# On the design of IIR filters

Robert G. Jenssen

September 30, 2022

$$\left(1+9^{-4^{6\times 7}}\right)^{3^{2^{85}}}$$

See: https://erich-friedman.github.io/mathmagic/0804.html

Copyright © 2017–2022 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


**I State Variable description of digital filters** **23**

**1 A review of the State Variable description of digital filters** **25**

1.1 The z-transform . . . 25

1.2 Filter difference equation . . . 26

1.3 Filter transfer function . . . 27

1.4 Filter signal flow graph . . . 27

1.5 State variable description of a signal flow graph . . . 31

1.6 Controllability . . . 31

1.7 Observability . . . 32

1.8 Coordinate Transformations . . . 32

1.9 State variable descriptions and the transfer function . . . 32

1.9.1 Transformation of a transfer function to a state variable description . . . 33

1.9.2 Transformation of a transfer function to a state variable description by continued fraction expansion . . . 34

1.9.3 Transformation of a state variable description to a transfer function . . . 36

1.9.4 Sensitivity of the state variable description of a transfer function . . . 39

1.10 Time domain description . . . 39

1.11 Unit Pulse Response . . . 39

1.12 Factored state variable descriptions . . . 40

1.12.1 Factored state variable filters with fractional delays . . . 40

1.12.2 Construction of the factored state variable description . . . 41

1.13 Block processing and decimation filters . . . 44

**2 Frequency transformations of state variable filters** **47**

2.1 Frequency Transformations . . . 47

2.2 Frequency Transformation of the Transfer Function . . . 48

2.3 Frequency Transformations of State Variable Filters . . . 49

2.4 An example: frequency transformations of a 5-th order elliptic filter . . . 51

**3 Round-off noise in state variable filters** **55**

3.1 Quantisation noise in digital filters . . . 55

3.2 Limit-cycle oscillations in digital filters . . . 56

3.3 State variable filters and wide sense stationary inputs . . . 56

3.3.1 The filter state covariance matrix . . . 56

3.3.2 The output response to white noise in a state variable . . . 57

3.3.3 Scaling State Variable Filters To Avoid Overflow . . . 58

3.4 Estimation of output round-off noise in state variable filters . . . 59

3.4.1 Rounding-to-minus-infinity quantisation noise . . . 60

3.5 Minimization of round-off noise in the calculation of the state vector . . . 61

3.6 Coefficient sensitivity . . . 63

3.7 Factored state variable filters and wide sense stationary inputs . . . 63

3.8 Frequency transformations and round-off noise . . . 64

**4 State variable filter realisation as a cascade of second order sections** **65**

4.1 Second Order State Variable Filters Optimised for Overflow and Round-Off Noise . . . 65

4.2 Design equations for optimised second order state variable filters . . . 65

4.3 Block optimal second order cascade filter realisations . . . 68

4.4 An example of a second-order state-variable cascade filter . . . 68

4.4.1 Comparison of calculated noise gains . . . 68

4.4.2 Simulation results . . . 69

4.4.3 Comparison with an N=10 example . . . 71

4.5 Coefficient sensitivity and round-off noise of first-order and second-order all-pass filter sections . . . 72

4.5.1 Searching for realisations of all-pass filter transfer functions . . . 73

4.5.2 Maximum phase gradient and round-off noise of some first-order all-pass filter sections . . . 74

4.5.3 Maximum phase gradient and round-off noise of some second-order all-pass filter sections . . . 76

**5 Filter synthesis by the Schur decomposition** **95**

5.1 The Schur algorithm . . . 95

5.1.1 Computation of Schur polynomials . . . 95

5.1.2 Orthonormality of Schur Polynomials . . . 96

5.1.3 Polynomial Expansion Algorithm . . . 98

5.1.4 Power calculation using the Schur algorithm . . . 98

5.2 Derivation of Digital Lattice Filters . . . 98

5.2.1 Derivation of FIR, All-Pole and All-Pass Lattice Filters . . . 99

5.3 Derivation of the One-Multiplier IIR Lattice Filter . . . 100

5.4 Derivation of the Normalised Lattice Filter . . . 102

5.5 Derivation of the Scaled Normalised Lattice Filter . . . 103

5.5.1 Example: synthesis of a 3rd order Butterworth lattice filter . . . 105

5.6 State Variable Descriptions for Schur Lattice Filters . . . 106

5.6.1 State variable description of the Schur FIR lattice filter . . . 106

5.6.2 State variable description of the one-multiplier IIR lattice filter . . . 107

5.6.3 State variable description of a retimed one-multiplier Schur lattice filter . . . 108

5.6.4 State variable description of the scaled-normalised IIR lattice filter . . . 110

5.7 Roundoff Noise Calculation in Schur Lattice Filters . . . 112

5.7.1 Round-off noise of the normalised-scaled lattice filter . . . 113

5.7.2 Round-off noise of the one multiplier lattice filter . . . 121

5.7.3 Frequency transformations and Schur normalised-scaled lattice filter round-off noise . . . 124

5.8 Examples of retiming Schur lattice filters . . . 124

5.8.1 Retiming a 3-rd order lattice filter . . . 124

5.8.2 Retiming a 4-th order Schur normalised-scaled lattice filter . . . 125

5.8.3 Retiming a 6th-order Schur one-multiplier lattice filter . . . 127

5.8.4 Frequency transformations of retimed Schur lattice filters . . . 129

5.9 Summary . . . 132

**6 Orthogonal state variable filters** **133**

6.1 Definition of orthogonal state variable filters . . . 133

6.2 The Lattice Orthogonal All-Pass Filter Section . . . 134

6.3 Noise gain of orthogonal filters . . . 134

6.4 Realisation of arbitrary filters from orthogonal sub-filters . . . 135

6.4.1 An example of structural variations . . . 137

**7 Feedforward and feedback of state quantisation error in state variable filters** **139**

7.1 Problem formulation . . . 139

7.2 Minimisation of round-off noise with $\delta = I$ and $\eta = 0$ . . . 141

**II Constrained optimisation of the IIR filter frequency response** **143**

**8 IIR filter design using Sequential Quadratic Programming with the transfer function defined by pole and zero locations** **145**

8.1 Problem Statement . . . 145

8.1.1 Solution of the constrained quasi-Newton optimisation problem . . . 146

8.1.2 Choice of active constraints . . . 147

8.1.3 Linearisation of peak constraints . . . 148

8.1.4 Ensuring the stability of the IIR filter . . . 149

8.1.5 Selecting an initial filter design . . . 149

8.2 Examples of IIR filter design with SQP and constrained pole and zero locations . . . 153

8.2.1 Introductory comments on the IIR filter design examples . . . 153

8.2.2 Tarczynski et al. Example 2 . . . 156

8.2.3 Deczky's Example 3 . . . 159

8.2.4 Deczky's Example 1 . . . 168

8.2.5 Low-pass R=2 decimation filter . . . 172

8.2.6 Band-pass R=2 decimation filter . . . 183

8.2.7 Hilbert transform R=2 decimation filter . . . 190

8.2.8 R=2 differentiator filter . . . 194

8.2.9 Low-pass differentiator filter . . . 198

8.2.10 Pink noise filter . . . 201

8.2.11 Minimum phase R=2 filter . . . 204

8.2.12 Non-linear phase FIR low-pass filter . . . 207

8.2.13 Minimum phase FIR bandpass filter . . . 209

**9 IIR filter design using Second Order Cone Programming** **213**

9.1 Second Order Cone Programming . . . 213

9.2 Design of IIR filters with SOCP . . . 213

9.3 Using the *SeDuMi* SOCP solver . . . 215

9.4 An example of SOCP design of an IIR filter expressed in *gain-zero-pole* format with *SeDuMi* . . . 215

9.5 SOCP design of a non-linear phase FIR low-pass filter . . . 224

9.6 Comparison of FIR and IIR low-pass filters having approximately flat pass-band group delay with symmetric FIR filters . . . 228

9.7 SOCP MMSE design of a bandpass R=2 filter expressed in *gain-zero-pole* format with *SeDuMi* . . . 230

9.8 SOCP MMSE design of a multi-band-pass filter expressed in *gain-zero-pole* format with *SeDuMi* . . . 234

**10 IIR filter design with a pre-defined structure** **237**

10.1 Design of an IIR filter composed of second-order sections . . . 237

10.1.1 Linear constraints on the stability of second-order filter sections . . . 237

10.1.2 Linear constraints on limit-cycle oscillations in second-order filter sections . . . 238

10.1.3 Design of an IIR filter composed of second order sections with *SeDuMi* . . . 239

10.1.4 Some notes on the design of an IIR filter composed of second order sections with *SeDuMi* . . . 244

10.2 Design of an IIR filter as the sum of two all-pass filters . . . 245

10.2.1 Design of an IIR filter as the sum of two all-pass filters each composed of second-order sections . . . 245

10.2.2 Design of an IIR filter as the sum of an all-pass filter composed of second-order sections and a delay . . . 253

10.2.3 Design of an IIR filter as the sum of two all-pass filters each represented in pole-zero form . . . 256

10.2.4 Design of an IIR filter as the sum of a delay and an all-pass filter represented in pole-zero form . . . 272

10.2.5 Design of an IIR filter as the polyphase decomposition into two all-pass filters each represented in pole-zero form . . . 279

10.3 Design of an IIR Schur lattice filter . . . 285

10.3.1 Design of an IIR one-multiplier Schur lattice low-pass filter using SOCP . . . 285

10.3.2 Design of an IIR one-multiplier Schur lattice low-pass filter using SQP . . . 288

10.3.3 Design of an IIR Schur normalised-scaled lattice low-pass filter using SQP . . . 291

10.3.4 Design of an IIR low-pass filter with parallel Schur one-multiplier all-pass lattice filters using SOCP . . . 294

10.3.5 Design of a parallel IIR Schur one-multiplier all-pass lattice band-pass filter using SOCP . . . 297

10.3.6 Design of an IIR one-multiplier Schur lattice band-pass filter using SQP . . . 299

10.3.7 Design of an IIR normalised-scaled Schur lattice band-pass filter using SQP . . . 304

10.3.8 Design of an IIR one-multiplier Schur lattice Hilbert filter using SOCP . . . 307

10.3.9 Design of an IIR one-multiplier Schur lattice Hilbert filter using SQP . . . 309

10.3.10 Design of a parallel IIR Schur one-multiplier all-pass lattice band-pass Hilbert filter using SOCP . . . 311

10.3.11 Design of a parallel IIR Schur one-multiplier all-pass lattice multi-band-pass filter using SOCP . . . 313

10.4 Design of IIR filters with a sharp transition band by frequency response masking . . . 318

10.4.1 Review of Frequency Response Masking digital filters . . . 318

10.4.2 Design of an FRM digital filter with an IIR model filter consisting of a cascade of second-order sections using SOCP . . . 321

10.4.3 Design of an FRM digital filter with an IIR model filter represented in gain-pole-zero form using SOCP and PCLS optimisation . . . 327

10.4.4 Design of an FRM digital filter with an allpass model filter represented in gain-pole-zero form using SOCP and PCLS optimisation . . . 333

10.4.5 Design of an FRM digital filter with an IIR model filter consisting of parallel allpass filters . . . 338

10.4.6 Design of an FRM low-pass digital filter with an all-pass Schur lattice model filter in parallel with a delay using SOCP and PCLS optimisation . . . 344

10.4.7 Design of an FRM half-band digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation . . . 348

10.4.8 Design of an FRM Hilbert digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation . . . 354

**III Design of IIR filters with integer coefficients** 359

**11 Signed-digit representation of filter coefficients** 362

11.1 Lim's method for allocating signed-digits to filter coefficients . . . 363

11.2 Ito's method for allocating signed-digits to filter coefficients . . . 364

11.3 Signed-digit allocation of the coefficients of a Schur one-multiplier lattice filter . . . 364

11.4 Signed-digit allocation of the coefficients of a symmetric FIR band-pass filter . . . 372

**12 Exhaustive search for integer and signed-digit filter coefficients** 377

**13 Searching for integer and signed-digit filter coefficients with the *bit-flipping* algorithm** 378

13.1 Bit-flipping search for the signed-digit coefficients of a direct-form bandpass IIR filter . . . 378

13.2 Bit-flipping search for the signed-digit coefficients of a normalised-scaled lattice bandpass IIR filter . . . 381

13.3 Bit-flipping search for the signed-digit coefficients of a one-multiplier lattice bandpass IIR filter . . . 382

13.4 Bit-flipping search for the signed-digit coefficients of a one-multiplier parallel-allpass lattice bandpass IIR filter . 385

13.5 Bit-flipping search for the signed-digit coefficients of a minimum-phase bandpass FIR filter . . . 388

13.6 Bit-flipping search for the signed-digit coefficients of a direct-form symmetric bandpass FIR filter . . . 389

**14 *Branch-and-bound* search for signed-digit coefficients** 392

14.1 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a direct-form symmetric bandpass FIR filter . 394

14.2 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a lattice band-pass IIR filter . . . 396

14.3 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a lattice band-pass IIR filter . . . 399

14.4 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass lattice low-pass IIR filter 402

14.5 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a parallel all-pass lattice IIR elliptic low-pass filter . . . 405

14.6 Branch-and-bound search for the 16-bit 4-signed-digit coefficients of a parallel all-pass lattice IIR elliptic low-pass filter . . . 407

14.7 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass IIR filter 410

14.8 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass Hilbert IIR filter . . . 413

14.9 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass Hilbert IIR filter . . . 417

14.10 Branch-and-bound search for the coefficients of an FRM low-pass filter implemented with 12-bits and an average of 3-signed-digits . . . 421

14.11 Branch-and-bound search for the coefficients of an FRM Hilbert filter implemented with 12-bits and 2-signed-digits 424

14.12 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of an FIR Hilbert filter . . . 428

14.13 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of an FIR Hilbert band-pass filter . . . 430

**15 Successive coefficient relaxation search for signed-digit filter coefficients** **432**

15.1 SOCP-relaxation search for the signed-digit coefficients of a direct-form symmetric bandpass FIR filter . . . . . 432

15.2 SQP-relaxation search for the signed-digit coefficients of a lattice bandpass IIR filter . . . . . 434

15.3 SQP-relaxation search for the signed-digit coefficients of a lattice lowpass IIR filter . . . . . 437

15.4 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice low-pass IIR filter . . . . . 440

15.5 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice band-pass IIR filter . . . . . 443

15.6 SOCP-relaxation search for the signed-digit coefficients of a lattice Hilbert IIR filter . . . . . 446

15.7 SOCP-relaxation search for the signed-digit coefficients of a one-multiplier lattice low-pass differentiator IIR filter 448

15.8 SOCP relaxation search for the 12-bit 3-signed-digit coefficients of a FRM low-pass filter . . . . . 450

15.9 SOCP-relaxation search for the signed-digit coefficients of an FRM Hilbert IIR filter with an all-pass lattice model filter . . . . . 452

15.10 SOCP-relaxation search for the signed-digit coefficients of a direct-form FIR Hilbert filter . . . . . 455

15.11 POP relaxation search for the signed-digit coefficients of a one-multiplier lattice bandpass filter . . . . . 457

15.12 SOCP-relaxation search for the signed-digit coefficients of a lattice FIR filter . . . . . 460

**16 Semi-definite programming search for integer and signed-digit filter coefficients** **465**

16.1 SDP optimisation of the signed-digit coefficients of a direct-form symmetric FIR filter . . . . . 466

16.2 SDP optimisation of the signed-digit coefficients of an FIR Hilbert filter . . . . . 469

16.3 SDP optimisation of the signed-digit coefficients of an FIR Hilbert band-pass filter . . . . . 471

16.4 SDP-relaxation search for the signed-digit coefficients of a lattice bandpass IIR filter . . . . . 473

16.5 SDP-relaxation search for the signed-digit coefficients of a parallel allpass lattice elliptic low-pass IIR filter . . . 477

16.6 SDP-relaxation search for the signed-digit coefficients of a parallel allpass lattice bandpass Hilbert IIR filter . . . 480

**17 Comparison of filter coefficient search methods for a 5th order elliptic filter with 6-bit integer and 2-signed-digit coefficients** **484**

17.1 Searching with the bit-flipping algorithm . . . . . 484

17.2 Searching with the *Nelder-Mead* simplex algorithm . . . . . 491

17.3 Searching with the *simulated annealing* algorithm . . . . . 497

17.4 Searching with the *differential evolution* algorithm . . . . . 504

17.5 Summary of the search algorithm comparison . . . . . 510

IV Appendixes 511

A Review of Complex Variables 513

A.1 Complex Functions . . . 513

A.2 Limit . . . 513

A.3 The Cauchy-Riemann Equations . . . 513

A.4 Line integrals in the complex plane . . . 514

A.5 Cauchy's Integral Theorem . . . 515

A.6 Cauchy's Integral Formula . . . 515

A.7 Derivatives of an analytic function . . . 516

A.8 Laurent's Theorem . . . 516

A.9 Residues . . . 516

A.10 Cauchy's Argument Principle . . . 517

A.11 Rouché's Theorem . . . 517

B Review of selected results from linear algebra 519

B.1 Norm of a matrix . . . 519

B.2 Trace of a matrix . . . 519

B.3 Rank, range, span and null-space of a matrix . . . 520

B.4 Matrix determinants . . . 520

B.4.1 Definitions . . . 520

B.4.2 Matrix exponential . . . 521

B.5 Positive-definite matrixes . . . 521

B.6 Schur complement . . . 521

B.6.1 Schur complement of a matrix . . . 522

B.6.2 Schur complement of a positive definite matrix . . . 523

B.7 Convex vector spaces . . . 524

B.7.1 Definitions on convex sets . . . 524

B.7.2 The separating hyperplane theorem . . . 524

B.7.3 The S-procedure . . . 525

B.7.4 The *log det A* penalty function . . . 526

**C Review of Chebyshev's polynomials** **527**

C.1 Recurrence relations . . . 529

C.2 Differentiation and integration of the Chebyshev polynomials . . . 529

C.3 Chebyshev differential equations . . . 530

C.4 Orthogonality of the Chebyshev polynomials . . . 530

C.5 Approximation of functions by Clenshaw's recurrence . . . 530

**D Review of Legendre's elliptic integrals and Jacobi's elliptic functions** **533**

D.1 Doubly periodic functions . . . 533

D.2 Legendre's elliptic integrals . . . 533

D.3 Computation of Legendre's elliptic integrals . . . 534

D.4 Jacobi's theta functions . . . 536

D.5 Computation of Jacobi's theta functions . . . 536

D.6 Jacobi's elliptic functions . . . 537

D.7 Inverses of Jacobi's elliptic functions . . . 538

D.8 Elementary identities for the elliptic integrals and elliptic functions . . . 538

D.9 Related functions . . . 539

D.10 Octave implementations . . . 540

**E Review of *Lanczos* tridiagonalisation of an unsymmetric matrix** **542**

**F Review of Lagrange Interpolation** **545**

F.1 The barycentric Lagrange polynomial . . . 545

F.2 Node distributions . . . 546

**G IIR filter amplitude, phase and group-delay frequency responses** **549**

G.1 IIR filter responses . . . 550

G.1.1 IIR filter amplitude response . . . 550

G.1.2 IIR filter phase response . . . 551

G.1.3 IIR filter group-delay response . . . 551

G.2 Partial derivatives of the IIR filter responses . . . 551

G.2.1 Partial derivatives of the IIR filter amplitude response . . . 551

G.2.2 Partial derivatives of the IIR filter phase response . . . 552

G.2.3 Partial derivatives of the IIR filter group-delay response . . . 553

G.3 Second partial derivatives of the IIR filter response . . . 553

G.3.1 Second partial derivatives of the IIR filter amplitude response . . . 553

G.3.2 Second partial derivatives of the IIR filter phase response . . . 556

G.3.3 Second partial derivatives of the IIR filter group-delay response . . . 557

**H Gradient of the IIR filter amplitude response with respect to frequency** **559**

**I Allpass filter frequency response** **561**

I.1 Allpass filter phase response . . . 561

I.2 Allpass filter group delay response . . . 562

**J Gradients of the state variable filter frequency response** **565**

J.1 Gradients of the state variable filter complex frequency response . . . 565

J.2 State variable filter squared-magnitude response . . . 565

J.3 State variable filter phase response . . . 566

J.4 State variable filter group-delay response . . . 566

**K Constrained non-linear optimisation** **569**

K.1 Newton's method for a quadratic function . . . 569

K.2 Lagrange multipliers . . . 569

K.3 The dual problem . . . 570

K.4 Karush-Kuhn-Tucker conditions for constrained optimisation . . . 570

K.5 Constrained optimisation using Newton's method . . . 571

K.6 Local convergence . . . 572

K.7 Quasi-Newton methods . . . 572

K.7.1 Updating the Hessian approximation with the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) formula . . . 573

K.7.2 A modified Cholesky factorisation of the Hessian . . . 573

K.7.3 Wright's modification for degenerate constraints . . . 575

K.7.4 Bertsekas' modification to the Hessian . . . 576

K.8 Penalty and barrier methods . . . 576

K.8.1 Penalty functions . . . 577

K.8.2 Barrier functions . . . 578

K.9 Finding the step size . . . 579

K.9.1 Line search with the Golden-Section . . . 579

K.9.2 Inexact step-size selection . . . 580

K.9.3 Lanczos step-size selection . . . 581

K.10 Initial solution with the Goldfarb-Idnani algorithm . . . 582

K.11 Implementation examples . . . 587

**L Fourier transform of the Gaussian function** **591**

L.1 Preliminary results . . . 591

L.1.1 Integral of the Gaussian function . . . 591

L.1.2 Fourier transform of the derivative of a function . . . 591

L.2 Derivation of the Fourier transform of the Gaussian function in the frequency domain . . . 592

**M Design of IIR digital filter transfer functions** **593**

M.1 Design of discrete time filters with the bilinear transform . . . 593

M.1.1 Design of Butterworth IIR filters . . . 594

M.2 Low passband sensitivity IIR filters . . . 596

M.2.1 Structural Boundedness . . . 596

M.2.2 Filter realisation as the sum of all-pass functions . . . 597

M.2.3 A note on the numerical calculation of the spectral factor . . . 599

M.2.4 Examples of parallel all-pass filter synthesis . . . 599

M.3 Design of Elliptic IIR filters with a reduced number of multipliers . . . 607

