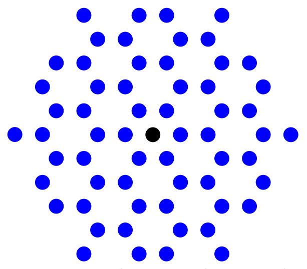
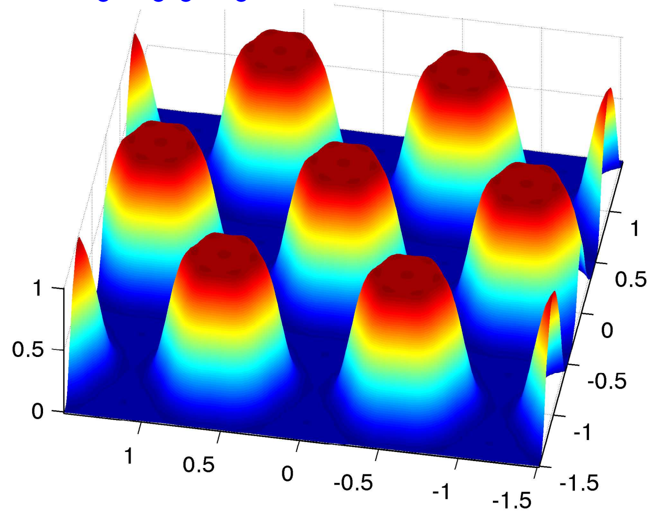


Figure 6: (Left) Locations in 2D of third-band-filter impulse-response samples.

Figure 7: (Below) Third-band frequency response optimized to roughly a 60 dB stopband depth.



References

[1] J. O. Coleman and D. W. Lytle, “Linear-programming techniques for the control of intersymbol interference with hybrid FIR/analog pulse shaping,” in *IEEE Int’l Conf. on Communications*, (Chicago), June 1992.

[2] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, “Applications of second-order cone programming,” *Linear Algebra and its Applications*, vol. 284, pp. 193–228, Nov. 1998.

[3] J. O. Coleman, “Systematic mapping of quadratic constraints on embedded FIR filters to linear matrix inequalities,” in *Conference on Information Sciences and Systems*, (http://www.ece.jhu.edu/ciss/index.html), Mar. 1998.

[4] J. O. Coleman, “Quadratic FIR-filter design as a generalized eigenproblem,” in *European Signal Processing Conf.*, (http://www.eurasip.org/conferences.htm), Sept. 1998.

[5] J. O. Coleman and D. P. Scholnik, “Design of nonlinear-phase FIR filters with second-order cone programming,” in *IEEE Midwest Symposium on Circuits and Systems*, (http://www.mwscas.org/), Aug. 1999.

[6] D. P. Scholnik and J. O. Coleman, “Constrained quadratic design of FIR data-communication filters with linear matrix inequalities,” in *Conf. on Inf. Sciences and Systems*, Mar. 1998.

[7] D. P. Scholnik and J. O. Coleman, “Nonuniformly offset polyphase synthesis of a bandpass signal from complex-envelope samples,” in *IEEE Int’l Symp. on Circuits and Systems*, (http://filter.ece.uci.edu/conf/), June 1999.

[8] D. P. Scholnik and J. O. Coleman, “Superdirectivity and SNR constraints in wideband array-pattern design,” in *IEEE International Radar Conference*, (http://ewh.ieee.org/soc/aes/Conferences.html), May 2001.