

On the design of IIR filters

Robert G. Jenssen

June 29, 2017

Copyright © 2017 Robert G. Jenssen

This work is licensed under the Creative Commons Attribution 4.0 International License.

View a copy of the license at: <https://creativecommons.org/licenses/by/4.0/legalcode>

Contents

1	Introduction	11
I	State Variable description of digital filters	15
2	A review of the State Variable description of digital filters	17
2.1	The z-transform	17
2.2	Filter difference equation	18
2.3	Filter transfer function	19
2.4	Filter signal flow graph	19
2.5	State variable description of a signal flow graph	21
2.6	Controllability	23
2.7	Observability	23
2.8	Coordinate Transformations	24
2.9	State variable descriptions and the transfer function	24
2.9.1	Transformation of a transfer function to a state variable description	25
2.9.2	Transformation of a state variable description to a transfer function	26
2.9.3	Sensitivity of the state variable description of a transfer function	28
2.10	Time domain description	29
2.11	Unit Pulse Response	29
2.12	Block processing and decimation filters	29
2.13	Continued fraction expansion of the transfer function	33
3	Frequency transformations of state variable filters	35
3.1	Frequency Transformations	35
3.2	Frequency Transformation of the Transfer Function	36
3.3	Frequency Transformations of State Variable Filters	37
3.4	An example: frequency transformations of a 5-th order elliptic filter	39

4 Factored state variable descriptions	43
4.1 Construction of the factored state variable description	43
4.1.1 Factoring cascaded state variable filters	44
4.1.2 Factoring block state variable filters	44
4.1.3 Factoring a feedback connection of state variable filters	45
4.2 Factored state variable filters with fractional delays	45
5 Round-off noise in state variable filters	47
5.1 Quantisation noise in digital filters	47
5.2 Limit-cycle oscillations in digital filters	48
5.3 State variable filters and wide sense stationary inputs	48
5.3.1 The filter state covariance matrix	48
5.3.2 The output response to white noise in a state variable	49
5.3.3 Scaling State Variable Filters To Avoid Overflow	50
5.4 Estimation of output round-off noise in state variable filters	51
5.4.1 Rounding-to-minus-infinity quantisation noise	52
5.5 Minimization of round-off noise in the calculation of the state vector	53
5.6 Coefficient sensitivity	55
5.7 Factored state variable filters and wide sense stationary inputs	55
5.8 Frequency transformations and round-off noise	56
6 State variable filter realisation as a cascade of second order sections	57
6.1 Second Order State Variable Filters Optimised for Overflow and Round-Off Noise	57
6.2 Design equations for optimised second order state variable filters	57
6.3 Block optimal second order cascade filter realisations	60
6.4 An example of a second-order state-variable cascade filter	60
6.4.1 Comparison of calculated noise gains	60
6.4.2 Simulation results	61
6.4.3 Comparison with an N=10 example	63

7 Filter synthesis by the Schur decomposition	65
7.1 The Schur algorithm	65
7.1.1 Computation of Schur polynomials	65
7.1.2 Orthonormality of Schur Polynomials	66
7.1.3 Polynomial Expansion Algorithm	68
7.1.4 Power calculation using the Schur algorithm	68
7.2 Derivation of Digital Lattice Filters	68
7.2.1 Derivation of FIR, All-Pole and All-Pass Lattice Filters	69
7.3 Derivation of the One-Multiplier IIR Lattice Filter	70
7.4 Derivation of the Normalised Lattice Filter	72
7.5 Derivation of the Scaled Normalised Lattice Filter	73
7.5.1 Example: synthesis of a 3rd order Butterworth lattice filter	75
7.6 State Variable Descriptions for Schur Lattice Filters	76
7.6.1 State variable description of the Schur FIR lattice filter	76
7.6.2 State variable description of the one-multiplier IIR lattice filter	77
7.6.3 State variable description of the scaled-normalised IIR lattice filter	78
7.7 Roundoff Noise Calculation in Schur Lattice Filters	80
7.7.1 Round-off noise of the normalised-scaled lattice filter	80
7.7.2 Round-off noise of the one multiplier lattice filter	88
7.8 Retiming Schur lattice filters	92
7.8.1 Retiming a 3-rd order lattice filter	92
7.8.2 Retiming a 4-th order Schur normalised-scaled lattice filter	92
7.8.3 Retiming a 6th-order Schur one-multiplier lattice filter	93
7.8.4 Frequency transformations of retimed Schur lattice filters	95
7.9 Frequency transformations and Schur normalised-scaled lattice filter round-off noise	97
7.10 Summary	98
8 Orthogonal state variable filters	99
8.1 Definition of orthogonal state variable filters	99
8.2 The Lattice Orthogonal All-Pass Filter Section	100
8.3 Noise gain of orthogonal filters	100
8.4 Realisation of arbitrary filters from orthogonal sub-filters	101
8.4.1 An example of structural variations	103

9 Feedforward and feedback of state quantisation error in state variable filters	105
9.1 Problem formulation	105
9.2 Minimisation of round-off noise with $\delta = I$ and $\eta = 0$	107
II Constrained optimisation of the IIR filter frequency response	109
10 IIR filter design using Sequential Quadratic Programming with the transfer function defined by pole and zero locations	111
10.1 Problem Statement	111
10.1.1 Solution of the constrained quasi-Newton optimisation problem	112
10.1.2 Choice of active constraints	113
10.1.3 Linearisation of peak constraints	114
10.1.4 Ensuring the stability of the IIR filter	115
10.1.5 Selecting an initial filter design	115
10.2 Examples of IIR filter design with SQP and constrained pole and zero locations	119
10.2.1 Introductory comments on the IIR filter design examples	119
10.2.2 Tarczynski et al. Example 2	122
10.2.3 Deczky's Example 3	125
10.2.4 Deczky's Example 1	132
10.2.5 Low-pass R=2 decimation filter	136
10.2.6 Band-pass R=2 decimation filter	142
10.2.7 Hilbert transform R=2 decimation filter	149
10.2.8 R=2 differentiator filter	152
10.2.9 Pink noise filter	156
10.2.10 Minimum phase R=2 filter	159
10.2.11 Minimum phase FIR bandpass filter	162
11 IIR filter design using Second Order Cone Programming	167
11.1 Second Order Cone Programming	167
11.2 Design of IIR filters with SOCP	167
11.3 Using the <i>SeDuMi</i> SOCP solver	169
11.4 An example of SOCP MMSE design of an IIR filter expressed in <i>gain-zero-pole</i> format with <i>SeDuMi</i>	170
11.5 SOCP MMSE design of a bandpass R=2 filter expressed in <i>gain-zero-pole</i> format with <i>SeDuMi</i>	178

12 IIR filter design with a pre-defined structure	181
12.1 Design of an IIR filter composed of second-order sections	181
12.1.1 Linear constraints on the stability of second-order filter sections	181
12.1.2 Design of an IIR filter composed of second order sections with <i>SeDuMi</i>	182
12.1.3 Some notes on the design of an IIR filter composed of second order sections with <i>SeDuMi</i>	188
12.2 Design of an IIR filter as the sum of two all-pass filters	189
12.2.1 Design of an IIR filter as the sum of two all-pass filters each composed of second-order sections	189
12.2.2 Design of an IIR filter as the sum of an all-pass filter composed of second-order sections and a delay	197
12.2.3 Design of an IIR filter as the sum of two all-pass filters each represented in pole-zero form	198
12.2.4 Design of an IIR filter as the sum of a delay and an all-pass filter represented in pole-zero form	210
12.2.5 Design of an IIR filter as the polyphase decomposition into two all-pass filters each represented in pole-zero form	217
12.3 Design of an IIR Schur lattice filter	222
12.3.1 Design of an IIR one-multiplier Schur lattice low-pass filter using SOCP	222
12.3.2 Design of an IIR one-multiplier Schur lattice low-pass filter using SQP	225
12.3.3 Design of an IIR one-multiplier Schur lattice band-pass filter using SQP	228
12.3.4 Design of an IIR one-multiplier Schur lattice Hilbert filter using SQP	231
12.3.5 Design of an IIR Schur normalised-scaled lattice low-pass filter using SQP	233
12.4 Design of IIR filters with a sharp transition band by frequency response masking	236
12.4.1 Review of Frequency Response Masking digital filters	236
12.4.2 Design of an FRM digital filter with an IIR model filter consisting of a cascade of second-order sections using SOCP	239
12.4.3 Design of an FRM digital filter with an IIR model filter represented in gain-pole-zero form using SOCP and PCLS optimisation	245
12.4.4 Design of an FRM digital filter with an allpass model filter represented in gain-pole-zero form using SOCP and PCLS optimisation	251
12.4.5 Design of an FRM digital filter with an IIR model filter consisting of parallel allpass filters	256
12.4.6 Design of an FRM half-band digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation	262
12.4.7 Design of an FRM Hilbert digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation	268
III Design of IIR filters with integer coefficients	273
13 Signed-digit representation of filter coefficients	275
13.1 Lim's method for allocating signed-digits to filter coefficients	276
13.2 Ito's method for allocating signed-digits to filter coefficients	277
13.3 Signed-digit allocation of the coefficients of a Schur one-multiplier lattice filter	277

14 Searching for integer and signed-digit filter coefficients	285
14.1 Exhaustive search for signed-digit filter coefficients	285
14.2 <i>Bit-flipping</i> search for integer and signed-digit filter coefficients	285
14.2.1 Bit-flipping algorithm examples	285
14.3 <i>Branch-and-bound</i> search for signed-digit coefficients	294
14.3.1 Branch-and-bound search for the 6 bit 3 signed-digit coefficients of a one-multiplier Schur lattice filter	294
14.3.2 Branch-and-bound search for the 10 bit 3 signed-digit coefficients of a one-multiplier Schur lattice filter	298
14.3.3 Branch-and-bound search for the 12 bit 2 signed-digit coefficients of an FRM Hilbert filter with an allpass model filter	301
14.4 Relaxation search for signed-digit filter coefficients	305
14.4.1 SQP relaxation search for the signed-digit coefficients of a Schur one-multiplier lattice bandpass filter	305
14.4.2 SQP relaxation search for the signed-digit coefficients of a Schur one-multiplier lattice lowpass filter	308
14.4.3 SOCP-relaxation search for the signed-digit coefficients of a Schur one-multiplier lattice Hilbert filter	310
14.4.4 SOCP-relaxation search for the signed-digit coefficients of an FRM Hilbert filter with a Schur one-multiplier all-pass lattice model filter	313
14.4.5 POP relaxation search for the signed-digit coefficients of a Schur one-multiplier lattice bandpass filter	316
14.4.6 SOCP-relaxation search for the signed-digit coefficients of an FIR lattice Gaussian filter	319
15 Comparison of filter coefficient search methods for a 5th order elliptic filter with 6-bit integer and 2-signed-digit coefficients	324
15.1 Searching with the bit-flipping algorithm	324
15.2 Searching with the <i>Nelder-Mead</i> simplex algorithm	331
15.3 Searching with the <i>simulated annealing</i> algorithm	337
15.4 Searching with the <i>differential evolution</i> algorithm	344
15.5 Summary of the search algorithm comparison	350
IV Appendixes	351
A Review of Complex Variables	353
A.1 Complex Functions	353
A.2 Limit	353
A.3 The Cauchy-Riemann Equations	353
A.4 Line integrals in the complex plane	354
A.5 Cauchy's Integral Theorem	355
A.6 Cauchy's Integral Formula	355
A.7 Derivatives of an analytic function	356
A.8 Laurent's Theorem	356
A.9 Residues	356
A.10 Cauchy's Argument Principle	357
A.11 Rouché's Theorem	357

B IIR filter amplitude, phase and group-delay frequency responses	359
C Gradient of the IIR filter amplitude response with respect to frequency	367
D Allpass filter frequency response	369
D.1 Allpass filter phase response	369
D.2 Allpass filter group delay response	370
E Gradients of the state variable filter frequency response	373
E.1 Gradients of the state variable filter complex frequency response	373
E.2 State variable filter squared-magnitude response	373
E.3 State variable filter phase response	374
E.4 State variable filter group-delay response	374
F Constrained non-linear optimisation	377
F.1 Newton's method for a quadratic function	377
F.2 Lagrange multipliers	377
F.3 Karush-Kuhn-Tucker conditions for constrained optimisation	378
F.4 Constrained optimisation using Newton's method	379
F.5 Local convergence	379
F.6 Quasi-Newton methods	380
F.6.1 Updating the Hessian approximation with the <i>Broyden-Fletcher-Goldfarb-Shanno</i> (BFGS) formula	380
F.6.2 A modified Cholesky factorisation of the Hessian	381
F.6.3 Wright's modification for degenerate constraints	382
F.6.4 Bertsekas' modification to the Hessian	384
F.7 Penalty and barrier methods	384
F.7.1 Penalty functions	384
F.7.2 Barrier functions	386
F.8 Finding the step size	387
F.8.1 Line search with the Golden-Section	387
F.8.2 Inexact step-size selection	388
F.8.3 Lanczos step-size selection	389
F.9 Initial solution with the Goldfarb-Idnani algorithm	389
F.10 Implementation examples	395

G Low passband sensitivity IIR filters	399
G.1 Structural Boundedness	399
G.2 Filter realisation as the sum of all-pass functions	400
G.3 A note on the numerical calculation of the spectral factor	402
G.4 Examples of parallel all-pass filter synthesis	404
G.4.1 3rd order Butterworth low-pass filter	404
G.4.2 6th order Butterworth band-pass	405
H Low passband sensitivity FIR lattice filters	409
H.1 Lattice decomposition of an FIR digital filter	409
H.2 Finite-wordlength properties of the lattice FIR filter	410
H.3 State variable description of the complementary FIR lattice filter	411
H.4 Design of the complementary FIR digital filter	412
H.4.1 <i>Orchard and Willson's</i> Newton-Raphson solution	412
H.4.2 <i>Mian and Nainer's</i> cepstral method	413
H.4.3 Example: the minimum-phase complementary filter of an FIR bandpass filter	415
I Review of Lanczos tridiagonalisation of an unsymmetric matrix	417
J Review of Butterworth second order sections	419
J.1 Continuous Time Second Order Butterworth Filter Prototypes	419
J.2 Design of discrete time filters with the bilinear transform	420
J.3 Design of discrete time second order Butterworth filters with the bilinear transform	420
K Fourier transform of the Gaussian function	423
K.1 Preliminary results	423
K.1.1 Integral of the Gaussian function	423
K.1.2 Fourier transform of the derivative of a function	423
K.2 Derivation of the Fourier transform of the Gaussian function in the frequency domain	424
Colophon	425
Bibliography	437

Chapter 1

Introduction

An IIR filter can approximate a desired response with fewer coefficients than an FIR filter. The design of IIR filters is more difficult than the design of FIR filters. An FIR filter design problem can be formulated as a convex optimisation problem with a global solution (see, for example, *Wu et al.* [87]). The coefficient-response surface of an IIR filter rational polynomial transfer function is more complicated than that of an FIR polynomial transfer function. An IIR filter design procedure must find a locally optimal solution that satisfies the specifications and the coefficients of the IIR transfer function denominator polynomial must be constrained to ensure that the IIR filter is stable.

This report describes my experiments in the design of IIR digital filters with given amplitude, phase and group delay responses and truncated or quantised coefficients. I programmed those experiments in the *Octave* language [46]. Octave is an “almost” compatible open-source-software clone of the commercial *MATLAB* package [93]. The minimum-mean-squared-error (MMSE) approximation to the required response is found by either a sequential-quadratic-programming (SQP) solver or by the *SeDuMi* second-order-cone-programming (SOCP) solver originally written by *Sturm* [90]. The stability of the filter is ensured by constraining the pole locations of the filter transfer function when expressed in gain-pole-zero form [3, 56] or by constraining the reflection coefficients of a tapped all-pass lattice filter implementation [4, 49]. A valid initial solution for the MMSE solver is found by “eye” or by using the WISE method of *Tarczynski et al.* [8]. A peak-constrained-least-squares (PCLS) solution is found by the exchange algorithm of *Selesnick et al.* [39]. The lattice filter implementation with integer coefficients has good round-off noise and coefficient sensitivity performance. For coefficient word lengths greater than 10-bits the coefficients are allocated signed-digits by the algorithm of *Lim et al.* [105] or that of *Ito et al.* [80] and branch-and-bound or relaxation search is used to find an acceptable response. For lesser coefficient word-lengths simulated-annealing gives the best results.

The state variable description of digital filters Part I is a review of the *state variable* description of digital filters. The state variable description models the internal structure and round-off noise performance of the digital filter. Chapter 7 shows the state variable description of the tapped all-pass lattice filter. This part summarises chapters 8 to 10 of the book “*Digital Signal Processing*” by *Roberts and Mullis* [76] and chapter 12 of “*VLSI Digital Signal Processing Systems : Design and Implementation*” by *Parhi* [49].

Optimising the IIR filter frequency response Part II reviews constrained optimisation of the IIR filter transfer function.

One formulation of the filter optimisation problem is to minimise the *weighted squared error* of the frequency response:

$$\begin{aligned} \text{minimise } & \mathcal{E}_H(x) = \int W(\omega) |H(x, \omega) - H_d(\omega)|^2 d\omega \\ \text{subject to } & H \text{ is stable} \end{aligned} \tag{1.1}$$

where x is the coefficient vector of the filter, \mathcal{E}_H is the weighted sum of the squared error, $W(\omega)$ is the frequency weighting, $H(x, \omega)$ is the filter frequency response and $H_d(\omega)$ is the desired filter frequency response. The solution proceeds by choosing an initial coefficient vector and calling the SQP solver to find the coefficient vector that optimises a second-order approximation to \mathcal{E}_H . The solution is repeated until the difference between successive errors or successive coefficient vectors is sufficiently small.

Alternatively, the optimisation problem can be expressed as a *weighted mini-max* problem:

$$\begin{aligned} \text{minimise } & \max W(\omega) |H(x, \omega) - H_d(\omega)| \\ \text{subject to } & H \text{ is stable} \end{aligned} \tag{1.2}$$

Similarly, in this case, given an initial coefficient vector, the solution proceeds by calling the SOCP solver to find the coefficient vector that minimises the maximum error of a first-order approximation to H .

Constraints on the filter response are applied by the *Peak-Constrained-Least-Squares* (PCLS) exchange algorithm of *Selesnick, Lang and Burrus* [39].

Chapter 10 applies the *Sequential Quadratic Programming* (SQP) method to the optimisation of the frequency of an IIR filter. The SQP method is reviewed in Appendix F. That review makes extensive use of the books by *Nocedal and Wright* [47], *Ruszczynski* [7] and *Bertsekas* [20]. I chose to write my own SQP solver in Octave. In chapter 10, I follow *Deczky* [3] and *Richards* [56] and optimise the filter response with respect to the gain and pole and zero locations of the filter rather than the coefficients of the transfer function polynomial. When the transfer function is expressed in *gain-pole-zero* form the stability of the filter is ensured by constraining the radius of the poles of the filter. Calculation of the response and gradient from the coefficients of the transfer function polynomials is simpler but the stability constraint on the transfer function denominator is more complex and may exclude valid alternative designs. Appendix B, derives expressions for the amplitude, phase and delay responses and gradients of an IIR filter in terms of the gain-pole-zero coefficients. The SQP method requires that at each step the optimisation problem is initialised with the second-order derivatives (ie: the Hessian matrix) of the response with respect to the coefficients.

Chapter 11 is based on the description of IIR filter design using *Second-Order-Cone-Programming* (SOCP) by *Lu and Hinamoto* [95]. SOCP is a subclass of *convex programming* [83]. See the review articles by *Alizadeh and Goldfarb* [25] and *Lobo et al.* [62] for a description of applications of SOCP. Unlike SQP, SOCP optimisation does not require the the Hessian of the response. In this report I use the public domain *SeDuMi* SOCP solver originally written by *Jos Sturm* [91].

Chapter 12 considers optimisation of the filter transfer function of several filter structures. The simplest IIR filter structure is the “*direct form*” implementation of the filter transfer function. This filter structure rarely has good round-off noise and coefficient sensitivity when the coefficients are truncated. *Regalia et al.* [69] and *Vaidyanathan et al.* [72] show that an IIR digital filter composed of two all-pass filters connected in parallel has desirable coefficient sensitivity and round-off noise performance. *Renfors and Saramäki* [60, 61], describe IIR digital filterbanks as a “*polyphase*” combination of all-pass digital filters. *Gray and Markel* [4] and *Parhi* [49, Chapter 12] describe the synthesis of digital filters as tapped lattice structures. The lattice filter is stable if the lattice coefficients, k_l , have $|k_l| < 1$. *Johansson and Wanhammar* [34], *Milić and Ćertić* [52] and *Lu and Hinamoto* [95] describe efficient, sharp transition-band IIR filter designs using the *frequency masking* approach described for FIR filters by *Lim* [104].

Truncating the IIR filter coefficients Part III considers algorithms for optimising the frequency response of the IIR filter with truncated or quantised rather than exact or floating point coefficients. The truncated coefficients are represented as *N-bit 2's complement* or *M-signed-digit* numbers. The signed-digit representation reduces the complexity and power requirements of the filter. *Lim et al.* [105] and *Ito et al.* [80] describe methods of allocating the number of signed-digits used by each coefficient. This part considers methods of searching the space of truncated coefficients for the best filter response. A brute force, exhaustive, search is likely to take too much time. Chapter 14 considers local optimisation of the truncated filter coefficients by *branch-and-bound* search [5] or by SQP *relaxation* of the coefficients. The latter repeatedly fixes one coefficient to an integer value and optimises the remaining coefficients subject to the filter specification. Figure 1.1 shows the response of a 20'th order tapped Schur lattice bandpass filter with denominator coefficients only in z^{-2} and 31 non-zero coefficients. The figure compares the filter responses for the exact coefficients and 10-bit signed-digit coefficients with an average of 3 signed-digits allocated to each coefficient by the method of *Ito et al.* [80]. After SQP-relaxation optimisation of the signed-digit coefficient values, 57 signed-digits and 26 shift-and-add operations are required to implement the coefficient multiplications.

Chapter 15, shows the results of searching for the truncated coefficients of a 5th order elliptic filter having various structures and 6-bit coefficients with the *bitflipping* algorithm of *Krukowski and Kale* [6], and the *simplex, differential evolution* and *simulated annealing* routines from the Octave-Forge *optim* package [64].

Reproducing my results The Octave scripts referred to in this report generate long sequences of floating point operations. I recommend reading “*What Every Computer Scientist Should Know About Floating-point Arithmetic*” by *Goldberg* [30] and “*The pernicious polynomial*” by *Wilkinson* [40]. In the latter, Wilkinson reminisces about programming polynomial root finding on early computers. He comments that “*explicit polynomial equations with ill-conditioned roots are remarkably common*” but that if “*one already knows the roots, then the polynomial can be evaluated without any loss of accuracy.*” As an example, Figure 1.2 shows the results of calling the Octave *roots* function to find the roots of the binomial polynomial of order 20 with the expression *roots (bincoef (20, 0:20))*. In fact, by definition, the roots are all -1 .

The results shown in this report were obtained on my system running with a particular combination of CPU architecture, operating system, library versions, compiler version and Octave version. In the *Colophon* I describe building and benchmarking Octave on my system. **YOUR SYSTEM WILL ALMOST CERTAINLY BE DIFFERENT. YOU MAY NEED TO MODIFY A SCRIPT TO RUN ON YOUR SYSTEM.** Try relaxing the constraints on the filter design, relaxing the tolerance on the optimised result or change the relative weights on the filter bands.

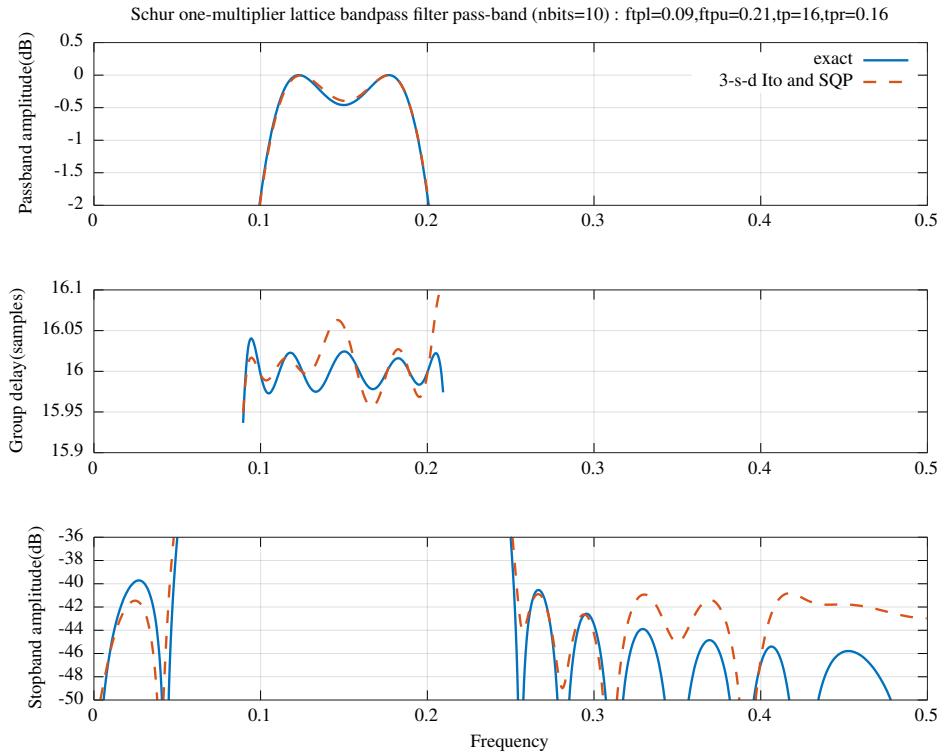


Figure 1.1: Comparison of the pass-band and stop-band amplitude responses and group delay responses for a Schur one-multiplier lattice bandpass filter with exact coefficients and with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation optimisation.

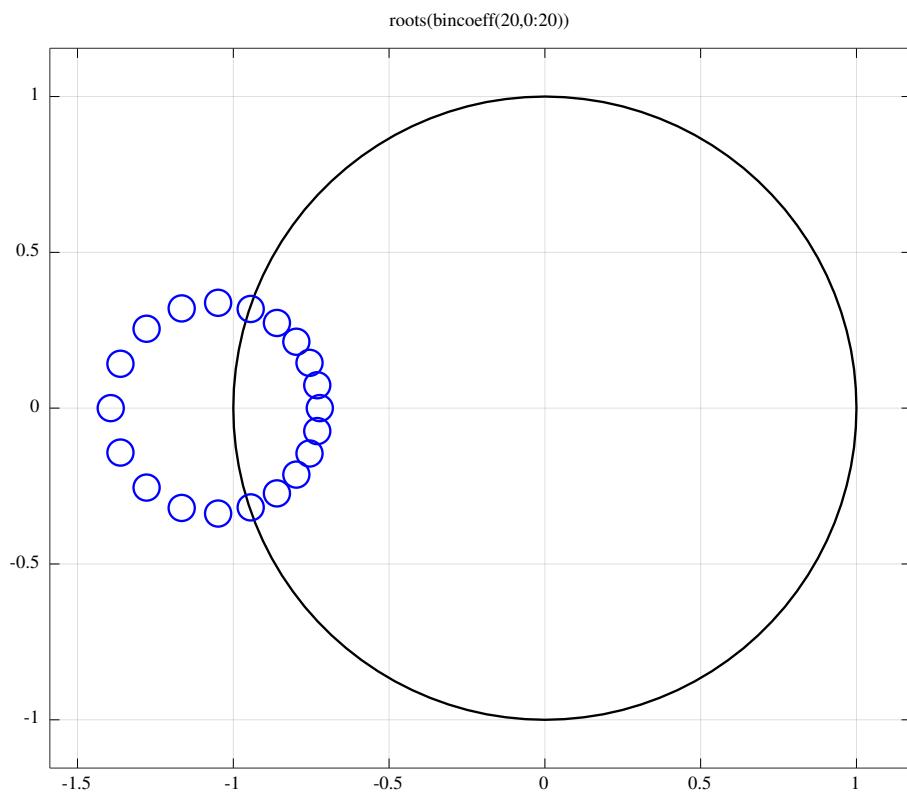


Figure 1.2: Plot of the roots of the binomial polynomial of order 20 calculated by the Octave *roots* function

Part I

State Variable description of digital filters

Chapter 2

A review of the State Variable description of digital filters

This chapter briefly summarises parts of the text *Digital Signal Processing* by *Roberts and Mullis* [76]. Appendix A reviews the necessary complex variables theory.

2.1 The z-transform

The bi-lateral *z-transform* of a sequence f is

$$F(z) = Z\{f\} = \sum_{k=-\infty}^{\infty} f(k) z^{-k}$$

where z is a complex variable. The region of convergence (ROC) of the series is that part of the z-plane for which

$$\sum_k |f(k) z^{-k}| < \infty$$

If f is causal then the z-transform is one-sided:

$$F(z) = \sum_{k=0}^{\infty} f(k) z^{-k}$$

The *Cauchy integral theorem*, shown in Appendix A.5, can be used to invert the z-transform. Start by selecting a circular contour, C , centred at the origin and lying in the ROC. Multiply each side of the z-transform by z^{n-1} and integrate along C :

$$\oint_C z^{n-1} F(z) dz = \sum_{k=-\infty}^{\infty} \oint_C f(k) z^{n-1-k} dz$$

Cauchy's *integral lemma* states

$$\oint_C z^n dz = \begin{cases} 2\pi i & n = -1 \\ 0 & n \neq -1, \text{ integral} \end{cases}$$

so the terms in the sum vanish except for $k = n$ and

$$f(n) = \frac{1}{2\pi i} \oint_C z^{n-1} F(z) dz$$

For rational z-transforms the contour integrals are usually evaluated by finding the residues of the integrand. See Appendix A.

The inversion integral can be used to derive Parseval's theorem for the z-transform of two sequences. Following *Roberts and Mullis* [76, Section 3.4], let f and g be causal sequences with z-transforms $F(z)$ and $G(z)$, respectively. Define the z-transform of the product as

$$Z\{f \cdot g\} = \sum_{k=0}^{\infty} f(k) g(k) z^{-k}$$

Substituting the inversion integral for \mathbf{f}

$$\begin{aligned} Z\{\mathbf{f} \cdot \mathbf{g}\} &= \frac{1}{2\pi i} \sum_{k=0}^{\infty} \oint_C g(k) s^{k-1} F(s) z^{-k} ds \\ &= \frac{1}{2\pi i} \oint_C s^{-1} F(s) \sum_{k=0}^{\infty} g(k) [zs^{-1}]^{-k} ds \\ &= \frac{1}{2\pi i} \oint_C s^{-1} F(s) G\left(\frac{z}{s}\right) ds \end{aligned}$$

If $|z| = 1$ lies in the ROC of the integrand then

$$Z\{\mathbf{f} \cdot \mathbf{g}\} = \frac{1}{2\pi i} \oint_C s^{-1} F(s) G\left(\frac{1}{s}\right) ds$$

If $\mathbf{f} = \mathbf{g}$ then we have the discrete form of Parseval's theorem:

$$\begin{aligned} Z\{\mathbf{f} \cdot \mathbf{f}\} &= \sum_{k=0}^{\infty} f^2(k) z^{-k} \\ &= \frac{1}{2\pi i} \oint_C z^{-1} F(z) F(z^{-1}) dz \end{aligned}$$

If C is the contour $|z| = 1$ then, after expanding terms in $e^{i\theta}$:

$$\sum_{k=0}^{\infty} f^2(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} |F(e^{i\theta})|^2 d\theta$$

2.2 Filter difference equation

The standard difference equation (SDE) of a digital filter is

$$\sum_{l=0}^n a_l y(k-l) = \sum_{l=0}^m b_l u(k-l), \quad a_0 = 1 \text{ and } k \geq 0 \quad (2.1)$$

or

$$\mathbf{a} * \mathbf{y} = \mathbf{b} * \mathbf{u}$$

$u(k)$ and $y(k)$ are the input and output sequences, respectively. If \mathbf{y}^0 is a solution of the homogenous equation (HE), $\mathbf{y} * \mathbf{a} = 0$, then

$$\mathbf{a} * (\mathbf{y} + \mathbf{y}^0) = \mathbf{b} * \mathbf{u}$$

The polynomial $a(z)$ is known as the *characteristic* polynomial of the HE. The n roots $\lambda_1, \dots, \lambda_n$ of \mathbf{a} define the solutions of the HE since

$$\sum_{l=0}^n a_l z^{k-l} = z^k a(z)$$

and if λ is a root of $a(z)$ then

$$\sum_{l=0}^n a_l \lambda^{k-l} = \lambda^k a(\lambda) = 0 \quad (2.2)$$

If the roots are distinct then there are n solutions of the form $y_l(k) = \lambda_l^k$, $1 \leq l \leq n$ and solutions of the HE are of the form

$$y^0(k) = \sum_{l=1}^n c_l \lambda_l^{k-l}$$

The case of repeated roots can be treated by differentiating Equation 2.2. Roots of multiplicity m generate m solutions $y_l(k) = k^{l-1} \lambda^{k-l+1}$, $1 \leq l \leq m$.

Every solution of the HE is a linear combination of these solutions

$$y^0(k) = \sum_{l=1}^n c_l y_l(k)$$

The constants c_l are usually chosen so that the filter is stable and causal.

The unit-pulse response sequence of the filter, \mathbf{h} , is found by setting the input $\mathbf{u} = \delta$

$$\sum_{l=0}^n a_l h(k-l) = \sum_{l=0}^n b_l \delta(k-l), \quad a_0 = 1 \text{ and } k \geq 0 \quad (2.3)$$

Further, if \mathbf{h} is causal, then it is of the form

$$h(k) = \begin{cases} 0, & k < 0 \\ b_0, & k = 0 \\ \sum_{l=1}^n c_l y_l(k), & k > 0 \end{cases}$$

The initial conditions $h(1), \dots, h(n)$ that determine the c_l are obtained by direct evaluation of Equation 2.3.

Finally, if the filter is Bounded-Input-Bounded-Output (BIBO) stable then

$$\sum_{l=-\infty}^{\infty} |h(k)| < \infty$$

For the case of distinct roots, \mathbf{h} has the form

$$h(k) = \sum_{l=1}^n c_l \lambda_l^k, \quad k > 0$$

If the unit-pulse response is causal, then BIBO stability requires that $|\lambda_l| < 1, l = 1, \dots, n$.

2.3 Filter transfer function

The filter transfer function is related to the z-transform of the filter difference equation, Equation 2.1, by

$$H(z) = \frac{b(z)}{a(z)} = \frac{\sum_{l=0}^m b_l z^{-l}}{\sum_{l=0}^n a_l z^{-l}}$$

The transfer function, $H(z)$, can also be written

$$H(z) = g \frac{\prod_{l=1}^n (z - z_l)}{\prod_{l=1}^m (z - p_l)}$$

The p_l are the roots of $a(z)$ and are called the poles of $H(z)$. The z_l are the roots of $b(z)$ and are called the zeros of $H(z)$. $g = H(0)$ is a gain factor.

2.4 Filter signal flow graph

Signal flow graphs are *primitive* if

1. All branch gains are either constant or z^{-1} (a unit delay)
2. There are no delay free loops in the graph
3. There are a finite number of nodes and branches

Algorithm 1 Procedure for reordering the nodes of a primitive signal flow graph

1. Examine each unit delay. If there is an incoming branch at the output then isolate the output of the unit delay by inserting a unit gain branch as shown in Figure 2.1
 2. Label all input nodes and all unit delay output nodes with 0
 3. All nodes that can be calculated from nodes labelled 0 are labelled 1
 4. If any nodes are unlabelled increment the label and repeat until all nodes are labelled
-

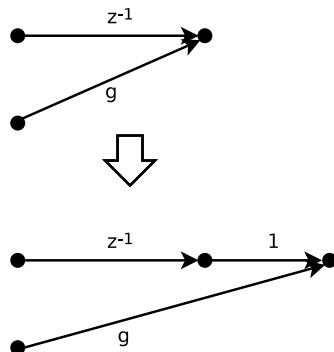


Figure 2.1: Isolating a unit delay from an incoming branch

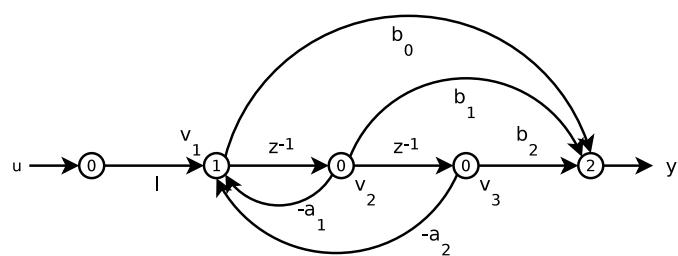


Figure 2.2: Second order direct form filter

The implementation of primitive signal flow graphs requires *node reordering*, shown in Algorithm 1. For example Figure 2.2 shows the signal flow graph of a second order *direct form* filter. The nodes are labelled according to Algorithm 1. The labelled nodes are:

- $S_0: u, v_2, v_3$
- $S_1: v_1 = u - a_1v_2 - a_2v_3$
- $S_2: y = b_0v_1 + b_1v_2 + b_2v_3$

With updates

- $v_2 \leftarrow v_1$
- $v_3 \leftarrow v_2$

Then in the z -domain

$$\begin{aligned} y(z) &= b_0z^2 + b_1z^1 + b_2 \\ u(z) &= z^2 + a_1z + a_0 \end{aligned}$$

so the transfer function is

$$H(z) = \frac{b_0z^2 + b_1z + b_2}{z^2 + a_1z + a_0}$$

n' th order direct form filters can be generated recursively by:

$$\begin{aligned} A_k(z) &= z^{-1}[a_k + A_{k+1}(z)] \\ B_k(z) &= z^{-1}[b_k + B_{k+1}(z)] \end{aligned}$$

where $A_{n+1}(z) = B_{n+1}(z) = 0$. The filter structure recursion is initialised with:

$$\begin{aligned} y(z) &= b_0v + B_1(z)v \\ u(z) &= v + a_0A_1(z)v \end{aligned}$$

so that

$$H(z) = \frac{b_0 + B_1(z)}{1 + a_0A_1(z)}$$

See Figure 2.3.

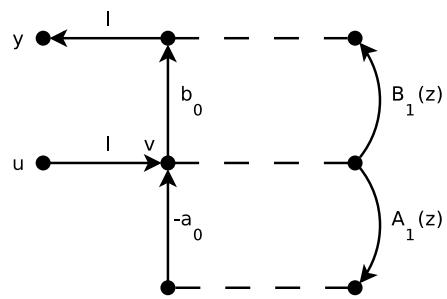
2.5 State variable description of a signal flow graph

The state variable description of a signal flow graph is derived by Algorithm 2.

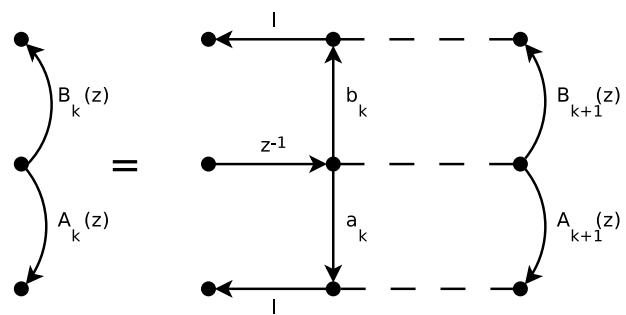
Algorithm 2 Derivation of the state variable description of a signal flow graph

1. Replace each unit delay with the equivalent path shown in Figure 2.4
 2. Remove the unit delays thereby creating new outputs x'_i and inputs x_i . Note that $x'_i(k) \triangleq x_i(k+1)$
 3. Eliminate all nodes that are not inputs or outputs
 4. Replace the unit delays
-

Figure 2.2 shows the signal flow graph for a second order direct form filter.



(a) Left hand side termination



(b) k_{th} internal section

Figure 2.3: Recursive generation of direct form filters

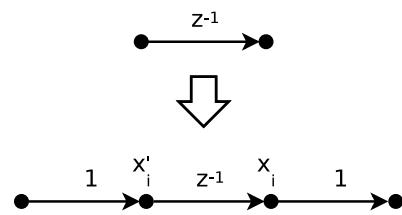


Figure 2.4: Unit delay equivalent

The time domain state variable equations for this filter are:

$$\begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \quad (2.4)$$

where:

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix} \\ B &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ C &= [b_2 - a_2 b_0, \quad b_1 - a_1 b_0] \\ D &= b_0 \end{aligned}$$

The z -domain state variable equations for this filter are:

$$\begin{bmatrix} zX \\ Y \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} X \\ U \end{bmatrix} \quad (2.5)$$

Figure 2.5 shows the signal flow graph for the state variable filter of Equation 2.5. The matrix A is called the *state transition matrix*.

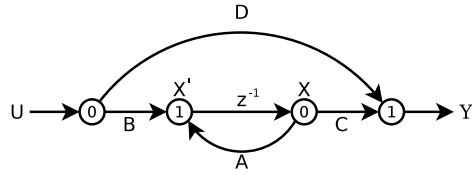


Figure 2.5: Signal flow graph of a state variable filter

2.6 Controllability

The matrix pair (A, B) is said to be *controllable* if and only if $\det [B \ AB \ \dots \ A^{n-1}B] \neq 0$. The n-by-n matrix $[B \ AB \ \dots \ A^{n-1}B]$ is called the *controllability matrix*. If the matrix pair (A, B) is controllable then the system

$$\begin{aligned} x(0) &= 0 \\ x(k+1) &= Ax(k) + Bu(k) \end{aligned}$$

can be driven to any specified vector $x(n)$.

PROOF: From above

$$\begin{aligned} x(n) &= \sum_{l=1}^n A^{l-1}Bu(n-l) \\ &= [B \ AB \ \dots \ A^{n-1}B] \begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(0) \end{bmatrix} \end{aligned}$$

The input $u(k)$, $0 \leq k < n$, producing $x(n)$ can be found by inverting the controllability matrix.

2.7 Observability

The matrix pair (A, C) is said to be *observable* if the matrix $[C \ CA \ \dots \ CA^{n-1}]$ is invertible (for a single output variable). If the matrix pair (A, C) is observable then the state $x(0)$ can be uniquely determined from $(u(k), y(k))$ for $0 \leq k < n$.

PROOF : From the matrix equations for $x(k)$ and $y(k)$

$$\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(n-1) \end{bmatrix} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} x(0) + \begin{bmatrix} D & 0 & \cdots & 0 \\ CB & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{n-2}B & CA^{n-3}B & \cdots & D \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ u(n-1) \end{bmatrix}$$

The right-hand term reduces to an n-by-1 column vector. $x(0)$ is found by inverting the observability matrix.

2.8 Coordinate Transformations

Let T be an n-by-n non-singular matrix and $q(k) = T^{-1}x(k)$ then the state variable equations have the same form except that $\{A, B, C, D\} \leftarrow \{T^{-1}AT, T^{-1}B, CT, D\}$. In matrix form:

$$\begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} = \begin{bmatrix} T^{-1}AT & T^{-1}B \\ CT & D \end{bmatrix} = \begin{bmatrix} T & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} T & 0 \\ 0 & 1 \end{bmatrix}$$

2.9 State variable descriptions and the transfer function

From the z -domain state variable matrix shown in Equation 2.5

$$X(z) = (zI - A)^{-1} BU$$

so

$$Y(z) = C(zI - A)^{-1} BU + DU$$

and the transfer function is

$$H(z) = D + C(zI - A)^{-1} B \quad (2.6)$$

The similarity transformation, $\{A, B, C, D\} \leftarrow \{T^{-1}AT, T^{-1}B, CT, D\}$, leaves the transfer function, $H(z)$, unchanged¹:

$$H(z) = D + C(zI - A)^{-1} B \quad (2.7)$$

$$= D + CTT^{-1}(zI - A)^{-1}(T^{-1})^{-1}T^{-1}B \quad (2.8)$$

$$= D + CT[T^{-1}(zI - A)T]^{-1}T^{-1}B \quad (2.9)$$

$$= D + CT(zI - T^{-1}AT)^{-1}T^{-1}B \quad (2.10)$$

The poles of $H(z)$ are the eigenvalues of A . The zeros of $H(z)$ are related to the $n+1$ components in C and D via a system of linear equations. This system of equations is invertible if the controllability matrix $[B \ AB \ \dots \ A^{n-1}B]$ is non-singular. The term $(zI - A)^{-1}$ is called the *matrix resolvent*. The determinant $p(z) = \det(zI - A)$ is called the *characteristic polynomial* of A .

The transfer function can be written

$$H(z) = \frac{b_0 z^n + b_1 z^{n-1} + \dots + b_{n-1} z + b_n}{z^n + a_1 z^{n-1} + \dots + a_{n-1} z + a_n} \quad (2.11)$$

The denominator of $H(z)$ is the characteristic polynomial of A . The unit impulse response is related to the coefficients of the numerator and denominator polynomials of $H(z)$ by

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ a_1 & 1 & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ a_n & a_{n-1} & \cdots & 1 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

Algorithm 3 Transformation of a transfer function to a state variable description

Given a transfer function

$$\begin{aligned} H(z) &= \frac{\hat{b}(z)}{\hat{a}(z)} \\ &= \frac{b_0 z^n + b_1 z^{n-1} + \cdots + b_n}{z^n + a_1 z^{n-1} + \cdots + a_n} \end{aligned}$$

the direct form state variable description is

$$H(z) = D + C(zI - A)^{-1}B$$

where

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & \cdots & -a_1 \end{bmatrix} \\ B &= [0 \ 0 \ \cdots \ 0 \ 1]^T \\ C &= [b_n - b_0 a_n \ \cdots \ b_1 - b_0 a_1] \\ D &= b_0 \end{aligned}$$

2.9.1 Transformation of a transfer function to a state variable description

Algorithm 3 converts a transfer function description to the equivalent direct form state variable description. An *Octave* implementation is shown in the file *tf2Abcd.m*.

PROOF of Algorithm 3: Firstly, show the identity

$$(zI - A)\Psi(z) = \hat{a}(z)B$$

with $\Psi(z) = [1 \ z \ \cdots \ z^{n-1}]^T$ uniquely defines matrix $A = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{22} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n,1} & \alpha_{n,2} & \cdots & \alpha_{n,n} \end{bmatrix}$ and column vector $B = [\beta_1 \ \cdots \ \beta_n]^T$. For row i ,

$$z^i - \beta_i \left(z^n + \sum_{j=1}^n a_j z^{n-j} \right) = \sum_{j=1}^n \alpha_{i,j} z^{j-1}$$

If $i \neq n$, then, equating coefficients, $\beta_i = 0$, so $\alpha_{i,i+1} = 1$ and $\alpha_{i,j} = 0$ if $i \neq j + 1$. If $i = n$ then $\beta_i = 1$ and

$$-\sum_{j=1}^n a_j z^{n-j} = \sum_{j=1}^n \alpha_{n,j} z^{j-1}$$

so

$$\begin{aligned} \alpha_{n,1} &= -a_n \\ &\vdots \\ \alpha_{n,n} &= -a_1 \end{aligned}$$

Now find vector $C = [c_1 \ c_2 \ \cdots \ c_n]$ and scalar $D = \delta$ that satisfy

$$D\hat{a}(z) + C\Psi = \hat{b}(z)$$

Expanding

$$\delta \left(z^n + \sum_{j=1}^n a_j z^{n-j} \right) + \sum_{j=1}^n c_j z^{j-1} = \sum_{j=0}^n b_j z^{n-j}$$

¹Recall that $(AB)^{-1} = B^{-1}A^{-1}$

Equating coefficients

$$\begin{aligned}\delta &= b_0 \\ c_j &= b_{n-j+1} - b_0 a_{n-j+1} \quad 1 \leq j \leq n\end{aligned}$$

2.9.2 Transformation of a state variable description to a transfer function

This section describes three methods for finding the transfer function of a state variable description.

Using the controllability matrix

Algorithm 4 uses the controllability matrix to find the characteristic polynomial of the state transition matrix, A . The transfer function follows from Equation 2.6.

Algorithm 4 Transformation of state variable description to direct form

If

1. the matrix pair (A, B) is controllable so that $\det [B \ AB \ A^2B \ \dots \ A^{n-1}B] \neq 0$.
2. the characteristic function of the state transition matrix, A , is $\det(zI - A) = a_0z^n + a_1z^{n-1} + \dots + a_n$ with $a_0 = 1$.
3. the state vectors are: $\begin{cases} x(0) = 0 \\ x(k+1) = Ax(k) + Ba_k \end{cases}$

then

1. $x(n+1) = 0$
2. If $T = [x(n) \ x(n-1) \ \dots \ x(1)]$ then

$$T^{-1}AT = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & \cdots & -a_1 \end{bmatrix}$$

$$T^{-1}B = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

PROOF of Algorithm 4: By repeated application, the state vectors, x , at time $k+1$ are:

$$x(k+1) = \sum_{i=0}^k A^{k-i}Ba_i$$

and

$$x(n+1) = \left(\sum_{i=0}^n A^{n-i}a_i \right) B$$

The Cayley-Hamilton theorem states that a matrix is a solution of its own characteristic polynomial so the expression in the parentheses is zero. Factorising

$$\begin{aligned}T &= [x(n) \ x(n-1) \ \dots \ x(1)] \\ &= [A^{n-1}B \ A^{n-2}B \ \dots \ B] \begin{bmatrix} 1 & 0 & \cdots & 0 \\ a_1 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ a_{n-1} & a_{n-2} & \cdots & 1 \end{bmatrix}\end{aligned}$$

If the matrix pair (A, B) is controllable then T is invertible. Now consider the state variables $q(k)$ of the direct form filter with the transfer function denominator polynomial $z^n + a_1 z^{n-1} + \dots + a_n$ and make the input to the filter be the coefficients of this denominator polynomial (recall that by definition $q(k) = T^{-1}x(k)$):

$$\begin{aligned}
q(0) &= 0 \\
q(1) &= \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & \cdots & -a_1 \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \\
q(2) &= \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & \cdots & -a_1 \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} a_1 = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix} \\
&\vdots \\
q(n) &= \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & \cdots & -a_1 \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} a_n = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\
q(n+1) &= 0
\end{aligned}$$

Form the matrix with columns consisting of the states $\{q(n+1), \dots, q(2)\}$:

$$\begin{aligned}
[q(n+1) \quad q(n) \quad \cdots \quad q(2)] &= \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\
&= T^{-1}AT \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_n & a_{n-1} & \cdots & a_1 \end{bmatrix}
\end{aligned}$$

The result follows.

LeVerrier's algorithm

In practice, I have found that the simplest method of finding $(zI - A)^{-1}$ (sometimes referred to as the matrix *resolvent*) is by matrix inversion. An alternative, slower, but more accurate method for finding the resolvent of a matrix is LeVerrier's algorithm [76, Algorithm 8A.12] shown in Algorithm 5. Note that Leverrier's algorithm calculates the characteristic polynomial

Algorithm 5 Leverrier's algorithm for finding the resolvent of the state transition matrix, A [76, Algorithm 8A.12]

To find $p(z) = \sum_{k=0}^n z^{n-k} a_k$ and $(zI - A)^{-1} = \frac{1}{p(z)} \sum_{k=0}^{n-1} z^{n-1-k} \bar{A}_k$ perform the following recursion:

```

 $a_0 = 1, \bar{A}_0 = I$ 
for  $k = 1, \dots, n$  do
   $a_k = -\frac{1}{k} \text{trace}(A\bar{A}_{k-1})$ 
   $\bar{A}_k = A\bar{A}_{k-1} + a_k I$ 
end for

```

of A as a by-product. In this case the characteristic polynomial of the state-transition matrix is the denominator polynomial of the transfer function, which is known. The recursion in \bar{A}_k is justified by rearranging the expansion of the resolvent shown in Algorithm 5 and then equating coefficients of z :

$$\begin{aligned}
\sum_{k=0}^n z^{n-k} a_k I &= (zI - A) \sum_{k=0}^{n-1} z^{n-1-k} \bar{A}_k \\
&= z^n \bar{A}_0 + \sum_{k=1}^{n-1} z^{n-k} (\bar{A}_k - A\bar{A}_{k-1}) - A\bar{A}_{n-1}
\end{aligned}$$

La Budde's algorithm for finding the characteristic polynomial of an upper Hessenberg matrix

Wilkinson [41, Section 57, Chapter 6], Rehman [82, Chapter 6], and Rehman and Ipsen [77] describe La Budde's algorithm for the calculation of the characteristic polynomial of an *upper Hessenberg* matrix, reproduced here as Algorithm 6 [77, Algorithm 2]. Rehman [82, Appendix B] provides a MATLAB implementation, *labudde.m*². Rehman shows that the numerical performance of La Budde's method compares favourably with the eigenvalue method, $p(z) = \prod_{k=1}^n (z - \lambda_k)$, where the λ_k are the (possibly repeated) eigenvalues of A . When the characteristic polynomial is known, the resolvent can be calculated with the matrix recursion of LeVerrier's algorithm.

Algorithm 6 La Budde's algorithm for finding the characteristic polynomial of A [77, Algorithm 2]

Given: $H = \begin{bmatrix} \alpha_1 & h_{1,2} & \cdots & \cdots & h_{1,n} \\ \beta_2 & \alpha_2 & h_{2,3} & & \vdots \\ \ddots & \ddots & \ddots & & \vdots \\ & \ddots & \ddots & h_{n-1,n} & \\ 0 & & \beta_n & \alpha_n & \end{bmatrix}$, the $n \times n$ *upper Hessenberg* reduction of A , find $a_1 \dots a_n$:

$$\begin{aligned} a_1^{(1)} &= -\alpha_1 \\ a_1^{(2)} &= a_1^{(1)} - \alpha_2 \\ a_2^{(2)} &= \alpha_1 \alpha_2 - h_{1,2} \beta_2 \\ \textbf{for } k = 3, \dots, n \textbf{ do} \\ a_1^{(k)} &= a_1^{(k-1)} - \alpha_k \\ \textbf{for } l = 2, \dots, k-1 \textbf{ do} \\ a_l^{(k)} &= a_l^{(k-1)} - \alpha_k a_{l-1}^{(k-1)} - \sum_{m=1}^{l-2} h_{k-m,k} \beta_k \cdots \beta_{k-m+1} a_{l-m-1}^{(k-m-1)} - h_{k-l+1,k} \beta_k \cdots \beta_{k-l+2} \\ \textbf{end for} \\ a_k^{(k)} &= -\alpha_k a_{k-1}^{(k-1)} - \sum_{m=1}^{k-2} h_{k-m,k} \beta_k \cdots \beta_{k-m+1} a_{k-m-1}^{(k-m-1)} - h_{1,k} \beta_k \cdots \beta_2 \\ \textbf{end for} \\ a_k &= a_k^{(n)}, \quad 1 \leq k \leq n \end{aligned}$$

The first stage of La Budde's method reduces the non-symmetric matrix, A , to *upper Hessenberg* form, H (see Golub and van Loan [26, Section 7.4]). The determinant of a matrix is unchanged by replacing a row of the matrix by linear combinations with other rows, so H and A have the same characteristic polynomial. Rehman and Ipsen [77, pp.10-11] justify La Budde's algorithm as follows (lightly edited for consistency with my notation):

La Budde's method computes the characteristic polynomial of an upper Hessenberg matrix, H , by successively computing the characteristic polynomials of leading principal submatrices H_k of order k . Denote the characteristic polynomial of H_k by $p_k(z) = \det(zI - H_k)$, $1 \leq k \leq n$, where $p(z) = p_n(z)$. The recursion for computing $p(z)$ is [41, Section 6.57.1]

$$\begin{aligned} p_0(z) &= 1 \\ p_1(z) &= z - \alpha_1 \\ p_k(z) &= (z - \alpha_1)p_{k-1}(z) - \sum_{m=1}^{k-1} h_{k-m,k} \beta_k \cdots \beta_{k-m+1} p_{k-m-1}(z) \end{aligned} \tag{2.12}$$

where $2 \leq k \leq n$. The recursion for $p_k(z) = \sum_{m=0}^k z^{k-m} a_m^{(k)}$ is obtained by developing the determinant of $zI - H_k$ along the last row of H_k . Each term in the sum contains an element in the last column of H_k and a product of subdiagonal elements. Equating like powers of z in Equation 2.12 gives recursions for individual coefficients a_k .

2.9.3 Sensitivity of the state variable description of a transfer function

Thiele [54] analyses the sensitivity of the frequency response of linear state space digital filters with respect to the coefficients. Differentiation of the resolvent, $R = (zI - A)^{-1}$, is simplified by the matrix identity:

$$\begin{aligned} RR^{-1} &= I \\ \frac{dR}{d\rho} R^{-1} + R \frac{dR^{-1}}{d\rho} &= 0 \end{aligned}$$

²I have edited *labudde.m* to make it compatible with Octave

$$\frac{dR^{-1}}{d\rho} = -R^{-1} \frac{dR}{d\rho} R^{-1}$$

where ρ represents the components of the matrix R . With this identity, the gradients of $H(z)$ are found by differentiating Equation 2.6:

$$\begin{aligned}\frac{\partial H(z)}{\partial z} &= -C(zI - A)^{-2} B \\ \frac{\partial H(z)}{\partial \alpha} &= C(zI - A)^{-1} \frac{\partial A}{\partial \alpha} (zI - A)^{-1} B \\ \frac{\partial H(z)}{\partial \beta} &= C(zI - A)^{-1} \\ \frac{\partial H(z)}{\partial \gamma} &= (zI - A)^{-1} B \\ \frac{\partial H(z)}{\partial \delta} &= I\end{aligned}$$

where α, β, γ and δ represent the components of A, B, C and D respectively.

2.10 Time domain description

If the input sequence is zero then in the time-domain, given $x(k_0)$

$$x(k+1) = Ax(k), \quad k > k_0$$

The matrix powers of A tend to 0 as k approaches infinity if and only if the eigenvalues λ_k of A satisfy

$$|\lambda_k| < 1, \quad k = 1, \dots, n$$

This is equivalent to the stability condition that the poles of $H(z)$ lie inside the unit circle in the z -plane since the eigenvalues of A are the roots of the polynomial $\hat{a}(z) = \det(zI - A)$ which is the denominator polynomial of the transfer function $H(z)$. If the matrix A is stable then

$$x(k) = \sum_{l=1}^{\infty} A^{l-1} Bu(k-l)$$

2.11 Unit Pulse Response

The input and output sequences are related by $y = h * u$. The state variable form is

$$\begin{aligned}y(k) &= Cx(k) + Du(k) \\ &= \sum_{l=1}^{\infty} CA^{l-1} Bu(k-l) + Du(k) \\ &= \sum_{l=-\infty}^{\infty} h(l) u(k-l)\end{aligned}$$

For these to agree

$$h(k) = \begin{cases} 0 & k < 0 \\ D & k = 0 \\ CA^{k-1}B & k > 0 \end{cases}$$

2.12 Block processing and decimation filters

See *Roberts and Mullis* [76, Section 10.2]. Block processing digital filters are multi-input, multi-output (MIMO) filters which are equivalent to a single-input, single-output (SISO) filter. MIMO filters have the following advantages:

- parallel computation increases the output data rate
- output noise is reduced and other finite register effects are improved when compared to the corresponding SISO filter
- the number of multiplies per output is reduced when compared to the corresponding SISO filter.

In general, for a state variable filter with state updates every P samples:

$$\begin{bmatrix} x(k+P) \\ y'(k) \end{bmatrix} = \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} \begin{bmatrix} x(k) \\ u'(k) \end{bmatrix}$$

where:

$$u'(k) = \begin{bmatrix} u(k) \\ u(k+1) \\ \vdots \\ u(k+P-1) \end{bmatrix}$$

$$y'(k) = \begin{bmatrix} y(k) \\ y(k+1) \\ \vdots \\ y(k+P-1) \end{bmatrix}$$

and:

$$A' = A^P$$

$$B' = [A^{P-1}B \quad A^{P-2}B \quad \dots \quad B]$$

$$C' = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{P-1} \end{bmatrix}$$

$$D' = \begin{bmatrix} D & 0 & 0 & \dots & 0 \\ CB & D & 0 & \dots & 0 \\ CAB & CB & D & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{P-2}B & \dots & \dots & CB & D \end{bmatrix}$$

For example, if $P = 2$:

$$\begin{bmatrix} x(k+1) \\ y(k) \\ u(k+1) \end{bmatrix} = \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \\ u(k+1) \end{bmatrix}, \text{ first update} \quad (2.13)$$

$$\begin{bmatrix} x(k+2) \\ y(k) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ C & 0 & D \end{bmatrix} \begin{bmatrix} x(k+1) \\ y(k) \\ u(k+1) \end{bmatrix}, \text{ second update} \quad (2.14)$$

Roberts and Mullis [76, Table 10.2.1] list the number of multipliers-per-output for three classes of N -th order, block length P block processing state variable filters, reproduced in Table 2.1.

Filter structure	Number of multipliers per output	Optimal values for block length P
Full state-space filter	$\frac{N^2}{P} + 2N + \frac{P+1}{2}$	$N\sqrt{2}$
m -th order cascaded sections	$\left(\frac{m}{P} + 2 + \frac{P+1}{2m}\right)N$	$m\sqrt{2}$
m -th order parallel sections	$\left(\frac{m}{P} + 2 + \frac{P-1}{2m}\right)N + 1$	$m\sqrt{2}$

Table 2.1: The number of multipliers for three classes of N -th order block processing filters. Reproduced from [76, Table 10.2.1].

If we are decimating then output only $y(k)$ needs to be calculated in each block:

$$y(k) = Cx(k) + Du(k)$$

The Octave function *sv2block* converts a block length 1 state variable filter to a block length P state variable filter. The Octave function *svf* implements a state variable filter with block length 1 or more. The Octave function *Abcd2cc* generates an *oct*-file implementation of a block processing state variable filter. The Octave script *Abcd2cc_test.m* implements an 8-th order elliptic filter with block length 12 and 8 bit precision coefficients as an *oct*-file. Figure 2.6 shows the overall simulated response and Figure 2.7 compares the passband responses. The block length 12 filter is implemented as an *oct*-file. The pass-band response of the block length 1 filter is less accurate than that of the block length 12 filter. The noise gain of the block filter with truncated coefficients is accurately estimated by dividing the noise gain of the *untruncated* block length 1 filter by P . Use of the noise gain of the untruncated filter reflects the improvement in round-off noise performance obtained by block processing. The accuracy of the estimate of the noise gain of the truncated block length 1 filter improves when the number of bits is increased to 10. From Table 2.1, the number of multiply-accumulate operations per output is 81 for the block length 1 filter and 27.83 for the block length 12 filter.

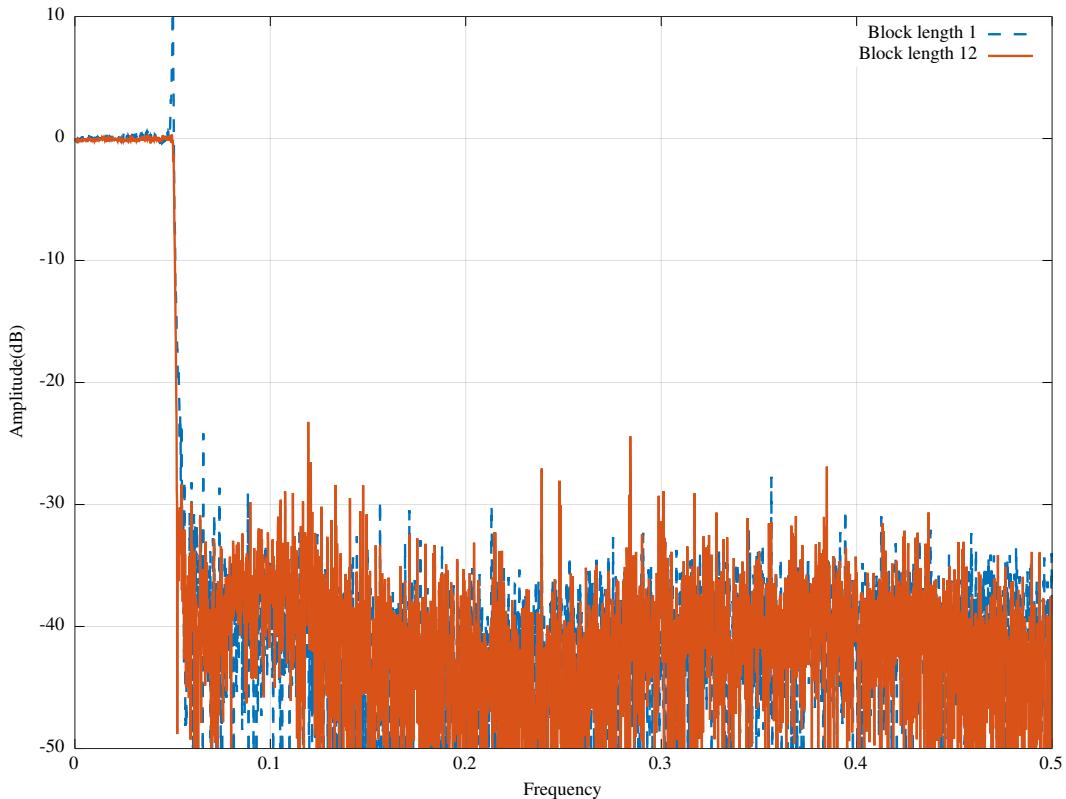


Figure 2.6: Simulated response of a 8-th order elliptic filter with 8-bit precision coefficients for block lengths of 1 and 12. The block length 12 filter is implemented as an *oct*-file.

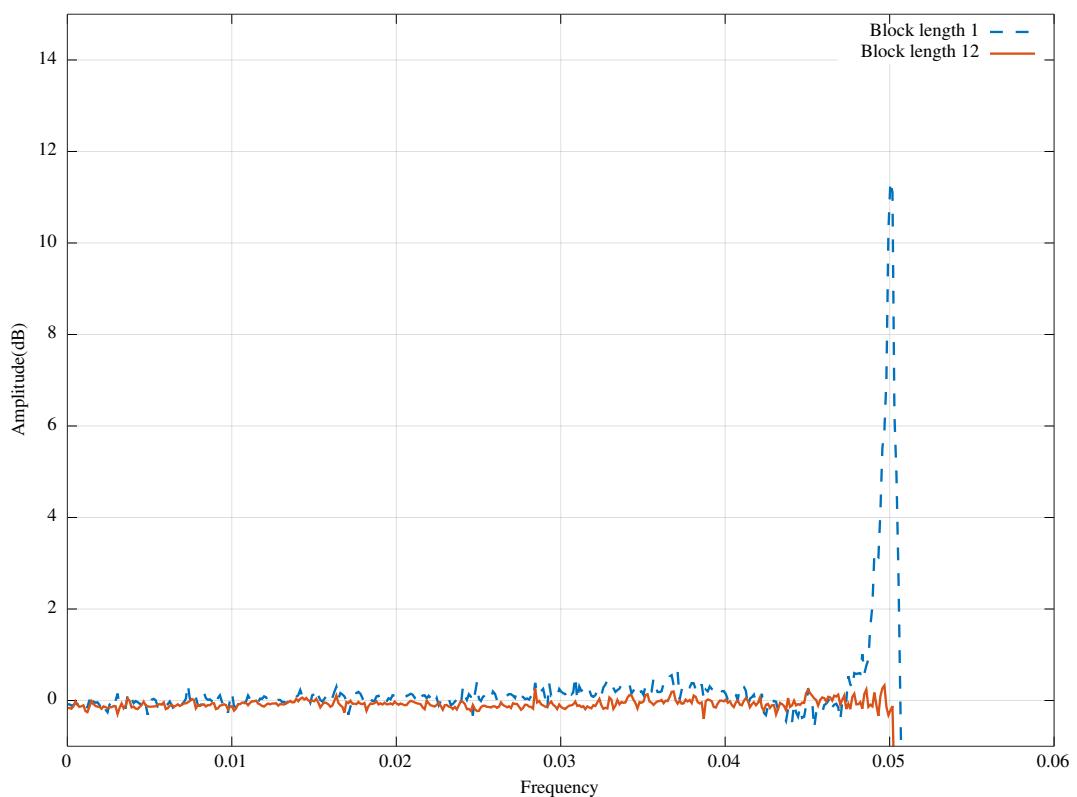


Figure 2.7: Simulated passband response of a 8-th order elliptic filter with 8-bit precision coefficients for block lengths of 1 and 12. The block length 12 filter is implemented as an *oct*-file.

2.13 Continued fraction expansion of the transfer function

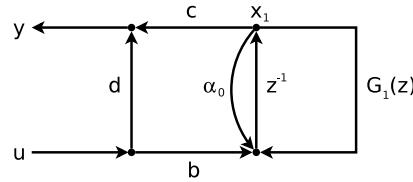
See *Roberts and Mullis* [76, Problems 8.16, 10.13 to 10.16] and *Mitra and Sherwood* [86]. Algorithm 7 calculates the continued fraction expansion of a rational transfer function.

Algorithm 7 Continued fraction expansion of a rational transfer function

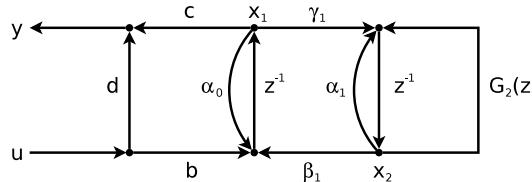
Given a rational transfer function $H(z) = \frac{\hat{b}(z)}{\hat{a}(z)} = \frac{\sum_{j=0}^N b_j z^{N-j}}{z^N + \sum_{j=1}^N a_j z^{N-j}}$ find the continued fraction expansion:

```

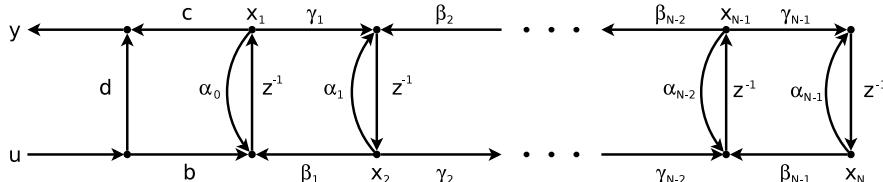
 $\hat{a}_1(z) = \hat{a}(z)$ 
 $\hat{b}_1(z) = \hat{b}(z) - b_0 \hat{a}_1(z)$ 
for  $k = 1, \dots, N$  do
    Find  $q_k(z)$  and  $\hat{b}_{k+1}(z)$  so that  $\hat{a}_k(z) = q_k(z) \hat{b}_k(z) - \hat{b}_{k+1}(z)$ 
     $\hat{a}_{k+1}(z) = \hat{b}_k(z)$ 
end for
```



(a) Signal flow graph of a first order approximation to a state variable filter



(b) Signal flow graph of a second order approximation to a state variable filter



(c) Signal flow graph of a continued fraction expansion of a state variable filter

Figure 2.8: Continued fraction expansion of a state variable filter

The continued fraction expansion of a state variable filter is shown in Figure 2.8. Figure 2.8a shows the signal flow graph for a first order approximation to a state variable filter. The state variable equations for Figure 2.8a are:

$$\begin{aligned} zX_1 &= \alpha_0 X_1 + bU + G_1 X_1 \\ Y &= cX_1 + dU \end{aligned}$$

The corresponding transfer function, $G(z)$, is:

$$G(z) = d + \frac{bc}{z - \alpha_0 - G_1(z)}$$

Figure 2.8b shows the addition of a second section for $G_1(z)$ with:

$$\begin{aligned} zX_2 &= \alpha_1 X_2 + \gamma_1 X_1 + bU + G_2 X_2 \\ Y_1 &= \beta_1 X_2 \\ G_1(z) &= \frac{\beta_1 \gamma_1}{z - \alpha_1 - G_2(z)} \end{aligned}$$

Finally, in Figure 2.8c, for a filter of order N :

$$G_{N-1}(z) = \frac{\beta_{N-1}\gamma_{N-1}}{z - \alpha_{N-1}}$$

The state variable equations for Figure 2.8c are:

$$\begin{aligned} x_1(k+1) &= \alpha_0 x_1 + \beta_1 x_2 + bu \\ x_2(k+1) &= \gamma_1 x_1 + \alpha_1 x_2 + \beta_2 x_3 \\ &\vdots \\ x_{N-1}(k+1) &= \gamma_{N-2} x_{N-2} + \alpha_{N-2} x_{N-1} + \beta_{N-1} x_N \\ x_N(k+1) &= \gamma_{N-1} x_{N-1} + \alpha_{N-1} x_N \\ y(k) &= cx_1 + du \end{aligned}$$

or:

$$\left[\begin{array}{cc} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{array} \right] = \left[\begin{array}{ccccccc|c} \alpha_0 & \beta_1 & 0 & 0 & 0 & \cdots & 0 & b \\ \gamma_1 & \alpha_1 & \beta_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \gamma_2 & \alpha_2 & \beta_3 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \gamma_3 & \alpha_3 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \gamma_{N-2} & \alpha_{N-2} & \beta_{N-1} & 0 \\ 0 & 0 & \cdots & 0 & 0 & \gamma_{N-1} & \alpha_{N-1} & 0 \end{array} \right] \quad \left[\begin{array}{ccccc|c} c & 0 & \cdots & 0 & 0 & 0 & d \end{array} \right]$$

The tridiagonal state transition matrix means that the state variable description is “pipelined”. The continued fraction expansion apparently has N additional parameters compared to the direct form but, by construction, the ratios $\frac{\alpha_i}{\gamma_i}$ and $\frac{\beta_i}{\gamma_i}$ are fixed. Furthermore, only a diagonal similarity transform maintains the tridiagonal filter structure so only state rescaling is possible. Golub and Van Loan [26, Section 9.4.3] describe *unsymmetric Lanczos tridiagonalisation* of an arbitrary matrix, summarised in Appendix I. The Octave function *lanczos_tridiag* implements *Lanczos* tridiagonalisation.

The Octave function *conffrac* implements Algorithm 7 and returns the equivalent state variable description. The Octave script *conffrac_test.m* implements the continued fraction expansion of a 5th order elliptic low-pass filter with cutoff frequency $0.05f_S$. The noise gains of the continued fraction filter and the corresponding minimum noise and direct-form state variable filter implementations are 127, 0.928 and $583e3$ respectively³.

³See Chapter 5.

Chapter 3

Frequency transformations of state variable filters

3.1 Frequency Transformations

See *Roberts and Mullis* [76, Section 6.7]. A *generalised band-pass* filter has a frequency response function $H(e^{j\theta})$ which is zero in each *stop-band* and one in each *pass-band*. Given a prototype low-pass filter $H(z)$ we wish to design a frequency transformation $F(z)$ such that the composition $G(z) = H(F(z))$ is a filter with the properties:

1. $F(z)$ should map the unit circle into itself, ie: $F(e^{j\phi}) = e^{j\theta(\phi)}$, so that $G(e^{j\phi}) = H(e^{j\theta(\phi)})$.
2. If $H(z)$ is stable and minimum phase, then $G(z)$ is stable and minimum phase. If λ is a pole (or zero) $G(z)$, then $F(\lambda)$ is a pole (or zero) of $H(z)$. Thus if $|\lambda| < 1$ implies $|F(\lambda)| < 1$ then these properties are preserved.

Consequently, the complex function $F(z)$ is a *frequency transformation* if

1. $|z| > 1 \Leftrightarrow |F(z)| > 1$
2. $|z| = 1 \Leftrightarrow |F(z)| = 1$
3. $|z| < 1 \Leftrightarrow |F(z)| < 1$

Products $F_1(z)F_2(z)$ of frequency transformations are frequency transformations. Compositions $F_1(F_2(z))$ of frequency transformations are also frequency transformations. If $F(z)$ is a frequency transformation, then $1/F(z)$ is a stable all-pass filter. In the composition $G(z) = H(F(z))$ the unit delay z^{-1} is replaced with the all-pass filter $[F(z)]^{-1}$. For IIR filter design $F(z)$ must be a rational function. A first order frequency transformation has the form

$$F(z) = \pm \frac{z - \alpha}{1 - \alpha^* z} \quad |\alpha|^2 < 1$$

This is a frequency transformation since

$$\begin{aligned} |F(z)|^2 &= \frac{(z - \alpha)(z^* - \alpha^*)}{(1 - \alpha^* z)(1 - \alpha z^*)} \\ &= 1 + \frac{zz^* - 1 + \alpha\alpha^* - \alpha\alpha^*zz^*}{1 - \alpha z^* - \alpha^* z + \alpha\alpha^*zz^*} \\ &= 1 + \frac{[|z|^2 - 1][1 - |\alpha|^2]}{|1 - \alpha^* z|^2} \end{aligned}$$

Order n frequency transformations are known as *Blaschke products*

$$F(z) = \pm \prod_{i=1}^m \left(\frac{z - \alpha_i}{1 - \alpha_i^* z} \right) = \pm \frac{p(z)}{\tilde{p}(z)}$$

where

$$p(z) = \sum_{i=0}^n p_i z^{-i} = p_0 z^{-n} \prod_{i=1}^n (z - \alpha_i)$$

has roots $|\alpha_i| < 1$ and $\tilde{p}(z) = z^{-n} p(z^{-1})$.

Suppose the prototype $H(z)$ has cut-off frequency $\theta = \pi/2$ and $G(z)$ has the pass band edges shown in Figure 3.1 (where $G(z)$ is low-stop).

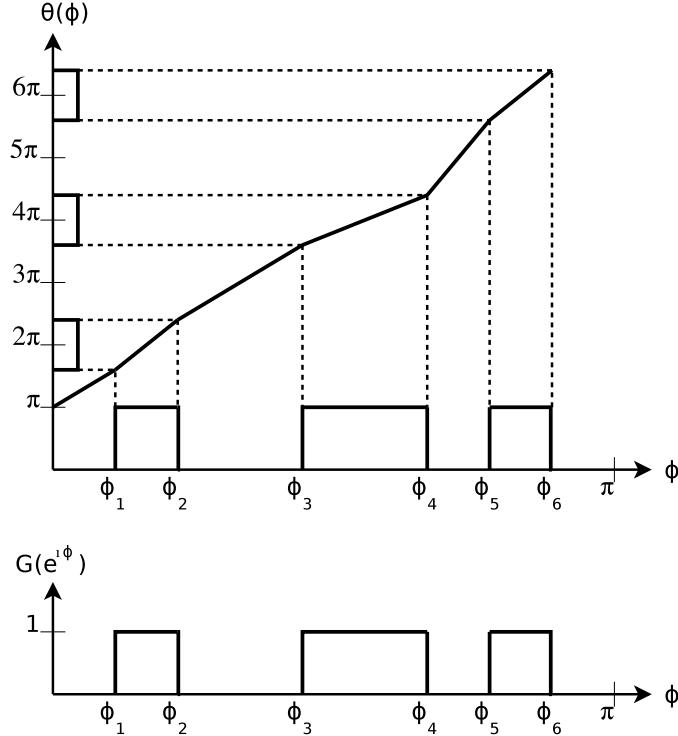


Figure 3.1: Frequency warping function (after *Roberts and Mullis* [76, Figure 6.7.2])

$F(z)$ must map the band edges as shown by the frequency warp $\theta(\phi)$. There are two cases:

$$\text{low-pass : } \begin{cases} F(1) = 1 \\ \theta(0) = 0 \\ \theta_k = \theta(\phi_k) = (k - \frac{1}{2})\pi \end{cases}$$

$$\text{low-stop : } \begin{cases} F(1) = -1 \\ \theta(0) = \pi \\ \theta_k = \theta(\phi_k) = (k + \frac{1}{2})\pi \end{cases}$$

Roberts and Mullis [76, Figure 6.7.3] show an algorithm for calculating the parameters of the frequency transformation $F(z)$. This algorithm is implemented by the Octave [46] function *phi2p*.

3.2 Frequency Transformation of the Transfer Function

See *Roberts and Mullis* [76, Problem 6.26]. Given a low-pass prototype:

$$\begin{aligned} H(z) &= \frac{\beta(z)}{\alpha(z)} \\ &= \frac{\sum_{k=0}^L \alpha_k z^{-k}}{\sum_{k=0}^L \beta_k z^{-k}} \end{aligned}$$

and a frequency transformation function

$$F(z) = \frac{p(z)}{\tilde{p}(z)}$$

where $p(z) = \sum_{k=0}^M p_k z^{-k}$ and $\tilde{p}(z) = z^{-M} p(z^{-1})$, find the filter

$$G(z) = H(\sigma F(z)) = \frac{b(z)}{a(z)}$$

where $\sigma = 1$ for low-pass and $\sigma = -1$ for low-stop.

First expand $a(z)$ and $b(z)$ in $\alpha(z)$, $\beta(z)$ and $p(z)$:

$$\begin{aligned} \frac{a(z)}{b(z)} &= \frac{\sum_{k=0}^{LM} a_k z^{-k}}{\sum_{k=0}^{LM} b_k z^{-k}} \\ &= \frac{\sum_{k=0}^L \alpha_k [\sigma p(z)/\tilde{p}(z)]^{-k}}{\sum_{k=0}^L \beta_k [\sigma p(z)/\tilde{p}(z)]^{-k}} \\ &= \frac{\sum_{k=0}^L \alpha_k [\sigma z^{-M} p(z^{-1})]^k [p(z)]^{L-k}}{\sum_{k=0}^L \beta_k [\sigma z^{-M} p(z^{-1})]^k [p(z)]^{L-k}} \end{aligned}$$

A common factor of $[p(z)]^L$ has been introduced. Next choose $N \geq LM+1$ and define the following *Discrete Fourier Transform* pairs:

$$\begin{bmatrix} p_0 & \cdots & p_M & 0 & \cdots & 0 \end{bmatrix} \xrightarrow{DFT} \begin{bmatrix} P(0) & P(1) & \cdots & P(N-1) \end{bmatrix} \\ \begin{bmatrix} a_0 & \cdots & a_{LM} & 0 & \cdots & 0 \end{bmatrix} \xrightarrow{DFT} \begin{bmatrix} A(0) & A(1) & \cdots & A(N-1) \end{bmatrix} \\ \begin{bmatrix} b_0 & \cdots & b_{LM} & 0 & \cdots & 0 \end{bmatrix} \xrightarrow{DFT} \begin{bmatrix} B(0) & B(1) & \cdots & B(N-1) \end{bmatrix} \end{math>$$

where

$$\begin{aligned} A(n) &= \sum_{k=0}^{N-1} a_k W_N^{-nk} \\ &= a(W_N^n) \\ &= \sum_{k=0}^L \alpha_k \sigma^k W_N^{-knM} e^{-2k\angle\{P(n)\}} \{P(n)\}^L \\ &= \alpha(e^{j\theta_n}) \{P(n)\}^L \end{aligned}$$

and

$$\theta_n = \frac{2\pi n M}{N} + \left(\frac{1-\sigma}{2} \right) \pi + 2\angle\{P(n)\}$$

Here $W_N = e^{j\frac{2\pi}{N}}$ and \angle means “angle of”. Similarly

$$B(n) = \beta(e^{j\theta_n}) \{P(n)\}^L$$

This algorithm is implemented by the Octave function *tfp2g.m*.

3.3 Frequency Transformations of State Variable Filters

See *Mullis and Roberts* [16, Section 6.7]. Given a (usually low-pass) prototype filter $H(z)$ parameterised by the state variable description $\{A, b, c, d\}$ construct the filter $G(z) = H(F(z))$ where

$$F(z) = \pm \prod_{i=1}^m \left(\frac{z - \alpha_i^*}{1 - \alpha_i z} \right), \quad |\alpha_i| < 1$$

If the order of $H(z)$ is n then the order of $G(z)$ is mn . The map $z \rightarrow F(z)$ preserves the disk $|z| < 1$, the circle $|z| = 1$ and the set $|z| > 1$ since

$$1 - \left| \frac{z - \alpha^*}{1 - \alpha z} \right|^2 = \frac{(1 - |z|^2)(1 - |\alpha|^2)}{|1 - \alpha z|^2}$$

Therefore if $H(z)$ is stable (minimum phase) then $G(z)$ is stable (minimum phase). We want to find the matrices $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \mathfrak{D}$ such that

$$\mathfrak{D} + \mathfrak{C}(zI - \mathfrak{A})^{-1} \mathfrak{B} = G(z) = H(F(z))$$

Let matrices $\{\alpha, \beta, \gamma, \delta\}$ parameterise the filter $1/F(z)$ so that

$$\frac{1}{F(z)} = \delta + \gamma(zI - \alpha)^{-1} \beta$$

Note that $1/F(z)$ is a stable, all-pass filter.

A heuristic construction of $H(F(z))$ follows. The filter $1/F(z)$ replaces each delay branch in the original filter, $H(z)$, so for a $m \times n$ matrix $S(k)$

$$\begin{aligned} S(k+1) &= \alpha S(k) + \beta [Ax(k) + bu(k)]^T \\ x(k) &= S^T(k) \gamma^T + \delta [Ax(k) + bu(k)] \\ y(k) &= cx(k) + du(k) \end{aligned}$$

Eliminating $x(k)$ gives

$$\begin{aligned} S(k+1) &= \alpha S(k) + \beta \gamma S(k) \left[A(I - \delta A)^{-1} \right]^T + \beta \left[(I - \delta A)^{-1} \right]^T u(k) \\ y(k) &= \gamma S(k) \left[c(I - \delta A)^{-1} \right]^T + \left[d + \delta c(I - \delta A)^{-1} b \right] u(k) \end{aligned}$$

Define $V(X)$ as the vector formed by stacking the columns of X and define $G \otimes F$ as the *Kronecker product*¹, $\{G_{ij}F\}$, of the matrixes G and F . Then:

$$\begin{aligned} V(S(k+1)) &= \mathfrak{A}V(S(k)) + \mathfrak{B}u(k) \\ y(k) &= \mathfrak{C}V(S(k)) + \mathfrak{D}u(k) \end{aligned}$$

where

$$\begin{aligned} \mathfrak{A} &= I \otimes \alpha + A(I - \delta A)^{-1} \otimes \beta \gamma \\ \mathfrak{B} &= (I - \delta A)^{-1} b \otimes \beta \\ \mathfrak{C} &= c(I - \delta A)^{-1} \otimes \gamma \\ \mathfrak{D} &= d + \delta c(I - \delta A)^{-1} b \end{aligned}$$

and

$$H(F(z)) = \mathfrak{D} + \mathfrak{C}(zI - \mathfrak{A})^{-1} \mathfrak{B} \quad (3.1)$$

Mullis and Roberts [16] give a direct proof. Start with the identity:

$$(fI - A)b \otimes \beta = f \left[b \otimes \beta - Ab \otimes \beta \left(\frac{1}{f} \right) \right]$$

¹ Define $V(X)$ as the vector formed by stacking the columns of X . Suppose the matrix product FXG^T is defined. This represents a linear transformation applied to X . When this transformation is arranged as a vector

$$V(FXG^T) = [G \otimes F]V(X)$$

where $G \otimes F$ is defined as the *Kronecker product*, $\{G_{ij}F\}$, of matrices G and F . The Kronecker product is defined for any two matrices and satisfies the following:

$$\begin{aligned} [A \otimes B][C \otimes D] &= (AC) \otimes (BD) \\ (A + B) \otimes (C + D) &= A \otimes C + A \otimes D + B \otimes C + B \otimes D \\ [A \otimes B]^{-1} &= A^{-1} \otimes B^{-1} \\ [A \otimes B]^T &= A^T \otimes B^T \end{aligned}$$

By convention Kronecker products are performed *after* ordinary matrix products and *before* matrix addition.

Where, for convenience the scalar $F(z) = f$. So

$$\begin{aligned} b \otimes \beta &= f \left[(fI - A)^{-1} b \otimes \beta - A (fI - A)^{-1} b \otimes \beta \left(\delta + \gamma (zI - \alpha)^{-1} \beta \right) \right] \\ &= f [I \otimes (zI - \alpha) - \delta A \otimes (zI - \alpha) - A \otimes \beta \gamma] \cdot \left[(fI - A)^{-1} b \otimes (zI - \alpha)^{-1} \beta \right] \\ &= [(I - \delta A) \otimes (zI - \alpha) - A \otimes \beta \gamma] \cdot \left[f (fI - A)^{-1} b \otimes (zI - \alpha)^{-1} \beta \right] \end{aligned}$$

Therefore

$$\begin{aligned} \mathfrak{B} &= (I - \delta A)^{-1} b \otimes \beta \\ &= \left[I \otimes (zI - \alpha) - (I - \delta A)^{-1} A \otimes \beta \gamma \right] \cdot \left[f (fI - A)^{-1} b \otimes (zI - \alpha)^{-1} \beta \right] \\ &= (zI - \mathfrak{A}) \left[f (fI - A)^{-1} b \otimes (zI - \alpha)^{-1} \beta \right] \end{aligned}$$

so, substituting $(zI - \mathfrak{A})^{-1} \mathfrak{B}$

$$\begin{aligned} \mathfrak{D} + \mathfrak{C} (zI - \mathfrak{A})^{-1} \mathfrak{B} &= \left[d + \delta c (I - \delta A)^{-1} b \right] + \left[c (I - \delta A)^{-1} \otimes \gamma \right] \cdot \left[f (fI - A)^{-1} b \otimes (zI - \alpha)^{-1} \beta \right] \\ &= d + \delta c (I - \delta A)^{-1} b + f \left[c (I - \delta A)^{-1} (fI - A)^{-1} b \right] \cdot \left[\gamma (zI - \alpha)^{-1} \beta \right] \\ &= d + c (I - \delta A)^{-1} \left[\delta (fI - A) + f \left(\frac{1}{f} - \delta \right) I \right] (fI - A)^{-1} b \\ &= d + c (fI - A)^{-1} b \\ &= H(f) \\ &= H(F(z)) \end{aligned}$$

This algorithm is implemented by the Octave function *tfp2Abcd*.

3.4 An example: frequency transformations of a 5-th order elliptic filter

The Octave script *tfp2g_test.m* shows examples of frequency transformations of a 5th order elliptic low-pass filter. Figure 3.2 shows the prototype filter. Figure 3.3 shows the result of a low-pass to low-pass transformation. Figure 3.4 shows the result of a low-pass to high-pass transformation. Figure 3.5 shows the result of a low-pass to band-pass transformation. Figure 3.6 shows the result of a low-pass to triple band-stop transformation.

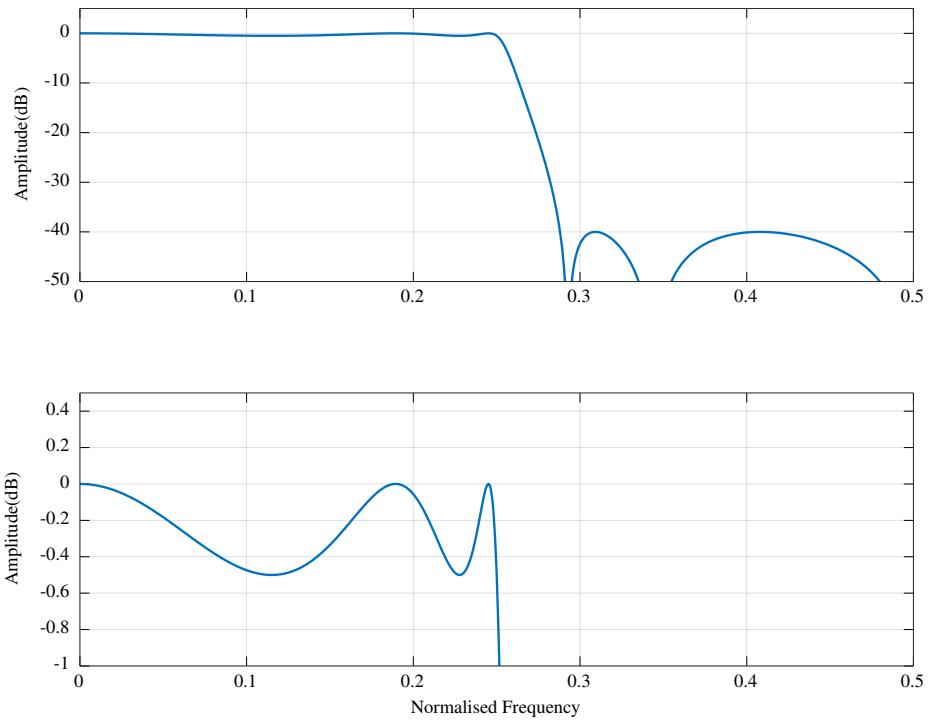


Figure 3.2: Prototype 5-th order elliptic low-pass filter

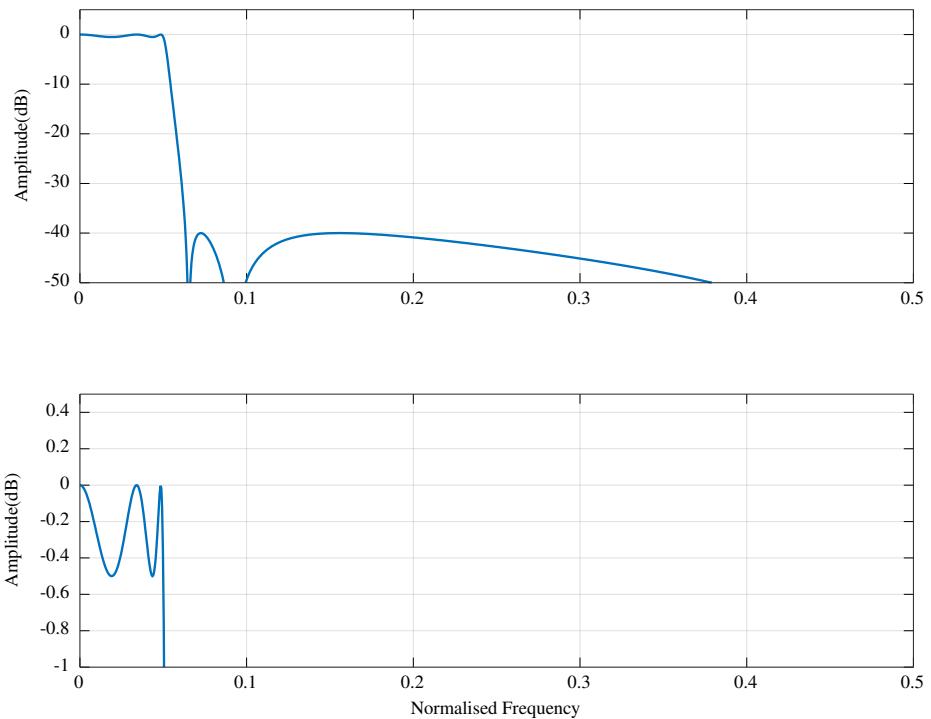


Figure 3.3: Low-pass to low-pass transformation of a 5-th order elliptic low-pass filter

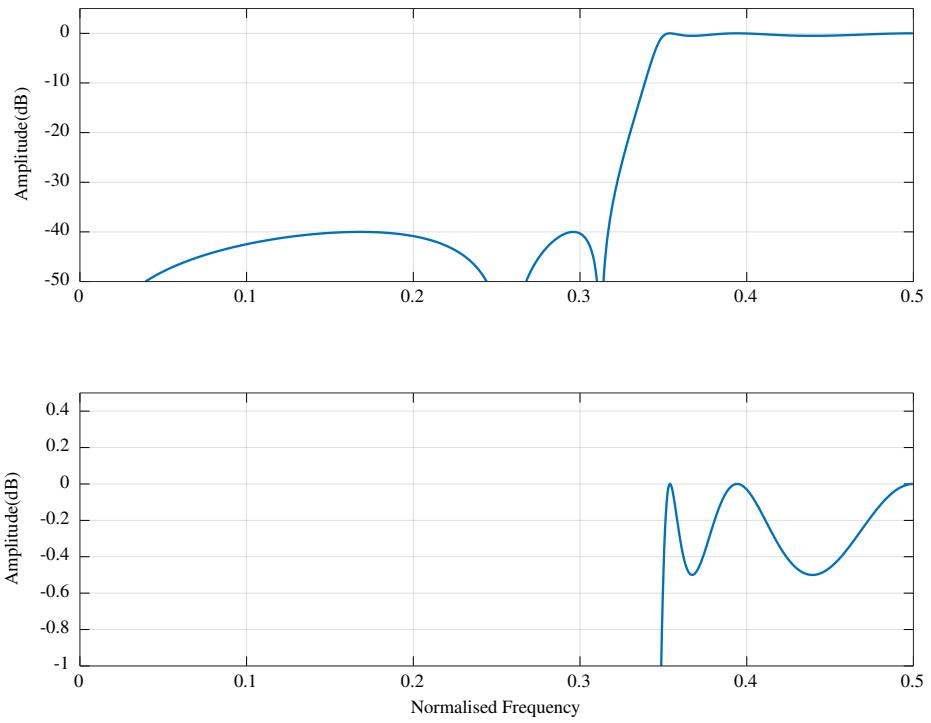


Figure 3.4: Low-pass to high-pass transformation of a 5-th order elliptic low-pass filter

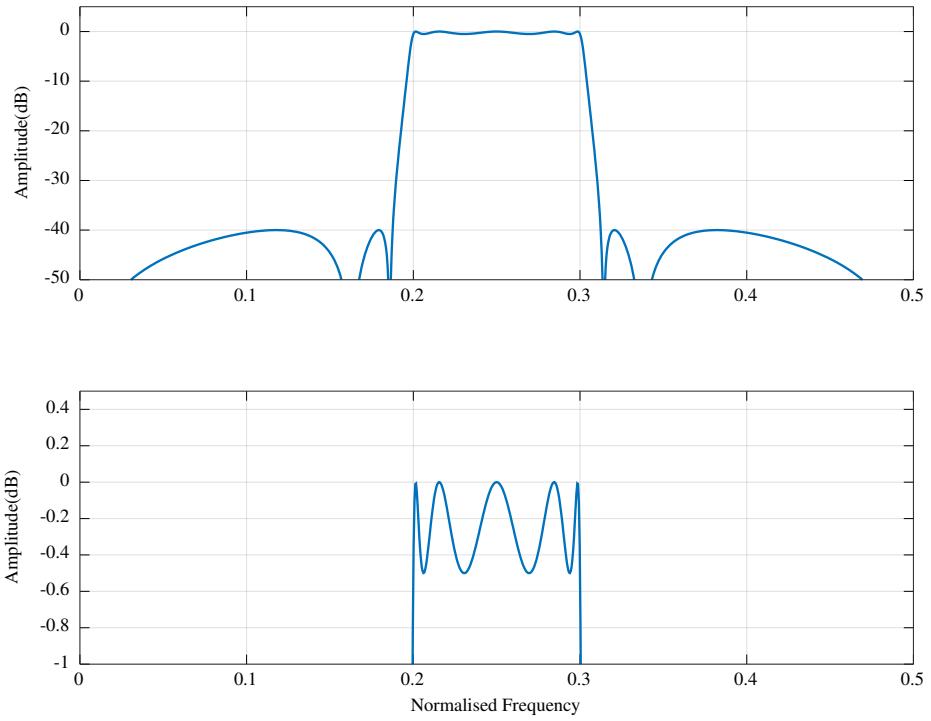


Figure 3.5: Low-pass to band-pass transformation of a 5-th order elliptic low-pass filter

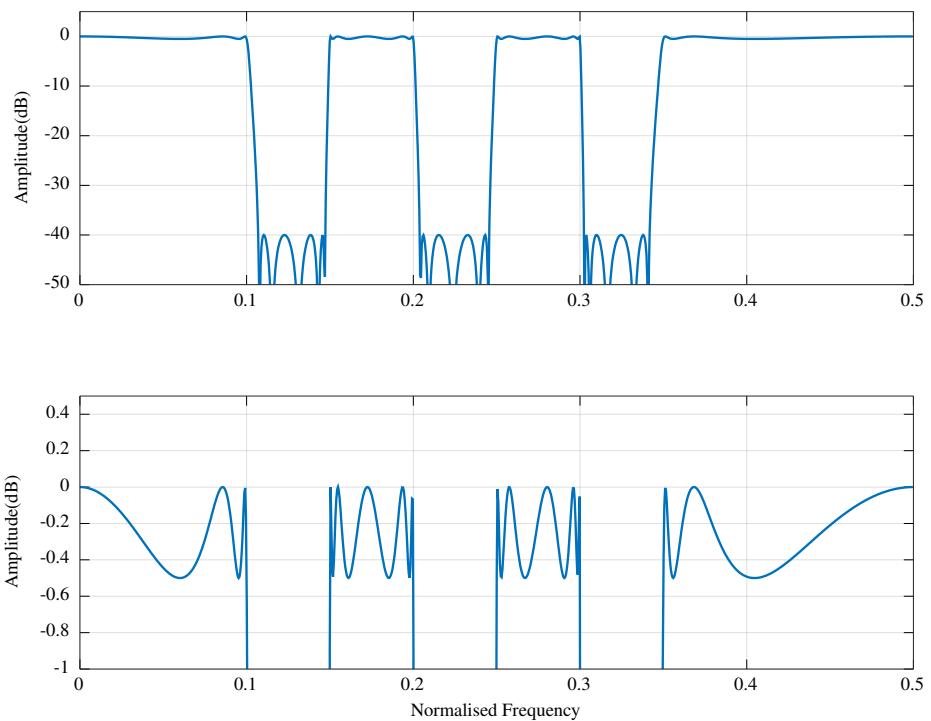


Figure 3.6: Low-pass to triple band-stop transformation of a 5-th order elliptic low-pass filter

Chapter 4

Factored state variable descriptions

See *Roberts and Mullis* [76, Section 8.4]. Equation 2.4 assumes that the left-hand side is calculated concurrently. If that is not the case, then the *factored* state variable description is more appropriate. This models individual computations that are done separately but in a fixed cyclic order. The number of factors is the number of delay free paths in the PSFG description. For L factors

$$\begin{aligned} q_0(k) &= \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \\ q_i(k) &= F_i q_{i-1}(k), 1 \leq i \leq L \\ \begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix} &= q_L(k) \end{aligned}$$

The corresponding state variable description is

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = F_L F_{L-1} \dots F_1$$

4.1 Construction of the factored state variable description

To construct a FSVD from a PSFG, first find the sets $\{S_0, S_1, \dots, S_L\}$ of node variables and let v_k be a vector representing the nodes in S_k . In particular:

$$v_0 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ u \end{bmatrix}$$

where the x_k are the outputs of the unit delays. Each variable in S_k is a linear combination of the variables in

$$\{S_0 \cup S_1 \cup \dots \cup S_{k-1}\}$$

so there is a coefficient matrix G_k for which:

$$v_k = G_k \begin{bmatrix} v_{k-1} \\ v_{k-2} \\ \vdots \\ v_0 \end{bmatrix} \quad 1 \leq k \leq L$$

Finally let G_0 be a matrix that picks out the next-state and outputs:

$$\begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \\ y \end{bmatrix} = G_0 \begin{bmatrix} v_L \\ v_{L-1} \\ \vdots \\ v_0 \end{bmatrix}$$

and the FSVD is:

$$\begin{bmatrix} x' \\ y \end{bmatrix} = G_0 \begin{bmatrix} G_L \\ I \end{bmatrix} \begin{bmatrix} G_{L-1} \\ I \end{bmatrix} \dots \begin{bmatrix} G_1 \\ I \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

4.1.1 Factoring cascaded state variable filters

Figure 4.1 shows two state variable filters connected in cascade. The node variable sets are:

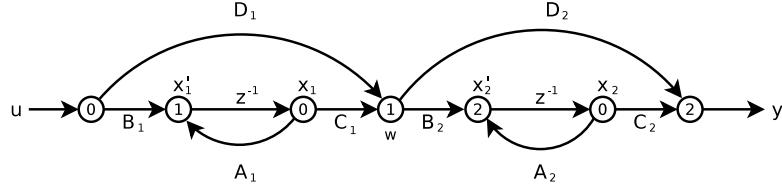


Figure 4.1: Signal flow graph of two cascaded state variable filters

$$S_0 = \{x_1, x_2, u\}$$

$$S_1 = \{w, x'_1\}$$

$$S_2 = \{y, x'_2\}$$

where:

$$\begin{bmatrix} w \\ x'_1 \\ x_1 \\ x_2 \\ u \end{bmatrix} = \begin{bmatrix} C_1 & 0 & D_1 \\ A_1 & 0 & B_1 \\ I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u \end{bmatrix}$$

$$\begin{bmatrix} y \\ x'_2 \\ w \\ x'_1 \\ x_1 \\ x_2 \\ u \end{bmatrix} = \begin{bmatrix} D_2 & 0 & 0 & C_2 & 0 \\ B_2 & 0 & 0 & A_2 & 0 \\ I & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} w \\ x'_1 \\ x_1 \\ x_2 \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ x'_2 \\ w \\ x'_1 \\ x_1 \\ x_2 \\ u \end{bmatrix}$$

The factored state variable equations are:

$$\begin{bmatrix} x'_1 \\ x'_2 \\ y \end{bmatrix} = \begin{bmatrix} 0 & I & 0 \\ B_2 & 0 & A_2 \\ D_2 & 0 & C_2 \end{bmatrix} \begin{bmatrix} C_1 & 0 & D_1 \\ A_1 & 0 & B_1 \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u \end{bmatrix}$$

The state variable description is:

$$\{A, B, C, D\} \leftarrow \left(\begin{bmatrix} A_1 & 0 \\ B_2 C_1 & A_2 \end{bmatrix}, \begin{bmatrix} B_1 \\ B_2 D_1 \end{bmatrix}, [D_2 C_1 \quad C_2], [D_2 D_1] \right)$$

4.1.2 Factoring block state variable filters

Section 2.12 shows the state variable filter equations for block processing of the input. The example shown in Equation 2.13 processes two inputs at a time to produce two outputs with a factored state variable description of:

$$\begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ C & 0 & D \end{bmatrix} \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & I \end{bmatrix}$$

4.1.3 Factoring a feedback connection of state variable filters

Suppose the state variable description of a *controller* is:

$$\begin{bmatrix} x'_C \\ y_C \end{bmatrix} = \begin{bmatrix} A_C & B_C \\ C_C & D_C \end{bmatrix} \begin{bmatrix} x_C \\ u_C \end{bmatrix}$$

and that the state variable description of the corresponding *plant* is:

$$\begin{bmatrix} x'_P \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & 0 \end{bmatrix} \begin{bmatrix} x_P \\ u_1 \\ u_2 \end{bmatrix}$$

where u_1 is the external input to the plant, y_1 is the external output of the plant, u_2 is the controller input to the plant and y_2 is the plant output to the controller. Note that y_2 does not depend directly on u_2 .

Apply the feedback connection between the plant and controller:

$$\begin{aligned} u_2 &= y_C \\ u_C &= y_2 \end{aligned}$$

so that, for the controller:

$$u_C = C_2 x_P + D_{21} u_1 y_C = C_C x_C + D_C y_2$$

After substituting and rearranging, the state variable equations for the feedback connection are:

$$\begin{bmatrix} x'_P \\ x'_C \\ y_1 \end{bmatrix} = \begin{bmatrix} A + B_2 D_C C_2 & B_2 C_C & B_1 + B_2 D_C D_{21} \\ B_C C_2 & A_C & B_C D_{21} \\ C_1 + D_{12} D_C C_2 & D_{12} C_C & D_{11} + D_{12} D_C D_{21} \end{bmatrix} \begin{bmatrix} x_P \\ x_C \\ u_1 \end{bmatrix}$$

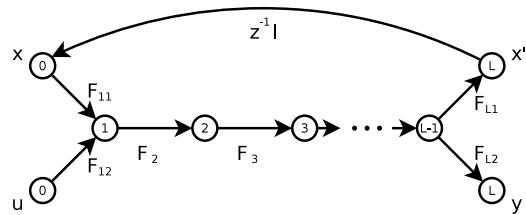
4.2 Factored state variable filters with fractional delays

Figure 4.2a shows the signal flow graph of a factored state variable filter. There are L factors and the longest delay free path has length L . The factors F_1 and F_L are:

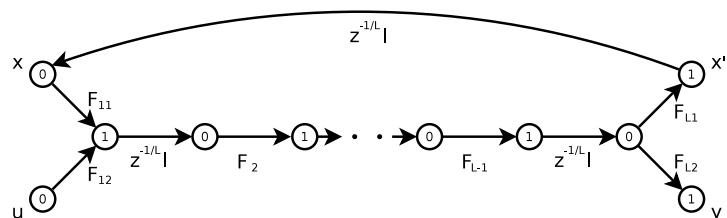
$$\begin{aligned} F_1 &= [F_{11} \quad F_{12}] \\ F_L &= [F_{L1} \\ F_{L2}] \end{aligned}$$

Figure 4.2b shows the result of retiming the filter with a delay of $z^{-1/L}$ after each factor. The loop gain and therefore the transfer function $H(z)$ are unchanged. The longest delay free path is now 1.

The factored and retimed filter has a reduced delay free path at the expense of a higher sub-sampling clock rate.



(a) Signal flow graph of a factored state variable filter



(b) Signal flow graph of a retimed factored state variable filter

Figure 4.2: Factored state variable filters (See [76, Figure 8.4.6].)

Chapter 5

Round-off noise in state variable filters

This chapter is based on Chapter 9 of *Roberts and Mullis* [76].

5.1 Quantisation noise in digital filters

The binary number generated by an ADC is assumed to be a fixed point number. Most often this number is represented in 2's complement format because that is easily implemented. The 2's complement representation of a real number, r , in a finite length register of B bits is:

$$[r]_Q = \Delta \left(-b_0 + \sum_{m=1}^{B-1} b_m 2^{-m} \right)$$

where:

$$\begin{aligned}\Delta &\leq r < \Delta \\ b_0 &= 1 \text{ for } r \leq 0 \\ b_0 &= 0 \text{ for } r \geq 0\end{aligned}$$

Here Δ is the maximum voltage represented and b_0 is the sign bit. The quantization step size is $q = \Delta 2^{-B+1}$. If the quantiser rounds to the nearest integer multiple of q then the error in the representation of r can be considered to be uncorrelated from sample to sample and uniformly distributed in the range $(-\frac{q}{2}, \frac{q}{2})$ with variance¹ $\sigma_e^2 = \frac{q^2}{12}$. This assumes that the frequency spectrum of the input, $u(k)$, of the quantiser is reasonably broad and that the probability density function of $u(k)$ is broad relative to q so that several quantization levels are crossed between samples of $u(k)$. The signal-to-noise ratio obtained depends on the statistics of both the quantization noise and the signal that is quantised, $u(k)$. If $u(k)$ is uniformly distributed over the range $[-\Delta, \Delta]$ then the variance of $u(k)$ is $\sigma_u^2 = \frac{1}{3}$ and the required number of bits, B , to obtain a signal to noise ratio of SN_Q is $b = \frac{\log_2 SN_Q}{2}$.

2's complement arithmetic does rounding-toward-zero rounding, also called magnitude truncation. The quantisation error is uniformly distributed in the range $(-q, q)$ with variance $\sigma_e^2 = \frac{q^2}{3}$. For positive inputs the error is negative and for negative inputs the error is positive. This distribution corresponds to a step input of $\frac{q}{2}$ at the state round off noise error input to the filter. Modern digital signal processing ICs use a few gates to provide a rounding-to-nearest arithmetic mode. If this is not present software rounding may be required to give adequate noise performance. The results presented in the following sections assume the rounding-to-nearest quantiser. Note that the 2's complement overflow characteristic has the useful property that immediate overflows in an accumulator cancel whenever the resulting sum is in range. An example will demonstrate the effect of the rounding-to-minus-infinity quantiser.

¹The probability density function of the quantisation error of r , e , is $p_e(x) = q^{-1}$ and the mean error is $\mu_e = 0$. The variance of the error is the *second central moment* of the pdf:

$$\sigma_e = E\{[e - \mu_e]^2\} = \int_{-\infty}^{\infty} (x - \mu_e)^2 p_e(x) dx = \int_{-q/2}^{q/2} \frac{x^2}{q} dx = \frac{q^2}{12}$$

5.2 Limit-cycle oscillations in digital filters

See *Roberts and Mullis* [76, Section 9.3]. If numbers are represented as 2's complement integers and there is no provision for explicitly detecting and correcting overflows then, depending on how the filter is scaled, there is a possibility that internal registers will overflow. What happens then depends on:

- the nature of the input
- the filter realization
- the number representation used in the filter
- the overflow characteristic used in the filter

The direct form filter has the minimum number of multipliers for a second order filter. However, with a 2's complement overflow characteristic the output of the filter after an overflow can, depending on the poles of the filter, become independent of the input sequence. This condition is called overflow oscillation. A second order direct form filter is free of overflow oscillations provided:

$$|\alpha_1| + |\alpha_2| \leq 1$$

where $1 + \alpha_1 z^{-1} + \alpha_2 z^{-2}$ is the denominator of the transfer function. Many other realizations are free of overflow oscillations by design.

5.3 State variable filters and wide sense stationary inputs

5.3.1 The filter state covariance matrix

Assume a causal filter $H(z)$ driven by white, wide-sense stationary unit variance white noise inputs, $u(k)$. The covariance matrix of the filter state is:

$$\begin{aligned} K &= E\{x(k)x^T(k)\} \\ &= E\{x(k+1)x^T(k+1)\} \\ &= E\{(Ax(k) + Bu(k))(Ax(k) + Bu(k))^T\} \\ &= E\{Ax(k)x^T(k)A^T + Bu(k)u^T(k)B^T\} \\ &= AKA^T + BB^T \end{aligned}$$

This is known as a *discrete Lyapunov equation*. Alternatively, since

$$x(k) = \sum_{l=0}^{\infty} A^l Bu(k-l-1)$$

the state covariance matrix is

$$K = \sum_{m=0}^{\infty} \sum_{l=0}^{\infty} A^l B r_{uu}(l-m) (A^m B)^T$$

For unit variance white inputs

$$K = \sum_{l=0}^{\infty} (A^l B) (A^l B)^T$$

Algorithm 8 is a simple recursive calculation of the covariance matrix implemented in the Octave function *dlyap_recursive*.

Alternative methods for calculating the covariance matrix are:

- find the auto-correlation sequence of the characteristic equation of the state transition matrix with the inverse-Levinson recursion. See *Roberts and Mullis* [76, Section 9.11, p. 393]. The Octave function *dlyap_levinson* implements this solution.
- use *Hammarling*'s solution of the discrete Lyapunov equation [84]. The Octave *Control* package contains the *dlyap* function that calls functions from an open-source version of the *SLICOT* library [24] to implement *Hammarling*'s algorithm.

Algorithm 8 Recursive calculation of the covariance matrix

```

 $F \leftarrow A$ 
 $K \leftarrow BB^T$ 
repeat
     $K \leftarrow FKF^T + K$ 
     $F \leftarrow F^2$ 
until  $F = 0$ 

```

5.3.2 The output response to white noise in a state variable

The unit pulse response is a cross-correlation of the input and output

$$h(k) = E\{y(k+l)u(l)\}$$

The autocorrelation sequence for the output is

$$\begin{aligned} r_{yy}(k) &= E\{y(k+l)y(l)\} \\ &= \sum_{l=0}^{\infty} h(k+l)h(l) \end{aligned}$$

Using a state variable description

$$h(k) = \begin{cases} 0 & k < 0 \\ D & k = 0 \\ CA^{k-1}B & k > 0 \end{cases} \quad \text{DFT} \quad H(e^{j\theta}) = D + C(e^{j\theta}I - A)^{-1}B$$

Since

$$r_{yy}(k) \quad \text{DFT} \quad S_{yy}(\theta) = |H(e^{j\theta})|^2$$

we can express the output autocorrelation sequence directly from $\{A, B, C, D\}$. The output autocorrelation sequence is

$$\begin{aligned} r_{yy}(k) &= E\{y(k+l)y^T(l)\} \\ &= E\{(Cx(k+l) + Du(k+l))(Cx(l) + Du(l))^T\} \\ &= E\{Cx(k+l)x^T(l)C^T + Du(k+l)x^T(l)C^T + Cx(k+l)u^T(l)D^T + Du(k+l)u^T(l)D^T\} \end{aligned}$$

The first term is

$$\begin{aligned} E\{x(k+l)x^T(l)\} &= E\left\{\left(\sum_{m=0}^{\infty} A^m Bu(k+l-m-1)\right)\left(\sum_{m'=0}^{\infty} A^{m'} Bu(l-m'-1)\right)^T\right\} \\ &= \sum_{m=0}^{\infty} \sum_{m'=0}^{\infty} (A^m B)(A^{m'} B)^T r_{uu}(k-m-m') \\ &= \sum_{m=0}^{\infty} A^k A^m B (A^m B)^T \\ &= A^k K \end{aligned}$$

where we have used the fact that $r_{uu}(k-m+m') = \delta(k-m+m')$. The cross-correlation between the state and a white noise input is

$$\begin{aligned} E\{x(k+l)u(l)\} &= E\left\{\sum_{j=0}^{\infty} A^j Bu(k+l-j-1)u(l)\right\} \\ &= \sum_{j=0}^{\infty} A^j B \delta(k-j-1) \\ &= \begin{cases} A^{k-1} B & k > 0 \\ 0 & k \leq 0 \end{cases} \end{aligned}$$

(This is also the unit-pulse response from the input to internal states.) The present state is uncorrelated with future inputs so the term containing $u(k+l)x^T(k)$ is zero. Finally,

$$\begin{aligned} r_{yy}(k) &= CA^k KC^T + CA^{k-1} BD^T \\ &= CA^k KC^T + h(k) D^T \end{aligned}$$

The energy in the unit-pulse response is given by

$$\begin{aligned} r_{yy}(0) &= E\{y^2(k)\} \\ &= \sum_{l=0}^{\infty} h^2(l) \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\theta})|^2 d\theta \\ &= D^2 + CKC^T \end{aligned}$$

5.3.3 Scaling State Variable Filters To Avoid Overflow

Round off noise is the dominant component of output error in digital filters only when overflows in internal registers are negligible. Overflows are avoided by properly scaling the realization. For a fixed point number representation the range of internal variables is:

$$|v(k)| \leq \Delta$$

The magnitude of Δ depends on the quantization step size and the number of bits. The unit pulse response from the input, $u(k)$ to an internal variable, $v(k)$, can be written:

$$v(k) = (f * u)(k)$$

where f is the unit pulse response from the input to $v(k)$ and $*$ is the convolution operator. The range of values of v depends on the nature of the input and on the sequence f .

For sinusoidal inputs:

$$\begin{aligned} u(k) &= \cos(k\theta) \\ |v(k)| &\leq \max |F(e^{j\theta})| \end{aligned}$$

where F is the transfer function from u to v .

For bounded inputs:

$$\begin{aligned} |u(k)| &\leq 1 \\ |v(k)| &\leq \sum_l |f(l)| = \|f\|_1 \end{aligned}$$

The right side of the expression for the range of v is known as the l_1 -norm of f . These are called “bang-bang” controller inputs in control theory. For filter design the l_1 norm is usually far too conservative.

For finite energy inputs:

$$\begin{aligned} \sum_{l \leq k} u^2(l) &\leq 1 \\ |v(k)| &\leq \left[\sum_l f^2(l) \right]^{\frac{1}{2}} = \|f\|_2 \end{aligned}$$

The right side of the expression for the range of v is known as the l_2 -norm of f .

To reconcile these bounds on the range of v with the constraint on the size of v given above we must constrain the “gain” of the unit-pulse response sequences for the internal variables. The l_2 -norm scaling rule is:

$$\delta \|f\|_2 = \delta \left[\sum_l f^2(l) \right]^{\frac{1}{2}} = 1$$

The parameter δ is chosen subjectively. It can be interpreted as the number of standard deviations representable in the register containing v if the input is unit-variance white noise. A good value for δ is 4.

The diagonal components of the state variable covariance matrix, K , given above are related to the l_2 -norm for each state variable, $x(k)$, by:

$$K_{ii} = \sum_{k=0}^{\infty} f_i^2(k) = \|f_i\|_2$$

so the scaling constraint applied to the i -th component of the state vector is:

$$\delta \sqrt{K_{ii}} = 1, \quad i = 1, 2, \dots, n$$

This can be achieved by applying the coordinate transformation:

$$T^{-1} = \text{Diag} \left[\frac{1}{t_1}, \dots, \frac{1}{t_n} \right]$$

where:

$$t_i = \delta \sqrt{K_{ii}}, \quad i = 1, 2, \dots, n$$

A geometric interpretation of scaling is as follows: For unit variance stationary Gaussian inputs the set of values of the state variables $\{x : x^T K^{-1} x \leq 1\}$ represents the “one standard deviation error ellipsoid” in the filter state

5.4 Estimation of output round-off noise in state variable filters

The z-domain state variable equations with round off noise inputs can be represented by the following signal flow graph shown in Figure 5.1. Where e models the round off noise due to calculation of the state vector, x , and n models the round off noise due

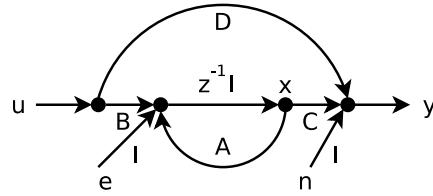


Figure 5.1: Signal flow graph of round-off noise of the state variable

to calculation of the output, y . The state variable difference equations for the flow graph are:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + e(k) \\ y(k) &= Cx(k) + Du(k) + n(k) \end{aligned}$$

The contribution of e to the state vector, x , is usually ignored when the filter realization is scaled. The round off noise at the output can be modeled as shown in Figure 5.2. Where:

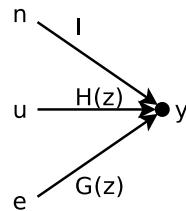


Figure 5.2: Signal flow graph of round-off noise at the filter output

$$\begin{aligned} H(z) &= D + C(zI - A)^{-1}B \\ G(z) &= C(zI - A)^{-1} \end{aligned}$$

$$g(k) = \begin{cases} 0 & k \leq 0 \\ CA^{k-1} & k > 0 \end{cases}$$

$G(z)$ is a vector of transfer functions from each state to the output. $g(k)$ is a vector of unit pulse responses, one for each state. If we call σ_i^2 the variance of that part of the output which is the response to $e(k)$, then:

$$\sigma_i^2 = \sigma_e^2 \sum_{k=1}^{\infty} g_i^2(k) = \sigma_e^2 \|g_i\|^2$$

Recall that q is the quantization step size and for a rounding-to-nearest characteristic $\sigma_e^2 = \frac{q^2}{12}$.

These estimates assume that the state variables are truncated after accumulation (ie: a double length accumulator is used).

In a similar fashion to the covariance matrix, K , of the state vector, the energy in the sequence g_i can be characterised by the matrix, W :

$$\begin{aligned} W &= E \left\{ g_i(k) g_i(k)^T \right\} \\ &= \sum_{k=0}^{\infty} (CA^k)^T (CA^k) \\ &= A^T W A + C^T C \end{aligned}$$

This is also a discrete *Lyapunov* equation and can be solved in the same manner as the equation for the state covariance matrix, K . The matrixes K and W are often called the *controllability Gramian* and *observability Gramian*, respectively. The Octave function *KW.m* returns both K and W given the state variable description $\{A, B, C, D\}$. The user can select the algorithm used to solve the corresponding discrete *Lyapunov* equation. The default algorithm uses the Octave function *dlyap* if it is found and the *Levinson* recursion otherwise.

The output noise variance due to round-off noise in calculating the i -th state can be written:

$$\sigma_i^2 = \sigma_e^2 W_{ii} \quad i = 1, \dots, n$$

After the filter is scaled, the total output error due to round off noise in the calculation of the state vector is:

$$\sigma_{total}^2 = \sigma_e^2 \delta^2 \sum_{i=1}^n W_{ii} K_{ii}$$

W_{ii} and K_{ii} are those for the unscaled filter. The sum $\sum_{i=1}^n W_{ii} K_{ii}$ is known as the “noise gain” of the filter realization. It is invariant under a diagonal (or scaling) coordinate transformation. Under a general coordinate transform, T :

$$\{A, B, C, D, K, W\} \leftarrow \{T^{-1}AT, T^{-1}B, CT, T^{-1}KT^{-T}, T^TWT\}$$

The other sources of output noise variance are those due to the input quantization and the round off in calculating the output. Neither term depends on the filter realization. Output roundoff contributes the amount:

$$\sigma_{other}^2 = \sigma_e^2 \sum_{j=1}^m \|g_j\|_2^2$$

to the output noise. m is the number of non-state variable summation nodes, g_j is the unit pulse response from node j to the output and the quantisation step size at the output and at each node is assumed to be the same ie: the word lengths used are the same for each state.

The following result can be used to calculate the l_2 -norm of a transfer function, $H(z)$, with unit pulse response $h(k)$ and state variable representation $\{A, B, C, D\}$:

$$\|h\|_2^2 = D^2 + CKC^T \tag{5.1}$$

5.4.1 Rounding-to-minus-infinity quantisation noise

As stated in Section 5.1, the results for round-off noise shown above assume rounding-to-nearest rounding. With rounding-to-nearest rounding, the quantisation error is assumed to be uniformly distributed in the range $(-\frac{q}{2}, \frac{q}{2})$ with variance $\frac{q^2}{12}$, where q

is the quantisation step size. For rounding-to-minus-infinity, the quantisation error is assumed to be uniformly distributed in the range $(0, q)$ with variance $\frac{q^2}{3}$. Consequently, rounding-to-minus-infinity rounding of the state variables is equivalent to adding a step of $\frac{q}{2}$ at the output of each state. The Octave script *gkstep_test.m* demonstrates the effect of using rounding-to-minus-infinity rounding in a 3rd order Butterworth filter with cut-off frequency $0.05f_S$. The Butterworth filter is globally optimised so that the states are equally scaled. The quantisation noise at the filter output is due to the filtered state variable quantisation noise in addition to the quantisation noise of the output calculation. The latter has a mean value of $\frac{q}{2}$. The state-to-filter-output unit impulse response has been given above. The state-to-filter-output step response is the sum over time of the impulse response. The expected step response at the filter output can be estimated as the sum of the state-to-filter-output step responses of each of the state variables. Figure 5.3 shows the simulated filtered state variable quantisation noise at the output of the filter superimposed on the summed state-output-to-filter-output step response to a step of $\frac{q}{2}$ at each state variable.

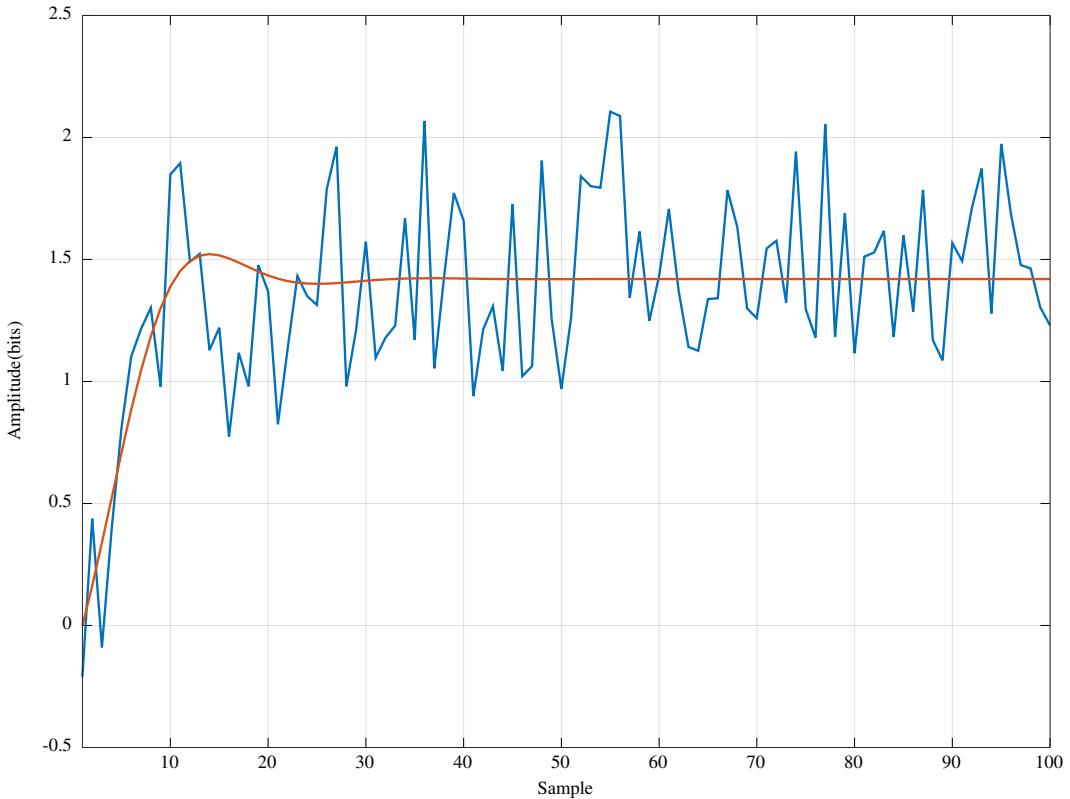


Figure 5.3: Summed state-to-output step response superimposed on the output quantisation noise of a 3rd order Butterworth filter due to rounding-to-minus-infinity at each state.

The rounding-toward-minus-infinity rounding error is not signal dependent. On the other hand, with the 2's-complement rounding-toward-zero rounding, the offset follows the sign of the input signal. The additional signal dependent quantisation noise introduced by rounding-toward-zero can significantly degrade filter noise performance.

5.5 Minimization of round-off noise in the calculation of the state vector

Minimising the noise gain minimises the round off noise. *Mullis and Roberts* [17] prove Algorithm 9 for equal word-length filters.

Algorithm 10 finds a coordinate transformation, T , that optimises the round-off noise of fixed point, equal wordlength state-variable filters given the K and W matrices. Algorithm 10 is implemented in the Octave function *optKW.m*.

The globally optimised state variable filter has $\mathcal{O}(n^2)$ coefficients. An alternative structure such as a lattice filter or a cascade of lower-order filters has fewer coefficients, $\mathcal{O}(n)$, but will have a larger than optimal noise gain. The noise gain for a non-optimal filter may be improved by distributing bits unevenly between the states. *Mullis and Roberts* [17, Section IV] show that if the state wordlengths are B_i with

$$\sum_{i=1}^n B_i = nB$$

Algorithm 9 Minimisation of the noise gain

If K and W are two n-by-n real symmetric positive definite matrices then:

$$\left[\frac{1}{n} \sum_{i=1}^n K_{ii} W_{ii} \right]^{\frac{1}{2}} \geq \frac{1}{n} \sum_{i=1}^n \mu_i$$

where μ_i^2 are the eigenvalues of the product KW . Equality holds if and only if for some diagonal matrix D :

$$K = DWD$$

and $K_{ii}W_{ii} = K_{jj}W_{jj}$ for all i and j .

Algorithm 10 Optimisation of the noise gain

1. Diagonalise K and W :

Since K and W are real, symmetric and positive definite, a decomposition into $W = U_1 D_1 U_1^T$ exists. Here U_1 is real and unitary (ie: $U_1^\dagger U_1 = I$, where \dagger means transpose conjugate) and D_1 is diagonal with real, positive elements. Recall that for non-singular matrices $(AB)^{-1} = B^{-1}A^{-1}$ and $(AB)^T = B^T A^T$. Let $T_1 = U_1 D_1^{-\frac{1}{2}}$ then

$$\begin{aligned} W_1 &= T_1^T W T_1 = I \\ K_1 &= T_1^{-1} K (T_1^{-1})^T = U_2 D_2 U_2^T \end{aligned}$$

where U_2 is real and unitary and D_2 is diagonal with real, positive elements. Let $T_2 = T_1 U_2 D_2^{\frac{1}{4}}$ then

$$\begin{aligned} W_2 &= T_2^T W T_2 = D_2^{\frac{1}{2}} \\ K_2 &= T_2^{-1} K (T_2^{-1})^T = D_2^{\frac{1}{2}} \end{aligned}$$

2. Balance $D_2^{\frac{1}{2}}$:

Now find a unitary transformation, U_3 , for which the diagonal elements of $U_3^T D_2^{\frac{1}{2}} U_3$ are nearly equal. U_3 can be found as a sequence of rotations that replace the largest and smallest diagonal elements of $D_2^{\frac{1}{2}}$ with their average. In two dimensions:

$$\begin{bmatrix} x' & y' \\ y' & z' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x & y \\ y & z \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Expand the matrix product and set $x' = z'$ so that

$$\tan 2\theta = \frac{z - x}{2y}$$

After eliminating y from x' and z'

$$x' = z' = \frac{x + z}{2}$$

3. The optimising transform:

$$T = U_1 D_1^{-\frac{1}{2}} U_2 D_2^{\frac{1}{4}} U_3$$

and the quantisation step size for each state is

$$q = 2^{-B_i+1}$$

then the choice of B_i is optimised by setting

$$\frac{K_{ii}W_{ii}}{2^{2B_i}} = c$$

where c is a constant. Each B_i is given by

$$B_i = B + \frac{1}{2} \log_2 K_{ii} W_{ii} - \frac{1}{2n} \sum_{j=1}^n \log_2 K_{jj} W_{jj} \quad (5.2)$$

The optimal output noise is then:

$$\sigma_{total}^2 = \left[\frac{n}{3} \left(\frac{\delta}{2^B} \right)^2 \right] \left[\prod_{i=1}^n W_{ii} K_{ii} \right]^{\frac{1}{n}}$$

5.6 Coefficient sensitivity

An additional effect of the use of finite length registers is the quantization of the filter parameters. This appears as a deterministic change in the filter transfer function. In fact, the sensitivities of the transfer function to the state variable coefficients are bounded reasonably tightly by the noise-gain. This means that low round off noise and low coefficient sensitivity generally occur together in digital filters. In general, finite register effects become more severe as the poles of the filter cluster together. The ratio of cut-off frequency, f_c , to sample rate, f_S , is a useful measure of this clustering. For small values the finite register effects determine the realization chosen.

5.7 Factored state variable filters and wide sense stationary inputs

The estimate of round-off noise shown in Section 5.4 only applies to the state variables and does not include the round-off noise due to arithmetic operations at other nodes in the filter. The *factored* state variable description can be used to find the variance of any variable in the realisation. Let

$$q_0(k) = \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}$$

then the covariance matrix for $q_0(k)$ is

$$\begin{aligned} E\{q_0(k) q_0^T(k)\} &= \begin{bmatrix} E\{x(k)x^T(k)\} & E\{x(k)u(k)\} \\ E\{u(k)x^T(k)\} & E\{u^2(k)\} \end{bmatrix} \\ &= \begin{bmatrix} K & 0 \\ 0 & 1 \end{bmatrix} \\ &\triangleq \tilde{K}_0 \end{aligned}$$

The factored state variable equations lead to

$$q_{l+1}^{(k)} = F_{l+1} q_l^{(k)}, \quad 0 \leq l \leq L-1$$

leading to

$$\begin{aligned} \tilde{K}_{l+1} &= E\left\{ q_{l+1}^{(k)} (k+1) \left(q_{l+1}^{(k)} (k+1) \right)^T \right\} \\ &= F_{l+1} \tilde{K}_l F_{l+1}^T \end{aligned}$$

Since the corresponding state variable description is

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = F_L F_{L-1} \dots F_1$$

we have

$$\begin{aligned}
\tilde{K}_L &= E \{ q_L(k) q_L^T(k) \} \\
&= E \left\{ \begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix} \begin{bmatrix} x^T(k+1) & y(k) \end{bmatrix} \right\} \\
&= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tilde{K}_0 \begin{bmatrix} A^T & B^T \\ C^T & D^T \end{bmatrix} \\
&= \begin{bmatrix} AKA^T + BBT & AKC^T + BD^T \\ CKA^T + DB^T & CKC^T + D^2 \end{bmatrix} \\
&= \begin{bmatrix} K & AKC^T + BD^T \\ CKA^T + DB^T & r_{yy}(0) \end{bmatrix}
\end{aligned}$$

5.8 Frequency transformations and round-off noise

Section 3.3 describes frequency transformations of state-variable filters. If the prototype filter, $H(z)$ of order n , is defined by the state-variable filter $\{A, B, C, D\}$ and the all-pass frequency transformation $1/F(z)$ of order m , is defined by $\{\alpha, \beta, \gamma, \delta\}$, then the transformed filter, $G(z) = H(F(z))$, defined by $\{\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \mathfrak{D}\}$ is:

$$\begin{aligned}
\mathfrak{A} &= I \otimes \alpha + A(I - \delta A)^{-1} \otimes \beta \gamma \\
\mathfrak{B} &= (I - \delta A)^{-1} B \otimes \beta \\
\mathfrak{C} &= C(I - \delta A)^{-1} \otimes \gamma \\
\mathfrak{D} &= D + \delta C(I - \delta A)^{-1} B
\end{aligned}$$

Mullis and Roberts [16, Section III], consider the state and output covariance matrixes, (also called the controllability and observability Gramians) \mathfrak{K} and \mathfrak{W} of the transformed filter:

$$\begin{aligned}
\mathfrak{K} &= \mathfrak{A} \mathfrak{K} \mathfrak{A}^T + \mathfrak{B} \mathfrak{B}^T \\
\mathfrak{W} &= \mathfrak{A} \mathfrak{W} \mathfrak{A}^T + \mathfrak{C}^T \mathfrak{C}
\end{aligned}$$

They prove that if $1/F(z)$ is a stable m th degree all-pass filter, then there is a positive-definite $m \times m$ symmetric matrix Q for which:

$$\begin{aligned}
Q &= \alpha Q \alpha^T + \beta \beta^T \\
Q^{-1} &= \alpha Q^{-1} \alpha^T + \gamma^T \gamma
\end{aligned}$$

Consequently, if

$$\begin{aligned}
K &= AKA^T + BB^T \\
W &= AWA^T + C^T C
\end{aligned}$$

then

$$\begin{aligned}
\mathfrak{K} &= K \otimes Q & (5.3) \\
\mathfrak{W} &= W \otimes Q^{-1} & (5.4)
\end{aligned}$$

and the nm second-order modes of the filter $G(z) = H(F(z))$ are m copies of the n second-order modes of $H(z)$.

Chapter 6

State variable filter realisation as a cascade of second order sections

6.1 Second Order State Variable Filters Optimised for Overflow and Round-Off Noise

Section 5.5 shows how to calculate the minimum noise state variable filter for an N -th order rational transfer function filter. The minimum noise filter has $\mathcal{O}(N^2)$ coefficients. This chapter describes the implementation of a rational filter transfer function as a cascade of optimised second order state variable sections having a total of $\mathcal{O}(N)$ coefficients¹. The cascade of second order sections can be pipelined for hardware implementation by inserting a delay between each section. Experience has shown that the filter realisation as a cascade of second order sections can successfully implement higher order filters than are possible with a single high order filter. The cascade of second order sections can be designed to have a round-off noise gain approaching the minimum possible. The round-off noise variance in the output of a cascade of second order sections includes:

- input quantisation noise filtered by the cascade transfer function
- quantisation noise at the output $y_j(k)$ of each sub-filter filtered by the remaining sub-filters
- quantisation noise at the output of the last sub-filter, in which case $\|g_m\|_2^2 = 1$.

6.2 Design equations for optimised second order state variable filters

Roberts and Mullis [76, Figure 9.14.1 with corrections] give the construction of the transformation matrix required to optimise a second order state variable filter, shown as Algorithm 11.

Bomar [14] gives design equations, shown in Algorithm 12 for a state variable second order section with scaling $\delta = 1$ and optimal noise performance. (See also *Roberts and Mullis* [76, Figure 9.12.1 with corrections]). *Bomar* assumes that transfer function of each second order section is arranged in the form:

$$H(z) = d + \frac{q_1 z^{-1} + q_2 z^{-2}}{1 + p_1 z^{-1} + p_2 z^{-2}} \quad (6.1)$$

Bomar [14] also gives design equations, shown in Algorithm 13, for a state variable second order section with scaling $\delta = 1$ and near optimal noise performance with one less multiplication (*Bomar* calls this a type III section). He shows experimentally for a 6th order low-pass Butterworth filter that the Type III section has a noise gain that is, as for the minimum-noise realisation, independent of the passband edge frequency.

The Octave function `pq2svcasc` converts the 2nd-order sections from the $d-p-q$ format of Equation 6.1 into 2nd-order *direct-form*, *Bomar-Type-III* or *minimum-noise* state-space sections.

Bomar [15] also describes realisation of second-order state-space sections that are “as computationally efficient as possible subject to preserving low-roundoff noise, low coefficient sensitivity and freedom from limit cycles”. In these realisations some matrix elements are replaced by single powers of 2.

¹ *Roberts and Mullis* [76, Table 10.2.1] show that a block processing implementation of the original SISO state variable filter may well have fewer arithmetic operations per output than a realisation by a cascade of second order sections.

Algorithm 11 Construction of optimised second order state variable filters[76, Figure 9.14.1].

Given $K = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix}$ and $W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$:

1. Transform K and W using a Cholesky transformation:

$$T_c = \begin{bmatrix} \sqrt{\frac{k_{11}k_{22}-k_{12}^2}{k_{22}}} & \frac{k_{12}}{\sqrt{k_{22}}} \\ 0 & \sqrt{k_{22}} \end{bmatrix}$$

so

$$K' = T_c^{-1} K T_c^{-T} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and

$$W' = T_c^T W T_c = \begin{bmatrix} w'_{11} & w'_{12} \\ w'_{21} & w'_{22} \end{bmatrix}$$

2. Apply a rotation transformation to W' so that $w'_{12} = w'_{21} = 0$. The eigenvalues of KW are thus the eigenvalues of W' . Let

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

$$\theta = \begin{cases} \frac{\pi}{4} & w'_{11} = w'_{22} \\ \frac{1}{2} \tan^{-1} \left(\frac{2w'_{12}}{w'_{11} - w'_{22}} \right) & w'_{11} \neq w'_{22} \end{cases}$$

Then

$$K'' = I$$

$$W'' = R(-\theta) W' (\theta)$$

$$= \begin{bmatrix} \mu_1^2 & 0 \\ 0 & \mu_2^2 \end{bmatrix}$$

3. Now apply a transformation:

$$\mu = \frac{\mu_2}{\mu_1}$$

$$T = \frac{\delta}{2} \begin{bmatrix} \sqrt{1+\mu} & -\sqrt{1+\mu} \\ \sqrt{1+\frac{1}{\mu}} & \sqrt{1+\frac{1}{\mu}} \end{bmatrix}$$

so that

$$K''' = \frac{1}{\delta^2} \begin{bmatrix} 1 & \frac{\mu_2-\mu_1}{\mu_2+\mu_1} \\ \frac{\mu_2-\mu_1}{\mu_2+\mu_1} & 1 \end{bmatrix}$$

$$W''' = \delta^2 \begin{bmatrix} \frac{(\mu_2+\mu_1)^2}{4} & \frac{\mu_2^2-\mu_1^2}{4} \\ \frac{\mu_2^2-\mu_1^2}{4} & \frac{(\mu_2+\mu_1)^2}{4} \end{bmatrix}$$

4. The optimising transform is $T_c R(\theta) T$.
-

Algorithm 12 Bomar second order optimised state variable filter sections [14, Equation 17]. (See also [76, Figure 9.12.1])

Compute: A_{11} , A_{12} , A_{21} , A_{22} , b_1 , b_2 , c_1 , c_2

$$\begin{aligned}
 v_1 &= \frac{q_2}{q_1} & \\
 v_2 &= \sqrt{v_1^2 - p_1 v_1 + p_2} & (\text{Bomar's } \mu) \\
 \\
 v_3 &= v_1 - v_2 & (\text{Bomar's } \gamma) \\
 v_4 &= v_1 + v_2 & (\text{Bomar's } \xi) \\
 v_5 &= p_2 - 1 \\
 v_6 &= p_2 + 1 \\
 v_7 &= v_5 (v_6^2 - p_1^2) & (\text{Bomar's } \lambda) \\
 \\
 v_8 &= \left(\frac{p_1}{2}\right)^2 - p_2 & (\text{Bomar's } \epsilon) \\
 A_{11} = A_{22} &= -\frac{p_1}{2} \\
 b_1 &= \sqrt{\frac{v_7}{2p_1 v_3 - v_6 (1 + v_3^2)}} \\
 b_2 &= \sqrt{\frac{v_7}{2p_1 v_4 - v_6 (1 + v_4^2)}} \\
 A_{21} &= \sqrt{v_8 \frac{v_5 + b_2^2}{v_5 + b_1^2}} \\
 A_{12} &= \frac{v_8}{A_{21}} \\
 c_1 &= \frac{q_1}{2b_1} \\
 c_2 &= \frac{q_1}{2b_2}
 \end{aligned}$$

Algorithm 13 Bomar Type III second order optimised state variable filter sections [14, Equation 23]

Compute: A_{11} , A_{12} , A_{21} , A_{22} , b_1 , b_2 , c_1 , c_2

$$\begin{aligned}
 A_{11} = A_{22} &= -\frac{p_1}{2} \\
 A_{12} &= \sqrt{1 + \left(\frac{p_1}{2}\right)^2 \left(\frac{p_2 - 3}{p_2 + 1}\right)} \\
 A_{21} &= \frac{\left(\frac{p_1}{2}\right)^2 - p_2}{A_{12}} \\
 b_1 &= 0 \\
 b_2 &= \sqrt{\frac{(1 - p_2) [(1 + p_2)^2 - p_1^2]}{(1 + p_2) \left[1 + \left(\frac{p_1}{2}\right)^2\right] - p_1^2}} \\
 c_1 &= \frac{q_2 + A_{11} q_1}{A_{12} b_2} \\
 c_2 &= \frac{q_1}{b_2}
 \end{aligned}$$

6.3 Block optimal second order cascade filter realisations

A cascade of individually optimised second order sections is not block optimal. That is, with the constraint that the sectional structure is maintained, the output round off noise of the cascade will not be minimised. This is so because for a white noise input the covariance matrix of the downstream sub-filters must be calculated for a coloured rather than white noise input. A cascade realization can be block optimised by:

1. Find the state variable description $\{A, B, C, D\}$ of the cascade
2. Find the $\{K, W\}$ matrixes of the cascade. For a cascade of second order sections, the 2×2 blocks on the diagonals are the covariance and noise gain matrices of the individual sections $\{K_i, W_i\}$
3. Find the transformation, T_i , that optimises $\{K_i, W_i\}$ for each section
4. Apply these T_i to each section in the cascade

6.4 An example of a second-order state-variable cascade filter

The Octave script `svcasc2noise_example_test.m` designs a 20th order Butterworth filter with cut-off frequency $f_c = 0.1 f_S$, where f_S is the sample rate, realised as a cascade of direct-form, Bomar Type III, minimum noise or block-optimised second-order state-space sections with a state variable scaling of $\delta = 4$. The Octave function `butter2pq` calculates the coefficients of a highpass or lowpass Butterworth filter with second order sections in the form of the rational transfer function shown in Equation 6.1. The sections are ordered with increasing pole angle.² The Octave function `pq2svcasc` converts these coefficients to second-order direct-form, Bomar Type III or minimum-noise state-space sections. The `pq2blockKWopt` function block-optimises the second-order direct-form cascade realisation. If the transfer function has odd order then `pq2blockKWopt` makes the final section a direct-form first-order section with an unused state variable. Note that, to avoid numerical problems, the `svcasc2Abcd` function will quietly remove an obviously unused state variable so that the state variable matrixes have the expected size for the odd filter order. For an odd order filter realised as a cascade of second-order minimum-noise or Bomar Type III sections `svcasc2Abcd` may not be able to remove the unused state variable. This may cause numerical problems when calculating the K and W covariance matrixes of the complete second-order cascade. In practice, the block-optimised second-order cascade, as generated by `pq2blockKWopt`, is the preferred realisation.

The example script uses the Octave function `svcasc2noise` to calculate the section noise-gain for each realisation generated by `pq2svcasc` and for the block-optimised realisation generated by `pq2blockKWopt`. `svcasc2noise` also calculates an estimate of the contribution of the output roundoff noise for that section at the overall cascade output. Finally, `svcasc2noise` estimates the optimal state variable bit distribution according to Equation 5.2.

The example script compares the overall noise gains for each cascade realisation with the section pole angles in increasing and decreasing order. That is, in the latter case the sections are in the reverse order to that calculated by `butter2pq`.

For comparison, the example script finds the overall state-space matrix with `svcasc2Abcd` and calculates the noise gain of the globally optimised filter. Recall that in the worst case, the globally optimised N th order filter requires $(N + 1)^2$ multiplies and the second-order cascade requires $4.5N$ multiplies.

Finally, the example script compares the estimated and simulated output roundoff noise variance of the block optimised second order cascade lowpass and highpass filters with the globally optimised state variable versions of those filters.

6.4.1 Comparison of calculated noise gains

Table 6.1 shows the section noise gains for each low-pass filter realisation. The coefficients used for these calculations are exact, not rounded, in order to illustrate the frequency independence of the optimised sections. As expected, the second-order minimum-noise, block-optimised and globally optimised realisations have the same noise gain in the high-pass and low-pass filters. Table 6.2 shows the section noise gains for each high-pass filter realisation.

Recall that the noise gain for each section estimates the contribution of the state variable roundoff noise from that section in the overall filter cascade output. This is *not* the same as the noise gain from the section state variables to the section output. If you calculate the latter separately for each section, then the state variable noise gain at each section output of the minimum-noise

²The Octave function `sos2pq` converts the output of the Octave-Forge `signal` package [66] `tf2sos` function to $p-q$ format

Section	Direct	Bomar III	Min. Noise	Block Opt.
1	5.5186	1.3967	1.3690	1.2794
2	10.9408	2.6388	2.4110	2.0731
3	15.0831	4.0774	3.5375	2.9953
4	13.6247	4.2012	3.5377	3.0481
5	9.0410	3.1335	2.6059	2.3064
6	4.8920	1.8552	1.5420	1.4031
7	2.3858	0.9592	0.8043	0.7520
8	1.1733	0.4837	0.4133	0.3974
9	0.6722	0.2766	0.2447	0.2420
10	0.5977	0.2652	0.2656	0.2585

Table 6.1: Section noise gains for the 20th order Butterworth lowpass filter

Section	Direct	Bomar III	Min. Noise	Block Opt.
1	2.4789	1.3466	1.3690	1.2794
2	4.7116	2.3861	2.4110	2.0731
3	10.3506	3.8681	3.5375	2.9953
4	14.0168	4.5558	3.5377	3.0481
5	12.3990	3.9417	2.6059	2.3064
6	8.0563	2.6454	1.5420	1.4031
7	4.2929	1.4971	0.8043	0.7520
8	2.0938	0.7954	0.4133	0.3974
9	1.0736	0.4607	0.2447	0.2420
10	0.8492	0.4276	0.2656	0.2585

Table 6.2: Section noise gains for the 20th order Butterworth highpass filter

filter will be found to be less than that of the corresponding section from the Bomar Type III filter. This explains the apparent discrepancies in Table 6.1 and Table 6.2. The noise-gains for each second-order minimum-noise section calculated according to Bomar's equations, as shown in Algorithm 12, agree with those calculated following the general method shown in Algorithm 11, although the state variable coefficients found by the two methods are different.

Tables 6.3 and 6.4 show the overall noise gains for each filter realisation with sections ranked in order of increasing and decreasing pole angle for the lowpass and highpass filters respectively. For the block optimised cascade the noise gain in parentheses shows that calculated if the cascade is not re-optimised after the section order is reversed.

Section design	Section pole angle increasing	Section pole angle decreasing
Direct	63.9292	63.5261
Bomar III	19.2876	15.8799
Min. Noise	16.7309	16.7309
Block Opt.	14.7554	14.7554 (21.48)
Global Opt.	1.6848	1.6848

Table 6.3: Overall noise gains for the 20th order Butterworth lowpass filter

6.4.2 Simulation results

Figure 6.1 shows the simulated response of the 20th order Butterworth lowpass filter realised as a block optimised cascade of second order sections with coefficients rounded to 10 bits and 10 bit state storage. Figure 6.2 shows the simulated response of the corresponding 20th order Butterworth highpass filter. Tables 6.5 and 6.6 show the simulated and estimated output roundoff noise variance for the 20th order Butterworth lowpass and highpass filter respectively.

The input signal is a uniformly distributed random noise signal with a nominal standard deviation of 2^8 . The state variables are scaled with $\delta = 4$ so that the nominal standard deviation of the state variables is 2^6 . In each case the section outputs, state variables and coefficients are rounded to 10 bits. The effect of coefficient truncation on the noise-gain is seen by comparison with

Section design	Section pole angle increasing	Section pole angle decreasing
Direct	60.3228	24.1559
Bomar III	21.9245	21.3406
Min. Noise	16.7309	16.7309
Block Opt.	14.7554	14.7554 (21.48)
Global Opt.	1.6848	1.6848

Table 6.4: Overall noise gains for the 20th order Butterworth highpass filter

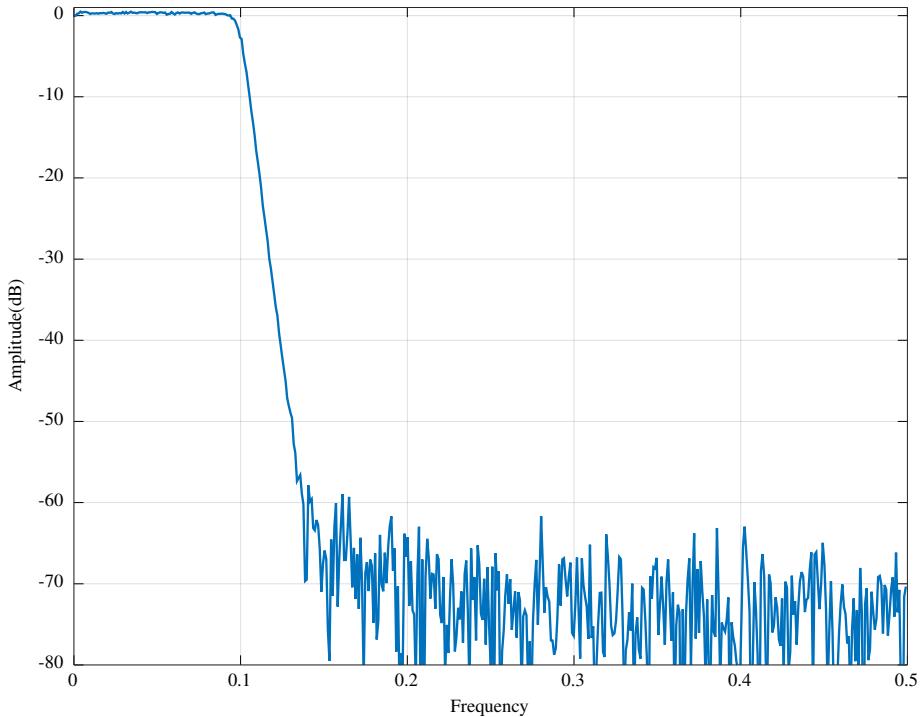


Figure 6.1: Simulated amplitude response of the 20th order lowpass Butterworth filter realised as a block optimised cascade of second-order sections with 10-bit coefficients and state storage

	Estimated noise gain	Estimated noise variance	Simulated noise variance
Scaled Direct	67.89	90.67	65.50
Block Opt.	16.34	21.94	20.35
Block Opt. (extra bits)	5.44	7.41	6.32
Global Opt.	1.64	2.27	2.34

Table 6.5: Estimated noise gain and estimated and simulated output roundoff noise variances for the 20th order Butterworth lowpass filter with 10 bit rounded coefficients.

	Estimated noise gain	Estimated noise variance	Simulated noise variance
Scaled Direct	61.11	81.97	61.33
Block Opt.	15.81	21.57	20.34
Block Opt. (extra bits)	5.26	7.50	7.25
Global Opt.	1.69	2.34	2.26

Table 6.6: Estimated noise gain and estimated and simulated output roundoff noise variances for the 20th order Butterworth highpass filter with 10 bit rounded coefficients.

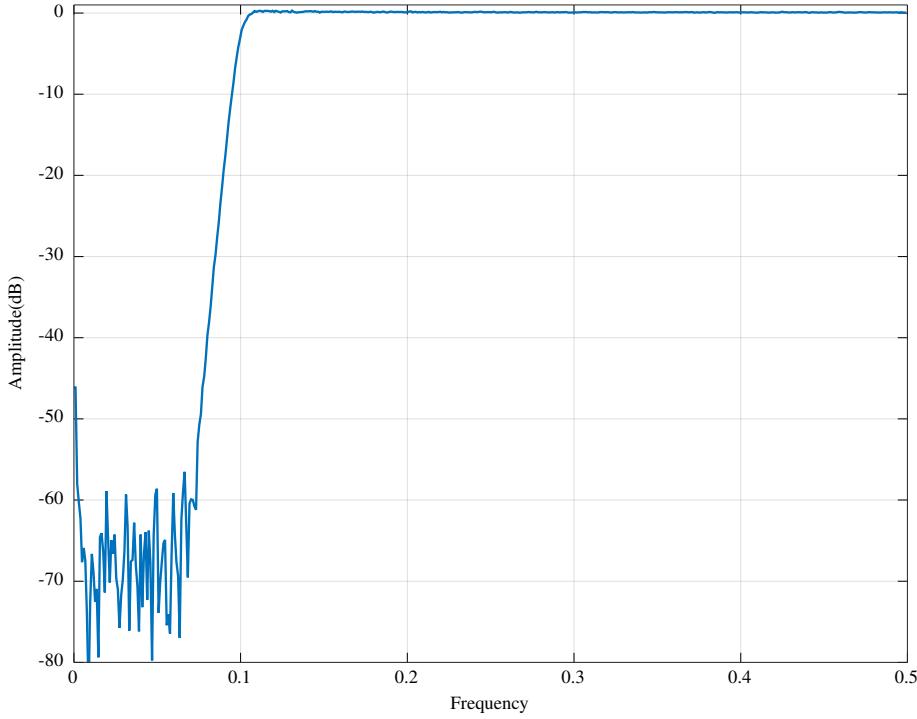


Figure 6.2: Simulated amplitude response of the 20th order highpass Butterworth filter realised as a block optimised cascade of second-order sections with 10-bit coefficients and state storage

the calculated values shown in Table 6.3 and Table 6.4. The simulation results suggest that, to avoid overflow, the intermediate section outputs should be kept in a double-length accumulator. For comparison, the tables show the simulation results for the globally-optimised filter and the improvement obtained by adding an extra bit to the state variables in the first 6 sections of the block-optimised filter. (See Equation 5.2). When calculating the output noise gain of the block optimised cascade with additional state variable bits in some sections, the noise gain for the individual section is scaled in proportion to the number of extra bits for that section. The nominal standard deviation of these state variables is 2^7 .

6.4.3 Comparison with an N=10 example

The order 20 Butterworth filter was chosen as an extreme example. Table 6.7 shows the simulation results obtained by setting $N = 10$ in *svcasc2noise_example_test.m*.

	Estimated noise gain	Estimated noise variance	Simulated noise variance
Scaled Direct	7.36	9.93	6.93
Block Opt.	1.92	2.69	2.62
Block Opt. (extra bits)	1.10	1.59	1.59
Global Opt.	1.03	1.45	1.42

Table 6.7: Estimated noise gain and estimated and simulated output roundoff noise variances for the 10th order Butterworth lowpass filter with 10 bit rounded coefficients.

Chapter 7

Filter synthesis by the Schur decomposition

This chapter follows *Parhi* [49, Chapter 12].

7.1 The Schur algorithm

From *Parhi* [49, Chapter 12]:

The Schur algorithm was originally used to test if a power series is analytic and bounded in the unit disk. If an N -th order polynomial $\Phi_N(z)$ has all zeros inside the unit circle then $N + 1$ polynomials

$$\{\Phi_i(z), i = N, N - 1, \dots, 0\}$$

can be generated by the Schur algorithm. One of the most important properties of the Schur algorithm is that these $N + 1$ polynomials form an orthonormal basis that can be used to expand any N -th order polynomial.

In this section, the inner product formulation used to demonstrate orthonormality is based on the calculation of the signal power at an internal node of a digital filter. Appendix A.6 contains a review of the complex variables theory required in this section.

7.1.1 Computation of Schur polynomials

The denominator of a stable IIR filter is a Schur polynomial because it has no zeros on or outside the unit circle. Define the N -th order denominator polynomial as:

$$D_N(z) = \sum_{i=0}^N d_i z^i$$

Initialise the N -th order Schur polynomial $\Phi_N(z)$ as:

$$\Phi_N(z) = D_N(z) = \sum_{i=0}^N \phi_i z^i$$

From $\Phi_N(z)$ form the polynomial $\Phi_{N-1}(z)$ by:

$$\begin{aligned}\Phi_{N-1}(z) &= \frac{z^{-1} \{\phi_N \Phi_N(z) - \phi_0 \Phi_N^*(z)\}}{\sqrt{\phi_N^2 - \phi_0^2}} \\ &= \frac{z^{-1} \{\Phi_N(z) - k_N \Phi_N^*(z)\}}{\sqrt{1 - k_N^2}}\end{aligned}$$

where $k_N = \phi_0/\phi_N$ and $\Phi_N^*(z)$ is the reverse polynomial of $\Phi_N(z)$ defined by:

$$\Phi_N^*(z) = z^N \Phi_N(z^{-1})$$

The degree of $\Phi_{N-1}(z)$ is 1 less than that of $\Phi_N(z)$ since, by a change of variables the numerator is:

$$\begin{aligned} z^{-1} \{ \phi_N \Phi_N(z) - \phi_0 \Phi_N^*(z) \} &= z^{-1} \sum_{i=0}^N \{ \phi_N \phi_i z^i - \phi_0 \phi_{N-i} z^i \} \\ &= \sum_{i=1}^N \{ \phi_N \phi_i - \phi_0 \phi_{N-i} \} z^{i-1} \end{aligned}$$

For $\Phi_N(z)$ to be a Schur polynomial, $|k_i| < 1$ for each polynomial in the set $\{\Phi_N(z), \Phi_{N-1}(z), \dots, \Phi_1(z)\}$. Parhi [49, Equation 12.6] points out that by inspection of $\Phi_{N-1}(z)$, the coefficients of increasing powers of z in $\Phi_{N-1}(z)$ are $\frac{1}{\sqrt{\phi_N^2 - \phi_0^2}}$ times the N determinants of the 2×2 submatrices formed by the first column and each succeeding column in the matrix:

$$\begin{bmatrix} \phi_N & \phi_{N-1} & \phi_{N-2} & \dots & \phi_2 & \phi_1 & \phi_0 \\ \phi_0 & \phi_1 & \phi_2 & \dots & \phi_{N-2} & \phi_{N-1} & \phi_N \end{bmatrix}$$

The Schur decomposition of a polynomial is implemented in the Octave function *schurdecomp*. The C++ file *schurdecomp.cc* implements *schurdecomp* as an *oct*-file using the *MPFR* arbitrary precision floating point library [51, 75] written by Fousse et al. [1]. The mantissa precision is set to 256 bits.

7.1.2 Orthonormality of Schur Polynomials

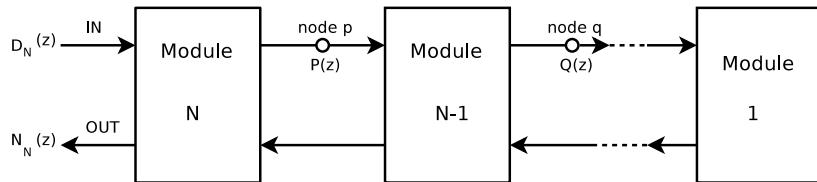


Figure 7.1: A filter structure implementing $H(z) = \frac{N_N(z)}{D_N(z)}$ (after Parhi [49, Fig. 12.1])

The filter structure shown in Figure 7.1 implements a real causal stable transfer function $H(z) = N_N(z)/D_N(z)$. The transfer function from the input node to node p is $P(z)/D_N(z)$. If the input signal is modeled as random and white with unit power, then the average power at node p is:

$$\begin{aligned} P_p &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left| \frac{P(e^{i\omega})}{D_N(e^{i\omega})} \right|^2 d\omega \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \frac{P(e^{i\omega}) P(e^{-i\omega})}{D_N(e^{i\omega}) D_N(e^{-i\omega})} d\omega \end{aligned}$$

On the unit circle, $z = e^{i\omega}$, so in the z-domain:

$$P_p = \frac{1}{2\pi i} \oint_C \frac{P(z) P(z^{-1}) z^{-1}}{D_N(z) D_N(z^{-1})} dz$$

Where the contour is taken in the anti-clockwise direction about the origin. Recall that all the zeros of $D_N(z)$ are inside the unit circle, C . Now define the inner product of two internal polynomials $P(z)$ and $Q(z)$ as:

$$\langle P(z), Q(z) \rangle = \frac{1}{2\pi i} \oint_C \frac{P(z) Q(z^{-1}) z^{-1}}{D_N(z) D_N(z^{-1})} dz$$

so that $P_p = \langle P(z), P(z) \rangle$. This is a valid inner product definition because it has the following three properties:

1. Conjugate symmetry:

$$\langle P(z), Q(z) \rangle = \langle Q(z), P(z) \rangle^\dagger$$

where (\dagger) represents the complex conjugate operation. This can be verified by a change of variables. The coefficients of $P(z)$ and $Q(z)$ are usually real.

2. Linearity: for any real constants α, β and γ :

$$\langle \alpha P(z), \beta Q(z) + \gamma R(z) \rangle = \alpha\beta \langle P(z), Q(z) \rangle + \alpha\gamma \langle P(z), R(z) \rangle$$

3. Positive norm:

$$\langle P(z), Q(z) \rangle \geq 0$$

with equality if and only if $P(z) = 0$.

Some identities:

$$\langle \Phi_N(z), z^i \rangle = \begin{cases} \frac{1}{\phi_N} & i = N \\ 0 & 0 \leq i \leq N-1 \end{cases}$$

PROOF: Recall that $\Phi_N^*(z) = z^N \Phi_N(z^{-1})$. Then

$$\begin{aligned} \langle \Phi_N(z), z^i \rangle &= \frac{1}{2\pi i} \oint_C \frac{\Phi_N(z) z^{-i}}{\Phi_N(z) \Phi_N(z^{-1})} dz \\ &= \frac{1}{2\pi i} \oint_C \frac{z^{N-i-1}}{\Phi_N^*(z)} dz \\ &= \begin{cases} \frac{1}{\phi_N} & i = N \\ 0 & otherwise \end{cases} \end{aligned}$$

The reverse Schur polynomial has all its roots outside the unit circle so the integrand is analytic within the unit circle if $0 \leq i < N$ and Cauchy's Integral Theorem applies. The result for $i = N$ follows from Cauchy's Integral Formula. Similarly:

$$\langle \Phi_N^*(z), z^i \rangle = \begin{cases} \frac{1}{\phi_N} & i = 0 \\ 0 & i \geq 1 \end{cases}$$

If

$$P(z) = \sum_{i=0}^N p_i z^i$$

then:

$$\begin{aligned} \langle \Phi_N(z), P(z) \rangle &= \frac{p_N}{\phi_N} \\ \langle \Phi_N^*(z), P(z) \rangle &= \frac{p_0}{\phi_N} \end{aligned}$$

Also:

$$\begin{aligned} \langle \Phi_N(z), \Phi_N(z) \rangle &= 1 \\ \langle \Phi_N^*(z), \Phi_N^*(z) \rangle &= 1 \\ \langle \Phi_N(z), \Phi_N^*(z) \rangle &= \frac{\phi_0}{\phi_N} \\ \langle z^{-j} \Phi_N(z), z^i \rangle &= \langle \Phi_N(z), z^{i+j} \rangle \\ \langle z^{-j} \Phi_N^*(z), z^i \rangle &= \langle \Phi_N^*(z), z^{i+j} \rangle \end{aligned}$$

The Schur polynomials satisfy the following orthonormality condition:

$$\langle \Phi_i(z), \Phi_j(z) \rangle = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

where $0 \leq i, j \leq N$. See the proof in *Parhi* [49, Appendix D]. The reverse Schur polynomials are not orthonormal. However:

$$\begin{aligned} \langle \Phi_i^*(z), z^{i-j} \Phi_j^*(z) \rangle &= \langle z^{j-i} \Phi_j^*(z^{-1}), \Phi_i^*(z^{-1}) \rangle \\ &= \langle z^j \Phi_j^*(z^{-1}), z^i \Phi_i^*(z^{-1}) \rangle \\ &= \langle \Phi_j(z), \Phi_i(z) \rangle \\ &= 0 \end{aligned}$$

where $0 \leq j \leq i \leq N$. Therefore the polynomials

$$\{\Phi_N^*(z), z\Phi_{N-1}^*(z), z^2\Phi_{N-2}^*(z), \dots, z^N\Phi_0^*(z)\}$$

also form an orthonormal basis. This basis can be used to synthesise an alternative lattice structure. *Parhi* [49, Section 12.7.2] shows that the reverse polynomial structure has inferior round off noise performance when compared with the forward polynomial structure described later in this summary.

7.1.3 Polynomial Expansion Algorithm

The Schur polynomials in the set $\{\Phi_N(z), \dots, \Phi_0(z)\}$ form an orthonormal basis. Therefore any N -th order polynomial $N_N(z)$ can be expanded as

$$N_N(z) = \sum_{i=0}^N c_i \Phi_i(z)$$

Algorithm 14 shows the Schur expansion of an arbitrary polynomial in a Schur basis.

Algorithm 14 Schur polynomial expansion (see *Parhi* [49, Section 12.2.3])

For any polynomial $N_m(z)$ of degree m , ($0 < m \leq N$):

```

 $Q(z) = N_m(z)$ 
 $c_i = 0$ , for  $m < i \leq N$ 
for  $i = m, m - 1, \dots, 0$  do
   $c_i = \frac{Q^*(0)}{\Phi_i^*(0)}$ 
   $Q(z) = Q(z) - c_i \Phi_i(z)$ 
end for
```

$Q^*(z)$ and $\Phi_i^*(z)$ are the reverse polynomials of $Q(z)$ and $\Phi_i(z)$.

For lattice filter implementations of a rational transfer function, the denominator is synthesised using the Schur algorithm and the numerator is synthesised using the polynomial expansion algorithm with the orthonormal functions obtained from the denominator. The Schur expansion of a polynomial is implemented in the Octave function *schurexpand*. The C++ *oct*-file *schurexpand.cc* implements *schurexpand* with the *MPFR* arbitrary precision floating point library [51, 75] written by *Fousse et al.* [1]. The mantissa precision is set to 256 bits.

7.1.4 Power calculation using the Schur algorithm

For the lattice filter structure in Figure 7.1, when the input signal is random and white with unit power then the average power at internal node p is

$$P_p = \langle P(z), P(z) \rangle$$

Since the denominator $D_N(z)$ is a Schur polynomial,

$$P(z) = \sum_i c_i \Phi_i(z)$$

and, since the Schur algorithm inner product is linear and orthonormal

$$P_p = \sum_i c_i^2$$

7.2 Derivation of Digital Lattice Filters

The Schur polynomials are obtained by the degree reduction procedure

$$\Phi_{i-1}(z) = \frac{z^{-1} \{\Phi_i(z) - k_i \Phi_i^*(z)\}}{s_i} \quad (7.1)$$

where s_i is a scaling factor and $k_i = \Phi_i(0) / \Phi_i^*(0)$. Note that the s_i cancel out when calculating k_i so the k_i are the same regardless of the choice of s_i . If $s_i = \sqrt{1 - k_i^2}$ then the Schur polynomials are orthonormal. In the following the i -th order Schur polynomial with this choice of s_i is denoted $\Phi_i(z)$; if $s_i = 1 - \epsilon_i k_i$ is chosen, by Λ_i , and if $s_i = 1 - k_i^2$ is chosen, by $\Psi_i(z)$.

Note that since

$$\begin{aligned} \Phi_{i-1}^*(z) &= z^{i-1} \Phi_{i-1}(z^{-1}) \\ &= \frac{\Phi_i^*(z) - k_i \Phi_i(z)}{s_i} \end{aligned}$$

by rearranging

$$\begin{aligned}\Phi_i(z) &= \frac{s_i}{1 - k_i^2} \{z\Phi_{i-1}(z) + k_i\Phi_{i-1}^*(z)\} \\ \Phi_i^*(z) &= \frac{s_i}{1 - k_i^2} \{zk_i\Phi_{i-1}(z) + \Phi_{i-1}^*(z)\}\end{aligned}$$

7.2.1 Derivation of FIR, All-Pole and All-Pass Lattice Filters

Initialise an N-th order Schur polynomial as

$$\Psi_N(z) = \sum_{i=0}^N \psi_i z^i$$

Form $\Psi_{N-1}(z)$ by degree reduction

$$\Psi_{N-1}(z) = \frac{z^{-1} \{\Psi_N(z) - k_N \Psi_N^*(z)\}}{1 - k_N^2} \quad (7.2)$$

The reverse Schur polynomial is

$$\Psi_{N-1}^*(z) = z^{N-1} \Psi_{N-1}(z^{-1}) \quad (7.3)$$

$$= \frac{\Psi_N^*(z) - k_N \Psi_N(z)}{1 - k_N^2} \quad (7.4)$$

so

$$\begin{aligned}\Psi_N^*(z) &= (1 - k_N^2) \Psi_{N-1}^*(z) + k_N \Psi_N(z) \\ &= \Psi_{N-1}^*(z) + k_N \{\Psi_N(z) - k_N \Psi_{N-1}^*(z)\}\end{aligned} \quad (7.5)$$

Substituting Equation 7.5 into Equation 7.2

$$\Psi_{N-1}(z) = z^{-1} \{\Psi_N(z) - k_N \Psi_{N-1}^*(z)\}$$

so

$$\Psi_N(z) = z\Psi_{N-1}(z) + k_N \Psi_{N-1}^*(z) \quad (7.6)$$

and, applying the definition of the reverse polynomial

$$\Psi_N^*(z) = zk_N \Psi_{N-1}(z) + \Psi_{N-1}^*(z) \quad (7.7)$$

Equations 7.6 and 7.7 represent the FIR filter shown in Figure 7.2. This structure has twice the number of multipliers of the direct form structure. However, the structure is useful for implementing adaptive filters. Equations 7.2 and 7.4 represent the

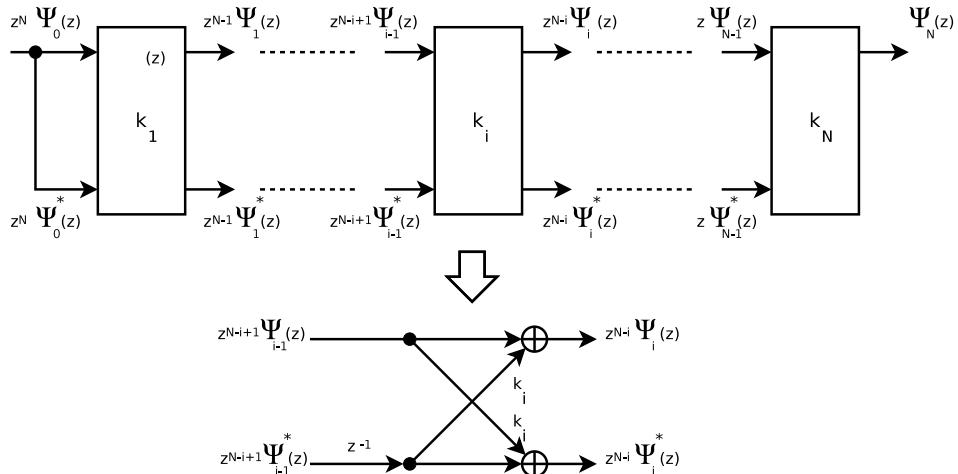


Figure 7.2: FIR filter structure (after Parhi [49, Fig. 12.8])

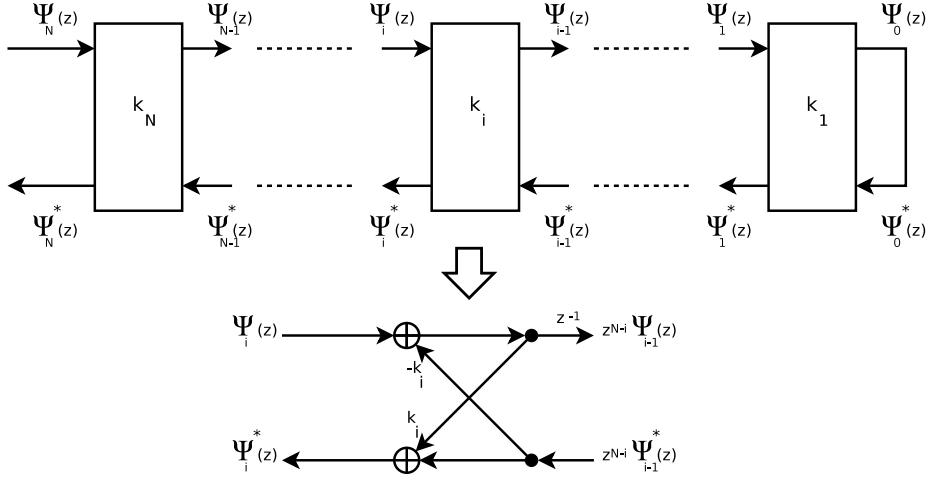


Figure 7.3: All-pass and all-pole filter structure (after Parhi [49, Fig. 12.5])

IIR filter section shown in Figure 7.3. This structure implements both an all-pole filter $\Psi_0(z)/\Psi_N(z)$ and an all-pass filter $\Psi_N^*(z)/\Psi_N(z)$. The relation between $\Phi_i(z)$ and $\Psi_i(z)$ is

$$\begin{aligned}\Psi_N(z) &= \Phi_N(z) \\ \Psi_i(z) &= \frac{\Phi_i(z)}{\sqrt{(1-k_N^2)(1-k_{N-1}^2)\cdots(1-k_{i+1}^2)}}, \quad 0 \leq i < N\end{aligned}$$

Note that $\Phi_i(z)$ and $\Psi_i(z)$ differ only by a scale factor so the k -parameters are unchanged

$$k_i = \Psi_i(0)/\Psi_i^*(0) = \Phi_i(0)/\Phi_i^*(0)$$

Also the Ψ_i are orthogonal but not orthonormal since

$$\begin{aligned}\langle \Psi_i(z), \Psi_i(z) \rangle &= \langle \Psi_i^*(z), \Psi_i^*(z) \rangle \\ &= \frac{1}{(1-k_N^2)(1-k_{N-1}^2)\cdots(1-k_{i+1}^2)}\end{aligned}$$

Clearly, the $\langle \Psi_i(z), \Psi_i(z) \rangle$ increase as i decreases since $k_i < 1$. When most of the k -parameters are nearly one, the difference of powers among the nodes in the filter is very large and the input needs to be scaled down by a large factor to prevent overflow at a critical node. As a result the effect of roundoff noise increases significantly.

7.3 Derivation of the One-Multiplier IIR Lattice Filter

If $s_i = 1 - \epsilon_i k_i$ in Equation 7.1 then

$$\Lambda_{i-1}(z) = \frac{z^{-1} \{ \Lambda_i(z) - k_i \Lambda_i^*(z) \}}{1 - \epsilon_i k_i}$$

where $\epsilon_i = \pm 1$ is a sign parameter. For an N -th order IIR transfer function, $H(z) = N_N(z)/D_N(z)$, initialise $\Lambda_N(z) = D_N(z)$. Then

$$\begin{aligned}\Lambda_{N-1}(z) &= \frac{z^{-1} \{ \Lambda_N(z) - k_N \Lambda_N^*(z) \}}{1 - \epsilon_N k_N} \\ \Lambda_{N-1}^*(z) &= \frac{\Lambda_N^*(z) - k_N \Lambda_N(z)}{1 - \epsilon_N k_N}\end{aligned}$$

where $k_i = \Lambda_i(0)/\Lambda_i^*(0)$. Rearranging

$$\Lambda_N^*(z) = k_N \Lambda_N(z) + (1 - \epsilon_N k_N) \Lambda_{N-1}^*(z) \quad (7.8)$$

$$\Lambda_{N-1}(z) = z^{-1} \{ (1 + \epsilon_N k_N) \Lambda_N - k_N \Lambda_{N-1}^*(z) \} \quad (7.9)$$

By repeated application for $i = N, N - 1, \dots, 1$, the denominator $D_N(z)$ is synthesised. The relation between $\Lambda_i(z)$ and $\Phi_i(z)$ is

$$\begin{aligned}\Lambda_N(z) &= \Phi_N(z) \\ \Lambda_i(z) &= \Phi_i(z) \sqrt{\frac{(1 + \epsilon_N k_N)(1 + \epsilon_{N-1} k_{N-1}) \cdots (1 + \epsilon_{i+1} k_{i+1})}{(1 - \epsilon_N k_N)(1 - \epsilon_{N-1} k_{N-1}) \cdots (1 - \epsilon_{i+1} k_{i+1})}}\end{aligned}$$

where $0 \leq i < N$. Note that $\Phi_i(z)$ and $\Lambda_i(z)$ differ only by a scale factor so the k -parameters are unchanged

$$k_i = \Lambda_i(0) / \Lambda_i^*(0) = \Phi_i(0) / \Phi_i^*(0)$$

Also the Λ_i are orthogonal but not orthonormal since

$$\langle \Lambda_i(z), \Lambda_i(z) \rangle = \langle \Lambda_i^*(z), \Lambda_i^*(z) \rangle \quad (7.10)$$

$$= \frac{(1 + \epsilon_N k_N)(1 + \epsilon_{N-1} k_{N-1}) \cdots (1 + \epsilon_{i+1} k_{i+1})}{(1 - \epsilon_N k_N)(1 - \epsilon_{N-1} k_{N-1}) \cdots (1 - \epsilon_{i+1} k_{i+1})} \quad (7.11)$$

The magnitude of $\langle \Lambda_i(z), \Lambda_i(z) \rangle$ can be adjusted by choosing the sign parameters so the 1-multiplier lattice filter can avoid the severe input scaling of the basic lattice filter and has better round-off noise behaviour. One criterion for choosing the sign parameters is to require that the node associated with the largest k -parameter in magnitude have the largest amplitude[4]. The sign parameters are found recursively by requiring that the amplitudes at other nodes be as large as possible without exceeding the maximum value. If the maximum occurs for k_l then the recursion proceeds for $m = l - 1, l - 2, \dots, 0$ and again for $m = l + 1, l + 2, \dots, N$. The recursion is simple because:

$$\frac{\langle \Lambda_i(z), \Lambda_i(z) \rangle}{\langle \Lambda_{i+1}(z), \Lambda_{i+1}(z) \rangle} = \frac{1 + \epsilon_{i+1} k_{i+1}}{1 - \epsilon_{i+1} k_{i+1}}$$

By changing the sign parameter, this ratio can always be made smaller or larger than one.

Algorithm 15 One-multiplier lattice sign assignment

Assume that k_l has the largest magnitude of the k_m for $m = 1, 2, \dots, N$. Define the quantities

$$\begin{aligned}Q_m &= \frac{\langle \Lambda_m(z), \Lambda_m(z) \rangle}{\langle \Lambda_l(z), \Lambda_l(z) \rangle} \\ q_m &= \frac{1 + |k_m|}{1 - |k_m|}\end{aligned}$$

so that $Q_l = 1$. Each Q_m should be as large as possible without exceeding Q_l . Successive ratios are:

$$\frac{Q_m}{Q_{m+1}} = \begin{cases} q_m & \text{if } \epsilon_m = \text{sgn}(k_m) \\ 1/q_m & \text{if } \epsilon_m = -\text{sgn}(k_m) \end{cases} \quad (7.12)$$

Now assign the ϵ_m :

```

for  $m = l - 1, l - 2, \dots, 1$  do
  if  $Q_{m+1} < 1/q_m$  then
     $\epsilon_m = -\text{sgn}(k_m)$ 
  else
     $\epsilon_m = \text{sgn}(k_m)$ 
  end if
end for
for  $m = l + 1, l + 2, \dots, N$  do
  if  $Q_m < 1/q_m$  then
     $\epsilon_m = \text{sgn}(k_m)$ 
  else
     $\epsilon_m = -\text{sgn}(k_m)$ 
  end if
end for
```

Since the polynomials $\{\Lambda_N(z), \Lambda_{N-1}(z), \dots, \Lambda_0(z)\}$ form an orthogonal basis, the numerator polynomial can be synthesised as

$$N_N(z) = \sum_{i=0}^N c_i \Lambda_i(z)$$

The synthesised filter structure is shown in Figure 7.4. The Octave function *tf2schurOneMlattice* calculates the coefficients of the one-multiplier Schur lattice from the transfer function.

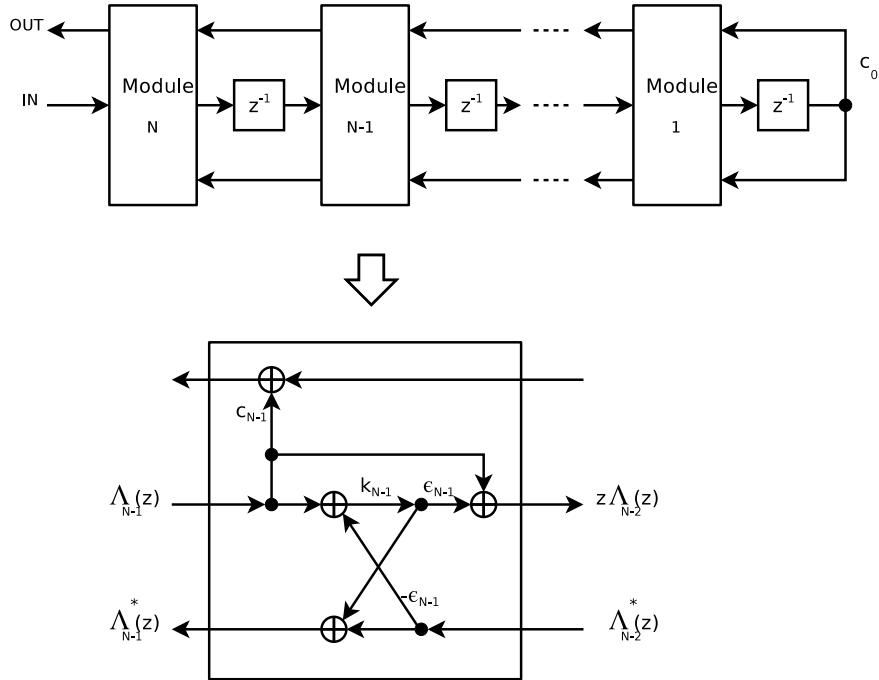


Figure 7.4: One-multiplier lattice structure (after Parhi [49, Fig. 12.11])

7.4 Derivation of the Normalised Lattice Filter

For an N-th order IIR transfer function $H_N(z) = N_N(z)/D_N(z)$ initialise the N-th order Schur polynomial as $\Phi_N(z) = D_N(z)$ and

$$\Phi_N(z) = \sum_{i=0}^N \phi_i z^i$$

Form $\Phi_{N-1}(z)$ by degree reduction

$$\Phi_{N-1}(z) = \frac{z^{-1} \{\Phi_N(z) - k_N \Phi_N^*(z)\}}{\sqrt{1 - k_N^2}}$$

where $k_i = \Phi_i(0)/\Phi_i^*(0)$. The reverse polynomial is

$$\begin{aligned} \Phi_{N-1}^*(z) &= z^{N-1} \Phi_{N-1}(z^{-1}) \\ &= \frac{\Phi_N^*(z) - k_N \Phi_N(z)}{\sqrt{1 - k_N^2}} \end{aligned}$$

So

$$\Phi_N^*(z) = \sqrt{1 - k_N^2} \Phi_{N-1}^*(z) + k_N \Phi_N(z) \quad (7.13)$$

$$\Phi_{N-1}(z) = z^{-1} \left\{ \sqrt{1 - k_N^2} \Phi_N(z) - k_N \Phi_{N-1}^*(z) \right\} \quad (7.14)$$

$$\Phi_N(z) = \frac{1}{\sqrt{1 - k_N^2}} \{ z \Phi_{N-1}(z) + k_N \Phi_{N-1}^*(z) \} \quad (7.15)$$

Figure 7.5 shows an implementation of Equations 7.13 and 7.15.