M.3.1 Elliptic filter design with the *Landen transformation* . . . 607

M.3.2 Design of elliptic filters with minimal-Q . . . 608

M.4 *Saramäki*'s method for the design of IIR filters with zeros on the unit circle . . . 613

M.4.1 Optimisation of a low-pass filter with denominator order higher . . . 614

M.4.2 Optimisation of a low-pass filter with denominator order lower . . . 616

M.4.3 Surma-aho and Saramäki method of unconstrained optimisation of an initial IIR filter . . . 619

M.5 Johansson and Saramäki design of all-pass complementary IIR filters . . . 625

**N Design of FIR digital filter transfer functions** **633**

N.1 Low passband sensitivity FIR lattice filters . . . 633

N.1.1 Lattice decomposition of an FIR digital filter . . . 633

N.1.2 Finite-wordlength properties of the lattice FIR filter . . . 634

N.1.3 State variable description of the complementary FIR lattice filter . . . 635

N.1.4 Design of the complementary FIR digital filter . . . 636

N.1.5 Example: the minium-phase complementary filter of an FIR bandpass filter . . . 641

N.1.6 Estimating the MA coefficients of a filtered noise sequence . . . 645

N.2 Design of an FIR lattice band-pass Hilbert filter . . . 650

N.3 Constrained least-squared error optimisation of FIR digital filters . . . 654

N.3.1 Examples of the design of constrained least-squared error symmetric FIR filters . . . 655

N.4 Constrained least-mean-squared error optimisation of FIR Hilbert digital filters . . . 660

N.4.1 Frequency response of a Hilbert filter . . . 660

N.4.2 Design of a least-mean-squared error even-order FIR Hilbert filter . . . 661

N.4.3 Design of a least-mean-squared error even-order FIR Hilbert band-pass filter . . . 662

N.5 Peak-constrained-least-squares design of non-symmetric FIR filters . . . 665

N.5.1 Frequency response and gradients of a non-symmetric FIR filter . . . 665

N.5.2 Examples of the PCLS design of non-symmetric FIR filters . . . 666

N.6 Constrained mini-max error optimisation of FIR digital filters . . . 670

N.6.1 The alternation theorem . . . 670

N.6.2 *Hofstetter*'s algorithm for mini-max FIR filter approximation . . . 671

N.6.3 *Parks-McClellan* algorithm for mini-max FIR filter approximation . . . 684

N.7 Design of FIR filters with maximally-linear pass-bands and equi-ripple stop-bands using the *Parks-McClellan* algorithm . . . 695

N.7.1 Design of FIR low-pass filters with maximally-flat pass-bands and equi-ripple stop-bands . . . 695

N.7.2 Design of FIR band-pass filters with maximally-flat pass-bands and equi-ripple stop-bands . . . 699

N.7.3 Design of maximally-linear FIR low-pass differentiators with equi-ripple stop-bands . . . 702

N.8 Closed-form design of maximally-linear FIR filters . . . 707

N.8.1 Closed-form design of maximally-flat low-pass FIR filters . . . 707

N.8.2 Closed-form design of maximally-flat FIR half-band filters . . . 716

N.8.3 Closed-form design of maximally-flat FIR Hilbert filters . . . 718

N.8.4 Closed-form design of maximally-linear FIR low-pass differentiators . . . 720

N.9 Linear Matrix Inequality(LMI) design of symmetric FIR filters . . . 725

N.9.1 The *Markov-Lukacs* theorem . . . 725

N.9.2 Trigonometric curves . . . 725

N.9.3 Moment matrix of trigonometric curves . . . 726

N.9.4 A *Markov-Lukacs* theorem for trigonometric curves . . . 726

N.9.5 The conic hull of $C_{a,b}$ . . . 727

N.9.6 Optimisation of the dual of a convex quadratic objective function . . . 728

N.9.7 Example of LMI design of a low pass symmetric FIR filter . . . 729

N.9.8 Example of LMI design of a band pass symmetric FIR filter . . . 731

N.10 Design of half-band FIR filters . . . 735

N.10.1 Vaidyanathan's "TRICK" for the design of FIR half-band filters . . . 735

N.10.2 Design of equi-ripple FIR half-band filters . . . 736

N.11 Design of equi-ripple FIR filters with Zolotarev polynomials . . . 745

N.11.1 The Zolotarev polynomials . . . 745

N.11.2 Narrow-band FIR filter design with the Zolotarev polynomials . . . 753

N.11.3 Almost equi-ripple low-pass FIR filter design with the Zolotarev polynomials . . . 764

N.12 Design of FIR filters as a tapped cascade of sub-filters . . . 772

N.12.1 Transformations of linear-phase FIR filters . . . 772

N.12.2 Frequency-domain constraints on the prototype and sub-filter . . . 773

N.12.3 Filter design . . . 775

N.12.4 Filter design examples . . . 777

**O Application of the *Kalman-Yakubovič-Popov* lemma to digital filter design** **783**

O.1 *Iwasaki* and *Hara*'s generalised KYP lemma . . . 783

O.1.1 The generalised S-procedure . . . 783

O.1.2 Frequency ranges as curves in the complex plane . . . 784

O.1.3 The generalised KYP lemma . . . 786

O.1.4 Examples of FIR filter design with the generalised KYP lemma . . . 787

O.2 Generalisation of the KYP lemma to a union of frequency intervals . . . 796

O.2.1 Union of frequency intervals . . . 796

O.2.2 Generalised KYP lemma over a union of frequency intervals . . . 797

O.2.3 Example of design of a non-symmetric band pass FIR filter with the union of the upper and lower stop bands . . . 798

**Colophon** **803**

**Bibliography** **819**

# List of Algorithms


1.1 Procedure for reordering the nodes of a primitive signal flow graph . . . 28
1.2 Derivation of the state variable description of a signal flow graph . . . 29
1.3 Transformation of a transfer function to a state variable description . . . 33
1.4 Continued fraction expansion of a rational transfer function . . . 34
1.5 Transformation of state variable description to direct form . . . 36
1.6 Leverrier's algorithm for finding the resolvent of the state transition matrix, A [168, Algorithm 8A.12] . . . 37
1.7 *La Budde*'s algorithm for finding the characteristic polynomial of A [173, Algorithm 2] . . . 38
3.1 Recursive calculation of the covariance matrix . . . 57
3.2 Minimisation of the noise gain . . . 61
3.3 Optimisation of the noise gain . . . 62
4.1 Construction of optimised second order state variable filters[168, Figure 9.14.1]. . . . 66
4.2 Bomar second order optimised state variable filter sections [24, Equation 17]. (See also [168, Figure 9.12.1]) . . . 67
4.3 Bomar Type III second order optimised state variable filter sections [24, Equation 23] . . . 67
5.1 Schur polynomial expansion (see *Parhi* [98, Section 12.2.3]) . . . 98
5.2 One-multiplier lattice sign assignment [4, Page 496] . . . 101
5.3 Construction of a state variable description of the Schur FIR lattice filter . . . 106
5.4 Construction of a state variable description of the Schur one-multiplier lattice filter . . . 107
5.5 Construction of a state variable description of the retimed Schur one-multiplier lattice filter . . . 109
5.6 Construction of a state variable description of the Scaled-Normalised Lattice . . . 111
7.1 Minimisation of round off noise in error feedback/feedforward state variable filters. (See *Williamson* [40, Theorem 5.2]). . . . 141
8.1 Exchange algorithm for multiband FIR filters. (*Selesnick, Lang and Burrus*. See [76, p.498]). . . . 148
8.2 Compute the powers of a matrix (see *Golub and van Loan*, Algorithm 11.2.2) . . . 151
11.1 Conversion of 2's complement numbers to the canonical signed-digit representation (*Parhi* [98, Section 13.6.1]). 363
11.2 Modified signed-digit allocation heuristic of *Ito et al.* [178] . . . 364
D.1 Carlson's algorithm for computing the $R_F$ function [15, Algorithm 1] . . . 535
D.2 Carlson's algorithm for computing the $R_C$ function [15, Algorithm 2] . . . 535
D.3 Carlson's algorithm for computing the $R_J$ function [15, Algorithm 3] . . . 535
D.4 Carlson's algorithm for computing the $R_D$ function [15, Algorithm 4] . . . 536
E.1 *Lanczos* tridiagonalisation of an unsymmetric matrix . . . 543
K.1 The sequential quadratic programming method . . . 572
K.2 Cholesky factorisation of an $n \times n$ positive-definite symmetric matrix, $W$ . . . 574
K.3 Line search using the Armijo rule . . . 580
K.4 Line search using Wolfe conditions . . . 581
K.5 Line search using the Goldstein conditions . . . 581
K.6 The *Goldfarb-Idnani* dual algorithm [33] . . . 583
M.1 Filter realisation as the sum of all-pass functions . . . 599
M.2 *Surma-aho* and *Saramäki* method for finding an initial IIR filter in *gain-pole-zero* form [96, pp. 958-959] . . . 619
N.1 Construction of a state variable description of the complementary FIR lattice filter . . . 635
N.2 Parks-McClellan FIR filter design [143, Section 7.4.3] [206, 78] . . . 685
N.3 *Vlček et al.*'s backwards recursion for the calculation of the impulse response of an odd-length, symmetric, maximally-flat, low-pass FIR filter [129, Table 1]. . . . 714
N.4 *Zahradník* and *Vlček*'s algorithm [155, Table 1] for the evaluation of the coefficients, $a_l$, of $\mathcal{U}_n$. . . . 740
N.5 *Vlček* and *Unbehauen*'s backwards recursion for $Z_{p,q}(w) = \sum_{m=0}^{n} b_m w^m(w)$ [127, Table IV], [128] . . . 748
N.6 Modified *Vlček* and *Unbehauen* backwards recursion for $Z_{p,q}(w) = \sum_{m=0}^{n} a_m T_m(w)$ [127, Table V], [128] . . . 749
N.7 Recursion of *Zahradník, Šusta et al.* for the calculation of the scaled coefficients of the expansion, in Chebyshev polynomials of the first kind, of the product of the $n$ factors, $(w - w_m)$, of a polynomial [157, Table I] . . . 760
N.8 *Vlček* and *Zahradník*'s algorithm for the evaluation of the impulse response, $h(m)$, of an "almost equi-ripple" low-pass FIR filter [132, Tables 4 and 5] . . . 765

# Introduction

An IIR filter can approximate a desired amplitude response with fewer coefficients than an FIR filter[a]. The design of IIR filters is more difficult than the design of FIR filters. FIR filters are inherently stable. An FIR filter design problem can be formulated as a convex optimisation problem with a global solution (see, for example, *Wu et al.* [186]). 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 must be constrained to ensure that the IIR filter is stable. This report describes my experiments in the design of IIR digital filters with constraints on the amplitude, phase and group delay responses. I intended to show that it is possible to design an "acceptable" or "good-enough" IIR digital filter with coefficients that use a limited number of shift-and-add operations and so do not require software or hardware multiplications.

My experiments are programmed in the *Octave* language [91][b]. Octave is an "almost" compatible open-source-software clone of the commercial *MATLAB* package [208]. 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* [193]. 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, 116] or by constraining the reflection coefficients of a tapped all-pass lattice filter implementation [4, 98]. A valid initial solution for the MMSE solver is found by "eye" or by using the WISE method of *Tarczynski et al.* [10]. A peak-constrained-least-squares (PCLS) solution is found by the exchange algorithm of *Selesnick et al.* [76]. 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.* [224] or that of *Ito et al.* [178] and branch-and-bound or relaxation search is used to find an acceptable response. For lesser coefficient word-lengths simulated-annealing of the signed-digit rounded coefficients 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 5 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* [168] and chapter 12 of *"VLSI Digital Signal Processing Systems:Design and Implementation"* by *Parhi* [98].

**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:

$$\begin{aligned} \textbf{minimise} \quad & \mathcal{E}_H(x) = \int W(\omega) \left| H(x,\omega) - H_d(\omega) \right|^2 d\omega \\ \textbf{subject to} \quad & H \text{ is stable} \end{aligned} \tag{1}$$

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:

$$\begin{aligned} \textbf{minimise} \quad & \max\ W(\omega) \left| H(x,\omega) - H_d(\omega) \right| \\ \textbf{subject to} \quad & H \text{ is stable} \end{aligned} \tag{2}$$

[a] Section 9.6 compares the frequency responses of FIR and IIR implementations of a low-pass filter with approximately flat pass-band group delay.
[b] The Colophon describes my local Octave build.

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 IIR filter response are applied by the *Peak-Constrained-Least-Squares* (PCLS) exchange algorithm of *Selesnick*, *Lang* and *Burrus* [76]. Appendix N.3 describes the use of the PCLS method of *Selesnick et al.* to design symmetric FIR filters with constraints on the filter amplitude response.

Chapter 8 applies the *Sequential Quadratic Programming* (SQP) method to the optimisation of the frequency of an IIR filter. The SQP method is reviewed in Appendix K. That review makes extensive use of the books by *Nocedal* and *Wright* [92], *Ruszczynski* [9] and *Bertsekas* [36]. I chose to write my own SQP solver in Octave. In Chapter 8, I follow *Deczky* [3] and *Richards* [116] 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 G, 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 9 is based on the description of IIR filter design using *Second-Order-Cone-Programming* (SOCP) by *Lu* and *Hinamoto* [212]. SOCP is a subclass of *convex programming* [181]. See the review articles by *Alizadeh* and *Goldfarb* [45] and *Lobo et al.* [125] 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* [193].

Chapter 10 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.* [145] and *Vaidyanathan et al.* [152] 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* [123, 124], describe IIR digital filterbanks as a "polyphase" combination of all-pass digital filters. *Gray* and *Markel* [4] and *Parhi* [98, 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* [64], *Milić* and *Ćertić* [104] and *Lu and Hinamoto* [212] describe efficient, sharp transition-band IIR filter designs using the *frequency masking* approach described for FIR filters by *Lim* [222].

**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.* [224] and *Ito et al.* [178] 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. Part III considers methods of searching for the best filter response with truncated filter coefficients. Figure 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.* [178]. After successive SQP-relaxation optimisations of the signed-digit coefficient values, 63 signed-digits and 33 shift-and-add operations are required to implement the coefficient multiplications. Chapter 17, 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* [7], and the *simplex*, *differential evolution* and *simulated annealing* routines from the Octave-Forge *optim* package [137].

**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* [56] and *"The perfidious polynomial"* by *Wilkinson* [79]. 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 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 (bincoeff (20,0:20))`. In fact the roots are all $-1$. Figure 3 repeats the example with the oct-file *qzsolve*, an implementation using 128-bit floating-point numbers based on *gsl_poly_complex_solve* from the GNU GSL library[163][c].

**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. YOUR SYSTEM WILL ALMOST CERTAINLY BE DIFFERENT. YOU MAY NEED TO MODIFY A SCRIPT TO RUN ON YOUR SYSTEM.**

[c] The *MPSolve* multi-precision polynomial solver [32] finds the expected roots. The package includes an Octave interface.

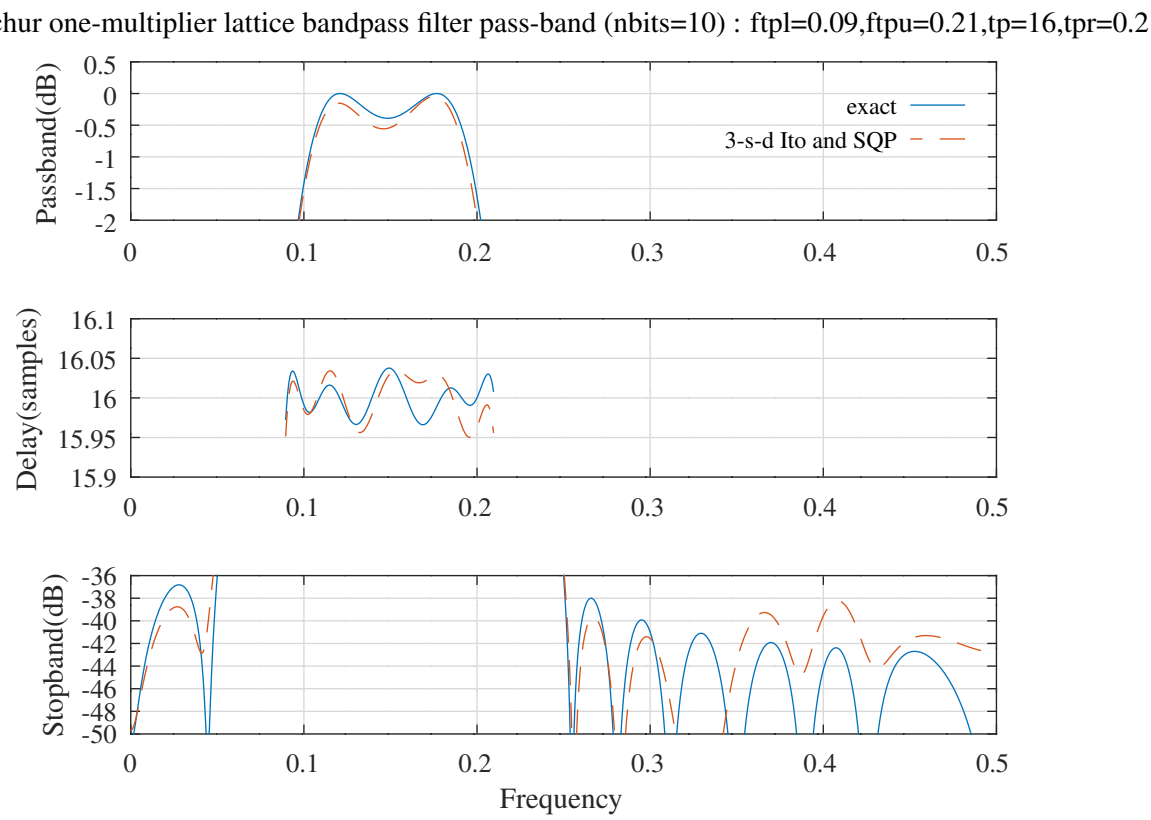



Figure 1: Comparison of the pass-band and stop-band amplitude responses and group delay responses for a Schur one-multiplier lattice bandpass filter with exact coefficients and with 10 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation optimisation.

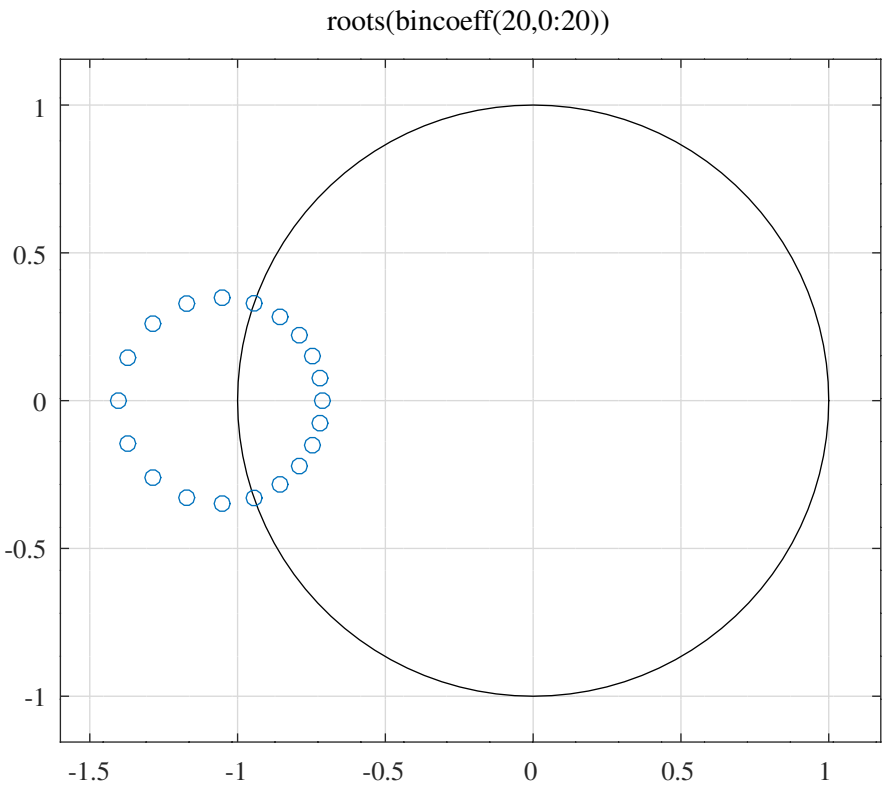



Figure 2: Plot of the roots of the binomial polynomial of order 20 calculated by the Octave *roots* function

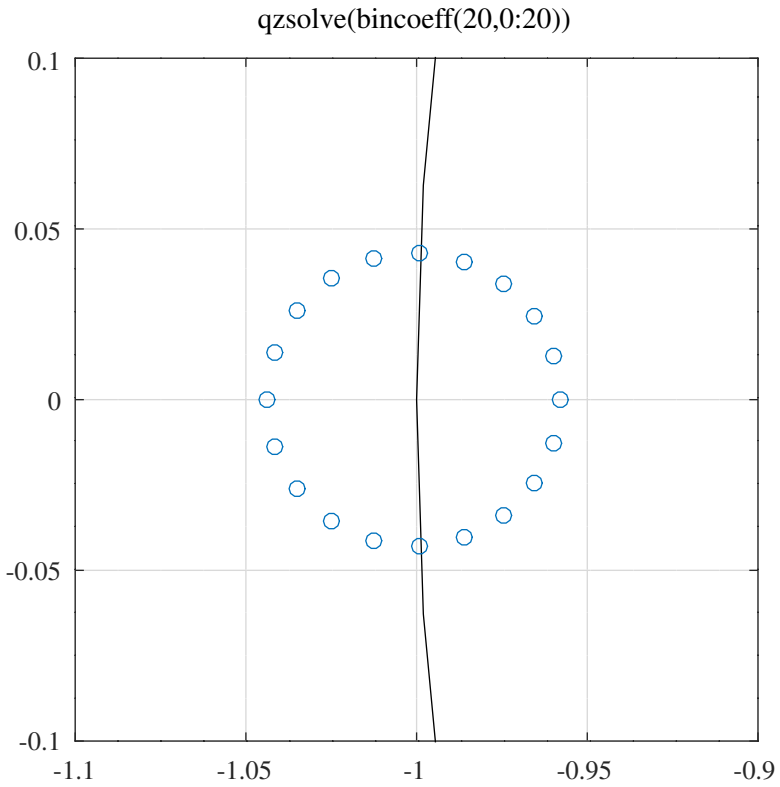



Figure 3: Plot of the roots of the binomial polynomial of order 20 calculated by the *qzsolve* oct-file function using 128-bit floating point numbers

# Part I

# State Variable description of digital filters

# Chapter 1

# 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* [168]. Appendix A reviews the necessary complex variables theory.

## 1.1 The z-transform

The bi-lateral *z-transform* of a sequence $\boldsymbol{f}$ is

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

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

$$\sum_{k} \left| f(k) z^{-k} \right| < \infty$$

If $\boldsymbol{f}$ 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

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

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

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

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* [168, Section 3.4], let $\boldsymbol{f}$ and $\boldsymbol{g}$ be causal sequences with z-transforms $F(z)$ and $G(z)$, respectively. Define the z-transform of the product as

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

Substituting the inversion integral for $\boldsymbol{f}$

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

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

$$Z\left\{\boldsymbol{f}\centerdot\boldsymbol{g}\right\} = \frac{1}{2\pi\imath}\oint_C s^{-1}F\left(s\right)G\left(\frac{1}{s}\right)ds$$

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

$$\begin{aligned} Z\left\{\boldsymbol{f}\centerdot\boldsymbol{f}\right\} &= \sum_{k=0}^{\infty}f^2\left(k\right)z^{-k} \\ &= \frac{1}{2\pi\imath}\oint_C z^{-1}F\left(z\right)F\left(z^{-1}\right)dz \end{aligned}$$

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

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

## 1.2 Filter difference equation

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

$$\sum_{l=0}^{n}a_l y\left(k-l\right) = \sum_{l=0}^{m}b_l u\left(k-l\right), \quad a_0 = 1 \text{ and } k \geq 0 \tag{1.1}$$

If $\star$ represents the convolution operation and $u\left(k\right)$ and $y\left(k\right)$ are the input and output sequences, respectively, then the difference equation is:

$$\boldsymbol{a}\star\boldsymbol{y} = \boldsymbol{b}\star\boldsymbol{u}$$

If $\boldsymbol{y^0}$ is a solution of the homogenous equation (HE), $\boldsymbol{y}\star\boldsymbol{a} = 0$, then

$$\boldsymbol{a}\star\left(\boldsymbol{y}+\boldsymbol{y^0}\right) = \boldsymbol{b}\star\boldsymbol{u}$$

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

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

and if $\lambda$ is a root of $a\left(z\right)$ then

$$\sum_{l=0}^{n}a_l\lambda^{k-l} = \lambda^k a\left(\lambda\right) = 0 \tag{1.2}$$

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

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

The case of repeated roots can be treated by differentiating Equation 1.2. 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, $\boldsymbol{h}$, is found by setting the input $\boldsymbol{u} = \boldsymbol{\delta}$

$$\sum_{l=0}^{n} a_l h(k-l) = \sum_{l=0}^{n} b_l \delta(k-l), \quad a_0 = 1 \text{ and } k \geq 0 \tag{1.3}$$

Further, if $\boldsymbol{h}$ 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 $h(1), \ldots, h(n)$ that determine the $c_l$ are obtained by direct evaluation of Equation 1.3.

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, $\boldsymbol{h}$ 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 $|\lambda_l| < 1$, $l = 1, \ldots, n$.

## 1.3 Filter transfer function

The filter transfer function is related to the z-transform of the filter difference equation, Equation 1.1, 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

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

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.

## 1.4 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

**Algorithm 1.1** 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 1.1
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

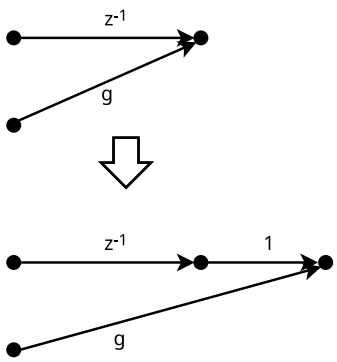



Figure 1.1: Isolating a unit delay from an incoming branch

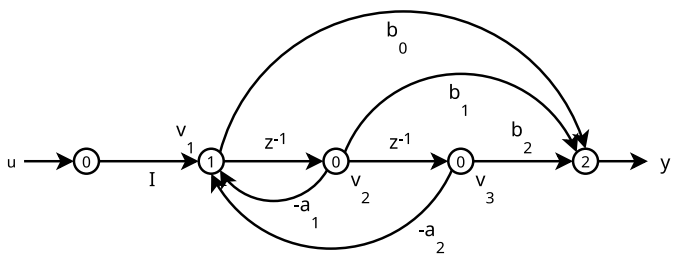



Figure 1.2: Second order direct form filter

**Algorithm 1.2** Derivation of the state variable description of a signal flow graph

1. Replace each unit delay with the equivalent path shown in Figure 1.4
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

3. There are a finite number of nodes and branches

The implementation of primitive signal flow graphs requires *node reordering*, shown in Algorithm 1.1. For example Figure 1.2 shows the signal flow graph of a second order *direct form* filter. The nodes are labelled according to Algorithm 1.1. The labelled nodes are:

- $S_0$: $u, v_2, v_3$
- $S_1$: $v_1 = u - a_1 v_2 - a_2 v_3$
- $S_2$: $y = b_0 v_1 + b_1 v_2 + b_2 v_3$

With updates

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

Then in the $z$-domain

$$\begin{aligned} y(z) &= b_0 z^2 + b_1 z^1 + b_2 \\ u(z) &= z^2 + a_1 z + a_0 \end{aligned}$$

so the transfer function is

$$H(z) = \frac{b_0 z^2 + b_1 z + b_2}{z^2 + a_1 z + a_0}$$

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

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

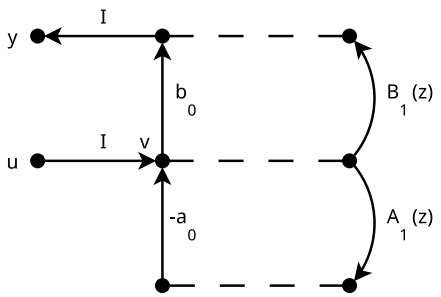
where $A_{n+1}(z) = B_{n+1}(z) = 0$. The filter structure recursion is initialised with:

$$\begin{aligned} y(z) &= b_0 v + B_1(z) v \\ u(z) &= v + a_0 A_1(z) v \end{aligned}$$

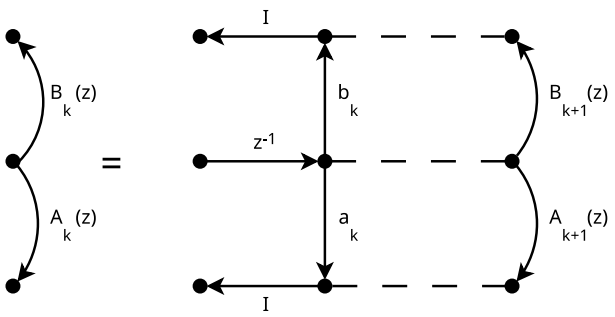
so that

$$H(z) = \frac{b_0 + B_1(z)}{1 + a_0 A_1(z)}$$

See Figure 1.3.




(a) Left hand side termination




(b) $k_{th}$ internal section

Figure 1.3: Recursive generation of direct form filters

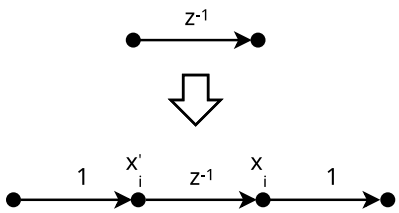



Figure 1.4: Unit delay equivalent

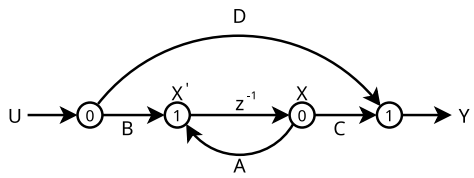



Figure 1.5: Signal flow graph of a state variable filter

## 1.5 State variable description of a signal flow graph

The state variable description of a signal flow graph is derived by Algorithm 1.2.

Figure 1.2 shows the signal flow graph for a second order direct form filter.

The time domain state variable equations for this filter are:

$$\begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \tag{1.4}$$

where:

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

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

$$\begin{bmatrix} zX \\ Y \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} X \\ U \end{bmatrix} \tag{1.5}$$

Figure 1.5 shows the signal flow graph for the state variable filter of Equation 1.5. The matrix $A$ is called the *state transition matrix*.

If the arrows in Figure 1.5 are reversed and the inputs to each node become the outputs then the state-variable description of the *transposed* signal flow graph is:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^\top = \begin{bmatrix} A^\top & C^\top \\ B^\top & D \end{bmatrix}^\top \tag{1.6}$$

## 1.6 Controllability

The matrix pair $(A, B)$ is said to be *controllable* if and only if $\det \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix} \neq 0$.

The n-by-n matrix $\begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix}$ 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

$$x(n) = \sum_{l=1}^{n} A^{l-1} B u(n-l)$$

$$= \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix} \begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(0) \end{bmatrix}$$

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

## 1.7 Observability

The matrix pair $(A, C)$ is said to be *observable* if the matrix $\begin{bmatrix} C & CA & \cdots & CA^{n-1} \end{bmatrix}$ 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.

## 1.8 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:

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

## 1.9 State variable descriptions and the transfer function

From the $z$-domain state variable matrix shown in Equation 1.5

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

so

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

and the transfer function is

$$H(z) = D + C(zI - A)^{-1} B \tag{1.7}$$

The transfer function of the *transpose* of the signal flow diagram shown in 1.5 is the same as that of the original diagram.

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

$$\begin{aligned} H(z) &= D + C(zI - A)^{-1} B && (1.8) \\ &= D + CTT^{-1}(zI - A)^{-1} \left(T^{-1}\right)^{-1} T^{-1}B && (1.9) \\ &= D + CT\left[T^{-1}(zI - A)T\right]^{-1} T^{-1}B && (1.10) \\ &= D + CT\left(zI - T^{-1}AT\right)^{-1} T^{-1}B && (1.11) \end{aligned}$$

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

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 $\begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix}$ 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

$$H(z) = \frac{b_0 z^n + b_1 z^{n-1} + \cdots + b_{n-1} z + b_n}{z^n + a_1 z^{n-1} + \ldots a_{n-1} z + a_n} \tag{1.12}$$

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

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

### 1.9.1 Transformation of a transfer function to a state variable description

Algorithm 1.3 converts a rational transfer function to the equivalent direct form state variable description[b]. An *Octave* implementation is shown in the file *tf2Abcd.m*.

**Algorithm 1.3** 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

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

PROOF of Algorithm 1.3: Firstly, show the identity

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

with $\Psi(z) = \begin{bmatrix} 1 & z & \cdots & z^{n-1} \end{bmatrix}^\top$ uniquely defines the matrix:

$$A = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{22} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n,1} & \alpha_{n,2} & \cdots & \alpha_{n,n} \end{bmatrix}$$

[b] Appendix 1.9.2 shows the continued fraction expansion of a rational transfer function into state variable form.

**Algorithm 1.4** 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 do** $k = 1, \ldots, N$
  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**

and the column vector: $B = \begin{bmatrix} \beta_1 & \cdots & \beta_n \end{bmatrix}^\top$. For row $i$:

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

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 $C = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}$ and scalar $D = \delta$ that satisfy

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

Expanding

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

Equating coefficients

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

### 1.9.2 Transformation of a transfer function to a state variable description by continued fraction expansion

See *Roberts* and *Mullis* [168, Problems 8.16, 10.13 to 10.16] and *Mitra* and *Sherwood* [185]. Algorithm 1.4 calculates the continued fraction expansion of a rational transfer function.

The continued fraction expansion of a state variable filter is shown in Figure 1.6. Figure 1.6a shows the signal flow graph for a first order approximation to a state variable filter. The state variable equations for Figure 1.6a are:

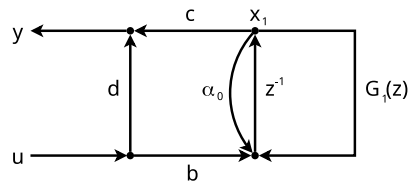
$$\begin{aligned} zX_1 &= \alpha_0 X_1 + bU + G_1 X_1 \\ Y &= cX_1 + dU \end{aligned}$$

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

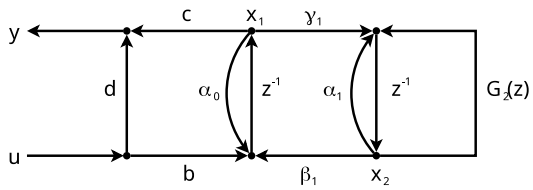
$$G(z) = d + \frac{bc}{z - \alpha_0 - G_1(z)}$$

Figure 1.6b shows the addition of a second section for $G_1(z)$ with:

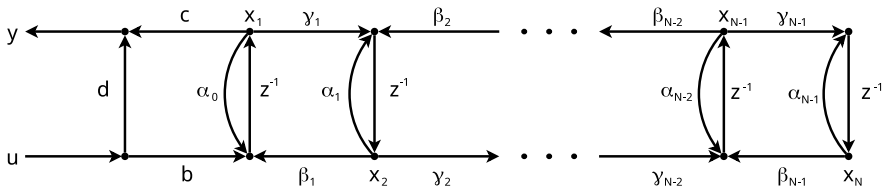
$$zX_2 = \alpha_1 X_2 + \gamma_1 X_1 + bU + G_2 X_2$$




(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 1.6: Continued fraction expansion of a state variable filter

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

Finally, in Figure 1.6c, 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 1.6c are:

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

or:

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

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* [49, Section 9.4.3] describe *unsymmetric Lanczos tridiagonalisation* of an arbitrary matrix, summarised in Appendix E. The Octave function *lanczos_tridiag* implements *Lanczos* tridiagonalisation.

The Octave function *contfrac* implements Algorithm 1.4 and returns the equivalent state variable description. The Octave script *contfrac_test.m* implements the continued fraction expansion of a 5th order elliptic low-pass filter with cutoff frequency $0.05 f_S$. 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 $583e3$ respectively[c].

### 1.9.3 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 1.5 uses the controllability matrix to find the characteristic polynomial of the state transition matrix, $A$. The transfer function follows from Equation 1.7.

**Algorithm 1.5** Transformation of state variable description to direct form

If

1. the matrix pair $(A, B)$ is controllable so that $\det \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \neq 0$.
2. the characteristic function of the state transition matrix, $A$, is $\det(zI - A) = a_0 z^n + a_1 z^{n-1} + \ldots + a_n$ 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 $T = \begin{bmatrix} x(n) & x(n-1) & \cdots & x(1) \end{bmatrix}$ 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 1.5: By repeated application, the state vectors, $x$, at time $k+1$ are:

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

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

$$T = \begin{bmatrix} x(n) & x(n-1) & \cdots & x(1) \end{bmatrix}$$

[c] See Chapter 3.

$$= \begin{bmatrix} A^{n-1}B & A^{n-2}B & \cdots & B \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ a_1 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ a_{n-1} & a_{n-2} & \cdots & 1 \end{bmatrix}$$

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 $z^n + a_1 z^{n-1} + \cdots + a_n$ 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 $\{q(n+1), \ldots, q(2)\}$:

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

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 [168, Algorithm 8A.12] shown in Algorithm 1.6. Note that Leverrier's algorithm calculates the characteristic polynomial

**Algorithm 1.6** Leverrier's algorithm for finding the resolvent of the state transition matrix, A [168, 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** $k = 1, \ldots, n$ **do**
  $a_k = -\frac{1}{k} \operatorname{trace}\left(A\bar{A}_{k-1}\right)$
  $\bar{A}_k = A\bar{A}_{k-1} + a_k I$
**end for**

of $A$ as a by-product. In this case the characteristic polynomial of the state-transition matrix is the denominator polynomial of

**Algorithm 1.7** *La Budde*'s algorithm for finding the characteristic polynomial of A [173, 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 $a_1 \ldots a_n$:

$a_1^{(1)} = -\alpha_1$
$a_1^{(2)} = a_1^{(1)} - \alpha_2$
$a_2^{(2)} = \alpha_1\alpha_2 - h_{1,2}\beta_2$
**for** $k = 3, \ldots, n$ **do**
  $a_1^{(k)} = a_1^{(k-1)} - \alpha_k$
  **for** $l = 2, \ldots, k-1$ **do**
    $a_l^{(k)} = a_l^{(k-1)} - \alpha_k a_{l-1}^{(k-1)} - \sum_{m=1}^{l-2} h_{k-m,k}\beta_k \cdots \beta_{k-m+1} a_{l-m-1}^{(k-m-1)} - h_{k-l+1,k}\beta_k \cdots \beta_{k-l+2}$
  **end for**
  $a_k^{(k)} = -\alpha_k a_{k-1}^{(k-1)} - \sum_{m=1}^{k-2} h_{k-m,k}\beta_k \cdots \beta_{k-m+1} a_{k-m-1}^{(k-m-1)} - h_{1,k}\beta_k \cdots \beta_2$
**end for**
$a_k = a_k^{(n)},\ 1 \le k \le n$

the transfer function, which is known. The recursion in $\bar{A}_k$ is justified by rearranging the expansion of the resolvent shown in Algorithm 1.6 and then equating coefficients of $z$:

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

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

*Wilkinson* [80, Section 57, Chapter 6], *Rehman* [180, Chapter 6], and *Rehman* and *Ipsen* [173] describe *La Budde*'s algorithm for the calculation of the characteristic polynomial of an *upper Hessenberg* matrix, reproduced here as Algorithm 1.7 [173, Algorithm 2]. *Rehman* [180, Appendix B] provides a *MATLAB* implementation, *labudde.m*[d]. *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 $\lambda_k$ 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.

The first stage of *La Budde*'s method reduces the non-symmetric matrix, $A$, to *upper Hessenberg* form, $H$ (see *Golub* and *van Loan* [49, 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* [173, 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 $p_k(z) = \det(zI - H_k),\ 1 \le k \le n$, where $p(z) = p_n(z)$. The recursion for computing $p(z)$ is [80, Section 6.57.1]
>
> $$\begin{aligned} p_0(z) &= 1 \\ p_1(z) &= z - \alpha_1 \\ p_k(z) &= (z - \alpha_1)p_{k-1}(z) - \sum_{m=1}^{k-1} h_{k-m,k}\beta_k \cdots \beta_{k-m+1} p_{k-m-1}(z) \end{aligned} \tag{1.13}$$
>
> where $2 \le k \le n$. 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 1.13 gives recursions for individual coefficients $a_k$.

[d] I have edited *labudde.m* to make it compatible with Octave

### 1.9.4 Sensitivity of the state variable description of a transfer function

*Thiele* [108] analyses the sensitivity of the frequency response of linear state variable digital filters with respect to the coefficients. Differentiation of the resolvent, $R = (zI - A)^{-1}$, is simplified by the matrix identity:

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

where $\rho$ represents the components of the matrix $R$. With this identity, the gradients of $H(z)$ are found by differentiating Equation 1.7:

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

where $\alpha$, $\beta$, $\gamma$ and $\delta$ represent the components of $A$, $B$, $C$ and $D$ respectively.

## 1.10 Time domain description

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

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

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

$$|\lambda_k| < 1\,,\ k = 1, \ldots, n$$

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

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

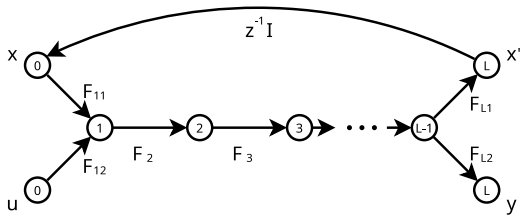
## 1.11 Unit Pulse Response

The input and output sequences are related by the convolution, $y = h \star u$. The state variable form is:

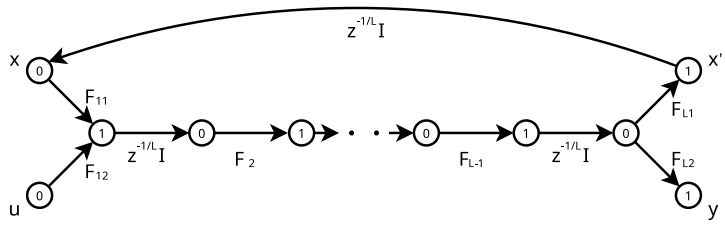
$$\begin{aligned} y(k) &= Cx(k) + Du(k) \\ &= \sum_{l=1}^{\infty} CA^{l-1}Bu(k-l) + Du(k) \\ &= \sum_{l=-\infty}^{\infty} h(l)\,u(k-l) \end{aligned}$$

For these to agree

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




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




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

Figure 1.7: Factored state variable filters (See [168, Figure 8.4.6].)

## 1.12 Factored state variable descriptions

See *Roberts* and *Mullis* [168, Section 8.4]. Equation 1.4 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

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

### 1.12.1 Factored state variable filters with fractional delays

Figure 1.7a shows the signal flow graph of a factored state variable filter with $L$ factors. The longest delay free path has length $L$. The factors $F_1$ and $F_L$ are:

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

Figure 1.7b 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.

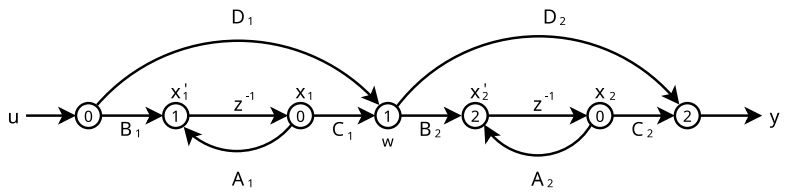



Figure 1.8: Signal flow graph of two cascaded state variable filters

### 1.12.2 Construction of the factored state variable description

To construct a FSVD from a PSFG, first find the sets $\{S_0, S_1, \ldots, S_L\}$ 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

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

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:

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

#### Factoring cascaded state variable filters

As an example, Figure 1.8 shows two state variable filters connected in cascade.