For module i

$$\begin{aligned} \sigma_{20}^{(i)} &= -\sigma_{02}^{(i)} = k_i \\ \sigma_{00}^{(i)} &= \sigma_{22}^{(i)} = \sqrt{1 - k_i^2} \end{aligned}$$

These equations synthesise the denominator $D_N(z)$ of the transfer function. The numerator is expanded in the orthonormal basis

$$N_N(z) = \sum_{i=0}^N c_i \Phi_i(z)$$

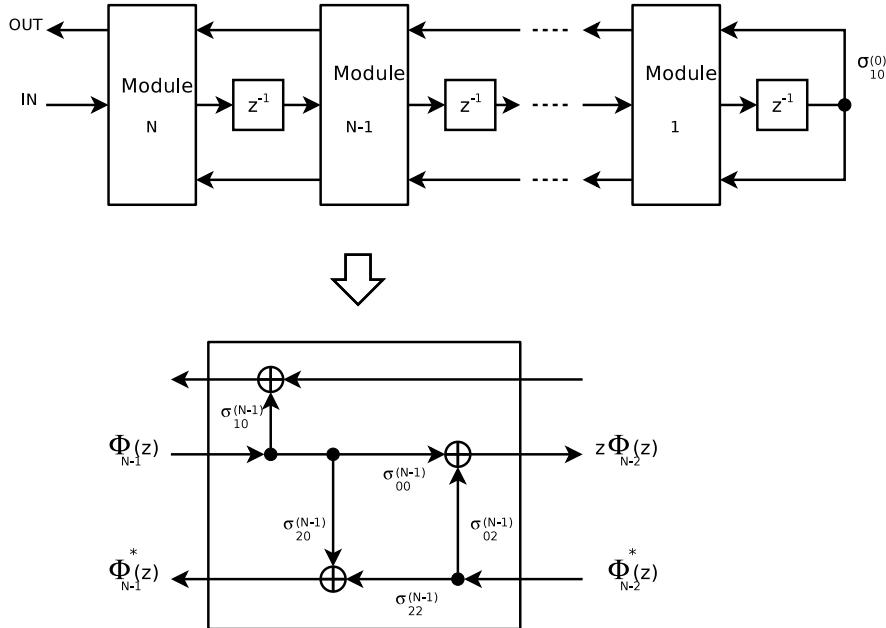


Figure 7.5: Normalised lattice filter (after Parhi [49, Fig. 12.20])

$$\sigma_{10}^{(i)} = c_i$$

This is a *normalised* lattice filter. The nodes in the feedback path have unit power since the $\Phi_i(z)$ form an orthonormal basis. For an all-pole filter the state covariance matrix, K , is the unit matrix and the structure is orthonormal. However, for a pole-zero filter the states corresponding to the numerator part are not scaled and the filter is not orthonormal.

7.5 Derivation of the Scaled Normalised Lattice Filter

We can introduce a delay at each node in the normalised lattice by making the transformation $z \rightarrow z^2$ and retiming so that each node corresponds to a state in the state variable description. Figure 7.6 shows one module of the retimed, slowed lattice. The state-variable description of the re-timed and pipelined lattice has a state for every node so that the signal at each node can be scaled¹. The orthogonality of the $\Phi_i(z)$ means that the additional diagonal elements of the state covariance matrix have the form $\sum c_i^2$. The elements of the diagonal scaling matrix have the form $T = \sqrt{\sum c_i^2}$. This suggests the following section-by-section scaling:

1. For module N:

$$\sigma_{10}^{(N)} = c_N \quad (7.16)$$

$$\sigma_{11}^{(N)} = \sqrt{\sum_{j=0}^N c_j^2} \quad (7.17)$$

2. For modules $N - 1$ to 1:

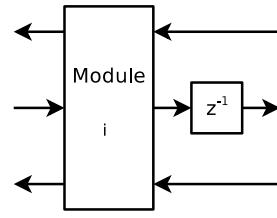
$$\sigma_{10}^{(i)} = \frac{c_i}{\sqrt{\sum_{j=0}^i c_j^2}} \quad (7.18)$$

$$\sigma_{11}^{(i)} = \frac{\sqrt{\sum_{j=0}^{i-1} c_j^2}}{\sqrt{\sum_{j=0}^i c_j^2}} \quad (7.19)$$

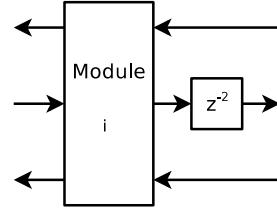
3. Any module N to 1:

$$\sigma_{20}^{(i)} = -\sigma_{02}^{(i)} = k_i \quad (7.20)$$

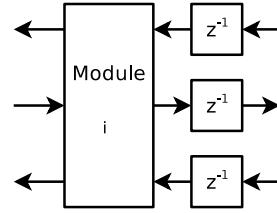
$$\sigma_{00}^{(i)} = \sigma_{22}^{(i)} = \sqrt{1 - k_i^2} \quad (7.21)$$



(a) Original lattice section



(b) After slow-down



(c) After slow-down and re-timing

Figure 7.6: Slowed and retimed lattice section

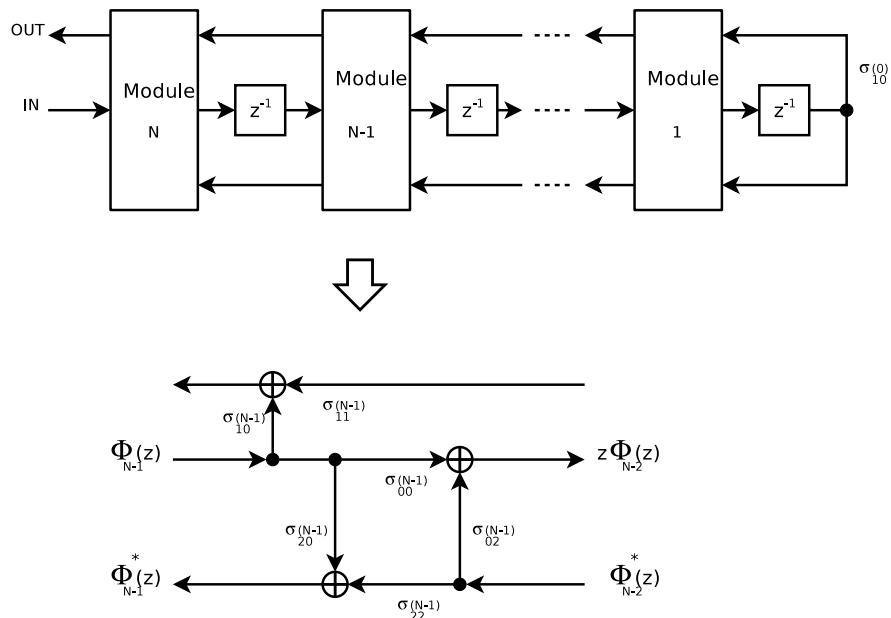


Figure 7.7: Normalised-scaled lattice filter (after Parhi [49, Fig. 12.19])

The structure of an N-th order normalised-scaled lattice filter is shown in Figure 7.7. Note that $\sigma_{10}^{(0)} = \text{sign } c_0$. The Octave function `schurNSscale` implements the scaling of the σ_{10} and σ_{11} lattice coefficients from the c_i expansion coefficients. For convenience, `schurNSscale` combines $\sigma_{10}^{(0)}$ and $\sigma_{11}^{(1)}$. The Octave function `tf2schurNSlattice` calculates the coefficients of the normalised-scaled Schur lattice from the transfer function.

7.5.1 Example: synthesis of a 3rd order Butterworth lattice filter

Parhi [49, Example 12.6.1] uses as an example a third order Butterworth low-pass filter with cutoff at angular frequency 0.1π (where the sampling frequency is normalised to 2π). The squared magnitude of the n -th order Butterworth response is defined to be

$$|\hat{H}(i\omega)|^2 = \left[1 + \left(\frac{\omega}{\omega_c} \right)^{2n} \right]^{-1}$$

The poles of the response are evenly distributed around the unit circle. For stability, we choose the poles in the left-hand s-plane

$$\lambda_k = \omega_c e^{i\theta_k}, \quad \theta_k = \frac{\pi}{2} \left(1 + \frac{(2k-1)}{n} \right) \quad 1 \leq k \leq n$$

and

$$\hat{H}(s) = \frac{-\lambda_0 \lambda_1 \lambda_1^*}{(s - \lambda_0)(s - \lambda_1)(s - \lambda_1^*)}$$

where $\lambda_0 = \Omega_c$, the cutoff frequency of the analog low pass filter, and $\lambda_1 = -\Omega_c [(1 - i\sqrt{3})/2]$ so

$$\hat{H}(s) = \frac{\Omega_c^3}{(s + \Omega_c)(s^2 + s\Omega_c + \Omega_c^2)}$$

Choose the bi-linear transformation from the s -plane to the z -plane so that the transfer function of the digital filter is $H(z) = \hat{H}\left(\frac{z-1}{z+1}\right)$. If the cutoff frequency of the digital filter $H(z)$ is $\theta_c = \omega_c t_0$ (where ω_c is the s -plane cutoff angular frequency and t_0 is the sampling interval) then pre-warp the s -plane frequency axis so that the corresponding cut-off in the s -plane is

$$\Omega(\omega_c) = \tan(\omega_c t_0 / 2)$$

(found by substituting $z = e^{i\omega t_0}$ into the bi-linear transformation). For the third order Butterworth filter

$$H(z) = \frac{\Omega^3(z+1)^3}{[(1+\Omega)z + (-1+\Omega)][(1+\Omega+\Omega^2)z^2 + (-2+2\Omega^2)z + (1-\Omega+\Omega^2)]}$$

In the example, the cut-off frequency is $0.05/t_0$ so $\omega_c t_0 = 0.1\pi$ and $\Omega = 0.158384$ giving

$$H(z) = \frac{0.0028982(z+1)^3}{z^3 - 2.37409z^2 + 1.92836z - 0.53208} = \frac{(z+1)^3}{345.04z^3 - 819.16z^2 + 665.71z - 183.59}$$

For the denominator of $H(z)$ the Schur basis is

$$\begin{aligned} \Phi_3(z) &= 345.1z^3 - 819.3z^2 + 665.8z - 183.6 \\ \Phi_2(z) &= 292.2072z^2 - 549.2662z + 271.5337 \\ \Phi_1(z) &= 107.956z - 105.1841 \\ \Phi_0(z) &= 24.3064 \end{aligned}$$

The Schur expansion of the numerator polynomial of $H(z)$ is

1. For $H(z)$ initialise $Q(z) = z^3 + 3z^2 + 3z + 1$. Then $c_3 = 1/345.1 = 0.0029$.
2. Update $Q(z) = 5.3741z^2 + 1.0707z + 1.532$. Then $c_2 = 5.3741/292.2072 = 0.0184$.
3. Update $Q(z) = 11.1725z - 3.4619$. Then $c_1 = 11.1725/107.956 = 0.10349$.
4. Update $Q(z) = 7.4238$. Then $c_0 = 7.4238/24.3064 = 0.3054$

The sum of the expansion coefficients, the output signal power, is 0.1043. The lattice “reflection coefficients” are $k_3 = -0.532$, $k_2 = 0.9293$ and $k_1 = -0.9743$.

Figure 7.8 shows the signal-flow graph of the normalised-scaled lattice implementation of $H(z)$.

¹The roundoff-noise performance of the transformed filter is the same as that of the original filter

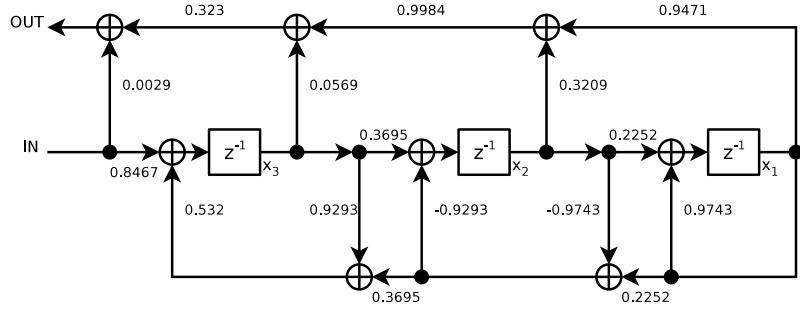


Figure 7.8: Butterworth 3rd order normalised-scaled lattice filter example (after Parhi [49, Fig. 12.20])

7.6 State Variable Descriptions for Schur Lattice Filters

7.6.1 State variable description of the Schur FIR lattice filter

Figure 7.2 shows the Schur FIR lattice structure. For convenience, call x'_n the input to state x_n of the n -th section, y_n the upper output of the n -th section and y_n^* the lower output of the n -th section. Construction of the state variable description of the Schur FIR lattice is summarised in Algorithm 16.

Algorithm 16 Construction of a state variable description of the Schur FIR lattice filter

```

 $y_0 = u$ 
 $y_0^* = u$ 
for  $n = 1, \dots, N$  do
     $x'_n = y_{n-1}^*$ 
     $y_n = k_n x_n + y_{n-1}$ 
     $y_n^* = x_n + k_n y_{n-1}$ 
end for
 $y = y_N$ 

```

As shown in Section 4.1, the state variable description can be expressed as a series of matrix multiplications linking the input and state outputs to the output and the next state inputs.

$$\begin{bmatrix} y_0 \\ y_0^* \\ x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ y_1 \\ y_1^* \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & k_1 & 0 & \cdots & 0 \\ k_1 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots & \vdots \\ 0 & \cdots & & \cdots & 1 & \end{bmatrix} \begin{bmatrix} y_0 \\ y_0^* \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ y_2 \\ y_2^* \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & k_2 & 0 & \cdots & 0 \\ 0 & k_2 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \ddots & \vdots & \vdots \\ 0 & \cdots & & \cdots & \cdots & 1 & \end{bmatrix} \begin{bmatrix} x'_1 \\ y_1 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \vdots \\ x'_{N-1} \\ y_{N-1} \\ y^*_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & \cdots & & \cdots & 0 \\ \vdots & \ddots & & & \vdots \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & 1 & 0 & k_{N-1} \\ 0 & \cdots & k_{N-1} & 0 & 1 \\ 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x'_1 \\ \vdots \\ y_{N-2} \\ y^*_{N-2} \\ x_{N-1} \\ x_N \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \vdots \\ x'_N \\ y_N \\ y^*_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & & & \vdots \\ 0 & & 0 & 1 & 0 \\ 0 & & k_N & 0 & 1 \\ 0 & \cdots & 1 & 0 & k_N \end{bmatrix} \begin{bmatrix} x'_1 \\ \vdots \\ y_{N-1} \\ y^*_{N-1} \\ x_N \end{bmatrix}$$

The Octave function `schurFIRlattice2Abcd` returns the state variable description of a Schur FIR lattice filter.

7.6.2 State variable description of the one-multiplier IIR lattice filter

In Figure 7.9, Figure 7.4 is redrawn with x'_n corresponding to the input to state x_n of the n -th section, y_n being the output of the n -th section and y^*_n the all-pass output of the n -th section. Construction of the state variable description is summarised in

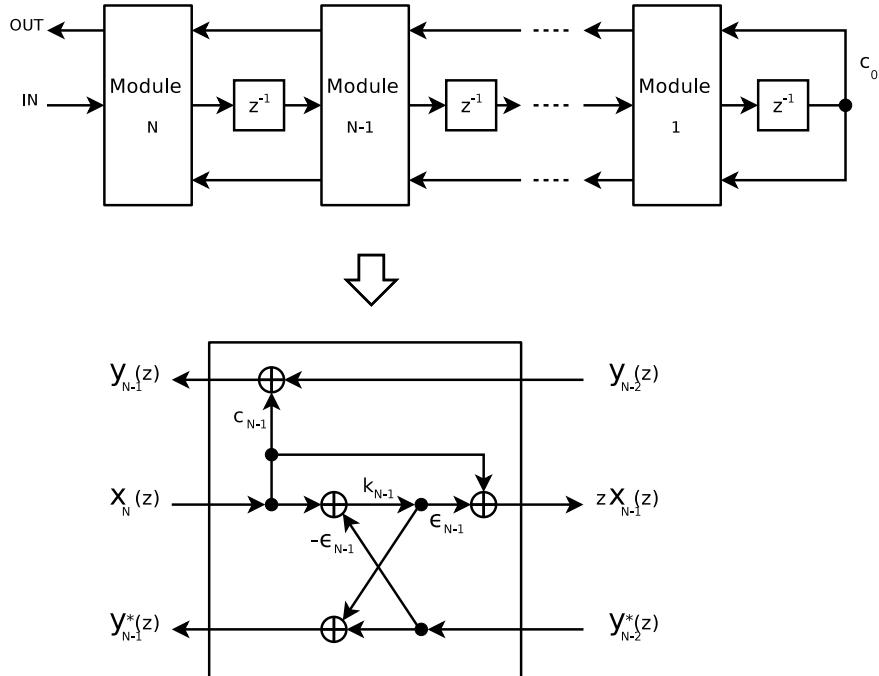


Figure 7.9: State variable description of the Schur one-multiplier lattice filter

Algorithm 17.

Algorithm 17 Construction of a state variable description of the Schur one-multiplier lattice filter

```

 $y_0^* = x_1$ 
for  $n = 1, \dots, N-1$  do
   $x'_n = -k_n y_{n-1}^* + (1 + k_n \epsilon_n) x_{n+1}$ 
   $y_n^* = (1 - k_n \epsilon_n) y_{n-1}^* + k_n x_{n+1}$ 
end for
 $x_N = -k_N y_{N-1}^* + (1 + k_N \epsilon_N) u$ 
 $y^* = (1 - k_N \epsilon_N) y_{N-1}^* + k_N u$ 
 $y = c_0 x_1 + c_1 x_2 + \dots + c_{N-1} x_N + c_N u$ 

```

The state variable description can be expressed as a series of matrix multiplications linking the input and state outputs to the

output and the next state inputs.

$$\begin{bmatrix} y_0^* \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ 0 & & & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ y_1^* \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} -k_1 & (1+k_1\epsilon_1) & 0 & \cdots & \cdots & 0 \\ (1-k_1\epsilon_1) & k_1 & 0 & & & \vdots \\ 0 & 0 & 1 & & & \vdots \\ \vdots & & & \ddots & & \vdots \\ 0 & & & & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_0^* \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ y_2^* \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & -k_2 & (1+k_2\epsilon_2) & 0 & & \vdots \\ 0 & (1-k_2\epsilon_2) & k_2 & 0 & & \vdots \\ \vdots & & & \ddots & & \vdots \\ 0 & & & & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ y_1^* \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_{N-1} \\ y_{N-1}^* \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 1 & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ 0 & & & -k_{N-1} & (1+k_{N-1}\epsilon_{N-1}) & 0 \\ 0 & & & (1-k_{N-1}\epsilon_{N-1}) & k_{N-1} & 0 \\ 0 & \cdots & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ \vdots \\ x'_{N-2} \\ y_{N-2}^* \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ \vdots \\ x'_N \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & & 1 & 0 & 0 \\ 0 & & 0 & -k_N & (1+k_N\epsilon_N) \\ 0 & \cdots & 0 & (1-k_N\epsilon_N) & k_N \end{bmatrix} \begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_{N-1} \\ y_{N-1}^* \\ u \end{bmatrix}$$

The Octave function *schurOneMlattice2Abcd* returns the state variable description of a one multiplier lattice filter (including the all-pass filter *Cap* and *Dap* output matrixes).

7.6.3 State variable description of the scaled-normalised IIR lattice filter

In Figure 7.10, Figure 7.7 is redrawn with x'_n corresponding to the input to state x_n of the n -th section, y_n being the output of the n -th section and y_n^* the all-pass output of the n -th section. Construction of the state variable description is summarised in Algorithm 18².

The state variable description can be expressed as a series of matrix multiplications linking the input and state outputs to the output and the next state inputs.

$$\begin{bmatrix} y_0^* \\ y_0 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ \sigma_{10}^{(0)} & 0 & \cdots & & \vdots \\ 0 & 1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & & & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

²The Octave function *schurNSlattice* combines $\sigma_{10}^{(0)}$ and $\sigma_{11}^{(1)}$

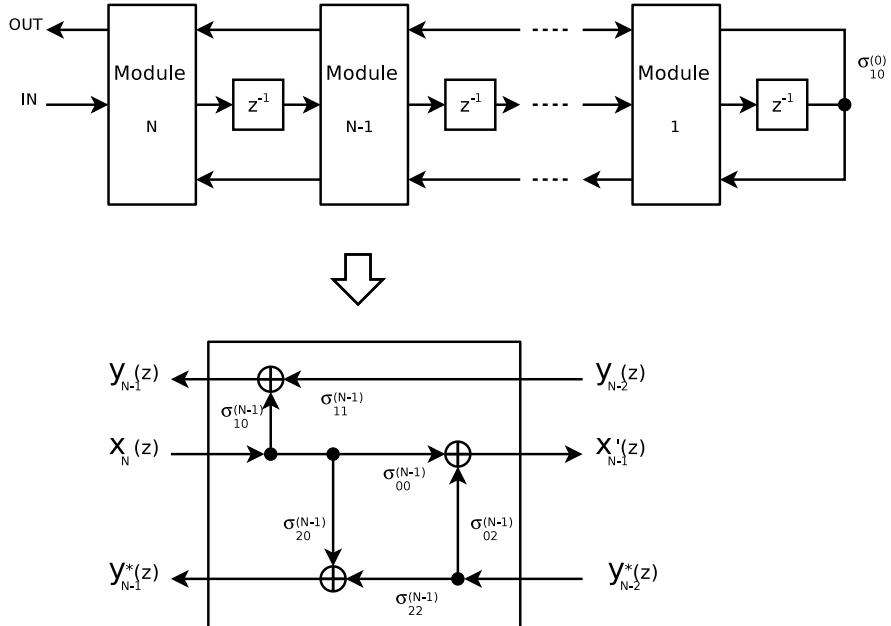


Figure 7.10: State variable description of the Schur Scaled-Normalised lattice filter

Algorithm 18 Construction of a state variable description of the Scaled-Normalised Lattice

```

 $y_0^* = x_1$ 
 $y_0 = \sigma_{10}^{(0)} x_1$ 
for  $n = 1, \dots, N - 1$  do
     $x'_n = \sigma_{02}^{(n)} y_{n-1}^* + \sigma_{00}^{(n)} x_{n+1}$ 
     $y_n^* = \sigma_{22}^{(n)} y_{n-1}^* + \sigma_{20}^{(n)} x_{n+1}$ 
     $y_n = \sigma_{11}^{(n)} y_{n-1} + \sigma_{10}^{(n)} x_{n+1}$ 
end for
 $x'_N = \sigma_{02}^{(N)} y_{N-1}^* + \sigma_{00}^{(N)} u$ 
 $y^* = \sigma_{22}^{(N)} y_{N-1}^* + \sigma_{20}^{(N)} u$ 
 $y = \sigma_{11}^{(N)} y_{N-1} + \sigma_{10}^{(N)} u$ 

```

$$\begin{bmatrix} x'_1 \\ y_1^* \\ y_1 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} \sigma_{02}^{(1)} & 0 & \sigma_{00}^{(1)} & 0 & 0 & \cdots & 0 \\ \sigma_{22}^{(1)} & 0 & \sigma_{20}^{(1)} & 0 & & & \vdots \\ 0 & \sigma_{11}^{(1)} & \sigma_{10}^{(1)} & 0 & & & \\ 0 & 0 & 0 & 1 & & & \\ & & & & \ddots & & \vdots \\ & & & & & 1 & 0 \\ 0 & \cdots & & & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_0^* \\ y_0 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ y_2^* \\ y_2 \\ \vdots \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_{02}^{(2)} & 0 & \sigma_{00}^{(2)} & & & \vdots \\ 0 & \sigma_{22}^{(2)} & 0 & \sigma_{20}^{(2)} & & & \\ 0 & 0 & \sigma_{11}^{(2)} & \sigma_{10}^{(2)} & & & \\ & & & & \ddots & & \vdots \\ 0 & 0 & & & & 0 & \\ 0 & \cdots & & & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ y_1^* \\ y_1 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_{N-1} \\ y_{N-1}^* \\ y_{N-1} \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 1 & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ 0 & & & \sigma_{02}^{(N-1)} & 0 & \sigma_{00}^{(N-1)} \\ 0 & & & \sigma_{22}^{(N-1)} & 0 & \sigma_{20}^{(N-1)} \\ 0 & & & 0 & \sigma_{11}^{(N-1)} & \sigma_{10}^{(N-1)} \\ 0 & \cdots & & & & \cdots & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ \vdots \\ y_{N-2}^* \\ y_{N-2} \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ \vdots \\ x'_N \\ y^* \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 1 & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ 0 & & & \sigma_{02}^{(N)} & 0 & \sigma_{00}^{(N)} \\ 0 & & & \sigma_{22}^{(N)} & 0 & \sigma_{20}^{(N)} \\ 0 & \cdots & & 0 & \sigma_{11}^{(N)} & \sigma_{10}^{(N)} \end{bmatrix} \begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ y_{N-1}^* \\ y_{N-1} \\ u \end{bmatrix}$$

The Octave function *schurNSlattice2Abcd* returns the state variable description of a normalised-scaled lattice filter (including the all-pass filter *Cap* and *Dap* output matrixes).

7.7 Roundoff Noise Calculation in Schur Lattice Filters

In the following I rely on the analysis of round-off noise in the state variable description presented by *Roberts and Mullis* [76], [16], [17]. Lattice filter roundoff noise can be calculated by the K and W matrices derived from the state variable description of the filter. An alternative method uses the Schur polynomials and the transposed graph of the filter. To compute the output roundoff noise, the transfer functions from the internal nodes to the output node are needed. These are the same as the transfer functions from the input to the internal nodes of the transposed filter. Again, these transfer functions can be derived from the Schur decomposition or the state variable description.

7.7.1 Round-off noise of the normalised-scaled lattice filter

Calculation of the normalised-scaled lattice filter round-off noise with the transposed signal flow graph

Figure 7.11 shows module m of $N, \dots, 1$ of the transposed graph of a normalised-scaled lattice filter (with $k = \sigma_{20} = -\sigma_{02}$, and $k_c = \sqrt{1 - k^2} = \sigma_{00} = \sigma_{22}$). The transposed graph of the module gives

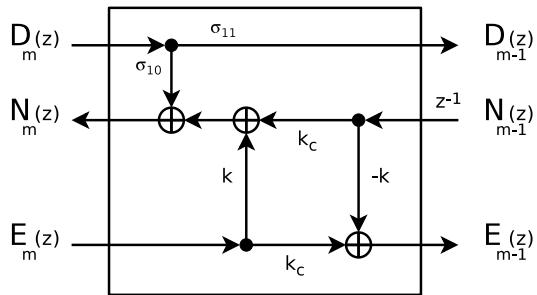


Figure 7.11: Transposed normalised-scaled lattice filter module (after Parhi [49, Fig. 12.24])

$$\begin{bmatrix} z^{-1}N_{m-1}(z) \\ D_{m-1}(z) \\ E_{m-1}(z) \end{bmatrix} = \frac{1}{k_c} \begin{bmatrix} 1 & -\sigma_{10} & -k \\ 0 & k_c\sigma_{11} & 0 \\ -k & k\sigma_{10} & 1 \end{bmatrix} \begin{bmatrix} N_m(z) \\ D_m(z) \\ E_m(z) \end{bmatrix}$$

For module N , the input, $D_N(z)$, and output, $N_N(z)$, are given by the transfer function $H(z) = N(z)/D(z)$ and $E_N(z) = 0$. Similarly, for the all-pass response, $D_N(z) = 0$, $N_N(z) = z^N D(z^{-1})$ and $E_N(z) = D(z)$. The transfer function polynomials of the modules to the right hand of module N are found by repeated matrix multiplication. Since $D(z)$ is a Schur polynomial, the output noise variance due to round off at each node is calculated from the coefficients of the Schur orthonormal basis decomposition of these transfer function polynomials.

The Octave script *butt3NS_test.m* uses the transposed transfer function to calculate the round-off noise of the normalised-scaled 3rd order low-pass Butterworth filter of the example in Section 7.5.1. The coefficients of the lattice are exact, not truncated. Coefficient truncation implies a different Schur basis and the polynomial division used to find the output noise is inaccurate. Annotated output from *butt3NS_test.m* follows.

The filter cutoff frequency is

```
fc = 0.050000
```

The denominator and numerator polynomials are

```
n = 0.0028982 0.0086946 0.0086946 0.0028982
```

```
d = 1.00000 -2.37409 1.92936 -0.53208
```

The Schur orthonormal basis corresponding to the denominator polynomial is

```
s = 0.07045 0.00000 0.00000 0.00000
-0.30483 0.31286 0.00000 0.00000
0.78677 -1.59152 0.84670 0.00000
-0.53208 1.92936 -2.37409 1.00000
```

The Schur expansion of the numerator polynomial is

```
c = 0.3053850 0.1034929 0.0183952 0.0028982
```

The coefficients of filter sections (input/output at the right end of each vector) are

```
s10 = 0.3209629 0.0569565 0.0028982
s11 = 0.94709 0.99838 0.32297
s20 = -0.97432 0.92923 -0.53208
s00 = 0.22518 0.36951 0.84670
s02 = 0.97432 -0.92923 0.53208
s22 = 0.22518 0.36951 0.84670
```

The noise gain of the filter (with un-quantised coefficients) is

$ng = 1.1906$

The nodes corresponding to $D_0(z)$ and $E_0(z)$, that is at the output of state x_1 , make no contribution to round-off noise and are omitted from the noise gain. The noise gain can be reduced slightly by summing the output in one pass (assuming a double-length accumulator holds the intermediate sums along the top edge of Figure 7.8).

The noise gain of the associated all-pass filter is

$ngap = 5.0000$

Recall that for a unit variance white noise input the internal nodes of the normalised-scaled filter have unit signal variance and that the transfer functions from the internal nodes of the all-pass filter to the all-pass output have unity gain by definition. Therefore the noise gain of the all-pass filter is simply the number of internal nodes at which arithmetic truncation occurs. In this case there are five internal nodes in the all-pass filter at which round-off occurs. Three of these are at the inputs to the internal delay element state storage and two are at calculation of the reverse Schur polynomial output, $\Phi_i^*(z)$. By convention the third of the reverse Schur polynomial output truncations is represented separately as the all-pass filter output truncation. (Truncation of the internal reverse Schur polynomial outputs could be avoided by storing them in temporary double precision storage).

For comparison the noise gain of a globally optimised Butterworth filter is $ngopt = 0.47049$ and for a direct form implementation, $ngdir = 68.980$. The corresponding noise gains for the globally optimised and direct-form all-pass filters are $ngoptap = 3.0000$ and $ngdirap = 818.90$.

The filters were tested with a uniformly distributed random noise signal with variance 0.5 of full-scale. The filter outputs were calculated with floating point arithmetic and again with rounding-to-nearest truncation and 10 bit word storage.

Figure 7.12 shows the frequency response of the Schur lattice implementation of the Butterworth filter and all-pass filter determined from the cross-correlation of the filter input and outputs.

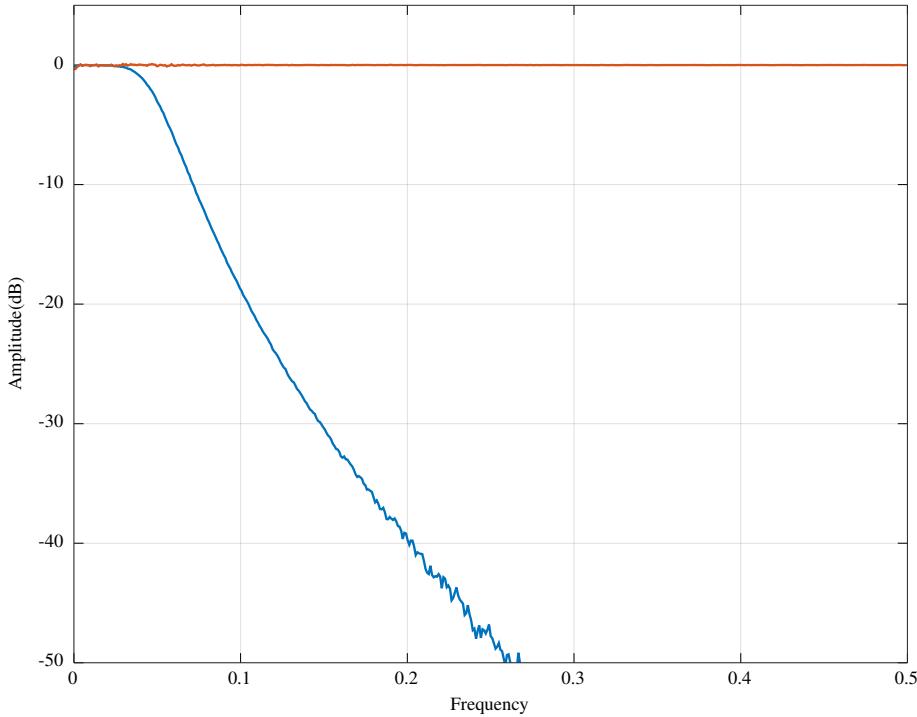


Figure 7.12: Frequency response of the 3rd order Butterworth filter implemented with a normalised-scaled lattice structure

The estimated and simulated round-off noise variances are for the Schur lattice implementation of the Butterworth filter ($\sigma^2 = (1 + ng) / 12$):

```
est_varyd = 0.18255
varyd = 0.18266
```

and for the Schur lattice all-pass filter

```
est_varyapd = 0.50000
varyapd = 0.49645
```

Similarly, the estimated and simulated round-off noise variances for the globally optimised Butterworth filter are:

```
est_varyoptd = 0.12254
varyoptd = 0.11981
```

and the estimated and simulated round-off noise variances for the scaled direct-form Butterworth filter are:

```
est_varydird = 5.8317
varydird = 1.8078
```

The factor of about 3 discrepancy between the estimated and measured output roundoff noise of the scaled direct-form filter suggests that the output roundoff noise of that filter is correlated with the input signal rather than having the uniform random distribution assumed in the noise calculations. Figure 7.13 compares the response of the output roundoff noise of the scaled direct-form implementation to that of the globally optimised filter.

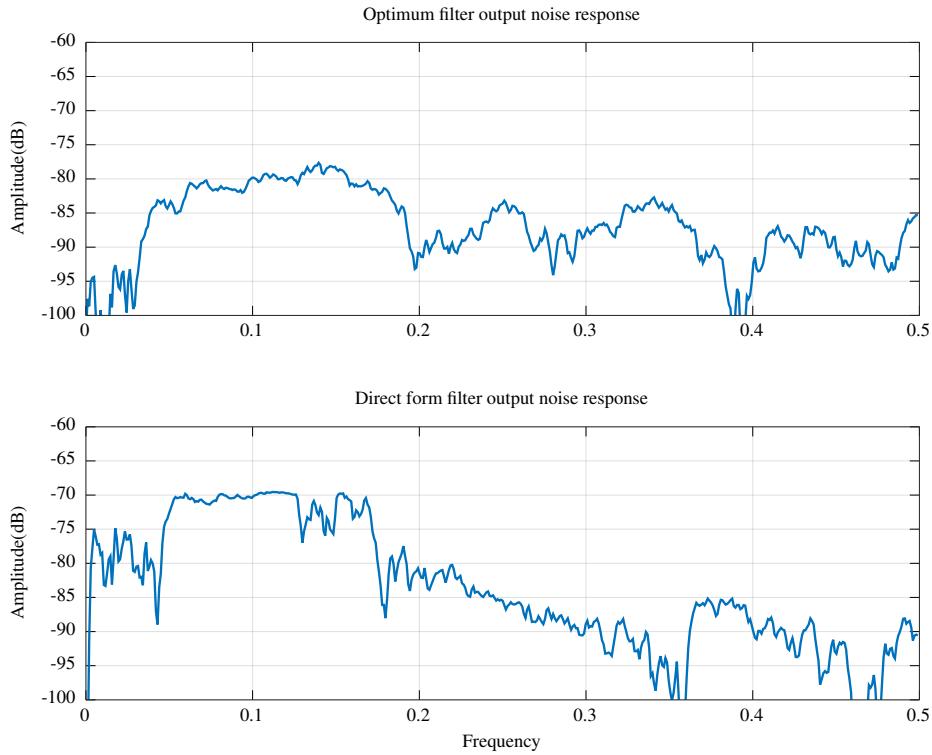


Figure 7.13: Comparison of the frequency response of the output noise of the 3rd order Butterworth filter when implemented with a scaled direct-form and globally optimised state variable structure

The standard deviations of the internal states of the Schur lattice filter are

```
stdxx = 131.32 129.47 127.97
```

The standard deviations of the internal states of the globally-optimum and scaled direct-form filters are similar. Figures 7.14, 7.15 and 7.16 show part of the state trajectories for the Schur lattice, globally-optimised and direct-form state variable versions of the Butterworth filter with a random noise input.

Figure 7.17 shows part of the state trajectories for the Schur lattice implementation of the Butterworth filter in response to a sine wave input.

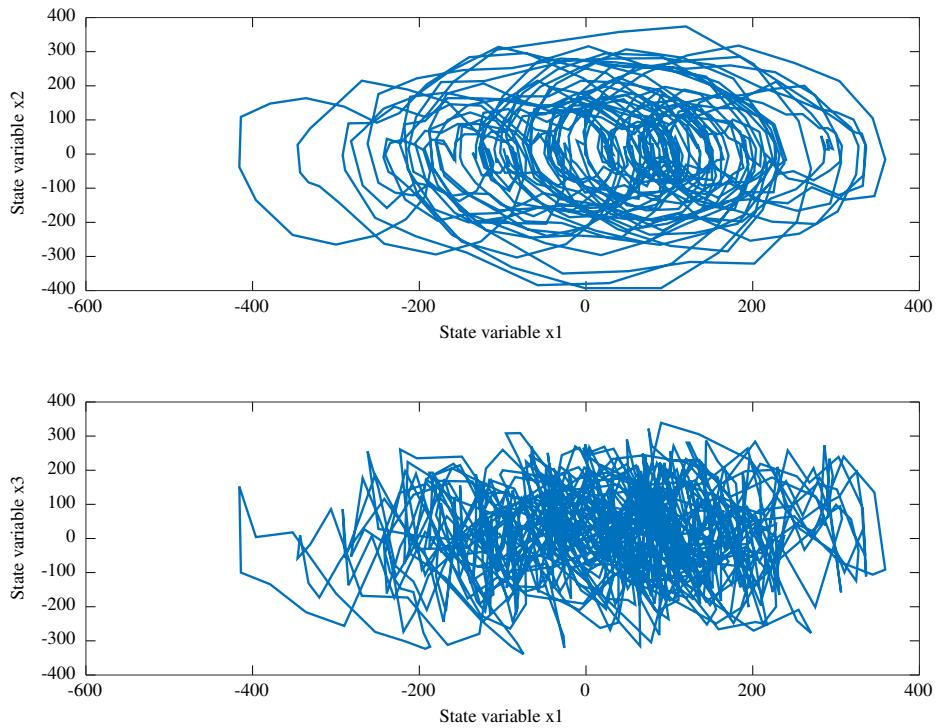


Figure 7.14: Internal states in the 3rd order Butterworth filter implemented in a normalised-scaled Schur lattice structure with a random noise input

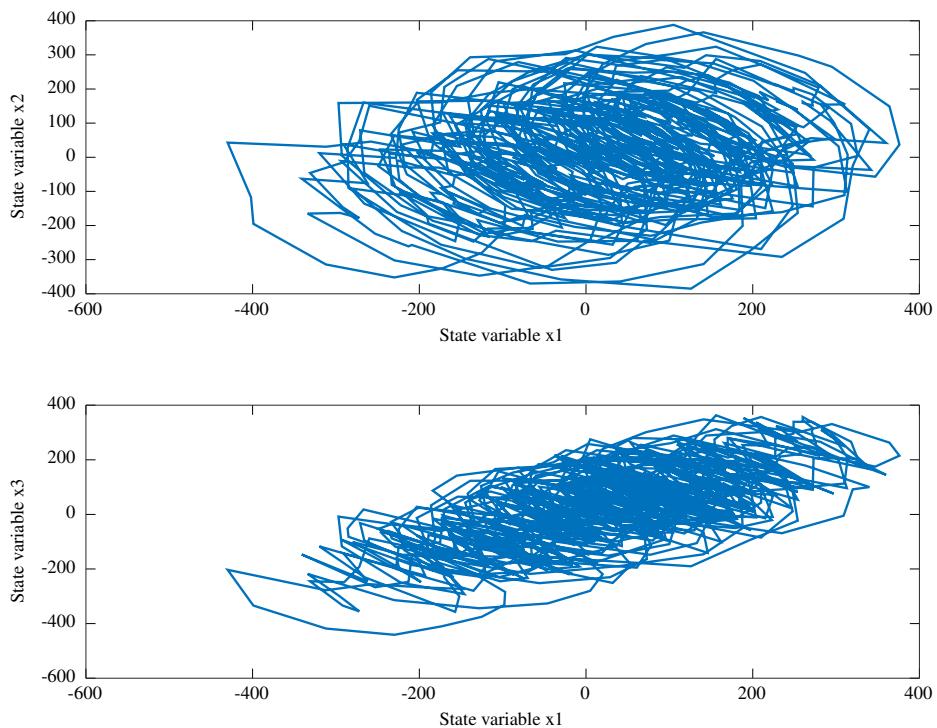


Figure 7.15: Internal states in the 3rd order Butterworth filter implemented in the globally optimised state variable structure with a random noise input

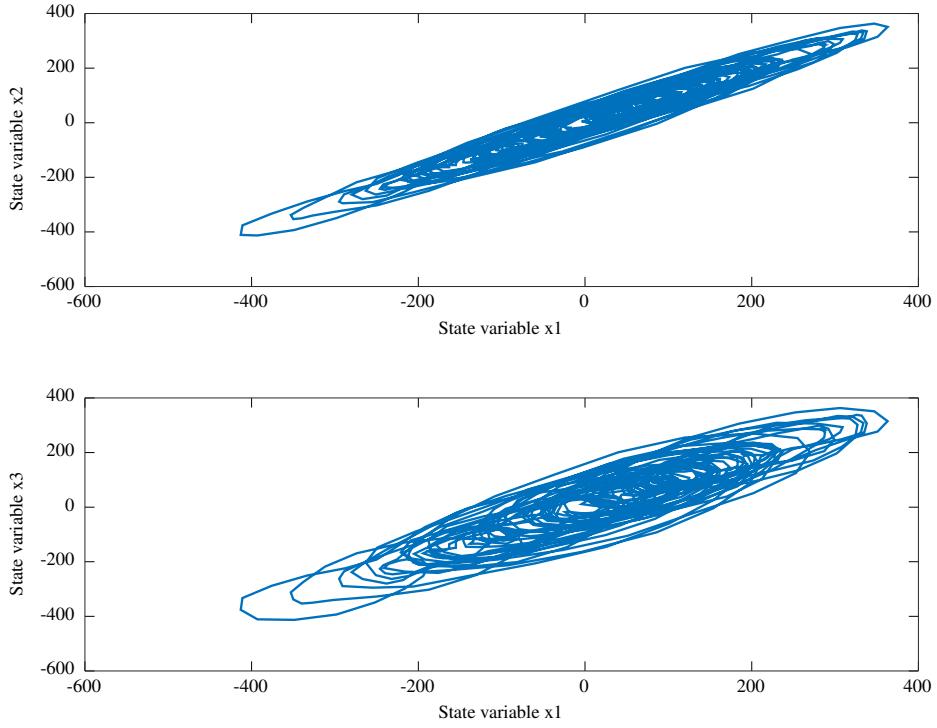


Figure 7.16: Internal states in the 3rd order Butterworth filter implemented in a direct-form structure with a random noise input

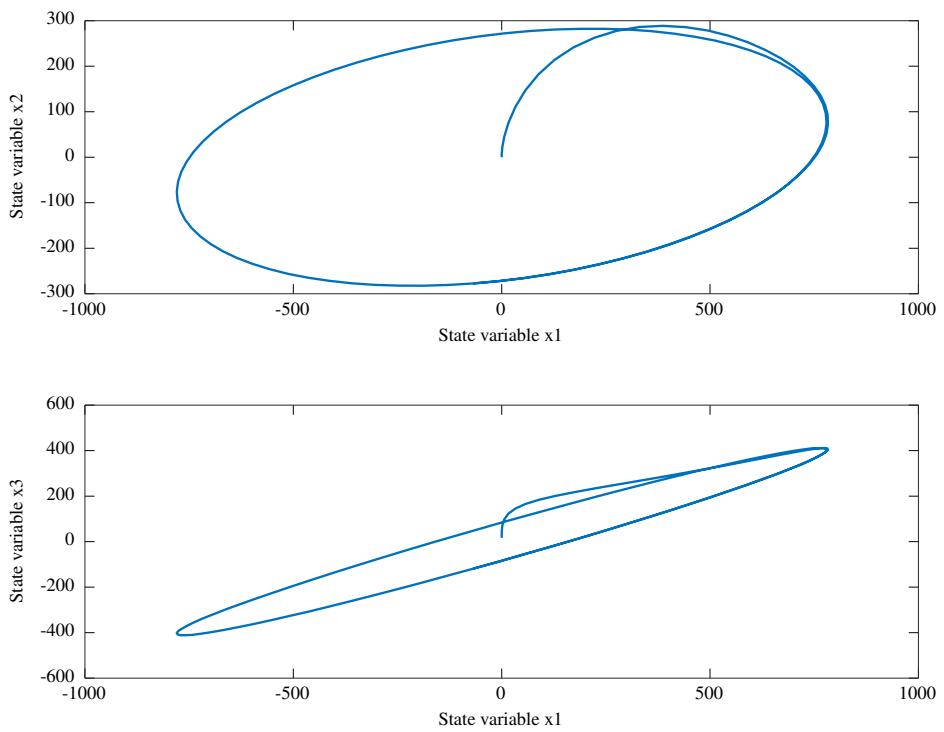


Figure 7.17: Internal states in the 3rd order Butterworth filter implemented with a Schur lattice structure with a sine wave input

Calculation of the normalised-scaled lattice filter round-off noise with a retimed state-variable description

Parhi [49, Chapter 12, p. 450] suggests that the normalised-scaled lattice filter round-off noise variance can also be determined from the state variable representation if the filter is slowed and retimed as shown in Figure 7.18. Each intermediate node is now associated with a state. The additional states alter the overall transfer function but the round-off noise gains are unchanged.

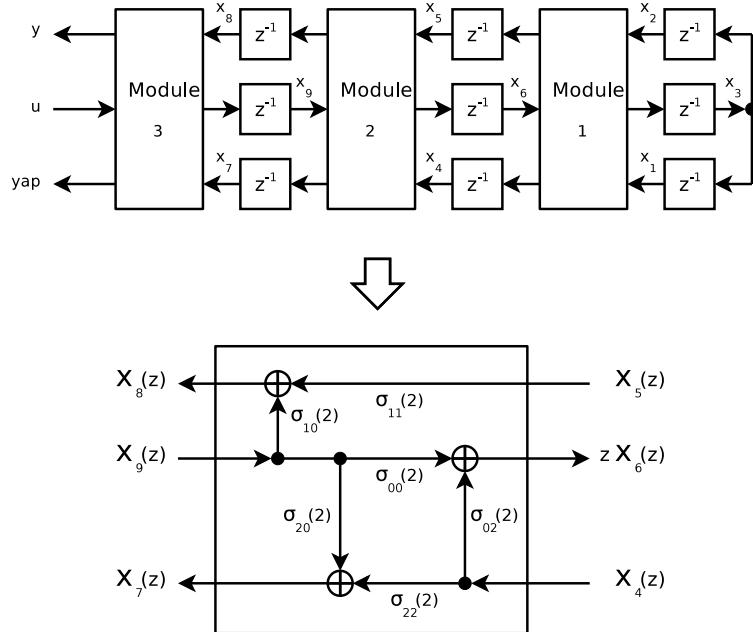


Figure 7.18: Slowed and retimed normalised-scaled lattice for the 3rd order Butterworth example

The Octave function `schurNSlatticeRetimed2Abcd` converts the Schur normalised-scaled lattice implementation to a retimed state variable form. The Octave script `butt3NSSV_test.m` demonstrates roundoff noise calculations in the retimed state variable form. The Butterworth and all-pass filter noise gains with exact coefficients are found to be the same as those for the transposed filter calculation above. With 10 bit 2 signed-digit coefficients the retimed state variable form of the Schur lattice Butterworth and all-pass filters have noise gains $ng = 1.1334$ and $ngap = 5.6989$. The equivalent floating point signed-digit coefficients are calculated by the Octave function `flt2SD` which calls `bin2SD`. The latter function approximates an integer by adding successive signed-digits³. The *oct*-file `bin2SD.cc` implements a C++ version of `bin2SD`. The filter responses for the corresponding Butterworth and all-pass filters with 10-bit 2-signed-digit coefficients are shown in Figure 7.19. The response demonstrates a drawback of the normalised-scaled lattice filters: they are not *structurally loss-less* in the sense of Vaidyanathan et al. [72, 71] (see Appendix G). The normalised-scaled lattice does not preserve the all-pass characteristic when its coefficients are truncated.

³ Parhi [49, Section 13.6.1, p.507] shows an algorithm that calculates the complete signed-digit representation. The Octave function `bin2SPT` and *oct*-file `bin2SPT.cc` implement this algorithm.

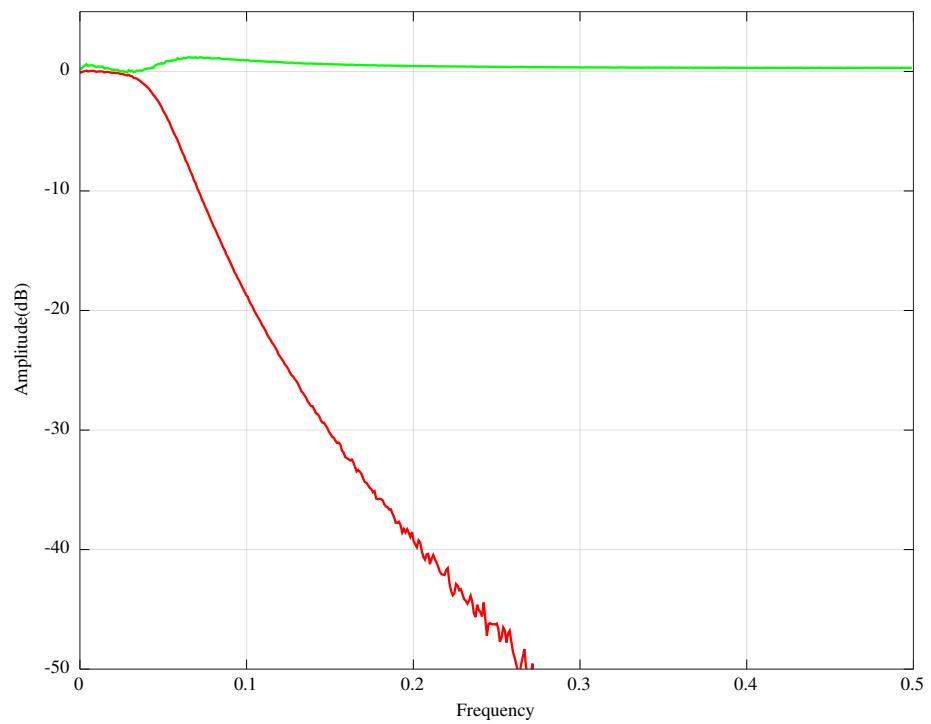


Figure 7.19: Frequency response of the 3rd order Butterworth filter implemented with a normalised-scaled lattice structure and 10-bit 2 signed-digit coefficients

7.7.2 Round-off noise of the one multiplier lattice filter

Calculation of the one multiplier lattice filter round-off noise with the transposed signal flow graph

Figure 7.20 shows module m of $N, \dots, 1$ of the transposed graph of a one multiplier lattice filter. The transposed graph of the

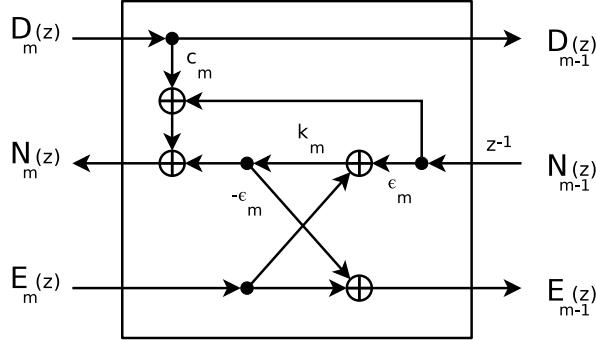


Figure 7.20: Transposed one multiplier lattice filter module

module gives

$$\begin{aligned} E_{m-1} &= -\epsilon_m^2 k_m z^{-1} N_{m-1} + (1 - k_m \epsilon_m) E_m \\ N_m &= (1 + \epsilon_m k_m) z^{-1} N_{m-1} + c_m D_m + k_m E_m \\ (1 + \epsilon_m k_m) z^{-1} N_{m-1} &= N_m - c_m D_m - k_m E_m \\ (1 + \epsilon_m k_m) E_{m-1} &= -k_m N_m + c_m k_m D_m + E_m \end{aligned}$$

Finally

$$\begin{bmatrix} z^{-1} N_{m-1}(z) \\ D_{m-1}(z) \\ E_{m-1}(z) \end{bmatrix} = \frac{1}{1 + \epsilon_m k_m} \begin{bmatrix} 1 & -c_m & -k_m \\ 0 & 1 + \epsilon_m k_m & 0 \\ -k_m & c_m k_m & 1 \end{bmatrix} \begin{bmatrix} N_m(z) \\ D_m(z) \\ E_m(z) \end{bmatrix}$$

The single-multiplier lattice basis functions, $\Lambda_i(z)$, are orthogonal but not orthonormal. The basis functions, $\Phi_i(z)$, of the normalised-scaled lattice are orthonormal so, for a unit-power white noise input, each node of the normalised-scaled lattice has unit-power. Equation 7.11 shows the expected power at each lattice node of the unscaled one multiplier lattice. Recall that this follows from the definition of the inner product of Schur polynomials and Parseval's equality:

$$\|g_i\|_2^2 = \frac{1}{2\pi i} \oint G(z) G_i(z^{-1}) \frac{dz}{z}$$

where $G_i(z)$ is the z -transform of g_i , the unit-impulse response from internal node i to the output. Similarly, for the scaled filter with unit-impulse response from the input to internal node i , $f_i(k)$, the ℓ_2 scaling rule gives

$$\|f_i\|^2 = \sum_{k=0}^{\infty} f_i^2(k) = 1$$

The noise gain calculation assumes that there is an equivalent white noise input at the i -th node of variance $q^2/12$ where q is the quantisation step size. The noise at the output is due to this node is $q \|g_i\|^2/12$. If the filter is not scaled then the unit impulse response from the input to internal node i is not unity. The filter is scaled by dividing coefficients on branches entering node i by $\|f_i\|$ and multiplying coefficients on branches leaving the i -th node by $\|f_i\|$. Therefore the output noise variance for scaled filters is

$$\sigma^2 = \frac{q^2}{12} \sum_i \|f_i\|^2 \|g_i\|^2$$

The Octave function `schurOneMlatticeFilter` implements this scaling.

The Octave script `butt3OneM_test.m` implements a 3rd order Butterworth filter with the single-multiplier lattice structure. Annotated results of the script follow.

The multiplier and sign coefficients are

```
k = -0.97432  0.92923 -0.53208
epsilon = -1  -1  -1
```

The scaling factors for each section are

```
p = 3.03862  0.34657  1.80947
```

The numerator polynomial expands in the orthonormal Schur basis, $\Phi_i(z)$, found above, as

```
c = 0.1005013  0.2986163  0.0101661  0.0028982
```

The scaled Butterworth filter noise gain with exact coefficients is

```
ng = 0.98228
```

and the scaled all-pass noise gain is

```
ngap = 5.0000
```

Using the 10-bit random test signal above, the estimated and measured round-off noise variance at the Butterworth output is

```
est_varyd = 0.16519
varyd = 0.16571
```

and at the all-pass output the estimated and measured round-off noise variance is

```
est_varyapd = 0.50000
varyapd = 0.49645
```

For the scaled filter, the signal at each internal unit delay storage element has standard deviation

```
stdxf = 131.32  129.47  127.97
```

Note that the noise gain for the low-pass filter is slightly smaller than for the normalised-scaled lattice filter.

Calculation of the one multiplier lattice filter round-off noise with a retimed state-variable description

The single-multiplier lattice round-off noise variance can also be determined from the state variable representation if the signal flow graph is slowed and retimed as shown in Figure 7.21. Each intermediate node is now associated with a state. The additional states alter the overall transfer function but the round-off noise gains are unchanged. The state-variable equations for this single-multiplier lattice example after slowing and re-timing are calculated in the Octave function *schurOneMlatticeRetimed2Abcd*. The Octave function *schurOneMlatticeFilter* calculates the upper, Butterworth output, edge of Figure 7.4 in a single operation, ie: with a single large accumulator, and the lower, all-pass output, edge with separate truncated accumulations. On the other hand, the Octave function *svf* implements a general state-variable filter with truncated accumulations for each state and a wide accumulator for both the outputs. The round-off noise estimation in *schurOneMlatticeRetimed2Abcd* illustrates that the retiming method of calculating noise gain is much more flexible than the transposed filter method. In addition, as noted in Section 7.7.1, the Schur basis is not truncated at the same time as the lattice coefficients but is still used to calculate the noise gain after truncation. Consequently, it is simpler to use the state-variable description to calculate the round-off noise of the one multiplier lattice with truncated coefficients.

The *schurOneMlatticeRetimed2Abcd* function can calculate round-off noise gain for three different one multiplier lattice filter implementations by selecting the states included through the *filterStr* argument:

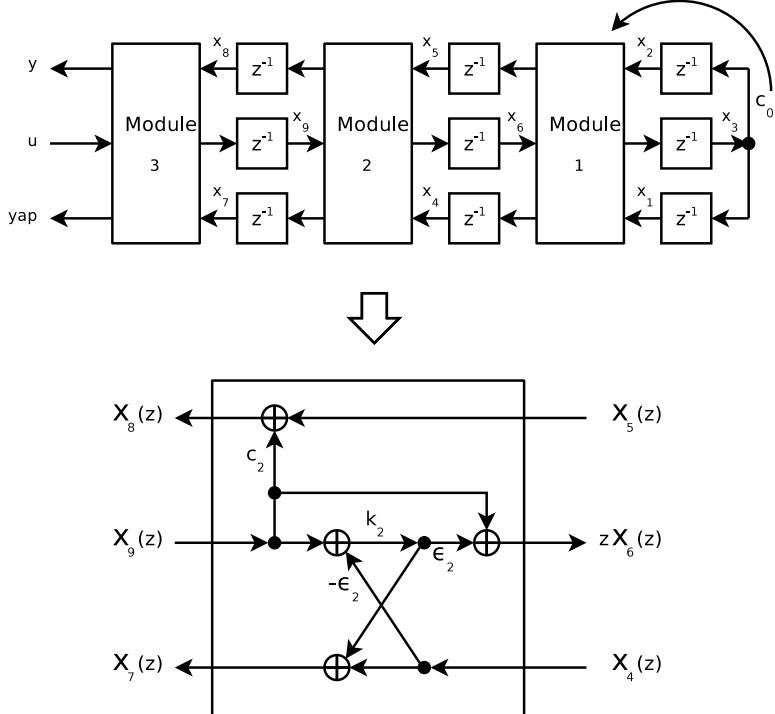


Figure 7.21: Slowed and retimed single-multiplier lattice for the 3rd order Butterworth example

- For “schur” *schurOneMlatticeFilter* calculates the Butterworth output (the upper row of Figure 7.21) in a single wide accumulator and ignores the states on the top edge
- For “ABCD” *svf* calculates both the Butterworth output and the all-pass output in wide accumulators and ignores the states on both the top and bottom edges

In both these cases, states x_1 and x_2 in Figure 7.21 do not contribute to the round-off noise. (In the case of x_2 , c_0 is included in the calculation of state x_5).

The Octave script *butt3OneMSV_test.m* implements this state-variable description and calculates the round-off noise. The results of the script are shown in the following.

When filter type “schur” is selected the round-off noise gains for exact filter coefficients calculated with the Octave function *schurOneMlatticeRetimed2Abcd* match those found with *schurOneMlatticeNoiseGain*.

When filter type “ABCD” is selected the Butterworth filter noise gain is $ngABCD = 0.75000$ and the all-pass noise gain is $ngABCDap = 3.0000$.

With exact coefficients, the globally optimised state-variable implementation has a noise gain of $ngopt = 0.47049$ for the Butterworth filter and $ngoptap = 3.0000$ for the all-pass filter.

After truncating the k and c coefficients to 10 bit 3-signed-digits the Butterworth and all-pass noise gains were $ngf = 1.1019$ and $ngfap = 5.0000$ respectively. Note that the state scaling factors, p , are not truncated. With 10 bit 2 signed-digit coefficients (as used for the normalised-scaled lattice in Section 7.7.1) the passband response of the filter was unacceptable.

With rounding-to-nearest arithmetic truncation in the accumulator, the estimated and measured round-off noise variance of the one multiplier Schur lattice filter with 10-bit 3-signed-digit truncated coefficients are, for the Butterworth filter:

```
est_varyd = 0.17516
varyd = 0.17410
```

and for the allpass filter:

```
est_varyapd = 0.50000
varyapd = 0.49283
```

The standard deviations of the internal states of the filter are

```
stdxf = 137.06 129.04 127.82
```

Figure 7.22 shows the transfer function of the filter found by cross-correlation of the input and Butterworth and all-pass filter outputs when the filter is calculated as a Schur scaled one multiplier lattice with 10-bit 3-signed-digit truncated coefficients in the Octave function *schurOneMlatticeFilter*.

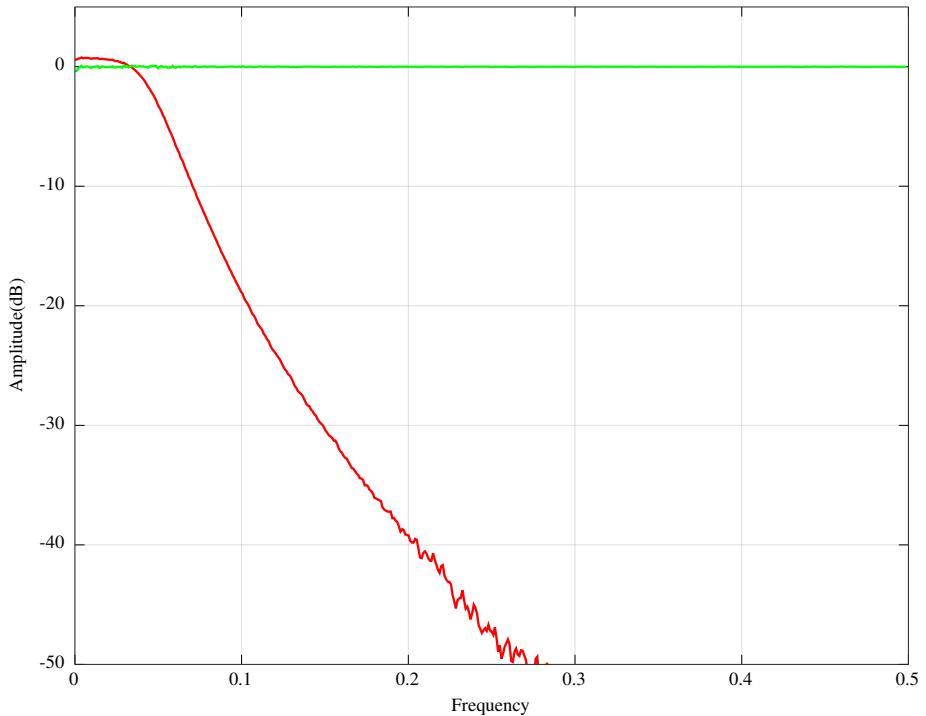
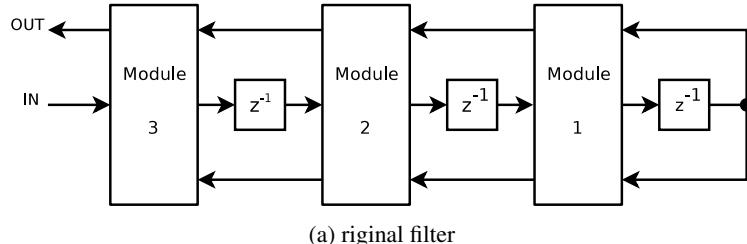
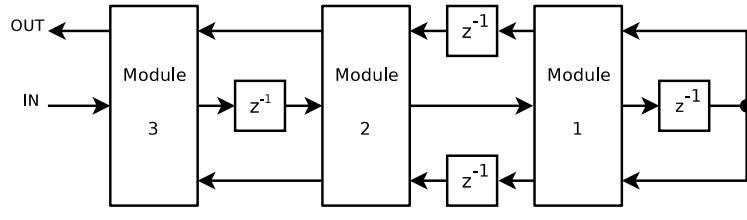


Figure 7.22: Transfer function of the slowed and retimed single-multiplier lattice 3rd order Butterworth filter implemented in Schur lattice form with 10-bit 3-signed-digit truncated coefficients



(a) original filter



(b) Filter after retiming

Figure 7.23: Example of retiming a 3rd order lattice filter

7.8 Retiming Schur lattice filters

7.8.1 Retiming a 3-rd order lattice filter

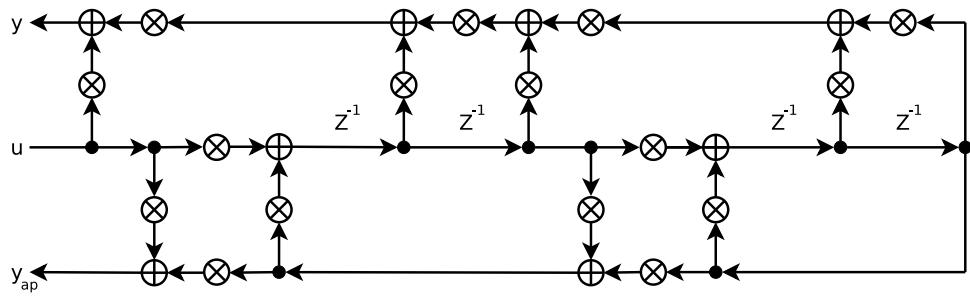
Figure 7.23a shows a representation 3rd order lattice filter and Figure 7.23b shows the filter after retiming by moving the second filter delay to the upper and lower branches of the signal flow graph. The total delay in each loop of the signal flow graph is unchanged so the transfer function of the filter is also unchanged. Retiming limits the maximum latency of each filter update calculation to the latency of two sections and changes the round-off noise performance of the filter.

7.8.2 Retiming a 4-th order Schur normalised-scaled lattice filter

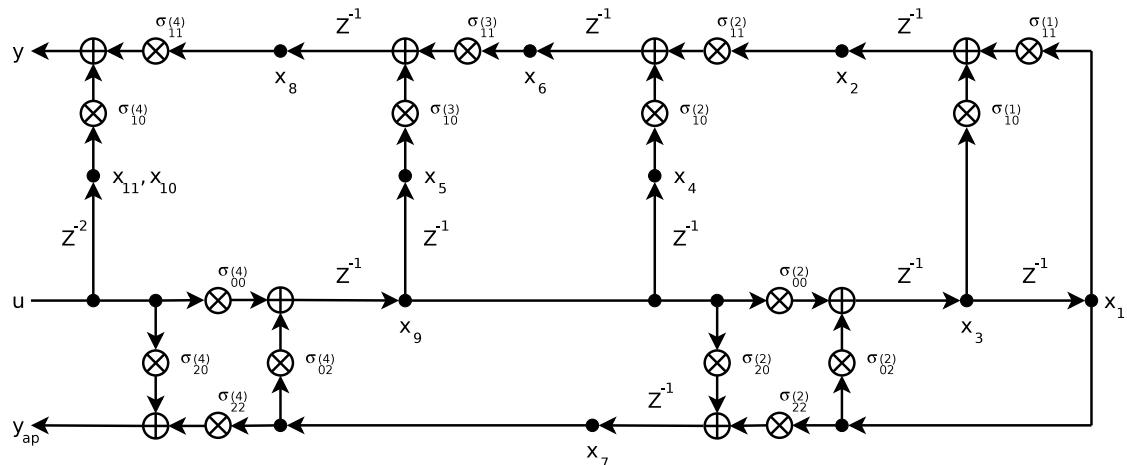
The normalised-scaled lattice filter of order N shown in Figure 7.7 has a delay-free path, or latency, of N adds and N multiplies to the tapped and all-pass outputs, y and y_{ap} respectively. Section 4.2 describes a procedure for inserting fractional delays in the signal flowgraph of the filter to reduce the length of the delay free path. The clock rate of the resulting filter is a multiple of the sample rate. Parhi [49, Chapter 4] discusses formal methods for retiming a signal flow graph. Figure 7.24a shows a simplified view of a normalised-scaled Schur lattice filter implementing a 4-th order transfer function with a denominator polynomial having coefficients only in z^{-2} . Figure 7.24b shows the example filter after redistributing the delay in each loop of the graph so that the total delay around that loop is unchanged. Each state update in the resulting filter has the form $pq + rs$. In this case the filter group delay is increased by 2 samples. In general, this retiming scheme increases the group delay by 1 sample for each second-order section.

Following the notation of Figure 7.10, the state variable equations for the filter shown in Figure 7.24b are:

$$\begin{aligned}
 x_1(k+1) &= x_3(k) \\
 x_2(k+1) &= \sigma_{11}^{(1)}x_1(k) + \sigma_{10}^{(1)}x_3(k) \\
 x_3(k+1) &= \sigma_{02}^{(2)}x_1(k) + \sigma_{00}^{(2)}x_9(k) \\
 x_4(k+1) &= x_9(k) \\
 x_5(k+1) &= x_9(k) \\
 x_6(k+1) &= \sigma_{11}^{(2)}x_2(k) + \sigma_{10}^{(2)}x_4(k) \\
 x_7(k+1) &= \sigma_{22}^{(2)}x_1(k) + \sigma_{20}^{(2)}x_9(k) \\
 x_8(k+1) &= \sigma_{10}^{(3)}x_5(k) + \sigma_{11}^{(3)}x_6(k) \\
 x_9(k+1) &= \sigma_{02}^{(4)}x_7(k) + \sigma_{00}^{(4)}u(k) \\
 x_{10}(k+1) &= x_{11}(k) \\
 x_{11}(k+1) &= u(k)
 \end{aligned}$$



(a) In original form



(b) After retiming

Figure 7.24: Signal flow graph of a 4th order normalised-scaled Schur lattice filter with a denominator polynomial having coefficients only in z^{-2}

$$y(k) = \sigma_{11}^{(4)}x_8(k) + \sigma_{10}^{(4)}x_{10}(k)$$

$$y_{ap}(k) = \sigma_{22}^{(4)}x_7(k) + \sigma_{20}^{(4)}u(k)$$

The Octave script *schur_retimed_test.m* designs a 4-th order low-pass filter with cutoff frequency $0.05f_S$ and denominator coefficients in z^{-2} . The script minimises the amplitude response error with the Octave *fminunc* function. The barrier function of Tarczynski *et al.* [8] is added to the response error to constrain the locations of the roots of the denominator polynomial and ensure that the filter is stable. The barrier function is implemented in Octave function *WISEJ.m*. Figure 7.25 shows the simulated amplitude response of the retimed Schur lattice filter.

7.8.3 Retiming a 6th-order Schur one-multiplier lattice filter

Figure 7.26 shows the signal flow graph of a retimed 6th order one-multiplier Schur lattice filter with a denominator polynomial having coefficients only in z^{-2} . Both filter implementations have the same number of delays around the corresponding loops of their signal flow graphs. Retiming the filter reduces the latency in the calculation of the outputs and does not change the amplitude and group delay responses. If N is the filter order then the retimed filter has an additional $\frac{N}{2} - 1$ states. The Octave function *schurOneMR2lattice2Abcd*, exercised by the Octave script *schurOneMR2lattice2Abcd_test.m*, implements retiming of a one-multiplier Schur lattice filter with a denominator polynomial having coefficients in z^{-2} only.

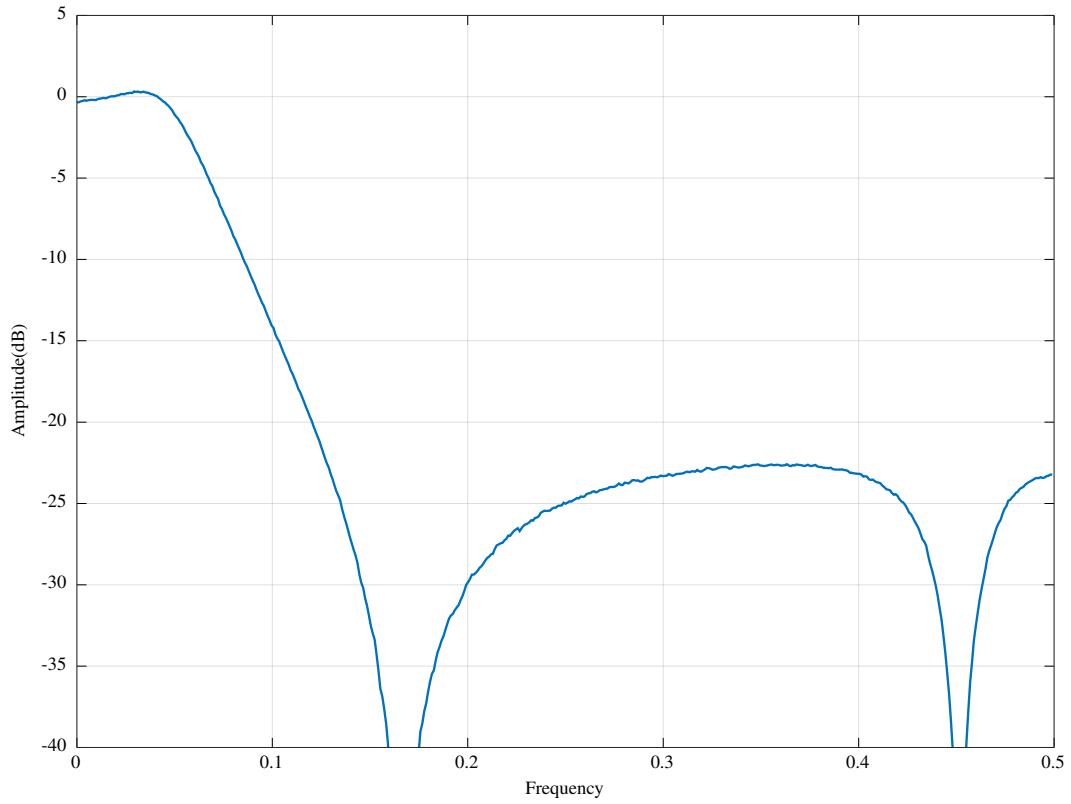
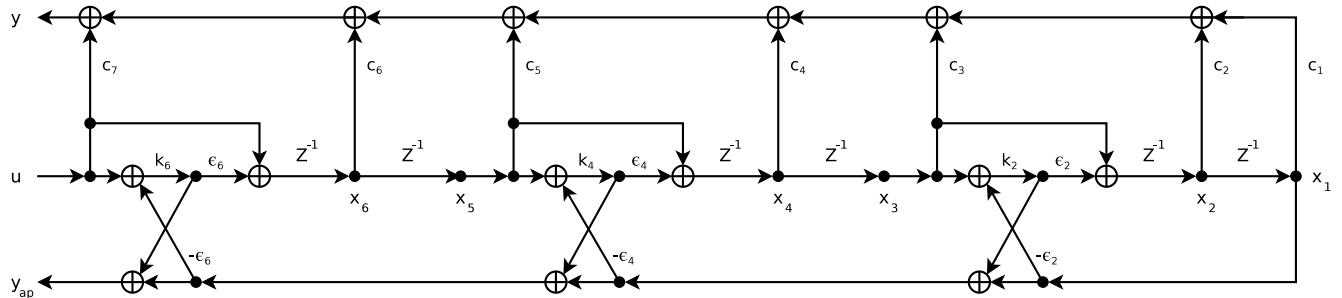
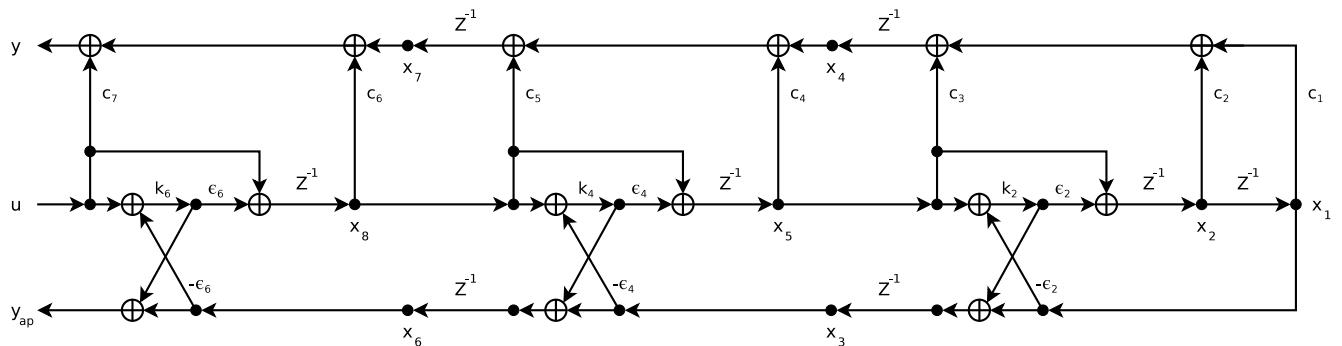


Figure 7.25: Simulated amplitude response of the retimed 4th order normalised-scaled Schur lattice filter



(a) Signal flow graph of a 6th order one-multiplier Schur lattice filter with a denominator polynomial having coefficients only in z^{-2}



(b) Signal flow graph of a 6th order one-multiplier Schur lattice filter with a denominator polynomial having coefficients only in z^{-2} after retiming

Figure 7.26: Original and retimed signal flow graphs of a 6th order one-multiplier Schur lattice filter with a denominator polynomial having coefficients only in z^{-2} . The number of delays around each loop of the signal flow graph is unchanged.

7.8.4 Frequency transformations of retimed Schur lattice filters

As illustrated in Section 7.8, an implementation of a rational transfer function having denominator coefficients only in powers of z^{-2} can be retimed and pipelined so that the resulting filter is suitable for hardware implementation. This section gives an example of the effect of frequency transformations on such a filter transfer function. The Octave script *freq_trans_structure_test.m* designs an 8-th order low-pass filter prototype with a denominator polynomial having coefficients only in powers of z^{-2} . The script minimises the amplitude response error with the Octave *fminunc* function. The barrier function of Tarczynski *et al.* [8] is added to the response error to constrain the locations of the roots of the denominator polynomial and ensure that the filter is stable. The barrier function is implemented in Octave function *WISEJ.m*. The relative weights of the pass-band and stop-band response errors are 1 : 100. The transfer function numerator and denominator polynomials for the low-pass filter prototype are, respectively:

```
n = [ 0.0857526461, 0.2721065334, 0.5924693555, 0.8872367637, ...
      1.0183181186, 0.8872367931, 0.5924694628, 0.2721066332, ...
      0.0857528266 ];
dR = [ 1.0000000000, 0.0000000000, 2.0227725009, 0.0000000000, ...
      1.3779306177, 0.0000000000, 0.3584982390, 0.0000000000, ...
      0.0507148469 ];
```

Figure 7.27a shows the response of the low-pass prototype filter.

Figure 7.27b shows the response of a band-pass filter generated from the low-pass prototype filter with the following frequency transformation:

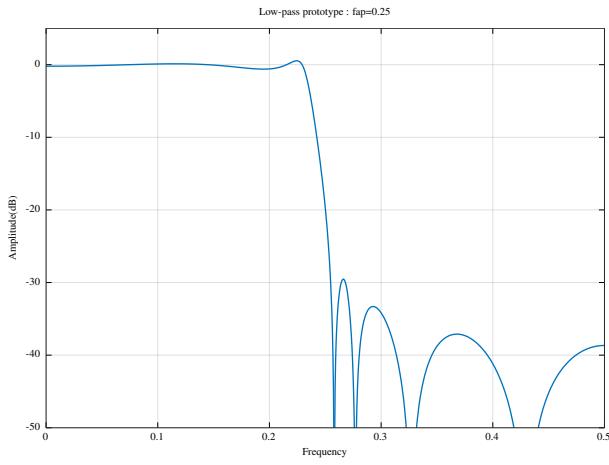
```
pA = phi2P([0.1 0.25])
pA = [ 1.0000e+00 -6.7508e-01 3.2492e-01 ]
```

The denominator polynomial of the resulting band-pass filter has coefficients in powers of z^{-1} .

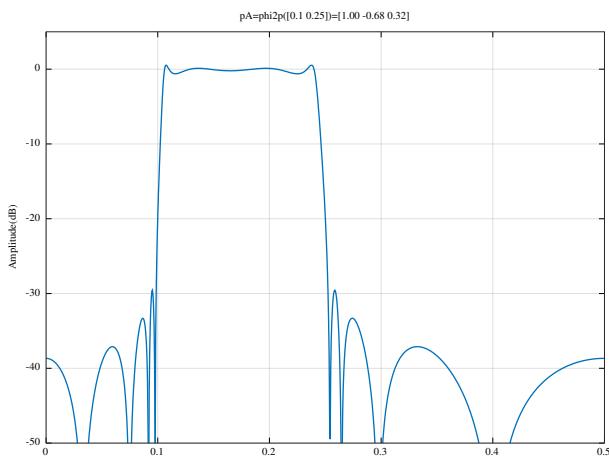
Figure 7.27c shows the response of a band-pass filter generated from the low-pass prototype filter with a frequency transformation that is symmetrical about $\frac{f_S}{4}$:

```
pB = phi2P([0.2 0.3])
pB = [ 1.0000e+00 0.0000e+00 5.0953e-01 ]
```

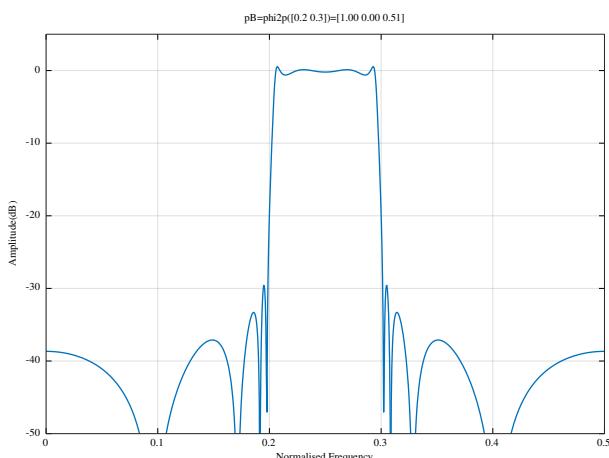
In this case, both the numerator and denominator polynomials of the resulting band-pass filter have coefficients only in powers of z^{-2} .



(a) Amplitude response of a low-pass filter prototype having denominator polynomial coefficients in z^{-2}



(b) Amplitude response of a band-pass filter with frequency transformation [0.1 0.25]



(c) Amplitude response of a band-pass filter with frequency transformation [0.2 0.3]

Figure 7.27: Frequency transformations of a low-pass filter prototype having denominator polynomial coefficients in powers of z^{-2} only

7.9 Frequency transformations and Schur normalised-scaled lattice filter round-off noise

Section 3.3 shows the frequency transformation of a state variable filter. Section 5.8 shows some results, given by *Mullis and Roberts* [16, Section III], concerning the noise gain of a frequency transformed state variable filter. *Koshita et al.* [89] point out that the normalised-scaled Schur lattice all-pass filter has controllability and observability Grammians $K = W = I$. It follows from Equation 5.4 that the frequency transformed filter constructed with the state variable implementation of that Schur normalised-scaled all-pass lattice filter has the same noise gain as the globally optimised implementation of the frequency transformed filter (see Section 5.5). Further, the state-transition matrix of the lattice all-pass frequency transformation filter is Hessenberg in form so the state-transition matrix of the frequency transformed filter has many zero entries. The Octave function `tfp2schurNSlattice2Abcd` implements the frequency transformation of a prototype filter as follows:

1. construct the state variable implementation, $\{\alpha, \beta, \gamma, \delta\}$, of the Schur normalised-scaled lattice filter corresponding to the all-pass frequency transformation, $F(z)$
2. construct the globally optimised state-variable implementation, $\{a, b, c, d\}$, of the low-pass filter prototype, $H(z)$
3. construct the state variable implementation, $\{A, B, C, D\}$, of the frequency transformed filter, $H(F(z))$

The Octave script `tfp2schurNSlattice2Abcd_test.m` exercises `tfp2schurNSlattice2Abcd` with the 5th order elliptic low-pass to multiple stop-band filter example of Section 3.4 shown in Figure 3.6. Table 7.1 shows the number of non-zero coefficients, the noise gain and the estimated noise variance in bits of the multiple band-stop filter implemented by `tfp2schurNSlattice2Abcd`, a globally optimised state variable filter, a Schur normalised-scaled lattice filter and a Schur one-multiplier lattice filter (neglecting the one-multiplier state scaling).

	Non-zero coefficients	Noise gain	Noise variance(bits)
ABCD transformed	286	6.44	0.62
Globally optimised	961	6.44	0.62
Schur normalised-scaled lattice	180	18.88	1.66
Schur one-multiplier lattice	60	13.60	1.22

Table 7.1: Schur NS lattice frequency transformation round-off noise example : number of non-zero coefficients, noise gain and estimated output roundoff noise variances for a prototype 5th order elliptic low-pass filter transformed to a multiple band-stop filter.

7.10 Summary

Given a transfer function $H(z) = N_N(z) / D_N(z)$ the design procedure for the normalised-scaled lattice filter is:

1. Compute the Schur polynomials from the denominator $D_N(z)$
2. Compute the parameters k_i for $i = N, N - 1, \dots, 1$
3. Expand the numerator $N_N(z)$ in the Schur polynomial basis by the polynomial expansion algorithm
4. Synthesise the filter by Equations 7.16, 7.17, 7.18, 7.19, 7.20 and 7.21

Given a transfer function $H(z) = N_N(z) / D_N(z)$ the design procedure for the one multiplier lattice filter is:

1. Compute the Schur polynomials from the denominator $D_N(z)$
2. Compute the parameters k_i for $i = N, N - 1, \dots, 1$
3. Find the k_i with greatest magnitude and recursively compute the sign parameters by Algorithm 15. Scale the Schur polynomial basis functions appropriately
4. Expand the numerator $N_N(z)$ in the Schur polynomial basis by the polynomial expansion algorithm
5. Synthesise the filter by Equations 7.8 and 7.9

The normalised-scaled lattice filter is not structurally passive, as is the one multiplier lattice filter, but the normalised-scaled lattice filter appears to be less sensitive to coefficient truncation than the one multiplier lattice filter. The normalised-scaled lattice filter is inherently scaled. The one multiplier lattice filter is not orthonormal and requires rather complex scaling.

Chapter 8

Orthogonal state variable filters

Chapter 7 described the Schur decomposition of an arbitrary rational filter transfer function into the normalised-scaled or one-multiplier tapped all-pass lattice representations with state covariance matrix $K = I$. The all-pass lattice filters so constructed are (in the terminology of *Roberts and Mullis*) *orthogonal* filters. In this chapter I review the method of *Roberts and Mullis* [76, Section 10.4] for decomposing an arbitrary transfer function into 2×2 block diagonal coordinate rotations that can be “*manipulated to achieve either high-speed, parallel computation using a sparsely connected array of processor modules or low-speed single processor realizations*”¹.

8.1 Definition of orthogonal state variable filters

If, for matrix O , $OO^T = I$, then $O \in \mathcal{O}$, the group of *orthogonal* matrixes. If U has *complex* elements and $UU^\dagger = I$ then $U \in \mathcal{U}$, where \dagger is the *conjugate transpose* matrix operator, and \mathcal{U} is the group of *unitary* matrixes. An all pass filter has magnitude response $|H(e^{j\theta})| = 1$ for all real θ . The $m \times m$ matrix transfer function $H(z)$ of m inputs and m outputs and complex elements is *all-pass* if $H(e^{j\theta})$ is unitary:

$$H(e^{j\theta}) \in \mathcal{U}(m) \quad \forall \text{ real } \theta$$

DEFINITION: A state variable filter $F = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ belongs to the set of orthogonal, all-pass filters $\mathcal{F}(m, n)$ if

1. $AA^T + BB^T = I_{(n \times n)}$
2. $H(e^{j\theta}) = D + C(e^{j\theta}I - A)^{-1}B \in \mathcal{U}(m) \quad \forall \text{ real } \theta$

In addition, F belongs to the sub-set $\mathcal{F}_0(m, n)$ if it has the following equivalent properties:

1. $F \in \mathcal{F}_0(m, n)$
2. (A, B) is controllable
3. (A, C) is observable
4. $\det(\lambda I - A) = 0 \Rightarrow |\lambda| < 1$

As noted above, apart from F_L , each factor in the factored state variable description is an orthogonal matrix. For all-pass filters F_L is also orthogonal. More formally

$$\begin{aligned} F \in \mathcal{O}(m+n) &\Rightarrow F \in \mathcal{F}(m, n) \\ F \in \mathcal{F}_0(m, n) &\Rightarrow F \in \mathcal{O}(m+n) \end{aligned}$$

¹In Appendix G, I review the method presented by *Vaidyanathan and Mitra* [72, 71] for decomposing an odd-order transfer function into the sum of two all-pass filters

For any orthogonal matrix $F \in \mathcal{O}(N)$, there are N possible filters, one each in $\mathcal{F}(m, N - m)$, $1 \leq m \leq N$. The choice of m depends on how F is partitioned

$$F = \begin{bmatrix} A_{(N-m) \times (N-m)} & B_{(N-m) \times m} \\ C_{m \times (N-m)} & D_{m \times m} \end{bmatrix}$$

Note that transformations of the all-pass filter by $T \in \mathcal{O}(N)$,

$$\begin{aligned} F' &= \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} \\ &= \left[\begin{array}{c|c} T^T AT & T^T B \\ \hline CT & D \end{array} \right] \\ &= \left[\begin{array}{cc} T & 0 \\ 0 & I \end{array} \right]^T \left[\begin{array}{cc} A & B \\ C & D \end{array} \right] \left[\begin{array}{cc} T & 0 \\ 0 & I \end{array} \right] \end{aligned}$$

are also members of \mathcal{F} and that F' represents an alternative realisation of $H(z) = D + C(zI - A)^{-1}B$.

8.2 The Lattice Orthogonal All-Pass Filter Section

Figure 8.1 shows a possible realisation of the orthogonal lattice all-pass filter section. There are two inputs and two outputs. The filter matrix is given by

$$\begin{bmatrix} x' \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & \cos \theta & \sin \theta \\ 1 & 0 & 0 \\ 0 & -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ u_1 \\ u_2 \end{bmatrix}$$

and the corresponding transfer function is

$$H(z) = \begin{bmatrix} z^{-1} \cos \theta & z^{-1} \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

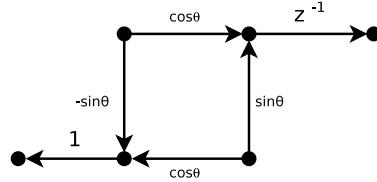


Figure 8.1: Lattice section

8.3 Noise gain of orthogonal filters

For an orthogonal state variable structure, $K = I$, and the noise gain is

$$\begin{aligned} g_{orth} &= \frac{1}{n} \sum_{i=1}^n K_{ii} W_{ii} \\ &= \frac{1}{n} \sum_{i=1}^n \mu_i^2 \end{aligned}$$

where the μ_i^2 are the *second order modes* or eigenvalues of KW . For comparison, a globally optimised minimum noise filter with equal word-lengths has

$$\begin{aligned} g_{min} &= \frac{1}{n} \sum_{i=1}^n K_{ii} W_{ii} \\ &= \left[\frac{1}{n} \sum_{i=1}^n \mu_i \right]^2 \end{aligned}$$

Thus the difference between g_{orth} and g_{min} is the difference between a second moment and the square of a first moment, which is the variance. The two are only equal if the μ_i are all equal. Recall that the second order modes of the globally optimised filter are invariant under a frequency transformation.

8.4 Realisation of arbitrary filters from orthogonal sub-filters

Let $G(z)$ be the transfer function for an order n filter with 1 input and 1 output. Suppose

$$G = \begin{bmatrix} A & B \\ C_1 & D_1 \end{bmatrix}$$

is an orthogonal filter implementation with

$$K = AA^T + BB^T = I$$

The first n rows of G are orthonormal, but have dimension $n + 1$ so there exist C and D for which

$$F = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \in \mathcal{O}(n+1)$$

is orthonormal. The Schur decomposition of G produces an all-pass filter corresponding to F . The conversion from F to G can be found from

$$\begin{aligned} G &= \begin{bmatrix} A & B \\ C_1 & D_1 \end{bmatrix} F^T F \\ &= \begin{bmatrix} I & 0 \\ C_1 A^T + D_1 B^T & C_1 C^T + D_1 D^T \end{bmatrix} F \\ &= G_0 F \end{aligned}$$

where

$$\begin{aligned} [\Gamma \ \delta] &= [C_1 \ D_1] F^T \\ G_0 &= \begin{bmatrix} I & 0 \\ \Gamma & \delta \end{bmatrix} \end{aligned}$$

G_0 can be simplified by finding the transformation

$$T = \begin{bmatrix} T_0 & 0 \\ 0 & 1 \end{bmatrix} \in \mathcal{O}(n+1)$$

so that

$$\Gamma T_0 = [0 \ \gamma]$$

where γ is a scalar. T_0 can be found as the product of a series of rotations (each of which are members of $\mathcal{O}(n)$) that zero out the leading elements of Γ

$$[\gamma_i \ \gamma_{i+1}] \begin{bmatrix} \cos \theta_i & \sin \theta_i \\ -\sin \theta_i & \cos \theta_i \end{bmatrix} = [0 \ \gamma'_{i+1}]$$

where θ_i is given by

$$\tan \theta_i = \frac{\gamma_i}{\gamma_{i+1}}$$

Now transform G by T

$$\begin{aligned} G' &= T^{-1} GT \\ &= T^{-1} (G_0 F) T \\ &= (T^{-1} G_0 T) (T^{-1} F T) \\ &= G'_0 F' \end{aligned}$$

where

$$G'_0 = \begin{bmatrix} I & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \gamma & \delta \end{bmatrix}$$

In other words, $F' \in \mathcal{F}(2, n-1)$ is a two-input, two-output all-pass filter and one of the all-pass outputs of F' becomes a state of $G(z)$. Figure 8.2 is a representation of the corresponding implementation of $G(z)$ (see [76, Figure 10.4.5]).

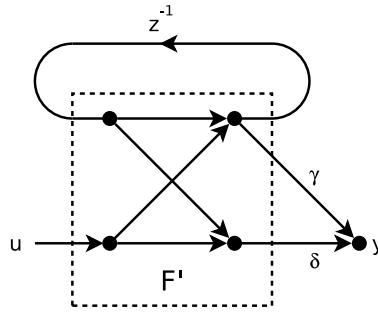


Figure 8.2: Signal flow graph of an arbitrary transfer function constructed from a two-input two-output all-pass filter

Roberts and Mullis [76, pp. 460-461] describe the following procedure for factoring a two-input, two-output all-pass filter, F , into the product of 2×2 block diagonal coordinate rotations of the form

$$\begin{bmatrix} T_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathcal{O}(n+1)$$

that preserve G'_0 ²:

1. Construct a series of similarity transformations that zero the elements of F which are more than 2 sub-diagonals below the main diagonal. For example, with $G(z)$ of order $n = 5$ so that $F \in \mathcal{O}(6)$:

$$F' = \begin{bmatrix} X & X & X & X & X & X \\ X & X & X & X & X & X \\ X & X & X & X & X & X \\ 0_6 & X & X & X & X & X \\ 0_4 & 0_5 & X & X & X & X \\ 0_1 & 0_2 & 0_3 & X & X & X \end{bmatrix} = T_6^T T_5^T T_4^T T_3^T T_2^T T_1^T F T_1 T_2 T_3 T_4 T_5 T_6$$

The indexes on the sub-diagonal zeros indicate the order of construction. The transformations T_i are constructed in the same manner as those that zero the leftmost elements of Γ . In the example:

$$\begin{aligned} & \begin{bmatrix} I_{4 \times 4} & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix}^T \begin{bmatrix} X & X & X & X & X & X \\ X & X & X & X & X & X \\ X & X & X & X & X & X \\ F_{4,1} & X & X & X & X & X \\ F_{5,1} & F_{5,2} & X & X & X & X \\ F_{6,1} & F_{6,2} & F_{6,3} & X & X & X \end{bmatrix} \begin{bmatrix} I_{4 \times 4} & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix} \\ &= \begin{bmatrix} X & X & X & X & X & X \\ X & X & X & X & X & X \\ X & X & X & X & X & X \\ F'_{4,1} & X & X & X & X & X \\ F'_{5,1} & F'_{5,2} & X & X & X & X \\ 0 & F'_{6,2} & F'_{6,3} & X & X & X \end{bmatrix} \end{aligned}$$

2. Factor F' with $q = 2n - 1$ coordinate rotations that zero the sub-diagonal elements

$$F'' = F' F_1 \cdots F_q$$

These are *not* similarity transformations. In fact, F'' will be diagonal with elements ± 1 ³. The diagonal elements can be included in the rotations so that:

$$F' = F_q^T \cdots F_1^T$$

For example:

$$F'' = \begin{bmatrix} X & X & X & X & X & X \\ 0_9 & X & X & X & X & X \\ 0_4 & 0_8 & X & X & X & X \\ 0 & 0_3 & 0_7 & X & X & X \\ 0 & 0 & 0_2 & 0_6 & X & X \\ 0 & 0 & 0 & 0_1 & 0_5 & X \end{bmatrix} = F' F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8 F_9$$

²This procedure is known as the *upper Hessenberg reduction* [26, Section 7.4.2].

³This is a consequence of F'' being upper-triangular and $F'' \in \mathcal{O}(n+1)$.

This realisation of $G(z)$ as a tapped lattice filter requires, as a starting point, an orthogonal state variable representation of $G(z)$ with $K = I$. The *singular value decomposition* [26, Theorem 2.5.2] provides the required similarity transform⁴. Alternatively, the Schur decomposition of a transfer function (reviewed in Chapter 7, based on Parhi [49, Chapter 12]) realises an orthogonal lattice filter without requiring the calculation of K to find an initial similarity transformation⁵. In practice, for larger filters, I have found that the Schur decomposition is more accurate.

The Octave function *orthogonaliseTF* returns the orthogonal decomposition of a rational polynomial transfer function. In the default configuration, *orthogonaliseTF* uses the Schur decomposition of the transfer function to find an orthogonal state transition matrix, A , and then finds the 2×2 block diagonal rotation matrixes in the orthogonal decomposition of the filter. As an example, the Octave script *orthogonaliseTF_test.m* finds the orthogonal decomposition of a 9th order elliptic low-pass filter with cutoff frequency $0.05f_S$. The orthogonal decomposition of the filter contains 17 non-trivial 2×2 block diagonal rotation matrixes. The noise gain of the orthogonal filter is 2.83. The noise gain of the corresponding minimum noise state variable filter is 1.64.

8.4.1 An example of structural variations

The decomposition of the filter transfer function into 2×2 block diagonal coordinate rotations permits straightforward modification of the filter factorisation. For example, suppose we design a 5-th order filter with the structure shown in Figure 8.3a. The 2×2 rotation matrixes on the diagonals are represented by the letters across two rows. All other elements are either 0 or, on the diagonal, 1. The structure shown in Figure 8.3a uses a single rotation time multiplexed across 9 matrixes (plus G'_0 , although that matrix is not orthogonal). The orthogonal matrixes above the arrow in Figure 8.3a can be combined into a single similarity transform, T , and applied to G_0F to form a new filter factorisation:

$$F' = T^{-1}FT$$

This is equivalent to a circular shift of the order of the orthogonal factor matrixes. Also, the individual factors commute if the 2×2 sub-matrixes on the diagonal have no common rows. Hence the filter matrix can be compacted as shown in Figure 8.3b, which represents a sequence of 2, 3, 2, 2 rotation elements in a depth-4 pipeline.

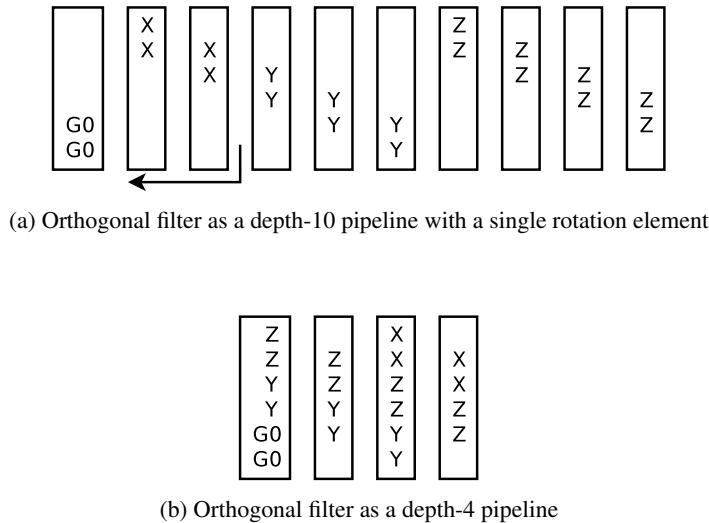


Figure 8.3: Orthogonal structures for a 5-th order filter

Roberts and Mullis [76, pp. 462-467] show other examples of orthogonal filter factorisation.

⁴The SVD of K gives $U^T KV = \text{diag}(\sigma_1, \dots, \sigma_n)$ where U and V are orthogonal $n \times n$ matrixes (ie: $U^T U = I$)

⁵In fact the Schur decomposition of the transfer function is equivalent to the orthogonal decomposition of the state transition matrix presented here.

Chapter 9

Feedforward and feedback of state quantisation error in state variable filters

Chapter 5 reviewed the optimisation of a state variable digital filter for minimum roundoff noise. The resulting filter has $\mathcal{O}(N^2)$ non-zero coefficients. Part 6 showed how a filter could be implemented as a cascade of second-order sections with section optimal roundoff noise performance. Chapters 7 and 8 reviewed the design of lattice filters with good roundoff noise performance and low coefficient sensitivity. An alternative technique for reducing the roundoff noise in the filter output is the feedback of the state quantisation error [32, 96, 21] in order to improve the roundoff noise performance of the filter with a small increase in complexity. *Mullis and Roberts* [18] show that complete or “optimal” error feedback is equivalent to increasing the word-lengths of the coefficients and state registers in the “usual” state variable filter without feedback. They point out that sub-optimal error feedback, that is choosing the registers that receive feedback and choosing coefficients that are a power-of-2, may provide improved noise performance with reduced complexity. Overflow oscillations and coefficient sensitivity must be considered on a case-by-case basis. *Li and Gevers* show that the “delta” operator method, where the usual time-shift operator, z , is replaced by $\delta = \frac{z-1}{\Delta}$ is a special case of the residue feed back method of *Williamson* [21]. *Lu and Hinamoto* [96] describe use of SQP-Relaxation non-linear optimisation to find the near-optimal signed-digit coefficients of a diagonal feedback matrix. Here I follow the paper by *Williamson* [21].

9.1 Problem formulation

Figure 9.1 [96, Figure 2] shows a block diagram of a state variable digital filter with feedback and feed-forward of the state quantisation error e . The state variable quantiser is Q , the feedback error transfer function is δ and the feed-forward error transfer function is η . As shown in Section 2.5 the unquantised state variable equations are

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned}$$

In his treatment *Williamson* [21] states that “the coefficients $\{A, B, C, D\}$, while not necessarily less than 1 in magnitude, are assumed to have an exact fractional B_0 bit representation. The filter states $x(\tilde{k})$ and the ouptut $y(\tilde{k})$ all have a fractional $B + B_0$ bit representation and the input $u(k)$ is a B bit fraction. The quantiser $Q[x(\tilde{k})]$ rounds the $B + B_0$ fraction $x(\tilde{k})$ to B bits after the arithmetic operations are complete. Fixed point arithmetic is implemented using a two’s complement representation (where the sign bit is not counted)”. The *roundoff residue*, $e(k)$, is a $B + B_0$ bit fraction having zero in the most significant B bits:

$$e(k) = \tilde{x}(k) - Q[\tilde{x}(k)]$$

The roundoff residue sequence, $e(k)$, is modelled as a zero-mean noise process with covariance

$$\begin{aligned} \sigma_e &= E\{e(k)e^T(k)\} \\ &= \frac{q^2}{12}I \end{aligned}$$

where $q = 2^{-B}$. When the state variables are quantised, the state variable equations for the error feedback/feed-forward filter of Figure 9.1 are

$$\tilde{x}(k+1) = AQ[\tilde{x}(k)] + Bu(k) + \delta e(k)$$

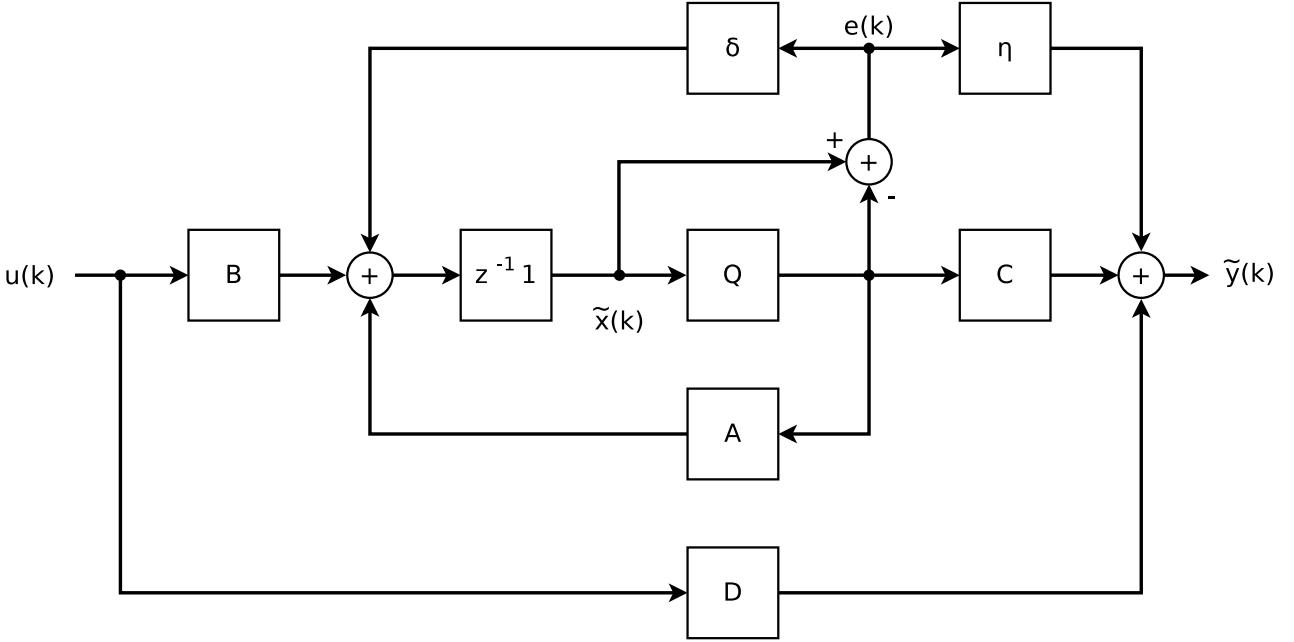


Figure 9.1: Error feedback and feed-forward in state variable digital filters. (Lu and Hinamoto [96, Figure 2])

$$\tilde{y}(k) = CQ[\tilde{x}(k)] + Du(k) + \eta e(k)$$

The round off noise for the filter is

$$\begin{aligned}\Delta x(k+1) &= A\Delta x(k) + (A - \delta)e(k) \\ \Delta y(k) &= C\Delta x(k) + (C - \eta)e(k)\end{aligned}$$

where $\Delta x(k) = \tilde{x}(k) - x(k)$ and $\Delta y(k) = \tilde{y}(k) - y(k)$. As shown in Section 2.9, the frequency domain transfer function from the state quantisation error to the output roundoff noise is:

$$\Delta H(z) = C(zI - A)^{-1}(A - \delta) + (C - \eta)$$

Section 5.3.3 shows that the probability of overflow can be minimised by appropriate l_2 -norm scaling of the state variables. If the input signal, $u(k)$, is zero-mean and unit variance, then the steady-state covariance of the state variables is, since $\tilde{x}(k)$ and $e(k)$ are uncorrelated

$$K = AKA^T + BB^T + q^2(A - \delta)(A - \delta)^T$$

If $q^2 \ll 1$ then

$$K \approx AKA^T + BB^T$$

For optimum l_2 -norm scaling $\text{diag}\{K\} = I$. As shown in Section 2.11, for $\Delta y(0) = 0$, the output roundoff noise error is

$$\Delta y(k) = \sum_{l=0}^{k-1} CA^l(A - \delta)e(k - m - 1) + (C - \eta)e(k)$$

As shown in Section 5.4, the output roundoff noise variance due to truncation of each state is $\sigma_e^2 \sigma^2$, where

$$\begin{aligned}\sigma^2 &= \text{trace} \left\{ (A - \delta)(A - \delta)^T W + (C - \eta)(C - \eta)^T \right\} \\ W &= AWA^T + CCT^T\end{aligned}$$

Williamson considers the noise minimisation problem of finding a realisation $\{A, B, C, D\}$ of $H(z)$ and integer-valued feedback gains $\{\delta, \eta\}$ such that the output noise gain, g , is minimised, subject to the state scaling constraint, $\text{diag}\{K\} = I$.

Given an initial realisation, $\{A_0, B_0, C_0, D\}$, of $H(z)$, Williamson [21, Section III] defines the *residue matrix* as

$$P_0 = (I - A)^T W_0 + W_0(I - A)$$

where

$$W_0 = A_0 W_0 A_0^T + C_0 C_0^T$$

He shows that the eigenvalues, $\{\mu_i^2\}$ of $K_0 W_0$, and $\{\rho_i^2\}$ of $K_0 P_0$, are invariant under a similarity transformation, T , and are positive. In Section 8.3, the eigenvalues $\{\mu_i^2\}$ are referred to as the *second-order modes* of $H(z)$. Williamson calls $\{\rho_i^2\}$ the *residue modes*. Further, he calls a realisation $\{A_0, B_0, C_0, D\}$ *input balanced* if $K_0 = I$ and $W_0 = M^2 = \text{diag}\{\mu_1^2, \dots, \mu_N^2\}$, where N is the order of $H(z)$ ¹. An input balanced structure can be transformed to an *internally balanced* structure with $K_1 = W_1 = M$ by means of the similarity transformation $T_1 = M^{-\frac{1}{2}}$.

9.2 Minimisation of round-off noise with $\delta = I$ and $\eta = 0$

Williamson [21, Section V] considers the “problem of finding the optimal transformation, T , and the optimal integer residue feedback matrixes δ_T and η_T , which together minimise the output round-off noise”. He proves the theorem shown as Algorithm 19 for the suboptimal case for a low-pass narrow-band filter in which $\delta_T = I$ and $\eta_T = 0$. (For a high-pass narrow-band filter $\delta_T = -I$). If $\delta_T = 0$ and $\eta_T = 0$ then the minimum noise gain is

$$g_M = \frac{\sum_{k=1}^N \mu_k}{\sqrt{N}}$$

Hence, error feedback reduces the noise gain only if the sum of the residue modes is less than the sum of the second-order modes.

Algorithm 19 Minimisation of round off noise in error feedback/feedforward state variable filters.

(See Williamson [21, Theorem 5.2]).

Let M define the second-order modes of a stable N th order filter, $H(z)$, with an input-balanced realisation, $\{A_0, B_0, C_0, D\}$. Consider the finite word length implementation $\{A_T = T^{-1}A_0T, B_T = T^{-1}B_0, C_T = T^TC_0, D\}$. Then subject to the l_2 -norm scaling constraint, the identity state residue correction, $\delta_T = I$, and the zero output residue correction, $\eta_T = 0$, the minimum noise filter is defined by

$$\begin{aligned} T &= R_1 \Pi R_0^T \\ \Pi &= \text{diag}\{\pi_1, \dots, \pi_N\} \end{aligned}$$

where

$$\pi_m^2 = \frac{1}{\rho_m} \frac{\sum_{k=1}^N \rho_k}{N}$$

and for unitary matrixes R_0 and R_1

$$\begin{aligned} \text{diag}\{R_0 \Pi^{-1} R_0^T\} &= I \\ R_1^T P_0 R_1 &\text{ is diagonal} \end{aligned}$$

where

$$P_0 = (I - A_0)^T M^2 + M^2 (I - A_0)$$

The residue modes $\{\rho_k\}$ are the square roots of the eigenvalues of P_0 . The corresponding minimum noise gain, g_I is

$$g_I = \frac{\sum_{k=1}^N \rho_k}{\sqrt{N}}$$

The Octave script *error_feedback_test.m* attempts to reproduce Williamson’s *Example 6.3*. This example considers error feedback for the 6th order filter given by $H(z) = \frac{q(z)}{p(z)}$ where

```
q=[0.0047079 -0.0251014 0.0584417 -0.0760820 0.0584417 -0.0251014 0.0047079];
p=[1 -5.6526064 13.3817570 -16.9792460 12.1764710 -4.6789191 0.7525573];
```

I found different values for the second order modes. I found that the optimum noise gain was 1.333 and the noise gain for an orthogonal filter was 2.086 whereas Williamson reports a noise gain of $g_M^2 = 5.8272$ in the latter case. I found the following values for the residue modes:

¹If $K = I$, then the filter is *orthogonal*. See Part 8. An SVD transformation of the corresponding W matrix gives the input balanced structure.

```
rho = [ 0.268561 0.141649 0.127042 0.058299 0.021251 0.005512]
```

and a noise gain of $g_I = 0.254$. The error-feedback filter was simulated with 8 bit coefficients by adding 2 bits to the state storage in the Octave function *svf.m* (as the mantissa, to the right of the binary point). The estimated output roundoff noise is 0.323 bits. The measured output roundoff noise is 0.308 bits. The output round-off noise is dominated by the noise due to the rounding of the output, $y(k)$: $\frac{1}{\sqrt{12}} = 0.2887$ bits. The simulated response is shown in Figure 9.2.

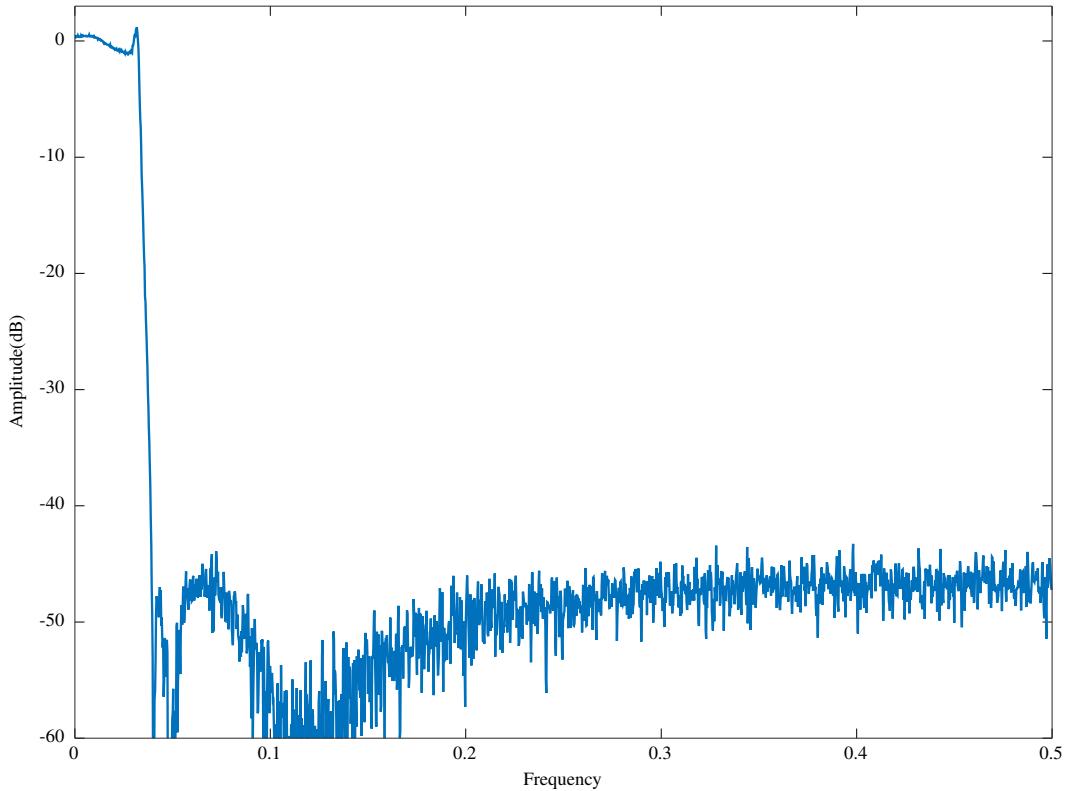


Figure 9.2: Simulated response of a 6-th order filter with 8 bit coefficients and a 2 bit mantissa (for error feedback) in the state variables. (After Williamson [21, Example 6.3]).

Part II

Constrained optimisation of the IIR filter frequency response

Chapter 10

IIR filter design using Sequential Quadratic Programming with the transfer function defined by pole and zero locations

The transfer function or response of an IIR digital filter is usually written in the z -transform domain similarly to

$$H(z) = \frac{b(z)}{a(z)} = \frac{\sum_{l=0}^m b_l z^{-l}}{1 + \sum_{l=1}^n a_l z^{-l}} \quad (10.1)$$

Where $0 \leq m \leq n$. Poles at zero may be added to ensure that the filter is *causal*. For a causal filter, the output of the filter only depends inputs from the past. When z is interpreted as a single sample delay this means that the order of the numerator is less than or equal to the order of the denominator. A non-causal filter can be used to process signals off-line. Alternatively, the non-causal filter can be decomposed by polynomial division into a parallel and cascade combination. A *stable* filter has all the roots of the denominator polynomial inside the unit circle in the z -plane. A *minimum phase* filter has both poles and zeros inside the unit circle in the z -plane so that the inverse filter is also stable.

Equation 1.1 minimises the error in the *complex* response (amplitude and phase). In contrast, in the following the weighted amplitude and group-delay response squared-error is minimised and filter stability is obtained by constraints on the pole location. The *sequential quadratic programming* (SQP) optimisation method linearises the response error at the current estimate of the solution and solves the corresponding constrained quadratic programming problem to find a new estimate that minimises the linearised error. The procedure stops at a *local* minimum of the error surface. Unfortunately, the filter design error surface usually has many, many local minima. Achieving success in finding a good solution (for which the error value at the local minimum is close to the *global* minimum) depends on the initial solution and the formulation and weighting of the error function¹.

10.1 Problem Statement

The magnitude and group delay responses of an IIR digital filter are optimised in terms of a coefficient vector containing the gain-zero-pole definition of the transfer function

$$x = \left[K, R_{0,1}, \dots, R_{0,U}, R_{p,1}, \dots, R_{p,V}, r_{0,1}, \dots, r_{0,\frac{M}{2}}, \phi_{0,1}, \dots, \phi_{0,\frac{M}{2}}, r_{p,1}, \dots, r_{p,\frac{Q}{2}}, \phi_{p,1}, \dots, \phi_{p,\frac{Q}{2}} \right] \quad (10.2)$$

This vector represents the poles and zeros of a filter with gain factor K , the radiiuses of U real zeros and V real poles and the radiiuses and angle of $\frac{M}{2}$ pairs of conjugate zeros and $\frac{Q}{2}$ pairs of conjugate poles. Note that for decimation factor, R , the poles are in fact on the z^R plane and each pole in x corresponds to R poles on the z plane. Appendix B shows the amplitude, phase and group delay responses and their gradients in terms of the coefficient vector.

¹An alternative to Equation 1.1 is to linearise the cost function, \mathcal{E}_H , by:

$$\hat{\mathcal{E}}_H = \sum_{j=1}^{NB} \int_{f_{l,j}}^{f_{u,j}} \frac{W_j(f)}{|\hat{D}(f)|^2} |N(f) - D(f) H_d(f)|^2 df$$

where \hat{D} is the previous estimate of D . See, for example, Dumitrescu and Niemistö [13]

The weighted squared-magnitude of the response error, \mathcal{E}_A , is:

$$\mathcal{E}_A(x) = \sum_{j=1}^{NB} \int_{f_{l,j}}^{f_{u,j}} W_{Aj}(f) [A(x, f) - A_D(x, f)]^2 df$$

where NB is the number of frequency bands, $f_{u,j}$ the upper and $f_{l,j}$ the lower frequency band edges, W_{Aj} the weighting function for each frequency band, and A is the actual and A_D the desired magnitude response. The weighted time delay error, \mathcal{E}_T , is similarly expressed in terms of W_{Tj} the weighting function for each frequency band, the actual group delay response, T , and the desired group delay response, T_D .

The optimal filter design minimises the total weighted squared response error, $\mathcal{E} = \mathcal{E}_A + \mathcal{E}_T$, subject to constraints $g_i(x) \geq 0$ where $x \in \mathcal{D} \subset \mathbb{R}_N$ and the filter is stable for $x \in \mathcal{D}$. The method of *Lagrange multipliers* minimises the *Lagrangian* function:

$$\mathcal{L}(x, \lambda_i) = \mathcal{E}(x) - \sum_{i \in \mathcal{A}(x)} \lambda_i g_i(x)$$

where $\mathcal{A}(x)$ represents the set of active constraints at x and λ_i are the Lagrange multipliers (or *dual variables*) for those constraints. The *Karush-Kuhn-Tucker* conditions for the minimum are:

$$\nabla_x \left\{ \mathcal{E}(x) - \sum_{i \in \mathcal{A}(x)} \lambda_i g_i(x) \right\} = 0 \quad (10.3)$$

$$g_i(x) \geq 0 \quad (10.4)$$

$$\lambda_i \geq 0 \quad (10.5)$$

$$\langle \lambda_i, g_i(x) \rangle = 0 \quad (10.6)$$

The gradient of the magnitude error is:

$$\nabla_x \mathcal{E}_A(x) = \sum_{j=1}^{NB} 2 \int_{f_{l,j}}^{f_{u,j}} W_{Aj}(f) [A(x, f) - A_D(x, f)] \nabla_x A(x, f) df$$

The gradient of the delay error is similar.

The literature often refers to the *dual* problem, finding the greatest lower bound of the dual function:

$$\Lambda(\lambda_i) = \inf_{x \in \mathcal{D}} \left\{ \mathcal{E}(x) - \sum_{i \in \mathcal{A}(x)} \lambda_i g_i(x) \right\}$$

10.1.1 Solution of the constrained quasi-Newton optimisation problem

Express the *Karush-Kuhn-Tucker* conditions (Equations 10.3 and 10.4) at x^k in terms of a linearised version of the gradient of the error and linearised constraints (see Appendix F.1):

$$\begin{aligned} \nabla_x \mathcal{E}(x^k) + \nabla_x^2 \mathcal{E}(x^k)(x - x^k) - \sum_{i \in \mathcal{A}(x)} \lambda_i \nabla_x g_i(x^k) &= 0 \\ g_i(x^k) + \nabla_x g_i(x^k)(x - x^k) &\geq 0 \end{aligned} \quad (10.7)$$

where the Hessian of the squared-magnitude response error, \mathcal{E}_A , is:

$$\nabla_x^2 \mathcal{E}_A(x) = \sum_{j=1}^{NB} 2 \int_{f_{l,j}}^{f_{u,j}} W_{Aj}(f) \left\{ [\nabla_x A(x, f)]^2 + [A(x, f) - A_D(x, f)] \nabla_x^2 A(x, f) \right\} df$$

and the Hessian of the group delay error, \mathcal{E}_T , is similar.

Appendix F.5 shows that for a particular set of constraints, $\{g_i(x^k) \mid i \in \mathcal{A}(x^k)\}$, the IIR design problem can be approximated by

$$\mathcal{W}_k d^k - \mathcal{B}_k \lambda^k = -\nabla_x \mathcal{E}(x^k) \quad (10.8)$$

$$-\mathcal{B}_k^T d^k = g(x^k)$$

where \mathcal{W}_k is a positive-definite approximation to the true Hessian matrix derived using the *Broyden-Fletcher-Goldfarb-Shanno* formula (see Appendix F.6.1) and \mathcal{B}_k is the matrix whose columns are the gradients of the active constraints. This is a matrix equation that can be solved for the Lagrange multipliers λ^k and the direction vector d^k :

$$\begin{aligned}\lambda^k &= -(\mathcal{B}_k^T \mathcal{W}_k^{-1} \mathcal{B}_k)^{-1} [g(x^k) - \mathcal{B}_k^T \mathcal{W}_k^{-1} \nabla_x \mathcal{E}(x^k)] \\ d^k &= -\mathcal{W}_k^{-1} [\nabla_x \mathcal{E}(x^k) - \mathcal{B}_k \lambda^k]\end{aligned}$$

At each iteration of the sequential programming method the coefficient vector is updated by

$$x^{k+1} = x^k + \tau^k d^k$$

where d^k is the step direction vector and $\tau^k \in [0, 1]$ is the step-size. τ^k is found by a line search for the minimum of the Lagrangian function

$$\mathcal{L}(\tau^k) = \mathcal{E}(x^k + \tau^k d^k) - \sum_{i \in \mathcal{A}(x^k)} \lambda_i^k g_i(x^k + \tau^k d^k)$$

subject to the constraints. The iteration finishes when the *Karush-Kuhn-Tucker* conditions are satisfied.

10.1.2 Choice of active constraints

Selesnick, Lang and Burrus [38] describe a simple algorithm for selecting the constraints on the magnitude response that apply at each iteration in the design of an FIR filter. The general discussion accompanying that description is applicable here. The following applies equally to the group delay error component, \mathcal{E}_T , of the total error, \mathcal{E} .

The amplitude $A(f)$ of the filter response minimising the L_2 error subject to the peak constraints will touch the upper and lower bound functions at the extremal frequencies of $A(f)$. At each iteration of the algorithm, the set of frequency points at which $A(f)$ touches the constraints is updated. The equality-constrained problem is then solved by the method of Lagrange multipliers. According to the *Karush-Kuhn-Tucker* conditions, the solution to the *equality*-constrained problem solves the corresponding *inequality* constrained problem if all the Lagrange multipliers are non-negative (where the signs of the multipliers are defined appropriately). If on some iteration a multiplier is negative, then the solution to the equality-constrained problem does not solve the corresponding inequality-constrained one. For this reason, constraints corresponding to negative multipliers are sequentially dropped from the set of constraints. Although not proved in theory, this technique appears to converge in practice.

Let the constraint set S be the set of frequencies $S = \{f_1, \dots, f_r\}$ where $f \in [0, \pi]$. Let S be partitioned into two sets S_L and S_U , where S_L is the set of frequencies where the lower bounds apply

$$A(x, f) = L(f)$$

and S_U is the set of frequencies where the upper bounds apply

$$A(x, f) = U(f)$$

Suppose $S_L = \{f_1, \dots, f_q\}$ and $S_U = \{f_{q+1}, \dots, f_r\}$. To minimise $\mathcal{E}_A(x, f)$ subject to these constraints, form the Lagrangian

$$\mathcal{L}(x, \lambda) = \mathcal{E}_A(x, f) - \sum_{i=1}^q \lambda_i [A(x, f) - L(f)] - \sum_{i=q+1}^r \lambda_i [U(f) - A(x, f)]$$

At the minimum of $\mathcal{E}_A(x, f)$ the gradient $\nabla_x \mathcal{L}(x, \lambda) = 0$ and

$$\begin{aligned}\nabla_x \mathcal{E}_A(x, f) - \sum_{i=1}^q \lambda_i \nabla_x A(x, f) + \sum_{i=q+1}^r \lambda_i \nabla_x A(x, f) &= 0 \\ A(x, f_i) &= L(f_i) \quad \text{for } 1 \leq i \leq q \\ A(x, f_i) &= U(f_i) \quad \text{for } q+1 \leq i \leq r\end{aligned}$$

According to the *Karush-Kuhn-Tucker* conditions, when the Lagrange multipliers $\lambda_1, \dots, \lambda_r$ are all non-negative, then the solution to these equations minimises $\epsilon(x, f)$ subject to the inequality constraints

$$A(x, f_i) \geq L(f_i) \quad \text{for } 1 \leq i \leq q$$

Algorithm 20 Exchange algorithm for multiband FIR filters. (*Selesnick, Lang and Burrus*. See [39, p.498]).

1. *Initialisation*: Initialise the constraint sets R and S to the empty set.
 2. *Minimisation with Equality Constraints*: Calculate the Lagrange multipliers associated with the filter that minimizes $\mathcal{E}_A(\omega)$ subject to the equality constraints $A(\omega_i) = L(\omega_i)$ for $\omega \in S_L$, and $A(\omega_i) = U(\omega_i)$ for $\omega \in S_U$.
 3. *Karush-Kuhn-Tucker Conditions*: If there is a constraint set frequency ω_i , for which the Lagrange multiplier λ_i is negative, then remove from the active constraint set, S , the frequency corresponding to the most negative multiplier, and go back to step 2. Otherwise go on to step 4.
 4. *Check for Violation over R*: Calculate the new filter response, $A(\omega_i)$ for frequencies in the alternate constraint set, R . If $A(\omega_i) < L(\omega_i)$ or $A(\omega_i) > H(\omega_i)$ for some $\omega_i \in R$, then remove the frequency corresponding to the greatest violation from R , append that frequency to S , and go back to step 2.
 5. *Multiple Exchange of Constraint Set*: Overwrite the previous constraint set, R , with the current constraint set, S . Set the current constraint set S equal to $S_L \cup S_U$, where S_L is the set of frequency points ω , in $[0, \pi]$ satisfying both $A'(\omega_i) = 0$ and $A(\omega_i) \leq L(\omega_i)$ (ie: troughs failing the constraint) and where S_U , is the set of frequency points ω , in $[0, \pi]$ satisfying both $A'(\omega_i) = 0$ and $A(\omega_i) \geq U(\omega_i)$ (ie: peaks failing the constraint).
 6. *Check for Convergence*: If $A(\omega) \geq L(\omega) - \epsilon$ for all frequency points in S_L and if $A(\omega) \leq U(\omega) + \epsilon$ for all frequency points in S_U , then convergence has been achieved. Otherwise, go back to step 2.
-

$$A(x, f_i) \leq U(f_i) \quad \text{for } q+1 \leq i \leq r$$

Selesnick et al. [39, p.498] modify the algorithm of *Selesnick et al.* [38] so that it can be used in the design of multiband FIR filters, as shown in Algorithm 20. In the following, this algorithm is referred to as the Peak-Constrained-Least-Square (PCLS) error algorithm. The modification referred to is the addition of a second set of constraints. It avoids the cycling of constraint sets that otherwise occurs with multi-band filter designs.

Selesnick et al. [38, Section IV.A] justify the removal of the most negative Lagrange multiplier as follows:

The constraints are on the values of $A(\omega_i)$ for the frequency points ω_i in a constraint set. On each iteration, the constraint set is updated so that at convergence, the only frequency points at which equality constraints are imposed are those where $A(\omega)$ touches the constraint. The equality constrained problem is solved with Lagrange multipliers. The algorithm below associates an inequality-constrained problem with each equality constrained one. According to the Kuhn-Tucker conditions, the solution to the *equality* constrained problem solves the corresponding *inequality* constrained problem if all the Lagrange multipliers are non-negative (where the signs of the multipliers are defined appropriately). If on some iteration a multiplier is negative, then the solution to the equality constrained problem does not solve the corresponding inequality constrained one. For this reason, before the constraint set is updated in the algorithm described below, constraints corresponding to negative multipliers (when they appear) are sequentially dropped from the constraint set. In this way, an inequality constrained problem is solved on each iteration, albeit over a smaller constraint set. It turns out that in the special case of a lowpass filter design considered here, this simple iterative technique converges in practice.

Selesnick and Burrus have published implementations at [88], namely *cl2lp.m* and *cl2bp.m*.

10.1.3 Linearisation of peak constraints

At each iteration the magnitude response and group-delay are linearised about x^k . I follow the description of the linearised constraints given by *Sullivan* [42, p.2855]. The linearised magnitude response is:

$$\hat{A}(x) = A(x^k) + \nabla_x A(x^k)^T (x - x^k)$$

The frequency response magnitude inequality constraints are, in the pass band

$$\begin{aligned} A_D - \hat{A}(x) &\geq 0 \\ \hat{A}(x) - [A_D - \Delta_{AP}] &\geq 0 \end{aligned}$$

and in the stop band

$$\Delta_{A_S} - \hat{A}(x) \geq 0$$

where Δ_{A_p} is the pass band ripple for the desired gain A_D and Δ_{A_S} is the stop band ripple. After linearising about x^k these constraints become

$$\begin{aligned} A_D - A(x^k) - \nabla_x A(x^k)^T (x - x^k) &\geq 0 \\ A(x^k) + \nabla_x A(x^k)^T (x - x^k) - [A_D - \Delta_{A_P}] &\geq 0 \\ \Delta_{A_S} - A(x^k) - \nabla_x A(x^k)^T (x - x^k) &\geq 0 \end{aligned}$$

Similarly, the linearised group-delay is

$$\hat{T}(x) = T(x^k) + \nabla_x T(x^k)^T (x - x^k)$$

and the group-delay inequality constraints in the pass band are

$$\begin{aligned} [T_D + \Delta_D] - \hat{T}(x) &\geq 0 \\ \hat{T}(x) - [T_D - \Delta_T] &\geq 0 \end{aligned}$$

where $\Delta_T = T_{\max} - T_D = T_D - T_{\min}$ is the tolerance for the group-delay error compared with the desired group-delay T_D . As before, the two corresponding linearised constraints on the group-delay in the pass band are

$$\begin{aligned} [T_D + \Delta_T] - T(x^k) - \nabla_x T(x^k)^T (x - x^k) &\geq 0 \\ T(x^k) + \nabla_x T(x^k)^T (x - x^k) - [T_D - \Delta_T] &\geq 0 \end{aligned}$$

The frequency response inequality constraints are calculated at a grid of frequency points. For a low-pass filter there are two amplitude constraints and two group-delay constraints in the pass band and one amplitude constraint in the stop band.

10.1.4 Ensuring the stability of the IIR filter

An IIR filter is stable if the poles of the filter transfer function lie within the unit circle: $|z| < R < 1$. Deczky [3] and Richards [56] define the transfer function in the form of gain, pole locations and zero locations. In this case filter stability is ensured by a simple constraint on the pole radius.

The partial differentials of the amplitude response with respect to the filter coefficients are simpler when expressed in terms of the numerator and denominator polynomials of the transfer function. This has lead many authors to suggest filter stability criteria expressed in terms of the coefficients of the denominator polynomial. Tarczynski *et al.* [8] ensure stability by adding a *barrier* function to the squared error based on the impulse response of the digital filter that corresponds to the denominator polynomial of the filter transfer function being optimised. Lang [57] describes a method for finding successive coefficient vectors based on Rouché's theorem. Dumitrescu and Niemistö [13] compare Lang's method with one that ensures that the updated denominator polynomial remains a Schur polynomial in the vicinity of the current coefficient vector. Lu and Hinamoto [95] express the denominator polynomial as a product of second-order sections and derive a linear inequality stability constraint on the denominator coefficients. Lu [101] describes a stability test based on Cauchy's Argument Principle.

In my opinion, expressing the filter transfer function in the gain-pole-zero form is preferable to the usual polynomial fraction form because the former provides the simplest possible stability criterion.

10.1.5 Selecting an initial filter design

The constraints on the filter design are the desired response and stability.

Windowed FIR initial filters

An FIR filter approximating the desired response can be designed with the Octave *remez* function. Alternatively, the FIR filter can be designed with the “windowing” method, summarised here. The frequency response of a digital filter

$$H(z) = \sum_{k=-\infty}^{\infty} h_k z^{-k}$$

with a lowpass response cutoff at ω_p is

$$\begin{aligned} H(\omega) &= \sum_{k=-\infty}^{\infty} h_k e^{-jk\omega} \\ &= \begin{cases} 1, & |\omega| < \omega_p \\ 0, & \omega_p < |\omega| < \pi \end{cases} \end{aligned}$$

The coefficients of the impulse response are

$$\begin{aligned} h_k &= \frac{1}{2\pi} \int_{-\omega_p}^{\omega_p} e^{jk\omega} d\omega \\ &= \frac{1}{\pi k} \sin k\omega_p \\ &= \frac{\omega_p}{\pi} \operatorname{sinc} k\omega_p \end{aligned}$$

To create an FIR response we truncate the response to length $L = 2N + 1$ with a window function. The window function is selected for main-lobe width and side-lobe suppression. A typical window function is

$$W_k = \begin{cases} [\alpha + (1 - \alpha) \cos \frac{2\pi k}{N}], & |k| \leq N \\ 0, & |k| > N \end{cases}$$

For the Hamming window, $\alpha = 0.54$. The Octave code to implement a windowed FIR filter is:

```
b=2*fc*sinc((-((L-1)/2):((L-1)/2))*2*fc).*hamming(L)
```

where L is the FIR filter length and $fc < 0.5$ is the low-pass cut-off frequency for sampling frequency $f_S = 1$.

Unconstrained optimisation for an IIR initial filter

An arbitrary filter can be refined by unconstrained optimisation with a “barrier” function (see Appendix F.7.2). Tarczynski *et al.* [8], propose minimising the error with the following, so-called, WISE barrier function:

$$(1 - \lambda) \mathcal{E}_H + \lambda \sum_{t=T+1}^{T+M} f^2(t) \quad (10.9)$$

where \mathcal{E}_H is the filter response error defined in Equation 1.1, λ , T and M are suitable constants and $f(t)$ is the impulse response of the filter $F(z) = \frac{1}{D(z)}$. Tarczynski *et al.* provide heuristics for selecting λ , T and M . Typically, $\lambda \in [10^{-10}, 10^{-3}]$, $T \in [100, 500]$ and $M = RK$. The barrier function (the second part of Equation 10.9) is intended to be small when the filter is stable and increase rapidly otherwise. Roberts and Mullis [76, Section 8.3] show that the state space description of the direct-form implementation of $F(z)$ is:

$$\begin{bmatrix} x(t+1) \\ y(t) \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}$$

where:

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -d_{N_d} & -d_{N_d-1} & \cdots & -d_1 \end{bmatrix} \\ B &= [0 \ 0 \ \cdots \ 0 \ 1]^T \\ C &= [-d_{N_d} \ \cdots \ -d_1] \\ D &= 1 \end{aligned}$$

Alternatively, for the filter transfer function, $H(z)$, given by Equation 10.1, Tarczynski *et al.* [8, Appendix 1] show that the filter $\frac{z^{-N_d}}{D(\rho z)}$ can be implemented as a cascade of first order sections, $\frac{z^{-1}}{1-p_i(\rho z)-1}$. $0 < \rho < 1$ is chosen to limit the pole

magnitudes. Let the output of the i th section, y_i , be the input to the $i + 1$ th section, u_{i+1} , and assume the poles are ordered so that $|p_1| \geq |p_2| \geq \dots \geq |p_{N_d}|$, then the state space model for each section is:

$$\begin{aligned}x_i(t+1) &= p_i \rho^{-1} x_i(t) + u_i(t) \\y_i(t) &= x_i(t)\end{aligned}$$

The overall state space model is (for $R = 1$):

$$\begin{aligned}A &= \begin{bmatrix} p_1 \rho^{-1} & 0 & 0 & \cdots & 0 \\ 1 & p_1 \rho^{-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{N_d} \rho^{-1} \end{bmatrix} \\B &= [1 \ 0 \ \dots \ 0 \ 0]^T \\C &= [0 \ 0 \ \dots \ 0 \ 1] \\D &= 0\end{aligned}$$

In both cases, the corresponding impulse response is:

$$f(t) = \begin{cases} 0 & t < 0 \\ D & t = 0 \\ CA^{t-1}B & t > 0 \end{cases}$$

Golub and van Loan [26, Algorithm 11.2.2] show an algorithm, reproduced as Algorithm 21, for efficiently computing the powers of a matrix by cumulative products of the binary powers of the matrix. This algorithm requires at most $2 \times \text{floor}[\log_2(s)]$ matrix multiplies. If s is a power of 2, then only $\log_2(s)$ matrix multiplies are needed.

Algorithm 21 Compute the powers of a matrix (see *Golub and van Loan*, Algorithm 11.2.2)

The following algorithm computes $F = A^s$ for matrix $A \in \mathbb{R}^{n \times n}$ and a positive integer s with binary expansion $s = \sum_{k=0}^K \beta_k 2^k$.

```
Z = A, q = 0
while β_q = 0 do
    Z = Z^2
    q = q + 1
end while
F = Z
for k = q + 1, ..., t do
    Z = Z^2
    if β_k ≠ 0 then
        F = FZ
    end if
end for
```

The Octave script *tarczynski_ex2_standalone_test.m*, designs a filter for the specifications of *Tarczynski et al.* Example 2 [8] with $nN = 24$, $nD = 2$ and $R = 2$ by unconstrained minimisation of Equation 10.9 with *fminunc*. The resulting response is shown in Figure 10.1 and the pole-zero plot of the filter is shown in Figure 10.2.

The Octave function *xInitHd* uses the WISE barrier function to design an initial filter in pole-zero form.

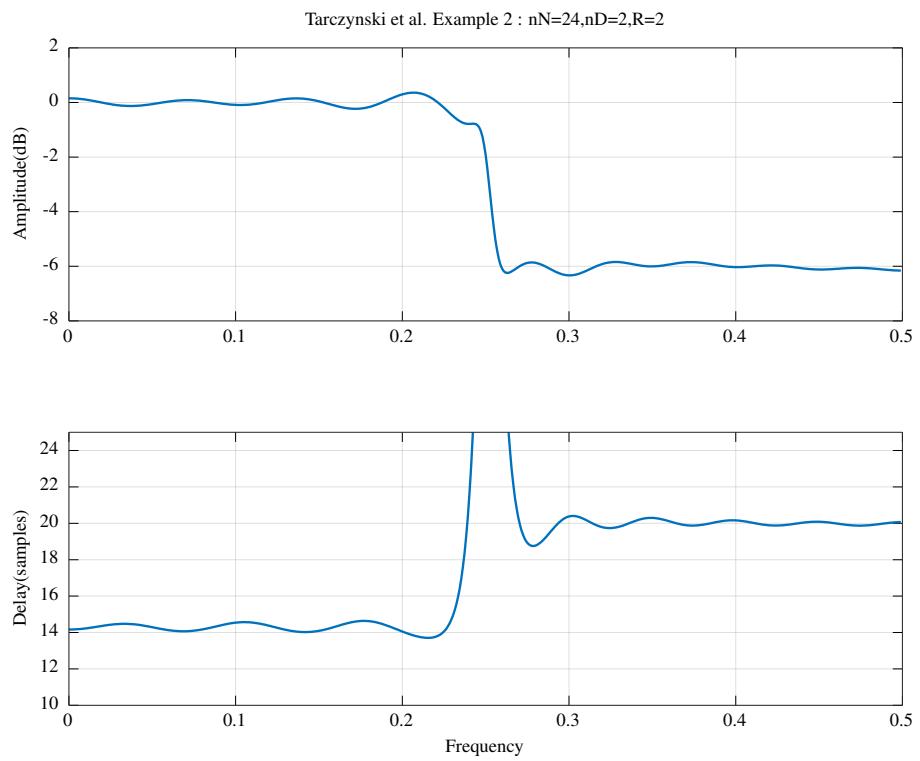


Figure 10.1: Tarczynski et al. Example 2 : Response for $nN=24$, $nD=2$ and $R=2$
with

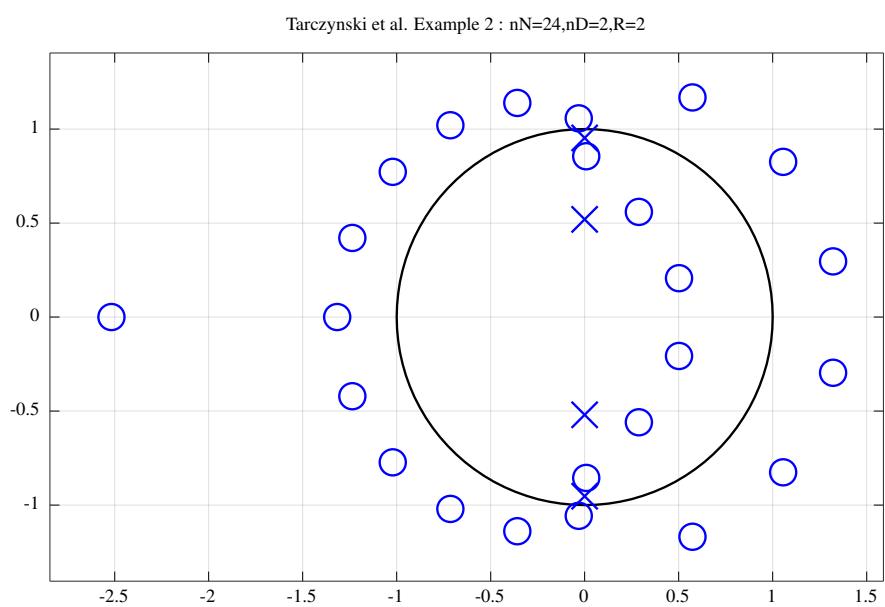


Figure 10.2: Tarczynski et al. Example 2 : Response for $nN=24$, $nD=2$ and $R=2$

10.2 Examples of IIR filter design with SQP and constrained pole and zero locations

10.2.1 Introductory comments on the IIR filter design examples

This section considers examples of IIR filter design. The design procedure is:

- find a valid initial design
- iteratively reduce the mean-square-error (MMSE) with the procedure described in Section 10.1.1.
- iteratively apply constraints with the peak-constrained-least-square-error (PCLS) algorithm described in Section 10.1.2.

For the filter design examples in this section, the filter amplitude, phase and group delay responses and their gradients are calculated by the Octave functions *iirA*, *iirP* and *iirT*, respectively.

Finding an initial IIR filter design

The initial IIR filter design must be stable and bear a passing resemblance to the desired response. The multi-dimensional IIR filter design surface has many local minima and the minimum reached depends on the initial point. After deciding the decimation ratio, R , a stable initial design can be found with:

- an FIR filter and an estimate of the number of poles required
- an arbitrary IIR filter optimised with the Octave function *xInitHd*

The Octave function *xInitHd* begins with an initial filter and designs a rational polynomial transfer function approximation to the desired response by repeated calls to the Octave *fminunc* function. Filter stability is assured by a barrier function based on the impulse response of a filter constructed from the poles of the approximate response. See Section 10.1.5 and *Tarczynski et al.* [8]. The coefficient constraints are defined in the Octave function *xConstraints*. In fact I only constrain the pole radiiuses and do not constrain the scale factor, K , or the complex conjugate pole angles. It is possible that the initial filter produced by *xInitHd* has a negative scale factor and the filter amplitude response is real but negative. See, for example, the initial filter of the $R=2$ decimator filter below. Presumably, the filter was found at a minimum so reversing the sign of the scale factor reverses the sign of the initial amplitude gradients. In practice, a first pass of MMSE optimisation will (hopefully!) reverse the sign of K to match the sign of the desired response while, at the same time, finding a new minimum. The constraints applied to the amplitude response by the implementation of PCLS optimisation in *iir_slb* assume that the amplitude function is positive so an attempt at PCLS optimisation of a filter with a negative scale factor will most likely fail.

Appendix F.9 works through the derivation of the *Goldfarb-Idnani* algorithm [19] for finding an initial solution that meets constraints. As an example, the Octave script *goldfarb_idnani_fir_minimum_phase_test.m* begins with a non-minimum phase FIR bandpass filter and finds a minimum phase FIR filter, that is, an FIR bandpass filter with the constraint that the zero locations lie on or within the unit circle. When the number of constraints on the frequency response is more than ten or so the script fails due to numerical problems with the Hessian matrix. The script does find a minimum-phase FIR filter albeit with a very poor response. In this case I found that a “by-eye” or “cut-and-fit” iterative approach to finding an initial filter is more useful. For example see the Octave script *iir_sqp_slb_fir_bandpass_test.m*.

MMSE optimisation

MMSE optimisation is performed by calling Octave function *iir_sqp_mmse*. The response error is minimised while the real and complex pole radiiuses are constrained to $|r| < rho < 1$. The other parts of the coefficient vector (scale factor, zero radiiuses and pole and zero angles) are not constrained. The *vS* argument to *iir_sqp_mmse* specifies the indexes into the frequency vectors ω_a etc. of linear constraints at the corresponding index into A_{du} or A_{dl} etc. Function *iir_sqp_mmse()* calls *sqp_bfgs* to minimise the objective function *iir_sqp_mmse_fx()* with linear constraints calculated by *iir_sqp_mmse_gx()*. These functions calculate the squared-error in the response amplitude and delay and linear constraints by calls to *iirE()*. That function calculates the error with trapezoidal integration. If the frequency bands are not contiguous then zeros in the weight vector can make frequency transition bands. I found by trial-and-error when running *iir_sqp_mmse* that the SQP Hessian matrix can be initialised with the diagonal elements of the amplitude squared error Hessian. Similarly, the current SQP search point can be updated with the *Armijo-Kim*

line-search algorithm and the *Broyden-Fletcher-Goldfarb-Shanno*(BFGS) Hessian matrix update algorithm (see Appendix F.8.2 and Appendix F.6.1 respectively). The error surface is not quadratic and has many, many local minima. The SQP loop is more stable if, at each iteration, the error surface is approximated by a quadratic surface with the BFGS estimate of the Hessian matrix. Typically, the MMSE optimisation process is iterative; the design process starts with a loose amplitude only specification and the constraints on amplitude, phase and delay are gradually tightened.

There are several reasons why an optimisation attempt might fail:

- the weighting factors for the combined squared error of, for example, amplitude and group delay, are not appropriate. Experiment with the weights.
- the number of iterations may exceed the limit. Increase the iteration limit, *maxiter*,
- the tolerance used may be too low. Increase the tolerance, *tol*.
- the line-search direction does not satisfy the constraints:

```
warning: searching for d within constraints but norm(d)<tol^2!
```

Reduce the maximum coefficient update size, *dmax*.

- the line-search algorithm may not find a minimum or the line-search algorithm may find that the objective function is not approximately quadratic in the search region:

```
Found tau = 0.000000 using goldensection search of Lagrangian
warning: norm(delta)<eps
```

Octave includes an SQP solver function, *sqp*. I have not used this function because it calls separate functions to calculate the error and the constraints. In *iir_sqp_mmse* and in the PCLS solver, *iir_slb*, the constraints are calculated at the same time as the error.

PCLS optimisation

The MMSE optimisation constrains the integrated error over a frequency interval. PCLS optimisation constrains the peaks of the amplitude, phase and group-delay responses. For example, the MMSE design of a low-pass filter may have amplitude peaks above 0dB. The PCLS algorithm of Selesnick, Lang and Burrus [39, Fig.4,p.499] (reproduced above as Algorithm 20), is implemented in the Octave function *iir_slb*. The function handle of the MMSE solver (in this case *iir_sqp_mmse*) is an argument to *iir_slb*. The peak-exchange algorithm of Selesnick *et al.* is much simpler than that of Adams and Sullivan [43, Section IV].

Figure 10.3 shows the failed constraints for the amplitude response of a low pass filter with an amplitude constraint mask. The figure was generated by the Octave script *iir_slb_update_constraints_test.m*. When switching from MMSE to PCLS optimisation, the introduction of constraints on the response peaks alters the Lagrangian and the pass-band and stop-band weights must be modified by trial-and-error.

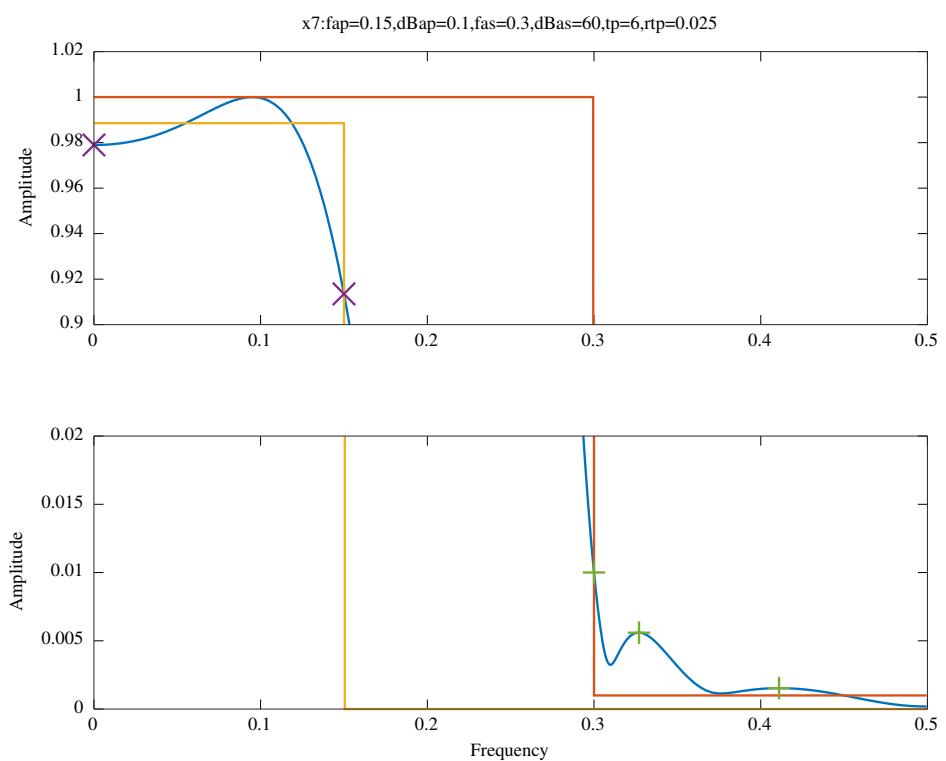


Figure 10.3: Example of failed constraints for a low pass filter

10.2.2 Tarczynski et al. Example 2

Figure 10.1 shows the response of a filter designed with the “WISE” method of *Tarczynski et al.* [8]. In gain-pole-zero format the filter is:

```
Ux=2,Vx=2,Mx=22,Qx=0,Rx=2
x = [ 0.0055318501, ...
      -2.5170628267, -1.3160752171, ...
      -0.9079560306, -0.2702693669, ...
      1.3053646150, 1.2801395738, 1.2456947672, 1.3543532252, ...
      1.3403287270, 1.3017511081, 1.1940391431, 1.0576999798, ...
      0.8556865803, 0.6295823844, 0.5427361878, ...
      2.8130739332, 2.4936224647, 2.1815962607, 0.2206288358, ...
      0.6636910430, 1.1146343826, 1.875693941, 1.6003195241, ...
      1.5609093563, 1.0945324853, 0.3906957551 ]';
```

In the above listing, the first line shows the gain, and subsequent lines show, respectively, 2 real zeros, 2 real poles, the zero radiiuses for 11 conjugate pairs of zeros and the zero angles for 11 conjugate pairs of zeros. In practice, since $R = 2$, the real poles are split into pairs of conjugate poles lying on the imaginary axis. After some experimentation I modified the filter to:

```
Ux0=3,Vx0=2,Mx0=20,Qx0=0,Rx0=2
x0 = [ 0.0400000000, ...
        -1.1000000000, 0.3617300000, 0.3617300000, ...
        -0.8842894000, -0.1495357000, ...
        1.3091281000, 1.2794257000, 1.3538295000, 1.3386509000, ...
        1.3011204000, 1.2418447000, 1.1899991000, 1.0533570000, ...
        0.8306859000, 0.6034298000, ...
        2.8187821000, 2.5029369000, 0.2204128000, 0.6631062000, ...
        1.1173639000, 2.1870677000, 1.8790233000, 1.6013209000, ...
        1.5328472000, 0.9197805000 ]';
```

In this case there are three real zeros. The Octave script *iir_sqp_mmse_tarczynski_ex2_test.m* calls the *iir_sqp_mmse()* function to MMSE optimise this filter. The optimised filter is:

```
Ux1=3,Vx1=2,Mx1=20,Qx1=0,Rx1=2
x1 = [ 0.0007402580, ...
        -1.4828851415, 0.3838193509, 0.3838193510, ...
        -0.2847251560, -0.0188954226, ...
        1.4719814067, 1.4466551512, 1.6138480001, 1.6013771461, ...
        1.5938155430, 1.4225447085, 1.4591287873, 1.3976136170, ...
        0.6772946132, 0.5472460777, ...
        2.8540823426, 2.5559841909, 0.2128541969, 0.6705754680, ...
        1.0878341366, 2.2457387807, 1.8689391245, 1.8351853805, ...
        1.4770396949, 0.8972756089 ]';
```

The corresponding transfer function numerator and denominator polynomials are, respectively:

```
N1 = [ 0.0007402580, 0.0010075454, -0.0000410788, -0.0012882376, ...
        -0.0001374465, 0.0034910791, 0.0004763081, -0.0106891505, ...
        -0.0064731484, 0.0203804486, 0.0158174043, -0.0470794373, ...
        -0.0435911340, 0.1676387802, 0.4570564980, 0.5015721577, ...
        0.2337489150, -0.0289323154, -0.0111170682, 0.1764889444, ...
        -0.2391975851, 0.1809139714, -0.0827777960, 0.0143840620 ]';
```

and

```
D1 = [ 1.0000000000, 0.0000000000, 0.3036205786, 0.0000000000, ...
        0.0053800021 ]';
```

Figure 10.4 shows the response of the optimised filter, Figure 10.5 shows the pass-band response and Figure 10.6 shows the corresponding pole-zero plot.

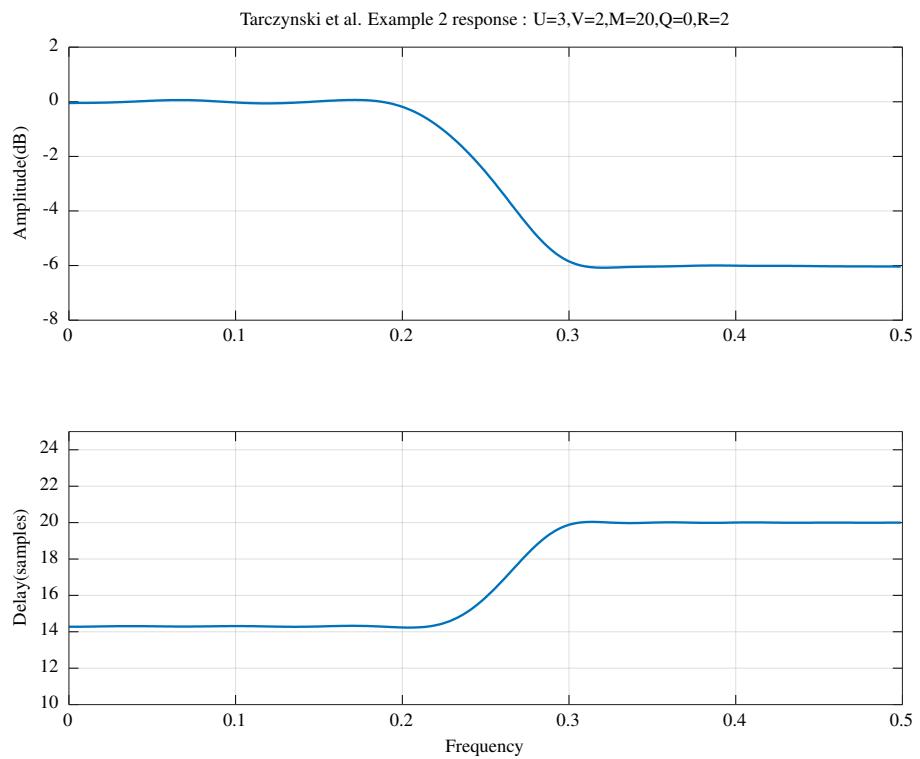


Figure 10.4: Tarczynski et al. Example 2 : Response after MMSE optimisation

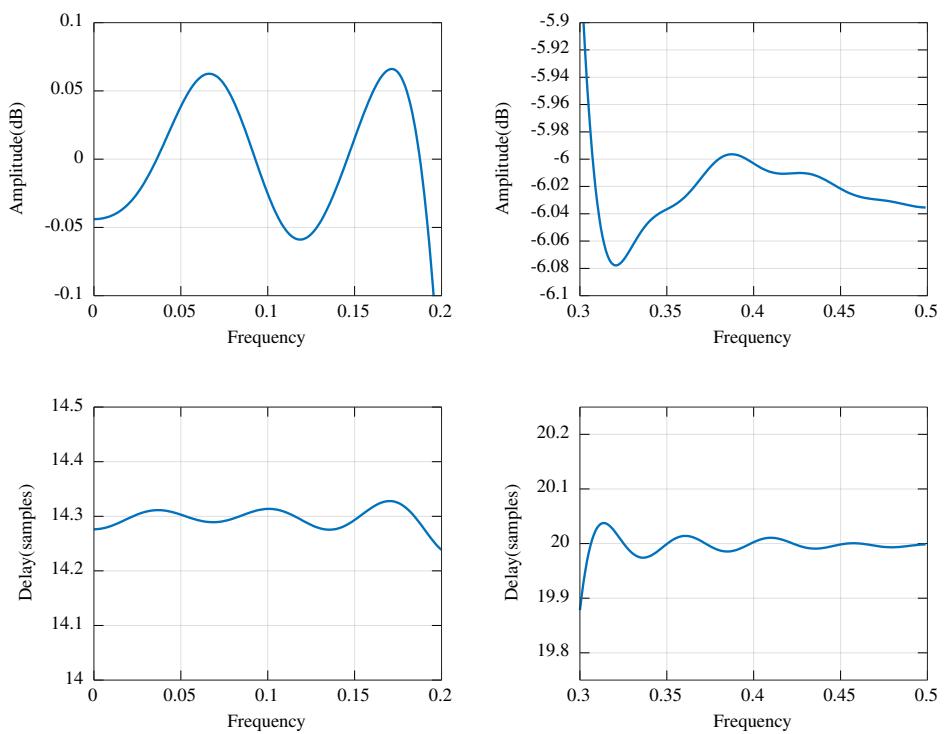


Figure 10.5: Tarczynski et al. Example 2 : Pass-band response after MMSE optimisation

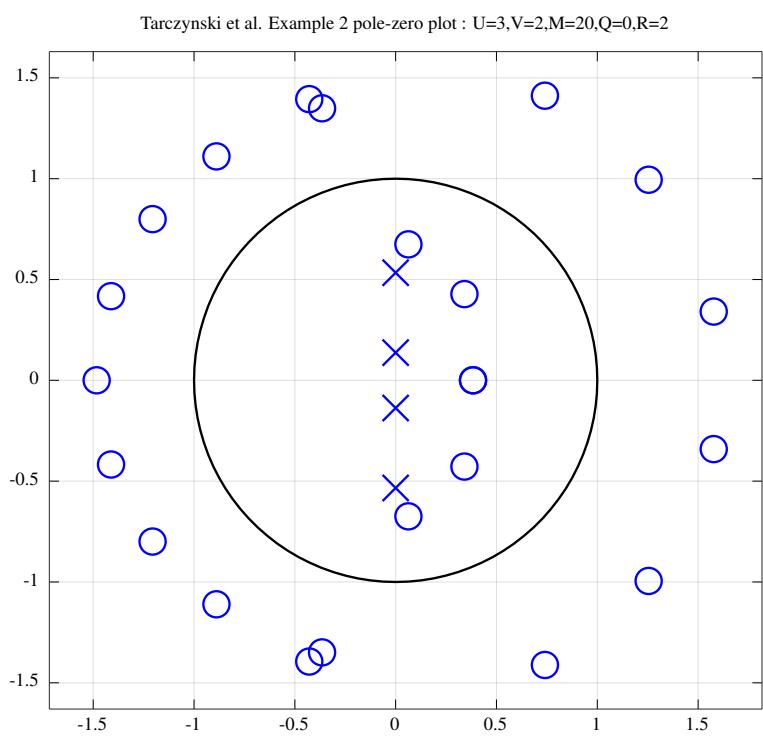


Figure 10.6: Tarczynski et al. Example 2 : Pole-zero plot after MMSE optimisation

10.2.3 Deczky's Example 3

Sullivan and Adams [42, p. 2859] refer to the following example IIR filter design specification as “*Filter 2-iv*”. It is a modified version of Deczky’s Example 3, [3].

$$\begin{aligned} U &= 0 \\ V &= 0 \\ M &= 10 \\ Q &= 6 \\ R &= 1 \\ A(f) &= \begin{cases} 1 & 0 < f < 0.15 \\ 0 & 0.3 < f < 0.5 \end{cases} \\ T(f) &= 10.00 \quad 0 < f < 0.25 \end{aligned}$$

The Octave script *deczky3_sqp_test.m* implements this example. The filter specification defined in that file is

```
n=1000 % Frequency points across the band
tol=0.0002 % Tolerance on relative coefficient update size
fap=0.15 % Pass band amplitude response edge
dBap=0.1 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.25 % Pass band group delay response edge
tp=10 % Nominal filter group delay
tpr=0.0035 % Pass band group delay peak-to-peak ripple
Wtp_mmse1=0.25 % Pass band group delay weight (MMSE pass 1)
Wtp_mmse2=0.5 % Pass band group delay weight (MMSE pass 2)
Wtp_pccls=2 % Pass band group delay weight (PCCLS pass)
fas=0.3 % Stop band amplitude response edge
dBas=30 % Stop band minimum attenuation
Was=1 % Stop band amplitude weight
U=0 % Number of real zeros
V=0 % Number of real poles
M=10 % Number of complex zeros
Q=6 % Number of complex poles
R=1 % Denominator polynomial decimation factor
```

Sullivan and Adams initialise the filter coefficients with a set of coefficients called “IPZS-1” [42, IPZS-1, p. 2860]

```
z=[exp(j*2*pi*0.41),exp(j*2*pi*0.305), ...
    1.5*exp(j*2*pi*0.2),1.5*exp(j*2*pi*0.14),1.5*exp(j*2*pi*0.08)];
p=[0.7*exp(j*2*pi*0.16),0.6*exp(j*2*pi*0.12),0.5*exp(j*2*pi*0.05)];
K=0.0096312406;
x0=[K,abs(z),angle(z),abs(p),angle(p)]';
```

The above listing shows that the initial filter has 0 real zeros, 0 real poles, 5 conjugate pairs of zeros, and 3 conjugate pairs of poles. The initial response is shown in Figure 10.7. The corresponding pole-zero plot is shown in Figure 10.8.

MMSE optimisation of Deczky's Example 3

With weights $W_{ap} = W_{as} = 1$ and $W_{tp} = 0.25$ the first MMSE optimisation pass gives the response shown in Figure 10.9. After some experimentation, the second iteration, with $W_{tp} = 0.5$, gives the response shown in Figure 10.10 with pass-band details shown in Figure 10.11 and the pole-zero plot shown in Figure 10.12.

PCCLS optimisation of Deczky's Example 3

The test script *deczky3_sqp_test.m* now switches to PCCLS optimisation of the MMSE filter. After some experimentation, the final specification becomes $fap = 0.15$, $dBap = 0.1dB$, $Wap = 1$, $fas = 0.30$, $dBas = 30dB$, $Was = 1$, $ftp = 0.25$, $Wtp = 2$, $tp = 10$ samples group delay and $tpr = 0.0035$ samples of peak-to-peak group delay ripple. The resulting amplitude and delay responses are shown in Figure 10.13 with pass-band detail shown in Figure 10.14. The corresponding pole-zero plot is shown in Figure 10.15. The last estimate of the optimised filter vector is, in the gain, zeros and poles form of Equation 10.2:

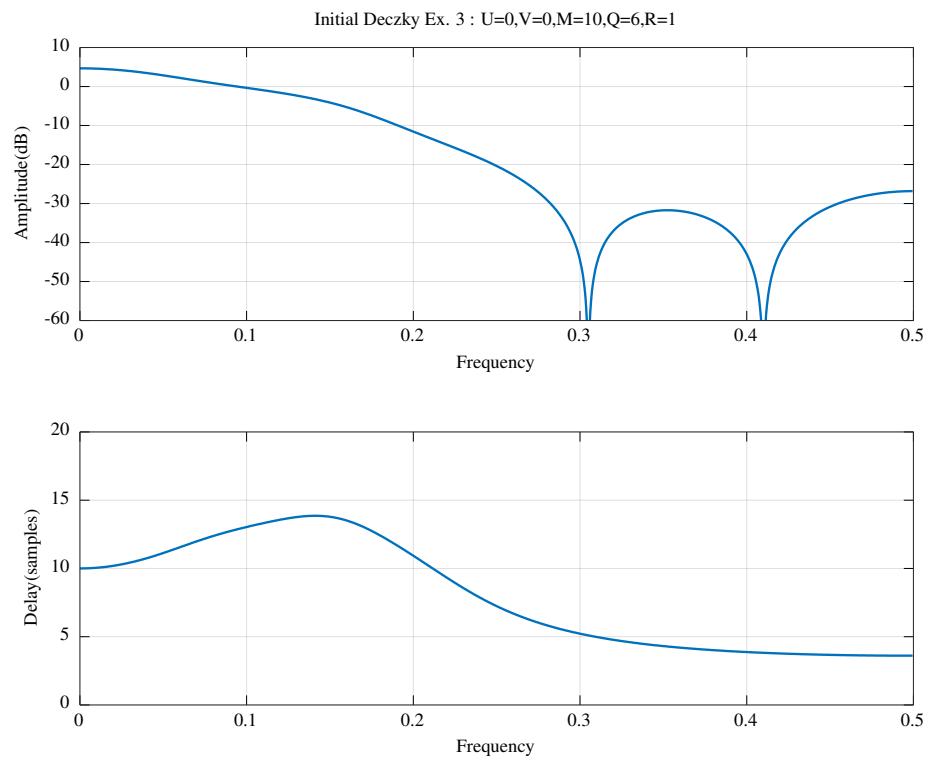


Figure 10.7: Deczky Example 3 : Response for initial coefficients

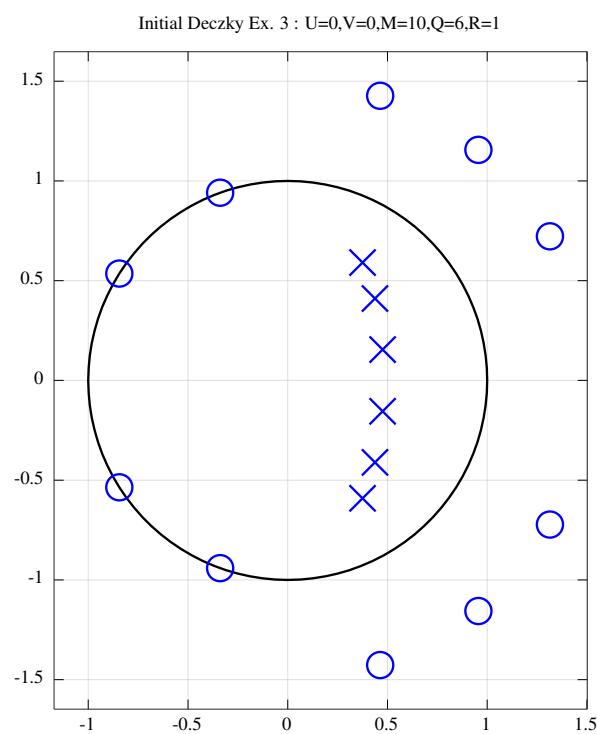


Figure 10.8: Deczky Example 3 : Pole-zero plot for initial coefficients

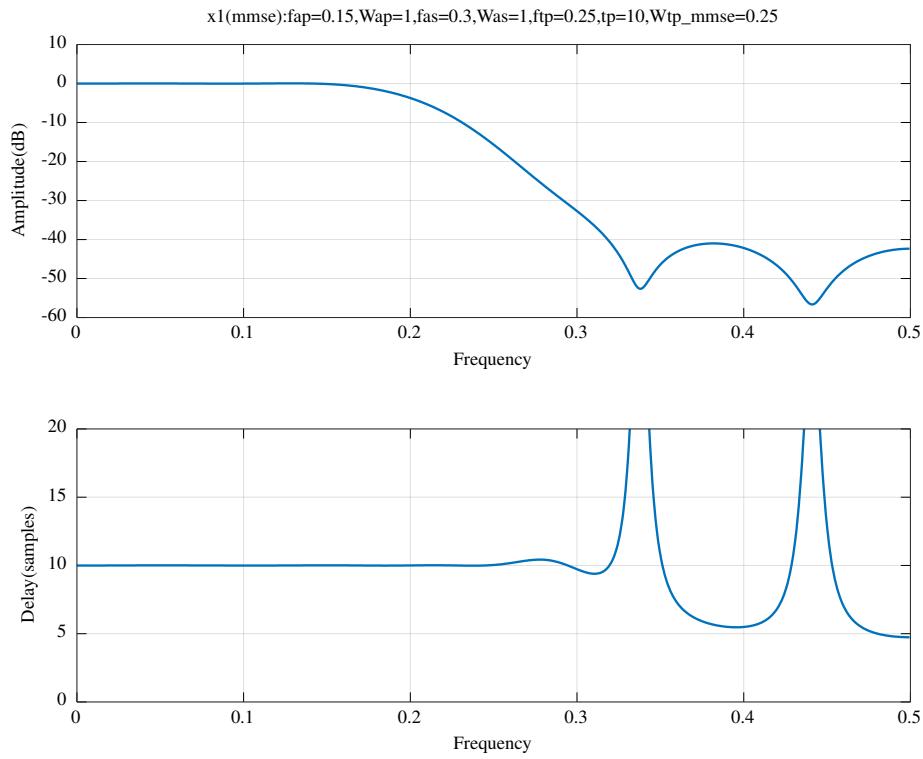


Figure 10.9: Deczky Example 3 MMSE pass 1 : Optimised response with Wap=1, Wtp=0.25, Was=1

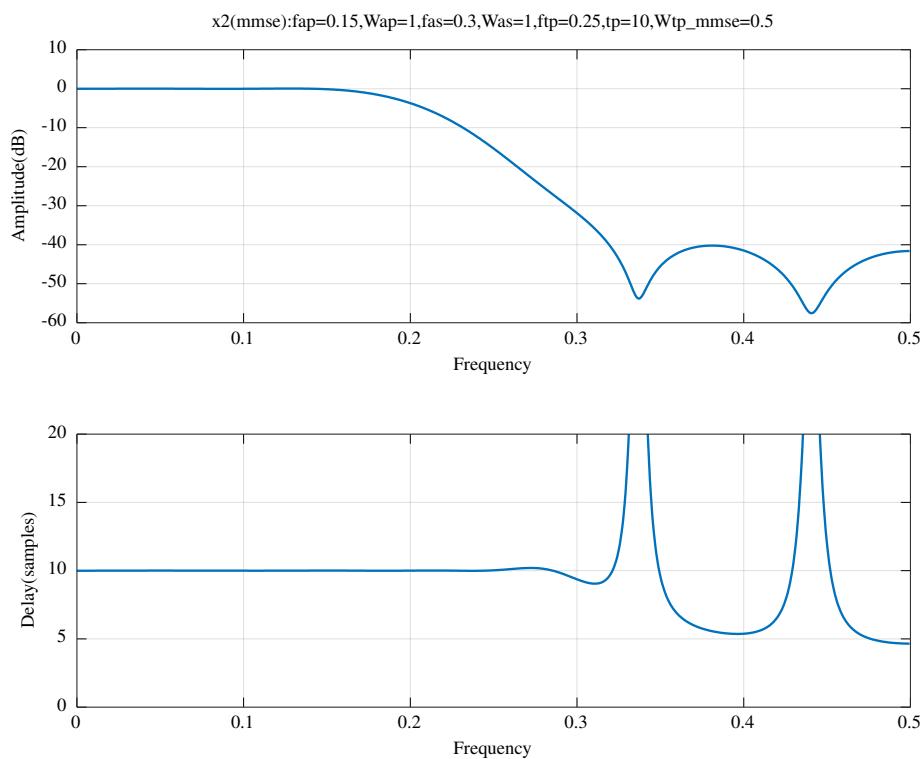


Figure 10.10: Deczky Example 3 MMSE pass 2: Optimised response with Wap=1, Wtp=0.5, Was=1

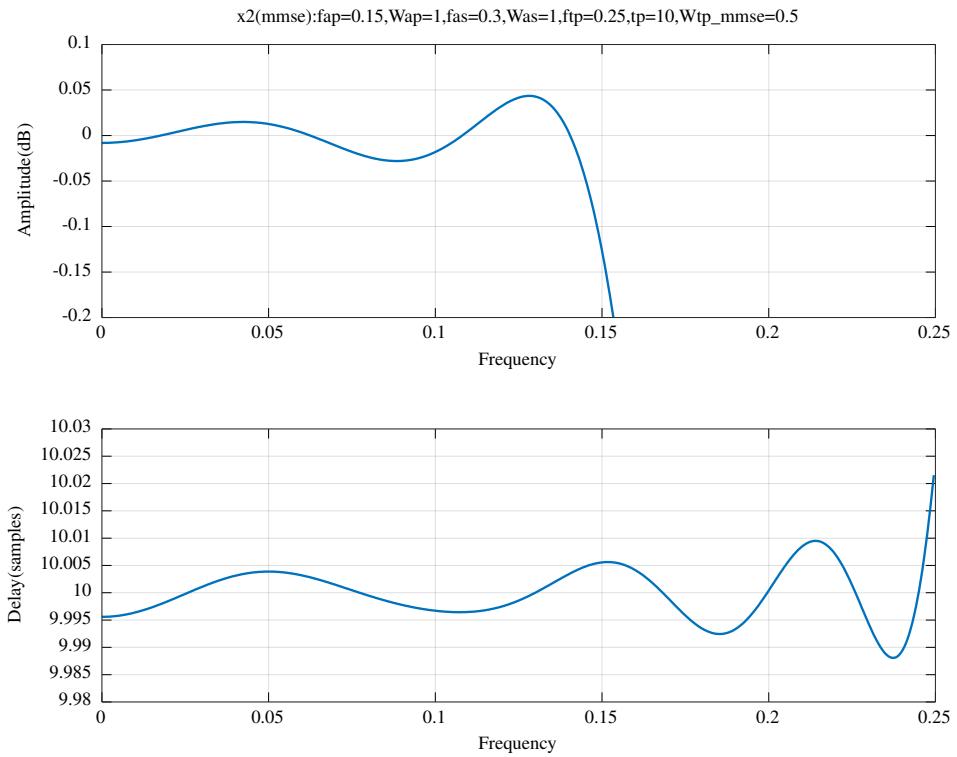


Figure 10.11: Deczky Example 3 MMSE pass 2: Optimised passband response with Wap=1, Wtp=0.5, Was=1

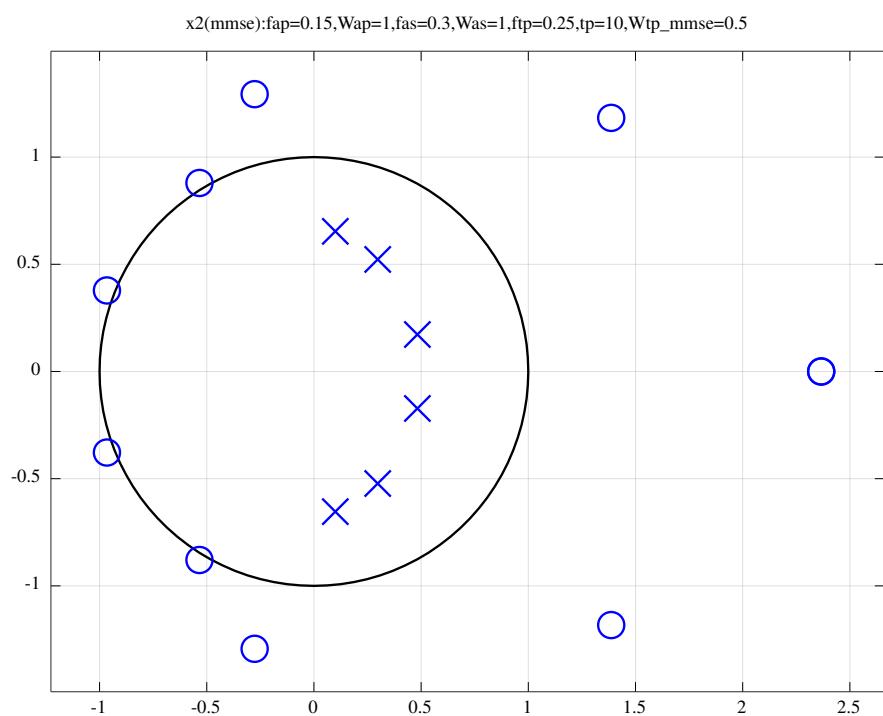


Figure 10.12: Deczky Example 3 MMSE pass 2: Pole-zero plot after optimisation with Wap=1, Wtp=0.5, Was=1

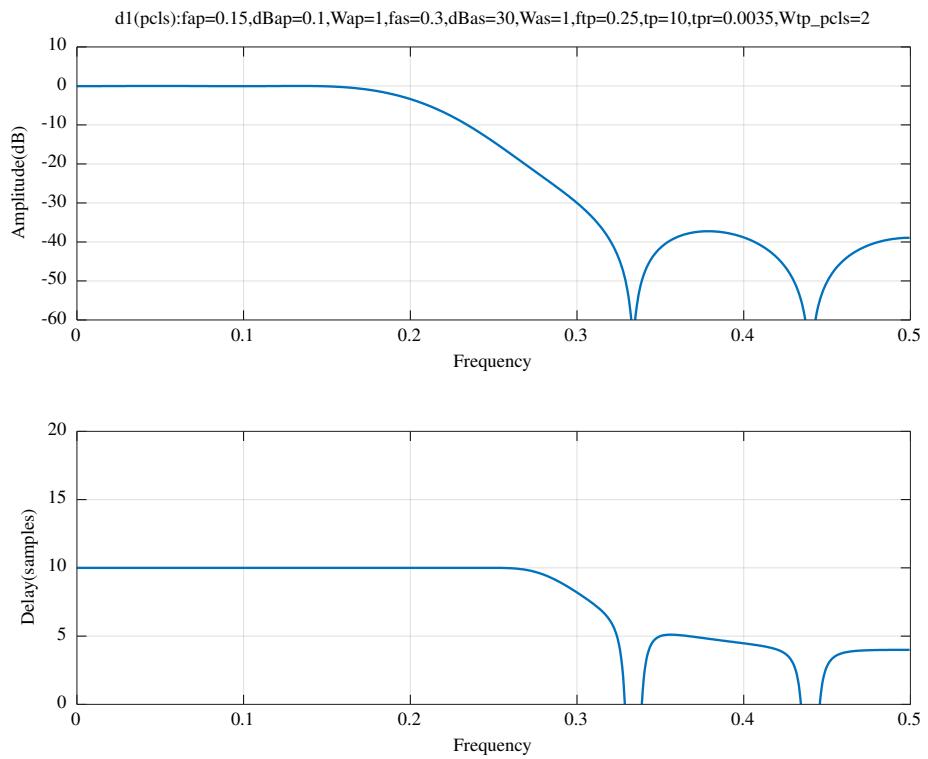


Figure 10.13: Deczky Example 3 PCLS : Optimised response with dBap=0.1, tp=10, tpr=0.0035, dBas=30

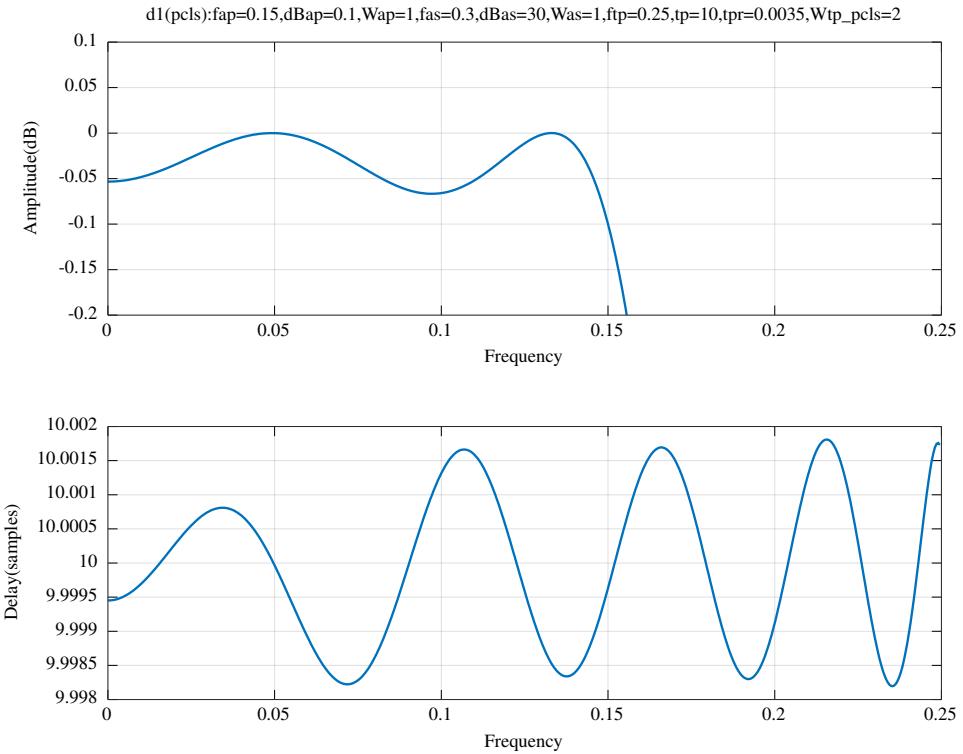


Figure 10.14: Deczky Example 3 PCLS : Optimised passband response with dBap=0.1, tp=10, tpr=0.0035, dBas=30

```
Ud1=0,Vd1=0,Md1=10,Qd1=6,Rd1=1
d1 = [ 0.0032730394, ...
        0.9932847469,    0.9945326862,    1.7705497191,    1.3603744029, ...
        2.2548365732, ...
        2.0965810695,    2.7643271361,    0.7257330778,    1.7449853615, ...]
```

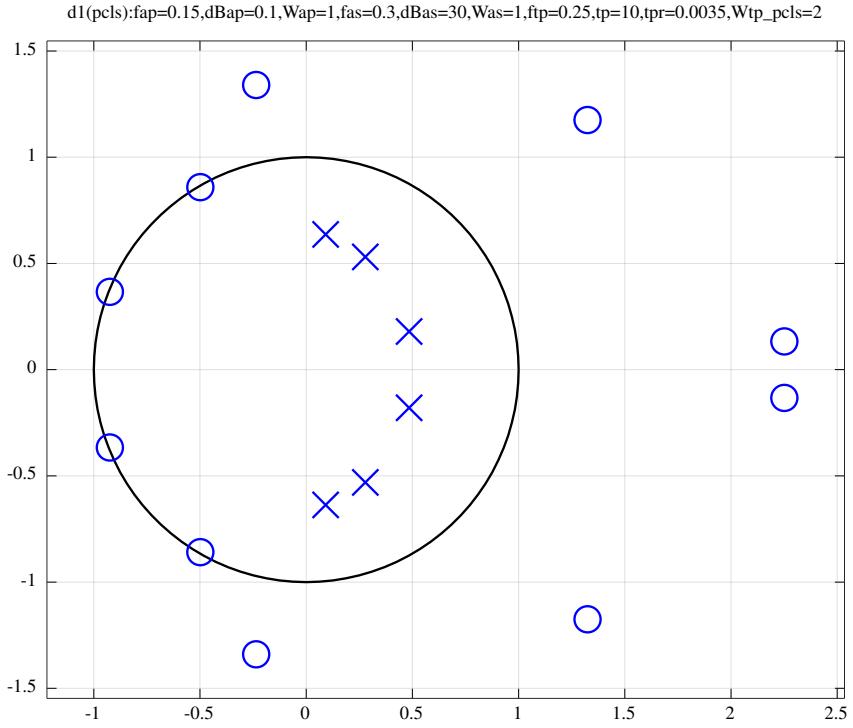


Figure 10.15: Deczky Example 3 PCLS : Pole-zero plot after optimisation with $\text{dBap}=0.1, \text{tp}=10, \text{tpr}=0.0035, \text{dBas}=30$

```
0.0589807149, ...
0.6432049445, 0.5993725631, 0.5165959328, ...
1.4279830777, 1.0882394044, 0.3551805244 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function $x2tf$) are, respectively:

```
N1 = [ 0.0032730394, -0.0125451026, 0.0112318460, -0.0033076342, ...
0.0144472234, -0.0093394545, -0.0348174946, 0.0005760261, ...
0.0967827377, 0.1323235521, 0.0942106910 ]';
```

and

```
D1 = [ 1.0000000000, -1.7080688389, 1.8579064523, -1.3406590224, ...
0.6687383057, -0.2229443678, 0.0396638102 ]';
```

An alternative implementation of Deczky's Example 3

The test script *deczky3a_sqp_test.m* is an alternative implementation of Deczky's Example 3 with $fap = 0.15, dBap = 0.2dB, Wap = 1, fas = 0.30, dBas = 51dB, Was = 10, ftp = 0.25, Wtp = 0.001, tp = 9.35$ samples group delay and $tpr = 0.3$ samples of peak-to-peak group delay ripple. The resulting amplitude and delay responses are shown in Figure 10.16 with pass-band detail shown in Figure 10.17. The corresponding pole-zero plot is shown in Figure 10.18. The PCLS optimised filter vector is, in gain-zero-pole form:

```
Ud1=0,Vd1=0,Md1=10,Qd1=6,Rd1=1
d1 = [ 0.0004163838, ...
1.1022479080, 1.0859218557, 1.0255043693, 4.5328682665, ...
1.9656503553, ...
2.7927633760, 2.1838682894, 1.9122550316, -0.0000009447, ...
-0.5153028320, ...
0.6819047747, 0.7449385654, 0.5826591428, ...
1.0587769010, 1.5272046939, -0.3763138297 ]';
```

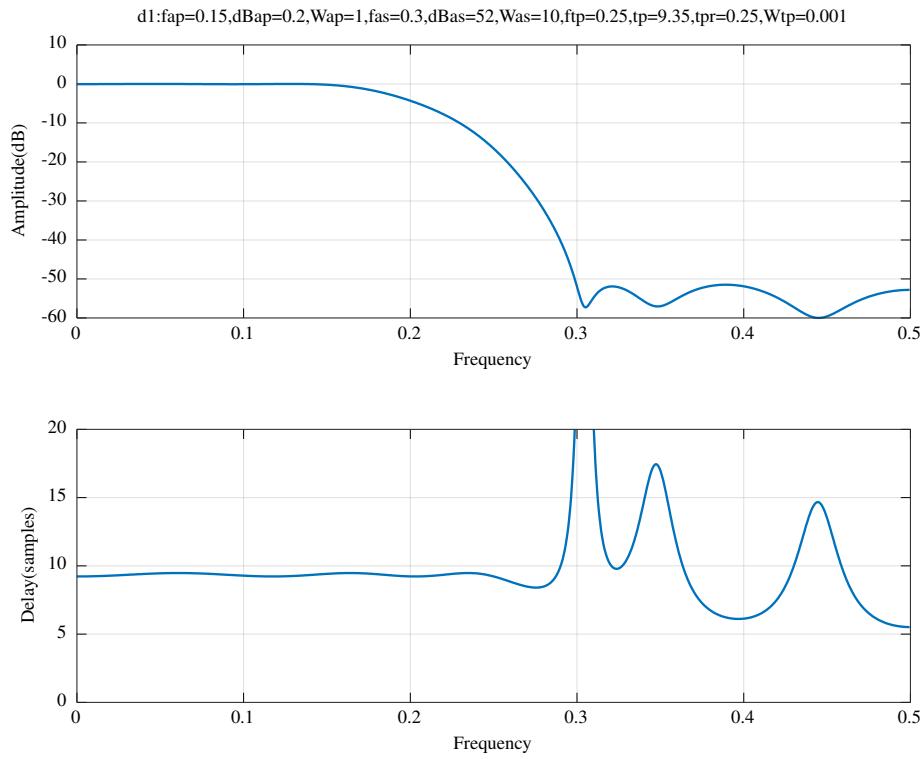


Figure 10.16: Deczky Example 3a PCLS : Optimised response with dBap=0.2, dBas=51, Was=10, tp=9.35, tpr=0.3

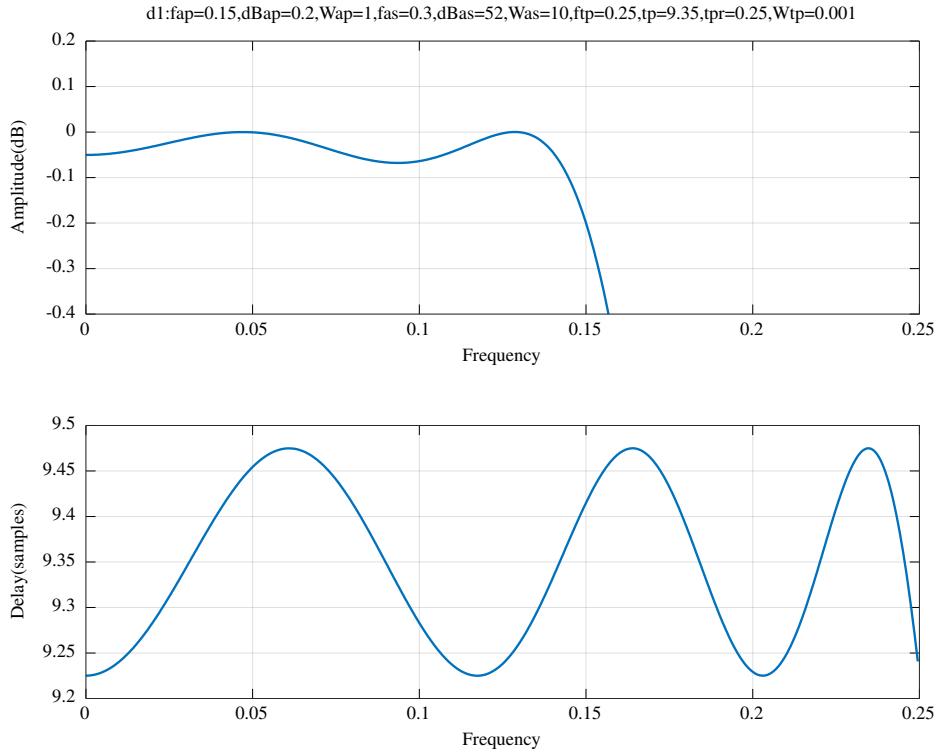


Figure 10.17: Deczky Example 3a PCLS : Optimised passband response with dBap=0.2, dBas=51, Was=10, tp=9.35, tpr=0.3

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [ 0.0004163838, -0.0035302517, 0.0057004664, 0.0099387279, ... ]
```

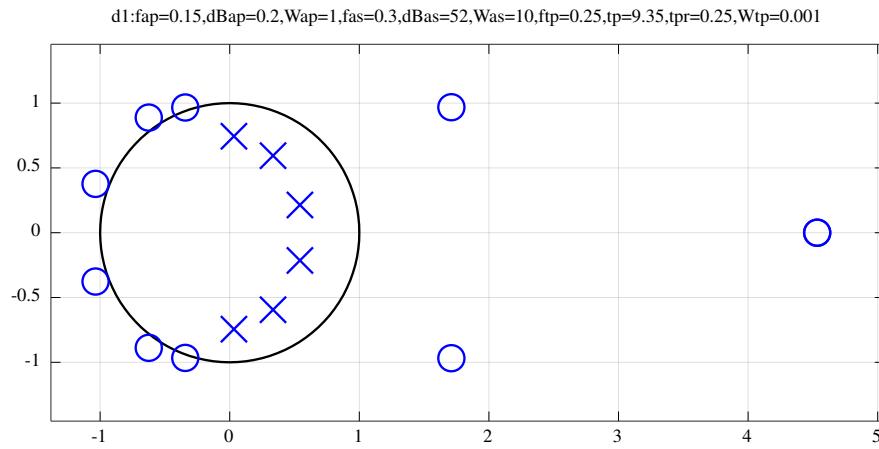


Figure 10.18: Deczky Example 3a PCLS : Pole-zero plot after optimisation with dBap=0.2, dBas=51, Was=10, tp=9.35, tpr=0.3

```
-0.0034393392, -0.0278623137, -0.0114295050, 0.0586187165, ...
0.1194646770, 0.1041648692, 0.0498061889 ]';
```

and

```
D1 = [ 1.0000000000, -1.8168845153, 2.1973269296, -1.8022609159, ...
1.0536058296, -0.4157901915, 0.0876027038 ]';
```

10.2.4 Deczky's Example 1

Antoniou and Lu [2, Example 16.3] refer to the following example IIR filter design specification as *Deczky's Example 1* [3]: the pass-band is [0.0, 0.25] with 0.02 maximum amplitude ripple, the pass-band ripple is 9 samples with 1 sample maximum ripple and the stop band is [0.3, 0.5) with at least 34 dB attenuation. The Octave script *deczky1_sqp_test.m* implements this example. The filter specification defined in that file is

```
n=200 % Frequency points across the band
tol=0.0002 % Tolerance on relative coefficient update size
fap=0.25 % Pass band amplitude response edge
dBap=1 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
Wat=0.02 % Transition band weight
ftp=0.25 % Pass band group delay response edge
tp=9 % Nominal filter group delay
tpr=1 % Pass band group delay peak-to-peak ripple
Wtp=0.02 % Pass band group delay weight
fas=0.3 % Stop band amplitude response edge
dBas=41 % Stop band minimum attenuation
Was=50 % Stop band amplitude weight
U=2 % Number of real zeros
V=0 % Number of real poles
M=10 % Number of complex zeros
Q=6 % Number of complex poles
R=1 % Denominator polynomial decimation factor
```

The initial filter design was found with the Octave script *tarczynski_deczky1_test.m*: The initial frequency response is shown in Figure 10.19 and the corresponding pole-zero plot is shown in Figure 10.20.