The node variable sets are:

$$\begin{aligned} S_0 &= \{x_1, x_2, u\} \\ S_1 &= \{w, x'_1\} \\ S_2 &= \{y, x'_2\} \end{aligned}$$

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:

$$\begin{aligned} \begin{bmatrix} x_1' \\ x_2' \\ y \end{bmatrix} &= \begin{bmatrix} 0 & I & 0 \\ B_2 & 0 & A_2 \\ D_2 & 0 & C_2 \end{bmatrix} \begin{bmatrix} C_1 & 0 & D_1 \\ A_1 & 0 & B_1 \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u \end{bmatrix} \\ &= \begin{bmatrix} A_1 & 0 & B_1 \\ B_2C_1 & A_2 & B_2D_1 \\ D_2C_1 & C_2 & D_2D_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u \end{bmatrix} \end{aligned}$$

**Factoring a feedback connection of state variable filters**

*Roberts* and *Mullis* [168, Problem 8.9] show a feedback connection of two MIMO state-variable systems:

$$x_1' = A_1x_1 + \begin{bmatrix} B_1 & B_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} x_1 + \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

and

$$x_2' = A_2x_2 + \begin{bmatrix} b_1 & b_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$
$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} x_2 + \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

The feedback connections are $u_2 = v_1$ and $w_1 = y_2$. A delay-free loop is avoided by setting $D_{22} = 0$ or $d_{11} = 0$. Choosing the latter, the state equations are, after substitution:

$$\begin{aligned} x_1' &= A_1x_1 + B_1u_1 + B_2v_1 \\ y_1 &= C_1x_1 + D_{11}u_1 + D_{12}v_1 \\ y_2 &= C_2x_1 + D_{21}u_1 + D_{22}v_1 \\ x_2' &= A_2x_2 + b_1y_2 + b_2w_2 \\ v_1 &= c_1x_2 + d_{12}w_2 \\ v_2 &= c_2x_2 + d_{21}y_2 + d_{22}w_2 \end{aligned}$$

The node variable sets are:

$$\begin{aligned} S_0 &= \{x_1, x_2, u_1, w_2\} \\ S_1 &= \{v_1\} \\ S_2 &= \{x_1', y_1, y_2\} \end{aligned}$$

$$S_3 = \{x_2', v_2\}$$

where:

$$\begin{bmatrix} v_1 \\ x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 & c_1 & 0 & d_{12} \\ I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix}$$

$$\begin{bmatrix} x_1' \\ y_1 \\ y_2 \\ v_1 \\ x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} B_2 & A_1 & 0 & B_1 & 0 \\ D_{12} & C_1 & 0 & D_{11} & 0 \\ D_{22} & C_2 & 0 & D_{21} & 0 \\ I & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} v_1 \\ x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix}$$

$$\begin{bmatrix} x_2' \\ v_2 \\ x_1' \\ y_1 \\ y_2 \\ v_1 \\ x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & b_1 & 0 & 0 & A_2 & 0 & b_2 \\ 0 & 0 & d_{21} & 0 & 0 & c_2 & 0 & d_{22} \\ I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} x_1' \\ y_1 \\ y_2 \\ v_1 \\ x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix}$$

$$\begin{bmatrix} x_1' \\ x_2' \\ y_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_2' \\ v_2 \\ x_1' \\ y_1 \\ y_2 \\ v_1 \\ x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix}$$

The factored state variable description is:

$$\begin{aligned}
\begin{bmatrix} x_1' \\ x_2' \\ y_1 \\ v_2 \end{bmatrix} &= \begin{bmatrix} 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & b_1 & 0 & 0 & A_2 & 0 & b_2 \\ 0 & 0 & d_{21} & 0 & 0 & c_2 & 0 & d_{22} \\ I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} x_1' \\ y_1 \\ y_2 \\ v_1 \\ x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix} \\
&= \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & 0 & b_1 & A_2 & b_2 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & d_{21} & c_2 & d_{22} \end{bmatrix} \begin{bmatrix} B_2 & A_1 & 0 & B_1 & 0 \\ D_{12} & C_1 & 0 & D_{11} & 0 \\ D_{22} & C_2 & 0 & D_{21} & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} 0 & c_1 & 0 & d_{12} \\ I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix} \\
&= \begin{bmatrix} A_1 & B_2c_1 & B_1 & B_2d_{12} \\ b_1C_2 & b_1D_{22}c_1 + A_2 & b_1D_{21} & b_1D_{22}d_{12} + b_2 \\ C_1 & D_{12}c_1 & D_{11} & D_{12}d_{12} \\ d_{21}C_2 & d_{21}D_{22}c_1 + c_2 & d_{21}D_{21} & d_{21}D_{22}d_{12} + d_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix}
\end{aligned}$$

## 1.13 Block processing and decimation filters

See *Roberts* and *Mullis* [168, 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:

$$\begin{aligned} A' &= A^P \\ B' &= \begin{bmatrix} A^{P-1}B & A^{P-2}B & \cdots & B \end{bmatrix} \\ C' &= \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{P-1} \end{bmatrix} \\ D' &= \begin{bmatrix} D & 0 & 0 & \cdots & 0 \\ CB & D & 0 & \cdots & 0 \\ CAB & CB & D & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{P-2}B & \cdots & \cdots & CB & D \end{bmatrix} \end{aligned}$$

For example, if the block length is $P = 2$:

$$\begin{bmatrix} x(k+1) \\ y(k) \\ u(k+1) \end{bmatrix} = \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \\ u(k+1) \end{bmatrix}, \text{ first update} \tag{1.14}$$

$$\begin{bmatrix} x(k+2) \\ y(k) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ C & 0 & D \end{bmatrix} \begin{bmatrix} x(k+1) \\ y(k) \\ u(k+1) \end{bmatrix}, \text{ second update} \tag{1.15}$$

The factored state variable description of the block-length 2 filter is:

$$\begin{bmatrix} x(k+2) \\ y(k) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ C & 0 & D \end{bmatrix} \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \\ u(k+1) \end{bmatrix} = \begin{bmatrix} A^2 & AB & B \\ C & D & 0 \\ CA & CB & D \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \\ u(k+1) \end{bmatrix}$$

*Roberts* and *Mullis* [168, 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 1.1. If the filter is decimating then only a single output, $y(k)$, needs to be calculated in each block and the number of multipliers-per-output is reduced accordingly.

| Filter structure | Number of multipliers per output | Optimal block length, P |
|---|---|---|
| Full state variable 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 1.1: The number of multipliers for three classes of $N$-th order block processing filters. (See [168, Table 10.2.1]).

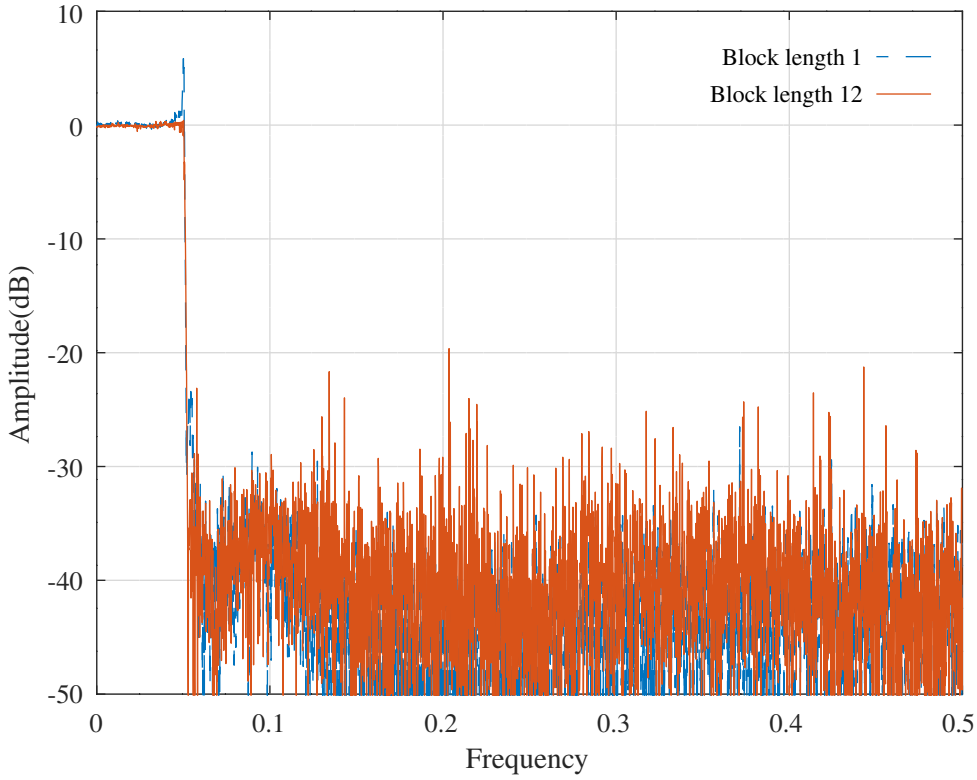



Figure 1.9: Simulated response of an 8-th order elliptic filter with 8-bit precision coefficients for block lengths of 1 and 12. The block length 12 filter is implemented as an *oct*-file.

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 MIMO state variable filter that can be applied to block filtering by rearranging the input and output matrixes appropriately. The Octave function *Abcd2cc* generates an *oct*-file implementation of a block processing state variable filter. The Octave script *Abcd2cc_test.m* tests the *oct*-file implementation of an 8-th order elliptic filter with block length 12 and 8 bit coefficients. Figure 1.9 shows the overall simulated response and Figure 1.10 compares the passband responses. The pass-band response of the block length 12 filter is more accurate than that of the block length 1 filter. The noise gain[e] 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 1.1, the number of multipliers per output is 81 for the block length 1 filter and 27.83 for the block length 12 filter.

[e] See Chapter 3.

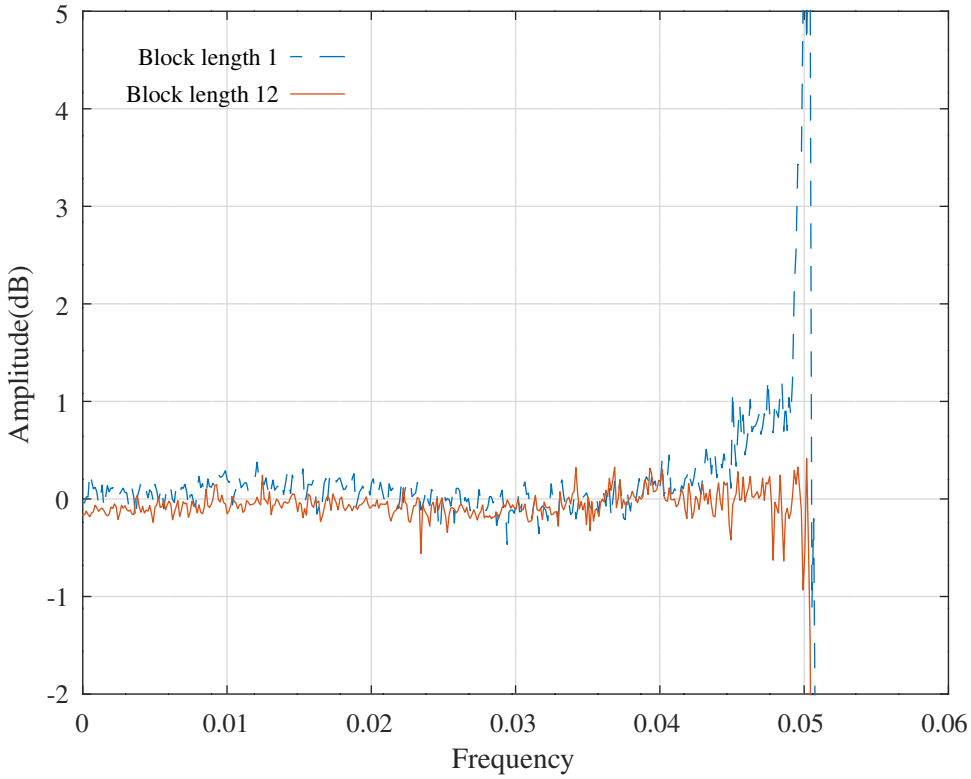



Figure 1.10: Simulated passband response of a 8-th order elliptic filter with 8-bit precision coefficients for block lengths of 1 and 12. The block length 12 filter is implemented as an *oct*-file.

# Chapter 2

# Frequency transformations of state variable filters

## 2.1 Frequency Transformations

See *Roberts* and *Mullis* [168, Section 6.7]. A *generalised band-pass* filter has a frequency response function $H\left(e^{\imath\theta}\right)$ 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: $F\left(e^{\imath\phi}\right) = e^{\imath\theta(\phi)}$, so that $G\left(e^{\imath\phi}\right) = H\left(e^{\imath\theta(\phi)}\right)$.
2. If $H(z)$ is stable and minimum phase, then $G(z)$ is stable and minimum phase. If $\lambda$ 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:

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

This is a frequency transformation since

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

Order $n$ frequency transformations are known as *Blaschke products*

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

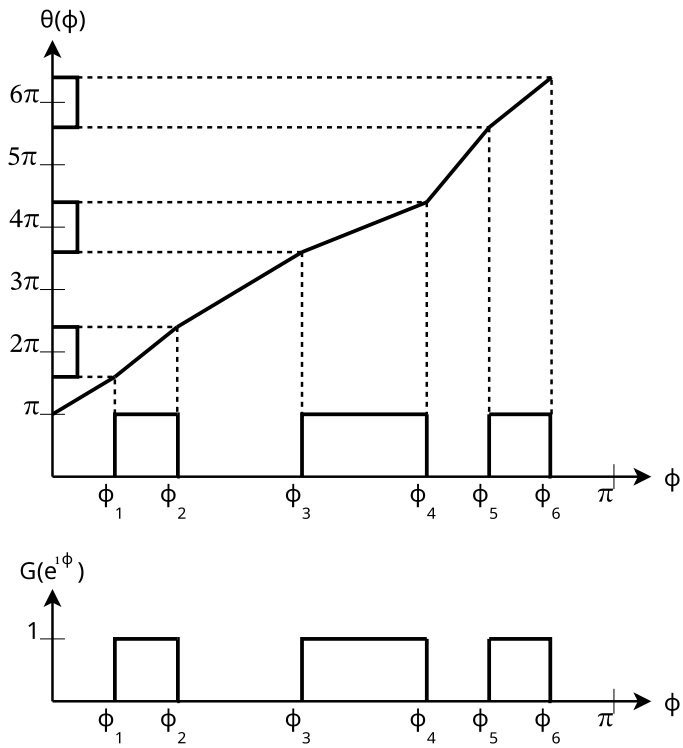



Figure 2.1: Frequency warping function (after *Roberts* and *Mullis* [168, Figure 6.7.2])

where

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

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

Suppose the prototype $H(z)$ has cut-off frequency $\theta = \pi/2$ and $G(z)$ has the pass band edges shown in Figure 2.1 (where $G(z)$ is low-stop).

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

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

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

*Roberts* and *Mullis* [168, Figure 6.7.3] show an algorithm for calculating the parameters of the frequency transformation $F(z)$. This algorithm is implemented by the Octave [91] function *phi2p*.

## 2.2 Frequency Transformation of the Transfer Function

See *Roberts* and *Mullis* [168, 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 $\tilde{p}(z) = z^{-M} p\left(z^{-1}\right)$, 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 $p(z)$:

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

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

$$\begin{aligned}
\left[\begin{array}{cccccc} p_0 & \cdots & p_M & 0 & \cdots & 0 \end{array}\right] &\overset{DFT}{\longleftrightarrow} \left[\begin{array}{cccc} P(0) & P(1) & \cdots & P(N-1) \end{array}\right] \\
\left[\begin{array}{cccccc} a_0 & \cdots & a_{LM} & 0 & \cdots & 0 \end{array}\right] &\overset{DFT}{\longleftrightarrow} \left[\begin{array}{cccc} A(0) & A(1) & \cdots & A(N-1) \end{array}\right] \\
\left[\begin{array}{cccccc} b_0 & \cdots & b_{LM} & 0 & \cdots & 0 \end{array}\right] &\overset{DFT}{\longleftrightarrow} \left[\begin{array}{cccc} B(0) & B(1) & \cdots & B(N-1) \end{array}\right]
\end{aligned}$$

where

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

and

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

Here $W_N = e^{\imath\frac{2\pi}{N}}$ and arg means *"angle of"*. Similarly:

$$B(n) = \beta\left(e^{\imath\theta_n}\right) \{P(n)\}^L$$

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

## 2.3 Frequency Transformations of State Variable Filters

See *Mullis and Roberts* [28, 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

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

(here † means complex conjugate transpose).

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

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

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 in the original filter, $H(z)$, so for a $m \times n$ matrix $S(k)$

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

Eliminating $x(k)$ gives

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

Define $V(X)$ as the vector formed by stacking the columns of $X$ and define $G \otimes F$ as the *Kronecker product*[a], $\{G_{ij}F\}$, 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

$$H(F(z)) = \mathfrak{D} + \mathfrak{C}(zI - \mathfrak{A})^{-1}\mathfrak{B} \tag{2.1}$$

*Mullis and Roberts* [28] give a direct proof. Start with the identity:

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

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

$$V\left(FXG^{\top}\right) = [G \otimes F]\, V(X)$$

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:

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

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

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

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

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*.

## 2.4 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 2.2 shows the prototype filter. Figure 2.3 shows the result of a low-pass to low-pass transformation. Figure 2.4 shows the result of a low-pass to high-pass transformation. Figure 2.5 shows the result of a low-pass to band-pass transformation. Figure 2.6 shows the result of a low-pass to triple band-stop transformation.

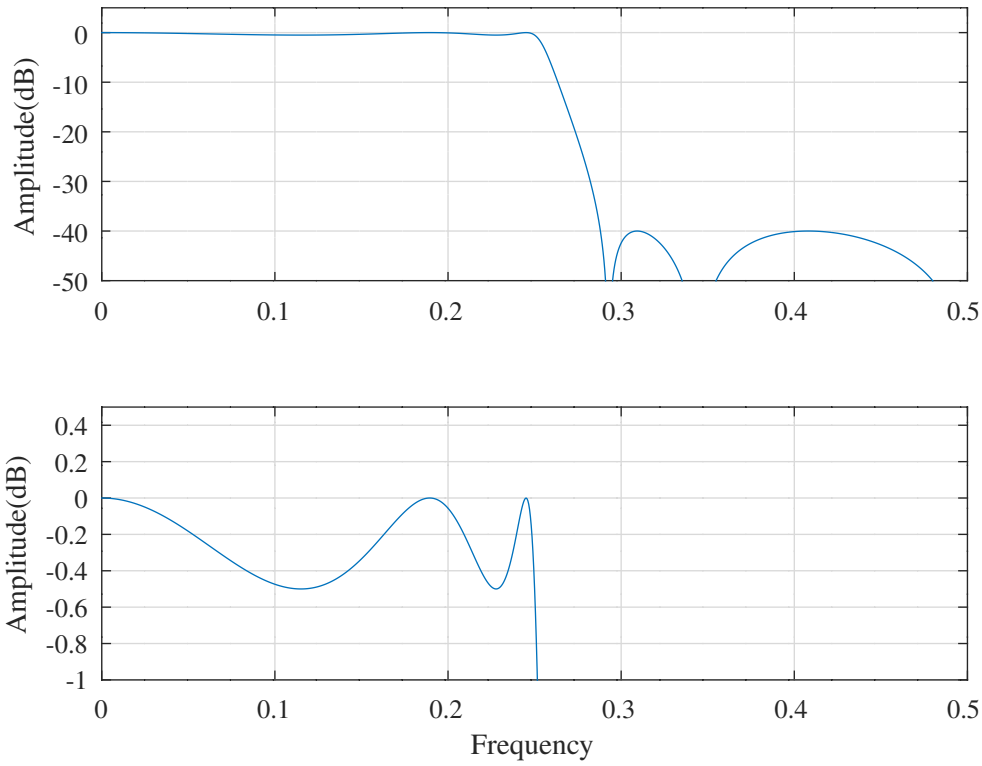



Figure 2.2: Prototype 5-th order elliptic low-pass filter

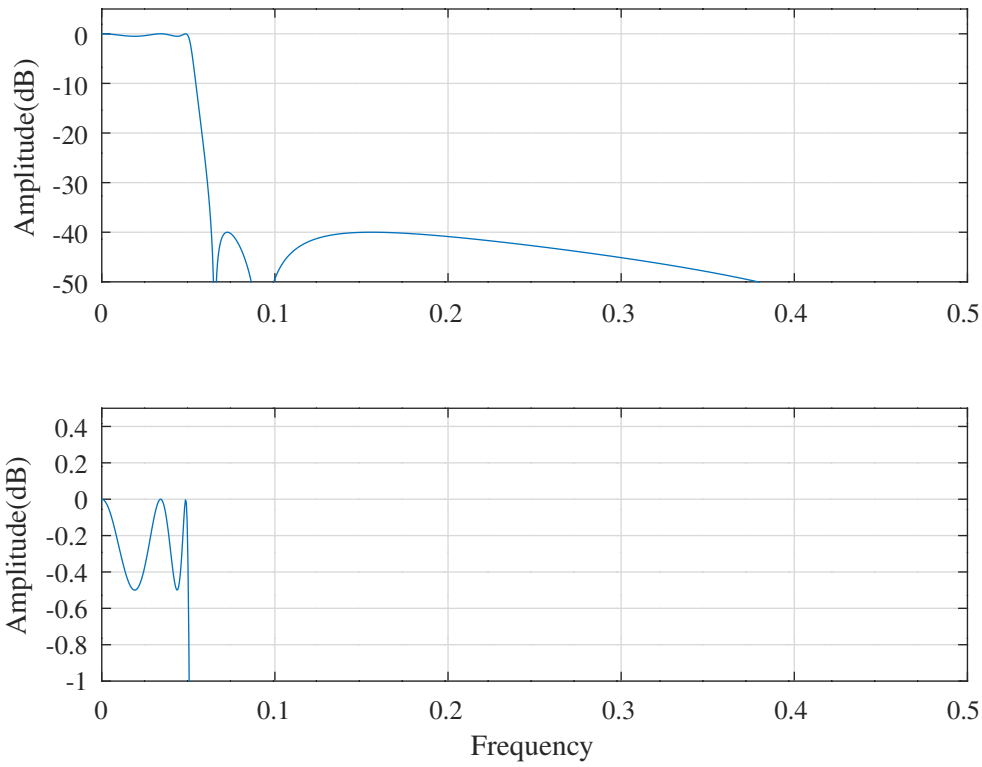



Figure 2.3: Low-pass to low-pass transformation of a 5-th order elliptic low-pass filter

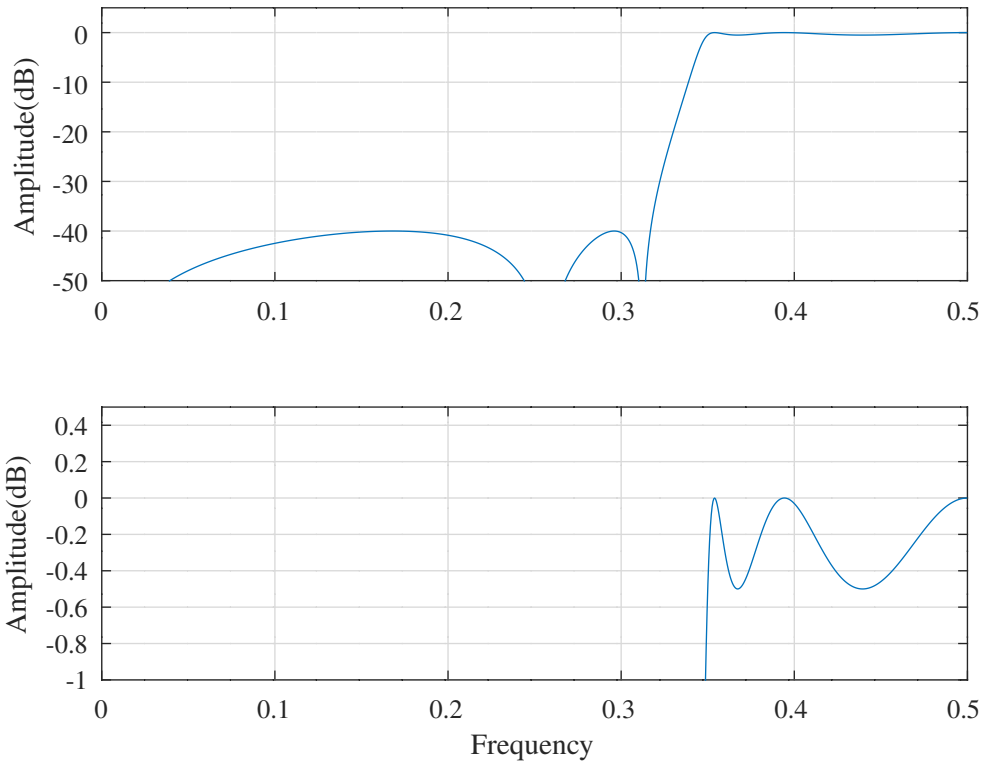



Figure 2.4: Low-pass to high-pass transformation of a 5-th order elliptic low-pass filter

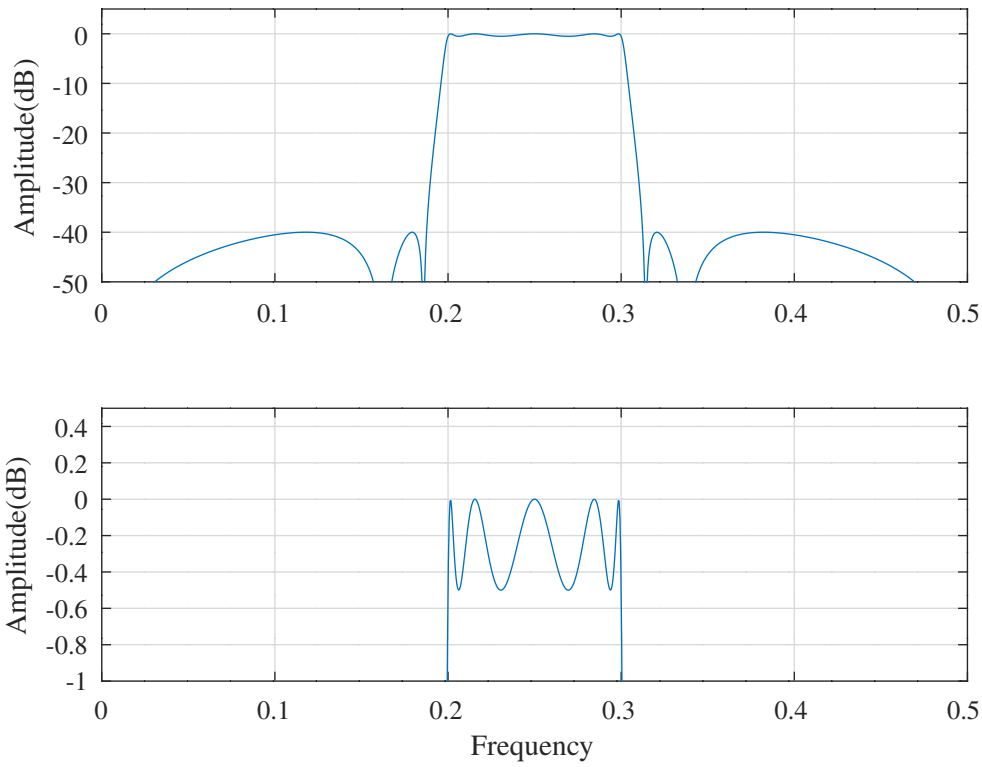



Figure 2.5: Low-pass to band-pass transformation of a 5-th order elliptic low-pass filter

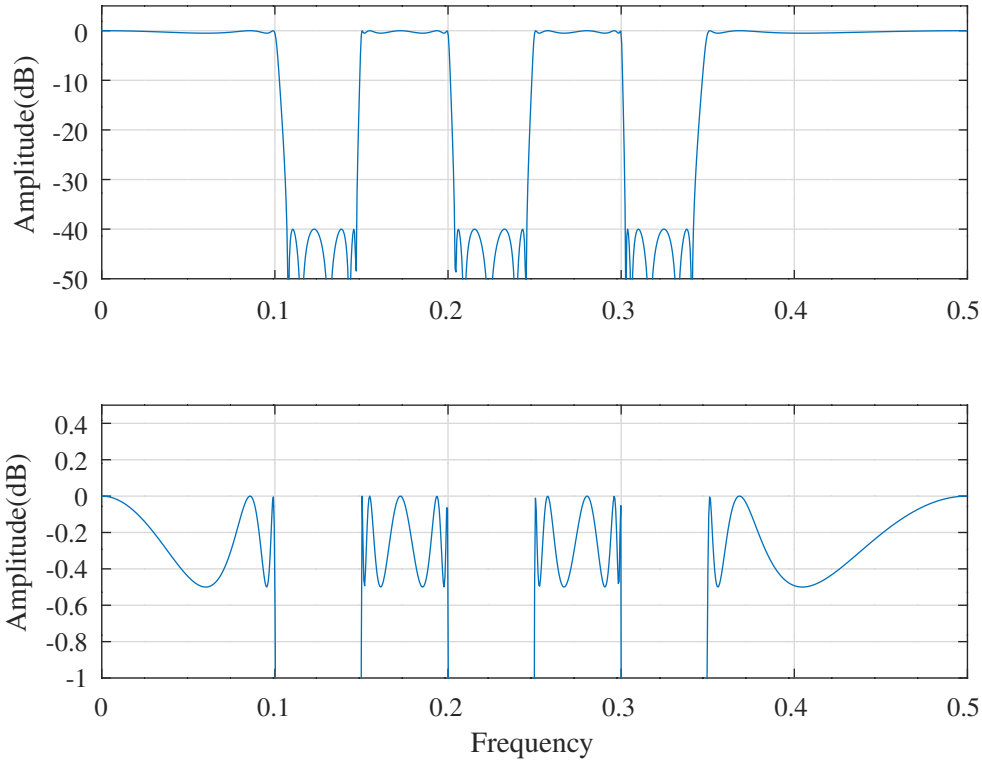



Figure 2.6: Low-pass to triple band-stop transformation of a 5-th order elliptic low-pass filter

# Chapter 3

# Round-off noise in state variable filters

This chapter is based on Chapter 9 of *Roberts* and *Mullis* [168].

## 3.1 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:

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

Here $\Delta$ 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 $\left(-\frac{q}{2}, \frac{q}{2}\right)$ with variance[a] $\sigma_e^2 = \frac{q^2}{12}$. 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.

[a] The 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:

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

## 3.2 Limit-cycle oscillations in digital filters

See *Roberts* and *Mullis* [168, 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.

## 3.3 State variable filters and wide sense stationary inputs

### 3.3.1 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:

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

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

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

For unit variance white inputs

$$K = \sum_{l=0}^{\infty}\left(A^l B\right)\left(A^l B\right)^\top$$

Algorithm 3.1 is a simple recursive calculation of the covariance matrix implemented in the Octave function *dlyap_recursive*.

Alternative methods for calulating 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* [168, Section 9.11, p. 393]. The Octave function *dlyap_levinson* implements this solution.
- use *Hammarling*'s solution of the discrete *Lyapunov* equation [182]. The Octave *Control* package contains the *dlyap* function that calls functions from an open-source version of the *SLICOT* library [44] to implement *Hammarling*'s algorithm.

**Algorithm 3.1** Recursive calculation of the covariance matrix

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

### 3.3.2 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

$$h(k) = \begin{cases} 0 & k < 0 \\ D & k = 0 \\ CA^{k-1}B & k > 0 \end{cases} \qquad \underset{\Leftrightarrow}{DFT} \qquad H\left(e^{\imath\theta}\right) = D + C\left(e^{\imath\theta}I - A\right)^{-1}B$$

Since

$$r_{yy}(k) \qquad \underset{\Leftrightarrow}{DFT} \qquad S_{yy}(\theta) = \left|H\left(e^{\imath\theta}\right)\right|^2$$

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

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

The first term is

$$\begin{aligned} E\left\{x(k+l)\,x^\top(l)\right\} &= E\left\{\left(\sum_{m=0}^{\infty} A^m Bu(k+l-m-1)\right)\left(\sum_{m'=0}^{\infty} A^{m'} Bu(l-m'-1)\right)^\top\right\} \\ &= \sum_{m=0}^{\infty}\sum_{m'=0}^{\infty} (A^m B)\left(A^{m'}B\right)^\top r_{uu}(k-m-m') \\ &= \sum_{m=0}^{\infty} A^k A^m B\,(A^m B)^\top \\ &= A^k K \end{aligned}$$

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

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

(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 is $u(k+l)\,x^{\top}(k)$ is zero. Finally,

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

The energy in the unit-pulse response is given by

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

### 3.3.3 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 $\Delta$ 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:

$$v(k) = (f \star u)(k)$$

where $f$ is the unit pulse response from the input to $v(k)$ and $\star$ 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:

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

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:

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

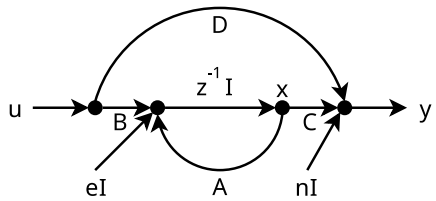



Figure 3.1: Signal flow graph of round-off noise of the state variable

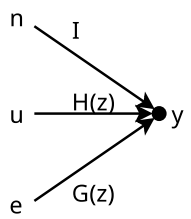



Figure 3.2: Signal flow graph of round-off noise at the filter output

The parameter $\delta$ 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 $\delta$ 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:

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

This can be achieved by applying the coordinate transformation:

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

where:

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

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

## 3.4 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 3.1. Where $e$ models the round off noise due to calculation of the state vector, $x$, and $n$ models the round off noise due 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 3.2. Where:

$$\begin{aligned} H(z) &= D + C(zI - A)^{-1}B \\ G(z) &= C(zI - A)^{-1} \\ g(k) &= \begin{cases} 0 & k \leq 0 \\ CA^{k-1} & k > 0 \end{cases} \end{aligned}$$

$G(z)$ 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 $\sigma_i^2$ 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, $W$:

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

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:

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

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, $T$:

$$\{A\,,\ B\,,\ C\,,\ D,\ K,\ W\} \quad \leftarrow \quad \left\{T^{-1}AT\,,\ T^{-1}B\,,\ CT\,,\ T^{-1}KT^{-1^\top}\,,\ T^\top W T\right\}$$

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 $\{A, B, C, D\}$:

$$\|h\|_2^2 = D^2 + CKC^\top \tag{3.1}$$

### 3.4.1 Rounding-to-minus-infinity quantisation noise

As stated in Section 3.1, 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 $\left(-\frac{q}{2}, \frac{q}{2}\right)$ 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* demonstates the effect of using rounding-to-minus-infinity rounding in a 3rd order Butterworth filter with cut-off frequency $0.05 f_S$. The Butterworth filter is globally optimised so that

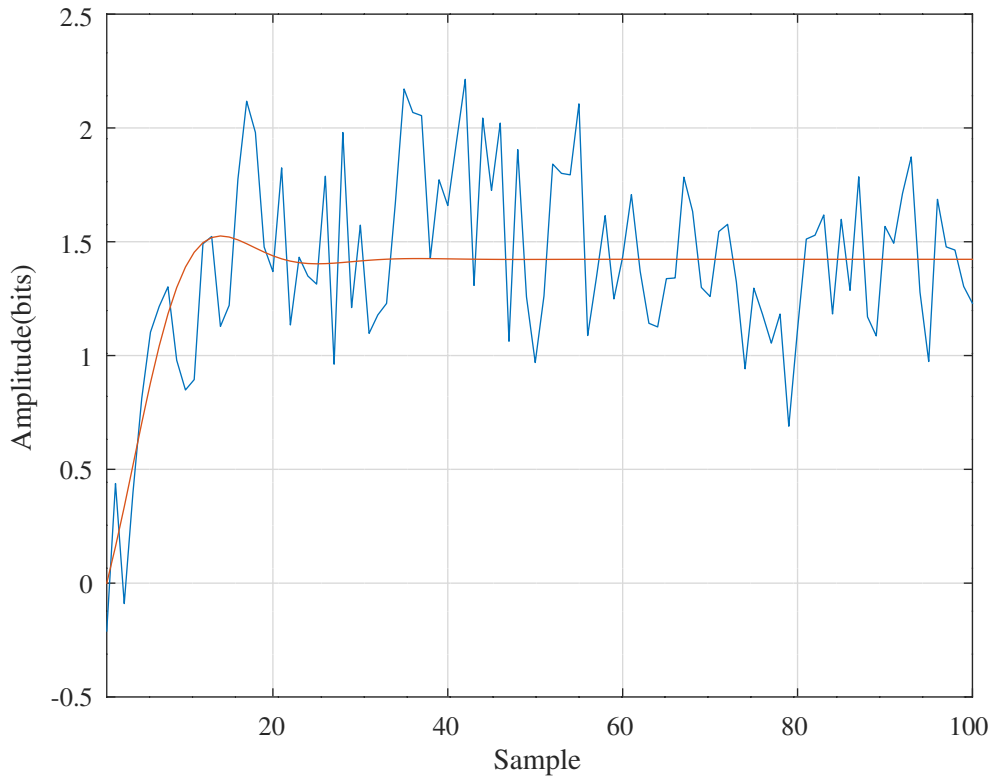



Figure 3.3: Summed state-to-output step response superimposed on the output quantisation noise of a 3rd order Butterworth filter due to rounding-to-minus-infinity at each state.

**Algorithm 3.2** 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 $\mu_i^2$ 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 $j$.

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 3.3 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.

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.

## 3.5 Minimization of round-off noise in the calculation of the state vector

Minimising the noise gain minimises the round off noise. *Mullis* and *Roberts* [29] prove Algorithm 3.2 for equal word-length filters.

Algorithm 3.3 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 3.3 is implemented in the Octave function *optKW.m*.

**Algorithm 3.3** Optimisation of the noise gain

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

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

   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

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

2. Balance $D_2^{\frac{1}{2}}$:
   Now find a unitary transformation, $U_3$, for which the diagonal elements of $U_3^\top D_2^{\frac{1}{2}} U_3$ 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:

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

The globally optimised state variable filter has $\mathcal{O}\left(n^2\right)$ coefficients. An alternative structure such as a lattice filter or a cascade of lower-order filters has fewer coefficients, $\mathcal{O}\left(n\right)$, 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* [29, Section IV] show that if the state wordlengths are $B_i$ with

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

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

$$B_i = B + \frac{1}{2}\log_2 K_{ii}W_{ii} - \frac{1}{2n}\sum_{j=1}^{n}\log_2 K_{jj}W_{jj} \tag{3.2}$$

The optimal output noise is then:

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

## 3.6 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.

## 3.7 Factored state variable filters and wide sense stationary inputs

The estimate of round-off noise shown in Section 3.4 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

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

then the covariance matrix for $q_0\left(k\right)$ is

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

The factored state variable equations lead to

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

leading to

$$\tilde{K}_{l+1} = E\left\{q_{l+1}^{(k)}\left(k+1\right)\left(q_{l+1}^{(k)}\left(k+1\right)\right)^\top\right\}$$

$$= F_{l+1}\tilde{K}_l F_{l+1}^\top$$

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

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

## 3.8 Frequency transformations and round-off noise

Section 2.3 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 $\{\mathfrak{ABCD}\}$ 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* [28, 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:

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

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:

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

Consequently, if

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

then

$$\mathfrak{K} = K \otimes Q \tag{3.3}$$

$$\mathfrak{W} = W \otimes Q^{-1} \tag{3.4}$$

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 4

# State variable filter realisation as a cascade of second order sections

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

Section 3.5 shows how to calculate the minimum noise state variable filter for an $N$-th order rational transfer function filter. The minimum noise filter has $\mathcal{O}\left(N^2\right)$ 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 $\mathcal{O}\left(N\right)$ coefficients[a]. 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 $\|g_m\|_2^2 = 1$.

## 4.2 Design equations for optimised second order state variable filters

*Roberts* and *Mullis* [168, 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 4.1.

*Bomar* [24] gives design equations, shown in Algorithm 4.2 for a state variable second order section with scaling $\delta = 1$ and optimal noise performance. (See also *Roberts* and *Mullis* [168, Figure 9.12.1 with corrections]). *Bomar* assumes that transfer function of each second order section is arranged in the form:

$$H\left(z\right) = d + \frac{q_1 z^{-1} + q_2 z^{-2}}{1 + p_1 z^{-1} + p_2 z^{-2}} \tag{4.1}$$

*Bomar* [24] also gives design equations, shown in Algorithm 4.3, 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 4.1 into 2nd-order *direct-form*, *Bomar-Type-III* or *minimum-noise* state variable sections.

*Bomar* [25] also describes realisation of second-order state variable 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.

[a] *Roberts* and *Mullis* [168, 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 4.1** Construction of optimised second order state variable filters[168, 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

$$K' = T_c^{-1} K T_c^{-1\top} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and

$$W' = T_c^\top W T_c = \begin{bmatrix} w'_{11} & w'_{12} \\ w'_{21} & w'_{22} \end{bmatrix}$$

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

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

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

Then

$$K'' = I$$

$$\begin{aligned} W'' &= R(-\theta)\, W'(\theta) \\ &= \begin{bmatrix} \mu_1^2 & 0 \\ 0 & \mu_2^2 \end{bmatrix} \end{aligned}$$

3. Now apply a transformation:

$$\begin{aligned} \mu &= \frac{\mu_2}{\mu_1} \\ T &= \frac{\delta}{2} \begin{bmatrix} \sqrt{1+\mu} & -\sqrt{1+\mu} \\ \sqrt{1+\frac{1}{\mu}} & \sqrt{1+\frac{1}{\mu}} \end{bmatrix} \end{aligned}$$

so that

$$\begin{aligned} K''' &= \frac{1}{\delta^2} \begin{bmatrix} 1 & \frac{\mu_2-\mu_1}{\mu_2+\mu_1} \\ \frac{\mu_2-\mu_1}{\mu_2+\mu_1} & 1 \end{bmatrix} \\ W''' &= \delta^2 \begin{bmatrix} \frac{(\mu_2+\mu_1)^2}{4} & \frac{\mu_2^2-\mu_1^2}{4} \\ \frac{\mu_2^2-\mu_1^2}{4} & \frac{(\mu_2+\mu_1)^2}{4} \end{bmatrix} \end{aligned}$$

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

**Algorithm 4.2** Bomar second order optimised state variable filter sections [24, Equation 17]. (See also [168, Figure 9.12.1])

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

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

**Algorithm 4.3** Bomar Type III second order optimised state variable filter sections [24, Equation 23]

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

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

## 4.3 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 $2\times2$ 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

## 4.4 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 variable 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 4.1. The sections are ordered with increasing pole angle[b]. The Octave function *pq2svcasc* converts these coefficients to second-order direct-form, Bomar Type III or minimum-noise state variable 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, *svcas2noise* estimates the optimal state variable bit distribution according to Equation 3.2.

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 variable 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.

### 4.4.1 Comparison of calculated noise gains

Table 4.1 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 4.2 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

[b] The Octave function *sos2pq* converts the output of the Octave-Forge *signal* package [140] *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 4.1: 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 4.2: 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 4.1 and Table 4.2. The noise-gains for each second-order minimum-noise section calculated according to Bomar's equations, as shown in Algorithm 4.2, agree with those calculated following the general method shown in Algorithm 4.1, although the state variable coefficients found by the two methods are different.