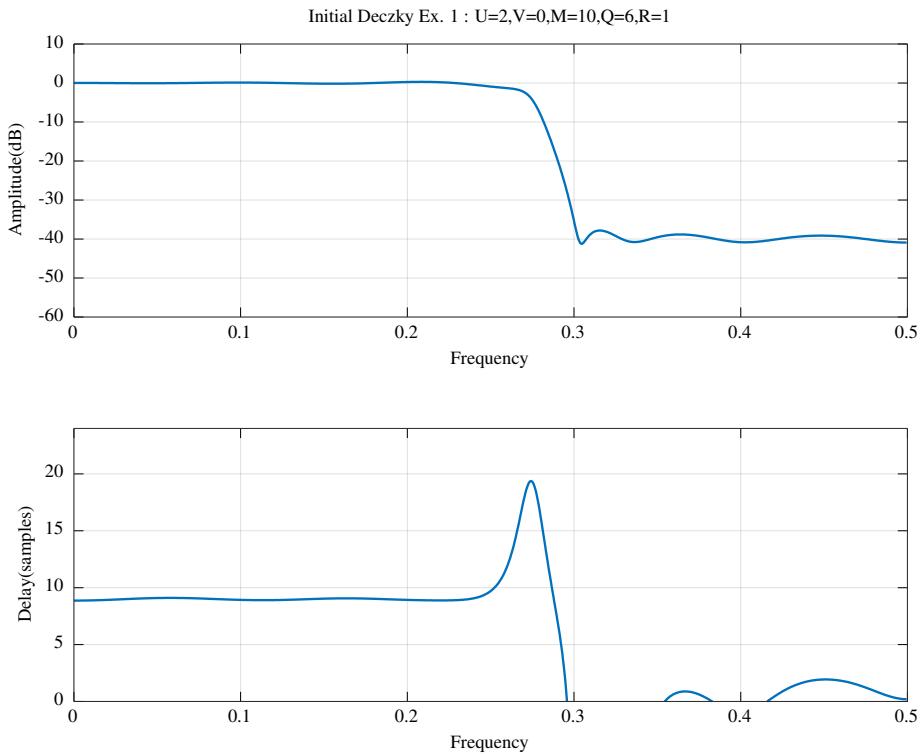


Figure 10.19: Deczky Example 1 : Response for initial coefficients

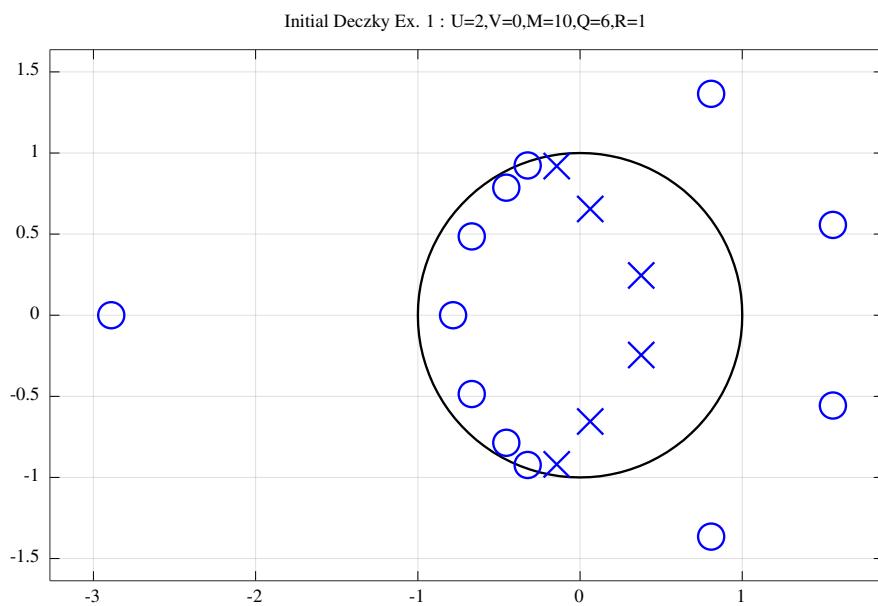


Figure 10.20: Deczky Example 1 : Pole-zero plot for initial coefficients

MMSE optimisation of Deczky's Example 1

The frequency response after MMSE optimisation is shown in Figure 10.21 and the pole-zero plot is shown in Figure 10.22.

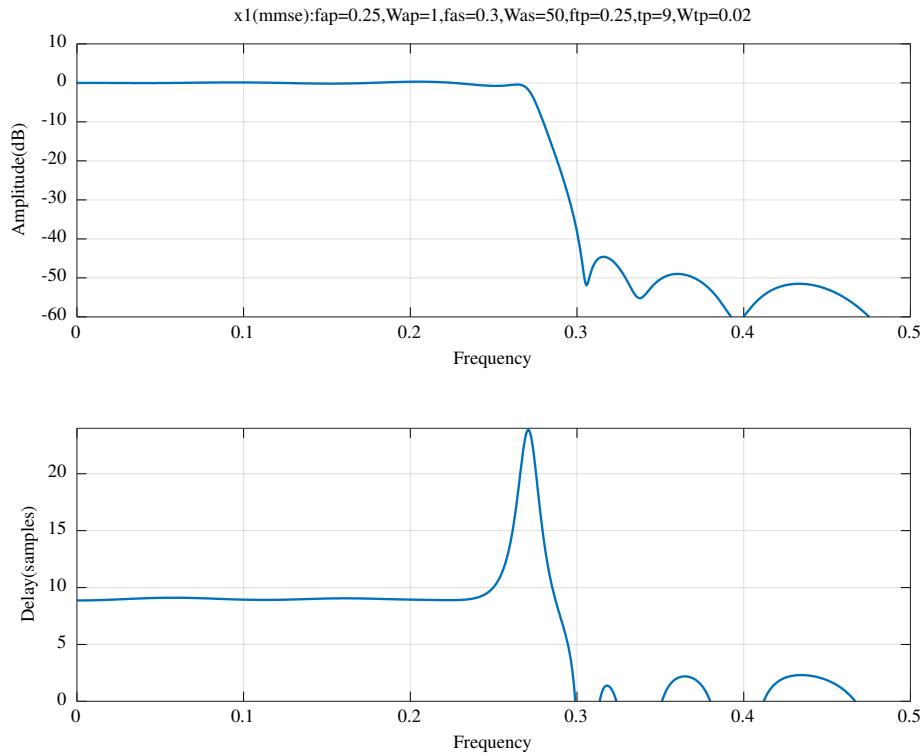


Figure 10.21: Deczky Example 1 MMSE : Optimised response with $W_{ap}=1$, $W_{tp}=0.02$, $W_{as}=3$

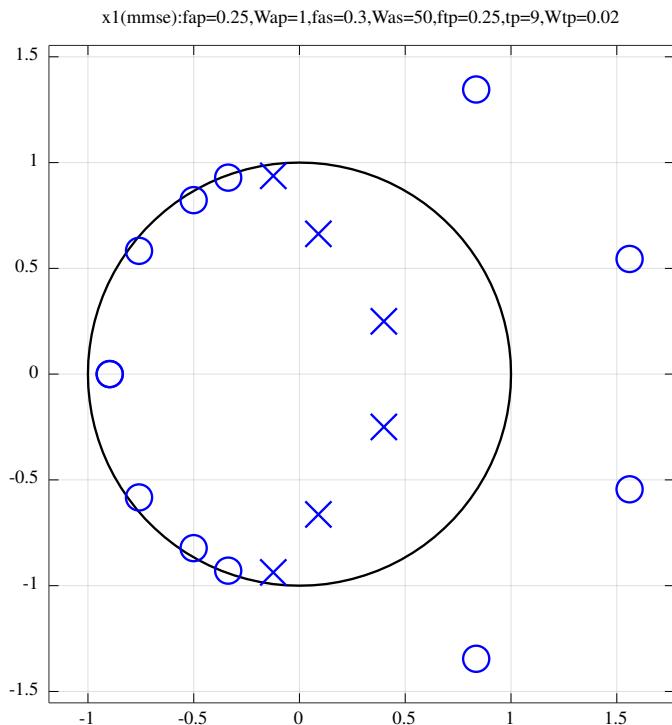


Figure 10.22: Deczky Example 1 MMSE : Pole-zero plot after optimisation with $W_{ap}=1$, $W_{tp}=0.02$, $W_{as}=3$

PCLS optimisation of Deczky's Example 1

The frequency response after PCLS optimisation is shown in Figure 10.23, the pass-band detail is shown in Figure 10.24 and the pole-zero plot is shown in Figure 10.25. The last estimate of the optimised filter vector is, in the gain, zeros and poles form of

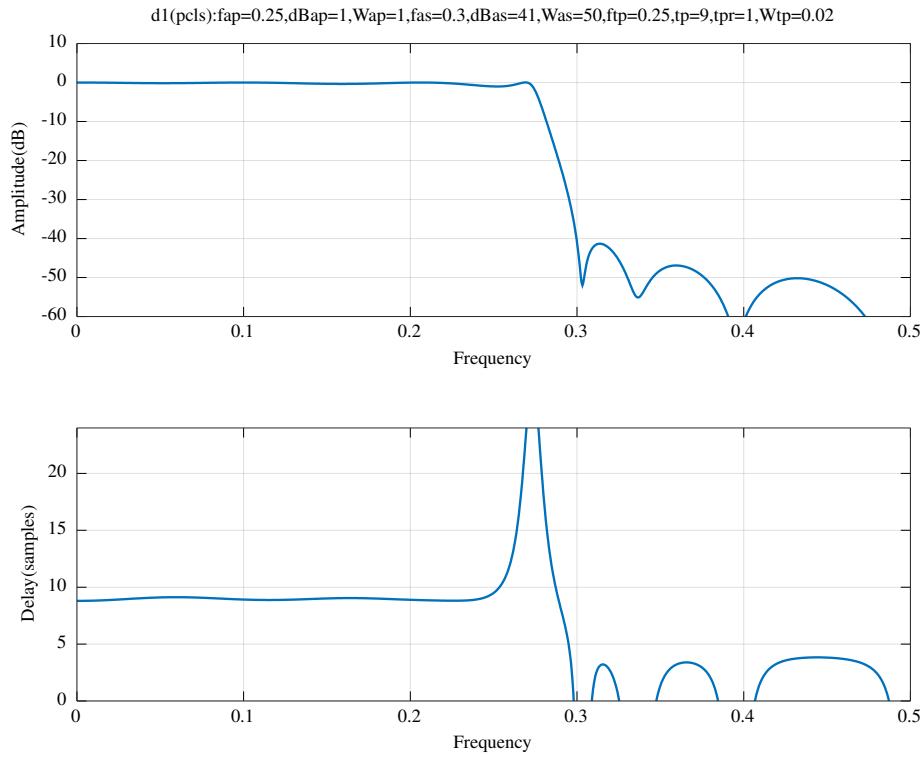


Figure 10.23: Deczky Example 1 PCLS : Optimised response with dBap=0.7, tp=10, tpr=0.6, dBas=36

Equation 10.2:

```
Ud1=2,Vd1=0,Md1=10,Qd1=6,Rd1=1
d1 = [ 0.0112122730, ...
        -0.9861004368, -0.9861004386, ...
        1.6110573997, 1.5722087253, 0.9922923829, 0.9715772075, ...
        0.9771125907, ...
        0.3501543945, 1.0364670361, 1.9044365672, 2.1121907965, ...
        2.4859615633, ...
        0.9551538736, 0.6593886914, 0.4529386321, ...
        1.7176976492, 1.4735817876, 0.5786060583 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [ 0.0112122730, 0.0061061917, -0.0072657516, 0.0004160889, ...
        0.0176332381, 0.0007219057, -0.0411494068, 0.0175031604, ...
        0.2108791812, 0.3942108798, 0.3864432741, 0.2208302697, ...
        0.0620728331 ]';
```

and

```
D1 = [ 1.0000000000, -0.6068077653, 1.4014847233, -0.9586349086, ...
        0.6620538480, -0.2998600159, 0.0813782601 ]';
```

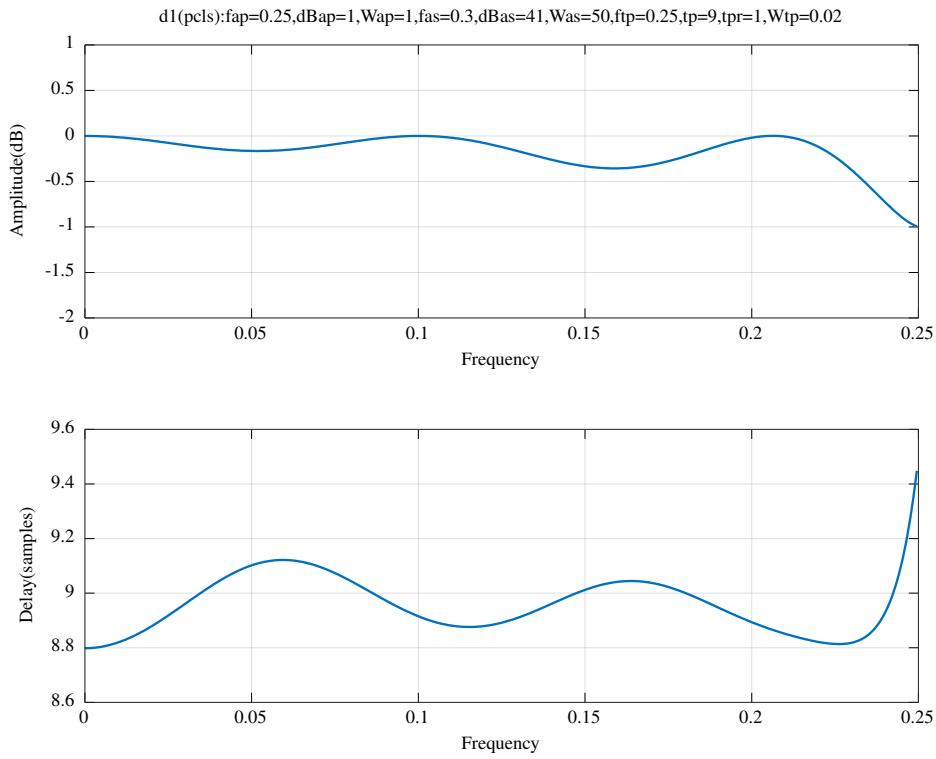


Figure 10.24: Deczky Example 1 PCLS : Optimised passband response with dBap=0.7, tp=10, tpr=0.6, dBas=36

d1(pcls):fap=0.25,dBap=1,Wap=1,fas=0.3,dBas=41,Was=50,ftp=0.25,tp=9,tpr=1,Wtp=0.02

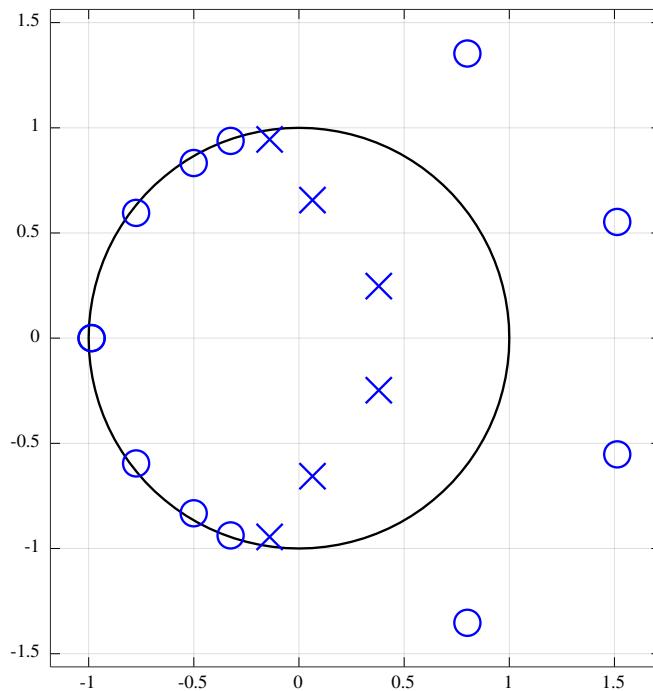


Figure 10.25: Deczky Example 1 PCLS : Pole-zero plot after optimisation with dBap=0.7, tp=10, tpr=0.6, dBas=36

10.2.5 Low-pass R=2 decimation filter

The second example design is a low-pass $R = 2$ decimation filter.

$$U = 2$$

$$\begin{aligned}
V &= 0 \\
M &= 10 \\
Q &= 6 \\
R &= 2 \\
A(f) &= \begin{cases} 1 & 0 < f < 0.10 \\ 0 & 0.25 < f < 0.5 \end{cases} \\
T(f) &= 12 \quad 0 < f < 0.125
\end{aligned}$$

The Octave script *decimator_R2_test.m* implements this example. The filter specification defined in that file is

```

n=1000 % Frequency points across the band
tol_wise=1e-07 % Tolerance on WISE relative coef. update
tol_mmse=1e-05 % Tolerance on MMSE relative coef. update
tol_pcls=0.001 % Tolerance on PCLS relative coef. update
fap=0.1 % Pass band amplitude response edge
dBap=0.2 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.125 % Pass band group delay response edge
tp=12 % Nominal filter group delay
tpr=0.06 % Pass band group delay peak-to-peak ripple
Wtp=1 % Pass band group delay weight
fas=0.25 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=2 % Stop band amplitude weight
U=2 % Number of real zeros
V=0 % Number of real poles
M=10 % Number of complex zeros
Q=6 % Number of complex poles
R=2 % Denominator polynomial decimation factor

```

The initial filter is found using the *WISE* barrier function implemented in the Octave function *xInitHd* function. That filter design is itself initialised with a “guess”:

```

xi=[ 0.0001, ...
[2,2], ...
[1,1,1,1,1], ...
(6:10)*pi/10, ...
0.7*[1,1,1], ...
(1:3)*pi/8]';

```

In the above listing, the first line shows the gain, and subsequent lines show, respectively, 2 real zeros, 0 real poles, the zero radiiuses for 5 conjugate pairs of zeros, the zero angles for 5 conjugate pairs of zeros, the pole radiiuses for 3 conjugate pairs of poles and the pole angles for 3 conjugate pairs of poles. In this case the denominator decimation factor is $R = 2$ and each complex pole pair is split into 2 complex pole pairs so that the overall filter has 6 complex conjugate pole pairs. The resulting initial response is shown in Figure 10.26.

MMSE optimisation of the low-pass R=2 decimator

After some experimentation and iteration, weights $Wap = 1$, $Was = 2$ and $Wtp = 1$ give the response shown in Figure 10.27 with the pass-band detail shown in Figure 10.28. The corresponding pole-zero plot is shown in Figure 10.29.

PCLS optimisation of the low-pass R=2 decimator

Starting with the MMSE filter of Figure 10.27, the weights $Wap = 1$, $Was = 2$ and $Wtp = 1$ and the constraints $dBap = 0.2$, $dBas = 40$ and $tpr = 0.06$ give the response shown in Figure 10.30 with pass-band detail shown in Figure 10.31. The corresponding pole-zero plot is shown in Figure 10.32. The estimate of the optimised filter vector is, in the gain, zeros and poles form of Equation 10.2:

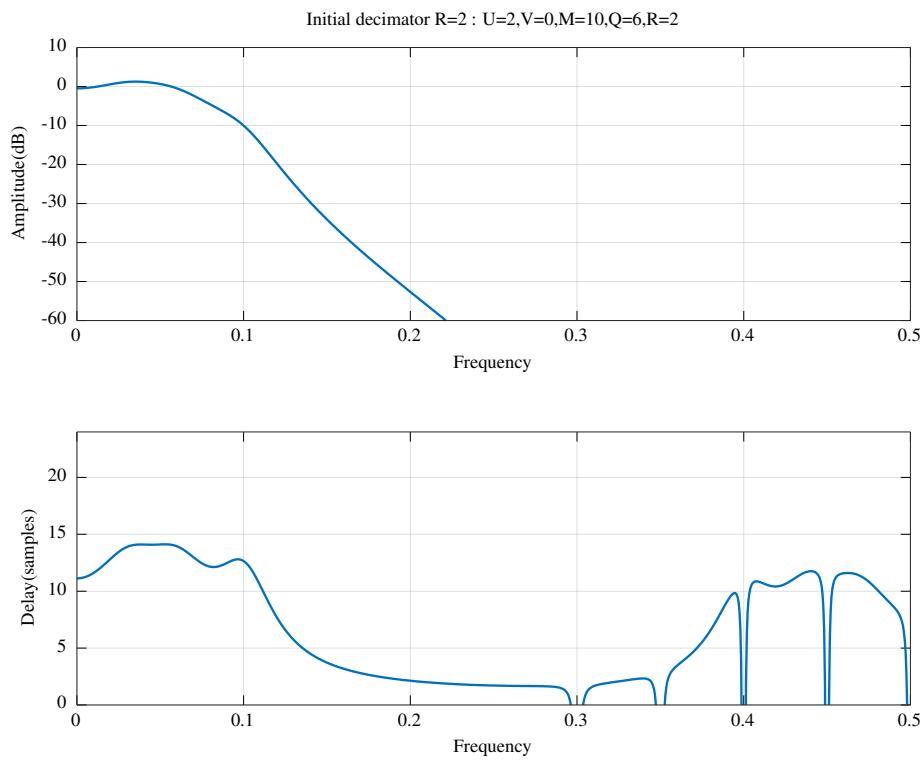


Figure 10.26: Low-pass decimator R=2 : Response for initial coefficients

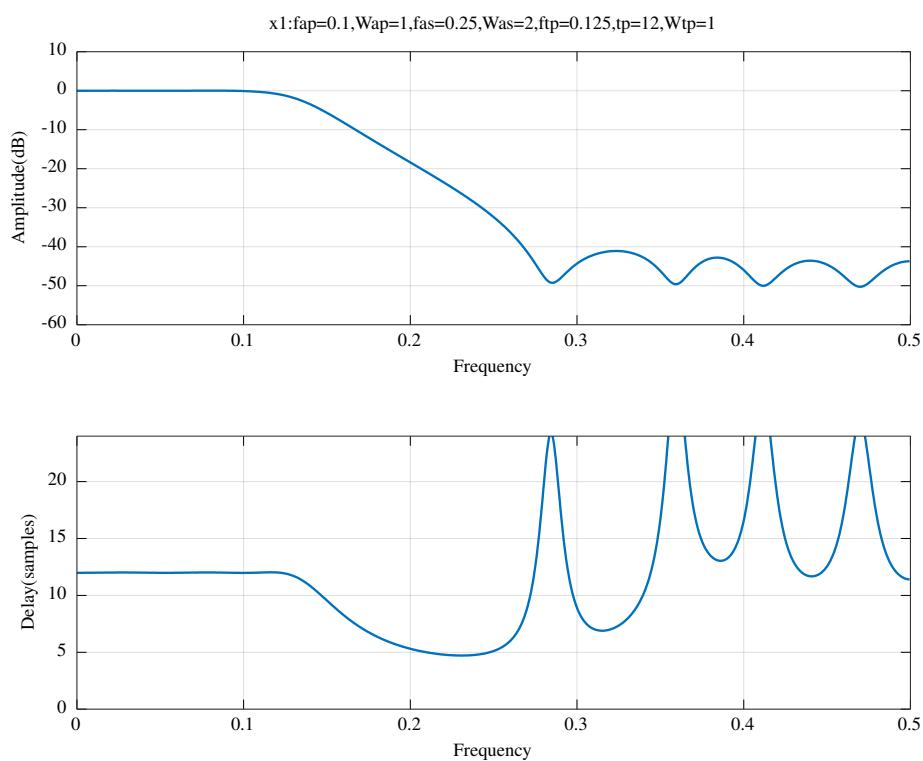


Figure 10.27: Low-pass decimator R=2 MMSE : Optimised response with Wap=1, Was=2, Wtp=1

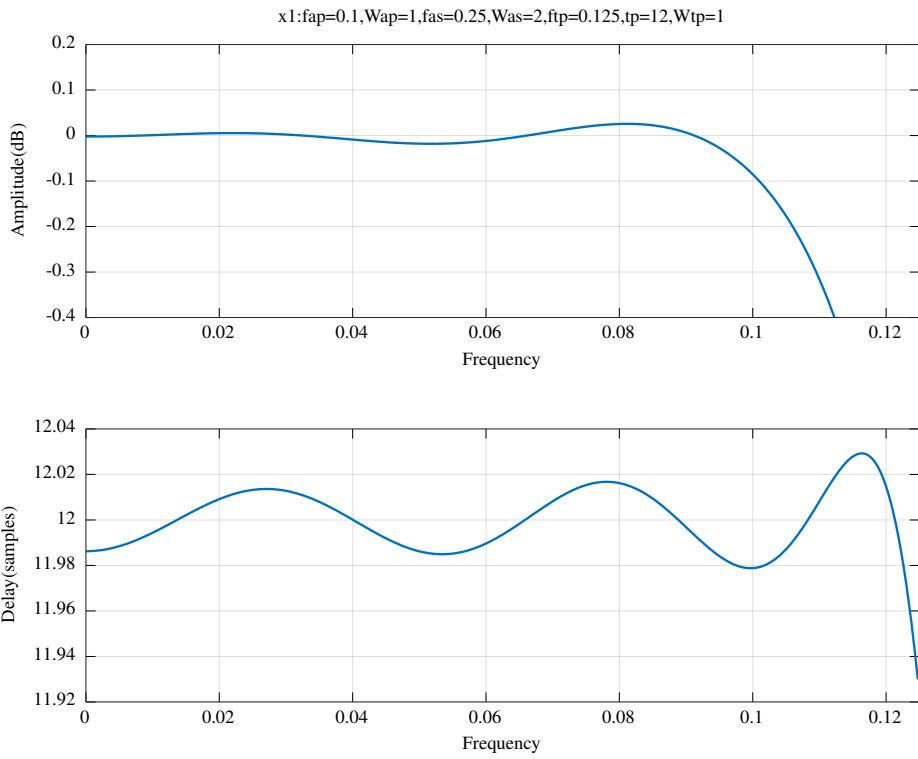


Figure 10.28: Low-pass decimator R=2 MMSE : Optimised passband response with Wap=1, Was=2, Wtp=1

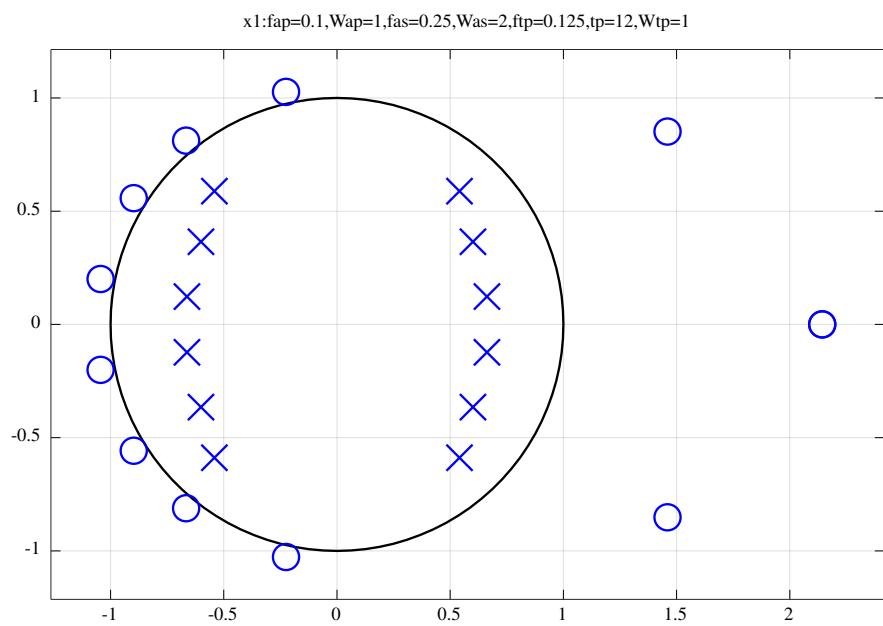


Figure 10.29: Low-pass decimator R=2 MMSE : Pole-zero plot after optimisation with Wap=1, Was=2, Wtp=1

```

Ud1=2,Vd1=0,Md1=10,Qd1=6,Rd1=2
d1 = [ 0.0007506908, ...
        3.2229433282, 3.2229433703, ...
        1.6609255336, 1.1027187059, 1.0696174079, 1.1454286988, ...
        1.1669818656, ...
        0.3785854665, 2.2357990079, 1.6704437538, 2.5726772013, ...
        3.3357771282, ...
        0.4803466628, 0.5326727129, 0.6515541977, ...
        0.3503189764, 1.1194133879, 1.6610716737 ]';

```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```

N1 = [ 0.0007506908, -0.0028066024, -0.0043561761, 0.0114126554, ...
        0.0268293835, 0.0024638252, -0.0498340859, -0.0640032070, ...
        0.0013891976, 0.1128716622, 0.1811914496, 0.1453552801, ...
        0.0534708989 ]';

```

and

```

D1 = [ 1.0000000000, 0.0000000000, -1.2495798645, 0.0000000000, ...
        1.1977280733, 0.0000000000, -0.8339015419, 0.0000000000, ...
        0.4192151506, 0.0000000000, -0.1465194627, 0.0000000000, ...
        0.0277927518 ]';

```

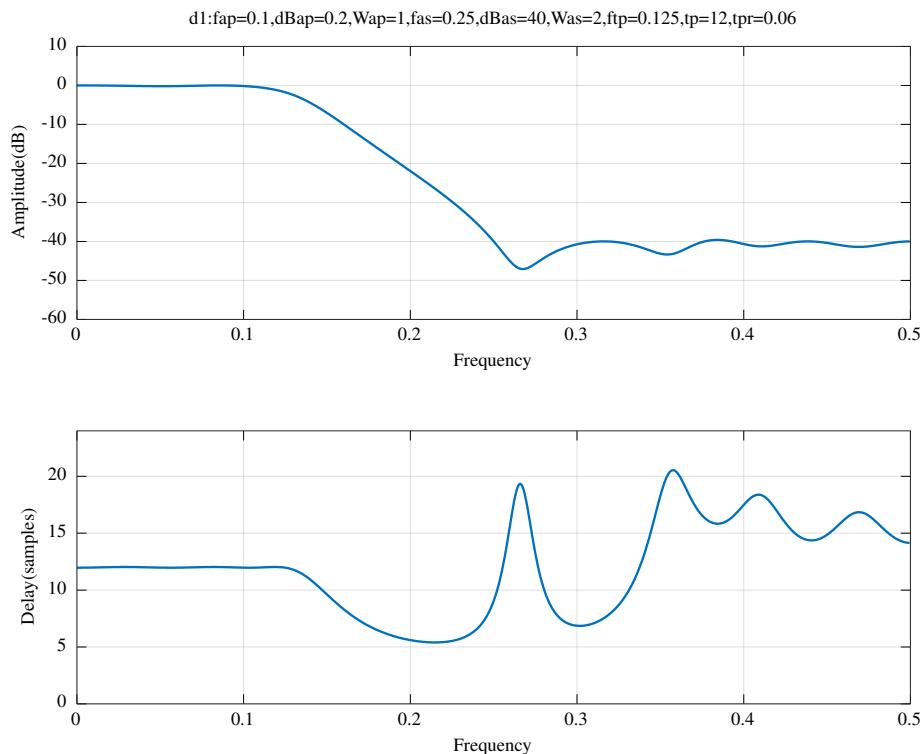


Figure 10.30: Low-pass decimator R=2 PCLS : Optimised response with Wap=1, Was=2, Wtp=1, dBap=0.2, dBas=40, tpr=0.06

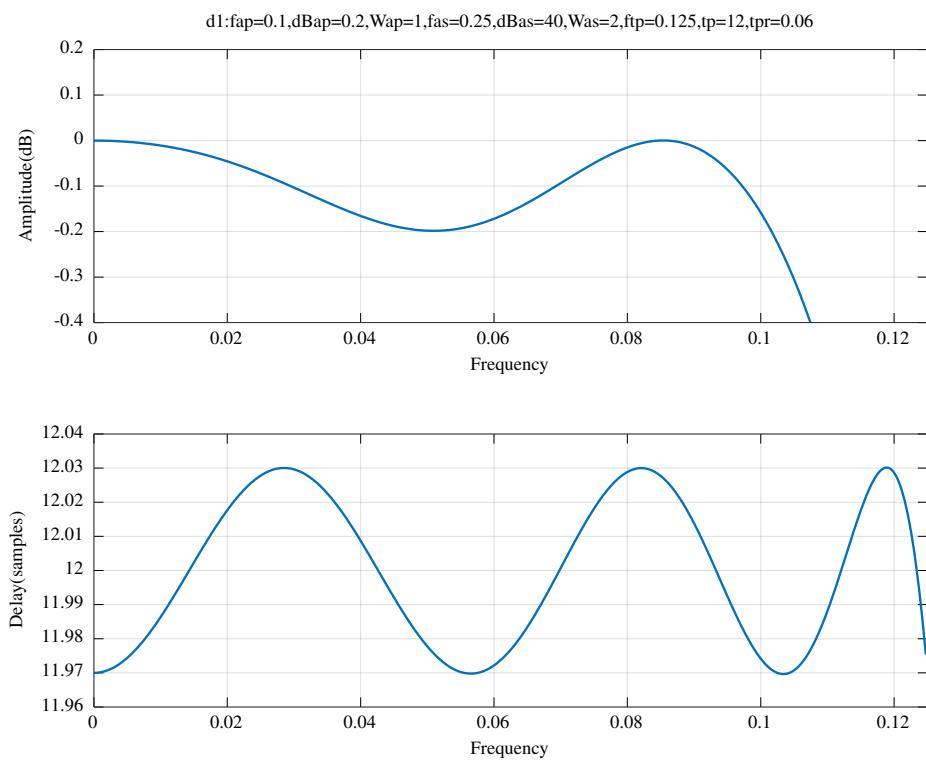


Figure 10.31: Low-pass decimator R=2 PCLS : Optimised passband response with Wap=1, Was=2, Wtp=1, dBap=0.2, dBas=40, tpr=0.06

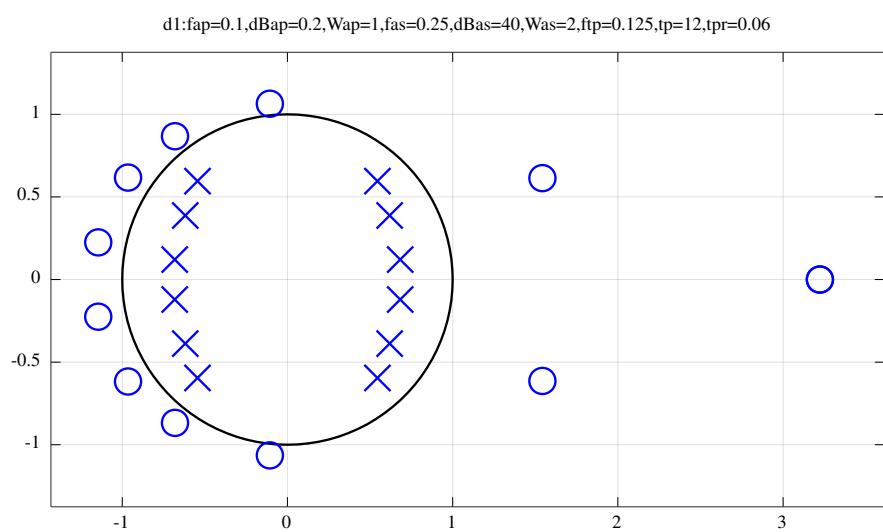


Figure 10.32: Low-pass decimator R=2 PCLS : Optimised pole-zero plot with Wap=1, Was=2, Wtp=1, dBap=0.2, dBas=40, tpr=0.06

10.2.6 Band-pass R=2 decimation filter

The fourth example design is an $R = 2$ decimation band-pass filter. The amplitude response pass-band edges are at frequencies $0.1f_s$ and $0.2f_s$. The amplitude response stop-band edges are at frequencies $0.05f_s$ and $0.25f_s$. The pass-band group-delay specification is 16 samples with band edges at frequencies $0.09f_s$ and $0.21f_s$. The Octave script *iir_sqp_slb_bandpass_test.m* implements this example. The *sqp* loop is implemented by the Octave function *iir_slb*. The initial filter was “guesstimated” and has a pole-zero specification of $U = 2$, $V = 0$, $M = 18$, $Q = 10$ and $R = 2$:

```
U=2,V=0,M=18,Q=10,R=2
x0=[ 0.00005, ...
1, -1, ...
0.9*ones(1,6), [1 1 1], ...
(11:16)*pi/20, (7:9)*pi/10, ...
0.81*ones(1,5), ...
(4:8)*pi/10 ]';
```

In the above listing, the first line shows the gain, and subsequent lines show, respectively, 2 real zeros, 0 real poles, the zero radiiuses for 9 conjugate pairs of zeros, the zero angles for 9 conjugate pairs of zeros, the pole radiiuses for 5 conjugate pairs of poles and the pole angles for 5 conjugate pairs of poles. In this case the denominator decimation factor is $R = 2$ and each complex pole pair is split into 2 complex pole pairs so that the overall filter has 10 complex conjugate pole pairs. The response of the initial filter is shown in Figure 10.33.

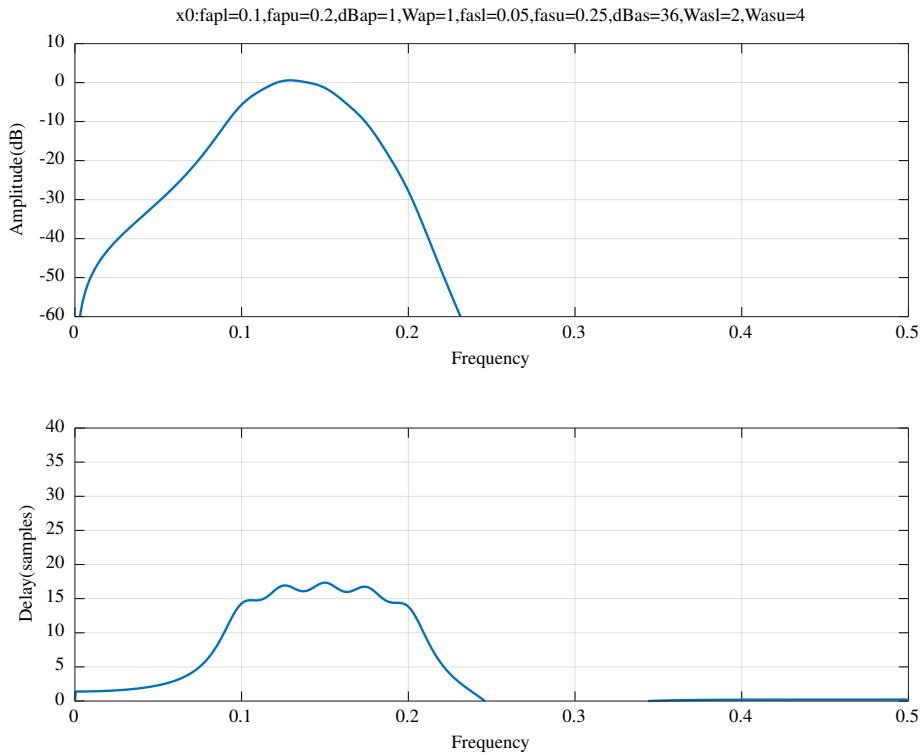


Figure 10.33: IIR band-pass R=2 decimation filter : response of the initial filter

MMSE optimisation of the band-pass R=2 decimator

First the initial filter is passed through *iir_sqp_mmse* with amplitude response pass-band weight $Wap = 1$, amplitude response lower and upper stop-band weights $Wasl = 2$ and $Wasu = 2$, respectively, and group-delay response pass-band weight $Wtp = 1$. The response of the MMSE-optimised filter is shown in Figure 10.34 with pass-band detail shown in Figure 10.35. The corresponding pole-zero plot is shown in Figure 10.36.

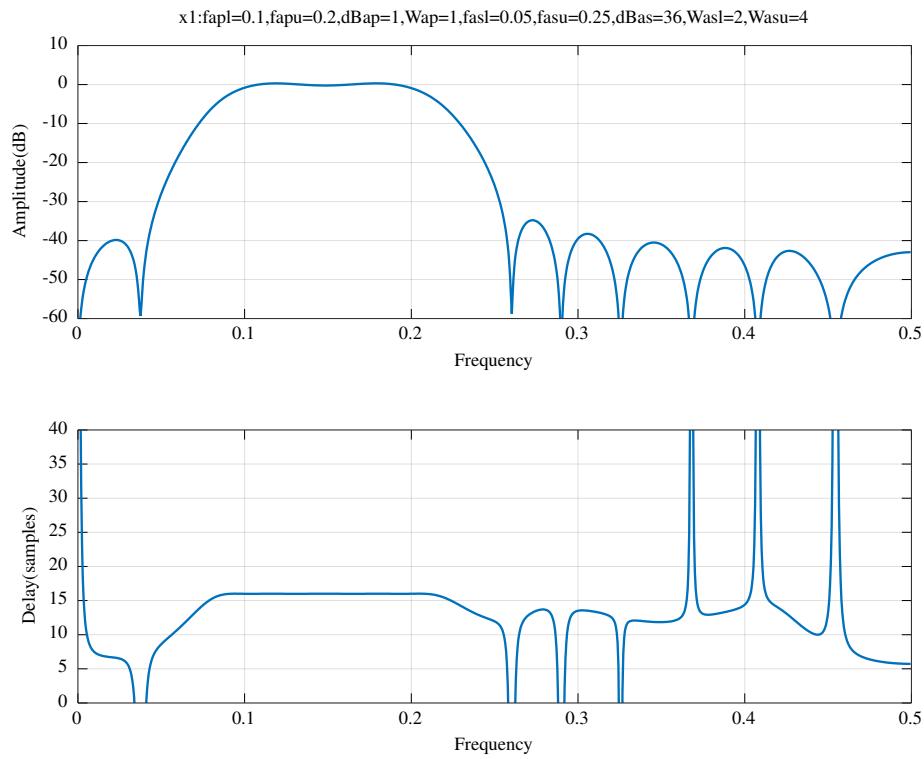


Figure 10.34: IIR band-pass R=2 decimation filter with delay tp=16 samples : response of the filter after MMSE optimisation

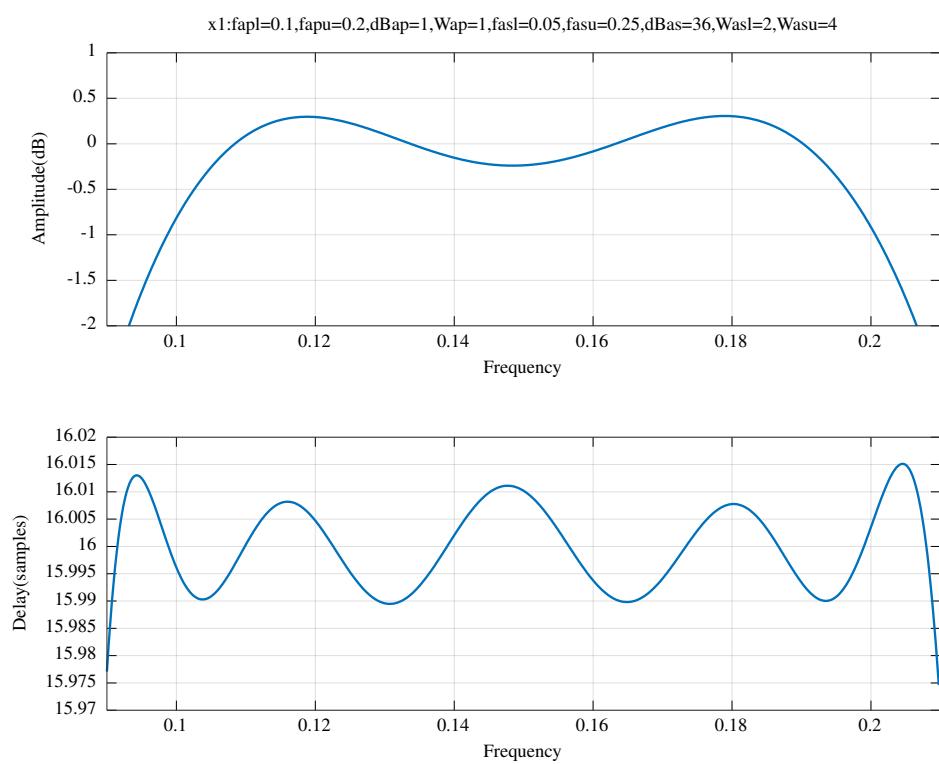


Figure 10.35: IIR band-pass R=2 decimation filter with delay tp=16 samples : passband response of the filter after MMSE optimisation

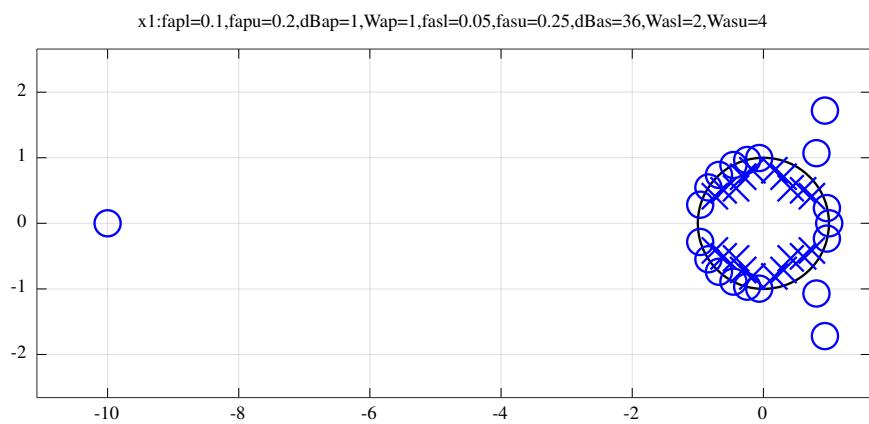


Figure 10.36: IIR band-pass R=2 decimation filter with delay tp=16 samples : pole-zero plot of the filter after MMSE optimisation

PCLS optimisation of the band-pass R=2 decimator

Next, the filter designed in Section 10.2.6 is passed through *iir_slb* for PCLS optimisation with $Wtp = 1$, $tpr = 0.08$ samples, $dBap = 1$ and $dBas = 36$. The response of the resulting filter is shown in Figure 10.37 with pass-band detail shown in Figure 10.38. The corresponding pole-zero plot is shown in Figure 10.39. The estimate of the optimised filter vector is, in the gain, zeros and poles form of Equation 10.2:

```
Ud1=2,Vd1=0,Md1=18,Qd1=10,Rd1=2
d1 = [ 0.0121797025, ...
        0.9737564861, -0.9407368121, ...
        1.4670056060, -0.9939270229,  0.9756166898,  0.9777463256, ...
        0.9805296265,  0.9795312589,  0.9955763363,  0.9797312081, ...
        1.3298964377, ...
        1.1054498390,   3.4230971323,   2.1541209613,   1.9135326087, ...
        1.7264436726,   2.4222348902,   4.6917485784,   2.6547478753, ...
        5.4146044096, ...
        0.7558653825,   0.6654897294,   0.5810684018,   0.6390408964, ...
        0.7256942915, ...
        1.0659341269,   1.4085319806,   1.9688769424,   2.4347284339, ...
        2.8280539303 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [ 0.0121797025,   0.0037181466,   0.0271208444,   0.0229180899, ...
        0.0527427390,   0.0310257396,   0.0310148929,   -0.0050816528, ...
        -0.0052190028,  -0.0414041061,  -0.0726994834,  -0.0996450244, ...
        -0.0564833866,   0.0506036935,   0.1386006506,   0.1492761564, ...
        0.0507378434,  -0.0443572576,  -0.1001730674,  -0.0681538294, ...
        -0.0335038915 ]';
```

and

```
D1 = [ 1.0000000000,   0.0000000000,   1.8567605231,   0.0000000000, ...
        2.1933886439,   0.0000000000,   2.2557980857,   0.0000000000, ...
        2.0335494061,   0.0000000000,   1.5306996820,   0.0000000000, ...
        0.9936532921,   0.0000000000,   0.5470027574,   0.0000000000, ...
        0.2511015052,   0.0000000000,   0.0839735161,   0.0000000000, ...
        0.0183734564 ]';
```

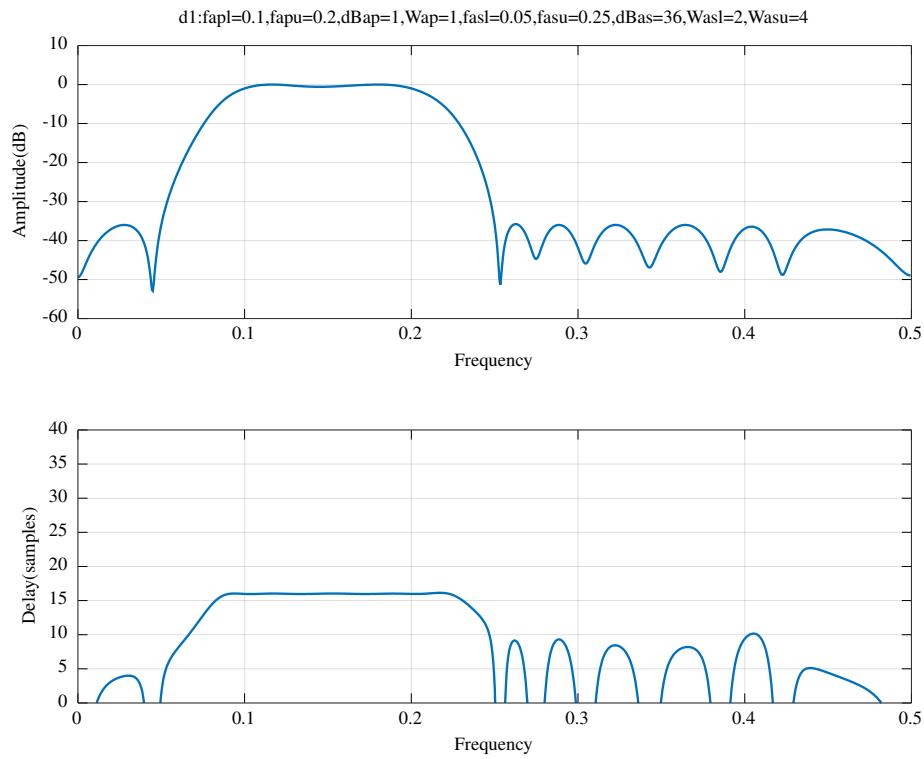


Figure 10.37: IIR band-pass R=2 decimation filter with delay tp=16 samples : response of the filter after PCLS optimisation

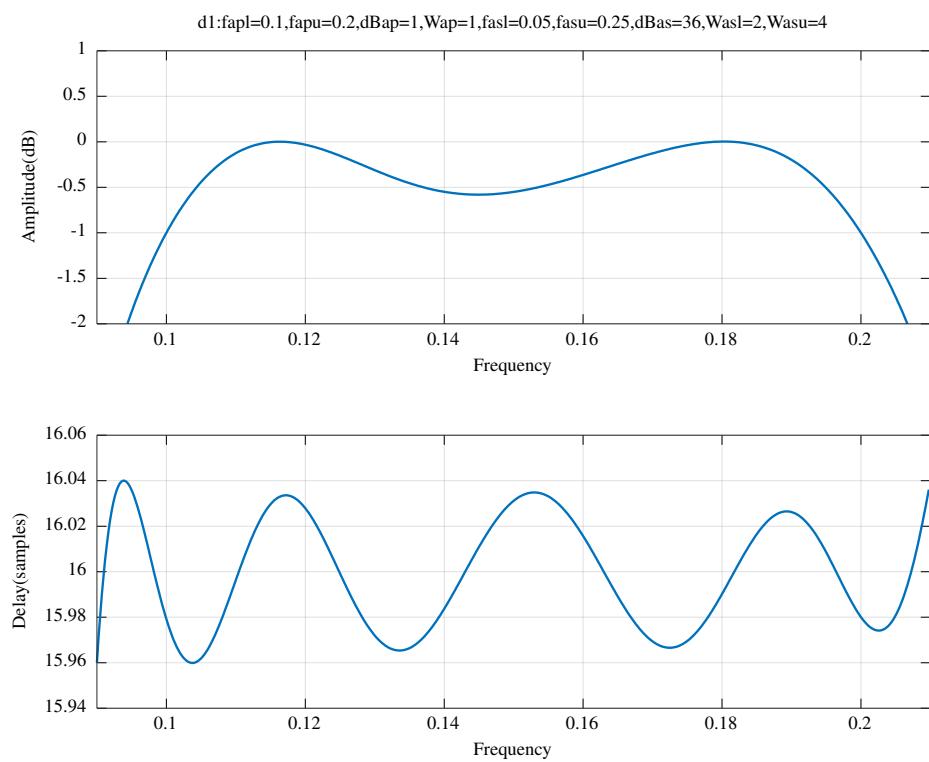


Figure 10.38: IIR band-pass R=2 decimation filter with delay tp=16 samples : passband response of the filter after PCLS optimisation

d1:fapl=0.1,fapu=0.2,dBap=1,Wap=1,fasl=0.05,fasu=0.25,dBas=36,Wasl=2,Wasu=4

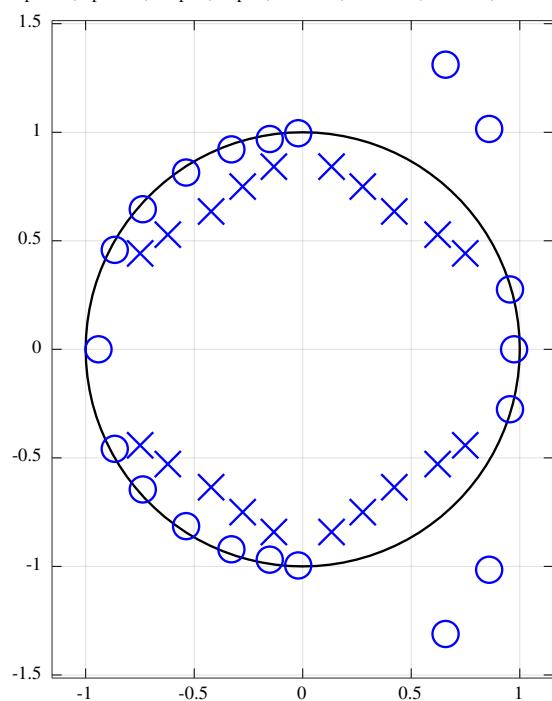


Figure 10.39: IIR band-pass R=2 decimation filter with delay tp=16 samples : pole-zero plot of the filter after PCLS optimisation

Comparison with FIR band-pass filter designs

For comparison, an FIR filter was designed with the *remez* Octave function:

```
N=1+U+V+M+Q;
brz=remez(N-1,2*[0 fasl fapl fapu fasu 0.5],[0 0 1 1 0 0], ...
[Wasl Wap Wasu],'bandpass');
```

Similarly, an FIR filter was designed with *Selesnick's* Octave function, *cl2bp*. The nominal passband has been adjusted to provide a similar response to the *remez* and PCLS IIR filters. The specifications of the *cl2bp* filter design are:

```
wl=fapl*2*pi*0.8;
wu=fapu*2*pi*1.1;
up=10.^([-dBas, 0, -dBas]/20);
lo=[-1,1,-1].*10.^([-dBas, -dBap, -dBas]/20);
Ccl=floor(N/2)
ncl=2048;
bcl = cl2bp(Ccl,wl,wu,up,lo,512);
```

The FIR filter designs are linear phase with filter length set to the number of coefficients of the IIR PCLS filter. Hence the group delay of the FIR filters is 15 samples compared to 16 samples for the IIR PCLS filter.

The responses of the FIR filters and the IIR PCLS filter are compared in Figure 10.40.

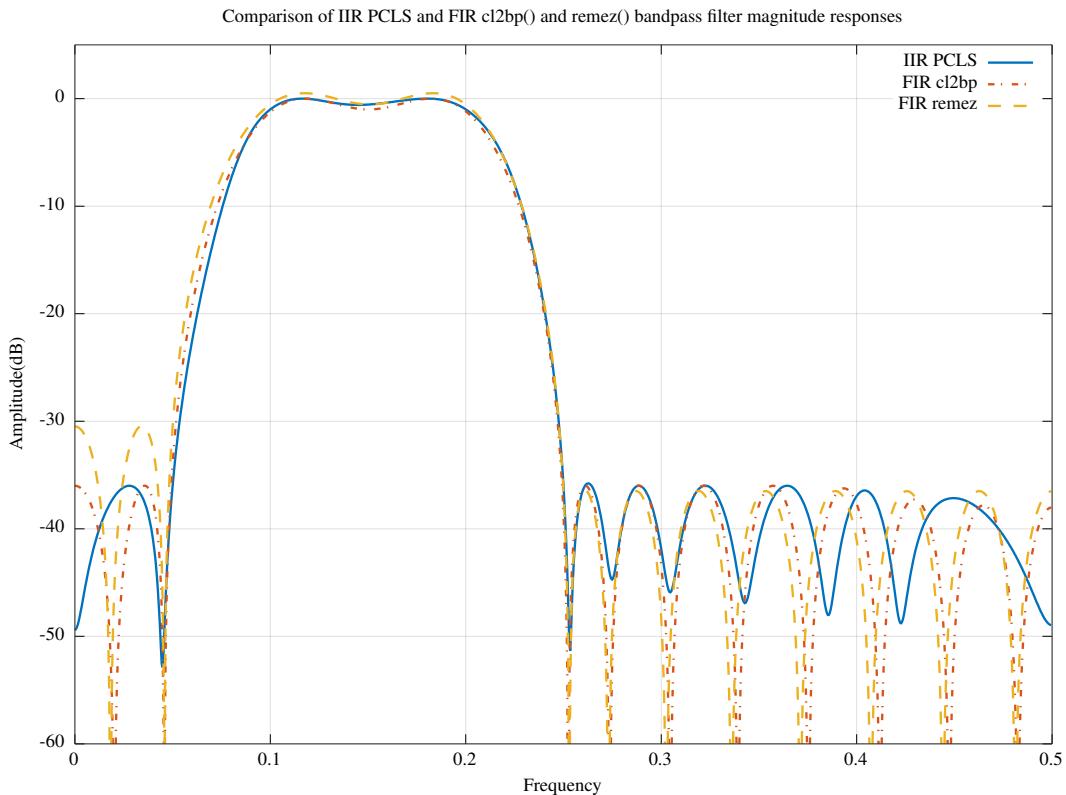


Figure 10.40: Comparison of the magnitude responses of the bandpass PCLS IIR filter and FIR filters designed by *cl2bp* and *remez*

10.2.7 Hilbert transform R=2 decimation filter

A Hilbert transform filter[9, p.118] has the transfer function

$$H_d(\omega) = -i \operatorname{sign}(\omega) e^{-i\omega t_d}$$

The Octave script *iir_sqp_slb_hilbert_test.m* designs a Hilbert transform filter with $U = 7$, $V = 2$, $M = 4$, $Q = 4$, $R = 2$ and the group-delay of the filter is $t_d = \frac{U+M}{2} = 5.5$ samples. An initial filter was designed by the method of Tarczynski et. al. with the Octave script *tarczynski_hilbert_test.m*. The initial filter coefficients are, in gain-pole-zero form:

```
Ux0=7,Vx0=2,Mx0=4,Qx0=4,Rx0=2
x0 = [ -0.0579060329, ...
        -1.8392370631, 1.0642628174, 0.8765050934, -0.9615119703, ...
        -0.6831013577, 0.4149883058, -0.1328122211, ...
        0.9249450019, -0.2221631134, ...
        1.7962303468, 1.6409698917, ...
        2.0743518726, 1.0002172438, ...
        0.2450511070, 0.3859900542, ...
        1.6836440518, 0.1440717597 ]';
```

The amplitude, group delay and phase response of the initial filter, as calculated by the Octave functions *iirA*, *iirT* and *iirP* are shown in Figure 10.41. The script defines a phase response “don’t-care” transition band of $f_{pt} = \pm 0.05$ about $\omega = 0$. Similarly, the group delay transition band is $f_{pt} = \pm 0.07$. Note that the *iirA* function can return a negative amplitude if, as in this case, the gain coefficient is negative. I do this so that $\frac{\partial A}{\partial K}$ is well-defined at $K = 0$ (since the absolute value of K is not differentiable at 0). Consequently, the phase returned by the *iirP* function does not include the sign of the gain coefficient. The phase response, relative to π , is shown adjusted for the group-delay by adding ωt_d where ω is the angular frequency vector for which the response is calculated.

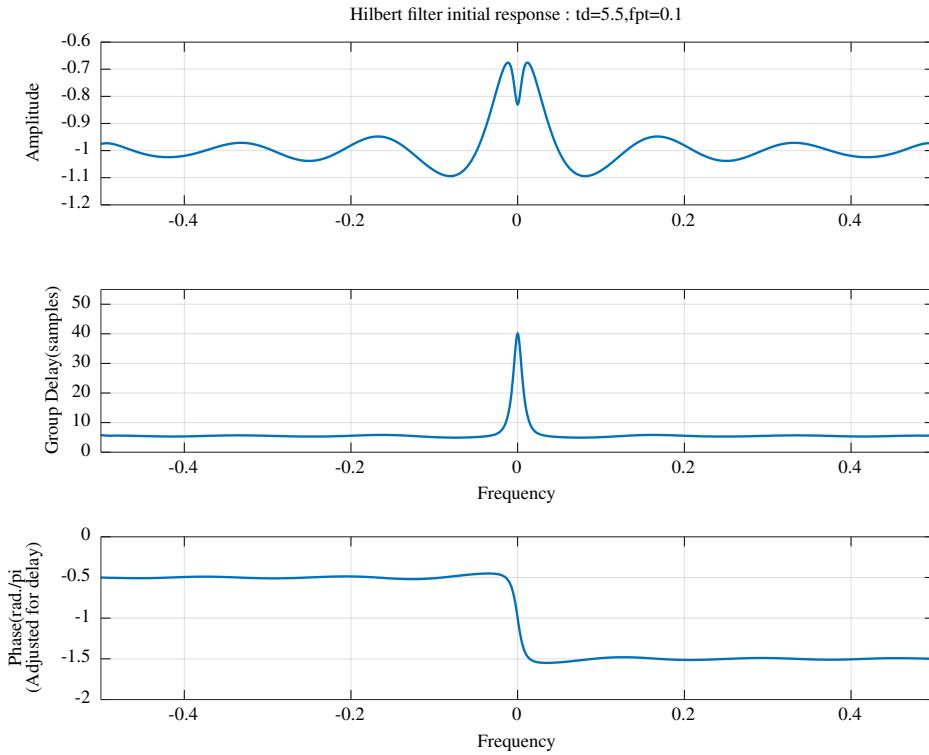


Figure 10.41: Response of the initial Hilbert filter designed by *tarczynski_hilbert_test.m* with $R = 2$ and $t_d = 5.5$

MMSE optimisation of the Hilbert transform R=2 decimator

Figure 10.42 shows the response after MMSE optimisation of the initial filter. The relative weights of the amplitude, group delay and phase errors were $W_{ap} = 1$, $W_{tp} = 0.00001$ and $W_{pp} = 0.001$ respectively. The desired PCLS constraints are superimposed

on the figure. The *iirP* function does not calculate the Hessian of the phase response. In the previous examples only the diagonal of the Hessian matrix of the error response is used to initialise the SQP loop. In this case the diagonal of the Hessian of the phase response is approximated with the Octave function *iirP_hessP_DiagonalApprox.m*.

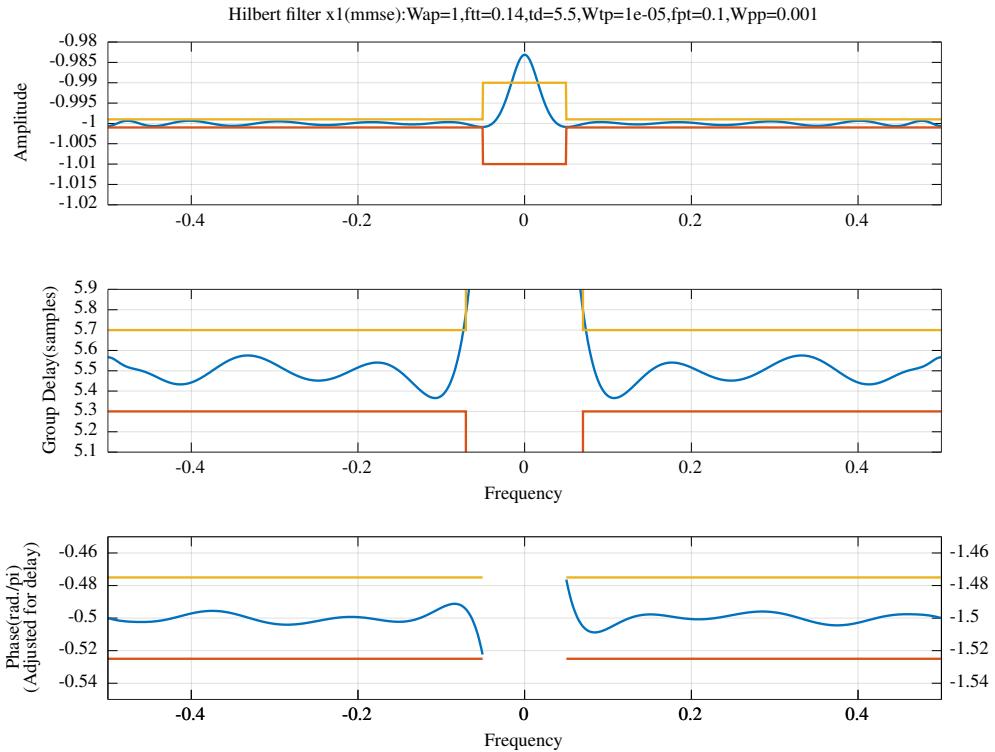


Figure 10.42: Response of the Hilbert filter after MMSE optimisation with $R = 2$, $W_{ap} = 1$, $W_{tp} = 0.00001$ and $W_{pp} = 0.001$

PCLS optimisation of the Hilbert transform R=2 decimator

Figure 10.43 shows the response after PCLS optimisation of the MMSE optimised filter of Figure 10.42. Figure 10.44 shows the pole-zero plot for the PCLS optimised filter. The peak-to-peak amplitude, group delay and phase ripple outside the transition band are $A_r = 0.002$, $t_{dr} = 0.4$ and $p_r = 0.1\frac{\pi}{2}$ respectively. The PCLS optimised filter coefficients are, in gain-pole-zero form

```
Ud1=7,Vd1=2,Md1=4,Qd1=4,Rd1=2
d1 = [ -0.0125534624, ...
        -2.4112464683, 1.2021616747, 0.5756082996, -0.8253308539, ...
        -0.6588645220, 0.5783866727, -0.1637264415, ...
        0.6855081703, -0.0953723689, ...
        2.3675675404, 2.2137974443, ...
        2.0588908825, 1.0189218691, ...
        0.1998070736, 0.3441380790, ...
        1.9143778104, -0.0320242450 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively:

```
N1 = [ -0.0125534624, -0.0201132234, -0.0263194587, -0.0583493637, ...
        -0.1218813747, -0.5018253204, 0.8581945662, 0.9021189554, ...
        -0.6685504990, -0.3523921905, 0.1444860283, 0.0296303566 ]';
```

and

```
D1 = [ 1.0000000000, 0.0000000000, -1.1434444742, 0.0000000000, ...
        0.3268981002, 0.0000000000, -0.0141478780, 0.0000000000, ...
        0.0072286544, 0.0000000000, -0.0020369770, 0.0000000000, ...
        -0.0003091166 ]';
```

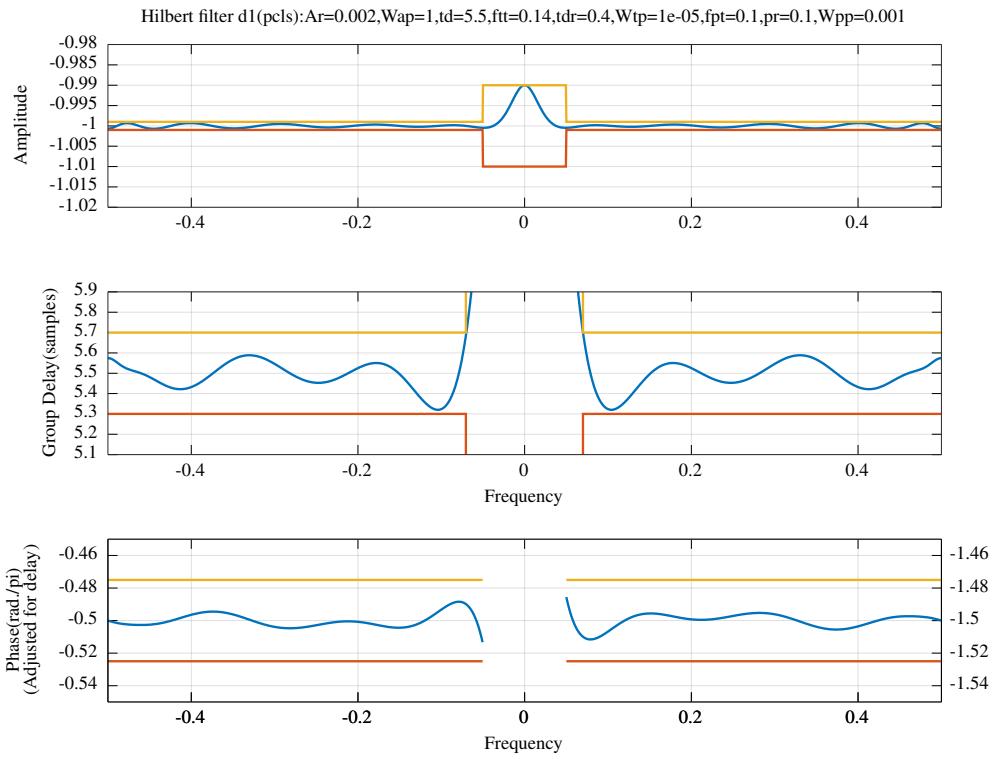


Figure 10.43: Response of the Hilbert filter after PCLS optimisation with $R = 2$, $A_r = 0.002$, $t_{dr} = 0.4$ and $p_r = 0.1\frac{\pi}{2}$

Hilbert filter d1:Ar=0.002,Wap=1,td=5.5,ftt=0.14,tdr=0.4,Wtp=1e-05,fpt=0.1,pr=0.1,Wpp=0.001

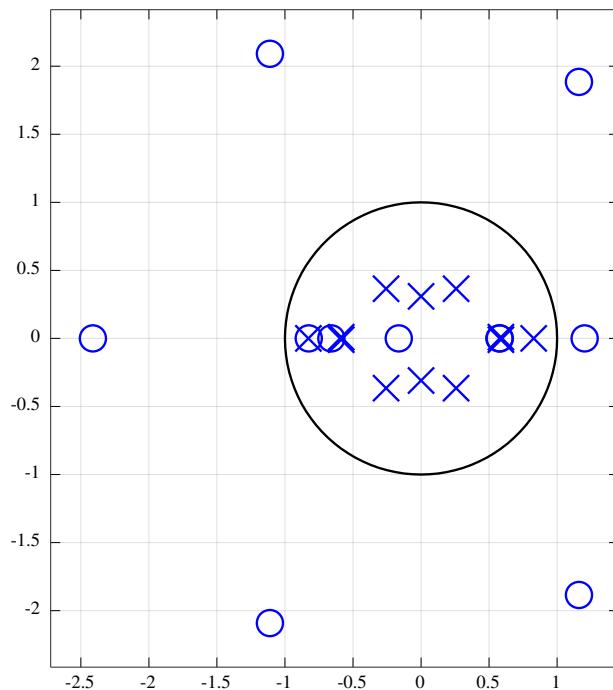


Figure 10.44: Pole-zero plot of the Hilbert filter after PCLS optimisation with $R = 2$, $A_r = 0.002$, $t_{dr} = 0.4$ and $p_r = 0.1\frac{\pi}{2}$

10.2.8 R=2 differentiator filter

A differentiator filter has transfer function $H_d(\omega) = \frac{w}{\pi} e^{-\omega t_d}$. The Octave script *iir_sqp_slb_differentiator_test.m* designs a differentiator filter with $U = 4$, $V = 2$, $M = 4$, $Q = 2$, $R = 2$ and the group-delay of the filter is $t_d = 5.5$ samples. An initial filter was designed by the method of *Tarczynski et al* with the Octave script *tarczynski_differentiator_test.m*. That script designs the transfer function polynomials of the filter and it seems to consistently find cancelling pairs of zeros and poles. The script gives a filter with $U = 6$, $V = 4$, $M = 6$, $Q = 2$, $R = 2$ and a group-delay of $t_d = \frac{U+M-1}{2} = 5.5$. After removing the poles and zeros of this filter that almost cancel, the initial filter coefficients used in this example are, in gain-pole-zero form:

```
Ux0=4,Vx0=2,Mx0=4,Qx0=2,Rx0=2
x0 = [ -0.0033301625, ...
        2.4432889231, 1.0036639004, 0.3309893840, -0.2476976266, ...
        0.2283263247, -0.1370501745, ...
        2.5725431250, 2.6080098957, ...
        2.3928644449, 1.2330172843, ...
        0.1584621214, ...
        1.5131179592 ]';
```

The initial filter amplitude, group delay and phase responses, as calculated by the Octave functions *iirA*, *iirT* and *iirP*, are shown in Figure 10.45. The script defines a “don’t-care” transition band of $ft = \pm 0.05$ at $\omega = 0$ and 0.5. The phase response shown is adjusted for the group-delay by adding $w * td$ where w is the angular frequency vector for which the response is calculated. As for the Hilbert filter example, the gain coefficient is negative, cancelling out the nominal phase of $\frac{3\pi}{2}$.

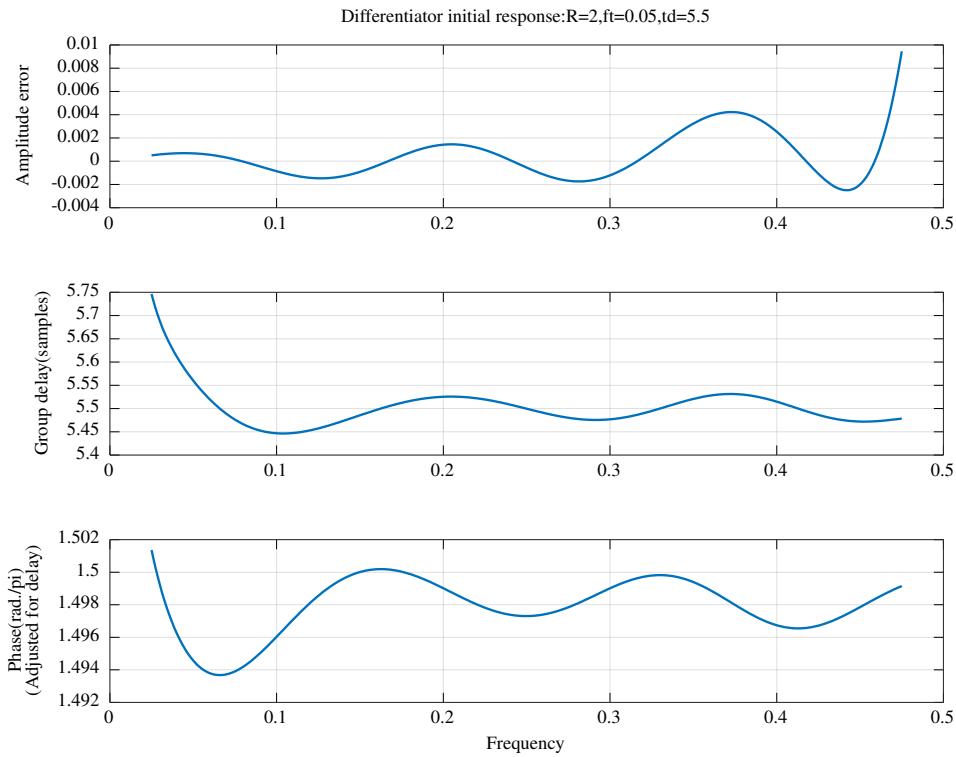


Figure 10.45: Response of the initial R=2 differentiator filter calculated by *iirA()*, *iirT()* and *iirP()*

MMSE optimisation of the R=2 differentiator filter

Figure 10.46 shows the response after MMSE optimisation of the initial filter. The relative weights of the amplitude, group delay and phase error are $W_{ap} = 1$, $W_{tp} = 0.001$ and $W_{pp} = 0.1$ respectively.

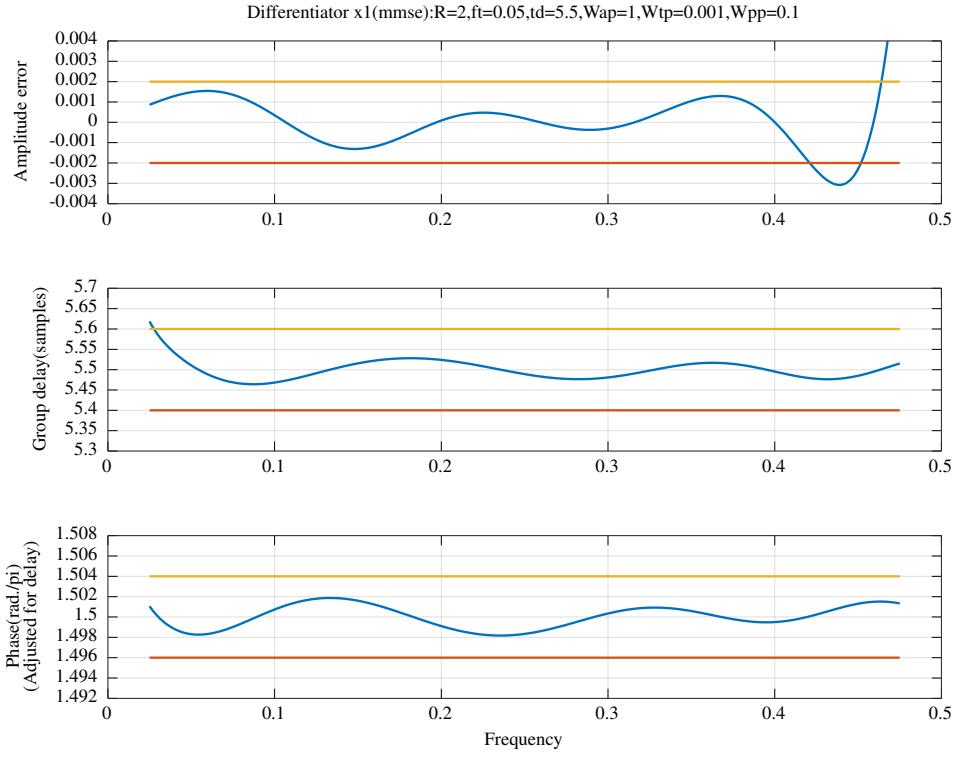


Figure 10.46: Response error of the $R=2$ differentiator filter after MMSE optimisation with $W_{ap} = 1$, $W_{tp} = 0.001$ and $W_{pp} = 0.1$

PCLS optimisation of the $R=2$ differentiator filter

Figure 10.47 shows the response after PCLS optimisation of the MMSE optimised filter of Figure 10.46. Figure 10.48 shows the pole-zero plot for the PCLS filter. The peak-to-peak amplitude, group delay and phase ripple outside the transition bands are $A_r = \pm 0.002$, $t_{dr} = \pm 0.1$ samples and $p_r = \pm 0.004\pi$ respectively. The PCLS optimised filter coefficients are, in gain-pole-zero form:

```
Ud1=4,Vd1=2,Md1=4,Qd1=2,Rd1=2
d1 = [ -0.0017172851, ...
        3.0213805068,    1.0012343545,    0.3949583156,   -0.3066209291, ...
        0.3258273040,   -0.2061751433, ...
        2.8093994324,    2.9950236368, ...
        2.3553802780,    1.1685509527, ...
        0.2438016292, ...
        1.5102452782 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively

```
N1 = [ -0.0017172851,    0.0042693670,   -0.0070979525,    0.0144846495, ...
        -0.0431131701,    0.4022657766,   -0.3979859824,   -0.0159790604, ...
        0.0445414252 ]';
```

and

```
D1 = [ 1.0000000000,    0.0000000000,   -0.1491590108,    0.0000000000, ...
        -0.0042076983,    0.0000000000,   -0.0051298367,    0.0000000000, ...
        -0.0039929786 ]';
```

As a check, Figure 10.49 shows the response of the PCLS optimised filter calculated with the Octave *freqz* and *grpdelay* functions. In this case the amplitude response is plotted as a proportion of the frequency.

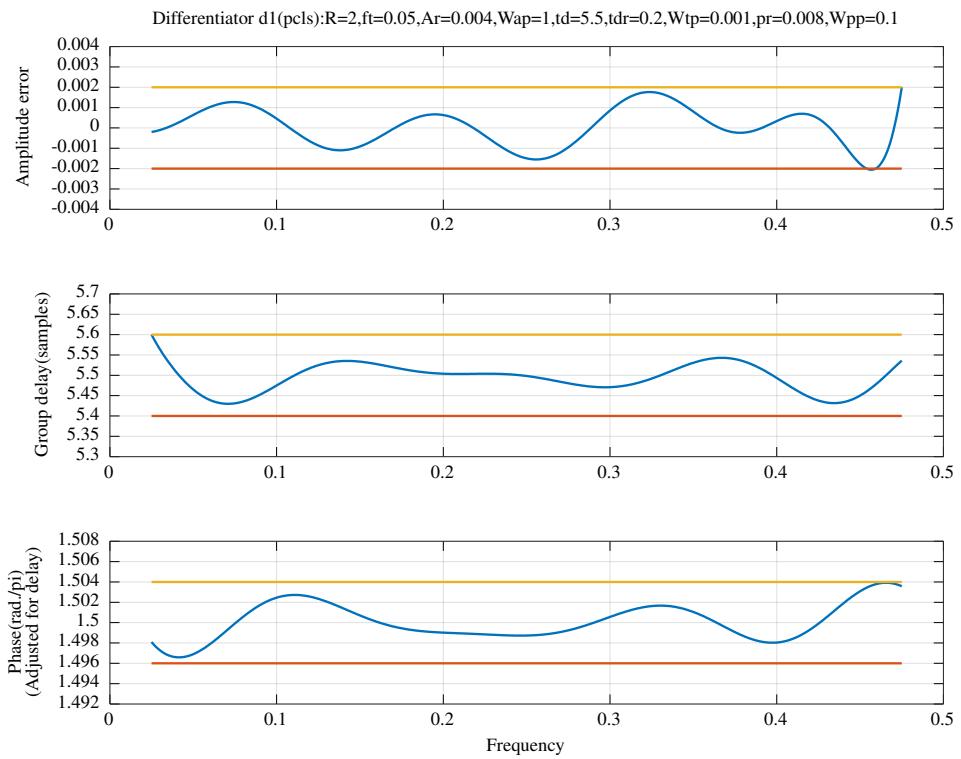


Figure 10.47: Response of the $R=2$ differentiator filter after PCLS optimisation with $A_r = \pm 0.002$, $t_{dr} = \pm 0.1$ and $p_r = \pm 0.004\pi$

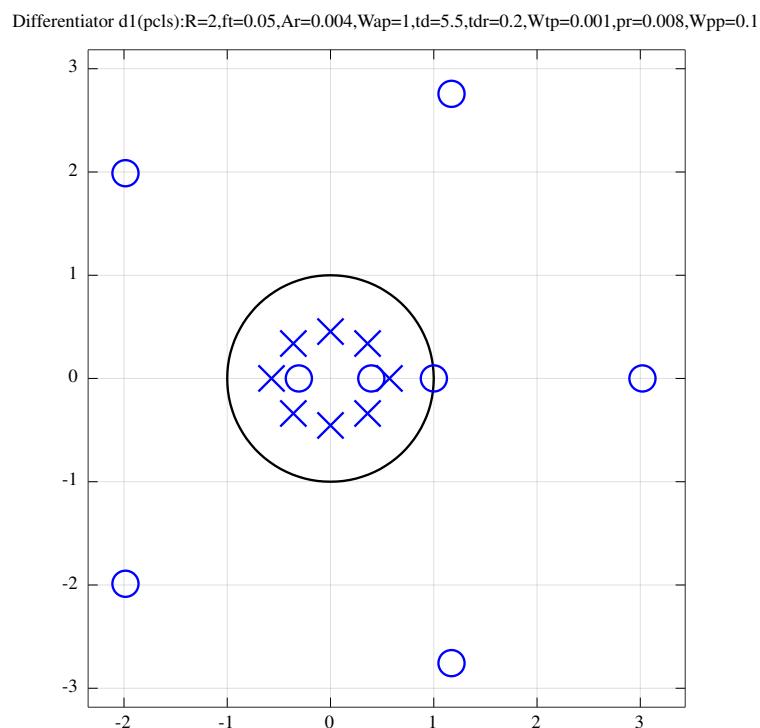


Figure 10.48: Pole-zero plot of the $R=2$ differentiator filter after PCLS optimisation with $A_r = \pm 0.002$, $t_{dr} = \pm 0.1$, $p_r = \pm 0.004\pi$, $W_{ap} = 1$, $W_{tp} = 0.001$ and $W_{pp} = 0.1$

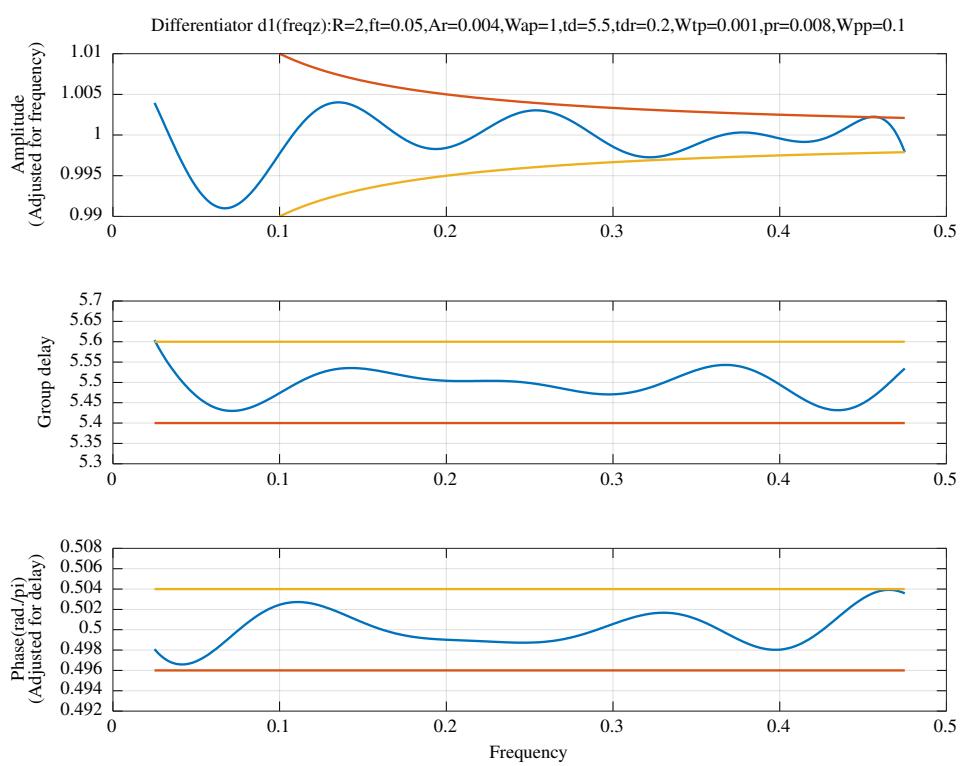


Figure 10.49: Response of the R=2 differentiator filter after PCLS optimisation calculated with *freqz* and *grpdelay*. The amplitude response is plotted as a proportion of the frequency.

10.2.9 Pink noise filter

Pink noise has equal noise power per octave. A pink noise filter has transfer function $H_d(\omega) = \frac{K}{\sqrt{\omega}} e^{-i\omega t_d}$ where K is a constant and t_d is the nominal filter group delay. The Octave script *iir_sqp_slb_pink_test.m* designs an 11th order IIR pink noise filter with the specification:

```
n=1000 % Frequency points across the band
tol_mmse=2e-05 % Tolerance on coef. update (MMSE pass)
tol_pccls=1e-05 % Tolerance on coef. update (PCCLS pass)
fat=0.005 % Amplitude transition band width
Ar=0.04 % Relative amplitude peak-to-peak ripple
Wap=100 % Amplitude weight
fft=0.02 % Group delay transition band width
tp=4.78 % Nominal filter group delay
tpr=0.04 % Filter group delay peak-to-peak ripple
Wtp=1 % Filter group delay weight
U=3 % Number of real zeros
V=1 % Number of real poles
M=8 % Number of complex zeros
Q=10 % Number of complex poles
R=1 % Denominator polynomial decimation factor
```

The script defines a “don’t-care” transition band from $f = 0$ to $fat = 0.005$ for the amplitude response and $fft = 0.02$ for the group delay response. An initial filter was designed by the method of *Tarczynski et al.* in the Octave script *tarczynski_pink_test.m*. The initial filter coefficients are, in gain-pole-zero form:

```
Ux0=3,Vx0=1,Mx0=8,Qx0=10,Rx0=1
x0 = [ 0.0255743587, ...
        -1.5619750054, 0.7463300234, 0.2738485191, ...
        0.8634288354, ...
        1.5918837826, 1.5129484262, 0.7411913002, 0.7167018561, ...
        1.9998197429, 0.8494959342, 2.6042678218, 1.5338409733, ...
        0.7315228779, 0.5728118773, 0.4989497591, 0.7058379436, ...
        0.5226274130, ...
        2.6170934551, 0.4365819189, 2.4902532121, 1.5575288887, ...
        1.3513009567 ]';
```

Figure 10.50 compares the amplitude response of the initial filter to the desired response. Figure 10.51 shows the amplitude response error and group delay of the filter after MMSE and PCCLS optimisation. Figure 10.52 shows the pole-zero plot of the filter.

The PCCLS optimised filter coefficients are, in gain-pole-zero form

```
Ud1=3,Vd1=1,Md1=8,Qd1=10,Rd1=1
d1 = [ 0.0012971188, ...
        -2.6287906907, 0.9150773792, 0.8293083940, ...
        0.9687500000, ...
        2.9360263904, 2.8119974337, -0.7042007037, 0.0544369296, ...
        1.9122515272, 0.7093805091, 3.1415941336, 0.0095301096, ...
        0.3343883578, 0.8293087164, 0.3730095625, 0.3303919106, ...
        0.6307290872, ...
        1.5136784874, 0.0000002052, 2.5185642402, 0.8117538823, ...
        0.0000069925 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively:

```
N1 = [ 0.0012971188, -0.0038055538, 0.0065848905, -0.0099545387, ...
        0.0261210955, 0.1748782423, -0.6920314897, 0.9055691740, ...
        -0.5360065488, 0.1382121156, -0.0108367423, 0.0002592027 ]';
```

and

```
D1 = [ 1.0000000000, -3.7759198213, 5.6464198336, -4.2134745544, ...
        1.6378486920, -0.3522971741, 0.0874912466, -0.0480316042, ...
        0.0269870584, -0.0114677772, 0.0030464356, -0.0004501227 ]';
```

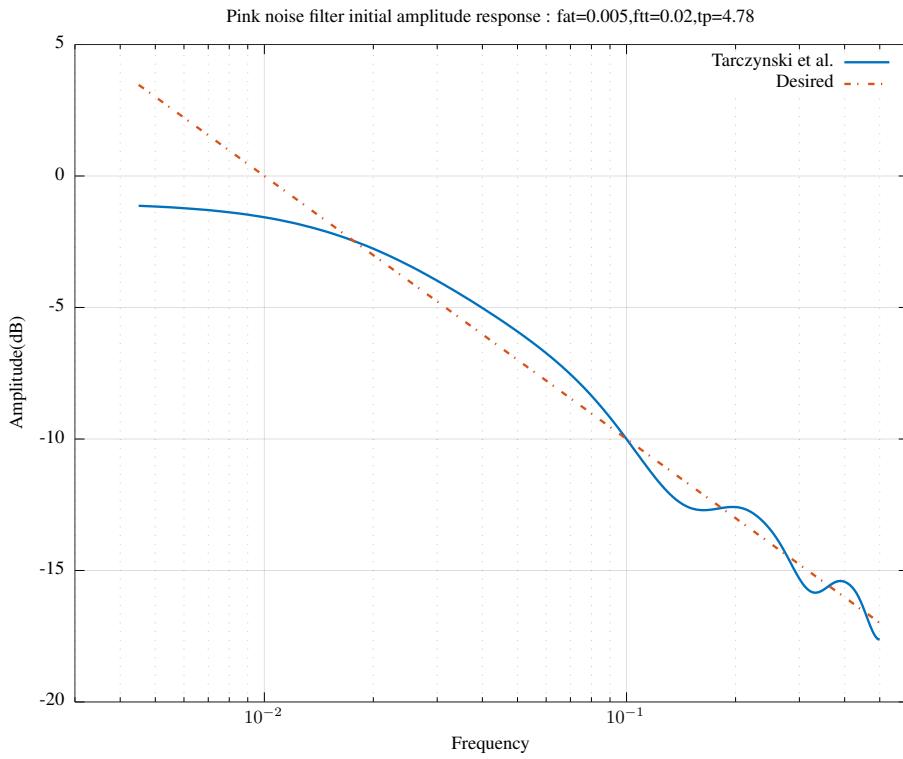


Figure 10.50: Response of the initial pink noise filter designed by *tarczynski_pink_test.m*

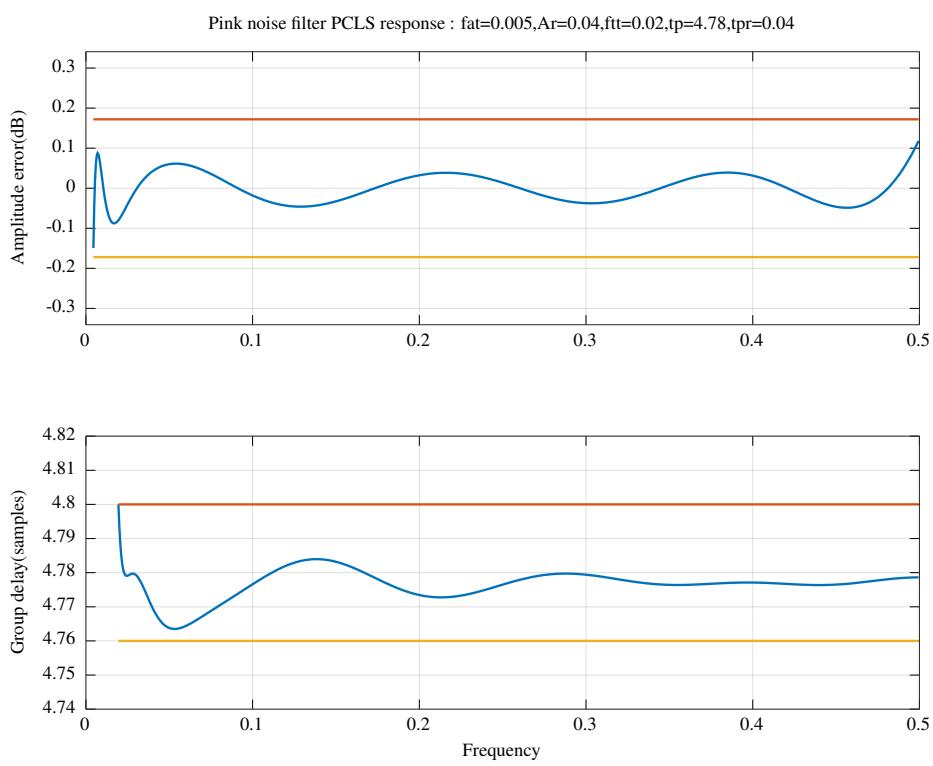


Figure 10.51: Response error of the pink noise filter after PCLS optimisation

Pink noise filter PCLS response : fat=0.005,Ar=0.04,ftt=0.02,tp=4.78,tpr=0.04

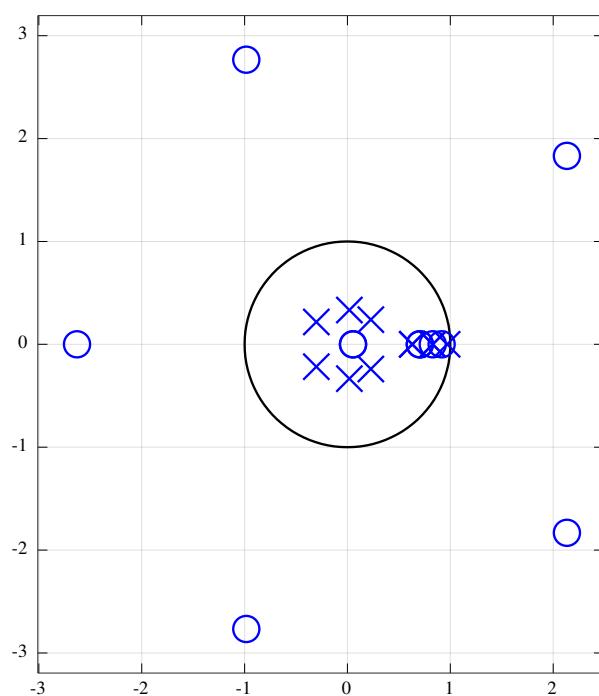


Figure 10.52: Pole-zero plot of the pink noise filter after PCLS optimisation

10.2.10 Minimum phase R=2 filter

A minimum phase filter has all the zeros of the transfer function on or within the unit circle in the z-plane $|z| \leq 1^2$. The Octave script *iir_sqp_slb_minimum_phase_test.m* designs a minimum phase bandpass filter specified by:

```
n=1000 % Frequency points across the band
tol=0.0005 % Tolerance on relative coefficient update size
fap=0.15 % Pass band amplitude response edge
dBap=0.2 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
fas=0.2 % Stop band amplitude response edge
dBas=46 % Stop band minimum attenuation
Was=10 % Stop band amplitude weight
U=2 % Number of real zeros
V=1 % Number of real poles
M=8 % Number of complex zeros
Q=4 % Number of complex poles
R=2 % Denominator polynomial decimation factor
```

The initial filter coefficients are, in gain-pole-zero form:

```
x0=[0.002;-1;-1;0.7;ones(4,1);pi*(12:15)'/16;0.7;0.7;2*pi/3;pi/2];
```

The initial filter is optimised with the PCLS algorithm. Figure 10.53 shows the response of the PCLS optimised filter. Figure 10.54 shows the pass-band detail of the response of the PCLS optimised filter. Figure 10.55 shows the pole-zero plot for the PCLS optimised filter. The resulting filter coefficients are, in gain-pole-zero form:

```
Ud1=2,Vd1=1,Md1=8,Qd1=4,Rd1=2
d1 = [ 0.0182425747, ...
        -1.0000000000, -1.0000000000, ...
        0.2633776048, ...
        1.0000000000, 1.0000000000, 1.0000000000, ...
        2.3400514660, 1.2857033998, 1.5770643441, 2.1023174339, ...
        0.8607712593, 0.5146947874, ...
        1.9522140924, 1.5366166636 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively

```
N1 = [ 0.0182425747, 0.0703239152, 0.1603609007, 0.2697788048, ...
        0.3607727265, 0.3985469637, 0.3607727265, 0.2697788048, ...
        0.1603609007, 0.0703239152, 0.0182425747 ]';
```

and

```
D1 = [ 1.0000000000, 0.0000000000, 0.3422664055, 0.0000000000, ...
        0.8237824266, 0.0000000000, -0.1152814105, 0.0000000000, ...
        0.1584330839, 0.0000000000, -0.0516956379 ]';
```

²If the inverse filter is required then the zeros must be within the unit circle, $|z| \leq \rho < 1$.

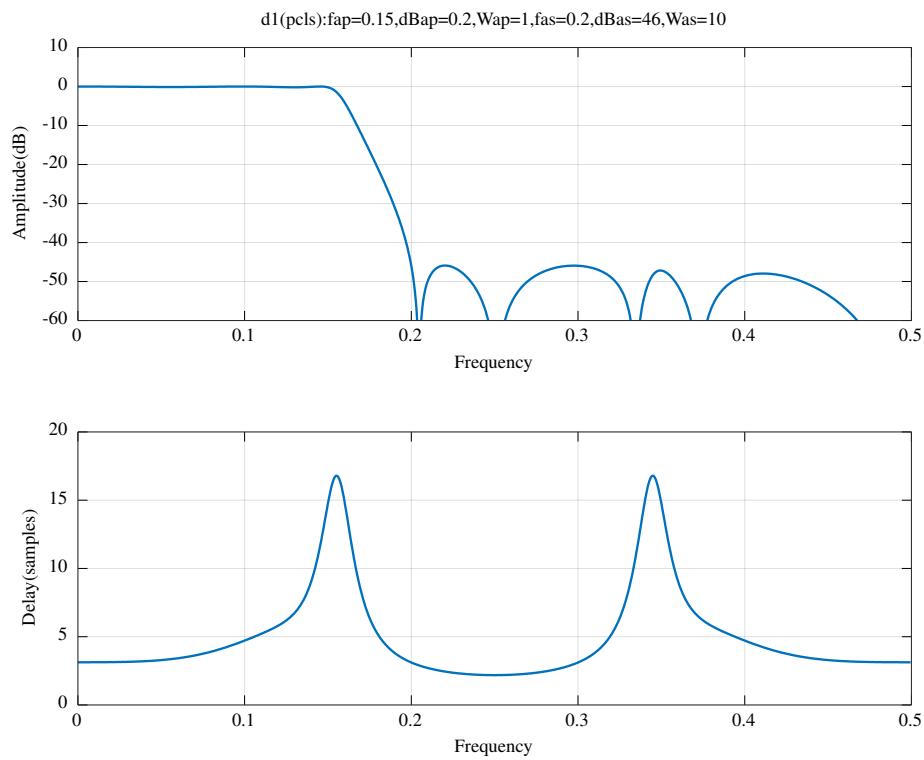


Figure 10.53: Response of the $R=2$ minimum phase filter after PCLS optimisation

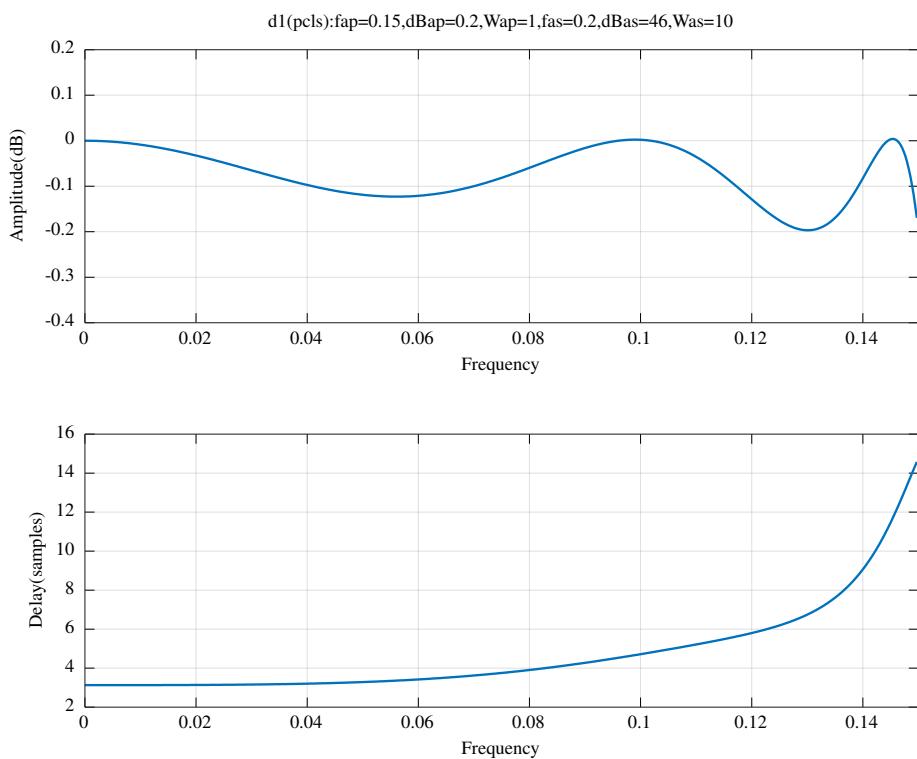


Figure 10.54: Passband response of the $R=2$ minimum phase filter after PCLS optimisation

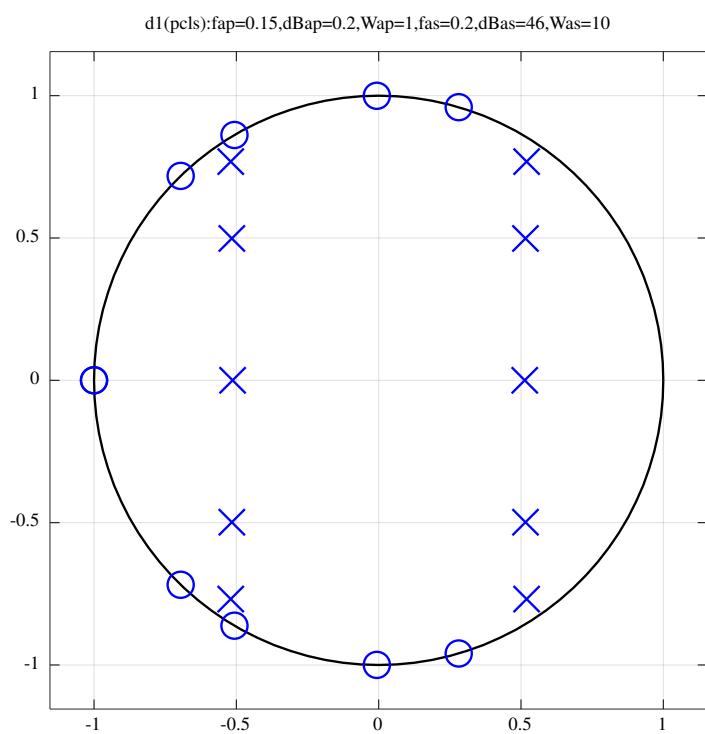


Figure 10.55: Pole-zero plot of the $R=2$ minimum phase filter after PCLS optimisation

10.2.11 Minimum phase FIR bandpass filter

The Octave script *iir_sqp_slb_fir_bandpass_test.m* designs a minimum phase FIR bandpass filter specified by:

```
U=2,V=0,M=28,Q=0,R=1
fapl=0.1,fapu=0.2,dBap=1,Wap=1
fasl=0.05,fasu=0.25,dBas=36,Wasl=5,Wasu=2
```

The initial filter coefficients are, in gain-pole-zero form:

```
x0=[ 0.005, -0.7, 0.7, 0.7*ones(1,14), pi*[(1:3)/80, (13:23)/24] ]';
```

The zero radiiuses are constrained by $|z| < \frac{31}{32}$ (so that the Schur FIR decomposition succeeds). The initial filter is first MMSE optimised and then optimised with the PCLS algorithm. The radiiuses of the zeros of the filter are constrained to $|z| \leq 1$. Figure 10.56 shows the response of the PCLS optimised filter. Figure 10.57 shows the pass-band detail of the response of the PCLS optimised filter. Figure 10.58 shows the pole-zero plot for the PCLS optimised filter. The coefficients of the PCLS optimised minimum phase FIR bandpass filter are, in gain-pole-zero form:

```
Ud1=2,Vd1=0,Md1=28,Qd1=0,Rd1=1
d1 = [ 0.0118204911, ...
        0.9687500000, 0.9687500000, ...
        0.9687500000, 0.9687500000, 0.8530768643, 0.7893758491, ...
        -0.7528760591, -0.8511201883, 0.9687500000, 0.9687500000, ...
        0.9687500000, 0.9687500000, 0.9687500000, 0.9687500000, ...
        0.9687500000, 0.9687500000, ...
        0.2367575237, -0.2987344392, -1.0548116544, 3.1427363676, ...
        0.0007789322, 2.3416969544, 1.5868679326, 1.6616535935, ...
        1.8909590271, 2.7479672987, 3.1403011039, 3.9285838726, ...
        4.1721299226, 3.2016867271 ]';
```

and in transfer function form the FIR polynomial is:

```
Nd1 = [ 0.0118204911, 0.0499523290, 0.0722229638, 0.0030388556, ...
        -0.1373128406, -0.2081326768, -0.0931656062, 0.1375112133, ...
        0.2643512300, 0.1636447194, -0.0461317685, -0.1507106757, ...
        -0.0922372369, -0.0029285497, 0.0012233951, -0.0411924040, ...
        -0.0313085300, 0.0339738863, 0.0714178198, 0.0416859642, ...
        -0.0068611011, -0.0181451409, -0.0008958506, 0.0034838656, ...
        -0.0130236880, -0.0246988421, -0.0133028092, 0.0069431806, ...
        0.0128540726, 0.0063194220, 0.0010946266 ]';
```

The script also calculates the FIR filter with complementary amplitude response. The combined response is shown in Figure 10.59. Figure 10.60 shows the pole-zero plot for complementary FIR filter.

The coefficients of the complementary FIR filter are, in gain-pole-zero form:

```
Uc1=2,Vc1=0,Mc1=28,Qc1=0,Rc1=1
c1 = [ 0.0378186153, ...
        -0.6709116611, -0.6709116610, ...
        1.2548858986, 0.7626702470, 1.0086521218, 1.0048027556, ...
        0.5009124311, 1.0000000001, 1.3164254837, 1.3672360981, ...
        1.3493770287, 1.3449299285, 0.7075598569, 0.7198058249, ...
        0.7373679962, 0.7434462718, ...
        0.3521054151, 0.1229356086, 0.7055539932, 0.9201597361, ...
        0.0000000960, 1.1636319784, 1.5157497250, 2.6629251529, ...
        2.2332627973, 1.8062414959, 1.5960590333, 2.8834083516, ...
        2.4469722618, 2.0204684276 ]';
```

and in transfer function form the FIR polynomial is:

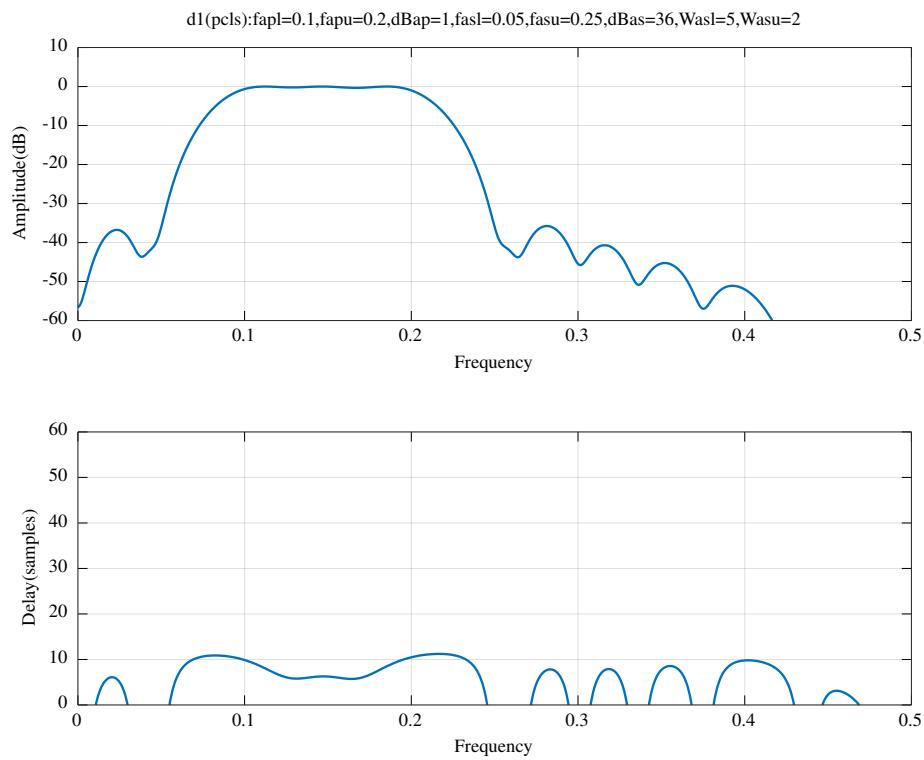


Figure 10.56: Response of the minimum phase FIR bandpass filter after PCLS optimisation

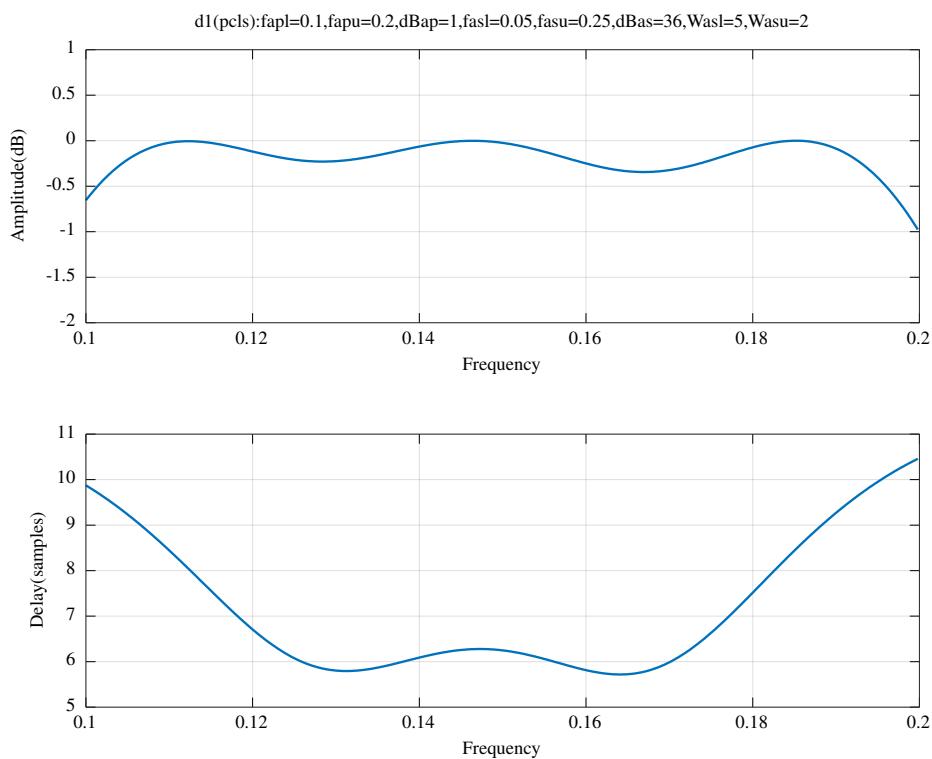


Figure 10.57: Passband response of the minimum phase FIR bandpass filter after PCLS optimisation

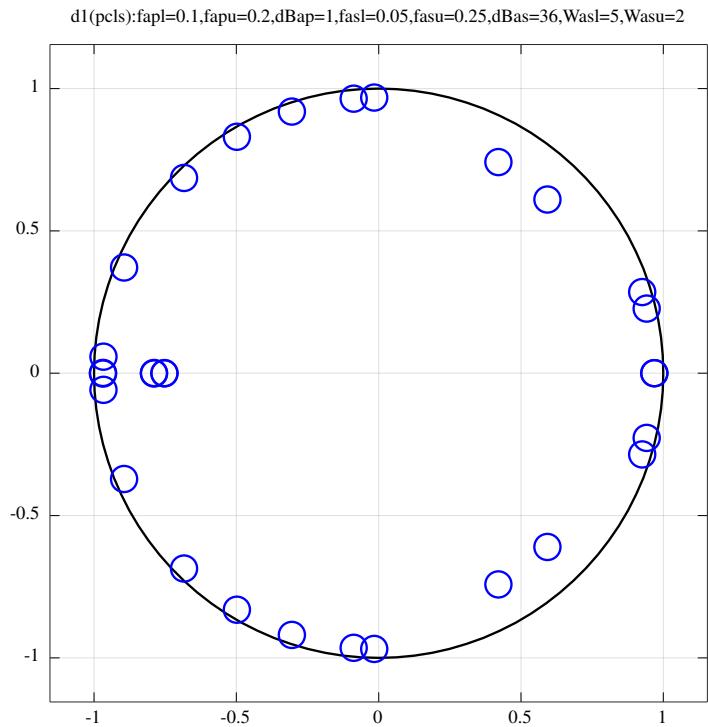


Figure 10.58: Pole-zero plot of the minimum phase FIR bandpass filter after PCLS optimisation

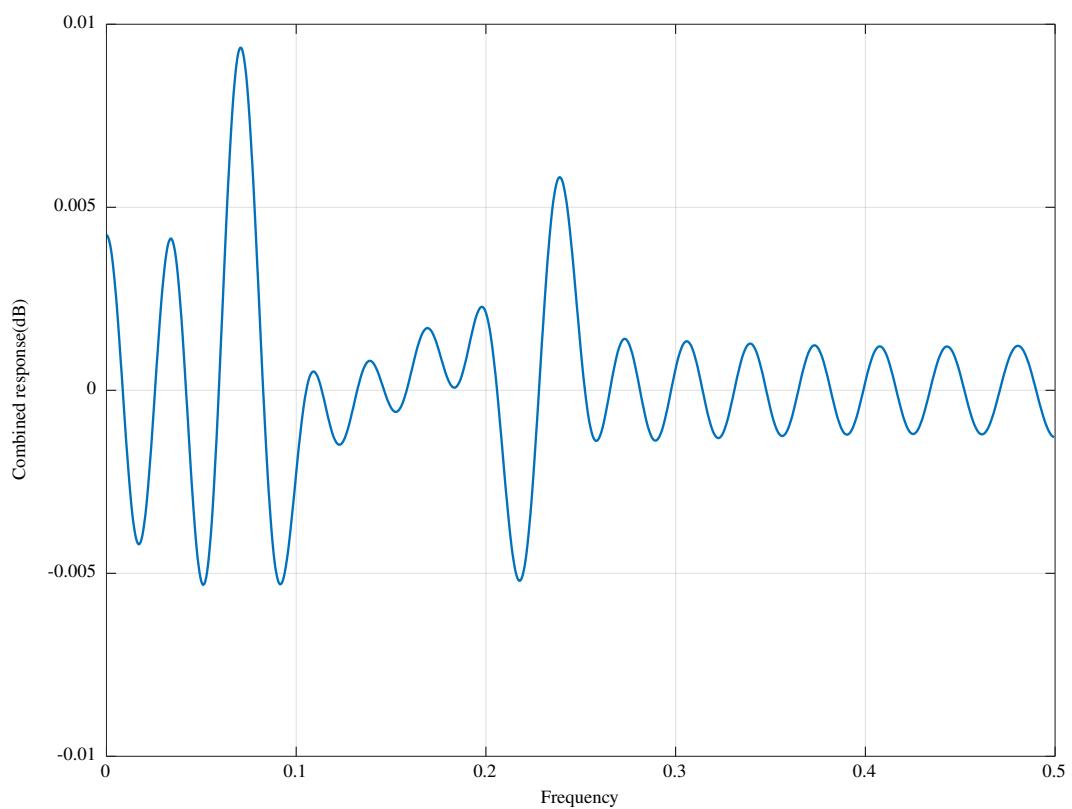


Figure 10.59: Combined response of the PCLS optimised minimum phase FIR bandpass filter and the complementary FIR filter

c1(mmse):fapl=0.1,fapu=0.2,dBap=1,fasl=0.05,fasu=0.25,dBas=36,Wasl=8,Wasu=12

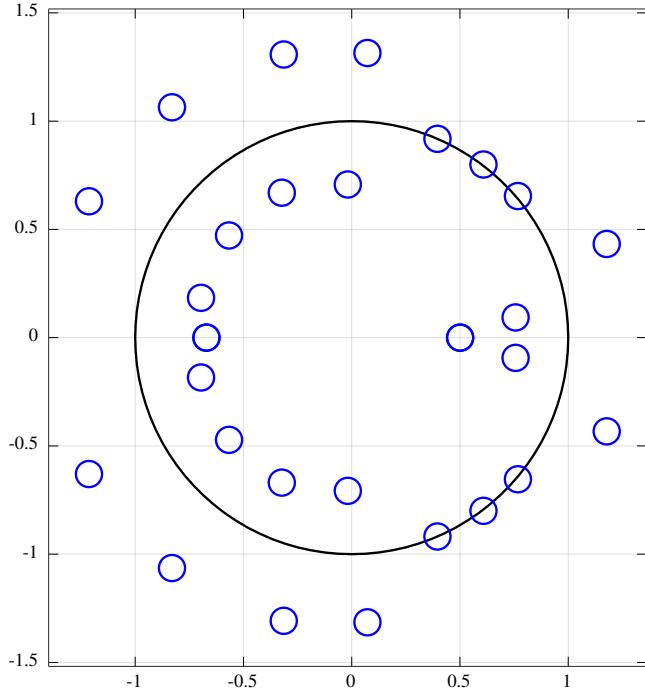


Figure 10.60: Pole-zero plot of the complementary FIR filter

```
Nc1 = [ 0.0378186153, 0.0265538137, 0.0594181653, -0.0196506142, ...
0.0087972059, -0.0697147987, 0.1205946598, 0.0025762583, ...
0.2295909398, -0.0790593575, 0.3472137078, -0.0771458974, ...
0.4866845172, 0.1002260475, -0.2548218258, 0.3729654717, ...
0.0938023550, -0.2247470948, -0.1105890235, 0.0207941800, ...
-0.0130942250, -0.1048206076, -0.0777922076, 0.0153895021, ...
0.0738911241, 0.0506254606, 0.0138766934, -0.0132304306, ...
-0.0185635996, -0.0004442674, 0.0033423877 ]';
```

Alternatively, given an FIR filter of order N , $H(z)$, with the desired magnitude response, $|H(e^{j\omega})| \leq 1$, a more accurate complementary minimum-phase FIR spectral factor of $1 - z^{-N} H(z^{-1}) H(z) = 1 - |H(z)|^2$ can be derived from the real cepstrum (see for example, *Mian and Nainer* [27]) or directly by *Orchard's* Newton-Raphson solution [33]. See Appendix H.

Chapter 11

IIR filter design using Second Order Cone Programming

11.1 Second Order Cone Programming

Following *Alizadeh* and *Goldfarb* [25], Second Order Cone Programming (SOCP) is a class of convex optimisation problems in which a linear function is minimised subject to a set of conic constraints

$$\begin{aligned} & \text{minimise} \quad \mathbf{f}^T \mathbf{x} \\ & \text{subject to} \quad \|A_i \mathbf{x} + b_i\| \leq c_i^T \mathbf{x} + h_i \quad i = 1, \dots, N \end{aligned}$$

where $\|\mathbf{u}\| = \sqrt{\mathbf{u}^T \mathbf{u}}$ is the Euclidean norm of the vector \mathbf{u} , $\mathbf{x} \in \mathbb{R}^{p \times 1}$, $\mathbf{f} \in \mathbb{R}^{p \times 1}$, $A_i \in \mathbb{R}^{(p-1) \times p}$, $b_i \in \mathbb{R}^{(p-1) \times 1}$, $c_i \in \mathbb{R}^{p \times 1}$ and $h_i \in \mathbb{R}$. Each constraint is equivalent to:

$$\begin{bmatrix} c_i^T \\ A_i \end{bmatrix} \mathbf{x} + \begin{bmatrix} h_i \\ b_i \end{bmatrix} \in \mathcal{C}_i$$

where \mathcal{C}_i is a cone in \mathbb{R}^p

$$\mathcal{C}_i = \left\{ \begin{bmatrix} t \\ u \end{bmatrix} : u \in \mathbb{R}^{(p-1) \times 1}, t \geq 0, \|u\| \leq t \right\}$$

11.2 Design of IIR filters with SOCP

Equation 1.2 defined the mini-max optimisation problem for the filter. Rewrite this in terms of an upper bound, ϵ , as

$$\begin{aligned} & \text{minimise} \quad \epsilon \\ & \text{subject to} \quad \|W(\omega_i) [H(\mathbf{x}, \omega_i) - H_d(\omega_i)]\| \leq \epsilon \quad \text{for } \omega_i \in \Omega \\ & \quad H(\mathbf{x}) \text{ is stable} \end{aligned}$$

where \mathbf{x} is the vector of coefficients, ϵ is an auxilliary optimisation variable, $W(\omega)$ is a weighting factor, $H(\mathbf{x}, \omega)$ is the frequency response with coefficients \mathbf{x} and $H_d(\omega)$ is the desired frequency response. $H(\mathbf{x}, \omega)$ and $H_d(\omega)$ usually have complex values in the pass band. The optimisation problem may be formulated so that $H(\mathbf{x}, \omega)$ and $H_d(\omega)$ are complex scalars or vectors of complex values. In the latter case the norm is assumed to be summed over a range of frequencies.

If $\nabla_{\mathbf{x}} H(\mathbf{x}_k, \omega)$ is known and the step-size, $\|\mathbf{x} - \mathbf{x}_k\|$, is small, then a useful *first-order* approximation to $H(\mathbf{x}, \omega)$ is:

$$H(\mathbf{x}, \omega) \approx H(\mathbf{x}_k, \omega) + \nabla_{\mathbf{x}} H(\mathbf{x}_k, \omega)^T (\mathbf{x} - \mathbf{x}_k)$$

so that

$$\begin{aligned} W(\omega) |H(\mathbf{x}, \omega) - H_d(\omega)| & \approx W(\omega) \left| \nabla_{\mathbf{x}} H(\mathbf{x}_k, \omega)^T (\mathbf{x} - \mathbf{x}_k) + H(\mathbf{x}_k, \omega) - H_d(\omega) \right| \\ & = W(\omega) \left| \nabla_{\mathbf{x}} H(\mathbf{x}_k, \omega)^T (\mathbf{x} - \mathbf{x}_k) + \mathbf{e}_k \right| \end{aligned}$$

where $e_k = H(\mathbf{x}_k, \omega) - H_d(\omega)$.

Similarly, a *second-order* approximation to $H(\mathbf{x}, \omega)$ is:

$$H(\mathbf{x}, \omega) \approx H(\mathbf{x}_k, \omega) + \nabla_x H(\mathbf{x}_k, \omega)^T (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_k)^T \nabla_{xx}^2 H(\mathbf{x}_k, \omega) (\mathbf{x} - \mathbf{x}_k)$$

Unfortunately the IIR filter Hessian matrix, $\nabla_{xx}^2 H(\mathbf{x}_k, \omega)$, is unlikely to be positive definite.

Lu and Hinamoto [95] add linear constraints on the maximum filter response in the transition band, Ω_t :

$$\|\nabla_x H(\mathbf{x}_k, \omega_t)^T (\mathbf{x} - \mathbf{x}_k) + H(\mathbf{x}_k, \omega_t)\| \leq \gamma \quad \text{for } \omega_t \in \Omega_t$$

Similar linear constraints can be added to control the peaks of the response in the pass and stop bands.

The linearised SOCP optimisation problem is

$$\begin{aligned} & \text{minimise} \quad \epsilon \\ & \text{subject to} \quad \|W(\omega_i) [\nabla_x H(\mathbf{x}_k, \omega_i)^T (\mathbf{x} - \mathbf{x}_k) + e_k]\| \leq \epsilon \quad \text{for } \omega_i \in \Omega \\ & \quad \|\nabla_x H(\mathbf{x}_k, \omega_t)^T (\mathbf{x} - \mathbf{x}_k) + H(\mathbf{x}_k, \omega_t)\| \leq \gamma \quad \text{for } \omega_t \in \Omega_t \\ & \quad \|\mathbf{x} - \mathbf{x}_k\| \leq \beta \\ & \quad H(\mathbf{x}) \text{ is stable} \end{aligned}$$

There are three sets of second-order cone constraints (for the desired response, the transition band response and for the coefficient step size) and one set of linear constraints (for filter stability). The optimisation is repeated until the step size, $(\mathbf{x} - \mathbf{x}_k)$, is satisfactory or the iteration limit is exceeded. Typically, ϵ minimises the weighted sum of, for example, the amplitude and delay errors, \mathcal{E}_A and \mathcal{E}_T . Separately minimising the amplitude and delay errors implies the minimisation of two auxilliary variables ϵ_A and ϵ_T .

The complex frequency response can be written in terms of amplitude and phase as

$$\begin{aligned} H(\mathbf{x}_k, \omega_i) &= H_a(\mathbf{x}_k, \omega_i) e^{iH_p(\mathbf{x}_k, \omega_i)} \\ &= H_a(\mathbf{x}_k, \omega_i) [\cos H_p(\mathbf{x}_k, \omega_i) + i \sin H_p(\mathbf{x}_k, \omega_i)] \end{aligned}$$

so that, at each ω_i , the gradient of the complex frequency response is

$$\begin{aligned} \nabla_x H(\mathbf{x}_k, \omega_i) &= [\cos H_p(\mathbf{x}_k, \omega_i) \nabla_x H_a(\mathbf{x}_k, \omega_i) - H_a(\mathbf{x}_k, \omega_i) \sin H_p(\mathbf{x}_k, \omega_i) \nabla_x H_p(\mathbf{x}_k, \omega_i)] \dots \\ &\quad + i [\sin H_p(\mathbf{x}_k, \omega_i) \nabla_x H_a(\mathbf{x}_k, \omega_i) + H_a(\mathbf{x}_k, \omega_i) \cos H_p(\mathbf{x}_k, \omega_i) \nabla_x H_p(\mathbf{x}_k, \omega_i)] \end{aligned}$$

The squared magnitude of the response is used if there is no design constraint on the phase of the filter response (for example in the stop band). In this case

$$\begin{aligned} |H(\mathbf{x}_k, \omega_i)|^2 &= H(\mathbf{x}_k, \omega_i) H(\mathbf{x}_k, \omega_i)^\dagger \\ \nabla_x |H(\mathbf{x}_k, \omega_i)|^2 &= (\nabla_x H(\mathbf{x}_k, \omega_i)) H(\mathbf{x}_k, \omega_i)^\dagger + H(\mathbf{x}_k, \omega_i) (\nabla_x H(\mathbf{x}_k, \omega_i))^\dagger \\ &= 2 \operatorname{Re} \nabla_x H(\mathbf{x}_k, \omega_i) \operatorname{Re} H(\mathbf{x}_k, \omega_i) + 2 \operatorname{Im} \nabla_x H(\mathbf{x}_k, \omega_i) \operatorname{Im} H(\mathbf{x}_k, \omega_i) \end{aligned}$$

where \dagger represents the element-wise complex conjugate. The squared-magnitude response can be used in linear constraints on the upper and lower peaks of the pass-band and stop-band magnitude responses

$$\begin{aligned} |H(\mathbf{x}_k, \omega_i)|^2 + (\mathbf{x} - \mathbf{x}_k)^T \nabla_x |H(\mathbf{x}_k, \omega_i)|^2 &\geq H_{d,l}^2(\omega_i) \\ |H(\mathbf{x}_k, \omega_i)|^2 + (\mathbf{x} - \mathbf{x}_k)^T \nabla_x |H(\mathbf{x}_k, \omega_i)|^2 &\leq H_{d,u}^2(\omega_i) \end{aligned}$$

where $H_{d,l}^2(\omega)$ and $H_{d,u}^2(\omega)$ are the lower and upper constraints on the squared-magnitude response. A similar linear constraint on the phase response uses

$$\begin{aligned} \arg H(\mathbf{x}_k, \omega_i) &= \tan^{-1} \frac{\operatorname{Im} H(\mathbf{x}_k, \omega_i)}{\operatorname{Re} H(\mathbf{x}_k, \omega_i)} \\ \nabla_x \arg H(\mathbf{x}_k, \omega_i) &= \frac{\operatorname{Re} H(\mathbf{x}_k, \omega_i) \operatorname{Im} \nabla_x H(\mathbf{x}_k, \omega_i) - \operatorname{Im} H(\mathbf{x}_k, \omega_i) \operatorname{Re} \nabla_x H(\mathbf{x}_k, \omega_i)}{|H(\mathbf{x}_k, \omega_i)|^2} \end{aligned}$$

11.3 Using the *SeDuMi* SOCP solver

In the following I use the *SeDuMi* (*Self-Dual-Minimisation*) SOCP solver originally written by *Jos Sturm* [91]. The *Colophon* shows construction of a *patch* file that makes *SeDuMi v1.3* compatible with Octave.

Lu [100, Section III] provides an example of expressing an optimisation problem in the form accepted by SeDuMi. In Lu's notation the problem is

$$\text{minimise} \quad \mathbf{b}^T \mathbf{x} \quad (11.1a)$$

$$\text{subject to} \quad \| \mathbf{A}_i^T \mathbf{x} + \mathbf{c}_i \| \leq \mathbf{b}_i^T \mathbf{x} + d_i \quad \text{for } i = 1, \dots, q \quad (11.1b)$$

$$\mathbf{D}^T \mathbf{x} + \mathbf{f} \geq \mathbf{0} \quad (11.1c)$$

where $\mathbf{x} \in \mathbb{R}^{m \times 1}$, $\mathbf{D} \in \mathbb{R}^{m \times p}$, $\mathbf{f} \in \mathbb{R}^{p \times 1}$, $\mathbf{b} \in \mathbb{R}^{m \times 1}$, $\mathbf{A}_i \in \mathbb{R}^{m \times (n_i - 1)}$ ¹, $\mathbf{b}_i \in \mathbb{R}^{m \times 1}$, $\mathbf{c}_i \in \mathbb{R}^{(n_i - 1) \times 1}$ and $d_i \in \mathbb{R}$ for $1 \leq i \leq q$. The problem is cast into SeDuMi format by defining

$$\begin{aligned} \mathbf{A}_t &= \begin{bmatrix} -\mathbf{D} & \mathbf{A}_t^{(1)} \dots \mathbf{A}_t^{(q)} \end{bmatrix} \\ \mathbf{A}_t^{(i)} &= -[\mathbf{b}_i \quad \mathbf{A}_i] \\ \mathbf{b}_i &= -\mathbf{b} = [-1 \quad 0 \quad \dots \quad 0]^T \\ \mathbf{c}_t &= \begin{bmatrix} \mathbf{f}; & \mathbf{c}_t^{(1)}; & \dots & \mathbf{c}_t^{(q)} \end{bmatrix} \\ \mathbf{c}_t^{(i)} &= [d_i; \quad \mathbf{c}_i] \end{aligned} \quad (11.2)$$

The Octave commands to solve the SOCP problem with SeDuMi are

```
K.p = p;
K.q = q;
[xs,ys,info] = sedumi(At,bt,ct,K);
info
x = ys;
```

where p is the number of linear constraints in Equation 11.1c and q is a vector giving the dimensions of the q sets of conic constraints in Equation 11.1b, $\mathbf{q} = [n_1 \quad n_2 \quad \dots \quad n_q]$.

¹ $n_i - 1$ to allow for d_i in the column vector $\mathbf{c}_t^{(i)}$

11.4 An example of SOCP MMSE design of an IIR filter expressed in *gain-zero-pole* format with *SeDuMi*

The Octave script *deczky3_socp_test.m* implements the design of Deczky's Example 3, used previously in Section 10.2.3, with MMSE optimisation of the weighted response error by the *SeDuMi* SOCP solver. As in Part 10, the coefficients of this example filter are expressed in *gain-zero-pole* form and the amplitude and group-delay are calculated with the *iirA* and *iirT* functions. Similarly, filter stability is ensured by linear constraints on the upper and lower values of the coefficients.

As for the filter design in Section 10.2.3, the *deczky3_socp_test.m* script has two phases. First, starting with the "IPSZ-1" coefficients the sum of the coefficient step-size and MMSE error is minimised in the the Octave function *iir_socp_mmse.m*. Linesearch along the minimal direction is not required. The minimisation variables are

$$\mathbf{x} = \begin{bmatrix} \epsilon \\ \beta \\ \delta \end{bmatrix}$$

where ϵ is the MMSE error, δ is the step direction from coefficient vector \mathbf{x}_k , and β is the constraint on the coefficient step-size, $\|\delta\| \leq \beta$.

In a similar fashion to the MMSE error used in Part 10, the MMSE frequency response constraint can be expressed as a weighted sum of pass-band amplitude response, A , stop-band amplitude response, S , and group-delay response, T . In this example I do not attempt to control the transition band amplitude response. In *SeDuMi* format

$$\begin{aligned} \mathbf{A}_i^T &= \begin{bmatrix} \mathbf{0} & W_a(\omega_s) \nabla_x A(\mathbf{x}_k, \omega_a) \\ \mathbf{0} & W_t(\omega_t) \nabla_x T(\mathbf{x}_k, \omega_t) \\ \mathbf{0} & W_s(\omega_s) \nabla_x S(\mathbf{x}_k, \omega_s) \end{bmatrix} \\ \mathbf{b}_i^T &= [1 \ 0 \ \mathbf{0}] \\ \mathbf{c}_i &= \begin{bmatrix} W_a(\omega_a) [A(\mathbf{x}_k, \omega_a) - A_d(\omega_a)] \\ W_t(\omega_t) [T(\mathbf{x}_k, \omega_t) - T_d(\omega_t)] \\ W_s(\omega_s) [S(\mathbf{x}_k, \omega_s) - S_d(\omega_s)] \end{bmatrix} \\ d_i &= 0 \end{aligned}$$

where $\omega_a \in \Omega_a$, $\omega_t \in \Omega_t$ and $\omega_s \in \Omega_s$ are the pass band amplitude, and pass band group delay and stop band amplitude response grid frequencies and $A_d(\omega_a)$, $T_d(\omega_t)$ and $S_d(\omega_s)$ are the desired pass band amplitude, pass band delay and stop band amplitude responses, respectively.

The gain-zero-pole coefficients have upper and lower constraints, \mathbf{x}_u and \mathbf{x}_l respectively, ensuring stability with

$$\begin{aligned} -(\mathbf{x} - \mathbf{x}_k) - (\mathbf{x}_k - \mathbf{x}_u) &\geq 0 \\ (\mathbf{x} - \mathbf{x}_k) - (\mathbf{x}_l - \mathbf{x}_k) &\geq 0 \end{aligned}$$

The stability constraint is a linear constraint on the coefficients

$$\begin{aligned} \mathbf{D}^T &= \begin{bmatrix} \mathbf{0} & -\mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ \mathbf{f} &= \begin{bmatrix} -(\mathbf{x}_k - \mathbf{x}_u) \\ -(\mathbf{x}_l - \mathbf{x}_k) \end{bmatrix} \end{aligned}$$

During the second, PCLS, phase, the Octave function, *iir_slb.m* minimises the sum of the coefficient step-size and the MMSE error subject to linear constraints on the pass-band amplitude response, stop-band amplitude response and group-delay response. For example, the pass-band amplitude response upper and lower constraints are

$$\begin{aligned} A_{du}(\mathbf{x}_k, \omega_{au}) - \left[A(\mathbf{x}_k, \omega_{au}) + \nabla_x A(\mathbf{x}_k, \omega_{au})^T \delta \right] &\geq 0 \\ -A_{dl}(\mathbf{x}_k, \omega_{al}) + \left[A(\mathbf{x}_k, \omega_{al}) + \nabla_x A(\mathbf{x}_k, \omega_{al})^T \delta \right] &\geq 0 \end{aligned}$$

where $\omega_{au} \in \Omega_{au}$ and $\omega_{al} \in \Omega_{al}$ are the upper and lower pass-band amplitude response constraint frequencies. These constraints are added to the *SeDuMi* problem as linear constraints. The constraint frequencies are determined by the PCLS peak-exchange algorithm of Selesnick *et al.* shown in Algorithm 20.

The *deczky3_socp_test.m* script has somewhat tighter constraints on the pass-band group-delay ripple than the SQP design of *deczky3_sqp_test.m* shown in Section 10.2.3. The filter specification is

```

n=500 % Frequency points across the band
tol=2e-05 % Tolerance on relative coefficient update size
fap=0.15 % Pass band amplitude response edge
dBap=0.1 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.25 % Pass band group delay response edge
tp=10 % Nominal filter group delay
tpr=0.003 % Pass band group delay peak-to-peak ripple
Wtp=1 % Pass band group delay weight
fas=0.3 % Stop band amplitude response edge
dBas=33 % Stop band minimum attenuation
Was=1.1 % Stop band amplitude weight
U=0 % Number of real zeros
V=0 % Number of real poles
M=10 % Number of complex zeros
Q=6 % Number of complex poles
R=1 % Denominator polynomial decimation factor

```

The result of the PCLS SOCP pass in *deczky3_socp_test.m* is shown in Figure 11.1 with pass-band details shown in Figure 11.2 and pole-zero plot shown in Figure 11.3.

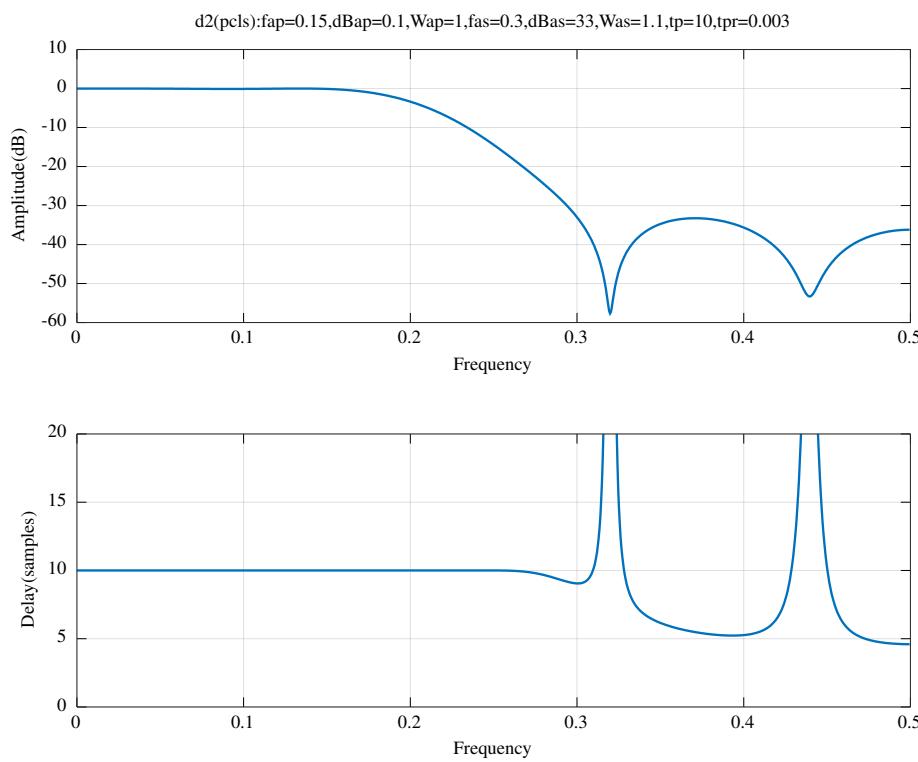


Figure 11.1: Deczky Example 3 PCLS : Response after SOCP optimisation

The estimate of the SOCP PCLS optimised filter vector is, in the gain, zeros and poles form of Equation 10.2:

```

Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [ 0.0035162429, ...
        1.0318419574, 1.0092377454, 1.3833181691, 1.7931632656, ...
        2.1785594333, ...
        2.7612792576, 2.0096447479, 1.7570038190, 0.7267396652, ...
        0.1872795892, ...
        0.6347159223, 0.5923001343, 0.4948392664, ...
        1.4364417161, 1.0937339837, 0.3404444899 ]';

```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

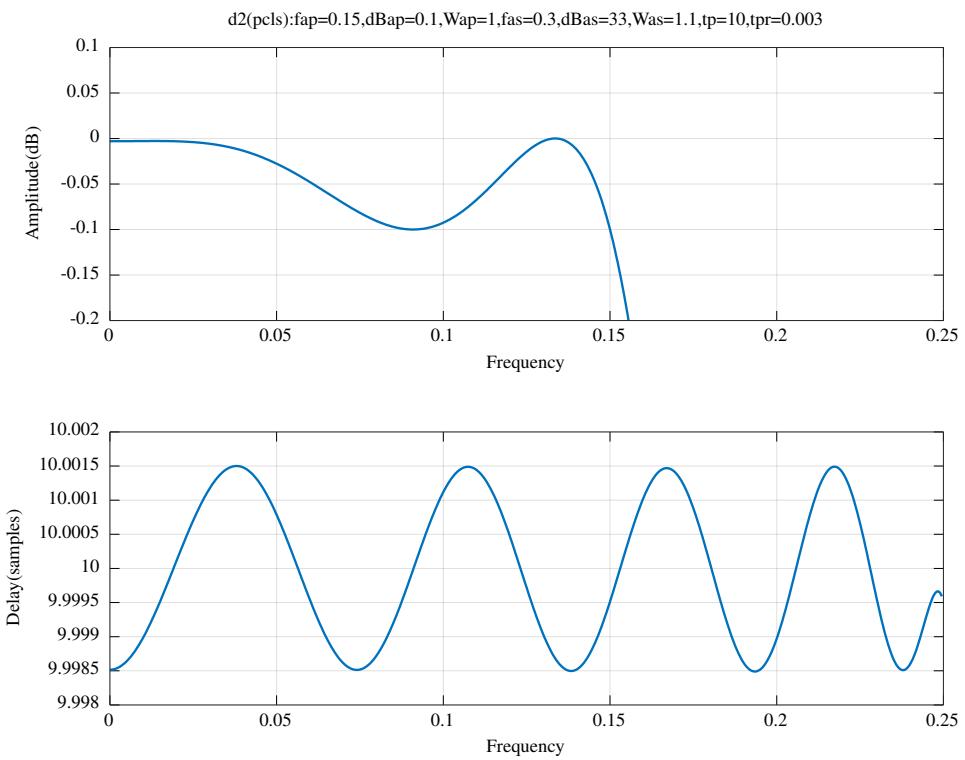


Figure 11.2: Deczky Example 3 PCLS : Passband response after SOCP optimisation

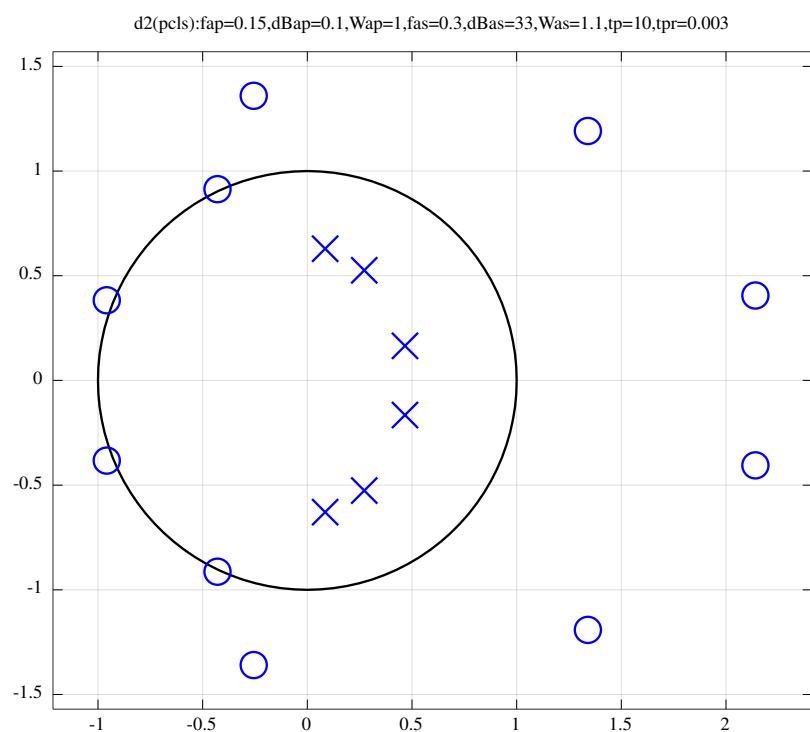


Figure 11.3: Deczky Example 3 PCLS : Pole-zero plot after SOCP optimisation

```
N2 = [ 0.0035162429, -0.0129224822, 0.0127344097, -0.0059454417, ...
0.0174537200, -0.0080671839, -0.0387572439, 0.0010177540, ...
0.0991325203, 0.1307251199, 0.1113563162 ]';
```

and

```
D2 = [ 1.0000000000, -1.6468533105, 1.7570930697, -1.2429916272, ...
0.6086048976, -0.2001111066, 0.0346075422 ]';
```

For comparison, the *deczky3a_socp_test.m* script has looser constraints on the pass-band group-delay ripple and increases the required stop-band attenuation:

```
n=500 % Frequency points across the band
tol=2e-05 % Tolerance on relative coefficient update size
fap=0.15 % Pass band amplitude response edge
dBap=0.2 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.25 % Pass band group delay response edge
tp=10 % Nominal filter group delay
tpr=0.5 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
fas=0.3 % Stop band amplitude response edge
dBas=50 % Stop band minimum attenuation
Was=10 % Stop band amplitude weight
U=0 % Number of real zeros
V=0 % Number of real poles
M=10 % Number of complex zeros
Q=6 % Number of complex poles
R=1 % Denominator polynomial decimation factor
```

The result of the PCLS SOCP pass in *deczky3a_socp_test.m* is shown in Figure 11.4 with pass-band details shown in Figure 11.5 and pole-zero plot shown in Figure 11.6. The estimate of the SOCP PCLS optimised filter vector is, in the gain, zeros and poles

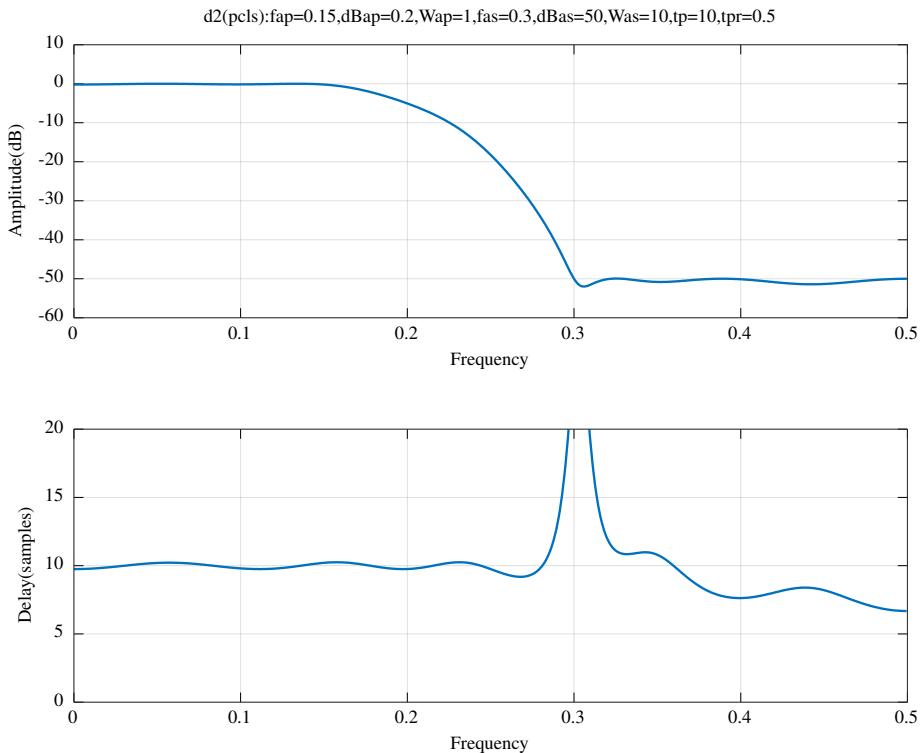


Figure 11.4: Deczky Example 3 PCLS : Response after SOCP optimisation, alternative specification

form of Equation 10.2:

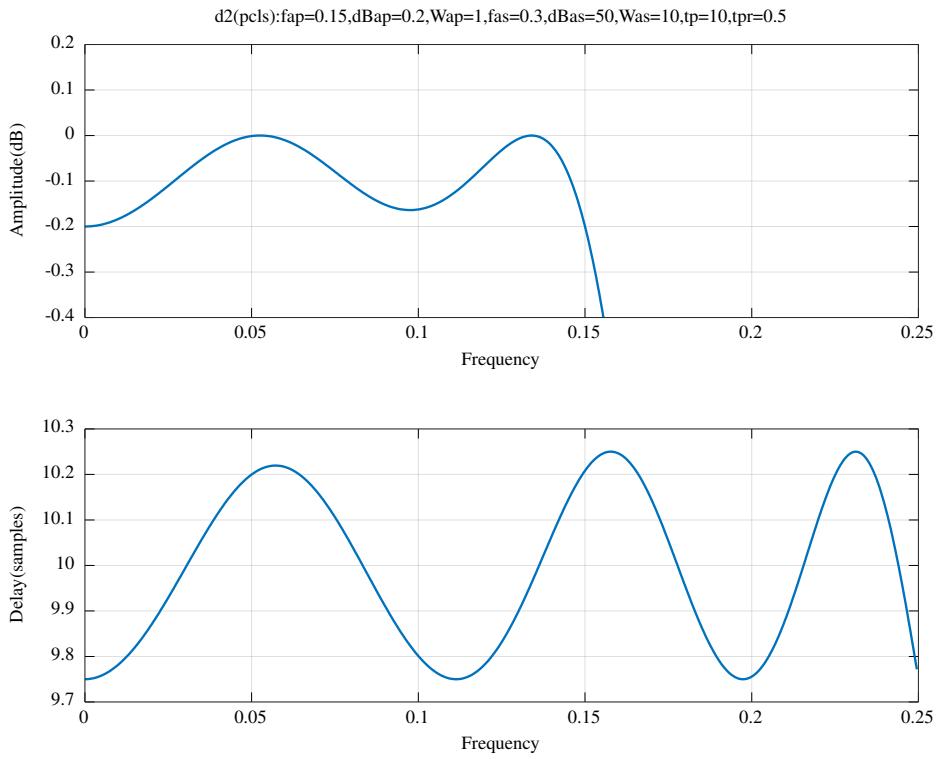


Figure 11.5: Deczky Example 3 PCLS : Passband response after SOCP optimisation, alternative specification

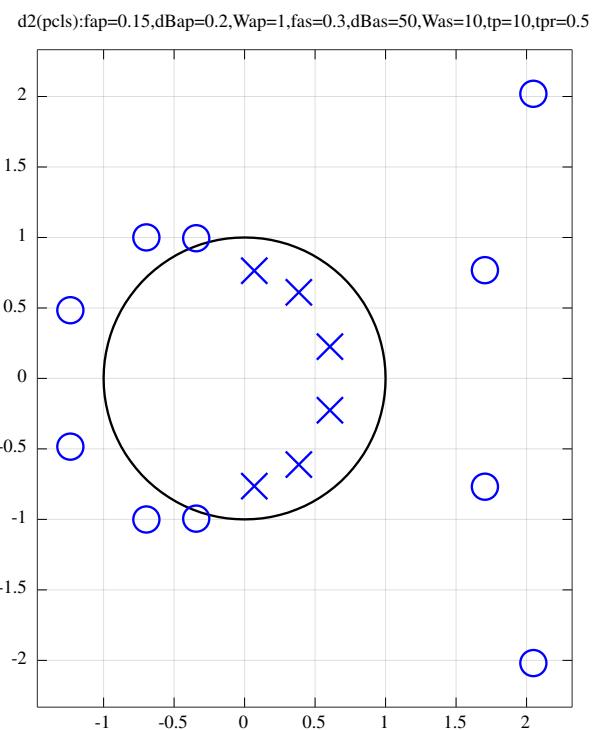


Figure 11.6: Deczky Example 3 PCLS : Pole-zero plot after SOCP optimisation, alternative specification

```

Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [ 0.0006905613, ...
        1.3275431889, 1.2194863172, 1.0520555188, 2.8760541710, ...
        1.8697000593, ...
        2.7683046481, 2.1792330150, 1.9029266805, 0.7780558185, ...
        0.4233389601, ...
        0.7673693479, 0.7218819024, 0.6460293455, ...
        1.4807046578, 1.0084810310, 0.3570842436 ]';

```

In Section 10.1.1, I expressed the filter design problem as a *sequential quadratic programming*, or SQP, problem:

$$\text{minimise } q(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{a}^T \mathbf{x} + \beta$$

where \mathbf{Q} is positive definite and symmetric and linear constraints are neglected for clarity. This SQP optimisation problem can be converted to an SOCP problem by rearranging $q(\mathbf{x})$:

$$q(\mathbf{x}) = \frac{1}{2} \|\mathbf{Q}^{\frac{1}{2}} \mathbf{x} + \mathbf{Q}^{-\frac{1}{2}} \mathbf{a}\|^2 + \beta - \frac{1}{2} \mathbf{a}^T \mathbf{Q}^{-1} \mathbf{a}$$

See Alizadeh and Goldfarb [25, Section 2.1] or Antoniou and Lu [2, Section 14.7.2].

The Octave script *deczky3_socp_bfgs_test.m* implements the design of Deczky's Example 3, with the SeDuMi SOCP solver, as in Section 11.4, but with MMSE optimisation of the weighted response error, $\mathcal{E}(\mathbf{x})$, represented as $\|\mathbf{Q}^{\frac{1}{2}} \mathbf{x} + \mathbf{Q}^{-\frac{1}{2}} \nabla_x \mathcal{E}(\mathbf{x})\|$, where \mathbf{Q} is initialised with the diagonal of $\nabla_{xx}^2 \mathcal{E}(\mathbf{x})$ and updated with the BFGS Hessian matrix update algorithm shown in Appendix F.6.1.

The filter specification is

```

U=0 % Number of real zeros
V=0 % Number of real poles
M=10 % Number of complex zeros
Q=6 % Number of complex poles
R=1 % Denominator polynomial decimation factor
n=1000 % Frequency points across the band
tol=1e-06 % Tolerance on relative coefficient update size
fap=0.15 % Pass band amplitude response edge
dBap=0.1 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.25 % Pass band group delay response edge
tp=10 % Nominal filter group delay
tpr=0.08 % Pass band group delay peak-to-peak ripple
Wtp=0.2 % Pass band group delay weight
fas=0.3 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=10 % Stop band amplitude weight

```

The result of the PCLS SOCP pass in *deczky3_socp_bfgs_test.m* is shown in Figure 11.7 with pass-band details shown in Figure 11.8 and pole-zero plot shown in Figure 11.9.

The estimate of the SOCP PCLS optimised filter vector is, in the gain, zeros and poles form of Equation 10.2:

```

Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [ 0.0028603102, ...
        1.1326277305, 1.1153480379, 1.1339469768, 1.8249200760, ...
        2.1304863117, ...
        2.0494174809, 2.7293066409, 1.8375913246, 0.7412308698, ...
        0.2300938399, ...
        0.7101803596, 0.6426477892, 0.5590032038, ...
        1.4478415971, 1.0470003236, 0.3514326072 ]';

```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

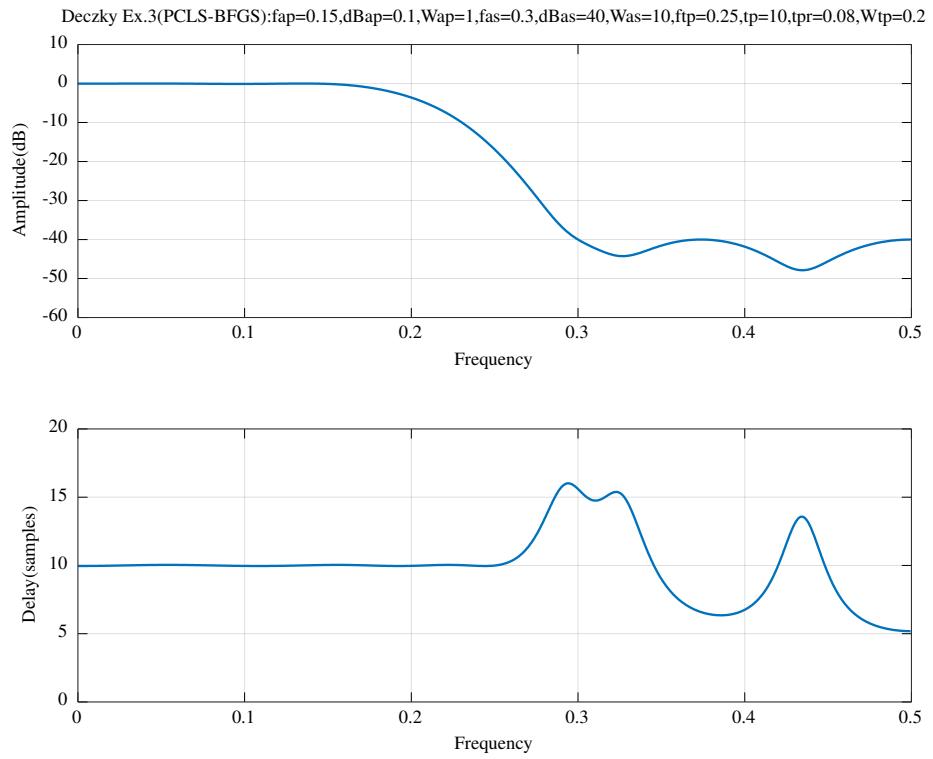


Figure 11.7: Deczky Example 3 PCLS : Response after SOCP optimisation with BFGS update of the Hessian

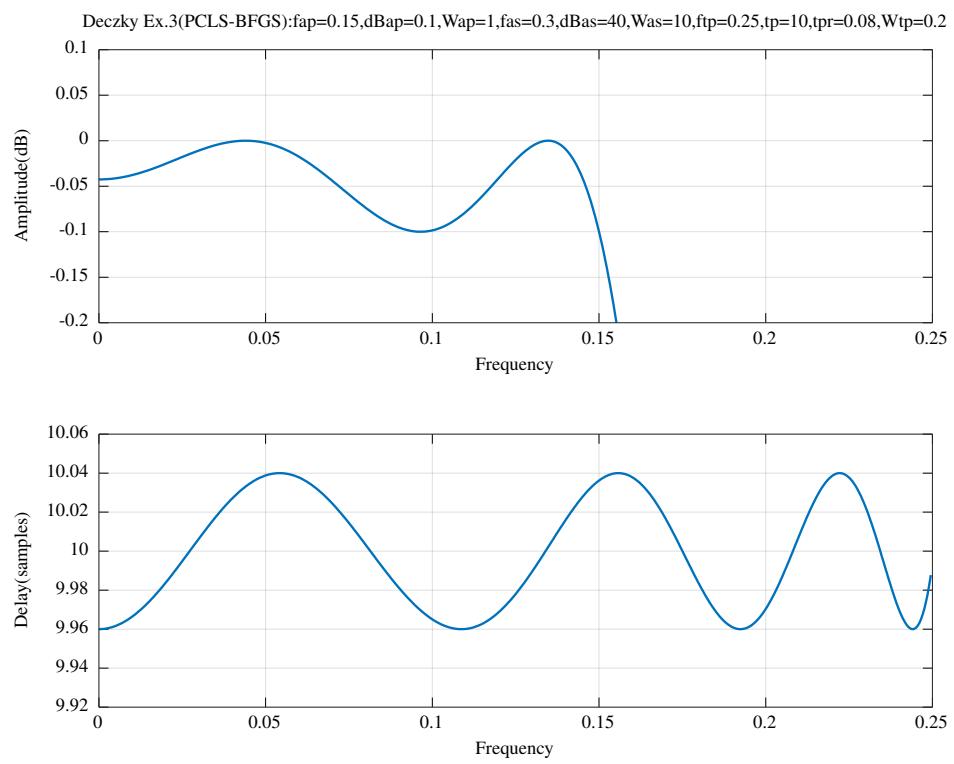


Figure 11.8: Deczky Example 3 PCLS : Passband response after SOCP optimisation with BFGS update of the Hessian

Deczky Ex.3(PCLS-BFGS):fap=0.15,dBap=0.1,Wap=1,fas=0.3,dBas=40,Was=10,ftp=0.25,tp=10,tpr=0.08,Wtp=0.2

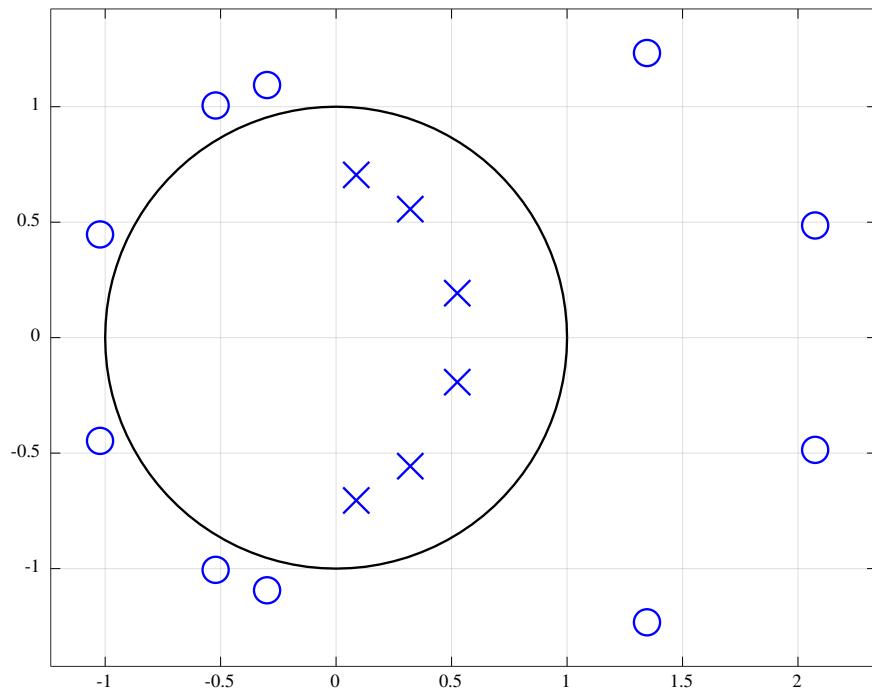


Figure 11.9: Deczky Example 3 PCLS : Pole-zero plot after SOCP optimisation with BFGS update of the Hessian

```
N2 = [ 0.0028603102, -0.0090270509, 0.0046356885, 0.0042885520, ...
       0.0125876512, -0.0167148529, -0.0287899486, 0.0082897144, ...
       0.0946954301, 0.1063564411, 0.0887235713 ]';
```

and

```
D2 = [ 1.0000000000, -1.8667421079, 2.1994796790, -1.7319711422, ...
       0.9458081436, -0.3424437713, 0.0650896512 ]';
```

11.5 SOCP MMSE design of a bandpass R=2 filter expressed in gain-zero-pole format with *SeDuMi*

The Octave script *iir_socp_slb_bandpass_test.m* uses the *SeDuMi* SOCP solver to design an $R = 2$ bandpass filter similar to that designed with the SQP solver in Section 10.2.6. After some experimentation, the filter specification is:

```
n=500 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
fasl=0.05 % Stop band amplitude response lower edge
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
fasu=0.25 % Stop band amplitude response upper edge
dBap=1 % Pass band amplitude peak-to-peak ripple
dBas=28 % Stop band amplitude peak-to-peak ripple
Wasl=2 % Lower stop band weight
Wap=1 % Pass band weight
Wasu=2 % Upper stop band weight
ftpl=0.09 % Pass band group delay response lower edge
ftpnu=0.21 % Pass band group delay response upper edge
tp=16 % Nominal filter group delay
tpr=0.04 % Pass band group delay peak-to-peak ripple
Wtp=0.25 % Pass band group delay weight
U=2 % Number of real zeros
V=0 % Number of real poles
M=18 % Number of complex zeros
Q=10 % Number of complex poles
R=2 % Denominator polynomial decimation factor
```

The *SeDuMi* solver seems to require the initial filter to be a closer approximation than that required by the SQP solver. In this case, the initial filter is calculated by the Octave function *tarczynski_bandpass_test.m*. The response of the initial filter is shown in Figure 11.10. After PCLS SOCP optimisation, the response of the resulting filter is shown in Figure 11.11 with pass-band

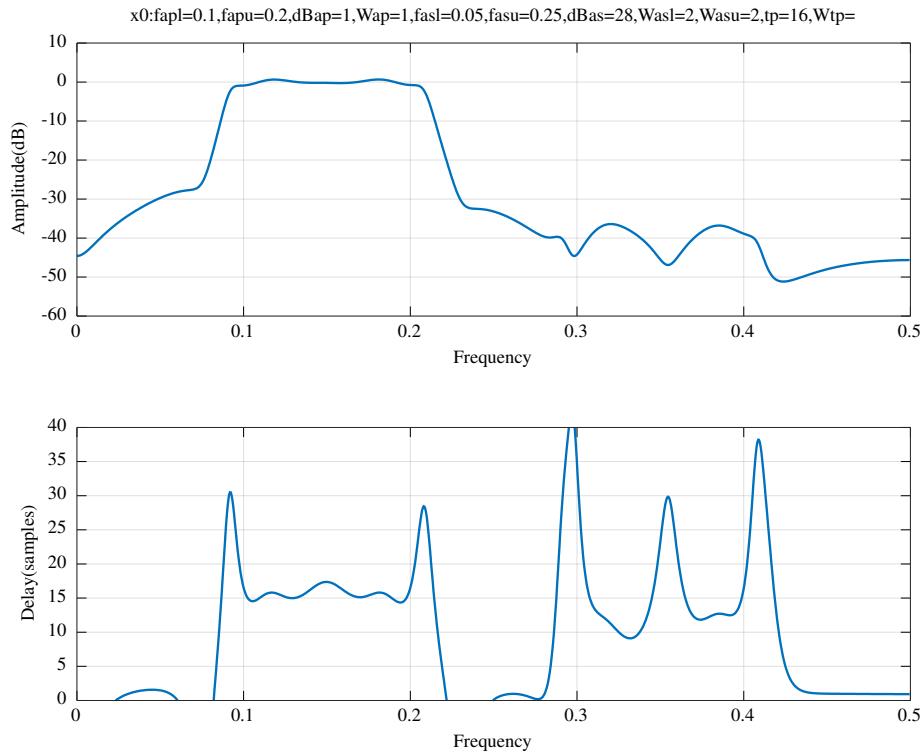


Figure 11.10: IIR band-pass R=2 decimation filter : response of the initial filter calculated with the WISE method of *Tarczynski et al.*

detail shown in Figure 11.12. The corresponding pole-zero plot is shown in Figure 11.13. The estimate of the optimised filter vector is, in the gain, zeros and poles form of Equation 10.2:

```

Ud1=2,Vd1=0,Md1=18,Qd1=10,Rd1=2
d1 = [ 0.0002963101, ...
        2.6259837779, 1.3069430770, ...
        1.2931230772, 0.9853880141, 0.9901079752, 1.5513139750, ...
        1.0084464780, 1.4806163982, 1.3560239792, 1.5228531555, ...
        1.3263961794, ...
        0.9376529564, 0.2649459226, 1.6038066463, 2.1832519229, ...
        1.8531165008, 2.6291165346, 2.6670600922, 2.3246853748, ...
        2.2743635581, ...
        0.6955309527, 0.6211003619, 0.6158345184, 0.5952781504, ...
        0.1621688198, ...
        1.0025152827, 1.4288446716, 2.8466776361, 2.3406230402, ...
        2.0739458129 ]';

```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function `x2tf`) are, respectively:

```

N1 = [ 0.0002963101, 0.0011377763, 0.0010698268, -0.0049426838, ...
        -0.0180166331, -0.0271451616, -0.0154709779, 0.0085498729, ...
        0.0229347243, 0.0257463910, 0.0566709789, 0.1066044130, ...
        0.1229687737, 0.0357599263, -0.0967412347, -0.1840345697, ...
        -0.1172577290, 0.0136030716, 0.1259380335, 0.1110247957, ...
        0.0651541550 ]';

```

and