Tables 4.3 and 4.4 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 4.3: Overall noise gains for the 20th order Butterworth lowpass filter

### 4.4.2 Simulation results

Figure 4.1 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 4.2 shows the simulated response of the corresponding 20th order Butterworth highpass filter. Tables 4.5 and 4.6 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 4.4: Overall noise gains for the 20th order Butterworth highpass filter

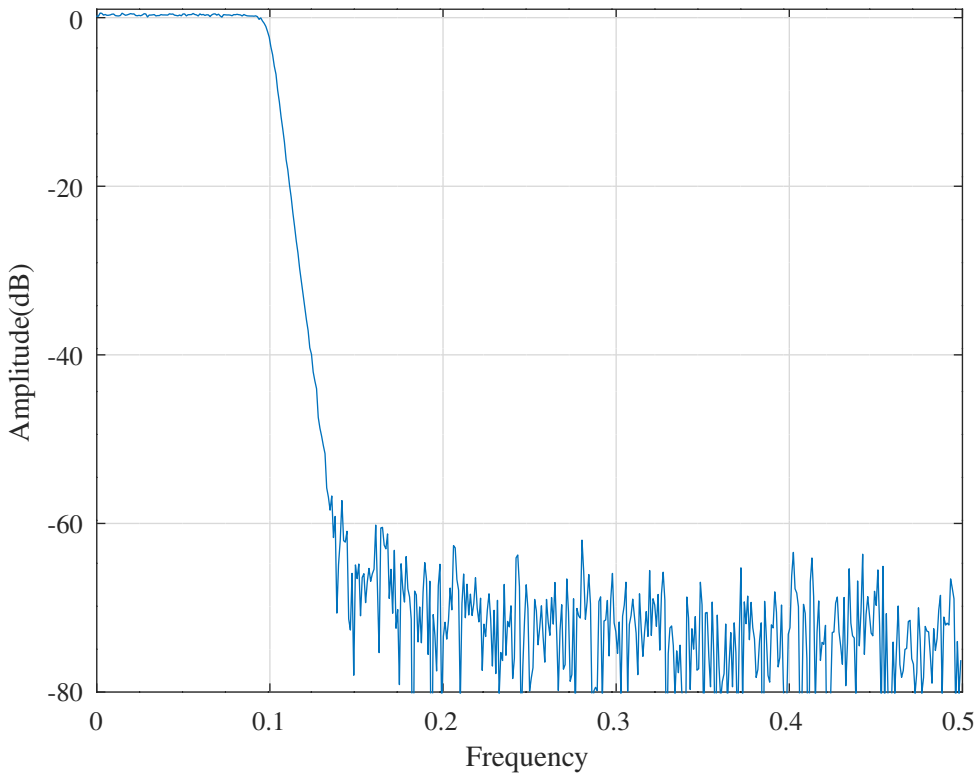



Figure 4.1: Simulated amplitude response of the 20th order lowpass Butterworth filter realised as a block optimised cascade of second-order sections with 10-bit coefficents and state storage

| | Estimated noise gain | Estimated noise variance | Simulated noise variance |
|---|---|---|---|
| Scaled Direct | 67.89 | 90.67 | 65.31 |
| Block Opt. | 16.34 | 21.94 | 19.94 |
| Block Opt. (extra bits) | 5.44 | 7.41 | 6.43 |
| Global Opt. | 1.70 | 2.36 | 2.42 |

Table 4.5: 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 | 61.50 |
| Block Opt. | 15.81 | 21.57 | 20.50 |
| Block Opt. (extra bits) | 5.26 | 7.50 | 7.22 |
| Global Opt. | 1.65 | 2.28 | 2.33 |

Table 4.6: Estimated noise gain and estimated and simulated output roundoff noise variances for the 20th order Butterworth highpass filter with 10 bit rounded coefficients.

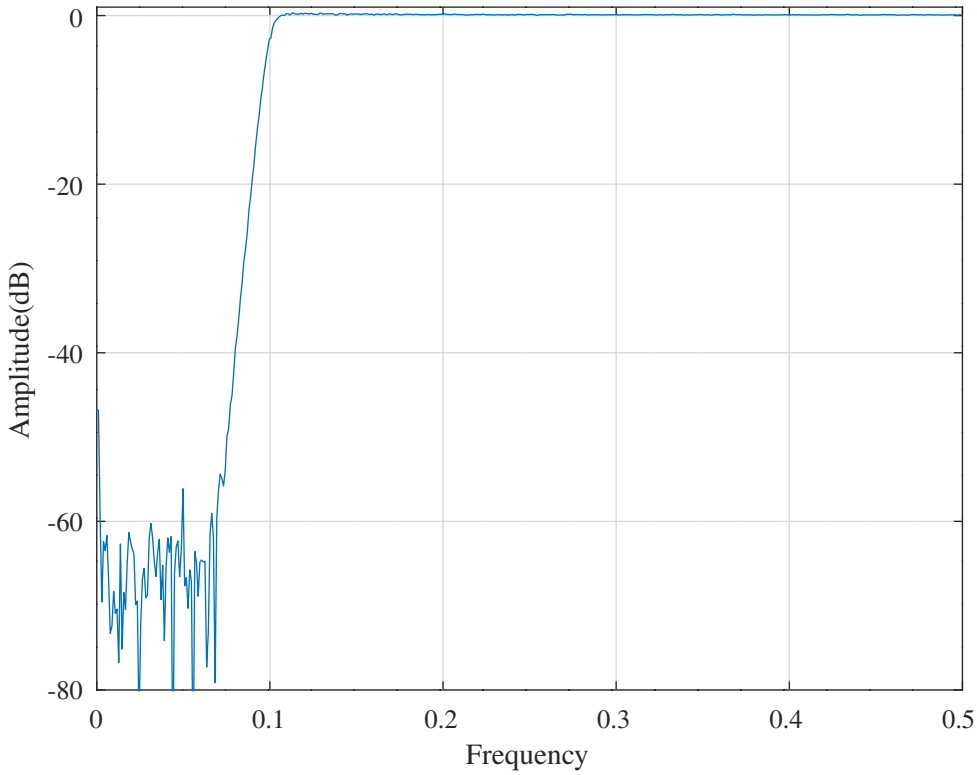



Figure 4.2: Simulated amplitude response of the 20th order highpass Butterworth filter realised as a block optimised cascade of second-order sections with 10-bit coefficents and state storage

| | 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. | 1.01 | 1.43 | 1.43 |

Table 4.7: Estimated noise gain and estimated and simulated output roundoff noise variances for the 10th order Butterworth lowpass filter with 10 bit rounded coefficients.

the calculated values shown in Table 4.3 and Table 4.4. 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 3.2). 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 $2^7$.

### 4.4.3 Comparison with an N=10 example

The order 20 Butterworth filter was chosen as an extreme example. Table 4.7 shows the simulation results obtained by setting $N = 10$ in *svcasc2noise_example_test.m*.

## 4.5 Coefficient sensitivity and round-off noise of first-order and second-order all-pass filter sections

Digital filters implemented as the parallel combination of two or more allpass filters typically have low coefficient sensitivity [152]. The transfer function of a filter consisting of parallel all-pass filters $A(z)$ and $B(z)$ is:

$$H(z) = \frac{A(z) + B(z)}{2}$$

The frequency response of this filter is:

$$H(\omega) = \frac{e^{\phi_A(\omega)} + e^{\phi_B(\omega)}}{2}$$

where $\phi_A(\omega)$ and $\phi_B(\omega)$ are the phase responses of the all-pass filters. The corresponding squared-amplitude, phase and group delay responses are:

$$\begin{aligned}
|H(\omega)|^2 &= \frac{1 + \cos(\phi_A(\omega) - \phi_B(\omega))}{2} \\
\phi_H(\omega) &= \frac{\phi_A(\omega) + \phi_B(\omega)}{2} \\
T(\omega) &= -\frac{1}{2}\left[\frac{\partial \phi_A(\omega)}{\partial \omega} + \frac{\partial \phi_B(\omega)}{\partial \omega}\right]
\end{aligned}$$

When the all-pass branches of a parallel all-pass filter are realised as the cascaded connection of first and second order all-pass filters then the phase response of each branch is, without loss of generality, $\phi_A(\omega) = \sum_l \phi_{A_l}(\omega)$ where $\phi_{A_l}$ is the phase response of the $l$'th first or second order all-pass section. If $x$ represents a multiplier coefficient in the realisation of section $A_l$, then the squared-amplitude and group delay sensitivities with respect to $x$ are:

$$\begin{aligned}
S_x^{|H|^2}(\omega) &= -\frac{1}{|H(\omega)|^2}\frac{\sin(\phi_A(\omega) - \phi_B(\omega))}{2}\frac{\partial \phi_{A_l}(\omega)}{\partial x} \\
S_x^{\top}(\omega) &= -\frac{1}{T(\omega)}\frac{1}{2}\frac{\partial^2 \phi_{A_l}(\omega)}{\partial \omega \partial x}
\end{aligned}$$

The transfer function of a first order all-pass filter section is:

$$H(z) = -\frac{r - z^{-1}}{1 - rz^{-1}}$$

where $r$ is real. The filter is stable if $|r| < 1$.

The transfer function of a second order all-pass filter section is:

$$H(z) = \frac{a_2 + a_1 z^{-1} + z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

The transfer function of a second order all-pass filter section with complex poles in the $z$-plane at $z = re^{\pm \imath\theta}$ is:

$$\begin{aligned}
H(z) &= \frac{\left(re^{\imath\theta} - z^{-1}\right)\left(re^{-\imath\theta} - z^{-1}\right)}{\left(1 - re^{\imath\theta}z^{-1}\right)\left(1 - re^{-\imath\theta}z^{-1}\right)} \\
&= \frac{r^2 - 2r\cos\theta z^{-1} + z^{-2}}{1 - 2r\cos\theta z^{-1} + r^2 z^{-2}}
\end{aligned}$$

where $r$ is real. The filter is stable if $|r| < 1$. The transfer function of a second order all-pass filter section with real poles in the $z$-plane at $z = r_1, r_2$ is:

$$\begin{aligned}
H(z) &= \frac{\left(r_1 - z^{-1}\right)\left(r_2 - z^{-1}\right)}{\left(1 - r_1 z^{-1}\right)\left(1 - r_2 z^{-1}\right)} \\
&= \frac{r_1 r_2 - (r_1 + r_2) z^{-1} + z^{-2}}{1 - (r_1 + r_2) z^{-1} + r_1 r_2 z^{-2}}
\end{aligned}$$

The filter is stable if $|r_1| < 1$ and $|r_2| < 1$.

This section surveys the maximum phase response gradient, $\frac{\partial \phi}{\partial x}$, and round-off noise performance of a selection of first-order and second-order all-pass filter section transfer functions and realisations. The Maxima script *allpass_filter.max* performs the algebra required to find the state variable description of each realisation and the Octave scripts *Abcd2H.m* and *H2P.m* calculate the phase response gradient (see Appendix J).

### 4.5.1 Searching for realisations of all-pass filter transfer functions

*Mitra* and *Hirano* [184] show a catalogue of realizations of minimum multiplier first and second order all-pass filters. They consider realizations of the *Type 1* first order all-pass transfer function:

$$H_1(z) = \frac{z^{-1} - b_1}{1 - b_1 z^{-1}} \tag{4.2}$$

as a *two-port* network with a constraint:

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$
$$X_2 = b_1 Y2$$

Eliminating variables $X_2$ and $Y_2$:

$$\frac{Y_1}{X_1} = \frac{t_{11} - b_1 (t_{11} t_{22} - t_{12} t_{21})}{1 - t_{22} b_1}$$

Comparing with Equation 4.2:

$$t_{11} = t_{22} = z^{-1}$$
$$t_{12} t_{21} = z^{-2} - 1$$

There are four possible realisations of $t_{12}$ and $t_{21}$: $t_{12} = z^{-2} - 1, t_{21} = 1$ and $t_{12} = z^{-1} - 1, t_{21} = z^{-1} + 1$ and their transposed equivalents [184, Fig. 2].

In a similar fashion, *Mitra* and *Hirano* find realisations of the *Type 2* and *Type 3* second order all-pass transfer functions:

$$H_{MH2}(z) = \frac{z^{-2} - b_1 z^{-1} + b_1 b_2}{1 - b_1 z^{-1} + b_1 b_2 z^{-2}} \tag{4.3a}$$

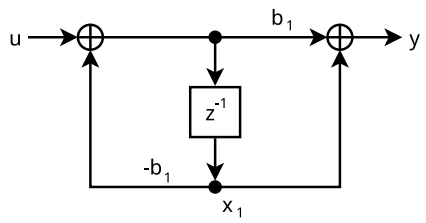
$$H_{MH3}(z) = \frac{z^{-2} - b_1 z^{-1} + b_2}{1 - b_1 z^{-1} + b_2 z^{-2}} \tag{4.3b}$$

Their realisations of these transfer functions are constrained to have only 2 multipliers but may have more than 2 delays. *Mitra* and *Hirano* show 4 Type 2 and 8 Type 3 realisations[c]. *Mitra* and *Hirano* show plots of the estimated output round-off noise of each realisation [184, Fig.9, Fig.10 and Fig.11] due to truncation at the multiplier outputs. This estimate does not include the round-off noise due to truncation at the register inputs.

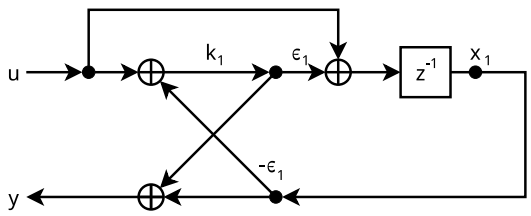
*Nishihara* and *Sugahara* [8] present a catalogue of general (not just all-pass) second order filter realisations with low pole sensitivity with respect to each of the two multipliers in the realisation. They show 37 realisations[d].

*Szczupak et. al* [84] describe a computer based search for realisations in which the two coefficients of the second order all-pass filter transfer function are themselves functions of two multipliers. They find 646 distinct realisations.

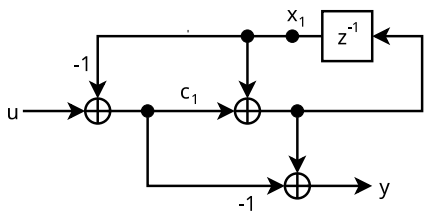
[c] There are equal numbers of transposed realisations.
[d] Note the complexity of the classification map shown in Figure 2!




(a) Direct form first order all-pass filter section




(b) Gray and Markel first order all-pass filter section




(c) Stoyanov et al. low sensitvity first order all-pass filter section

Figure 4.3: First order all-pass filter sections

### 4.5.2 Maximum phase gradient and round-off noise of some first-order all-pass filter sections

Equation 4.4 is the transfer function of the first order direct form all-pass filter section shown in Figure 4.3a. This realisation has 2 multipliers.

$$H_{Dir1}(z) = \frac{b_1 + z^{-1}}{1 + b_1 z^{-1}} \tag{4.4}$$

The *Gray* and *Markel* [4] first order all-pass filter section shown in Figure 4.3b also implements the transfer function of Equation 4.4. The $\epsilon_1 = \pm 1$ are chosen to scale the state for good numerical performance in the implementation. The choice of $\epsilon$ does not alter the transfer function or the noise gain.

Equation 4.5 is the transfer function of the first order all-pass filter section of *Stoyanov et al.* [57] shown in Figure 4.3c. This realisation has low sensitivity for filter poles near $z = 1$.

$$H_{LS1}(z) = \frac{-(1 - c_1) + z^{-1}}{1 - (1 - c_1) z^{-1}} \tag{4.5}$$

Figure 4.5 plots the maximum of the gradient of the first-order all-pass filter phase response with respect to the section coefficient against the pole radius. It is the same for each realisation. The noise gain of each scaled realisation was calculated with the Octave function *Abcd2ng.m*, as shown in Chapter 3. The noise gain is found to be 1 for each realisation regardless of pole radius. For the direct-form section, a *slowed* and *retimed* realisation, shown in Figure 4.4, was analysed. *Slowing* is the replacement of each $z^{-1}$ delay by $z^{-M}$, reducing the sample rate. The realisation is *retimed* by distributing the delays so that there is a register, or state, at each multiplier output. This allows calculation by the state-variable method of the output round-off noise due to truncation at both the register inputs and the multiplier outputs. See *Parhi* [98, Chapter 4] for a description of retiming algorithms. The Maxima script *allpass_filter_retimed.max* performs the algebra required to find the state variable description of each retimed filter realisation.

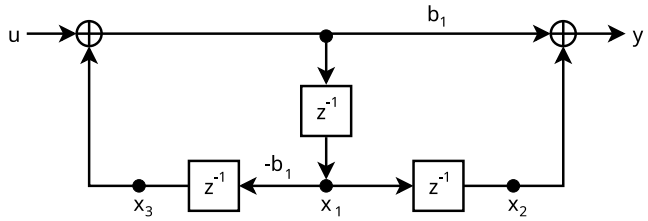



Figure 4.4: Retimed first-order direct form all-pass filter section

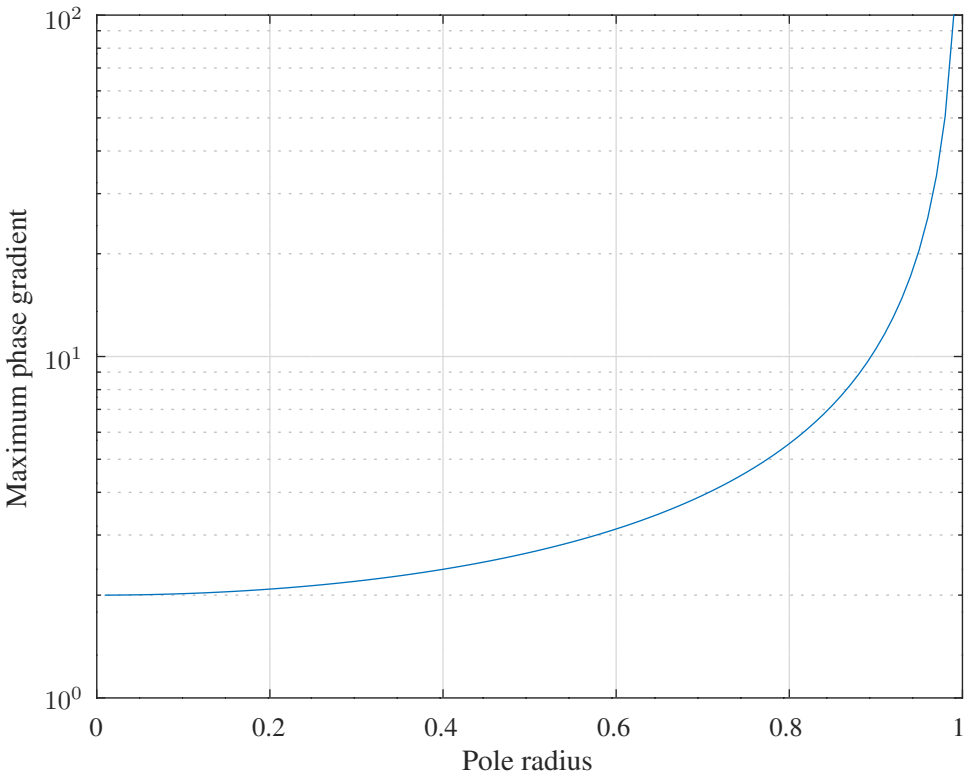



Figure 4.5: Maximum phase gradient plotted against pole radius of first order all-pass filter sections

### 4.5.3 Maximum phase gradient and round-off noise of some second-order all-pass filter sections

Equation 4.6 is the transfer function of the second-order direct-form all-pass filter section shown in Figure 4.6a. This realisation has $4$ multipliers.

$$H_{Dir2}(z) = \frac{b_2 + b_1 z^{-1} + z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \tag{4.6}$$

Equation 4.7 is the transfer function of the second order Gray-and-Markel all-pass filter section shown in Figure 4.6b. The $\epsilon_1, \epsilon_2 = \pm 1$ are chosen to scale the states.

$$H_{GM2}(z) = \frac{k_2 + k_1 (1 + k_2) z^{-1} + z^{-2}}{1 + k_1 (1 + k_2) z^{-1} + k_2 z^{-2}} \tag{4.7}$$

Figure 4.6c shows a realisation of Equation 4.7 due to *Ansari* and *Lui* [169]. This realisation has low noise gain for poles near $z = \pm\imath$.

The three port transfer matrix of the *Mitra* and *Hirano* [184] type 2d realisation of Equation 4.3a is:

$$\begin{bmatrix} z^{-2} & 1 & 0 \\ z^{-1}\left(z^{-2} - 1\right) & z^{-1} & 1 \\ z^{-4} - 1 & z^{-2} & 0 \end{bmatrix} \tag{4.8}$$

and that of the type 3d realisation of Equation 4.3b is:

$$\begin{bmatrix} z^{-2} & 1 & 1 \\ z^{-1}\left(z^{-2} - 1\right) & z^{-1} & z^{-1} \\ z^{-4} - 1 & z^{-2} & z^{-2} \end{bmatrix} \tag{4.9}$$

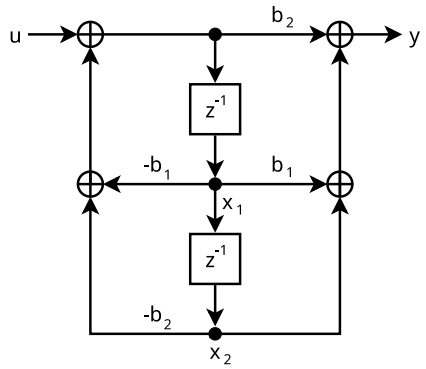
Figures 4.6d and 4.6e show the type 2d and transposed type 2d realisations of Equation 4.3a. Figures 4.6f and 4.6g show the type 3d and transposed type 3d realisations of Equation 4.3b. The state variable implementations of the transposed realisations have duplicated state updates. In other words, the rows of the state-transition matrix are not linearly independent.

Equation 4.10 is the transfer function of the second order all-pass filter section of *Stoyanov et al.* [57] shown in Figure 4.6h. This realisation has low sensitivity for filter poles near $z = 1$.

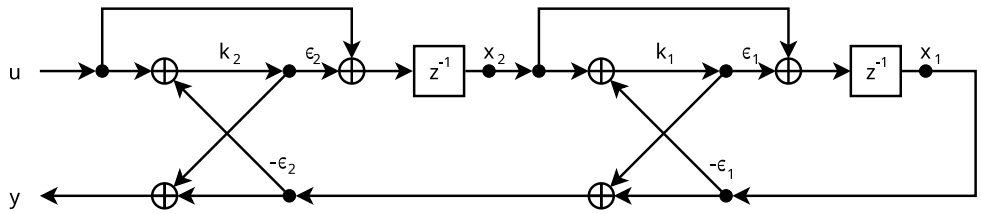
$$H_{LS2}(z) = \frac{(1 - c_2) + (2c_1 + c_2 - 2) z^{-1} + z^2}{1 + (2c_1 + c_2 - 2) z^{-1} + (1 - c_2) z^{-2}} \tag{4.10}$$

Equation 4.11 is the transfer function of the second order all-pass filter section of *Ivanova* and *Stoyanov* [95] shown in Figure 4.6i. This realisation has low sensitivity for filter poles near $z = 0$.