```

D1 = [ 1.0000000000, 0.0000000000, 1.2391372950, 0.0000000000, ...
        1.0515123142, 0.0000000000, 0.9488660137, 0.0000000000, ...
        0.7517197900, 0.0000000000, 0.4592145110, 0.0000000000, ...
        0.2497964204, 0.0000000000, 0.1235487230, 0.0000000000, ...
        0.0433195404, 0.0000000000, 0.0061929889, 0.0000000000, ...
        0.0006595673 ]';

```

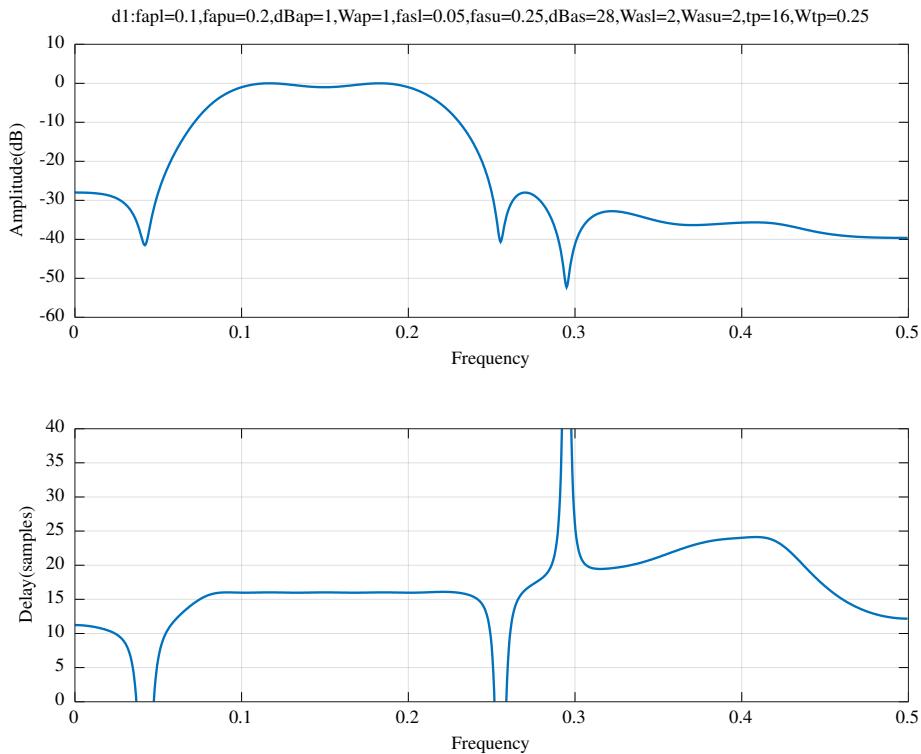


Figure 11.11: IIR band-pass $R=2$ decimation filter with delay $tp=16$ samples : response of the filter after PCLS SOCP optimisation

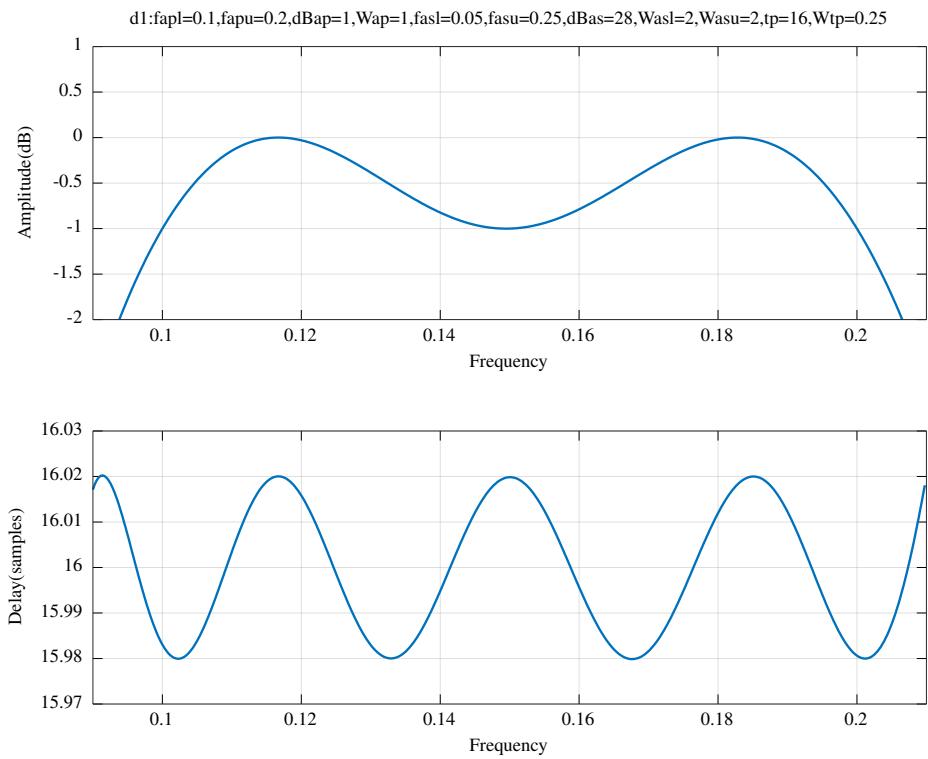


Figure 11.12: IIR band-pass R=2 decimation filter with delay tp=16 samples : passband response of the filter after PCLS SOCP optimisation

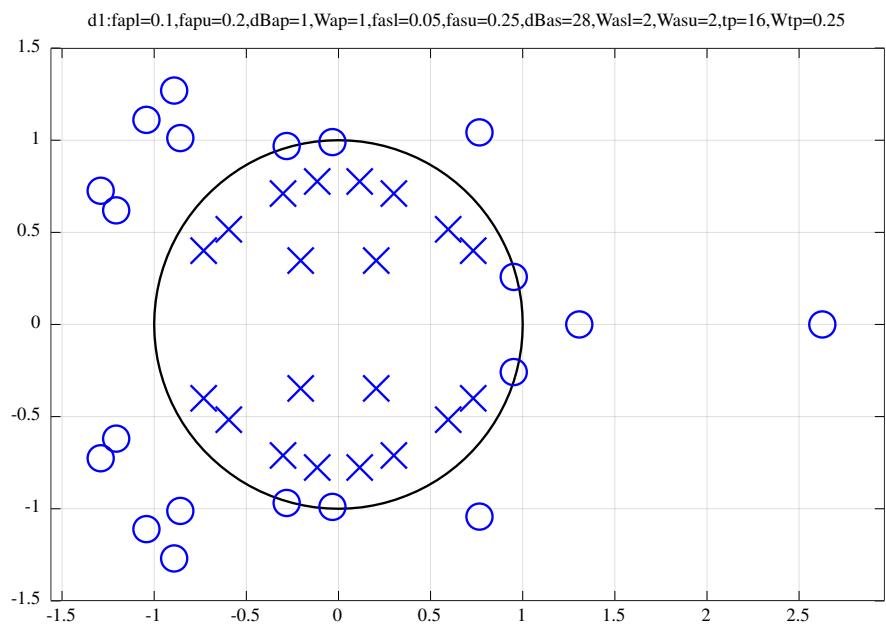


Figure 11.13: IIR band-pass R=2 decimation filter with delay tp=16 samples : pole-zero plot of the filter after PCLS SOCP optimisation

Chapter 12

IIR filter design with a pre-defined structure

In this chapter I describe the design of IIR filters with a predefined structure.

12.1 Design of an IIR filter composed of second-order sections

In this section I consider the MMSE design of an IIR filter consisting of a cascade of second-order sections (with one first-order section if the transfer function polynomial order is odd). The stability of the IIR filter is ensured by a linear constraint on the coefficients of the denominator second-order sections. (See *Lu and Hinamoto* [94]).

12.1.1 Linear constraints on the stability of second-order filter sections

A digital filter is stable if its poles (the zeros of the denominator polynomial of the filter transfer function) lie within the unit circle in the z-plane, $|z| < 1$. Suppose the filter denominator polynomial is decomposed into one first order section, $z + d_0$, and second order sections of the form $z^2 + d_1 z + d_2$. The pole location of the first order section is constrained by

$$\begin{aligned} d_0 &> -1 \\ -d_0 &> -1 \end{aligned}$$

If the roots of the second order polynomial lie within the unit circle $|z| < 1$ then

$$\left| \frac{-d_1 \pm \sqrt{d_1^2 - 4d_2}}{2} \right| < 1$$

If the roots are complex then $d_1^2 \leq 4d_2$ and

$$\frac{d_1^2 + 4d_2 - d_1^2}{4} < 1$$

so $-d_2 > -1$.

If the roots are real then $d_1^2 \geq 4d_2$ and

$$\begin{aligned} \left| \frac{-d_1 \pm \sqrt{d_1^2 - 4d_2}}{2} \right| &< 1 \\ \pm \sqrt{d_1^2 - 4d_2} &< 2 \pm d_1 \end{aligned}$$

Squaring both sides

$$\pm d_1 + d_2 > -1$$

If a margin $0 < \tau \ll 1$ is applied to the first and second order sections then:

$$d_0 > -1 + \tau$$

$$\begin{aligned} -d_0 &> -1 + \tau \\ d_1 + d_2 &> -1 + \tau \\ -d_1 + d_2 &> -1 + \tau \\ -d_2 &> -1 + \tau \end{aligned}$$

Suppose the denominator polynomial has odd order $2L + 1$. Define the coefficients of the corresponding first and second order sections and the stability constraint matrixes as

$$\begin{aligned} \mathbf{d} &= \begin{bmatrix} d_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_L \end{bmatrix} & \text{where } \mathbf{d}_i = \begin{bmatrix} d_{i1} \\ d_{i2} \end{bmatrix} \\ \mathbf{C} &= \begin{bmatrix} c_1 & & \mathbf{0} \\ & c_2 & \\ & & \ddots \\ \mathbf{0} & & c_2 \end{bmatrix} & \text{where } c_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ and } c_2 = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 0 & 1 \end{bmatrix} \\ \mathbf{e} &= \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_2 \end{bmatrix} & \text{where } e_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } e_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \end{aligned}$$

The stability constraint on the filter denominator polynomial is

$$\mathbf{C}\mathbf{d} + (1 - \tau)\mathbf{e} \geq \mathbf{0}$$

Suppose the coefficient vector, \mathbf{d} , is perturbed by $\boldsymbol{\delta}$. Then the stability constraint becomes

$$\mathbf{C}\boldsymbol{\delta} + \mathbf{h} \geq \mathbf{0}$$

where $\mathbf{h} = \mathbf{C}\mathbf{d} + (1 - \tau)\mathbf{e}$.

12.1.2 Design of an IIR filter composed of second order sections with *SeDuMi*

For the IIR filter transfer function

$$H(z) = \frac{a(z)}{d(z)}$$

where

$$a(z) = \sum_{i=0}^n a_i z^{-i}$$

and $d(z)$ is a polynomial of order r expressed as a product of second order sections (with one first order section if r is odd)

$$d(z) = \begin{cases} (1 + d_0 z^{-1}) \prod_{i=1}^{\frac{r-1}{2}} (1 + d_{i1} z^{-1} + d_{i2} z^{-2}), & \text{if } r \text{ odd} \\ \prod_{i=1}^{\frac{r}{2}} (1 + d_{i1} z^{-1} + d_{i2} z^{-2}), & \text{if } r \text{ even} \end{cases}$$

Define the two filter coefficient vectors as

$$\mathbf{a} = [a_0 \ a_1 \ \dots \ a_n]^T$$

and

$$\mathbf{d} = \begin{bmatrix} d_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_L \end{bmatrix}$$

$$\begin{aligned}\mathbf{d}_i &= \begin{bmatrix} d_{i1} \\ d_{i2} \end{bmatrix}, \quad \text{for } 1 \leq i \leq L \\ L &= \begin{cases} \frac{r-1}{2}, & \text{if } r \text{ odd} \\ \frac{r}{2}, & \text{if } r \text{ even} \end{cases}\end{aligned}$$

The frequency response of the filter is

$$H(\omega) = \frac{\mathbf{a}^T \mathbf{v}(\omega)}{d(\omega)}$$

where

$$\begin{aligned}\mathbf{v}(\omega) &= \mathbf{c}(\omega) - \imath \mathbf{s}(\omega) \\ \mathbf{c}(\omega) &= [1 \dots \cos n\omega]^T \\ \mathbf{s}(\omega) &= [0 \dots -\sin n\omega]^T \\ v_1(\omega) &= \cos \omega - \imath \sin \omega \\ \mathbf{v}_2(\omega) &= \begin{bmatrix} \cos \omega \\ \cos 2\omega \end{bmatrix} - \imath \begin{bmatrix} \sin \omega \\ \sin 2\omega \end{bmatrix}\end{aligned}$$

and

$$d(\omega) = \begin{cases} [1 + d_0 v_1(\omega)] \prod_{i=1}^L [1 + \mathbf{d}_i^T \mathbf{v}_2(\omega)], & \text{if } r \text{ odd} \\ \prod_{i=1}^L [1 + \mathbf{d}_i^T \mathbf{v}_2(\omega)], & \text{if } r \text{ even} \end{cases}$$

The gradients of $H(\omega)$ with respect to the coefficients are

$$\begin{aligned}\frac{\partial H(\omega)}{\partial \mathbf{a}} &= \frac{\mathbf{v}(\omega)}{d(\omega)} \\ \frac{\partial H(\omega)}{\partial d_0} &= -H(\omega) \frac{v_1(\omega)}{1 + d_0 v_1(\omega)} \\ \frac{\partial H(\omega)}{\partial \mathbf{d}_i} &= -H(\omega) \frac{\mathbf{v}_2(\omega)}{1 + \mathbf{d}_i^T \mathbf{v}_2(\omega)} \quad \text{for } 1 \leq i \leq L\end{aligned}$$

In this case we optimise the complex frequency response (gain and delay). For each of the response frequencies the format for solution by *SeDuMi* is

$$\begin{aligned}\mathbf{A}_i^T &= W(\omega_i) \begin{bmatrix} 0 & \operatorname{Re} \nabla_x H(\omega_i)^T \\ 0 & \operatorname{Im} \nabla_x H(\omega_i)^T \end{bmatrix} \\ \mathbf{b}_i^T &= [1 \ 0] \\ \mathbf{c}_i &= W(\omega_i) \begin{bmatrix} 0 & \operatorname{Re}[H(\omega_i) - H_d(\omega_i)] \\ 0 & \operatorname{Im}[H(\omega_i) - H_d(\omega_i)] \end{bmatrix} \\ d_i &= 0\end{aligned}$$

where x represents the vector of coefficients and the desired low pass filter frequency response is

$$H_d(\omega) = \begin{cases} e^{-\imath \omega t_d} & \text{if } 0 \leq \omega \leq \omega_{pass} \\ 10^{-\frac{dB_{stop}}{20}} & \text{if } \omega_{pass} < \omega \end{cases}$$

The Octave script *lowpass2ndOrderCascade_socp_test.m* calls the Octave function *lowpass2ndOrderCascade_socp* to implement MMSE SOCP design of a low pass filter similar to that of Deczky's Example 3:

```
tol=1e-06 % Tolerance on coefficient update vector
mn=10 % Numerator order (mn+1 coefficients)
mr=6 % Denominator order (mr coefficients)
tau=0.1 % Second order section stability parameter
n=400 % Number of frequency points
td=10 % Pass band group delay
```

```

fpass= 0.15 % Pass band edge
Wpass=1 % Pass band weight
fstop= 0.3 % Stop band edge
Wstop=100 % Stop band weight
dBstop=80 % Stop band attenuation

```

The filter coefficients are initialised with the “IPZS-1” set. After SOCP MMSE optimisation, the numerator and denominator polynomials are respectively

```

a = [ -0.0021092647, 0.0004069329, 0.0076883394, 0.0051931733, ...
-0.0115231077, -0.0210543663, 0.0019486082, 0.0417050319, ...
0.0578424345, 0.0390044901, 0.0123776721 ]';

```

and

```

d = [ 1.0000000000, -2.4277939779, 3.0446749932, -2.3351332887, ...
1.1383562667, -0.3371599411, 0.0468622432 ]';

```

The overall frequency response of the MMSE SOCP design is shown in Figure 12.1 with pass-band details shown in Figure 12.2 and pole-zero plot shown in Figure 12.3.

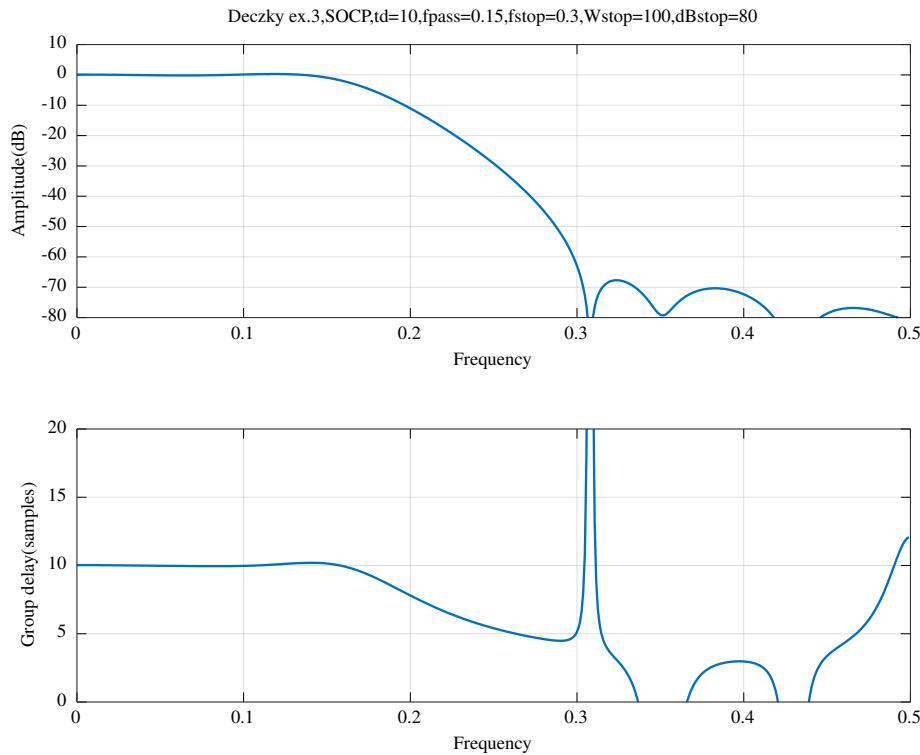


Figure 12.1: Deczky Example 3 : Response with 2nd order sections and MMSE SOCP optimisation

Additionally, the Octave function *lowpass2ndOrderCascade_socp.m* optimises only the squared-magnitude response and ignores the group delay response. The resulting overall magnitude and delay responses are shown in Figure 12.4, the passband responses are shown in Figure 12.5 and the pole-zero plot is shown in Figure 12.6.

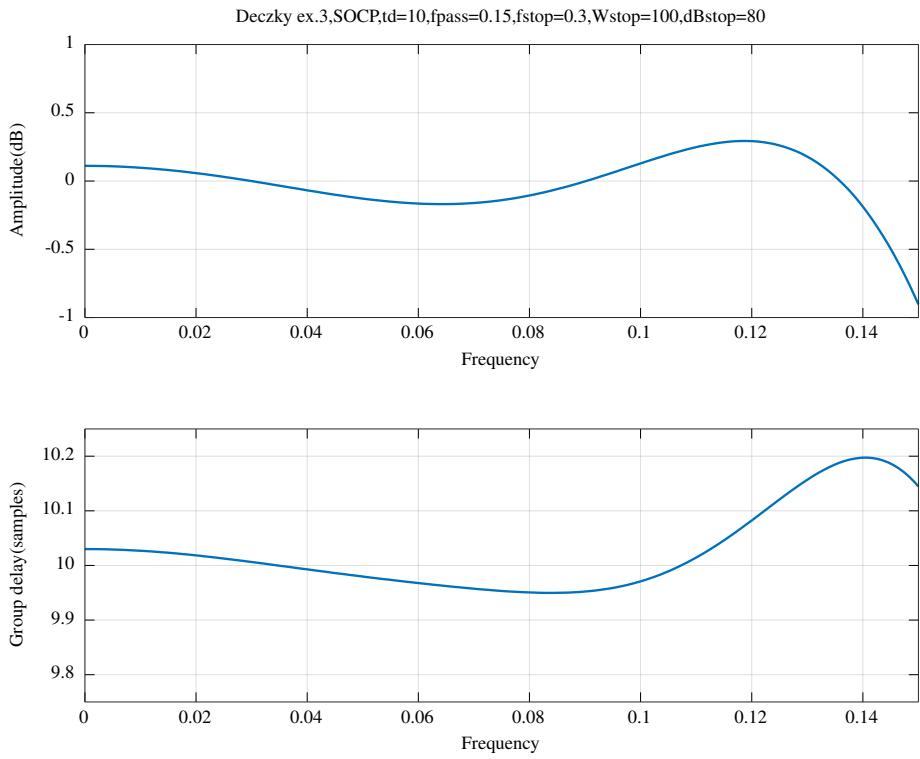


Figure 12.2: Deczky Example 3 : Passband response with 2nd order sections and MMSE SOCP optimisation

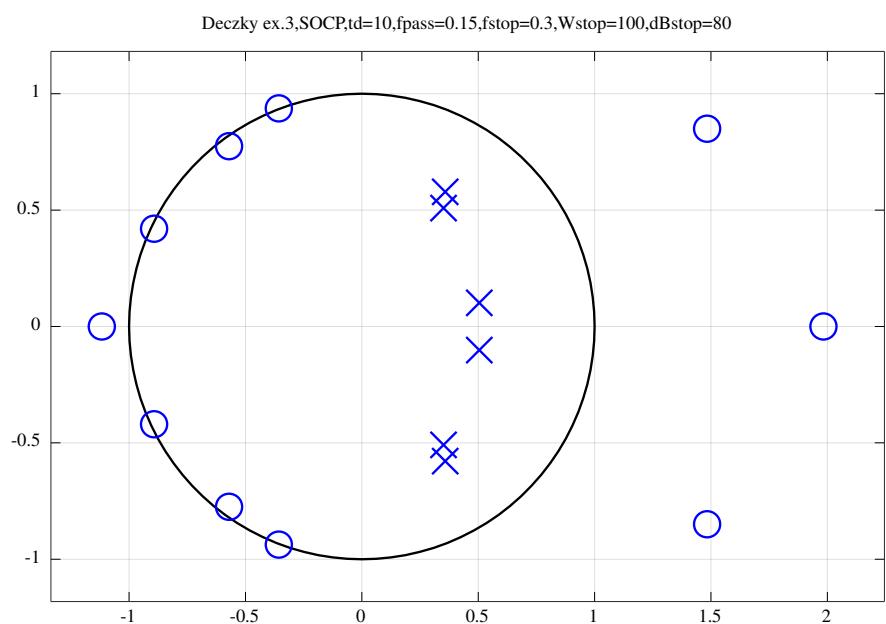


Figure 12.3: Deczky Example 3 : Pole-zero plot with 2nd order sections and MMSE SOCP optimisation

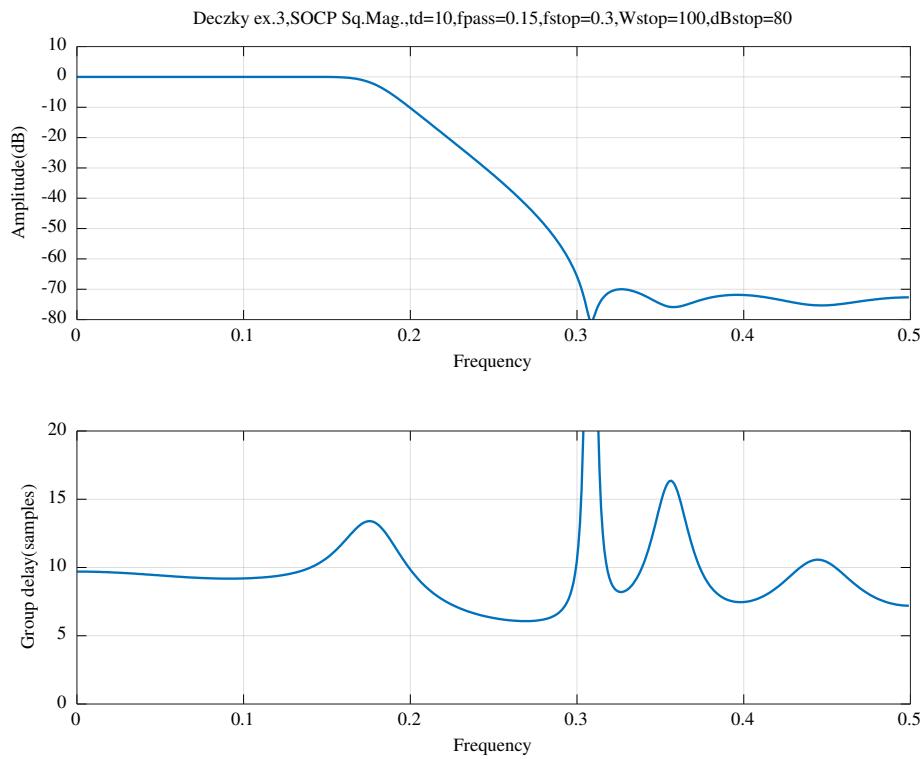


Figure 12.4: Deczky Example 3 : Response with 2nd order sections and MMSE SOCP optimisation of the squared-magnitude response

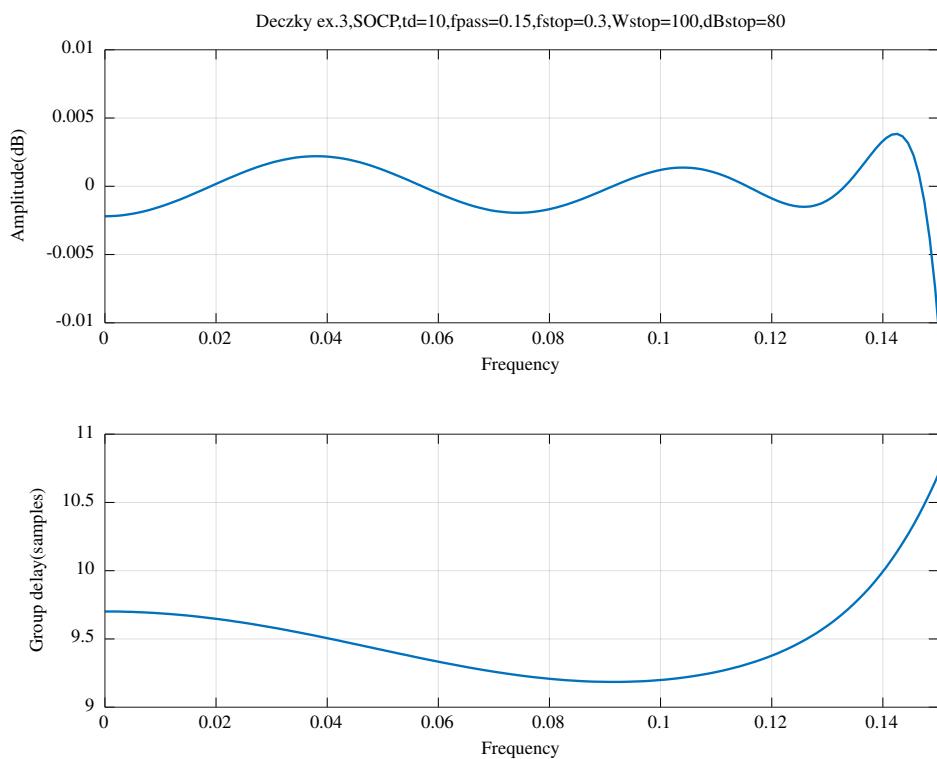


Figure 12.5: Deczky Example 3 : Passband response with 2nd order sections and MMSE SOCP optimisation of the squared-magnitude response

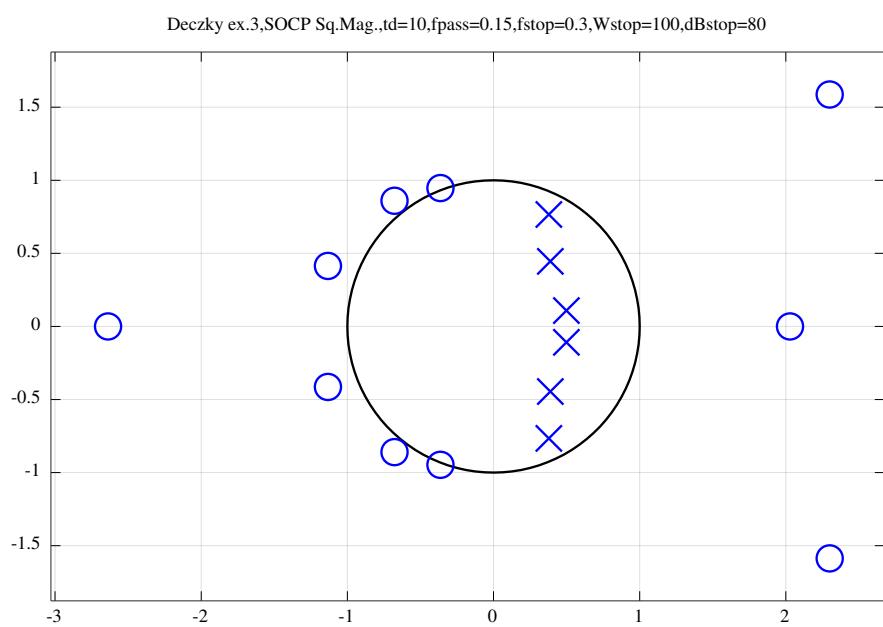


Figure 12.6: Deczky Example 3 : Pole-zero plot with 2nd order sections and MMSE SOCP optimisation of the squared-magnitude response

12.1.3 Some notes on the design of an IIR filter composed of second order sections with *SeDuMi*

Some notes on SOCP optimisation of a filter composed of a cascade of second order sections with *SeDuMi*:

- when running SeDuMi under Octave with the default options, SeDuMi sometimes fails due to numerical problems. This indicates that the optimisation problem is not feasible with the current constraints.
- a better response is obtained by using a single constraint on the summed response error (MMSE) rather than a large number of separate frequency response constraints.
- the response is sensitive to the stop band weight.
- the passband response is improved if the desired stop band amplitude response is set to a small non-zero value.
- I did not find a useful filter when attempting to optimise the complex response in the pass band simultaneously with the squared-magnitude response in the stop band.
- a second order section may have complex conjugate roots or two real roots. In the gain-zero-pole format the total number of real and complex conjugate roots is specified in advance.

12.2 Design of an IIR filter as the sum of two all-pass filters

In this section I consider the design of a low-pass IIR filter that is the sum of two parallel all-pass filters, $A(z)$ and $B(z)$

$$H(z) = \frac{A(z) + B(z)}{2} \quad (12.1)$$

Vaidyanathan *et al.* [72] demonstrate that the “classical” odd-order lowpass digital filter approximations can be implemented as the sum of two allpass filters. The resulting filters have low sensitivity to coefficient quantisation.

12.2.1 Design of an IIR filter as the sum of two all-pass filters each composed of second-order sections

Following Lu and Hinamoto [94], the transfer functions of the $A(z)$ and $B(z)$ filters are expressed as the product of second-order sections. As shown in Section 12.1.1, the second-order filter sections permit a simple linear stability constraint on the coefficients of each section.

Assume that the order of $H(z)$ is odd and that that

$$\begin{aligned} a(z) &= (1 + a_0 z^{-1}) \prod_{k=1}^{\frac{m-1}{2}} 1 + a_{k1} z^{-1} + a_{k2} z^{-2} \\ A(z) &= z^{-m} \frac{a(z^{-1})}{a(z)} \end{aligned}$$

and

$$\begin{aligned} b(z) &= \prod_{k=1}^{\frac{n}{2}} 1 + b_{k1} z^{-1} + b_{k2} z^{-2} \\ B(z) &= z^{-n} \frac{b(z^{-1})}{b(z)} \end{aligned}$$

Here the coefficients a_{k1} etc. are real, m is odd, n is even and the order of H is $p = n + m$. In the following, references to $A(z)$ apply to $B(z)$ as appropriate. The frequency response of $A(z)$ is:

$$\begin{aligned} A(\omega) &= e^{-im\omega} \frac{a(-\omega)}{a(\omega)} \\ a(\omega) &= (1 + a_0 v_1) \prod_{k=1}^{\frac{m-1}{2}} 1 + \mathbf{v}_2 \mathbf{a}_k \end{aligned}$$

where $\mathbf{a}_k = \begin{bmatrix} a_{k1} \\ a_{k2} \end{bmatrix}$, $v_1 = \cos \omega - i \sin \omega$, $\mathbf{v}_2 = [\cos \omega \cos 2\omega] - i [\sin \omega \sin 2\omega]$ and $A(\omega)$ is understood to mean $A(e^{i\omega})$.

The gradients of $A(z)$ with respect to its coefficients a_0 , a_{k1} and a_{k2} are

$$\begin{aligned} \frac{\partial A(z)}{\partial a_0} &= A(z) \frac{z - z^{-1}}{1 + a_0^2 + a_0(z + z^{-1})} \\ \frac{\partial A(z)}{\partial a_{k1}} &= A(z) \frac{(z - z^{-1})(1 - a_{k2})}{1 + a_{k1}^2 + a_{k2}^2 + a_{k1}(1 + a_{k2})(z + z^{-1}) + a_{k2}(z^2 + z^{-2})} \\ \frac{\partial A(z)}{\partial a_{k2}} &= A(z) \frac{(z - z^{-1})(a_{k1} + z + z^{-1})}{1 + a_{k1}^2 + a_{k2}^2 + a_{k1}(1 + a_{k2})(z + z^{-1}) + a_{k2}(z^2 + z^{-2})} \end{aligned}$$

The Octave function *allpass2ndOrderCascade.m* returns the complex frequency response and gradient of an allpass filter consisting of a cascade of 2nd-order allpass sections (with a single additional first order section if the filter order is odd).

Similarly to Section 11.4, the parallel allpass filter design problem can be expressed in SOCP form as

$$\text{minimise } \epsilon, \beta$$

subject to $\|W(\omega_i) \begin{bmatrix} \operatorname{Re} \nabla H(\mathbf{a}_k, \mathbf{b}_k, \omega_i)^T \\ \operatorname{Im} \nabla H(\mathbf{a}_k, \mathbf{b}_k, \omega_i)^T \end{bmatrix} \boldsymbol{\delta} + W(\omega_i) \begin{bmatrix} \operatorname{Re} H(\mathbf{a}_k, \mathbf{b}_k, \omega_i) - \operatorname{Re} H_d(\omega_i) \\ \operatorname{Im} H(\mathbf{a}_k, \mathbf{b}_k, \omega_i) - \operatorname{Im} H_d(\omega_i) \end{bmatrix}\| \leq \epsilon$

$$\|\boldsymbol{\delta}\| \leq \beta$$

$$C\boldsymbol{\delta} + \mathbf{h} \geq \mathbf{0}$$

where $\boldsymbol{\delta} = \begin{bmatrix} \mathbf{a} - \mathbf{a}_k \\ \mathbf{b} - \mathbf{b}_k \end{bmatrix}$

$$H(\mathbf{a}_k, \mathbf{b}_k, \omega_i) = 0.5 [A(\mathbf{a}_k, \omega_i) + B(\mathbf{b}_k, \omega_i)]$$

$$\nabla H(\mathbf{a}_k, \mathbf{b}_k, \omega_i) = 0.5 [\nabla_a A(\mathbf{a}_k, \omega_i) + \nabla_b B(\mathbf{b}_k, \omega_i)]$$

A quadratic constraint on the squared-magnitude response at the stop-band frequencies is

$$\|\nabla H^2(\mathbf{a}_k, \mathbf{b}_k, \omega_i)^T \boldsymbol{\delta} + H^2(\mathbf{a}_k, \mathbf{b}_k, \omega_i)\| \leq |H_d(\omega_i)|^2$$

where

$$H^2(\mathbf{a}_k, \mathbf{b}_k, \omega_i) = \|H(\mathbf{a}_k, \mathbf{b}_k, \omega_i)\|^2$$

$$= 0.25 [\operatorname{Re} A(\mathbf{a}_k, \omega_i) + \operatorname{Re} B(\mathbf{b}_k, \omega_i)]^2 + 0.25 [\operatorname{Im} A(\mathbf{a}_k, \omega_i) + \operatorname{Im} B(\mathbf{b}_k, \omega_i)]^2$$

$$\nabla H^2(\mathbf{a}_k, \mathbf{b}_k, \omega_i) = 0.5 [\operatorname{Re} A(\mathbf{a}_k, \omega_i) + \operatorname{Re} B(\mathbf{b}_k, \omega_i)] [\operatorname{Re} \nabla_a A(\mathbf{a}_k, \omega_i) + \operatorname{Re} \nabla_b B(\mathbf{b}_k, \omega_i)] + \dots$$

$$0.5 [\operatorname{Im} A(\mathbf{a}_k, \omega_i) + \operatorname{Im} B(\mathbf{b}_k, \omega_i)] [\operatorname{Im} \nabla_a A(\mathbf{a}_k, \omega_i) + \operatorname{Im} \nabla_b B(\mathbf{b}_k, \omega_i)]$$

Design of an IIR filter as the sum of two 2nd order cascade all-pass filters with MMSE optimisation of the complex response

The Octave script *allpass2ndOrderCascade_socp_test.m* calls the Octave function *allpass2ndOrderCascade_socp* to design a low-pass filter composed of two parallel all-pass filters with MMSE optimisation of the complex response of the filter. The filter specification is similar to Deczky's Example 3. The desired low pass filter frequency response is:

$$H_d(\omega) = \begin{cases} e^{-i\omega t_d} & \text{if } 0 \leq \omega \leq \omega_{pass} \\ 0 & \text{if } \omega_{pass} < \omega \end{cases}$$

and the filter specification is

```
tol=1e-06 % Tolerance on coefficient update vector
ma=11 % Order of filter A
mb=12 % Order of filter B
tau=0.001 % Second order section stability parameter
n=500 % Number of frequency points
resp=complex % Flat passband group delay or squared-magnitude
fp= 0.15 % Pass band edge
td=11.5 % Pass band nominal group delay
Wp=2 % Pass band weight
fs= 0.2 % Stop band edge
Ws=20 % Stop band weight
```

The initial allpass filters were designed by the Octave script *tarczynski_allpass2ndOrderCascade_test.m* with *flat_delay = true*. The initial response is shown in Figure 12.7.

After optimisation with the *SeDuMi* SOCP solver the response is shown in Figure 12.8 with passband detail shown in Figure 12.9 and pole-zero plot shown in Figure 12.10. The second-order section coefficients are

```
a1 = [ -0.3886778788, -1.1714215254, 0.5348178883, -0.3155085230, ...
        0.8306967580, 0.9425118218, 0.6632816869, -0.5279545715, ...
        0.6123199111, -0.3999511418, -0.1735918137 ]';
```

and

```
b1 = [ -1.2672786021, 0.4645351038, -0.5300346634, 0.4986290583, ...
        0.9445419509, 0.6640912419, -0.2937208160, -0.1642197261, ...
        -0.3151019717, 0.8341889523, -0.8545522695, 0.8310544334 ]';
```

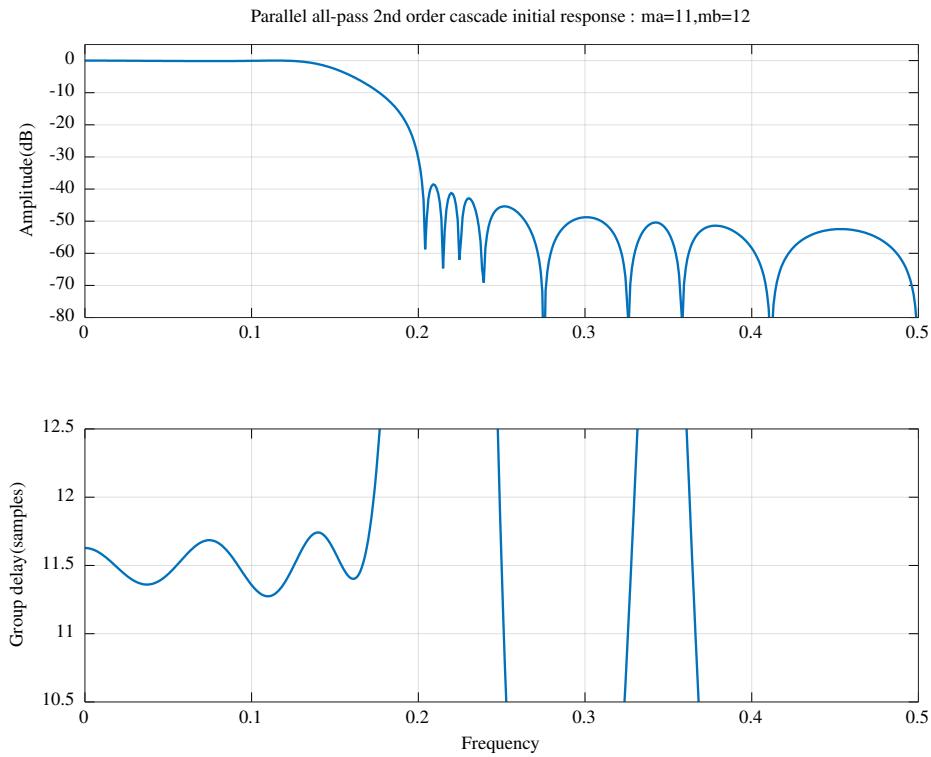


Figure 12.7: Parallel 2nd order cascaded allpass filters : initial response found with the WISE barrier function

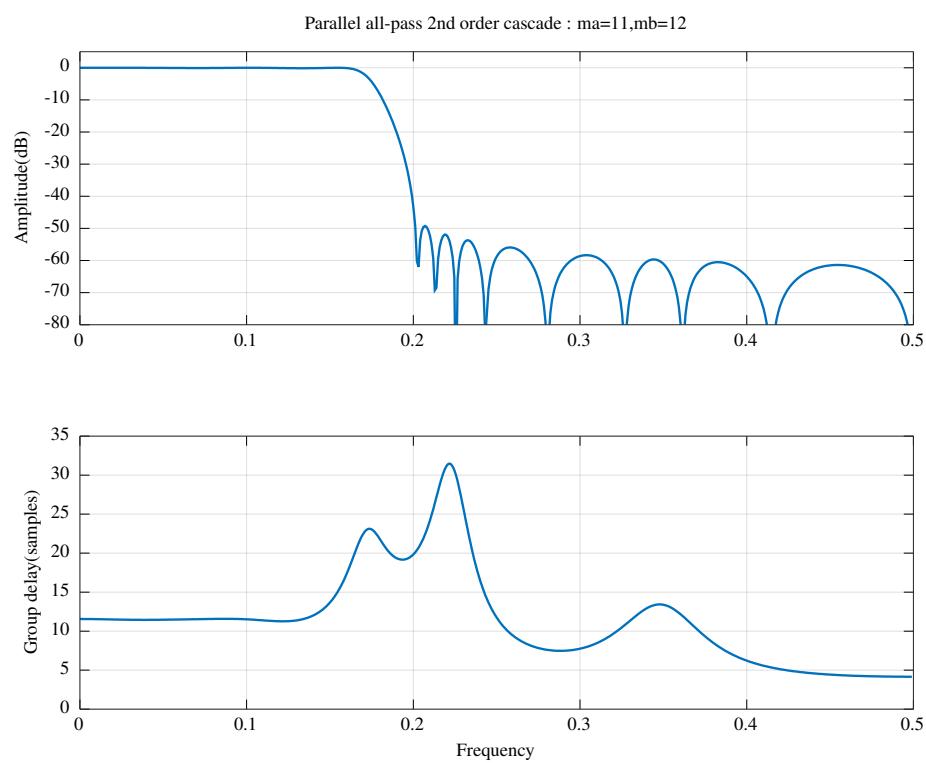


Figure 12.8: Parallel 2nd order cascade allpass filters : response after SOCP optimisation

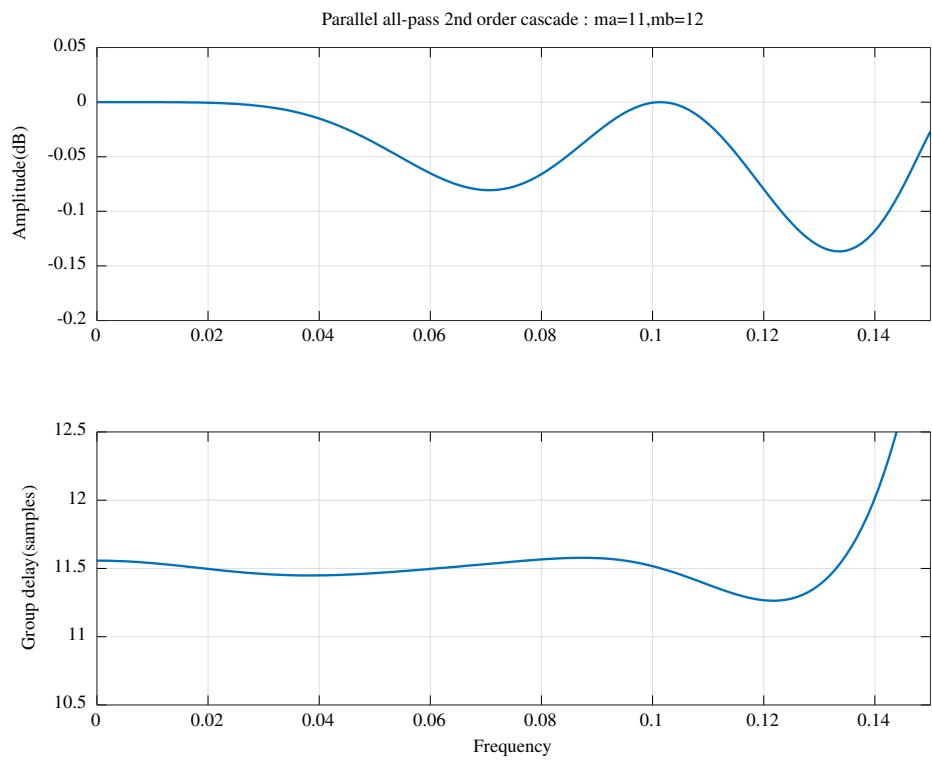


Figure 12.9: Parallel 2nd order cascade allpass filters : pass-band response after SOCP optimisation

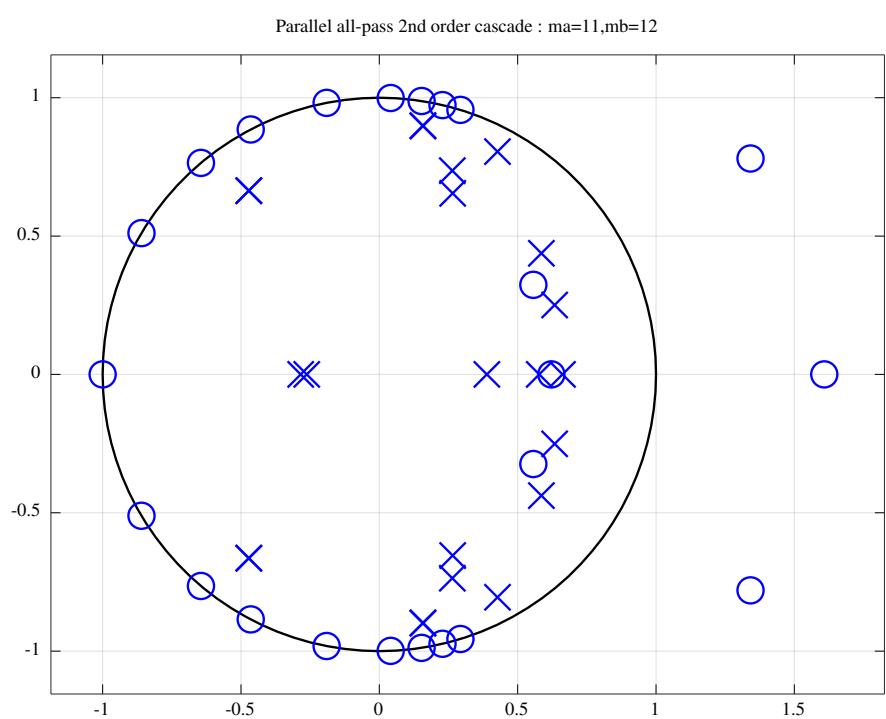


Figure 12.10: Parallel 2nd order cascade allpass filters : pole-zero plot after SOCP optimisation

The corresponding allpass filter denominator polynomials are

```
Da1 = [ 1.0000000000, -1.8610018186, 2.7242533194, -2.7612866728, ...
        2.3493318754, -1.8576319283, 1.3958154086, -0.8765621465, ...
        0.3906071642, -0.0823298781, -0.0277600484, 0.0121742970 ]';
```

and

```
Db1 = [ 1.0000000000, -2.3161463716, 3.9630900469, -4.5609374358, ...
        4.3447740265, -3.6136893943, 2.7522609252, -1.9326538217, ...
        1.0948152350, -0.4101533179, 0.0621345369, 0.0347822087, ...
        -0.0175122965 ]';
```

Design of an IIR filter as the sum of two 2nd order cascade all-pass filters with MMSE optimisation of the squared-magnitude response

The Octave script *allpass2ndOrderCascade_socp_sqmag_test.m* calls the Octave function *allpass2ndOrderCascade_socp* to design a low-pass filter composed of two parallel all-pass filters with MMSE optimisation of the squared-magnitude response of the filter. The filter specification is

```
tol=1e-08 % Tolerance on coefficient update vector
ma=5 % Order of filter A
mb=6 % Order of filter B
tau=0.001 % Second order section stability parameter
n=500 % Number of frequency points
resp=sqmag % Flat passband group delay or squared-magnitude
fp= 0.15 % Pass band edge
Wp=1 % Pass band weight
fs= 0.17 % Stop band edge
Ws=550 % Stop band weight
```

The initial allpass filters were designed by the Octave script *tarczynski_allpass2ndOrderCascade_test.m* with *flat_delay = false*. The initial response is shown in Figure 12.11.

After optimisation with the *SeDuMi* SOCP solver the response is shown in Figure 12.12 and the pole-zero plot is shown in Figure 12.13. The second-order section coefficients are

```
a1 = [ -0.6602263246, -1.2419223879, 0.6645926533, -1.1533561351, ...
        0.9104910243 ]';
```

and

```
b1 = [ -1.2965726505, 0.5078966454, -1.1470496928, 0.9740141482, ...
        -1.1876338035, 0.8110697901 ]';
```

The corresponding allpass filter denominator polynomials are

```
Da1 = [ 1.0000000000, -3.0555048476, 4.5888884186, -3.8828771025, ...
        1.8577340374, -0.3995066764 ]';
```

and

```
Db1 = [ 1.0000000000, -3.6312561468, 6.6823423421, -7.3536669040, ...
        5.0946155638, -2.0843198064, 0.4012350235 ]';
```

For comparison, Figures 12.14 and 12.15 show the response and pole-zero plot of an elliptic filter designed with a similar specification:

```
ma=5,mb=6,fap=0.15,dBap=0.02,fas=0.17,dBas=84
[Nellip,Dellip]=ellip(ma+mb,dBap,dBas,fap*2);
```

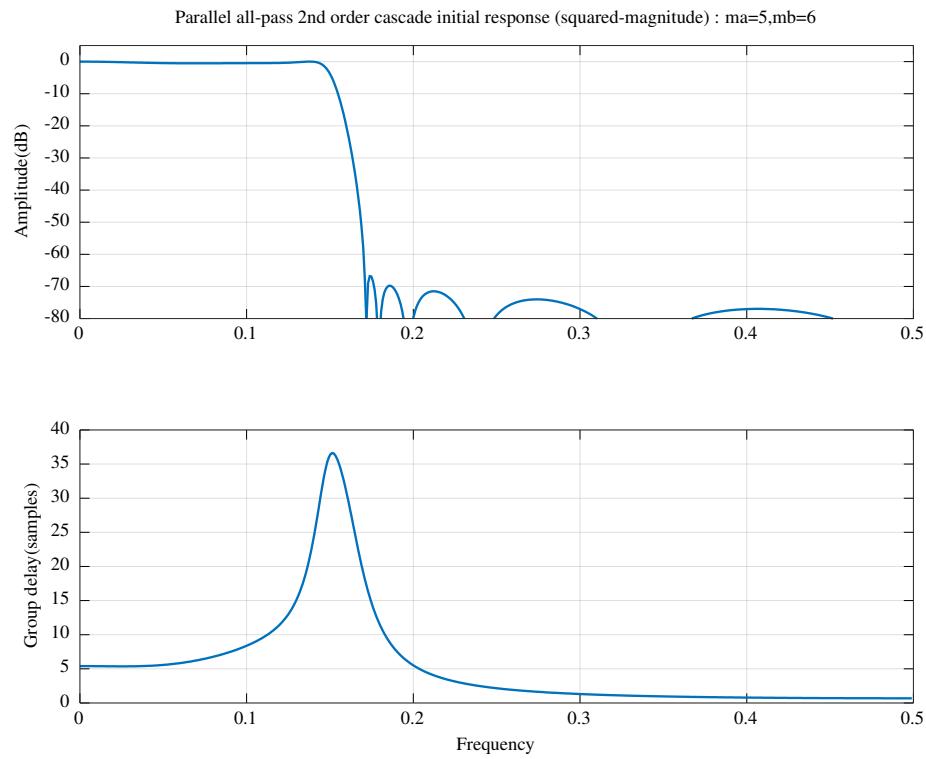


Figure 12.11: Parallel 2nd order allpass cascade filters (squared magnitude) : Initial response found with the WISE barrier function

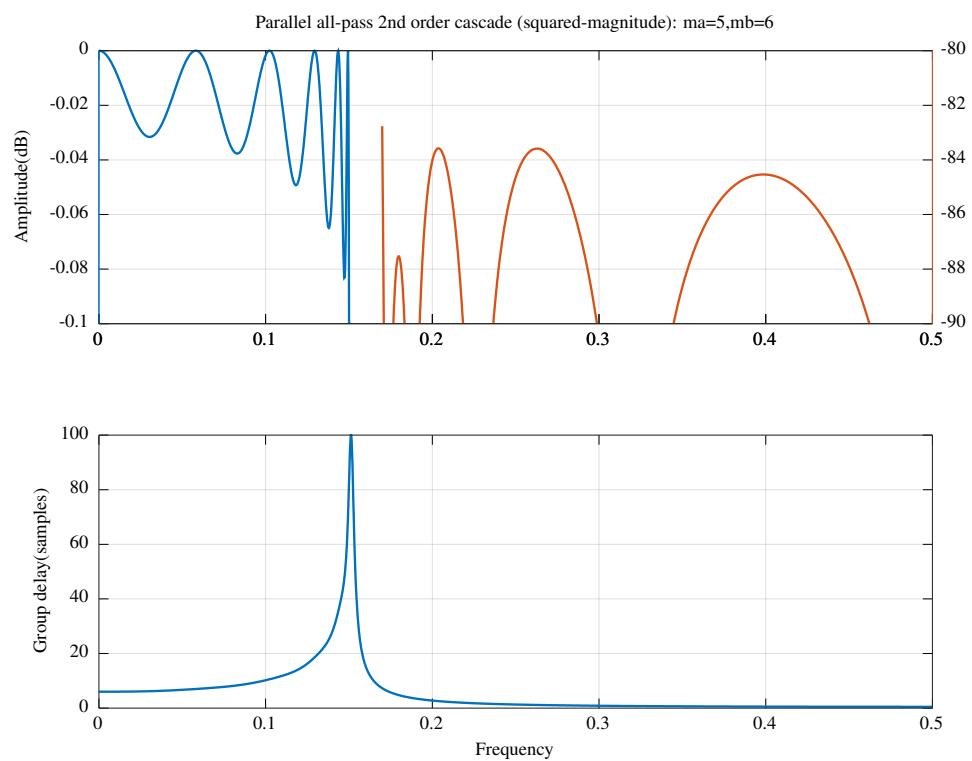


Figure 12.12: Parallel 2nd order cascade allpass filters (squared magnitude) : response after SOCP optimisation

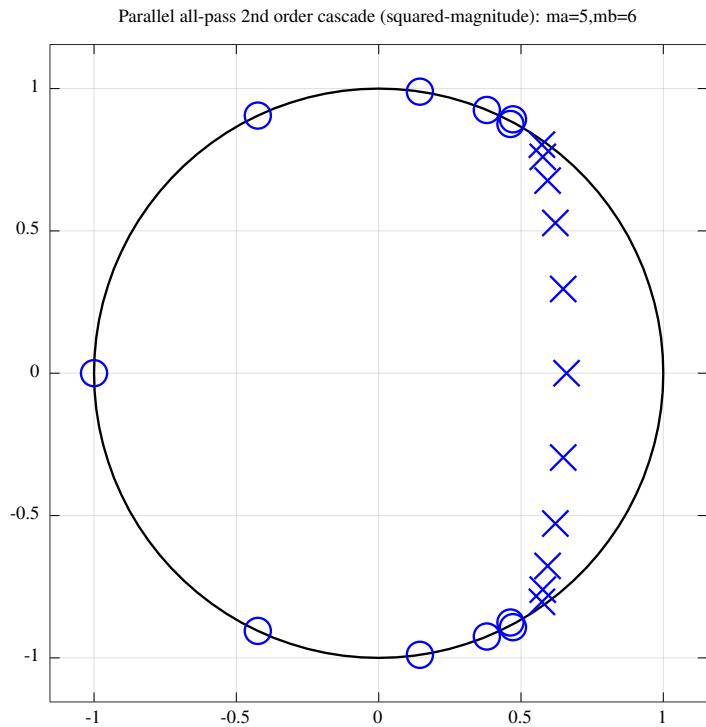


Figure 12.13: Parallel 2nd order cascade allpass filters (squared magnitude) : pole-zero plot after SOCP optimisation

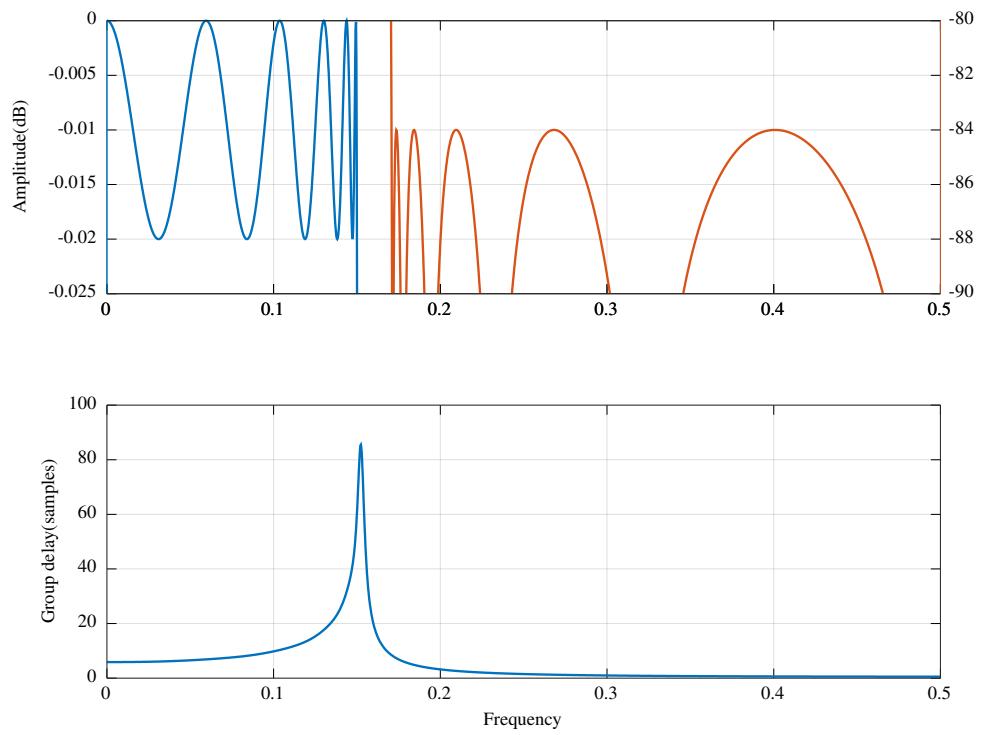


Figure 12.14: Response of an elliptic filter with $f_{ap}=0.15$, $d_{Bap}=0.02$, $d_{Bas}=84$

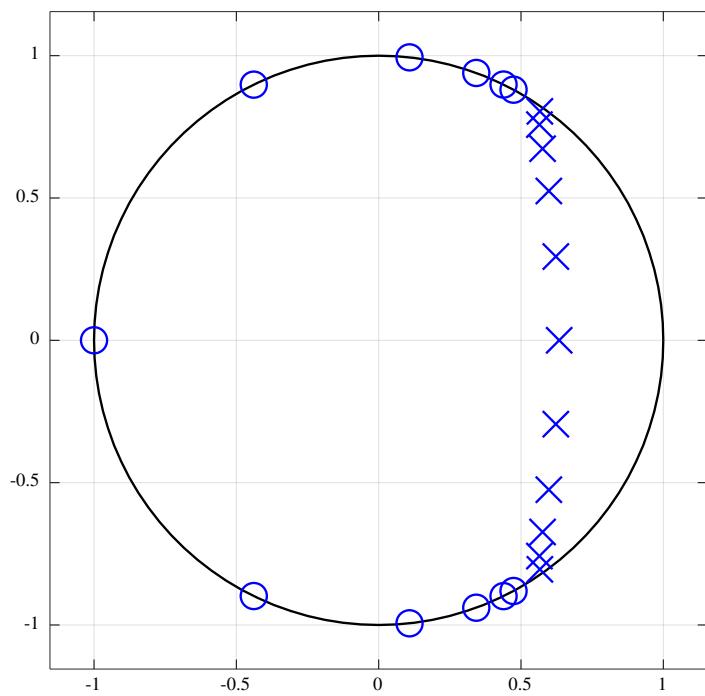


Figure 12.15: Pole-zero plot of an elliptic filter with $f_{ap}=0.15$, $d_{Bap}=0.02$, $d_{Bas}=84$

12.2.2 Design of an IIR filter as the sum of an all-pass filter composed of second-order sections and a delay

The Octave script `allpass2ndOrderCascadeDelay_socp_test.m` designs a lowpass filter consisting of an allpass filter in parallel with a pure delay. The allpass filter is composed of a cascade of second-order sections, as described in Section 12.2.1. The filter specification is:

```
n=500 % Number of frequency points
tol=1e-06 % Tolerance on coefficient update vector
tau=0.05 % Second order section stability parameter
ma=11 % Order of allpass filter
D=10 % Parallel delay in samples
td=10 % Nominal filter group delay in samples
fap= 0.15 % Pass band edge
Wap=1 % Pass band weight
fas= 0.2 % Stop band edge
Was=10 % Stop band weight (complex response)
Was_sqm=200 % Stop band weight (squared-magnitude)
```

This script produces two designs. The first attempts to optimise the filter for flat group-delay in the passband and the second ignores the phase response and optimises the weighted squared-magnitude response. The relative weights in each case are shown in the specification. The initial allpass filter was designed by the Octave script `tarczynski_allpass_phase_shift_test.m` to have a phase shift of 0 radians in the pass band and π radians in the stop band. The phase response of the initial allpass filter is shown in Figure 12.16. The phase has been adjusted by ωD , where D is the number of samples in the pure delay branch.

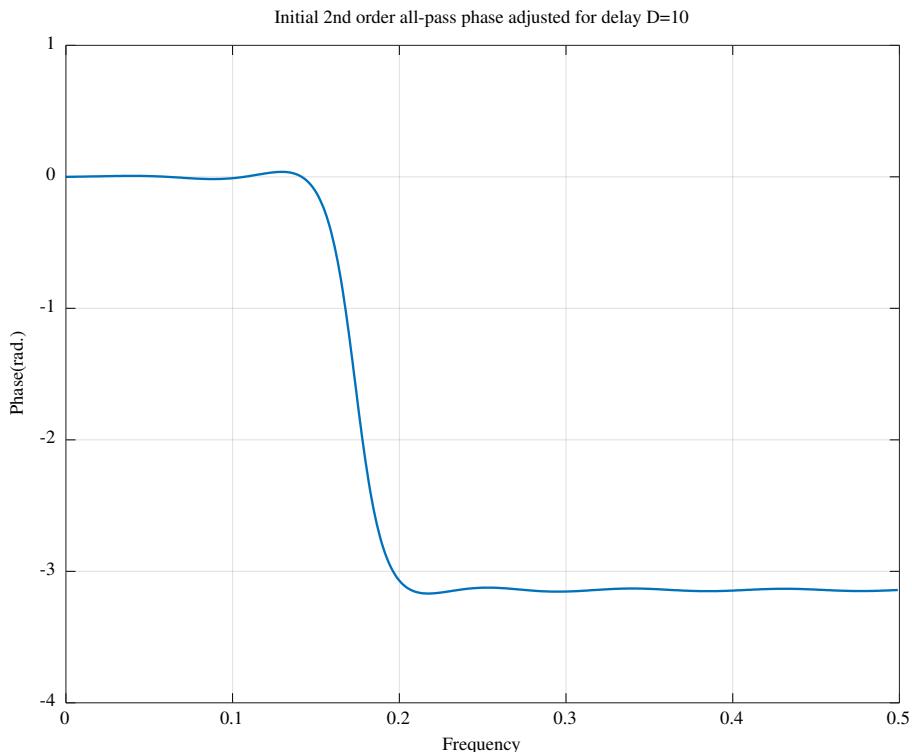


Figure 12.16: Parallel all-pass 2nd order cascade filter and delay : phase response of the initial all-pass filter adjusted for the group delay of the fixed delay branch

The allpass filter polynomial found for the optimised delay case is:

```
Da1 = [ 1.0000000000, -0.4940632894, 0.3737009604, 0.1807528317, ...
0.0198375896, -0.0547182785, -0.0506744667, -0.0144919038, ...
0.0113201551, 0.0138227871, 0.0045376384, -0.0041454497 ]';
```

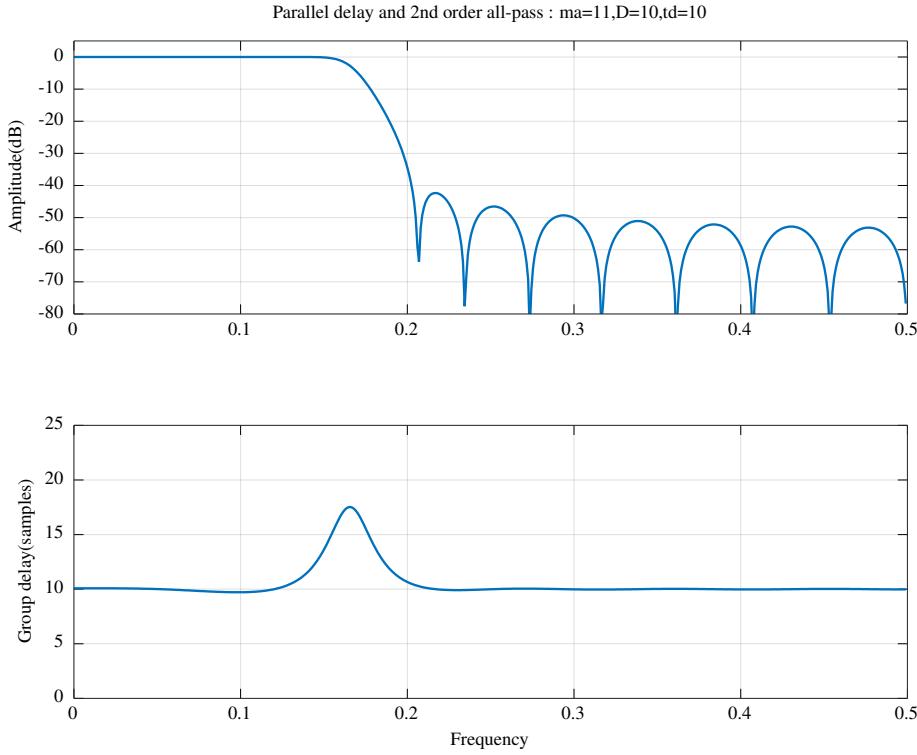


Figure 12.17: Parallel all-pass 2nd order cascade filter and delay : Overall response optimised for flat pass band group delay

with the overall filter response shown in Figure 12.17.

The allpass filter polynomial found for the optimised squared-magnitude case is:

```
Dalsqm = [ 1.0000000000, -0.5292155378, 0.3637414575, 0.1968593505, ...
0.0368342545, -0.0661347221, -0.0989858962, -0.0823773532, ...
-0.0478345116, -0.0194755024, -0.0051849836, -0.0007373676 ]';
```

with the overall filter response shown in Figure 12.18. At the filter band edges the change in phase of the allpass branch produces a corresponding transient in the group delay response.

12.2.3 Design of an IIR filter as the sum of two all-pass filters each represented in pole-zero form

Rewriting Equation 12.1 in terms of the phase responses of the component all-pass filters gives

$$H(z) = \frac{e^{\phi_1(z)} + e^{\phi_2(z)}}{2}$$

where $\phi_1(z) = \arg A_1(z)$ and $\phi_2(z) = \arg A_2(z)$. The squared magnitude response and phase of the frequency response, $H(\omega)$ are, with simple trigonometry

$$\begin{aligned} |H(\omega)|^2 &= \frac{1 + \cos(\phi_1(\omega) - \phi_2(\omega))}{2} \\ \arg H(\omega) &= \frac{\phi_1(\omega) + \phi_2(\omega)}{2} \end{aligned}$$

The group delay response is

$$T(\omega) = -\frac{1}{2} \left[\frac{\partial \phi_1(\omega)}{\partial \omega} + \frac{\partial \phi_2(\omega)}{\partial \omega} \right]$$

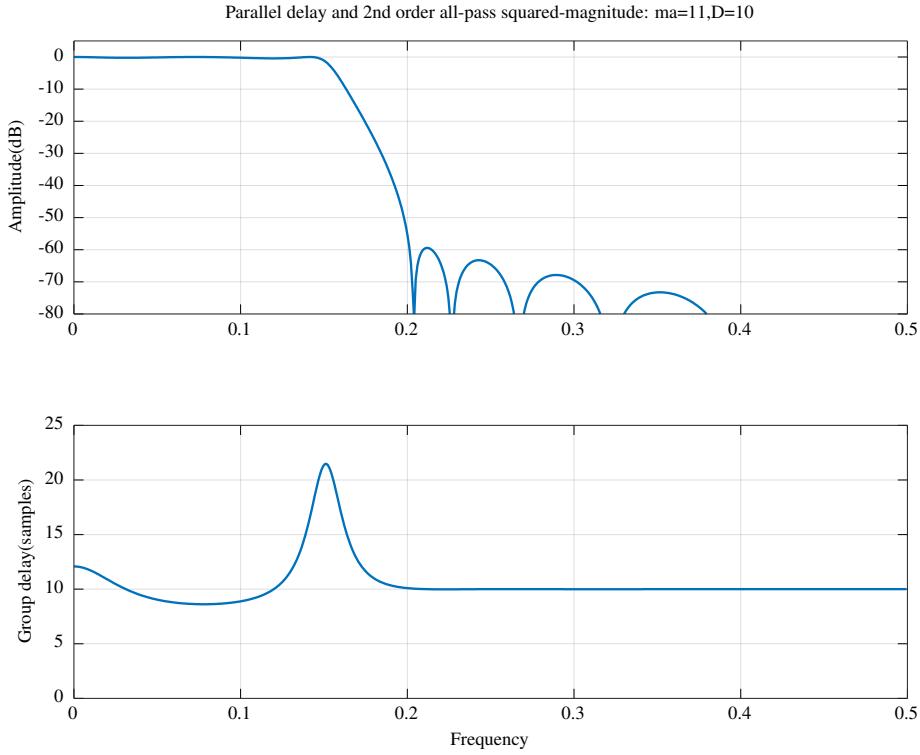


Figure 12.18: Parallel all-pass 2nd order cascade filter and delay : Overall response optimised for squared-magnitude

The partial derivatives of $|H(\omega)|^2$, $\arg H(\omega)$ and $T(\omega)$ with respect to the real and complex conjugate pole radiiuses of filters A_1 and A_2 (for convenience all represented here by r), are

$$\begin{aligned}\frac{\partial |H(\omega)|^2}{\partial r} &= -\frac{1}{2} \sin(\phi_1(\omega) - \phi_2(\omega)) \left\{ \frac{\partial \phi_1(\omega)}{\partial r} - \frac{\partial \phi_2(\omega)}{\partial r} \right\} \\ \frac{\partial \arg H(\omega)}{\partial r} &= \frac{1}{2} \left\{ \frac{\partial \phi_1(\omega)}{\partial r} + \frac{\partial \phi_2(\omega)}{\partial r} \right\} \\ \frac{\partial T(\omega)}{\partial r} &= -\frac{1}{2} \left\{ \frac{\partial^2 \phi_1(\omega)}{\partial r \partial \omega} + \frac{\partial^2 \phi_2(\omega)}{\partial r \partial \omega} \right\}\end{aligned}$$

The partial derivatives with respect to the filter pole angles, θ , are similar. Appendix D.1 derives the phase response of an all-pass filter and the partial derivatives of the phase response with respect to the real and complex conjugate pole locations. The Octave function *allpassP* calculates the phase response and partial derivatives of the phase response of an all-pass IIR filter. The Octave function *allpassT* calculates the group delay response and partial derivatives of the group delay response of an all-pass IIR filter. The Octave function *parallel_allpassAsq* calls *allpassP* to calculate the squared-magnitude and partial derivatives of the squared magnitude response of the parallel combination of two allpass IIR filters and is exercised by the test script *parallel_allpassAsq_test.m*. Similarly, the Octave function *parallel_allpassP* calls *allpassP* to calculate the phase and partial derivatives of the phase response of the parallel combination of two all-pass IIR filters and is exercised by the test script *parallel_allpassP_test.m*. Finally, the Octave function *parallel_allpassT* calls *allpassT* to calculate the group delay and partial derivatives of the group delay response of the parallel combination of two all-pass IIR filters and is exercised by the test script *parallel_allpassT_test.m*.

Design of an IIR filter as the sum of two all-pass filters each represented in pole-zero form with no constraints on the group delay

The Octave script *parallel_allpass_socp_slb_test.m* calls the Octave function *parallel_allpass_slb* to perform the PCLS design of a low-pass filter composed of two parallel all-pass filters in terms of the all-pass filter pole locations. This script does not constrain the group delay of the filter. The PCLS algorithm of Selesnick, Lang and Burrus was reviewed in Section 10.1.2. At each iteration of the PCLS optimisation the Octave function *parallel_allpass_socp_mmse* optimises the filter response subject to the current constraints. The filter specification is

```
polyphase=0 % Use polyphase combination
```

```

tol=0.0001 % Tolerance on coefficient update vector
ctol=7e-09 % Tolerance on constraints
n=1000 % Frequency points across the band
ma=5 % Allpass model filter A denominator order
Va=1 % Allpass model filter A no. of real poles
Qa=4 % Allpass model filter A no. of complex poles
Ra=1 % Allpass model filter A decimation
mb=6 % Allpass model filter B denominator order
Vb=0 % Allpass model filter B no. of real poles
Qb=6 % Allpass model filter B no. of complex poles
Rb=1 % Allpass model filter B decimation
fap=0.15 % Pass band amplitude response edge
dBap=0.050000 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas=0.17 % Stop band amplitude response edge
dBas=83.000000 % Stop band amplitude response ripple
Was=1000 % Stop band amplitude response weight
rho=0.999000 % Constraint on allpass pole radius

```

The initial parallel all-pass filters were designed by the Octave script *tarczynski_parallel_allpass_test.m* (with *flat_delay = false*), using the WISE method described in Section 10.1.5. The response of the initial filter is shown in Figure 12.19 and the pole-zero plot of the initial filter is shown in Figure 12.20. The final filter response is shown in Figure 12.21 with passband detail

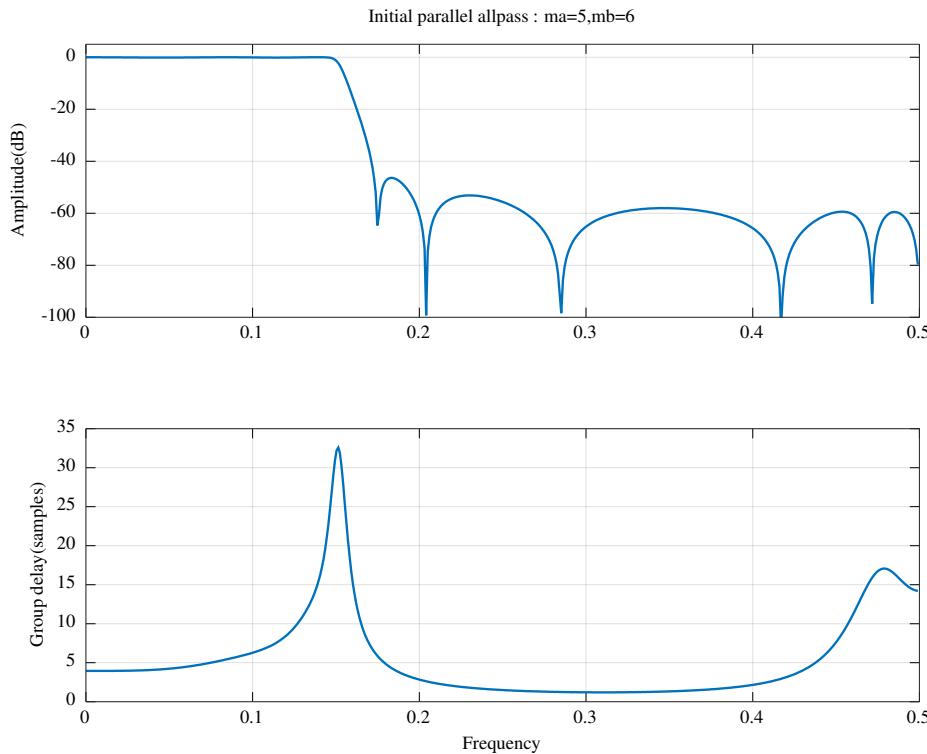


Figure 12.19: Parallel all-pass filters : Initial response found with the WISE barrier function

shown in Figure 12.22 and pole-zero plot shown in Figure 12.23. The pole-zero plots of the branch allpass filters are shown in Figures 12.24 and 12.25. The denominator polynomials of the two all-pass filters are

```

Da1 = [ 1.0000000000, -2.8330463002, 4.0371210157, -3.2282596781, ...
1.4628447196, -0.2936225622 ]';

```

and

```

Db1 = [ 1.0000000000, -3.4038084317, 5.9908536454, -6.2950667259, ...
4.1745726517, -1.6254957473, 0.2961665555 ]';

```

The corresponding filter numerator and denominator polynomials of the overall filter are

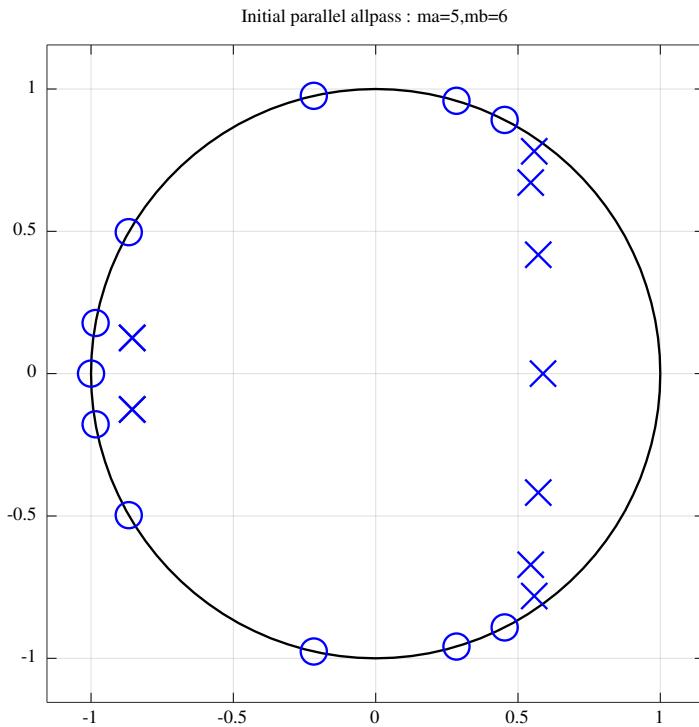


Figure 12.20: Parallel all-pass filters : Pole zero plot of the initial response found with the WISE barrier function

```
Nab1 = [ 0.0012719966, -0.0011348195, 0.0043924618, -0.0013445763, ...
0.0049871171, 0.0017789635, 0.0017789635, 0.0049871171, ...
-0.0013445763, 0.0043924618, -0.0011348195, 0.0012719966 ]';
```

and

```
Dab1 = [ 1.0000000000, -6.2368547319, 19.6711215452, -40.2372787124, ...
58.6458115349, -63.4790965418, 51.8397599086, -31.8457360881, ...
14.3983077846, -4.5596991378, 0.9105279080, -0.0869611829 ]';
```

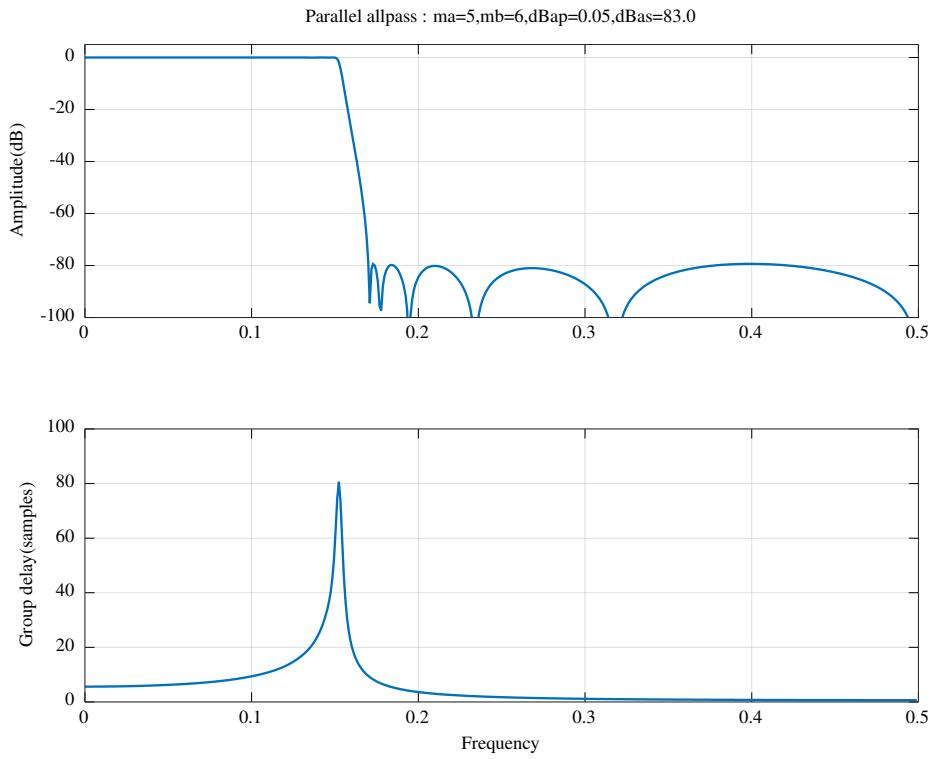


Figure 12.21: Parallel all-pass filters : Response after PCLS SOCP optimisation

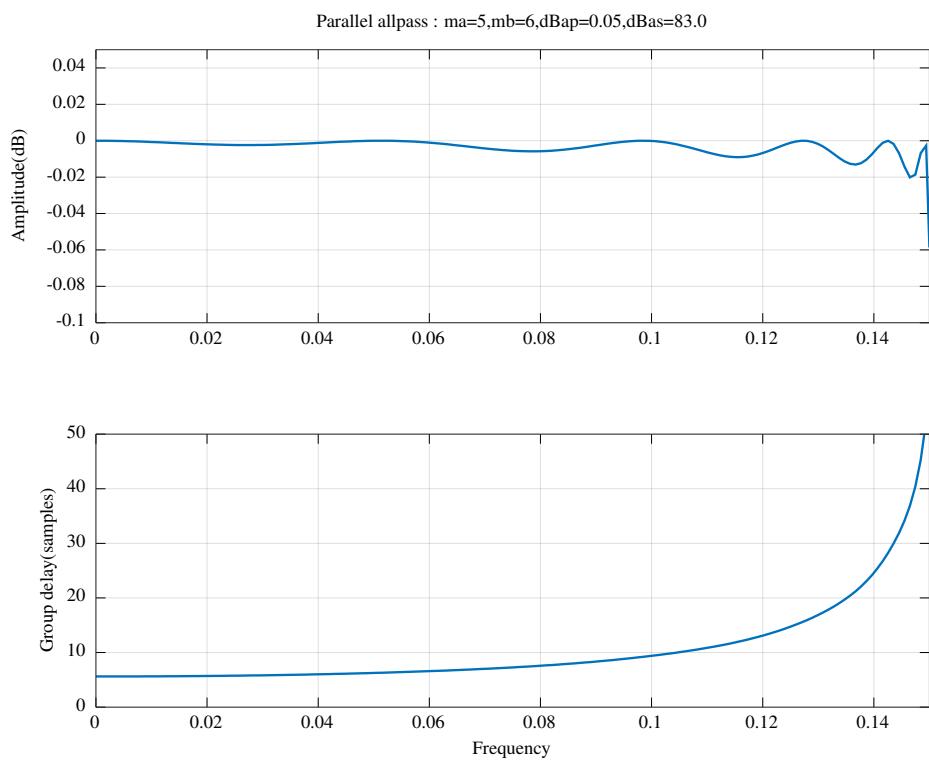


Figure 12.22: Parallel all-pass filters : Pass-band response after PCLS SOCP optimisation

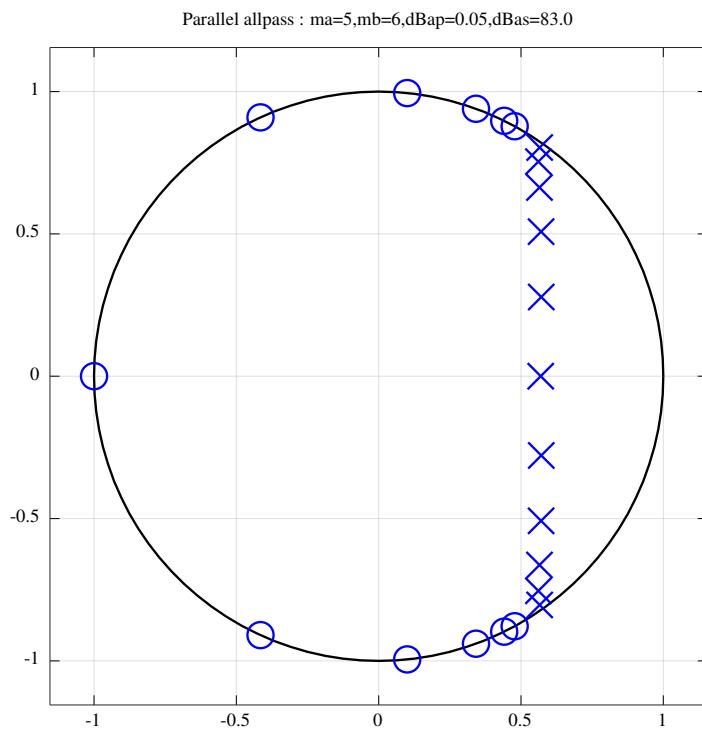


Figure 12.23: Parallel all-pass filters : Pole-zero plot after PCLS SOCP optimisation

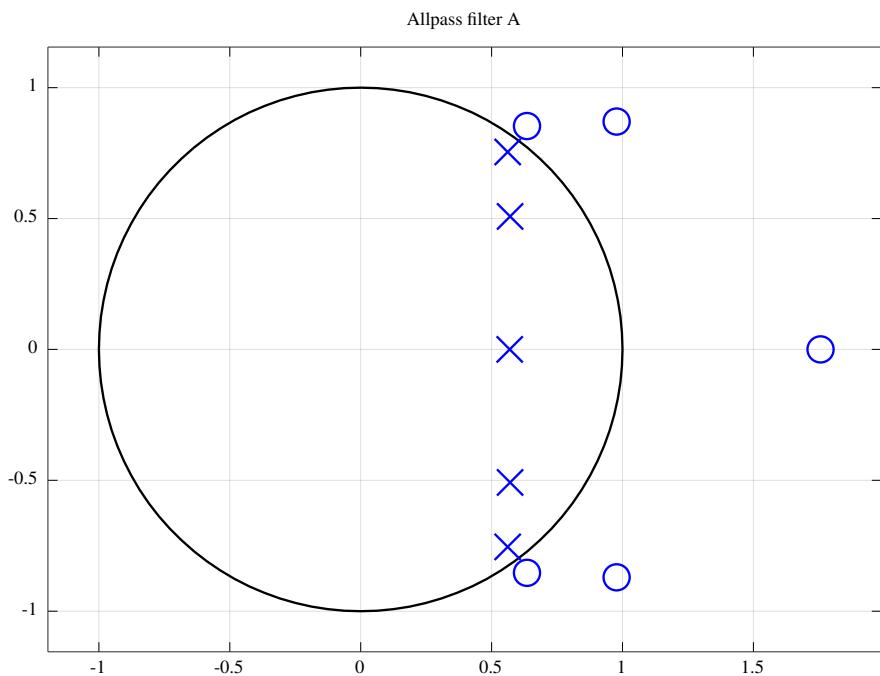


Figure 12.24: Parallel all-pass filters : Pole-zero plot of the A allpass filter branch after PCLS SOCP optimisation

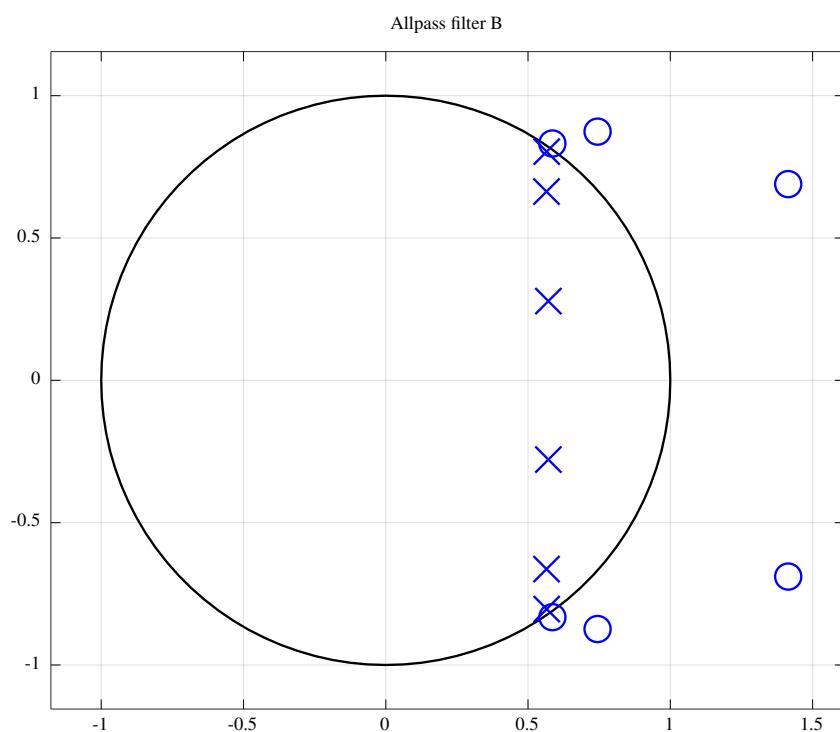


Figure 12.25: Parallel all-pass filters : Pole-zero plot of the B allpass filter branch after PCLS SOCP optimisation

Design of an IIR filter as the sum of two all-pass filters each represented in pole-zero form with constraints on the group delay

The Octave script `parallel_allpass_socp_slb_flat_delay_test.m` calls the Octave function `parallel_allpass_slb` to perform the PCLS design of a low-pass filter composed of two parallel all-pass filters in terms of the all-pass filter pole locations. This script does constrain the group delay of the filter. The PCLS algorithm of *Selesnick, Lang and Burrus* was reviewed in Section 10.1.2. At each iteration of the PCLS optimisation the Octave function `parallel_allpass_socp_mmse` optimises the filter response subject to the current constraints. The filter specification is

```

polyphase=0 % Use polyphase combination
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
ma=11 % Allpass model filter A denominator order
Va=1 % Allpass model filter A no. of real poles
Qa=10 % Allpass model filter A no. of complex poles
Ra=1 % Allpass model filter A decimation
mb=12 % Allpass model filter B denominator order
Vb=2 % Allpass model filter B no. of real poles
Qb=10 % Allpass model filter B no. of complex poles
Rb=1 % Allpass model filter B decimation
fap=0.15 % Pass band amplitude response edge
dBap=3.000000 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas=0.2 % Stop band amplitude response edge
dBas=40.000000 % Stop band amplitude response ripple
Was=750 % Stop band amplitude response weight
ftp=0.175 % Pass band group delay response edge
td=11.5 % Pass band nominal group delay
tdr=0.08 % Pass band nominal group delay ripple
Wtp=1 % Pass band group delay response weight
rho=0.992188 % Constraint on allpass pole radius

```

The initial parallel allpass filters were designed by the Octave script `tarczynski_parallel_allpass_test.m` (with `flat_delay = true`), using the *WISE* method described in Section 10.1.5. The response of the initial filter is shown in Figure 12.26 and the pole-zero plot of the initial filter is shown in Figure 12.27. The final filter response is shown in Figure 12.28 with passband detail shown in Figure 12.29 and pole-zero plot shown in Figure 12.30. The pole-zero plots of the branch allpass filters are shown in Figures 12.31 and 12.32. The denominator polynomials of the two all-pass filters are

```

Da1 = [ 1.0000000000, 0.3931432341, -0.2660133321, -0.0850275861, ...
-0.2707651069, -0.0298153197, 0.1338823243, -0.0589362474, ...
0.1650490792, 0.0296371262, -0.1113859180, 0.0372881323 ]';

```

and

```

Db1 = [ 1.0000000000, -0.1344939785, -0.0918734630, 0.4461033862, ...
-0.1115261080, 0.1180340147, 0.0396352218, -0.2006006436, ...
0.2105512466, -0.0838522576, -0.1001537312, 0.1080994566, ...
-0.0610732672 ]';

```

The corresponding filter numerator and denominator polynomials of the overall filter are

```

Nab1 = [ -0.0118925675, -0.0161560163, -0.0001083932, 0.0205704164, ...
0.0369335724, 0.0211196767, -0.0280717678, -0.0596562038, ...
-0.0358505121, 0.0676811446, 0.2182340832, 0.3207798419, ...
0.3207798419, 0.2182340832, 0.0676811446, -0.0358505121, ...
-0.0596562038, -0.0280717678, 0.0211196767, 0.0369335724, ...
0.0205704164, -0.0001083932, -0.0161560163, -0.0118925675 ]';

```

and

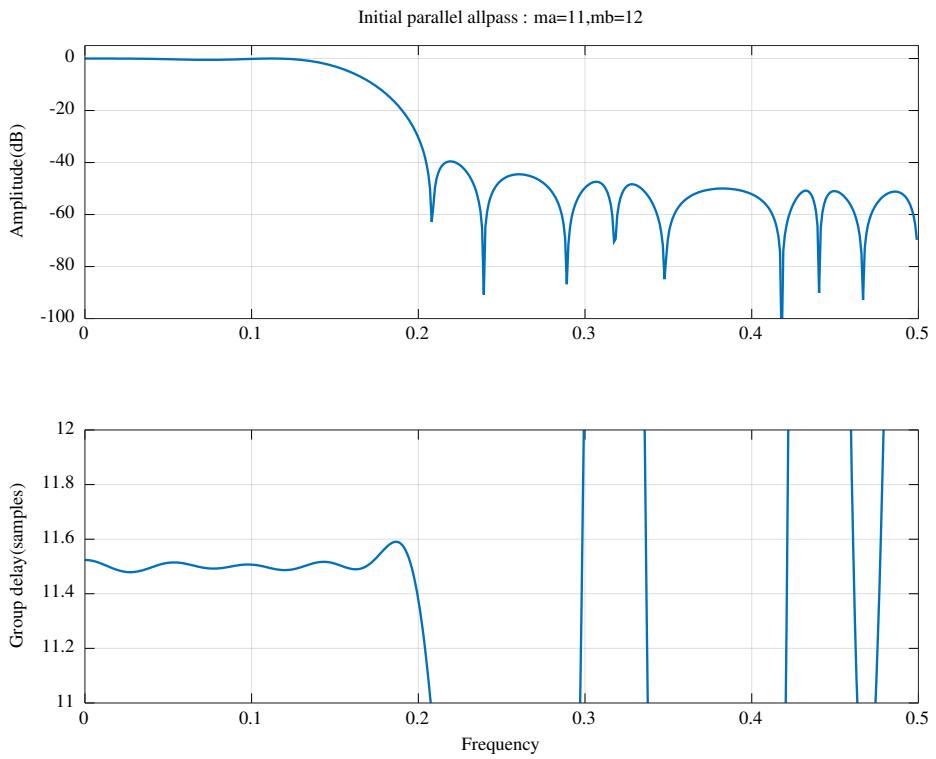


Figure 12.26: Parallel all-pass filters with flat pass-band delay : Initial response found with the WISE barrier function

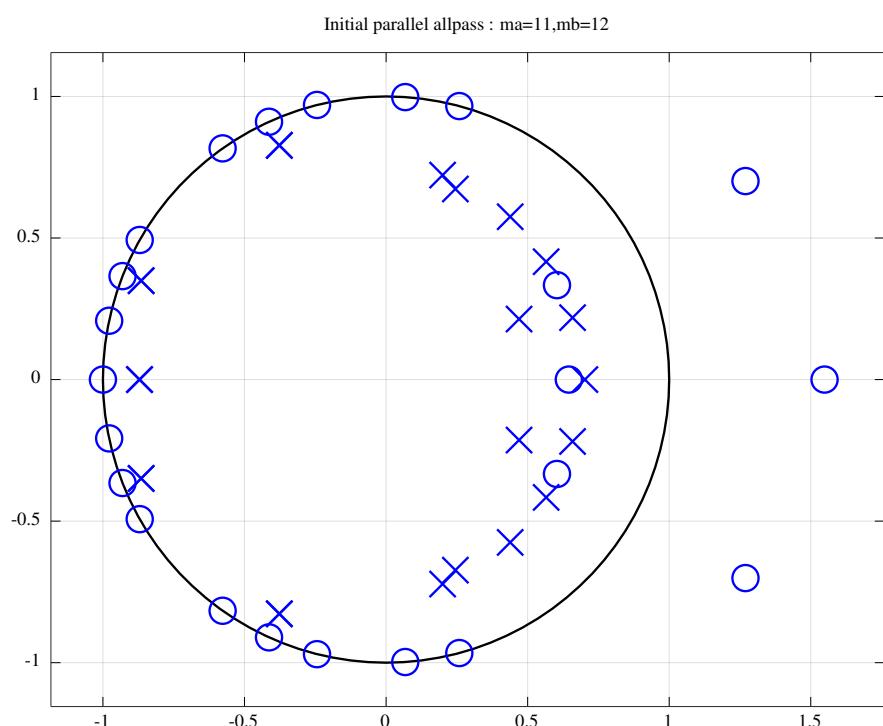


Figure 12.27: Parallel all-pass filters with flat pass-band delay : Pole zero plot of the initial response found with the WISE barrier function

```

Dab1 = [ 1.0000000000,  0.2586492556, -0.4107621927,  0.3607335610, ...
-0.1710334226, -0.0300684328,  0.2405442669, -0.4019267583, ...
0.2886788963,  0.0928616471, -0.3580821837,  0.2718123207, ...
-0.0374493775, -0.0939561603,  0.1157815704, -0.0939661732, ...
0.0336391381,  0.0384186168, -0.0644954956,  0.0356638996, ...
0.0011526919, -0.0175853389,  0.0108335288, -0.0022773081 ]';

```

The overall numerator polynomial is symmetric to within the precision of the calculations and has pairs of reciprocal zeros that do not lie on the unit circle. This should be compared with the pole-zero plot resulting when the delay is not constrained, Figure 12.23, and the pole-zero plot for the elliptic filter example, Figure 12.15.

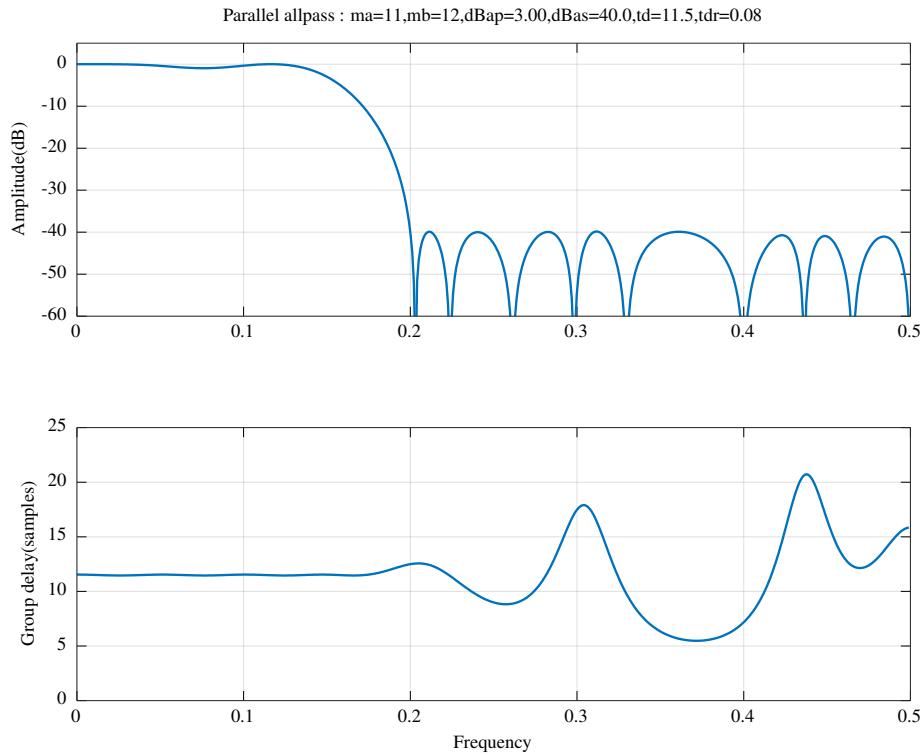


Figure 12.28: Parallel all-pass filters with flat pass-band delay : Response after PCLS SOCP optimisation

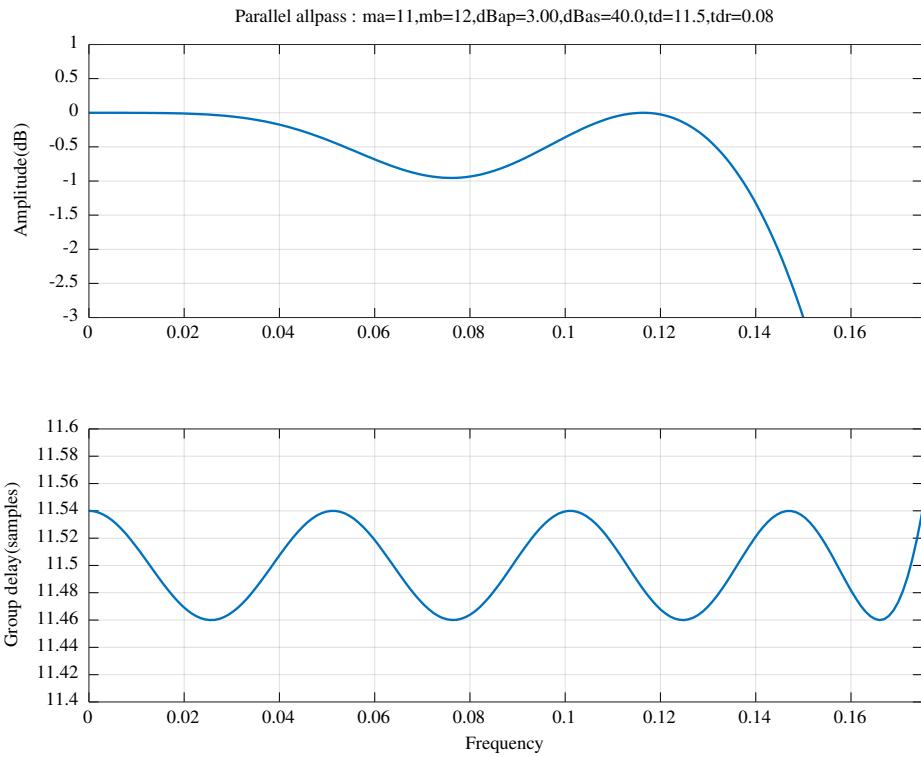


Figure 12.29: Parallel all-pass filters with flat pass-band delay : Pass-band response after PCLS SOCP optimisation

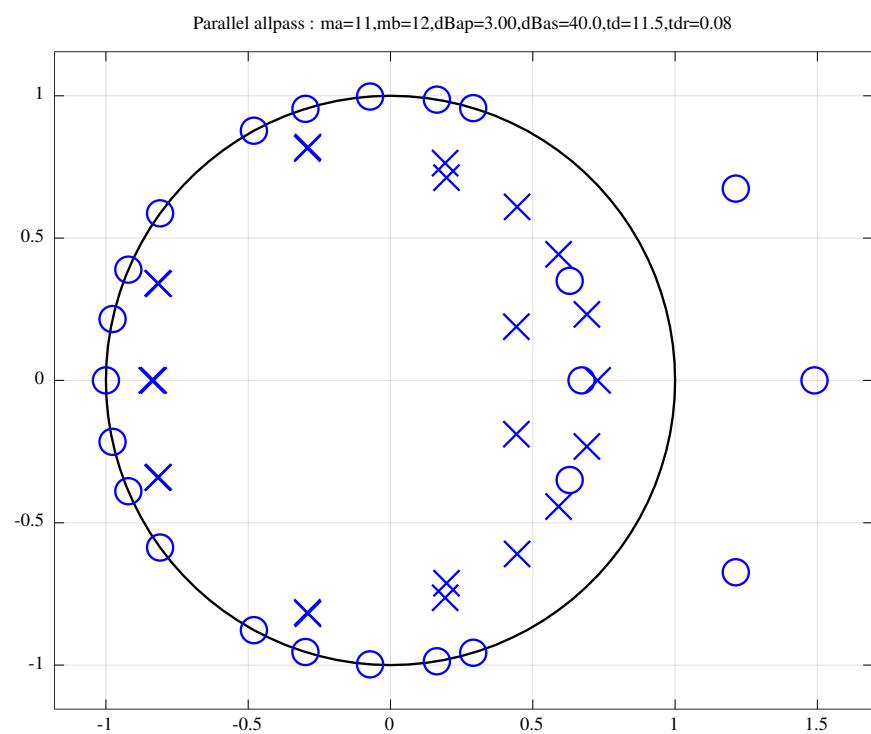


Figure 12.30: Parallel all-pass filters with flat pass-band delay : Pole-zero plot after PCLS SOCP optimisation

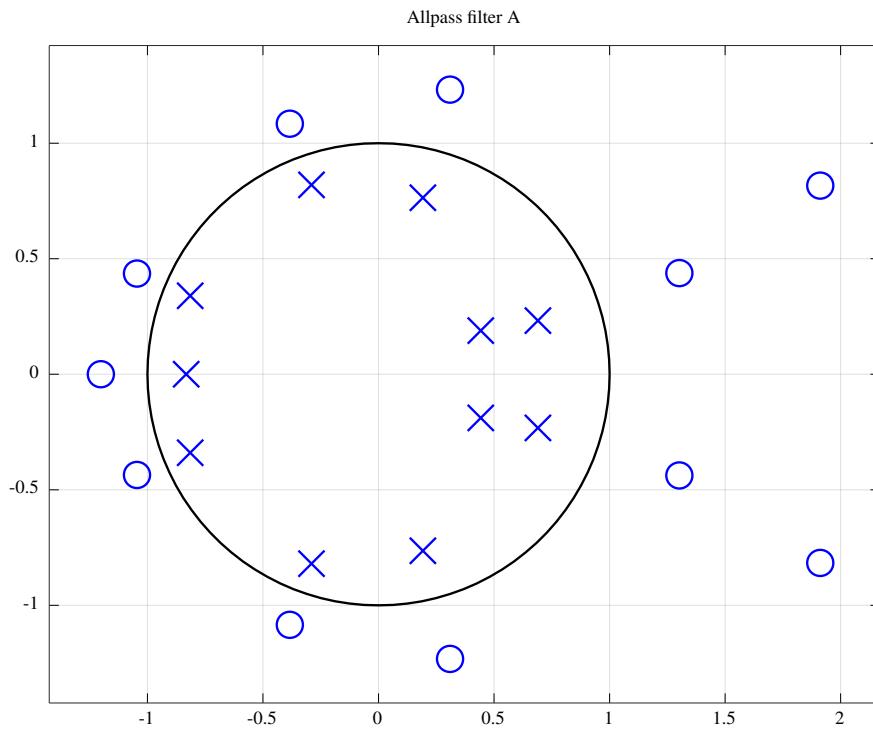


Figure 12.31: Parallel all-pass filters : Pole-zero plot of the A allpass filter branch after PCLS SOCP optimisation

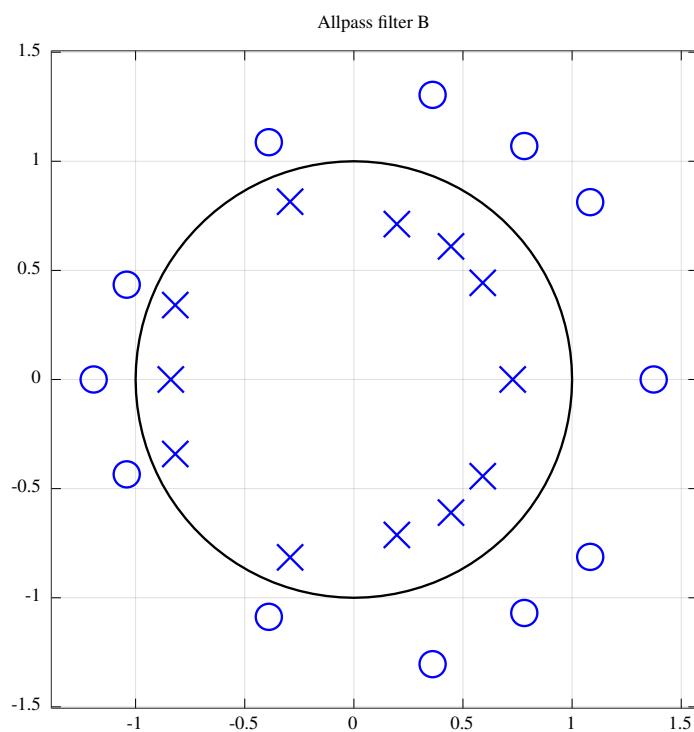


Figure 12.32: Parallel all-pass filters with flat pass-band delay : Pole-zero plot of the B allpass filter branch after PCLS SOCP optimisation

12.2.4 Design of an IIR filter as the sum of a delay and an all-pass filter represented in pole-zero form

Kunold [37] suggests realisation of a low-pass filter with flat pass-band delay by the parallel combination of a wave-digital lattice filter and a pure delay. The parallel fixed delay means that the all-pass filter phase change during the pass-band to stop-band transition of the overall amplitude response must result in a large peak in the group delay response over that transition band.

Design of an IIR filter as the sum of a delay and an all-pass filter represented in pole-zero form using SOCP

The Octave script *parallel_allpass_delay_socp_slb_test.m* uses the SeDuMi SOCP solver to design a low-pass filter consisting of an all-pass filter in parallel with a delay. The script calls the Octave function *parallel_allpass_delay_slb* to perform the PCLS design of the filter in terms of the all-pass filter pole locations. The PCLS algorithm of *Selesnick, Lang and Burrus* was reviewed in Section 10.1.2. At each iteration of the PCLS optimisation the Octave function *parallel_allpass_delay_socp_mmse* optimises the filter response subject to the current constraints. The filter specification is

```
tol=1e-05 % Tolerance on coefficient update vector
ctol=1e-08 % Tolerance on constraints
n=1000 % Frequency points across the band
m=12 % Allpass filter denominator order
V=0 % Allpass filter no. of real poles
Q=12 % Allpass filter no. of complex poles
R=1 % Allpass filter decimation
DD=11 % Parallel delay
fap=0.15 % Pass band amplitude response edge
dBap=0.500000 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas=0.2 % Stop band amplitude response edge
dBas=66.000000 % Stop band amplitude response ripple
Was=100000 % Stop band amplitude response weight
rho=0.992188 % Constraint on allpass pole radius
```

There are no group delay constraints.

The initial allpass filter was designed by the Octave script *tarczynski_parallel_allpass_delay_test.m* using the *WISE* method described in Section 10.1.5. The response of the initial filter is shown in Figure 12.33 and the pole-zero plot of the initial filter is shown in Figure 12.34.

The final filter response is shown in Figure 12.35 with pass-band and stop-band details shown in Figure 12.36. The pole-zero plot of the all-pass branch of the filter is shown in Figure 12.37. The denominator polynomial of the final all-pass filter is

```
Da1 = [ 1.0000000000, -0.5309207779, 0.3596778262, 0.1918749899, ...
        0.0368764697, -0.0537876390, -0.0708690080, -0.0412674263, ...
        -0.0028480451, 0.0194176547, 0.0215207341, 0.0129527154, ...
        0.0044735847 ]';
```

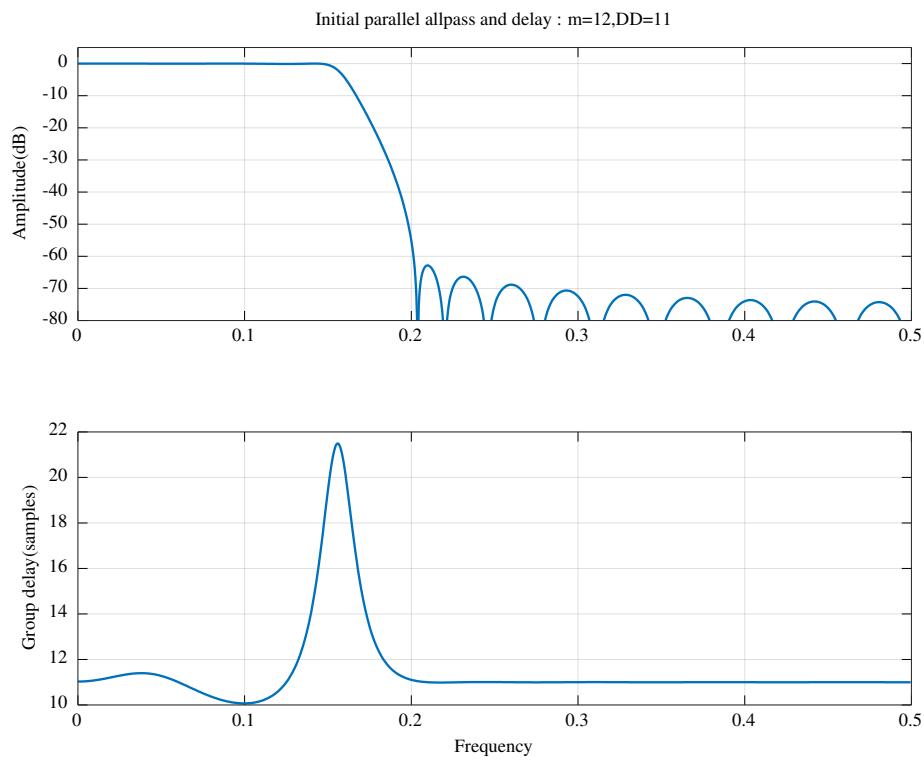


Figure 12.33: Parallel delay and all-pass filter : Initial response found with the WISE barrier function

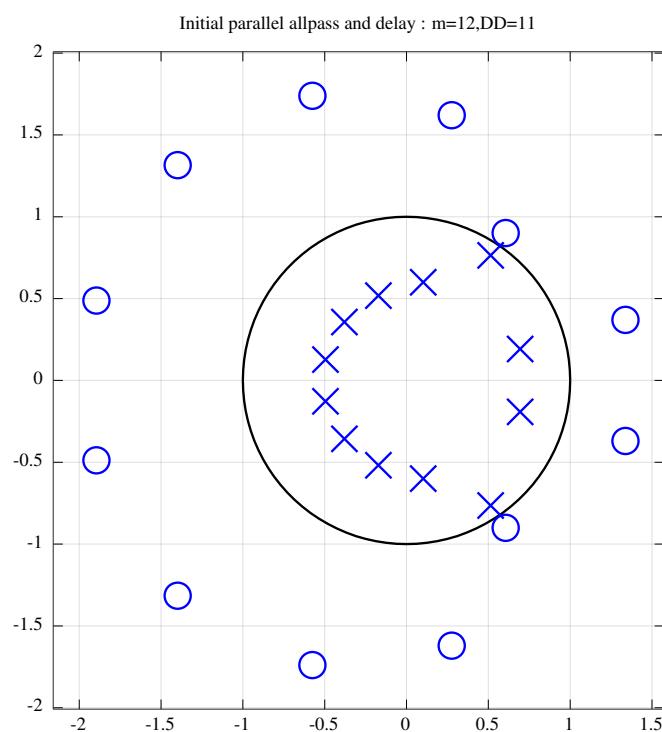


Figure 12.34: Parallel delay and all-pass filter : Pole zero plot of the initial filter found with the WISE barrier function

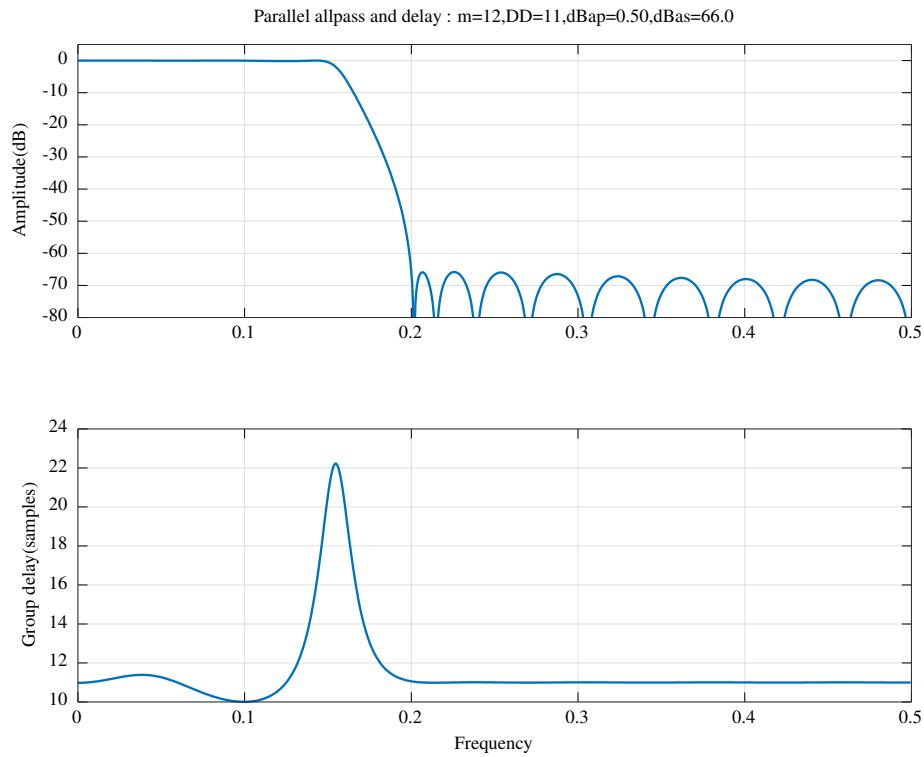


Figure 12.35: Parallel delay and all-pass filter : Response after PCLS SOCP optimisation

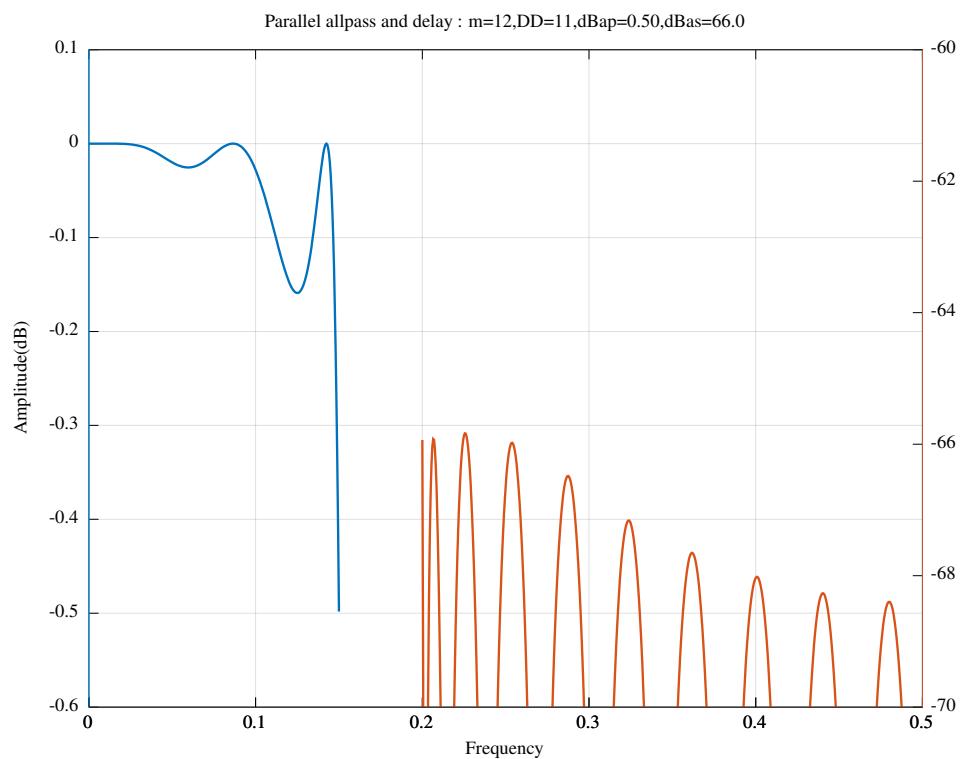


Figure 12.36: Parallel delay and all-pass filter : Detail of the pass-band and stop-band responses after PCLS SOCP optimisation

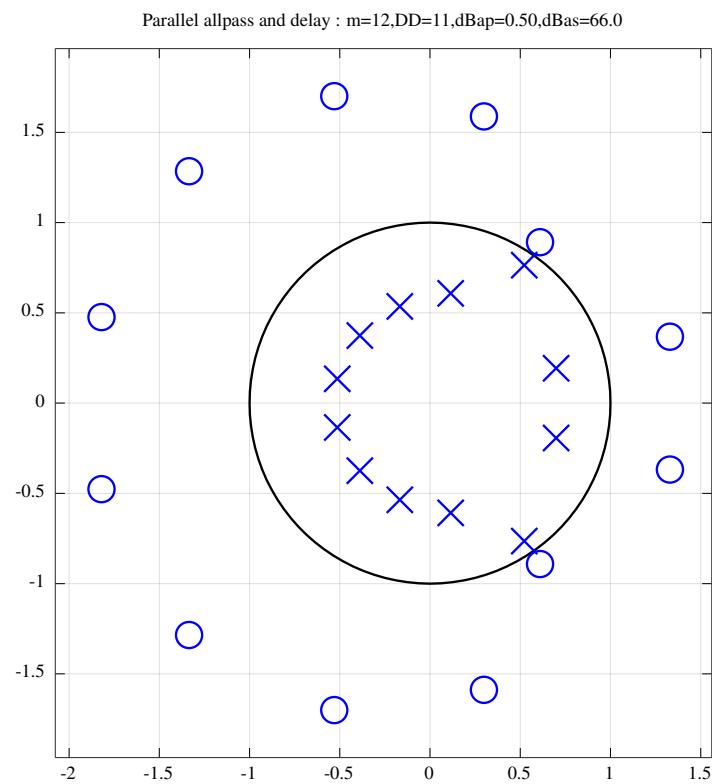


Figure 12.37: Parallel delay and all-pass filter : Pole-zero plot of the all-pass filter branch after PCLS SOCP optimisation

Design of an IIR filter as the sum of a delay and an all-pass filter represented in pole-zero form using SQP

The Octave script *parallel_allpass_delay_socp_slb_test.m* repeats the design example of Section 12.2.4 with the SQP solver. The script calls the Octave function *parallel_allpass_delay_slb* to perform the PCLS design of the filter in terms of the all-pass filter pole locations. At each iteration of the PCLS optimisation the Octave function *parallel_allpass_delay_sqp_mmse* optimises the filter response subject to the current constraints. The filter specification is

```
tol=1e-05 % Tolerance on coefficient update vector
ctol=2e-07 % Tolerance on constraints
n=1000 % Frequency points across the band
m=12 % Allpass filter denominator order
V=0 % Allpass filter no. of real poles
Q=12 % Allpass filter no. of complex poles
R=1 % Allpass filter decimation
DD=11 % Parallel delay
fap=0.15 % Pass band amplitude response edge
dBap=0.050000 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas=0.2 % Stop band amplitude response edge
dBas=46.000000 % Stop band amplitude response ripple
Was=100 % Stop band amplitude response weight
rho=0.992188 % Constraint on allpass pole radius
dmax=0.050000 % Constraint on coefficient step-size
```

There are no group delay constraints.

The final filter response is shown in Figure 12.38 with pass-band and stop-band details shown in Figure 12.39. Figure 12.40 shows the phase response of the all-pass branch of the filter after adjustment for the delay of the fixed delay branch. The denominator polynomial of the final all-pass filter is

```
Da1 = [ 1.0000000000, -0.4972014446, 0.3729579920, 0.1821060380, ...
        0.0210300485, -0.0543946712, -0.0485183537, -0.0067262912, ...
        0.0252898245, 0.0288436289, 0.0196632773, -0.0006042797, ...
        0.0000284523 ]';
```

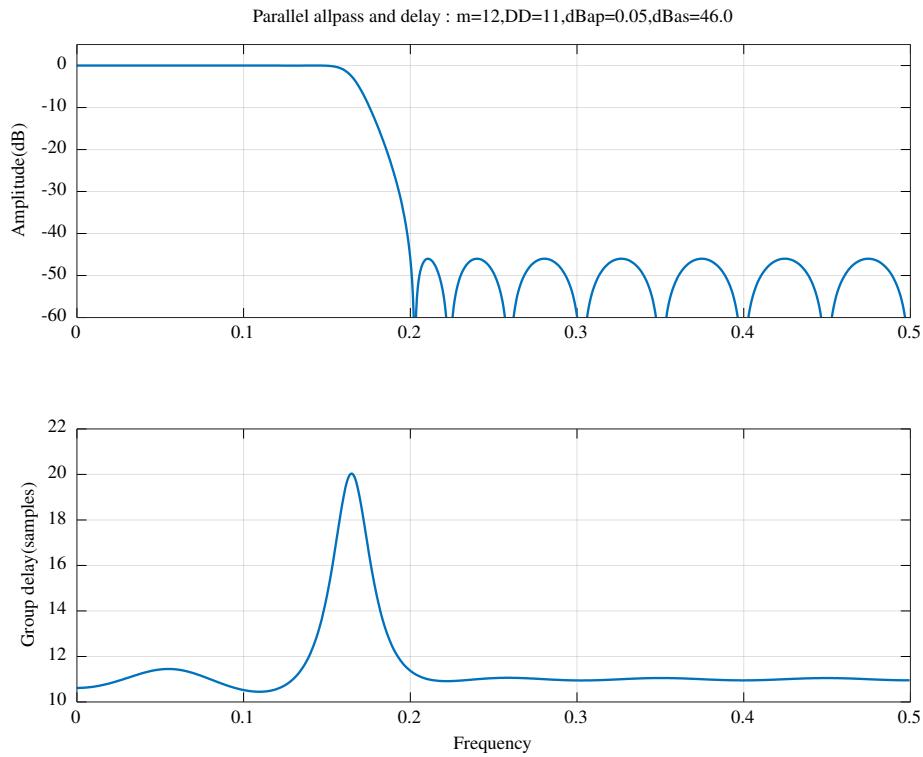


Figure 12.38: Parallel delay and all-pass filter : Response after PCLS SQP optimisation

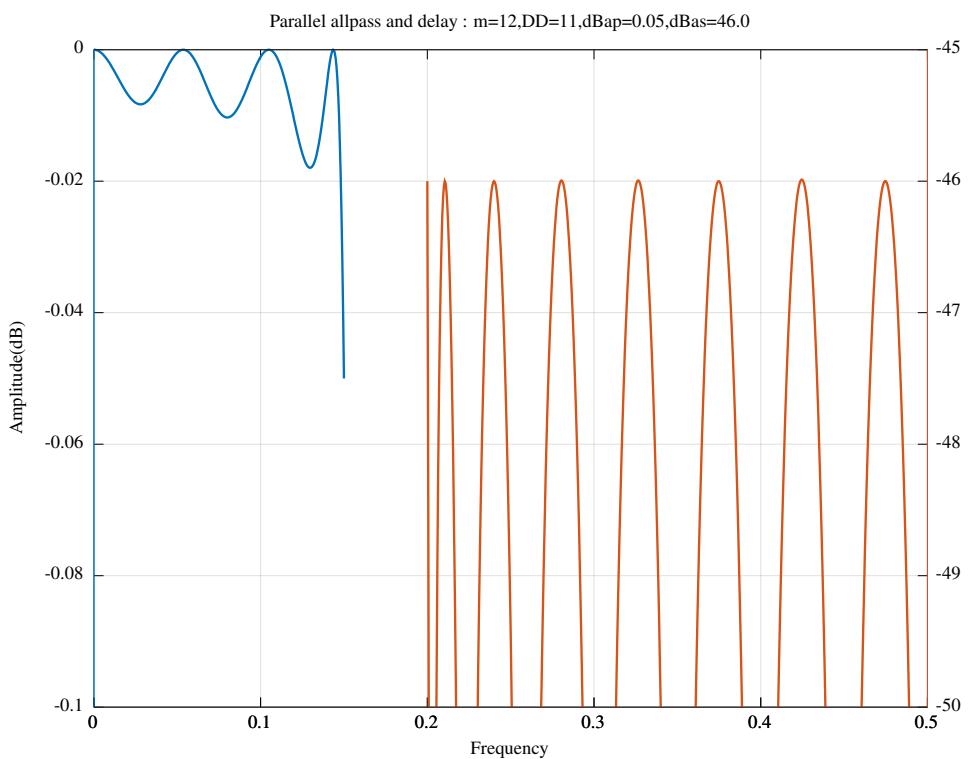


Figure 12.39: Parallel delay and all-pass filter : Detail of the pass-band and stop-band responses after PCLS SQP optimisation

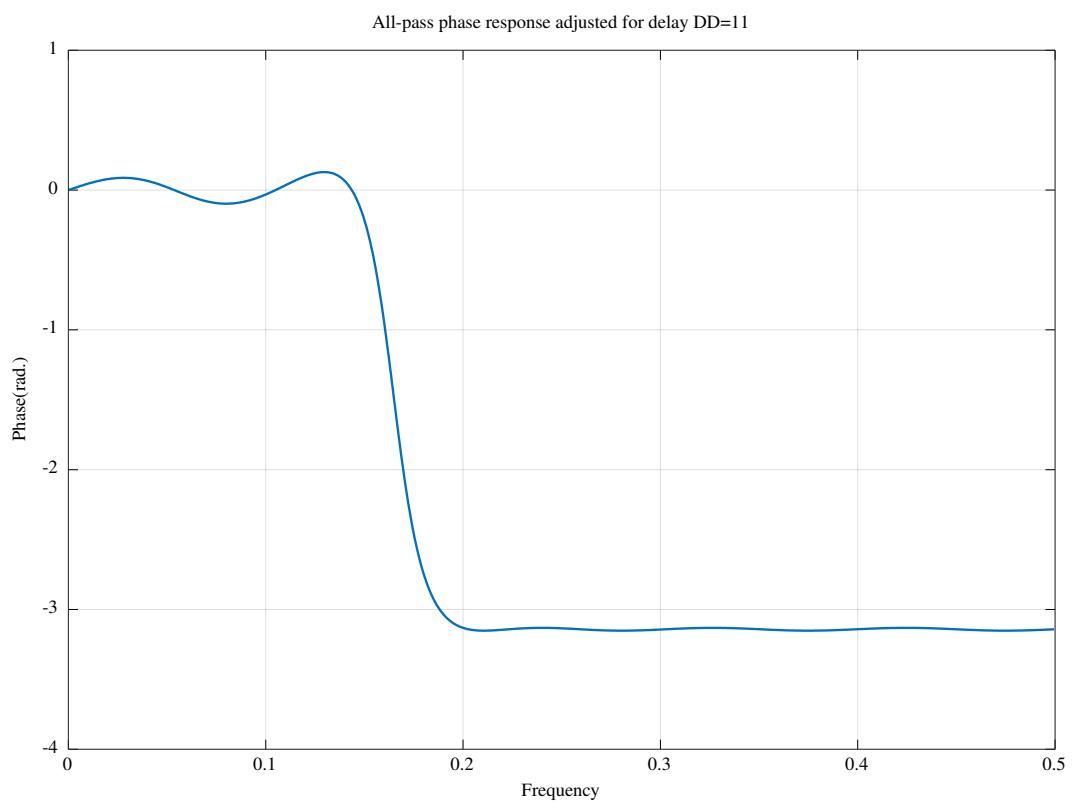


Figure 12.40: Parallel delay and all-pass filter : phase response of the all-pass filter branch after PCLS SQP optimisation. The response has been adjusted for the group delay of the fixed delay branch

12.2.5 Design of an IIR filter as the polyphase decomposition into two all-pass filters each represented in pole-zero form

The IIR filter of Equation 12.1 can be represented in “polyphase” form [60, 61] as

$$H(z) = \frac{1}{R} \sum_{k=0}^{R-1} z^{-k} A_k(z^R) \quad (12.2)$$

Where the A_k are all-pass filters. In the following I consider an example with $R = 2$, in other words, a half-band decimation filter suitable for use in an efficient half-band filter-bank. The Octave functions *parallel_allpassP* and *parallel_allpassAsq* support the calculation of the phase and squared magnitude response, respectively, of the polyphase combination of two all-pass filters with $R = 2$.

Design of an IIR filter as the polyphase decomposition into two all-pass filters each represented in pole-zero form with no constraints on the group delay

The Octave script *polyphase_allpass_socp_slb_test.m* calls the Octave function *parallel_allpass_slb* to design a low-pass filter composed of the polyphase combination of two parallel all-pass filters in terms of the allpass filter pole locations. The response of the filter is optimised with the PCLS algorithm described in Section 10.1.2. The filter specification is:

```

polyphase=1 % Use polyphase combination
tol=0.0001 % Tolerance on coefficient update vector
ctol=5e-08 % Tolerance on constraints
n=500 % Frequency points across the band
ma=11 % Allpass model filter A denominator order
Va=3 % Allpass model filter A no. of real poles
Qa=8 % Allpass model filter A no. of complex poles
Ra=2 % Allpass model filter A decimation
mb=11 % Allpass model filter B denominator order
Vb=3 % Allpass model filter B no. of real poles
Qb=8 % Allpass model filter B no. of complex poles
Rb=2 % Allpass model filter B decimation
fap=0.24 % Pass band amplitude response edge
dBap=0.001000 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas=0.26 % Stop band amplitude response edge
dBas=70.000000 % Stop band amplitude response ripple
Was=1000 % Stop band amplitude response weight
rho=0.999000 % Constraint on allpass pole radius

```

As in Section 12.2.3, the script does not optimise for a flat filter group delay.

The initial parallel all-pass filters were designed by the Octave script *tarczynski_polyphase_allpass_test.m* using the *WISE* method described in Section 10.1.5. In this case the script has *flat_delay = false*. The response of the initial filter is shown in Figure 12.41. After PCLS optimisation with the *SeDuMi* SOCP solver the response is shown in Figure 12.42. The denominator polynomials of the two all-pass filters are

```

Da1 = [ 1.0000000000, 0.0000000000, 0.2174624884, 0.0000000000, ...
-0.9121950515, -0.0000000000, -0.1497527454, -0.0000000000, ...
0.0006129877, 0.0000000000, 0.0124771222, 0.0000000000, ...
0.0063389050, 0.0000000000, -0.0219137665, -0.0000000000, ...
0.1313130566, 0.0000000000, 0.0259969767, 0.0000000000, ...
-0.0426390011, -0.0000000000, -0.0067999380 ]';

```

and

```

Db1 = [ 1.0000000000, 0.0000000000, 0.7171785131, 0.0000000000, ...
-0.9281154090, -0.0000000000, -0.5707270897, -0.0000000000, ...
0.0144870391, 0.0000000000, -0.0067639320, -0.0000000000, ...
0.0243146407, 0.0000000000, -0.0277931670, -0.0000000000, ...
0.1254789366, 0.0000000000, 0.0905347219, 0.0000000000, ...
-0.0448843722, -0.0000000000, -0.0246495096 ]';

```

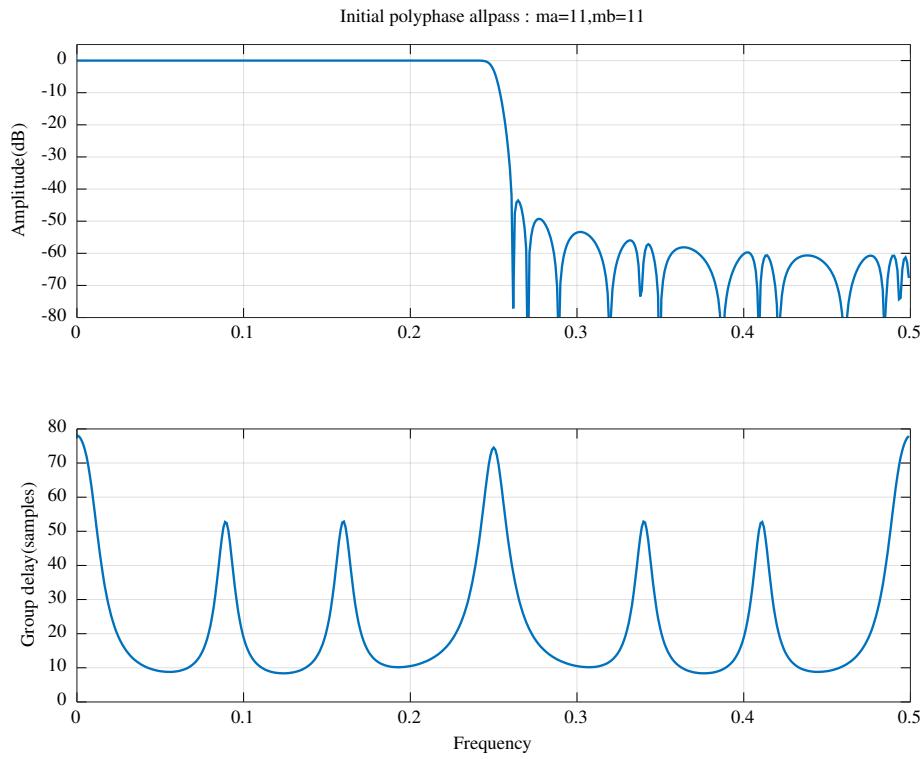


Figure 12.41: Polyphase combination of allpass filters : Initial response found with the WISE barrier function

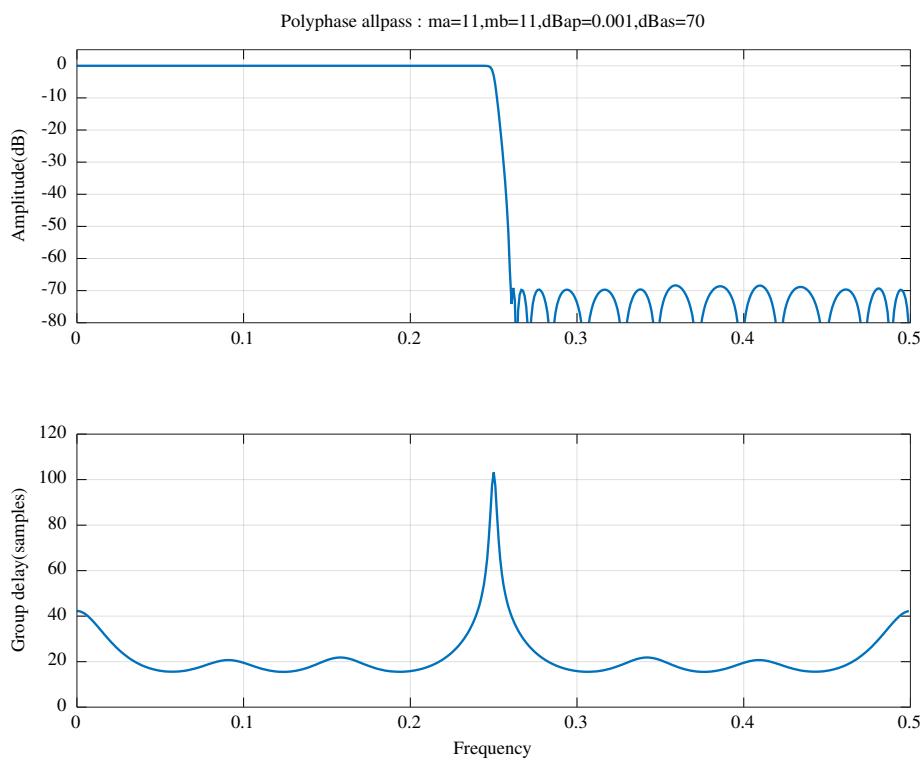


Figure 12.42: Polyphase combination of two allpass filters : Response after PCLS SOCP optimisation

Design of an IIR filter as the polyphase decomposition into two all-pass filters each represented in pole-zero form with constraints on the group delay

The Octave script *polyphase_allpass_socp_slb_flat_delay_test.m* calls the Octave function *parallel_allpass_slb* to design a low-pass filter composed of the polyphase combination of two parallel all-pass filters in terms of the all-pass filter pole locations. The response of the filter is optimised with the PCLS algorithm described in Section 10.1.2. The filter specification is:

```

polyphase=1 % Use polyphase combination
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-07 % Tolerance on constraints
n=500 % Frequency points across the band
ma=11 % Allpass model filter A denominator order
Va=1 % Allpass model filter A no. of real poles
Qa=10 % Allpass model filter A no. of complex poles
Ra=2 % Allpass model filter A decimation
mb=11 % Allpass model filter B denominator order
Vb=1 % Allpass model filter B no. of real poles
Qb=10 % Allpass model filter B no. of complex poles
Rb=2 % Allpass model filter B decimation
fap=0.22 % Pass band amplitude response edge
dBap=0.000005 % Pass band amplitude response ripple
Wap=0 % Pass band amplitude response weight
fas=0.28 % Stop band amplitude response edge
dBas=60.000000 % Stop band amplitude response ripple
Was=500 % Stop band amplitude response weight
ftp=0.22 % Pass band group delay response edge
td=22.02 % Pass band nominal group delay
tdr=0.08 % Pass band nominal group delay ripple
Wtp=1 % Pass band group delay response weight
rho=0.968750 % Constraint on allpass pole radius

```

In this case the script does optimise for a flat filter group delay across the pass band. The nominal pass-band group delay was found by trial-and-error.

The initial parallel allpass filters were designed by the Octave script *tarczynski_polyphase_allpass_test.m* using the WISE method described in Section 10.1.5. In this case the script has *flat_delay = true*. The response of the initial filter is shown in Figure 12.43. After PCLS optimisation with the *SeDuMi* SOCP solver the response is shown in Figure 12.44. The pass band response is shown in Figure 12.45. The denominator polynomials of the two all-pass filters are

```

Da1 = [ 1.0000000000, 0.0000000000, -0.0124595911, -0.0000000000, ...
        0.0058711516, 0.0000000000, -0.0035359582, -0.0000000000, ...
        0.0023544691, 0.0000000000, -0.0016881797, -0.0000000000, ...
        0.0012551890, 0.0000000000, -0.0009219425, -0.0000000000, ...
        0.0005568385, 0.0000000000, -0.0002856279, -0.0000000000, ...
        0.0000004558, 0.0000000000, -0.0001635960 ]';

```

and

```

Db1 = [ 1.0000000000, 0.0000000000, 0.4841852737, 0.0000000000, ...
        -0.1203448092, -0.0000000000, 0.0572996623, 0.0000000000, ...
        -0.0330158562, -0.0000000000, 0.0206100182, 0.0000000000, ...
        -0.0133245377, -0.0000000000, 0.0087223870, 0.0000000000, ...
        -0.0057815254, -0.0000000000, 0.0037161533, 0.0000000000, ...
        -0.0022954704, -0.0000000000, 0.0019208385 ]';

```

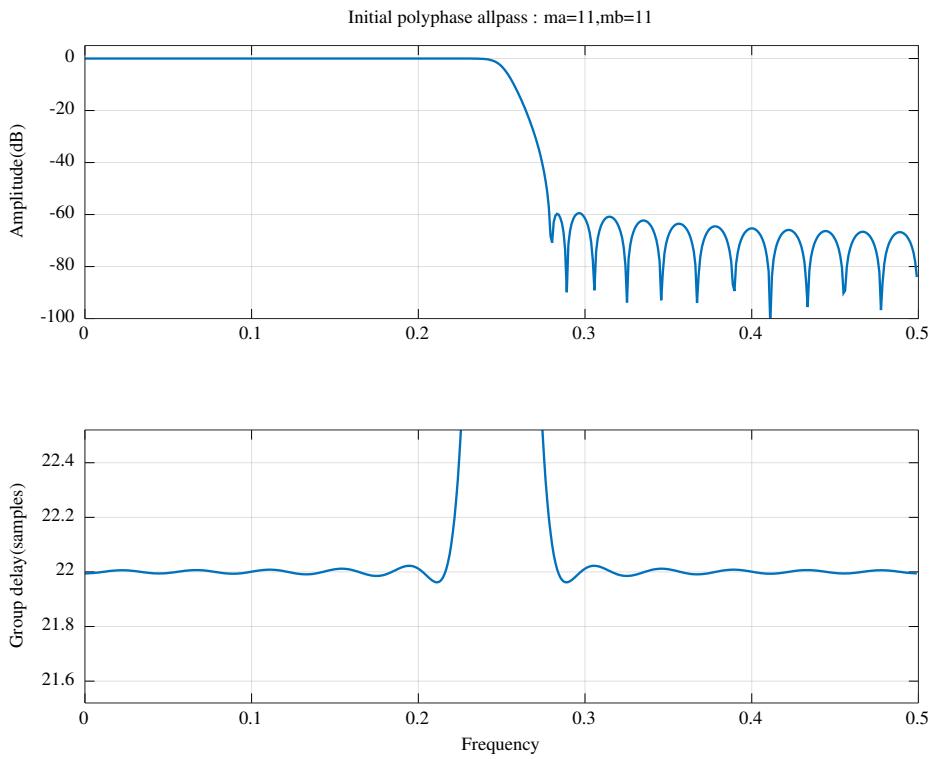


Figure 12.43: Polyphase combination of all-pass filters with flat pass-band delay : Initial response found with the WISE barrier function

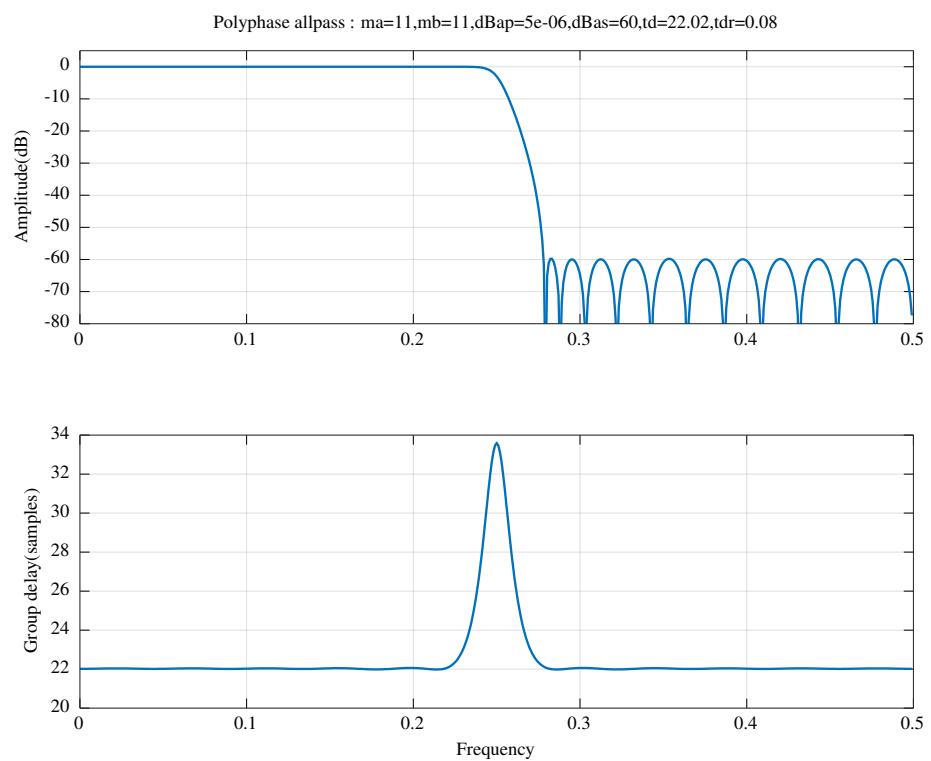


Figure 12.44: Polyphase combination of two all-pass filters with flat pass-band delay : Response after PCLS SOCP optimisation

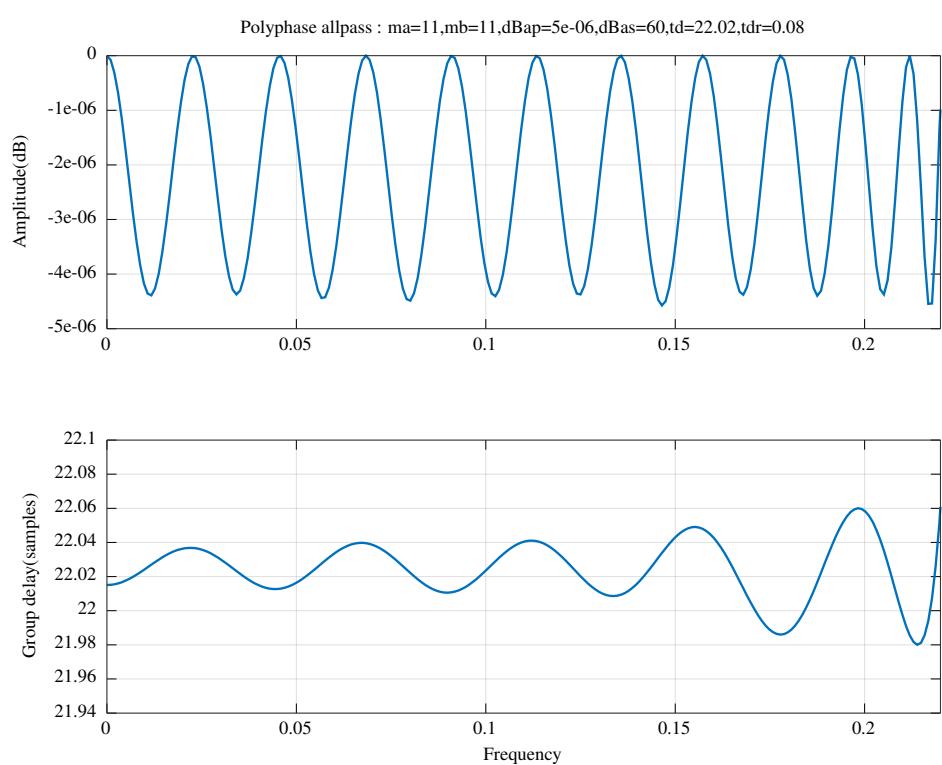


Figure 12.45: Polyphase combination of two all-pass filters with flat pass-band delay : Pass-band response after PCLS SOCP optimisation

12.3 Design of an IIR Schur lattice filter

There is a large literature describing adaptive IIR lattice filters. For example, *Regalia* [68] describes adaptive IIR lattice filter algorithms with $\mathcal{O}(N)$ complexity. The lattice structure has the advantage that the allpass “reflection” coefficients, k_i , have a simple filter stability criterion: $|k_i| \leq 1$. This section describes the PCLS design of the squared magnitude, phase and group delay responses of one-multiplier and normalised scaled IIR Schur lattice filters by SOCP and SQP optimisation of the lattice coefficients. The calculation of the Schur lattice filter squared-magnitude, phase and group delay responses and gradients is calculated in three steps:

1. calculate the state variable representation of the lattice filter and the gradients of the state variable matrixes with respect to each of the lattice coefficients
2. calculate the lattice filter complex frequency response and gradients
3. calculate the lattice filter squared magnitude, phase and group delay responses and gradients

Chapter 7 reviews the Schur decomposition of an IIR transfer function into an IIR tapped-lattice structure. Algorithm 17 shows the factorisation of the state variable description of the *one-multiplier* lattice filter. Algorithm 18 shows the factorisation of the state variable description of the *normalised-scaled* lattice filter. Section 2.9.3 reviews the sensitivities of the complex transfer function with respect to the state variable coefficients. Appendix E shows the gradients of the squared-magnitude, phase and group delay responses with respect to the components of the state variable matrixes. The state-transition matrix, A , generated by Algorithm 17 or Algorithm 18 is lower Hessenberg. The matrix resolvent, $(zI - A)^{-1}$ is calculated by direct matrix inversion rather than Leverrier’s algorithm (see Algorithm 5). Xu Zhong [103] gives an algorithm that calculates the inverse of a lower Hessenberg matrix. That algorithm is implemented in Octave as the function *zhong_inverse.m* and, for a matrix with complex elements, in the *oct-file* *complex_zhong_inverse.cc*. The Octave function *Abcd2H*, exercised by the Octave script *Abcd2H_test.m*, calculates the response of a state variable filter and the gradients of the complex response with respect to frequency and with respect to the components of the state variable matrixes. The Octave function *Abcd2H* assumes that the components of the state variable matrixes are first order combinations of the lattice coefficients. In other words, *Abcd2H* does not support gradients with respect to the square-roots of the k and c lattice coefficients derived in Section 7. The results of this section are intended to be used in the following Part III to optimise the choice of fixed-point lattice coefficients. The k and c lattice coefficients of an “exact” transfer function can be calculated from a transfer function optimised in the *gain-pole-zero* form.

12.3.1 Design of an IIR one-multiplier Schur lattice low-pass filter using SOCP

The Octave script *schurOneMlattice_socp_slb_lowpass_test.m* implements the design of a lowpass IIR one-multiplier Schur lattice filter using the *SeDuMi* SOCP solver with PCLS constraints. The filter specification is similar to that of the *Deczky3* gain-pole-zero SQP and SOCP optimisation examples in Sections 10.2.3 and 11.4:

```

tol=1e-06 % Tolerance on coefficient update vector
n=400 % Frequency points across the band
length(c0)=11 % Tap coefficients
length(k0~=0)=10 % Num. non-zero all-pass coef.s
rho=0.992188 % Constraint on allpass coefficients
fap=0.15 % Amplitude pass band edge
dBap=0.1 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftp=0.25 % Delay pass band edge
tp=10 % Nominal pass band filter group delay
tpr=0.02 % Delay pass band peak-to-peak ripple
Wtp=1 % Delay pass band weight
fas=0.3 % Amplitude stop band edge
dBas=38 % amplitude stop band peak-to-peak ripple
Was=100 % Amplitude stop band weight

```

The initial filter is the “IPZS-1” filter of Section 10.2.3. Figures 12.46 and 12.47 show the overall and passband response after SOCP PCLS optimisation. Figure 12.48 shows the pole-zero plot of the resulting filter after SOCP PCLS optimisation.

The PCLS SOCP optimised Schur one-multiplier all-pass lattice and numerator tap coefficients of the low-pass filter are:

```

k2 = [ -0.6632301475,    0.6513264680,   -0.5324959391,    0.3544483872, ...
       -0.1741518210,    0.0475000230,    0.0000000000,    0.0000000000, ...
       0.0000000000,   -0.0000000000 ];

```

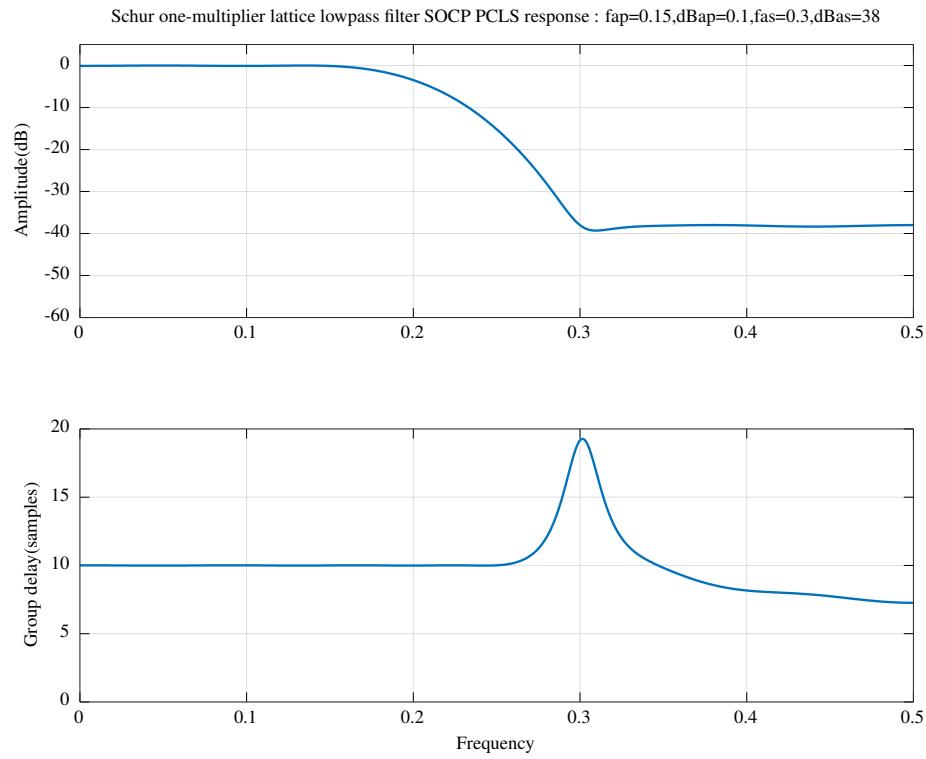


Figure 12.46: Schur one-multiplier lattice lowpass filter : response after PCLS SOCP optimisation

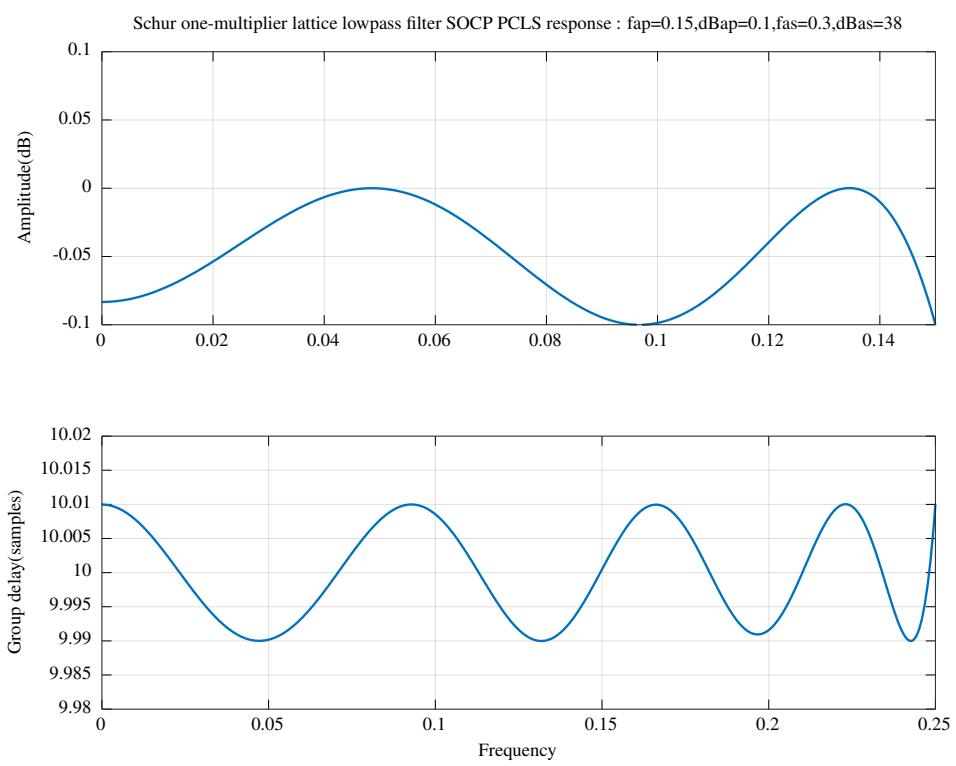


Figure 12.47: Schur one-multiplier lattice lowpass filter : passband response after PCLS SOCP optimisation

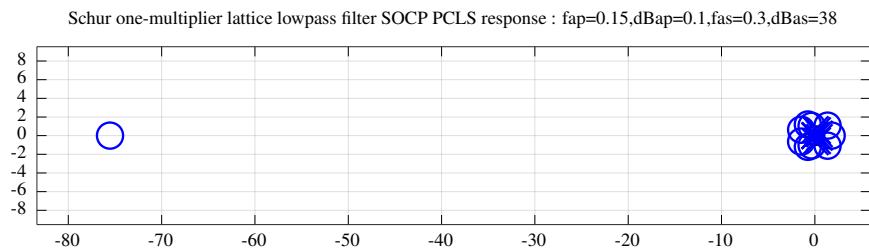


Figure 12.48: Schur one-multiplier lattice lowpass filter : pole-zero plot after PCLS SOCP optimisation

```

epsilon2 = [ -1, -1, -1, -1, ...
            1, -1, -1, -1, ...
           -1,  1 ];

p2 = [ 1.0207920141,    0.4593330479,    0.9996193941,    0.5521126543, ...
        0.7997301072,    0.9535763389,    1.0000000000,    1.0000000000, ...
       1.0000000000,    1.0000000000 ];

c2 = [ 0.4474906376,    0.8866772744,    0.0987277698,   -0.1374366129, ...
        -0.0510516872,    0.0175758447,    0.0222791564,    0.0003803473, ...
       -0.0078482368,   -0.0034971737,   -0.0000449277 ];

```

12.3.2 Design of an IIR one-multiplier Schur lattice low-pass filter using SQP

The Octave script *schurOneMlattice_sqp_slb_lowpass_test.m* implements the design of a lowpass IIR one-multiplier Schur lattice filter with SQP and PCLS. The specification of the filter is:

```

tol=4e-05 % Tolerance on coefficient update vector
n=400 % Frequency points across the band
length(c0)=11 % Tap coefficients
sum(k0~=0)=6 % Num. non-zero lattice coefficients
dmax=0.050000 % Constraint on norm of coefficient SQP step size
rho=0.992188 % Constraint on lattice coefficient magnitudes
fap=0.15 % Amplitude pass band edge
dBap=0.4 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftp=0.25 % Delay pass band edge
tp=10 % Nominal pass band filter group delay
tpr=0.08 % Delay pass band peak-to-peak ripple
Wtp_mmse=0.1 % Delay pass band weight for MMSE
Wtp_pcls=1 % Delay pass band weight for PCLS
fas=0.3 % Amplitude stop band edge
dBas=46 % amplitude stop band peak-to-peak ripple
Was_mmse=1e+08 % Amplitude stop band weight for MMSE
Was_pcls=1e+08 % Amplitude stop band weight for PCLS

```

The initial filter is the “IPZS-1” of Section 10.2.3. As for the examples of Chapter 10 the SQP BFGS update is initialised by the diagonal of the Hessian matrix of the squared error. The diagonals of the Hessian matrixes of the squared-magnitude, phase and group delay are given in Appendix E.

Figures 12.49 and 12.50 show the overall and passband response of the filter after SQP PCLS optimisation. Figure 12.51 shows the pole-zero plot of the filter after SQP PCLS optimisation.

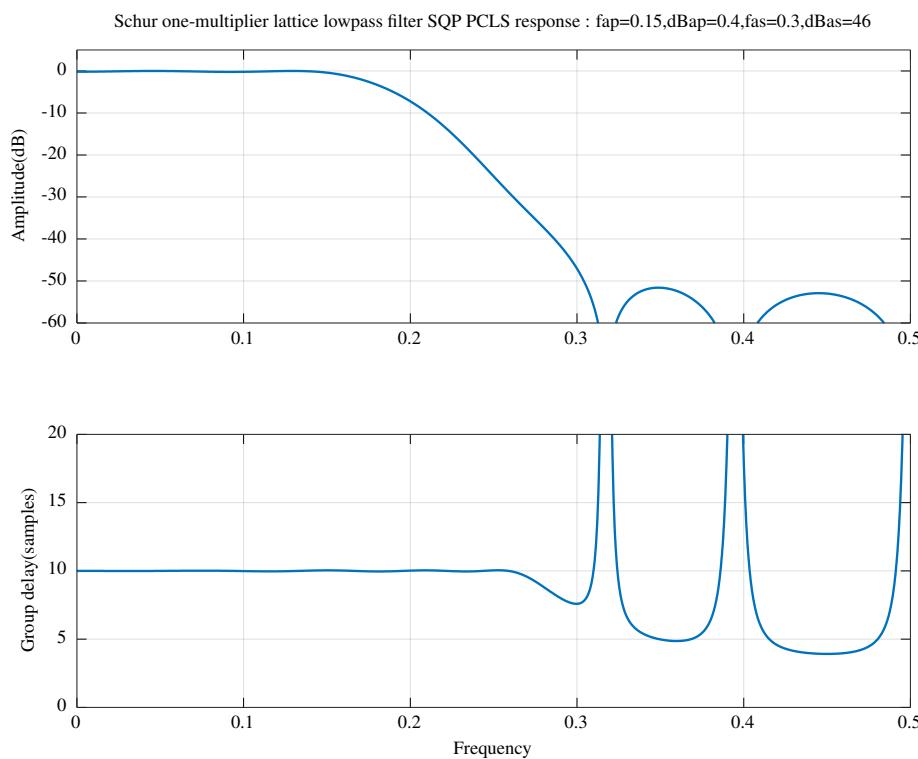


Figure 12.49: Schur one-multiplier lattice lowpass filter : response after SQP PCLS optimisation

The SQP PCLS optimised Schur one-multiplier all-pass lattice and numerator tap coefficients of the low-pass filter are:

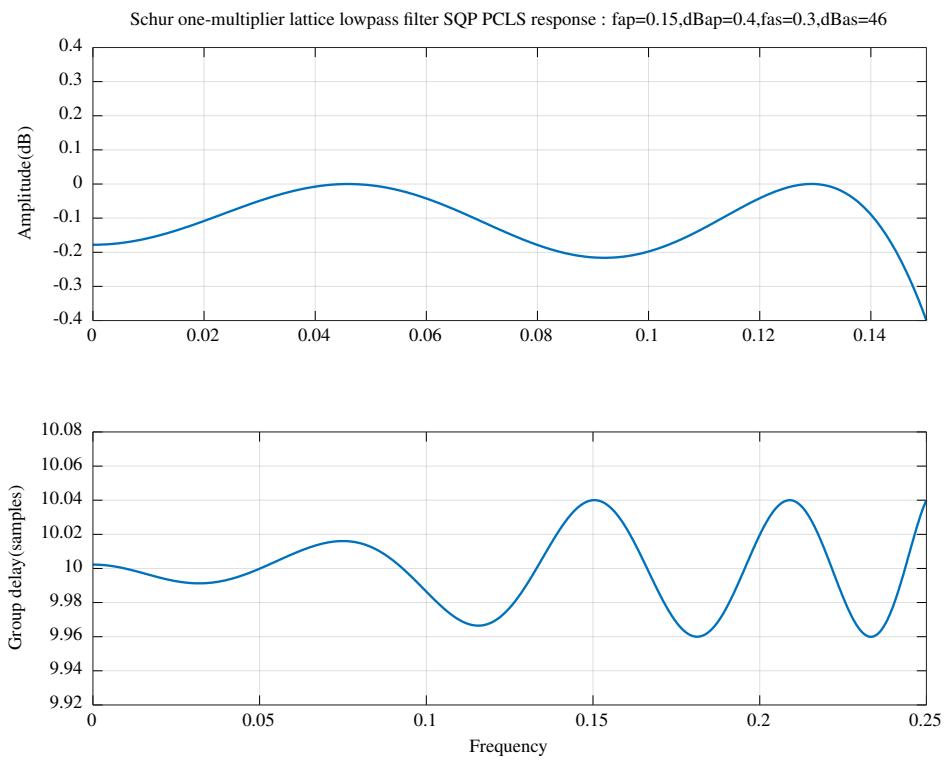


Figure 12.50: Schur one-multiplier lattice lowpass filter : passband response after SQP PCLS optimisation

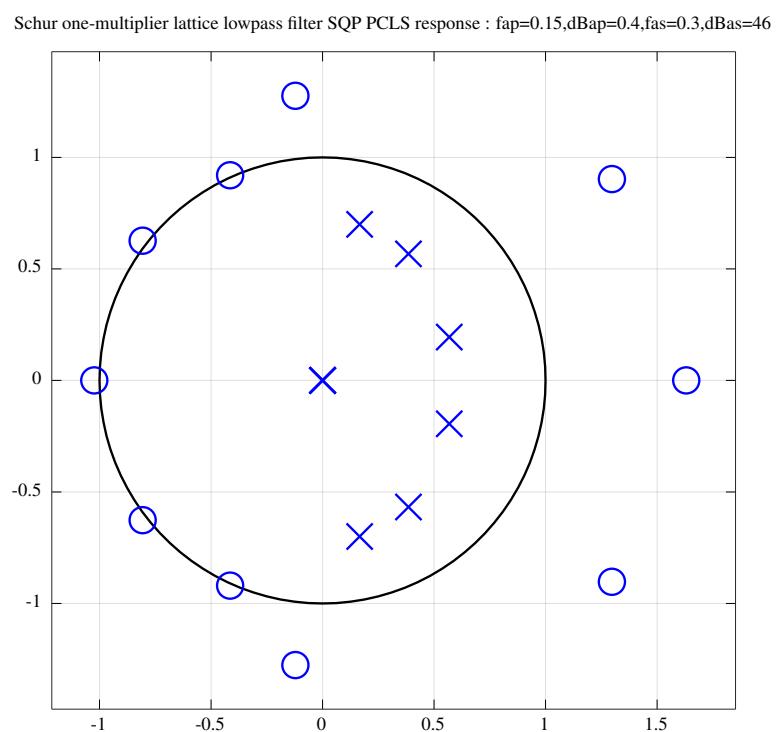


Figure 12.51: Schur one-multiplier lattice lowpass filter : pole-zero plot after SQP PCLS optimisation

```

k2 = [ -0.7375771181,    0.7329492530,   -0.6489740765,    0.4943363006, ...
       -0.2824593372,    0.0878212134,   -0.0000000000,   -0.0000000000, ...
       0.0000000000,    0.0000000000 ];

epsilon2 = [ -1, -1, -1, -1, ...
            -1, -1,  1,  1, ...
            -1, -1 ];

p2 = [  1.5591179441,    0.6059095195,   1.5434899280,    0.7121418051, ...
       1.2242208323,    0.9157168850,   1.0000000000,   1.0000000000, ...
       1.0000000000,    1.0000000000 ];

c2 = [  0.2287455707,    0.7274997803,   0.1030199742,   -0.0659731942, ...
       -0.0487519200,   -0.0063882219,   0.0239044428,   0.0157765877, ...
       -0.0023124161,   -0.0090627662,   -0.0052625492 ];

```

12.3.3 Design of an IIR one-multiplier Schur lattice band-pass filter using SQP

The Octave script *schurOneMlattice_sqp_slb_bandpass_test.m* implements the design of a band-pass IIR one-multiplier Schur lattice filter with SQP and PCLS. The specification of the filter is:

```

tol_mmse=2e-05 % Tolerance on coef. update for MMSE
tol_pcls=2e-05 % Tolerance on coef. update for PCLS
n=500 % Frequency points across the band
length(c0)=21 % Tap coefficients
sum(k0~=0)=10 % Num. non-zero all-pass coef.s
dmax=0.050000 % Constraint on norm of coefficient SQP step size
rho=0.992188 % Constraint on allpass coefficients
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftpl=0.09 % Delay pass band lower edge
ftpdu=0.21 % Delay pass band upper edge
tp=16 % Nominal passband filter group delay
tpr=0.08 % Delay pass band peak-to-peak ripple
Wtp_mmse=6 % Delay pass band weight(MMSE)
Wtp_pcls=6 % Delay pass band weight(PCLS)
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
dBas=36 % Amplitude stop band peak-to-peak ripple
Wasl_mmse=1000000 % Ampl. lower stop band weight(MMSE)
Wasu_mmse=1000000 % Ampl. upper stop band weight(MMSE)
Wasl_pcls=1000000 % Ampl. lower stop band weight(PCLS)
Wasu_pcls=1000000 % Ampl. upper stop band weight(PCLS)

```

The initial filter is that of the Octave script *iir_sqp_slb_bandpass_test.m*, as shown in Section 10.2.6. The denominator polynomial of the filter has coefficients in z^2 only.

Figure 12.52 shows the overall response of the band-pass filter after SQP MMSE optimisation. Figures 12.53 and 12.54 show the overall and passband response of the band-pass filter after SQP PCLS optimisation. Figure 12.55 shows the pole-zero plot of the band-pass filter after SQP PCLS optimisation.

The SQP PCLS optimised Schur one-multiplier allpass lattice and numerator tap coefficients of the band-pass filter are:

```

k2 = [ 0.0000000000, 0.6578120679, 0.0000000000, 0.5164877092, ...
        0.0000000000, 0.3527537927, 0.0000000000, 0.4280317934, ...
        0.0000000000, 0.2989556994, 0.0000000000, 0.2525473176, ...
        0.0000000000, 0.1498954992, 0.0000000000, 0.1010893608, ...
        0.0000000000, 0.0332806170, 0.0000000000, 0.0134002323 ];

epsilon2 = [ 0, 1, 0, -1, ...
             0, 1, 0, -1, ...
             0, 1, 0, -1, ...
             0, -1, 0, 1, ...
             0, -1, 0, -1 ];

p2 = [ 1.0859542638, 1.0859542638, 0.4933739682, 0.4933739682, ...
        0.8737595176, 0.8737595176, 0.6043900367, 0.6043900367, ...
        0.9549932244, 0.9549932244, 0.7015778220, 0.7015778220, ...
        0.9081991060, 0.9081991060, 1.0562679428, 1.0562679428, ...
        0.9543794437, 0.9543794437, 0.9866883596, 0.9866883596 ];

c2 = [ 0.0751717907, -0.0082427906, -0.2880852630, -0.4887366083, ...
        -0.1709389778, 0.1065369928, 0.3845903475, 0.3074219368, ...
        0.0242136637, -0.0804302473, -0.0844066957, -0.0154037568, ...
        -0.0063029471, -0.0302681729, -0.0243108556, 0.0037923197, ...
        0.0245894675, 0.0177004014, 0.0024010069, -0.0015285861, ...
        0.0023859122 ];

```

A second PCLS pass with the constraints $tpr = 0.06$ and $Wtp_pcls = 8$ succeeds (eventually). Alternatively, the Octave script *schurOneMlattice_socp_slb_bandpass_test.m* runs a PCLS SOCP pass starting with the $k2$, ϵ_2 , $p2$ and $c2$ found above and the constraints $dB_{as} = 37$, $tpr = 0.05$ and $Wtp_pcls = 6$.

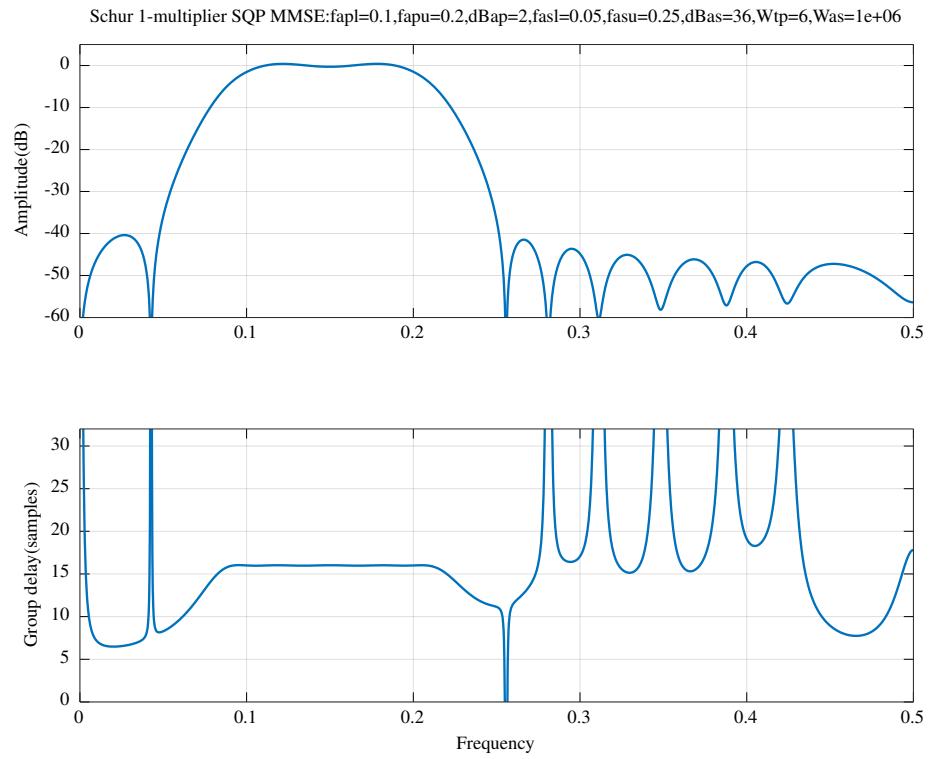


Figure 12.52: Schur one-multiplier lattice band-pass filter : response after SQP MMSE optimisation

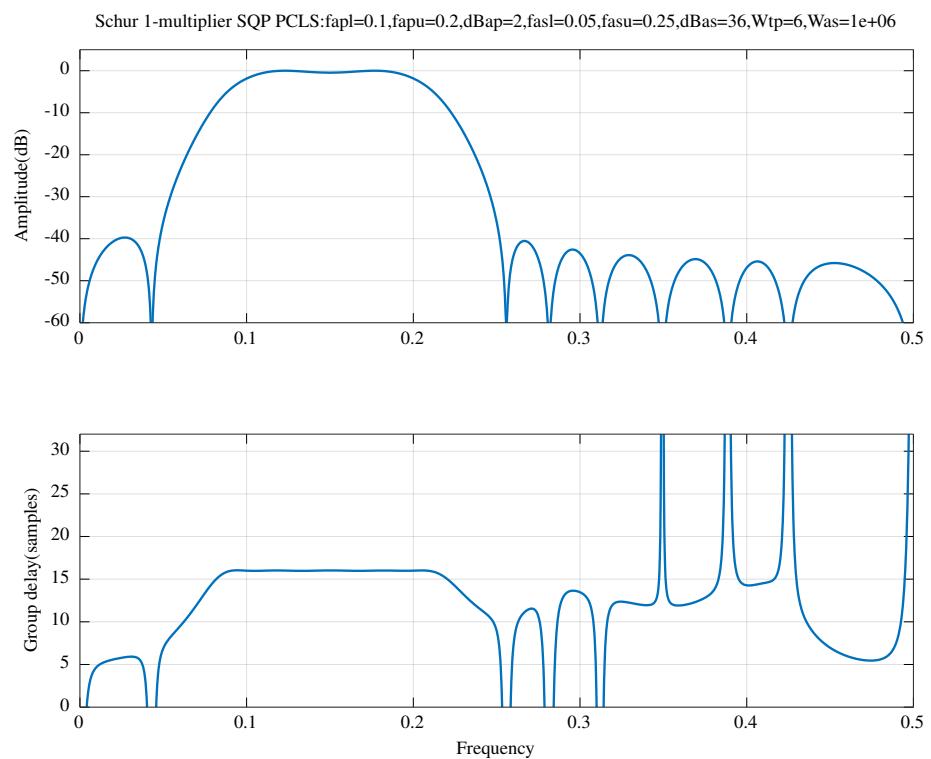


Figure 12.53: Schur one-multiplier lattice band-pass filter : response after SQP PCLS optimisation

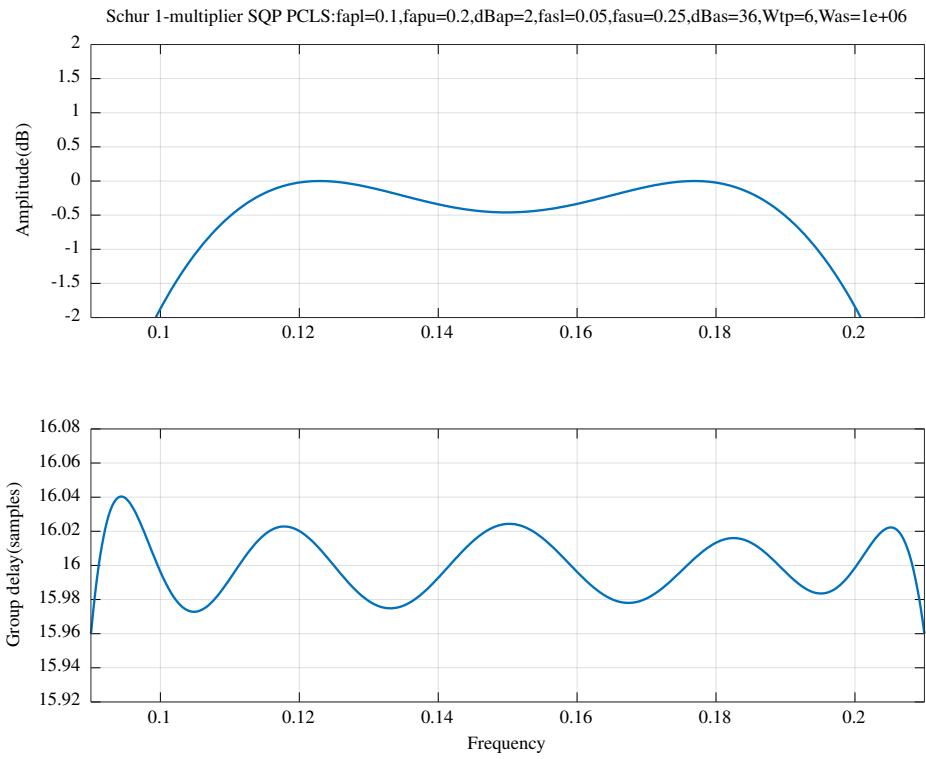


Figure 12.54: Schur one-multiplier lattice band-pass filter : passband response after SQP PCLS optimisation

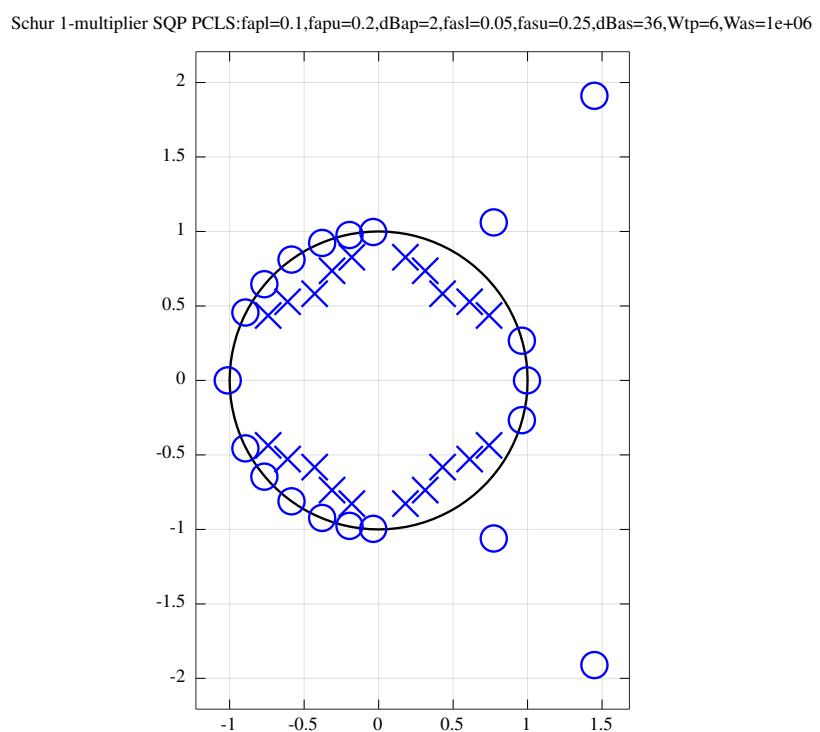


Figure 12.55: Schur one-multiplier lattice band-pass filter : pole-zero plot after SQP PCLS optimisation

12.3.4 Design of an IIR one-multiplier Schur lattice Hilbert filter using SQP

The Octave script *schurOneMlattice_sqp_slb_hilbert_test.m* implements the design of an IIR one-multiplier Schur lattice Hilbert filter with SQP and PCLS. The specification of the filter is:

```

tol=0.0001 % Tolerance on coefficient update vector
n=400 % Frequency points across the band
dmax=0.100000 % Constraint on norm of coefficient SQP step size
rho=0.999900 % Constraint on lattice coefficient magnitudes
ft1=0.05 % First transition band [0,ft1]
ft2=0.075 % Second transition band [ft1,ft2]
dBap=0.1 % Amplitude pass band peak-to-peak ripple
Wat=0.05 % Amplitude transition band weight
Wap=1 % Amplitude pass band weight
tp=5.5 % Nominal pass band filter group delay
tpr=0.5 % Pass band filter group delay peak-to-peak ripple
Wtp=0.005 % Pass band group delay weight
pr=0.01 pi/2 % Pass band peak-to-peak phase ripple
Wpp=1 % Pass band phase weight

```

The initial filter is that of the Octave script *iir_sqp_slb_hilbert_test.m* designed by the method of Tarczynski et. al. with the Octave script *tarczynski_hilbert_test.m*, as shown in Section 10.2.7. The denominator polynomial of the filter has coefficients in z^2 only. Figure 12.56 shows the response of the Hilbert filter after SQP PCLS optimisation. The phase response shown has been adjusted by the nominal group delay, $\omega\tau_p$. The values shown on the phase axis of the phase response plot are multiples of $\frac{\pi}{2}$. Figure 12.57 shows the pole-zero plot of the Hilbert filter after SQP PCLS optimisation.

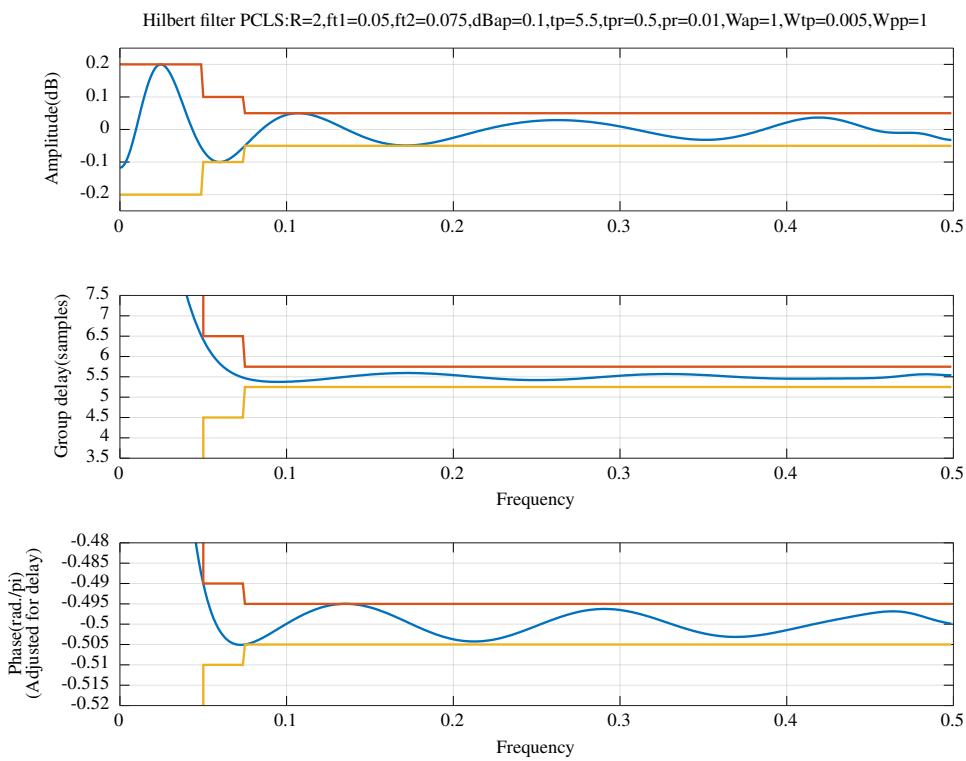


Figure 12.56: Schur one-multiplier lattice Hilbert filter : response after SQP PCLS optimisation

The SQP PCLS optimised Schur one-multiplier all-pass lattice and numerator tap coefficients of the Hilbert filter are:

```

k2 = [ 0.0000000000, -0.9135081383, 0.0000000000, 0.5546428065, ...
0.0000000000, -0.0966755700, 0.0000000000, 0.0007462345, ...
0.0000000000, -0.0020582169, 0.0000000000, 0.0015840382 ];

```

Hilbert filter PCLS:R=2,ft1=0.05,ft2=0.075,dBap=0.1,tp=5.5,tpr=0.5,pr=0.01,Wap=1,Wtp=0.005,Wpp=1

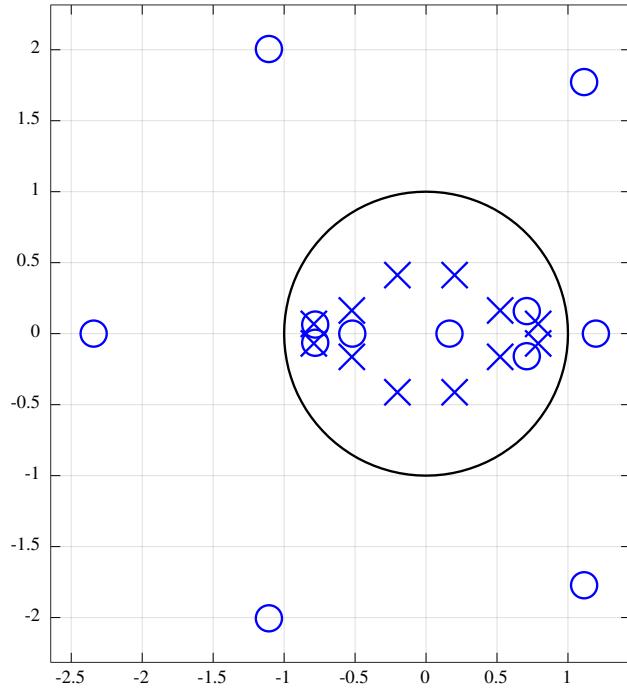


Figure 12.57: Schur one-multiplier lattice Hilbert filter : pole-zero plot after SQP PCLS optimisation