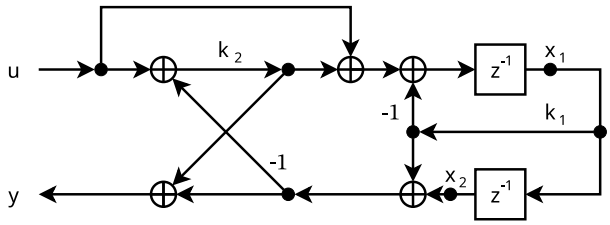
$$H_{IS}(z) = \frac{d_2 + (d_1 d_2 - d_1 - 2d_2) z^{-1} + z^2}{1 + (d_1 d_2 - d_1 - 2d_2) z^{-1} + d_2 z^{-2}} \tag{4.11}$$



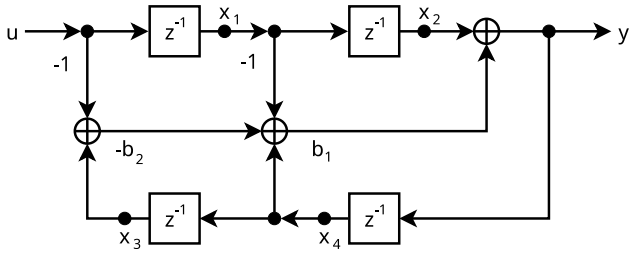

(a) Direct form second order all-pass filter section



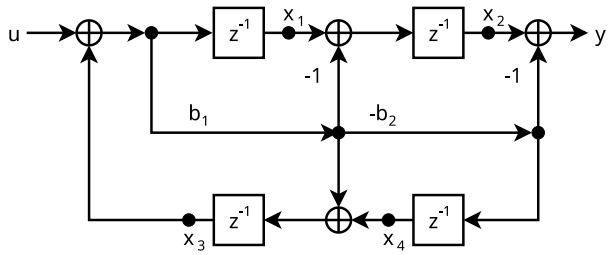

(b) Gray-and-Markel second order all-pass filter section



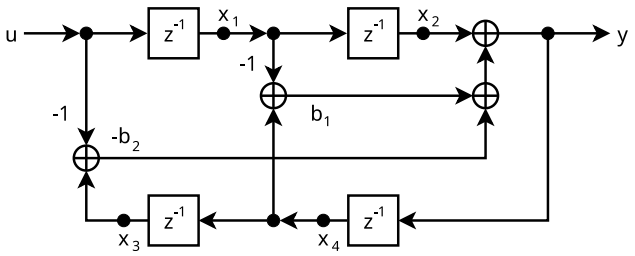

(c) Ansari-and-Lui second order all-pass filter section



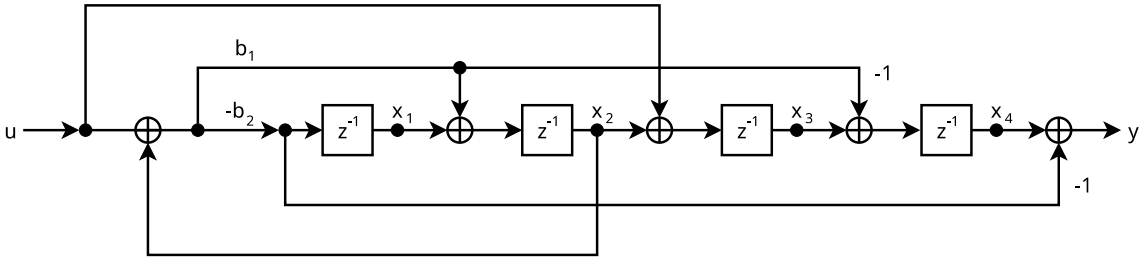

(d) Mitra-and-Hirano Type 2d second order all-pass filter section



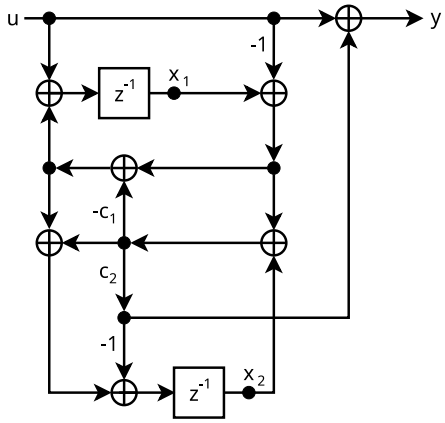

(e) Mitra-and-Hirano transposed type 2d second order all-pass filter section



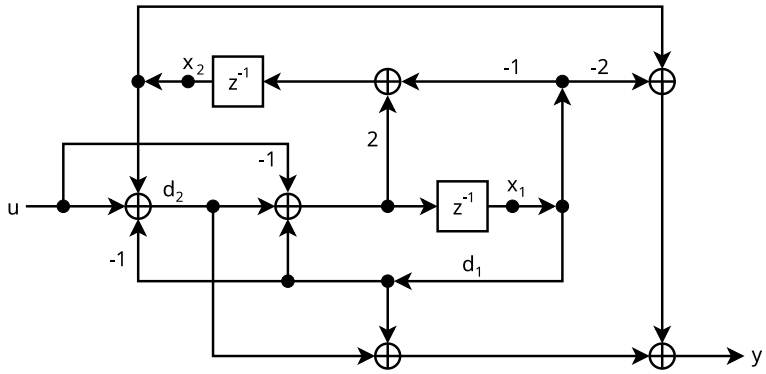

(f) Mitra-and-Hirano type 3d second order all-pass filter section




(g) Mitra-and-Hirano transposed type 3d second order all-pass filter section




(h) Stoyanov et al. low sensitvity near $z = 1$ second order all-pass filter section




(i) Ivanova-and-Stoyanov low sensitivity near $z = 0$ second order all-pass filter section

Figure 4.6: Second order all-pass filter sections

**Maximum phase gradient of some second-order all-pass filter sections**

Figure 4.7 shows the maximum of the gradient of the phase plotted against the second-order direct-form coefficients $b_1$ and $b_2$ in Equation 4.6 for real poles plotted against the real pole radius. Figure 4.8 shows the maximum of the gradient of the phase plotted against the same coefficients for complex conjugate poles plotted against pole angle.

Figure 4.9 shows the maximum of the gradient of the phase plotted against the second-order *Gray* and *Markel* and *Ansari* and *Lui* coefficients $k_1$ and $k_2$ in Equation 4.7 for real poles plotted against pole radius. Figure 4.10 shows the maximum of the gradient of the phase plotted against the same coefficients for complex conjugate poles plotted against pole angle.

Figure 4.11 shows the maximum of the gradient of the phase plotted against the *Mitra* and *Hirano* type 2d second-order all-pass filter section coefficients $b_1$ and $b_2$ in Equation 4.3a for real poles. Figure 4.12 shows the maximum of the gradient of the phase plotted against the same coefficients for complex conjugate poles. The phase gradient is not defined at a pole angle of $\frac{\pi}{2}$.

Figure 4.13 shows the maximum of the gradient of the phase plotted against the *Mitra* and *Hirano* type 3d second-order all-pass filter section coefficients $b_1$ and $b_2$ in Equation 4.3b for real poles. Figure 4.14 shows the maximum of the gradient of the phase plotted against the same coefficients for complex conjugate poles.

Figure 4.15 shows the maximum of the gradient of the phase plotted against the coefficients $c_1$ and $c_2$ of the second-order low-sensitivity near $z = 1$ section of *Stoyanov et al.* having the transfer function shown in Equation 4.10 for real poles. Figure 4.16 shows the maximum of the gradient of the phase plotted against the same coefficients for complex conjugate poles.

Figure 4.17 shows the maximum of the gradient of the phase plotted against the coefficients $d_1$ and $d_2$ of the second-order low-sensitivity near $z = 0$ section of *Ivanova* and *Stoyanov* having the transfer function shown in Equation 4.11 for real poles. Figure 4.18 shows the maximum of the gradient of the phase plotted against the same coefficients for complex conjugate poles.

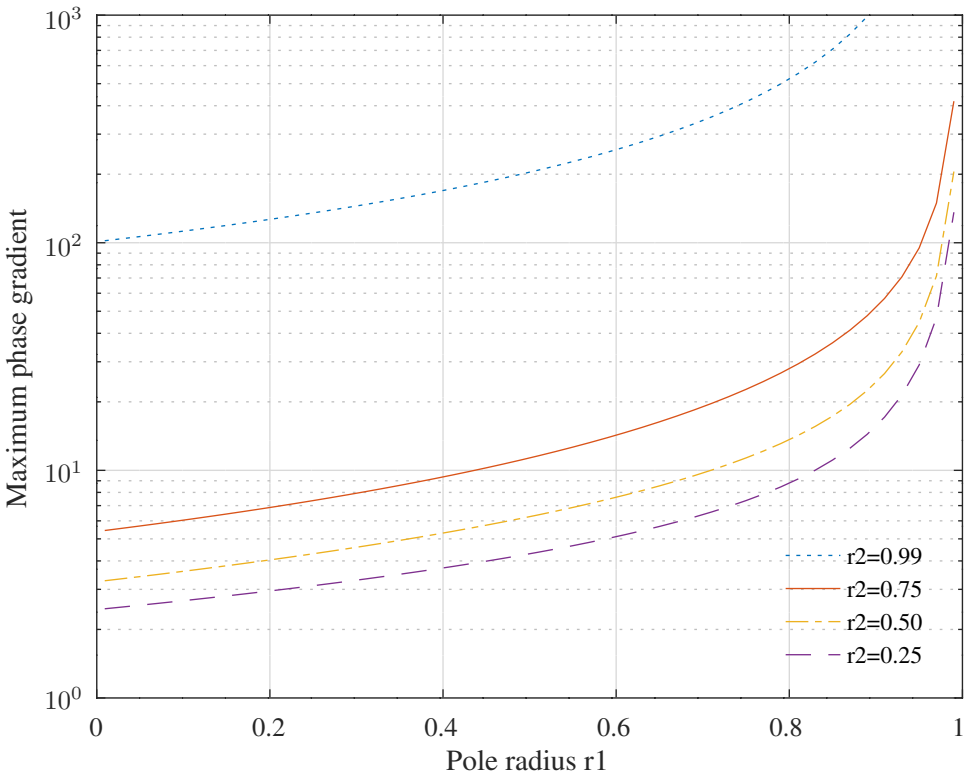



Figure 4.7: Maximum phase gradient plotted against pole radius of the second-order direct-form all-pass filter section with real poles

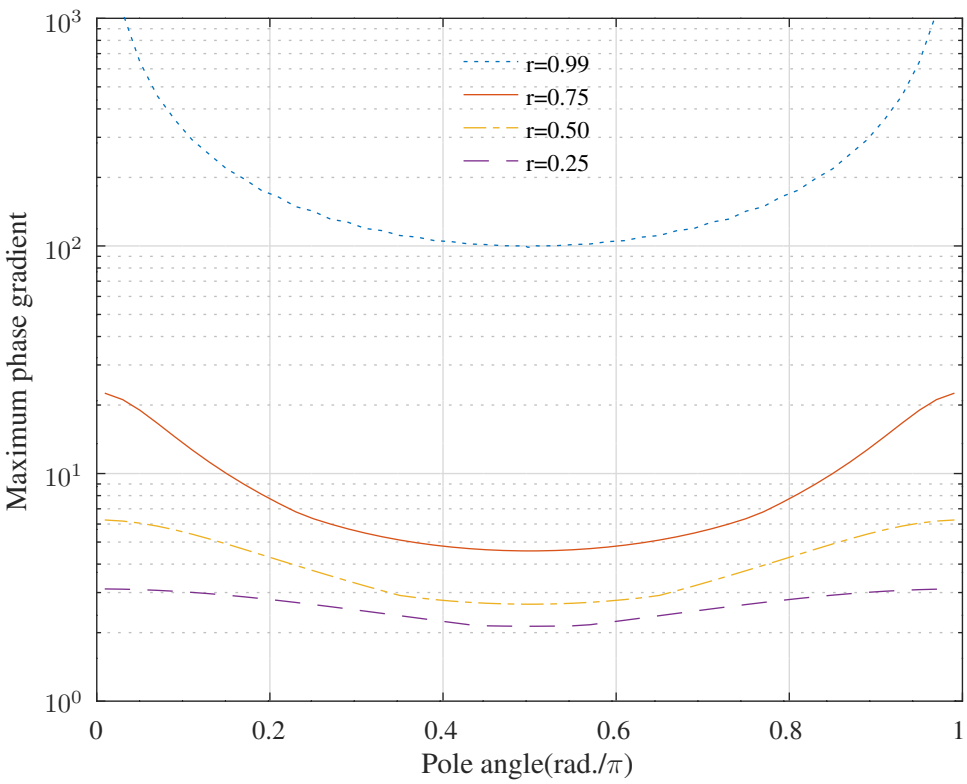



Figure 4.8: Maximum phase gradient plotted against pole angle of the second-order direct-form all-pass filter section with complex conjugate poles

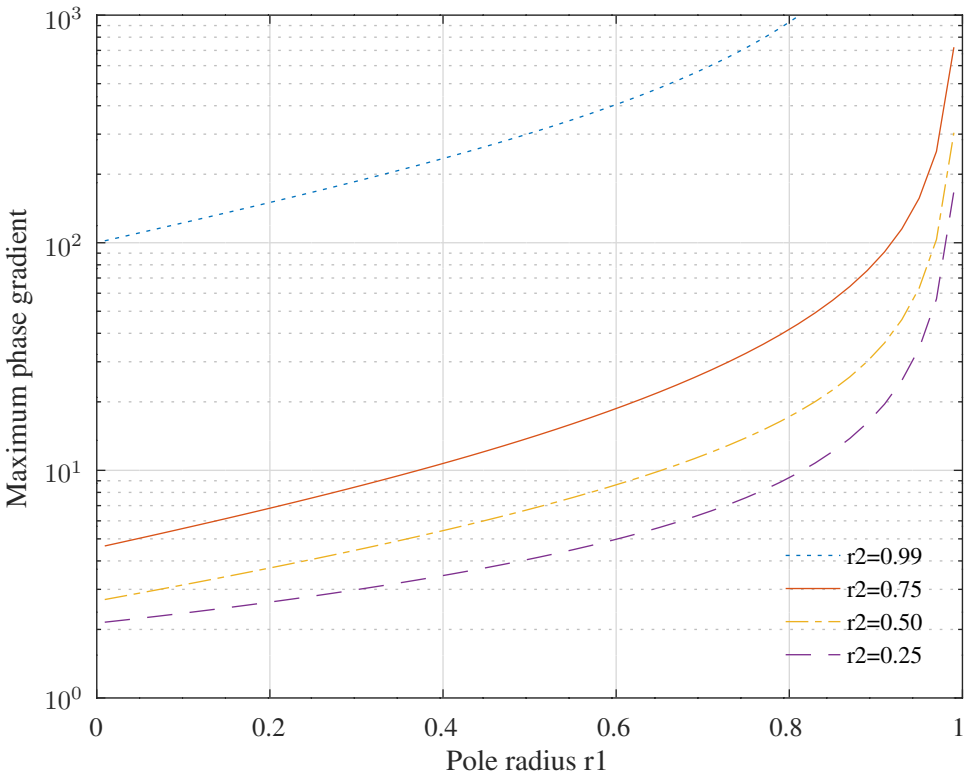



Figure 4.9: Maximum phase gradient plotted against pole radius of the second-order Gray-and-Markel and Ansari-and-Lui all-pass filter section with real poles

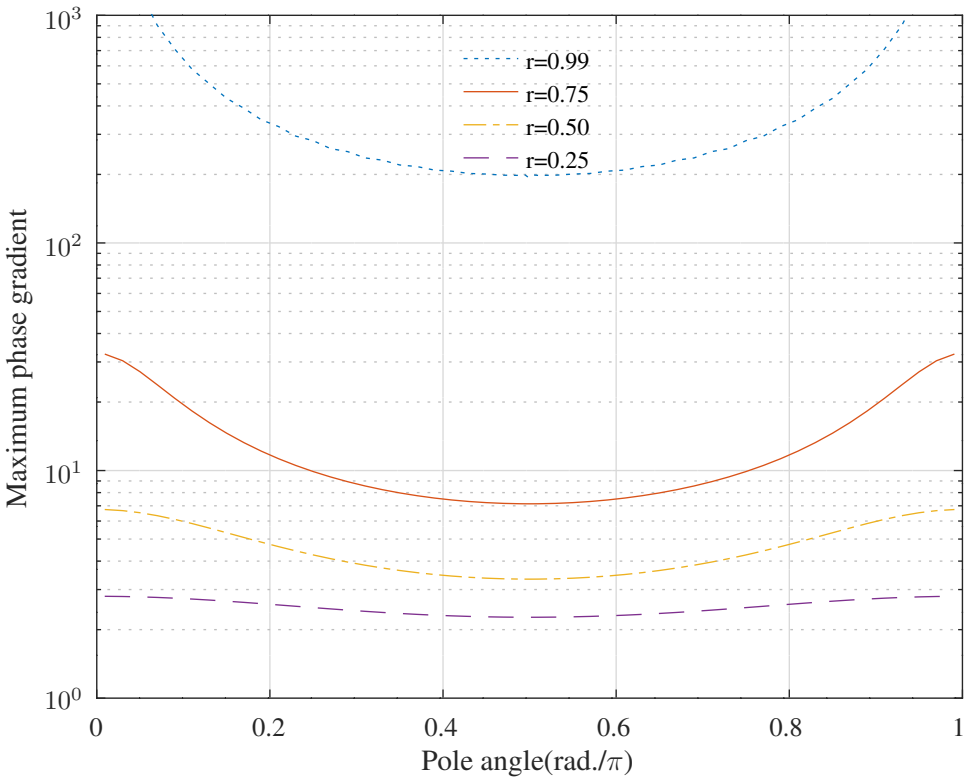



Figure 4.10: Maximum phase gradient plotted against pole angle of the second-order Gray-and-Markel and Ansari-and-Lui all-pass filter section with complex conjugate poles

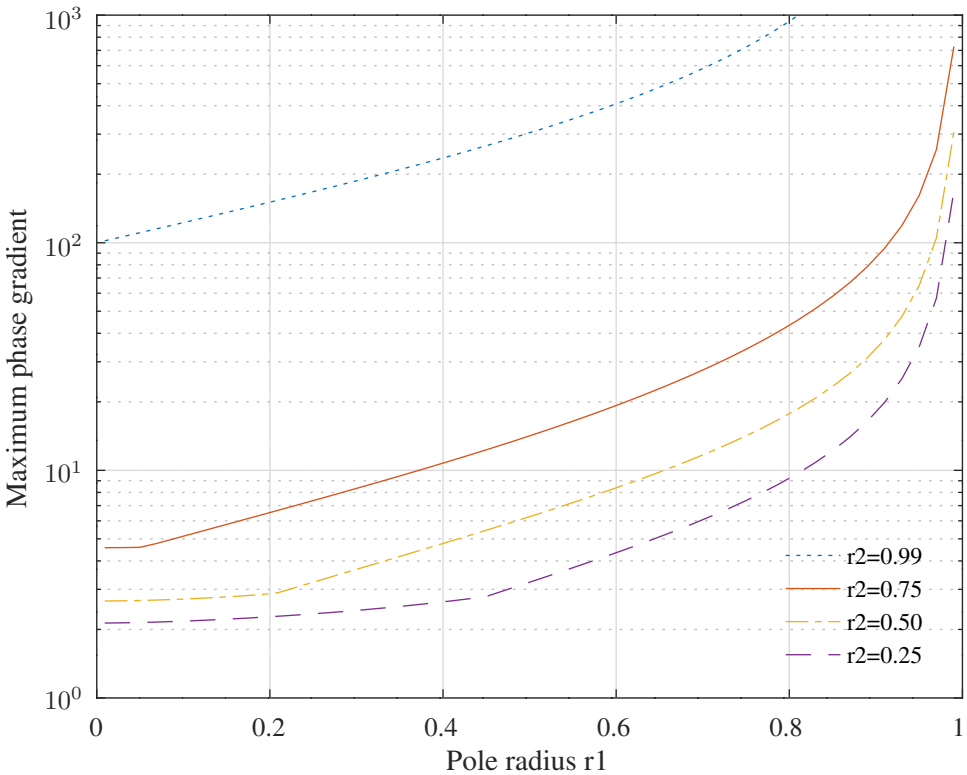



Figure 4.11: Maximum phase gradient plotted against pole radius of the second-order Mitra-and-Hirano type 2d all-pass filter section with real poles

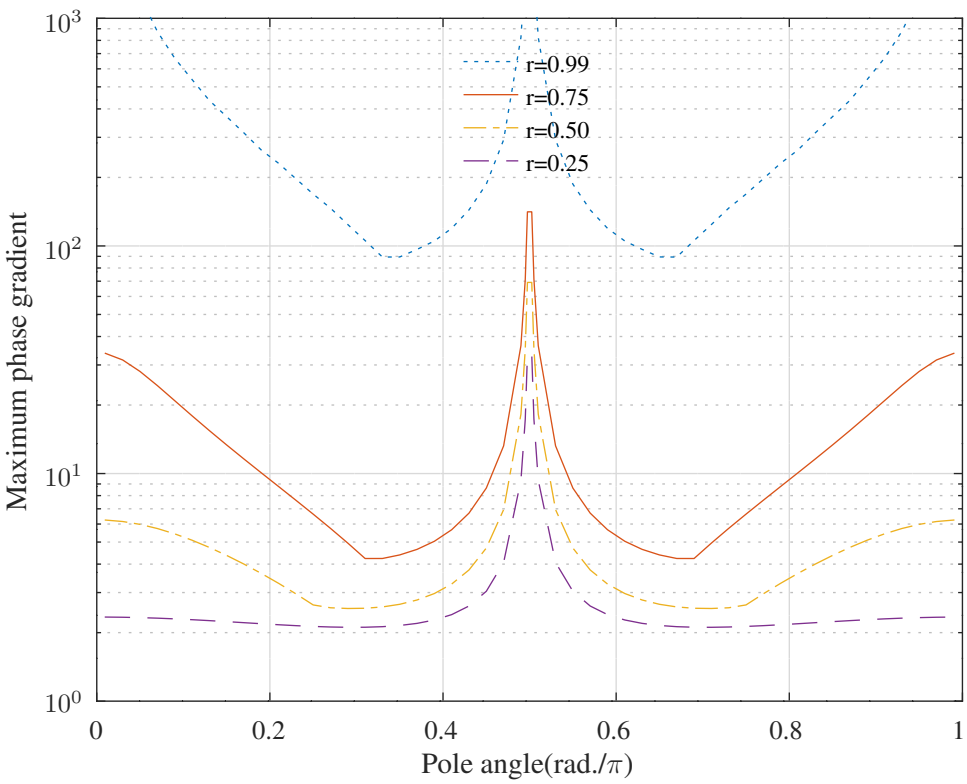



Figure 4.12: Maximum phase gradient plotted against pole angle of the second-order Mitra-and-Hirano type 2d all-pass filter section with complex conjugate poles