```

epsilon2 = [ 0, -1, 0, -1, ...
            0, 1, 0, -1, ...
            0, 1, 0, -1 ];

p2 = [ 2.2748007025, 2.2748007025, 0.4836328807, 0.4836328807, ...
        0.9036013990, 0.9036013990, 0.9956211215, 0.9956211215, ...
        0.9963643657, 0.9963643657, 0.9984172144, 0.9984172144 ];

c2 = [ 0.0435189924, 0.0508921396, 0.2428519491, 0.2882277479, ...
        0.1877845008, 0.2854412257, 0.6862739755, -0.5909939642, ...
        -0.1639093077, -0.0790058478, -0.0435858017, -0.0252833112, ...
        -0.0155222725 ];

```

Alternatively, the Octave script *schurOneMlattice_socp_slb_hilbert_test.m* implements the design of an IIR one-multiplier Schur lattice Hilbert filter with SOCP and PCLS.

12.3.5 Design of an IIR Schur normalised-scaled lattice low-pass filter using SQP

The Octave script *schurNSlattice_sqp_slb_lowpass_test.m* implements the design of a lowpass IIR normalised-scaled structure Schur lattice filter with SQP and PCLS. The filter coefficients are allowed to vary independently and the resulting filter is not, in fact, normalised-scaled. The specification of the filter is:

```

tol_mmse=0.001 % Tolerance on coefficient update vector for MMSE
tol_pcls=1e-05 % Tolerance on coefficient update vector for PCLS
n=400 % Frequency points across the band
sxx_symmetric=0 % Enforce s02=-s20 and s22=s00
dmax=0.050000 % Constraint on norm of coefficient SQP step size
rho=0.999990 % Constraint on lattice coefficient magnitudes
fap=0.15 % Amplitude pass band edge
dBap=0.2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftp=0.25 % Delay pass band edge
tp=9.5 % Nominal pass band filter group delay
tpr=0.1 % Delay pass band peak-to-peak ripple
Wtp_mmse=0.01 % Delay pass band weight for MMSE
Wtp_pcls=0.1 % Delay pass band weight for PCLS
fas=0.3 % Amplitude stop band edge
dBas=46 % amplitude stop band peak-to-peak ripple
Was_mmse=10000 % Amplitude stop band weight for MMSE
Was_pcls=1e+06 % Amplitude stop band weight for PCLS

```

The initial filter is the “IPZS-1” of Section 10.2.3. As for the examples of Chapter 10 the SQP BFGS update is initialised by the diagonal of the Hessian matrix of the squared error.

Figures 12.58 and 12.59 show the overall and passband response of the filter after SQP PCLS optimisation. Figure 12.60 shows the pole-zero plot of the filter after SQP PCLS optimisation.

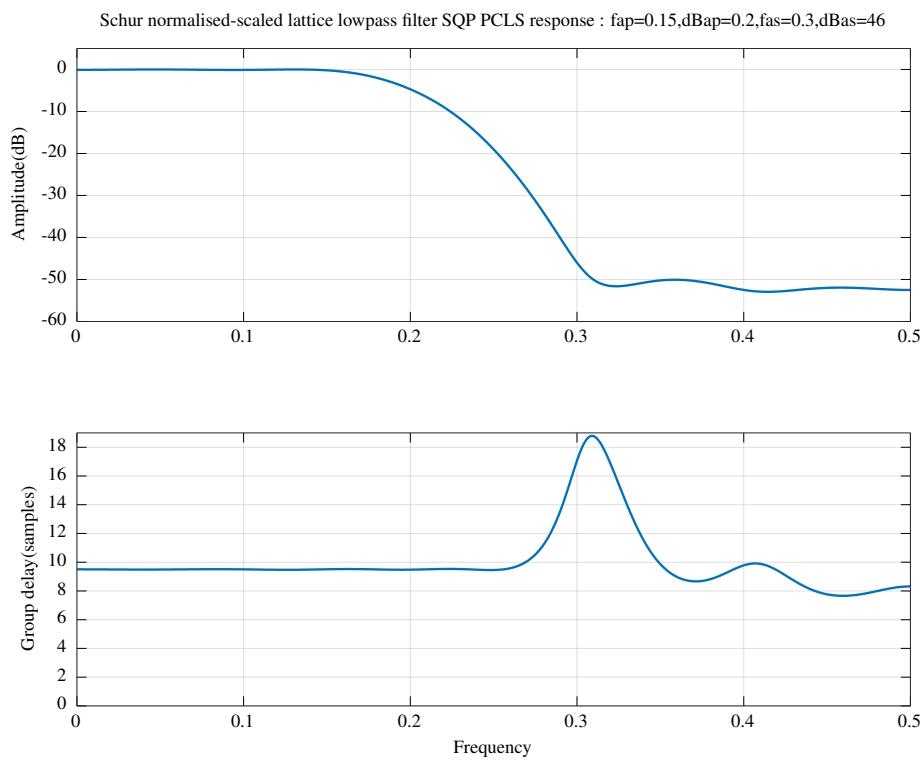


Figure 12.58: Schur normalised-scaled lattice lowpass filter : response after SQP PCLS optimisation

The SQP PCLS optimised Schur normalised-scaled all-pass lattice and numerator tap coefficients of the low-pass filter are:

```

s10_2 = [ 1.2139594607, 0.4304107497, -0.0657111836, -0.1192373164, ...
-0.0293352869, 0.0362349326, 0.0262123098, -0.0035236357, ...
-0.0110221712, -0.0028759866 ];

```

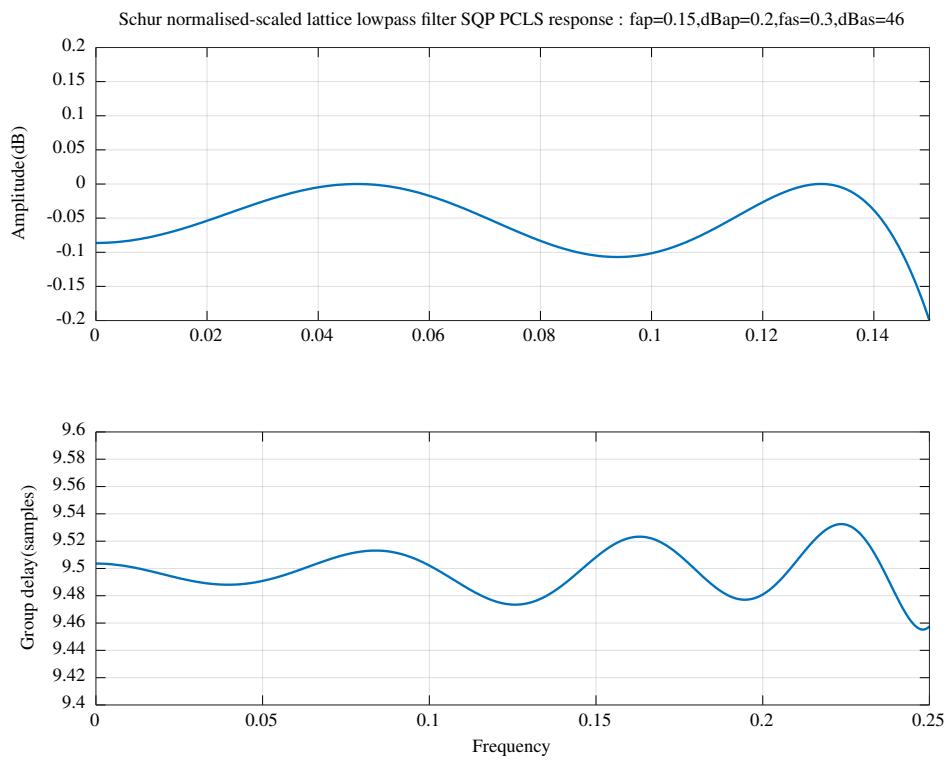


Figure 12.59: Schur normalised-scaled lattice lowpass filter : passband response after SQP PCLS optimisation

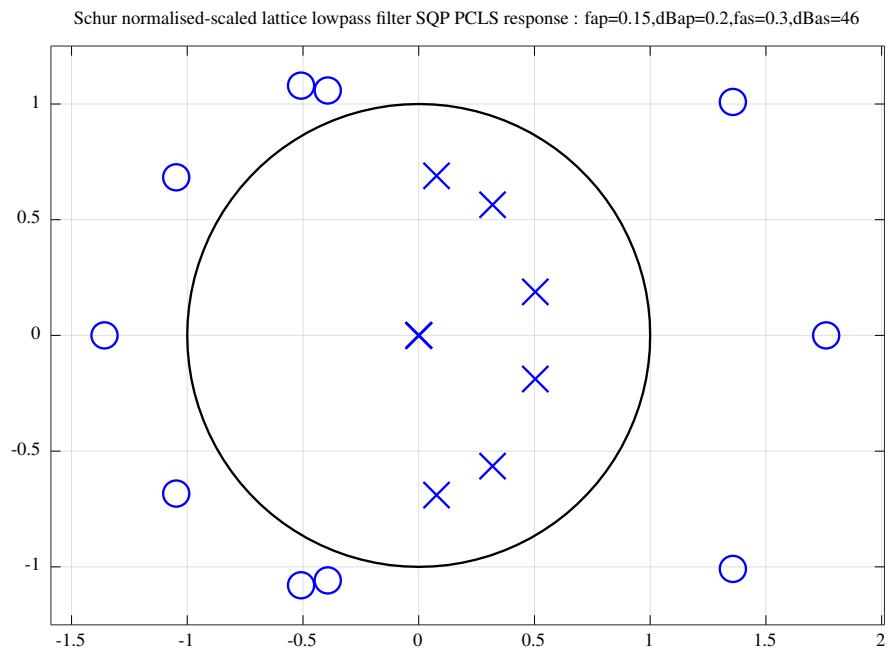


Figure 12.60: Schur normalised-scaled lattice lowpass filter : pole-zero plot after SQP PCLS optimisation

```

s11_2 = [ 0.8919273371, 0.9504891000, 1.0238729354, 0.9816052186, ...
          1.0024899597, 0.9974867686, 0.9867379008, 0.9784074832, ...
          0.9798971101, 0.6734090254 ];

s20_2 = [ -0.5844569147, 0.7162787641, -0.5418853725, -0.1260310922, ...
          0.9385382288, 0.0441000000, 0.0000000000, 0.0000000000, ...
          0.0000000000, 0.0000000000 ];

s00_2 = [ 0.7910940345, 0.6134643655, 0.7865221075, 0.8637318641, ...
          0.9722121569, 0.9974928780, 1.0000000000, 1.0000000000, ...
          1.0000000000, 1.0000000000 ];

s02_2 = [ 0.8659608466, -0.7738763974, 0.6742136699, -0.5215951411, ...
          0.3939054328, -0.2505521537, -0.0000000000, -0.0000000000, ...
          -0.0000000000, -0.0000000000 ];

s22_2 = [ 0.7710543164, 0.9130256527, 0.9672191401, 0.9355878103, ...
          0.6110976463, 0.9990271218, 1.0000000000, 1.0000000000, ...
          1.0000000000, 1.0000000000 ];

```

The diagonal of the state covariance matrix, K , of the optimised filter is:

```

diag(K) = [ 0.6029,      0.5846,      0.8568,      1.0115, ...
            1.1926,      1.0839,      1.0000,      1.0000, ...
            1.0000,      1.0000 ]';

```

The SQP loop for this example is called from the Octave function *schurNSlattice_sqp_mmse* exercised by the Octave script *schurNSlattice_sqp_mmse_test.m*. The *schurNSlattice_sqp_mmse* function includes code that forces $s_{02} = -s_{20}$ and $s_{22} = s_{00}$. Enabling this code in the test script results in a filter with worse state-variable scaling.

12.4 Design of IIR filters with a sharp transition band by frequency response masking

12.4.1 Review of Frequency Response Masking digital filters

Frequency response masking (FRM) is a technique for designing digital filters with sharp transition bands. Given a prototype or “model” low-pass filter, $G(z)$, the filter $G(z^M)$ has a pass-band width and pass-band to stop-band transition width that is reduced by a factor M compared to the model low-pass filter and also has M images across the whole frequency band. The “masking” filter, $F(z)$ selects the required filter image frequency response. The resulting filter is $G(z^M)F(z)$, illustrated in Figure 12.61. This simple frequency response masking filter technique is only suitable for designing narrow-band filters. Lim [104] describes

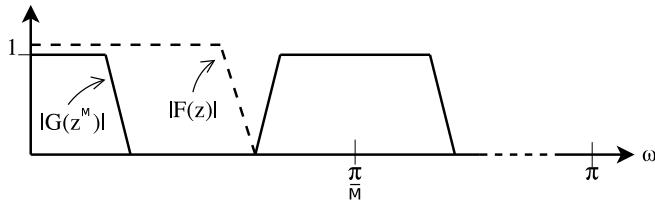


Figure 12.61: Simple frequency response masking filter

the design of digital filters with sharp transition bands with wider bandwidths using FIR model and masking filters. Lu and Hinamoto [95] describe the design of digital filters with an IIR model filter and FIR masking filters using a structure, shown in Figure 12.62, that is similar to Lim’s. The passband delay of $F_a(z)$ is D so that $F_c = z^{-D} - F_a(z)$ is complementary to $F_a(z)$.

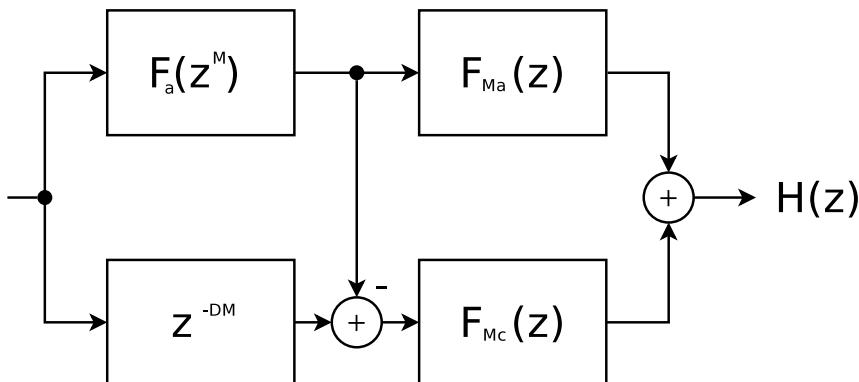


Figure 12.62: Lim’s frequency response masking filter structure

Johansson and Wanhammar [34] describe design of frequency response masking filters with a model filter consisting of parallel allpass filters, FIR masking filters and the structure shown in Figure 12.63. Johansson and Wanhammar specify a delay line in one arm of the model filter if approximately linear phase response is desired.

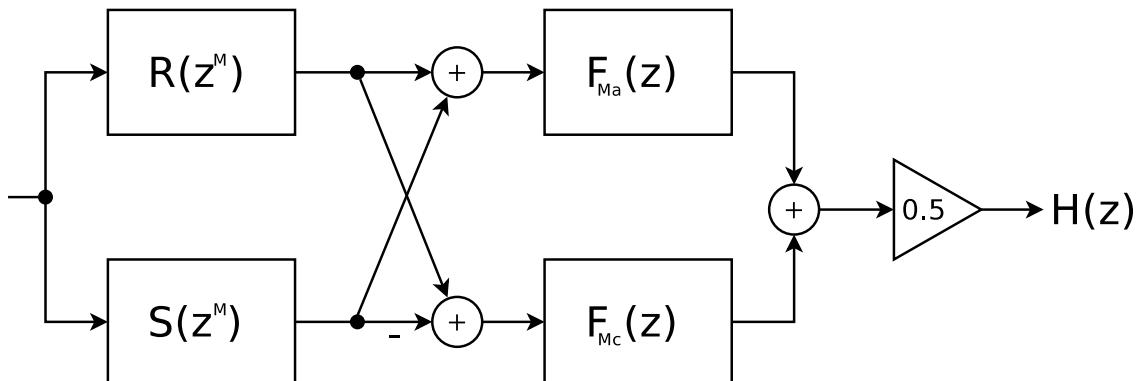


Figure 12.63: Johansson and Wanhammar’s frequency response masking filter structure

In the following I use Lim's description of frequency response masking for the two cases shown in Figure 12.64 [104, Section III, Figure 4]. Figure 12.64a shows the response of a prototype filter a with frequency response, $F_a(\omega)$, having passband and stopband edge frequencies of θ and ϕ respectively. Figure 12.64b shows the response of the complementary filter $F_c(z) = z^{-D} - F_a(z)$. Figure 12.64c shows the responses of the filters $F'_a(z) = F_a(z^M)$ and $F'_c(z) = F_c(z^M)$. Each response is aliased into M images across the frequency band. The aliased response of the model filter is masked by F_{M_a} and the aliased response of the complement to the model filter is masked by F_{M_c} . For example, in Figure 12.64d, F_{M_a} has passband and stopband edge frequencies of $\omega_{M_{ap}} = \frac{2m\pi+\theta}{M}$ and $\omega_{M_{as}} = \frac{2(m+1)\pi-\phi}{M}$ respectively. Here $\omega_{M_{ap}}$ includes the m 'th image band of the model filter and $\omega_{M_{as}}$ excludes the $m + 1$ 'th image band of the model filter. The complementary masking filter, F_{M_c} , has passband and stopband edge frequencies of $\omega_{M_{cp}} = \frac{2m\pi-\theta}{M}$ and $\omega_{M_{cs}} = \frac{2m\pi+\phi}{M}$ respectively so that the $m - 1$ 'th image of the complementary filter is included and the m 'th image is excluded. The resulting frequency response masking filter, shown in Figure 12.64e, has passband edge $\omega_p = \frac{2m\pi+\phi}{M}$ and stopband edge $\omega_s = \frac{2m\pi+\phi}{M}$. Alternatively, Figures 12.64f and 12.64g show the frequency response masking filter for which the passband edge is $\omega_p = \frac{2m\pi-\phi}{M}$ and the stopband edge is $\omega_s = \frac{2m\pi-\theta}{M}$. In this case the filter includes $m - 1$ images of the model filter and $m - 1$ images of the complementary filter.

In order that $0 < \theta < \phi < \pi$ in Figure 12.64a, for Figure 12.64e

$$\begin{aligned} m &= \lfloor \frac{\omega_p M}{2\pi} \rfloor \\ \theta &= \omega_p M - 2m\pi \\ \phi &= \omega_s M - 2m\pi \end{aligned}$$

and for Figure 12.64g

$$\begin{aligned} m &= \lceil \frac{\omega_s M}{2\pi} \rceil \\ \theta &= 2m\pi - \omega_s M \\ \phi &= 2m\pi - \omega_p M \end{aligned}$$

where $\lfloor x \rfloor$ and $\lceil x \rceil$ represent the largest integer less than x and the smallest integer larger than x respectively.

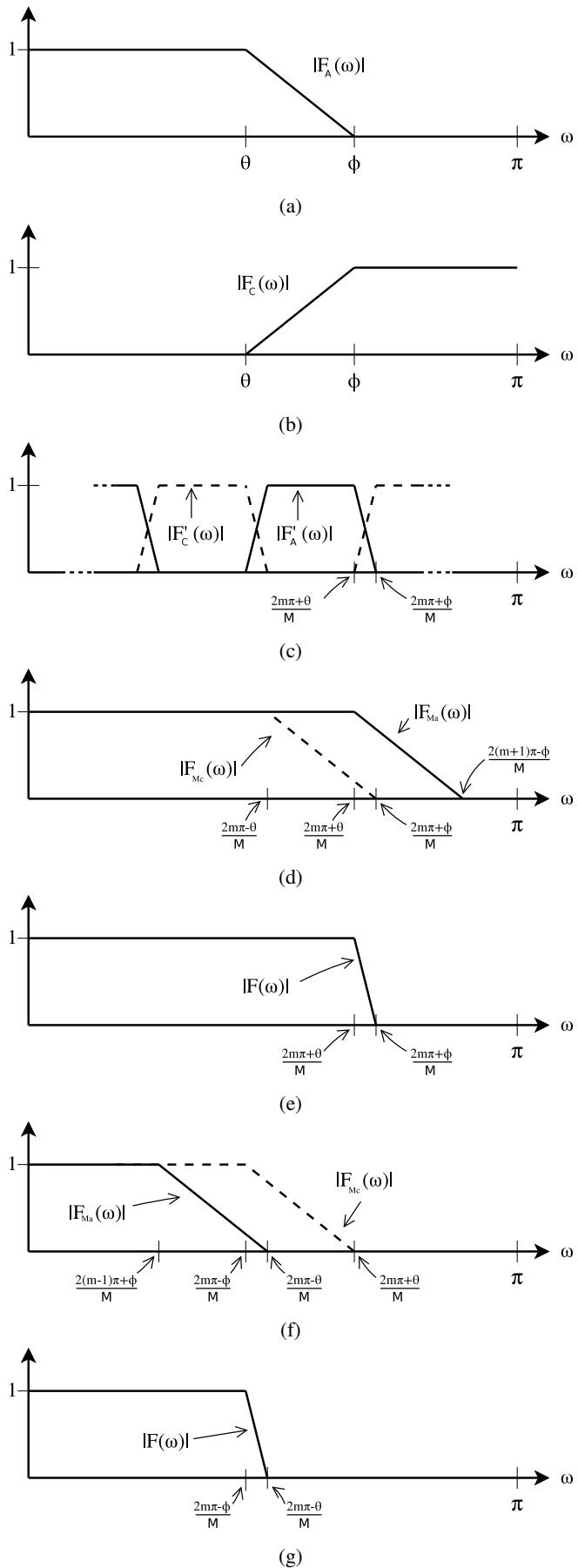


Figure 12.64: Frequency response masking with Lim's filter structure

12.4.2 Design of an FRM digital filter with an IIR model filter consisting of a cascade of second-order sections using SOCP

In this section I follow Lu and Hinamoto [95, Section V] The FRM filter structure is shown in Figure 12.62. The model filter is an IIR filter of the form

$$F_a(z) = \frac{a(z)}{d(z)}$$

where

$$a(z) = \sum_{k=0}^n a_k z^{-k}$$

and $d(z)$ is a product of second order sections (with one first order section if r is odd)

$$d(z) = \begin{cases} (1 + d_0 z^{-1}) \prod_{k=1}^{\frac{r-1}{2}} (1 + d_{k1} z^{-1} + d_{k2} z^{-2}), & \text{if } r \text{ odd} \\ \prod_{k=1}^{\frac{r}{2}} (1 + d_{k1} z^{-1} + d_{k2} z^{-2}), & \text{if } r \text{ even} \end{cases}$$

Define the two model filter coefficient vectors as

$$\mathbf{a} = [a_0 \ a_1 \ \dots \ a_n]^T$$

and

$$\begin{aligned} \mathbf{d} &= \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_L \end{bmatrix} \\ \mathbf{d}_k &= \begin{bmatrix} d_{k1} \\ d_{k2} \end{bmatrix}, \quad \text{for } 1 \leq k \leq L \\ L &= \begin{cases} \frac{r-1}{2}, & \text{if } r \text{ odd} \\ \frac{r}{2}, & \text{if } r \text{ even} \end{cases} \end{aligned}$$

The masking filters are

$$\begin{aligned} F_{M_a}(z) &= \sum_{k=0}^{n_a-1} p_k z^{-k} \\ F_{M_c}(z) &= \sum_{k=0}^{n_c-1} q_k z^{-k} \end{aligned}$$

$F_{M_a}(z)$ and $F_{M_c}(z)$ are assumed to have linear phase responses; the lengths n_a and n_c are assumed to be both even or odd; and the group delays of $F_{M_a}(z)$ and $F_{M_c}(z)$ are both $d = \max\left\{\frac{n_a-1}{2}, \frac{n_c-1}{2}\right\}$ (if necessary, one filter is padded with extra delays).

The frequency response of the symmetric, linear phase, FIR masking filters is (in the case of F_{M_a})

$$\begin{aligned} F_{M_a}(\omega) &= \sum_{k=0}^{n_a-1} p_k e^{-i\omega k} \\ &= e^{-i\omega \frac{n_a-1}{2}} \begin{cases} p_{\frac{n_a-1}{2}} + 2 \sum_{k=1}^{\frac{n_a-1}{2}} p_{k+\frac{n_a-1}{2}} \cos k\omega, & \text{if } n_a \text{ odd} \\ 2 \sum_{k=1}^{\frac{n_a}{2}} p_{k+\frac{n_a}{2}-1} \cos(k - \frac{1}{2})\omega, & \text{if } n_a \text{ even} \end{cases} \end{aligned}$$

The desired passband group delay of the FRM filter is $D_s = d + MD$ where D is the group delay of the model filter. The desired frequency response of the FRM filter can be expressed as

$$e^{-iD_s \omega} H(\mathbf{x}, \omega)$$

where

$$H(\mathbf{x}, \omega) = \tilde{H}_a(M\omega) [\mathbf{a}_a^T \mathbf{c}_a(\omega) - \mathbf{a}_c^T \mathbf{c}_c(\omega)] + \mathbf{a}_c^T \mathbf{c}_c(\omega)$$

$$\begin{aligned}
\tilde{H}_a(\omega) &= e^{jD\omega} \frac{a(\omega)}{d(\omega)} = \frac{\mathbf{a}^T \mathbf{v}(\omega)}{d(\omega)} \\
\mathbf{v}(\omega) &= \mathbf{c}(\omega) + j\mathbf{s}(\omega) \\
\mathbf{c}(\omega) &= [\cos D\omega \quad \dots \quad \cos(D-n)\omega]^T \\
\mathbf{s}(\omega) &= [\sin D\omega \quad \dots \quad \sin(D-n)\omega]^T \\
v_1(\omega) &= \cos \omega - j \sin \omega \\
\mathbf{v}_2(\omega) &= \begin{bmatrix} \cos \omega \\ \cos 2\omega \end{bmatrix} - j \begin{bmatrix} \sin \omega \\ \sin 2\omega \end{bmatrix}
\end{aligned}$$

and

$$\begin{aligned}
d(\omega) &= \begin{cases} [1 + d_0 v_1(\omega)] \prod_{k=1}^L [1 + \mathbf{d}_k^T \mathbf{v}_2(\omega)], & \text{if } r \text{ odd} \\ \prod_{k=1}^L [1 + \mathbf{d}_k^T \mathbf{v}_2(\omega)], & \text{if } r \text{ even} \end{cases} \\
\mathbf{a}_a &= \begin{cases} \begin{bmatrix} p_{\frac{n_a-1}{2}} & p_{\frac{n_a+1}{2}} & \dots & p_{n_a-1} \end{bmatrix}^T, & \text{if } n_a \text{ odd} \\ \begin{bmatrix} p_{\frac{n_a}{2}} & p_{\frac{n_a}{2}+1} & \dots & p_{n_a-1} \end{bmatrix}^T, & \text{if } n_a \text{ even} \end{cases} \\
\mathbf{c}_a(\omega) &= \begin{cases} \begin{bmatrix} 1 & 2 \cos \omega & \dots & 2 \cos \frac{(n_a-1)}{2} \omega \end{bmatrix}^T, & \text{if } n_a \text{ odd} \\ \begin{bmatrix} 2 \cos \frac{1}{2}\omega & \dots & 2 \cos \frac{(n_a-1)}{2} \omega \end{bmatrix}^T, & \text{if } n_a \text{ even} \end{cases} \\
\mathbf{a}_c &= \begin{cases} \begin{bmatrix} q_{\frac{n_c-1}{2}} & q_{\frac{n_c+1}{2}} & \dots & q_{n_c-1} \end{bmatrix}^T, & \text{if } n_c \text{ odd} \\ \begin{bmatrix} q_{\frac{n_c}{2}} & q_{\frac{n_c}{2}+1} & \dots & q_{n_c-1} \end{bmatrix}^T, & \text{if } n_c \text{ even} \end{cases} \\
\mathbf{c}_c(\omega) &= \begin{cases} \begin{bmatrix} 1 & 2 \cos \omega & \dots & 2 \cos \frac{(n_c-1)}{2} \omega \end{bmatrix}^T, & \text{if } n_c \text{ odd} \\ \begin{bmatrix} 2 \cos \frac{1}{2}\omega & \dots & 2 \cos \frac{(n_c-1)}{2} \omega \end{bmatrix}^T, & \text{if } n_c \text{ even} \end{cases}
\end{aligned}$$

The gradient of $H(\mathbf{x}, \omega)$ is

$$\frac{\partial H(\mathbf{x}, \omega)}{\partial \mathbf{x}} = \begin{bmatrix} y(\omega) \frac{\partial \tilde{H}_a(M\omega)}{\partial \mathbf{a}} \\ y(\omega) \frac{\partial \tilde{H}_a(M\omega)}{\partial \mathbf{d}} \\ \tilde{H}_a(M\omega) \mathbf{c}_a(\omega) \\ [1 - \tilde{H}_a(M\omega)] \mathbf{c}_c(\omega) \end{bmatrix}^T$$

where

$$\begin{aligned}
y(\omega) &= \mathbf{a}_a^T \mathbf{c}_a(\omega) - \mathbf{a}_c^T \mathbf{c}_c(\omega) \\
\frac{\partial \tilde{H}_a(\omega)}{\partial \mathbf{a}} &= \frac{\mathbf{v}(\omega)}{d(\omega)} \\
\frac{\partial \tilde{H}_a(\omega)}{\partial \mathbf{d}} &= \left[\frac{\partial \tilde{H}_a(\omega)}{\partial d_0} \quad \frac{\partial \tilde{H}_a(\omega)}{\partial d_1} \quad \dots \quad \frac{\partial \tilde{H}_a(\omega)}{\partial d_L} \right]
\end{aligned}$$

with

$$\begin{aligned}
\frac{\partial \tilde{H}_a(\omega)}{\partial d_0} &= -\tilde{H}_a(\omega) \frac{v_1(\omega)}{1 + d_0 v_1(\omega)} \\
\frac{\partial \tilde{H}_a(\omega)}{\partial d_k} &= -\tilde{H}_a(\omega) \frac{\mathbf{v}_2(\omega)}{1 + \mathbf{d}_k^T \mathbf{v}_2(\omega)}, \quad \text{for } 1 \leq k \leq L
\end{aligned}$$

The overall parameter vector is

$$\mathbf{x} = \begin{bmatrix} \mathbf{a} \\ \mathbf{d} \\ \mathbf{a}_a \\ \mathbf{a}_c \end{bmatrix}$$

The calculation of the FRM zero-phase frequency response, $H(\mathbf{x}, \omega)$, and its gradient with respect to the coefficients is implemented in the Octave function `frm2ndOrderCascade()`.

The SOCP optimisation of an FRM filter is implemented in the Octave function `frm2ndOrderCascade_socp()`. The Octave script `frm2ndOrderCascade_socp_test()` designs a FRM filter similar to that of the example shown by Lu and Hinamoto [95, Section V.E]. The specification of the FRM filter is:

```

tol=1e-06 % Tolerance on coefficient update vector
n=1200 % Frequency points across the band
mn=10 % Model filter numerator order (mn+1 coefficients)
mr=10 % Model filter denominator order (mr coefficients)
na=33 % Model masking filter FIR length
nc=33 % Model complementary masking filter FIR length
M=9 % Decimation
Dmodel=7 % Model filter pass band group delay
dmask=16 % Masking filter nominal delay
fpass= 0.3 % Pass band edge
fstop=0.305 % Stop band edge
dBas=55 % Stop band attenuation
Wap=1 % Pass band weight
Wapextra=0 % Extra weight for extra pass band points
Wasextra=0 % Extra weight for extra stop band points
Was=9 % Stop band weight
tau=0.1 % Stability parameter
edge_factor=0.1 % Add extra frequencies near band edges
edge_ramp=0 % Linear change in extra weights over edge region

```

For comparison, the example of Lu and Hinamoto uses $mn = 14$, $n_a = 41$ and $D = 9$ and has a nominal passband delay of 101 samples. The SOCP pass minimises the combined error of the pass and stop bands at the constraint frequencies. Those frequencies are chosen so that half are concentrated in the union of the regions $[0.9f_p \quad f_p]$ and $[f_s \quad 1.1f_s]$. I found during debugging that the best results were obtained if the model filter denominator polynomial second order sections are rearranged after each of the first few SOCP passes. The initial filter uses FIR filter and IIR model filter numerator polynomials calculated by the Octave *remez* function and a model filter denominator polynomial of $d = 1$. This initial FRM filter results in a better FRM filter than the initial FRM filter calculated by the Octave script *tarczynski_frm_iir_test.m*.

The model filter numerator polynomial is

```
a = [ -0.0670638355, 0.2075231646, -0.2557629487, 0.0429412824, ...
      0.2510381494, -0.0220238387, -0.8452013645, -0.4404853589, ...
      -0.6561340777, 1.1930262627, -0.1191646168 ]';
```

The model filter denominator polynomial is

```
d = [ 1.0000000000, -1.1297298975, 1.0703348345, -1.0136151448, ...
      0.3939445988, 0.0719575589, -0.2341804517, 0.1795060709, ...
      -0.0743305673, 0.0163517321, -0.0015651193 ]';
```

The masking filter polynomial is

```
aa = [ 0.0037135196, -0.0090258670, -0.0029427402, 0.0147479739, ...
      -0.0084170312, -0.0129877846, 0.0226913959, 0.0019769306, ...
      -0.0212855058, 0.0346447841, -0.0133448209, -0.0453932272, ...
      0.0734263389, -0.0106279856, -0.1385485540, 0.2789950982, ...
      0.6782975754, 0.2789950982, -0.1385485540, -0.0106279856, ...
      0.0734263389, -0.0453932272, -0.0133448209, 0.0346447841, ...
      -0.0212855058, 0.0019769306, 0.0226913959, -0.0129877846, ...
      -0.0084170312, 0.0147479739, -0.0029427402, -0.0090258670, ...
      0.0037135196 ]';
```

The complementary masking filter polynomial is

```
ac = [ 0.0017342802, -0.0047535671, -0.0004251481, 0.0073708021, ...
      -0.0065701932, -0.0050468569, 0.0135548278, -0.0030063076, ...
      -0.0087769243, 0.0278469608, -0.0228460103, -0.0298698217, ...
      0.0717752758, -0.0266304594, -0.1221601869, 0.2866321256, ...
      0.6524596979, 0.2866321256, -0.1221601869, -0.0266304594, ...
      0.0717752758, -0.0298698217, -0.0228460103, 0.0278469608, ...
      -0.0087769243, -0.0030063076, 0.0135548278, -0.0050468569, ...
      -0.0065701932, 0.0073708021, -0.0004251481, -0.0047535671, ...
      0.0017342802 ]';
```

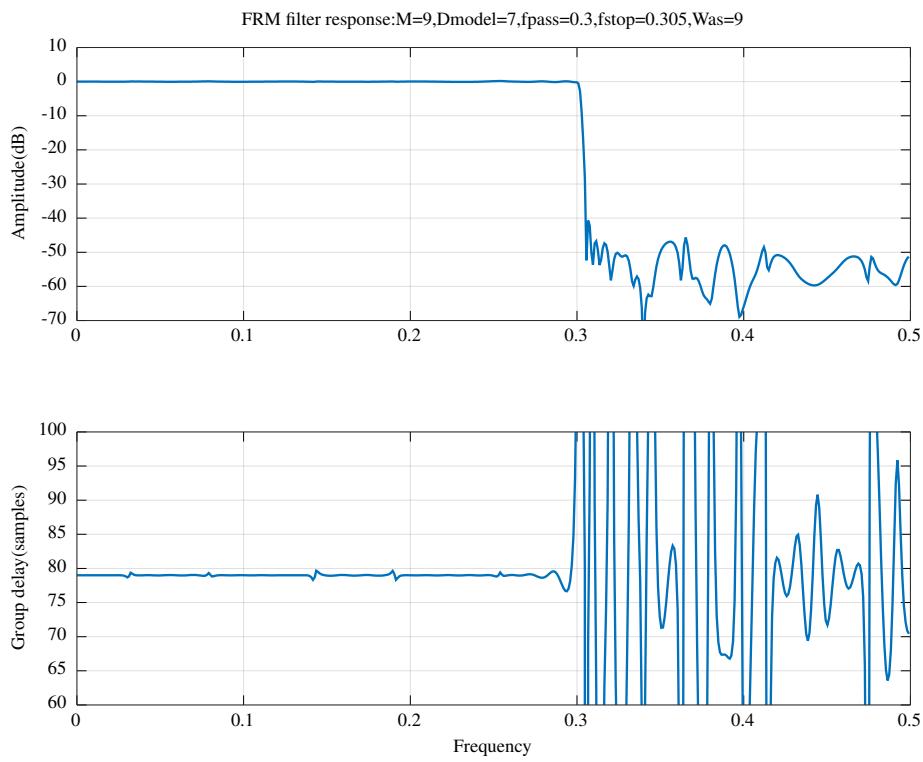


Figure 12.65: Second order cascade FRM filter response

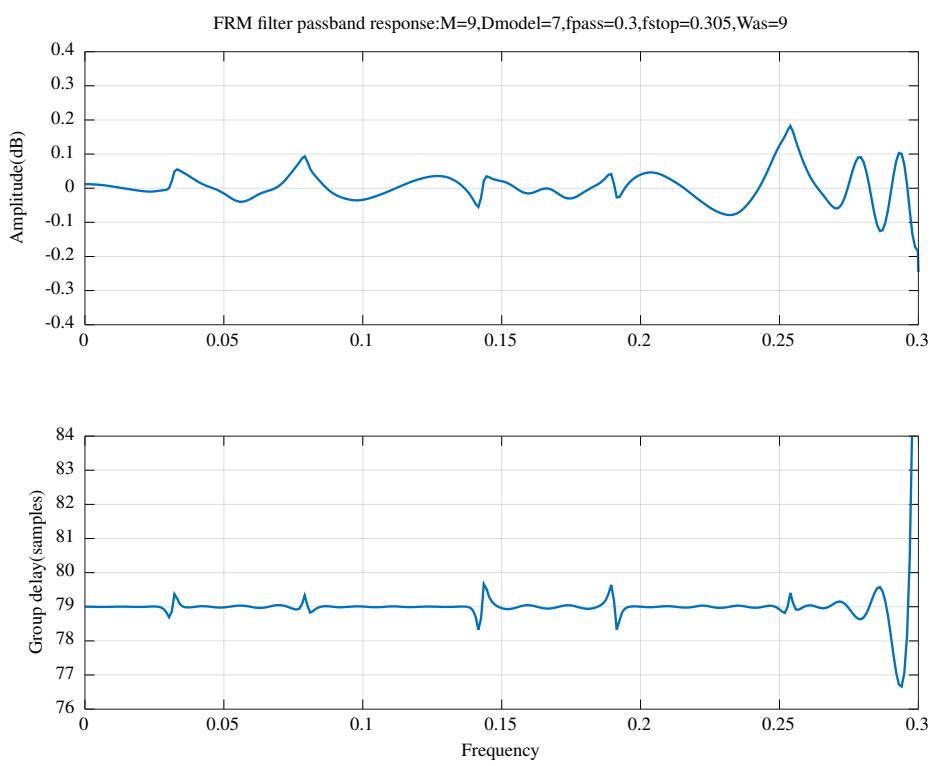


Figure 12.66: Second order cascade FRM filter passband response

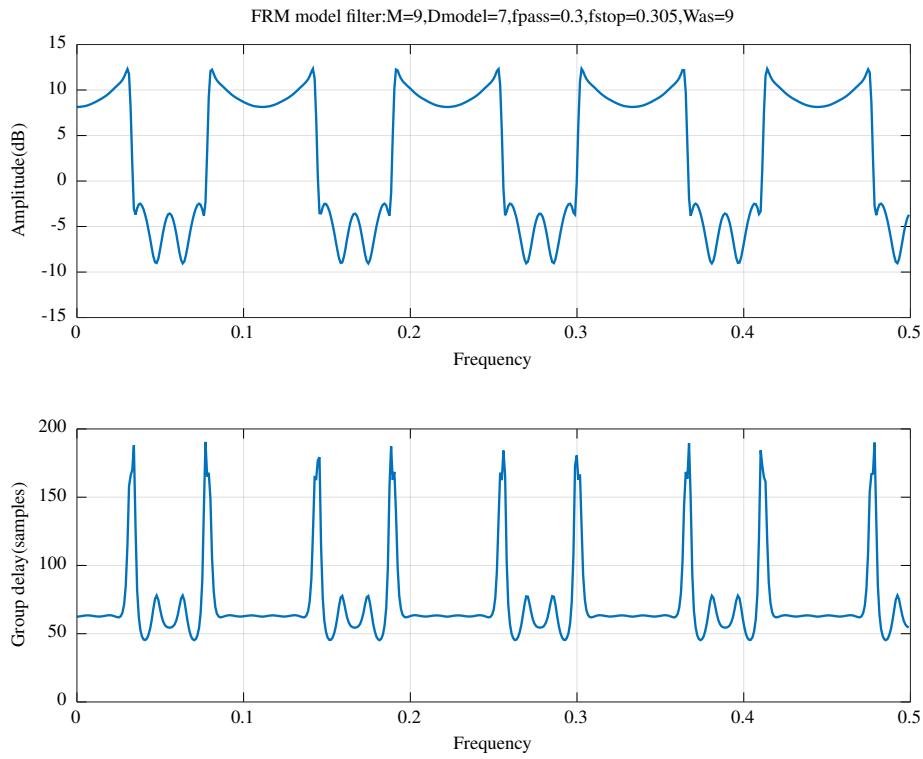


Figure 12.67: Second order cascade FRM model filter response

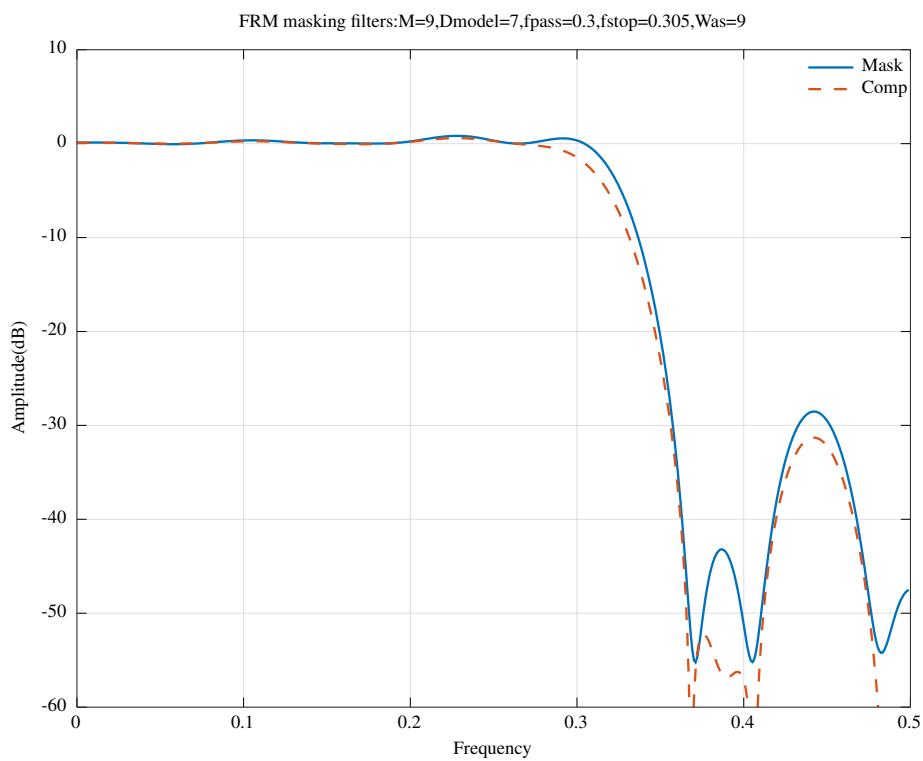


Figure 12.68: Second order cascade FRM masking filter responses

Figure 12.65 shows the overall response of the resulting FRM filter. Figure 12.66 shows the passband response of the FRM filter. Figure 12.67 shows the decimated response of the FRM model filter. Figure 12.68 shows the responses of the FRM masking filters. Figure 12.69 compares the amplitude response of the FRM filter with that of a linear-phase FIR filter having a group delay of 79 samples designed with the *remez* function:

```
br=remez((M*D)+d)*2, [0 fpass fstop 0.5]*2, [1 1 0 0], [1 Was];
```

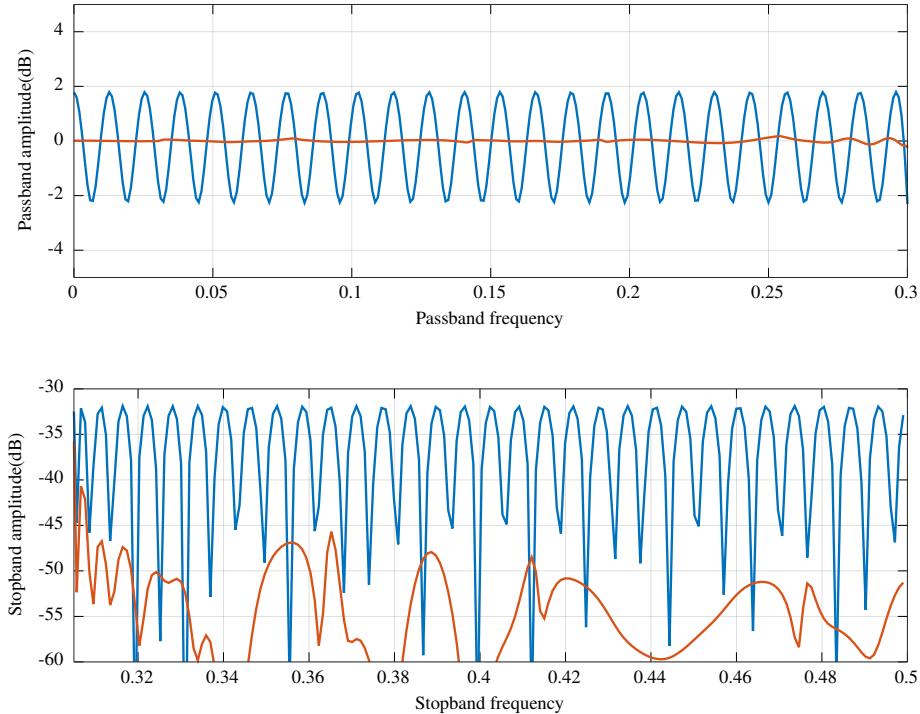


Figure 12.69: Comparison of the second order cascade FRM filter with a linear-phase FIR filter designed with *remez*

12.4.3 Design of an FRM digital filter with an IIR model filter represented in gain-pole-zero form using SOCP and PCLS optimisation

This section describes an FRM filter with the *Lim* FRM filter structure shown in Figure 12.62 in which the model filter is represented in gain-pole-zero form and the FIR masking filters are symmetric (ie: linear phase). The squared-amplitude and the group delay of the MMSE optimised FRM filter response are constrained by the PCLS algorithm of *Selesnick, Lang and Burrus*, described in Section 10.1.2.

Using the notation of Section 12.4.2, the desired passband group delay of the FRM filter is $D_s = d_{mask} + M_{model}D_{model}$, where D_{model} is the delay of the pure delay branch of the FRM filter, M_{model} is the decimation factor of the IIR model filter and d_{mask} is the group delay of the linear phase FIR masking filters. If $R(\omega)e^{i\phi_R(\omega)}$ is the frequency response of the IIR model filter then the *zero phase* response of the FRM filter is

$$H(x, \omega) = A(\omega)R(M\omega)e^{i\phi_Z(M\omega)} + B(\omega)$$

where

$$\begin{aligned}\phi_Z(\omega) &= D\omega + \phi_R(\omega) \\ A(\omega) &= \mathbf{a}_a^T \mathbf{c}_a(\omega) - \mathbf{a}_c^T \mathbf{c}_c(\omega) \\ B(\omega) &= \mathbf{a}_c^T \mathbf{c}_c(\omega)\end{aligned}$$

The squared-magnitude and phase responses of the zero phase response of the FRM filter are

$$\begin{aligned}|H(\omega)|^2 &= A^2(\omega)R^2(M\omega) + B^2(\omega) + 2A(\omega)B(\omega)R(M\omega)\cos\phi_Z(M\omega) \\ \arg H(\omega) &= \arctan \frac{A(\omega)R(M\omega)\sin\phi_Z(M\omega)}{A(\omega)R(M\omega)\cos\phi_Z(M\omega) + B(\omega)}\end{aligned}$$

The group delay response, $T(\omega)$, of the zero phase response of the FRM filter (ie: the group delay error of the FRM filter, $-\frac{\partial \arg H}{\partial \omega}$) is given by

$$\begin{aligned}|H(\omega)|^2 T(\omega) &= - (A^2(\omega)R^2(M\omega) + A(\omega)B(\omega)R(M\omega)\cos\phi_Z(M\omega)) \frac{\partial\phi_Z(M\omega)}{\partial\omega} \dots \\ &\quad - A(\omega)B(\omega)\sin\phi_Z(M\omega) \frac{\partial R(M\omega)}{\partial\omega} \dots \\ &\quad + R(M\omega)\sin\phi_Z(M\omega) \left(A(\omega) \frac{\partial B(\omega)}{\partial\omega} - B(\omega) \frac{\partial A(\omega)}{\partial\omega} \right)\end{aligned}$$

where

$$\begin{aligned}\frac{\partial\phi_Z(M\omega)}{\partial\omega} &= DM + \frac{\partial\phi_R(M\omega)}{\partial\omega} \\ \frac{\partial A(\omega)}{\partial\omega} &= \mathbf{a}_a^T \mathbf{s}_a(\omega) - \mathbf{a}_c^T \mathbf{s}_c(\omega) \\ \frac{\partial B(\omega)}{\partial\omega} &= \mathbf{a}_c^T \mathbf{s}_c(\omega) \\ \mathbf{s}_a(\omega) &= \begin{cases} -2 \left[\begin{array}{cccc} 0 & \sin\omega & \dots & \frac{(n_a-1)}{2} \sin \frac{(n_a-1)}{2}\omega \end{array} \right]^T, & \text{if } n_a \text{ odd} \\ -2 \left[\begin{array}{cccc} \sin \frac{1}{2}\omega & \dots & \frac{(n_a-1)}{2} \sin \frac{(n_a-1)}{2}\omega \end{array} \right]^T, & \text{if } n_a \text{ even} \end{cases} \\ \mathbf{s}_c(\omega) &= \begin{cases} -2 \left[\begin{array}{cccc} 0 & \sin\omega & \dots & \frac{(n_c-1)}{2} \sin \frac{(n_c-1)}{2}\omega \end{array} \right]^T, & \text{if } n_c \text{ odd} \\ -2 \left[\begin{array}{cccc} \sin \frac{1}{2}\omega & \dots & \frac{(n_c-1)}{2} \sin \frac{(n_c-1)}{2}\omega \end{array} \right]^T, & \text{if } n_c \text{ even} \end{cases}\end{aligned}$$

The gradients of $|H(\omega)|^2$ with respect to the coefficients are

$$\begin{aligned}\frac{\partial |H(\omega)|^2}{\partial \mathbf{r}} &= 2(A^2(\omega)R(M\omega) + A(\omega)B(\omega)\cos\phi_Z(M\omega)) \frac{\partial R(M\omega)}{\partial \mathbf{r}} \dots \\ &\quad - 2A(\omega)B(\omega)R(M\omega)\sin\phi_Z(M\omega) \frac{\partial\phi_R(M\omega)}{\partial \mathbf{r}} \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{a}} &= 2(A(\omega)R^2(M\omega) + B(\omega)R(M\omega)\cos\phi_Z(M\omega)) \frac{\partial A(\omega)}{\partial \mathbf{a}} \dots\end{aligned}$$

$$+ 2(B(\omega) + A(\omega)R(M\omega)\cos\phi_Z(M\omega)) \frac{\partial B(\omega)}{\partial \mathbf{a}}$$

where \mathbf{r} represents the coefficients of the IIR model filter, and \mathbf{a} represents the coefficients of both the masking filter, \mathbf{a}_a , and the complementary masking filter, \mathbf{a}_c .

The gradients of $T(\omega)$ with respect to the coefficients are given by

$$\begin{aligned} \frac{\partial |H(\omega)|^2}{\partial \mathbf{r}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \mathbf{r}} &= \dots \\ - (A^2(\omega)R^2(M\omega) + A(\omega)B(\omega)R(M\omega)\cos\phi_Z(M\omega)) \frac{\partial^2 \phi_R(M\omega)}{\partial \omega \partial \mathbf{r}} \dots \\ - (2A^2(\omega)R(M\omega) + A(\omega)B(\omega)\cos\phi_Z(M\omega)) \frac{\partial \phi_Z(M\omega)}{\partial \omega} \frac{\partial R(M\omega)}{\partial \mathbf{r}} \dots \\ + A(\omega)B(\omega)R(M\omega)\sin\phi_Z(M\omega) \frac{\partial \phi_Z(M\omega)}{\partial \omega} \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \dots \\ - A(\omega)B(\omega)\cos\phi_Z(M\omega) \frac{\partial R(M\omega)}{\partial \omega} \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \dots \\ - A(\omega)B(\omega)\sin\phi_Z(M\omega) \frac{\partial^2 R(M\omega)}{\partial \omega \partial \mathbf{r}} \dots \\ + \sin\phi_Z(M\omega) \frac{\partial R(M\omega)}{\partial \mathbf{r}} \left(A(\omega) \frac{\partial B(\omega)}{\partial \omega} - B(\omega) \frac{\partial A(\omega)}{\partial \omega} \right) \dots \\ + R(M\omega)\cos\phi_Z(M\omega) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \left(A(\omega) \frac{\partial B(\omega)}{\partial \omega} - B(\omega) \frac{\partial A(\omega)}{\partial \omega} \right) \end{aligned}$$

and

$$\begin{aligned} \frac{\partial |H(\omega)|^2}{\partial \mathbf{a}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \mathbf{a}} &= \dots \\ - (2A(\omega)R^2(M\omega) + B(\omega)R(M\omega)\cos\phi_Z(M\omega)) \frac{\partial \phi_Z(M\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial \mathbf{a}} \dots \\ - A(\omega)R(M\omega)\cos\phi_Z(M\omega) \frac{\partial \phi_Z(M\omega)}{\partial \omega} \frac{\partial B(\omega)}{\partial \mathbf{a}} \dots \\ - \sin\phi_Z(M\omega) \frac{\partial R(M\omega)}{\partial \omega} \left(A(\omega) \frac{\partial B(\omega)}{\partial \mathbf{a}} + B(\omega) \frac{\partial A(\omega)}{\partial \mathbf{a}} \right) \dots \\ - R(M\omega)\sin\phi_Z(M\omega) \left(\frac{\partial A(\omega)}{\partial \omega} \frac{\partial B(\omega)}{\partial \mathbf{a}} - \frac{\partial B(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial \mathbf{a}} \right) \dots \\ + R(M\omega)\sin\phi_Z(M\omega) \left(A(\omega) \frac{\partial^2 B(\omega)}{\partial \omega \partial \mathbf{a}} - B(\omega) \frac{\partial^2 A(\omega)}{\partial \omega \partial \mathbf{a}} \right) \end{aligned}$$

The amplitude, phase and group delay responses and gradients of the IIR model filter are calculated by the Octave functions *iirA*, *iirP* and *iirT*. The derivative with respect to frequency of the IIR model filter amplitude response, $\frac{\partial R}{\partial \omega}$, is derived in Appendix C and calculated by the Octave function *iirdelAdelw*. The Octave function *iir_frm.m* returns the low pass FRM filter squared-magnitude and group delay error responses and their gradients.

The Octave function *iir_frm_socp_mmse.m* finds the SOCP solution that optimises the coefficients of a filter response calculated by *iir_frm.m* with the required linear amplitude and group delay constraints. The Octave function *iir_frm_slb.m* implements the PCLS algorithm for finding the constraint frequencies. The Octave script *iir_frm_socp_slb_test.m* designs an FRM filter with the following specification:

```
n=400 % Frequency points across the band
tol=0.0002 % Tolerance on relative coefficient update size
ma=10 % IIR model filter numerator order
md=10 % IIR model filter denominator order
na=41 % FIR masking filter length (order+1)
nc=41 % FIR complementary masking filter length (order+1)
Mmodel=9 % Model filter decimation factor
Dmodel=7 % Model filter nominal pass band group delay
dmask=20 % FIR masking filter delay
Tnominal=83 % FIR masking filter delay
fap=0.3 % Pass band edge
dBap=0.1 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
tpr=1 % Pass band delay peak-to-peak ripple
```

```

Wtp=0.05 % Pass band delay weight
Wat=4e-08 % Transition band amplitude weight
fas=0.31125 % Stop band edge
dBas=40 % Stop band attenuation ripple
fasA=0.31125 % Additional stop band edge
dBAsA=40 % Additional stop band attenuation ripple
Was=50 % Stop band weight

```

The passband edge frequency is the same as that of the design example given by Lu and Hinamoto [95, Section V.E], namely 0.300, but the stop band edge frequency is relaxed from 0.305 to 0.31125. Both FIR masking filters have length 41. The initial filter is designed by the Octave script *tarczynski_fir_iir_test.m* with the WISE method of Tarczynski *et al.* (as shown in Section 10.1.5). Figure 12.70 shows the overall response of the initial FRM filter. After SOCP and PCLS optimisation of the

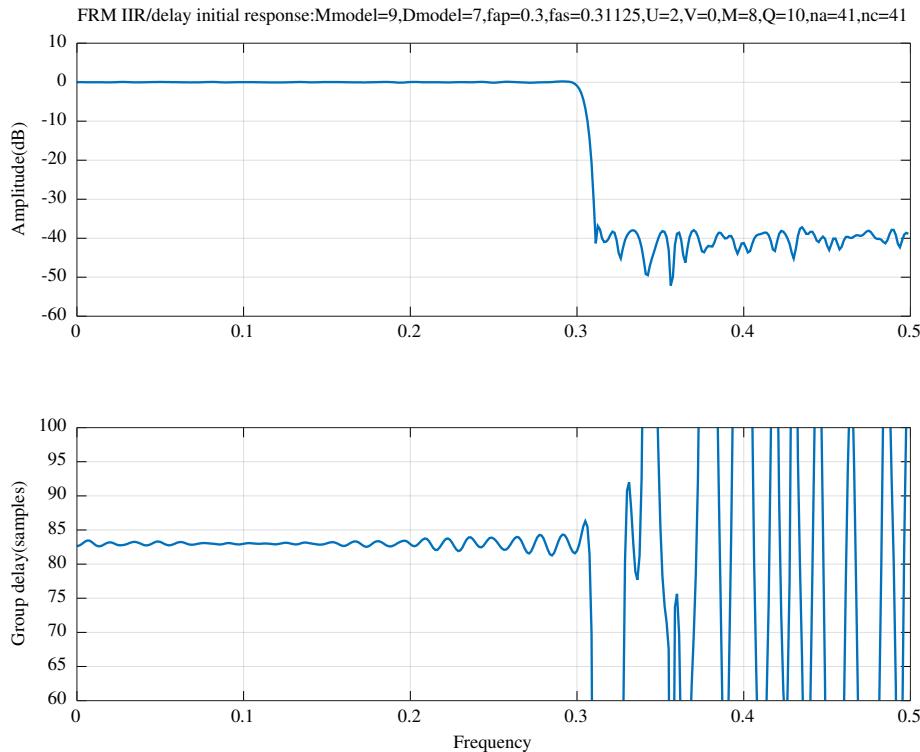


Figure 12.70: FRM gain-pole-zero format IIR model filter with WISE optimisation : initial response

initial response the resulting IIR model filter numerator polynomial is

$$a = [\begin{array}{ccccccc} 0.0076278129, & 0.0415842834, & -0.0436800082, & -0.0714979522, & \dots \\ 0.1085671506, & 0.1497643029, & -0.2114258252, & 0.4429788162, & \dots \\ -0.5367456391, & 0.2661302741, & 0.1162560522 \end{array}]';$$

and the IIR model filter denominator polynomial is

$$d = [\begin{array}{ccccccc} 1.0000000000, & 0.1042601496, & 0.5355166651, & 0.1081586819, & \dots \\ -0.0598776864, & -0.0284049728, & -0.0034188664, & 0.0200735744, & \dots \\ 0.0164328945, & 0.0056710998, & 0.0015312570 \end{array}]';$$

The FIR masking filter polynomial is

$$aa = [\begin{array}{ccccccc} -0.0034746550, & 0.0030234856, & -0.0001398567, & -0.0021948577, & \dots \\ 0.0017451188, & 0.0003487298, & -0.0064408521, & 0.0066375025, & \dots \\ -0.0013765107, & -0.0209767262, & 0.0236087830, & -0.0038054662, & \dots \\ -0.0208125940, & 0.0257229528, & -0.0014666336, & -0.0517010363, & \dots \\ 0.0636831303, & -0.0015554928, & -0.1430831955, & 0.2806659316, & \dots \\ 0.6620956646, & 0.2806659316, & -0.1430831955, & -0.0015554928, & \dots \end{array}]$$

```

0.0636831303, -0.0517010363, -0.0014666336, 0.0257229528, ...
-0.0208125940, -0.0038054662, 0.0236087830, -0.0209767262, ...
-0.0013765107, 0.0066375025, -0.0064408521, 0.0003487298, ...
0.0017451188, -0.0021948577, -0.0001398567, 0.0030234856, ...
-0.0034746550 ]';

```

The complementary FIR masking filter polynomial is

```

ac = [ 0.0157711738, -0.0157990184, -0.0011270081, 0.0164261558, ...
-0.0097246399, -0.0126618740, 0.0352388908, -0.0133314023, ...
-0.0289985951, 0.0145909018, 0.0258882622, -0.0443994626, ...
0.0090112500, 0.0439635077, -0.0460850933, -0.0422553962, ...
0.1033167590, -0.0416916996, -0.1093247926, 0.2895974014, ...
0.6298718280, 0.2895974014, -0.1093247926, -0.0416916996, ...
0.1033167590, -0.0422553962, -0.0460850933, 0.0439635077, ...
0.0090112500, -0.0443994626, 0.0258882622, 0.0145909018, ...
-0.0289985951, -0.0133314023, 0.0352388908, -0.0126618740, ...
-0.0097246399, 0.0164261558, -0.0011270081, -0.0157990184, ...
0.0157711738 ]';

```

Figure 12.71 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 12.72 shows the passband response of the resulting FRM filter. Figure 12.73 shows the responses of the resulting FRM masking filters. Figure 12.74 shows the decimated response of the resulting FRM model filter.

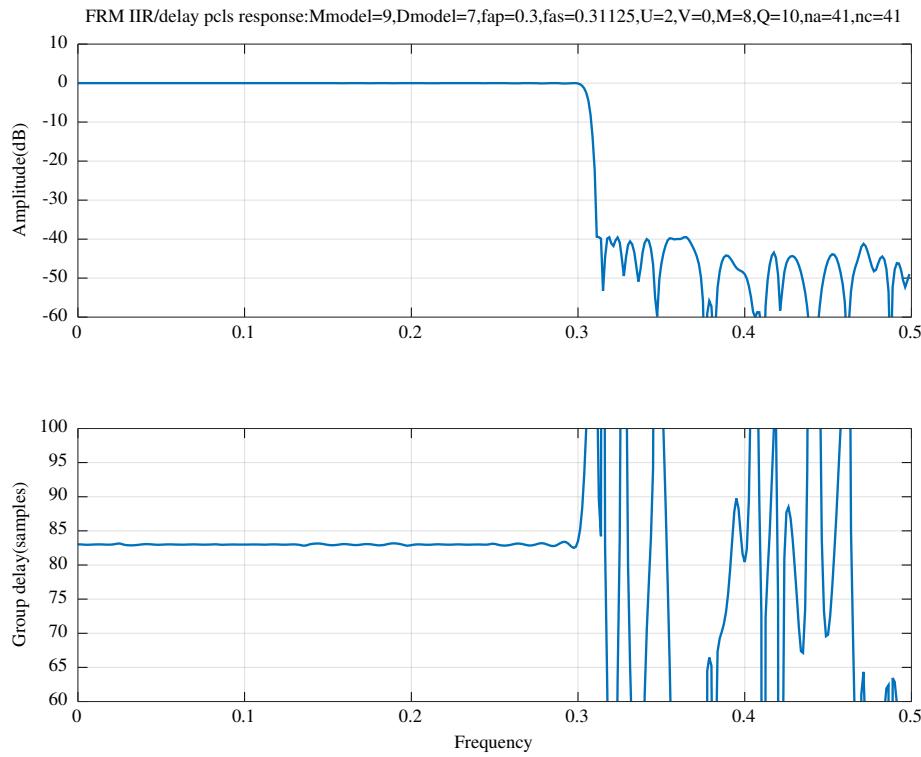


Figure 12.71: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation : overall response

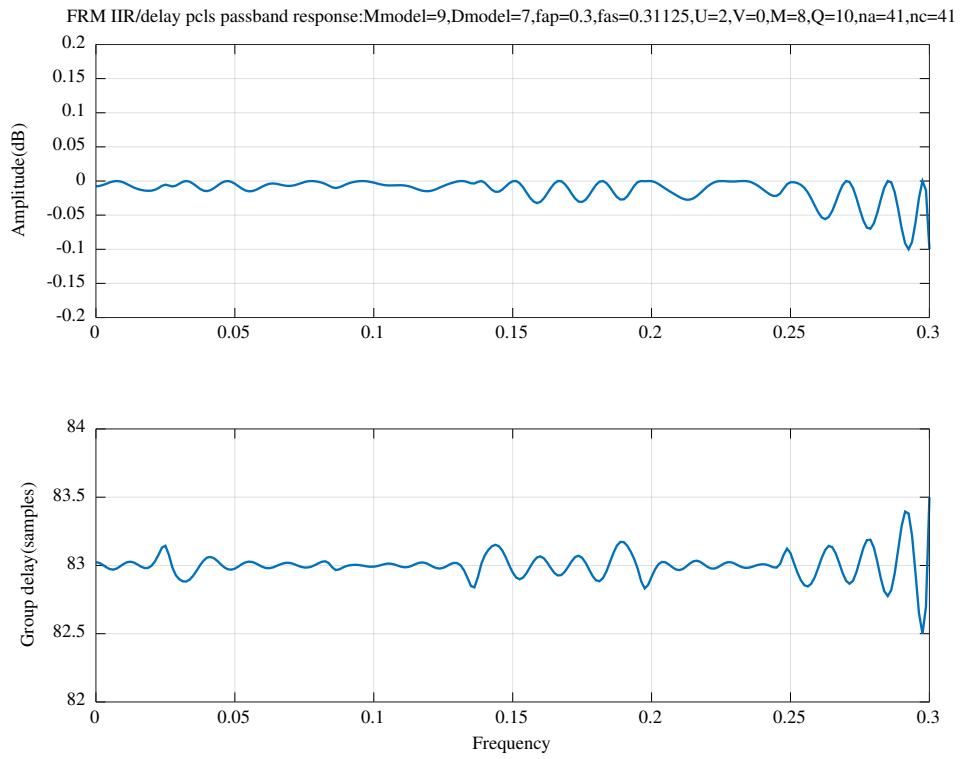


Figure 12.72: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation : passband response

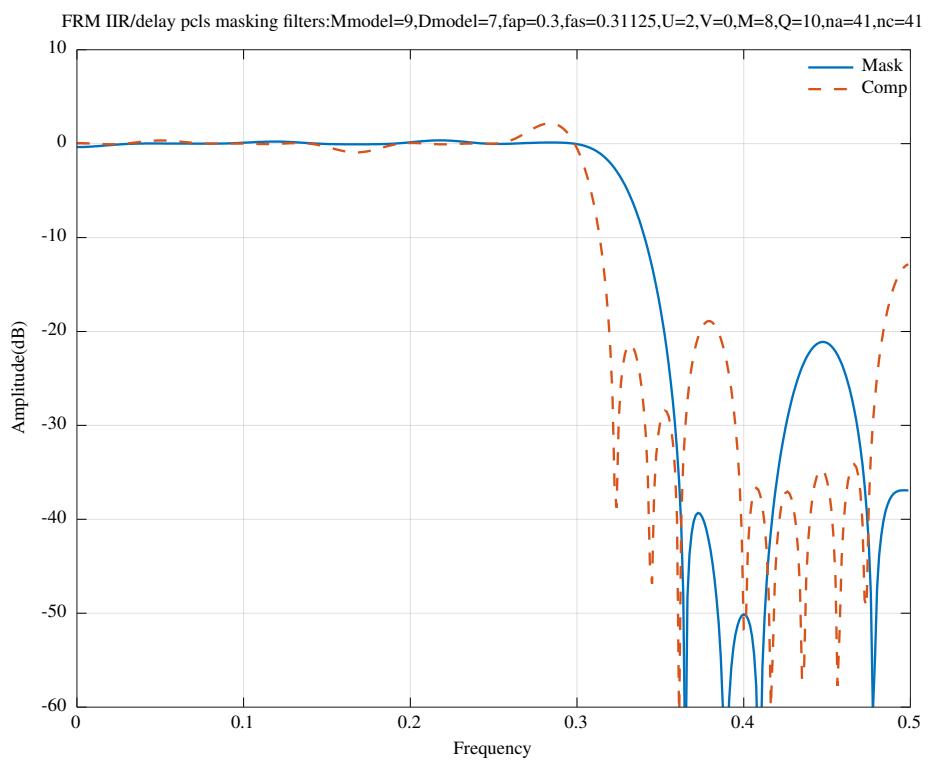


Figure 12.73: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation : masking filter responses

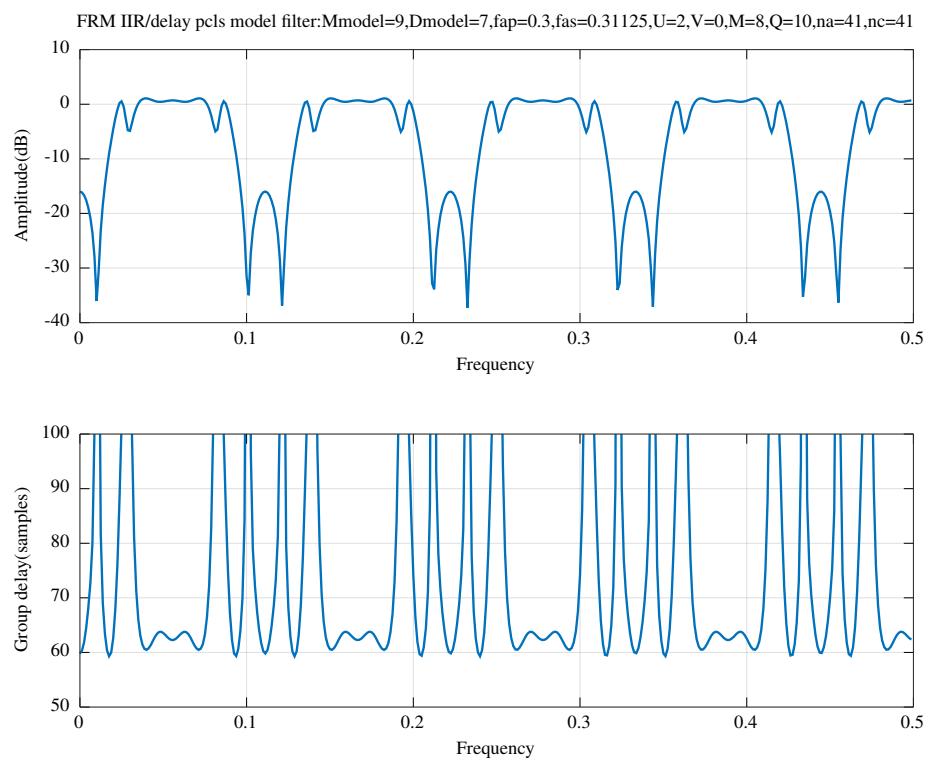


Figure 12.74: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation : model filter response

12.4.4 Design of an FRM digital filter with an allpass model filter represented in gain-pole-zero form using SOCP and PCLS optimisation

This section describes an FRM filter with the structure shown in Figure 12.63, and an all-pass filter. For simplicity the model filters consist of an allpass filter in parallel with a pure delay and the FIR masking filters are of equal odd length and are symmetric (ie: the FIR masking filters are even order and linear phase). The squared-amplitude and the group delay of the MMSE optimised FRM filter response are constrained by the PCLS algorithm of *Selesnick, Lang and Burrus*, described in Section 10.1.2.

Using the notation of Section 12.4.2, the desired passband group delay of the FRM filter is $D_s = d + MD$, where D is the group delay of the model filter, M is the decimation factor of the model filter and d is the group delay of the linear phase FIR masking filters. The *zero phase* response of the FRM filter is

$$H(\mathbf{x}, \omega) = e^{i[D M \omega + \phi_R(M\omega)]} A(\omega) + B(\omega)$$

where $\phi_R(\omega)$ is the phase response of the allpass branch of the model filter and:

$$\begin{aligned} A(\omega) &= \frac{\mathbf{a}_a^T \mathbf{c}_a(\omega) + \mathbf{a}_c^T \mathbf{c}_c(\omega)}{2} \\ B(\omega) &= \frac{\mathbf{a}_a^T \mathbf{c}_a(\omega) - \mathbf{a}_c^T \mathbf{c}_c(\omega)}{2} \end{aligned}$$

The squared-magnitude and phase responses of the zero phase response of the FRM filter are:

$$\begin{aligned} |H(\omega)|^2 &= A^2(\omega) + B^2(\omega) + 2A(\omega)B(\omega)\cos\phi_Z(M\omega) \\ \arg H(\omega) &= \arctan \frac{A(\omega)\sin\phi_Z(M\omega)}{A(\omega)\cos\phi_Z(M\omega) + B(\omega)} \end{aligned}$$

where $\phi_Z(\omega) = D\omega + \phi_R(\omega)$.

The group delay response, $T(\omega)$, of the zero phase response of the FRM filter (ie: the group delay error of the FRM filter) is given by:

$$\begin{aligned} |H(\omega)|^2 T(\omega) &= - (A^2(\omega) + A(\omega)B(\omega)\cos\phi_Z(M\omega)) \frac{\partial\phi_Z(M\omega)}{\partial\omega} \dots \\ &\quad - B(\omega)\sin\phi_Z(M\omega) \frac{\partial A(\omega)}{\partial\omega} \dots \\ &\quad + A(\omega)\sin\phi_Z(M\omega) \frac{\partial B(\omega)}{\partial\omega} \end{aligned}$$

where:

$$\begin{aligned} \frac{\partial\phi_Z(M\omega)}{\partial\omega} &= DM + \frac{\partial\phi_R(M\omega)}{\partial\omega} \\ \frac{\partial A(\omega)}{\partial\omega} &= \frac{\mathbf{a}_a^T \mathbf{s}_a(\omega) + \mathbf{a}_c^T \mathbf{s}_c(\omega)}{2} \\ \frac{\partial B(\omega)}{\partial\omega} &= \frac{\mathbf{a}_a^T \mathbf{s}_a(\omega) - \mathbf{a}_c^T \mathbf{s}_c(\omega)}{2} \\ \mathbf{s}_a(\omega) &= -2 \left[\begin{array}{cccc} 0 & \sin\omega & \dots & \frac{(n_a-1)}{2} \sin \frac{(n_a-1)}{2}\omega \end{array} \right]^T \\ \mathbf{s}_c(\omega) &= -2 \left[\begin{array}{cccc} 0 & \sin\omega & \dots & \frac{(n_c-1)}{2} \sin \frac{(n_c-1)}{2}\omega \end{array} \right]^T \end{aligned}$$

The gradients of $|H(\omega)|^2$ with respect to the coefficients are:

$$\begin{aligned} \frac{\partial |H(\omega)|^2}{\partial \mathbf{r}} &= -2A(\omega)B(\omega)\sin\phi_Z(M\omega) \frac{\partial\phi_R(M\omega)}{\partial\mathbf{r}} \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{a}} &= 2(A(\omega) + B(\omega)\cos\phi_Z(M\omega)) \frac{\partial A(\omega)}{\partial \mathbf{a}} + 2(B(\omega) + A(\omega)\cos\phi_Z(M\omega)) \frac{\partial B(\omega)}{\partial \mathbf{a}} \end{aligned}$$

where \mathbf{r} represents the coefficients of the allpass model filter, and \mathbf{a} represents the coefficients of both the masking filter, \mathbf{a}_a , and the complementary masking filter, \mathbf{a}_c .

The gradients of $\arg H(\omega)$ with respect to the coefficients are:

$$|H(\omega)|^2 \frac{\partial \arg H(\omega)}{\partial \mathbf{r}} = A(\omega)[A(\omega) + B(\omega)\cos\phi_Z(M\omega)] \frac{\partial\phi_R(M\omega)}{\partial\mathbf{r}}$$

$$|H(\omega)|^2 \frac{\partial \arg H(\omega)}{\partial \mathbf{a}} = \sin \phi_Z(M\omega) \left[B(\omega) \frac{\partial A(\omega)}{\partial \mathbf{a}} - A(\omega) \frac{\partial B(\omega)}{\partial \mathbf{a}} \right]$$

The gradients of $T(\omega)$ with respect to the coefficients are given by:

$$\begin{aligned} \frac{\partial |H(\omega)|^2}{\partial \mathbf{r}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \mathbf{r}} &= - (A^2(\omega) + A(\omega)B(\omega) \cos \phi_Z(M\omega)) \frac{\partial^2 \phi_Z(M\omega)}{\partial \mathbf{r} \partial \omega} \dots \\ &\quad + A(\omega)B(\omega) \sin \phi_Z(M\omega) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \frac{\partial \phi_Z(M\omega)}{\partial \omega} \dots \\ &\quad + A(\omega) \cos \phi_Z(M\omega) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \frac{\partial B(\omega)}{\partial \omega} \dots \\ &\quad - B(\omega) \cos \phi_Z(M\omega) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \frac{\partial A(\omega)}{\partial \omega} \dots \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{a}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \mathbf{a}} &= - (2A(\omega) + B(\omega) \cos \phi_Z(M\omega)) \frac{\partial A(\omega)}{\partial \mathbf{a}} \frac{\partial \phi_Z(M\omega)}{\partial \omega} \dots \\ &\quad - A(\omega) \cos \phi_Z(M\omega) \frac{\partial B(\omega)}{\partial \mathbf{a}} \frac{\partial \phi_Z(M\omega)}{\partial \omega} \dots \\ &\quad + \sin \phi_Z(M\omega) \frac{\partial A(\omega)}{\partial \mathbf{a}} \frac{\partial B(\omega)}{\partial \omega} - \sin \phi_Z(M\omega) \frac{\partial B(\omega)}{\partial \mathbf{a}} \frac{\partial A(\omega)}{\partial \omega} \dots \\ &\quad + A(\omega) \sin \phi_Z(M\omega) \frac{\partial^2 B(\omega)}{\partial \mathbf{a} \partial \omega} - B(\omega) \sin \phi_Z(M\omega) \frac{\partial^2 A(\omega)}{\partial \mathbf{a} \partial \omega} \end{aligned}$$

The values and gradients of $\phi_R(\omega)$ are calculated by the Octave *allpassP* and *allpassT* functions.

The Octave function *iir_frm_allpass.m* returns the low pass FRM filter squared-magnitude and group delay error responses and their gradients. It is exercised by the Octave script *iir_frm_allpass_test.m*. The Octave function *iir_frm_allpass_socp_mmse* finds the SOCP solution that optimises the coefficients of a filter response calculated by *iir_frm_allpass* with the required amplitude and group delay constraints. The Octave function *iir_frm_allpass_slb* implements the PCLS algorithm for finding the constraint frequencies.

The Octave script *iir_frm_allpass_socp_slb_test.m* designs an FRM filter with the following specification:

```
tol=2e-05 % Tolerance on coefficient update vector
n=1000 % Frequency points across the band
mr=10 % Allpass model filter denominator order
R=1 % Allpass model filter decimation factor
na=41 % FIR masking filter length (order+1)
nc=41 % FIR complementary masking filter length (order+1)
Mmodel=9 % Model filter decimation factor
Dmodel=9 % Model filter nominal pass band group delay
dmask=20 % FIR masking filter delay
Tnominal=101 % Nominal FRM filter group delay
fap=0.3 % Pass band edge
dBap=0.05 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
tpr=1 % Pass band delay peak-to-peak ripple
Wtp=0.025 % Pass band delay weight
fas=0.3105 % Stop band edge
dBas=40 % Stop band attenuation ripple
Was=50 % Stop band weight
rho=0.968750 % Constraint on allpass pole radius
```

The passband edge frequency is the same as for the design example given by Lu and Hinamoto [95, Section V.E], namely 0.300, but the stop band edge frequency is relaxed slightly from 0.305 to 0.31. Both FIR masking filters have length 41. The initial filter is designed by the Octave script *tarczynski_frm_allpass_test.m* with the WISE method of Tarczynski *et al.* as shown in Section 10.1.5. Figure 12.75 shows the overall response of the initial FRM filter. After SOCP and PCLS optimisation of the initial filter the resulting model filter allpass filter denominator polynomial is

```
r = [ 1.0000000000, -0.0320590553, 0.4903054365, 0.0138034084, ...
-0.1089307638, -0.0101913920, 0.0412530056, 0.0024491534, ...
-0.0241234993, -0.0091388456, 0.0028791749 ]';
```

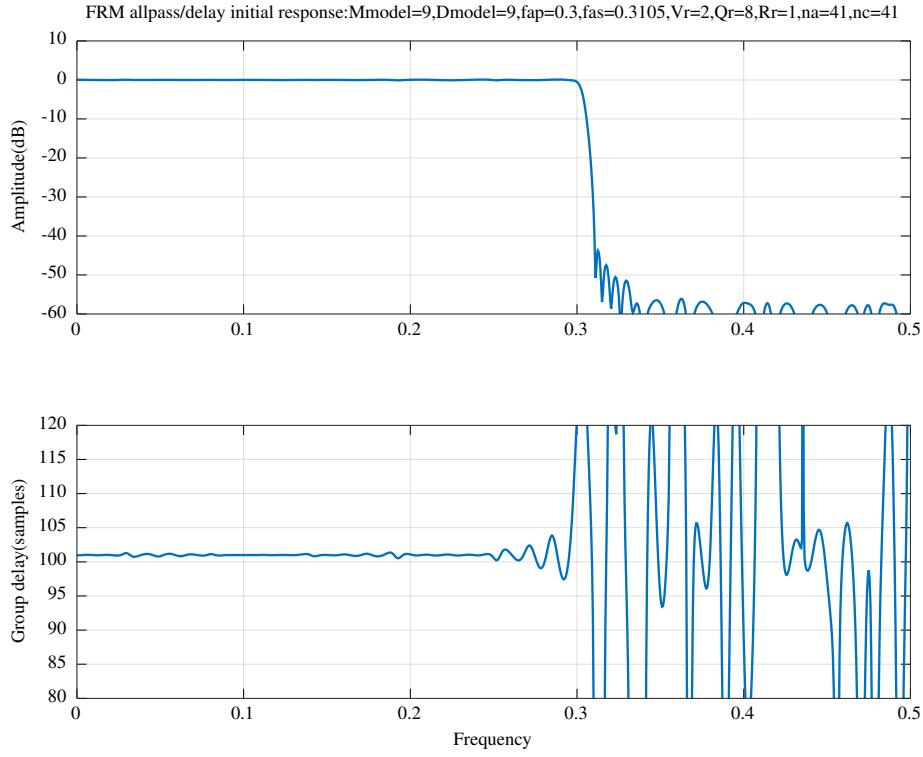


Figure 12.75: FRM gain-pole-zero format allpass model filter with WISE optimisation : initial response

The FIR masking filter polynomial is

```
aa = [ -0.0036033362, 0.0023926406, 0.0039878742, -0.0071721701, ...
0.0004490460, 0.0075461637, -0.0049793052, -0.0077327162, ...
0.0099886216, 0.0115694409, -0.0176965859, -0.0099131298, ...
0.0343976340, -0.0074552626, -0.0423815161, 0.0350458672, ...
0.0494685434, -0.0854543929, -0.0604500199, 0.3074416518, ...
0.5677661186, 0.3074416518, -0.0604500199, -0.0854543929, ...
0.0494685434, 0.0350458672, -0.0423815161, -0.0074552626, ...
0.0343976340, -0.0099131298, -0.0176965859, 0.0115694409, ...
0.0099886216, -0.0077327162, -0.0049793052, 0.0075461637, ...
0.0004490460, -0.0071721701, 0.0039878742, 0.0023926406, ...
-0.0036033362 ]';
```

The complementary FIR masking filter polynomial is

```
ac = [ 0.0017788085, -0.0008725081, -0.0028356131, 0.0073446083, ...
-0.0071881587, 0.0011331737, 0.0079524582, -0.0100275676, ...
0.0014423097, 0.0138372500, -0.0147224124, -0.0045872568, ...
0.0315104968, -0.0354388082, 0.0028332925, 0.0490774544, ...
-0.0650752653, 0.0030699275, 0.1324731576, -0.2693511644, ...
-0.6714880942, -0.2693511644, 0.1324731576, 0.0030699275, ...
-0.0650752653, 0.0490774544, 0.0028332925, -0.0354388082, ...
0.0315104968, -0.0045872568, -0.0147224124, 0.0138372500, ...
0.0014423097, -0.0100275676, 0.0079524582, 0.0011331737, ...
-0.0071881587, 0.0073446083, -0.0028356131, -0.0008725081, ...
0.0017788085 ]';
```

Figure 12.76 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 12.77 shows the passband response of the resulting FRM filter. Figure 12.78 shows the responses of the resulting FRM masking filters. Figure 12.79 shows the response of the resulting FRM model filter.

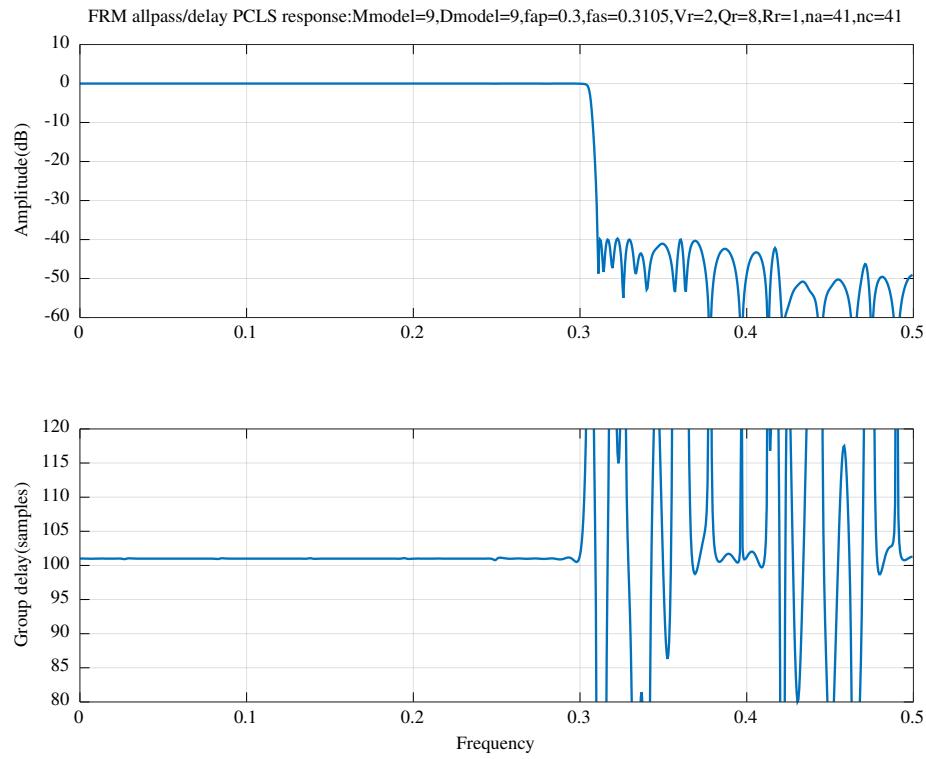


Figure 12.76: FRM gain-pole-zero format allpass model filter with SOCP and PCLS optimisation : overall response

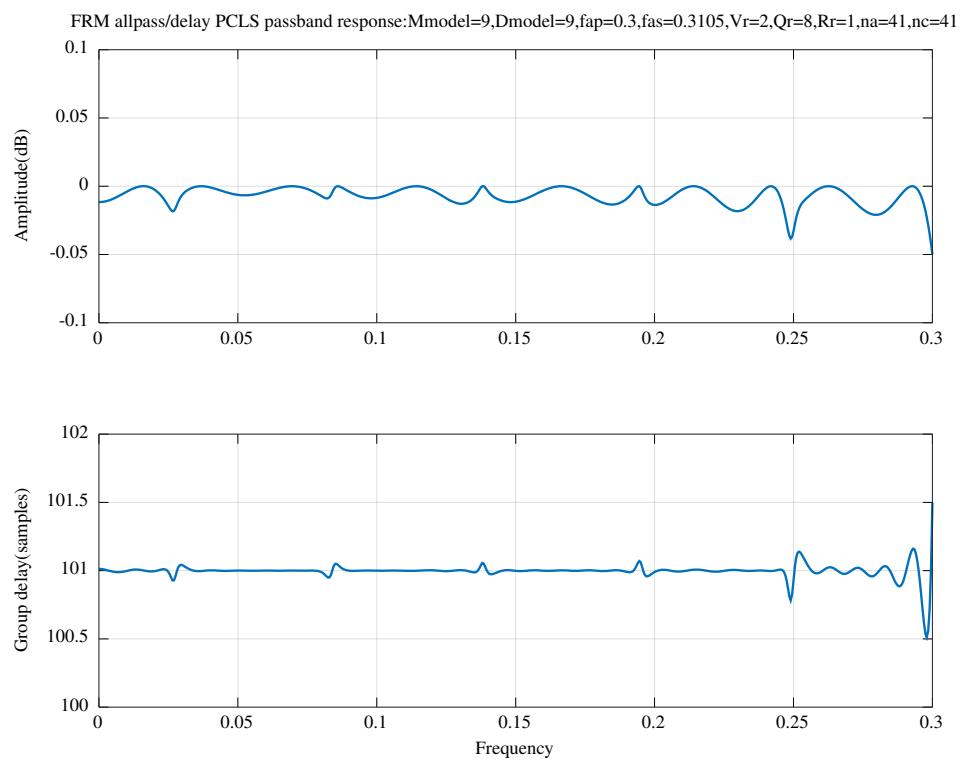


Figure 12.77: FRM gain-pole-zero format allpass model filter : passband response with SOCP and PCLS optimisation

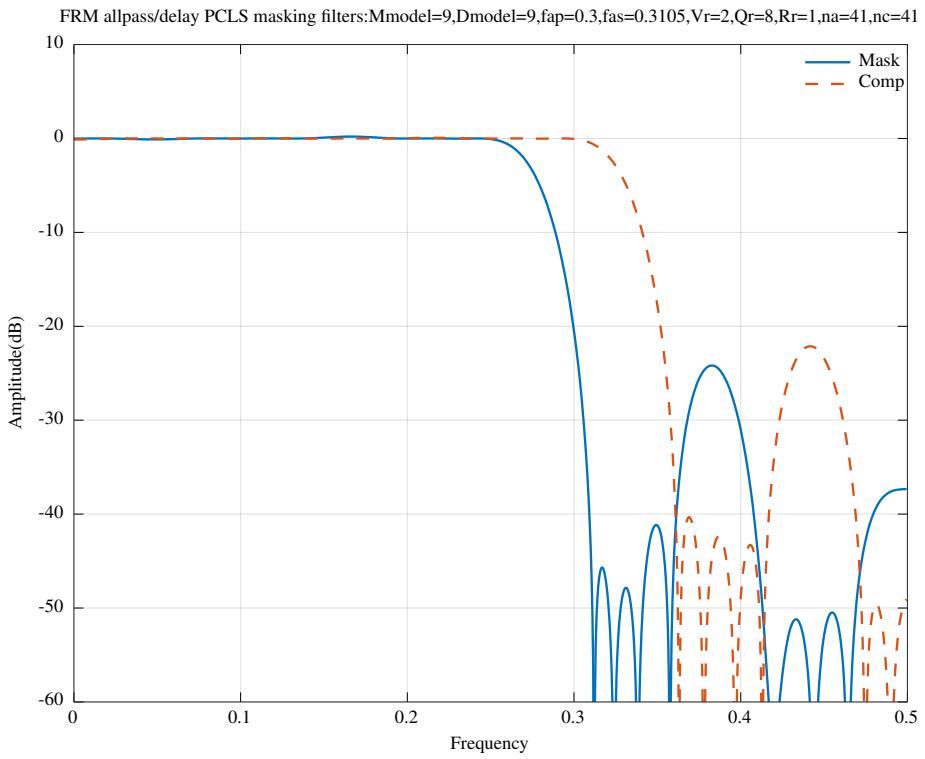


Figure 12.78: FRM gain-pole-zero format allpass model filter with SOCP and PCLS optimisation : masking filter responses

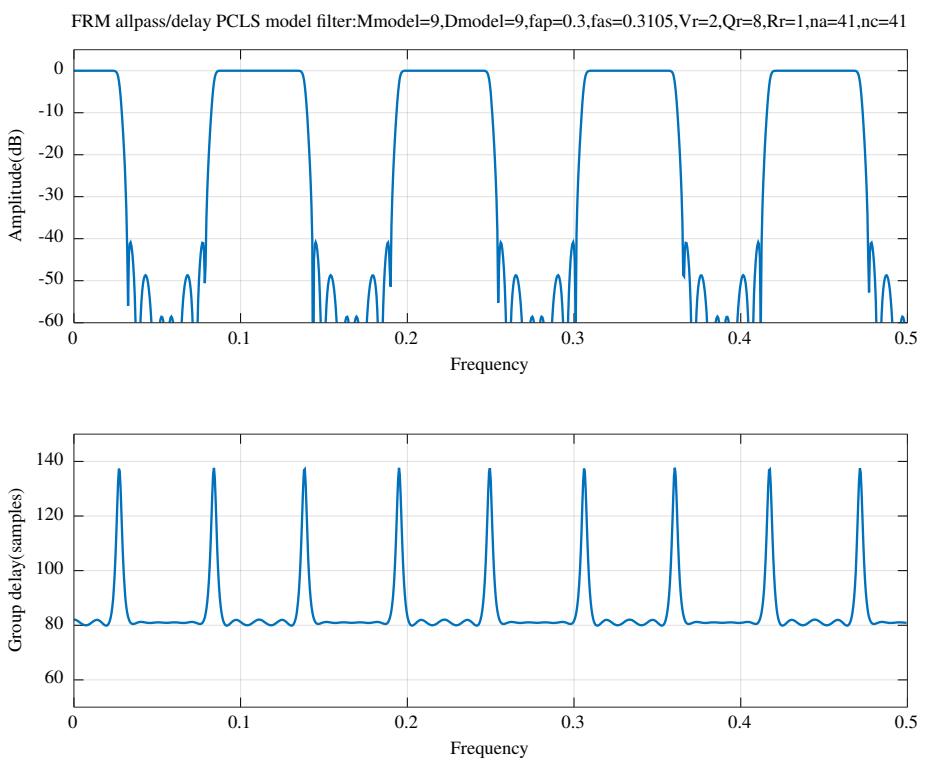


Figure 12.79: FRM gain-pole-zero format allpass model filter with SOCP and PCLS optimisation : model filter response

12.4.5 Design of an FRM digital filter with an IIR model filter consisting of parallel allpass filters

Figure 12.63 shows an FRM filter with a model filter consisting of parallel allpass filters. The filter transfer function is

$$H(z) = \frac{1}{2} [R(z^M) + S(z^M)] F_{M_a}(z) + \frac{1}{2} [R(z^M) - S(z^M)] F_{M_c}(z)$$

where $R(z)$ and $S(z)$ are allpass filters and, as previously, $F_{M_a}(z)$ and $F_{M_c}(z)$ are the masking and complementary masking filters

$$F_{M_a}(z) = \sum_{k=0}^{n_a-1} p_k z^{-k}$$

$$F_{M_c}(z) = \sum_{k=0}^{n_c-1} q_k z^{-k}$$

An equivalent filter is

$$H(z) = R(z^M) A(z) + S(z^M) B(z)$$

where

$$A(z) = \frac{1}{2} [F_{M_a}(z) + F_{M_c}(z)] = \sum_{k=0}^{n_m-1} a_k z^{-k}$$

$$B(z) = \frac{1}{2} [F_{M_a}(z) - F_{M_c}(z)] = \sum_{k=0}^{n_m-1} b_k z^{-k}$$

n_m is the larger of n_a and n_c and the a_k or b_k are zero-padded as required. If $\phi_R(\omega)$ and $\phi_S(\omega)$ are the phase responses of the allpass filters $R(z)$ and $S(z)$, then

$$H(\omega) = e^{i\phi_R(M\omega)} \sum_{k=0}^{n_m-1} a_k e^{-ik\omega} + e^{i\phi_S(M\omega)} \sum_{k=0}^{n_m-1} b_k e^{-ik\omega}$$

$$= \mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b} - i(\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b})$$

where

$$\begin{aligned} \mathbf{a} &= [a_0 \dots a_{n_m-1}]^T \\ \mathbf{b} &= [b_0 \dots b_{n_m-1}]^T \\ \mathbf{v} &= [0 \dots n_m - 1] \\ \mathbf{c}_R &= \cos[\mathbf{v}\omega - \phi_R(M\omega)] \\ \mathbf{c}_S &= \cos[\mathbf{v}\omega - \phi_S(M\omega)] \\ \mathbf{s}_R &= \sin[\mathbf{v}\omega - \phi_R(M\omega)] \\ \mathbf{s}_S &= \sin[\mathbf{v}\omega - \phi_S(M\omega)] \end{aligned}$$

The squared-amplitude and phase responses of $H(z)$ are

$$|H(\omega)|^2 = (\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b})^2 + (\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b})^2$$

$$\arg H(\omega) = -\arctan \frac{\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}}{\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}}$$

The group delay response is given by

$$|H(\omega)|^2 T(\omega) = (\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \left(\frac{\partial \mathbf{s}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{s}_S}{\partial \omega} \mathbf{b} \right) - (\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \left(\frac{\partial \mathbf{c}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{c}_S}{\partial \omega} \mathbf{b} \right)$$

where

$$\frac{\partial \mathbf{c}_R}{\partial \omega} = - \left[\mathbf{v} - \frac{\partial \phi_R(M\omega)}{\partial \omega} \right] \circ \mathbf{s}_R$$

$$\begin{aligned}\frac{\partial \mathbf{c}_S}{\partial \omega} &= - \left[\mathbf{v} - \frac{\partial \phi_S(M\omega)}{\partial \omega} \right] \circ \mathbf{s}_S \\ \frac{\partial \mathbf{s}_R}{\partial \omega} &= \left[\mathbf{v} - \frac{\partial \phi_R(M\omega)}{\partial \omega} \right] \circ \mathbf{c}_R \\ \frac{\partial \mathbf{s}_S}{\partial \omega} &= \left[\mathbf{v} - \frac{\partial \phi_S(M\omega)}{\partial \omega} \right] \circ \mathbf{c}_S\end{aligned}$$

The \circ symbol represents the *Hadamard* or element-wise product.

The gradients of the squared amplitude response are

$$\begin{aligned}\frac{\partial |H(\omega)|^2}{\partial \mathbf{r}} &= 2(\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \frac{\partial \mathbf{c}_R}{\partial \mathbf{r}} \mathbf{a} + 2(\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \frac{\partial \mathbf{s}_R}{\partial \mathbf{r}} \mathbf{a} \\ &= 2((\mathbf{c}_S \mathbf{b})(\mathbf{s}_R \mathbf{a}) - (\mathbf{s}_S \mathbf{b})(\mathbf{c}_R \mathbf{a})) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{s}} &= 2(\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \frac{\partial \mathbf{c}_S}{\partial \mathbf{s}} \mathbf{b} + 2(\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \frac{\partial \mathbf{s}_S}{\partial \mathbf{s}} \mathbf{b} \\ &= -2((\mathbf{c}_S \mathbf{b})(\mathbf{s}_R \mathbf{a}) - (\mathbf{s}_S \mathbf{b})(\mathbf{c}_R \mathbf{a})) \frac{\partial \phi_S(M\omega)}{\partial \mathbf{s}} \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{a}} &= 2(\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \mathbf{c}_R + 2(\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \mathbf{s}_R \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{b}} &= 2(\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \mathbf{c}_S + 2(\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \mathbf{s}_S\end{aligned}$$

The gradients of the group delay response are given by

$$\begin{aligned}\frac{\partial |H(\omega)|^2}{\partial \mathbf{r}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \mathbf{r}} &= \mathbf{s}_R \mathbf{a} \left(\frac{\partial \mathbf{s}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{s}_S}{\partial \omega} \mathbf{b} \right) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} + (\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \frac{\partial^2 \mathbf{s}_R}{\partial \mathbf{r} \partial \omega} \mathbf{a} \\ &\quad + \mathbf{c}_R \mathbf{a} \left(\frac{\partial \mathbf{c}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{c}_S}{\partial \omega} \mathbf{b} \right) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} - (\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \frac{\partial^2 \mathbf{c}_R}{\partial \mathbf{r} \partial \omega} \mathbf{a} \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{s}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \mathbf{s}} &= \mathbf{s}_S \mathbf{b} \left(\frac{\partial \mathbf{s}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{s}_S}{\partial \omega} \mathbf{b} \right) \frac{\partial \phi_S(M\omega)}{\partial \mathbf{s}} + (\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \frac{\partial^2 \mathbf{s}_S}{\partial \mathbf{s} \partial \omega} \mathbf{b} \\ &\quad + \mathbf{c}_S \mathbf{b} \left(\frac{\partial \mathbf{c}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{c}_S}{\partial \omega} \mathbf{b} \right) \frac{\partial \phi_S(M\omega)}{\partial \mathbf{s}} - (\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \frac{\partial^2 \mathbf{c}_S}{\partial \mathbf{s} \partial \omega} \mathbf{b} \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{a}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \mathbf{a}} &= \left(\frac{\partial \mathbf{s}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{s}_S}{\partial \omega} \mathbf{b} \right) \mathbf{c}_R + (\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \frac{\partial \mathbf{s}_R}{\partial \omega} \\ &\quad - \left(\frac{\partial \mathbf{c}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{c}_S}{\partial \omega} \mathbf{b} \right) \mathbf{s}_R - (\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \frac{\partial \mathbf{c}_R}{\partial \omega} \\ \frac{\partial |H(\omega)|^2}{\partial \mathbf{b}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \mathbf{b}} &= \left(\frac{\partial \mathbf{s}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{s}_S}{\partial \omega} \mathbf{b} \right) \mathbf{c}_S + (\mathbf{c}_R \mathbf{a} + \mathbf{c}_S \mathbf{b}) \frac{\partial \mathbf{s}_S}{\partial \omega} \\ &\quad - \left(\frac{\partial \mathbf{c}_R}{\partial \omega} \mathbf{a} + \frac{\partial \mathbf{c}_S}{\partial \omega} \mathbf{b} \right) \mathbf{s}_S - (\mathbf{s}_R \mathbf{a} + \mathbf{s}_S \mathbf{b}) \frac{\partial \mathbf{c}_S}{\partial \omega}\end{aligned}$$

where

$$\begin{aligned}\frac{\partial^2 \mathbf{c}_R}{\partial \mathbf{r} \partial \omega} \mathbf{a} &= (\mathbf{s}_R \mathbf{a}) \frac{\partial^2 \phi_R(M\omega)}{\partial \mathbf{r} \partial \omega} + \left(\frac{\partial \mathbf{s}_R}{\partial \omega} \mathbf{a} \right) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \\ \frac{\partial^2 \mathbf{c}_S}{\partial \mathbf{s} \partial \omega} \mathbf{b} &= (\mathbf{s}_S \mathbf{b}) \frac{\partial^2 \phi_S(M\omega)}{\partial \mathbf{s} \partial \omega} + \left(\frac{\partial \mathbf{s}_S}{\partial \omega} \mathbf{b} \right) \frac{\partial \phi_S(M\omega)}{\partial \mathbf{s}} \\ \frac{\partial^2 \mathbf{s}_R}{\partial \mathbf{r} \partial \omega} \mathbf{a} &= -(\mathbf{c}_R \mathbf{a}) \frac{\partial^2 \phi_R(M\omega)}{\partial \mathbf{r} \partial \omega} - \left(\frac{\partial \mathbf{c}_R}{\partial \omega} \mathbf{a} \right) \frac{\partial \phi_R(M\omega)}{\partial \mathbf{r}} \\ \frac{\partial^2 \mathbf{s}_S}{\partial \mathbf{s} \partial \omega} \mathbf{b} &= -(\mathbf{c}_S \mathbf{b}) \frac{\partial^2 \phi_S(M\omega)}{\partial \mathbf{s} \partial \omega} - \left(\frac{\partial \mathbf{c}_S}{\partial \omega} \mathbf{b} \right) \frac{\partial \phi_S(M\omega)}{\partial \mathbf{s}}\end{aligned}$$

The calculation of the filter response and gradients is implemented in the Octave function *iir_frm_parallel_allpass.m* which is exercised by the Octave script *iir_frm_parallel_allpass_test.m*.

The Octave script *iir_frm_parallel_allpass_socp_slb_test.m* designs an FRM filter with the following specification:

```

tol=0.002 % Tolerance on coefficient update vector
ctol=2e-05 % Tolerance on constraints
n=400 % Frequency points across the band
mr=8 % R model filter denominator order
ms=7 % S model filter denominator order
na=25 % FIR masking filter length (order+1)
nc=25 % FIR complementary masking filter length (order+1)
Mmodel=9 % Model filter decimation factor
Dmodel=0 % Model filter nominal pass band group delay
dmask=0 % FIR masking filter delay
fap=0.3 % Pass band edge
dBap=0.05 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
tpr=Inf % Pass band delay peak-to-peak ripple
Wtp=0 % Pass band delay weight
fas=0.31125 % Stop band edge
dBas=40 % Stop band attenuation ripple
Was=100 % Stop band weight
rho=0.968750 % Constraint on allpass pole radius

```

In this example the FIR masking filters are not required to be linear phase and there is no constraint on the group delay of the filter. The Octave script *iir_frm_parallel_allpass_socp_slb_test.m* calls the Octave function *iir_frm_parallel_allpass_socp_mmse* to minimise the response error and the coefficient step size with the SeDuMi SOCP solver. The initial filter was calculated by the Octave script *tarczynski_frm_parallel_allpass_test.m* with the WISE method of Tarczynski et al. (see Section 10.1.5). The response of the initial filter is shown in Figure 12.80. The initial filter was SOCP and PCLS optimised with the Octave function

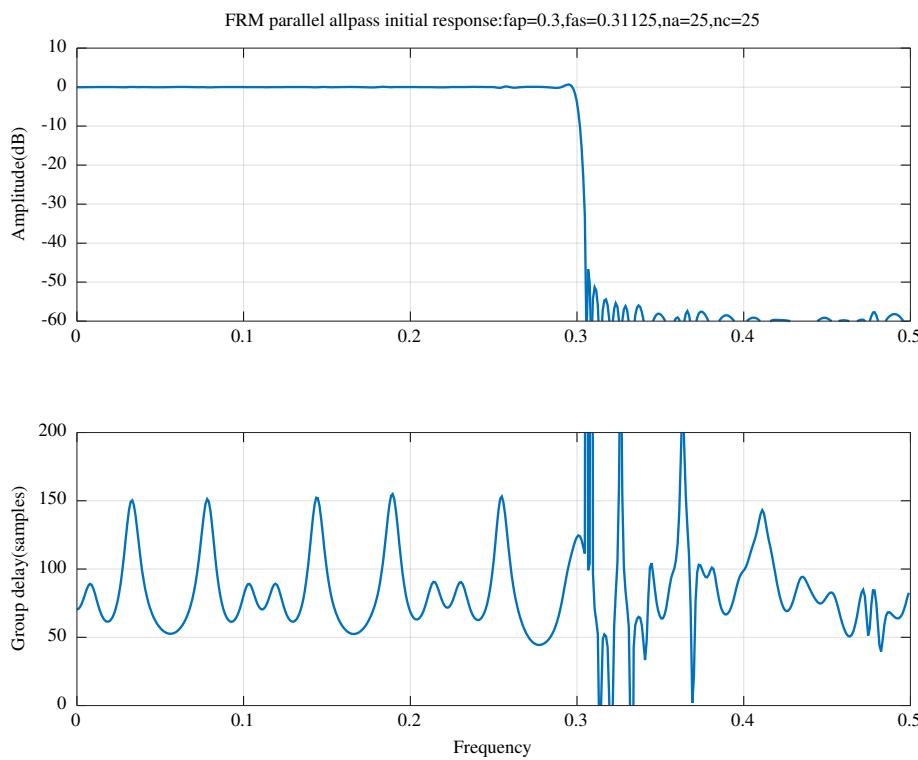


Figure 12.80: FRM filter with gain-pole-zero format parallel allpass model filter : initial response with WISE optimisation

iir_frm_parallel_allpass_slb. The resulting model filter parallel allpass filter denominator polynomials are

```
r = [ 1.0000000000, -0.4000476421, 0.8174669849, -0.5071247226, ...
0.1411606488, -0.0686720573, 0.0811456489, 0.0268175181, ...
0.0087385004 ]';
```

and

```
s = [ 1.0000000000, -0.5043138273, 0.5732659143, -0.4482102603, ...
0.1394717006, -0.0223037751, 0.0647311896, 0.0080494330 ]';
```

The FIR masking filter polynomial is

```
aa = [ 0.0068716189, 0.0089742952, -0.0194582484, -0.0453051604, ...
-0.0014628206, 0.0271722689, -0.0171769260, -0.0012857000, ...
0.0507631991, -0.0088078805, 0.0279492906, 0.3549249362, ...
0.5550132250, 0.2295824107, -0.1715499525, -0.1188162719, ...
0.0828646318, 0.0369642189, -0.0617437736, -0.0037534672, ...
0.0615865024, 0.0261401931, -0.0095240170, -0.0058666616, ...
-0.0048744084 ]';
```

and the complementary FIR masking filter polynomial is

```
ac = [ 0.0102485092, 0.0424757474, -0.0399408499, -0.1033522294, ...
0.0512446527, 0.0821716218, -0.1230568186, -0.0364256416, ...
0.1765263303, -0.0561118967, -0.0648088626, 0.4636909465, ...
0.5674588849, 0.1136927378, -0.0980924037, -0.0546817682, ...
-0.0454089995, 0.0546627793, 0.0523121745, -0.0613078522, ...
-0.0137168240, 0.0860744060, 0.0153586832, -0.0544355531, ...
-0.0150301135 ]';
```

Figure 12.81 shows the overall response of the resulting SOCP and PCLS optimised FRM filter. Figure 12.82 shows the passband response of the resulting FRM filter. Figure 12.83 shows the impulse response of the resulting FRM masking filters. Figure 12.84 shows the responses of the resulting FRM masking filters. Figure 12.85 shows the response of the resulting FRM model filter.

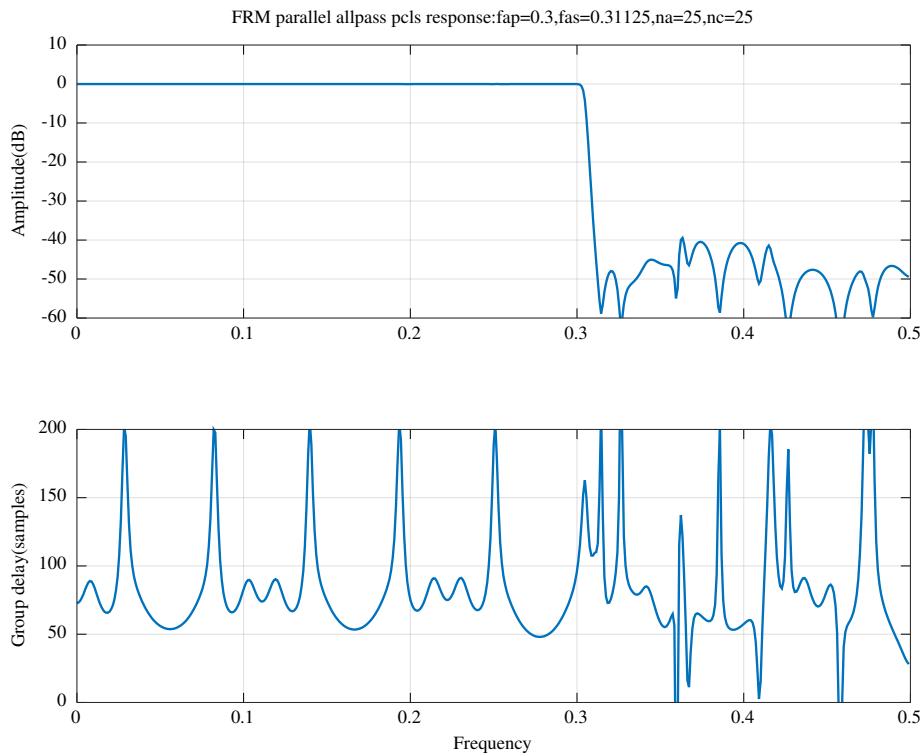


Figure 12.81: FRM filter with gain-pole-zero format parallel allpass model filter : SOCP PCLS optimised overall response

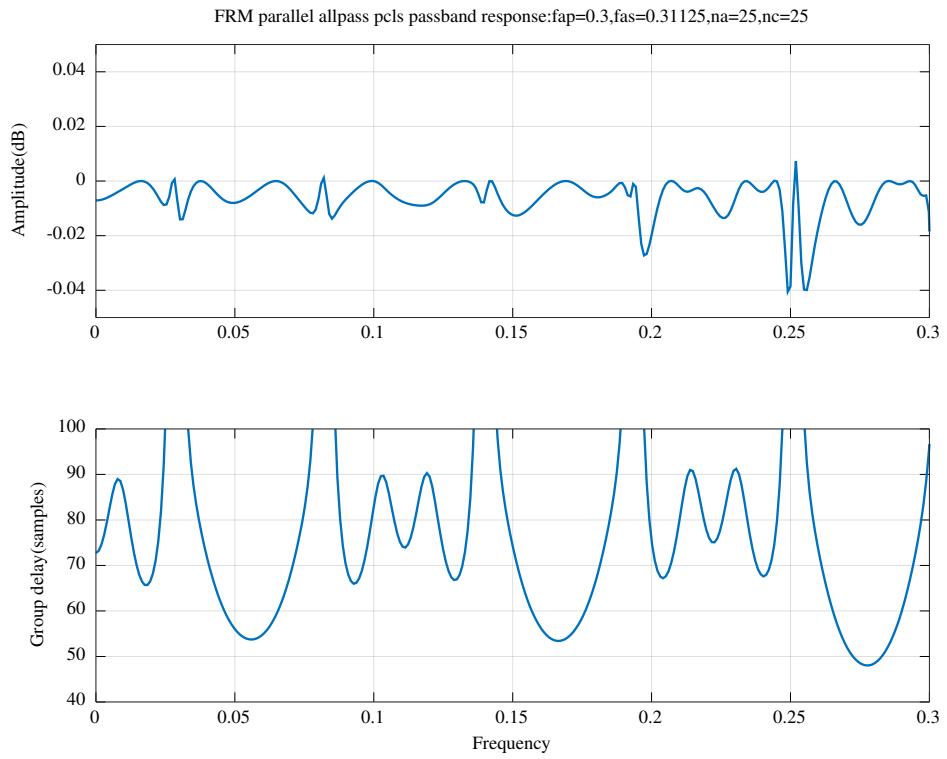


Figure 12.82: FRM filter with gain-pole-zero format parallel allpass model filter : SOCP PCLS optimised passband response

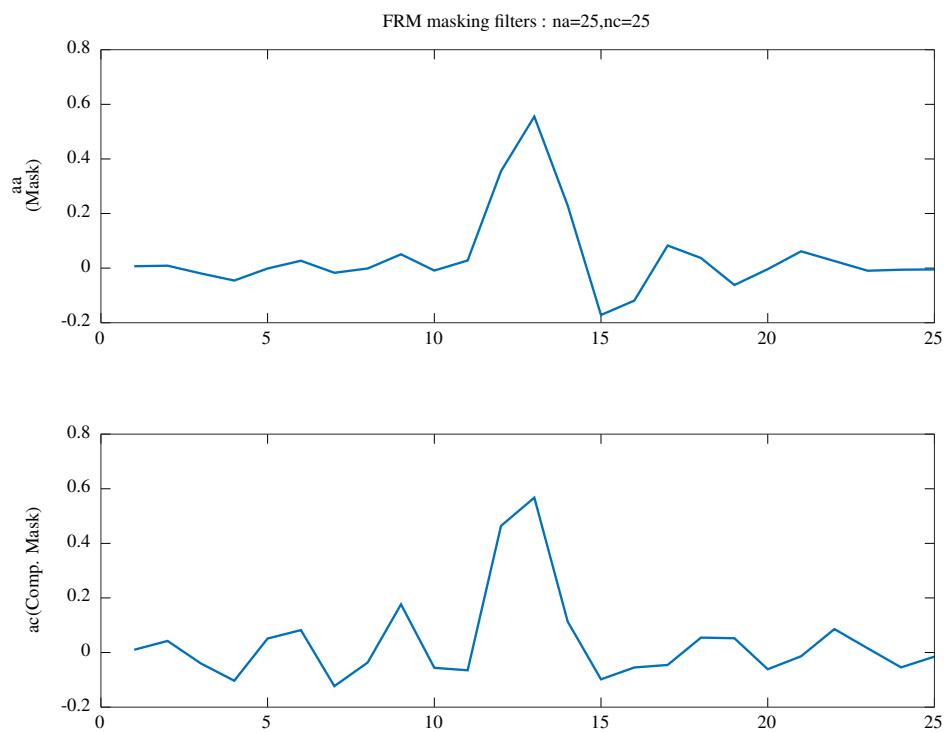


Figure 12.83: FRM filter with gain-pole-zero format parallel allpass model filter : SOCP PCLS optimised FIR masking filters

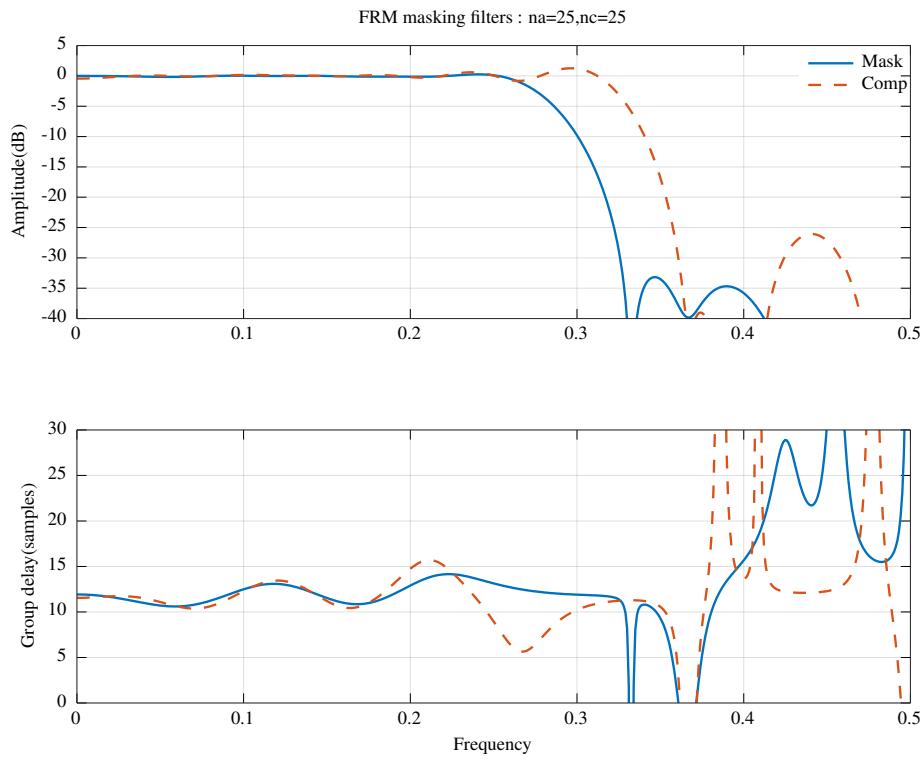


Figure 12.84: FRM filter with gain-pole-zero format parallel allpass model filter : SOCP PCLS optimised FIR masking filter responses

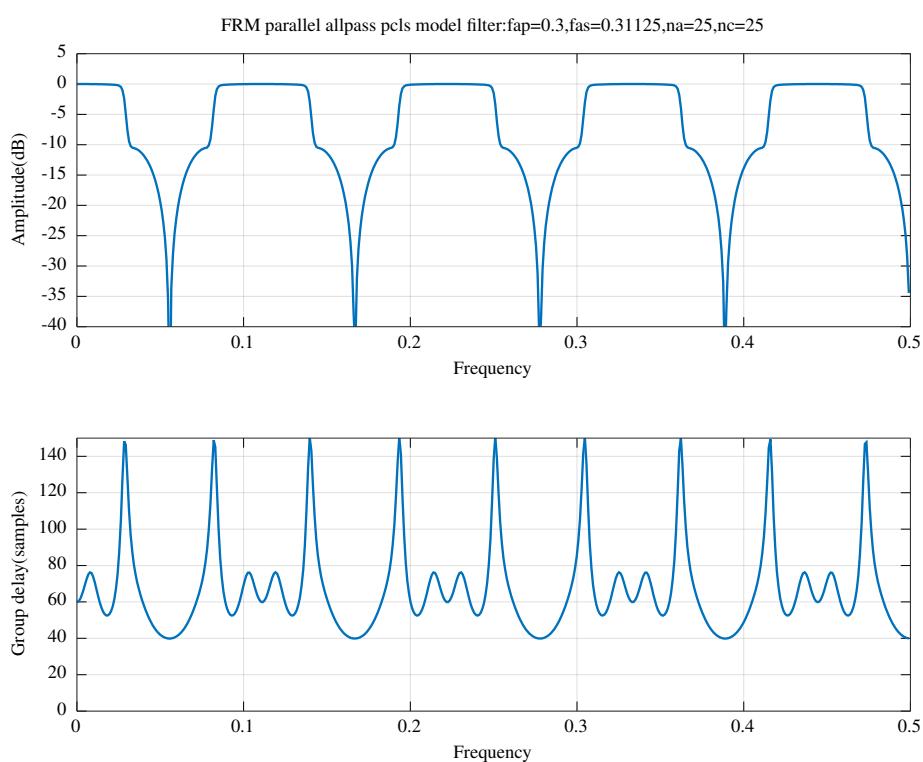


Figure 12.85: FRM filter with gain-pole-zero format parallel allpass model filter : SOCP PCLS optimised model filter response

12.4.6 Design of an FRM half-band digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation

Milić et al. describe the design of an FRM half-band filter with the structure shown in Figure 12.63. In this case the model filters are synthesised as the parallel combination of a pure delay, $S(z) = z^{-D}$ and an all-pass filter with coefficients only in z^2 , $R(z^2)$. The coefficients of the denominator polynomial of $R(z^2)$ are r_k . The masking filters are symmetric FIR filters with even order N_m , a multiple of 4 and delay $d = \frac{N_m}{2}$. The transfer function of the FRM half-band filter is:

$$\begin{aligned} H(z) &= \frac{1}{2} [R(z^{2M}) + z^{-DM}] F_{M_a}(z) + \frac{1}{2} [R(z^{2M}) - z^{-DM}] F_{M_c}(z) \\ &= \frac{1}{2} R(z^{2M}) [F_{M_a}(z) + F_{M_c}(z)] + \frac{1}{2} z^{-DM} [F_{M_a}(z) - F_{M_c}(z)] \end{aligned}$$

where M is the decimation factor of the model filter. The nominal delay of the FRM filter is $DM + d$.

The power-complementary branches of a half-band filter are symmetric in frequency. Section 12.4.1 shows the calculation of the band edges of the masking filters required for the FRM filter corresponding to these model filters. Saramäki et al. [92, Figure 2] show that the masking filters of the FRM half-band filter are related by:

$$F_{M_c}(z) = -z^{-d} + F_{M_a}(-z)$$

The polyphase decomposition of $F_{M_a}(z)$ is:

$$F_{M_a}(z) = U(z^2) + z^{-1}V(z^2)$$

where $U(z)$ is a symmetric FIR filter with even order $\frac{N_m}{2}$ and $V(z)$ is a symmetric FIR filter with odd order $\frac{N_m}{2} - 1$.

The transfer function of the FRM half-band filter is, in terms of the polyphase decomposition of F_{M_a} :

$$\begin{aligned} F_{M_a}(z) + F_{M_c}(z) &= 2U(z^2) - z^{-d} \\ F_{M_a}(z) - F_{M_c}(z) &= 2z^{-1}V(z^2) + z^{-d} \end{aligned}$$

so that:

$$H(z) = \frac{1}{2} z^{-DM-d} + \frac{1}{2} [R(z^{2M}) (2U(z^2) - z^{-d}) + 2z^{-DM} z^{-1} V(z^2)]$$

It is convenient to express $U(z^2)$ and $z^{-1}V(z^2)$ in terms of the coefficients, u_k and v_k :

$$\begin{aligned} U(z^2) &= \sum_{k=0}^d u_k z^{-2k} \\ &= u_{\frac{d}{2}} z^{-d} + \sum_{k=0}^{\frac{d}{2}-1} u_k z^{-2k} + \sum_{k=\frac{d}{2}+1}^d u_k z^{-2k} \\ &= u_{\frac{d}{2}} z^{-d} + \sum_{k=0}^{\frac{d}{2}-1} u_k (z^{-2k} + z^{2k-2d}) \\ &= z^{-d} \left[u_{\frac{d}{2}} + \sum_{k=0}^{\frac{d}{2}-1} u_k (z^{-2k+d} + z^{2k-d}) \right] \end{aligned}$$

and:

$$\begin{aligned} z^{-1}V(z^2) &= z^{-1} \sum_{k=0}^{d-1} v_k z^{-2k} \\ &= z^{-1} \sum_{k=0}^{\frac{d}{2}-1} v_k z^{-2k} + z^{-1} \sum_{k=\frac{d}{2}}^{d-1} v_k z^{-2k} \\ &= z^{-1} \sum_{k=0}^{\frac{d}{2}-1} v_k (z^{-2k} + z^{2k-2d+2}) \end{aligned}$$

$$= z^{-d} \sum_{k=0}^{\frac{d}{2}-1} v_k (z^{-2k+d-1} + z^{2k-d+1})$$

Rearranging $H(z)$:

$$H(z) = z^{-DM-d} \left[z^{DM} R(z^{2M}) \left(-\frac{1}{2} + z^d U(z^2) \right) + \left(\frac{1}{2} + z^d z^{-1} V(z^2) \right) \right]$$

so that the zero-phase frequency response of the half-band FRM filter, $H(z)$, is:

$$H(\omega) = e^{j[DM\omega + \phi_R(2M\omega)]} A(\omega) + B(\omega)$$

where $\phi_R(2M\omega)$ is the phase response of the all-pass filter $R(z^{2M})$ and:

$$\begin{aligned} A(\omega) &= -\frac{1}{2} + u_{\frac{d}{2}} + \sum_{k=0}^{\frac{d}{2}-1} 2u_k \cos[\omega(2k-d)] \\ B(\omega) &= \frac{1}{2} + \sum_{k=0}^{\frac{d}{2}-1} 2v_k \cos[\omega(2k-d+1)] \end{aligned}$$

The squared-magnitude and phase responses of the zero phase response of the FRM half-band filter are similar to those shown in Section 12.4.4:

$$\begin{aligned} |H(\omega)|^2 &= A^2(\omega) + B^2(\omega) + 2A(\omega)B(\omega)\cos\phi_Z(M\omega) \\ \arg H(\omega) &= \arctan \frac{A(\omega)\sin\phi_Z(M\omega)}{A(\omega)\cos\phi_Z(M\omega) + B(\omega)} \end{aligned}$$

where $\phi_Z(\omega) = D\omega + \phi_R(2\omega)$.

The group delay response, $T(\omega)$, of the zero phase response of the FRM half-band filter is given by:

$$\begin{aligned} |H(\omega)|^2 T(\omega) &= - (A^2(\omega) + A(\omega)B(\omega)\cos\phi_Z(M\omega)) \frac{\partial\phi_Z(M\omega)}{\partial\omega} \dots \\ &\quad - \sin\phi_Z(M\omega) \left[B(\omega) \frac{\partial A(\omega)}{\partial\omega} - A(\omega) \frac{\partial B(\omega)}{\partial\omega} \right] \end{aligned}$$

where:

$$\begin{aligned} \frac{\partial A(\omega)}{\partial\omega} &= -2 \sum_{k=0}^{\frac{d}{2}-1} (2k-d) u_k \sin[\omega(2k-d)] \\ \frac{\partial B(\omega)}{\partial\omega} &= -2 \sum_{k=0}^{\frac{d}{2}-1} (2k-d+1) v_k \sin[\omega(2k-d+1)] \end{aligned}$$

The gradients of $|H(\omega)|^2$ with respect to the coefficients are:

$$\begin{aligned} \frac{\partial |H(\omega)|^2}{\partial r_k} &= -2A(\omega)B(\omega)\sin\phi_Z(M\omega) \frac{\partial\phi_R(2M\omega)}{\partial r_k} \\ \frac{\partial |H(\omega)|^2}{\partial u_k} &= 2(A(\omega) + B(\omega)\cos\phi_Z(M\omega)) \frac{\partial A(\omega)}{\partial u_k} \\ \frac{\partial |H(\omega)|^2}{\partial v_k} &= 2(B(\omega) + A(\omega)\cos\phi_Z(M\omega)) \frac{\partial B(\omega)}{\partial v_k} \end{aligned}$$

and:

$$\begin{aligned} \frac{\partial A(\omega)}{\partial u_k} &= \begin{cases} 1 & k = \frac{d}{2} \\ 2 \cos[\omega(2k-d)] & otherwise \end{cases} \\ \frac{\partial B(\omega)}{\partial v_k} &= 2 \cos[\omega(2k-d+1)] \end{aligned}$$

The gradients of $T(\omega)$ with respect to the coefficients are given by:

$$\begin{aligned}\frac{\partial |H(\omega)|^2}{\partial r_k} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial r_k} &= (A^2(\omega) + A(\omega)B(\omega)\cos\phi_Z(M\omega)) \frac{\partial^2 \phi_R(2M\omega)}{\partial \omega \partial r_k} \dots \\ &\quad + A(\omega)B(\omega)\sin\phi_Z(M\omega) \frac{\partial \phi_Z(M\omega)}{\partial \omega} \frac{\partial \phi_R(2M\omega)}{\partial r_k} \dots \\ &\quad - \cos\phi_Z(M\omega) \left[B(\omega) \frac{\partial A(\omega)}{\partial \omega} - A(\omega) \frac{\partial B(\omega)}{\partial \omega} \right] \frac{\partial \phi_R(2M\omega)}{\partial r_k} \dots \\ \frac{\partial |H(\omega)|^2}{\partial u_k} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial u_k} &= -(2A(\omega) + B(\omega)\cos\phi_Z(M\omega)) \frac{\partial \phi_Z(M\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial u_k} \dots \\ &\quad - \sin\phi_Z(M\omega) \left[B(\omega) \frac{\partial^2 A(\omega)}{\partial \omega \partial u_k} - \frac{\partial B(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial u_k} \right] \dots \\ \frac{\partial |H(\omega)|^2}{\partial v_k} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial v_k} &= -A(\omega)\cos\phi_Z(M\omega) \frac{\partial \phi_Z(M\omega)}{\partial \omega} \frac{\partial B(\omega)}{\partial v_k} \dots \\ &\quad - \sin\phi_Z(M\omega) \left[\frac{\partial A(\omega)}{\partial \omega} \frac{\partial B(\omega)}{\partial v_k} - A(\omega) \frac{\partial^2 B(\omega)}{\partial \omega \partial v_k} \right] \dots\end{aligned}$$

The Octave script *schurOneMAPlattice_frm_halfband_socp_slb_test.m* designs an FRM half-band filter with a model filter implemented as a Schur one-multiplier all-pass lattice. The filter specification is:

```
tol=1e-06 % Tolerance on coefficient update vector
n=800 % Frequency points across the band
mr=5 % Allpass model filter denominator order
na=33 % FIR masking filter length (order+1)
Mmodel=7 % Model filter FRM decimation factor
Dmodel=9 % Model filter nominal pass band group delay
dmask=16 % FIR masking filter delay
Tnominal=79 % Nominal FRM filter group delay
fap=0.24 % Pass band edge
dBap=0.05 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
tpr=0.334 % Pass band delay peak-to-peak ripple
Wtp=0.2 % Pass band delay weight
fas=0.26 % Stop band edge
dBas=45 % Stop band attenuation ripple
Was=100 % Stop band weight
rho=0.968750 % Constraint on all-pass lattice coefficients
```

The initial filter is designed by the Octave script *tarczynski_frm_halfband_test.m* with the WISE method of *Tarczynski et al.* as shown in Section 10.1.5. Figure 12.86 shows the overall response of the initial FRM filter.

SOCP and PCLS optimisation of the initial filter results in a model filter allpass filter with following lattice coefficients:

```
k2 = [ 0.5541159714, -0.1235054400, 0.0419578980, -0.0136390237, ...  
0.0020565211 ]';
```

```
epsilon0 = [ 1, 1, -1, -1, ...  
1 ];
```

The corresponding denominator polynomial of the allpass model filter is:

```
r2 = [ 1.0000000000, 0.4798972922, -0.1016479773, 0.0351951888, ...  
-0.0126520471, 0.0020565211 ];
```

The FIR masking filter polynomials are:

```
u2 = [ -0.0005745653, 0.0023384306, -0.0070502278, 0.0130140428, ...  
-0.0310645326, 0.0344544514, -0.0508990361, 0.0577324312, ...  
0.4385696117 ]';
```

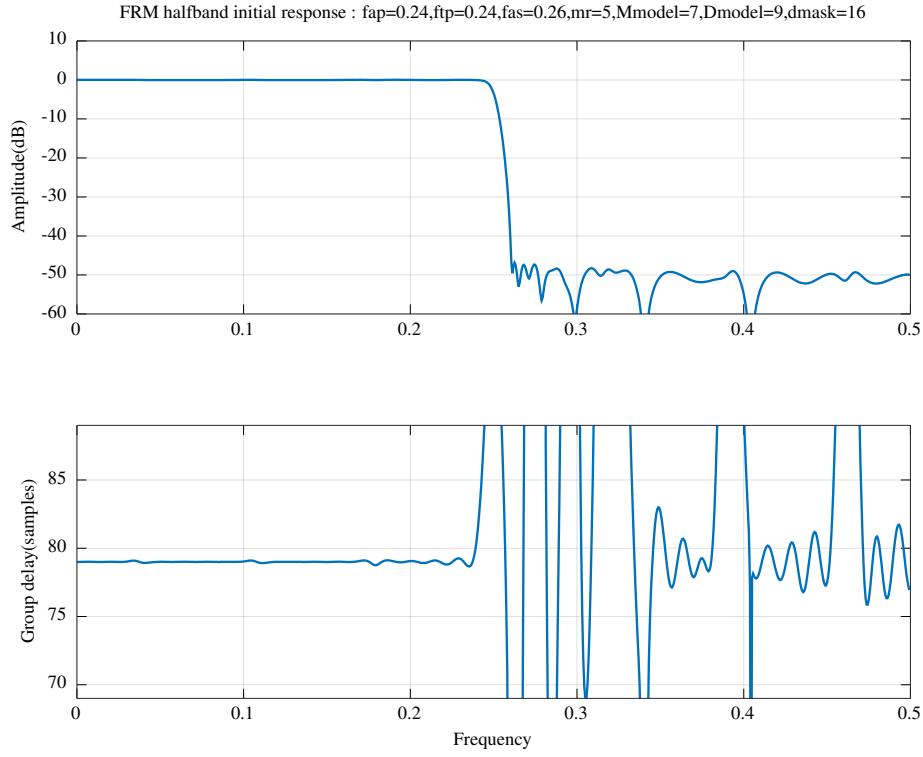


Figure 12.86: FRM half-band filter : initial response

```
v2 = [ 0.0066212307, -0.0044127894, 0.0067211174, -0.0028602101, ...
-0.0069802620, 0.0308565055, -0.0815328874, 0.3143514031 ]';
```

Figure 12.87 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 12.88 shows the passband response of the resulting FRM filter. Figure 12.89 shows the responses of the resulting FRM masking filters. Figure 12.90 shows the response of the resulting FRM model filter.

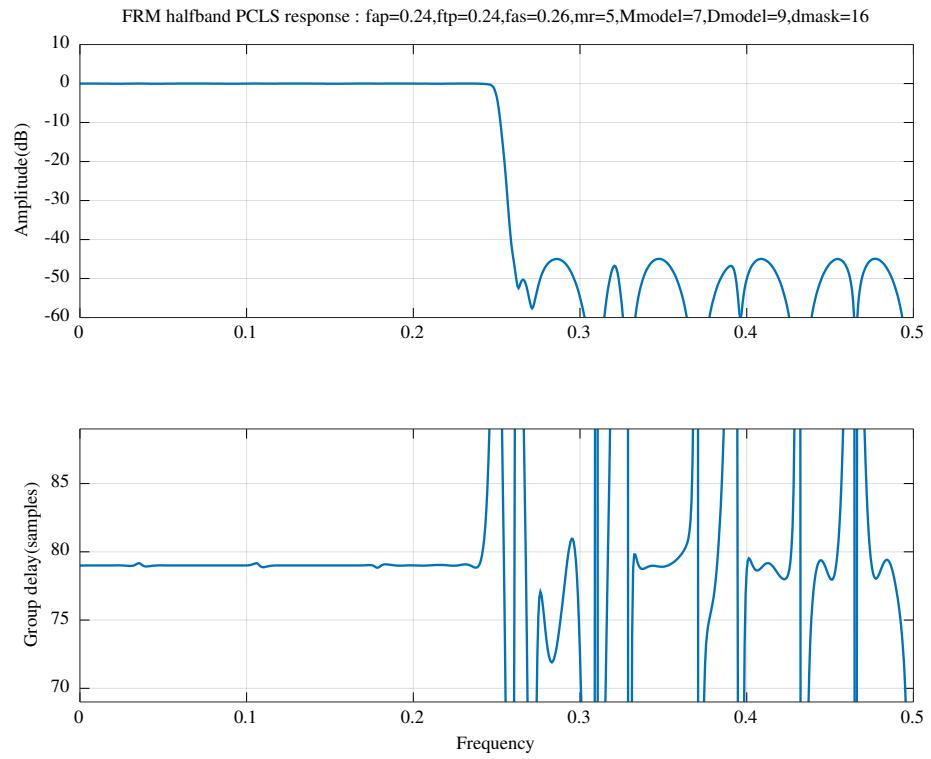


Figure 12.87: FRM half-band filter : overall response after SOCP and PCLS optimisation

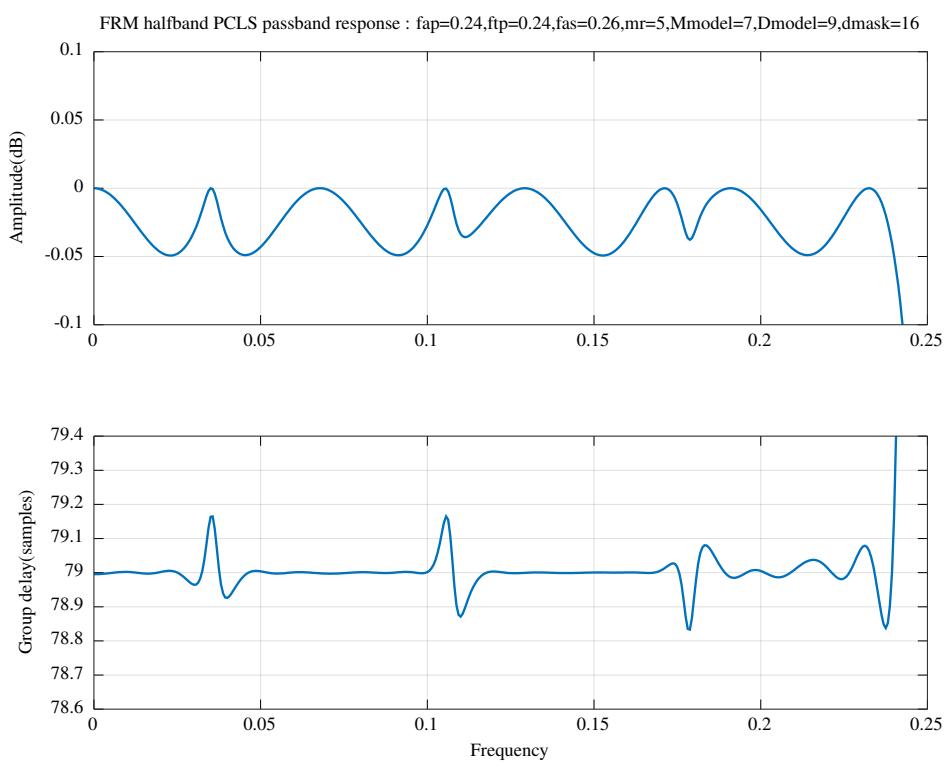


Figure 12.88: FRM half-band filter : passband response after SOCP and PCLS optimisation

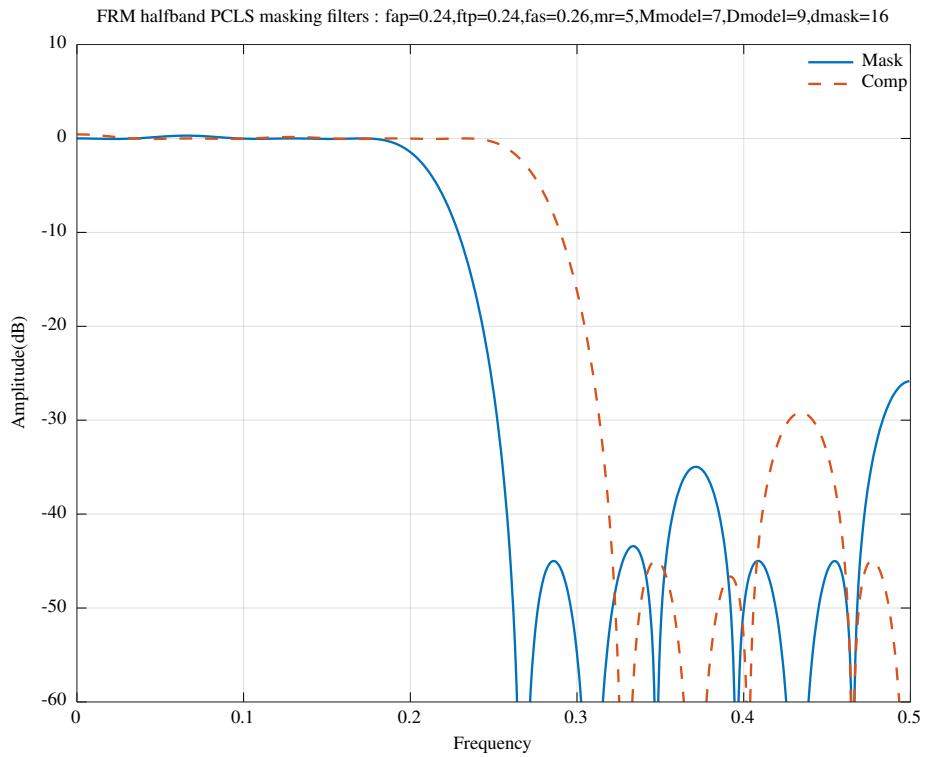


Figure 12.89: FRM half-band filter with SOCP and PCLS optimisation : masking filter responses

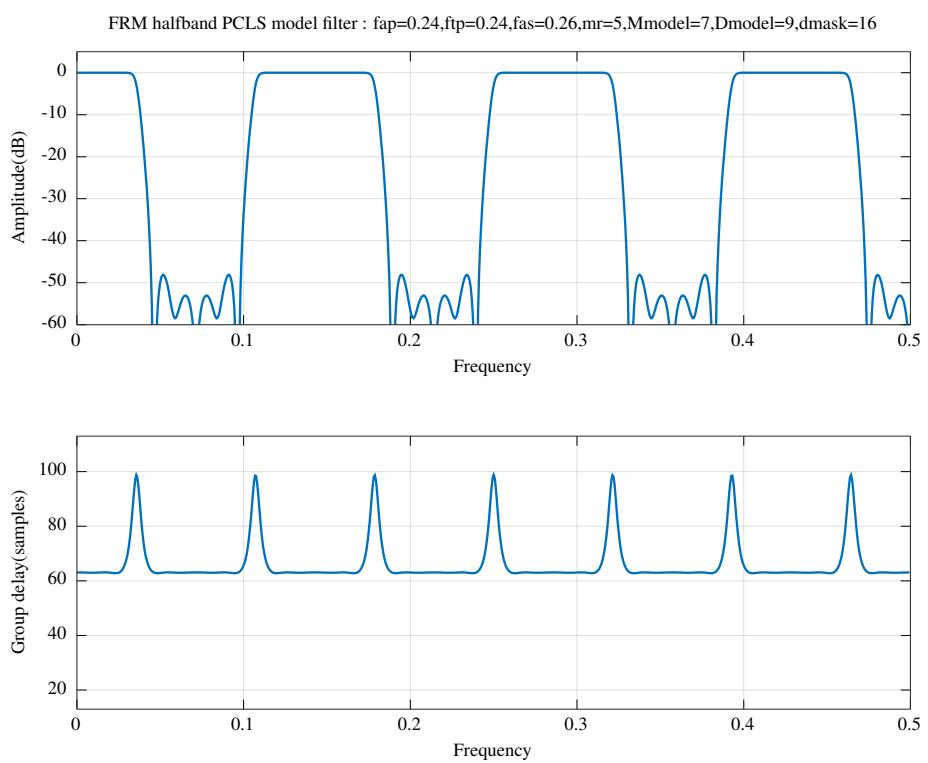


Figure 12.90: FRM half-band filter with SOCP and PCLS optimisation : model filter response

12.4.7 Design of an FRM Hilbert digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation

Milić et al. [53] point out that shifting the transfer function of the FRM half-band filter, $H(z)$, of Section 12.4.6, right-wards by $\frac{\pi}{2}$ along the frequency axis results in a filter, $H_A(z)$, that generates the *analytic* signal. Suppose the half-band filter response is:

$$H(z) = \frac{1}{2}z^{-N} + \frac{1}{2}Q(z^2)$$

where $N + 1$ is a multiple of 4. The filter, $H_A(z)$, that generates the analytic signal is:

$$\begin{aligned} H_A(z) &= 2\imath H(-\imath z) \\ &= z^{-N} + \imath Q(-z^2) \end{aligned}$$

In other words, $H_H(z) = Q(-z^2)$ is a Hilbert transform filter. For convenience, assume that the masking filter order, N_m , is a multiple of 8, that M and D are odd and that $DM + 1$ is a multiple of 4. Then:

$$H_H(z) = R(-z^{2M})(2U(-z^2) - z^{-d}) + 2z^{-DM-1}V(-z^2)$$

where:

$$\begin{aligned} U(-z^2) &= \sum_{k=0}^d u_k (-z^2)^{-k} \\ &= (-1)^{-\frac{d}{2}} u_{\frac{d}{2}} z^{-d} + \sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} u_k z^{-2k} + \sum_{k=\frac{d}{2}+1}^d (-1)^{-k} u_k z^{-2k} \\ &= u_{\frac{d}{2}} z^{-d} + \sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} u_k (z^{-2k} + z^{2k-2d}) \\ &= z^{-d} \left[u_{\frac{d}{2}} + \sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} u_k (z^{-2k+d} + z^{2k-d}) \right] \end{aligned}$$

and:

$$\begin{aligned} z^{-1}V(-z^2) &= z^{-1} \sum_{k=0}^{d-1} v_k (-z^2)^{-k} \\ &= z^{-1} \sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} v_k z^{-2k} + z^{-1} \sum_{k=\frac{d}{2}}^{d-1} (-1)^{-k} v_k z^{-2k} \\ &= z^{-d} \sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} v_k (z^{-2k+d-1} - z^{2k-d+1}) \end{aligned}$$

If $r(z^2)$ is the denominator polynomial of the prototype all-pass filter prototype, $R(z^2)$, then:

$$\begin{aligned} R(-z^2) &= \frac{(-z^{-2})^{N_{model}} r(-z^{-2})}{r(-z^2)} \\ &= (-1)^{N_{model}} \frac{z^{-2N_{model}} r(-z^{-2})}{r(-z^2)} \end{aligned}$$

where N_{model} is the order of $r(z)$. If N_{model} is odd then the zero frequency gain of $R(-z^2)$ is -1 .

For convenience, rename the FRM Hilbert filter coefficients in terms of the coefficients of the FRM half-band filter as $r'_k = (-1)^{-k} r_k$, $u'_k = (-1)^{-k} u_k$ and $v'_k = (-1)^{-k} v_k$. The zero-phase frequency response of the Hilbert filter, $H_H(z)$, is:

$$H_H(\omega) = e^{i[D M \omega + \phi_{R_H}(2M\omega)]} A_H(\omega) + \imath B_H(\omega)$$

where:

$$A_H(\omega) = -1 + 2u'_{\frac{d}{2}} + 4 \sum_{k=0}^{\frac{d}{2}-1} u'_k \cos[\omega(2k-d)]$$

$$B_H(\omega) = -4 \sum_{k=0}^{\frac{d}{2}-1} v'_k \sin [\omega(2k-d+1)]$$

and $\phi_{R_H}(2M\omega)$ is the phase response of the modified all-pass filter $R_H(z^{2M}) = R(-z^{2M})$.

The squared-magnitude and phase responses of the zero phase response of the FRM Hilbert filter are:

$$|H_H(\omega)|^2 = A_H^2(\omega) + B_H^2(\omega) + 2A_H(\omega)B_H(\omega)\sin\phi_{Z_H}(M\omega)$$

and:

$$\arg H_H(\omega) = \arctan \frac{A_H(\omega)\sin\phi_{Z_H}(M\omega) + B_H(\omega)}{A_H(\omega)\cos\phi_{Z_H}(M\omega)}$$

where $\phi_{Z_H}(\omega) = D\omega + \phi_{R_H}(2\omega)$.

The group delay response, $T_H(\omega)$, of the zero phase response of the FRM Hilbert filter is given by:

$$\begin{aligned} |H_H(\omega)|^2 T_H(\omega) &= - (A_H^2(\omega) + A_H(\omega)B_H(\omega)\sin\phi_{Z_H}(M\omega)) \frac{\partial\phi_{Z_H}(M\omega)}{\partial\omega} \dots \\ &\quad + \cos\phi_{Z_H}(M\omega) \left[B_H(\omega) \frac{\partial A_H(\omega)}{\partial\omega} - A_H(\omega) \frac{\partial B_H(\omega)}{\partial\omega} \right] \end{aligned}$$

where:

$$\begin{aligned} \frac{\partial A_H(\omega)}{\partial\omega} &= -4 \sum_{k=0}^{\frac{d}{2}-1} (2k-d) u'_k \sin [\omega(2k-d)] \\ \frac{\partial B_H(\omega)}{\partial\omega} &= -4 \sum_{k=0}^{\frac{d}{2}-1} (2k-d+1) v'_k \cos [\omega(2k-d+1)] \end{aligned}$$

The gradients of $|H_H(\omega)|^2$ with respect to the coefficients are:

$$\begin{aligned} \frac{\partial |H_H(\omega)|^2}{\partial r'_k} &= 2A_H(\omega)B_H(\omega)\cos\phi_{Z_H}(M\omega) \frac{\partial\phi_{R_H}(2M\omega)}{\partial r'_k} \\ \frac{\partial |H_H(\omega)|^2}{\partial u'_k} &= 2(A_H(\omega) + B_H(\omega)\sin\phi_{Z_H}(M\omega)) \frac{\partial A_H(\omega)}{\partial u'_k} \\ \frac{\partial |H_H(\omega)|^2}{\partial v'_k} &= 2(B_H(\omega) + A_H(\omega)\sin\phi_{Z_H}(M\omega)) \frac{\partial B_H(\omega)}{\partial v'_k} \end{aligned}$$

where:

$$\begin{aligned} \frac{\partial A_H(\omega)}{\partial u'_k} &= \begin{cases} 1 & k = \frac{d}{2} \\ 4\cos[\omega(2k-d)] & otherwise \end{cases} \\ \frac{\partial B_H(\omega)}{\partial v'_k} &= -4\sin[\omega(2k-d+1)] \end{aligned}$$

The gradients of $\arg H_H(\omega)$ with respect to the coefficients are given by:

$$\begin{aligned} |H_H(\omega)|^2 \frac{\partial \arg H_H(\omega)}{\partial r'_k} &= (A_H^2(\omega) + A_H(\omega)B_H(\omega)\sin\phi_{Z_H}(M\omega)) \frac{\partial\phi_{Z_H}(2M\omega)}{\partial\omega} \\ |H_H(\omega)|^2 \frac{\partial \arg H_H(\omega)}{\partial u'_k} &= -B_H(\omega)\cos\phi_{Z_H}(M\omega) \frac{\partial A_H(\omega)}{\partial u'_k} \\ |H_H(\omega)|^2 \frac{\partial \arg H_H(\omega)}{\partial v'_k} &= A_H(\omega)\cos\phi_{Z_H}(M\omega) \frac{\partial B_H(\omega)}{\partial v'_k} \end{aligned}$$

The gradients of $T_H(\omega)$ with respect to the coefficients are given by:

$$\frac{\partial |H_H(\omega)|^2}{\partial r'_k} T_H(\omega) + |H_H(\omega)|^2 \frac{\partial T_H(\omega)}{\partial r'_k} = -A_H(\omega)B_H(\omega)\cos\phi_{Z_H}(M\omega) \frac{\partial\phi_{Z_H}(M\omega)}{\partial\omega} \frac{\partial\phi_{R_H}(2M\omega)}{\partial r'_k} \dots$$

$$\begin{aligned}
& - (A_H^2(\omega) + A_H(\omega)B_H(\omega)\sin\phi_{Z_H}(M\omega)) \frac{\partial^2\phi_{Z_H}(M\omega)}{\partial\omega\partial r'_k} \dots \\
& - \sin\phi_{Z_H}(M\omega) \left[B_H(\omega) \frac{\partial A_H(\omega)}{\partial\omega} - A_H(\omega) \frac{\partial B_H(\omega)}{\partial\omega} \right] \frac{\partial\phi_{R_H}(2M\omega)}{\partial r'_k} \\
\frac{\partial |H_H(\omega)|^2}{\partial u'_k} T_H(\omega) + |H_H(\omega)|^2 \frac{\partial T_H(\omega)}{\partial u'_k} & = - (2A_H(\omega) + B_H(\omega)\sin\phi_{Z_H}(M\omega)) \frac{\partial\phi_{Z_H}(M\omega)}{\partial\omega} \frac{\partial A_H(\omega)}{\partial u'_k} \dots \\
& + \cos\phi_{Z_H}(M\omega) \left[B_H(\omega) \frac{\partial^2 A_H(\omega)}{\partial\omega\partial u'_k} - \frac{\partial B_H(\omega)}{\partial\omega} \frac{\partial A_H(\omega)}{\partial u'_k} \right] \\
\frac{\partial |H_H(\omega)|^2}{\partial v'_k} T_H(\omega) + |H_H(\omega)|^2 \frac{\partial T_H(\omega)}{\partial v'_k} & = - A_H(\omega)\sin\phi_{Z_H}(M\omega) \frac{\partial\phi_{Z_H}(M\omega)}{\partial\omega} \frac{\partial B_H(\omega)}{\partial v'_k} \dots \\
& + \cos\phi_{Z_H}(M\omega) \left[\frac{\partial A_H(\omega)}{\partial\omega} \frac{\partial B_H(\omega)}{\partial v'_k} - A_H(\omega) \frac{\partial^2 B_H(\omega)}{\partial\omega\partial v'_k} \right]
\end{aligned}$$

where:

$$\begin{aligned}
\frac{\partial^2 A_H(\omega)}{\partial\omega\partial u'_k} &= -4(2k-d)\sin[\omega(2k-d)] \\
\frac{\partial^2 B_H(\omega)}{\partial\omega\partial v'_k} &= -4(2k-d+1)\cos[\omega(2k-d+1)]
\end{aligned}$$

The Octave script *schurOneMAPlattice_frm_hilbert_socp_slb_test.m* designs an FRM Hilbert filter with an all-pass model filter implemented as a Schur one-multiplier lattice. The filter specification is:

```

n=800 % Frequency points
tol=7.5e-05 % Tolerance on coefficient update vector
Mmodel=7 % Model filter decimation
Dmodel=9 % Desired model filter passband delay
mr=5 % Model filter order
dmask=16 % FIR masking filter delay
fap=0.01 % Amplitude pass band edge
fas=0.49 % Amplitude stop band edge
dBap=0.1 % Pass band amplitude ripple
Wap=1 % Pass band amplitude weight
ftp=0.01 % Delay pass band edge
fts=0.49 % Delay stop band edge
tp=79 % Nominal FRM filter group delay
tpr=tp/103.947 % Peak-to-peak pass band delay ripple
Wtp=0.01 % Pass band delay weight
fpp=0.01 % Phase pass band edge
fps=0.49 % Phase stop band edge
pp=-0.5*pi % Nominal passband phase (adjusted for delay)
ppr=pi/500 % Peak-to-peak pass band phase ripple
Wpp=0.1 % Pass band phase weight

```

The initial filter is based on the half-band filter designed by the Octave script *tarczynski_frm_halfband_test.m* with the WISE method of *Tarczynski et al.* as shown in Section 10.1.5. Figure 12.91 shows the response of the initial FRM Hilbert filter. SOCP and PCLS optimisation of the initial filter results in a model filter allpass filter with following lattice coefficients:

```

k2 = [ -0.5737912482, -0.1357861405, -0.0532745521, -0.0211256540, ...
       -0.0087697088 ];
epsilon2 = [ -1, 1, 1, 1, ...
             1 ];

```

and FIR masking filter polynomials:

```

u2 = [ -0.0009005864, -0.0025457761, -0.0071130803, -0.0128019220, ...
       -0.0309485917, -0.0343335606, -0.0517736811, -0.0570207655, ...
       0.4398895843 ]';

```

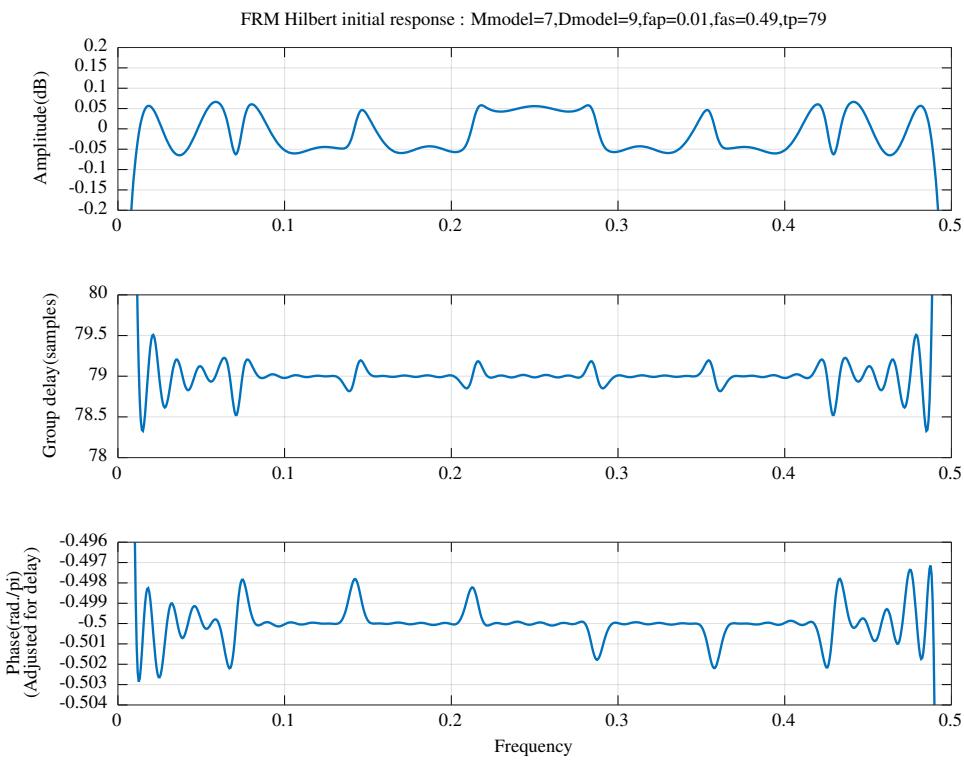


Figure 12.91: FRM Hilbert filter : initial response

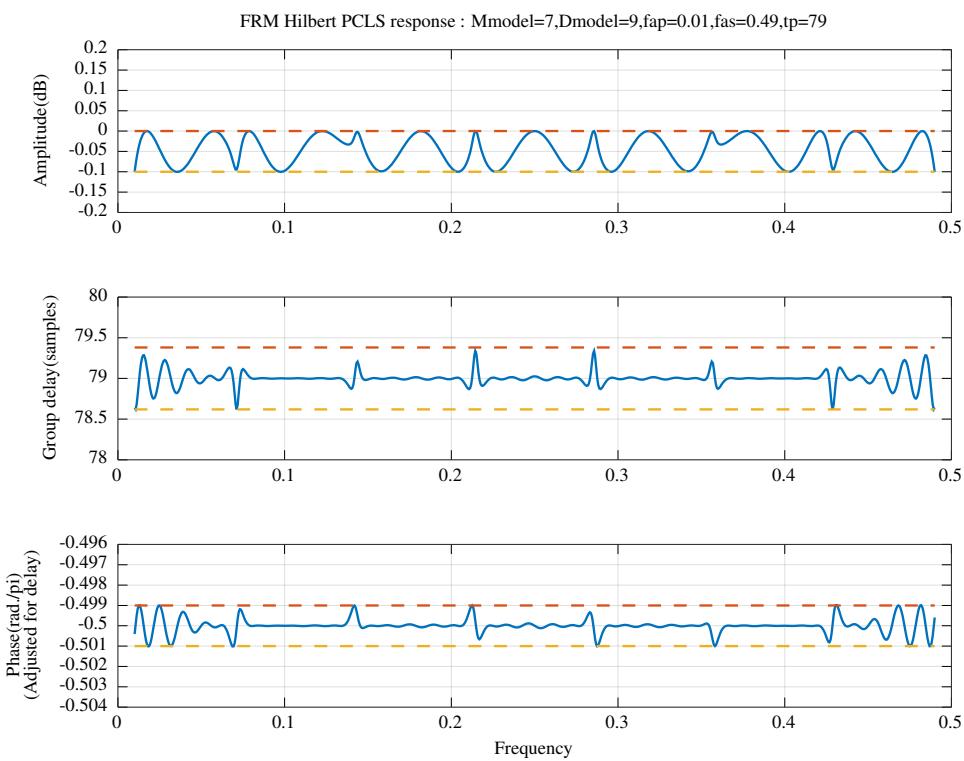


Figure 12.92: FRM Hilbert filter : response after SOCP and PCLS optimisation

```
v2 = [ 0.0065311035, 0.0043827833, 0.0072166026, 0.0020996443, ...
-0.0078831931, -0.0311746387, -0.0808425030, -0.3143749022 ]';
```

Figure 12.92 shows the response of the FRM Hilbert filter after SOCP and PCLS optimisation.

Part III

Design of IIR filters with integer coefficients

Chapter 13

Signed-digit representation of filter coefficients

Hwang [48, Section 1.5] defines *signed-digit* numbers as follows:

Given a radix r , each digit of a signed-digit number can assume the following $2\alpha + 1$ values:

$$\Sigma_r = \{-\alpha, \dots, -1, 0, 1, \dots, \alpha\}$$

where the maximum digit magnitude, α , must be within the following region:

$$\lceil \frac{r-1}{2} \rceil \leq \alpha \leq r-1$$

Because integer $\alpha \geq 1, r \geq 2$ must be assumed. In order to yield *maximum redundancy* in the balanced digit set Σ_r , one can choose the following value for the maximum magnitude

$$\alpha = \lfloor \frac{r}{2} \rfloor$$

Further:

The original motivation of using signed-digit number system is to eliminate carry propagation chains in addition (or subtraction). To break the carry chain, the lower bound on α should be made tighter as

$$\lceil \frac{r+1}{2} \rceil \leq \alpha \leq r-1$$

Parhi [49, Section 13.6] lists the following properties of the *canonical* binary signed-digit (CSD) representation of a binary signed-digit number $A = a_{W-1}a_{W-2}\cdots a_1a_0$ where each $a_i \in \{-1, 0, 1\}$:

- no 2 consecutive bits in a CSD number are non-zero
- the CSD representation of a number contains the minimum possible number of non-zero bits, thus the name *canonical*
- the CSD representation of a number is unique
- CSD numbers cover the range $(-\frac{4}{3}, \frac{4}{3})$, out of which the values in the range $[-1, 1]$ are of greatest interest
- among the W -bit CSD numbers in the range $[-1, 1]$, the average number of nonzero bits is $\frac{W}{3} + \frac{1}{9} + \mathcal{O}(2^{-W})$. Hence, on average, the CSD representation contains about two-thirds the number of non-zero bits of the equivalent two's complement representation.

Parhi [49, Section 13.6.1] shows an algorithm that calculates the *canonical* binary signed-digit representation from the two's complement representation, reproduced here as Algorithm 22.

The Octave function *bin2SPT* converts a two's complement number to the canonical signed-digit representation. The Octave function *bin2SD* approximates an *nbits* two's complement number by the *ndigits* signed-digit representation.

Algorithm 22 Conversion of 2's complement numbers to the canonical signed-digit representation (*Parhi* [49, Section 13.6.1]).

Denote the two's complement representation of the number A as $A = \hat{a}_{W-1}\hat{a}_{W-2}\cdots\hat{a}_1\hat{a}_0$.

Denote the CSD representation of A as $A = a_{W-1}a_{W-2}\cdots a_1a_0$.

```

 $\hat{a}_{-1} = 0$ 
 $\gamma_{-1} = 0$ 
 $\hat{a}_W = \hat{a}_{W-1}$ 
for  $k = 0, \dots, W-1$  do
     $\theta_i = \hat{a}_i \oplus \hat{a}_{i-1}$ 
     $\gamma_i = \bar{\gamma}_{i-1}\theta_i$ 
     $a_i = (1 - 2\hat{a}_{i+1})\gamma_i$ 
end for
```

13.1 Lim's method for allocating signed-digits to filter coefficients

Lim et al. [105] describe a method of allocating a limited number of signed power-of-two digit terms to the fixed-point coefficients of a digital filter. The method is based on the belief that “*allocating the SPT terms in such a way that all the coefficient values have the same quantization step-size to coefficient sensitivity ratio will lead to a good design*”.

Lim et al. first prove properties of the signed-digit representation [105, Section II]. In particular:

Property 1: Define S_Q as the set of contiguous integers that can be represented by up to Q signed digits. The largest integer in S_Q is $J_Q = \sum_{l=0}^{Q-1} 2^{2l+1}$.

Property 2: For $n = \sum_{l=0}^{L-1} s_l 2^l$ with $s_l \in \{-1, 0, 1\}$ it is always possible to find a representation for n such that no two consecutive signed-digits are non-zero: $s_l s_{l+1} = 0$ for all l .

Property 5: On average, $0.72Q$ signed-digits are required to represent the integers in S_Q .

Lim et al. show that an estimate of the number of signed digits, Q , required to represent J_Q is:

$$Q \approx \frac{1}{2} \log_2 J_Q + 0.31$$

Replacing J_Q by an integer $n \in S_Q$, an estimate of the average number of terms, Q_A , required to represent n is:

$$\begin{aligned} Q_A &\approx 0.72Q \\ &\approx 0.36 \log_2 n + 0.22 \end{aligned}$$

Now suppose that R signed digits are available to represent two positive integers n_1 and n_2 . If $n_1 \approx n_2$ then each integer is allocated $\frac{R}{2}$ bits. If $n_1 > n_2$ then *Lim et al.* argue that the number of additional signed-digits, Q_E , required to represent n_1 is:

$$Q_E \approx 0.36 \log_2 \left\lfloor \frac{n_1}{n_2} \right\rfloor$$

where $\lfloor x \rfloor$ represents the integer part of x . In general, Q_E is not an integer.

Lim et al. go on to consider the allocation of signed digits to the coefficients of a symmetric FIR filter. The change in the frequency response, $\Delta H(\omega)$ of a filter due to a change Δx_k in coefficient x_k is:

$$\Delta H(\omega, k) \approx \frac{\partial H(\omega)}{\partial x_k} \Delta x_k$$

Lim et al. use the average of the coefficient sensitivity to define a cost, c_k , for the k 'th coefficient:

$$c_k = 0.36 \log_2 |x_k| + 0.36 \log_2 \int_0^\pi \left| \frac{\partial H(\omega)}{\partial x_k} \right| d\omega$$

Given a total of R signed-digits, *Lim et al.* assign a single signed-digit at a time to the coefficient with the largest cost. After a coefficient is given a signed-digit, its cost is decreased by one. The process is repeated until all R digits have been allocated.

13.2 Ito's method for allocating signed-digits to filter coefficients

Ito *et al.* [80] describe a heuristic for allocating signed-digits to the coefficients of an FIR filter. Suppose $\mathbf{x} = \{x_1, \dots, x_K\}$ are the exact coefficients of the filter and that each x_k is approximated by an L signed-digit number, \hat{x}_k , and there are a total of R signed digits to be allocated:

$$\hat{x}_k = \sum_{l=1}^{n_k} b_{k,l} 2^{-q_{k,l}}$$

where

$$\begin{aligned} b_{k,l} &\in \{-1, 1\} \\ q_{k,l} &\leq L \\ R &\geq \sum_{k=1}^K n_k \end{aligned}$$

The heuristic of Ito *et al.* allocates the R available signed digits to the coefficients \mathbf{x} . $c(\mathbf{x})$ is a cost function for the filter design and \mathbf{e}_k is the unit vector with a 1 in the k 'th position and 0 elsewhere. In this case, $\lceil x_k \rceil$ is defined to be the least CSD upper bound to x_k and $\lfloor x_k \rfloor$ is defined to be the greatest CSD lower bound to x_k . I have modified the heuristic described by Ito *et al.* to that shown in Algorithm 23 by beginning with an allocation of $2N$ signed digits to each non-zero coefficient (where N is the desired average number of signed-digits per coefficient) and then iteratively removing digits from coefficients with the lowest cost. At each iteration the new cost for the coefficient is recalculated.

Algorithm 23 Modified signed-digit allocation heuristic of Ito *et al.* [80]

Initialise n_k :

```
for k = 1, ..., K do
    if |x_k| < ε then
        n_k = 0
    else
        n_k = 2N
    end if
end for
```

Allocate n_k :

```
for r = 2R, ..., R do
    for k = 1, ..., K do
        if n_k ≥ 1 then
            c_k^U = c(x + (lceil x_k - x_k) e_k)
            c_k^L = c(x - (x_k - lfloor x_k)) e_k
            c_k = min {c_k^L, c_k^U}
        end if
    end for
    c_{k_min} = min {c_k}
    n_{k_min} -= 1
end for
```

13.3 Signed-digit allocation of the coefficients of a Schur one-multiplier lattice filter

This section compares the performance of the Schur one-multiplier bandpass filter of Section 12.3.3 with coefficients that are exact, rounded, approximated by two signed-digits and approximated by an average of two signed-digits allocated by the Lim and Ito heuristics, implemented in the Octave script *schurOneMlattice_bandpass_allocsd_test.m*. That script designs a Schur IIR tapped-lattice band-pass filter for which the denominator of the transfer function has coefficients only in z^{-2} the amplitude passband is $[0.1, 0.2]$, the lower amplitude stopband is $[0, 0.05]$, the upper amplitude stopband is $[0.25, 0.5]$ and the passband group delay is $t_d = 16$ samples over $[0.09, 0.21]$. There are 31 non-zero lattice coefficients to be truncated. I assume that the internal state scaling for round-off noise reduction is approximated by bit-shifts. The heuristic of Lim *et al.* is implemented in the Octave function *schurOneMlattice_allocsd_Lim* and the heuristic of Ito *et al.* is implemented in the Octave function *schurOneMlattice_allocsd_Ito.m*. The function *schurOneMlattice_allocsd_Lim* allocates signed digits according to the gradients of the un-weighted sum of the squared-magnitude and group-delay errors. Figure 13.1 compares the cost for each allocation

method, Figure 13.2 compares the maximum stopband response in the frequency range [0.26, 0.5), Figure 13.3 compares the total number of signed-digits required to implement the coefficient multipliers for 2 signed-digits allocated to each non-zero coefficient and Figure 13.4 compares the estimated filter noise-gain using word-sizes of 6 to 16 bits. Figure 13.5 compares the responses for 10-bit coefficients and Figure 13.6 compares the passband responses for 10-bit coefficients.

Figures 13.7, 13.8, 13.9, 13.10, 13.11 and 13.12 show the corresponding results for an allocation of 3 signed-digits to each non-zero coefficient. For the heuristic of *Ito et al.*, when 3 bits are allocated to each non-zero coefficient approximately 2 signed-digits per coefficient are in fact used.

This filter has been designed with denominator polynomial coefficients only in powers of z^{-2} so that the filter can be retimed with reduced latency for the state update and filter output calculations. (As shown in Figure 7.26). The noise gain of the retimed filter is calculated by converting the lattice filter representation to state variable form with the Octave function *schurOneMR2lattice2Abcd*. The noise gain of the filter with exact coefficients is 3.44804. A fixed point implementation of the filter would either scale the states with bit-shifts (ie: by a power of two) or by adding bits to the state registers as required. In the results shown here the noise gain calculated for the retimed filter with truncated coefficients does not include the round-off noise due to state scaling.

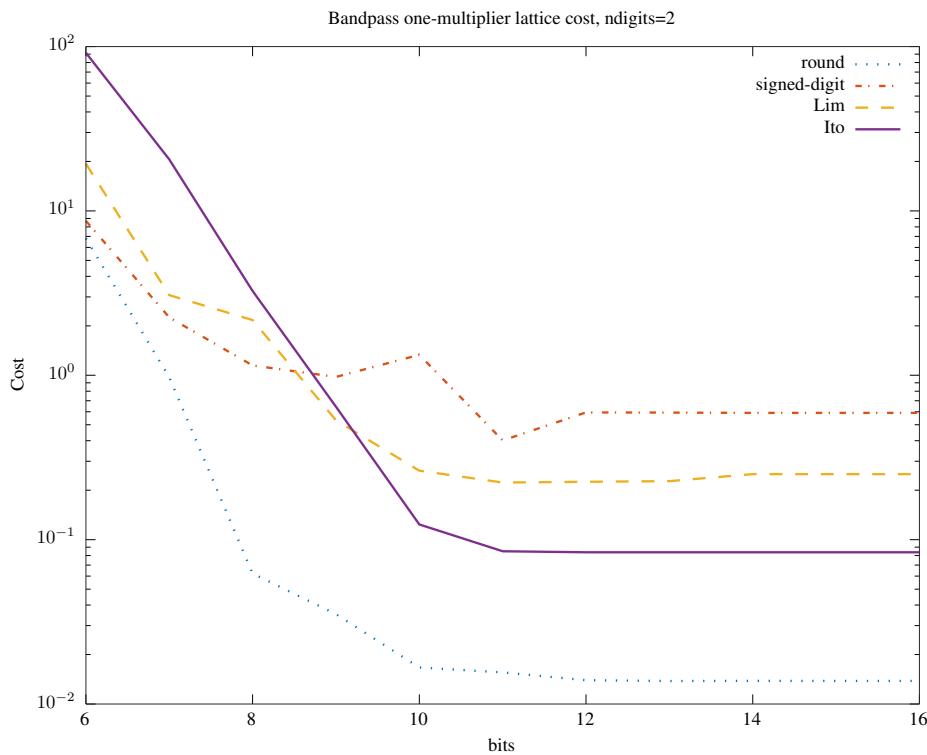


Figure 13.1: Comparison of the cost function for a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

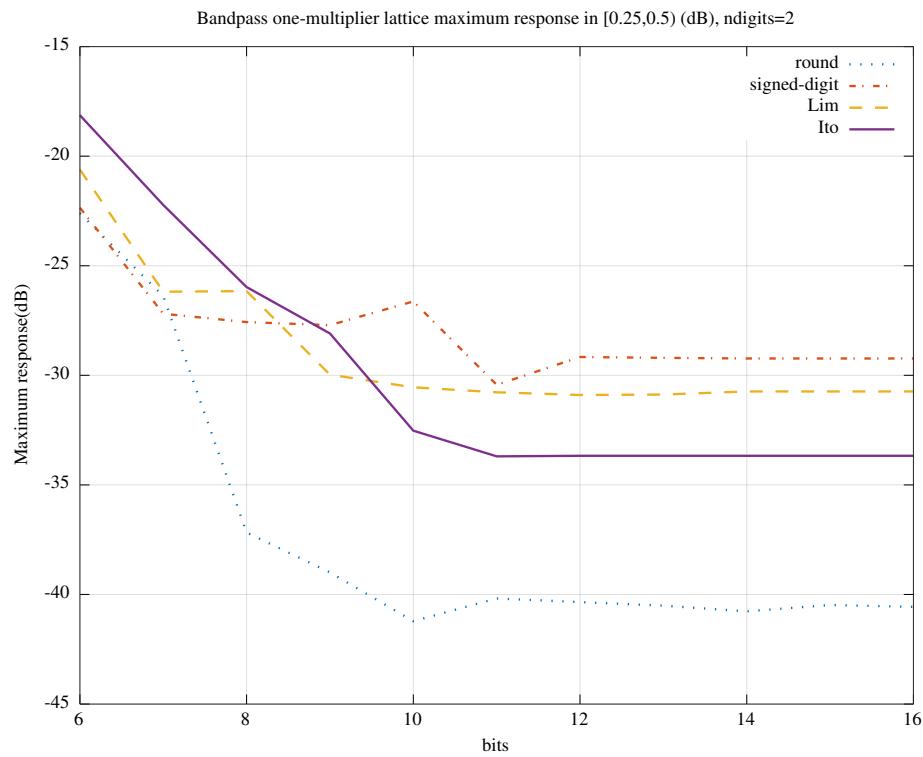


Figure 13.2: Comparison of the maximum stopband response in the region [0.26,0.5) for a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

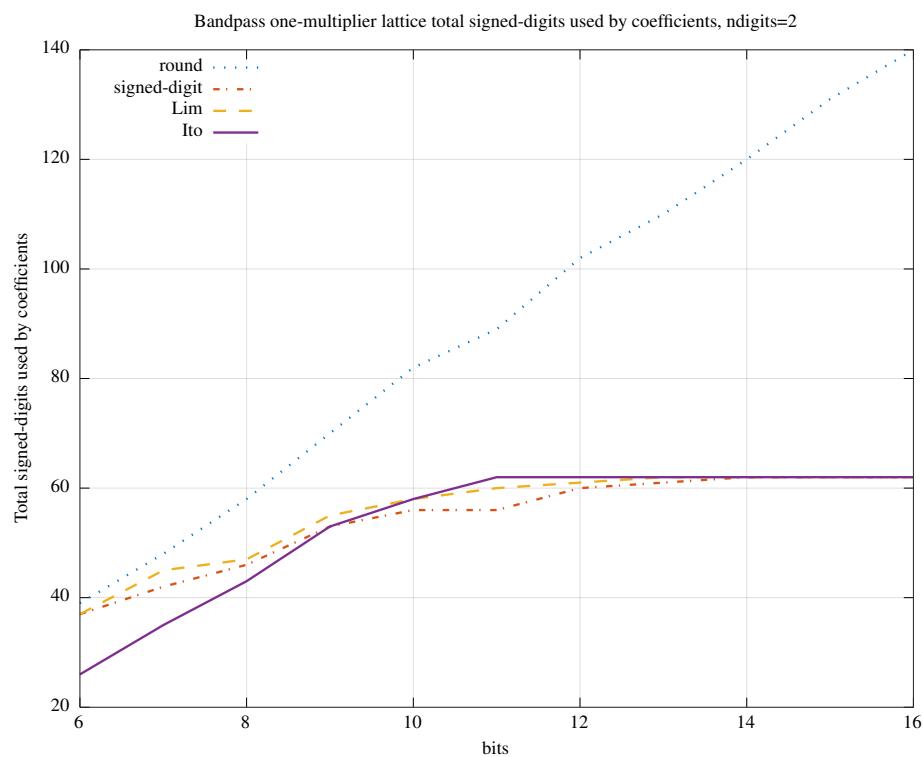


Figure 13.3: Comparison of the total number of signed-digits required to implement the coefficients of a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

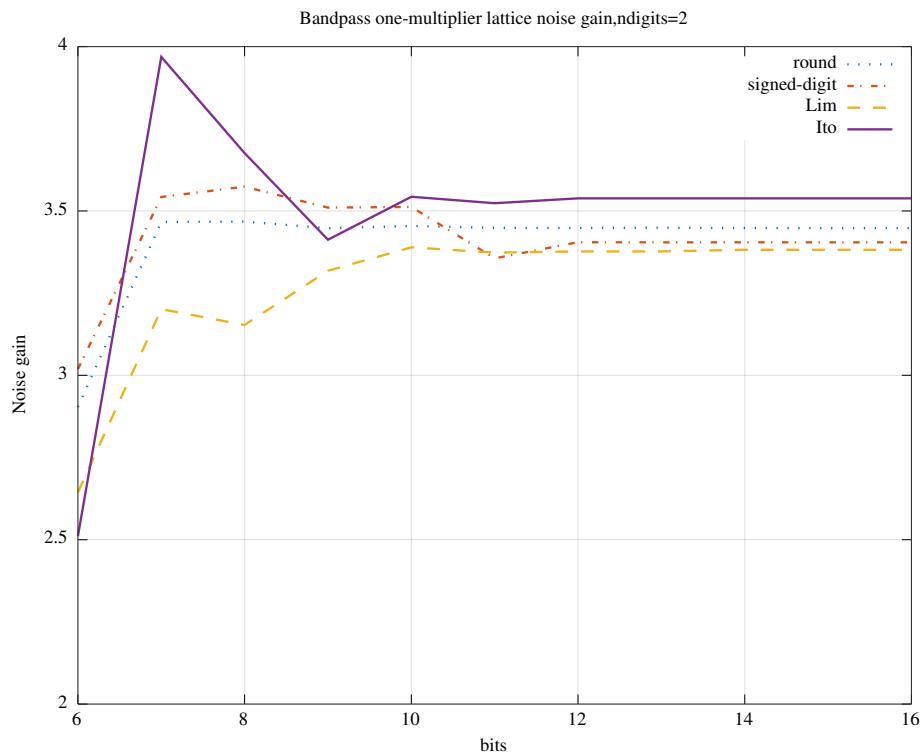


Figure 13.4: Comparison of the estimated noise gain of a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*.

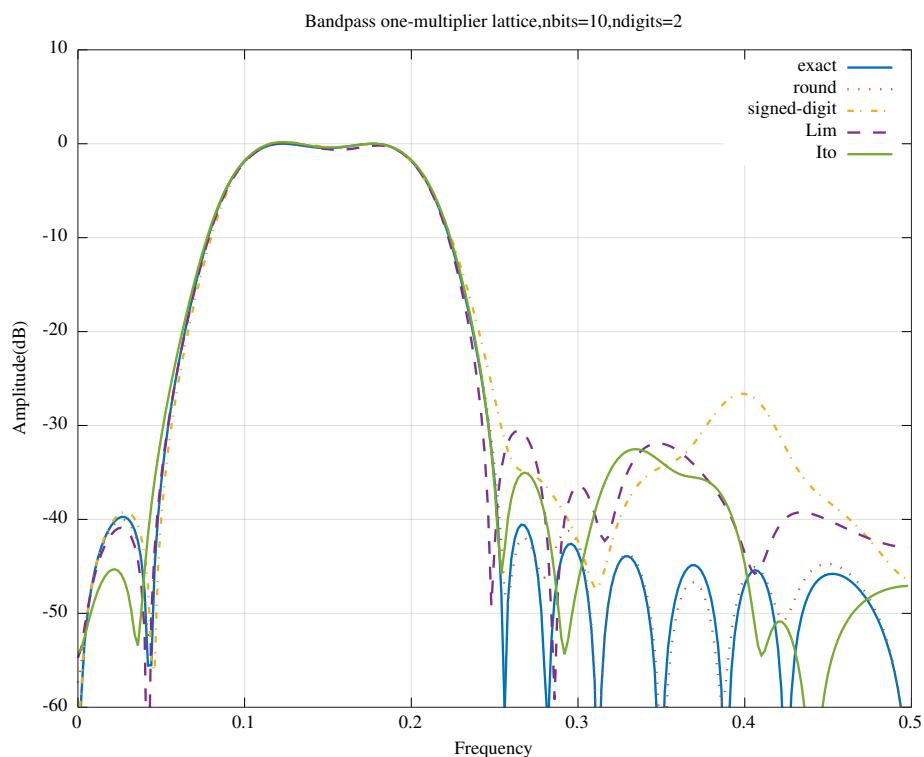


Figure 13.5: Comparison of the responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

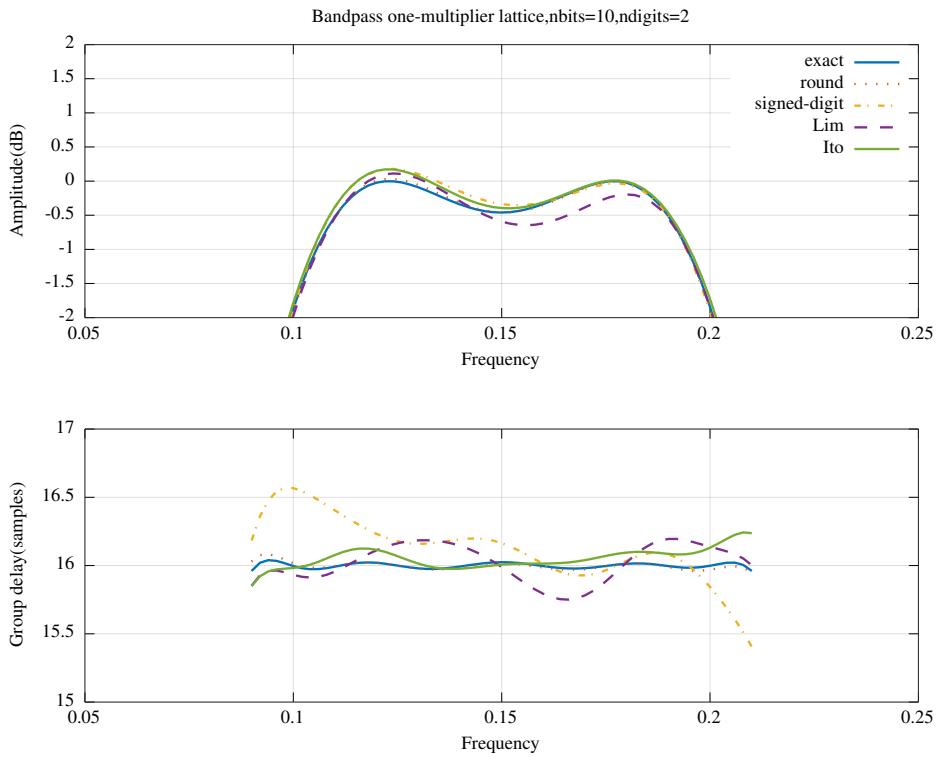


Figure 13.6: Comparison of the passband responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

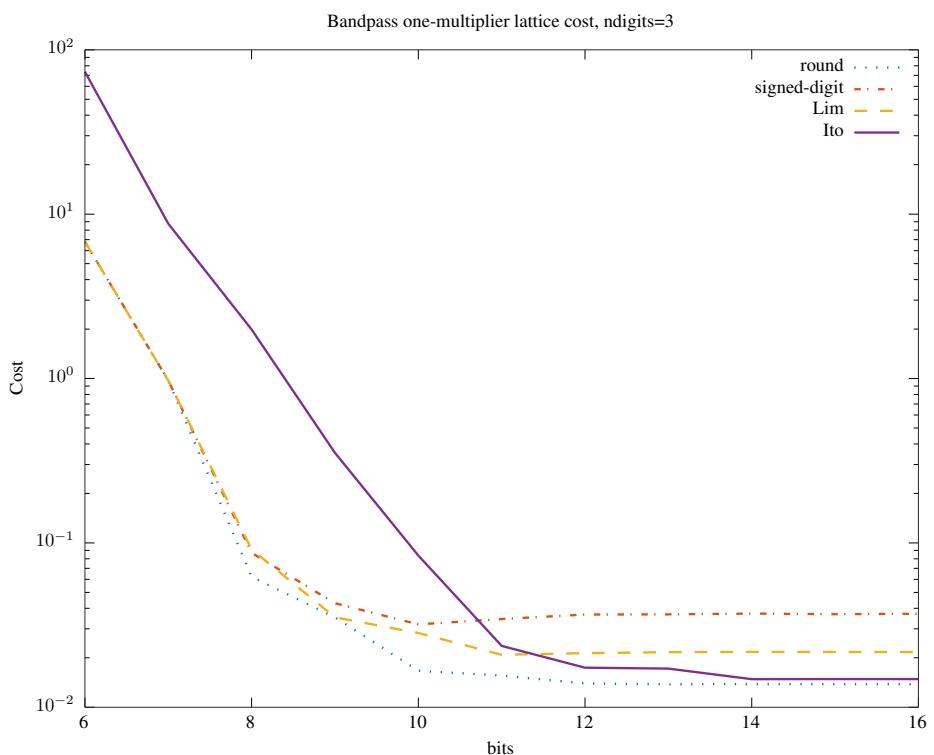


Figure 13.7: Comparison of the cost function for a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

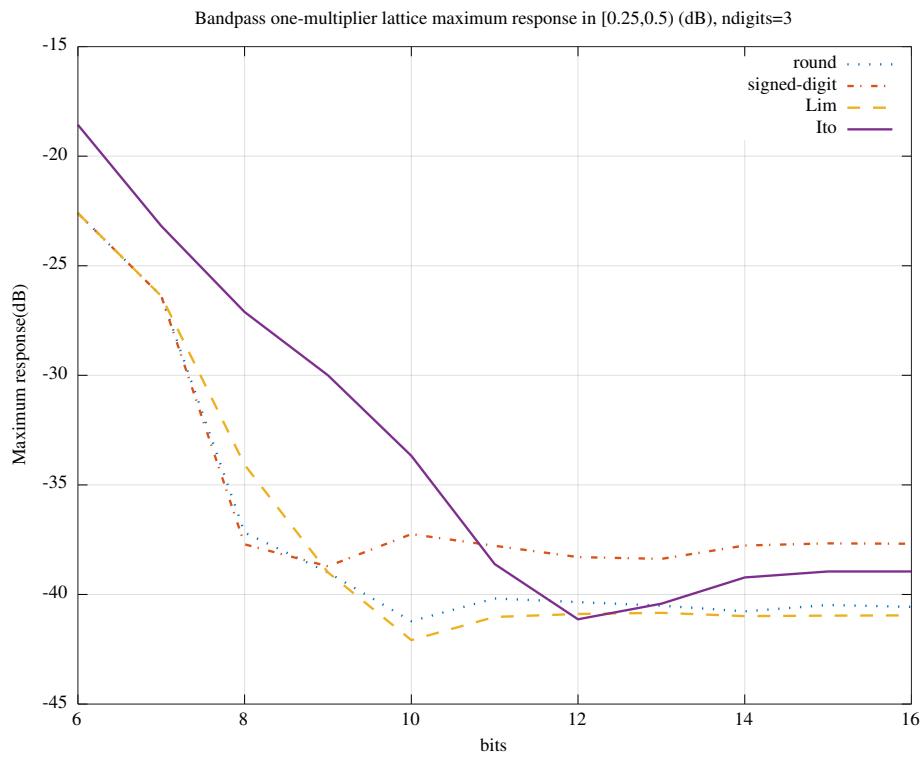


Figure 13.8: Comparison of the maximum stopband response in the region [0.26,0.5) for a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

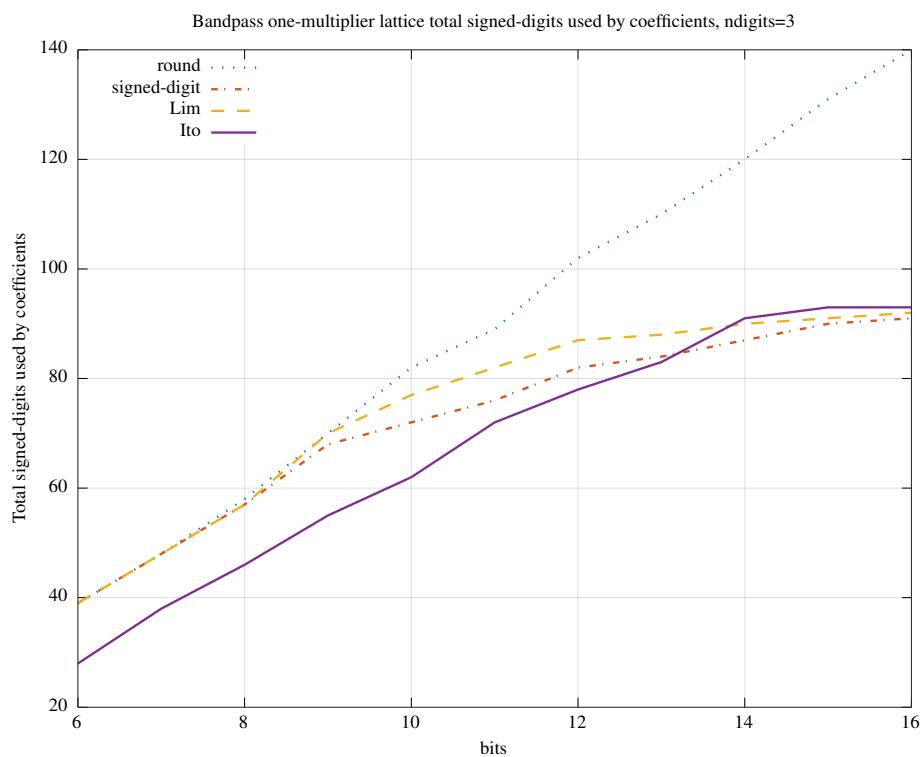


Figure 13.9: Comparison of the total number of signed-digits required to implement the coefficients of a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

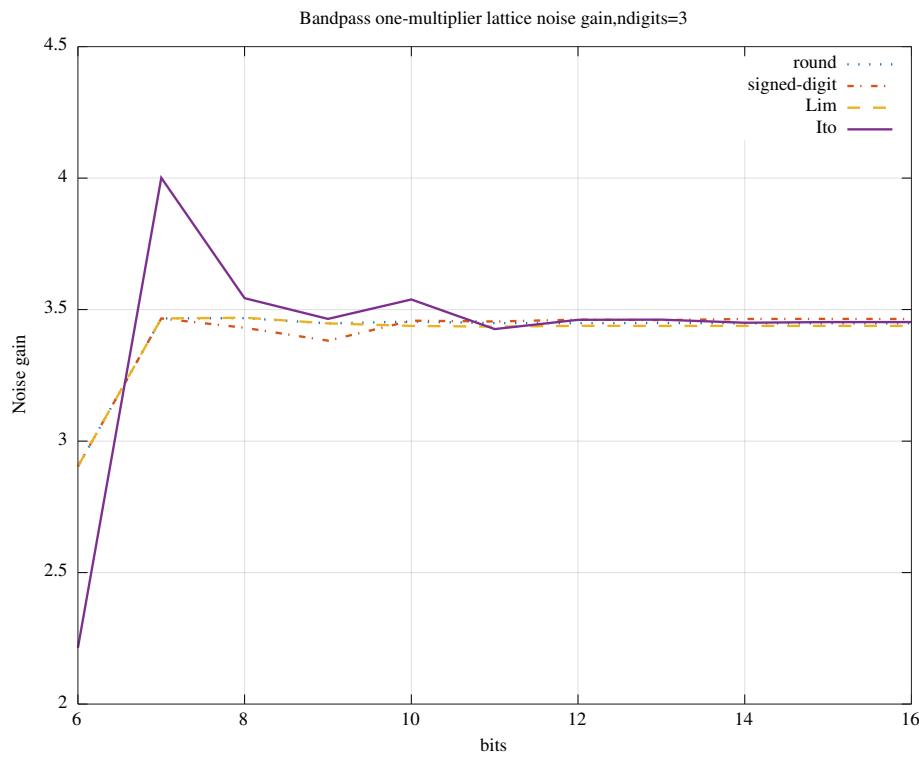


Figure 13.10: Comparison of the estimated noise gain of a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*.

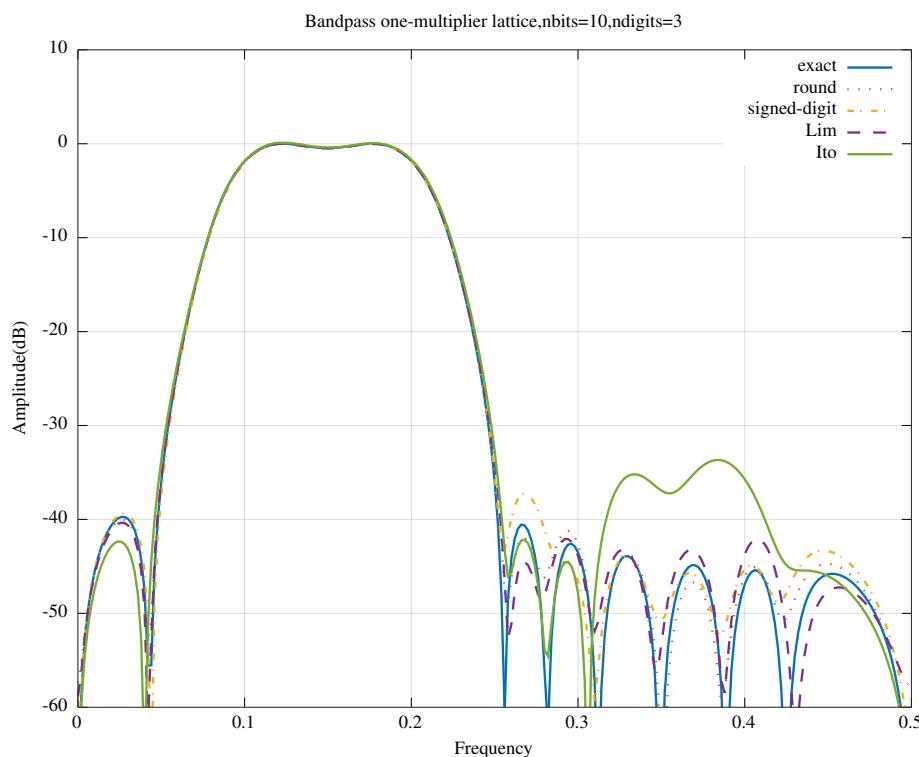


Figure 13.11: Comparison of the responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

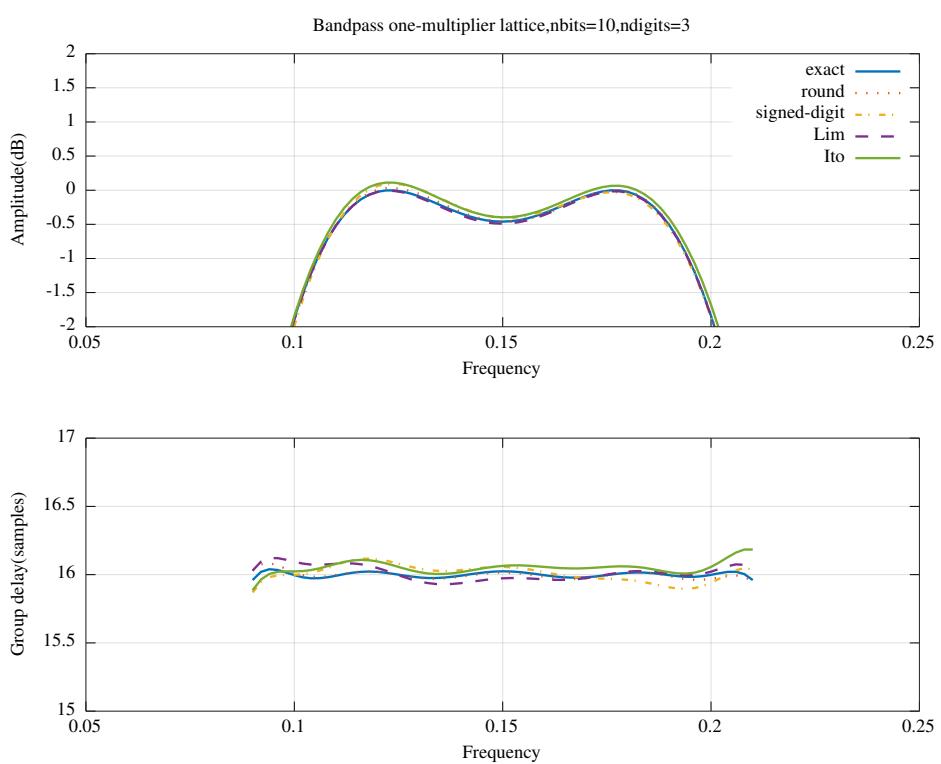


Figure 13.12: Comparison of the passband responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

Chapter 14

Searching for integer and signed-digit filter coefficients

14.1 Exhaustive search for signed-digit filter coefficients

The Octave script *exhaustive_bandpass_OneM_lattice_test.m* implements an exhaustive search for the best signed-digit approximation to the exact coefficients of the Schur one-multiplier lattice band-pass filter designed in Section 12.3.3. The cost function for the search calculates the weighted root-squared-error of the amplitude and group-delay responses of the filter when compared to an ideal “brick-wall” response. The band-pass filter has 31 non-zero coefficients and each coefficient is selected from an upper and lower bound on the exact value so the exhaustive search will perform 2^{31} evaluations of the cost function. Preliminary experiments showed that this script would require about month of processing time on my PC (which has an Intel i7-7700K CPU running at 4.2GHz with 4 cores and 2 threads-per-core).

14.2 Bit-flipping search for integer and signed-digit filter coefficients

Krukowski and Kale [6], describe a “bit-flipping” algorithm for fixed-point filter design, shown in flow-graph form in Figure 14.1 (from [6, Figure 2]).

The bit-flipping algorithm searches for an improvement in a cost-function by testing each combination of bits within a window for each coefficient. When no further improvement is found the window is shifted toward the least-significant-bits and the search is repeated. The bit-flipping algorithm is implemented in the Octave function *bitflip*.

14.2.1 Bit-flipping algorithm examples

In the following examples I apply the bit-flipping algoritm to direct-form, Schur normalised-scaled lattice and Schur one-multiplier lattice IIR filter implementations. The initial filter is that designed by the Octave script *iir_sqp_slb_bandpass_test.m* with the following filter specification:

```
R=2 % Denominator coefficients only in z^2
fapl=0.1 % Amplitude passband lower edge (fs=1)
faph=0.2 % Amplitude passband upper edge
dBap=1 % Amplitude passband ripple (dB)
ftpl=0.09 % Group delay passband lower edge
ftph=0.21 % Group delay passband upper edge
tp=16 % Passband group delay (samples)
tpr=0.08 % Passband group delay ripple
fasl=0.05 % Amplitude lower stopband upper edge
fasl=0.25 % Amplitude upper stopband lower edge
dBas=36 % Amplitude stopband attenuation
```

and with the following exact filter transfer function polynomials:

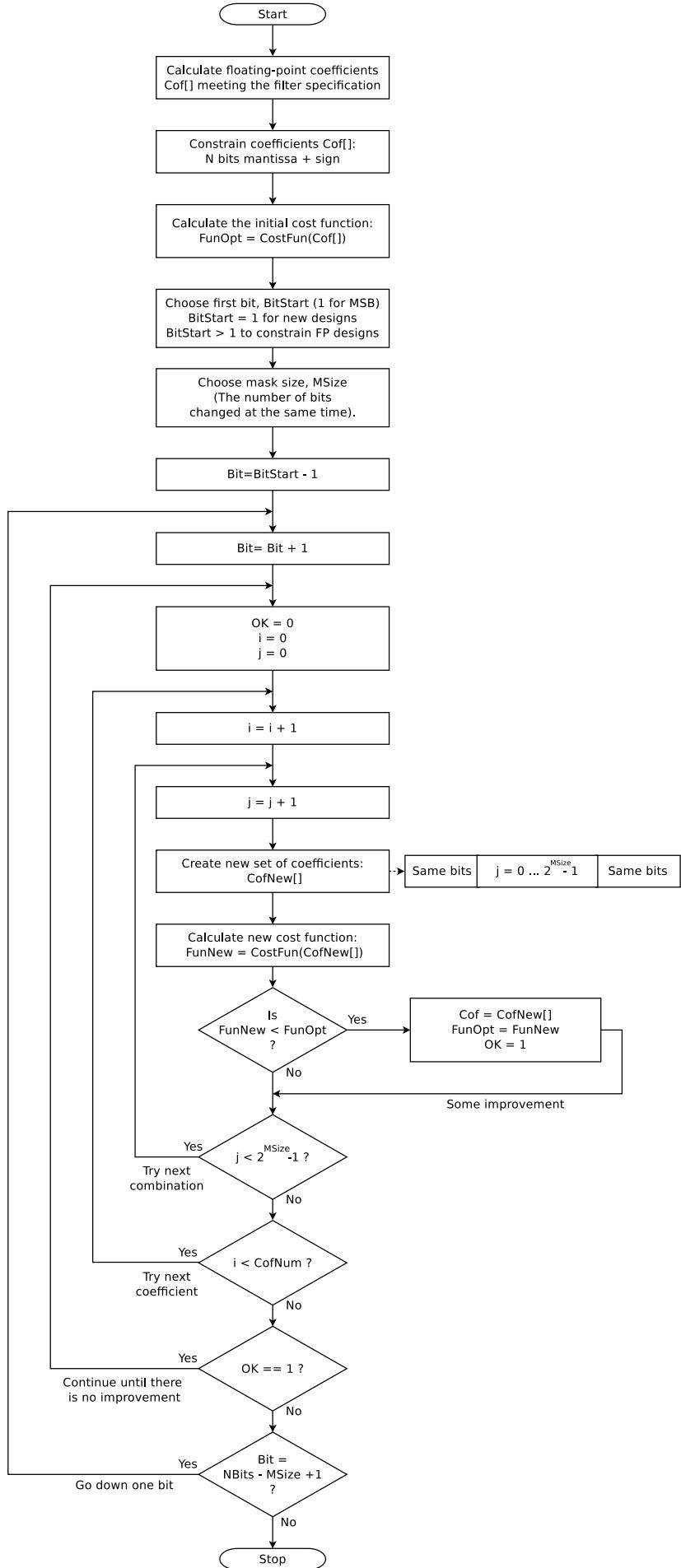


Figure 14.1: The bit-flipping algorithm. (*Krukowski and Kale* [6, Figure 2]).

```

n0 = [ 0.0121797025, 0.0037181466, 0.0271208444, 0.0229180899, ...
        0.0527427390, 0.0310257396, 0.0310148929, -0.0050816528, ...
    -0.0052190028, -0.0414041061, -0.0726994834, -0.0996450244, ...
    -0.0564833866, 0.0506036935, 0.1386006506, 0.1492761564, ...
    0.0507378434, -0.0443572576, -0.1001730674, -0.0681538294, ...
    -0.0335038915 ]';

d0 = [ 1.0000000000, 0.0000000000, 1.8567605231, 0.0000000000, ...
        2.1933886439, 0.0000000000, 2.2557980857, 0.0000000000, ...
        2.0335494061, 0.0000000000, 1.5306996820, 0.0000000000, ...
        0.9936532921, 0.0000000000, 0.5470027574, 0.0000000000, ...
        0.2511015052, 0.0000000000, 0.0839735161, 0.0000000000, ...
        0.0183734564 ]';

```

For each example a cost function compares an ideal, “brick-wall” response with the responses for the exact filter coefficients and the filter coefficients approximated by 6-bit rounded, 6-bit rounding optimised with the bit-flipping algorithm, 6-bit 3 signed-digits and 6-bit 3 signed-digits optimised with the bit-flipping algorithm. The cost function weights the stop-band amplitude error by $W_{asl} = W_{asu} = 30$. Bit-flipping starts at bit 4 of 6 and uses a mask size of 3 bits.

For comparison, I also apply the bit-flipping algorithm to a minimum-phase Schur FIR lattice having $dBap = 3$ and $dBas = 25$.

Truncation of the coefficients of a direct-form bandpass IIR filter with the bit-flipping algorithm

The Octave script *bitflip_bandpass_direct_test.m* applies the bit-flipping algorithm to a direct-form bandpass filter. There is no attempt to ensure that the bit-flipped filter transfer function is stable. The filter responses are shown in Figure 14.2.

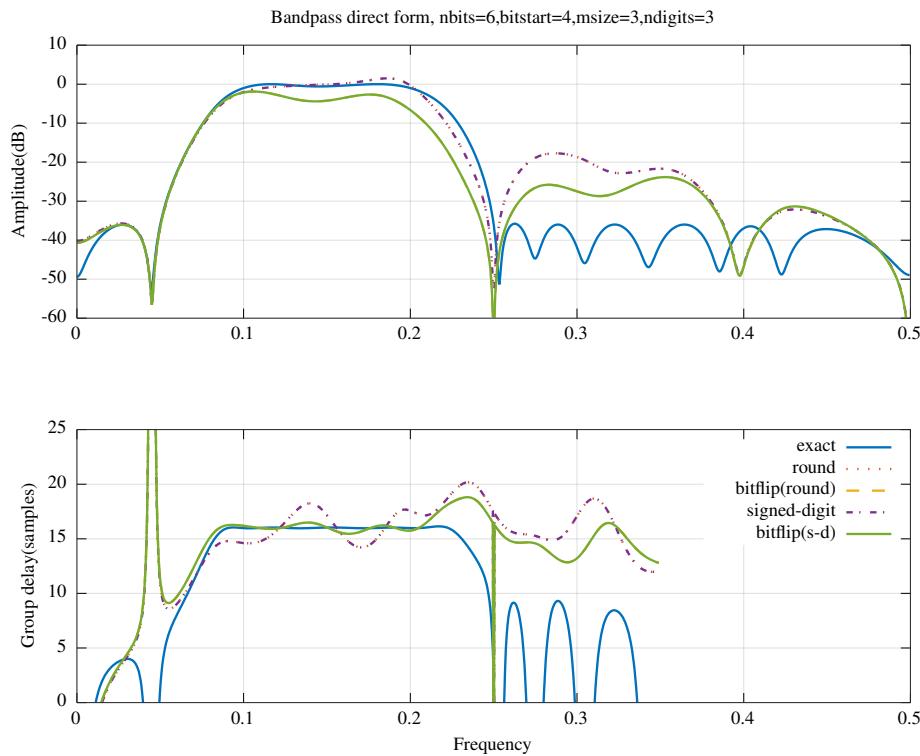


Figure 14.2: Amplitude and group-delay responses of a direct-form band-pass filter with exact coefficients, 6-bit rounded coefficients, 6-bit rounded coefficients optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients and 6-bit 3 signed-digit coefficients optimised with the bit-flipping algorithm

Table 14.1 compares the cost result for each test.

The 6-bit 3 signed-digit direct-form transfer function coefficients found by the bit-flipping algorithm are:

Bandpass direct form	Cost
Exact	1.0803
6-bit rounded	14.5007
6-bit rounded with bitflipping	5.3815
6-bit 3-signed-digit	14.5007
6-bit 3-signed-digit with bitflipping	5.3815

Table 14.1: Summary of the cost results for the direct form bandpass filter with exact coefficients, 6-bit rounded coefficients, 6-bit rounded coefficients optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients and 6-bit 3 signed-digit coefficients optimised with the bit-flipping algorithm

```
n_bfsd = [ 0.00000, 0.00000, 0.03125, 0.03125, ...
0.06250, 0.03125, 0.03125, 0.00000, ...
0.00000, -0.03125, -0.06250, -0.09375, ...
-0.06250, 0.06250, 0.12500, 0.15625, ...
0.06250, -0.03125, -0.09375, -0.06250, ...
-0.03125 ];

d_bfsd = [ 1.37500, 0.03125, 1.93750, 0.06250, ...
2.25000, 0.00000, 2.25000, 0.06250, ...
2.00000, 0.06250, 1.50000, 0.00000, ...
1.00000, 0.00000, 0.56250, 0.00000, ...
0.28125, 0.00000, 0.09375, 0.00000, ...
0.03125 ];
```

A total of 12 adders is required to implement these signed-digit multipliers.

Truncation of the coefficients of a normalised-scaled Schur lattice bandpass IIR filter with the bit-flipping algorithm

The Octave script *bitflip_bandpass_NS_lattice_test.m* applies the bit-flipping algorithm to the bandpass filter implemented as a Schur normalised-scaled lattice filter. The filter responses are shown in Figure 14.3.

Table 14.2 compares the cost result for each test.

Bandpass Schur normalised-scaled	Cost
Exact	1.0803
6-bit rounded	3.9584
6-bit rounded with bitflipping	2.6834
6-bit 3-signed-digit	3.9584
6-bit 3-signed-digit with bitflipping	2.1823

Table 14.2: Summary of the cost results for the bandpass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6-bit rounded coefficients, 6-bit rounded coefficients optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients and 6-bit 3 signed-digit coefficients optimised with the bit-flipping algorithm

The 6-bit 3 signed-digit lattice filter coefficients are:

```
s10_bfsd = [ 0.37500, -0.71875, -0.87500, -0.62500, ...
0.03125, 0.50000, 0.46875, 0.15625, ...
-0.09375, -0.09375, 0.00000, 0.00000, ...
-0.06250, -0.09375, 0.00000, 0.03125, ...
0.03125, 0.00000, 0.00000, 0.00000 ]';

s11_bfsd = [ 0.93750, 0.68750, 0.46875, 0.78125, ...
1.00000, 0.87500, 0.87500, 1.06250, ...
1.00000, 1.00000, 1.06250, 1.00000, ...
1.00000, 1.00000, 1.00000, 1.00000, ...
1.00000, 1.00000, 1.00000, 0.43750 ]';
```

```

s20_bfsd = [ 0.00000, 0.62500, 0.00000, 0.53125, ...
0.00000, 0.37500, 0.00000, 0.31250, ...
0.00000, 0.15625, 0.00000, 0.43750, ...
0.00000, 0.46875, 0.00000, 0.00000, ...
0.00000, 0.00000, 0.00000, 0.03125 ]';

s00_bfsd = [ 1.00000, 0.62500, 1.00000, 0.87500, ...
1.00000, 0.93750, 1.00000, 0.90625, ...
1.00000, 0.93750, 1.00000, 0.96875, ...
1.00000, 0.96875, 1.00000, 1.00000, ...
1.00000, 1.00000, 1.00000, 1.00000 ]';

s02_bfsd = [ 0.00000, -0.78125, 0.00000, -0.53125, ...
0.00000, -0.37500, 0.00000, -0.40625, ...
0.00000, -0.34375, 0.00000, -0.28125, ...
0.00000, -0.18750, 0.00000, -0.12500, ...
0.00000, -0.06250, 0.00000, -0.03125 ]';

s22_bfsd = [ 1.00000, 0.53125, 1.00000, 0.75000, ...
1.00000, 0.90625, 1.00000, 0.93750, ...
1.00000, 0.84375, 1.00000, 0.96875, ...
1.00000, 0.96875, 1.00000, 1.00000, ...
1.00000, 1.00000, 1.00000, 1.00000 ]';

```

A total of 55 adders is required to implement these signed-digit multipliers.

Truncation of the coefficients of a one-multiplier Schur lattice bandpass IIR filter with the bit-flipping algorithm

The Octave script *bitflip_bandpass_OneM_lattice_test.m* applies the bit-flipping algorithm to the bandpass filter implemented as a Schur one-multiplier lattice filter. Note that, in this example, the one-multiplier state scaling coefficients are not truncated.

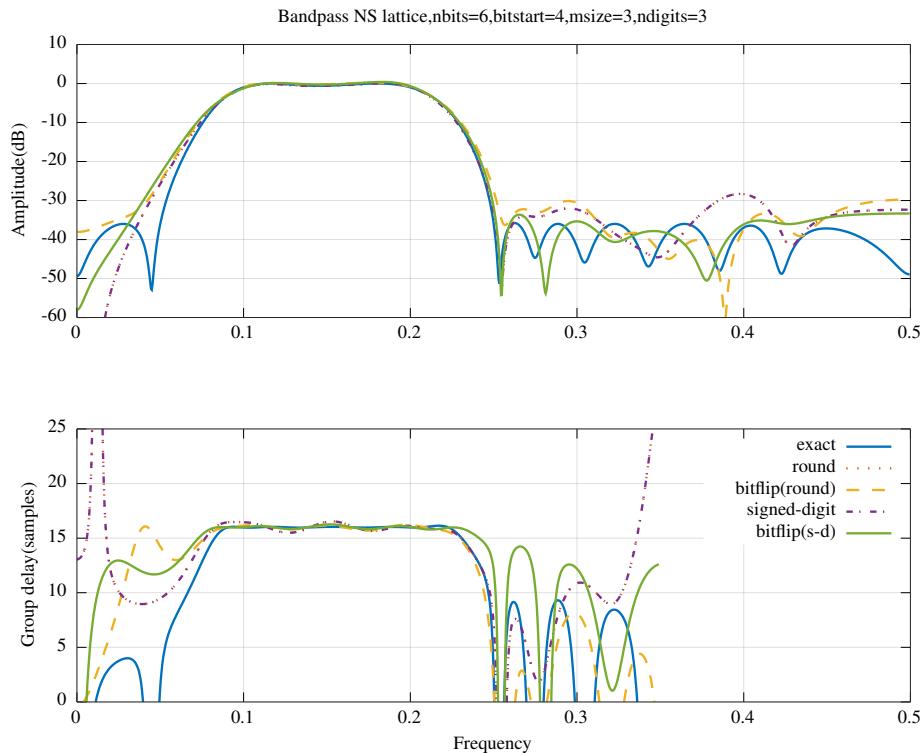


Figure 14.3: Amplitude and group-delay responses of a band-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6-bit rounded coefficients, 6-bit rounded coefficients optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients and 6-bit 3 signed-digit coefficients optimised with the bit-flipping algorithm

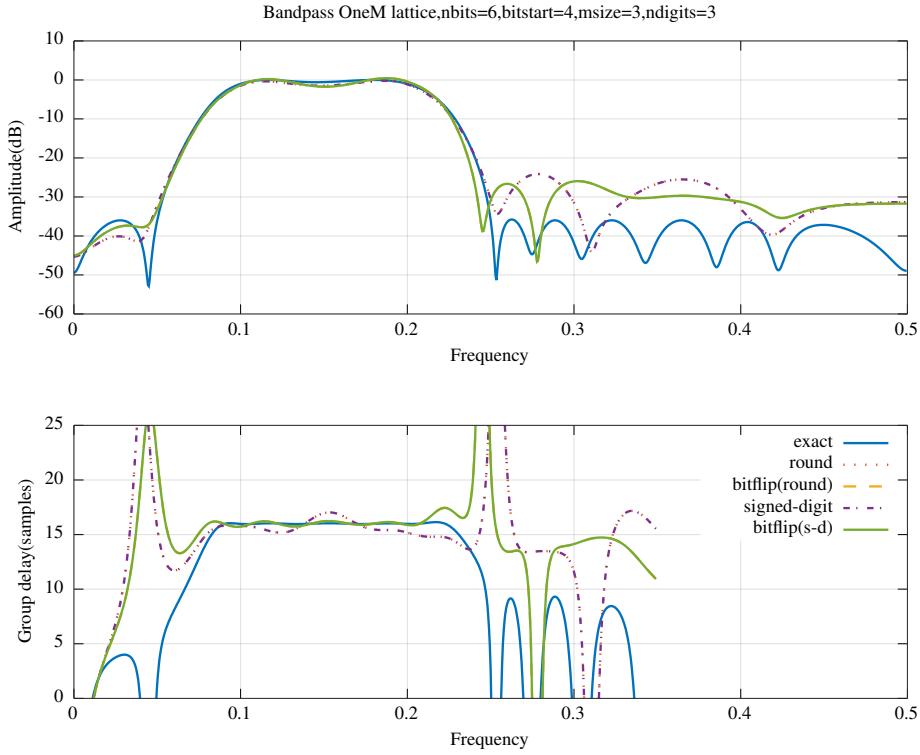


Figure 14.4: Amplitude and group-delay responses of a band-pass filter synthesised as a one-multiplier lattice filter with exact coefficients, 6-bit rounded coefficients, 6-bit rounded coefficients optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients and 6-bit 3 signed-digit coefficients optimised with the bit-flipping algorithm

The filter responses are shown in Figure 14.4.

The filter responses for the signed-digit coefficients with 3 signed-digits, 3 signed-digits allocated with Lim's algorithm and 3 signed-digits allocated with Ito's algorithm are shown in Figure 14.5.

Table 14.3 compares the cost result for each test.

Bandpass Schur one-multiplier	Cost
Exact	1.0803
6-bit rounded	7.2092
6-bit rounded with bit-flipping	3.2642
6-bit 3-signed-digit	7.2092
6-bit 3-signed-digit with bit-flipping	3.2642
6-bit 3-signed-digit(Lim alloc.)	9.2479
6-bit 3-signed-digit(Lim alloc.) with bit-flipping	4.1526
6-bit 3-signed-digit(Ito alloc.)	10.9197
6-bit 3-signed-digit(Ito alloc.) with bit-flipping	7.5414

Table 14.3: Summary of the cost results for the bandpass filter synthesised as a one-multiplier lattice filter with exact coefficients, 6-bit rounded coefficients, 6-bit rounded coefficients optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients and 6-bit 3 signed-digit coefficients optimised with the bit-flipping algorithm. The signed digits are allocated with 3 digits each, Lim's allocation method with an average of 3 signed-digits each and Ito's allocation method with an average of 3 signed-digits each.

The 6-bit 3 signed-digits (allocated with Lim's algorithm) one-multiplier lattice filter coefficients are:

```
k_bfsdl = [ 0.00000,  0.81250,  0.00000,  0.56250, ...
            0.00000,  0.31250,  0.00000,  0.43750, ...
            0.00000,  0.34375,  0.00000,  0.28125, ...]
```

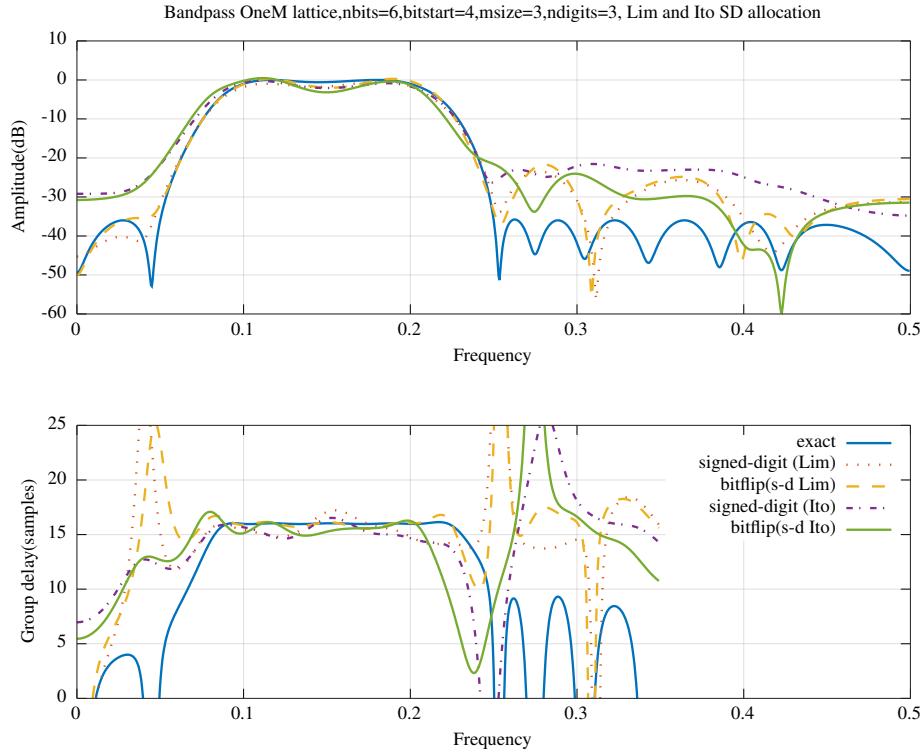


Figure 14.5: Amplitude and group-delay responses of a band-pass filter synthesised as a one-multiplier lattice filter with exact coefficients, 6-bit 3 signed-digit coefficients allocated with Lim's algorithm, 6-bit 3 signed-digit coefficients allocated with Lim's algorithm and optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients allocated with Ito's algorithm and 6-bit 3 signed-digit coefficients allocated with Ito's algorithm and optimised with the bit-flipping algorithm

```
0.00000, 0.18750, 0.00000, 0.15625, ...
0.00000, 0.06250, 0.00000, 0.03125 ];
```

```
c_bfsdl = [ 0.06250, 0.03125, -0.25000, -0.65625, ...
-0.31250, 0.03125, 0.21875, 0.21875, ...
0.09375, -0.06250, -0.06250, 0.00000, ...
0.00000, -0.06250, -0.06250, 0.00000, ...
0.03125, 0.03125, 0.00000, 0.00000, ...
0.00000 ];
```

A total of 16 adders is required to implement these signed-digit multipliers.

Truncation of the coefficients of a minimum-phase bandpass FIR filter with the bit-flipping algorithm

Section 10.2.11 shows the design of a minimum-phase FIR bandpass filter with an amplitude response that is similar to the example above. A minimum-phase FIR filter has all zeros within the unit circle so that the filter polynomial has a Schur decomposition. The filter is not linear phase and does not have a flat group delay response. The Schur FIR lattice is assumed to have lesser coefficient sensitivity than the direct form filter [70]. The FIR Schur lattice has two real multipliers for each reflection coefficient [106].

The Octave script, *bitflip_bandpass_schur_FIR_lattice_test.m* implements the band-pass filter as a minimum-phase Schur FIR lattice filter. In contrast to the previous examples in this chapter, the cost function does not include the group-delay error. As for the Schur one-multiplier lattice IIR filter example, the FIR state scaling coefficients are not truncated. The initial bandpass Schur lattice FIR filter polynomial is that calculated by the Octave script *iir_sqp_slb_fir_17_bandpass_test.m*:

```
b0 = [ 0.0920209477, 0.1200456229, 0.0238698802, -0.1608522791, ...
-0.2457451579, -0.1127458202, 0.1191739637, 0.2218613695, ...
0.1246356125, -0.0268745986, -0.0766798541, -0.0325005729, ...
0.0009004091, -0.0151564755, -0.0332760730, -0.0216108491, ...
0.0235001427 ]';
```

The Schur FIR lattice multiplier coefficients of the initial filter are

```
k_ex = [ 0.4593466811, 0.0207755511, -0.4543038955, -0.5752634543, ...
-0.3661751825, 0.1513717747, 0.4876634116, 0.4458838158, ...
0.0642635216, -0.3825130725, -0.5970105299, -0.7283202009, ...
-0.8093612951, 0.6805119912, -0.6076281309, 0.2553781860 ];
```

The initial FIR filter has order 17, 16 lattice coefficents and 32 multiplies per sample which is similar to the one-multiplier Schur lattice bandpass filter of the previous example.

The initial filter response and the filter responses after coefficient truncation are shown in Figure 14.6.

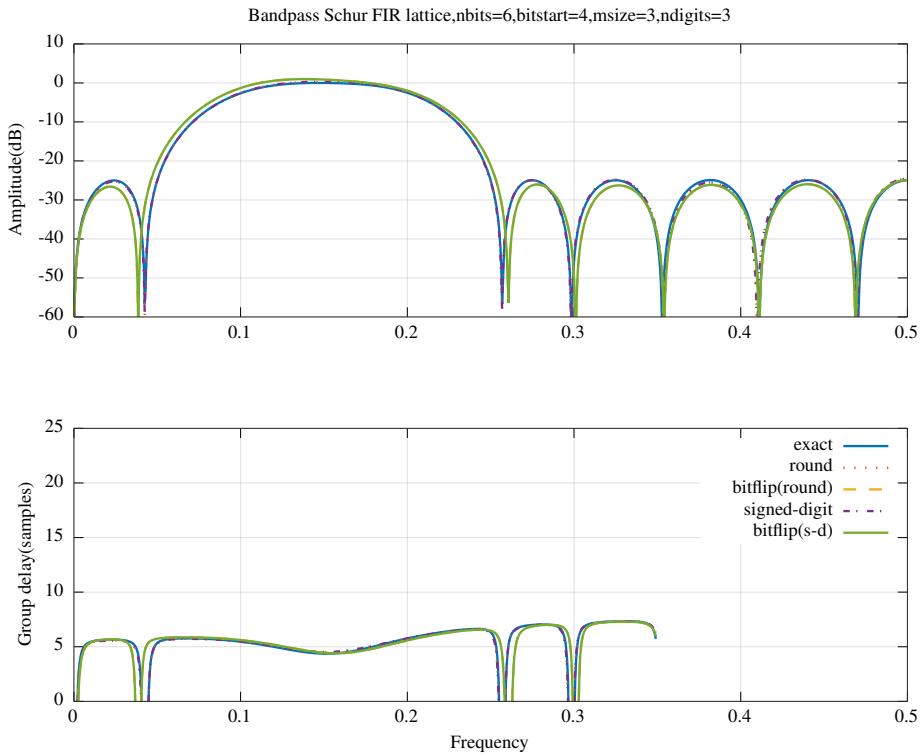


Figure 14.6: Amplitude and group-delay responses of a minimum-phase band-pass filter synthesised as a Schur FIR lattice filter with exact coefficients, 6-bit rounded coefficients, 6-bit rounded coefficients optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients and 6-bit 3 signed-digit coefficients optimised with the bit-flipping algorithm

Table 14.4 compares the cost results for each truncation method.

Bandpass Schur FIR	Cost
Exact	2.7927
6-bit rounded	2.7629
6-bit rounded with bitflipping	2.5878
6-bit 3-signed-digit	2.7629
6-bit 3-signed-digit with bitflipping	2.5878

Table 14.4: Summary of the cost results for the FIR minimum-phase bandpass filter synthesised as a Schur FIR lattice filter with exact coefficients, 6-bit rounded coefficients, 6-bit rounded coefficients optimised with the bit-flipping algorithm, 6-bit 3 signed-digit coefficients and 6-bit 3 signed-digit coefficients optimised with the bit-flipping algorithm

The 6-bit 3 signed-digit Schur FIR lattice coefficients found by the bit-flipping algorithm are:

```
k_bfsd = [ 0.46875,  0.06250, -0.46875, -0.56250, ...
           -0.37500,  0.34375,  0.56250,  0.43750, ...
           0.12500, -0.37500, -0.59375, -0.71875, ...
           -0.81250,  0.65625, -0.59375,  0.25000 ];
```

A total of 38 adders is required to implement these signed-digit multipliers.

14.3 Branch-and-bound search for signed-digit coefficients

Given the K coefficients, x_k , of a filter, an exhaustive search of the upper and lower bounds on the integer or signed-digit approximations to these coefficients would require $\mathcal{O}(2^K)$ comparisons. *Branch-and-bound* [5], [78, p.627] is a heuristic for reducing the number of branches searched in a binary decision tree. At each branch of the binary tree the solution is compared to the estimated lower bounds on the cost of the full path proceeding from that branch. If the cost of that full path is greater than that of the best full path found so far then further search on that path is abandoned. Figure 14.7 shows a flow diagram of an implementation of the algorithm using a stack. The exact filter coefficients x are approximated by the signed-digit coefficients \bar{x} . Each coefficient, x_k and \bar{x}_k is bounded by the corresponding signed-digit numbers u_k and l_k so that $l_k \leq x_k \leq u_k$ and $l_k \leq \bar{x}_k \leq u_k$. The search for the set of coefficients with minimum cost is “depth-first”, starting at the root of the search tree and fixing successive coefficients. *Ito et al.* [80] recommend choosing, at each branch, the x_k with the greatest difference $u_k - l_k$. The two sub-problems at that branch fix x_k to l_k and u_k . One of the two sub-problems is pushed onto a stack and the other is solved and the cost calculated. I assume that this cost is the least possible for the remaining coefficients on the current branch. If the cost of the current sub-problem is greater than the current minimum cost then the current branch is abandoned and a new sub-problem is popped off the problem stack. Otherwise, if search has reached the maximum depth of the tree then the current solution is a new signed-digit minimum cost solution. If the current cost is less than the minimum cost and the search has not reached the maximum depth then the search continues with a new branch.

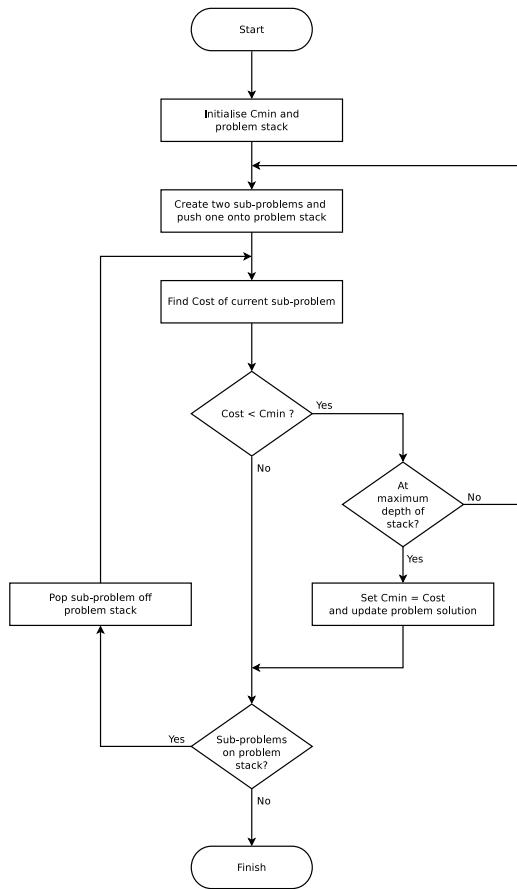


Figure 14.7: The branch-and-bound tree search algorithm.

The following sections show the results of experiments with branch-and-bound search for the signed-digit coefficients of a Schur one-multiplier lattice filter based on that found in Section 12.3.3. The cost function used is *schurOneMlatticeEsq*.

14.3.1 Branch-and-bound search for the 6 bit 3 signed-digit coefficients of a one-multiplier Schur lattice filter

The Octave script *branch_bound_bandpass_OneM_lattice_6_nbts_test.m* uses the branch-and-bound heuristic to optimise the response of the band-pass Schur one-multiplier lattice filter of Section 12.3.3 with coefficients truncated to 6 bit and 3 signed-digits. The filter specification is:

```
nbts=6 % Coefficient bits
```

```

ndigits=3 % Nominal average coefficient signed-digits
tol=0.001 % Tolerance on coefficient. update
maxiter=400 % SQP iteration limit
npoints=250 % Frequency points across the band
length(c0)=21 % Num. tap coefficients
sum(k0~=0)=10 % Num. non-zero all-pass coef.s
dmax=0.250000 % Constraint on norm of coefficient SQP step size
rho=0.992188 % Constraint on alpass coefficients
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
Wap=1 % Amplitude pass band weight
ftpl=0.09 % Delay pass band lower edge
ftpu=0.21 % Delay pass band upper edge
tp=16 % Nominal passband filter group delay
Wtp=8 % Delay pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
Wasl=10000 % Amplitude lower stop band weight
Wasu=100000 % Amplitude upper stop band weight

```

The 31 non-zero 6-bit 3 signed-digit one-multiplier lattice coefficients found by the branch-and-bound search are:

```

32*k_min = [      0,      22,      0,      17, ...
              0,      11,      0,      14, ...
              0,      10,      0,       8, ...
              0,       5,      0,       3, ...
              0,       1,      0,       0 ]';

32*c_min = [      2,       0,      -9,     -15, ...
              -6,       3,      12,      10, ...
              1,      -2,      -2,       0, ...
              0,      -1,      -1,       0, ...
              1,       1,       0,       0, ...
              0 ]';

```

I assume that the internal filter state scaling is approximated by bit-shifts. Figure 14.8 compares the pass-band responses of the filter with exact coefficients, the initial 6 bit 3 signed-digit coefficients and 6 bit 3 signed-digit coefficients found by branch-and-bound search. Figure 14.9 shows the filter stop-band response and Figure 14.10 shows the filter pass-band group delay response. Table 14.5 compares the cost and number of 6 bit shift-and-add operations required to implement the 31 coefficient multiplications for the initial signed-digit coefficients and the coefficients found by the branch-and-bound search. A further 41 additions are required by the lattice filter structure¹. Although 3 signed-digits are allocated to each coefficient, many of these signed-digits are not used.

	Cost	Signed-digits	Additions
Exact	0.0123		
6-bit 3-signed-digit	1.1916	39	16
6-bit 3-signed-digit(branch-and-bound)	0.5047	38	15

Table 14.5: Comparison of the cost and number of additions required to implement the coefficient multiplications for a Schur one-multiplier lattice bandpass filter with 6 bit 3 signed-digit coefficients found by branch-and-bound search

¹A one-multiplier lattice filter with order 21 and denominator polynomial coefficients in z^2 would normally require 1 multiplication and 3 additions for each of 10 lattice sections and 21 filter output tap multiplications and additions. In this case 1 of the lattice filter coefficients and 7 of the tap coefficients are zero. See Figure 7.4.

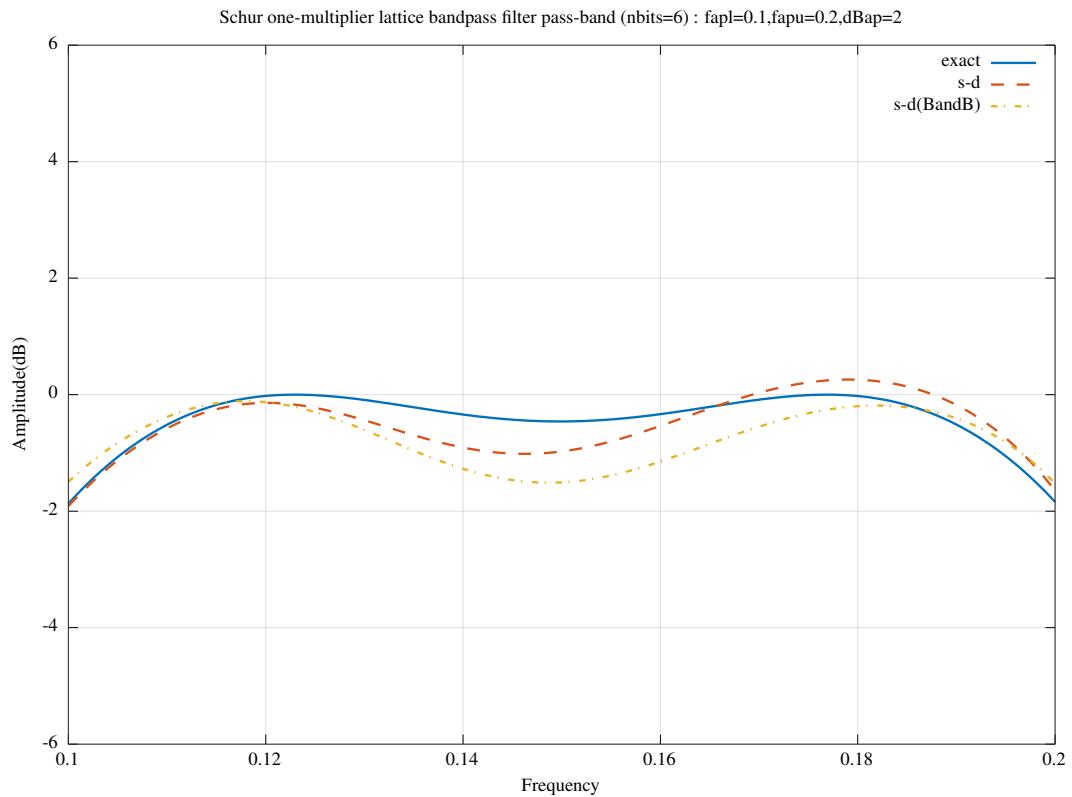


Figure 14.8: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 6 bit 3 signed-digit coefficients found by branch-and-bound search

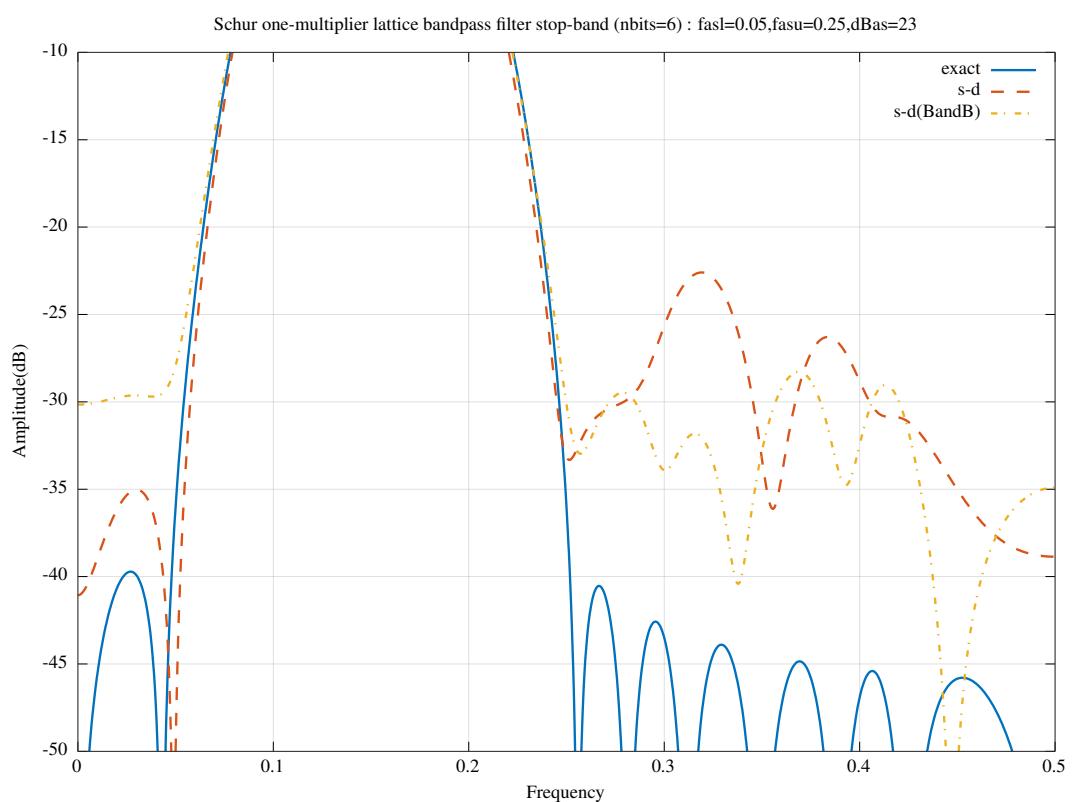


Figure 14.9: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 6 bit 3 signed-digit coefficients found by branch-and-bound search

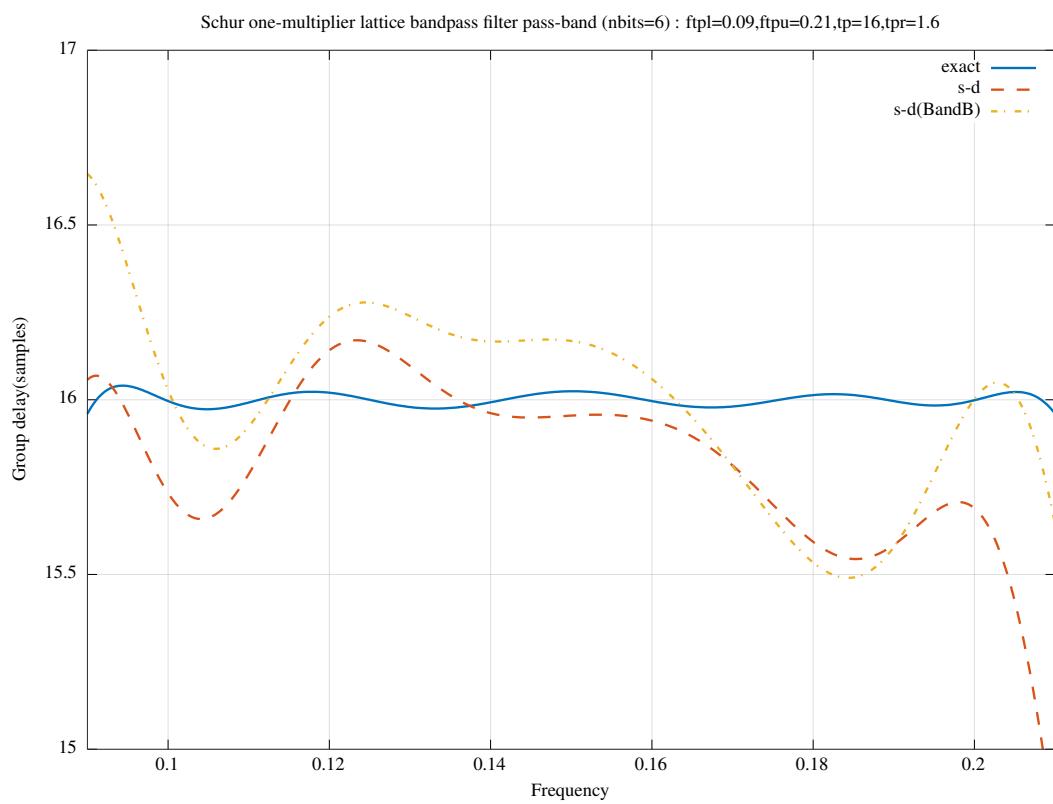


Figure 14.10: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 6 bit 3 signed-digit coefficients found by branch-and-bound search

14.3.2 Branch-and-bound search for the 10 bit 3 signed-digit coefficients of a one-multiplier Schur lattice filter

The Octave script `branch_bound_bandpass_OneM_lattice_10_nbites_test.m` uses the branch-and-bound heuristic to optimise the response of the band-pass Schur one-multiplier lattice filter of Section 12.3.3. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
tol=0.0001 % Tolerance on coefficient update
maxiter=400 % SQP iteration limit
npoints=250 % Frequency points across the band
length(c0)=21 % Num. tap coefficients
sum(k0~=0)=10 % Num. non-zero all-pass coef.s
dmax=0.250000 % Constraint on norm of coefficient SQP step size
rho=0.992188 % Constraint on allpass coefficients
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftpl=0.09 % Delay pass band lower edge
ftpup=0.21 % Delay pass band upper edge
tp=16 % Nominal passband filter group delay
tpr=0.16 % Delay pass band peak-to-peak ripple
Wtp=6 % Delay pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
dBas=33 % Amplitude stop band peak-to-peak ripple
Wasl=100000 % Amplitude lower stop band weight
Wasu=1000000 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 10 bits with an average of 3 signed-digits allocated to each coefficient. The number of signed-digits allocated to each coefficient is determined by the heuristic of *Ito et al.* as shown in Section 13.2. The average number of non-zero signed-digits per coefficient is close to 2. I assume that the internal filter state scaling is approximated by bit-shifts.

At each branch the script fixes the coefficient with the largest difference between upper and lower 3 signed-digit approximations to the floating-point value. For each sub-problem the coefficients “higher-up-the-tree” are fixed and the remaining free coefficients are SQP PCLS optimised with the filter specification given above. For this example the branch-and-bound search makes thousands of branches and requires several hours CPU time on my PC.

The filter coefficients found by the branch-and-bound search are:

```
512*k_min = [      0,     328,      0,     260, ...
                0,     168,      0,     216, ...
                0,     145,      0,     125, ...
                0,      71,      0,      51, ...
                0,      16,      0,       7 ]';
512*c_min = [    40,      -2,    -144,    -256, ...
               -95,      48,    190,     160, ...
                17,     -36,     -40,      -8, ...
                -4,     -16,     -15,      0, ...
               12,      10,      2,      -1, ...
                2 ]';
```

Figure 14.11 compares the pass-band responses of the filter with exact, 10 bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and 10 bit signed-digits allocated with the algorithm of *Ito et al.* and branch-and-bound search. Figure 14.12 shows the filter stop-band response and Figure 14.13 shows the filter pass-band group delay response. Table 14.6 compares the cost and the number of 10 bit shift-and-add operations required to implement the 31 coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the branch-and-bound search. A further 51 additions are required by the lattice filter structure. The truncation of the last coefficient is, by necessity, not PCLS optimised so the final set of coefficients may not meet the PCLS specifications.

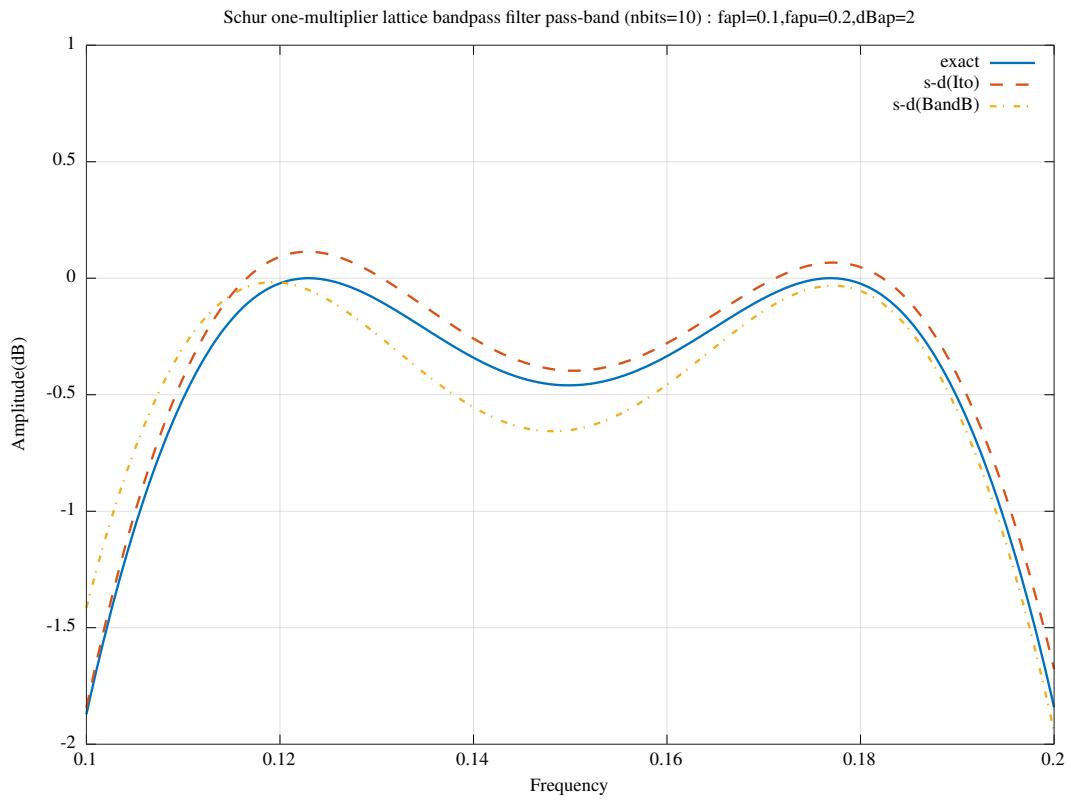


Figure 14.11: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search

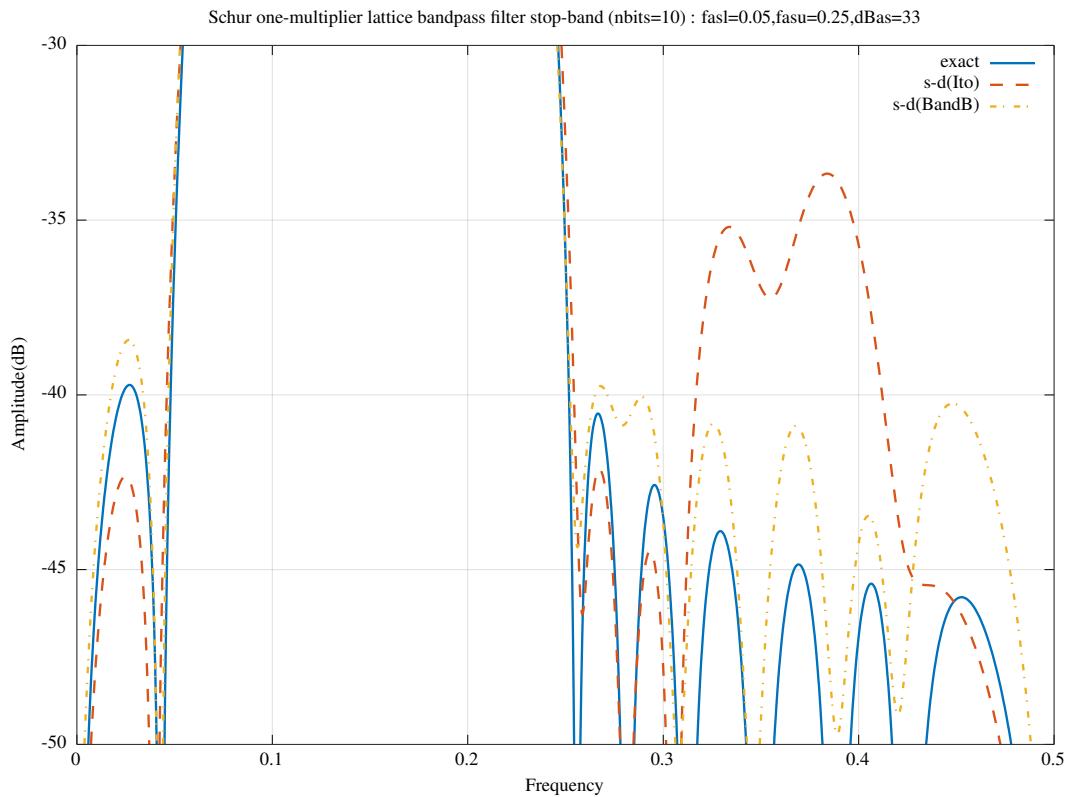


Figure 14.12: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search

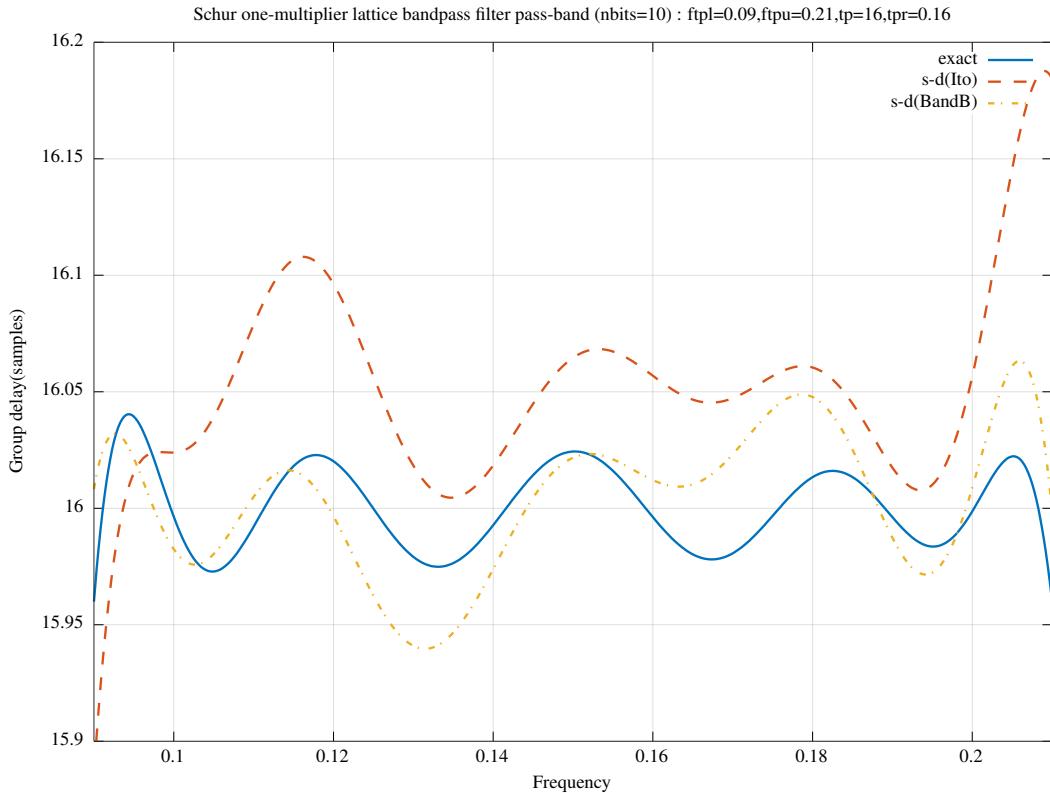


Figure 14.13: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search

	Cost	Signed-digits	Shift-and-adds
Exact	0.0138		
10-bit 3-signed-digit(Ito)	0.0838	62	31
10-bit 3-signed-digit(branch-and-bound)	0.0212	61	31

Table 14.6: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search

14.3.3 Branch-and-bound search for the 12 bit 2 signed-digit coefficients of an FRM Hilbert filter with an allpass model filter

The Octave script *branch_bound_schurOneMAPlattice_frm_hilbert_12_nbis_test.m* uses the branch-and-bound heuristic to optimise the response of the FRM Hilbert filter of Section 12.4.7 with 12 bit 2 signed-digit coefficients. The filter specification is:

```
n=800 % Frequency points across the band
tol=1e-05 % Tolerance on coefficient update vector
n=800 % Frequency points across the band
mr=5 % Allpass model filter denominator order
Mmodel=7 % Model filter FRM decimation factor
Dmodel=9 % Model filter nominal pass band group delay
dmask=16 % FIR masking filter delay
fap=0.01 % Magnitude-squared pass band edge
fas=0.49 % Magnitude-squared stop band edge
dBap=0.2 % Pass band magnitude-squared peak-to-peak ripple
Wap=1 % Pass band magnitude-squared weight
ftp=0.01 % Delay pass band edge
fts=0.49 % Delay stop band edge
tp=79 % Pass band nominal delay
tpr=tp/90 % Pass band delay peak-to-peak ripple
Wtp=1 % Pass band magnitude-squared weight
fpp=0.01 % Phase pass band edge
fps=0.49 % Phase stop band edge
pp=-0.5*pi % Pass band phase peak-to-peak ripple (rad.)
ppr=pi/360 % Pass band phase peak-to-peak ripple (rad.)
Wpp=0.005 % Phase pass band weight
```

The truncation of the last coefficient is, by necessity, not PCLS optimised so the final set of coefficients may not meet the PCLS specifications. The filter coefficients found by the branch-and-bound search are:

```
2048*k_min = [ -1152,    -264,     -96,     -36, ...
                -12 ]';

2048*u_min = [      0,     -3,    -16,    -24, ...
                -63,    -72,   -112,   -120, ...
                896 ]';

2048*v_min = [      0,     -3,    -16,    -24, ...
                -63,    -72,   -112,   -120, ...
                896 ]';
```

Figures 14.14, 14.15 and 14.16 compare the amplitude, phase and delay responses of the FRM Hilbert filter with exact and 12 bit 2 signed-digit coefficients and with the 12 bit 2 signed-digit coefficients found by branch-and-bound search. Figure 14.17 compares the amplitude and phase responses of a linear phase FIR Hilbert filter with 12 bit 2 signed-digit coefficients and the FRM Hilbert filter with 12 bit 2-signed digit integer coefficients found by branch-and-bound search. The group delay of the FIR Hilbert filter is the nominal delay of the FRM Hilbert filter:

```
b=remez(2*tp,2*[fap fas],[1 1],1,"hilbert");
```

The FIR Hilbert filter coefficients have not been optimised. Table 14.7 compares the cost and the number of 12 bit shift-and-add operations required to implement the coefficient multiplications found by the branch-and-bound search.

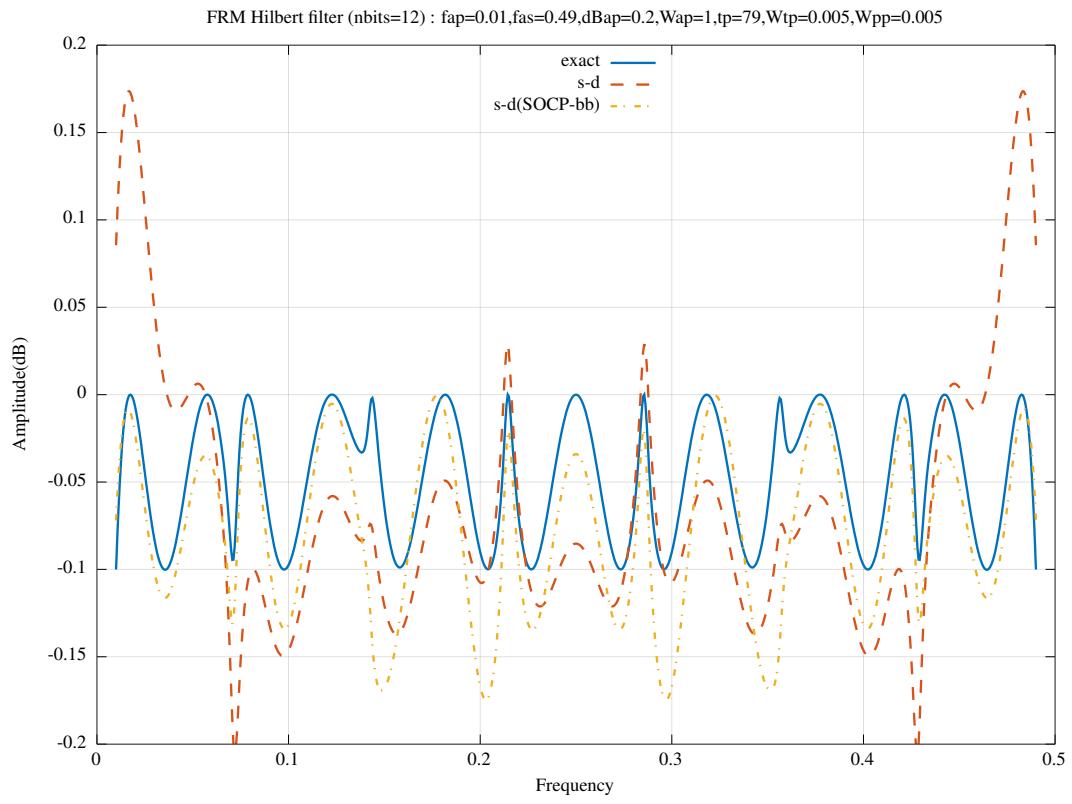


Figure 14.14: Comparison of the amplitude response of an FRM Hilbert filter with exact and 12 bit 2-signed digit integer coefficients found by branch-and-bound search

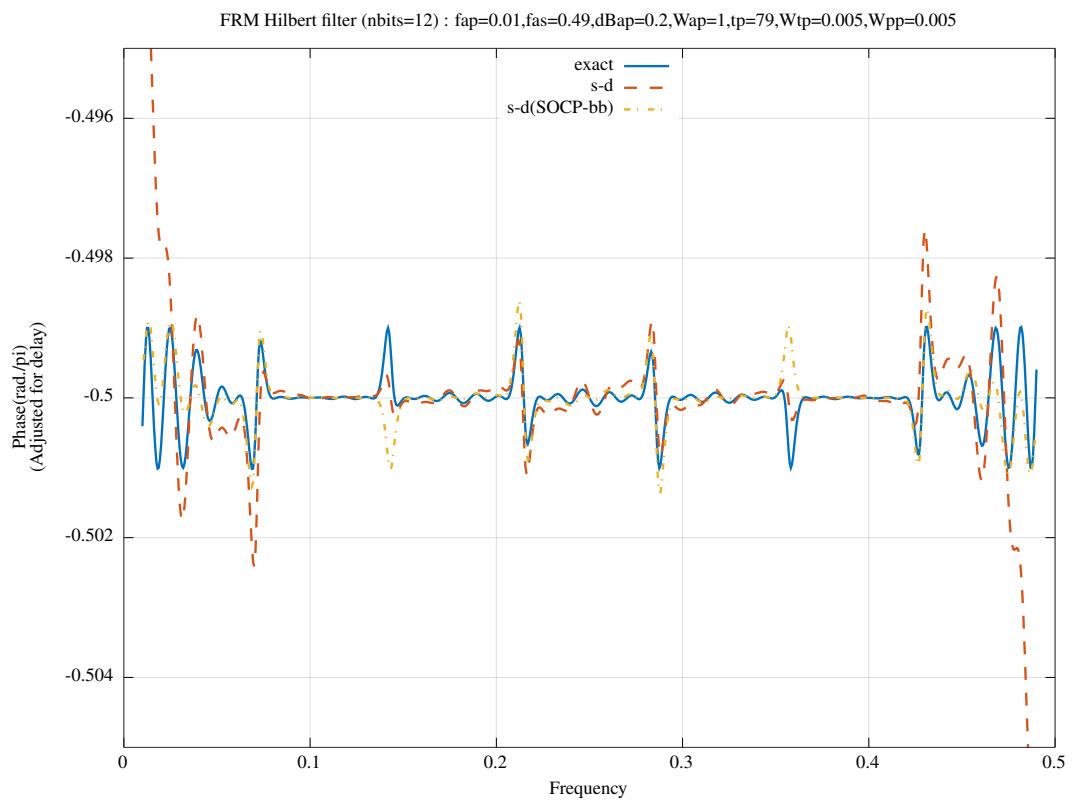


Figure 14.15: Comparison of the phase response of an FRM Hilbert filter with exact and 12 bit 2-signed digit integer coefficients found by branch-and-bound search

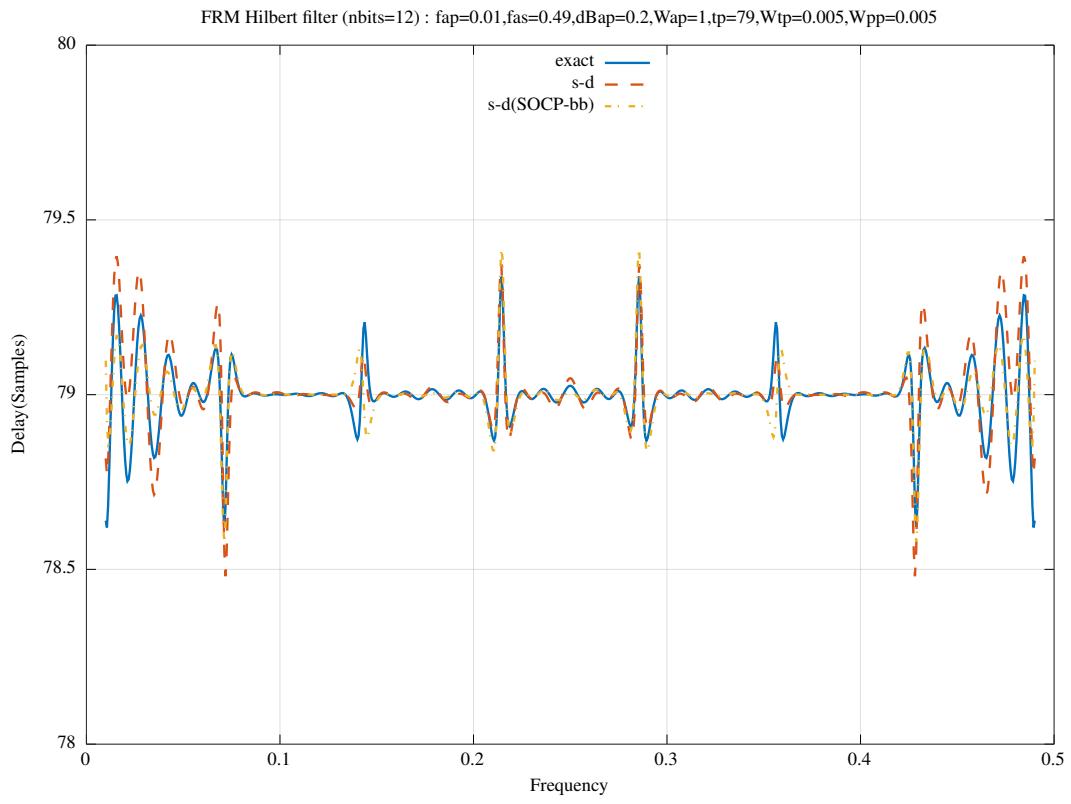


Figure 14.16: Comparison of the delay response of an FRM Hilbert filter with exact and 12 bit 2-signed digit integer coefficients found by branch-and-bound search

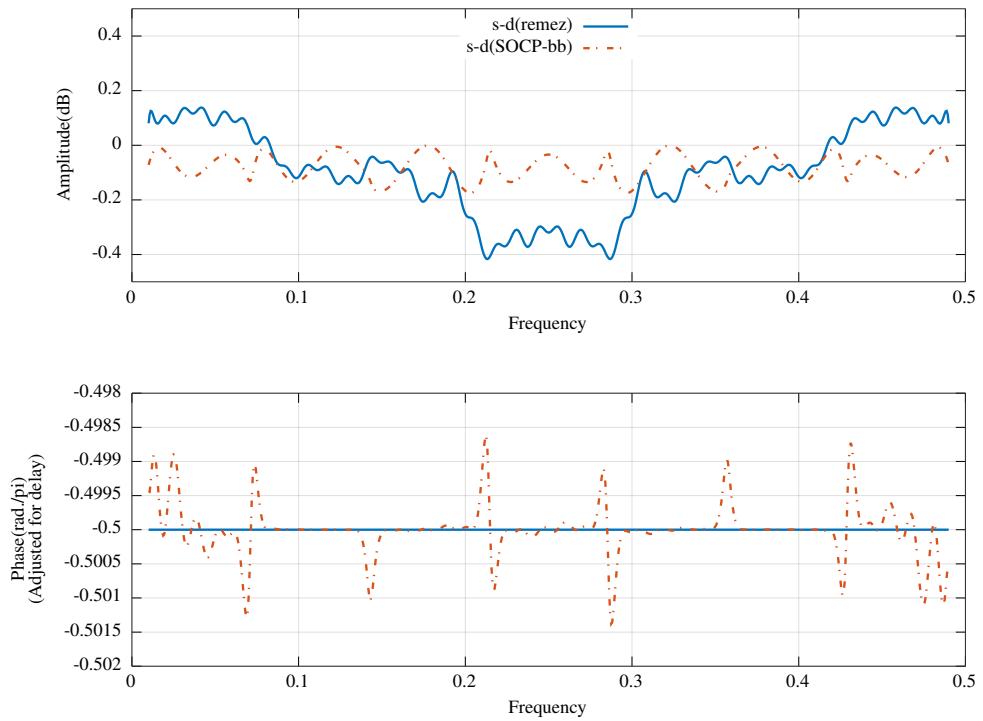


Figure 14.17: Comparison of the amplitude and phase responses of an FIR Hilbert filter with 12 bit 2 signed-digit coefficients and an FRM Hilbert filter with 12 bit 2-signed digit integer coefficients found by branch-and-bound search

	Cost	Signed-digits	Shift-and-adds
Exact	0.000655		
12-bit 2-signed-digit	0.001686	40	18
12-bit 2-signed-digit(branch-and-bound)	0.001387	40	19
12-bit 2-signed-digit(remez)	0.002565	73	33

Table 14.7: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for an FRM Hilbert filter with 12 bit 2 signed-digit coefficients found by branch-and-bound search

14.4 Relaxation search for signed-digit filter coefficients

The branch-and-bound algorithm of Section 14.3 performs SQP optimisation at each coefficient relaxation step. Alternatively, the optimisation problem can be converted to an *integer programming* relaxation problem. Given a set of exact coefficients, \mathbf{x} , and signed-power-of-two upper and lower bounds, \mathbf{u} and \mathbf{l} , corresponding to stable filter implementations such that $u_i \geq x_i \geq l_i$ for each of the components of \mathbf{x} , define

$$\hat{\mathbf{x}} = \frac{(\mathbf{u} + \mathbf{l})}{2}$$

$$\boldsymbol{\delta} = \frac{(\mathbf{u} - \mathbf{l})}{2}$$

Then $u_i = \hat{x}_i + y_i \delta_i$ if $y_i = 1$ and $l_i = \hat{x}_i + y_i \delta_i$ if $y_i = -1$. The response error at $\hat{\mathbf{x}} + (\mathbf{y} \circ \boldsymbol{\delta})$ is approximately:

$$\mathcal{E}(\hat{\mathbf{x}} + (\mathbf{y} \circ \boldsymbol{\delta})) \approx \nabla_x \mathcal{E}(\hat{\mathbf{x}})^T (\mathbf{y} \circ \boldsymbol{\delta}) + \frac{1}{2} (\mathbf{y} \circ \boldsymbol{\delta})^T \nabla_{xx}^2 \mathcal{E}(\hat{\mathbf{x}}) (\mathbf{y} \circ \boldsymbol{\delta})$$

where \circ represents element-by-element multiplication. The signed-digit filter coefficients can be found by the optimisation:

$$\begin{aligned} & \text{minimise} \quad \nabla_x \mathcal{E}(\hat{\mathbf{x}})^T (\mathbf{y} \circ \boldsymbol{\delta}) + \frac{1}{2} (\mathbf{y} \circ \boldsymbol{\delta})^T \nabla_{xx}^2 \mathcal{E}(\hat{\mathbf{x}}) (\mathbf{y} \circ \boldsymbol{\delta}) \\ & \text{subject to} \quad y_i \in \{1, -1\} \end{aligned} \quad (14.1)$$

$$(14.2)$$

Lu [102] and *Ito et al.* [81] point out that this is similar to the *maximum-cut* problem of graph theory. A simple relaxation of the optimisation problem is:

$$\begin{aligned} & \text{minimise} \quad \nabla_x \mathcal{E}(\hat{\mathbf{x}})^T (\mathbf{y} \circ \boldsymbol{\delta}) + \frac{1}{2} (\mathbf{y} \circ \boldsymbol{\delta})^T \nabla_{xx}^2 \mathcal{E}(\hat{\mathbf{x}}) (\mathbf{y} \circ \boldsymbol{\delta}) \\ & \text{subject to} \quad -1 \leq y_i \leq 1 \end{aligned}$$

14.4.1 SQP relaxation search for the signed-digit coefficients of a Schur one-multiplier lattice band-pass filter

The Octave script *sqp_relaxation_bandpass_OneM_lattice_10_nbites_test.m* performs successive SQP relaxations to optimise the response of the band-pass Schur one-multiplier lattice filter of Section 12.3.3. The filter specification is:

```
nbits=10 % Coeficient bits
ndigits=3 % Nominal average coeficient signed-digits
tol=0.0001 % Tolerance on coef. update
maxiter=150 % SQP iteration limit
npoints=250 % Frequency points across the band
length(c0)=21 % Num. tap coefficients
sum(k0~=0)=10 % Num. non-zero all-pass coef.s
dmax=0.250000 % Constraint on norm of coefficient SQP step size
rho=0.992188 % Constraint on allpass coefficients
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftpl=0.09 % Delay pass band lower edge
ftpu=0.21 % Delay pass band upper edge
tp=16 % Nominal passband filter group delay
tpr=0.16 % Delay pass band peak-to-peak ripple
Wtp=6 % Delay pass band weight
fasl=0.05 % Amplitude stop band(1) lower edge
fasu=0.25 % Amplitude stop band(1) upper edge
dBas=33 % Amplitude stop band(1) peak-to-peak ripple
fasll=0.04 % Amplitude stop band(2) lower edge
fasuu=0.26 % Amplitude stop band(2) upper edge
dBass=36 % Amplitude stop band(2) peak-to-peak ripple
Wasl=100000 % Amplitude lower stop band weight
Wasu=1000000 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 10 bits allocated with an average of 3 signed-digits by the heuristic of *Ito et al.* as shown in Section 13.2.

At each coefficient relaxation step the script finds the upper and lower signed-digit approximations to the current set of active coefficients and selects the coefficient with the largest difference in those approximations. The corresponding bounds are set in the bounds for that coefficient passed to the Octave function *schurOneMlattice_sqp_mmse*. The results of this MMSE optimisation are then passed to *schurOneMlattice_slb* for PCLS optimisation. The resulting PCLS coefficient value selects the closer of the upper or lower signed-digit values as the final choice for that coefficient. The truncation of the last coefficient is, by necessity, not PCLS optimised so the final set of coefficients may not meet the PCLS specifications.

The filter coefficients found by the SQP-relaxation search are:

```
512*k_min = [ 0,    336,    0,    264, ...
               0,    176,    0,    216, ...
               0,    145,    0,    126, ...
               0,    72,     0,    51, ...
               0,    16,     0,     8 ]';
512*c_min = [ 40,    -2,   -144,   -256, ...
               -94,    48,   192,   160, ...
                15,   -40,   -40,    -8, ...
                -2,   -16,   -12,     1, ...
               12,     8,     1,   -1, ...
                1 ]';
```

Figure 14.18 compares the pass-band responses of the filter with exact, 10 bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and 10 bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and SQP-relaxation search. Figure 14.19 shows the filter stop-band response and Figure 14.20 shows the filter pass-band group delay response. Table 14.8 compares the cost and the number of 10 bit shift-and-add operations required to implement the 31 coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the SQP-relaxation search. A further 51 additions are required by the lattice filter structure.

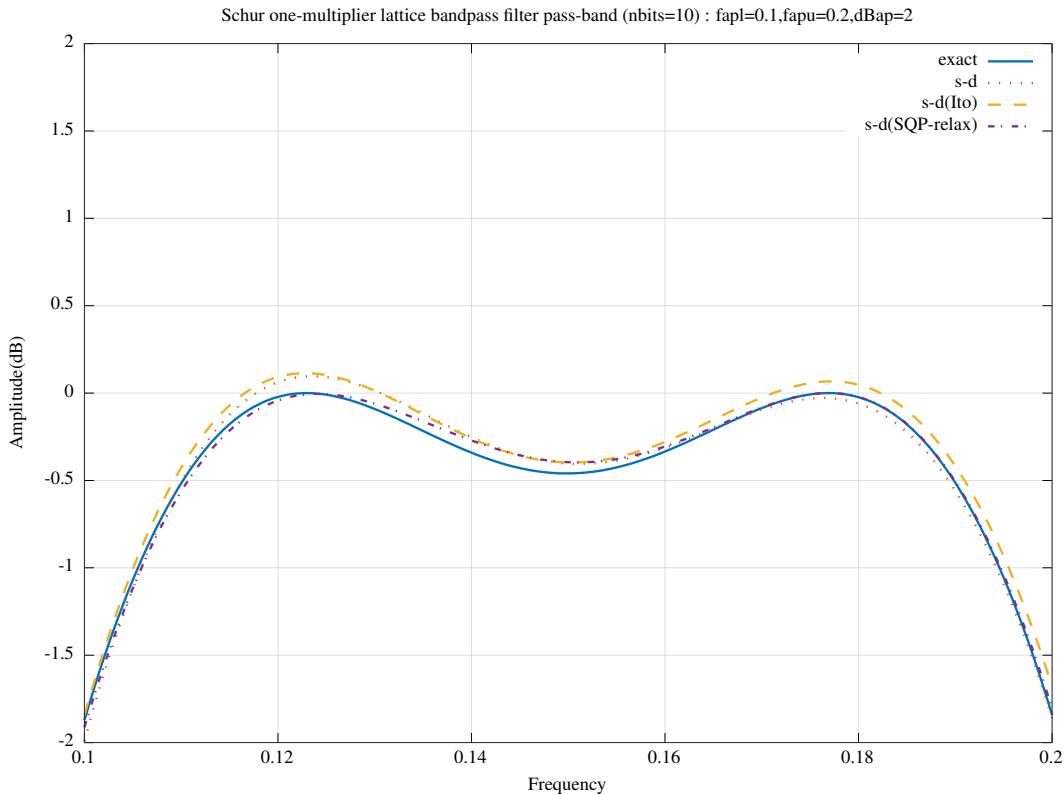


Figure 14.18: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation search

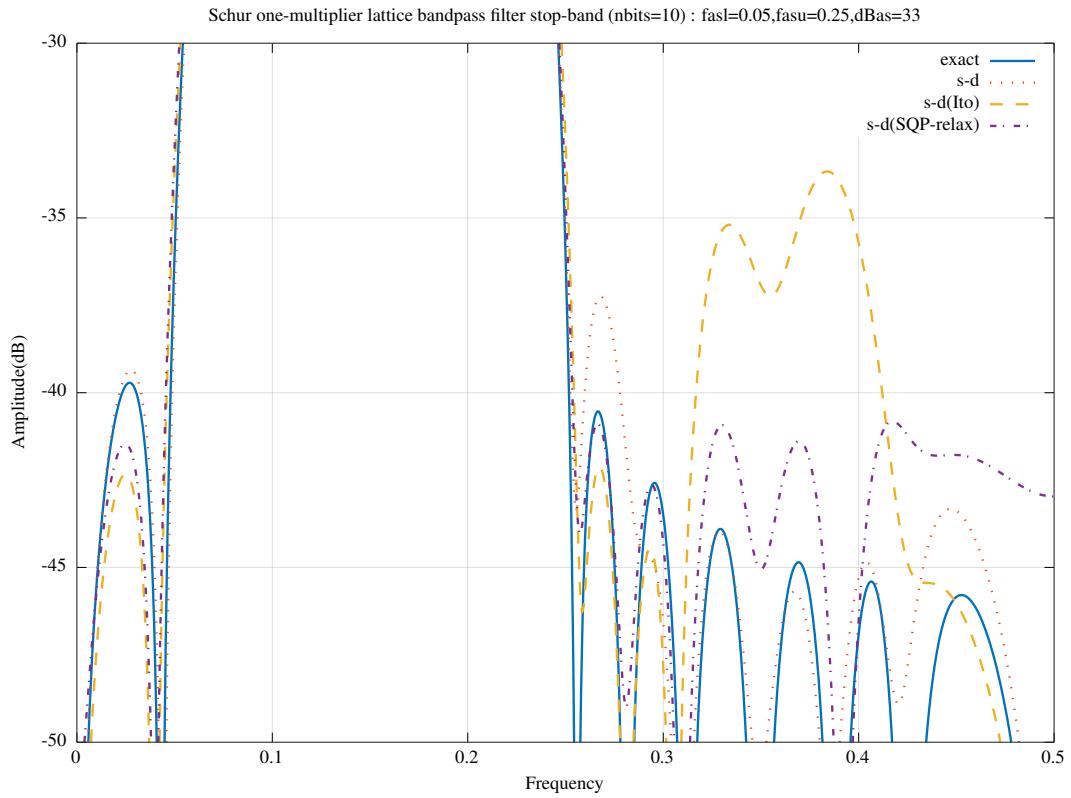


Figure 14.19: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation search

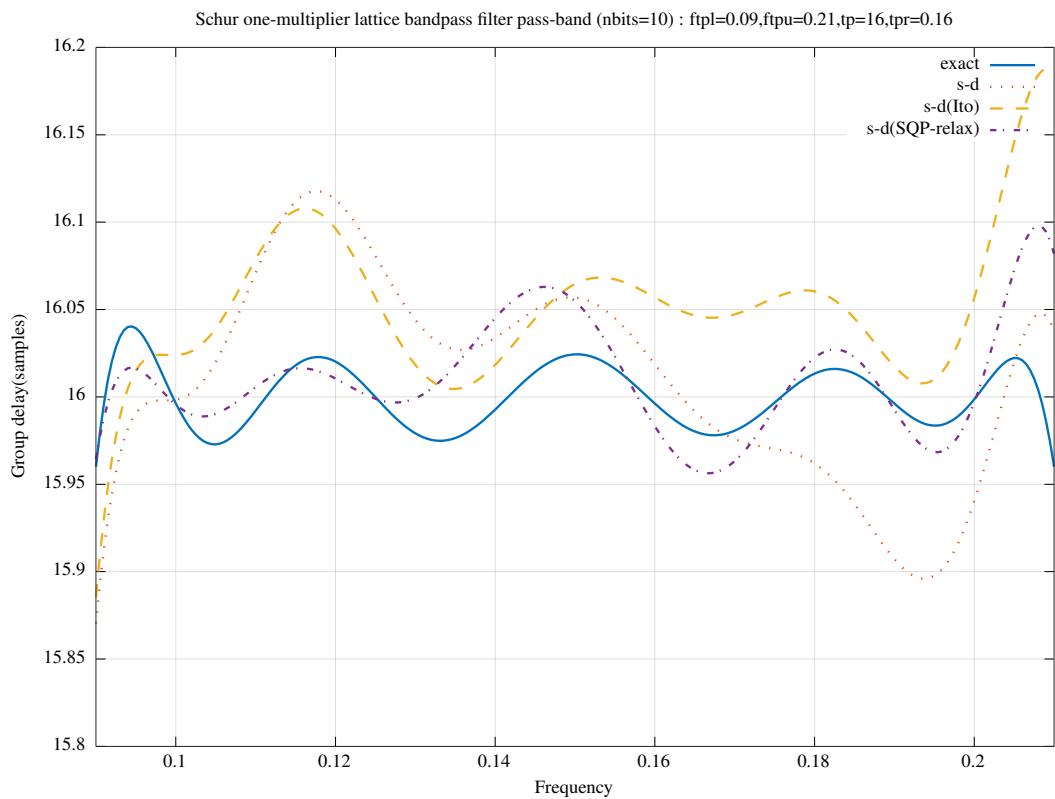


Figure 14.20: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation search

	Cost	Signed-digits	Shift-and-adds
Exact	0.0138		
10-bit 3-signed-digit	0.0319	72	41
10-bit 3-signed-digit(Ito)	0.0838	62	31
10-bit 3-signed-digit(SQP-relax)	0.0222	57	26

Table 14.8: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation search

14.4.2 SQP relaxation search for the signed-digit coefficients of a Schur one-multiplier lattice low-pass filter

The Octave script *sqp_relaxation_lowpass_OneM_lattice_10_nbites_test.m* performs successive SQP relaxations to optimise the response of the low-pass Schur one-multiplier lattice filter of Section 12.3.2 with 10-bit integer coefficients. The filter specification is:

```

tol=0.0001 % Tolerance on coefficient update vector
nbits=10 % coefficient length in bits
ndigits=3 % signed-digits per coefficient
n=400 % Frequency points across the band
length(c0)=11 % Tap coefficients
sum(k0~=0)=6 % Num. non-zero lattice coefficients
dmax=0.050000 % Constraint on norm of coefficient SQP step size
rho=0.992188 % Constraint on lattice coefficient magnitudes
fap=0.15 % Amplitude pass band edge
dBap=0.4 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftp=0.25 % Delay pass band edge
tp=10 % Nominal pass band filter group delay
tpr=0.1 % Delay pass band peak-to-peak ripple
Wtp=1 % Delay pass band weight
fas=0.3 % Amplitude stop band edge
dBas=37 % amplitude stop band peak-to-peak ripple
Was=10000 % Amplitude stop band weight

```

The filter coefficients are truncated to 10 bits allocated with an average of 3 signed-digits by the heuristic of *Ito et al.* as shown in Section 13.2. At each coefficient relaxation step the script finds the upper and lower signed-digit approximations to the current set of active coefficients and selects the coefficient with the largest difference in those approximations. The corresponding bounds are set in the bounds for that coefficient passed to the Octave function *schurOneMlattice_sqp_mmse*. The results of this MMSE optimisation are then passed to *schurOneMlattice_slb* for PCLS optimisation. The resulting PCLS coefficient value selects the closer of the upper or lower signed-digit values as the final choice for that coefficient. The truncation of the last coefficient is, by necessity, not PCLS optimised so the final set of coefficients may not meet the PCLS specifications.

The filter coefficients found by the SQP-relaxation search are:

```

512*k_min = [ -361,      355,     -302,      215, ...
               -111,       32,        0,        0, ...
                  0,       0 ]';

512*c_min = [    138,      354,       41,      -44, ...
                -22,        5,       11,        4, ...
                  -4,      -4,        0 ]';

```

Figure 14.21 shows the amplitude and group-delay responses of the filter with 10-bit 3-signed-digit coefficients allocated with the algorithm of *Lim et al.* and SQP-relaxation search. Figure 14.22 shows the corresponding filter pass-band response and Figure 14.23 shows the filter pole-zero plot. Table 14.9 compares the cost and the number of 10 bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Lim et al.* with the SQP-relaxation search.

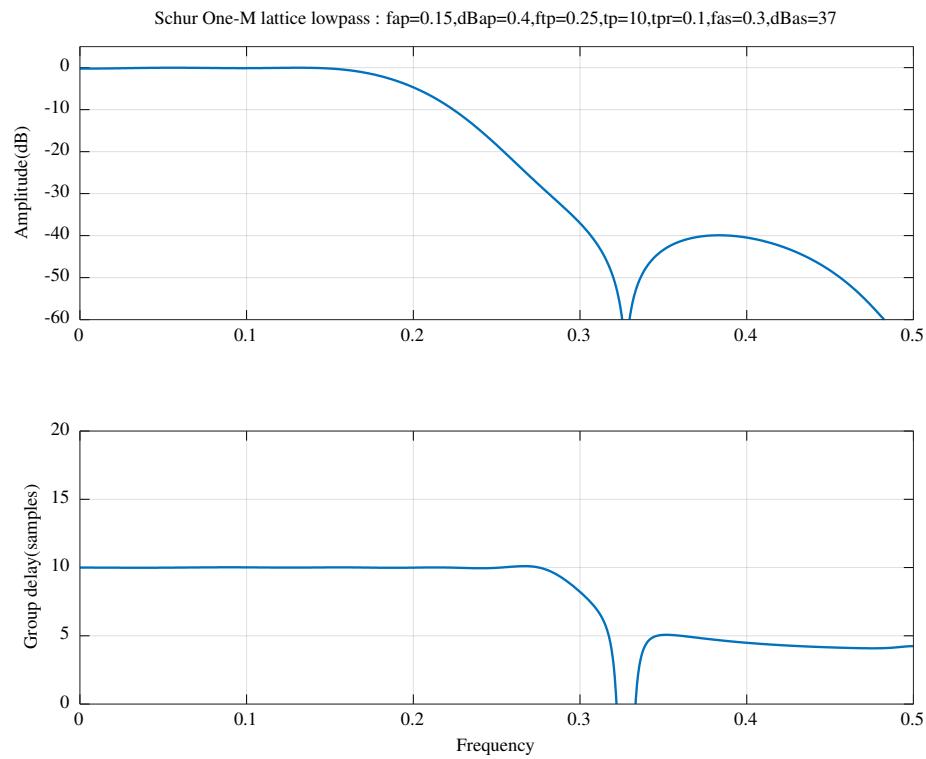


Figure 14.21: Amplitude and group-delay responses for a Schur one-multiplier lattice low-pass filter with 10 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SQP-relaxation search.

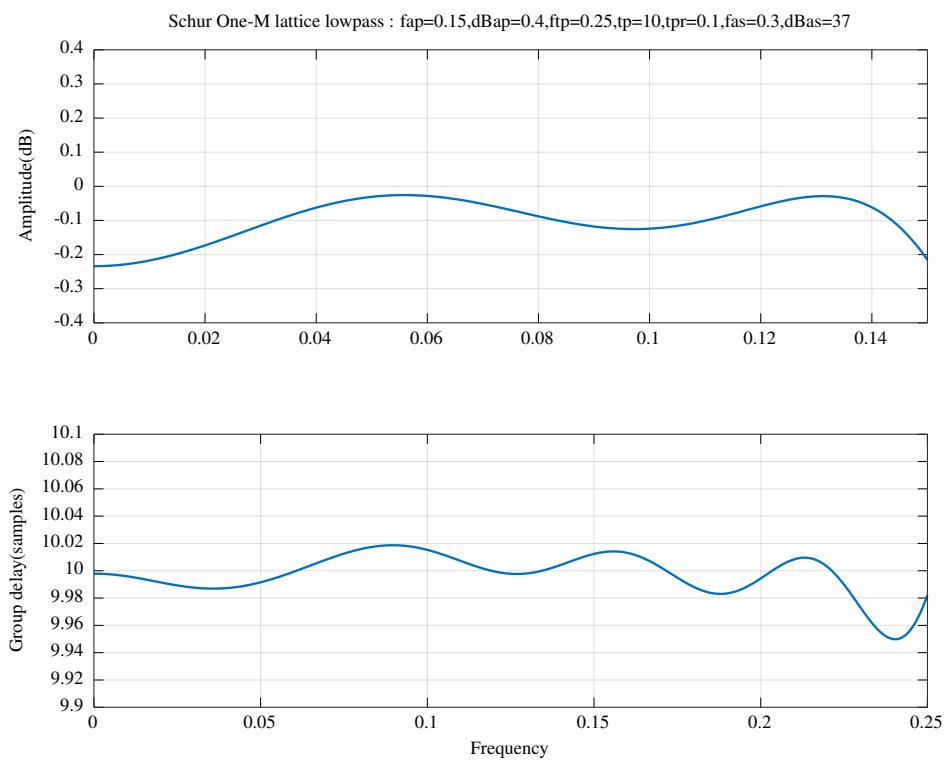


Figure 14.22: Amplitude and group-delay pass-band responses for a Schur one-multiplier lattice low-pass filter with 10 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SQP-relaxation search.

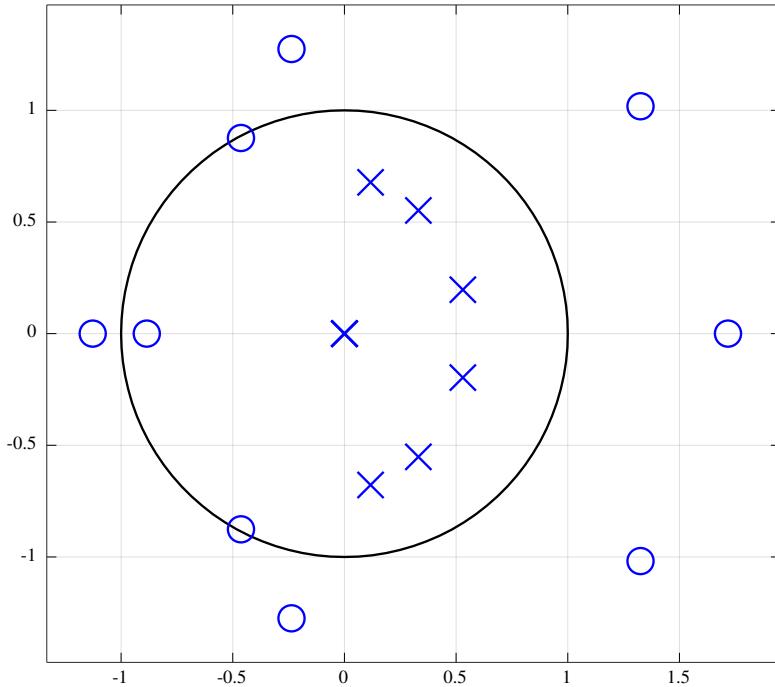


Figure 14.23: Pole-zero plot for a Schur one-multiplier lattice low-pass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SQP-relaxation search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.001633		
10-bit 3-signed-digit(Lim)	0.036110	42	26
10-bit 3-signed-digit(SQP-relax)	0.001063	46	30

Table 14.9: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier lattice low-pass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SQP-relaxation search.

14.4.3 SOCP-relaxation search for the signed-digit coefficients of a Schur one-multiplier lattice Hilbert filter

The Octave script *socp_relaxation_hilbert_OneM_lattice_10_nbts_test.m* performs successive SOCP relaxations to optimise the response of the band-pass Schur one-multiplier lattice filter of Section 12.3.4 with 10-bit integer coefficients. The filter specification is:

```

tol=1e-08 % Tolerance on coefficient update vector
n=400 % Frequency points across the band
rho=1.000000 % Constraint on lattice coefficient magnitudes
ft=0.05 % Transition band width [0,ft]
dBap=0.17 % Amplitude pass band peak-to-peak ripple
Wat=1e-08 % Amplitude transition band weight
Wap=1 % Amplitude pass band weight
tp=5.5 % Nominal pass band filter group delay (samples)
pr=0.011 * pi % Phase pass band peak-to-peak ripple (rad.)
Wpp=1 % Phase pass band weight

```

The filter coefficients are truncated to 10 bits allocated with an average of 3 signed-digits by the heuristic of *Lim et al.* as shown in Section 13.1. In this example, the group delay response is not constrained. I did not succeed in finding a Hilbert filter with integer coefficients using the SQP solver or using the signed-digit allocation heuristic of *Ito et al.*.

As in Section 14.4.1, at each coefficient relaxation step the script finds the upper and lower signed-digit approximations to the current set of active coefficients and selects the coefficient with the largest difference in those approximations. The corresponding bounds are set in the bounds for that coefficient passed to the Octave function `schurOneMlattice_socp_mmse`. The results of this MMSE optimisation are then passed to `schurOneMlattice_slb` for PCLS optimisation. The resulting PCLS coefficient value selects the closer of the upper or lower signed-digit values as the final choice for that coefficient. The truncation of the last coefficient is, by necessity, not PCLS optimised so the final set of coefficients may not meet the PCLS specifications.

The signed-digit filter coefficients found by the heuristic of *Lim et al.* are:

```
512*k0_sd = [      0,    -468,      0,    284, ...
                0,    -49,      0,      0, ...
                0,    -1,      0,      1 ]';
```

```
512*c0_sd = [    22,     26,    124,    148, ...
                96,    146,    351,   -303, ...
               -84,   -40,   -22,   -12, ...
                -8 ]';
```

The signed-digit filter coefficients found by the SOCP-relaxation search are:

```
512*k_min = [      0,    -468,      0,    284, ...
                0,    -50,      0,      0, ...
                0,    -1,      0,      1 ]';
```

```
512*c_min = [    22,     26,    124,    148, ...
                96,    146,    351,   -303, ...
               -84,   -41,   -23,   -12, ...
                -9 ]';
```

Figure 14.24 compares the amplitude, group delay and phase responses of the filter with exact, 10 bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and 10 bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and SOCP-relaxation search. The phase response shown is adjusted to allow for the filter group-delay and then scaled by π . Table 14.10 compares the cost, the number of signed-digits and the number of 10-bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Lim et al.* with SOCP-relaxation search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.000161		
10-bit 3-signed-digit(Lim)	0.000218	47	29
10-bit 3-signed-digit(SOCP-relax)	0.000259	48	30

Table 14.10: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier lattice Hilbert filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search.

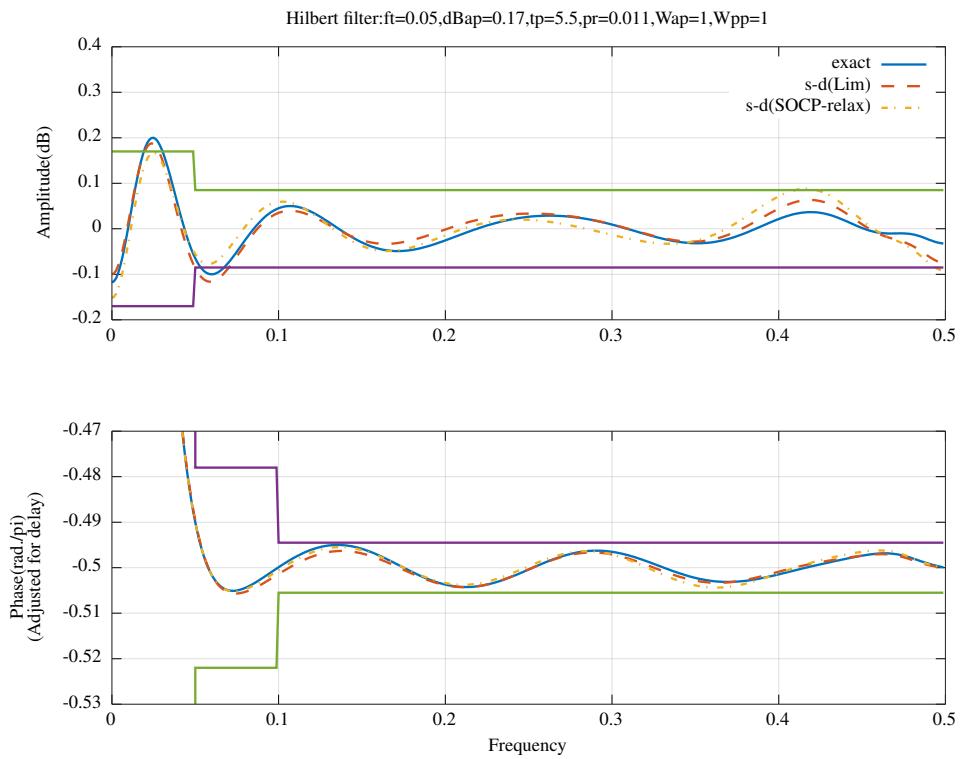


Figure 14.24: Comparison of the amplitude, group delay and phase responses for a Schur one-multiplier lattice Hilbert filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The phase response shown is adjusted to allow for the filter group-delay and then scaled by π .

14.4.4 SOCP-relaxation search for the signed-digit coefficients of an FRM Hilbert filter with a Schur one-multiplier all-pass lattice model filter

The Octave script `socp_relaxation_schurOneMAPlattice_frm_hilbert_12_nbis_test.m` performs successive SOCP relaxations to optimise the response of the FRM Hilbert filter of Section 12.4.7 with 12-bit integer coefficients. The FRM model model filter is implemented as a Schur one-multiplier all-pass lattice filter. The filter specification is:

```
n=800 % Frequency points across the band
tol=5e-05 % Tolerance on coefficient update vector
n=800 % Frequency points across the band
mr=5 % Allpass model filter denominator order
Mmodel=7 % Model filter FRM decimation factor
Dmodel=9 % Model filter nominal pass band group delay
dmask=16 % FIR masking filter delay
fap=0.01 % Magnitude-squared pass band edge
fas=0.49 % Magnitude-squared stop band edge
dBap=0.22 % Pass band magnitude-squared peak-to-peak ripple
Wap=1 % Pass band magnitude-squared weight
ftp=0.01 % Delay pass band edge
fts=0.49 % Delay stop band edge
tp=79 % Pass band nominal delay
tpr=tp/50 % Pass band delay peak-to-peak ripple
Wtp=1 % Pass band magnitude-squared weight
fpp=0.01 % Phase pass band edge
fps=0.49 % Phase stop band edge
pp=-0.5*pi % Pass band phase peak-to-peak ripple (rad.)
ppr=pi/100 % Pass band phase peak-to-peak ripple (rad.)
Wpp=0.005 % Phase pass band weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 2 signed-digits by the heuristic of *Lim et al.* shown in Section 13.1.

As in Section 14.4.1, at each coefficient relaxation step the script finds the upper and lower signed-digit approximations to the current set of active coefficients and selects the coefficient with the largest difference in those approximations. The corresponding filter is PCLS optimised by the function `schurOneMAPlattice_frm_hilbert_slb`. The resulting PCLS coefficient value selects the nearer of the upper or lower signed-digit values as the final choice for that coefficient. The truncation of the last coefficient is necessarily not PCLS optimised so the final set of coefficients may not meet the PCLS specifications.

The signed-digit filter coefficients found by the heuristic of *Lim et al.* are:

```
2048*k0_sd = [ -1152, -272, -112, -32, ...
                 -16 ]';
2048*u0_sd = [ -2, -4, -15, -24, ...
                 -64, -72, -104, -116, ...
                 901 ]';
2048*v0_sd = [ 16, 9, 15, 4, ...
                 -16, -64, -168, -644 ]';
```

The signed-digit filter coefficients found by the SOCP-relaxation search are:

```
2048*k_min = [ -1152, -272, -112, -32, ...
                 -16 ]';
2048*u_min = [ -1, -4, -14, -24, ...
                 -64, -72, -104, -116, ...
                 901 ]';
2048*v_min = [ 16, 10, 14, 4, ...
                 -16, -64, -164, -641 ]';
```

Table 14.11 compares the cost, the number of signed-digits and the number of 12-bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Lim et al.* optimised by SOCP-relaxation search. The one-multiplier lattice implementation requires a further 15 additions (3 per coefficient) and the FIR masking filter requires a further 33 additions. Figures 14.25, 14.26 and 14.27 compare the amplitude, delay and phase responses of the filter with exact, 12 bit 2 signed-digit coefficients allocated with the algorithm of *Lim et al.* and 12 bit 2 signed-digit coefficients allocated with the algorithm of *Lim et al.* and optimised by SOCP-relaxation search. The phase response is adjusted for the nominal delay and normalised to π radians.

	Cost	Signed-digits	Shift-and-adds
Exact	0.000655		
12-bit 2-signed-digit(Lim)	0.001206	41	19
12-bit 2-signed-digit(SOCP-relax)	0.001701	41	19

Table 14.11: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for an FRM Hilbert filter with with 12 bit integer coefficients found by allocating an average of 2 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The FRM model model filter is implemented as a Schur one-multiplier all-pass lattice filter.

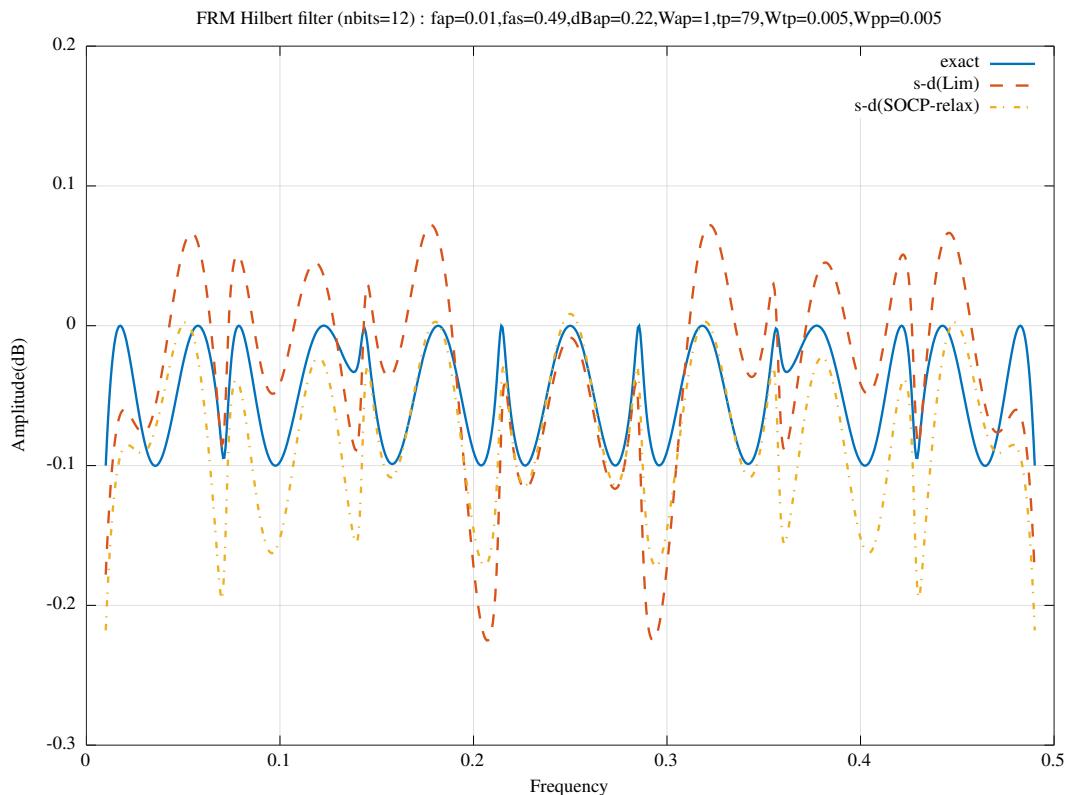


Figure 14.25: Comparison of the amplitude responses for an FRM Hilbert filter with with 12 bit integer coefficients found by allocating an average of 2 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The FRM model model filter is implemented as a Schur one-multiplier all-pass lattice filter.

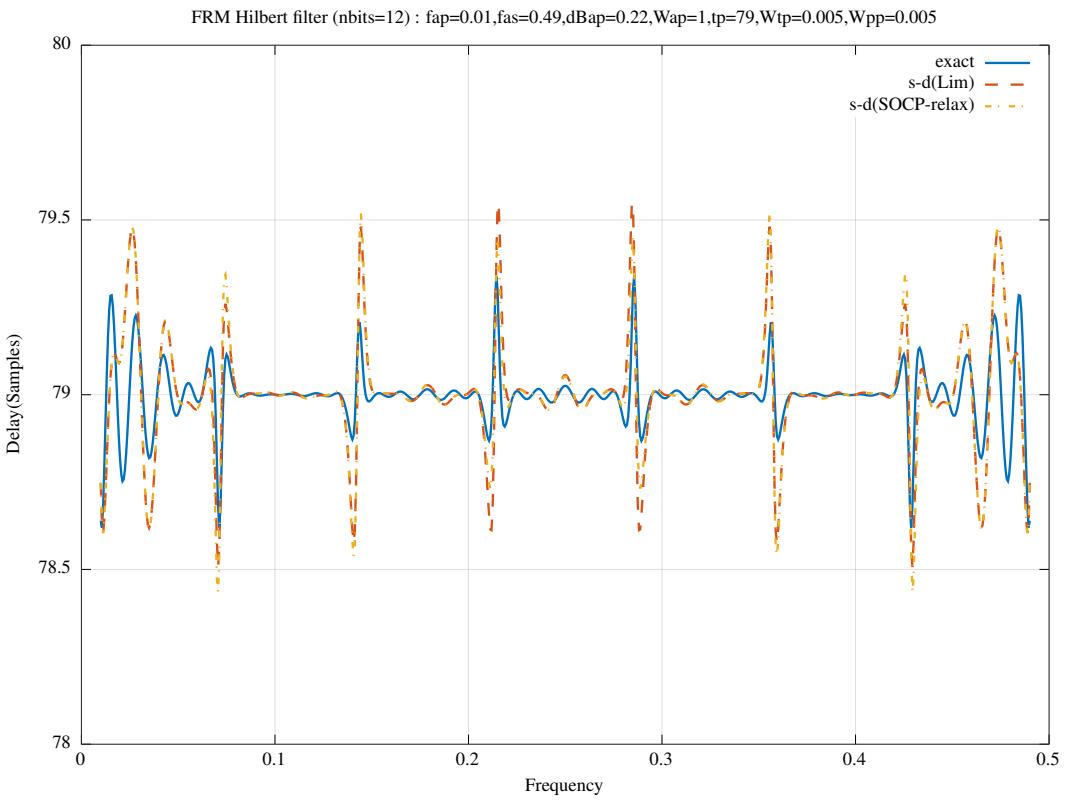


Figure 14.26: Comparison of the delay responses for an FRM Hilbert filter with with 12 bit integer coefficients found by allocating an average of 2 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The FRM model model filter is implemented as a Schur one-multiplier all-pass lattice filter.

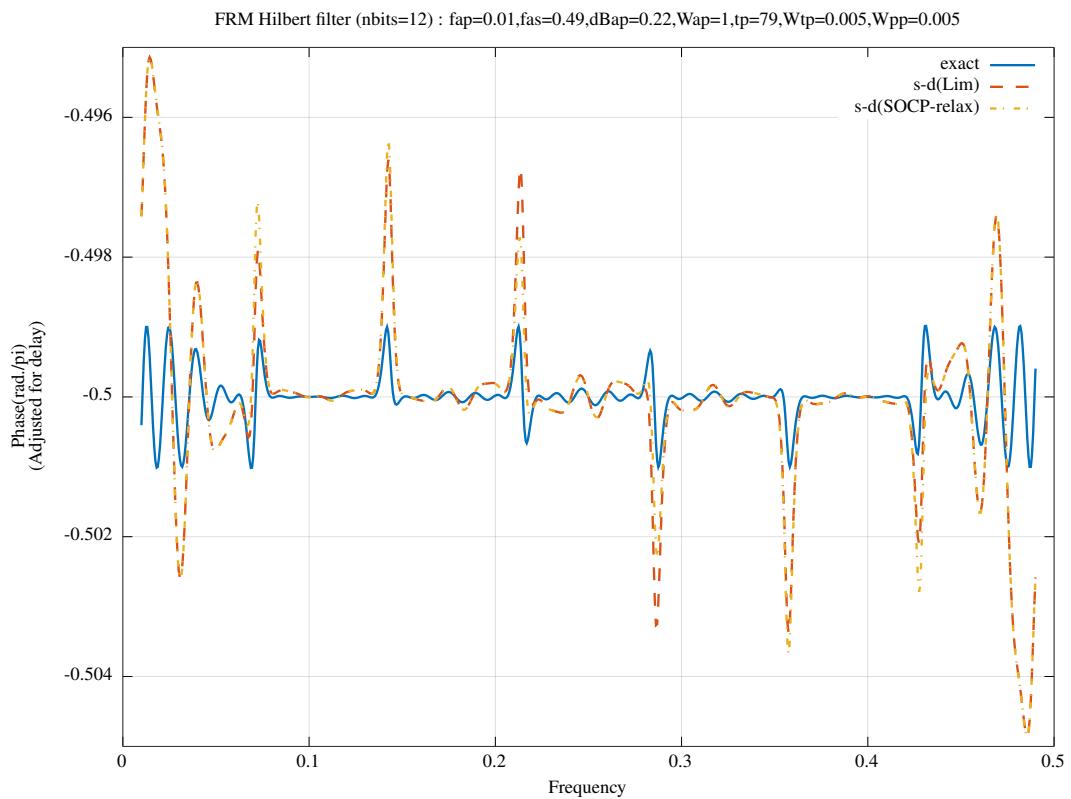


Figure 14.27: Comparison of the phase responses for an FRM Hilbert filter with with 12 bit integer coefficients found by allocating an average of 2 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The FRM model model filter is implemented as a Schur one-multiplier all-pass lattice filter.

14.4.5 POP relaxation search for the signed-digit coefficients of a Schur one-multiplier lattice band-pass filter

Lu [55] and *Lu* and *Hinamoto* [97] point out that the integer programming optimisation of the truncated filter coefficients shown in Equation 14.2 can be rewritten as a *polynomial optimisation problem* (POP) in SOCP form as:

$$\begin{aligned} \text{minimise} \quad & \mathcal{E}(\mathbf{x}_k) + \nabla_x \mathcal{E}(\mathbf{x}_k)^T \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^T \nabla_{xx}^2 \mathcal{E}(\mathbf{x}_k) \Delta \mathbf{x} \\ \text{subject to} \quad & (\mathbf{x}_k + \Delta \mathbf{x} - \mathbf{x}_u)^T (\mathbf{x}_k + \Delta \mathbf{x} - \mathbf{x}_l) = 0 \end{aligned}$$

where $\Delta \mathbf{x}$ is the coefficient update vector, \mathcal{E} is the weighted response error at \mathbf{x}_k and \mathbf{x}_u and \mathbf{x}_l are the upper and lower bounds on the truncated coefficients, $\mathbf{x}_k + \Delta \mathbf{x}$.

Lasserre [44] and *Waki et al.* [35] show that polynomial optimisation can be reduced to the solution of “an often finite sequence of convex linear matrix inequality problems”. *Waki et al.* have written *SparsePOP* [36], a “Matlab implementation of sparse semidefinite programming (SDP) relaxation for polynomial optimization problems”. SparsePOP runs under Octave with minor modifications. SparsePOP converts the POP problem to a larger SOCP problem that is solved by SeDuMi.

The Octave script *pop_relaxation_bandpass_OneM_lattice_socp_slb_test.m* implements SparsePOP optimisation of the truncated coefficients of the band-pass Schur one-multiplier lattice filter described in Section 12.3.3. The filter specification is:

```
nbits=10 % Coeficient bits
ndigits=3 % Nominal average coeficient signed-digits
tol=5e-05 % Tolerance on coef. update
maxiter=2000 % SOCP iteration limit
npoints=250 % Frequency points across the band
length(c0)=21 % Num. tap coefficients
sum(k0~=0)=10 % Num. non-zero all-pass coef.s
rho=0.992188 % Constraint on allpass coefficients
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftpl=0.09 % Delay pass band lower edge
ftpup=0.21 % Delay pass band upper edge
tp=16 % Nominal passband filter group delay
tpr=0.2 % Delay pass band peak-to-peak ripple
Wtp=6 % Delay pass band weight
fasl=0.05 % Amplitude stop band(1) lower edge
fasu=0.25 % Amplitude stop band(1) upper edge
dBas=33 % Amplitude stop band(1) peak-to-peak ripple
fasll=0.04 % Amplitude stop band(2) lower edge
fasuu=0.26 % Amplitude stop band(2) upper edge
dBass=36 % Amplitude stop band(2) peak-to-peak ripple
Wasl=100000 % Amplitude lower stop band weight
Wasu=1000000 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 10 bits allocated with an average of 3 signed-digits by the heuristic of *Ito et al.* as shown in Section 13.2. At each coefficient relaxation step the script finds the upper and lower signed-digit approximations to the current set of active coefficients, selects the coefficient with the largest difference in those approximations and performs two optimisations. The first is an SOCP PCLS optimisation of the response with the current set of active coefficients. The second optimisation calls *SparsePOP* with a 2nd order polynomial equality constraint requiring that the previously selected coefficient be equal to either the signed-digit coefficient upper or lower bound. The remaining active coefficients are allowed to vary within the corresponding signed-digit upper and lower bounds. *SparsePOP* failed when I attempted to apply linear constraints on the response for PCLS optimisation (see *schurOneMlattice_pop_socp_mmse_test.m*).

The filter coefficients found by the SparsePOP relaxation search are:

```
512*k_min = [      0,      336,       0,      264, ...
                0,      176,       0,      216, ...
                0,      146,       0,      127, ...
                0,       74,       0,       52, ...
                0,       17,       0,        8 ]';
```

```

512*c_min = [      40,      -2,     -144,     -256, ...
               -94,      48,     193,     160, ...
                15,     -40,     -40,      -8, ...
               -2,     -16,     -12,       1, ...
               12,       8,       0,      -1, ...
                2 ] ';

```

Table 14.12 compares the cost, the number of signed-digits and the number of 10-bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* and optimised by POP-relaxation search. Figure 14.28 compares the pass-band responses of the filter with exact and 10 bit coefficients allocated an average of 3 signed-digits found by the heuristic of *Ito et al.* and optimised by POP-relaxation search. Similarly, Figure 14.29 shows the filter stop-band response and Figure 14.30 shows the filter pass-band group delay response.

	Cost	Signed-digits	Shift-and-adds
Exact	0.0138		
10-bit 3-signed-digit(Ito)	0.0838	62	31
10-bit 3-signed-digit(POP-relax)	0.0236	58	28

Table 14.12: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing POP-relaxation search.

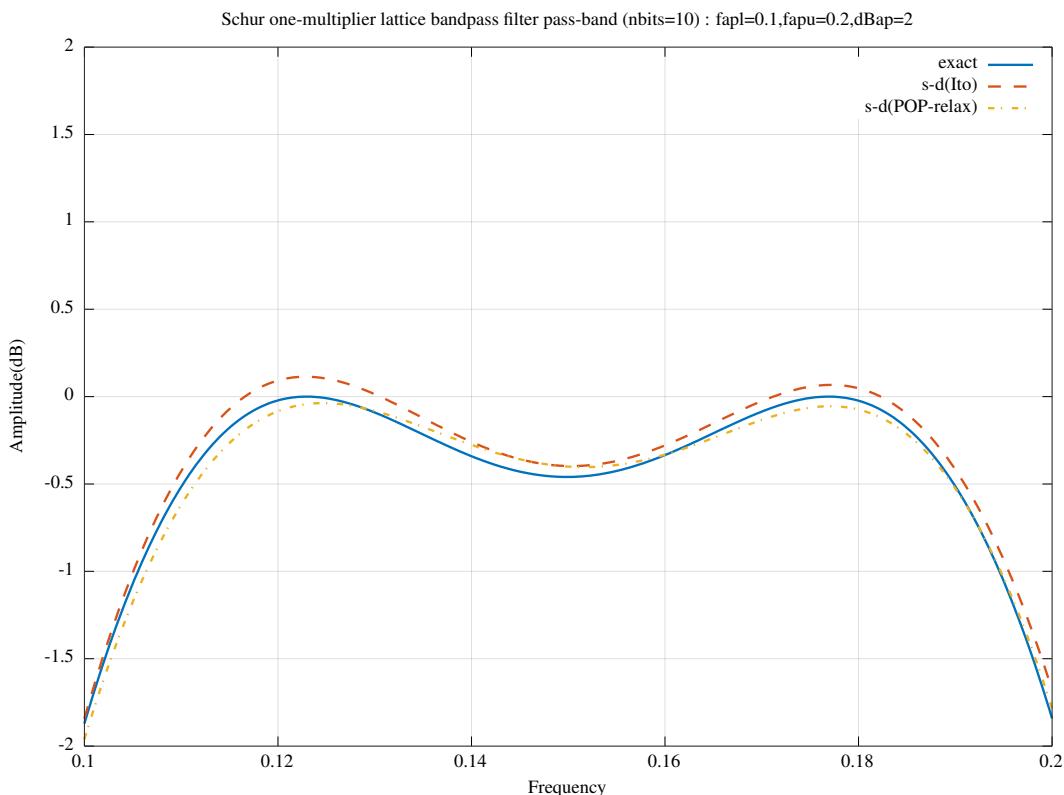


Figure 14.28: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing POP-relaxation search

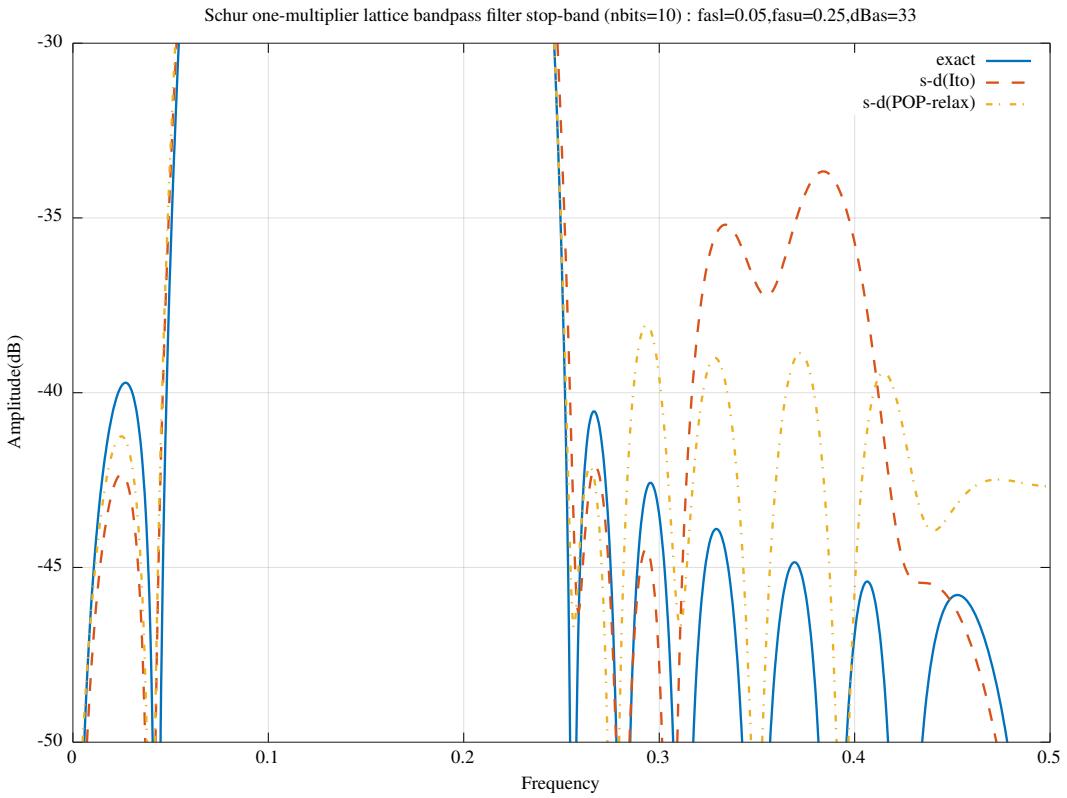


Figure 14.29: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing POP-relaxation search

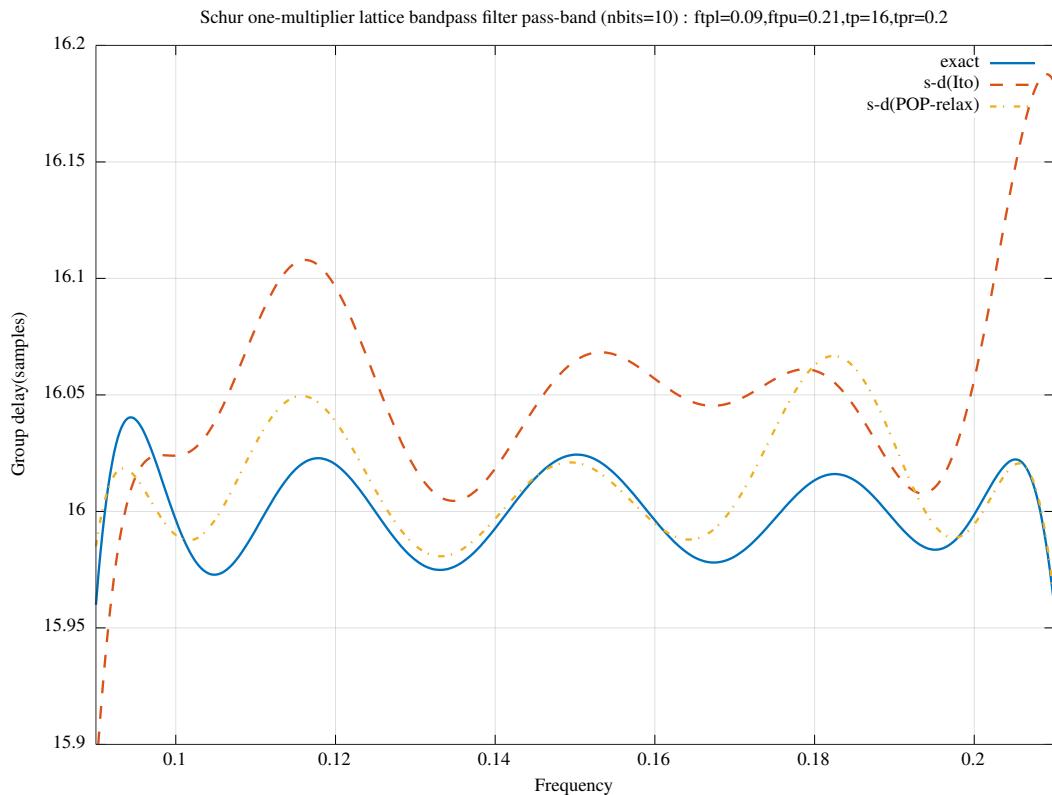


Figure 14.30: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 10 bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Ito et al.* and performing POP-relaxation search

14.4.6 SOCP-relaxation search for the signed-digit coefficients of an FIR lattice Gaussian filter

The Gaussian function in the continuous angular frequency domain is:

$$G(\omega) = e^{-(\frac{\omega}{\alpha})^2}$$

Appendix K shows that the corresponding time domain Fourier transform pair function is:

$$g(t) = \frac{\alpha}{2\sqrt{\pi}} e^{-(\frac{t\alpha}{2})^2}$$

If the symbol interval is T_S then the normalised one-sided half-power bandwidth-symbol time product, BT_S , is given by:

$$e^{-(\frac{2\pi BT_S}{T_S \alpha})^2} = \frac{1}{\sqrt{2}}$$

so:

$$\alpha = \frac{2\pi BT_S}{T_S} \sqrt{\frac{2}{\ln 2}}$$

Assume that the Gaussian filter has a length of N_S symbols and an over-sampling rate of R samples per symbol. After shifting and truncation the sampled filter coefficients of an odd length filter are:

$$g(k) = \frac{T_S}{R} \frac{1}{2\sqrt{\pi}} \frac{2\pi BT_S}{T_S} \sqrt{\frac{2}{\ln 2}} e^{-\left[\left(\frac{\pi BT_S}{T_S} \sqrt{\frac{2}{\ln 2}}\right) \frac{T_S}{R} \left(k - \frac{RN_S}{2}\right)\right]^2}$$

where $k = 0 \dots RN_S$. Assume that the modulation scheme uses $BT_S = 0.3$, that the Gaussian filter has a length of $N_S = 4$ symbols and that the over-sampling rate is $R = 8$ samples per symbol. The filter coefficients are:

```
g0= = [ 0.000003984, 0.000013788, 0.000044042, 0.000129853, ...
0.000353389, 0.000887708, 0.002058274, 0.004405067, ...
0.008701970, 0.015867136, 0.026705172, 0.041486623, ...
0.059488999, 0.078737407, 0.096192548, 0.108471994, ...
0.112904093, 0.108471994, 0.096192548, 0.078737407, ...
0.059488999, 0.041486623, 0.026705172, 0.015867136, ...
0.008701970, 0.004405067, 0.002058274, 0.000887708, ...
0.000353389, 0.000129853, 0.000044042, 0.000013788, ...
0.000003984 ]';
```

Figure 14.31 shows the impulse response of the filter.

The Octave script *socp_relaxation_gaussian_FIR_lattice_16_nbts_test.m* performs SOCP relaxation to find the 16-bit coefficients of the Gaussian filter implemented as a complementary FIR lattice. (See Appendix H for a description of the complementary FIR lattice filter). The response of this filter is compared with filters using 16-bit 3-signed-digit coefficients of the direct form implementation and the 16-bit signed-digit coefficients of the lattice implementation. The coefficients of the lattice filters are allocated an average of 3 signed-digits with the algorithm of Lim *et al.* shown in Section 13.2. The script does not implement signed-digit allocation or SOCP-relaxation optimisation of the direct-form coefficients. The filter specification is:

```
BTs=0.3 % Bandwidth-Symbol-rate product
Ns=4 % Filter width in symbols
R=8 % Samples-per-symbol
tol=1e-08 % Tolerance on coefficient update vector
nplot=1024 % Frequency points across the band
dBap=0.175 % Amplitude pass band peak-to-peak ripple
dBas=50 % Amplitude stop band peak-to-peak ripple
dBasu=63 % Amplitude upper stop band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Was=1e+06 % Amplitude stop band weight
tp=16 % Nominal pass band filter group delay (samples)
tpr=0.4 % Delay pass band peak-to-peak ripple (rad.)
Wtp=0.01 % Delay pass band weight
```

The amplitude response in the “pass-band” (up to $dBas$) found by SOCP-relaxation is required to be within $\pm \frac{dBap}{2}$ of the exact response and below $dBasu$ in the “stop-band”.

The 16-bit 3-signed-digit direct form coefficients are:

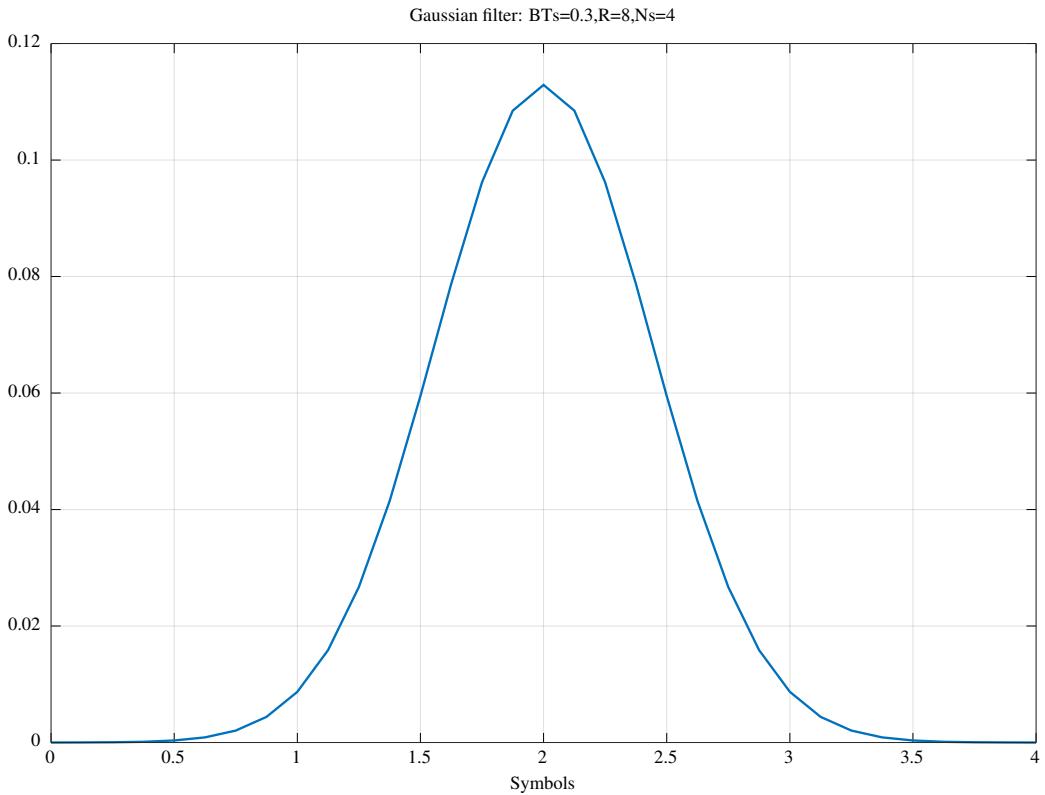


Figure 14.31: Impulse response of a sampled Gaussian filter with half-power bandwidth-symbol-interval product $BT_S = 0.3$, length $N_S = 4$ symbols and $R = 8$ samples per symbol

```
32768*g0_sd = [
    0,      0,      1,      4, ...
   12,     29,     67,    144, ...
  284,    520,    880,   1344, ...
 1952,   2576,   3136,  3552, ...
 3712,   3552,   3136,  2576, ...
 1952,   1344,   880,   520, ...
 284,    144,    67,    29, ...
 12,     4,      1,      0, ...
 0 ]';
```

The exact lattice coefficients are:

```
k0= [ 1.000000000, 1.000000000, 0.999999999, 0.999999987, ...
 0.999999904, 0.999999379, 0.999996554, 0.999983626, ...
 0.999933289, 0.999766665, 0.999298569, 0.998187370, ...
 0.995980833, 0.992394921, 0.987837620, 0.983761807, ...
 0.982093942, 0.983761807, 0.987837620, 0.992394921, ...
 0.995980833, 0.998187370, 0.999298569, 0.999766665, ...
 0.999933289, 0.999983626, 0.999996554, 0.999999379, ...
 0.999999904, 0.999999987, 0.999999999, 1.000000000, ...
 0.000004422 ]';
```

```
khat0= [ -0.000004422, -0.000016216, -0.000053022, -0.000158639, ...
 -0.000437458, -0.001114397, -0.002625192, -0.005722584, ...
 -0.011550656, -0.021601291, -0.037448233, -0.060182849, ...
 -0.089566627, -0.123094763, -0.155489023, -0.179478986, ...
 -0.188391851, -0.179478986, -0.155489023, -0.123094763, ...
 -0.089566627, -0.060182849, -0.037448233, -0.021601291, ...
 -0.011550656, -0.005722584, -0.002625192, -0.001114397, ...
 -0.000437458, -0.000158639, -0.000053022, -0.000016216, ...
 1.000000000 ]';
```

The 16-bit lattice coefficients with an average of 3-signed-digits each are:

```

32768*k0_sd = [ 32768, 32768, 32768, 32768, ...
                  32768, 32768, 32768, 32767, ...
                  32766, 32760, 32745, 32709, ...
                  32636, 32519, 32368, 32236, ...
                  32180, 32236, 32368, 32519, ...
                  32636, 32709, 32745, 32760, ...
                  32766, 32767, 32768, 32768, ...
                  32768, 32768, 32768, 32768, ...
                  0 ]';


```

```

32768*khat0_sd = [      0,      0,      0,     -4, ...
                     -14,    -32,    -80,   -188, ...
                     -376,   -704,  -1224, -1972, ...
                     -2936,  -4034, -5096, -5880, ...
                     -6173,  -5880, -5096, -4034, ...
                     -2936,  -1972, -1224,  -704, ...
                     -376,   -188,   -80,   -32, ...
                     -14,    -4,      0,      0, ...
                     32768 ]';


```

The 16-bit lattice coefficients with an average of 3-signed-digits optimised by SOCP-relaxation are:

```

32768*k_min = [ 32768, 32768, 32768, 32768, ...
                  32768, 32768, 32768, 32767, ...
                  32766, 32760, 32745, 32709, ...
                  32636, 32519, 32368, 32236, ...
                  32182, 32236, 32368, 32519, ...
                  32636, 32709, 32745, 32760, ...
                  32766, 32767, 32768, 32768, ...
                  32768, 32768, 32768, 32768, ...
                  0 ]';


```

```

32768*khat_min = [      0,      0,      0,     -4, ...
                     -15,    -32,    -80,   -188, ...
                     -376,   -704,  -1224, -1972, ...
                     -2936,  -4032, -5096, -5880, ...
                     -6174,  -5880, -5096, -4033, ...
                     -2936,  -1972, -1224,  -704, ...
                     -376,   -184,   -80,   -32, ...
                     -14,    -4,      0,      0, ...
                     32768 ]';


```

Table 14.13 compares the total number of signed-digits and 16-bit shift-and-add operations required to implement the coefficient multiplications. The values for the direct form filter assume that the implementation makes use of the symmetry of the impulse response, in which case the direct form filter has 17 multipliers compared to the 130 required by the lattice implementation (though a number of the latter multipliers are 1 or 0).

	Signed-digits	Shift-and-adds
16-bit 3-signed-digit(direct-folded)	73	44
16-bit 3-signed-digit(lattice)	158	98
16-bit 3-signed-digit(SOCP-relax)	156	96

Table 14.13: Comparison of the number of 16-bit shift-and-add operations required to implement the coefficient multiplications for a direct form and a complementary FIR lattice implementation of a Gaussian filter with 16-bit 3-signed-digit coefficients.

Figure 14.32 compares the overall amplitude response of the filter for exact coefficients, direct form coefficients implemented with 16-bit 3-signed-digits, complementary FIR lattice coefficients implemented with 16-bit, 3-signed-digits having an average of 3 signed-digits per coefficient allocated with the algorithm of Lim *et al.* and lattice coefficients optimised with SOCP-relaxation. Figure 14.33 compares the amplitude responses in the “pass-band” (up to d_{Bas}). Figure 14.34 compares the group delay responses in the “pass-band” (up to d_{Bas}).

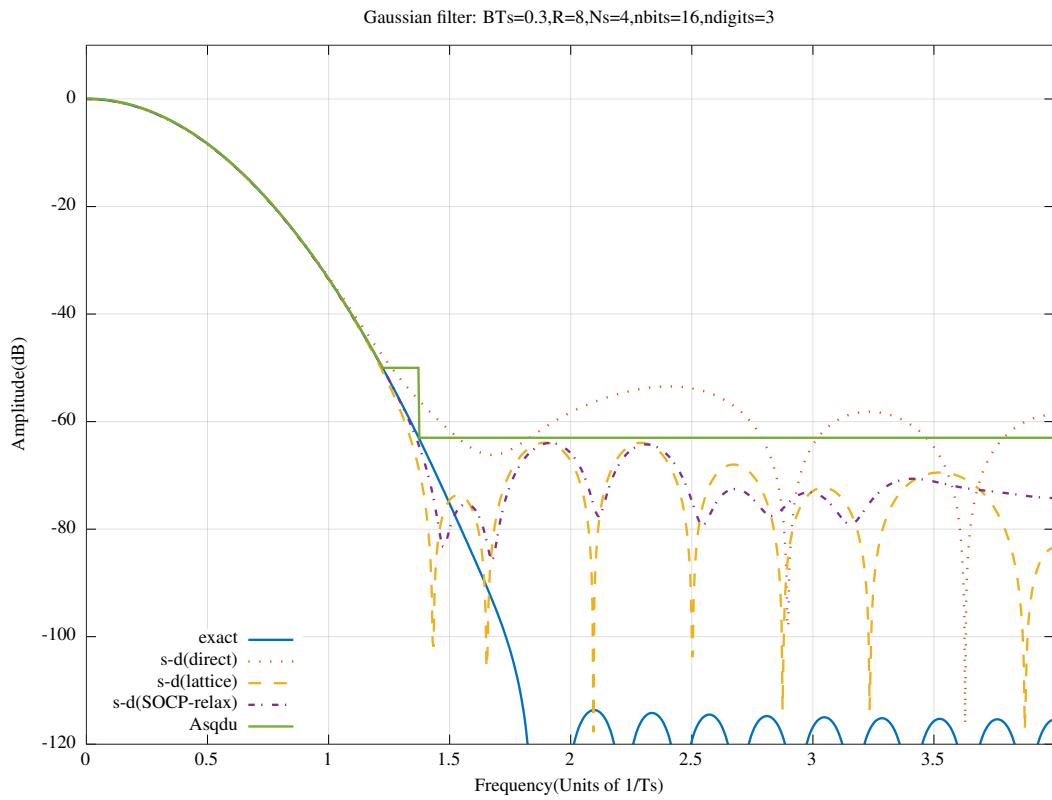


Figure 14.32: Comparison of the amplitude response of the FIR Gaussian filter for exact direct form coefficients, 16-bit 3-signed-digit direct form coefficients, 16-bit coefficients with an average of 3-signed-digits per coefficient and 16-bit coefficients with an average of 3-signed-digits per coefficient optimised by SOCP-relaxation.

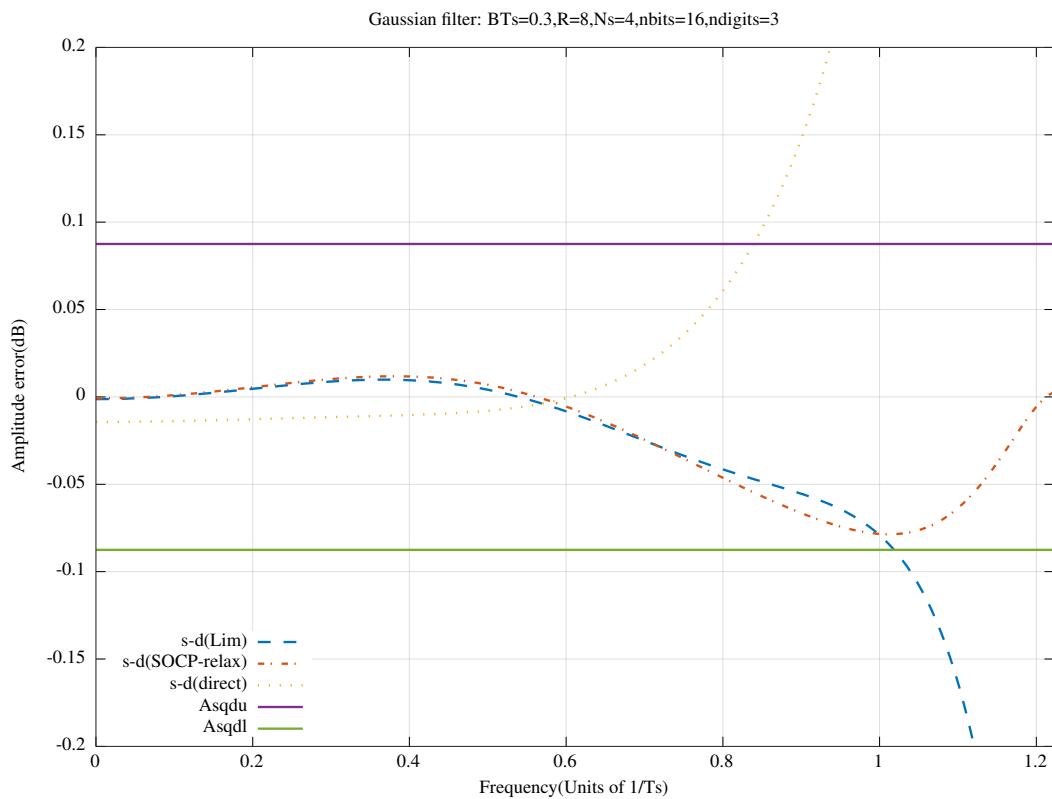


Figure 14.33: Comparison of the pass-band amplitude response of the FIR Gaussian filter for 16-bit coefficients with an average of 3-signed-digits per coefficient and 16-bit coefficients with an average of 3-signed-digits per coefficient optimised by SOCP-relaxation.

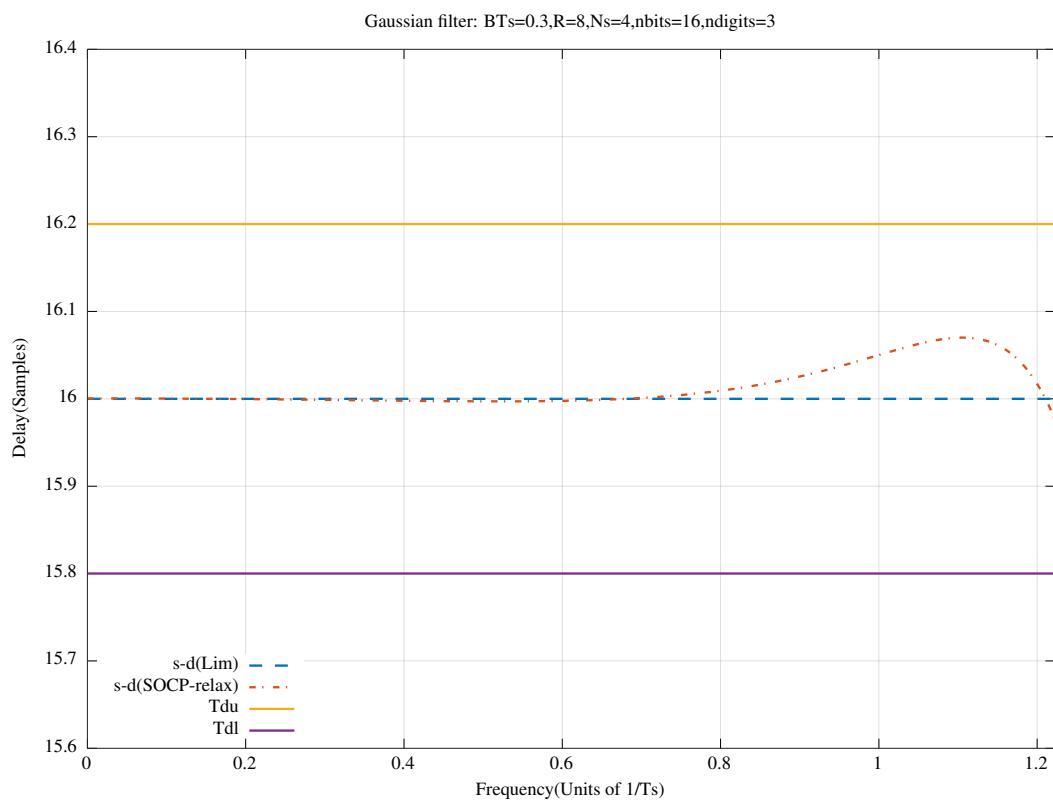


Figure 14.34: Comparison of the “pass-band” (up to d_{BAS}) group-delay response of the FIR Gaussian filter for 16-bit coefficients with an average of 3-signed-digits per coefficient and 16-bit coefficients with an average of 3-signed-digits per coefficient optimised by SOCP-relaxation.

Chapter 15

Comparison of filter coefficient search methods for a 5th order elliptic filter with 6-bit integer and 2-signed-digit coefficients

The signed-digit allocation algorithms shown in Chapter 13 performed poorly when the coefficient word-length is less than 10. This chapter compares the results of “brute force” search for the 6-bit integer coefficients of a 5th order elliptic filter. The filter responses are compared to an ideal, “brick-wall”, amplitude response:

$$A(f) = \begin{cases} 1 & f \leq 0.125 \\ 0 & f > 0.150 \end{cases}$$

The response errors are weighted as follows:

$$W_a(f) = \begin{cases} 1 & f \leq 0.125 \\ 0.1 & 0.125 < f < 0.150 \\ 10 & f \geq 0.150 \end{cases}$$

The prototype filter is a 5-th order elliptic filter with pass-band edge at $f_{pass} = 0.125$, 1dB pass-band ripple and 40dB stop-band ripple:

```
% Specify elliptic low pass filter
norder=5; dBpass=1; dBstop=40; fpass=0.125;
[n0, d0]=ellip(norder, dBpass, dBstop, 2*fpass);
```

The prototype filter is implemented as a normalised-scaled lattice filter, parallel allpass normalised-scaled lattice filters, a one-multiplier lattice filter, parallel all-pass one-multiplier lattice filters and a cascade of two 2nd order minimum noise state variable sections followed by a first order section. The stability of each filter is maintained by checking the pole locations of the transfer function denominator polynomial in the corresponding cost function. For each optimisation algorithm and filter, I compare the cost function of the exact filter response to the response for each filter with the coefficients truncated as 6 bit rounded and 6 bit 2 signed-digits. I anticipate that the average number of signed digits used by the signed-digit coefficients will be less than 2. The 6 bit rounded coefficients can be represented exactly by 3 or fewer signed digits. Individual cases may be improved by “tweaking” the optimisation parameters.

15.1 Searching with the bit-flipping algorithm

The Octave script *bitflip_NS_lattice_test.m* implements the prototype elliptic filter as a normalised-scaled lattice and optimises the truncated coefficients with the bit-flipping algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 15.1 and 15.2 show the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

The Octave script *bitflip_OneM_lattice_test.m* implements the prototype elliptic filter as a one-multiplier lattice and optimises the truncated coefficients with the bit-flipping algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 15.3 and 15.4 show the overall and passband responses of the filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm. The one-multiplier lattice state scaling coefficients are not truncated.

Appendix G shows how odd-order “classical” digital filter transfer functions can be implemented as the sum of two parallel all-pass filters. The Octave script *bitflip_NSPA_lattice_test.m* implements the 5th order elliptic filter as the sum of two normalised-scaled all-pass lattice filters and optimises the truncated coefficients with the bit-flipping algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 15.5 and 15.6 show the overall and passband responses of the parallel all-pass filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients and 2 signed-digit coefficients optimised with the bit-flipping algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

The Octave script *bitflip_OneMPA_lattice_test.m* implements the 5th order elliptic filter as the sum of two one-multiplier all-pass lattice filters and optimises the truncated coefficients with the bit-flipping algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 15.7 and 15.8 show the overall and passband responses of the parallel all-pass filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm. The one-multiplier lattice state scaling coefficients are not truncated. The structural boundedness of the parallel all-pass one-multiplier lattice filter is evident.

Finally, the Octave script *bitflip_svcasc_test.m* implements the 5th order elliptic filter as a pair of 2nd order minimum-noise state variable sections followed by a 1st order state variable section. (See Section 6). The script optimises the truncated coefficients with the bit-flipping algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 15.9 and 15.10 show the overall and passband responses of the filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm. Note that the 2nd order state variable cascade filter obtained by simply converting the exact coefficients to 6 bit 2 signed-digit coefficients is unstable.

Table 15.1 compares the cost result for each test of the bit-flipping algorithm.

Cost	Normalised-scaled lattice	One-multiplier lattice	Normalised-scaled PA lattice	One-multiplier PA lattice	2nd Order cascade
Exact	1.0008	1.0008	1.0008	1.0008	1.0008
Rounded	1.5432	1.5299	1.2459	1.1097	1.2362
Rounded with bit-flipping	1.1289	1.5299	0.9861	1.1097	0.9929
Signed-digit	2.3373	2.7157	3.6059	3.9013	∞
Signed-digit with bit-flipping	0.8744	2.1571	1.3824	3.1746	1.1334

Table 15.1: Summary of the cost results for the example 5th order elliptic low-pass filter synthesised as a normalised-scaled or one-multiplier lattice filter or as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

The normalised-scaled 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm are:

```
s10_bfsd = [ 0.968750, 0.875000, 0.468750, 0.187500, 0.031250 ]';
s11_bfsd = [ 0.000000, 0.875000, 0.937500, 1.000000, 0.468750 ]';
s20_bfsd = [ -0.750000, 0.937500, -0.875000, 0.750000, -0.468750 ]';
s00_bfsd = [ 0.625000, 0.312500, 0.468750, 0.562500, 0.875000 ]';
s02_bfsd = [ 0.750000, -0.968750, 0.875000, -0.750000, 0.437500 ]';
s22_bfsd = [ 0.625000, 0.218750, 0.468750, 0.500000, 0.875000 ]';
```

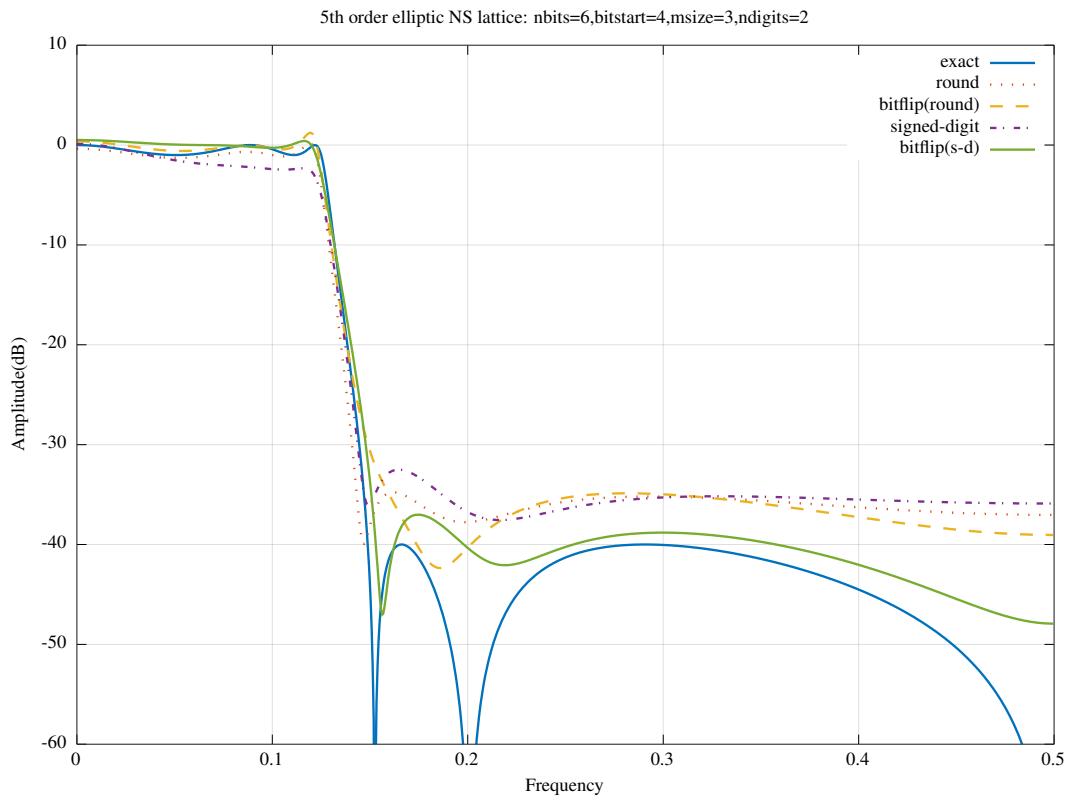


Figure 15.1: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

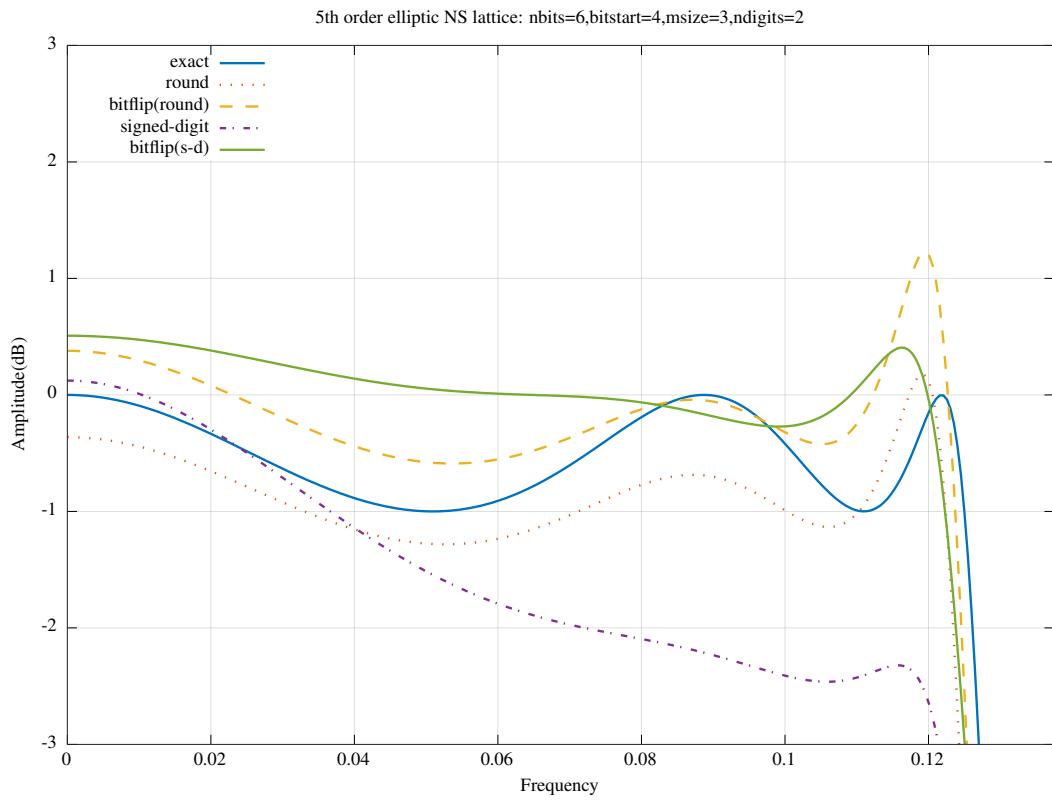


Figure 15.2: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

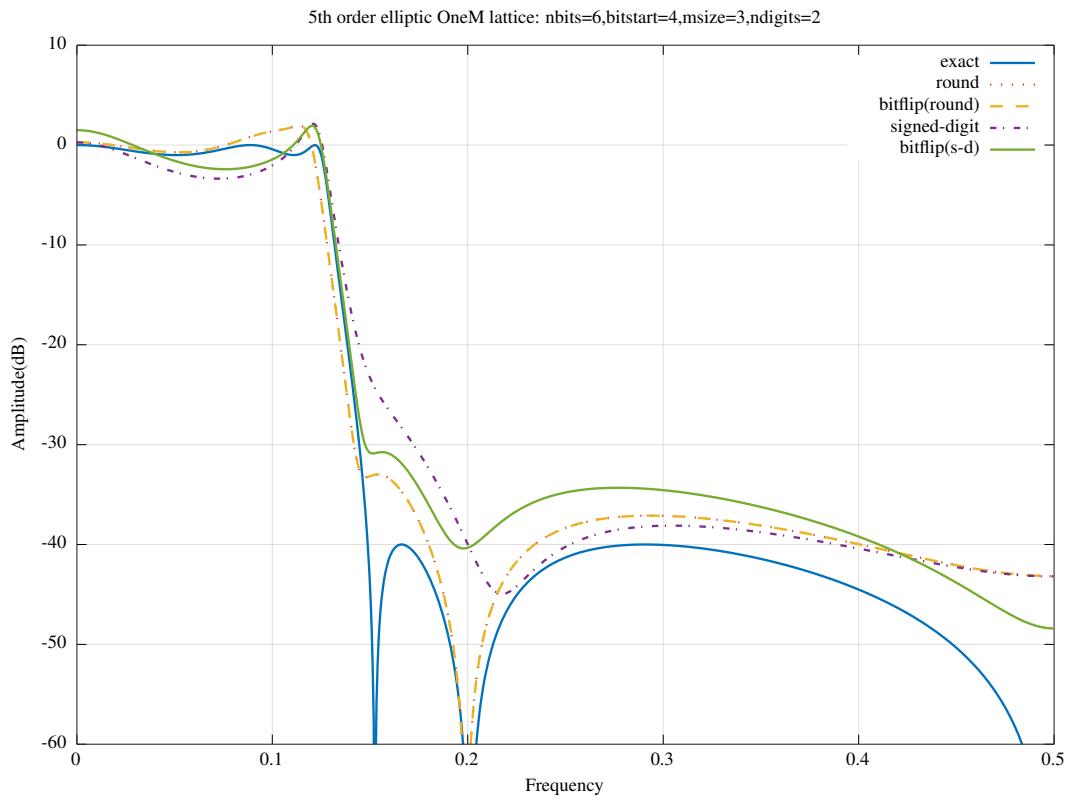


Figure 15.3: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

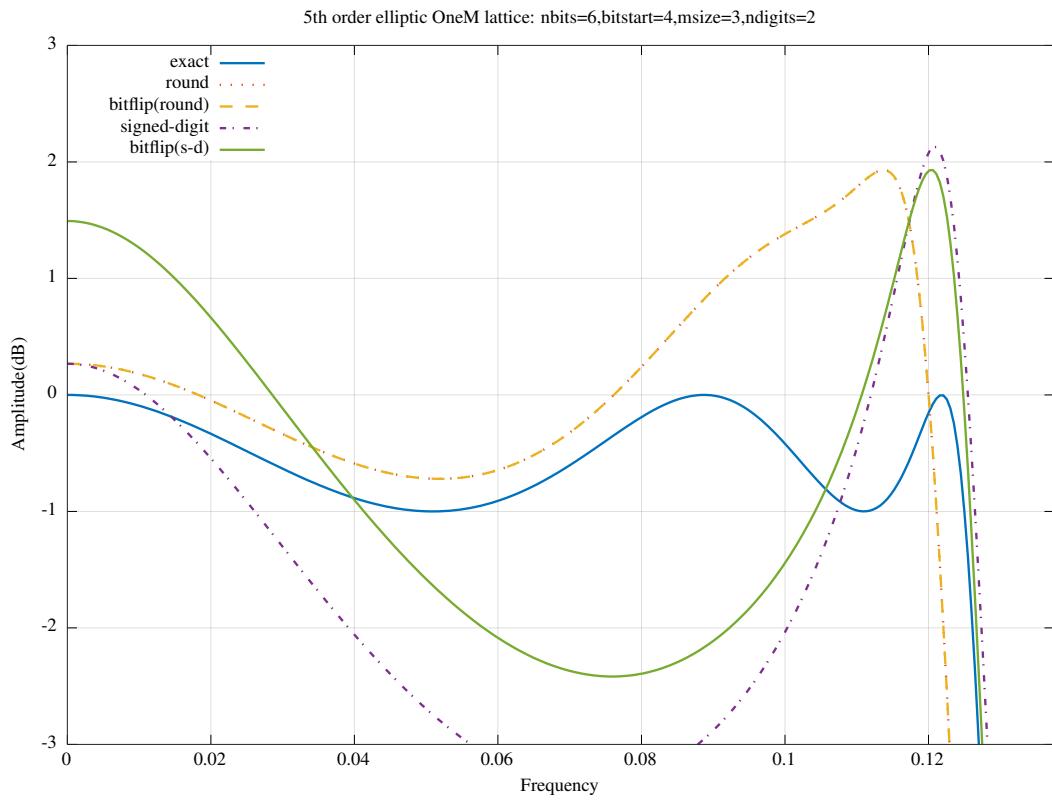


Figure 15.4: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

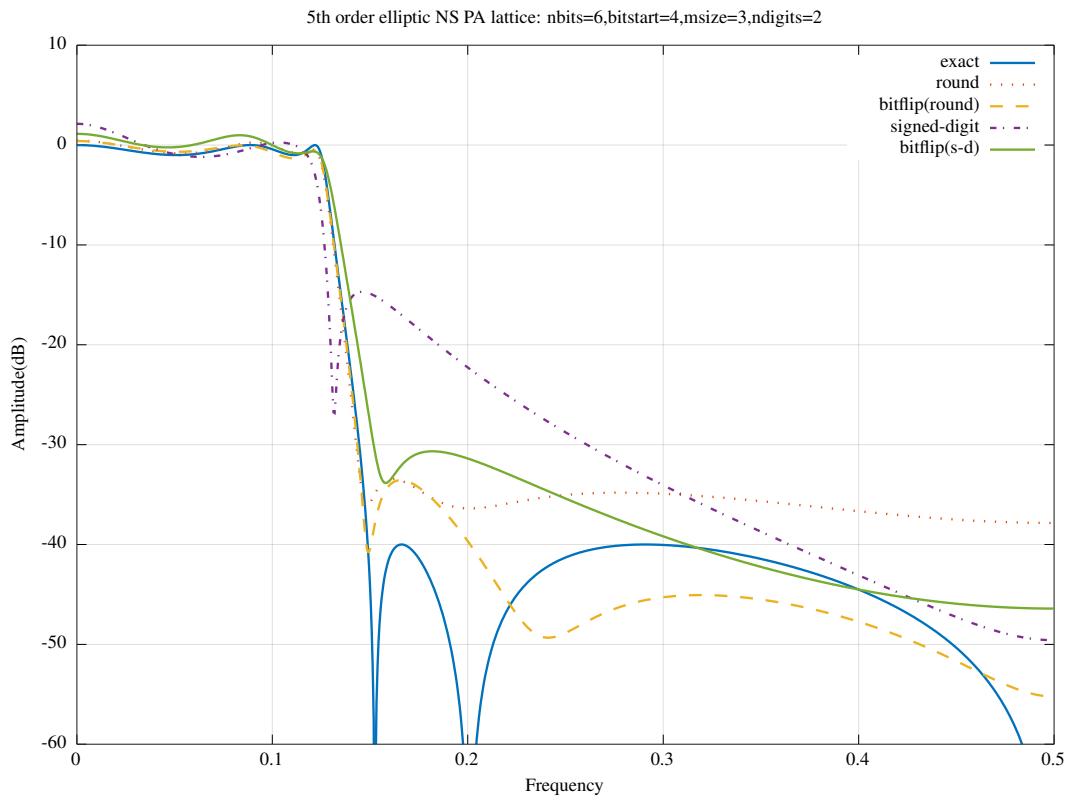


Figure 15.5: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

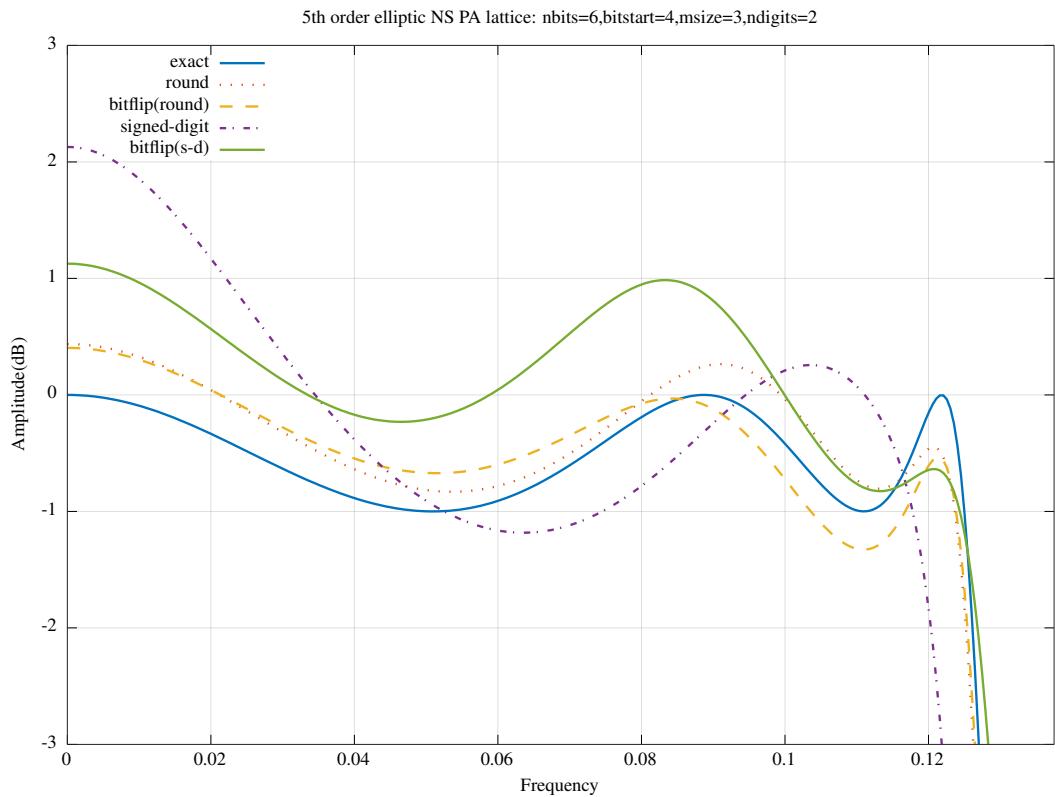


Figure 15.6: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

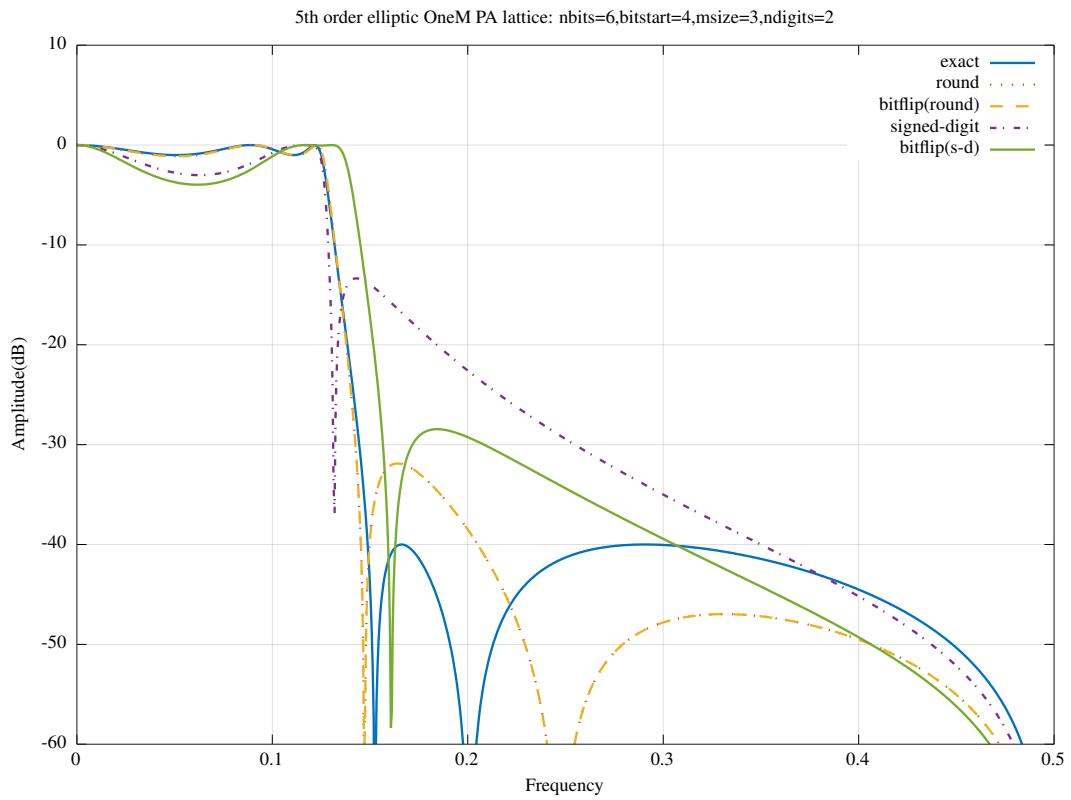


Figure 15.7: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

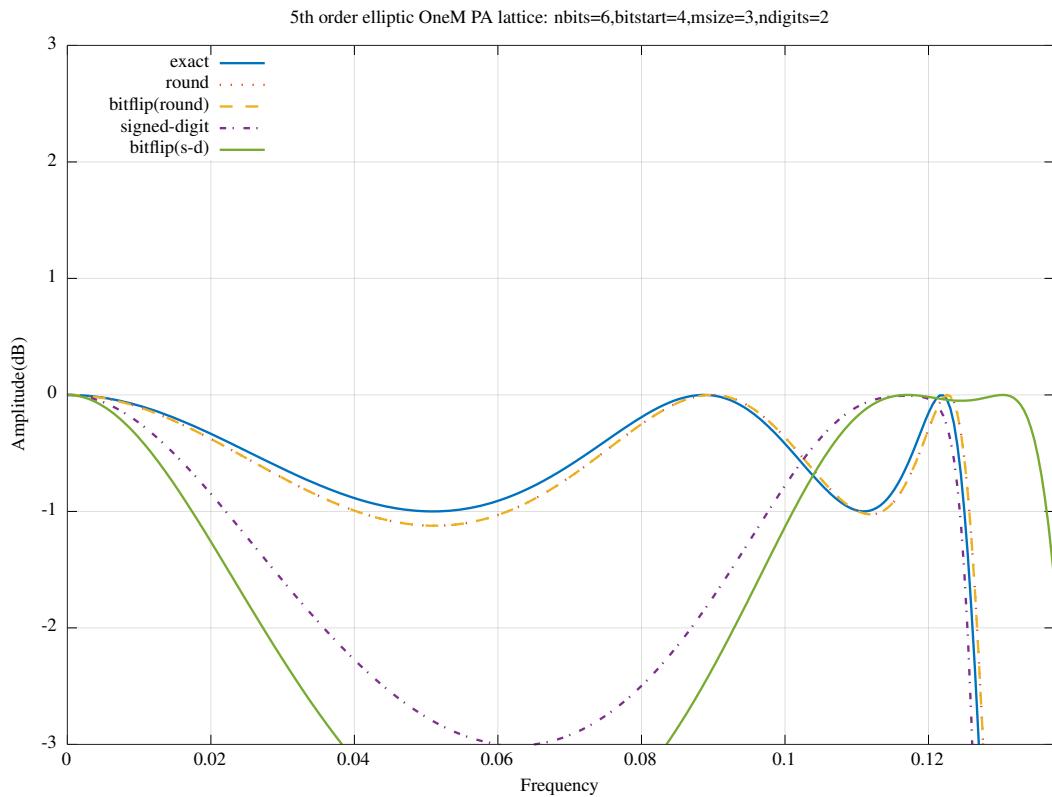


Figure 15.8: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

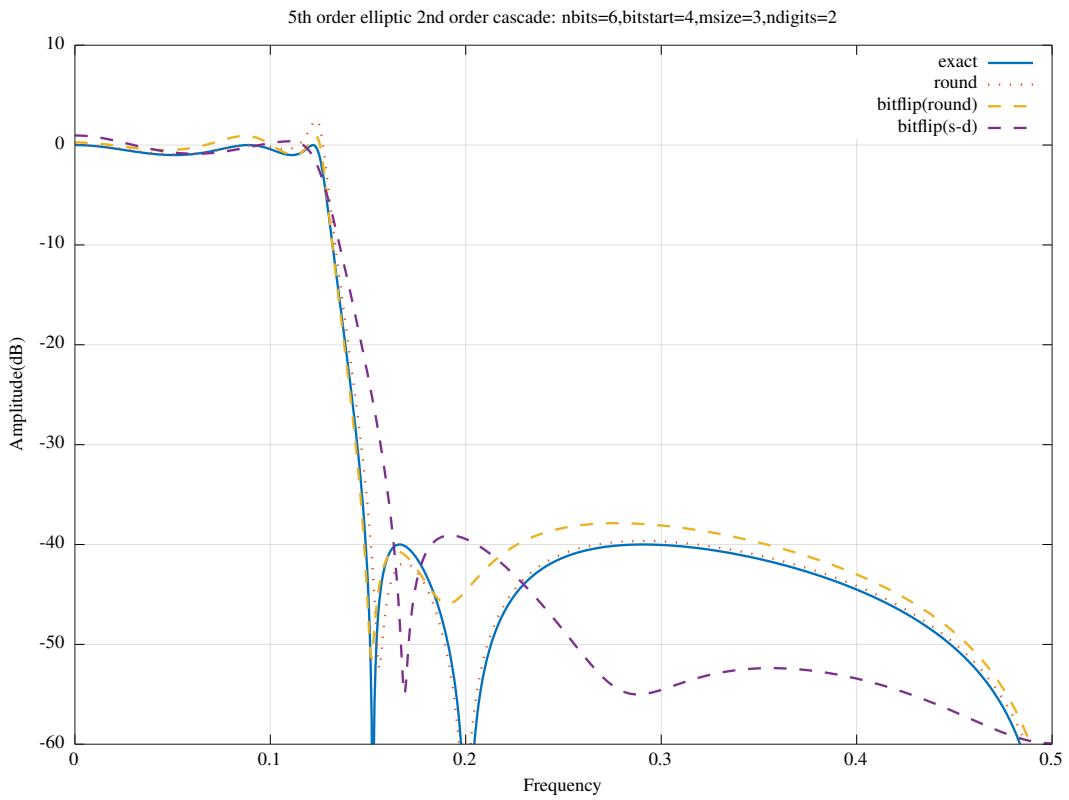


Figure 15.9: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

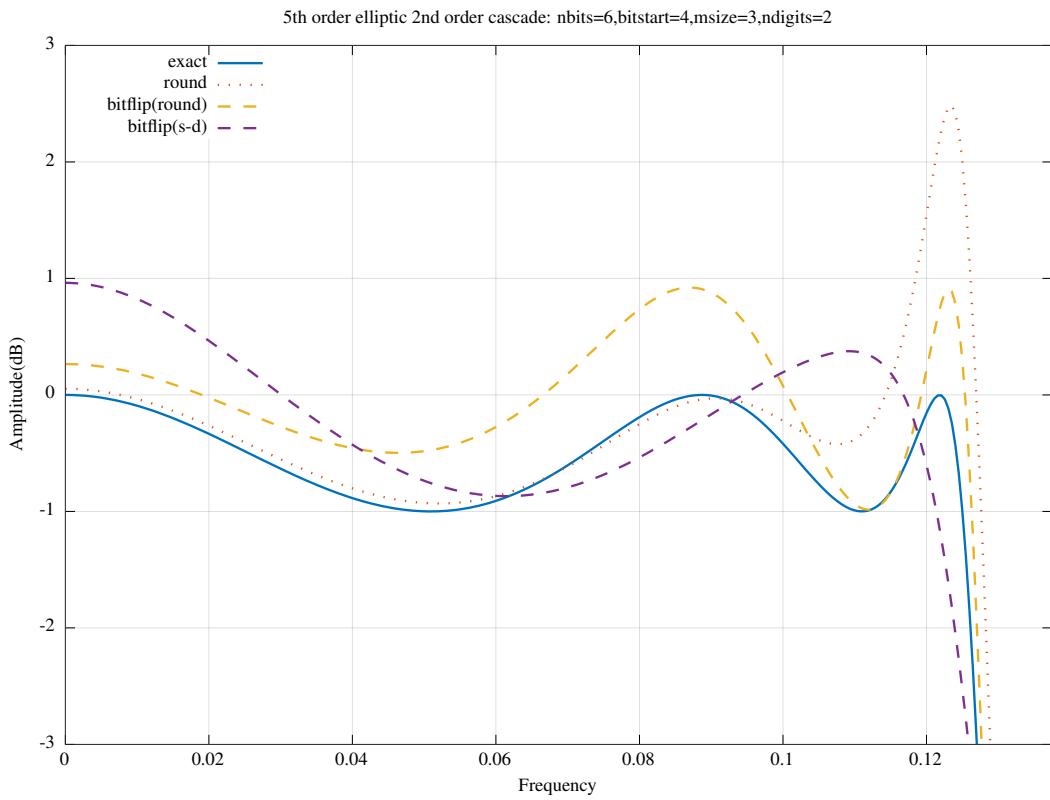


Figure 15.10: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

15.2 Searching with the Nelder-Mead simplex algorithm

The Octave script *simplex_NS_lattice_test.m* implements the prototype elliptic filter as a normalised-scaled lattice and optimises the truncated coefficients with the *nelder_mead_min* simplex algorithm from the Octave-Forge *optim* package [64]. Figures 15.11 and 15.12 show the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

The Octave script *simplex_OneM_lattice_test.m* implements the prototype elliptic filter as a one-multiplier lattice and optimises the truncated coefficients with the simplex algorithm. Figures 15.13 and 15.14 show the overall and passband responses of the filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm. The one-multiplier lattice state scaling coefficients are not truncated.

The Octave script *simplex_NS_PA_lattice_test.m* implements the 5th order elliptic filter as the sum of two normalised-scaled all-pass lattice filters and optimises the truncated coefficients with the simplex algorithm. Figures 15.15 and 15.16 show the overall and passband responses of the parallel all-pass filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

The Octave script *simplex_OneMPA_lattice_test.m* implements the 5th order elliptic filter as the sum of two one-multiplier all-pass lattice filters and optimises the truncated coefficients with the simplex algorithm. Figures 15.17 and 15.18 show the overall and passband responses of the parallel all-pass filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm. The one-multiplier lattice state scaling coefficients are not truncated.

Finally, the Octave script *simplex_svccasc_test.m* implements the 5th order elliptic filter as a pair of 2nd order minimum-noise state variable sections followed by a 1st order state variable section. (See Section 6). The script optimises the truncated coefficients with the simplex algorithm. Figures 15.19 and 15.20 show the overall and passband responses of the filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm. Note that the 2nd order state variable cascade filter obtained by simply converting the exact coefficients to 6 bit 2 signed-digit coefficients is unstable.

Table 15.2 compares the cost result for each test of the simplex algorithm.

Cost	Normalised-scaled lattice	One-multiplier lattice	Normalised-scaled PA lattice	One-multiplier PA lattice	2nd Order cascade
Exact	1.0008	1.0008	1.0008	1.0008	1.0008
Rounded	1.5432	1.5299	1.2459	1.1097	1.2362
Rounded with simplex	1.0397	1.4617	0.9861	1.1097	0.8395
Signed-digit	2.3373	2.7157	3.6059	3.9013	∞
Signed-digit with simplex	1.6581	2.3918	3.2559	3.1746	3.2134

Table 15.2: Summary of the cost results for the example 5th order elliptic low-pass filter synthesised as a normalised-scaled or one-multiplier lattice filter or as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

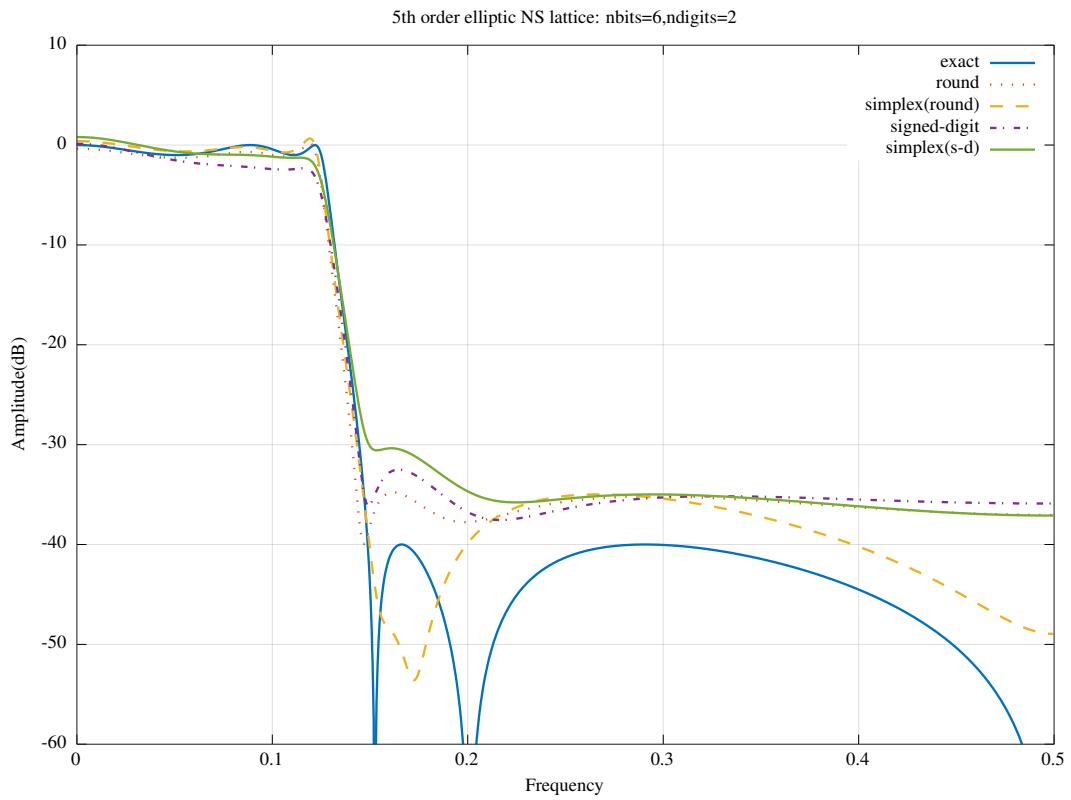


Figure 15.11: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

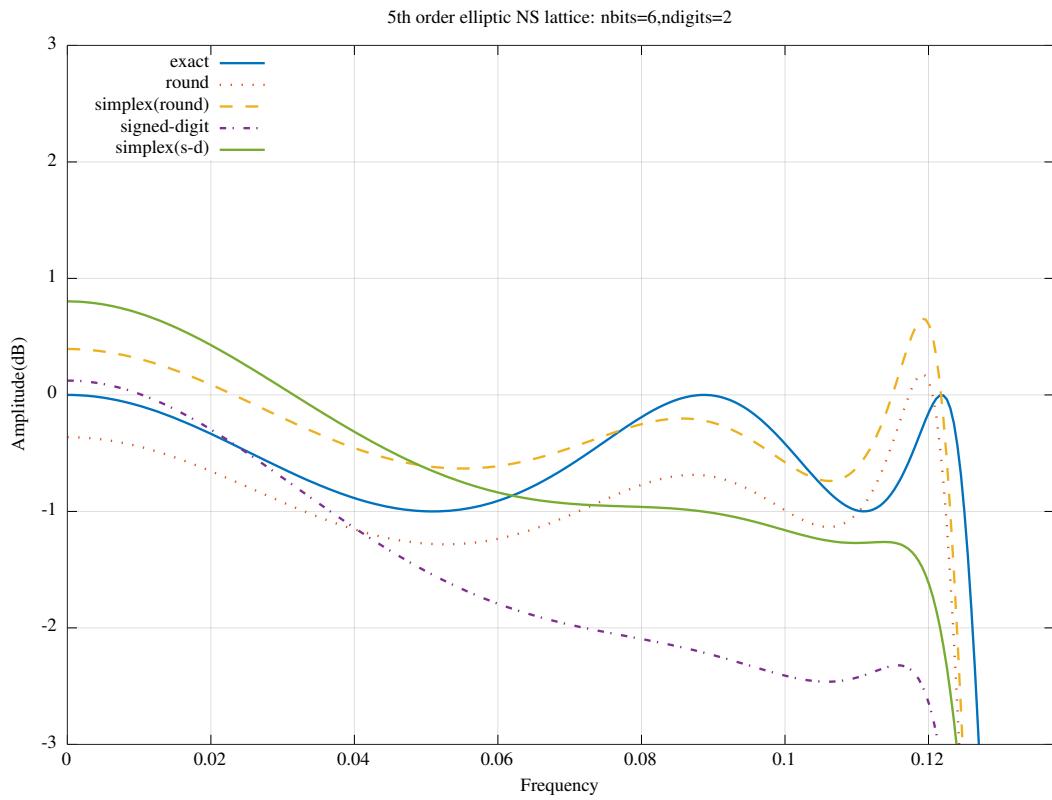


Figure 15.12: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

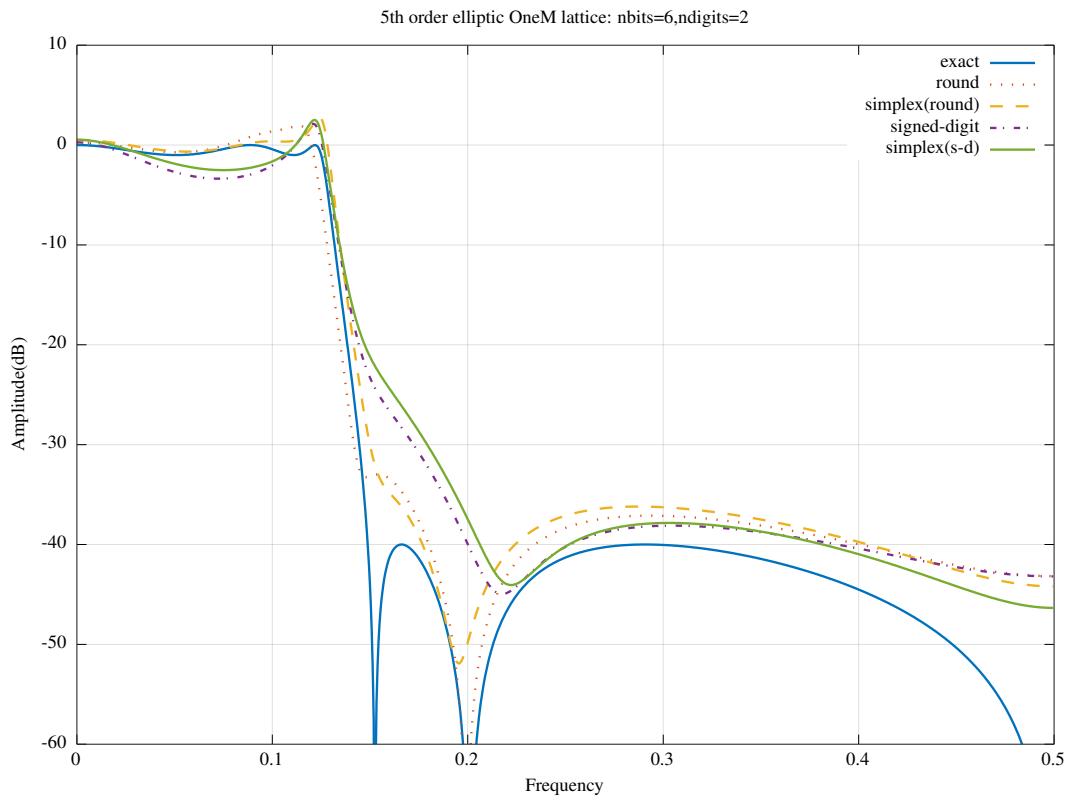


Figure 15.13: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

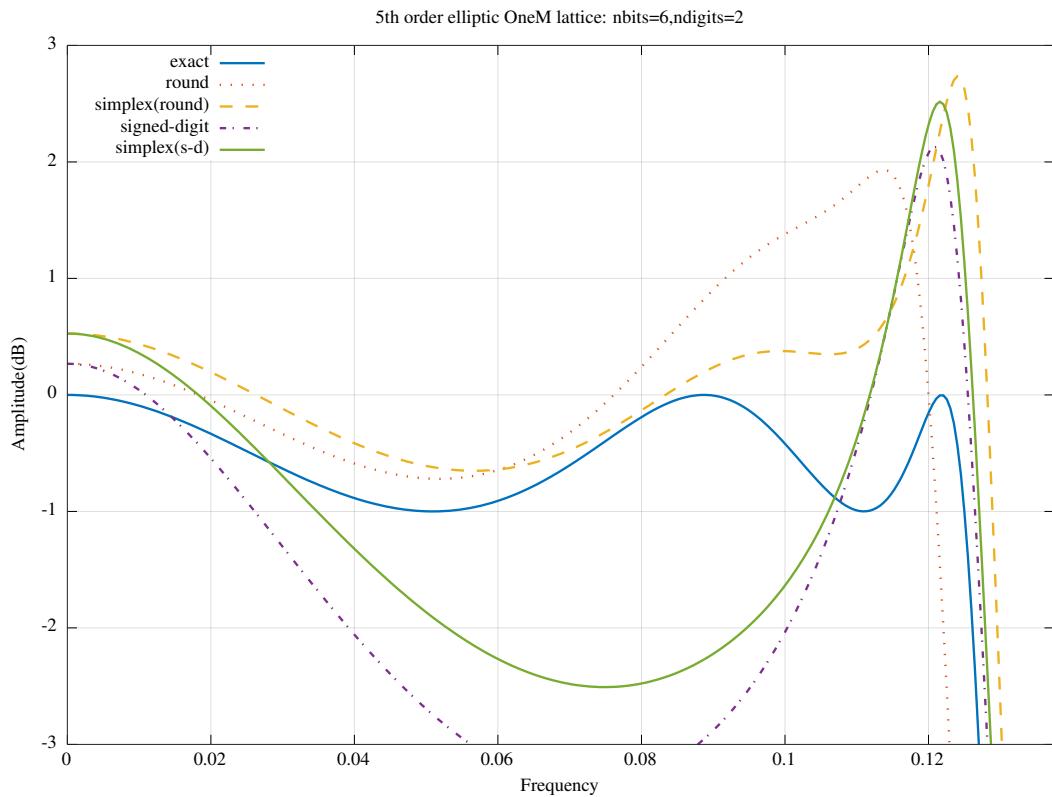


Figure 15.14: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

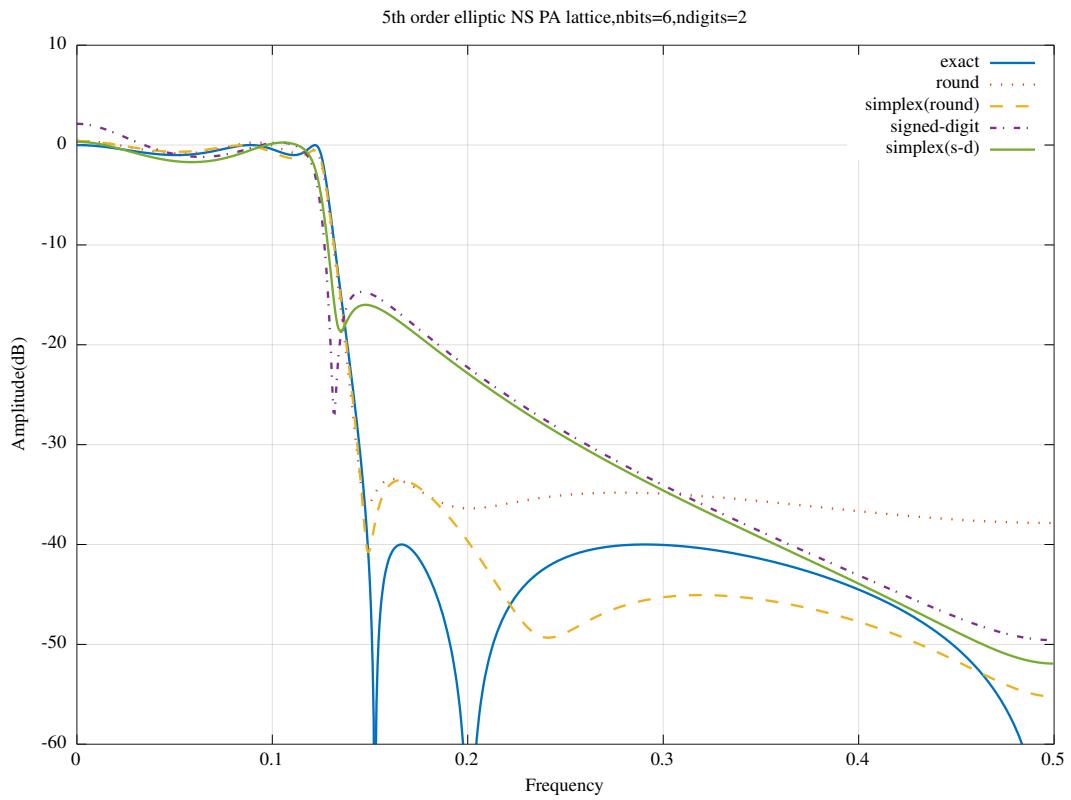


Figure 15.15: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

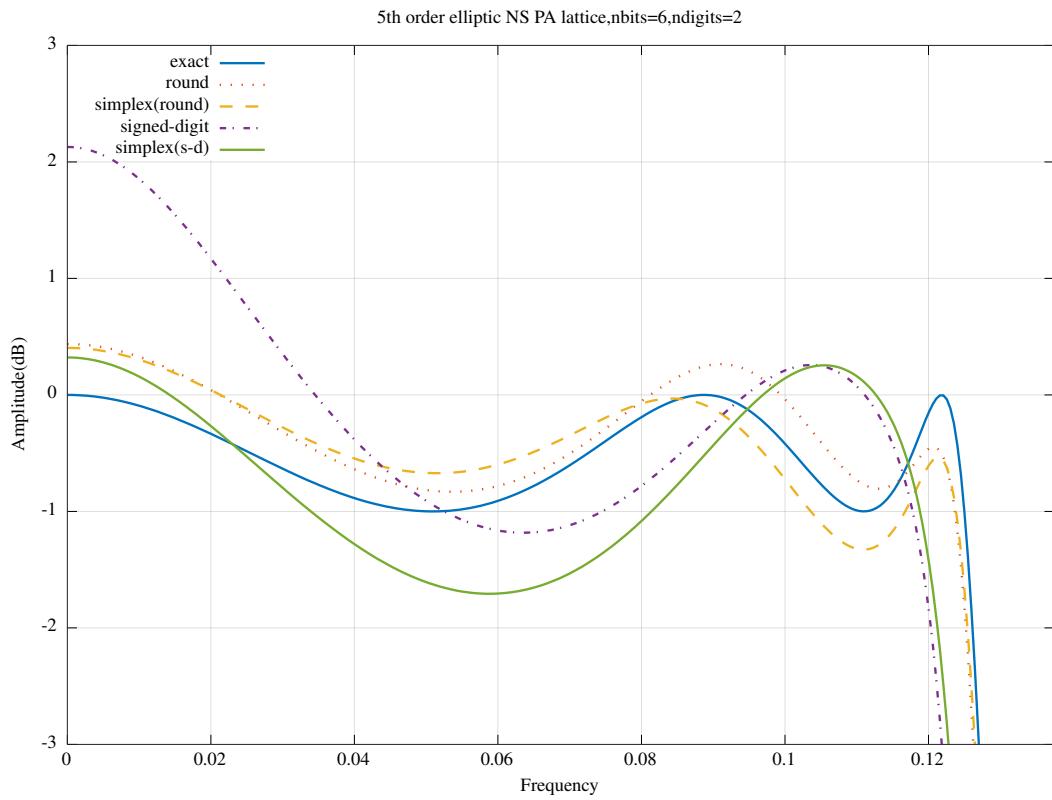


Figure 15.16: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

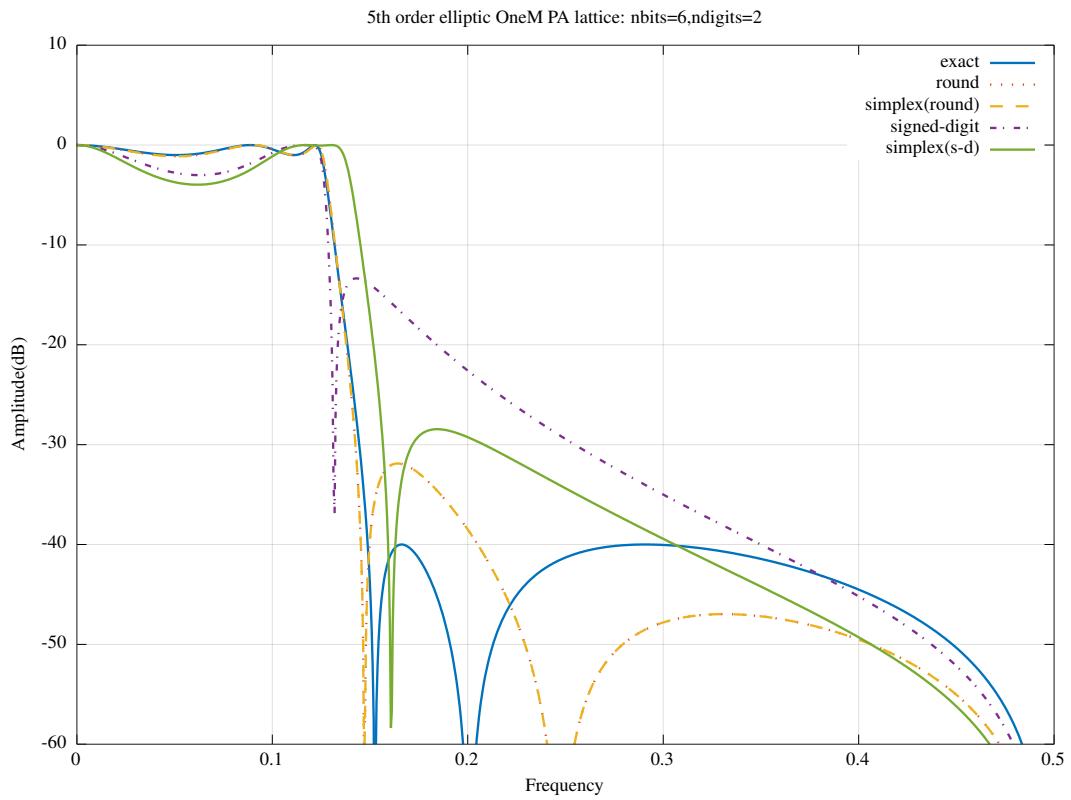


Figure 15.17: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

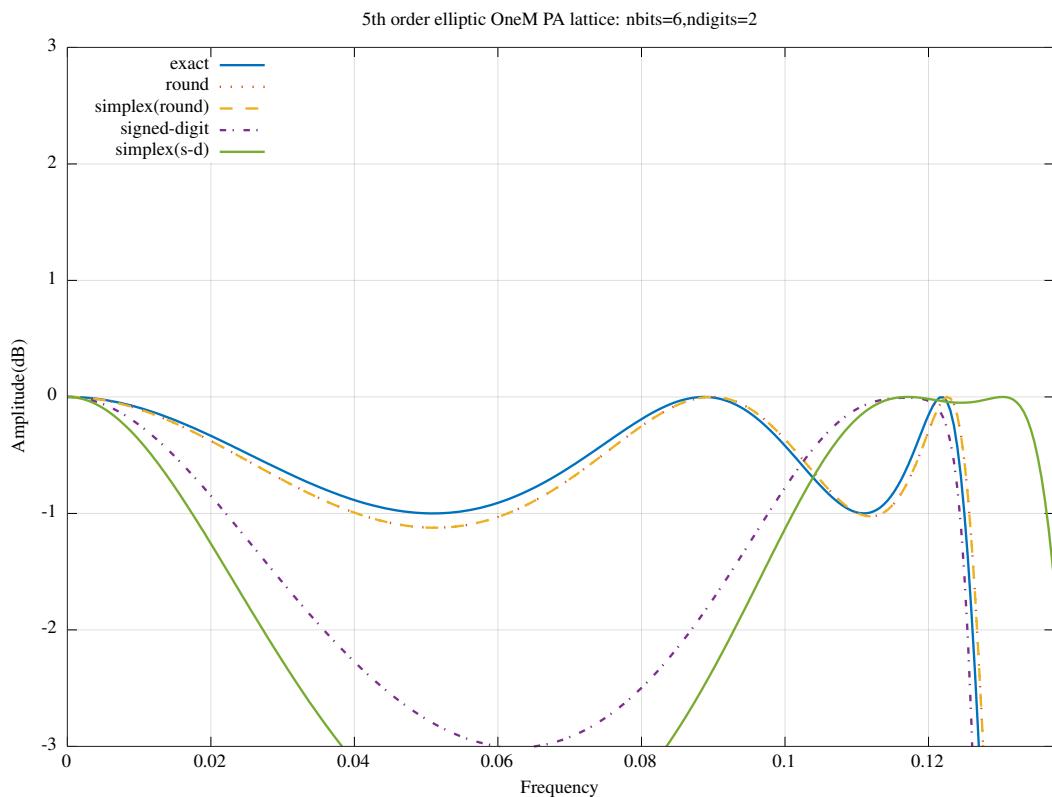


Figure 15.18: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

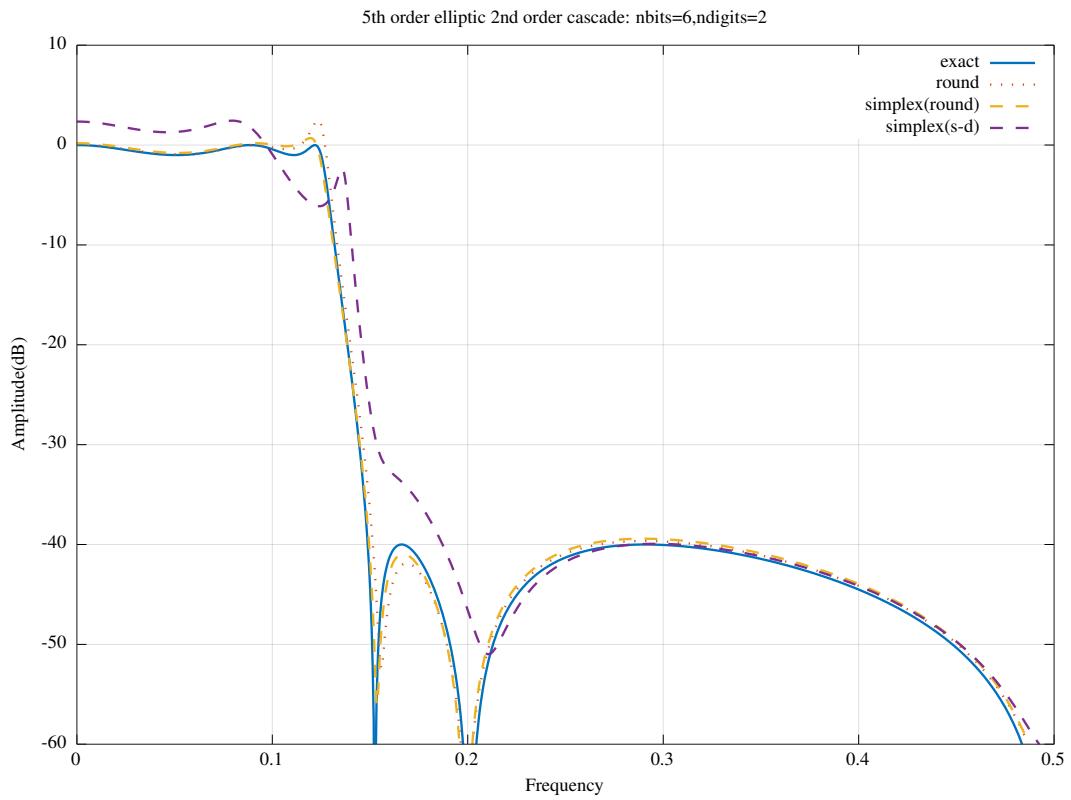


Figure 15.19: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

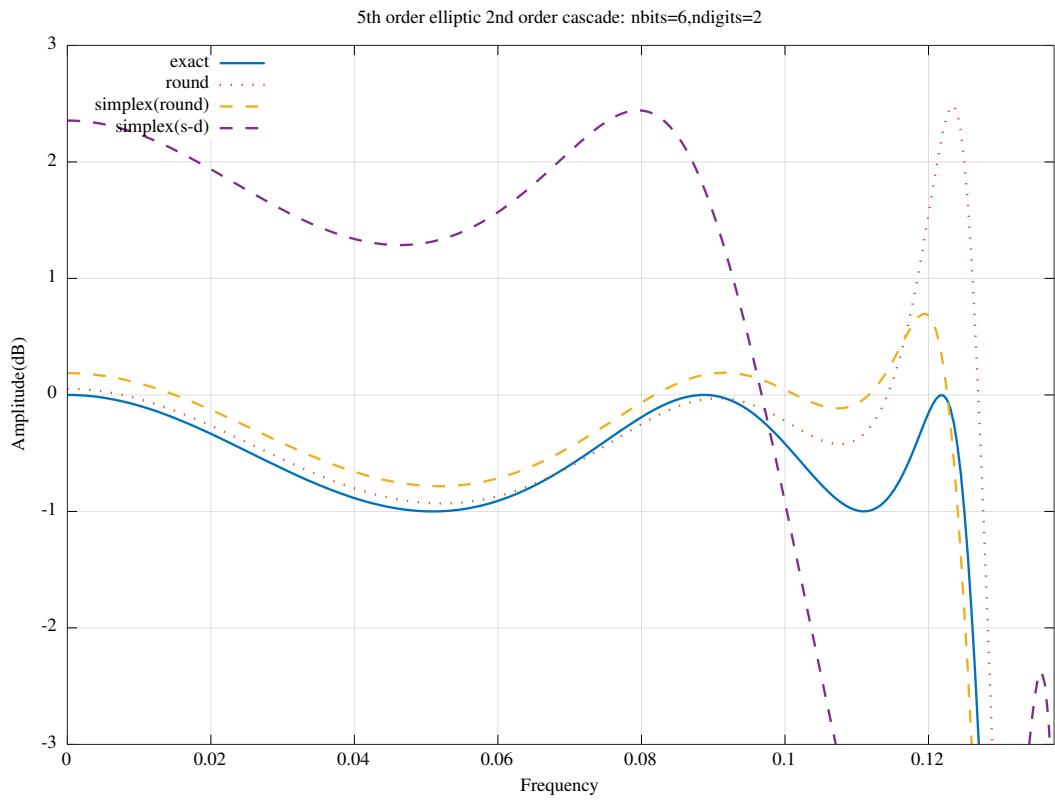


Figure 15.20: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

15.3 Searching with the *simulated annealing* algorithm

This section shows the result of searching for the coefficients of a 5-th order low-pass filter with the Octave-Forge *optim* package [64] implementation of the simulated annealing algorithm, *samin*. Unfortunately, the minimum cost found by *samin* may vary by a factor of 2 or more from run to run. In this section I show the best results from 10 runs of each test.

The Octave script *samin_NS_lattice_test.m* implements the prototype elliptic filter as a normalised-scaled lattice and optimises the truncated coefficients with the *samin* simulated annealing algorithm from the Octave-Forge *optim* package [64]. Figures 15.21 and 15.22 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

The Octave script *samin_OneM_lattice_test.m* implements the prototype elliptic filter as a one-multiplier lattice and optimises the truncated coefficients with the simulated annealing algorithm. Figures 15.23 and 15.24 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm. The one multiplier lattice state scaling coefficients are not truncated.

The Octave script *samin_NSPA_lattice_test.m* implements the 5th order elliptic filter as the sum of two normalised-scaled all-pass lattice filters and optimises the truncated coefficients with the simulated annealing algorithm. Figures 15.25 and 15.26 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

The Octave script *samin_OneMPA_lattice_test.m* implements the 5th order elliptic filter as the sum of two one-multiplier all-pass lattice filters and optimises the truncated coefficients with the simulated annealing algorithm. Figures 15.27 and 15.28 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm. The one multiplier lattice state scaling coefficients are not truncated.

Finally, the Octave script *samin_svccasc_test.m* implements the 5th order elliptic filter as a pair of 2nd order minimum-noise state variable sections followed by a 1st order state variable section. (See Section 6). The script optimises the truncated coefficients with the simulated annealing algorithm. Figures 15.29 and 15.30 show the overall and passband responses of the filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm. Note that the 2nd order state variable cascade filter obtained by simply converting the exact coefficients to 6 bit 2 signed-digit coefficients is unstable.

Table 15.3 shows the minimum cost result for 10 runs of each test.

Cost	Normalised-scaled lattice	One-multiplier lattice	Normalised-scaled PA lattice	One-multiplier PA lattice	2nd Order cascade
Exact	1.0008	1.0008	1.0008	1.0008	1.0008
Rounded	1.5432	1.5299	1.2459	1.1097	1.2362
Rounded with samin	0.7882	0.9516	0.9171	0.8803	0.6625
Signed-digit	2.3373	2.7157	3.6059	3.9013	∞
Signed-digit with samin	0.7532	1.7492	0.9371	1.3505	0.7473

Table 15.3: Summary of the cost results for the example 5th order elliptic low-pass filter synthesised as a normalised-scaled or one-multiplier lattice filter or as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

The lowest cost 6 bit 2 signed-digit coefficients of the normalised-scaled lattice filter found with the simulated annealing algorithm are:

```
s10_sasd = [ 0.625000, 0.531250, 0.187500, 0.062500, 0.000000 ]';
s11_sasd = [ 0.250000, 0.750000, 0.750000, 1.000000, 0.375000 ]';
```

```
s20_sasd = [ -0.750000,  0.625000, -0.968750,  0.968750, -0.625000 ]';
s00_sasd = [  0.625000,  0.531250,  0.500000,  0.750000,  0.968750 ]';
s02_sasd = [  0.750000, -0.968750,  1.000000, -1.000000,  0.375000 ]';
s22_sasd = [  0.625000,  0.156250,  0.093750,  0.875000, -0.375000 ]';
```

The lowest cost 6 bit rounded coefficients of the parallel-allpass one-multiplier lattice filter found with the simulated annealing algorithm are¹.

```
A1k_sa = [ -0.812500  0.625000 ]';
A2k_sa = [ -0.781250  0.906250 -0.593750 ]';
```

The lowest cost 6 bit 2 signed-digit coefficients of the cascade of 2nd order sections found with the simulated annealing algorithm are:

```
a11_sasd = [  0.750000,  0.750000,  0.000000 ];
a12_sasd = [ -0.500000, -0.468750, -1.500000 ];
a21_sasd = [  0.312500,  0.937500,  0.000000 ];
a22_sasd = [  0.562500,  0.625000,  0.562500 ];
b1_sasd = [  0.500000,  0.500000,  0.000000 ];
b2_sasd = [ -0.093750, -0.875000,  0.625000 ];
c1_sasd = [  0.156250,  0.625000,  0.000000 ];
c2_sasd = [  0.468750, -0.031250,  1.000000 ];
dd_sasd = [  0.156250,  0.281250,  0.437500 ];
```

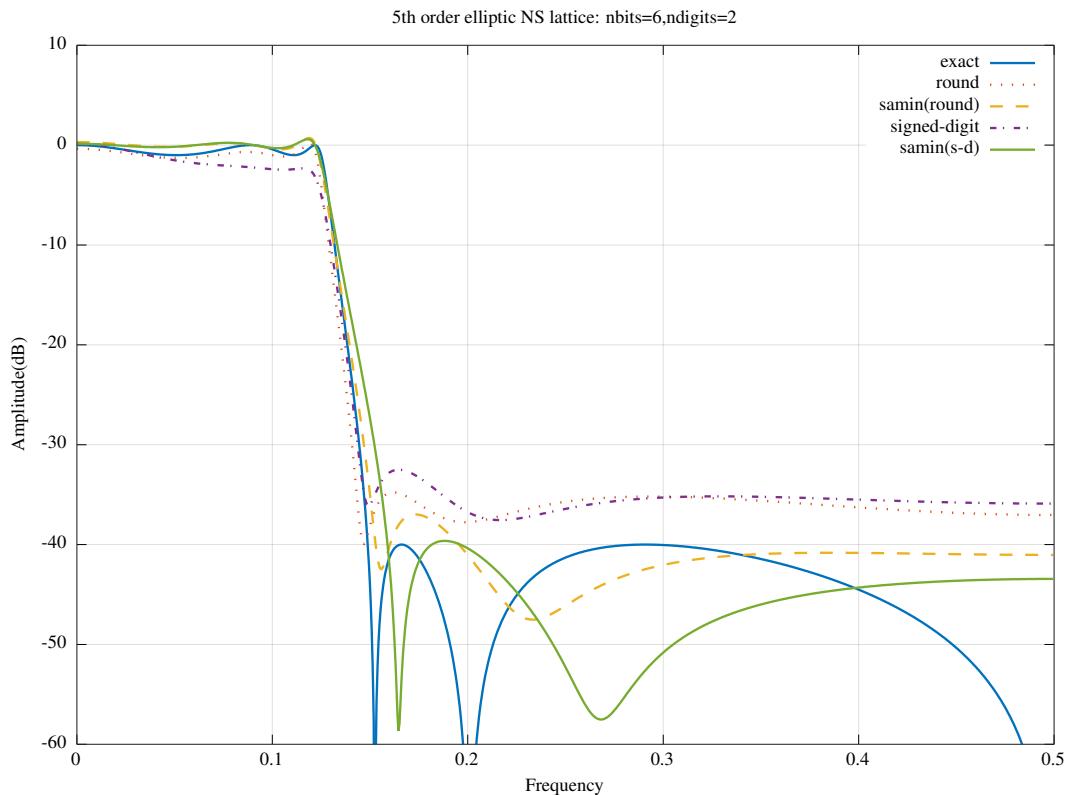


Figure 15.21: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

¹These coefficient multiplications can be implemented with a total of 14 signed-digits

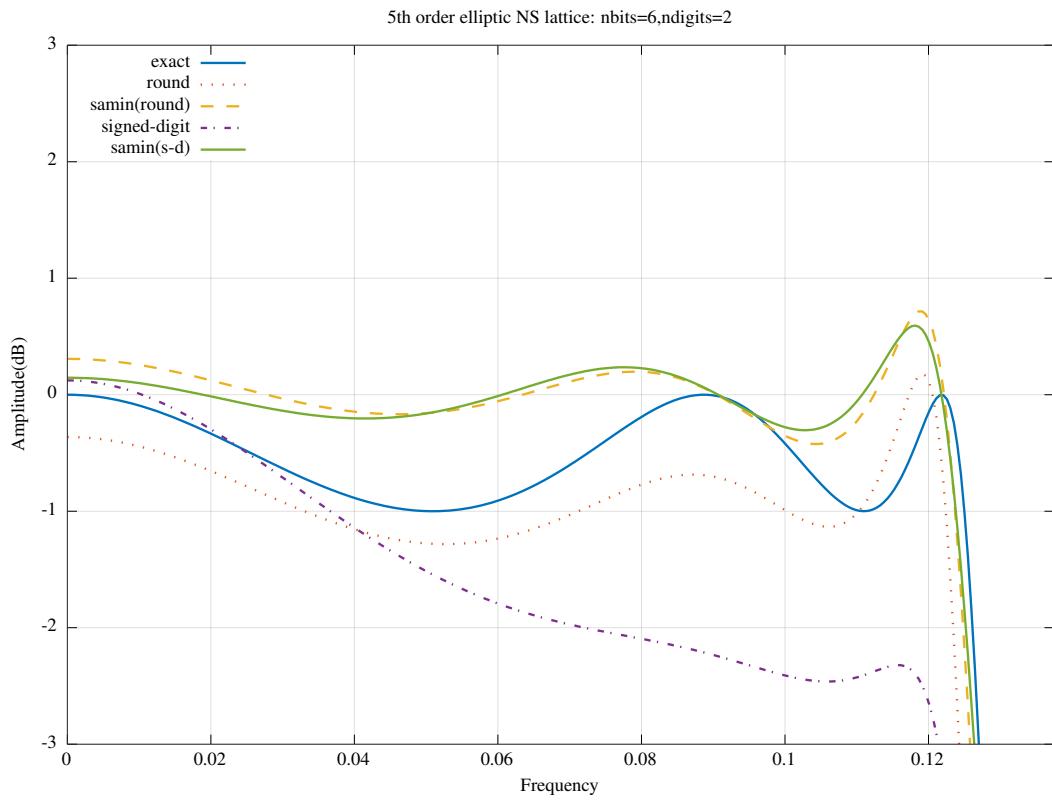


Figure 15.22: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

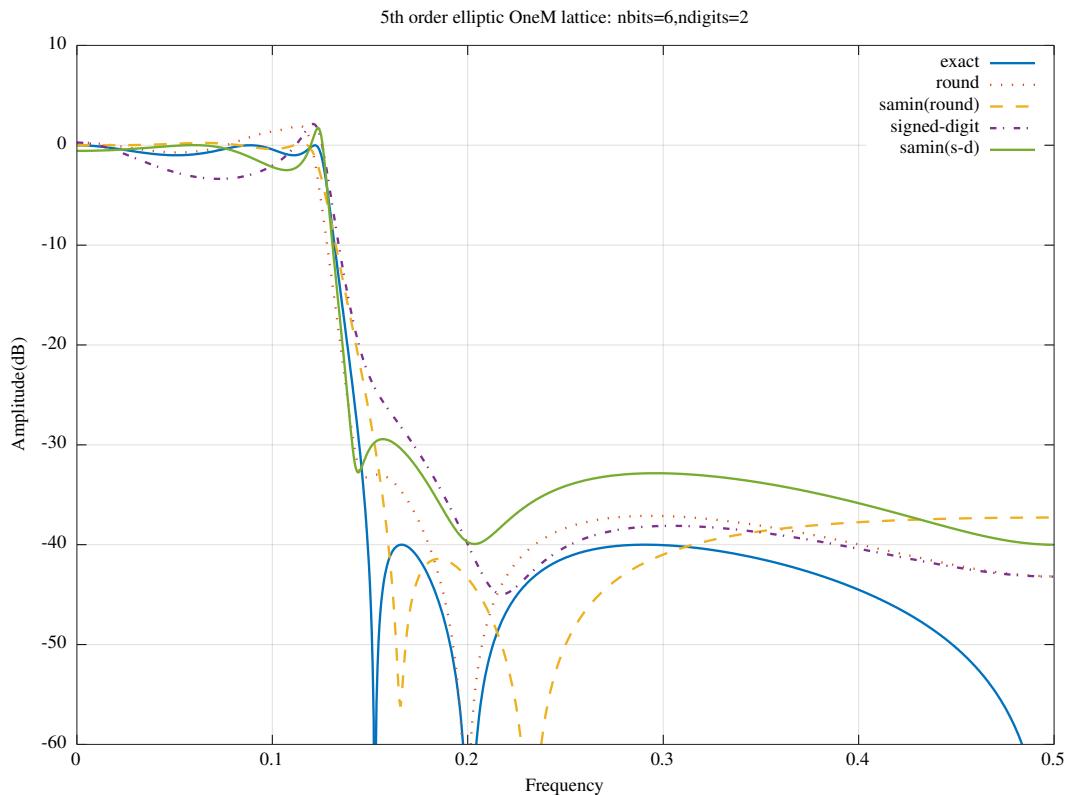


Figure 15.23: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

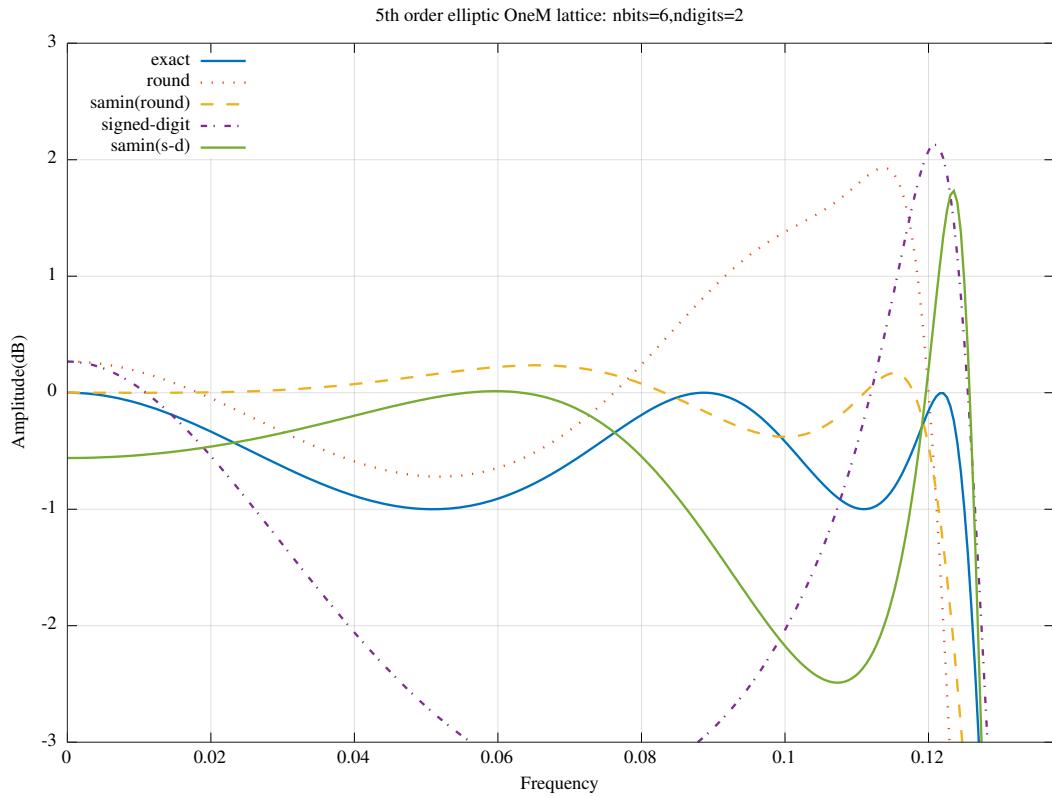


Figure 15.24: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

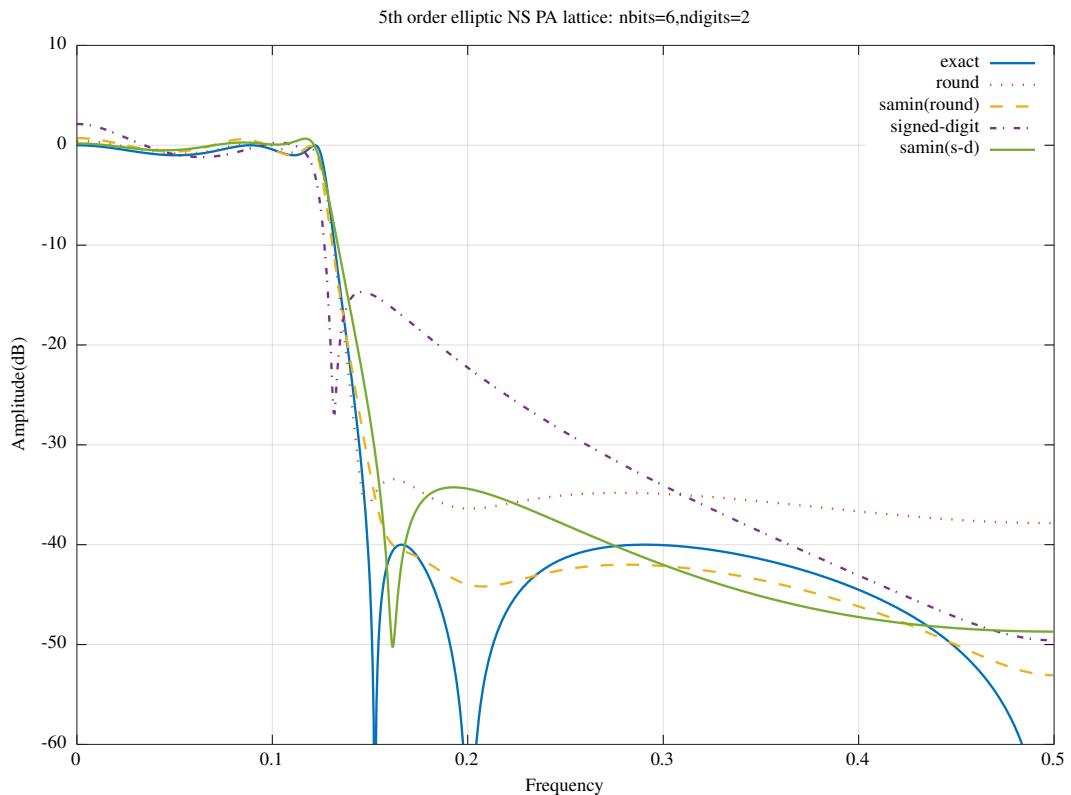


Figure 15.25: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

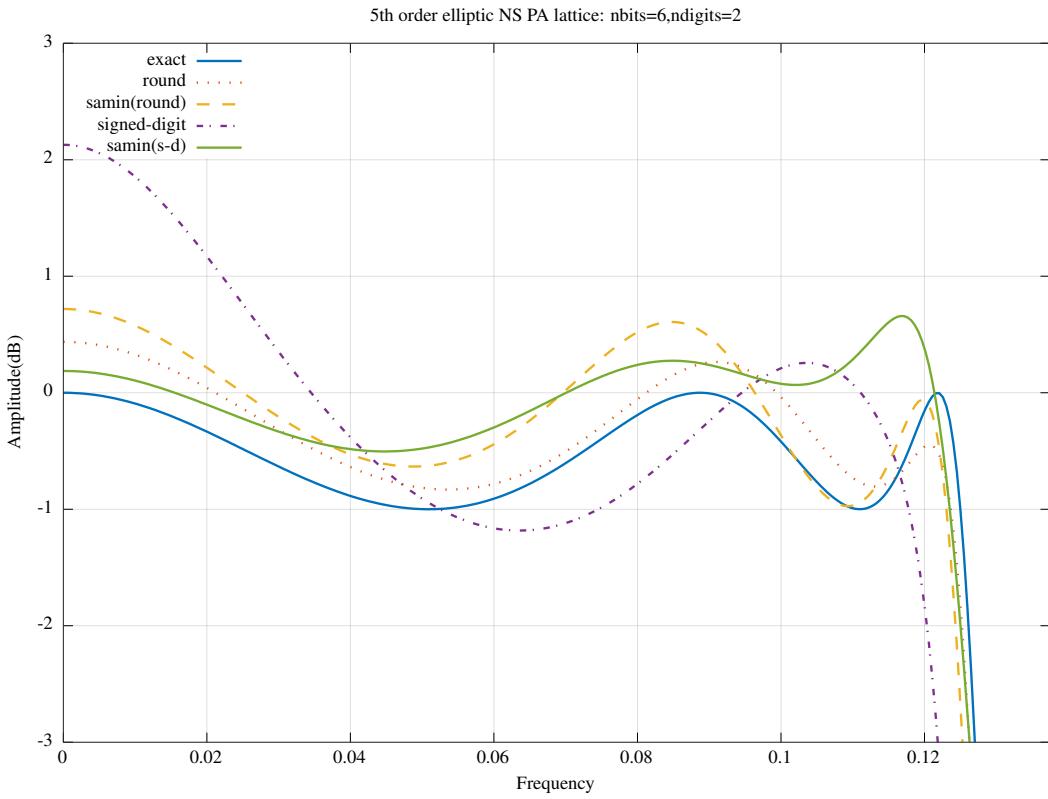


Figure 15.26: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

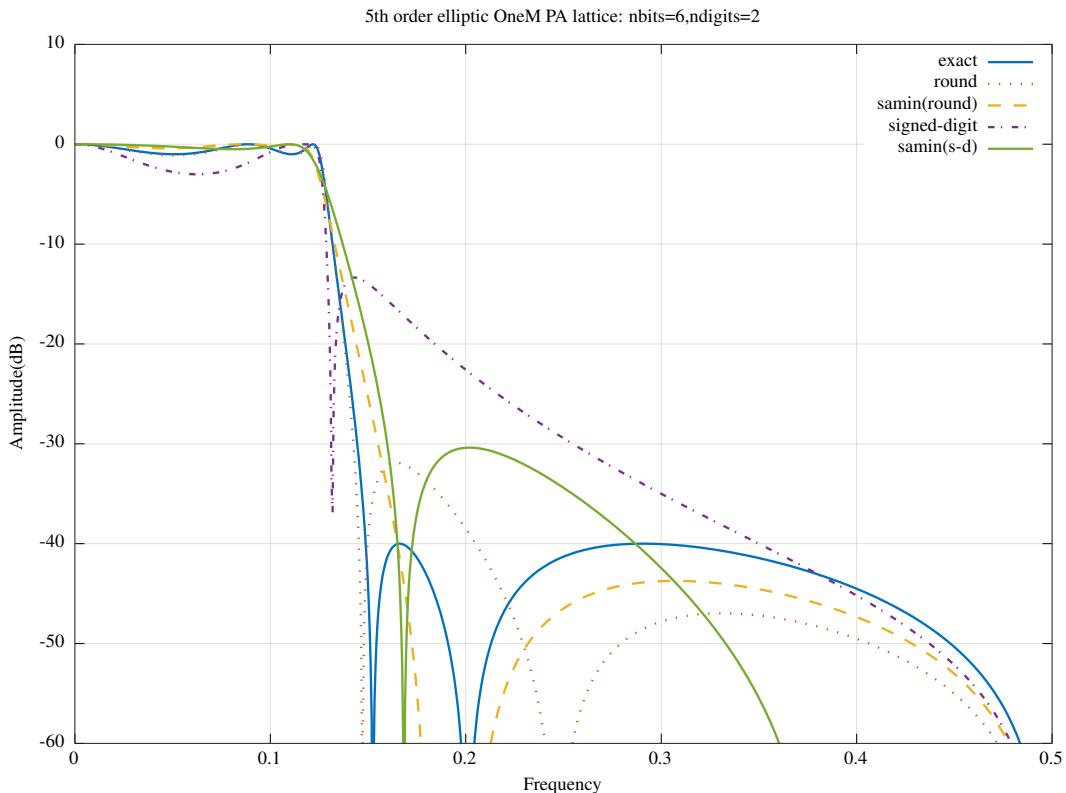


Figure 15.27: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

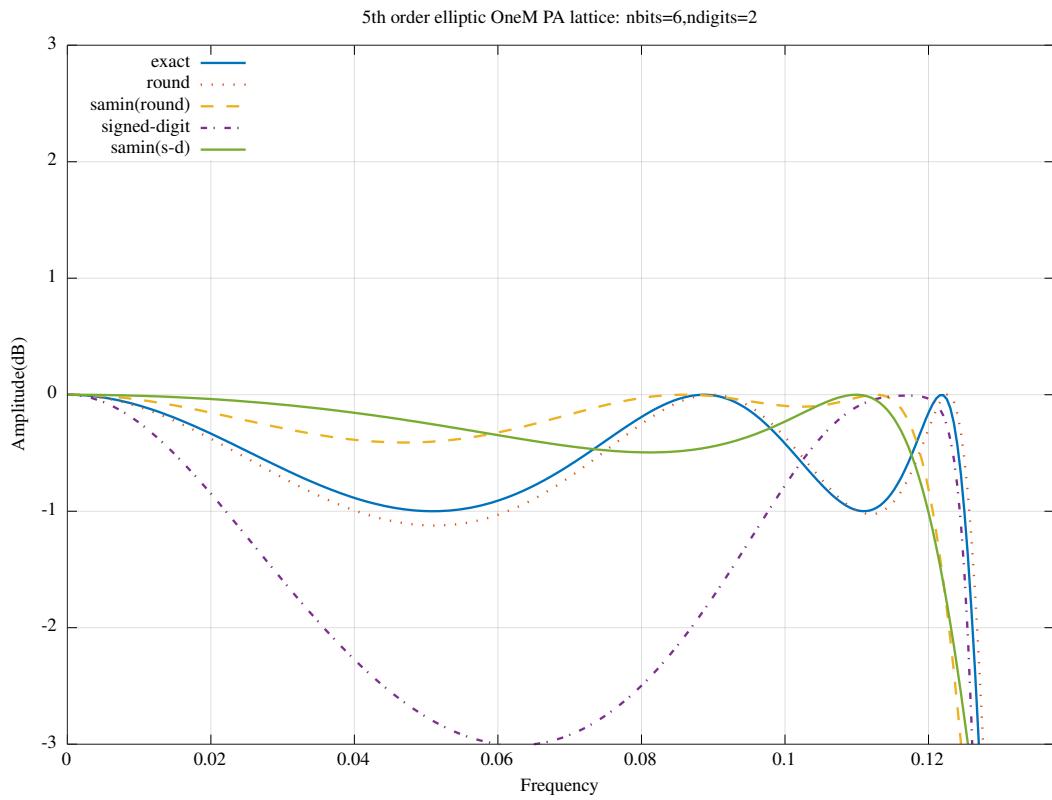


Figure 15.28: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

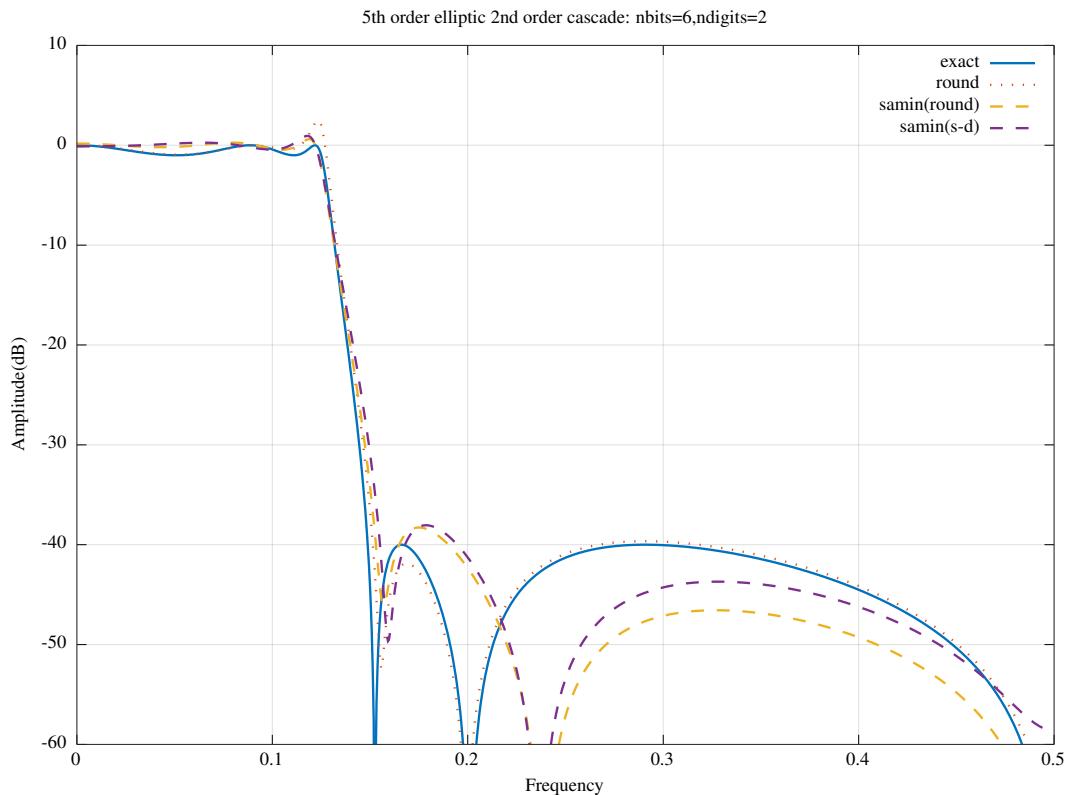


Figure 15.29: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

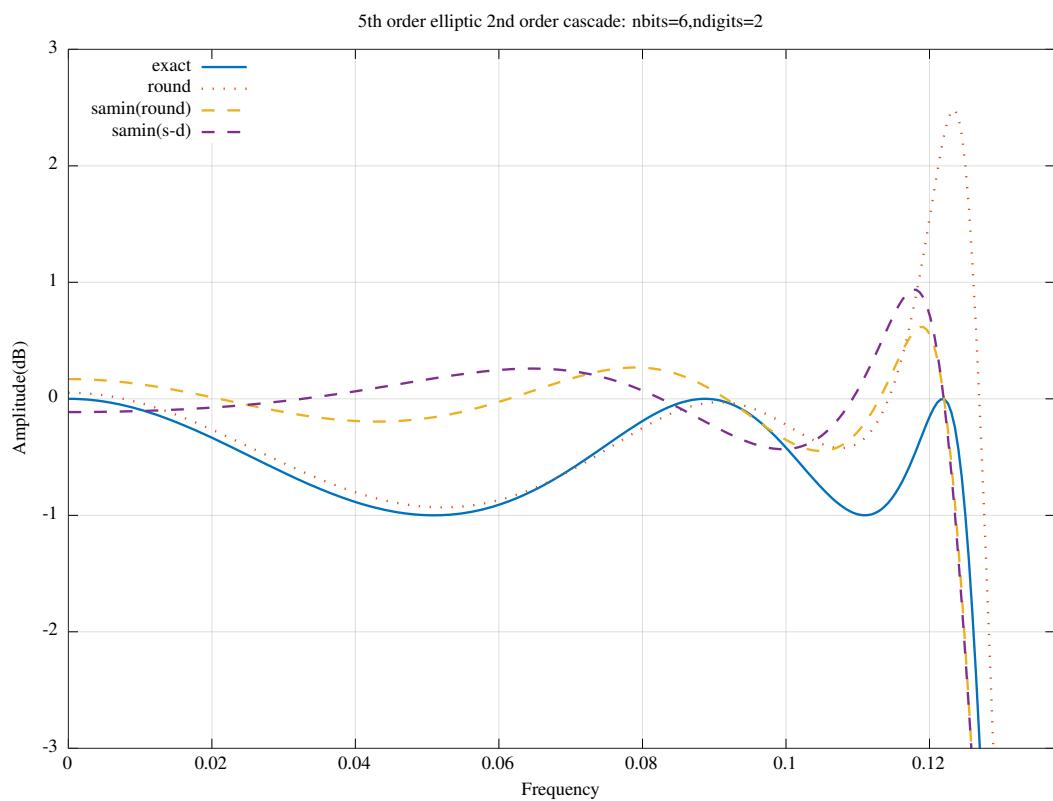


Figure 15.30: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

15.4 Searching with the *differential evolution* algorithm

This section shows the result of searching for the coefficients of a 5-th order low-pass filter with the Octave-Forge *optim* package [64] implementation of the *differential evolution* algorithm of *Storn and Price* [79], *de_min*. Unfortunately, the minimum cost found by *de_min* may vary from run to run. In this section I show the best results from 10 runs of each test. These tests use the default control settings for *de_min*.

The Octave script *de_min_NS_lattice_test.m* implements the prototype elliptic filter as a normalised-scaled lattice and optimises the truncated coefficients with the *de_min* differential evolution algorithm from the Octave-Forge *optim* package [64]. Figures 15.31 and 15.32 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

The Octave script *de_min_OneM_lattice_test.m* implements the prototype elliptic filter as a one-multiplier lattice and optimises the truncated coefficients with the differential evolution algorithm. Figures 15.33 and 15.34 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm. The one multiplier lattice state scaling coefficients are not truncated.

The Octave script *de_min_NSPA_lattice_test.m* implements the 5th order elliptic filter as the sum of two normalised-scaled all-pass lattice filters and optimises the truncated coefficients with the differential evolution algorithm. Figures 15.35 and 15.36 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

The Octave script *de_min_OneMPA_lattice_test.m* implements the 5th order elliptic filter as the sum of two one-multiplier all-pass lattice filters and optimises the truncated coefficients with the differential evolution algorithm. Figures 15.37 and 15.38 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm. The one multiplier lattice state scaling coefficients are not truncated.

Finally, the Octave script *de_min_svcasc_test.m* implements the 5th order elliptic filter as a pair of 2nd order minimum-noise state variable sections followed by a 1st order state variable section. (See Section 6). The script optimises the truncated coefficients with the differential evolution algorithm. Figures 15.39 and 15.40 show the overall and passband responses of the filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm. Note that the 2nd order state variable cascade filter obtained by simply converting the exact coefficients to 6 bit 2 signed-digit coefficients is unstable.

Table 15.4 shows the minimum cost result for 10 runs of each test.

Cost	Normalised-scaled lattice	One-multiplier lattice	Normalised-scaled PA lattice	One-multiplier PA lattice	2nd Order cascade
Exact	1.0008	1.0008	1.0008	1.0008	1.0008
Rounded	1.5432	1.5299	1.2459	1.1097	1.2362
Rounded with de_min	0.7931	1.2547	0.8669	0.8803	0.6571
Signed-digit	2.3373	2.7157	3.6059	3.9013	∞
Signed-digit with de_min	0.8539	1.4949	1.2004	2.2915	0.7047

Table 15.4: Summary of the cost results for the example 5th order elliptic low-pass filter synthesised as a normalised-scaled or one-multiplier lattice filter or as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

The lowest cost 6 bit rounded coefficients of the parallel-allpass one-multiplier lattice filter found after 10 runs of the differential evolution algorithm are the same as those found with the simulated-annealing algorithm in Section 15.3.

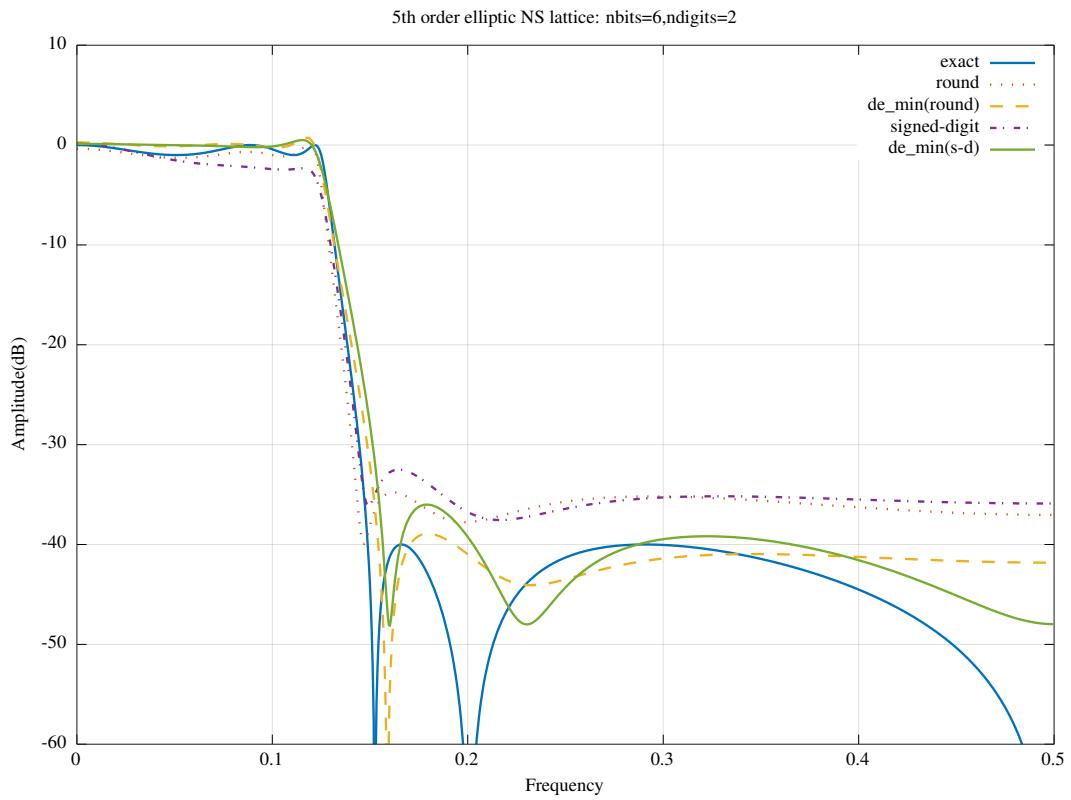


Figure 15.31: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

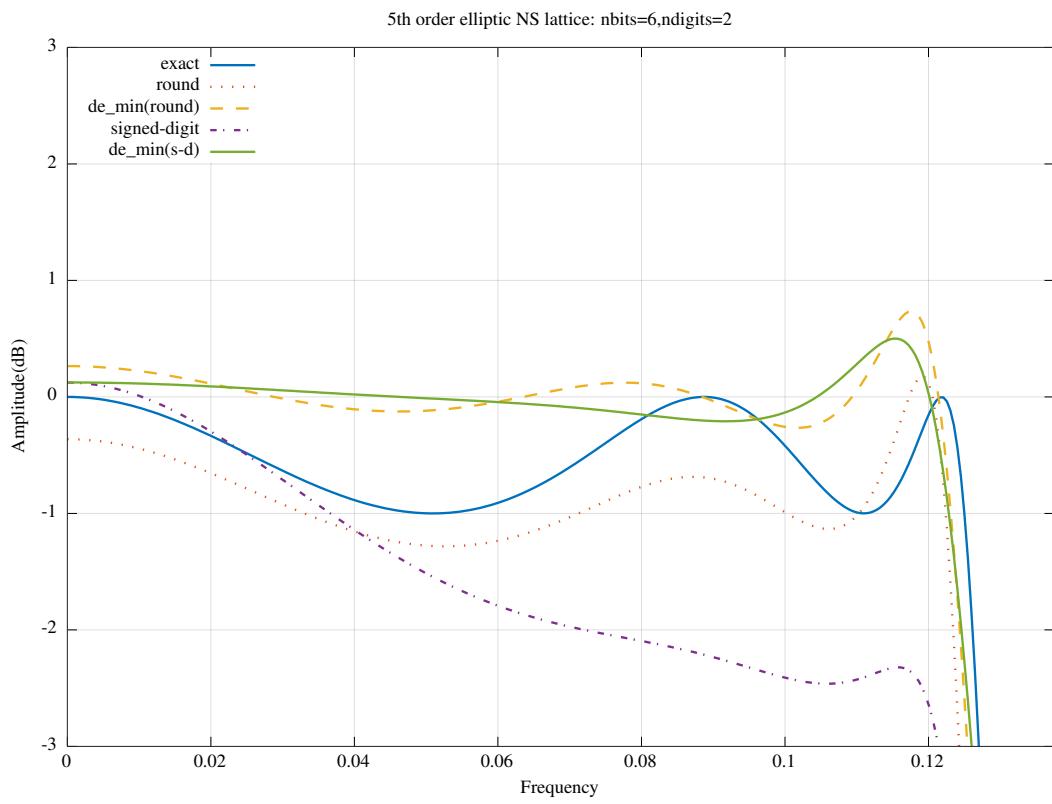


Figure 15.32: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

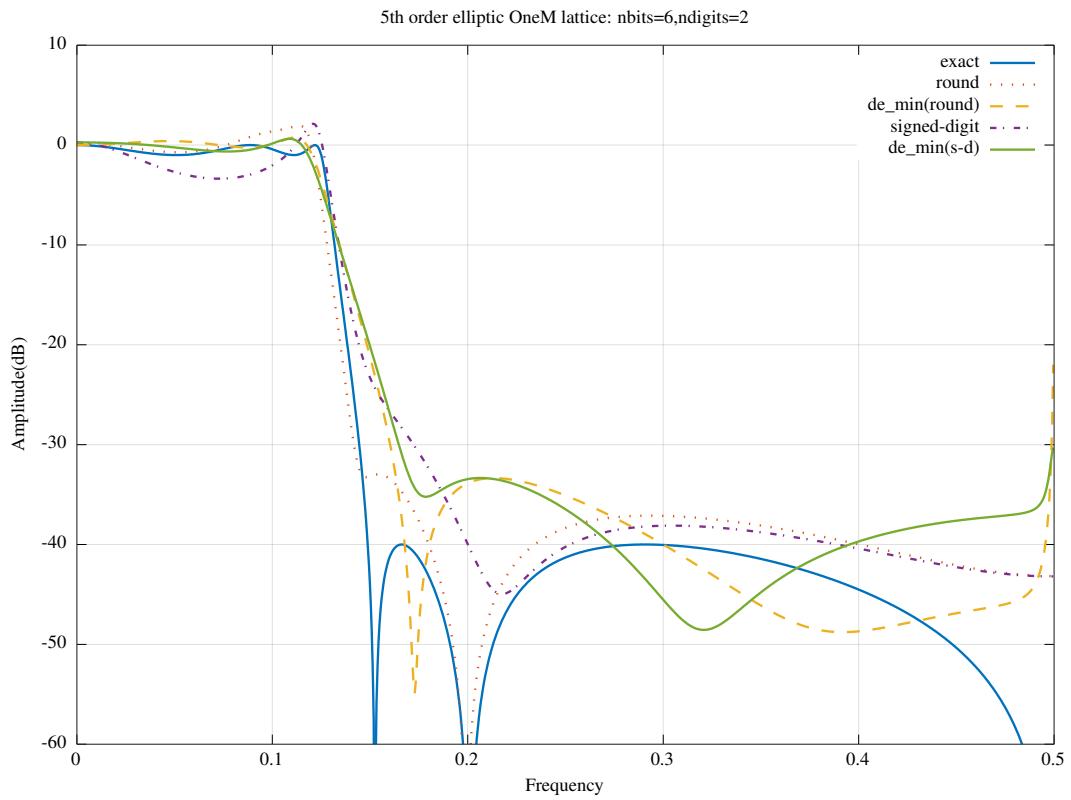


Figure 15.33: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

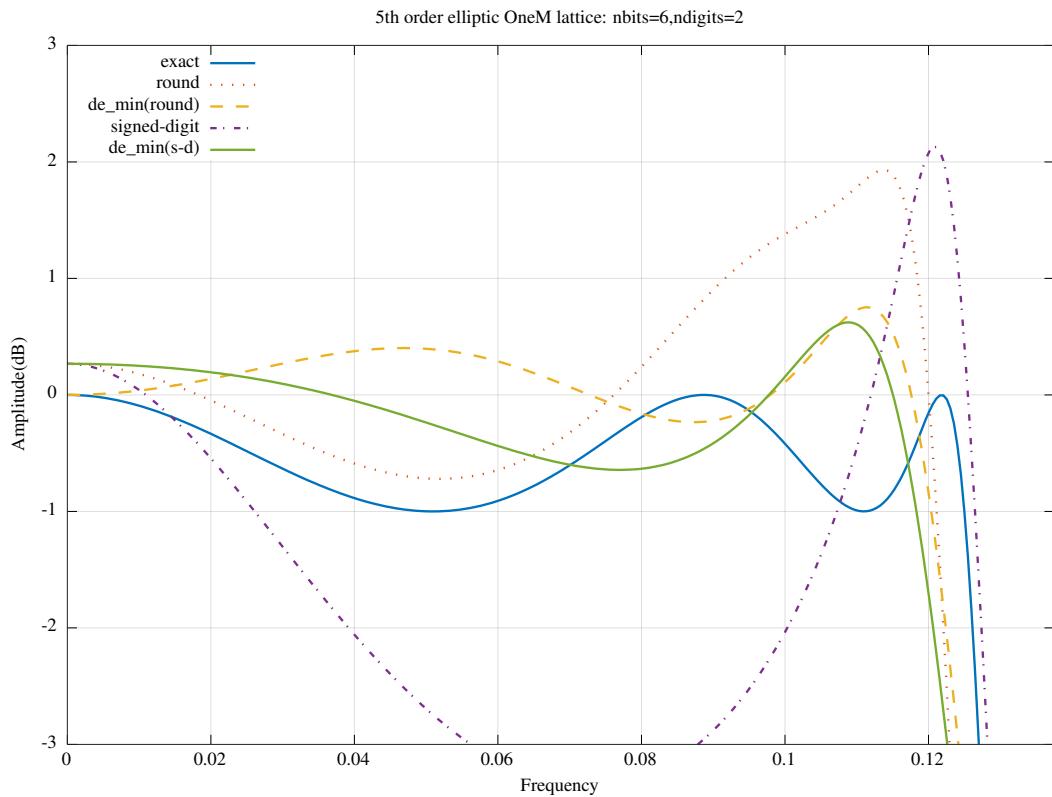


Figure 15.34: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

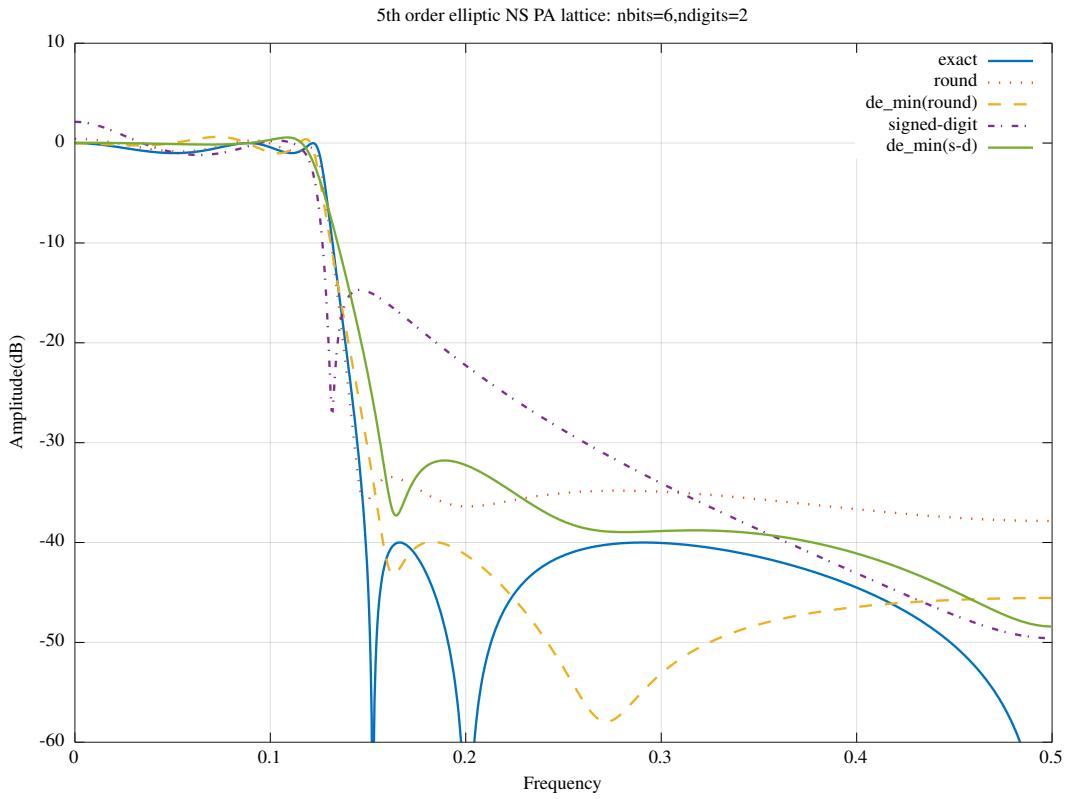


Figure 15.35: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

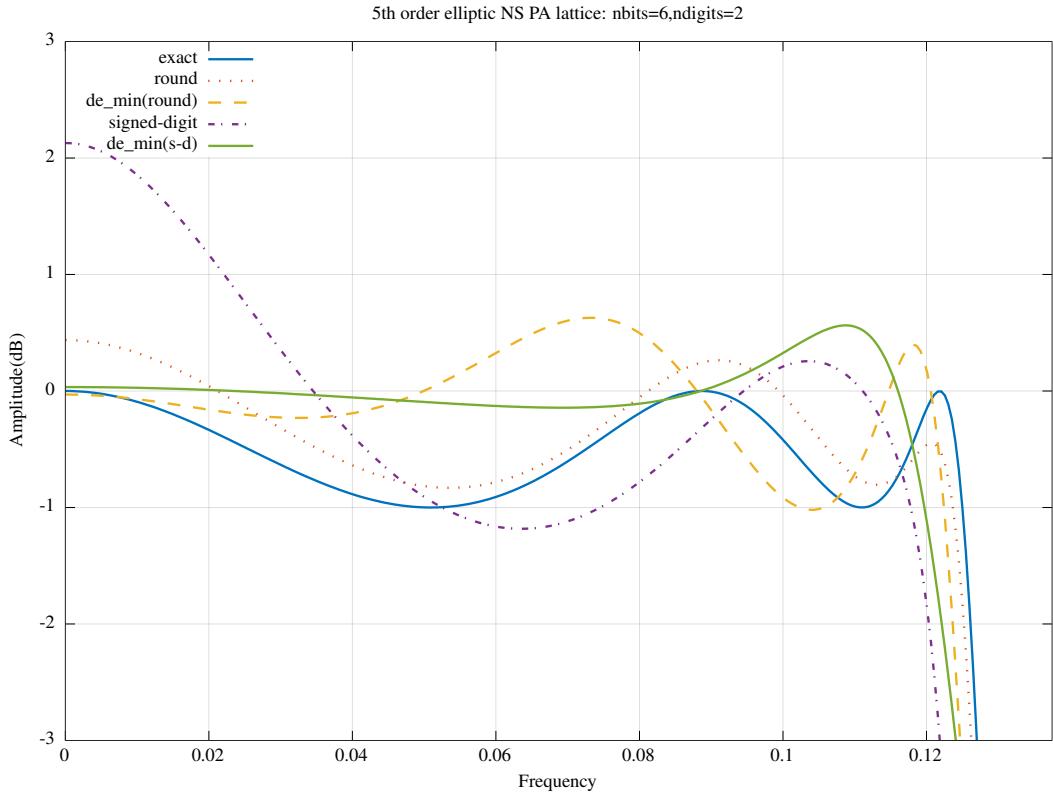


Figure 15.36: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

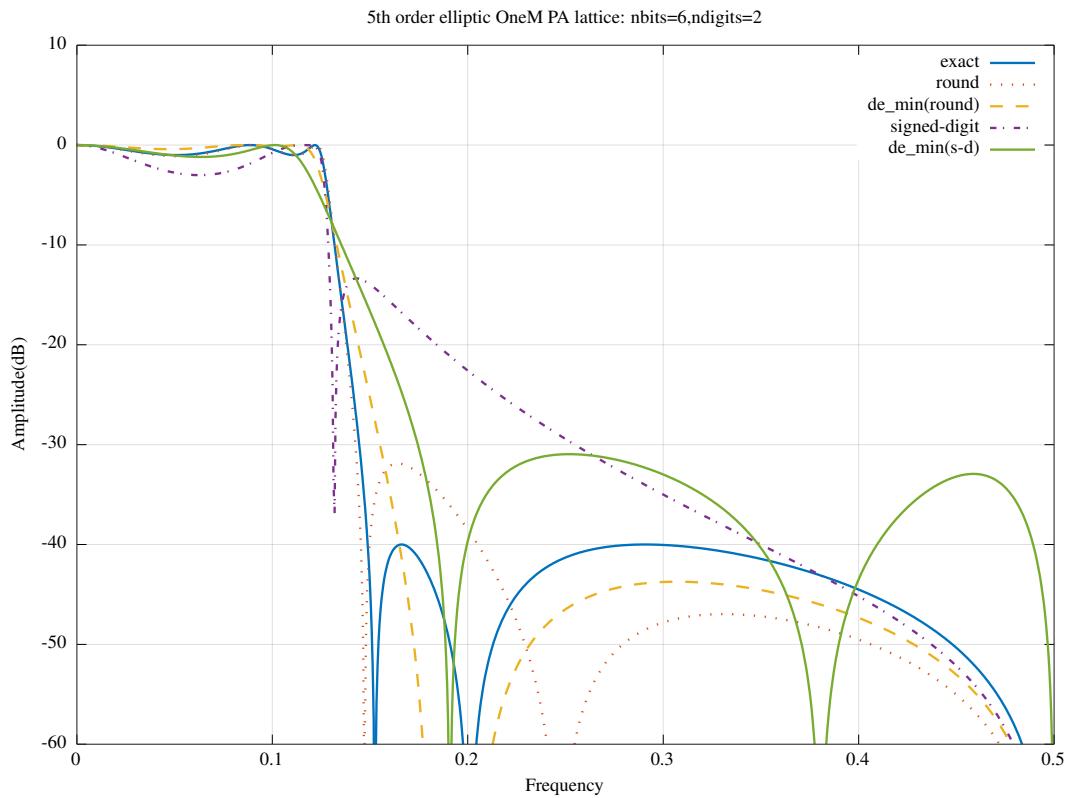


Figure 15.37: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

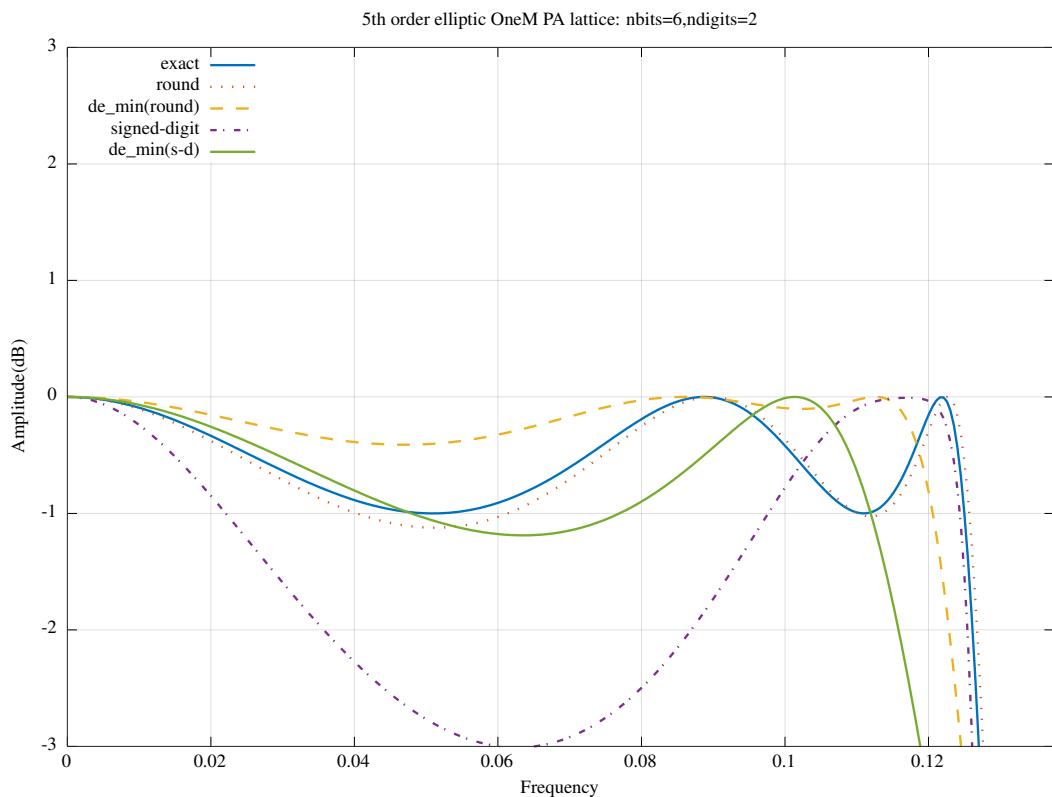


Figure 15.38: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

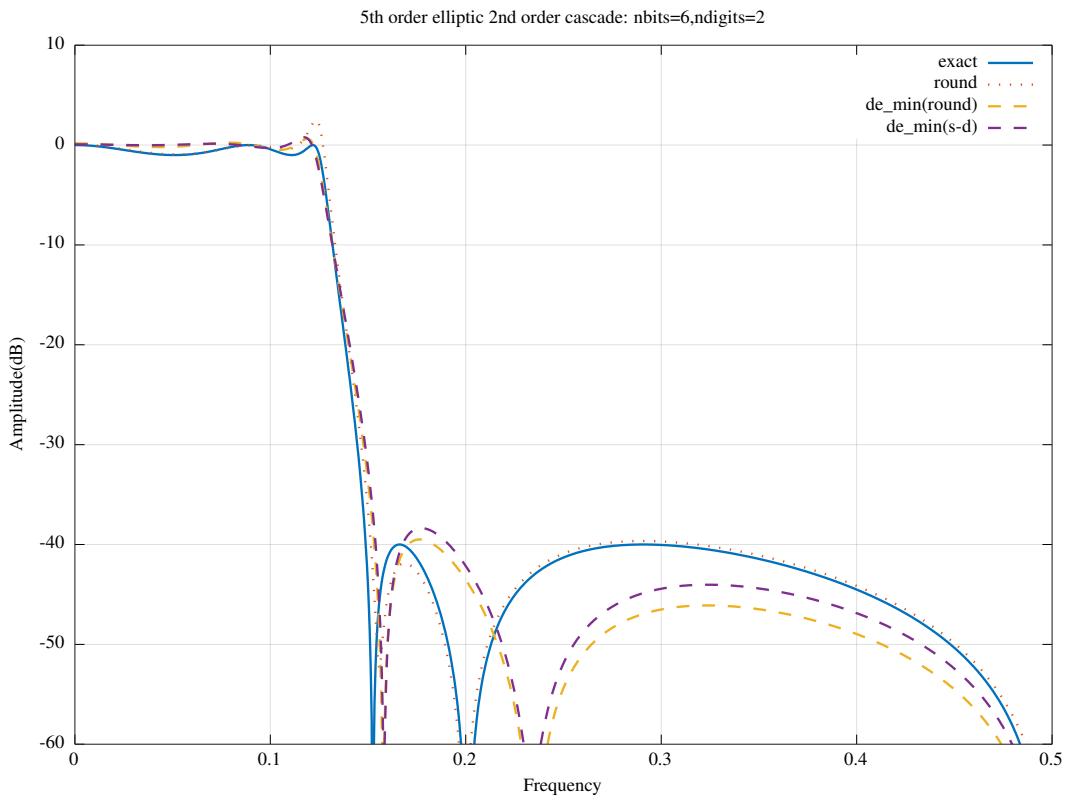


Figure 15.39: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

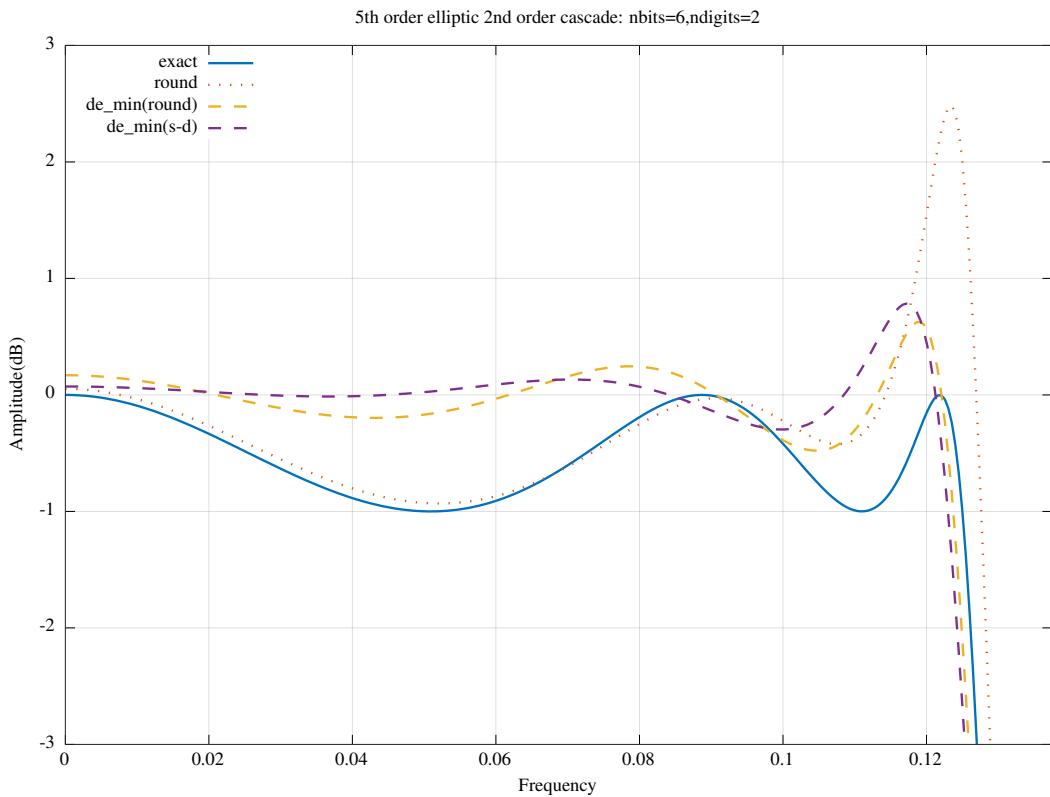


Figure 15.40: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

15.5 Summary of the search algorithm comparison

Table 15.5 compares the cost result for each of the search algorithms. The relative time consumed by the algorithms is, in increasing order: simplex, bit-flipping, differential evolution and simulated annealing. The simulated annealing and differential evolution costs shown are the minimum found for 10 runs of the corresponding test script.

Cost	Normalised-scaled lattice	One-multiplier lattice	Normalised-scaled PA lattice	One-multiplier PA lattice	2nd Order cascade
Exact	1.0008	1.0008	1.0008	1.0008	1.0008
Rounded	1.5432	1.5299	1.2459	1.1097	1.2362
Rounded with bit-flipping	1.1289	1.5299	0.9861	1.1097	0.9929
Rounded with simplex	1.0397	1.4617	0.9861	1.1097	0.8395
Rounded with samin	0.7882	0.9516	0.9171	0.8803	0.6625
Rounded with de_min	0.7931	1.2547	0.8669	0.8803	0.6571
Signed-digit	2.3373	2.7157	3.6059	3.9013	∞
Signed-digit with bit-flipping	0.8744	2.1571	1.3824	3.1746	1.1334
Signed-digit with simplex	1.6581	2.3918	3.2559	3.1746	3.2134
Signed-digit with samin	0.7532	1.7492	0.9371	1.3505	0.7473
Signed-digit with de_min	0.8539	1.4949	1.2004	2.2915	0.7047

Table 15.5: Comparison of the cost results for the example 5th order elliptic low-pass filter synthesised as a normalised-scaled or one-multiplier lattice filter or as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with each algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with each algorithm

Table 15.6 shows the number of coefficients for each filter implementation. The figure in parentheses for the normalised-scaled filters is the number of signed-digit coefficients optimised.

Cost	Normalised-scaled lattice	One-multiplier lattice	Normalised-scaled PA lattice	One-multiplier PA lattice	2nd Order cascade
Number of filter coefficients	20(30)	11	10(20)	5	23

Table 15.6: Comparison of the number of coefficients for the example 5th order elliptic low-pass filter synthesised as a normalised-scaled or one-multiplier lattice filter or as a cascade of 2nd order state variable sections. The figure in parentheses for the normalised-scaled filters is the number of signed-digit coefficients that are optimised.

Part IV

Appendices

Appendix A

Review of Complex Variables

This chapter summarises Kreysig [23, Chapter 12(Sections 4 and 5), Chapter 14].

A.1 Complex Functions

Define a function f on the complex numbers, \mathbb{C} , $w = f(z)$, where z varies in S and is called a “*complex variable*”. S is called the *domain* of z and the set of complex numbers, w , that $f(z)$ assumes is called the *range* of $f(z)$. Write the real and imaginary parts of w in terms of the real and imaginary parts of $z = x + iy$ as $w = f(z) = u(x, y) + iv(x, y)$ where u and v are real valued functions of the real variables x and y and, of course, $i = \sqrt{-1}$.

A.2 Limit

A function $f(z)$ is said to have a limit ℓ as z approaches $w = f(z) = u(x, y) + iv(x, y)$ if $f(z)$ is defined in a neighbourhood of z_0 (except perhaps at z_0 itself) and if for any positive, non-zero real number ϵ we can find a real positive δ such that, for all $z \neq z_0$ in the disk $|z - z_0| < \delta$, $|f(z) - \ell| < \epsilon$. We write:

$$\lim_{z \rightarrow z_0} f(z) = \ell$$

A function $f(z)$ is *continuous* at $z = z_0$ if $f(z_0)$ is defined and

$$\lim_{z \rightarrow z_0} f(z) = f(z_0)$$

A function $f(z)$ is *differentiable* at $z = z_0$ if the limit

$$f'(z) = \lim_{z \rightarrow z_0} \frac{f(z_0 + \Delta z) - f(z_0)}{\Delta z}$$

exists. This limit is called the *derivative* of $f(z)$ at $z = z_0$. $f(z)$ is said to be *analytic* (or *holomorphic*) in a domain D , if $f(z)$ is defined and differentiable at all points of D . For example, $f(z) = x - iy$ is not differentiable. If $\Delta z = \Delta x + i\Delta y$ then:

$$\frac{f(z + \Delta z) - f(z)}{\Delta z} = \frac{\Delta x - i\Delta y}{\Delta x + i\Delta y}$$

If Δz approaches z with $\Delta y = 0$ then the derivative is -1 but if Δz approaches z with $\Delta x = 0$ then the derivative is 1 . Hence the limit approaches different values along different paths to $z = x + iy$.

A.3 The Cauchy-Riemann Equations

Suppose $f(z) = u(x, y) + iv(x, y)$ is defined and continuous within a neighbourhood of an arbitrary fixed point z and differentiable at z so that $f'(z)$ exists. Set $\Delta z = \Delta x + i\Delta y$. On a path for which Δz approaches z with $\Delta y = 0$, then:

$$f'(z) = \lim_{\Delta x \rightarrow 0} \frac{u(x + \Delta x, y) - u(x, y)}{\Delta x} + i \lim_{\Delta x \rightarrow 0} \frac{v(x + \Delta x, y) - v(x, y)}{\Delta x}$$

$$= \frac{\partial u}{\partial x} + i \frac{\partial v}{\partial x}$$

Similarly, for a path on which Δz approaches z with $\Delta x = 0$

$$f'(z) = \frac{\partial v}{\partial y} - i \frac{\partial u}{\partial y}$$

Equating real and imaginary parts, we obtain the Cauchy-Riemann equations

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$$

and

$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

If $f(z) = u(x, y) + iv(x, y)$ is analytic in D then the real and imaginary parts of f satisfy Laplace's equation in D and have continuous second partial derivatives in D

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

and

$$\nabla^2 v = 0$$

A.4 Line integrals in the complex plane

Let C be a smooth curve in the z plane with $z(t) = x(t) + iy(t)$ with $a \leq t \leq b$ where $z(t)$ has a continuous derivative $\dot{z}(t) \neq 0$ for all t . Let $f(z)$ be a continuous function defined at each point of C . Divide the interval $a \leq t \leq b$ into sub-intervals $t_0 = a \leq t_1 \leq \dots \leq t_n = b$ corresponding to $z_0 = z(t_0), z_1, \dots, z_n = z(t_n)$. For each sub-interval of C choose an arbitrary point, ζ_i , between z_{i-1} and z_i . Then form the sums

$$S_n = \sum_{i=1}^n f(\zeta_i) \Delta z_i$$

where $\Delta z_i = z_i - z_{i-1}$. Define $\zeta_i = \varepsilon_i + i\eta_i$ and $\Delta z_i = \Delta x_i + i\Delta y_i$ and let $f(z) = u(x, y) + iv(x, y)$. Then S_n consists of four real sums

$$\begin{aligned} S_n &= \sum_{i=1}^n (u + iv)(\Delta x_i + i\Delta y_i) \\ &= \sum_{i=1}^n u\Delta x_i - \sum_{i=1}^n v\Delta y_i + i \left[\sum_{i=1}^n u\Delta y_i + \sum_{i=1}^n v\Delta x_i \right] \end{aligned}$$

and the *line integral* of $f(z)$ along C is defined as the limit of the sums S_n and

$$\begin{aligned} \int_C f(z) dz &= \lim_{n \rightarrow \infty} S_n \\ &= \int_C u dx - \int_C v dy + i \left[\int_C u dy + \int_C v dx \right] \\ &= \int_a^b u \dot{x} dt - \int_a^b v \dot{y} dt + i \left[\int_a^b u \dot{y} dt + \int_a^b v \dot{x} dt \right] \\ &= \int_c f[z(t)] \dot{z} dt \end{aligned}$$

The absolute value of the line integral is bounded by

$$\left| \int_C f(z) dz \right| \leq Ml$$

where l is the length of the path C and M is a real constant such that $|f(z)| \leq M$ everywhere on the path C .

A.5 Cauchy's Integral Theorem

A domain D in the complex plane is called *simply connected* if every simple closed curve in D encloses only points in D . Such a domain D is said to be *bounded* if D lies entirely within a circle about the origin. If $f(z)$ is analytic in a simply connected bounded domain D then for each simple closed path C in D

$$\oint_C f(z) dz = 0$$

PROOF: Cauchy made the additional assumption that $f'(z)$ is continuous and applied Green's theorem. Write

$$\oint_C f(z) dz = \int_C u dx - v dy + i \int_C u dy + v dx$$

$f(z)$ is analytic so $f'(z)$ exists. For the real part, using the Cauchy-Riemann equations, by Green's theorem

$$\begin{aligned} \int_C u dx - v dy &= \iint_R \left[-\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right] dx dy \\ &= 0 \end{aligned}$$

where R is the region bounded by C . Similarly for the imaginary part. Goursat provided a proof that does not require $f'(z)$ to be continuous. A corollary of Cauchy's Integral theorem is that the line integral of $f(z)$ is independent of the path in D .

A.6 Cauchy's Integral Formula

Let $f(z)$ be analytic in a simply connected domain, D . Then for any point z_0 in D and any simple closed path C in D which encloses z_0

$$\oint_C \frac{f(z)}{z - z_0} dz = 2\pi i f(z_0)$$

PROOF: Let

$$f(z) = f(z_0) + [f(z) - f(z_0)]$$

so

$$\oint_C \frac{f(z)}{z - z_0} dz = f(z_0) \oint_C \frac{dz}{z - z_0} + \oint_C \frac{f(z) - f(z_0)}{z - z_0} dz$$

For the first term use the identity

$$\int_c (z - z_0)^m dz = \begin{cases} 2\pi i & m = -1 \\ 0 & m \neq -1, \text{ integral} \end{cases}$$

found by setting $z(t) = z_0 + \rho e^{it}$ and integrating over t . For the second term, replace C by a small circle K , with centre z_0 . $f(z)$ is analytic so for any $\epsilon > 0$ we can find a $\delta > 0$ so that $|f(z) - f(z_0)| < \epsilon$ for all z in $|z - z_0| < \delta$. Choose the radius ρ of K smaller than δ then

$$\left| \frac{f(z) - f(z_0)}{z - z_0} \right| < \frac{\epsilon}{\rho}$$

at each point of K . Since the length of K is $2\pi\rho$

$$\left| \int_K \frac{f(z) - f(z_0)}{z - z_0} dz \right| < 2\pi\epsilon$$

at each point of K and the second term is shown to be zero.

A.7 Derivatives of an analytic function

If $f(z)$ is analytic in D then it has derivatives of all orders in D which are also analytic functions in D

$$f^{(n)}(z_0) = \frac{n!}{2\pi i} \oint_C \frac{f(z)}{(z - z_0)^{n+1}} dz, n = 1, 2, \dots$$

PROOF: For $f'(z_0)$

$$f'(z_0) = \lim_{\Delta z \rightarrow 0} \frac{f(z_0 + \Delta z) f(z_0)}{\Delta z}$$

Applying Cauchy's Integral formula

$$\begin{aligned} f'(z_0) &= \lim_{\Delta z \rightarrow 0} \frac{1}{2\pi i \Delta z} \left[\int_C \frac{f(z)}{z - (z_0 + \Delta z)} dz - \int_C \frac{f(z)}{z - z_0} dz \right] \\ &= \frac{1}{2\pi i} \int_C \frac{f(z)}{(z - z_0)^2} dz + \lim_{\Delta z \rightarrow 0} \frac{\Delta z}{2\pi i} \int_C \frac{f(z)}{(z - z_0 - \Delta z)(z - z_0)^2} dz \end{aligned}$$

So we need to establish that the second term on the right is zero. On C the function $f(z)$ is continuous. Hence $f(z)$ is bounded in absolute value on C , say $|f(z)| < M$. Let d be the distance of the point or points of C which are closest to z_0 . Then for all z on C , $|z - z_0| \geq d$ hence $|z - z_0|^{-1} \leq \frac{1}{d}$. Also, if $|\Delta z| \leq \frac{d}{2}$, then for all z on C we have the inequality $|z - z_0 - \Delta z| \geq \frac{d}{2}$ hence $|z - z_0 - \Delta z|^{-1} \leq \frac{2}{d}$. Denoting the length of C by L

$$\left| \frac{\Delta z}{2\pi i} \int_C \frac{f(z)}{(z - z_0 - \Delta z)(z - z_0)^2} dz \right| < \frac{|\Delta z|}{2\pi} \frac{M}{\frac{1}{2}dd^2} L$$

As Δz approaches zero the right hand side approaches zero. The general formula follows by induction.

A.8 Laurent's Theorem

If $f(z)$ is analytic on two concentric circles with centre a and in the annulus between them, then $f(z)$ can be represented by the *Laurent series*

$$f(z) = \sum_{n=0}^{\infty} b_n (z - a)^n + \sum_{n=1}^{\infty} \frac{c_n}{(z - a)^n}$$

where

$$\begin{aligned} b_n &= \frac{1}{2\pi i} \oint_C \frac{f(z^*)}{(z^* - a)^{n+1}} dz^* \\ c_n &= \frac{1}{2\pi i} \oint_C (z^* - a)^{n-1} f(z^*) dz^* \end{aligned}$$

each integral being taken in the counter-clockwise direction around any simple closed path, C , which lies in the annulus and encloses the inner circle. This series converges and represents $f(z)$ in the open annulus obtained from the given annulus by continuously increasing the circle C_1 and decreasing C_2 until each if the two circles reaches a point where $f(z)$ is singular.

For a proof see Kreysig [23, Section 16.7]. A common case occurs when $z = a$ is the only singular point of $f(z)$ in C_2 . Then the Laurent expansion converges for all z in C_1 except at $z = a$.

A.9 Residues

If $f(z)$ is analytic in the neighbourhood of a point $z = a$, then by Cauchy's integral theorem

$$\oint_C f(z) dz = 0$$

for any closed path, C , in that neighbourhood. If, however, $f(z)$ has an isolated singularity at $z = a$ and a lies in the interior of C , then we may represent $f(z)$ by the Laurent series

$$f(z) = \sum_{n=0}^{\infty} b_n (z-a)^n + \frac{c_1}{(z-a)} + \frac{c_2}{(z-a)^2} + \dots$$

which converges on the domain $0 < |z - a| < R$. Consequently

$$\oint_C f(z) dz = 2\pi i c_1$$

c_1 is called the *residue* of $f(z)$ at $z = a$. The integral of $f(z)$ over C can be extended to paths that contain finitely many singular points.

A.10 Cauchy's Argument Principle

A *meromorphic* function on an open subset, D , of the complex plane is a function that is holomorphic on all D except at a set of isolated points (the *poles* of the function) at which it must have a Laurent series. If $f(z)$ is a meromorphic function inside and on a closed contour, C , and has no poles or zeros on C , then

$$\oint_C \frac{f'(z)}{f(z)} dz = 2\pi i (k - m)$$

where k and m denote the number of zeros and poles, respectively, of $f(z)$ inside C . Each zero and pole is counted as many times as its multiplicity and order, respectively, indicate.

PROOF: Let z_N be a zero of $f(z)$ with multiplicity k , then

$$f(z) = (z - z_N)^k g(z)$$

where $g(z_N) \neq 0$. Differentiating

$$f'(z) = k(z - z_N)^{k-1} g(z) + (z - z_N)^k g'(z)$$

and

$$\frac{f'(z)}{f(z)} = \frac{k}{z - z_N} + \frac{g'(z)}{g(z)}$$

Since $g(z_N) \neq 0$, $\frac{g'(z)}{g(z)}$ has no singularities and is analytic at z_N and the residue of $\frac{f'(z)}{f(z)}$ at z_N is k .

Similarly, let z_P be a pole of $f(z)$ with order k , then

$$f(z) = (z - z_P)^{-m} h(z)$$

where $h(z_P) \neq 0$. Differentiating

$$f'(z) = -m(z - z_P)^{-m-1} h(z) + (z - z_P)^{-m} h'(z)$$

and

$$\frac{f'(z)}{f(z)} = \frac{-m}{z - z_P} + \frac{h'(z)}{h(z)}$$

Since $h(z_P) \neq 0$, $\frac{h'(z)}{h(z)}$ has no singularities and is analytic at z_P and the residue of $\frac{f'(z)}{f(z)}$ at z_P is $-m$.

A.11 Rouché's Theorem

If $f(z)$ and $g(z)$ are analytic inside and on a closed contour C , and $|g(z)| < |f(z)|$ on C , then $f(z)$ and $f(z) + g(z)$ have the same number of zeros inside C .

Appendix B

IIR filter amplitude, phase and group-delay frequency responses

This section describes the derivation of the amplitude (magnitude) and group delay responses and their first and second partial derivatives (gradient vector and Hessian matrix) with respect to the pole and zero locations of the transfer function. Some possibilities for constructing the first and second partial derivatives are:

- numerical differentiation by evaluation of the function at a perturbed input coefficient vector
- symbolic differentiation by a symbolic algebra package. For example, *Maxima* [99] or the *OctaveForge* symbolic calculation toolbox [67]
- differentiation by hand
- so-called *automatic differentiation*: “a technology for automatically augmenting computer programs, including arbitrarily complex simulations, with statements for the computation of derivatives”. See [11, *tools*].

Below I use a mixture of differentiation by hand and symbolic differentiation to derive the formulas implemented in Octave code and use numerical differentiation to test the formulas¹.

Richards [56] shows expressions for the magnitude and group-delay of an IIR filter with an integer decimation factor, R , and transfer function

$$H(z) = \frac{N(z)}{D(z)} = K \frac{\sum_{j=0}^J n_j z^{-j}}{1 + \sum_{l=1}^L d_l z^{-Rl}} \quad (\text{B.1})$$

The polar coordinates of the zeros of $H(z)$ are assumed to be $\{z_{0j}\} = \{(R_{0j}, 0), (r_{0j}, \pm\theta_{0j})\}$. Similarly, the poles of $H(z)$ are assumed to be $\{v_{pk}\} = \{(R_{pk}, 0), (r_{pk}, \pm\theta_{pk})\}$ where $v = z^R$ (so that each pole on the v plane corresponds to R equally spaced poles on the z plane). All coefficients are allowed to be negative so that the responses are differentiable with respect to the coefficients. This means that the amplitude response derived below can be negative if the gain coefficient is negative and, consequently, the phase of the gain coefficient is not included in the phase response listed below. If $R \geq 2$, the gradients of the amplitude response are undefined for real and complex poles with radius 0.

For convenience in the following, rewrite Equation B.1 as:

$$H(z) = K \frac{z^{-J}}{z^{-RL}} \frac{\sum_{j=0}^J n_j z^{J-j}}{z^{RL} + \sum_{l=1}^L d_l z^{R(L-l)}}$$

Suppose there is a zero at $v = re^{i\theta}$, then the magnitude response due to this zero is

$$\begin{aligned} |(e^{i\omega} - re^{i\theta})| &= |(\cos \omega - r \cos \theta) + i(\sin \omega - r \sin \theta)| \\ &= \{\cos^2 \omega - 2r \cos \omega \cos \theta + r^2 \cos^2 \theta + \sin^2 \omega - 2r \sin \omega \sin \theta + r^2 \sin^2 \theta\}^{\frac{1}{2}} \end{aligned}$$

¹In the examples above, the Hessian of the phase response is calculated numerically and that matrix initialises the diagonal of the BFGS positive-definite approximation to the Hessian.

$$= \{1 - 2r \cos(\omega - \theta) + r^2\}^{\frac{1}{2}}$$

and the phase due to this zero is

$$\arg\{e^{i\omega} - re^{i\theta}\} = \arctan\left\{\frac{\sin\omega - r\sin\theta}{\cos\omega - r\cos\theta}\right\}$$

so that the group delay due to this zero is

$$\begin{aligned} -\frac{d}{d\omega} \arg\{e^{i\omega} - re^{i\theta}\} &= -\left[1 + \left(\frac{\sin\omega - r\sin\theta}{\cos\omega - r\cos\theta}\right)^2\right]^{-1} \left[\frac{(\cos\omega - r\cos\theta)\cos\omega + (\sin\omega - r\sin\theta)\sin\omega}{(\cos\omega - r\cos\theta)^2}\right] \\ &= -\frac{1 - r\cos\theta\cos\omega - r\sin\theta\sin\omega}{(\cos\omega - r\cos\theta)^2 + (\sin\omega - r\sin\theta)^2} \\ &= -\frac{1 - r\cos(\omega - \theta)}{1 - 2r\cos(\omega - \theta) + r^2} \end{aligned}$$

Using these results, for decimation factor R , $s = \frac{1}{R}$, U real zeros, V real poles, $\frac{M}{2}$ conjugate zero pairs and $\frac{Q}{2}$ conjugate pole pairs, the magnitude response of $H(z)$ is

$$\begin{aligned} A(\omega) &= K \times \frac{\prod_{j=1}^U \{1 - 2R_{0j}\cos\omega + R_{0j}^2\}^{\frac{1}{2}}}{\prod_{j=1}^V \prod_{i=0}^{R-1} \{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}\}^{\frac{1}{2}}} \times \dots \\ &\quad \dots \frac{\prod_{j=1}^{\frac{M}{2}} \{1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\}^{\frac{1}{2}}}{\prod_{j=1}^{\frac{Q}{2}} \prod_{i=0}^{R-1} \{1 - 2r_{pj}^s \cos(\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}\}^{\frac{1}{2}}} \times \dots \\ &\quad \dots \frac{\prod_{j=1}^{\frac{M}{2}} \{1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\}^{\frac{1}{2}}}{\prod_{j=1}^{\frac{Q}{2}} \prod_{i=0}^{R-1} \{1 - 2r_{pj}^s \cos(\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}\}^{\frac{1}{2}}} \end{aligned}$$

the phase response of $H(z)$ is²

$$\begin{aligned} P(\omega) &= [(V+Q)R - (U+M)]\omega + \dots \\ &\quad \dots \sum_{j=1}^U \arctan\left(\frac{\sin\omega}{\cos\omega - R_{0j}}\right) - \dots \\ &\quad \dots \sum_{j=1}^V \left\{ \sum_{i=0}^{R-1} \arctan\left(\frac{\sin\omega - R_{pj}^s \sin(s2\pi i)}{\cos\omega - R_{pj}^s \cos(s2\pi i)}\right) \right\} + \dots \\ &\quad \dots \sum_{j=1}^{\frac{M}{2}} \arctan\left(\frac{\sin 2\omega - 2r_{0j}\cos\theta_{0j}\sin\omega}{\cos 2\omega - 2r_{0j}\cos\theta_{0j}\cos\omega + r_{0j}^2}\right) - \dots \\ &\quad \dots \sum_{j=1}^{\frac{Q}{2}} \left\{ \sum_{i=0}^{R-1} \arctan\left(\frac{\sin 2\omega - 2r_{pj}^s \cos[s(\theta_{pj} + 2\pi i)]\sin\omega}{\cos 2\omega - 2r_{pj}^s \cos[s(\theta_{pj} + 2\pi i)]\cos\omega + r_{pj}^{2s}}\right) \right\} \end{aligned}$$

and the group-delay response of $H(z)$ is³

$$\begin{aligned} T(\omega) &= -[(V+Q)R - (U+M)] - \dots \\ &\quad \dots \sum_{j=1}^U \frac{1 - R_{0j}\cos\omega}{1 - 2R_{0j}\cos\omega + R_{0j}^2} + \dots \end{aligned}$$

²Note that $\arg\{K\}$ is not included here. The sign of K is included in the amplitude response. This means that the amplitude response returned may be negative. This is allowed so that $\frac{\partial A(\omega)}{\partial K}$ is well defined at $K = 0$.

³An alternative derivation of the group delay, $T(\omega)$, is:

$$\begin{aligned} H(e^{i\omega T}) &= A(\omega) e^{i\Theta(\omega)} \\ \ln H(e^{i\omega T}) &= \ln A(\omega) + i\Theta(\omega) \\ \frac{d}{d\omega} \{\ln H(e^{i\omega T})\} &= \frac{A'(\omega)}{A(\omega)} + i\Theta'(\omega) \\ T(\omega) &= -\frac{d}{d\omega} \Theta(\omega) = -\text{imag}\left\{\frac{H'(e^{i\omega T})}{H(e^{i\omega T})}\right\} \end{aligned}$$

$$\begin{aligned}
& \cdots \sum_{j=1}^V \left\{ \sum_{i=0}^{R-1} \frac{1 - R_{pj}^s \cos(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}} \right\} - \cdots \\
& \cdots \sum_{j=1}^{\frac{M}{2}} \left\{ \frac{1 - r_{0j} \cos(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \frac{1 - r_{0j} \cos(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\} + \cdots \\
& \cdots \sum_{j=1}^{\frac{Q}{2}} \left\{ \sum_{i=0}^{R-1} \frac{1 - r_{pj}^s \cos(\omega - s(\theta_{pj} + 2\pi i))}{1 - 2r_{pj}^s \cos(\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}} \right\} + \cdots \\
& \cdots \sum_{j=1}^{\frac{Q}{2}} \left\{ \sum_{i=0}^{R-1} \frac{1 - r_{pj}^s \cos(\omega + s(\theta_{pj} + 2\pi i))}{1 - 2r_{pj}^s \cos(\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}} \right\}
\end{aligned}$$

The partial derivatives of the magnitude response are

$$\begin{aligned}
\frac{\partial A(\omega)}{\partial K} &= \frac{A(\omega)}{K} \\
\frac{\partial A(\omega)}{\partial R_{0j}} &= A(\omega) \left\{ \frac{R_{0j} - \cos \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} \right\} \\
\frac{\partial A(\omega)}{\partial R_{pj}} &= -sR_{pj}^{s-1} A(\omega) \left\{ \sum_{i=0}^{R-1} \frac{R_{pj}^s - \cos(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}} \right\} \\
\frac{\partial A(\omega)}{\partial r_{0j}} &= A(\omega) \left\{ \frac{r_{0j} - \cos(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \frac{r_{0j} - \cos(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\} \\
\frac{\partial A(\omega)}{\partial \theta_{0j}} &= A(\omega) \left\{ \frac{r_{0j} \sin(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} - \frac{r_{0j} \sin(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} \right\} \\
\frac{\partial A(\omega)}{\partial r_{pj}} &= -sr_{pj}^{s-1} A(\omega) \sum_{i=0}^{R-1} \left\{ \frac{r_{pj}^s - \cos(\omega - s(\theta_{pj} + 2\pi i))}{1 - 2r_{pj}^s \cos(\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}} + \cdots \right. \\
&\quad \left. \cdots \frac{r_{pj}^s - \cos(\omega + s(\theta_{pj} + 2\pi i))}{1 - 2r_{pj}^s \cos(\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}} \right\} \\
\frac{\partial A(\omega)}{\partial \theta_{pj}} &= -sr_{pj}^s A(\omega) \sum_{i=0}^{R-1} \left\{ \frac{\sin(\omega + s(\theta_{pj} + 2\pi i))}{1 - 2r_{pj}^s \cos(\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}} - \cdots \right. \\
&\quad \left. \cdots \frac{\sin(\omega - s(\theta_{pj} + 2\pi i))}{1 - 2r_{pj}^s \cos(\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}} \right\}
\end{aligned}$$

The partial derivatives of the phase are

$$\begin{aligned}
\frac{\partial P(\omega)}{\partial K} &= 0 \\
\frac{\partial P(\omega)}{\partial R_{0j}} &= \frac{\sin \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} \\
\frac{\partial P(\omega)}{\partial R_{pj}} &= -sR_{pj}^{s-1} \sum_{i=0}^{R-1} \frac{\sin(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}}
\end{aligned}$$

For convenience, write

$$\begin{aligned}
P_{0N} &= \sin 2\omega - 2r_{0j} \cos \theta_{0j} \sin \omega \\
P_{0D} &= \cos 2\omega - 2r_{0j} \cos \theta_{0j} \cos \omega + r_{0j}^2
\end{aligned}$$

then

$$\begin{aligned}
\frac{\partial P(\omega)}{\partial r_{0j}} &= P_{0N} \frac{(2 \cos \theta_{0j} \cos \omega - 2r_{0j})}{P_{0N}^2 + P_{0D}^2} - P_{0D} \frac{2 \cos \theta_{0j} \sin \omega}{P_{0N}^2 + P_{0D}^2} \\
\frac{\partial P(\omega)}{\partial \theta_{0j}} &= P_{0D} \frac{2r_{0j} \sin \theta_{0j} \sin \omega}{P_{0N}^2 + P_{0D}^2} - P_{0N} \frac{2r_{0j} \sin \theta_{0j} \cos \omega}{P_{0N}^2 + P_{0D}^2}
\end{aligned}$$

For convenience, write

$$P_{pN} = \sin 2\omega - 2r_{pj}^s \cos [s(\theta_{pj} + 2\pi i)] \sin \omega$$

$$P_{pD} = \cos 2\omega - 2r_{pj}^s \cos [s(\theta_{pj} + 2\pi i)] \cos \omega + r_{pj}^{2s}$$

then

$$\frac{\partial P(\omega)}{\partial r_{pj}} = - \sum_{i=0}^{R-1} \left\{ P_{pD} \frac{-2sr_{pj}^{s-1} \cos [s(\theta_{pj} + 2\pi i)] \sin \omega}{P_{pN}^2 + P_{pD}^2} - P_{pN} \frac{-2sr_{pj}^{s-1} \cos [s(\theta_{pj} + 2\pi i)] \cos \omega + 2sr_{pj}^{2s-1}}{P_{pN}^2 + P_{pD}^2} \right\}$$

$$\frac{\partial P(\omega)}{\partial \theta_{pj}} = - \sum_{i=0}^{R-1} \left\{ P_{pD} \frac{2sr_{pj}^s \sin [s(\theta_{pj} + 2\pi i)] \sin \omega}{P_{pN}^2 + P_{pD}^2} - P_{pN} \frac{2sr_{pj}^s \sin [s(\theta_{pj} + 2\pi i)] \cos \omega}{P_{pN}^2 + P_{pD}^2} \right\}$$

The partial derivatives of the group delay are

$$\begin{aligned} \frac{\partial T(\omega)}{\partial K} &= 0 \\ \frac{\partial T(\omega)}{\partial R_{0j}} &= \frac{2R_{0j} - (R_{0j}^2 + 1) \cos \omega}{[1 - 2R_{0j} \cos \omega + R_{0j}^2]^2} \\ \frac{\partial T(\omega)}{\partial R_{pj}} &= sR_{pj}^{s-1} \sum_{i=0}^{R-1} \frac{(R_{pj}^{2s} + 1) \cos (\omega - s2\pi i) - 2R_{pj}^s}{[1 - 2R_{pj}^s \cos (\omega - s2\pi i) + R_{pj}^{2s}]^2} \\ \frac{\partial T(\omega)}{\partial r_{0j}} &= \frac{2r_{0j} - (r_{0j}^2 + 1) \cos (\omega - \theta_{0j})}{[1 - 2r_{0j} \cos (\omega - \theta_{0j}) + r_{0j}^2]^2} + \frac{2r_{0j} - (r_{0j}^2 + 1) \cos (\omega + \theta_{0j})}{[1 - 2r_{0j} \cos (\omega + \theta_{0j}) + r_{0j}^2]^2} \\ \frac{\partial T(\omega)}{\partial \theta_{0j}} &= \frac{r_{0j}(r_{0j}^2 - 1) \sin (\omega - \theta_{0j})}{[1 - 2r_{0j} \cos (\omega - \theta_{0j}) + r_{0j}^2]^2} - \frac{r_{0j}(r_{0j}^2 - 1) \sin (\omega + \theta_{0j})}{[1 - 2r_{0j} \cos (\omega + \theta_{0j}) + r_{0j}^2]^2} \\ \frac{\partial T(\omega)}{\partial r_{pj}} &= sr_{pj}^{s-1} \sum_{i=0}^{R-1} \left\{ \frac{(r_{pj}^{2s} + 1) \cos (\omega - s(\theta_{pj} + 2\pi i)) - 2r_{pj}^s}{[1 - 2r_{pj}^s \cos (\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}]^2} + \dots \right. \\ &\quad \left. \dots \frac{(r_{pj}^{2s} + 1) \cos (\omega + s(\theta_{pj} + 2\pi i)) - 2r_{pj}^s}{[1 - 2r_{pj}^s \cos (\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}]^2} \right\} \\ \frac{\partial T(\omega)}{\partial \theta_{pj}} &= sr_{pj}^s (1 - r_{pj}^{2s}) \sum_{i=0}^{R-1} \left\{ \frac{\sin (\omega - s(\theta_{pj} + 2\pi i))}{[1 - 2r_{pj}^s \cos (\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}]^2} - \dots \right. \\ &\quad \left. \dots \frac{\sin (\omega + s(\theta_{pj} + 2\pi i))}{[1 - 2r_{pj}^s \cos (\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}]^2} \right\} \end{aligned}$$

The second partial derivatives of the magnitude response are (with $j \neq k$, and otherwise assumed to vary across all indexes):

$$\begin{aligned} \frac{\partial^2 A(\omega)}{\partial K^2} &= 0 \\ \frac{\partial^2 A(\omega)}{\partial K \partial R_{0j}} &= \frac{1}{K} \frac{\partial A(\omega)}{\partial R_{0j}} \\ \frac{\partial^2 A(\omega)}{\partial K \partial R_{pj}} &= \frac{1}{K} \frac{\partial A(\omega)}{\partial R_{pj}} \\ \frac{\partial^2 A(\omega)}{\partial K \partial r_{0j}} &= \frac{1}{K} \frac{\partial A(\omega)}{\partial r_{0j}} \\ \frac{\partial^2 A(\omega)}{\partial K \partial \theta_{0j}} &= \frac{1}{K} \frac{\partial A(\omega)}{\partial \theta_{0j}} \\ \frac{\partial^2 A(\omega)}{\partial K \partial r_{pj}} &= \frac{1}{K} \frac{\partial A(\omega)}{\partial r_{pj}} \\ \frac{\partial^2 A(\omega)}{\partial K \partial \theta_{pj}} &= \frac{1}{K} \frac{\partial A(\omega)}{\partial \theta_{pj}} \\ \frac{\partial^2 A(\omega)}{\partial R_{0j}^2} &= A(\omega) \frac{\sin^2 \omega}{[1 - 2R_{0j} \cos \omega + R_{0j}^2]^2} \end{aligned}$$

$$\begin{aligned}
\frac{\partial^2 A(\omega)}{\partial R_{0k} \partial R_{0j}} &= \frac{\partial A(\omega)}{\partial R_{0k}} \left\{ \frac{R_{0j} - \cos \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial R_{pj} \partial R_{0j}} &= \frac{\partial A(\omega)}{\partial R_{pj}} \left\{ \frac{R_{0j} - \cos \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{0j} \partial R_{0j}} &= \frac{\partial A(\omega)}{\partial r_{0j}} \left\{ \frac{R_{0j} - \cos \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{0j} \partial R_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{0j}} \left\{ \frac{R_{0j} - \cos \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{pj} \partial R_{0j}} &= \frac{\partial A(\omega)}{\partial r_{pj}} \left\{ \frac{R_{0j} - \cos \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{pj} \partial R_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{pj}} \left\{ \frac{R_{0j} - \cos \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial R_{pj}^2} &= -sR_{pj}^{s-1} \frac{\partial A(\omega)}{\partial R_{pj}} \left\{ \sum_{i=0}^{R-1} \frac{R_{pj}^s - \cos(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}} \right\} + \dots \\
&\dots sR_{pj}^{s-2} A(\omega) \sum_{i=0}^{R-1} \left\{ \frac{R_{pj}^{3s} + (s-3)R_{pj}^{2s} \cos(\omega - s2\pi i)}{[1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}]^2} + \dots \right. \\
&\dots \left. \frac{(2 \cos^2(\omega - s2\pi i) - 2s + 1)R_{pj}^s + (s-1)\cos(\omega - s2\pi i)}{[1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}]^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial R_{pk} \partial R_{pj}} &= -sR_{pj}^{s-1} \frac{\partial A(\omega)}{\partial R_{pk}} \left\{ \sum_{i=0}^{R-1} \frac{R_{pj}^s - \cos(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{0j} \partial R_{pj}} &= -sR_{pj}^{s-1} \frac{\partial A(\omega)}{\partial r_{0j}} \left\{ \sum_{i=0}^{R-1} \frac{R_{pj}^s - \cos(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{0j} \partial R_{pj}} &= -sR_{pj}^{s-1} \frac{\partial A(\omega)}{\partial \theta_{0j}} \left\{ \sum_{i=0}^{R-1} \frac{R_{pj}^s - \cos(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{pj} \partial R_{pj}} &= -sR_{pj}^{s-1} \frac{\partial A(\omega)}{\partial r_{pj}} \left\{ \sum_{i=0}^{R-1} \frac{R_{pj}^s - \cos(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{pj} \partial R_{pj}} &= -sR_{pj}^{s-1} \frac{\partial A(\omega)}{\partial \theta_{pj}} \left\{ \sum_{i=0}^{R-1} \frac{R_{pj}^s - \cos(\omega - s2\pi i)}{1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{0j}^2} &= \frac{\partial A(\omega)}{\partial r_{0j}} \left\{ \frac{r_{0j} - \cos(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \frac{r_{0j} - \cos(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\} + \dots \\
&\dots A(\omega) \left\{ \frac{2 \sin^2(\omega - \theta_{0j})}{[1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2]^2} - \frac{1}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \dots \right. \\
&\dots \left. \frac{2 \sin^2(\omega + \theta_{0j})}{[1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2]^2} - \frac{1}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{0k} \partial r_{0j}} &= \frac{\partial A(\omega)}{\partial r_{0k}} \left\{ \frac{r_{0j} - \cos(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \frac{r_{0j} - \cos(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{0j} \partial r_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{0j}} \left\{ \frac{r_{0j} - \cos(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \frac{r_{0j} - \cos(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\} + \dots \\
&\dots A(\omega) \left\{ \frac{(1 - r_{0j}^2) \sin(\omega + \theta_{0j})}{[1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2]^2} - \frac{(1 - r_{0j}^2) \sin(\omega - \theta_{0j})}{[1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2]^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{0k} \partial r_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{0k}} \left\{ \frac{r_{0j} - \cos(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \frac{r_{0j} - \cos(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{pj} \partial r_{0j}} &= \frac{\partial A(\omega)}{\partial r_{pj}} \left\{ \frac{r_{0j} - \cos(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \frac{r_{0j} - \cos(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial^2 A(\omega)}{\partial \theta_{pj} \partial r_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{pj}} \left\{ \frac{r_{0j} - \cos(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} + \frac{r_{0j} - \cos(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{0j}^2} &= \frac{\partial A(\omega)}{\partial \theta_{0j}} \left\{ \frac{r_{0j} \sin(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} - \frac{r_{0j} \sin(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} \right\} + \dots \\
\dots A(\omega) &\left\{ \frac{(r_{0j} + r_{0j}^3) \cos(\omega + \theta_{0j}) - 2r_{0j}^2}{[1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2]^2} + \frac{(r_{0j} + r_{0j}^3) \cos(\omega - \theta_{0j}) - 2r_{0j}^2}{[1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2]^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{0k} \partial \theta_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{0k}} \left\{ \frac{r_{0j} \sin(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} - \frac{r_{0j} \sin(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{pj} \partial \theta_{0j}} &= \frac{\partial A(\omega)}{\partial r_{pj}} \left\{ \frac{r_{0j} \sin(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} - \frac{r_{0j} \sin(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{pj} \partial \theta_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{pj}} \left\{ \frac{r_{0j} \sin(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} - \frac{r_{0j} \sin(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} \right\}
\end{aligned}$$

For convenience set

$$\begin{aligned}
N_n &= r_{pj}^{2s-1} - r_{pj}^{s-1} \cos(\omega - s(\theta_{pj} + 2\pi i)) \\
N_p &= r_{pj}^{2s-1} - r_{pj}^{s-1} \cos(\omega + s(\theta_{pj} + 2\pi i)) \\
D_n &= 1 - 2r_{pj}^s \cos(\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s} \\
D_p &= 1 - 2r_{pj}^s \cos(\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}
\end{aligned}$$

so, to continue

$$\begin{aligned}
\frac{\partial^2 A(\omega)}{\partial r_{pj}^2} &= -s \frac{\partial A(\omega)}{\partial r_{pj}} \sum_{i=0}^{R-1} \left\{ \frac{N_n}{D_n} + \frac{N_p}{D_p} \right\} - \dots \\
&\dots sA(\omega) \sum_{i=0}^{R-1} \left\{ \frac{(2s-1)r_{pj}^{2s-2} - (s-1)r_{pj}^{s-2} \cos(\omega - s(\theta_{pj} + 2\pi i))}{D_n} - \frac{2sN_n^2}{D_n^2} + \dots \right. \\
&\quad \left. \dots \frac{(2s-1)r_{pj}^{2s-2} - (s-1)r_{pj}^{s-2} \cos(\omega + s(\theta_{pj} + 2\pi i))}{D_p} - \frac{2sN_p^2}{D_p^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{pk} \partial r_{pj}} &= -s \frac{\partial A(\omega)}{\partial r_{pk}} \sum_{i=0}^{R-1} \left\{ \frac{N_n}{D_n} + \frac{N_p}{D_p} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{pj} \partial r_{pj}} &= -s \frac{\partial A(\omega)}{\partial \theta_{pj}} \sum_{i=0}^{R-1} \left\{ \frac{N_n}{D_n} + \frac{N_p}{D_p} \right\} - \dots \\
&\dots sA(\omega) \sum_{i=0}^{R-1} \left\{ sr_{pj}^{s-1} \sin(\omega + s(\theta_{pj} + 2\pi i)) \left[\frac{1}{D_p} - \frac{2N_p r_{pj}}{D_p^2} \right] - \dots \right. \\
&\quad \left. \dots sr_{pj}^{s-1} \sin(\omega - s(\theta_{pj} + 2\pi i)) \left[\frac{1}{D_n} - \frac{2N_n r_{pj}}{D_n^2} \right] \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{pk} \partial r_{pj}} &= -s \frac{\partial A(\omega)}{\partial \theta_{pk}} \sum_{i=0}^{R-1} \left\{ \frac{N_n}{D_n} + \frac{N_p}{D_p} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{pj}^2} &= -s \frac{\partial A(\omega)}{\partial \theta_{pj}} \sum_{i=0}^{R-1} \left\{ \frac{r_{pj}^s \sin(\omega + s(\theta_{pj} + 2\pi i))}{D_p} - \frac{r_{pj}^s \sin(\omega - s(\theta_{pj} + 2\pi i))}{D_n} \right\} - \dots \\
&\dots sA(\omega) \sum_{i=0}^{R-1} \left\{ \frac{sr_{pj}^s \cos(\omega + s(\theta_{pj} + 2\pi i))}{D_p} - \frac{2sr_{pj}^{2s} \sin^2(\omega + s(\theta_{pj} + 2\pi i))}{D_p^2} + \dots \right. \\
&\quad \left. \dots \frac{sr_{pj}^s \cos(\omega - s(\theta_{pj} + 2\pi i))}{D_n} - \frac{2sr_{pj}^{2s} \sin^2(\omega - s(\theta_{pj} + 2\pi i))}{D_n^2} \right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{pk} \partial \theta_{pj}} &= -s \frac{\partial A(\omega)}{\partial \theta_{pk}} \sum_{i=0}^{R-1} \left\{ \frac{r_{pj}^s \sin(\omega + s(\theta_{pj} + 2\pi i))}{D_p} - \frac{r_{pj}^s \sin(\omega - s(\theta_{pj} + 2\pi i))}{D_n} \right\}
\end{aligned}$$

The non-zero second partial derivatives of the delay response are

$$\frac{\partial^2 T(\omega)}{\partial R_{0j}^2} = \frac{2(R_{0j}^3 \cos \omega - 3R_{0j}^2 + 3R_{0j} \cos \omega - \cos 2\omega)}{[1 - 2R_{0j} \cos \omega + R_{0j}^2]^3}$$

$$\begin{aligned}
\frac{\partial^2 T(\omega)}{\partial R_{pj}^2} &= -s R_{pj}^{s-2} \sum_{i=0}^{R-1} \left\{ \frac{(s+1) \cos(\omega - s2\pi i) R_{pj}^{4s} + 2[(s-1) \cos^2(\omega - s2\pi i) - (2s+1)] R_{pj}^{3s}}{[1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}]^3} + \dots \right. \\
&\quad \dots \left. \frac{6 \cos(\omega - s2\pi i) R_{pj}^{2s} + 2[(2s-1) - (s+1) \cos^2(\omega - s2\pi i)] R_{pj}^s - (s-1) \cos(\omega - s2\pi i)}{[1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}]^3} \right\} \\
\frac{\partial^2 T(\omega)}{\partial r_{0j}^2} &= \frac{2[r_{0j}^3 \cos(\omega - \theta_{0j}) - 3r_{0j}^2 + 3r_{0j} \cos(\omega - \theta_{0j}) - \cos 2(\omega - \theta_{0j})]}{[1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2]^3} + \dots \\
&\quad \frac{2[r_{0j}^3 \cos(\omega + \theta_{0j}) - 3r_{0j}^2 + 3r_{0j} \cos(\omega + \theta_{0j}) - \cos 2(\omega + \theta_{0j})]}{[1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2]^3} \\
\frac{\partial^2 T(\omega)}{\partial \theta_{0j} \partial r_{0j}} &= \frac{\sin(\omega + \theta_{0j}) [r_{0j}^4 + 2r_{0j}^3 \cos(\omega + \theta_{0j}) - 6r_{0j}^2 + 2r_{0j} \cos(\omega + \theta_{0j}) + 1]}{[1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2]^3} - \dots \\
&\quad \frac{\sin(\omega - \theta_{0j}) [r_{0j}^4 + 2r_{0j}^3 \cos(\omega - \theta_{0j}) - 6r_{0j}^2 + 2r_{0j} \cos(\omega - \theta_{0j}) + 1]}{[1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2]^3} \\
\frac{\partial^2 T(\omega)}{\partial \theta_{0j}^2} &= -\frac{r_{0j} (r_{0j}^2 - 1) [r_{0j}^2 \cos(\omega - \theta_{0j}) - 2r_{0j} (1 + \sin^2(\omega - \theta_{0j})) + \cos(\omega - \theta_{0j})]}{[1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2]^3} - \dots \\
&\quad \frac{r_{0j} (r_{0j}^2 - 1) [r_{0j}^2 \cos(\omega + \theta_{0j}) - 2r_{0j} (1 + \sin^2(\omega + \theta_{0j})) + \cos(\omega + \theta_{0j})]}{[1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2]^3} \\
\frac{\partial^2 T(\omega)}{\partial r_{pj}^2} &= -s \sum_{i=1}^{R-1} r_{pj}^{s-2} \left\{ \frac{2r_{pj}^s (s(r_{pj}^{2s} - 1) - (r_{pj}^{2s} + 1)) \cos^2(\omega + s(\theta_{pj} + 2\pi i))}{D_p^3} + \dots \right. \\
&\quad \dots \left. \frac{(s(r_{pj}^{4s} - 1) + (r_{pj}^{4s} + 6r_{pj}^{2s} + 1)) \cos(\omega + s(\theta_{pj} + 2\pi i)) - 4sr_{pj}^s (r_{pj}^{2s} - 1) - 2r_{pj}^s (r_{pj}^{2s} + 1)}{D_p^3} + \dots \right. \\
&\quad \dots \left. \frac{2r_{pj}^s (s(r_{pj}^{2s} - 1) - (r_{pj}^{2s} + 1)) \cos^2(\omega - s(\theta_{pj} + 2\pi i))}{D_n^3} + \dots \right. \\
&\quad \dots \left. \frac{(s(r_{pj}^{4s} - 1) + (r_{pj}^{4s} + 6r_{pj}^{2s} + 1)) \cos(\omega - s(\theta_{pj} + 2\pi i)) - 4sr_{pj}^s (r_{pj}^{2s} - 1) - 2r_{pj}^s (r_{pj}^{2s} + 1)}{D_n^3} \right\} \\
\frac{\partial^2 T(\omega)}{\partial \theta_{pj} \partial r_{pj}} &= s^2 \sum_{i=1}^{R-1} r_{pj}^{s-1} \left\{ \frac{[2r_{pj}^s (r_{pj}^{2s} + 1) \cos(\omega - s(\theta_{pj} + 2\pi i)) + (r_{pj}^{4s} - 6r_{pj}^{2s} + 1)] \sin(\omega - s(\theta_{pj} + 2\pi i))}{D_n^3} - \dots \right. \\
&\quad \dots \left. \frac{[2r_{pj}^s (r_{pj}^{2s} + 1) \cos(\omega + s(\theta_{pj} + 2\pi i)) + (r_{pj}^{4s} - 6r_{pj}^{2s} + 1)] \sin(\omega + s(\theta_{pj} + 2\pi i))}{D_p^3} \right\} \\
\frac{\partial^2 T(\omega)}{\partial \theta_{pj}^2} &= s^2 \sum_{i=1}^{R-1} r_{pj}^s (r_{pj}^{2s} - 1) \left\{ \frac{2r_{pj}^s \cos^2(\omega + s(\theta_{pj} + 2\pi i)) + (r_{pj}^{2s} + 1) \cos(\omega + s(\theta_{pj} + 2\pi i)) - 4r_{pj}^s}{D_p^3} + \dots \right. \\
&\quad \dots \left. \frac{2r_{pj}^s \cos^2(\omega - s(\theta_{pj} + 2\pi i)) + (r_{pj}^{2s} + 1) \cos(\omega - s(\theta_{pj} + 2\pi i)) - 4r_{pj}^s}{D_n^3} \right\}
\end{aligned}$$

The results for group delay were calculated with the assistance of the *Maxima* [99] script *delay.max*.

The Octave function *iirA* calculates the amplitude, and partial derivatives and Hessian of the amplitude response of an IIR filter with decimation factor R , U real zeros, V real poles, $\frac{M}{2}$ conjugate zero pairs and $\frac{Q}{2}$ conjugate pole pairs. The Octave script *iirA_test.m* exercises *iirA*. Note that R_0 and r_0 are moved off the unit circle when testing the gradients. Likewise, Octave function *iirT* calculates the group delay, partial derivatives and Hessian of the group delay response of an IIR filter and is exercised by the test script *iirT_test.m*. Again, Octave function *iirP* calculates the phase and partial derivatives of the phase response of an IIR filter and is exercised by the test script *iirP_test.m*.

The Octave function *iirA_parallel.m* uses the *parcellfun* function included in the Octave-Forge *parallel* package [65] to test “parallelising” the calculation of amplitude response and gradients by the *iirA* function over 4 processes. (My Intel i7-7700K CPU has 4 CPUs). No benefit was seen on my PC until the frequency vector length was much longer than those used in the examples.

Appendix C

Gradient of the IIR filter amplitude response with respect to frequency

This section describes the gradient of the amplitude response, $A(\omega)$ of an IIR filter expressed in gain-pole-zero form (as derived in Section B) with respect to frequency. This section derives a formula for $\frac{\partial A(\omega)}{\partial \omega}$ and formulas for the gradients with respect to the gain-pole-zero coefficients, $\frac{\partial^2 A(\omega)}{\partial \omega \partial K}$, $\frac{\partial^2 A(\omega)}{\partial \omega \partial R_{0j}}$, $\frac{\partial^2 A(\omega)}{\partial \omega \partial R_{pj}}$, etc. In this case the denominator polynomial is *not* decimated by R . The squared amplitude response is

$$A^2(\omega) = K^2 \frac{\prod_{j=1}^U \{1 - 2R_{0j} \cos \omega + R_{0j}^2\}}{\prod_{j=1}^V \{1 - 2R_{pj} \cos \omega + R_{pj}^2\}} \frac{\prod_{j=1}^{\frac{M}{2}} \{(1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2)(1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2)\}}{\prod_{j=1}^{\frac{Q}{2}} \{(1 - 2r_{pj} \cos(\omega - \theta_{pj}) + r_{pj}^2)(1 - 2r_{pj} \cos(\omega + \theta_{pj}) + r_{pj}^2)\}}$$

The gradient of the amplitude response with respect to frequency is given by

$$\begin{aligned} \frac{1}{A(\omega)} \frac{\partial A(\omega)}{\partial \omega} &= \sum_{j=1}^U \frac{R_{0j} \sin \omega}{1 - 2R_{0j} \cos \omega + R_{0j}^2} - \sum_{j=1}^V \frac{R_{pj} \sin \omega}{1 - 2R_{pj} \cos \omega + R_{pj}^2} \dots \\ &+ \sum_{j=1}^{\frac{M}{2}} \frac{r_{0j} \sin(\omega - \theta_{0j})}{1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2} - \sum_{j=1}^{\frac{Q}{2}} \frac{r_{pj} \sin(\omega - \theta_{pj})}{1 - 2r_{pj} \cos(\omega - \theta_{pj}) + r_{pj}^2} \dots \\ &+ \sum_{j=1}^{\frac{M}{2}} \frac{r_{0j} \sin(\omega + \theta_{0j})}{1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2} - \sum_{j=1}^{\frac{Q}{2}} \frac{r_{pj} \sin(\omega + \theta_{pj})}{1 - 2r_{pj} \cos(\omega + \theta_{pj}) + r_{pj}^2} \end{aligned}$$

The gradients with respect to the coefficients are given by

$$\begin{aligned} -\frac{1}{A^2(\omega)} \frac{\partial A(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial K} + \frac{1}{A(\omega)} \frac{\partial^2 A(\omega)}{\partial \omega \partial K} &= 0 \\ -\frac{1}{A^2(\omega)} \frac{\partial A(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial R_{0j}} + \frac{1}{A(\omega)} \frac{\partial^2 A(\omega)}{\partial \omega \partial R_{0j}} &= \frac{(1 - R_{0j}^2) \sin \omega}{(1 - 2R_{0j} \cos \omega + R_{0j}^2)^2} \\ -\frac{1}{A^2(\omega)} \frac{\partial A(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial R_{pj}} + \frac{1}{A(\omega)} \frac{\partial^2 A(\omega)}{\partial \omega \partial R_{pj}} &= -\frac{(1 - R_{pj}^2) \sin \omega}{(1 - 2R_{pj} \cos \omega + R_{pj}^2)^2} \\ -\frac{1}{A^2(\omega)} \frac{\partial A(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial r_{0j}} + \frac{1}{A(\omega)} \frac{\partial^2 A(\omega)}{\partial \omega \partial r_{0j}} &= \frac{(1 - r_{0j}^2) \sin(\omega - \theta_{0j})}{(1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2)^2} + \frac{(1 - r_{0j}^2) \sin(\omega + \theta_{0j})}{(1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2)^2} \\ -\frac{1}{A^2(\omega)} \frac{\partial A(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial \theta_{0j}} + \frac{1}{A(\omega)} \frac{\partial^2 A(\omega)}{\partial \omega \partial \theta_{0j}} &= -\frac{r_{0j} (1 + r_{0j}^2) \cos(\omega - \theta_{0j}) - 2r_{0j}^2}{(1 - 2r_{0j} \cos(\omega - \theta_{0j}) + r_{0j}^2)^2} + \frac{r_{0j} (1 + r_{0j}^2) \cos(\omega + \theta_{0j}) - 2r_{0j}^2}{(1 - 2r_{0j} \cos(\omega + \theta_{0j}) + r_{0j}^2)^2} \\ -\frac{1}{A^2(\omega)} \frac{\partial A(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial r_{pj}} + \frac{1}{A(\omega)} \frac{\partial^2 A(\omega)}{\partial \omega \partial r_{pj}} &= -\frac{(1 - r_{pj}^2) \sin(\omega - \theta_{pj})}{(1 - 2r_{pj} \cos(\omega - \theta_{pj}) + r_{pj}^2)^2} - \frac{(1 - r_{pj}^2) \sin(\omega + \theta_{pj})}{(1 - 2r_{pj} \cos(\omega + \theta_{pj}) + r_{pj}^2)^2} \\ -\frac{1}{A^2(\omega)} \frac{\partial A(\omega)}{\partial \omega} \frac{\partial A(\omega)}{\partial \theta_{pj}} + \frac{1}{A(\omega)} \frac{\partial^2 A(\omega)}{\partial \omega \partial \theta_{pj}} &= \frac{r_{pj} (1 + r_{pj}^2) \cos(\omega - \theta_{pj}) - 2r_{pj}^2}{(1 - 2r_{pj} \cos(\omega - \theta_{pj}) + r_{pj}^2)^2} - \frac{r_{pj} (1 + r_{pj}^2) \cos(\omega + \theta_{pj}) - 2r_{pj}^2}{(1 - 2r_{pj} \cos(\omega + \theta_{pj}) + r_{pj}^2)^2} \end{aligned}$$

Appendix D

Allpass filter frequency response

D.1 Allpass filter phase response

This section describes the phase response and the gradient of the phase response of an all-pass filter in terms of the pole and zero locations of the transfer function. If $D(z)$ has V real zeros and $\frac{Q}{2}$ pairs of complex conjugate zeros within the unit circle

$$D(z) = \left\{ \prod_{k=1}^V z - R_{pk} \right\} \left\{ \prod_{k=1}^{\frac{Q}{2}} (z - r_{pk} e^{i\theta_{pk}})(z - r_{pk} e^{-i\theta_{pk}}) \right\}$$

where R_{pk} are the real zeros and $r_{pk} e^{\pm i\theta_{pk}}$ are the complex conjugate zeros of $D(z)$.

The all-pass filter transfer function with decimation factor, R , is:

$$\begin{aligned} A(z) &= z^{-R(V+Q)} \frac{D(z^{-R})}{D(z^R)} \\ &= \left\{ \prod_{k=1}^V \frac{z^{-R} - R_{pk}}{1 - R_{pk} z^{-R}} \right\} \left\{ \prod_{k=1}^{\frac{Q}{2}} \frac{z^{-R} - r_{pk} e^{i\theta_{pk}}}{1 - r_{pk} e^{-i\theta_{pk}} z^{-R}} \right\} \left\{ \prod_{k=1}^{\frac{Q}{2}} \frac{z^{-R} - r_{pk} e^{-i\theta_{pk}}}{1 - r_{pk} e^{i\theta_{pk}} z^{-R}} \right\} \end{aligned}$$

The magnitude response of $A(z)$ is $|A(e^{i\omega})| = 1$. The phase response of $A(z)$ is

$$\begin{aligned} P(\omega) &= - \sum_{k=1}^V \left\{ \arctan \frac{R_{pk} \sin R\omega}{1 - R_{pk} \cos R\omega} + \arctan \frac{\sin R\omega}{\cos R\omega - R_{pk}} \right\} \dots \\ &\quad - \sum_{k=1}^{\frac{Q}{2}} \left\{ \arctan \frac{r_{pk} \sin (R\omega + \theta_{pk})}{1 - r_{pk} \cos (R\omega + \theta_{pk})} - \arctan \frac{\sin R\omega + r_{pk} \sin \theta_{pk}}{\cos R\omega - r_{pk} \cos \theta_{pk}} \right\} \dots \\ &\quad - \sum_{k=1}^{\frac{Q}{2}} \left\{ \arctan \frac{r_{pk} \sin (R\omega - \theta_{pk})}{1 - r_{pk} \cos (R\omega - \theta_{pk})} - \arctan \frac{\sin R\omega - r_{pk} \sin \theta_{pk}}{\cos R\omega - r_{pk} \cos \theta_{pk}} \right\} \end{aligned}$$

The partial derivatives of the phase response are:

$$\begin{aligned} \frac{\partial P(\omega)}{\partial R_{pk}} &= - \frac{2 \sin R\omega}{R_{pk}^2 - 2R_{pk} \cos R\omega + 1} \\ \frac{\partial P(\omega)}{\partial r_{pk}} &= - \frac{2 \sin (R\omega - \theta_{pk})}{r_{pk}^2 - 2r_{pk} \cos (R\omega - \theta_{pk}) + 1} - \frac{2 \sin (R\omega + \theta_{pk})}{r_{pk}^2 - 2r_{pk} \cos (R\omega + \theta_{pk}) + 1} \\ \frac{\partial P(\omega)}{\partial \theta_{pk}} &= - \frac{2r_{pk}^2 - 2r_{pk} \cos (R\omega - \theta_{pk})}{r_{pk}^2 - 2r_{pk} \cos (R\omega - \theta_{pk}) + 1} + \frac{2r_{pk}^2 - 2r_{pk} \cos (R\omega + \theta_{pk})}{r_{pk}^2 - 2r_{pk} \cos (R\omega + \theta_{pk}) + 1} \end{aligned}$$

The second partial derivatives of the phase response (on the diagonal of the Hessian matrix only) are:

$$\frac{\partial^2 P(\omega)}{\partial R_{pk}^2} = \frac{4 [R_{pk} - \cos R\omega] \sin R\omega}{[R_{pk}^2 - 2R_{pk} \cos R\omega + 1]^2}$$

$$\begin{aligned}\frac{\partial^2 P(\omega)}{\partial r_{pk}^2} &= \frac{4[r_{pk} - \cos(R\omega - \theta_{pk})]\sin(R\omega - \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1]^2} + \frac{4[r_{pk} - \cos(R\omega + \theta_{pk})]\sin(R\omega + \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1]^2} \\ \frac{\partial^2 P(\omega)}{\partial \theta_{pk}^2} &= \frac{2r_{pk}\sin(R\omega - \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1} - \frac{4[r_{pk}^2 - r_{pk}\cos(R\omega - \theta_{pk})]r_{pk}\sin(R\omega - \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1]^2} \dots \\ &\quad \frac{2r_{pk}\sin(R\omega + \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1} - \frac{4[r_{pk}^2 - r_{pk}\cos(R\omega + \theta_{pk})]r_{pk}\sin(R\omega + \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1]^2}\end{aligned}$$

The second partial derivatives of the phase response (on the diagonal of the Hessian matrix only) are:

$$\begin{aligned}\frac{\partial^2 P(\omega)}{\partial R_{pk}^2} &= \frac{4[R_{pk} - \cos R\omega]\sin R\omega}{[R_{pk}^2 - 2R_{pk}\cos R\omega + 1]^2} \\ \frac{\partial^2 P(\omega)}{\partial r_{pk}^2} &= \frac{4[r_{pk} - \cos(R\omega - \theta_{pk})]\sin(R\omega - \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1]^2} + \frac{4[r_{pk} - \cos(R\omega + \theta_{pk})]\sin(R\omega + \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1]^2} \\ \frac{\partial^2 P(\omega)}{\partial \theta_{pk}^2} &= \frac{2r_{pk}\sin(R\omega - \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1} - \frac{4[r_{pk}^2 - r_{pk}\cos(R\omega - \theta_{pk})]r_{pk}\sin(R\omega - \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1]^2} \dots \\ &\quad \frac{2r_{pk}\sin(R\omega + \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1} - \frac{4[r_{pk}^2 - r_{pk}\cos(R\omega + \theta_{pk})]r_{pk}\sin(R\omega + \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1]^2}\end{aligned}$$

The Octave function `allpassP` calculates the phase and partial derivatives of the phase response of an allpass IIR filter and is exercised by the test script `allpassP_test.m`.

D.2 Allpass filter group delay response

This section describes the group delay response and the gradient of the group delay response of an all-pass filter in terms of the pole and zero locations of the transfer function. The phase response, $P(\omega)$, of the allpass filter is derived in Section D.1. The group delay of the allpass filter is:

$$\begin{aligned}T(\omega) &= -R \sum_{k=1}^V \frac{R_{pk}^2 - 1}{R_{pk}^2 - 2R_{pk}\cos R\omega + 1} \dots \\ &\quad -R \sum_{k=1}^{\frac{Q}{2}} \frac{r_{pk}^2 - 1}{r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1} \dots \\ &\quad -R \sum_{k=1}^{\frac{Q}{2}} \frac{r_{pk}^2 - 1}{r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1}\end{aligned}$$

The partial derivatives of the group delay with respect to the coefficients are:

$$\begin{aligned}\frac{\partial T(\omega)}{\partial R_{pk}} &= 2R \frac{(R_{pk}^2 + 1)\cos R\omega - 2R_{pk}}{[R_{pk}^2 - 2R_{pk}\cos R\omega + 1]^2} \\ \frac{\partial T(\omega)}{\partial r_{pk}} &= 2R \frac{(r_{pk}^2 + 1)\cos(R\omega + \theta_{pk}) - 2r_{pk}}{[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1]^2} + 2R \frac{(r_{pk}^2 + 1)\cos(R\omega - \theta_{pk}) - 2r_{pk}}{[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1]^2} \\ \frac{\partial T(\omega)}{\partial \theta_{pk}} &= 2R \frac{r_{pk}(r_{pk}^2 - 1)\sin(R\omega + \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1]^2} - 2R \frac{r_{pk}(r_{pk}^2 - 1)\sin(R\omega - \theta_{pk})}{[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1]^2}\end{aligned}$$

The second partial derivatives of the group delay response (on the diagonal of the Hessian matrix only) are:

$$\frac{\partial^2 T(\omega)}{\partial R_{pk}^2} = -4R \frac{R_{pk}^3 \cos R\omega - 3R_{pk}^2 + 3R_{pk} \cos R\omega + 1 - 2\cos^2 R\omega}{[R_{pk}^2 - 2R_{pk}\cos R\omega + 1]^3}$$

$$\begin{aligned} \frac{\partial^2 T(\omega)}{\partial r_{pk}^2} &= -4R \frac{r_{pk}^3 \cos(R\omega + \theta_{pk}) - 3r_{pk}^2 + 3r_{pk} \cos(R\omega + \theta_{pk}) + 1 - 2\cos^2(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk} \cos(R\omega + \theta_{pk}) + 1\right]^3} \dots \\ &\quad - 4R \frac{r_{pk}^3 \cos(R\omega - \theta_{pk}) - 3r_{pk}^2 + 3r_{pk} \cos(R\omega - \theta_{pk}) + 1 - 2\cos^2(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk} \cos(R\omega - \theta_{pk}) + 1\right]^3} \dots \\ \frac{\partial^2 T(\omega)}{\partial \theta_{pk}^2} &= 2R \frac{r_{pk}^5 \cos(R\omega + \theta_{pk}) - 2r_{pk}^2 (r_{pk}^2 - 1) [1 + \sin^2(R\omega + \theta_{pk})] - r_{pk} \cos(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk} \cos(R\omega + \theta_{pk}) + 1\right]^3} \dots \\ &\quad + 2R \frac{r_{pk}^5 \cos(R\omega - \theta_{pk}) - 2r_{pk}^2 (r_{pk}^2 - 1) [1 + \sin^2(R\omega - \theta_{pk})] - r_{pk} \cos(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk} \cos(R\omega - \theta_{pk}) + 1\right]^3} \end{aligned}$$

The Octave function *allpassT* calculates the phase and partial derivatives of the group delay response of an allpass IIR filter and is exercised by the test script *allpassT_test.m*.

Appendix E

Gradients of the state variable filter frequency response

Section 2.9.3 shows results for the gradients of the complex response, $H(z)$, with respect to the matrix components of the corresponding state variable filter. This section shows results for the gradients of the squared-magnitude, phase and group delay of a state variable filter. In the following, the components of the state variable filter matrixes A , B , C and D are represented by α , β , γ and δ or more generally, x . The matrix components may themselves be functions of other variables (the k and c coefficients of a Schur lattice filter, for example).

E.1 Gradients of the state variable filter complex frequency response

If the components of the state variable matrixes are functions of a variable x then given $\frac{dA}{dx}$, $\frac{dB}{dx}$, $\frac{dC}{dx}$ and $\frac{dD}{dx}$ and defining $R = (e^{i\omega} I - A)^{-1}$, the gradients of the complex frequency response are:

$$\begin{aligned}\frac{dH}{dx} &= \frac{dC}{dx} RB + CR \frac{dA}{dx} RB + CR \frac{dB}{dx} + \frac{dD}{dx} \\ \frac{d^2H}{dxd\omega} &= -ie^{i\omega} \left[\frac{dC}{dx} RRB + CR \frac{dA}{dx} RRB + CRR \frac{dA}{dx} RB + CRR \frac{dB}{dx} \right] \\ \frac{d^2H}{dx^2} &= \frac{d^2C}{dx^2} RB + 2 \frac{dC}{dx} R \frac{dA}{dx} RB + 2 \frac{dC}{dx} R \frac{dB}{dx} + 2CR \frac{dA}{dx} R \frac{dA}{dx} RB + CR \frac{d^2A}{dx^2} RB + 2CR \frac{dA}{dx} R \frac{dB}{dx} + CR \frac{d^2B}{dx^2} + \frac{d^2D}{dx^2}\end{aligned}$$

and

$$\begin{aligned}\frac{d^3H}{dx^2d\omega} &= -ie^{i\omega} \left[\frac{d^2C}{dx^2} RRB + 2 \frac{dC}{dx} R \frac{dA}{dx} RRB + 2 \frac{dC}{dx} RR \frac{dA}{dx} RB + 2 \frac{dC}{dx} RR \frac{dB}{dx} + \dots \right. \\ &\quad \left. 2CR \frac{dA}{dx} R \frac{dA}{dx} RRB + CR \frac{d^2A}{dx^2} RRB + 2CR \frac{dA}{dx} RR \frac{dA}{dx} RB + 2CR \frac{dA}{dx} RR \frac{dB}{dx} + \dots \right. \\ &\quad \left. 2CRR \frac{dA}{dx} R \frac{dA}{dx} RB + CRR \frac{d^2A}{dx^2} RB + 2CRR \frac{dA}{dx} R \frac{dB}{dx} + CRR \frac{d^2B}{dx^2} \right]\end{aligned}$$

The Octave function `Abcd2H` calculates these gradients under the assumption that the components of the state variable matrixes A , B , C and D are linear functions of x and $\frac{d^2A}{dx^2} = \frac{d^2B}{dx^2} = \frac{d^2C}{dx^2} = \frac{d^2D}{dx^2} = 0$.

E.2 State variable filter squared-magnitude response

The squared-magnitude response of a state variable filter is:

$$|H(z)|^2 = \operatorname{Re} H(z)^2 + \operatorname{Im} H(z)^2$$

The gradients of the squared-magnitude response of the filter with respect to the state variable coefficients, α , β , etc., are:

$$\frac{\partial}{\partial x} |H(z)|^2 = 2 \left[\operatorname{Im} H(z) \operatorname{Im} \frac{\partial H(z)}{\partial x} + \operatorname{Re} H(z) \operatorname{Re} \frac{\partial H(z)}{\partial x} \right]$$

where x represents the components of the state variable matrixes. The diagonal of the Hessian matrix of the squared-magnitude (that is the second differentials of the squared-magnitude) is:

$$\frac{\partial^2 |H(z)|^2}{\partial x^2} = 2 \left[\left| \frac{\partial H(z)}{\partial x} \right|^2 + \operatorname{Im} H(z) \operatorname{Im} \frac{\partial^2 H(z)}{\partial x^2} + \operatorname{Re} H(z) \operatorname{Re} \frac{\partial^2 H(z)}{\partial x^2} \right]$$

where:

$$\frac{\partial^2 H(z)}{\partial \alpha^2} = 2C(zI - A)^{-1} \frac{\partial A}{\partial \alpha} (zI - A)^{-1} \frac{\partial A}{\partial \alpha} (zI - A)^{-1} B + C(zI - A)^{-1} \frac{\partial^2 A}{\partial \alpha^2} (zI - A)^{-1} B$$

and

$$\frac{\partial^2 H(z)}{\partial \beta^2} = \frac{\partial^2 H(z)}{\partial \gamma^2} = \frac{\partial^2 H(z)}{\partial \delta^2} = 0$$

E.3 State variable filter phase response

The phase response, $P(z)$, of the state variable filter is:

$$P(z) = \tan^{-1} \frac{\operatorname{Im} H(z)}{\operatorname{Re} H(z)}$$

The gradients of the phase response of the filter with respect to the state variable coefficients, α , β , etc., are given by:

$$|H(z)|^2 \frac{\partial P(z)}{\partial x} = \operatorname{Re} H(z) \operatorname{Im} \frac{\partial H(z)}{\partial x} - \operatorname{Im} H(z) \operatorname{Re} \frac{\partial H(z)}{\partial x}$$

where x represents the components of the state variable matrixes. The diagonal of the Hessian matrix of the phase (that is the second differentials of the phase) is given by:

$$\frac{\partial |H(z)|^2}{\partial x} \frac{\partial P(z)}{\partial x} + |H(z)|^2 \frac{\partial^2 P(z)}{\partial x^2} = \operatorname{Re} H(z) \operatorname{Im} \frac{\partial^2 H(z)}{\partial x^2} - \operatorname{Im} H(z) \operatorname{Re} \frac{\partial^2 H(z)}{\partial x^2}$$

E.4 State variable filter group-delay response

The group delay response, $T(\omega)$, of the state variable filter is

$$T(\omega) = -\frac{\partial}{\partial \omega} \tan^{-1} \frac{\operatorname{Im} H(\omega)}{\operatorname{Re} H(\omega)}$$

so that

$$\left[\operatorname{Re} H(\omega)^2 + \operatorname{Im} H(\omega)^2 \right] T(\omega) = - \left[\operatorname{Re} H(\omega) \operatorname{Im} \frac{\partial H(\omega)}{\partial \omega} - \operatorname{Im} H(\omega) \operatorname{Re} \frac{\partial H(\omega)}{\partial \omega} \right]$$

where

$$\frac{\partial H(\omega)}{\partial \omega} = -ie^{i\omega} C(e^{i\omega} I - A)^{-2} B$$

The gradients of the group delay response with respect to the state variable coefficients, α , β , etc., are given by:

$$\begin{aligned} & 2 \left[\operatorname{Re} H(\omega) \operatorname{Re} \frac{\partial H(\omega)}{\partial \alpha} + \operatorname{Im} H(\omega) \operatorname{Im} \frac{\partial H(\omega)}{\partial \alpha} \right] T(\omega) + \left[\operatorname{Re} H(\omega)^2 + \operatorname{Im} H(\omega)^2 \right] \frac{\partial T(\omega)}{\partial \alpha} = \\ & - \left[\operatorname{Re} \frac{\partial H(\omega)}{\partial \alpha} \operatorname{Im} \frac{\partial H(\omega)}{\partial \omega} + \operatorname{Re} H(\omega) \operatorname{Im} \frac{\partial^2 H(\omega)}{\partial \alpha \partial \omega} - \operatorname{Im} \frac{\partial H(\omega)}{\partial \alpha} \operatorname{Re} \frac{\partial H(\omega)}{\partial \omega} - \operatorname{Im} H(\omega) \operatorname{Re} \frac{\partial^2 H(\omega)}{\partial \alpha \partial \omega} \right] \end{aligned}$$

etc. where

$$\frac{\partial^2 H(\omega)}{\partial \alpha \partial \omega} = -ie^{i\omega} C(e^{i\omega} I - A)^{-1} \left[(e^{i\omega} I - A)^{-1} \frac{\partial A}{\partial \alpha} + \frac{\partial A}{\partial \alpha} (e^{i\omega} I - A)^{-1} \right] (e^{i\omega} I - A)^{-1} B$$

$$\frac{\partial^2 H(\omega)}{\partial \beta \partial \omega} = -\imath e^{\imath \omega} C (e^{\imath \omega} I - A)^{-2}$$

$$\frac{\partial^2 H(\omega)}{\partial \gamma \partial \omega} = -\imath e^{\imath \omega} (e^{\imath \omega} I - A)^{-2} B$$

$$\frac{\partial^2 H(\omega)}{\partial \delta \partial \omega} = 0$$

The diagonal of the Hessian matrix of the group-delay (that is the second differentials of the group-delay) is given by:

$$\begin{aligned} \frac{\partial^2 |H|^2}{\partial x^2} T + 2 \frac{\partial |H|^2}{\partial x} \frac{\partial T}{\partial x} + |H|^2 \frac{\partial^2 T}{\partial x^2} &= -\operatorname{Re} \frac{\partial^2 H}{\partial x^2} \operatorname{Im} \frac{\partial H}{\partial \omega} - \operatorname{Re} \frac{\partial H}{\partial x} \operatorname{Im} \frac{\partial^2 H}{\partial x \partial \omega} - \operatorname{Re} \frac{\partial H}{\partial x} \operatorname{Im} \frac{\partial^2 H}{\partial x \partial \omega} - \operatorname{Re} H \operatorname{Im} \frac{\partial^3 H}{\partial x^2 \partial \omega} \dots \\ &\quad + \operatorname{Im} \frac{\partial^2 H}{\partial x^2} \operatorname{Re} \frac{\partial H}{\partial \omega} + \operatorname{Im} \frac{\partial H}{\partial x} \operatorname{Re} \frac{\partial^2 H}{\partial x \partial \omega} + \operatorname{Im} \frac{\partial H}{\partial x} \operatorname{Re} \frac{\partial^2 H}{\partial x \partial \omega} + \operatorname{Im} H \operatorname{Re} \frac{\partial^3 H}{\partial x^2 \partial \omega} \end{aligned}$$

where x represents the components of the state variable matrixes and:

$$\begin{aligned} \frac{\partial^3 H(\omega)}{\partial \alpha^2 \partial \omega} &= -\imath e^{\imath \omega} C (e^{\imath \omega} I - A)^{-1} \left[2 \frac{\partial A}{\partial \alpha} (e^{\imath \omega} I - A)^{-2} \frac{\partial A}{\partial \alpha} + \dots \right. \\ &\quad 2 (e^{\imath \omega} I - A)^{-1} \frac{\partial A}{\partial \alpha} (e^{\imath \omega} I - A)^{-1} \frac{\partial A}{\partial \alpha} + \dots \\ &\quad 2 \frac{\partial A}{\partial \alpha} (e^{\imath \omega} I - A)^{-1} \frac{\partial A}{\partial \alpha} (e^{\imath \omega} I - A)^{-1} + \dots \\ &\quad \left. (e^{\imath \omega} I - A)^{-1} \frac{\partial^2 A}{\partial \alpha^2} + \frac{\partial^2 A}{\partial \alpha^2} (e^{\imath \omega} I - A)^{-1} \right] (e^{\imath \omega} I - A)^{-1} B \end{aligned}$$

Appendix F

Constrained non-linear optimisation

The following largely follows *Ruszczynski* [7] with some contributions concerning heuristics from *Powell* [58] and *Nocedal and Wright* [47].

F.1 Newton's method for a quadratic function

Consider a quadratic approximation, $f^k(x)$, to a function $f(x)$ evaluated at a point x^k

$$f^k(x) = f(x^k) + \nabla_x f(x^k)(x - x^k) + \frac{1}{2}(x - x^k)^T \nabla_{xx}^2 f(x^k)(x - x^k)$$

where $\nabla_{xx}^2 f(x^k)$ is a positive-definite matrix¹ (the Hessian matrix). At an optimum point $\nabla_x f^k(x) = 0$ so, approximating the gradient with the first two terms of $f^k(x)$:

$$\nabla_x f(x^k) + \nabla_{xx}^2 f(x^k)(x - x^k) = 0$$

Accordingly, given a pair $(x^k, f(x^k))$, *Newton's method of tangents* gives the next approximation to the location of the optimum point as

$$x^{k+1} = x^k - \tau^k [\nabla_{xx}^2 f(x^k)]^{-1} \nabla_x f(x^k)$$

where τ^k is a stepsize.

F.2 Lagrange multipliers

Consider the problem

$$\begin{aligned} & \text{minimise} && f(x) \\ & \text{subject to} && g(x) = c \end{aligned}$$

where $g(x)$ may represent a set of constraints $\{g_j(x) \mid j \in \mathcal{K}\}$. When a contour of $g(x)$ is tangent to a contour of $f(x)$ the gradients $\nabla_x f(x)$ and $\nabla_x g(x)$ are parallel at that point. (See [47, p. 309] for a justification). The method of *Lagrange multipliers* defines an auxilliary function (or *Lagrangian*):

$$\mathcal{L}(x, \lambda) = f(x) - \lambda(g(x) - c)$$

and solves

$$\nabla_{x,\lambda} \mathcal{L}(x, \lambda) = 0$$

Note that the solution may be a saddle point of $\mathcal{L}(x, \lambda)$. The coefficients of the vector λ are the *Lagrange multipliers* (also called the *dual variables*). Define the *dual function* as

$$q(\lambda) = \min_x \mathcal{L}(x, \lambda)$$

¹ $Q \in \mathbb{R}^{n \times n}$ is positive-definite if $x^T Q x > 0$ for all nonzero $x \in \mathbb{R}^n$

The *dual problem* is

$$\begin{aligned} & \text{maximise} && q(\lambda) \\ & \text{subject to} && \lambda \geq 0 \end{aligned}$$

Bertsekas [20, Proposition 3.4.2] gives the following duality theorem:

1. If the primal dual problem (P) has an optimal solution, the dual problem (D) also has an optimal solution and the corresponding optimal values are equal.
2. In order for \tilde{x} to be an optimal primal solution and $\tilde{\lambda}$ to be an optimal dual solution, it is necessary and sufficient that \tilde{x} is primal feasible, $\tilde{\lambda} \geq 0$, $\tilde{\lambda}_j = 0$ for all $j \in \mathcal{A}(\tilde{x}) = \{j \mid g_j(\tilde{x}) = 0\}$ (the set of active constraints) and

$$\tilde{x} \in \arg \min_x \mathcal{L}(x, \tilde{\lambda})$$

F.3 Karush-Kuhn-Tucker conditions for constrained optimisation

The *Karush-Kuhn-Tucker* conditions generalise the method of Lagrange multipliers to handle inequality constraints. Consider the minimisation problem:

$$\begin{aligned} & \text{minimise} && f(x), \quad x \in \mathbb{R}^n \\ & \text{subject to} && g_i(x) \geq 0, \quad i = 1, \dots, m \\ & && \text{and} \quad h_j(x) = 0, \quad j = 1, \dots, p \end{aligned}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ and $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$ are continuous differentiable functions. Define the Lagrangian of the problem as:

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \sum_{i=1}^m \lambda_i g_i(x) - \sum_{j=1}^p \mu_j h_j(x) \triangleq f(x) - \langle \lambda, g \rangle - \langle \mu, h \rangle$$

Define a saddle point of the Lagrangian as:

$$(\tilde{x}, \tilde{\lambda}, \tilde{\mu}) \quad \text{such that} \quad \mathcal{L}(\tilde{x}, \tilde{\lambda}, \tilde{\mu}) = \max_{\lambda, \mu \geq 0} \min_x \mathcal{L}(x, \lambda, \mu)$$

so that

$$\min_x \mathcal{L}(x, \lambda, \mu) \leq \max_{\lambda, \mu \geq 0} \min_x \mathcal{L}(x, \lambda, \mu) \leq \max_{\lambda, \mu \geq 0} \mathcal{L}(x, \lambda, \mu)$$

It can be shown that each such saddle point satisfies the *Karush-Kuhn-Tucker* conditions:

$$\begin{aligned} \nabla_x \mathcal{L}(\tilde{x}, \tilde{\lambda}, \tilde{\mu}) &= \nabla_x f(\tilde{x}) - \langle \lambda, \nabla_x g(\tilde{x}) \rangle - \langle \mu, \nabla_x h(\tilde{x}) \rangle = 0 \\ g_i(\tilde{x}) &\geq 0 \\ h_j(\tilde{x}) &= 0 \\ \tilde{\lambda}_i &\geq 0 \\ \tilde{\mu}_j &\geq 0 \\ \langle \lambda, g(\tilde{x}) \rangle &= 0 \end{aligned}$$

For the cone ² $C = \mathbb{R}_+^m \times \mathbb{R}^p$, then these conditions can be expressed

$$\begin{aligned} \nabla_x \mathcal{L}(\tilde{x}, \tilde{\lambda}, \tilde{\mu}) &= 0 \\ \begin{bmatrix} g(\tilde{x}) \\ h(\tilde{x}) \end{bmatrix} &\in N_C(\tilde{\lambda}, \tilde{\mu}) \end{aligned}$$

Furthermore, these conditions are necessary and sufficient for \tilde{x} to be a minimiser of $f(x)$. In other words:

²From Ruszczynski [7, p.26], a set $K \subset \mathbb{R}^n$ is called a cone if for every $x \in K$ and all $\alpha > 0$ has $\alpha x \in K$. From [7, p.28], the set $K^\circ \triangleq \{y \in \mathbb{R}^n : \langle y, x \rangle \leq 0 \forall x \in K\}$ is called the polar cone of K . From [7, p.37], for a convex closed set $X \subset \mathbb{R}^n$ and a point $x \in X$, the set $N_X(x) \triangleq [cone(X - x)]^\circ$ is called the normal cone to X at x .

Algorithm 24 The sequential quadratic programming method

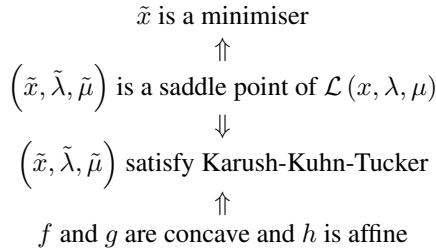
At iteration k , given the current approximation of the solution x^k and multipliers (λ^k, μ^k) solve the tangent quadratic programming problem:

$$\begin{aligned} & \text{minimise} \quad \langle \nabla_x f(x^k), d^k \rangle + \frac{1}{2} \left\langle d^{kT}, \nabla_{xx}^2 \mathcal{L}(x^k, \lambda^k, \mu^k), d^k \right\rangle \\ & \text{subject to} \quad g(x^k) + \nabla_x g(x^k) d^k \leq 0 \\ & \quad h(x^k) + \nabla_x h(x^k) d^k = 0 \end{aligned}$$

Denote the solution to this problem by d^k and the Lagrange multipliers associated with the constraints by $\hat{\lambda}^k$ and $\hat{\mu}^k$. Update the approximate solution by:

$$\begin{aligned} x^{k+1} &= x^k + \tau^k d^k \\ \lambda^{k+1} &= \hat{\lambda}^k \\ \mu^{k+1} &= \hat{\mu}^k \end{aligned}$$

and continue. Here $\tau^k \in (0, 1]$ is a step-size coefficient.



For any point x , the set of inequality constraints, \mathcal{K} , can be partitioned into a set of *active* constraints

$$\mathcal{A}(x) = \{i \mid g_i(x) = 0\}$$

and the corresponding set of *inactive* constraints, for which $i \notin \mathcal{A}(x)$. Following *Bertsekas* [20, Section 3.3], note that “if \tilde{x} is a local minimum of the inequality constrained problem, then \tilde{x} is also a local minimum of the identical problem for which the inactive constraints at \tilde{x} have been discarded. On the other hand, at a local minimum, active inequality constraints can be treated to a large extent as equalities”.

F.4 Constrained optimisation using Newton’s method

Consider the non-linear optimization problem of Section F.3. Motivated by Newton’s method, at a given point $(\bar{x}, \bar{\lambda}, \bar{\mu})$ construct an approximation to the *Karush-Kuhn-Tucker* conditions that is linearised at \bar{x} :

$$\begin{aligned} \nabla_x \mathcal{L}(\bar{x}, \bar{\lambda}, \bar{\mu}) + \nabla_{xx}^2 \mathcal{L}(\bar{x}, \bar{\lambda}, \bar{\mu})(x - \bar{x}) &= 0 \\ \begin{bmatrix} g(\bar{x}) + \nabla_x g(\bar{x})(x - \bar{x}) \\ h(\bar{x}) + \nabla_x h(\bar{x})(x - \bar{x}) \end{bmatrix} &\in N_C(\lambda, \mu) \end{aligned}$$

These are the necessary conditions of optimality for *sequential quadratic programming*, shown in Algorithm 24.

F.5 Local convergence

For simplicity, $\tau^k = 1$ and assume there are only inequality constraints ($p = 0$). Also assume that the objective function $f(x)$ has a minimum \tilde{x} , at which the gradients of active constraints, $\nabla_x g_i(\tilde{x})$, $i \in \mathcal{A}(\tilde{x}) = \{1 \leq i \leq m \mid g_i(\tilde{x}) = 0\}$ are linearly independent. In this case the Lagrange multipliers $\hat{\lambda}$ exist and are unique.

Consider the system of non-linear equations:

$$\nabla_x f(x) - \sum_{i \in \mathcal{A}(x)} \lambda_i \nabla_x g_i(x) = 0$$

$$g_i(x) = 0$$

These are the necessary conditions of a modification of the original minimisation problem in which the active inequality constraints are treated as equalities. The iterates of the sequential programming method are the iterates of Newton's method for this system. For the current iterate (x^k, λ^k) , define the matrices:

$$\begin{aligned}\mathcal{H}_k &= \nabla_{xx}^2 \mathcal{L}(x^k, \lambda^k) \\ \mathcal{B}_k &= \{\nabla_x g_i(x^k)\}\end{aligned}$$

where $i \in \mathcal{A}(x^k)$ and the columns of \mathcal{B}_k correspond to the gradient vectors. When the above equations are linearised, Newton's method takes the form:

$$\begin{aligned}[\nabla_x f(x^k) - \mathcal{B}_k \lambda^{k+1}] + \mathcal{H}_k d^k &= 0 \\ g_i(x^k) + \mathcal{B}_k^T d^k &= 0\end{aligned}$$

where λ^{k+1} and $g(x^k)$ are the vectors with coordinates λ_i and $g_i(x^k)$, for $i \in \mathcal{A}(x^k)$. This system can be simplified to:

$$\begin{aligned}\mathcal{H}_k d^k - \mathcal{B}_k \lambda^{k+1} &= -\nabla_x f(x^k) \\ -\mathcal{B}_k^T d^k &= g(x^k)\end{aligned}$$

which can be solved for the direction d^k and the multipliers λ^{k+1} :

$$\lambda^{k+1} = -(\mathcal{B}_k^T \mathcal{H}_k^{-1} \mathcal{B}_k)^{-1} [g(x^k) - \mathcal{B}_k^T \mathcal{H}_k^{-1} \nabla_x f(x^k)] \quad (\text{F.1})$$

$$d^k = -\mathcal{H}_k^{-1} [\nabla_x f(x^k) - \mathcal{B}_k \lambda^{k+1}] \quad (\text{F.2})$$

F.6 Quasi-Newton methods

Consider a modified version of the tangent quadratic programming problem

$$\begin{aligned}\text{minimise} \quad & \nabla_x f(x^k) d^k + \frac{1}{2} d^k T \mathcal{W}_k d^k \\ \text{subject to} \quad & g(x^k) + \nabla_x g(x^k) d^k \leq 0 \\ & h(x^k) + \nabla_x h(x^k) d^k = 0\end{aligned}$$

where the Hessian of the Lagrangian $\nabla_{xx}^2 \mathcal{L}(x^k, \lambda^k, \mu^k)$ is replaced by the positive definite matrix \mathcal{W}_k , guaranteeing that this is a convex quadratic programming problem. The following subsections describe methods for constructing the matrix \mathcal{W}_k .

F.6.1 Updating \mathcal{W}_k with the *Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula*

Typically, $\mathcal{W}_0 = I$. For convenience, set $\delta_k = \tau^k d^k$. The update to \mathcal{W}_k should depend on \mathcal{W}_k and the difference in gradients of the Lagrangian

$$\gamma_k = \nabla_x \mathcal{L}(x^k + \delta_k, \lambda^k, \mu^k) - \nabla_x \mathcal{L}(x^k, \lambda^k, \mu^k)$$

When there are no constraints it is possible to choose the step-length, τ^k , so that the scalar product $\delta_k^T \gamma_k$ is positive. On the other hand, when there are constraints, it can happen that $\delta_k^T \gamma_k$ is negative for all non-zero values of τ^k . In this case the usual methods for updating \mathcal{W}_k would fail to make the update positive definite. Powell [58, p.148] proposes updating \mathcal{W}_k as follows. First replace γ_k with

$$\eta_k = \theta_k \gamma_k + (1 - \theta_k) \mathcal{W}_k \delta_k, \quad 0 \leq \theta_k \leq 1$$

where

$$\theta_k = \begin{cases} 1 & \delta_k^T \gamma_k \geq 0.2 \delta_k^T \mathcal{W}_k \delta_k \\ \frac{0.8 \delta_k^T \mathcal{W}_k \delta_k}{\delta_k^T \mathcal{W}_k \delta_k - \delta_k^T \gamma_k} & \delta_k^T \gamma_k < 0.2 \delta_k^T \mathcal{W}_k \delta_k \end{cases} \quad (\text{F.3})$$

The factor of 0.2 was chosen empirically. Now update \mathcal{W}_k with the *Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula*

$$W_{k+1} = \mathcal{W}_k - \frac{\mathcal{W}_k \delta_k (\mathcal{W}_k \delta_k)^T}{\delta_k^T \mathcal{W}_k \delta_k} + \frac{\eta_k \eta_k^T}{\delta_k^T \eta_k}$$

and update \mathcal{W}_k^{-1} with the *Sherman-Morrison formula*

$$W_{k+1}^{-1} = \mathcal{W}_k^{-1} + \left[1 + \frac{\eta_k^T \mathcal{W}_k^{-1} \eta_k}{\delta_k^T \eta_k} \right] \left(\frac{\delta_k \delta_k^T}{\delta_k^T \eta_k} \right) - \left[\frac{\mathcal{W}_k^{-1} \eta_k \delta_k^T + \delta_k \eta_k^T \mathcal{W}_k^{-1}}{\delta_k^T \eta_k} \right]$$

Note that η and δ are column vectors and $\delta \eta^T$ is a *dyadic* or *Kronecker* product also written $\delta \otimes \eta$. Nocedal and Wright [47, Procedure 18.2] refer to Equation F.3 as *damped BFGS updating*.

Algorithm 25 Cholesky factorisation of an $n \times n$ positive-definite symmetric matrix, W

$$L_1 = \sqrt{w_{1,1}}$$

$$W_1 = L_1 L_1^T$$

for $i = 2, \dots, n$ **do**

$$W_i = \begin{bmatrix} W_{i-1} & \beta_i \\ \beta_i^T & w_{i,i} \end{bmatrix} \text{ where } \beta_i = \begin{bmatrix} w_{1,i} \\ \vdots \\ w_{i-1,i} \end{bmatrix}$$

$$W_i = L_i L_i^T \text{ where } L_i = \begin{bmatrix} L_{i-1} & 0 \\ l_i^T & \lambda_{ii} \end{bmatrix}, l_i = L_{i-1}^{-1} \beta_i \text{ and } \lambda_{i,i} = \sqrt{w_{i,i} - l_i^T l_i}$$

end for

$$W = L_n L_n^T$$

F.6.2 A modified Cholesky factorisation of the Hessian

Bertsekas [20, Appendix D] describes implementation of Newton's method by Cholesky factorisation of an approximation to the Hessian, $\nabla_{xx}^2 \mathcal{L}(x^k) + \Delta_k$, that is positive definite.

Algorithm 25 shows the Cholesky factorisation of a positive definite symmetric matrix, W , as LL^T where L is lower triangular. Firstly, define W_i to be the i -th leading principal submatrix of W :

$$W_i = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,i} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i,1} & w_{i,2} & \cdots & w_{i,i} \end{bmatrix}$$

This matrix is positive definite since, for any $y \in \mathbb{R}^i$, $y \neq 0$

$$y^T W_i y = \begin{bmatrix} y^T & 0 \end{bmatrix} W \begin{bmatrix} y \\ 0 \end{bmatrix} > 0$$

The scalar λ_{ii} can be seen to be well-defined by setting $b = W_{i-1}^T \beta_i$ and recalling that W_i is positive definite:

$$\begin{aligned} 0 < \begin{bmatrix} b^T & -1 \end{bmatrix} W_i \begin{bmatrix} b \\ -1 \end{bmatrix} &= b^T W_{i-1} b - 2b^T \beta_i + w_{ii} \\ &= w_{ii} - b^T \beta_i \\ &= w_{ii} - \beta_i^T W_{i-1}^{-1} \beta_i \\ &= w_{ii} - (L_{i-1}^{-1} \beta_i)^T (L_{i-1}^{-1} \beta_i) \\ &= w_{ii} - l_i^T l_i \end{aligned}$$

Note that matrix inversion of an arbitrary matrix is an $\mathcal{O}(n^3)$ operation whereas matrix inversion of a triangular matrix via backward or forward substitution is an $\mathcal{O}(n^2)$ operation³.

We wish to add a diagonal correction, Δ_k , to the Hessian matrix such that the resulting matrix is positive definite whilst simultaneously factoring $\nabla_{xx}^2 \mathcal{L}(x^k) + \Delta_k$. Firstly, fix two positive scalars μ_1 and μ_2 , where $\mu_1 < \mu_2$. The first column of the factor L is given by

$$\begin{aligned} l_{11} &= \begin{cases} \sqrt{w_{11}} & \mu_1 < w_{11} \\ \sqrt{\mu_2} & \text{otherwise} \end{cases} \\ l_{i1} &= \frac{w_{i1}}{l_{11}} \quad i = 2, \dots, n \end{aligned}$$

Given columns $1, 2, \dots, j-1$ of L the elements of the j -th column are

$$l_{jj} = \begin{cases} \sqrt{w_{jj} - \sum_{m=1}^{j-1} l_{jm}^2} & \mu_1 < w_{jj} - \sum_{m=1}^{j-1} l_{jm}^2 \\ \sqrt{\mu_2} & \text{otherwise} \end{cases}$$

³The LDL^T factorisation of a symmetric matrix avoids taking the square-root. See [26, Algorithm 4.1.2].

$$l_{ij} = \frac{w_{ij} - \sum_{m=1}^{j-1} l_{jm} l_{im}}{l_{jj}} \quad i = j+1, \dots, n$$

This scheme can be used in a modified Newton's method, where at the k -th iteration, we add a diagonal correction, Δ^k , to the Hessian, $\nabla_{xx}^2 \mathcal{L}(x^k)$, and simultaneously obtain a Cholesky factorisation of $\nabla_{xx}^2 \mathcal{L}(x^k) + \Delta^k$. The direction vector, d^k , is found by solving the triangular systems

$$\begin{aligned} L_k y &= -\nabla_x f(x^k) \\ L_k^T d^k &= y \end{aligned}$$

The next point is found by

$$x^{k+1} = x^k + \tau^k d^k$$

where τ^k is a step size. The scalars μ_1 and μ_2 are selected as follows: at each iteration we find the maximum absolute value of the Hessian diagonal elements

$$w^k = \max \text{ element } \{\nabla_{xx}^2 \mathcal{L}(x^k)\}$$

and set $\mu_1 = r_1 w^k$ and $\mu_2 = r_2 w^k$. The scalar r_1 is set at some “small” (or zero) value. The scalar r_2 is modified at each iteration: if $\tau^k < 0.2$ then $r_2 = 5r_2$; otherwise if $\tau^k > 0.9$ then $r_2 = \frac{r_2}{5}$.

F.6.3 Wright's modification for degenerate constraints

For the filter design problem, the gradients of upper and lower constraints on the radius and angle of the poles and zeros are not linearly independent and the resulting Jacobian matrix is degenerate. The *singular value decomposition* [26, Sections 2.5.3 and 5.5] finds the minimum 2-norm solution of a degenerate matrix equation as follows: if B is a real $m \times n$ matrix, then there exist orthogonal matrices $U = [u_1, \dots, u_m] \in \mathbb{R}^{m \times m}$ and $V = [v_1, \dots, v_n] \in \mathbb{R}^{n \times n}$ such that

$$U^T B V = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m \times n}$$

where $[u_1, \dots, u_m]$ is a column partitioning of U , $p = \min(m, n)$ and $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$. From *Golub and Van Loan* [26, Section 5.5.4], the *pseudo-inverse* of B is defined to be $B^* = V \Sigma U^T$ where

$$\Sigma = \text{diag}\left(\frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_r}, 0, \dots, 0\right) \in \mathbb{R}^{n \times m}$$

and $\text{rank}(B) = r$. It is the unique minimal Frobenius ⁴ norm solution to the problem

$$\min_{X \in \mathbb{R}^{n \times m}} \|BX - I_m\|_F$$

If $\text{rank}(B) = n$, then $B^* = (B^T B)^{-1} B^T$, while if $m = n = \text{rank}(B)$, then $B^* = B^{-1}$. Typically, B^* is defined to be the unique matrix $X \in \mathbb{R}^{n \times m}$ that satisfies the four *Moore-Penrose conditions*:

$$\begin{aligned} B X B &= B \\ X B X &= X \\ (B X)^T &= B X \\ (X B)^T &= X B \end{aligned}$$

⁴*Golub and Van Loan* [26, Section 2.3.1, Section 2.2.1] define the Frobenius norm (also called the Euclidean norm) of a matrix A as

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}$$

and the set of vector norms

$$\begin{aligned} \|x\|_p &= (|x_1|^p + \dots + |x_n|^p)^{\frac{1}{p}} \quad p \geq 1 \\ \|x\|_\infty &= \max_{1 \leq i \leq n} |x_i| \end{aligned}$$

$\|x\|$ is assumed to mean $\|x\|_2$.

On the other hand, the Octave manual defines the Frobenius norm of a matrix as

$$\text{norm}(A, "fro") = \text{sqrt}\left(\text{sum}\left(\text{diag}\left(A^T A\right)\right)\right)$$

These conditions amount to the requirement that BB^* and B^*B be orthogonal projections onto range (B) and range (B^T) respectively⁵.

Wright [85] points out that the existence of linearly dependent constraint gradients can interfere with the super-linear convergence of the sequential quadratic programming problem (SQP) and provides a stabilised algorithm for which super-linear convergence is still possible. Given a set of active constraints $\mathcal{A}(x) \subset \mathcal{K}$ at a point x and the complement $\mathcal{N}(x) = \mathcal{K} \setminus \mathcal{A}(x)$, Wright defines $g_{\mathcal{A}}(x) = \{g_i(x) \mid i \in \mathcal{A}(x)\}$, $g_{\mathcal{N}}(x) = \{g_i(x) \mid i \in \mathcal{N}(x)\}$, $\lambda_{\mathcal{A}}(x) = \{\lambda_i(x) \mid i \in \mathcal{A}(x)\}$, $\lambda_{\mathcal{N}}(x) = \{\lambda_i(x) \mid i \in \mathcal{N}(x)\}$. Wright considers the usual SQP problem and makes the following assumptions

1. a primal-dual solution point $(\tilde{x}, \tilde{\lambda})$ exists
2. for at least one of the active constraints, $\tilde{\lambda}_i > 0$ where $i \in \mathcal{A}(\tilde{x})$
3. $\tilde{\lambda}_i = 0 \forall i \in \mathcal{N}(\tilde{x})$
4. there exists a $\sigma > 0$ such that $w^T \nabla_{xx} \mathcal{L}(\tilde{x}, \tilde{\lambda}) w \geq \sigma \|w\|^2$ for all $\tilde{\lambda}$ such that $(\tilde{x}, \tilde{\lambda})$ satisfies the *Karush-Kuhn-Tucker* conditions and all $w \in \text{null}(\nabla g_{\mathcal{A}}(\tilde{x}))$
5. $\nabla g_{\mathcal{A}}(\tilde{x}) d < 0$ for some $d \in \mathbb{R}^n$

The latter assumption (known as the *Mangasarian-Fromovitz* constraint qualification) is weaker than the assumption that the matrix of constraint gradients has full column rank. *Wright* [85] considers the linearised SQP method described in Section F.4. He points out that the Lagrange multipliers $\lambda_{\mathcal{A}}$ may not be uniquely determined if the Jacobian $\nabla_x g_{\mathcal{A}}(x)$ is rank deficient. Consequently, the Hessian $\nabla_{xx} \mathcal{L}(x, \lambda_{\mathcal{A}})$ may not be uniquely defined at the next SQP iteration. He proposes a stabilised version of the SQP algorithm

$$\min_x \max_{\lambda_{\mathcal{A}} \geq 0} \langle \nabla f(\bar{x}), x - \bar{x} \rangle - \langle \lambda_{\mathcal{A}}, g(\bar{x}) + \nabla_x^T g(\bar{x})(x - \bar{x}) \rangle + \frac{1}{2} \langle x - \bar{x}, [\nabla_{xx}^2 \mathcal{L}(\bar{x}, \bar{\lambda}_{\mathcal{A}})](x - \bar{x}) \rangle - \frac{1}{2} \nu \|\lambda_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}\|^2$$

where ν is defined as

$$\nu(x, \lambda_{\mathcal{A}}) = \|(\nabla_x \mathcal{L}(x, \lambda_{\mathcal{A}}), g_{\mathcal{A}}(x), \langle \lambda_{\mathcal{A}}, g_{\mathcal{A}}(x) \rangle)\|$$

The optimality conditions for a candidate solution $(\tilde{x}, \tilde{\lambda}_{\mathcal{A}})$ are likewise similar

$$\begin{aligned} \nabla_x f(\bar{x}) - \nabla g_{\mathcal{A}}(\bar{x}) \tilde{\lambda}_{\mathcal{A}} + \nabla_{xx} \mathcal{L}(\bar{x}, \bar{\lambda}_{\mathcal{A}})(\tilde{x} - \bar{x}) &= 0 \\ g_{\mathcal{A}}(\bar{x}) + \nabla_x^T g_{\mathcal{A}}(\bar{x})(\tilde{x} - \bar{x}) - \nu(\tilde{\lambda}_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}) &\geq 0 \\ \tilde{\lambda}_{\mathcal{A}} &\geq 0 \\ \langle \tilde{\lambda}_{\mathcal{A}}, g_{\mathcal{A}}(\bar{x}) + \nabla g_{\mathcal{A}}^T(\bar{x})(\tilde{x} - \bar{x}) - \nu(\tilde{\lambda}_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}) \rangle &= 0 \end{aligned}$$

Wright shows that for $(\bar{x}, \bar{\lambda}_{\mathcal{A}})$ sufficiently close to a primal-dual solution $(x, \lambda_{\mathcal{A}})$ there is a unique solution $(\tilde{x}, \tilde{\lambda}_{\mathcal{A}})$ for which $\|(\tilde{x} - \bar{x}, \tilde{\lambda}_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}})\| = \mathcal{O}(\nu)$. This solution satisfies the following linear system

$$\begin{bmatrix} \nabla_{xx} \mathcal{L}(\bar{x}, \bar{\lambda}_{\mathcal{A}}) & -\nabla_x g_{\mathcal{A}}(\bar{x}) \\ -\nabla_x^T g_{\mathcal{A}}(\bar{x}) & \nu I \end{bmatrix} \begin{bmatrix} x - \bar{x} \\ \lambda_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}} \end{bmatrix} = \begin{bmatrix} -(\nabla_x f(\bar{x}) - \nabla_x g_{\mathcal{A}}(\bar{x}) \bar{\lambda}_{\mathcal{A}}) \\ g_{\mathcal{A}}(\bar{x}) \\ \lambda_{\mathcal{N}} = 0 \end{bmatrix}$$

⁵*Golub and Van Loan* [26, Section 2.1.2] define the *range* and *null space* or *kernel* of B as

$$\begin{aligned} \text{range}(B) &= \{y \in \mathbb{R}^n \mid y = Bx, x \in \mathbb{R}^n\} \\ \text{null}(A) &= \{x \in \mathbb{R}^n \mid Bx = 0\} \end{aligned}$$

If $B = [b_1, \dots, b_n]$ is a column vector partitioning then the *span* of B is the set of all linear combinations of those column vectors

$$\text{span}\{b_1, \dots, b_n\} = \left\{ \sum_{j=1}^n \beta_j b_j \mid \beta_j \in \mathbb{R} \right\}$$

Consequently, given the SVD of a matrix B with $\text{rank}(B) = r$

$$\begin{aligned} \text{null}(B) &= \text{span}\{v_{r+1}, \dots, v_n\} \\ \text{range}(B) &= \text{span}\{u_1, \dots, u_r\} \end{aligned}$$

F.6.4 Bertsekas' modification to the Hessian

Bertsekas [20, p. 461] describes a modification to the Hessian that can ensure that the Hessian is positive definite and invertible. From the usual Newton system

$$\begin{aligned}\mathcal{H}_k d^k - \mathcal{B}_k \lambda^{k+1} &= -\nabla_x f(x^k) \\ -\mathcal{B}_k^T d^k &= g(x^k)\end{aligned}$$

define a new matrix:

$$\bar{\mathcal{H}}_k = \mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^T$$

where the scalar, c^k , is chosen so that $\bar{\mathcal{H}}_k$ is positive definite. Bertsekas points out that

$$d^{k^T} \mathcal{B}_k \mathcal{B}_k^T d^k = \|g(x^k)\|^2$$

is a constant for *equality* constraints. Consequently, the modified optimisation problem

$$\begin{aligned}\text{minimise} \quad & \nabla_x f(x^k) d^k + \frac{1}{2} d^{k^T} (\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^T) d^k \\ \text{subject to} \quad & -\mathcal{B}_k^T d^k = g(x^k)\end{aligned}$$

has a larger minimum value but the location of that minimum is unchanged.

Given the scalar, c^k , write:

$$c^k \mathcal{B}_k \mathcal{B}_k^T d^k = -c^k \mathcal{B}_k g(x^k)$$

Adding:

$$(\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^T) d^k - \mathcal{B}_k \lambda^{k+1} = -[\nabla_x f(x^k) + c^k \mathcal{B}_k g(x^k)]$$

As in Section F.5:

$$\begin{aligned}\lambda^{k+1} &= -\left(\mathcal{B}_k^T (\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^T)^{-1} \mathcal{B}_k\right)^{-1} \left[g(x^k) - \mathcal{B}_k^T (\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^T)^{-1} (\nabla_x f(x^k) + c^k \mathcal{B}_k g(x^k))\right] \\ d^k &= -(\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^T)^{-1} [(\nabla_x f(x^k) + c^k \mathcal{B}_k g(x^k)) - \mathcal{B}_k \lambda^{k+1}]\end{aligned}$$

F.7 Penalty and barrier methods

From Bertsekas [20, Section 4.2], “The basic idea in penalty methods is to eliminate some or all of the constraints and add to the cost function a penalty term that prescribes a high cost to infeasible points.” Barrier functions are penalty functions that approach infinity as the constrained variable approaches the constraint. Clearly a barrier function requires that a feasible initial point is known.

F.7.1 Penalty functions

In the unconstrained case τ^k is found by a line search that minimises the objective function $f(x)$ along the ray $x^k + \tau d^k$ over $\tau \geq 0$. Ruszczynski [7, p.329] suggests that when there are constraints on x we use the *penalty function*

$$\Psi_\varrho(x) \triangleq f(x) + \varrho \left[\sum_{i=1}^p |h_i(x)| - \sum_{i=1}^m \min(0, g_i(x)) \right]$$

and states that a local minimum of the objective function is also a local minimum of $\Psi_\varrho(x)$, provided that the penalty coefficient ϱ is larger than the “max” norm of the Lagrange multipliers $(\hat{\lambda}, \hat{\mu})$ at the solution. Similarly, Powell [58, p.151] suggests using the penalty function

$$\Psi(x) \triangleq f(x) - \left[\sum_{i=1}^m \Omega_i \min(0, g_i(x)) + \sum_{i=1}^p \Upsilon_i |h_i(x)| \right]$$

and requiring that τ^k satisfies

$$\Psi(x^k + \tau^k d^k, \Omega, \Upsilon) \leq \Psi(x^k, \Omega, \Upsilon)$$

Υ_i and Ω_k are defined below. This condition can be fulfilled if the function

$$\Phi(\tau^k) = \Psi(x^k + \tau^k d^k, \Omega, \Upsilon)$$

decreases initially when τ^k is made positive. *Powell* [58] asserts that this happens if \mathcal{W}_k is positive definite and if

$$\begin{aligned}\Omega_i &\geq |\lambda_i| \quad i = 1, \dots, m \\ \Upsilon_i &\geq |\mu_i| \quad i = 1, \dots, p\end{aligned}$$

Powell [58, p.151] suggests $\Omega_i = |\lambda_i|$ for the first iteration and subsequently

$$\Omega_i = \max \left(|\lambda_i|, \frac{1}{2} [\bar{\Omega}_i + |\lambda_i|] \right)$$

where $\bar{\Omega}_i$ is the value of Ω_i used on the previous iteration. Similarly for Υ_i . The line-search procedure given by *Powell* [58, p.152] is obscure but it appears to be a variation of the “two-slope” test that is also described by *Ruszczynski* [7, p. 216] and *Bertsekas* [20, p. 28]. The test assumes that $\Phi'(0) < 0$. *Powell* [58, p.152] suggests that the line-search terminate when

$$\Phi(\hat{\tau}_k) \leq \Phi(0) + 0.1\tau^k \Phi'(0)$$

Bertsekas [20, p. 405], describes use of a quadratic penalty function with inequality constraints. The inequality constraints are converted to equality constraints by the addition of slack variables, z_i , giving the SQP problem

$$\begin{aligned}&\text{minimise} \quad f(x) \\ &\text{subject to} \quad h_i(x) = 0 \\ &\quad g_i(x) - z_i^2 = 0\end{aligned}$$

The Quadratic Penalty Function method (*Bertsekas* [20, Section 4.2.1] gives a series of unconstrained minimisations of the form

$$\min_{x,z} \bar{\mathcal{L}}_{c^k}(x, z, \mu^k, \lambda^k) = f(x) + \left\{ \frac{c^k}{2} \|h(x)\|^2 - (\mu^k)^T h(x) \right\} + \sum_{i=1}^m \left\{ \frac{c^k}{2} |g_i(x) - z_i^2|^2 - \lambda_i^k (g_i(x) - z_i^2) \right\}$$

where $c^k < c^{k+1}$ and $c^k \rightarrow \infty$. Minimising the sum with respect to z is equivalent to

$$\mathcal{L}_{c^k}(x, v, \mu^k, \lambda^k) = f(x) + \left\{ \frac{c^k}{2} \|h(x)\|^2 - (\mu^k)^T h(x) \right\} + \min_{v_i \geq 0} \sum_{i=1}^m \left\{ \frac{c^k}{2} |g_i(x) - v_i|^2 - \lambda_i^k (g_i(x) - v_i) \right\}$$

If the constrained minimum of the second term is at $v_i^* = \max\{0, \bar{v}_i\}$ (where $\bar{v}_i \geq 0$ is the unconstrained minimum at which $\lambda_i + c^k (g_i(x) - \bar{v}_i)$ is zero) then

$$v_i^* = \max \left\{ 0, g_i(x) + \frac{\lambda_i^k}{c^k} \right\}$$

and

$$\begin{aligned}g_i(x) - v_i^* &= \begin{cases} -\frac{\lambda_i^k}{c^k} & g_i(x) > -\frac{\lambda_i^k}{c^k} \\ g_i(x) & g_i(x) \leq -\frac{\lambda_i^k}{c^k} \end{cases} \\ &= \min \left(g_i(x), -\frac{\lambda_i^k}{c^k} \right)\end{aligned}$$

The Lagrangian summation term becomes

$$\sum_{i=1}^m \lambda_i^k \min \left(g_i(x), -\frac{\lambda_i^k}{c^k} \right) + \frac{c^k}{2} \left| \min \left(g_i(x), -\frac{\lambda_i^k}{c^k} \right) \right|^2 = \begin{cases} \frac{1}{2c^k} \sum_{i=1}^m \left[(\lambda_i^k + c^k g_i(x))^2 - (\lambda_i^k)^2 \right] & g_i(x) \leq -\frac{\lambda_i^k}{c^k} \\ -\frac{1}{2c^k} \sum_{i=1}^m (\lambda_i^k)^2 & g_i(x) > -\frac{\lambda_i^k}{c^k} \end{cases}$$

Substituting into the Lagrangian

$$\mathcal{L}_{c^k}(x, \mu^k, \lambda^k) = f(x) + \left\{ \frac{c^k}{2} \|h(x)\|^2 - (\mu^k)^T h(x) \right\} + \frac{1}{2c^k} \sum_{i=1}^m \left\{ \min [0, (\lambda_i^k + c^k g_i(x))]^2 - \lambda_i^k \right\}$$

The penalty term for the inequality constraints is continuously differentiable in x if $g_i(x)$ is continuously differentiable. The optimisation problem is now unconstrained minimisation of \mathcal{L}_{c^k} with updating of the Lagrange multipliers by

$$\lambda^{k+1} = \min(0, \lambda^k + c^k g(x))$$

Unfortunately, difficulties arise because the Hessian matrix of \mathcal{L}_{c^k} is discontinuous at the x for which $g_i(x) = -\frac{\lambda_i^k}{c_i^k}$.

An alternative augmented Lagrangian function that does have a continuous Hessian adds an exponential penalty function

$$\psi(t) = e^{-t} - 1$$

The constrained optimisation problem becomes unconstrained optimisation of the following Lagrangian (for inequality constraints only)

$$\mathcal{L}_{c^k}(x, \lambda^k) = f(x) + \sum_{i=1}^m \frac{\lambda_i^k}{c_i^k} \psi(c_i^k g_i(x))$$

with gradient

$$\nabla \mathcal{L}_{c^k}(x, \lambda^k) = \nabla f(x) - \sum_{i=1}^m \lambda_i^k g_i(x) \exp(-c_i^k g_i(x))$$

The $\{c_j^k\}$ are a positive penalty parameter sequence for each j . The λ^k are updated by

$$\lambda^{k+1} = \lambda^k \exp(c^k g(x))$$

Two implementation details

- to avoid overflow $\psi(t)$ should be defined as the exponential $e^{-t} - 1$ only for the interval in which the exponential is within the floating point range
- the penalty parameter for each constraint should depend on the corresponding multiplier by

$$c_j^k = \frac{w^k}{\lambda_j^k}$$

where $\{w^k\}$ is a positive sequence with $w^k \leq w^{k+1}$

F.7.2 Barrier functions

Bertsekas [20, Section 4.1], describes adding a so-called *barrier* function, $B(x)$, to the cost function. This function is continuous and goes to ∞ as any one of the constraints approaches 0 from positive values. Clearly, the barrier function is only defined at iterates, x^k , that satisfy the constraints

$$S = \{x \in X \mid g_j(x) \geq 0, j = 1, \dots, m\}$$

The two most common barrier functions are

$$\begin{aligned} B(x) &= - \sum_{i=1}^m \ln\{g_i(x)\} \\ B(x) &= \sum_{i=0}^m \frac{1}{g_i(x)} \end{aligned}$$

The barrier method consists of finding

$$x^k \in \arg \min_{x \in S} \{f(x) + \epsilon^k B(x), k = 0, 1, \dots\}$$

where the sequence $\{\epsilon^k\}$ is defined by

$$0 < \epsilon^{k+1} < \epsilon^k \quad k = 0, 1, \dots \quad \epsilon^k \rightarrow 0$$

F.8 Finding the step size

The coefficient solution space for the IIR filter design problem is highly non-linear and has many local minima. At each iteration of the SQP method the maximum step-size to the next estimate must maintain the positive semi-definiteness of the Hessian (or its approximation) but at the same time be large enough that solution approaches a minimum efficiently.

E.8.1 Line search with the Golden-Section

The *Golden Section* (see *Ruszczynski* [7, p.213] and *Bertsekas* [20, Appendix C.3]) is a simple means of generating the sequence of intervals required for line-search on a *convex* function. First, given two endpoint points $\alpha < \delta$, construct a sequence of four points

$$\alpha < \beta < \gamma < \delta$$

with

$$\begin{aligned}\beta &= \alpha + (1 - q)(\delta - \alpha) \\ \gamma &= \delta - (1 - q)(\delta - \alpha)\end{aligned}$$

where

$$\begin{aligned}q &= \frac{-1 + \sqrt{5}}{2} \\ 1 - q &= q^2\end{aligned}$$

Note that

$$\begin{aligned}\frac{\beta - \alpha}{\delta - \alpha} &= 1 - q \\ \frac{\gamma - \alpha}{\delta - \alpha} &= q \\ \frac{\gamma - \beta}{\delta - \beta} &= 1 - q \\ &\text{etc.}\end{aligned}$$

At each iteration select a new endpoint based on the function values at each point. The Octave function *goldensection.m* shows an implementation.

A quicker search method makes use of the fact that the gradient $\Phi'(0)$ is known and is negative [7, p.215]. First, search for a point $\beta > 0$ such that $\Phi(\beta) \geq \Phi(0)$, then quadratic interpolation gives

$$\tau^k = \frac{-\Phi'(0)\beta^2}{2[\Phi(\beta) - \Phi(0) - \Phi'(0)\beta]}$$

Since $\Phi(\beta) \geq \Phi(0)$, we have $0 < \tau^k \leq \frac{\beta}{2}$. If we still have $\Phi(\tau) > \Phi(0)$, then we replace β with τ^k and repeat the above interpolation. If $\Phi(\tau) < \Phi(0)$, then we have three points, with the best one in the middle, and we can apply a Golden Section search.

Alternatively, interpolate $\Phi(x)$. Given $\alpha_k < \tau^k < \beta_k$ such that $\Phi(\alpha_k) > \Phi(\tau^k) < \Phi(\beta_k)$ then a second-order estimate of the minimum is

$$\gamma_k = \frac{\Phi(\alpha_k)[\beta_k^2 - (\tau^k)^2] + \Phi(\tau^k)[\alpha_k^2 - \beta_k^2] + \Phi(\beta_k)[(\tau^k)^2 - \alpha_k^2]}{2(\Phi(\alpha_k)[\beta_k - \tau^k] + \Phi(\tau^k)[\alpha_k - \beta_k] + \Phi(\beta_k)[\tau^k - \alpha_k])}$$

The initial three points can be found by scanning the values $\Phi(0), \Phi(\tau_0), \dots$ or $\Phi(0), \Phi(-\tau_0), \dots$ depending on whether $\Phi(\tau_0) < \Phi(0)$.

F.8.2 Inexact step-size selection

The previous sub-section describes a brute-force search for the best step-size. *Bertsekas* [20, p. 28] describes selection rules for finding a step-size that is neither too large nor too small but “good enough”. This approach usually requires far fewer function evaluations than an “exact” line search for a problem with a convex, differentiable objective function.

The inexact step-size rules are motivated by the solution to the locally convergent system of equations shown in Equation F.2. Ignoring the constraints:

$$\mathcal{H}_k d^k = -\nabla_x f(x^k)$$

Since we approximate \mathcal{H}_k by a positive-definite matrix, \mathcal{W}_k :

$$\langle \nabla_x f(x^k), d^k \rangle = -\langle \nabla_x f(x^k), \mathcal{W}_k^{-1} \nabla_x f(x^k) \rangle < 0$$

and d^k is a descent direction.

Bertsekas [20, p. 29], recommends the use of the *Armijo* rule. Given an initial step size $\sigma > 0$ and $\beta \in (0, 1)$, choose τ^k to be the largest in $\{\sigma, \sigma\beta, \sigma\beta^2, \dots\}$ so that

$$f(x^k + \tau^k d^k) \leq f(x^k) + c_1 \tau^k \langle \nabla_x f(x^k), d^k \rangle \quad (\text{W1})$$

The *Armijo* rule reduces the step size geometrically. An implementation of line search using the *Armijo* rule is shown in Algorithm 26.

Algorithm 26 Line search using the Armijo rule

```

 $\beta \in [0.1, 0.5]$ 
 $\sigma \in [10^{-5}, 10^{-1}]$ 
while  $f(x^k) - f(x^k + \beta^m d^k) \geq -c_1 \sigma \beta^m \langle \nabla_x f(x^k), d^k \rangle$  do
     $m = m + 1$ 
end while
 $\tau = \sigma \beta^m$ 

```

Kim et al. [63] claim improved efficiency with the following modification to the *Armijo* rule:

$$f(x^k + \tau^k d^k) \leq f(x^k) + c_1 \tau^k \left[\langle \nabla_x f(x^k), d^k \rangle + \frac{1}{2} \tau^k \langle d^k, \mathcal{W}_k d^k \rangle \right] \quad (\text{F.4})$$

To rule out unacceptably short steps, the *Wolfe* step-size rules add the following to the *Armijo* rule (or W1):

$$\langle \nabla_x f(x^k + \tau^k d^k), d^k \rangle \geq c_2 \langle \nabla_x f(x^k), d^k \rangle \quad (\text{W2})$$

where $0 < c_1 < c_2 < 1$. Common choices for the Newton or quasi-Newton methods are $c_1 = 10^{-4}$ and $c_2 = 0.9$. The second *Wolfe* condition may be strengthened to

$$|\langle \nabla_x f(x^k + \tau^k d^k), d^k \rangle| \leq |c_2 \langle \nabla_x f(x^k), d^k \rangle| \quad (\text{F.5})$$

This stronger *Wolfe* condition does not allow the gradient to be too positive, excluding points that are far from the minimum. *Christianson* [12] gives Algorithm 27 for line search using the *Wolfe* conditions.

The *Goldstein* conditions for step-size selection are similar to the *Wolfe* conditions:

$$f(x^k + \tau^k d^k) < f(x^k) + c_1 \tau^k \langle \nabla_x f(x^k), d^k \rangle \quad (\text{G1})$$

$$f(x^k + \tau^k d^k) > f(x^k) + c_2 \tau^k \langle \nabla_x f(x^k), d^k \rangle \quad (\text{G2})$$

where $0 < c_1 < c_2 < 1$. Usually $c_1 < 0.5$ and $c_2 = 1 - c_1$. *Christianson* gives Algorithm 28 for line search using the *Goldstein* conditions. The first iteration in Algorithm 28 must terminate because $f(x)$ is differentiable at x^k . The second iteration must terminate because $f(x)$ is bounded below. Note that the two while loops can be placed in either order, and that at most one of them will ever be performed.

Algorithm 27 Line search using Wolfe conditions

```
R = 1, r = 0.5, τ = 1, a = 0, b = ∞
while a ≠ b do
    if W1(τ) then
        a = τ
    else
        b = τ
    end if
    if W1(τ) ∧ W2(τ) then
        b = τ
    else
        if b = ∞ then
            τ = max(τ, Ra)
        else
            τ = max((1 - r)a + rb, min(τ, ra + (1 - r)b))
        end if
    end if
end while
```

Algorithm 28 Line search using the Goldstein conditions

```
τ = 1, R = 2
while ¬G1(τ) do
    τ =  $\frac{\tau}{R}$ 
end while
while ¬G2(τ) do
    τ = τR
end while
```

F.8.3 Lanczos step-size selection

Toh [50] describes estimation of the step-size by the *Lanczos* iteration for finding the eigenvalues of a matrix. The updated approximation to the Hessian is expressed as:

$$\mathcal{W}_{k+1} = \mathcal{W}_k + \rho_k \Delta \mathcal{W}_k$$

Since \mathcal{W}_k is symmetric and positive-definite, it has a Cholesky factorisation $\mathcal{W}_k = L_k^T L_k$. Let

$$\mathcal{B}_k = -L_k^{-T} \Delta \mathcal{W}_k L_k^{-1}$$

then the condition that $\mathcal{W}_{k+1} \succeq 0$ is equivalent to $I - \rho_k \mathcal{B}_k \succeq 0$. In other words

$$\max \rho_k = \begin{cases} \frac{1}{\lambda_1} & \lambda_1 > 0 \\ \infty & \text{otherwise} \end{cases}$$

where λ_1 is the maximum eigenvalue of \mathcal{B}_k and ρ_k provides an upper bound on the corresponding step-size $\tau^k d^k$. Toh suggests that the required computation can be reduced by finding λ_1 with the *Lanczos* iteration method. Given a symmetric matrix, $A \in \mathbb{R}^{N \times N}$, the *Lanczos* method generates a sequence of tridiagonal matrices $T_k \in \mathbb{R}^{k \times k}$ having the property that the extremal eigenvalues of T_k are progressively better estimates of the extremal eigenvalues of A . Golub and van Loan describe efficient methods for finding the eigenvalues of a symmetric tridiagonal matrix [26, Section 8.5] and have an extensive discussion of *Lanczos* methods [26, Chapter 9].

F.9 Initial solution with the Goldfarb-Idnani algorithm

Goldfarb and Idnani [19] observe that the origin in the space of dual variables (ie: Lagrange multipliers) is always a feasible solution of the dual problem. In Algorithm 29 I reproduce their dual algorithm for solving the tangent quadratic problem with linear constraints.

Algorithm 29 The Goldfarb-Idnani dual algorithm [19]

0. Find the unconstrained minimum of $f(x)$, then:

Set $x \leftarrow H^{-1}a$, $f \leftarrow \frac{1}{2}a^T x$, $E \leftarrow H^{-1}$, $\mathcal{A} \leftarrow \emptyset$, $q \leftarrow 0$

1. Choose the most violated constraint, if any:

Compute $g_j(x)$, for all $j \in \mathcal{K} \setminus \mathcal{A}$.

If $\mathcal{V} = \{j \in \mathcal{K} \setminus \mathcal{A} \mid g_j(x) < 0\} = \emptyset$ then **stop**, the current solution x is both feasible and optimal.

Otherwise, choose $p \in \mathcal{V}$ and set $n^+ \leftarrow n_p$ and $\lambda^+ \leftarrow \begin{pmatrix} \lambda \\ 0 \end{pmatrix}$.

If $q = 0$ then set $\lambda^+ \leftarrow 0$.

Set $\mathcal{A}^+ = \mathcal{A} \cup \{p\}$.

2. Check for feasibility and determine a new solution pair:

(a) Determine the step direction:

Compute $d = En^+$ (the step direction in the primal space).

If $q > 0$ then $r = B^*n^+$ (the negative of the step direction in the dual space).

(b) Compute the step length:

i. Partial step length, τ_1 :

This is the maximum step in dual space without violating dual feasibility.

If $r \leq 0$ or $q = 0$ then set $\tau_1 \leftarrow \infty$.

Otherwise, set

$$\tau_1 \leftarrow \min_{r_j > 0, j=1,\dots,q} \left\{ \frac{\lambda_j^+(x)}{r_j} \right\} = \frac{\lambda_l^+(x)}{r_l}$$

In Step 2c below, element $k \in \mathcal{K}$ corresponds to the l -th element in \mathcal{A} .

ii. Full step length, τ_2 :

This is the minimum step in primal space such that the p -th constraint becomes feasible.

If $|d| = 0$ then set $\tau_2 \leftarrow \infty$.

Otherwise, set $\tau_2 \leftarrow -\frac{g_p(x)}{d^T n^+}$.

iii. Step length, τ :

Set $\tau \leftarrow \min(\tau_1, \tau_2)$

(c) Determine a new solution pair and take the step:

i. No step in primal or dual space:

If $\tau = \infty$ then **stop**. The sub-problem $P(\mathcal{A}^+)$ and hence the quadratic problem are infeasible.

ii. Step in dual space:

If $\tau_2 = \infty$, then set $\lambda^+ \leftarrow \lambda^+ + \tau \begin{pmatrix} -r \\ 1 \end{pmatrix}$, and drop constraint k ;

i.e. set $\mathcal{A} \leftarrow \mathcal{A} \setminus \{k\}$, $q \leftarrow q - 1$, update E and B^* , and go to Step 2a.

iii. Step in primal and dual space:

Set $x \leftarrow x + \tau d$, $f \leftarrow f + \tau d^T n^+ (\frac{1}{2}\tau + \lambda_{q+1}^+)$, $\lambda^+ \leftarrow \lambda^+ + \tau \begin{pmatrix} -r \\ 1 \end{pmatrix}$.

If $\tau = \tau_2$ (a full step) then set $\lambda \leftarrow \lambda^+$ and add constraint p ;

i.e. set $\mathcal{A} \leftarrow \mathcal{A} \cup \{p\}$, $q \leftarrow q + 1$, update E and B^* and go to Step 1.

If $\tau = \tau_1$ (partial step) then drop constraint k ;

i.e. set $\mathcal{A} \leftarrow \mathcal{A} \setminus \{k\}$, $q \leftarrow q - 1$, update E and B^* , and go to Step 2a.

Some explanation of the notation is required. The problem statement is:

$$\begin{aligned} \text{minimise} \quad & f(x) = a^T x + \frac{1}{2} x^T H x \\ \text{subject to} \quad & g(x) \equiv G^T x - b \geq 0 \end{aligned}$$

This is equivalent to the linearised optimisation problem defined in Section F.4. The *Goldfarb-Idnani* dual algorithm commences with the unconstrained minimum $x = -H^{-1}a$ and adds constraints to the active constraint set, \mathcal{A} , having cardinality q , until all constraints are satisfied. In the following the set of all constraints is denoted \mathcal{K} , \mathcal{A}^+ denotes $\mathcal{A} \cup \{p\}$ where $p \in \mathcal{K} \setminus \mathcal{A}$ and \mathcal{A}^- represents the subset of \mathcal{A} that contains one fewer element than \mathcal{A} . A *subproblem* $P(\mathcal{A})$ is defined to be the quadratic programming problem subject only to the subset of the constraints indexed by $\mathcal{A} \subseteq \mathcal{K}$. If the solution x of a subproblem $P(\mathcal{J})$ lies on a linearly independent set of constraints indexed by $\mathcal{A} \subseteq \mathcal{J}$ then (x, \mathcal{A}) is called a *solution-pair* or *S-pair*. B , B^+ and B^- represent the matrices of constraint gradients corresponding to \mathcal{A} , \mathcal{A}^+ and \mathcal{A}^- . n^+ represents the gradient vector added to B to form B^+ . Similarly for n^- . I_k denotes the $k \times k$ identity matrix and e_j represents the j th column of I_k .

When the constraint gradients (columns of B) are linearly independent one can define the operators

$$\begin{aligned} B^* &= (B^T H^{-1} B)^{-1} B^T H^{-1} \\ E &= H^{-1} (I - BB^*) \end{aligned}$$

where B^* is the *pseudo-inverse* or *Moore-Penrose generalised inverse* of B . E is a reduced inverse Hessian operator for the quadratic $f(x)$ subject to the active set of constraints. In particular, as in [19], if \hat{x} is a point in the $(n-q)$ dimensional manifold $\mathcal{M} = \{x \in \mathbb{R}^n \mid n_i^T x = b_i, i \in \mathcal{A}\}$ and $\nabla_x f(\hat{x}) = H\hat{x} + a$ is the gradient of $f(x)$ at \hat{x} then the minimum of $f(x)$ over \mathcal{M} is attained at $\tilde{x} = \hat{x} - E\nabla_x f(\hat{x})$. For \tilde{x} to be the optimal solution for the subproblem $P(\mathcal{A})$

$$\nabla_x f(\tilde{x}) = B\lambda(\tilde{x})$$

where the vector of Lagrange multipliers $\lambda(\tilde{x}) \geq 0$. Multiplying both sides by B^* gives

$$\lambda(\tilde{x}) \equiv B^* \nabla_x f(\tilde{x}) \geq 0$$

Also multiplying by E gives

$$\begin{aligned} E \nabla f(\tilde{x}) &= H^{-1} (I - BB^*) B\lambda \\ &= H^{-1} (B - B) \lambda \\ &= 0 \end{aligned}$$

These conditions are necessary and sufficient for \tilde{x} to be the optimal solution to $P(\mathcal{A})$. In addition, the dual algorithm makes use of a set of, so-called, *infeasibility* multipliers

$$r = B^* n^+$$

Some properties of B^* and E :

$$\begin{aligned} Ew &= 0 \Leftrightarrow w = B\alpha \\ E &\text{ is positive semidefinite} \\ EWE &= E \\ B^*WE &= 0 \\ EE^+ &= E^+ \end{aligned}$$

When justifying Algorithm 29, *Goldfarb* and *Idnani* [19, p. 7] first point out that for a given S-pair (x, \mathcal{A}) and a violated constraint p , if the columns of B^+ (ie: those of B and $n^+ = n_p$) are linearly independent, then

$$\begin{aligned} g_p(x) &< 0 \\ g_i(x) &= 0 \quad \forall i \in \mathcal{A} \\ E^+ \nabla_x f(x) &= 0 \\ \lambda^+(x) &\triangleq (B^+)^* \nabla_x f(x) \geq 0 \end{aligned}$$

Definition 1: A triple (x, \mathcal{A}, p) consisting of a point x and a set of indices $\mathcal{A}^+ = \mathcal{A} \cup \{p\}$ where $p \in \mathcal{K} \setminus \mathcal{A}$ is said to be a V(violated)-triple if the columns of B^+ are linearly independent and the above conditions apply.

The point \hat{x} corresponding to the V-triple $(\hat{x}, \mathcal{A}, p)$ is the optimal solution to the subproblem obtained by replacing the constraint $g_p(x) \geq 0$ in $P(\mathcal{A}^+)$ by $g_p(x) \geq g_p(\hat{x})$ i.e. by a parallel constraint which passes through \hat{x} . Let x^* be the point on the manifold $\mathcal{M}^+ = \{x \in \mathbb{R}^n \mid n_i^T x = b_i, i \in \mathcal{A}^+\}$ at which $f(x)$ is minimized. The following lemma shows how to move to such a point from a point x corresponding to a V-triple (x, \mathcal{A}, p) .

Lemma 1: Let (x, \mathcal{A}, p) be a V-triple and consider points of the form $\tilde{x} = x + \tau d$ where $d = En^+$. Then

$$E^+ \nabla_x f(\tilde{x}) = 0 \quad (\text{F.6})$$

$$g_i(\tilde{x}) = 0 \quad \forall i \in \mathcal{A} \quad (\text{F.7})$$

$$\lambda^+(\tilde{x}) \triangleq (B^+)^* \nabla_x f(\tilde{x}) = \lambda^+(x) + \tau \begin{bmatrix} -r \\ 1 \end{bmatrix} \quad (\text{F.8})$$

where $r = B^* n^+$ and $g_p(\tilde{x}) = g_p(x) + \tau d^T n^+$.

Proof: The lemma follows from the properties of the V-triple (x, \mathcal{A}, p) and

$$\nabla_x f(\tilde{x}) = \nabla_x f(x) + \tau H d \quad (\text{F.9})$$

where

$$\begin{aligned} Hd &= HEn^+ \\ &= (I - BB^*) n^+ \\ &= n^+ - Br \\ &= [B \quad n^+] \begin{bmatrix} -r \\ 1 \end{bmatrix} \\ &= B^+ \begin{bmatrix} -r \\ 1 \end{bmatrix} \end{aligned}$$

Recalling that $EB = H^{-1}(B - BB^* B) = 0$, the results follow from multiplying Equation F.9 on the left by E^+ and B^+ .

It follows from this lemma that the point $\tilde{x} = x + \tau_2 d$, where $\tau_2 = -\frac{g_p(x)}{d^T n^+}$, minimises the quadratic function $f(x)$ over \mathcal{M}^+ (since then $g_p(\tilde{x}) = 0$). If $\lambda^+(\tilde{x}) \geq 0$ as well, then \tilde{x} is an optimal solution to $P(\mathcal{A}^+)$ and $(\tilde{x}, \mathcal{A}^+)$ is an S-pair. If not, then Equation F.8 implies that there is a smallest value τ_1 of τ , $\tau_1 < \tau_2$, such that some component of $\lambda^+(\tilde{x}(\tau)) < 0$ for $\tau > \tau_1$. If the constraint, say $k \in \mathcal{A}$, corresponding to this component is dropped from the active set, then $(\tilde{x}(\tau_1), \mathcal{A}^-, p)$, where $\mathcal{A}^- = \mathcal{A} \setminus \{k\}$, is again a V-triple. These remarks are formalised by the following two theorems.

Theorem 1: Given a V-triple (x, \mathcal{A}, p) if \tilde{x} is defined as in Lemma 1 with $\tau = \min\{\tau_1, \tau_2\}$ where

$$\begin{aligned} \tau_1 &= \min \left\{ \min_{r_j > 0, 1 \leq j \leq q} \left\{ \frac{\lambda_j^+(x)}{r_j^+(x)} \right\}, \infty \right\} \\ \tau_2 &= -\frac{g_p(x)}{d^T n^+} \end{aligned}$$

then

$$\begin{aligned} g_P(\tilde{x}) &\geq g_p(x) \\ f(\tilde{x}) - f(x) &= \tau d^T n^+ \left(\frac{1}{2}\tau + \lambda_{q+1}^+(x) \right) \geq 0 \end{aligned}$$

Moreover, if $\tau = \tau_1 = \frac{\lambda_l^+(x)}{r_l}$, then $(\tilde{x}, \mathcal{A} \setminus \{k\}, p)$ is a V-triple, where element $k \in \mathcal{K}$ corresponds to the l -th element in \mathcal{A} . Alternatively, if $\tau = \tau_2$, then $(\tilde{x}, \mathcal{A} \cup \{p\})$ is an S-pair.

Proof: Since (x, \mathcal{A}, p) is a V-triple,

$$d^T n^+ = n^{+T} E n^+ = n^{+T} E H E n^+ = d^T H d > 0$$

and $\tau \geq 0$. Hence, from Lemma 1, $g_P(\tilde{x}) \geq g_p(x)$. Also, from Taylor's theorem

$$f(\tilde{x}) - f(x) = \tau d^T \nabla_x f(x) + \frac{1}{2} \tau^2 d^T H d$$

Since $E^+ \nabla_x f(x) = 0$ implies that $\nabla_x f(x) = B^+ \lambda^+(x)$, it follows that $E \nabla_x f(x) = E n^+ \lambda_{q+1}^+(x)$ and

$$d^T \nabla_x f(x) = n^{+T} E \nabla_x f(x) = d^T n^+ \lambda_{q+1}^+(x) \geq 0$$

The result follows by substitution. Moreover, as long as $\tau > 0$, $f(\tilde{x}) > f(x)$. From the definition of τ and Lemma 1 it is evident that $E^+ \nabla_x f(\tilde{x}) = 0$, $g_i(\tilde{x}) = 0$, $i \in \mathcal{A}$ and $\lambda^+(\tilde{x}) \geq 0$. If $\tau = \tau_2$, then $g_p(\tilde{x}) = 0$ and $(x, \mathcal{A} \cup \{p\})$ is an S-pair. If $\tau = \tau_1 < \tau_2$, then $\lambda_l^+(\tilde{x}) = 0$ and $g_p(\tilde{x}) < 0$. Since $E^+ \nabla_x f(\tilde{x}) = 0$ and $\lambda_l^+(\tilde{x}) = 0$ we can write

$$\begin{aligned} \nabla_x f(\tilde{x}) &= B^+ \lambda^+(\tilde{x}) \\ &= \sum_{i \in \mathcal{A} \cup \{p\} \setminus \{k\}} \lambda_{j(i)}^+ n_i \end{aligned}$$

where i is the $j(i)$ -th index in $\mathcal{A}^+ = \mathcal{A} \cup \{p\}$. As the set of normals $\{n_i \mid i \in \mathcal{A} \cup \{p\} \setminus \{k\}\}$ is clearly linearly independent $(\tilde{x}, \mathcal{A} \setminus \{k\}, p)$ is a V-triple.

It follows from the above theorem that starting from a V-triple (x, \mathcal{A}, p) one can obtain an S-pair $(\tilde{x}, \tilde{\mathcal{A}} \cup \{p\})$ with $\tilde{\mathcal{A}} \subseteq \mathcal{A}$ and $f(\tilde{x}) > f(x)$ after at most q partial steps and one full step.

If n^+ is a linear combination of the columns of B at the start of Step 2, then (x, \mathcal{A}, p) is not a V-triple. In this case, either the subproblem $P(\mathcal{A} \cup \{p\})$ is infeasible or a constraint can be dropped from the active set \mathcal{A} so that (x, \mathcal{A}^-, p) is a V-triple. In the former case, the original quadratic problem must also be feasible, while in the latter case, one may proceed according to Theorem 1 to obtain a new S-pair with a higher function value.

Theorem 2: Let (x, \mathcal{A}) be an S-pair and p be an index of a constraint in $\mathcal{K} \setminus \mathcal{A}$ such that $n^+ \triangleq n_p = Br$ and $g_p(x) < 0$. If $r \leq 0$, then $P(\mathcal{A} \cup \{p\})$ is infeasible; otherwise the k -th component can be dropped from the active set, where k is determined by

$$\frac{\lambda_l(x)}{r_l} = \min_{r_j > 0, j=1,\dots,q} \left\{ \frac{\lambda_j(x)}{r_j} \right\}, \quad l = j(k)$$

to give $\mathcal{A}^- = \mathcal{A} \setminus \{k\}$ and the V-triple (x, \mathcal{A}^-, p) .

Proof: If there is a feasible solution $\tilde{x} = x + d$ satisfying the constraints $\mathcal{A} \cup \{p\}$, it is necessary that $n^{+T} d = r^T B^T d > 0$ and $B^T d \geq 0$ since $g_i(x) = 0$ for all $i \in \mathcal{A}$. But if $r \leq 0$, the two requirements in the theorem cannot be simultaneously satisfied; hence in this case $P(\mathcal{A} \cup \{p\})$ is infeasible. If a component of r is positive, it follows that $r_l > 0$ and that

$$n^+ = n_k r_l + \sum_{i \in \mathcal{A}^-} r_{j(i)} n_i$$

so

$$n_k = \frac{1}{r_l} \left[- \sum_{i \in \mathcal{A}^-} r_{j(i)} n_i + n^+ \right]$$

Since (x, \mathcal{A}) is an S-pair

$$\begin{aligned} \nabla_x f(x) &= \sum_{i \in \mathcal{A}^-} \lambda_{j(i)} n_i + \lambda_l n_k \\ &= \sum_{i \in \mathcal{A}^-} \left(\lambda_{j(i)} - \frac{\lambda_l}{r_l} r_{j(i)} \right) n_i + \frac{\lambda_l}{r_l} n^+ \end{aligned}$$

If we define $\hat{\mathcal{A}} = \mathcal{A}^- \cup \{p\}$, then it is clear that \hat{B} has full column rank, $\hat{E} \nabla_x f(x) = 0$ and

$$\begin{aligned} \hat{\lambda}(x) &= \hat{B}^* \nabla_x f(x) \\ &= \begin{cases} \lambda_{j(i)} - \frac{\lambda_l}{r_l} r_{j(i)} \geq 0 & i \in \mathcal{A}^- \\ \frac{\lambda_l}{r_l} \geq 0 & \end{cases} \end{aligned}$$

and hence that (x, \mathcal{A}^-, p) is a V-triple.

Goldfarb and *Idnani* [19, Section 4] base their implementation on the *Cholesky* factorisation

$$H = LL^T$$

of the positive definite symmetric Hessian matrix H and the *QR* factorisation

$$C = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_1 \mid Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix}$$

of the $(n \times q)$ matrix

$$C = L^{-1}B$$

where L is an $(n \times n)$ lower triangular matrix, R is a $(q \times q)$ upper triangular matrix and $Q = [Q_1 \mid Q_2]$ is a $(n \times n)$ orthogonal matrix partitioned so that Q_1 has q columns. By substitution,

$$\begin{aligned} B^* &= (B^T L^{-T} L^{-1} B)^{-1} B^T L^{-T} L^{-1} \\ &= (C^T C)^{-1} C^T L^{-1} \\ &= R^{-1} R^{-T} C^T L^{-1} \\ &= R^{-1} Q_1^T L^{-1} \\ &= R^{-1} J_1^T \end{aligned}$$

and

$$\begin{aligned} E &= L^{-T} L^{-1} - L^{-T} C (C^T C)^{-1} C^T L^{-1} \\ &= L^{-T} L^{-1} - L^{-T} C R^{-1} R^{-T} C^T L^{-1} \\ &= L^{-T} Q Q^T L^{-1} - L^{-T} Q_1 Q_1^T L^{-1} \\ &= L^{-T} Q_2 Q_2^T L^{-1} \\ &= J_2 J_2^T \end{aligned}$$

where

$$\begin{aligned} C^T C &= \begin{bmatrix} R^T & 0 \end{bmatrix} Q^T Q \begin{bmatrix} R \\ 0 \end{bmatrix} \\ &= R^T R \end{aligned}$$

and

$$\begin{aligned} C R^{-1} &= \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} R^{-1} \\ &= Q_1 \end{aligned}$$

and

$$\begin{aligned} J &= \begin{bmatrix} J_1 & J_2 \end{bmatrix} \\ &= \begin{bmatrix} L^{-T} Q_1 & L^{-T} Q_2 \end{bmatrix} \\ &= L^{-T} Q \end{aligned}$$

In the dual algorithm the vectors $d = En^+$ and $r = B^*n^+$ are required. Compute the intermediate vector

$$\begin{aligned} v &= J^T n^+ \\ &= \begin{bmatrix} J_1^T \\ J_2^T \end{bmatrix} n^+ \\ &= \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \end{aligned}$$

from which it follows that

$$d = J_2 v_2$$

and

$$r = R^{-1} v_1$$

Goldfarb and *Idnani* describe an efficient method for updating the factors J and R when a constraint is added or deleted.

F.10 Implementation examples

The archive distributed with this document contains Octave functions for the line-search and non-linear optimisation techniques described above.

The Octave file *sqp_common.m* contains the function to be optimised, constraints, gradients and the Hessian function:

$$\begin{aligned} \text{minimise } f(x) &= x_1^4 + x_2^4 + x_3^4 + x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + x_3 + 5 \\ \text{subject to } g(x) &= [x_2 - 1; -x_1 - \sqrt{2}; -x_3 - 0.5] \geq 0 \end{aligned}$$

linesearch_test.m implements unconstrained quasi-Newton optimisation of this function with step size found by line search with the Armijo, Goldstein, golden-section or quadratic-interpolation methods. Sample output of *linesearch_test.m* is:

```
Testing nosearch linesearch:
lastx=[ -0.930424,0.550527,-0.428572,],lastfx=4.717132
lastx=[ -0.246129,0.104520,-0.179015,],lastfx=4.683817
At x = [ -5.74793e-01 3.34409e-01 -5.74793e-01 ]
f(x) = 4.361595, 0.019958 secs
LINESEARCH nosearch 92 iterations 47 f(x) calls
LINESEARCH nosearch f(x)= 4.361595 x=[ -0.574793 0.334409 -0.574793 ]

Testing armijo linesearch:
At x = [ -5.74793e-01 3.34409e-01 -5.74793e-01 ]
f(x) = 4.361595, 0.016281 secs
LINESEARCH armijo 84 iterations 86 f(x) calls
LINESEARCH armijo f(x)= 4.361595 x=[ -0.574793 0.334409 -0.574793 ]

Testing armijo_kim linesearch:
At x = [ -5.74793e-01 3.34409e-01 -5.74793e-01 ]
f(x) = 4.361595, 0.017486 secs
LINESEARCH armijo_kim 84 iterations 86 f(x) calls
LINESEARCH armijo_kim f(x)= 4.361595 x=[ -0.574793 0.334409 -0.574793 ]

Testing goldstein linesearch:
lastx=[ 0.375450,-0.299914,0.050046,],lastfx=5.044727
lastx=[ 0.302120,-0.252487,0.022310,],lastfx=5.009165
lastx=[ 0.259218,-0.225077,0.005148,],lastfx=4.988203
At x = [ -5.74793e-01 3.34408e-01 -5.74792e-01 ]
f(x) = 4.361595, 0.023704 secs
LINESEARCH goldstein 94 iterations 157 f(x) calls
LINESEARCH goldstein f(x)= 4.361595 x=[ -0.574793 0.334408 -0.574792 ]

Testing goldensection linesearch:
At x = [ -5.74820e-01 3.34448e-01 -5.74814e-01 ]
f(x) = 4.361595, 0.118241 secs
LINESEARCH goldensection 124 iterations 1149 f(x) calls
LINESEARCH goldensection f(x)= 4.361595 x=[ -0.574820 0.334448 -0.574814 ]

Testing quadratic linesearch:
At x = [ -5.74836e-01 3.34460e-01 -5.74793e-01 ]
f(x) = 4.361595, 0.023496 secs
LINESEARCH quadratic 118 iterations 119 f(x) calls
LINESEARCH quadratic f(x)= 4.361595 x=[ -0.574836 0.334460 -0.574793 ]
```

sqp_gi_test.m tests the *Goldfarb-Idnani* algorithm implemented in *goldfarb_idnani.m*. Sample output is:

```
Initial x0 = [ 30.000000 -20.000000 10.000000 ]
Active constraints are []
Step 1: Trying constraint 2 at g(2) = [ -31.414214 ]
Step 2a: step direction in primal space d = [ -0.000093 0.000000 0.000000 ]
Step 2b)iii): full step length t2 = 339273.473761
Step 2b)iii): selecting step length t = 339273.473761
Step 2c)iii): Step in primal and dual space.
Next x = [ -1.414214 -19.993461 10.026173 ]
```

```

f(x) =169707.209457
Adding constraint 2
Active constraints are [ 2 ]
Step 1: Trying constraint 1 at g(1) = [ -20.993461 ]
Step 2a: step direction in primal space d = [ -0.000000 0.000208 -0.000000 ]
Step 2a: step direction in dual space r = [ 0.000208 ]
Step 2b)i): partial step length t1 = 1629334390.615776
Step 2b)ii): full step length t2 = 100768.594465
Step 2b)iii): selecting step length t = 100768.594465
Step 2c)iii): Step in primal and dual space.
Next x = [ -1.414214 1.000000 10.008679 ]
f(x) =10048.793782
Adding constraint 1
Active constraints are [ 2 1 ]
Step 1: Trying constraint 3 at g(3) = [ -10.508679 ]
Step 2a: step direction in primal space d = [ -0.000000 -0.000000 -0.000833 ]
Step 2a: step direction in dual space r = [ -0.000831 0.000830 ]
Step 2b)i): partial step length t1 = 121345621.364403
Step 2b)ii): full step length t2 = 12610.414214
Step 2b)iii): selecting step length t = 12610.414214
Step 2c)iii): Step in primal and dual space.
Next x = [ -1.414214 1.000000 -0.500000 ]
f(x) =7.941180
Adding constraint 3
Active constraints are [ 2 1 3 ]
All constraints satisfied. Feasible solution found.
x=[ -1.414214 1.000000 -0.500000] fx=7.941180 4 iterations

```

The script *sqp_bfgs_test.m* tests constrained quasi-Newton optimisation with combinations of Hessian initialisation, Hessian update and linesearch type. Note that the *goldensection* linesearch requires many more function calls than the other types. Sample output filtered for “SQP” is:

```

SQP init hessian linesearch x feasible iter fiter liter
SQP GI exact nosearch [ -1.414214 1.000000 -0.527104 ] 1 3 9 0
SQP GI exact quadratic [ -1.414214 1.000000 -0.526902 ] 1 8 22 8
SQP GI exact armijo [ -1.414214 1.000000 -0.527104 ] 1 3 12 3
SQP GI exact armijo_kim [ -1.414214 1.000000 -0.527104 ] 1 3 12 3
SQP GI exact goldstein [ -1.414214 1.000000 -0.527104 ] 1 3 15 6
SQP GI exact goldensection [ -1.414214 1.000000 -0.526910 ] 1 8 62 48
SQP GI bfgs nosearch [ -1.414214 1.000000 -0.527057 ] 1 6 12 0
SQP GI bfgs quadratic [ -1.414214 1.000000 -0.526930 ] 1 11 28 11
SQP GI bfgs armijo [ -1.414214 1.000000 -0.527057 ] 1 6 18 6
SQP GI bfgs armijo_kim [ -1.414214 1.000000 -0.527057 ] 1 6 18 6
SQP GI bfgs goldstein [ -1.414214 1.000000 -0.527369 ] 1 5 29 18
SQP GI bfgs goldensection [ -1.414214 1.000000 -0.526935 ] 1 11 83 66
SQP GI diagonal nosearch [ -1.414214 1.000000 -0.527100 ] 1 4 10 0
SQP GI diagonal quadratic [ -1.414214 1.000000 -0.526902 ] 1 8 22 8
SQP GI diagonal armijo [ -1.414214 1.000000 -0.527100 ] 1 4 14 4
SQP GI diagonal armijo_kim [ -1.414214 1.000000 -0.527100 ] 1 4 14 4
SQP GI diagonal goldstein [ -1.414214 1.000000 -0.527100 ] 1 4 18 8
SQP GI diagonal goldensection [ -1.414214 1.000000 -0.526910 ] 1 8 62 48
SQP GI eye nosearch [ -1.414214 1.000000 -0.527057 ] 1 6 12 0
SQP GI eye quadratic [ -1.414214 1.000000 -0.526930 ] 1 11 28 11
SQP GI eye armijo [ -1.414214 1.000000 -0.527057 ] 1 6 18 6
SQP GI eye armijo_kim [ -1.414214 1.000000 -0.527057 ] 1 6 18 6
SQP GI eye goldstein [ -1.414214 1.000000 -0.527369 ] 1 5 29 18
SQP GI eye goldensection [ -1.414214 1.000000 -0.526935 ] 1 11 83 66
SQP eye exact nosearch [ -1.414214 1.000000 -0.527127 ] 1 4 6 0
SQP eye exact quadratic [ -1.414192 0.999986 -0.527111 ] 1 19 40 19
SQP eye exact armijo [ -1.414214 1.000000 -0.527104 ] 1 6 27 19
SQP eye exact armijo_kim [ -1.414214 1.000000 -0.527104 ] 1 6 27 19
SQP eye exact goldstein [ -1.414214 1.000000 -0.527104 ] 1 6 33 25
SQP eye exact goldensection [ -1.414191 0.999985 -0.527110 ] 1 20 321 299
SQP eye bfgs nosearch [ -1.414214 1.000000 -0.527108 ] 1 6 8 0
SQP eye bfgs quadratic [ -1.414185 0.999973 -0.527072 ] 1 33 68 33
SQP eye bfgs armijo [ -1.414214 1.000000 -0.527095 ] 1 9 44 33
SQP eye bfgs armijo_kim [ -1.414214 1.000000 -0.527095 ] 1 9 44 33
SQP eye bfgs goldstein [ -1.414214 1.000000 -0.527097 ] 1 9 54 43

```

```

SQP eye bfgs goldensection [ -1.414182 0.999999 -0.527114 ] 1 33 699 664
SQP eye diagonal nosearch [ -1.414214 1.000000 -0.527100 ] 1 6 8 0
SQP eye diagonal quadratic [ -1.414180 0.999986 -0.527180 ] 1 18 38 18
SQP eye diagonal armijo [ -1.414214 1.000000 -0.527100 ] 1 7 29 20
SQP eye diagonal armijo_kim [ -1.414214 1.000000 -0.527100 ] 1 7 29 20
SQP eye diagonal goldstein [ -1.414214 1.000000 -0.527100 ] 1 7 36 27
SQP eye diagonal goldensection [ -1.414193 0.999991 -0.527150 ] 1 20 319 297
SQP eye eye nosearch [ -1.414214 1.000000 -0.527108 ] 1 6 8 0
SQP eye eye quadratic [ -1.414185 0.999973 -0.527072 ] 1 33 68 33
SQP eye eye armijo [ -1.414214 1.000000 -0.527095 ] 1 9 44 33
SQP eye eye armijo_kim [ -1.414214 1.000000 -0.527095 ] 1 9 44 33
SQP eye eye goldstein [ -1.414214 1.000000 -0.527097 ] 1 9 54 43
SQP eye eye goldensection [ -1.414182 0.999999 -0.527114 ] 1 33 699 664
SQP none exact nosearch [ -1.414214 1.000000 -0.527104 ] 1 4 6 0
SQP none exact quadratic [ -1.414191 0.999985 -0.527110 ] 1 17 36 17
SQP none exact armijo [ -1.414214 1.000000 -0.527104 ] 1 4 12 6
SQP none exact armijo_kim [ -1.414214 1.000000 -0.527104 ] 1 4 12 6
SQP none exact goldstein [ -1.414214 1.000000 -0.527104 ] 1 4 16 10
SQP none exact goldensection [ -1.414190 0.999984 -0.527110 ] 1 18 264 244
SQP none bfgs nosearch [ -1.414214 1.000000 -0.527073 ] 1 8 10 0
SQP none bfgs quadratic [ -1.414213 1.000000 -0.526831 ] 1 25 52 25
SQP none bfgs armijo [ -1.414214 1.000000 -0.527073 ] 1 8 20 10
SQP none bfgs armijo_kim [ -1.414214 1.000000 -0.527073 ] 1 8 20 10
SQP none bfgs goldstein [ -1.414214 1.000000 -0.527227 ] 1 7 38 29
SQP none bfgs goldensection [ -1.414213 1.000000 -0.526940 ] 1 23 309 284
SQP none diagonal nosearch [ -1.414214 1.000000 -0.527100 ] 1 5 7 0
SQP none diagonal quadratic [ -1.414178 0.999985 -0.527182 ] 1 16 34 16
SQP none diagonal armijo [ -1.414214 1.000000 -0.527100 ] 1 5 13 6
SQP none diagonal armijo_kim [ -1.414214 1.000000 -0.527100 ] 1 5 13 6
SQP none diagonal goldstein [ -1.414214 1.000000 -0.527100 ] 1 5 18 11
SQP none diagonal goldensection [ -1.414192 0.999990 -0.527151 ] 1 18 262 242
SQP none eye nosearch [ -1.414214 1.000000 -0.527108 ] 1 6 8 0
SQP none eye quadratic [ -1.414185 0.999973 -0.527072 ] 1 33 68 33
SQP none eye armijo [ -1.414214 1.000000 -0.527095 ] 1 9 44 33
SQP none eye armijo_kim [ -1.414214 1.000000 -0.527095 ] 1 9 44 33
SQP none eye goldstein [ -1.414214 1.000000 -0.527097 ] 1 9 54 43
SQP none eye goldensection [ -1.414182 0.999999 -0.527114 ] 1 33 699 664

```


Appendix G

Low passband sensitivity IIR filters

Vaidyanathan *et al.* [72, 71] describe the synthesis of IIR digital filters as the parallel connection of two all-pass filters. The resulting filter has low coefficient sensitivity if the all-pass filter implementation is *structurally loss-less* (ie: the all-pass transfer function is preserved when the coefficients are truncated).

G.1 Structural Boundedness

Consider the N -th order IIR filter

$$G(z) = \frac{P(z)}{D(z)} = \frac{p_0 + p_1 z^{-1} + \cdots + p_N z^{-N}}{1 + d_1 z^{-1} + \cdots + d_N z^{-N}}$$

where the coefficients p_i and d_i are real. We wish to design a structure with multiplier coefficients m_0, m_1, \dots such that the sensitivity of $|G(e^{i\omega})|$ with respect to each m_i is very small in the pass-band. If $|G(e^{i\omega})| \leq 1$ for all ω regardless of the values of the multipliers (so long as they are within a certain range) then the implementation is called *structurally bounded* or *structurally passive* and $G(z)$ is called *bounded real*. If $|G(e^{i\omega})| = 1$ for certain frequencies ω_k in the passband, then at those frequencies, when an m_i is perturbed the value of $|G(e^{i\omega})|$ can only decrease. In other words the first-order sensitivity is zero at ω_k

$$\left. \frac{\partial |G(e^{i\omega_k})|}{\partial m_i} \right|_{m_i=m_{i_0}} = 0 \quad \forall i, \forall k$$

Now consider a stable all-pass function, $A_1(z)$

$$A_1(z) = \frac{a_m + a_{m-1}z^{-1} + \cdots + z^{-m}}{1 + a_1 z^{-1} + \cdots + a_m z^{-m}}$$

$$|A_1(e^{i\omega})| = 1$$

where a_k are real. A number of well-known structures exist for which the mirror images of the denominator and numerator coefficients are preserved in spite of multiplier quantization. Such implementations are called *structurally lossless*.

Now consider a parallel connection of two stable all-pass filters $A_1(z)$ and $A_2(z)$ with

$$A_2(z) = \frac{b_n + b_{n-1}z^{-1} + \cdots + z^{-n}}{1 + b_1 z^{-1} + \cdots + b_n z^{-n}}$$

and

$$G(z) = \frac{1}{2} [A_1(z) + A_2(z)]$$

Since $A_1(z)$ and $A_2(z)$ are all-pass functions:

$$A_1(z) = z^{-n_1} \frac{\hat{D}_1(z)}{D_1(z)}$$

$$A_2(z) = z^{-n_2} \frac{\hat{D}_2(z)}{D_2(z)}$$

where n_1 and n_2 are non-negative, $\hat{D}_1(z)$ denotes the mirror image of $D_1(z)$ and

$$G(z) = \frac{1}{2} \left[\frac{z^{-n_1} D_2(z) \hat{D}_1(z) + z^{-n_2} D_1(z) \hat{D}_2(z)}{D_1(z) D_2(z)} \right]$$

If $A_1(z)$ and $A_2(z)$ are minimal and if $D_1(z)$ and $D_2(z)$ have no common factors then there is no pole-zero cancellation and $G(z)$ has order $N = n + m$.

On the unit circle

$$G(e^{i\omega}) = \frac{1}{2} [e^{i\theta_1(\omega)} + e^{i\theta_2(\omega)}]$$

where $\theta_1(\omega)$ and $\theta_2(\omega)$ are real-valued functions of ω . Thus,

$$|G(e^{i\omega})| = \frac{1}{2} \left| 1 + e^{i[\theta_2(\omega) - \theta_1(\omega)]} \right|$$

and if $A_1(z) \neq A_2(z)$ then $G(z)$ cannot be all-pass. Also, if $A_1(z)$ and $A_2(z)$ are implemented so that they remain all-pass in spite of parameter quantisation, then $|G(e^{i\omega})| \leq 1$ for all ω . Accordingly, the structural lossless-ness of $A_1(z)$ and $A_2(z)$ induces structural bounded-ness in $G(z)$.

G.2 Filter realisation as the sum of all-pass functions

Consider a typical N -th order bounded real transfer function $G(z) = P(z)/D(z)$ where $P(z)$ is symmetric, i.e.: $p_k = p_{N-k}$. Now consider another transfer function $H(z)$ where

$$\begin{aligned} H(z) &= \frac{Q(z)}{D(z)} \\ &= \frac{q_0 + q_1 z^{-1} + \cdots + q_N z^{-N}}{1 + d_1 z + \cdots + d_N z^{-N}} \end{aligned}$$

and $H(z)$ is complementary to $G(z)$

$$|H(e^{i\omega})|^2 = 1 - |G(e^{i\omega})|^2$$

In terms of the z -variable

$$\tilde{P}(z)P(z) + \tilde{Q}(z)Q(z) = \tilde{D}(z)D(z)$$

where $\tilde{P}(z) = P(z^{-1})$ etc.

Computation of the spectral factor $Q(z)$ is simplified by the anti-symmetric nature of its coefficients.

$$Q^2(z) = P^2(z) - z^{-N} D(z^{-1}) D(z)$$

$D(z)$ and $P(z)$ are known so the right hand side can be written

$$R(z) = \sum_{n=0}^{2N} r_n z^{-n}$$

Then the coefficients of $Q(z)$ are related to r_n by

$$r_n = \sum_{k=0}^n q_k q_{n-k}$$

and the q_k can be computed recursively

$$\begin{aligned} q_0 &= \sqrt{r_0} \\ q_1 &= \frac{r_1}{2q_0} \\ q_n &= \frac{r_n - \sum_{k=1}^{n-1} q_k q_{n-k}}{2q_0}, \quad 2 \leq n \leq N \end{aligned}$$

The spectral factor, $Q(z)$ is calculated by Octave function *spectralfactor*.

Assume $G(z)$ is such that $Q(z)$ is anti-symmetric i.e.: $q_i = -q_{N-i}$. So

$$\begin{aligned}\tilde{P}(z) &\triangleq P(z^{-1}) = z^N P(z) \\ \tilde{Q}(z) &\triangleq Q(z^{-1}) = -z^N Q(z)\end{aligned}$$

and

$$\begin{aligned}[P(z) + Q(z)][P(z) - Q(z)] &= z^{-N} D(z) D(z^{-1}) \\ [P(z) + Q(z)][P(z^{-1}) + Q(z^{-1})] &= D(z) D(z^{-1})\end{aligned}$$

Moreover

$$P(z^{-1}) + Q(z^{-1}) = z^N [P(z) - Q(z)]$$

Hence the zeros of $P(z) + Q(z)$ are the reciprocals of the zeros of $P(z) - Q(z)$. Since $G(z)$ is stable, none of its poles are on the unit circle and

$$P(z) + Q(z) \neq 0, \quad |z| = 1$$

Let z_1, z_2, \dots, z_r be the zeros of $P(z) + Q(z)$ inside the unit circle and let $z_{r+1}, z_{r+2}, \dots, z_N$ be those outside. Then

$$D(z) = \prod_{k=1}^r (1 - z^{-1} z_k) \prod_{k=r+1}^N (1 - z^{-1} z_k^{-1})$$

and

$$[P(z) + Q(z)][P(z) - Q(z)] = \left[\prod_{k=1}^r (1 - z^{-1} z_k) \prod_{k=r+1}^N (1 - z^{-1} z_k^{-1}) \right] \left[\prod_{k=1}^r (z^{-1} - z_k) \prod_{k=r+1}^N (z^{-1} - z_k^{-1}) \right]$$

From which

$$\begin{aligned}P(z) + Q(z) &= \alpha \left[\prod_{k=1}^r (1 - z^{-1} z_k) \prod_{k=r+1}^N (z^{-1} - z_k^{-1}) \right] \\ P(z) - Q(z) &= \frac{1}{\alpha} \left[\prod_{k=1}^r (z^{-1} - z_k) \prod_{k=r+1}^N (1 - z^{-1} z_k^{-1}) \right]\end{aligned}$$

where α is a real constant. This leads to the equations

$$\begin{aligned}G(z) + H(z) &= \frac{P(z) + Q(z)}{D(z)} = \alpha \prod_{k=r+1}^N \left(\frac{z^{-1} - z_k^{-1}}{1 - z^{-1} z_k^{-1}} \right) \\ G(z) - H(z) &= \frac{P(z) - Q(z)}{D(z)} = \frac{1}{\alpha} \prod_{k=1}^r \left(\frac{z^{-1} - z_k}{1 - z^{-1} z_k} \right)\end{aligned}$$

Thus

$$\begin{aligned}G(z) + H(z) &= \alpha A_1(z) \\ G(z) - H(z) &= \frac{1}{\alpha} A_2(z)\end{aligned}$$

where $A_1(z)$ and $A_2(z)$ are stable all-pass functions of order $N - r$ and r respectively

$$\begin{aligned}A_1(z) &= \prod_{k=r+1}^N \left(\frac{z^{-1} - z_k^{-1}}{1 - z^{-1} z_k^{-1}} \right) \\ A_2(z) &= \prod_{k=1}^r \left(\frac{z^{-1} - z_k}{1 - z^{-1} z_k} \right)\end{aligned}$$

The complementarity condition requires $\alpha^2 = 1$ so, finally

$$G(z) = \frac{1}{2} [A_1(z) + A_2(z)]$$

Algorithm 30 Filter realisation as the sum of all-pass functions

Let $G(z) = P(z)/D(z)$ be a bounded real function of order N and let $P(z)$ be symmetric i.e.: $p_i = p_{N-i}$. In addition, let $G(z)$ be such that an anti-symmetric polynomial $Q(z)$ (i.e.: $q_i = -q_{N-i}$) exists such that

$$\tilde{P}(z)P(z) + \tilde{Q}(z)Q(z) = \tilde{D}(z)D(z)$$

$G(z)$ can be implemented as the combination of two stable all-pass functions $A_1(z)$ and $A_2(z)$

$$G(z) = \frac{1}{2} [A_1(z) + A_2(z)]$$
$$H(z) = \frac{1}{2} [A_1(z) - A_2(z)]$$

Furthermore, $H(z)$ is also bounded real and is doubly complementary to $G(z)$.

$$H(z) = \frac{1}{2} [A_1(z) - A_2(z)]$$

and $G(z)$ is implemented as the parallel combination of two all-pass functions.

This low-sensitivity realisation is summarised in Algorithm 30.

The classical Butterworth, Chebychev and Cauer low-pass digital filters of odd order satisfy the following conditions and can be implemented as the combination of two stable all-pass functions

1. N is odd
2. $\{\partial^k |G(e^{i\omega})| / \partial \omega^k\}|_{\omega=0} = 0$ for $k = 1, 2, \dots, n_0$ where n_0 is some odd integer
(In other words, $|G(e^{i\omega})|$ has odd-order tangency at zero frequency).
3. $|G(1)| = 1$
4. There are $(N - n_0)/2$ frequencies in the range $0 < \omega < \pi$ where $|G(e^{i\omega})| = 1$

Other filter pass-bands are obtained by frequency transformation.

The Octave script *vaidyanathan_allpass_example.m* implements the example transfer function shown in [72, Section V].

G.3 A note on the numerical calculation of the spectral factor

The Octave implementation of the calculation of the spectral factor is in the Octave function file *spectralfactor.m*. The file *spectralfactor.cc* shows a C++ version of *spectralfactor* using the *MPFR* arbitrary precision floating point library [51, 75, 1]. The mantissa precision is set to 256 bits.

The Octave script *spectralfactor_test.m* illustrates calculation of the spectral factor for a 13th order elliptic filter with cut-off frequency $0.05f_S$, pass-band ripple 0.0005dB and stop-band ripple 40dB using *spectralfactor.oct*. Figures G.1 and G.2 show the transfer functions of the elliptic filter, the spectral factor complementary filter and the summed transfer function. The summed response ripple is satisfactory for order 13 but fails catastrophically for order 15. I suspect that this is due to roundoff error in the original calculation of the elliptic filter coefficients that is visible in the transition band detailed view. The prototype elliptic filter response appears to be not structurally bounded.

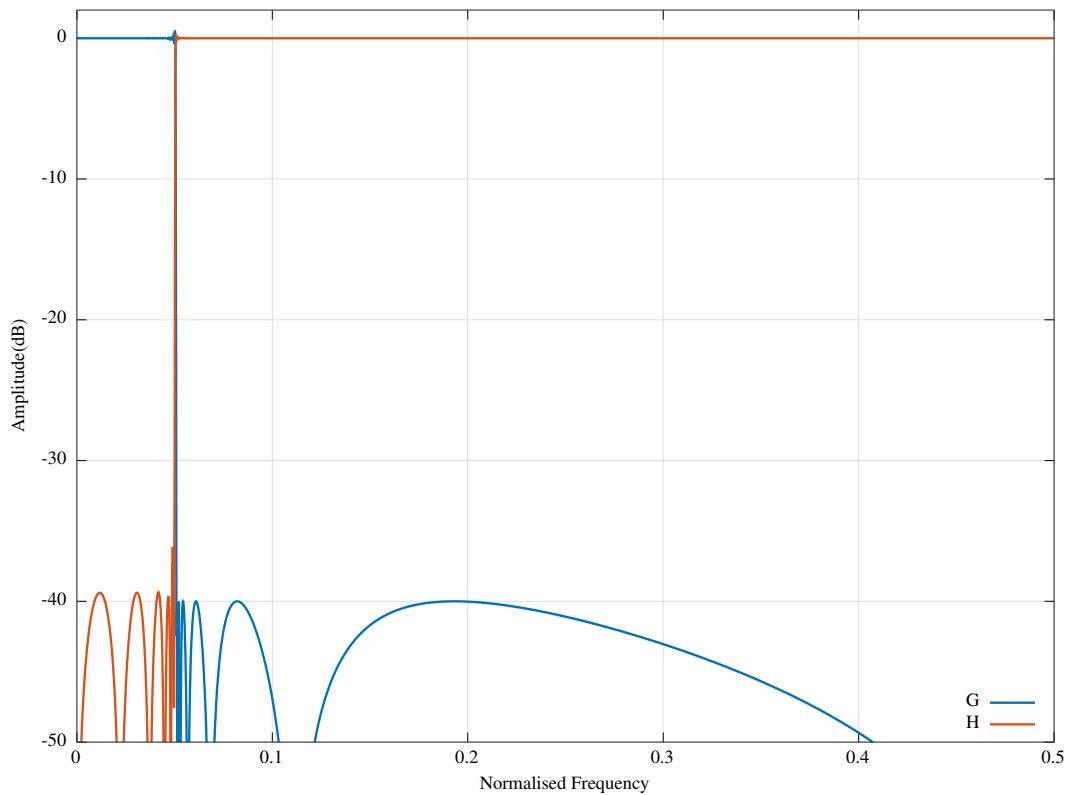


Figure G.1: 13th order elliptic filter, complementary filter and summed transfer functions

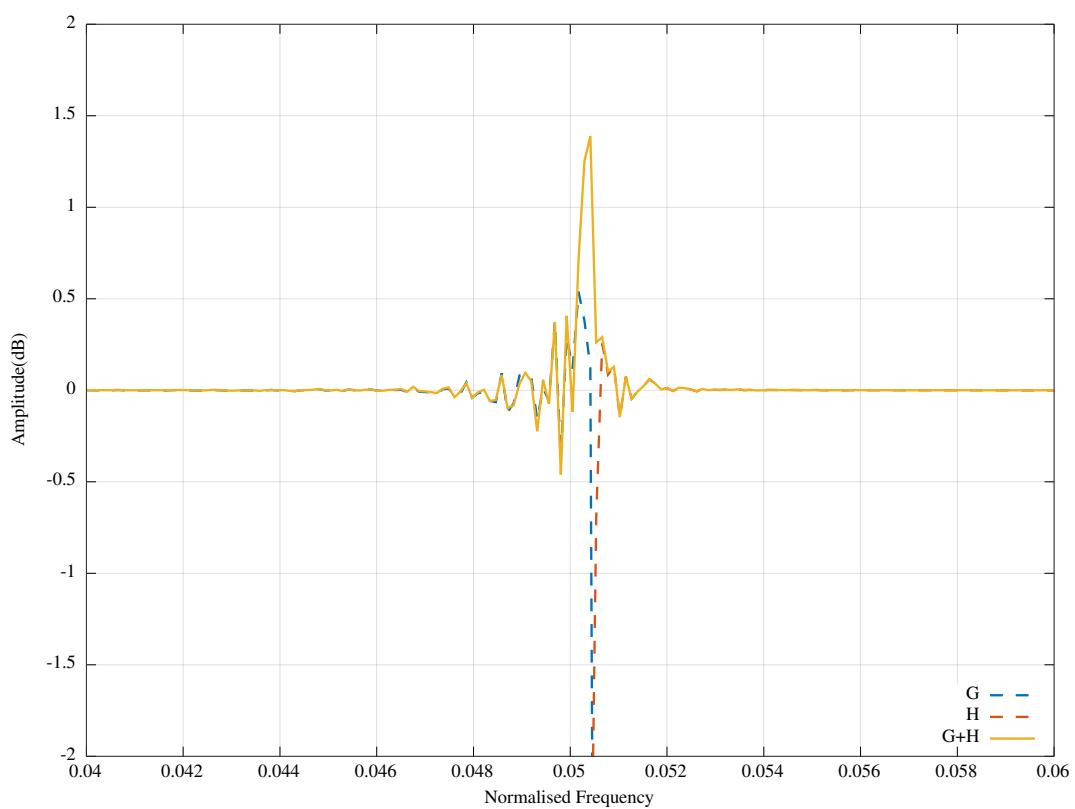


Figure G.2: 13th order elliptic filter, complementary filter and summed transfer functions transition band detail

G.4 Examples of parallel all-pass filter synthesis

G.4.1 3rd order Butterworth low-pass filter

The Octave script *butt3NSPA_test.m* shows synthesis as a parallel combination of two all-pass filters of the 3rd order Butterworth filter used in the example of Part I. Annotated results of the script follow.

The numerator and denominator polynomials, $N(z)$ and $D(z)$ are:

```
n = 0.0028982 0.0086946 0.0086946 0.0028982
d = 1.00000 -2.37409 1.92936 -0.53208
```

The spectral factor $Q(z)$ is:

```
q = 0.72944 -2.18832 2.18832 -0.72944
```

The sum $N(z) + Q(z)$ has roots:

```
z = 1.12486 + 0.31652i
    1.12486 - 0.31652i
    0.72654 + 0.00000i
```

The polynomials for the all-pass components are:

```
A1 = 0.73234 -1.64755 1.00000
A2 = -0.72654 1.00000
```

After quantising to three 10 bit signed-digits the coefficients for the normalised-scaled lattice implementation are:

```
A1s10f = [ -9.53125000e-01 7.34375000e-01 ]
A1s11f = [ 3.08593750e-01 6.87500000e-01 ]
A1s20f = [ -9.53125000e-01 7.34375000e-01 ]
A1s00f = [ 3.08593750e-01 6.87500000e-01 ]
A1s02f = [ 9.53125000e-01 -7.34375000e-01 ]
A1s22f = [ 3.08593750e-01 6.87500000e-01 ]
```

and

```
A2s10f = [ -7.18750000e-01 ]
A2s11f = [ 6.87500000e-01 ]
A2s20f = [ -7.18750000e-01 ]
A2s00f = [ 6.87500000e-01 ]
A2s02f = [ 7.18750000e-01 ]
A2s22f = [ 6.87500000e-01 ]
```

The noise gains of each part with three signed-digit coefficients are:

```
A1ngapABCDf = 3.1899
A2ngapABCDf = 0.95500
```

The estimated and measured round-off noise variances at the combined all-pass output are:

```
est_varA1yapd = 0.34916
varA1yapd = 0.34396
est_varA2yapd = 0.16292
varA2yapd = 0.16222
est_varyapd = 0.25302
varyapd = 0.26062
```

The standard deviations of the internal state delay storage elements are:

```
A1stdxf = 133.64 130.73
A2stdxf = 128.18
```

The amplitude response found from the cross-correlation of the input and output is shown in Figure G.3.

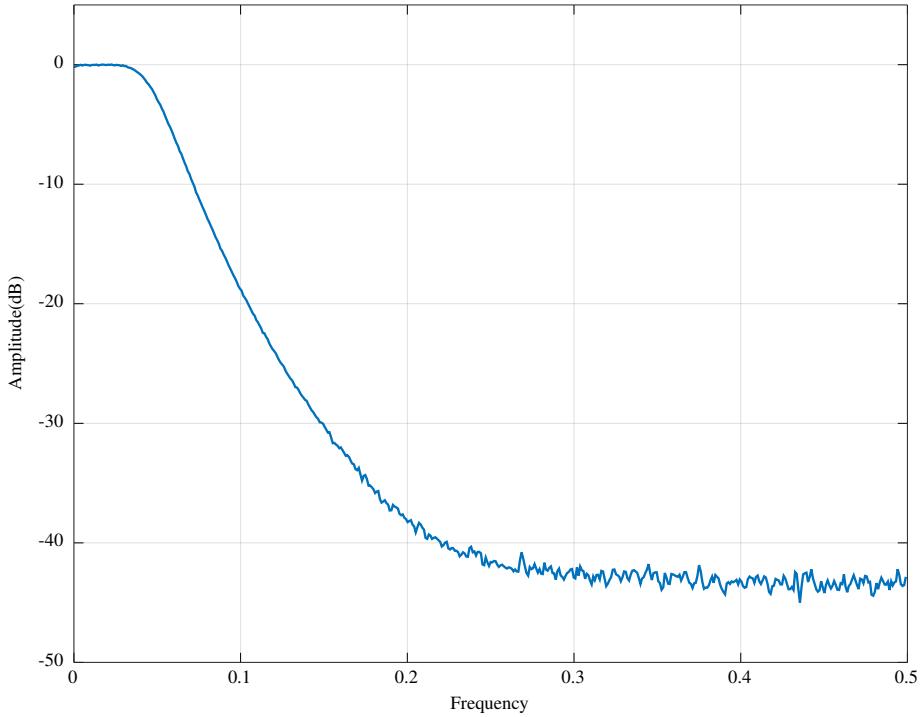


Figure G.3: Simulated amplitude response of the 3rd order Butterworth filter synthesised as the parallel combination of two all-pass filters implemented as normalised-scaled lattices with 10 bit 3 signed-digit coefficients

G.4.2 6th order Butterworth band-pass

The Octave script *butt6NSPABP_test.m* shows synthesis as a parallel combination of two all-pass filters of a 6th order band-pass Butterworth filter. Annotated results of the script follow.

The prototype filter has cutoff frequency:

```
fC = 0.25000
```

The numerator and denominator polynomials, $N(z)$ and $D(z)$ are:

```
n = 0.16667 0.50000 0.50000 0.16667
d = 1.0000e+00 -3.0531e-16 3.3333e-01 -1.8504e-17
```

The spectral factor $Q(z)$ is:

```
q = 0.16667 -0.50000 0.50000 -0.16667
```

The sum $N(z) + Q(z)$ has roots

```
z = 0.00000 + 1.73205i
    0.00000 - 1.73205i
    0.00000 + 0.00000i
```

The polynomials for the all-pass components of the prototype filter are:

```
Aap1 = 3.3333e-01 0.0000e+00 1.0000e+00
Aap2 = 0.0000e-00 1.0000e+00
```

The frequency transformation polynomial for a band-pass filter with band edges $0.2f_S$ and $0.25f_S$ is

```
p = 1.00000 -0.27346 0.72654
```

The all-pass polynomials of the parallel components of the band-pass filter are

```
A1BP = 1.0000e+00 -6.7309e-01 2.3655e+00 -7.8886e-01 1.3655e+00
```

and

```
A2BP = 1.0000e+00 -3.7638e-01 1.3764e+00
```

After truncating to 10 bit 3-signed-digits the coefficients for the normalised-scaled lattice implementation are:

```
A1s10f = [ -1.64062500e-01 9.5312500e-01 -1.4843750e-01 7.3437500e-01 ]
A1s11f = [ 9.86328125e-01 3.0859375e-01 9.8828125e-01 6.8750000e-01 ]
A1s20f = [ -1.64062500e-01 9.5312500e-01 -1.4843750e-01 7.3437500e-01 ]
A1s00f = [ 9.86328125e-01 3.0859375e-01 9.8828125e-01 6.8750000e-01 ]
A1s02f = [ 1.64062500e-01 -9.5312500e-01 1.4843750e-01 -7.3437500e-01 ]
A1s22f = [ 9.84375000e-01 3.0859375e-01 9.8828125e-01 6.8750000e-01 ]
```

and

```
A2s10f = [ -1.58203125e-01 7.1875000e-01 ]
A2s11f = [ 9.88281250e-01 6.8750000e-01 ]
A2s20f = [ -1.58203125e-01 7.1875000e-01 ]
A2s00f = [ 9.88281250e-01 6.8750000e-01 ]
A2s02f = [ 1.58203125e-01 -7.1875000e-01 ]
A2s22f = [ 9.88281250e-01 6.8750000e-01 ]
```

The noise gains of each part are:

```
A1ngapABCDf = 7.5062e+00
A2ngapABCDf = 2.8995e+00
```

The estimated and measured round-off noise variances at the combined all-pass output are:

```
est_varA1yapd = 7.0885e-01
varA1yapd = 6.8465e-01
est_varA2yapd = 3.2496e-01
varA2yapd = 3.3069e-01
est_varyapd = 3.8345e-01
varyapd = 3.8520e-01
```

The standard deviations of the internal state delay storage elements are:

```
A1stdxf = 1.2867e+02 1.2865e+02 1.2928e+02 1.2937e+02
A2stdxf = 1.2674e+02 1.2657e+02
```

The amplitude response found from the cross-correlation of the input and output is shown in Figure G.4.

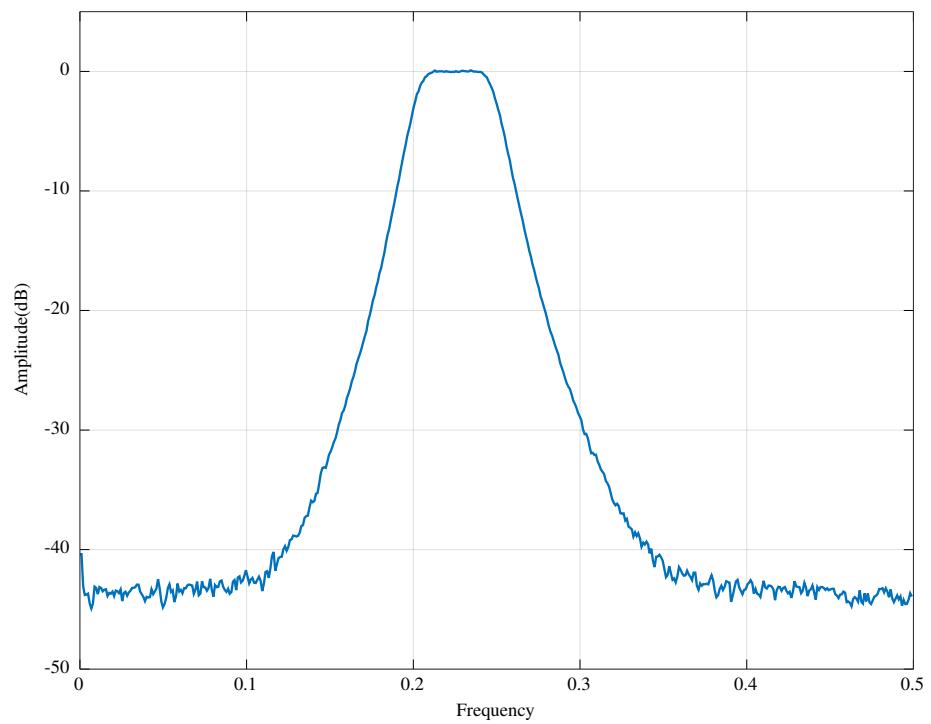


Figure G.4: Simulated amplitude response of the 6th order Butterworth band-pass filter synthesised as the parallel combination of two all-pass filters implemented as normalised-scaled lattices with 10 bit 3 signed-digit coefficients

Appendix H

Low passband sensitivity FIR lattice filters

H.1 Lattice decomposition of an FIR digital filter

Vaidyanathan [70] describes the design of low passband sensitivity FIR digital filters based on the lattice decomposition of a bounded-real FIR filter, $H(z)$, such that $|H(e^{i\omega})| \leq 1$, and the bounded-real complementary filter, $G(z)$, with $|H(e^{i\omega})|^2 + |G(e^{i\omega})|^2 = 1$. Assume that $H_{m+1}(z)$ and $G_{m+1}(z)$ are complementary FIR filter transfer function polynomials of order $m+1$:

$$H_{m+1}(z) = \sum_{n=0}^{m+1} h_{m+1,n} z^{-n} \quad (\text{H.1})$$

$$G_{m+1}(z) = \sum_{n=0}^{m+1} g_{m+1,n} z^{-n} \quad (\text{H.2})$$

and write $\tilde{H}_{m+1}(z) = H_{m+1}^T(z^{-1})$, where T denotes transpose.

$A_{m+1}(z) = [H_{m+1}(z) \ G_{m+1}(z)]^T$ represents the corresponding all-pass filter:

$$\tilde{A}_{m+1}(z) A_{m+1}(z) = 1$$

or

$$\tilde{H}_{m+1}(z) H_{m+1}(z) + \tilde{G}_{m+1}(z) G_{m+1}(z) = 1 \quad (\text{H.3})$$

After expanding Equation H.3 with Equations H.1 and H.2, the term in z^{m+1} is, by inspection:

$$h_{m+1,m+1} h_{m+1,0} + g_{m+1,m+1} g_{m+1,0} = 0 \quad (\text{H.4})$$

If $h_{m+1,m+1} = 0$ then either $g_{m+1,m+1} = 0$, in which case the current order reduction step is not necessary, or $g_{m+1,0} = 0$ in which case:

$$\begin{aligned} H_m(z) &= H_{m+1}(z) \\ G_m(z) &= zG_{m+1}(z) \end{aligned}$$

For the general case, Vaidyanathan derives the following order reduction of $A_{m+1}(z)$ to $A_m(z)$:

$$\begin{bmatrix} H_m(z) \\ G_m(z) \end{bmatrix} = \begin{bmatrix} k_{m+1} & \hat{k}_{m+1} \\ -\hat{k}_{m+1}z & k_{m+1}z \end{bmatrix} \begin{bmatrix} H_{m+1}(z) \\ G_{m+1}(z) \end{bmatrix}$$

where [70, Equations 13 and 14]:

$$k_{m+1} = \frac{-g_{m+1,m+1}}{\sqrt{h_{m+1,m+1}^2 + g_{m+1,m+1}^2}}, \quad \hat{k}_{m+1} = \frac{h_{m+1,m+1}}{\sqrt{h_{m+1,m+1}^2 + g_{m+1,m+1}^2}} \quad (\text{H.5})$$

or [70, Equation 23]:

$$k_{m+1} = \frac{h_{m+1,0}}{\sqrt{h_{m+1,0}^2 + g_{m+1,0}^2}}, \hat{k}_{m+1} = \frac{g_{m+1,0}}{\sqrt{h_{m+1,0}^2 + g_{m+1,0}^2}} \quad (\text{H.6})$$

The final order reduction step is:

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} k_0 & \hat{k}_0 \\ -\hat{k}_0 z & k_0 z \end{bmatrix} \begin{bmatrix} H_0(z) \\ G_0(z) \end{bmatrix}$$

The $m+1$ 'th lattice filter section is related to the m 'th section by:

$$A_{m+1}(z) = \tau_{m+1}(z) A_m(z)$$

where

$$\tau_{m+1}(z) = \begin{bmatrix} k_{m+1} & -\hat{k}_{m+1} z^{-1} \\ \hat{k}_{m+1} & k_{m+1} z^{-1} \end{bmatrix}$$

$A_m(z)$ also represents an allpass filter since $\tilde{\tau}_{m+1}(z) \tau_{m+1}(z) = I$ and:

$$\begin{aligned} I &= \tilde{A}_{m+1}(z) A_{m+1}(z) \\ &= \tilde{A}_m(z) \tilde{\tau}_{m+1}(z) \tau_{m+1}(z) A_m(z) \\ &= \tilde{A}_m(z) A_m(z) \end{aligned}$$

The Octave function *complementaryFIRdecomp.m* implements FIR lattice decomposition with filter order reduction by Equation H.6. (I found that Equation H.6 has better numerical performance than Equation H.5).

H.2 Finite-wordlength properties of the lattice FIR filter

Section H.3 shows that transfer function of the complementary FIR lattice is the result of N orthogonal transformations:

$$\begin{bmatrix} H_N & G_N \end{bmatrix}^T = \tau_N \cdots \tau_1 \begin{bmatrix} H_0 & G_0 \end{bmatrix}^T$$

where the τ_m are orthogonal matrixes:

$$\tau_m(z) = \begin{bmatrix} k_m & -\hat{k}_m z^{-1} \\ \hat{k}_m & k_m z^{-1} \end{bmatrix}$$

Vaidyanathan [70][Section VII] shows that this determines the finite-wordlength properties of the lattice FIR filter:

1. The noise gain from the inputs to the combined complementary outputs of the filter is N
2. The multiplier inputs are scaled since if $H_0^2 + G_0^2 = 1$ then $H_m^2 + G_m^2 = 1$. (Section 5.3.3 describes state scaling).
3. The coefficient sensitivities to k_m are:

$$\frac{\partial}{\partial k_m} \begin{bmatrix} H_N & G_N \end{bmatrix}^T = \tau_N \cdots \tau_{m+1} \begin{bmatrix} 1 & 0 \\ 0 & z^{-1} \end{bmatrix} \tau_{m-1} \cdots \tau_1 \begin{bmatrix} H_0 & G_0 \end{bmatrix}^T$$

so the coefficient sensitivities of the FIR lattice are bounded:

$$\left| \frac{\partial H_N}{\partial k_m} \right|^2 + \left| \frac{\partial G_N}{\partial k_m} \right|^2 = 1$$

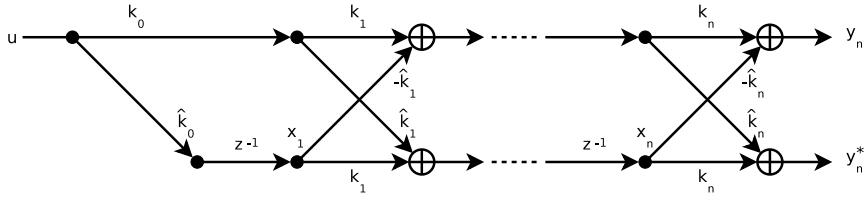


Figure H.1: Structure of the complementary FIR lattice filter (see Vaidyanathan [70, Figure 3])

Algorithm 31 Construction of a state variable description of the complementary FIR lattice filter

```

 $y_0^* = \hat{k}_0 u$ 
 $y_0 = k_0 u$ 
for  $n = 1, \dots, N$  do
     $x'_n = y_{n-1}^*$ 
     $y_n^* = \hat{k}_n y_{n-1} + k_n x_n$ 
     $y_n = k_n y_{n-1} - \hat{k}_n x_n$ 
end for
 $y = y_N$ 

```

H.3 State variable description of the complementary FIR lattice filter

Figure H.1 (see Figure 3 of Vaidyanathan [70]) shows the complementary FIR lattice structure.

For convenience, call x'_n the input to state x_n of the n -th section, y_n^* the lower output of the n -th section and y_n the upper output of the n -th section. Construction of the state variable description of the complementary FIR lattice is summarised in Algorithm 31.

As shown in Section 4.1, the state variable description can be expressed as a series of matrix multiplications linking the input and state outputs to the output and the next state inputs.

$$\begin{bmatrix} y_0^* \\ y_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 & \cdots & \cdots & 0 & \hat{k}_0 \\ 0 & \cdots & \cdots & 0 & k_0 \\ 1 & \cdots & \cdots & 0 & 0 \\ \vdots & \ddots & & \vdots & \\ \vdots & & \ddots & \vdots & \\ 0 & \cdots & & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ y_1^* \\ y_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \hat{k}_1 & k_1 & 0 & \cdots & 0 \\ 0 & k_1 & -\hat{k}_1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots & \\ 0 & \cdots & & \cdots & 1 & \end{bmatrix} \begin{bmatrix} y_0^* \\ y_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ y_2^* \\ y_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \hat{k}_2 & k_2 & 0 & \cdots & 0 \\ 0 & 0 & k_2 & -\hat{k}_2 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \ddots & \vdots & \\ 0 & \cdots & & \cdots & 1 & & \end{bmatrix} \begin{bmatrix} x'_1 \\ y_1^* \\ y_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \vdots \\ x'_{N-1} \\ y_{N-1}^* \\ y_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & \cdots & & \cdots & 0 \\ \vdots & \ddots & & & \vdots \\ 0 & \cdots & 1 & 0 & 0 & 0 \\ 0 & \cdots & 0 & \hat{k}_{N-1} & k_{N-1} & 0 \\ 0 & \cdots & 0 & k_{N-1} & -\hat{k}_{N-1} & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ \vdots \\ x'_{N-2} \\ y_{N-2}^* \\ y_{N-2} \\ x_{N-1} \\ x_N \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \vdots \\ x'_N \\ y_N^* \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & \cdots & \cdots & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & 1 & 0 & 0 \\ 0 & \cdots & 0 & \hat{k}_N & k_N \\ 0 & \cdots & 0 & k_N & -\hat{k}_N \end{bmatrix} \begin{bmatrix} x'_1 \\ \vdots \\ y_{N-1}^* \\ y_{N-1} \\ x_N \end{bmatrix}$$

The Octave function `complementaryFIRlattice2Abcd` returns the state variable description of a complementary FIR lattice filter.

H.4 Design of the complementary FIR digital filter

The previous sections of this chapter assume that the complementary FIR filter is known. This section shows two methods for finding that filter. The first, *Orchard and Willson's* Newton-Raphson solution [33], is simpler and more accurate than the cepstral method of *Mian and Nainer* [27].

H.4.1 *Orchard and Willson's* Newton-Raphson solution

If an order N FIR filter $H(z)$ is bounded-real then $|H(e^{j\omega})| \leq 1$. The magnitude-squared complementary filter $1 - H^\dagger(z)H(z)$, where \dagger means *conjugate*, is linear-phase and has double zeros on the unit-circle when $|H(e^{j\omega})| = 1$. *Orchard and Willson* [33] find the minimum-phase spectral factor of the magnitude-squared complementary filter by a Newton-Raphson solution of the non-linear system of equations linking the coefficients of the two filters. They demonstrate that the Newton-Raphson recursion converges linearly when the linear phase filter being factored has zeros on the unit circle. *Orchard and Willson* show a MATLAB (ie: Octave) implementation, `minphase.m`, in Appendix A of the reference.

Orchard and Willson justify their method with the following example. Firstly, they define a short linear phase FIR filter:

$$H(z) = z^{-3} [h_0 + h_1(z + z^{-1}) + h_2(z^2 + z^{-2}) + h_3(z^3 + z^{-3})]$$

$H(z)$ is assumed to be the product of a minimum phase factor:

$$F_1(z) = a_0 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3}$$

and the corresponding maximum phase factor:

$$\begin{aligned} F_2(z) &= z^{-3}F_1(z^{-1}) \\ &= a_3 + a_2z^{-1} + a_1z^{-2} + a_0z^{-3} \end{aligned}$$

Equating the coefficients of terms on either side of the equality sign in $H(z) = F_1(z)F_2(z)$ gives the system of non-linear equations:

$$\begin{aligned} h_0 &= a_0^2 + a_1^2 + a_2^2 + a_3^2 \\ h_1 &= a_0a_1 + a_1a_2 + a_2a_3 \\ h_2 &= a_1a_3 + a_0a_2 \\ h_3 &= a_0a_3 \end{aligned}$$

The Newton-Raphson method solves a system of equations $f(\mathbf{a}) = 0$ by successive approximations:

$$f(\mathbf{a}_k) + J(\mathbf{a}_k)[\mathbf{a}_{k+1} - \mathbf{a}_k] = 0$$

or:

$$\mathbf{a}_{k+1} = \mathbf{a}_k - J^{-1}(\mathbf{a}_k)f(\mathbf{a}_k)$$

where $J(\mathbf{a})$ is the Jacobian matrix of $f(\mathbf{a})$.

In this case:

$$f(\mathbf{a}) = \begin{bmatrix} h_0 - a_0^2 - a_1^2 - a_2^2 - a_3^2 \\ h_1 - a_0a_1 - a_1a_2 - a_2a_3 \\ h_2 - a_1a_3 - a_0a_2 \\ h_3 - a_0a_3 \end{bmatrix}$$

$$J(\mathbf{a}) = - \begin{bmatrix} 2a_0 & 2a_1 & 2a_2 & 2a_3 \\ a_1 & a_2 + a_0 & a_3 + a_1 & a_2 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & 0 & 0 & a_0 \end{bmatrix}$$

$$= - \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 & 0 \\ a_2 & a_3 & 0 & 0 \\ a_3 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ 0 & a_0 & a_1 & a_2 \\ 0 & 0 & a_0 & a_1 \\ 0 & 0 & 0 & a_0 \end{bmatrix}$$

The function *minphase.m* solves for the coefficient update, $\mathbf{d}_{k+1} = \mathbf{a}_{k+1} - \mathbf{a}_k$, by matrix left division. The C++ file *minphase.cc* implements the *minphase* algorithm as an *oct-file* using the *Eigen* [31] C++ template library with *long double* matrix elements.

I experimented with the Octave-Forge *optim* package *leasqr* function, an implementation of Levenberg-Marquardt non-linear optimisation. I found that *leasqr* needed to be initialised with the *minphase* solution and did not improve that solution by more than a few machine *epsilon*.

H.4.2 Mian and Nainer's cepstral method

Mian and Nainer [27] describe finding the minimum-phase spectral factor of the magnitude-squared complementary response by calculating the cepstrum of that response along a z-plane contour slightly off the unit circle. This contour reduces aliasing of the cepstrum due to the response zeros that lie on the unit circle. See Appendix A for a review of the complex variables theory used in this section.

Properties of the cepstrum Firstly, recall that the *z*-transform of a real sequence, \mathbf{x} , is

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n}$$

and that the inverse *z*-transform is

$$x(n) = \frac{1}{2\pi i} \oint_C X(z) z^{n-1} dz$$

where the contour, C , is centred on the origin and lies within the region of convergence. Cauchy's integral lemma is

$$\oint_C z^n dz = \begin{cases} 2\pi i & n = -1 \\ 0 & n \neq -1, \text{ integral} \end{cases}$$

The *complex cepstrum* is the inverse *z*-transform of $\hat{X}(z) = \log X(z) = \log |X(z)| + i \arg X(z)$ on the unit circle:

$$\hat{x}(n) = \frac{1}{2\pi i} \oint_C \log X(z) z^{n-1} dz$$

The *real cepstrum* is the inverse *z*-transform of $\log |X(z)|$ on the unit circle.

For example, suppose $X(z) = K(1 - az^{-1})(1 - bz)$, where $|a|, |b| < 1$. By Cauchy's integral lemma and the definition of the *z*-transform, $K = X(0) = x(0)$. The complex cepstrum is:

$$\hat{x}(n) = \frac{1}{2\pi i} \oint_C [\log K + \log(1 - az^{-1}) + \log(1 - bz)] z^{n-1} dz$$

Applying Cauchy's integral lemma and the series expansion $\log(1 + z) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{z^k}{k}$, where $|z| < 1$:

$$\hat{x}(n) = \begin{cases} -\frac{a^n}{n}, & n > 0 \\ \log K, & n = 0 \\ \frac{b^n}{n}, & n < 0 \end{cases}$$

This example demonstrates the motivation for using the cepstrum to find a minimum-phase spectral factor: if \mathbf{x} is minimum-phase then the cepstrum, $\hat{\mathbf{x}}$, is causal. Similarly, if \mathbf{x} is maximum-phase then the cepstrum is anti-causal.

The complex cepstrum is related to the original sequence, x , by a recursion. In the z-domain:

$$\begin{aligned}\frac{d}{dz} \hat{X}(z) &= \frac{1}{X(z)} \frac{d}{dz} X(z) \\ \left[-z \frac{d}{dz} \hat{X}(z) \right] X(z) &= -z \frac{d}{dz} X(z)\end{aligned}$$

So, in the time domain:

$$n\hat{x}(n) * x(n) = nx(n)$$

In other words:

$$x(n) = \begin{cases} e^{\hat{x}(0)} & n = 0 \\ \sum_{k=-\infty}^{\infty} \frac{k}{n} \hat{x}(n) x(n-k) & n \neq 0 \end{cases} \quad (\text{H.7})$$

Using the cepstrum to find the minimum-phase spectral factor Given a bounded-real transfer function $H(z)$ of order N , the z-transform of the magnitude-squared complementary filter is:

$$F(z) = 1 - |H(z)|^2 = 1 - H^\dagger(z) H(z)$$

where \dagger means *conjugate*. By construction, $F(z)$ is real, even, $0 \leq F(e^{j\omega}) \leq 1$ and any zeros of $F(z)$ on the unit circle are double zeros. If $F(z)$ has zeros on the unit circle, then $\hat{F}(z) = \log F(z)$ is not defined at those zeros. However we can make use of the z-transform

$$X(\alpha z) = \sum_{n=-\infty}^{\infty} \alpha^{-n} x(n) z^{-n}$$

to calculate the cepstrum over a contour that is slightly outside the unit circle. In this case, $F(\alpha z)$ is real and even so that, by construction, the cepstrum, $\hat{f}_\alpha(n)$, is real and even. Therefore the causal part of the cepstrum is

$$\hat{h}_\alpha(n) = \begin{cases} \frac{\hat{f}_\alpha(n)}{2} & n = 0 \\ \hat{f}_\alpha(n) & n > 0 \\ 0 & n < 0 \end{cases}$$

and the impulse response of the minimum-phase spectral factor is $h(n) = \alpha^n h_\alpha(n)$ where $h_\alpha(n)$ is found from the recursion given above in Equation H.7. Unfortunately this method fails if $F(\alpha z)$ is not positive.

A simple example Here is *Octave* code for a simple example recovering the minimum-phase spectral factor from a magnitude-squared filter with double zeros on the unit circle¹:

```
% Construct a minimum phase example with zeros on the unit circle
a1=[1 -1 0.5];a2=[1 -0.5];a3=[1 0 0.81];a4=[1 -1];a5=[1 -sqrt(2) 1];
a=conv(conv(conv(conv(a1,a2),a3),a4),a5);
% Construct the squared-magnitude filter for a contour of |z|=alpha
Na=length(a)-1;
alpha=1.05;
aalpha=a.* (alpha.^[0:-1:-Na]);
asq=conv(fliplr(aalpha),aalpha);
% The zero-phase squared-magnitude frequency response is real, positive and even
Hasq=freqz(asq,[zeros(1,Na) 1],4096,"whole");
Hasq=real(Hasq);
% The cepstrum is real and even
hhatasq=iifft(log(Hasq));
hhatasq=real(hhatasq(:)');
% Use the causal part of the cepstrum
ha=zeros(1,Na+1);
ha(1)=exp(hhatasq(1)/2);
for n=2:(Na+1)
    ha(n)=sum([1:(n-1)].*hhatasq(2:n).*fliplr(ha(1:(n-1))))/(n-1);
endfor
% Recover the original minimum-phase impulse response
h=ha.* (alpha.^[0:Na]);
```

¹Note that the Octave convention is that the polynomial $[a \ b \ c]$ corresponds to the z-transform $a + bz^{-1} + cz^{-2}$ so the zero-phase squared-magnitude response calculated by *freqz* must be scaled by a denominator polynomial z^{-N} .

H.4.3 Example: the minimum-phase complementary filter of an FIR bandpass filter

The Octave script *minphase_test.m* compares the use of *Orchard and Willson's* Newton-Raphson method with the cepstral method of *Mian and Nainer*. The script uses both methods to find the minimum-phase complementary filter for an FIR bandpass filter designed with the Octave *remez* function. The combined response of the bandpass filter and the complementary filter found by the Newton-Raphson method is allpass to within an order of magnitude of the machine precision ($\text{eps} = 2.2204e - 16$). The bandpass and complementary filter magnitude responses are shown in Figure H.2.

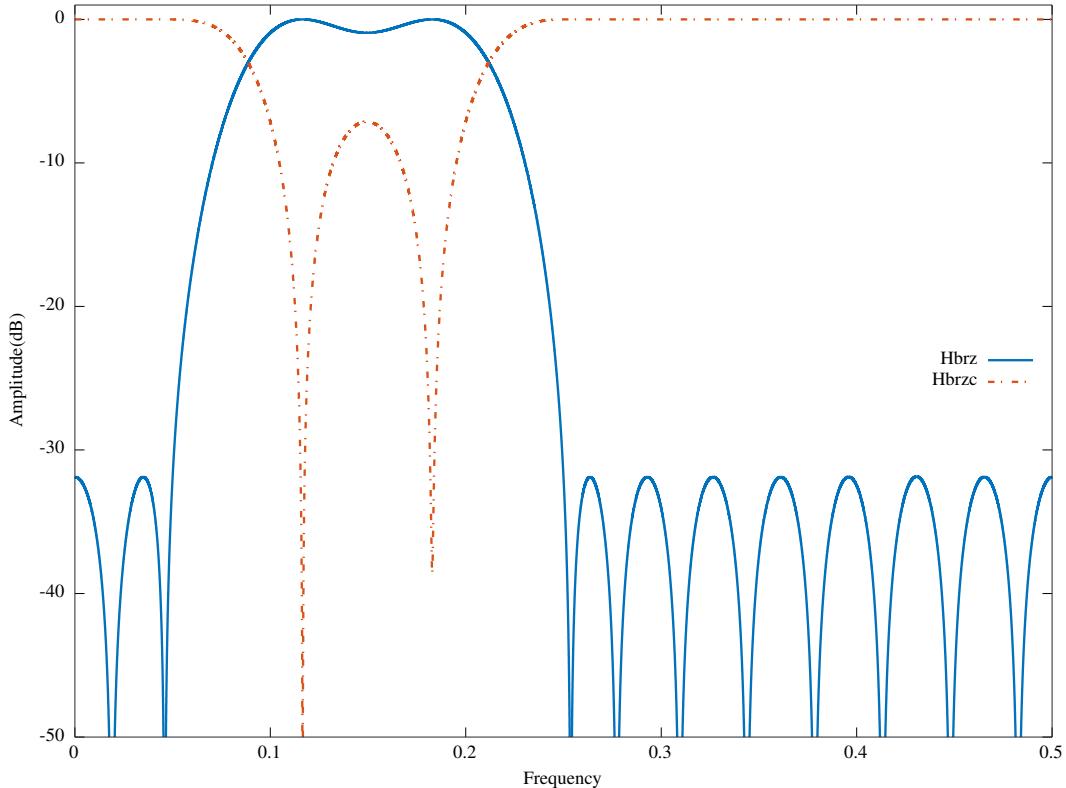


Figure H.2: Amplitude responses of the FIR band-pass filter and the minimum-phase complementary filter found with *Orchard and Willson's* Newton-Raphson method

Unfortunately, in the case of the cepstral method, $F(\alpha z)$ with $\alpha \approx 1.15$ is real and even but not positive. Fortunately, when using $\alpha = 1$ with a long FFT the sampling grid did not include the filter zeros. Response aliasing due to the discontinuities in $\log F(\alpha z)$ gave a result that was less accurate than that obtained with *minphase.m* but was still acceptable. The combined amplitude response found with the cepstral method is shown in Figure H.3.

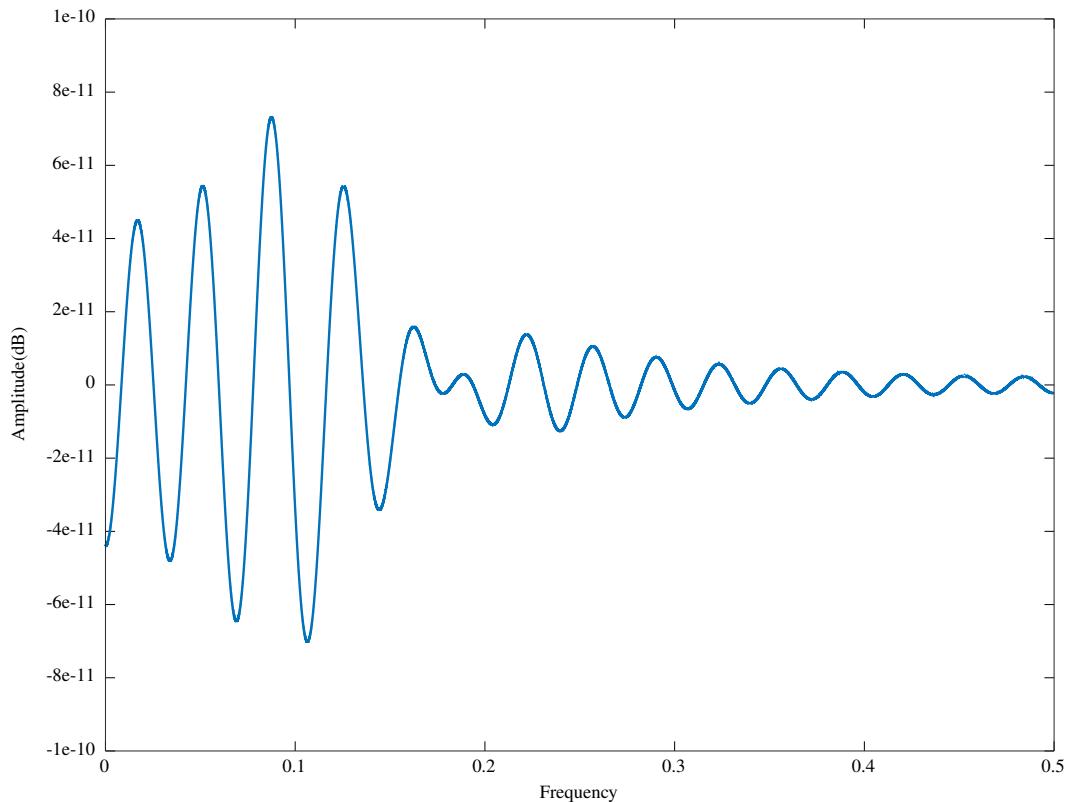


Figure H.3: Combined amplitude responses of the FIR band-pass filter and the minimum-phase complementary filter found with *Mian and Nainer's* cepstral method

Appendix I

Review of *Lanczos* tridiagonalisation of an unsymmetric matrix

See *Golub and Van Loan* [26, Section 9.4.3]. The *Lanczos* tridiagonalisation calculates an orthogonal similarity transform, T , that reduces a matrix $A \in \mathbb{R}^{N \times N}$ to tridiagonal form, \mathcal{A} :

$$T^{-1}AT = \mathcal{A} = \begin{bmatrix} \alpha_0 & \beta_1 & 0 & 0 & \cdots & 0 \\ \gamma_1 & \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ 0 & \gamma_2 & \alpha_2 & \beta_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \gamma_{N-2} & \alpha_{N-2} & \beta_{N-1} \\ 0 & 0 & \cdots & 0 & \gamma_{N-1} & \alpha_{N-1} \end{bmatrix}$$

The *Lanczos* tridiagonalisation calculates the column partitionings:

$$\begin{aligned} T &= [t_1 \ \cdots \ t_N] \\ T^{-T} &= P = [p_1 \ \cdots \ p_N] \end{aligned}$$

Comparing columns in $AT = T\mathcal{A}$ and $A^T P = P\mathcal{A}^T$, for $k \in [1, N - 1]$:

$$\begin{aligned} At_k &= \beta_{k-1}t_{k-1} + \alpha_{k-1}t_k + \gamma_k t_{k+1} && \text{with } \beta_0 t_0 \equiv 0 \\ A^T p_k &= \gamma_{k-1}p_{k-1} + \alpha_{k-1}p_k + \beta_k p_{k+1} && \text{with } \gamma_0 p_0 \equiv 0 \end{aligned}$$

In addition, since T is orthogonal and $P^T T = I_{N \times N}$:

$$\begin{aligned} \alpha_{k-1} &= p_k^T At_k \\ \beta_k p_{k+1} &\equiv r_k = (A - \alpha_{k-1}I)^T p_k - \gamma_{k-1}p_{k-1} \\ \gamma_k t_{k+1} &\equiv s_k = (A - \alpha_{k-1}I) t_k - \beta_{k-1}t_{k-1} \end{aligned}$$

Note that:

$$1 = p_{k+1}^T t_{k+1} = \left(\frac{r_k}{\beta_k} \right)^T \left(\frac{s_k}{\gamma_k} \right) \quad (\text{I.1})$$

and so if γ_k is specified in advance:

$$\beta_k = \frac{r_k^T s_k}{\gamma_k} \quad (\text{I.2})$$

A common choice is $\gamma_k = \|s_k\|_2$ ¹. Algorithm 32 calculates the *Lanczos tridiagonalisation* of an unsymmetric matrix.

¹Recall that $\|s_k\|_2 = (s_k^T s_k)^{\frac{1}{2}}$.

Algorithm 32 Lanczos tridiagonalisation of an unsymmetric matrix

Convert the matrix A to *tri-diagonal* form:

r_0 and s_0 are given unit 2-norm vectors with $r_0^T s_0 \neq 0$

$t_0 = 0$

$p_0 = 0$.

$k = 0$

while $r_k \neq 0 \wedge s_k \neq 0 \wedge r_k^T s_k \neq 0$ **do**

$$\gamma_k = \|s_k\|_2$$

$$\beta_k = \frac{r_k^T s_k}{\gamma_k}$$

$$t_{k+1} = \frac{\gamma_k}{\beta_k}$$

$$p_{k+1} = \frac{\gamma_k}{\beta_k}$$

$$k = k + 1$$

$$\alpha_{k-1} = p_k^T A t_k$$

$$r_k = (A - \alpha_{k-1} I)^T p_k - \gamma_{k-1} p_{k-1}$$

$$s_k = (A - \alpha_{k-1} I) t_k - \beta_{k-1} t_{k-1}$$

end while

If

$$\mathcal{A}_k = \begin{bmatrix} \alpha_0 & \beta_1 & 0 & 0 & \cdots & 0 \\ \gamma_1 & \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ 0 & \gamma_2 & \alpha_2 & \beta_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \gamma_{k-2} & \alpha_{k-2} & \beta_{k-1} \\ 0 & 0 & \cdots & 0 & \gamma_{k-1} & \alpha_{k-1} \end{bmatrix}$$

then, when the loop of Algorithm 32 terminates:

$$A^T \begin{bmatrix} p_1 & \cdots & p_k \end{bmatrix} = \begin{bmatrix} p_1 & \cdots & p_k \end{bmatrix} \mathcal{A}_k^T + r_k e_k^T$$

$$A \begin{bmatrix} t_1 & \cdots & t_k \end{bmatrix} = \begin{bmatrix} t_1 & \cdots & t_k \end{bmatrix} \mathcal{A}_k + s_k e_k^T$$

If $r_k = 0$ when the iteration terminates then $\text{span} \{ p_1 \dots p_k \}$ is a subspace for A . If $s_k = 0$ when the iteration terminates then $\text{span} \{ t_1 \dots t_k \}$ is a subspace for A . If neither of these conditions is true and $r_k^T s_k = 0$, then the tridiagonalisation process has failed. *Golub and Van Loan* [26, Section 9.4.4] give a brief review of the *Look-Ahead* technique for solving this problem. The Octave function *lanczos_tridiag* implements Algorithm 32.

Appendix J

Review of Butterworth second order sections

This section reviews the Butterworth filter transfer function. This material is covered in many textbooks (for example, *Roberts and Mullis* [76, Chapter 6]). I have included it here to justify the implementation in *butter2pq.m*.

J.1 Continuous Time Second Order Butterworth Filter Prototypes

In the continuous time *s-plane* a low-pass Butterworth filter has, by definition, a squared magnitude frequency response of:

$$\left| \hat{H}(\omega) \right|^2 = \frac{1}{1 + \omega^{2n}}$$

The response with a cutoff angular frequency of ω_c is found by the s-plane transformation $s \rightarrow \frac{s}{\omega_c}$. The high-pass response is found by the s-plane transformation $s \rightarrow \frac{1}{s}$.

The $2n$ poles of the Butterworth squared magnitude response are evenly spaced around the unit circle in the s-plane. For stability, the filter realization as a cascade of second order sections uses the poles in the left-hand half plane, λ_k :

$$\begin{aligned} \lambda_k &= \omega_c e^{j\theta_k} \\ \theta_k &= \frac{\pi}{2} \left(1 + \frac{(2k-1)}{n} \right), \quad 1 \leq k \leq n \end{aligned}$$

For each conjugate pole pair the corresponding low-pass filter second-order section with unity DC gain has transfer function¹:

$$\begin{aligned} \hat{H}_k(s) &= \frac{\lambda_k \lambda_k^\dagger}{(s - \lambda_k)(s - \lambda_k^\dagger)} \\ &= \frac{\lambda_k \lambda_k^\dagger}{s^2 - (\lambda_k + \lambda_k^\dagger)s + \lambda_k \lambda_k^\dagger} \\ &= \frac{\omega_c^2}{s^2 - 2\omega_c \cos \theta_k s + \omega_c^2} \end{aligned}$$

If n is odd, there is a single real pole at $s = -\omega_c$, $k = \frac{n+1}{2}$, and the corresponding low-pass first-order section is:

$$\hat{H}_k(s) = \frac{\omega_c}{s + \omega_c}$$

The Butterworth high-pass filter with cut-off frequency ω_c is found with the s-plane transformation $\frac{s}{\omega_c} \rightarrow \frac{\omega_c}{s}$. The second order high-pass sections are:

$$\hat{H}_k(s) = \frac{s^2}{s^2 - 2\omega_c \cos \theta_k s + \omega_c^2}$$

and, if n is odd, the first order high-pass section is:

$$\hat{H}_k(s) = \frac{s}{\omega_c + s}$$

¹† denotes complex conjugate

J.2 Design of discrete time filters with the bilinear transform

Various methods can be used to transform the analog prototype response to the discrete time domain. The “bi-linear” transform is:

$$H(z) = \hat{H} \left(\frac{1 - z^{-1}}{1 + z^{-1}} \right)$$

The bi-linear transform has the following properties:

- the s-plane ω axis maps to the z-plane unit circle $z = e^{i\Omega T}$, where T is the sampling interval, and the s-plane left-half plane maps to the z-plane unit disc, $|z| \leq 1$.
- if the analog response is stable, the transformed response is also stable
- the zero frequency response is the same in both domains

The bilinear transform, $H(z)$ of the analog frequency response, $\hat{H}(s)$ is

$$H(e^{i\Omega T}) = \hat{H} \left(i \tan \frac{\Omega T}{2} \right)$$

The design frequencies of the analog prototype filter on the s-plane ω axis must be “pre-warped” to the z-plane unit circle by the mapping $\tan \frac{\Omega T}{2} \rightarrow \omega$. In other words, the discrete-time filter behaves at frequency Ω in the same way that the continuous-time filter behaves at frequency $\omega = \tan \frac{\Omega T}{2}$. For example, if the desired cut-off frequency in the z-domain is $\Omega_c T = \frac{\pi}{2}$, then we choose the cutoff frequency of the s-plane prototype filter to be:

$$\omega_c = \tan \frac{\Omega_c T}{2} = \tan \frac{\pi}{4} = 1$$

In this work I have assumed that the sampling interval, T , is 1. Alternatively, one can choose that the *Nyquist* frequency corresponds to 1 so that the sampling frequency is 2. The Octave *signal* toolbox filter design functions use the latter convention.

Filter design using analog prototypes proceeds as follows:

- the critical frequencies in the z-plane are determined
- the critical frequencies are mapped to the s-plane and used to design the s-plane prototype
- the bi-linear transform is used to convert that prototype to the z-plane

J.3 Design of discrete time second order Butterworth filters with the bilinear transform

For the Butterworth first order low-pass section:

$$\begin{aligned} H_{\frac{n+1}{2}}(z) &= \frac{\omega_c (1 + z^{-1})}{(\omega_c + 1) + (\omega_c - 1) z^{-1}} \\ &= d + \frac{q z^{-1}}{1 + p z^{-1}} \end{aligned}$$

where

$$\begin{aligned} d &= \frac{\omega_c}{\omega_c + 1} \\ q &= \frac{2\omega_c}{(\omega_c + 1)^2} \\ p &= \frac{\omega_c - 1}{\omega_c + 1} \end{aligned}$$

For the Butterworth second order low-pass section:

$$\begin{aligned} H_k(z) &= \frac{g [1 + z^{-1}]^2}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \\ &= d_0 + \frac{q_1 z^{-1} + q_2 z^{-2}}{1 + p_1 z^{-1} + p_2 z^{-2}} \end{aligned}$$

where:

$$\begin{aligned} g &= \omega_c^2 \\ a_0 &= 1 - 2\omega_c \cos \theta_k + \omega_c^2 \\ a_1 &= 2(\omega_c^2 - 1) \\ a_2 &= 1 + 2\omega_c \cos \theta_k + \omega_c^2 \\ d_0 &= \frac{g}{a_0} \\ p_1 &= \frac{a_1}{a_0} \\ p_2 &= \frac{a_2}{a_0} \\ q_1 &= d_0(2 - p_1) \\ q_2 &= d_0(1 - p_2) \end{aligned}$$

Similarly, for the first order high-pass section:

$$\begin{aligned} H_{\frac{n+1}{2}}(z) &= \frac{(1 - z^{-1})}{(\omega_c + 1) + (\omega_c - 1)z^{-1}} \\ &= \frac{d}{w_c} - \frac{qz^{-1}}{1 + pz^{-1}} \end{aligned}$$

and for the Butterworth second order high-pass section:

$$\begin{aligned} H_k(z) &= \frac{[1 - z^{-1}]^2}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \\ &= t_0 + \frac{t_1 z^{-1} + t_2 z^{-2}}{1 + p_1 z^{-1} + p_2 z^{-2}} \end{aligned}$$

where

$$\begin{aligned} t_0 &= \frac{d_0}{g} \\ t_1 &= -\frac{d_0}{g}(2 + p_1) \\ t_2 &= \frac{q_2}{g} \end{aligned}$$

Appendix K

Fourier transform of the Gaussian function

See *NIST Digital Library of Mathematical Functions* [22, Sections 7.2.1 and 1.14.1]

K.1 Preliminary results

The Gaussian function is e^{-t^2} .

K.1.1 Integral of the Gaussian function

Poisson introduced this method of integrating the Gaussian function:

$$\begin{aligned} \left[\int_{-\infty}^{\infty} e^{-t^2} dt \right]^2 &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-x^2-y^2} dx dy \\ &= \int_0^{2\pi} \int_0^{\infty} e^{-r^2} r dr d\theta \\ &= 2\pi \int_0^{\infty} e^{-r^2} r dr \\ &= \pi \int_0^{\infty} e^{-s} ds \\ &= \pi [-e^{-s}]_0^{\infty} \\ &= \pi \end{aligned}$$

So:

$$\int_0^{\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{2}$$

K.1.2 Fourier transform of the derivative of a function

If $g(t)$ is absolutely integrable, bounded and differentiable on $(-\infty, \infty)$ then the Fourier transform, \mathcal{F} , of $g(t)$ is:

$$G(\omega) = \mathcal{F}g(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(t) e^{i\omega t} dt$$

and the inverse Fourier transform is:

$$g(t) = \mathcal{F}^{-1}G(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} G(\omega) e^{-i\omega t} d\omega$$

The Fourier transform of $\frac{dg(t)}{dt}$ is:

$$\begin{aligned}\mathcal{F} \frac{dg(t)}{dt} &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \frac{dg(t)}{dt} e^{i\omega t} dt \\ &= \frac{1}{\sqrt{2\pi}} \left\{ [g(t) e^{i\omega t}]_{-\infty}^{\infty} - i\omega \int_{-\infty}^{\infty} g(t) e^{i\omega t} dt \right\} \\ &= -i\omega \mathcal{F}g(t)\end{aligned}$$

Similarly:

$$\begin{aligned}\mathcal{F}^{-1} \frac{dG(\omega)}{d\omega} &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \frac{dG(\omega)}{d\omega} e^{-i\omega t} d\omega \\ &= \frac{1}{\sqrt{2\pi}} \left\{ [G(\omega) e^{-i\omega t}]_{-\infty}^{\infty} + it \int_{-\infty}^{\infty} G(\omega) e^{-i\omega t} dt \right\} \\ &= it \mathcal{F}^{-1} G(\omega)\end{aligned}$$

K.2 Derivation of the Fourier transform of the Gaussian function in the frequency domain

The Gaussian function in the frequency domain is:

$$G(\omega) = e^{-(\frac{\omega}{\alpha})^2}$$

Differentiating both sides:

$$\frac{dG(\omega)}{d\omega} = -\frac{2\omega}{\alpha^2} G(\omega)$$

Taking the Fourier transform of both sides:

$$itg(t) = -\frac{2i}{\alpha^2} \frac{dg(t)}{dt}$$

Rearranging and integrating both sides:

$$\begin{aligned}\int_0^t -\frac{\tau \alpha^2}{2} d\tau &= \int_0^t \frac{\frac{dg(\tau)}{d\tau}}{g(\tau)} d\tau \\ \ln g(t) - \ln g(0) &= -\left(\frac{t\alpha}{2}\right)^2 \\ g(t) &= g(0) e^{-\left(\frac{t\alpha}{2}\right)^2}\end{aligned}$$

where:

$$\begin{aligned}g(0) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-(\frac{\omega}{\alpha})^2} d\omega \\ &= \frac{2\alpha}{\sqrt{2\pi}} \int_0^{\infty} e^{-\omega^2} d\omega \\ &= \frac{\alpha}{\sqrt{2}}\end{aligned}$$

Colophon

Software requirements

This document is created with *GNU Make* and the *Makefile* in project archive. I assume:

- *octave-4.2.1* or later and the *struct*, *control*, *signal*, *optim* and *parallel* Octave-Forge packages
- *octave* is linked to the Netlib *lapack* and *blas* libraries either at build-time or using *LD_PRELOAD*
- *octave* uses the *gnuplot* graphics toolkit and *gnuplot-5.0* or later
- *texlive* and various packages are installed

I recommend building a local version of *octave-4.2.1* or later, as shown below in “*Building Octave with LTO and PGO*”. “*Testing with octave-4.0*”, shown below, lists the modifications required to run most scripts with the standard *octave-4.0.3* installed by Fedora25.

Aegis

I use the *aegis* software change management system written by Peter Miller [98, 73]. To build *aegis* with the patch provided in this project:

```
cd /usr/local/src/aegis
tar -xf aegis-4.24.tar.gz
cd aegis-4.24
patch -p1 < ../aegis-4.24.patch
./configure CXXFLAGS="-fpermissive -std=c++98" CFLAGS="-std=c99"
make && make install
```

To build *fhist* [74] with the patch provided in this project:

```
cd /usr/local/src/fhist
tar -xf fhist-1.21.D001.tar.gz
cd fhist-1.21
patch -p1 < ../fhist-1.21.D001.patch
./configure
make && make install
```

Makefile

This project is built with *GNU Make*. The *Makefile* in the project archive builds the PDF version of this document from the *LATEX* source in *DesignOfIIRFilters.tex*. The project archive contains the files required to generate the figures shown in this document. The diagrams are in *dia* format [29] in directory *fig*. The source code is in directory *src*. The source code languages are C++, Maxima and Octave. The *Makefile* includes a *.mk* file for each of the Octave test script dependencies listed in the variable *OCTAVE_SCRIPTS*. For example, the Octave script *butt3NS_test.m*, referred to in Section 7.7.1, has the corresponding *Makefile* fragment, *butt3NS_test.mk*:

```

butt3NS_test FIGURES=butt3NS_test_response \
butt3NS_test_expected_response \
butt3NS_test_response_direct_form_noise \
butt3NS_test_sv_noise_schur_lattice \
butt3NS_test_sv_noise_global_optimum \
butt3NS_test_sv_noise_direct_form \
butt3NS_test_sv_sine_schur_lattice

butt3NS_test_FILES = \
butt3NS_test.m test_common.m schurexpand.oct schurdecomp.oct \
schurNSscale.oct tf2schurNSlattice.m schurNSlatticeNoiseGain.m \
schurNSlattice2Abcd.oct schurNSlatticeFilter.m svf.m KW.m optKW.m \
tf2Abcd.m crossWelch.m

```

Lapack

The Octave-Forge *signal* package requires the *control* package which is based on the open-source version of the *SLICOT* library [24] which uses *lapack* routines that have been deprecated. Consequently, the *BUILD_DEPRECATED* flag must be set in *make.inc* when building *lapack*.

Building *lapack* shared libraries

My system has an Intel i7 CPU. The *lapack-3.6.1-2.fc25.x86_64* package is installed. The default *gcc* machine type used by the RedHat RPMs is given by *rpmbuild -showrc* as “*-m64 -mtune=generic*”. I could recompile and reinstall the *lapack* package with “*-mtune=intel*” as follows:

```

mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
echo '%_topdir %(%echo $HOME)/rpmbuild' > ~/.rpmmacros
echo 'optflags: x86_64 %{__global_cflags} -m64 -mtune=intel' > ~/.rpmrc
rpmbuild --rebuild lapack-3.6.1-2.fc25.src.rpm
sudo rpm -U --reinstall=~/rpmbuild/RPMS/RPMS/x86_64/lapack-3.6.1-2.fc25.x86_64.rpm

```

Alternatively, for the purposes of benchmarking with *LD_PRELOAD*, I can rebuild the shared library from the original *lapack-3.6.1.tgz* archive by adding the following lines to *lapack-3.6.1/make.inc.example*:

```

--- lapack-3.6.1/make.inc.example 2016-06-19 08:15:11.000000000 +1000
+++ lapack-3.6.1.new/make.inc.example 2017-05-04 15:01:44.316951216 +1000
@@ -16,17 +16,19 @@
# and handle these quantities appropriately. As a consequence, one
# should not compile LAPACK with flags such as -ffpe-trap=overflow.
#
-FORTRAN = gfortran
-OPTS = -O2 -frecursive
+BLDOPTS = -m64 -mtune=intel -fPIC
+FORTTRAN = gfortran -frecursive $(BLDOPTS)
+OPTS = -O2
DRVOPTS = $(OPTS)
-NOOPT = -O0 -frecursive
-LOADER = gfortran
+NOOPT = -O0
+LOADER = $(FORTTRAN)
LOADOPTS =
#
# Comment out the following line to include deprecated routines to the
# LAPACK library.
#
#MAKEDEPRECATED = Yes
+BUILD_DEPRECATED = Yes
#
# Timer for the SECOND and DSECND routines

```

```

#
@@ -52,7 +54,7 @@
 # CC is the C compiler, normally invoked with options CFLAGS.
 #
CC = gcc
-CFLAGS = -O3
+CFLAGS = -O3 $(BLDOPTS)
#
# The archiver and the flag(s) to use when building archive (library)
# If your system has no ranlib, set RANLIB = echo.
@@ -74,7 +76,7 @@
 # machine-specific, optimized BLAS library should be used whenever
 # possible.)
#
-BLASLIB      = ../../librefblas.a
+BLASLIB      = ../../libblas.a
CBLASLIB     = ../../libcblas.a
LAPACKLIB    = liblapack.a
TMGLIB        = libtmglib.a

```

and adding to *BLAS/SRC/Makefile*:

```
libblas.so: $(ALLOBJ)
$(FORTRAN) -shared -Wl,-soname,$@ -o $@ $(ALLOBJ)
```

and to *SRC/Makefile*:

```
liblapack.so: $(ALLOBJ) $(ALLXOBJ) $(DEPRECATED)
$(FORTRAN) -shared -Wl,-soname,$@ -o $@ $(ALLOBJ) $(ALLXOBJ) $(DEPRECATED)
```

Lapack static libraries

In order to take full advantage of LTO linking with static versions of *liblapack.a* and *libblas.a*, add the following to *make.inc*:

```
BLDOPTS = -m64 -mtune=intel -fPIC -flto=6 -ffat-lto-objects
```

Unfortunately, for *lapack-3.6.1*, the library tests fail with the *-frecursive* linker flag. Without this flag, the compiler allocates static memory and linking a static version of the *lapack* libraries with the Octave shared libraries fails.

Octave

The Octave version and list of installed packages used to prepare this document are:

```
computer=x86_64-pc-linux-gnu
kernel=octave version=4.2.1
Package Name | Version | Installation directory
-----+-----+-----
control *| 3.0.0 | /usr/local/share/octave/packages/control-3.0.0
optim *| 1.5.2 | /usr/local/share/octave/packages/optim-1.5.2
parallel | 3.1.1 | /usr/local/share/octave/packages/parallel-3.1.1
signal *| 1.3.2 | /usr/local/share/octave/packages/signal-1.3.2
struct *| 1.0.14 | /usr/local/share/octave/packages/struct-1.0.14
```

The section *Summary of important user-visible changes for version 4.0* in the *NEWS* file of the Octave source distribution includes the following comment:

Octave now automatically truncates intermediate calculations done with floating point values to 64 bits. Some hardware math co-processors, such as the x87, maintain extra precision, but this leads to disagreements in calculations when compared to reference implementations in software using the IEEE standard for double precision. There was no measurable performance impact to this change, but it may be disabled with the configure option `--disable-float-truncate`. MinGW and Cygwin platforms, as well as GCC compilers ≥ 5.0 require this feature. Non-x87 hardware, or hardware using SSE options exclusively, can disable float truncation if desired.

This project includes a number of Octave *oct-file* extensions written in C++. The Octave on-line FAQ states:

Code written using Octave's native plug-in interface (also known as a .oct file) necessarily links with Octave internals and is considered a derivative work of Octave and therefore must be released under terms that are compatible with the GPL.

Building Octave

Patching Octave

octave-4.2.1/liboctave/system/file-stat.cc must be altered as follows:

```
@@ -174,7 +174,7 @@
     update_internal ();
}

- inline file_stat::~file_stat () { }
+ file_stat::~file_stat () { }

void
file_stat::update_internal (bool force)
```

Cloning the Octave Mercurial repository

The latest development sources for Octave can be obtained as follows:

```
cd /usr/local/src/octave
hg clone http://www.octave.org/hg/octave
cd octave
hg -v pull
hg -v update
./bootstrap
```

Building Octave for debugging

```
export CFLAGS="-std=c11 -ggdb3 -O0"
export CXXFLAGS="-std=c++11 -ggdb3 -O0"
export FFLAGS="-ggdb3 -O0"
../$OCTAVE_DIR/configure --prefix=/usr/local/octave-dbg \
--disable-java --disable-atomic-refcount --disable-openmp --disable-threads \
--enable-bounds-check --with-blas="-lblas" --with-lapack="-llapack" \
--without-fltk --without-qt --without-opengl --without-OSMesa --without-magick \
--without-portaudio --without-sndfile
make -j 6 && make install
```

Don't use `--enable-address-sanitizer-flags` with *valgrind*.

To build and test an oct-file with *address-sanitizer*:

```
mkoctfile -g -O0 -fsanitize=address -fsanitize=undefined \
-fno-sanitize=vptr -fno-omit-frame-pointer an_oct_file.cc
LD_PRELOAD=/usr/lib64/libasan.so.3 octave-cli --eval 'an_oct_file_test_script'
```

Building Octave with LTO

To build Octave with static versions of the *lapack* libraries containing LTO symbol information:

```
BLDOPTS="-m64 -mtune=generic -O2 -fno-fat-lto-objects"
export CFLAGS="-std=c11 \"$BLDOPTS"
export CXXFLAGS="-std=c++11 \"$BLDOPTS
export FFLAGS=$BLDOPTS

$OCTAVE_DIR/configure --prefix=$OCTAVE_INSTALL_DIR \
--disable-java --without-fltk --without-qt --disable-atomic-refcount \
--with-blas=$LAPACK_DIR/libblas.a --with-lapack=$LAPACK_DIR/liblapack.a

make V=1 -j6
```

Building Octave with LTO and PGO

To build Octave with the system *lapack* and *blas* shared libraries and LTO and PGO optimisation:

```
BLDOPTS="-m64 -mtune=generic -O2 -fno-fat-lto-objects"
export CFLAGS="-std=c11 \"$BLDOPTS"
export CXXFLAGS="-std=c++11 \"$BLDOPTS
export FFLAGS=$BLDOPTS

../$OCTAVE_DIR/configure --prefix=$OCTAVE_INSTALL_DIR \
--disable-java --without-fltk --without-qt --disable-atomic-refcount \
--with-blas="-lblas" --with-lapack="-llapack"

make XTRA_CFLAGS="-fprofile-generate" XTRA_CXXFLAGS="-fprofile-generate" V=1 -j6
find . -name \*.gcda -exec rm -f {} ';' ;
make check
find . -name \*.o -exec rm -f {} ';' ;
find . -name \*.lo -exec rm -f {} ';' ;
find . -name \*.la -exec rm -f {} ';' ;
make XTRA_CFLAGS="-fprofile-use" XTRA_CXXFLAGS="-fprofile-use" V=1 -j6
```

The PGO profile information is generated by running the Octave test suite.

Building Octave with static *blas* and *lapack* libraries

To build Octave with static LTO versions of the reference NETLIB *blas* and *lapack* libraries:

```
BLDOPTS="-m64 -mtune=generic -O2 -fno-fat-lto-objects"
export CFLAGS="-std=c11 \"$BLDOPTS"
export CXXFLAGS="-std=c++11 \"$BLDOPTS
export FFLAGS=$BLDOPTS

$OCTAVE_DIR/configure --prefix=$OCTAVE_INSTALL_DIR \
--disable-java --without-fltk --without-qt --disable-atomic-refcount \
--with-blas=$LAPACK_DIR/libblas.a --with-lapack=$LAPACK_DIR/liblapack.a

make V=1 -j6
```

Configuring Octave with the ATLAS libraries

Alternatively, Octave can be built with the single-threaded *ATLAS* [10] libraries by:

```

BLDOPTS="-m64 -mtune=generic -O2 -fno-fat-lto-objects"
export CFLAGS="-std=c11 \"$BLDOPTS"
export CXXFLAGS="-std=c++11 \"$BLDOPTS
export FFLAGS=$BLDOPTS
export LDFLAGS="-L/usr/lib64/atlas"

./$OCTAVE_DIR/configure --prefix=/usr/local/octave-satlas-lto \
--disable-java --without-fltk --without-qt --disable-atomic-refcount \
--with-blas="-lsatlas" --with-lapack="-lsatlas"

make -j 6 && make check && make install

```

Similarly, for the multi-threaded *ATLAS* libraries call *configure* with:

```
--with-blas="-ltatlas" --with-lapack="-ltatlas"
```

Configuring Octave for 64-bit integers

This project uses the usual 32-bit integer version of Octave. To build a 64-bit integer version of Octave, first build and install the 64-bit versions of *arpack*, *glpk*, *lapack*, *qhull*, *qrupdate* and *SuiteSparse* in, for example, */opt/octave64*, then run *configure* with the *-enable-64* flag:

```

BLDOPTS="-O2 -m64 -mtune=generic -fno-fat-lto-objects -I/opt/octave64/include"
export CFLAGS=$BLDOPTS
export CXXFLAGS=$BLDOPTS
export FFLAGS=$BLDOPTS
export LDLAGS="-L/opt/octave64/lib"
export OCTAVE64_PREFIX=/opt/octave64/
export OCTAVE64_INCLUDE_DIR=$OCTAVE64_PREFIX/include/
export OCTAVE64_LIB_DIR=$OCTAVE64_PREFIX/lib/
export OCTAVE64_BIN_DIR=$OCTAVE64_PREFIX/bin/

./$OCTAVE_DIR/configure \
--prefix=$OCTAVE64_PREFIX \
--enable-64 \
--disable-java \
--without-fltk \
--without-qt \
--disable-atomic-refcount \
--with-blas=$OCTAVE64_LIB_DIR/libblas.a \
--with-lapack=$OCTAVE64_LIB_DIR/liblapack.a \
--with-qhull-includedir=$OCTAVE64_INCLUDE_DIR/libqhull" \
--with-qhull-libdir=$OCTAVE64_LIB_DIR \
--with-glpk-includedir=$OCTAVE64_INCLUDE_DIR \
--with-glpk-libdir=$OCTAVE64_LIB_DIR \
--with-qrupdate-libdir=$OCTAVE64_LIB_DIR \
--with-amd-includedir=$OCTAVE64_INCLUDE_DIR \
--with-amd-libdir=$OCTAVE64_LIB_DIR \
--with-camd-includedir=$OCTAVE64_INCLUDE_DIR \
--with-camd-libdir=$OCTAVE64_LIB_DIR \
--with-colamd-includedir=$OCTAVE64_INCLUDE_DIR \
--with-colamd-libdir=$OCTAVE64_LIB_DIR \
--with-ccolamd-includedir=$OCTAVE64_INCLUDE_DIR \
--with-ccolamd-libdir=$OCTAVE64_LIB_DIR \
--with-cholmod-includedir=$OCTAVE64_INCLUDE_DIR \
--with-cholmod-libdir=$OCTAVE64_LIB_DIR \
--with-umfpack-includedir=$OCTAVE64_INCLUDE_DIR \
--with-umfpack-libdir=$OCTAVE64_LIB_DIR \
--with-arpack-libdir=$OCTAVE64_LIB_DIR

make V=1 -j6

```

When compiling a 64-bit version of *SeDuMi*, *openblas/f77blas.h* requires *-DOPENBLAS_USE64BITINT*. All of the project Aegis test scripts passed with a 64-bit build of Octave.

Installing Octave-Forge packages

To install Octave packages from a remote Octave-Forge file-server:

```
octave-cli --eval 'pkg install -forge struct general miscellaneous optim control signal'
```

If *octave* was configured with *-disable-docs* then packages can be installed with this work-around:

```
/usr/local/octave-dbg/bin/octave-cli \
--eval 'texi_macros_file("/dev/null");pkg install package_file_name'
```

Benchmarking Octave

This section shows the results of benchmarking various builds with the Octave script *decimator_R2_test.m* and benchmarking various *blas* libraries with a *Link-Time-Optimisation(LTO)* and *Profile-Guided-Optimisation(PGO)* build of Octave running the Octave scripts *linpack.m* [59] (modified to produce repeatable residuals) and *decimator_R2_test.m*.

The Linux kernel used is *kernel-4.10.11-200.fc25.x86_64* from the RedHat Fedora25 Linux distribution. The Octave version is 4.2.1 built with *g++ version 6.3.1 20161221 (Red Hat 6.3.1-1)*. The CPU is an Intel i7-7700K for which the number of CPU cores is 4 (8 hyper-threaded). During the benchmark the CPU frequency was fixed at 4.5GHz. As the *root* user run:

```
for c in `seq 0 7`;do
  echo "4500000" > /sys/devices/system/cpu/cpu$c/cpufreq/scaling_min_freq;
  echo "performance" > /sys/devices/system/cpu/cpu$c/cpufreq/scaling_governor;
done
cpupower -c all frequency-info
```

Benchmarking Octave builds

The shell script *benchmark/build-benchmark.sh* generates the Octave builds shown in Table K.1. It is assumed to run in the *benchmark* sub-directory. The times given are the average of 10 runs. For the *debug* build the optimisation level is *-O0* and for the other builds it is *-O2 -m64 -mtune=generic*. PGO uses the profiler information generated by an intermediate *make check*. For the static LTO builds the executable is linked with local versions of the *lapack-3.6.1* libraries containing LTO symbols. For each case, the Octave build is *configured* for command-line only with the folowing options:

```
--disable-java --without-fltk --without-qt --disable-atomic-refcount
```

Benchmarking Octave *blas* libraries

The shell script *library-benchmark.sh* benchmarks the shared library LTO PGO version of Octave, built with the script *build-shared-lto-pgo.sh*, with the *linpack.m* [59] benchmark (modified to produce repeatable residuals) and *decimator_R2_test.m* filter design scripts using different versions of the *blas* shared library. It is assumed to run in the *benchmark* directory after *build-benchmark.sh*. The alternative *blas* and *lapack* library packages used in this section are:

atlas.x86_64	3.10.2-14.fc25
atlas-static.x86_64	3.10.2-14.fc25
gsl.x86_64	2.1-4.fc25
openblas.x86_64	0.2.19-4.fc25
openblas-threads.x86_64	0.2.19-4.fc25

The library to be used is specified by the *LD_PRELOAD* environment variable. For example:

```
LD_PRELOAD=/usr/lib64/libgslcblas.so.0 octave-cli -q linpack.m
```

The number of threads used by *libatlas* was set at compile-time by the package builder. For *libopenblas*, the number of threads is set to, for example 2, by:

```
export OPENBLAS_NUM_THREADS=2
```

Table K.2 shows the average time for 10 runs of each test using the shared library, profile-guided, LTO build. Note that the normalised residuals and calculated filter coefficients differ between implementations.

Octave build	decimator_R2_test.m
	Execution time (seconds)
Debug, shared reference lapack and blas	223.0
Generic, static reference lapack and blas	39.0
Generic, static reference lapack and blas, LTO	38.7
Generic, static reference lapack and blas, PGO	39.7
Generic, static reference lapack and blas, LTO, PGO	34.0
Generic, shared reference lapack and blas	39.1
Generic, shared reference lapack and blas, LTO	38.7
Generic, shared reference lapack and blas, PGO	40.0
Generic, shared reference lapack and blas, LTO, PGO	33.4

Table K.1: Average execution time for 10 runs of *decimator_R2_test.m* with various *Octave* builds.

blas library	linpack.m		decimator_R2_test.m
	MFLOPS	Normalised residual	Execution time (seconds)
Generic libblas/liblapack	4689	17.7	33.4
Intel libblas/liblapack	4653	17.7	33.5
Haswell libblas/liblapack	4465	15.1	34.5
Nehalem libblas/liblapack	4453	17.7	33.6
Skylake libblas/liblapack	4485	15.1	34.5
libblas/liblapack	4780	17.7	33.5
libgslcblas	4655	17.7	33.4
libsatlas	13925	13.6	34.8
libtatlas	30157	11.7	34.6
libopenblas	12465	8.1	35.5
libopenblas (1 thread)	12390	8.1	35.5
libopenblas (2 threads)	21273	7.8	38.3
libopenblas (4 threads)	35102	6.9	38.1

Table K.2: Benchmark results for *linpack.m* and *decimator_R2_test.m* with various *blas* implementations.

SeDuMi

The “official” SeDuMi source is available from *LeHigh University* as *SeDuMi_1_3.zip* [91]. Unfortunately, this version does not seem to be actively maintained and is not compatible with Octave without minor edits for various “MATLABisms”. A fork is available on *Github* as *sedumi-master.zip* [90]. The *Github* version is compatible with Octave. The *Github git* repository can be copied with:

```
git clone https://github.com/sqlp/sedumi.git
```

The source code in this project is based on the *Github git* version:

```
6c49c13e4fcb76e426c3f9b229d17c1088062caa
```

I have patched the *Github* version to fix some bugs and to remove some of the warning messages that occur when compiling the *mex* files. The patch file, *sedumi_master.patch*, was created from the *Github* version, *sedumi-master.zip* [90], as follows:

```
for file in `ls -1 SeDuMi_1_3/*.[cm]`; do \
    diff -wBbEq $file sedumi-master/`basename $file`; \
    if test $? -ne 0; then \
        diff -wBbErupN sedumi-master/`basename $file` $file >> sedumi_master.patch; \
    fi;
done
```

The local bug-fixes are in the file *blkchol2.c* in function *cholonBlk* and in the file *eigK.m*. The change in *cholonBlk* fixes an off-by-one error in the access to $x+inz+1$ [28]. This bug appears in both the *Github* and *LeHigh* versions. The edited version of *blkchol2.c* has the following *git diff*:

```
diff --git a/blkchol2.c b/blkchol2.c
index dc798d6..19761b4 100644
--- a/blkchol2.c
+++ b/blkchol2.c
@@ -113,16 +113,16 @@ void cholonBlk(double *x, double *d, mwIndex m, const mwIndex ncols
----- */
xkk = x[inz];
if(xkk > lb[k]) { /* now xkk > 0 */
-    if(xkk < ub) {
+    if((m>1) && (xkk < ub)) {
        ubk = maxabs(x+inz+1,m-1) / maxu;
        if(xkk < ubk) {
```

The change to *eigK.m* is that suggested in *Note4* of *SparsePOP302/readme.txt*. *SparsePOP* examples will fail without it. This bug appears only in the *Github* version.

As noted above, when compiling a 64-bit version of *SeDuMi* *openblas/f77blas.h* requires *-DOPENBLAS_USE64BITINT*.

SparsePOP

SparsePOP302 [36] compiles and runs on Octave with the minor modifications to *my_unique.m*, *compileSparsePOP.m* and *sparsePOP.m* shown in *SparsePOP302.patch*.

Testing

The *test* directory contains *aegis* regression testing shell scripts. The test scripts must be run from the project root directory. The test scripts pass on my custom build of Octave using the Fedora25 versions of Netlib *lapack* and *blas* (as shown above in *Building Octave with LTO and PGO*).

Testing on Fedora25 with *octave-4.0*

Almost all of the regression tests pass on a standard installation of *octave-4.0.3* on Fedora25 with “pre-loaded” *lapack* and *blas* shared libraries and the *svg* plotting device. For example, to run the regression test for *deczky3_sqp_test.m* under *octave-4.0.3*:

```
export LD_PRELOAD="/usr/lib64/liblapack.so.3.6.1 /usr/lib64/libblas.so.3.6.1"
sed -i -e "s/dpdflatex/dsvg/" deczky3_sqp_test.m
sh test/00/t0006a.sh
```

Reasons for regression test script failures under *octave-4.0.3*:

- *octave-4.0.3* does not support printing plots to the *pdflatex* device with the *gnuplot* Octave graphics toolkit
- *octave-4.0.3* does not support the *defaultaxestitlefontweight* graphics attribute. (Commented out in *test_common.m*).
- there are numerical differences between *octave-4.0.3* and *octave-4.2.1*
- *sedumi_hash_test.m* fails because *octave-4.0.3* does not have a built-in *hash* function
- *sparsePOP_test.m* fails with an internal error in *octave-4.0.3*

Testing on Windows10 with “ubuntu on Windows”

I have used *octave-4.2.1* in a “*ubuntu on Windows*” terminal running the 14.04.5 LTS Ubuntu release. The following tools and libraries were built from source:

- *gcc-6.3.0* with *configure* options *-enable-languages='c,c++,fortran'* *-disable-multilib*
- *make-4.2.1* (run *make-4.2.1* NOT *make* to build this document)
- *lapack-3.6.1* with deprecated routines
- *SuiteSparse-4.5.5*
- *qrupdate-1.1.2*
- *arpack-ng-master*
- *fftw-3.6.6* with *configure* options *-enable-shared -with-combined-threads -enable-threads* and with and without *-enable-single*
- *octave-4.2.1* with the following *configure* options:
-disable-java -without-fltk -without-qt -disable-atomic-refcount
-without-qhull -without-hdf5 -without-glpk
-with-blas=-lblas -with-lapack=-llapack
-with-arpack-libdir=/usr/local/lib -with-qrupdate-libdir=/usr/local/lib
-with-amd-includedir=/usr/local/include/ -with-amd-libdir=/usr/local/lib
-with-camd-includedir=/usr/local/include/ -with-camd-libdir=/usr/local/lib
-with-colamd-includedir=/usr/local/include/ -with-colamd-libdir=/usr/local/lib
-with-ccolamd-includedir=/usr/local/include/ -with-ccolamd-libdir=/usr/local/lib
-with-cholmod-includedir=/usr/local/include/ -with-cholmod-libdir=/usr/local/lib
-with-umfpack-includedir=/usr/local/include/ -with-umfpack-libdir=/usr/local/lib
-with-fft3-includedir=/usr/local/include/ -with-fft3-libdir=/usr/local/lib
-with-fft3f-includedir=/usr/local/include/ -with-fft3f-libdir=/usr/local/lib

export LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib64 is set in the shell environment. Various other packages were installed with *apt-get* including *libmpfr*, *libgmp*, *gnuplot* and *texlive*. I was able to build this document and all regression test shell scripts passed except those testing the Octave scripts *iir_sqp_mmse_tarczynski_ex2_test.m*, *tarczynski_parallel_allpass_test.m* and *tarczynski_polyphase_allpass_test.m* which failed due to differences in the calculated coefficients.

Testing on Windows10 with *octave-4.2.1-w64-installer*

The GNU Octave FTP repository provides a Windows installer for *octave-4.2.1* and its dependencies [45]. Of the scripts required to run in order to produce this document, the following scripts fail under *octave-4.2.1-w64*, apparently because of numerical differences:

- *iir_sqp_slb_differentiator_test.m*
- *iir_sqp_slb_pink_test.m*
- *iir_sqp_slb_hilbert_test.m*
- *schurOneMlattice_sqp_slb_bandpass_test.m*
- *iir_sqp_slb_minimum_phase_test.m*

Monochrome printing

Monochrome (or grey-scale) printing of this document is supported by:

- hiding coloured hyperlinks in the PDF document by compiling this document with:

```
pdflatex '\def\DesignOfIIRFiltersMono\input{DesignOfIIRFilters}'
```

The *DesignOfIIRFiltersMono* flag enables the following L^AT_EX code:

```
\usepackage[hidelinks]{hyperref}
```

- setting the Octave default line palette to all black by uncommenting the following line in *test_common.m*:

```
% set(0, "defaultaxescolororder", zeros(size(get(0, "defaultaxescolororder"))));
```

- modifying the *zplane.m* file from the Octave-Forge *signal* package to plot markers in the default colour:

```
--- signal-1.3.2(inst/zplane.m
+++ /usr/local/share/octave/packages/signal-1.3.2/zplane.m
@@ -100,12 +100,12 @@
    grid on;
    axis(1.05*[xmin, xmax, ymin, ymax]);
    if (!isempty(p))
-      h = plot(real(p), imag(p), "bx");
+      h = plot(real(p), imag(p), "x");
      set(h, 'MarkerSize', 7);
    endif
    if (!isempty(z))
-      h = plot(real(z), imag(z), "bo");
+      h = plot(real(z), imag(z), "o");
      set(h, 'MarkerSize', 7);
    endif
    hold off;
```


Bibliography

- [1] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier and Paul Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2), June 2007. [66](#), [68](#), [402](#)
- [2] A. Antoniou and W.-S. Lu. *Practical Optimization: Algorithms and Engineering Applications*. Springer Science+Business Media, LLC, 2007. ISBN 0-387-71106-6. [132](#), [175](#)
- [3] A. G. Deczky. Synthesis of recursive digital filters using the minimum p-error criterion. *IEEE Transactions on Audio and Electroacoustics*, 20:257–263, October 1972. [11](#), [12](#), [115](#), [125](#), [132](#)
- [4] A. H. Gray and J. D. Markel. Digital Lattice And Ladder Filter Synthesis. *IEEE Transactions on Audio and Electroacoustics*, 21(6):491–500, December 1973. [11](#), [12](#), [71](#)
- [5] A. H. Land and A. G. Doig. An Automatic Method Of Solving Discrete Programming Problems. *Econometrica*, 28(3):497–520, July 1960. [12](#), [294](#)
- [6] A. Krukowski and I. Kale. Two Approaches for fixed-point filter design: "bit-flipping" algorithm and constrained down-hill Simplex method. In *Proceedings of the Fifth International Symposium on Signal Processing and its Applications*, volume 2, pages 965–968, 1999. [12](#), [285](#), [286](#)
- [7] A. P. Ruszczyński. *Nonlinear Optimization*. Princeton University Press, 2006. ISBN 0-691-11915-5. [12](#), [377](#), [378](#), [384](#), [385](#), [387](#)
- [8] A. Tarczynski, G. Cain, E. Hermanowicz and M. Rojewski. A WISE Method for Designing IIR Filters. *IEEE Transactions on Signal Processing*, 49(7):1421–1432, July 2001. [11](#), [93](#), [95](#), [115](#), [116](#), [117](#), [119](#), [122](#)
- [9] Athanasios Papoulis. *Signal Analysis*. McGraw-Hill International, 1981. ISBN 0-07-066468-4. [149](#)
- [10] Automatically Tuned Linear Algebra Software. <http://math-atlas.sourceforge.net/>. [429](#)
- [11] Automatic Differentiation. <http://www.autodiff.org>. [359](#)
- [12] B. Christianson. Global Convergence using De-linked Goldstein or Wolfe Linesearch Conditions. *AMO - Advanced Modeling and Optimization*, 1(1):25–31, 2009. [388](#)
- [13] B. Dumitrescu and R. Niemistö. Multistage IIR Filter Design Using Convex Stability Domains Defined by Positive Realness. *IEEE Transactions on Signal Processing*, 52:962–974, April 2004. [111](#), [115](#)
- [14] B. W. Bomar. New second-order state-space structures for realizing low round off noise digital filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33(1):106–110, February 1985. [57](#), [59](#)
- [15] B. W. Bomar. On the Design of Second-Order State-Space Digital Filter Sections. *IEEE Transactions on Circuits and Systems*, 36(4):542–552, April 1989. [57](#)
- [16] C. T. Mullis and R. A. Roberts. Roundoff Noise in Digital Filters:Frequency Transformations and Invariants. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 24(6):538–550, December 1976. [37](#), [38](#), [56](#), [80](#), [97](#)
- [17] C. T. Mullis and R. A. Roberts. Synthesis of minimum roundoff noise fixed-point digital filters. *IEEE Transactions on Circuits and Systems*, 23:512–551, September 1976. [53](#), [80](#)
- [18] C. T. Mullis and R. A. Roberts. An Interpretation of Error Spectrum Shaping in Digital Filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 30(6):1013–1015, December 1982. [105](#)
- [19] D. Goldfarb and A. Idnani. A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs. *Mathematical Programming*, 27:1–33, 1983. [119](#), [389](#), [390](#), [391](#), [394](#)
- [20] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999. ISBN 1-886529-00-0. [12](#), [378](#), [379](#), [381](#), [384](#), [385](#), [386](#), [387](#), [388](#)

- [21] D. Williamson. Roundoff Noise Minimization and Pole-Zero Sensitivity in Fixed-point Digital Filters Using Residue Feedback. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(5):1210–1220, October 1986. 105, 106, 107, 108
- [22] NIST Digital Library of Mathematical Functions. <http://dlmf.nist.gov/>, Release 1.0.15 of 2017-06-01. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds. 423
- [23] Erwin Kreysig. *Advanced Engineering Mathematics*. John Wiley and Sons, Fifth edition, 1983. ISBN 0-471-88941-5. 353, 356
- [24] NICONET e.V. The Control and Systems Library SLICOT. <http://www.slicot.org>. 48, 426
- [25] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming*, 95:3–51, 2001. 12, 167, 175
- [26] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996. ISBN 0-8018-5413-8. 28, 34, 102, 103, 117, 381, 382, 383, 389, 417, 418
- [27] Gian Antonio Mian and Alberto Pio Nainer. A Fast Procedure to Design Equiripple Minimum-Phase FIR Filters. *IEEE Transactions on Circuits and Systems*, 29(5):327–331, May 1982. 165, 412, 413
- [28] GitHub. SeDuMi results are not apparently deterministic? <https://github.com/sqlp/sedumi/issues/15>. 433
- [29] gnome.org. Dia diagram editor. <https://wiki.gnome.org/Apps/Dia/>. 425
- [30] David Goldberg. What Every Computer Scientist Should Know About Floating-point Arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991. 12
- [31] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. <http://eigen.tuxfamily.org>, 2010. 413
- [32] H. A. Spang III and P. M. Shultheis. Reduction of quantizing noise by use of feedback. *IRE Transactions on Communications Systems*, 10:373–380, December 1962. 105
- [33] H. J. Orchard and Alan N. Willson. On the Computation of a Minimum-Phase Spectral Factor. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, 50(3):365–375, March 2003. 165, 412
- [34] H. Johansson and L. Wanhammar. High-Speed Recursive Digital Filters Based on the Frequency-Response Masking Approach. *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 47(1):48–61, January 2000. 12, 236
- [35] Masakazu Kojima Hayato Waki, Sunyoung Kim and Masakazu Muramatsu. Sums of Squares and Semidefinite Program Relaxations for Polynomial Optimization Problems with Structured Sparsity. *SIAM Journal on Optimization*, 17(1):218–242, 2006. 316
- [36] Hayato Waki, Sunyoung Kim, Masakazu Kojima, Masakazu Muramatsu, Hiroshi Sugimoto and Makoto Yamashita. SparsePOP (Sparse SDP Relaxation of Polynomial Optimization Problems). <http://sparsepop.sourceforge.net>. 316, 433
- [37] I. Kunold. Linear phase realization of wave digital lattice filters. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1455–1458, 1988. 210
- [38] I. W. Selesnick, M. Lang and C. S. Burrus. Constrained Least Square Design of FIR Filters without Specified Transition Bands. *IEEE Transactions on Signal Processing*, 44(8):1879–1892, August 1996. 113, 114
- [39] I. W. Selesnick, M. Lang and C. S. Burrus. A Modified Algorithm for Constrained Least Square Design of Multiband FIR Filters without Specified Transition Bands. *IEEE Transactions on Signal Processing*, 46(2):497–501, February 1998. 11, 12, 114, 120
- [40] J. H. Wilkinson. *Studies in Numerical Analysis, Ed. G.H.Golub*, volume 24 of *Studies in Mathematics*, chapter The perfidious polynomial, pages 1–28. Mathematical Association of America, 1984. 12
- [41] J. H. Wilkinson, editor. *The Algebraic Eigenvalue Problem*. Oxford University Press, Inc., 1988. ISBN 0-198-53418-3. 28
- [42] J. L. Sullivan and J. W. Adams. PCLS IIR Digital Filters with Simultaneous Frequency Response Magnitude and Group Delay Specifications. *IEEE Transactions on Signal Processing*, 46(11):2853–2861, November 1998. 114, 125
- [43] J. W. Adams and J. L. Sullivan. Peak Constrained Least-Squares Optimization. *IEEE Transactions on Signal Processing*, 46(2):306–321, February 1998. 120

- [44] J.B.Lasserre. Global Optimization with Polynomials and the Problem of Moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001. 316
- [45] Octave installer for Microsoft Windows. <https://ftp.gnu.org/gnu/octave/windows/octave-4.2.1-w64-installer.exe>. 435
- [46] John W. Eaton, David Bateman, Sören Hauberg and Rik Wehbring. *GNU Octave version 4.2.0 manual: a high-level interactive language for numerical computations*, 2016. 11, 36
- [47] Jorge Nocedal and Stephen J.Wright. *Numerical Optimization*. Springer, second edition, 2006. ISBN 0-387-30303-0. 12, 377, 380
- [48] Kai Hwang. *Computer Arithmetic : Principles, Architecture and Design*. John Wiley and Sons, 1979. ISBN 0-471-03496-7. 275
- [49] Keshab K. Parhi. *VLSI Digital Signal Processing Systems : Design and Implementation*. Wiley Inter-Science, 1999. ISBN 0-471-24186-5. 11, 12, 65, 66, 67, 68, 69, 70, 72, 73, 74, 75, 76, 81, 86, 92, 103, 275, 276
- [50] Kim-Chuan Toh. A note on the calculation of step-lengths in interior-point methods for semidefinite programming. *Computational Optimization and Applications*, 21(3):301–310, 2002. 389
- [51] L. Fousse et al. MPFR: A multiple-precision binary floating-point library with correct rounding. <http://www.mpfr.org>. 66, 68, 402
- [52] L. Milić and J. Ćertić. Two-Channel IIR Filter Banks Utilizing the Frequency-Response Masking Technique. *Telfor Journal*, 1(2):45–48, 2009. 12
- [53] L. Milić, J. Ćertić and M. Lutovac. A class of FRM-based all-pass digital filters with applications in half-band filters and Hilbert transformers. In *International Conference on Green Circuits and Systems*, pages 273–278. IEEE, 2010. 268
- [54] L. Theile. On the Sensitivity of Linear State-Space Systems. *IEEE Transactions on Circuits and Systems*, 33(5):502–510, May 1986. 28
- [55] Wu-Sheng Lu. Digital Filter Design: Global Solutions via Polynomial Optimization. *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems*, pages 49–52, 2006. 316
- [56] M. A. Richards. Application of Deczky’s Program for Recursive Filter Design to the Design of Recursive Decimators. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 30(5):811–814, October 1982. 11, 12, 115, 359
- [57] M. C. Lang. Least-Squares Design of IIR Filters with Prescribed Magnitude and Phase Responses and a Pole Radius Constraint. *IEEE Transactions on Signal Processing*, 48(11):3109–3121, November 2000. 115
- [58] M. J. D. Powell. A Fast Algorithm for Nonlinearly Constrained Optimization Calculations. *Lecture Notes in Mathematics*, 630:144–157, 1978. 377, 380, 384, 385
- [59] M. J. Rutter. Linpack benchmark in Octave/Matlab. <http://www.tcm.phy.cam.ac.uk/~mjr/linpack/linpack.m>, November 2015. 431
- [60] M. Renfors and T. Saramäki. Recursive Nth-Band Digital Filters-Part I:Design and Properties. *IEEE Transactions on Circuits and Systems*, 34(1):24–39, January 1987. 12, 217
- [61] M. Renfors and T. Saramäki. Recursive Nth-Band Digital Filters-Part II:Design of Multistage Decimators and Interpolators. *IEEE Transactions on Circuits and Systems*, 34(1):40–51, January 1987. 12, 217
- [62] M. S. Lobo, L. Vandenberghe, S. Boyd and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 1998. 12
- [63] Min Su Kim, Sun Joo Kwon and Se Young Oh. The Performance of a Modified Armijo Line Search Rule in BFGS Optimisation Method. *Journal of the ChungCheong Mathematical Society*, 21(1):117–127, March 2008. 388
- [64] Octave Forge. Non-linear optimization toolbox. <http://octave.sourceforge.net/optim/>. 12, 331, 337, 344
- [65] Octave Forge. Parallel execution package. <http://octave.sourceforge.net/parallel/>. 365
- [66] Octave Forge. Signal processing toolbox. <http://octave.sourceforge.net/signal/>. 60
- [67] Octave Forge. Symbolic calculation toolbox. <http://octave.sourceforge.net/symbolic/>. 359
- [68] P. A. Regalia. Stable and Efficient Lattice Algorithms for Adaptive IIR Filtering. *IEEE Transactions on Signal Processing*, 40(2):375–388, February 1992. 222

- [69] P. A. Regalia, S. K. Mitra and P. P. Vaidyanathan. The Digital All-Pass Filter: A Versatile Signal Processing Building Block. *Proceedings of the IEEE*, 76(1):19–37, January 1988. [12](#)
- [70] P. P. Vaidyanathan. Passive Cascaded-Lattice Structures for Low-Sensitivity FIR Filter Design, with Applications to Filter Banks. *IEEE Transactions on Circuits and Systems*, 33(11):1045–1064, November 1986. [291](#), [409](#), [410](#), [411](#)
- [71] P. P. Vaidyanathan and S. K. Mitra. Low Passband Sensitivity Digital Filters: A Generalized Viewpoint and Synthesis Procedures. *Proceedings of the IEEE*, 72(4):404–423, April 1984. [86](#), [99](#), [399](#)
- [72] P. P. Vaidyanathan, S. K. Mitra and Y. Nuevo. A New Approach to the Realization of Low-Sensitivity IIR Digital Filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(2):350–361, April 1986. [12](#), [86](#), [99](#), [189](#), [399](#), [402](#)
- [73] Peter Miller. aegis-4.24. <http://aegis.sourceforge.net>. [425](#)
- [74] Peter Miller. fhist-1.21.D001. <http://fhist.sourceforge.net>. [425](#)
- [75] GNU Project. GMP: The GNU Multiple Precision Arithmetic Library. <http://gmplib.org>. [66](#), [68](#), [402](#)
- [76] R. A. Roberts and C. T. Mullis. *Digital Signal Processing*. Addison Wesley, 1987. ISBN 0-201-16350-0. [11](#), [17](#), [27](#), [29](#), [30](#), [33](#), [35](#), [36](#), [43](#), [46](#), [47](#), [48](#), [57](#), [58](#), [59](#), [80](#), [99](#), [101](#), [102](#), [103](#), [116](#), [419](#)
- [77] R. Rehman and I. C. F. Ipsen. La Budde’s Method for Computing Characteristic Polynomials. *ArXiv e-prints*, April 2011. [28](#)
- [78] R. Sedgwick. *Algorithms in C++*. Addison-Wesley, 1990. ISBN 0-201-51059-6. [294](#)
- [79] R. Storn and K. Price. Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. <http://www1.icsi.berkeley.edu/~storn/TR-95-012.pdf>. [344](#)
- [80] Rika Ito, Tetsuya Fujie, Kenji Suyama and Ryuichi Hirabayashi. A powers-of-two term allocation algorithm for designing FIR filters with CSD coefficients in a min-max sense. <http://www.eurasip.org/Proceedings/Eusipco/Eusipco2004/defevent/papers/cr1722.pdf>. [11](#), [12](#), [277](#), [294](#)
- [81] Rika Ito, Tetsuya Fujie, Kenji Suyama, Ryuichi Hirabayashi. New design method of FIR filters with SP2 coefficients based on a new linear programming relaxation with triangle inequalities. In *11th European Signal Processing Conference (EUSIPCO 2002)*, 2002. [305](#)
- [82] Rizwana Rehman. *Numerical Computation of the Characteristic Polynomial of a Complex Matrix*. PhD thesis, North Carolina State University, 2010. [28](#)
- [83] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2008. ISBN 0-521-83378. [12](#)
- [84] S. Hammarling. Numerical solution of the discrete-time, convergent, non-negative definite Lyapunov equation. *Systems and Control Letters*, 17:137–139, 1991. [48](#)
- [85] S. J. Wright. Superlinear Convergence of a Stabilized SQP Method to a Degenerate Solution. *Computational Optimization and Applications*, 11:253–275, 1998. [383](#)
- [86] S. K. Mitra and R. J. Sherwood. Canonic Realizations of Digital Filters Using the Continued Fraction Expansion. *IEEE Transactions on Audio and Electroacoustics*, 20(3):185–194, August 1972. [33](#)
- [87] S.-P. Wu, S. Boyd, and L. Vandenberghe. FIR Filter Design via Semidefinite Programming and Spectral Factorization. In *IEEE Conference on Decision and Control*, volume 1, pages 271–276, December 1996. [11](#)
- [88] I. W. Selesnick. A collection of scripts for constrained-least-squares FIR filter design. <http://dsp.rice.edu/files/software/ConstrainedLeastSquaresAllprogs.tar.zip>. [114](#)
- [89] Shunsuke Koshita, Satoru Tanaka, Masahide Abe and Masayuki Kawamata. Grammian-Preserving Frequency Transformation for Linear Discrete-Time Systems Using Normalised Lattice Structure. *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1124–1127, 2008. [97](#)
- [90] J. Sturm. SeDuMi_1_3 at GitHub. <https://github.com/sqlp/sedumi>. [11](#), [433](#)
- [91] J. Sturm. SeDuMi_1_3 at LeHigh University. <http://sedumi.ie.lehigh.edu/sedumi>. [12](#), [169](#), [433](#)
- [92] T. Saramäki, Y.C. Lim and R. Yang. The synthesis of half-band filter using frequency-response masking technique. *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 42(1):58–60, January 1995. [262](#)
- [93] The MathWorks. MATLAB: The Language of Technical Computing. <http://mathworks.com>. [11](#)
- [94] W.-S. Lu and T. Hinamoto. Optimal Design of IIR Digital Filters With Robust Stability Using Conic Quadratic-Programming Updates. *IEEE Transactions on Signal Processing*, 51(6):1581–1592, June 2003. [181](#), [189](#)

- [95] W.-S. Lu and T. Hinamoto. Optimal Design of IIR Frequency-Response-Masking Filters Using Second-Order Cone Programming. *IEEE Transactions on Circuits and Systems-I:Fundamental Theory and Applications*, 50(11):1401–1412, November 2003. 12, 115, 168, 236, 239, 240, 247, 252
- [96] W.-S. Lu and T. Hinamoto. Jointly Optimized Error-Feedback and Realization for Roundoff Noise Minimization in State-Space Digital Filters. *IEEE Transactions on Signal Processing*, 53(6):2135–2145, June 2005. 105, 106
- [97] W.-S. Lu and T. Hinamoto. Design of FIR Filters with Discrete Coefficients via Polynomial Programming: Towards the Global Solution. *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2048–2051, 2007. 316
- [98] Wikipedia. Peter Miller. [https://en.wikipedia.org/wiki/Peter_Miller_\(software_engineer\)](https://en.wikipedia.org/wiki/Peter_Miller_(software_engineer)). 425
- [99] William Schelter. Maxima, a Computer Algebra System. <http://maxima.sourceforge.net/>. 359, 365
- [100] Wu-Sheng Lu. Use SeDuMi to Solve LP, SDP and SCOP Problems: Remarks and Examples. <http://www.ece.uvic.ca/~wslu/Talk/SeDuMi-Remarks.pdf>. 169
- [101] Wu-Sheng Lu. An Argument-Principle Based Stability Criterion and Application to the Design of IIR Digital Filters. *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 4431–4434, 2006. 115
- [102] Wu-Sheng Lu. Design of FIR Filters with Discrete Coefficients via Sphere Relaxation. *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2509–2512, 2006. 305
- [103] Xu Zhong. On Inverses and Generalized Inverses of Hessenberg Matrices. *Linear Algebra and its Applications*, 101:167–180, 1988. 222
- [104] Y. C. Lim. Frequency-Response Masking Approach for the Synthesis of Sharp Linear Phase Digital Filters. *IEEE Transactions on Circuits and Systems*, 33(24):357–364, April 1986. 12, 236, 237
- [105] Y. C. Lim, R. Yang, D. Li and J. Song. Signed Power-of-Two Term Allocation Scheme for the Design of Digital Filters. *IEEE Transactions on Circuits and Systems-II:Analog and Digital Signal Processing*, 46(5):577–584, May 1999. 11, 12, 276
- [106] Z. Doğonata and P. P. Vaidyanathan. On One-Multiplier Implementations of FIR Lattice Structures. *IEEE Transactions on Circuits and Systems*, 34(12):1608–1609, December 1987. 291