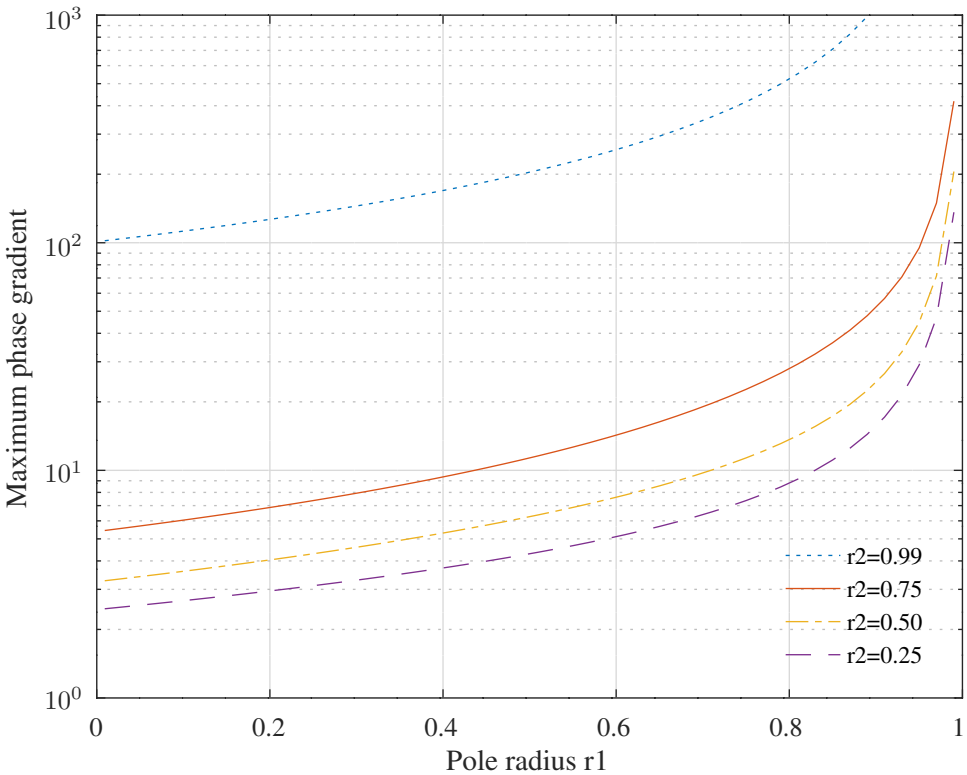



Figure 4.13: Maximum phase gradient plotted against pole radius of the second-order Mitra-and-Hirano type 3d all-pass filter section with real poles

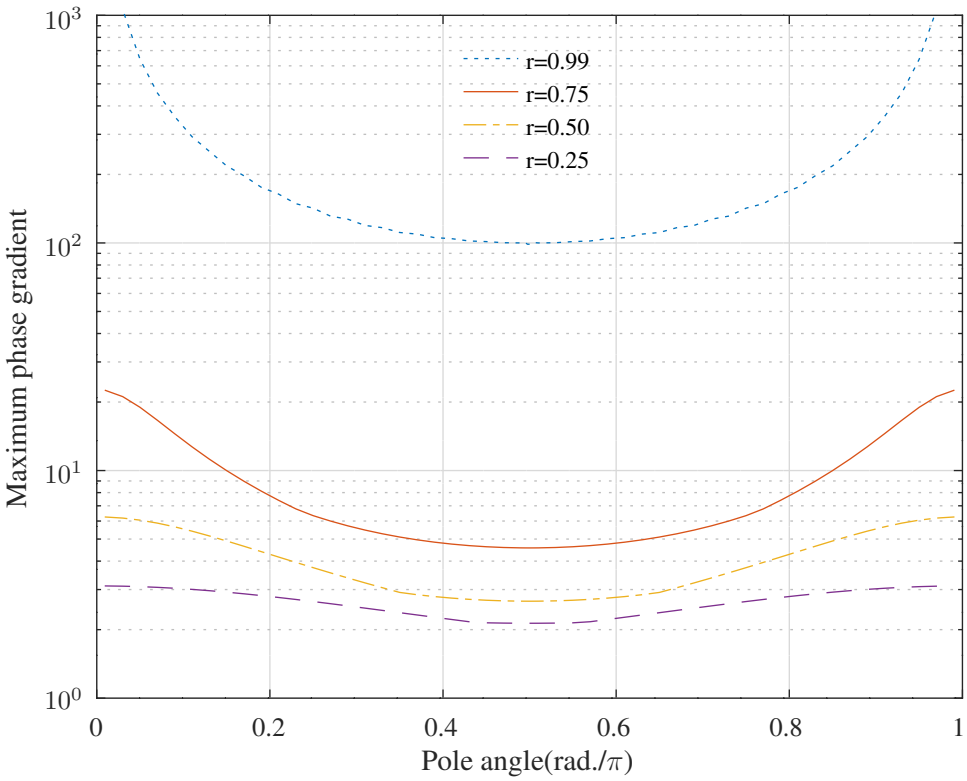



Figure 4.14: Maximum phase gradient plotted against pole angle of the second-order Mitra-and-Hirano type 3d all-pass filter section with complex conjugate poles

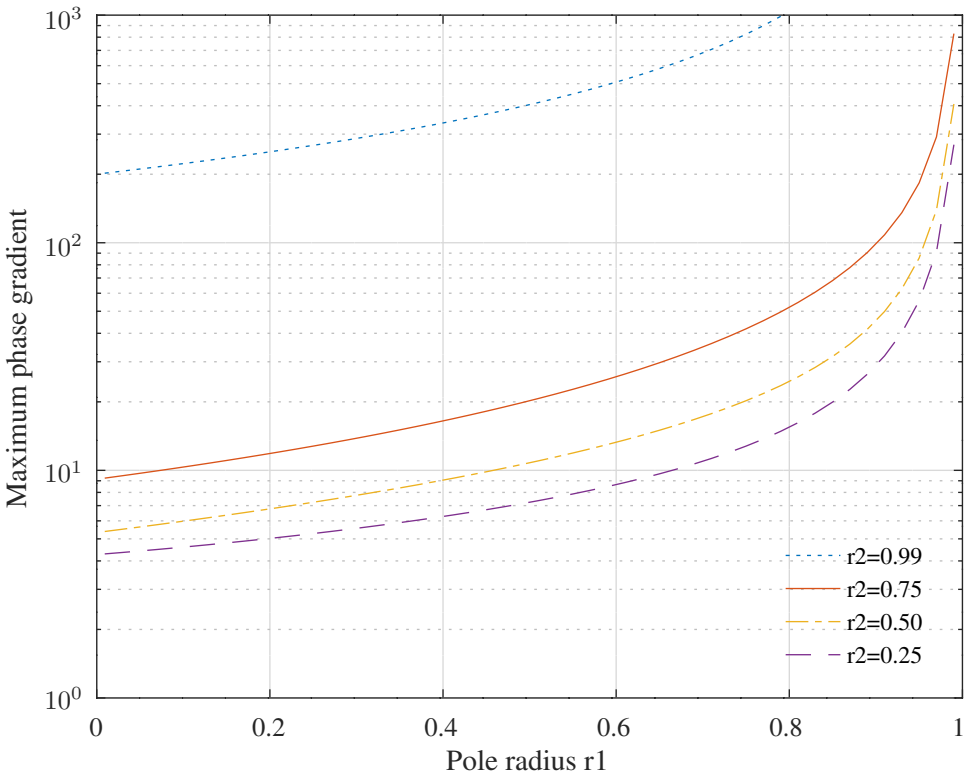



Figure 4.15: Maximum phase gradient plotted against pole radius of the second-order Stoyanov low-sensitivity mear $z = 1$ all-pass filter section with real poles

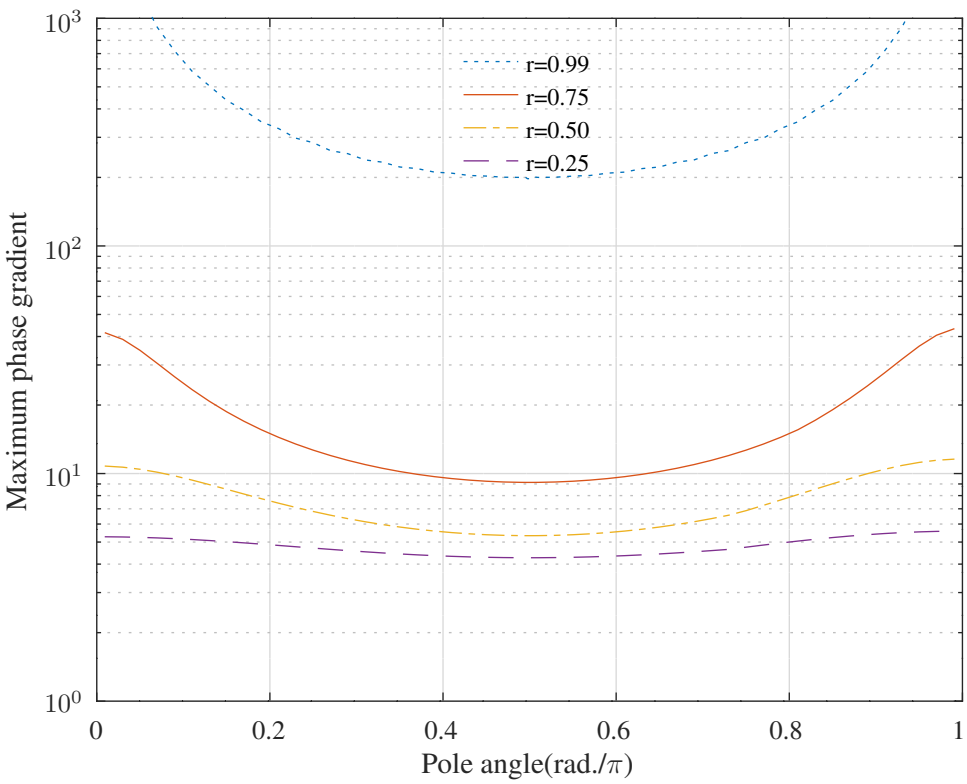



Figure 4.16: Maximum phase gradient plotted against pole angle of the second-order Stoyanov low-sensitivity near $z = 1$ all-pass filter section with complex conjugate poles

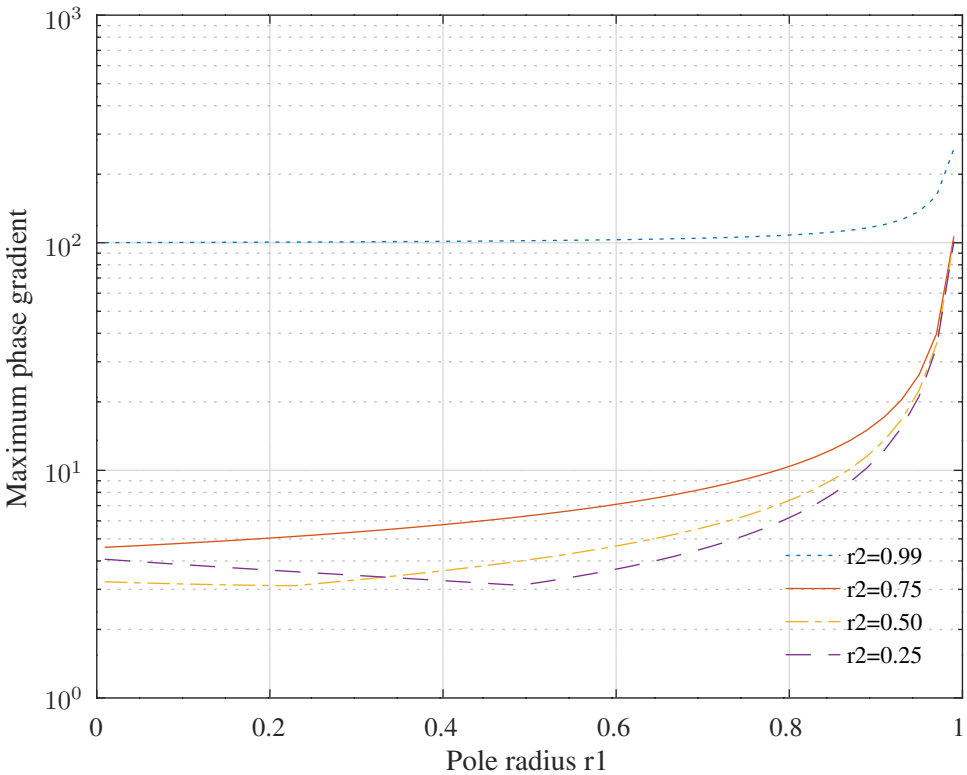



Figure 4.17: Maximum phase gradient plotted against pole radius of the second-order Ivanova and Stoyanov low-sensitivity near $z = 0$ all-pass filter section with real poles

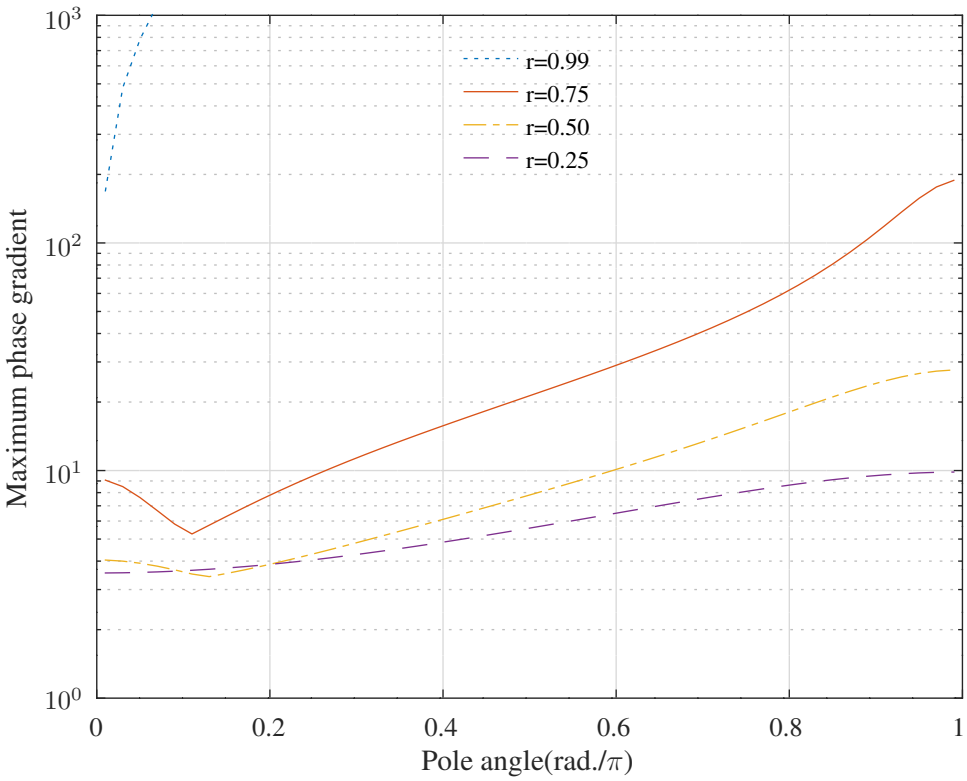



Figure 4.18: Maximum phase gradient plotted against pole angle of the second-order Ivanova and Stoyanov low-sensitivity near $z = 0$ all-pass filter section with complex conjugate poles

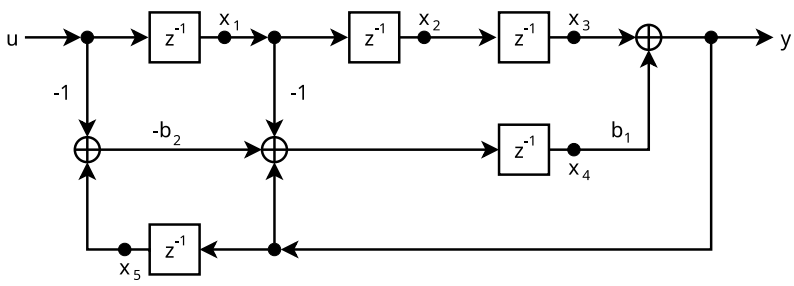



Figure 4.19: Retimed Mitra-and-Hirano type 2d second-order direct form all-pass filter section

**Noise gain of some second-order all-pass filter sections**

As for the first-order direct-form section the noise gain of the scaled second-order direct-form section can be estimated with a slowed and retimed realisation.

Figure 4.20 shows the noise gain of a scaled realisation with real poles of Equation 4.6. Figure 4.21 shows the noise gain of a scaled realisation with complex conjugate poles.

As for the second-order direct-form section the noise gain of the scaled second-order *Gray* and *Markel* section can be estimated with a slowed and retimed realisation. As for the first-order *Gray* and *Markel* section, the $\epsilon$ coefficients do not alter the transfer function or noise gain but must be selected for effective internal state scaling in a fixed-point implementation. Figure 4.22 shows the noise gain of a scaled realisation with real poles. Figure 4.23 shows the noise gain of a scaled realisation with complex conjugate poles.

The noise gain of the scaled second-order *Ansari* and *Lui* section can be estimated with a slowed and retimed realisation. Figure 4.24 shows the noise gain of a scaled realisation with real poles. Figure 4.25 shows the noise gain of a scaled realisation with complex conjugate poles.

The noise gain of the scaled second-order *Mitra* and *Hirano* type 2d section can be estimated with the retimed realisation shown in Figure 4.19. The section is retimed by adding a $z^{-1}$delay at the output and then redistributing the delay "backwards" until the output of the $b_1$ multiplier is registered.

Figure 4.26 shows the noise gain of a scaled realisation with real poles. Figure 4.27 shows the noise gain of a scaled realisation with complex conjugate poles.

The noise gain of the scaled second-order *Mitra* and *Hirano* type 3d section can be estimated without slowing or retiming the retimed realisation. Figure 4.28 shows the noise gain of a scaled realisation with real poles. Figure 4.29 shows the noise gain of a scaled realisation with complex conjugate poles.

The round-off noise of the second-order Stoyanov low-sensitivity section can be estimated without slowing and retiming the realisation. Figure 4.30 shows the noise gain of a scaled realisation with real poles. Figure 4.31 shows the noise gain of a scaled realisation with complex conjugate poles.

The round-off noise of the second-order *Ivanova* and *Stoyanov* low-sensitivity near $z = 0$ section can be estimated with a slowed and retimed realisation. Figure 4.32 shows the noise gain of a scaled realisation with real poles. Figure 4.33 shows the noise gain of a scaled realisation with complex conjugate poles.

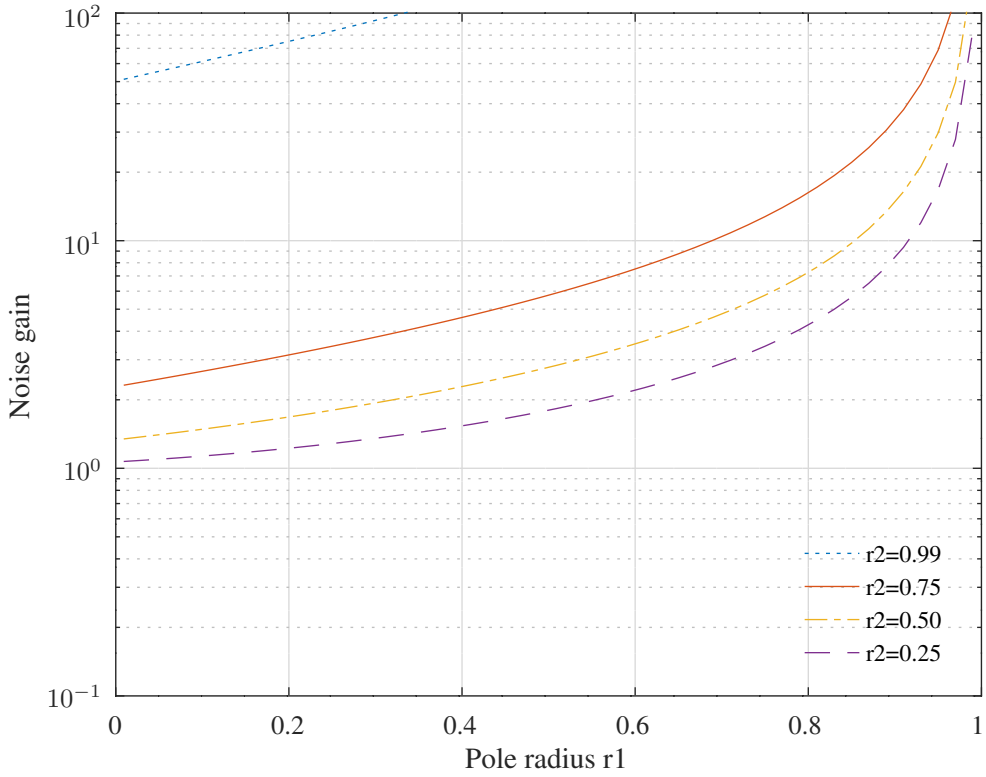



Figure 4.20: Noise gain of the scaled and retimed second-order direct-form all-pass filter section with real poles

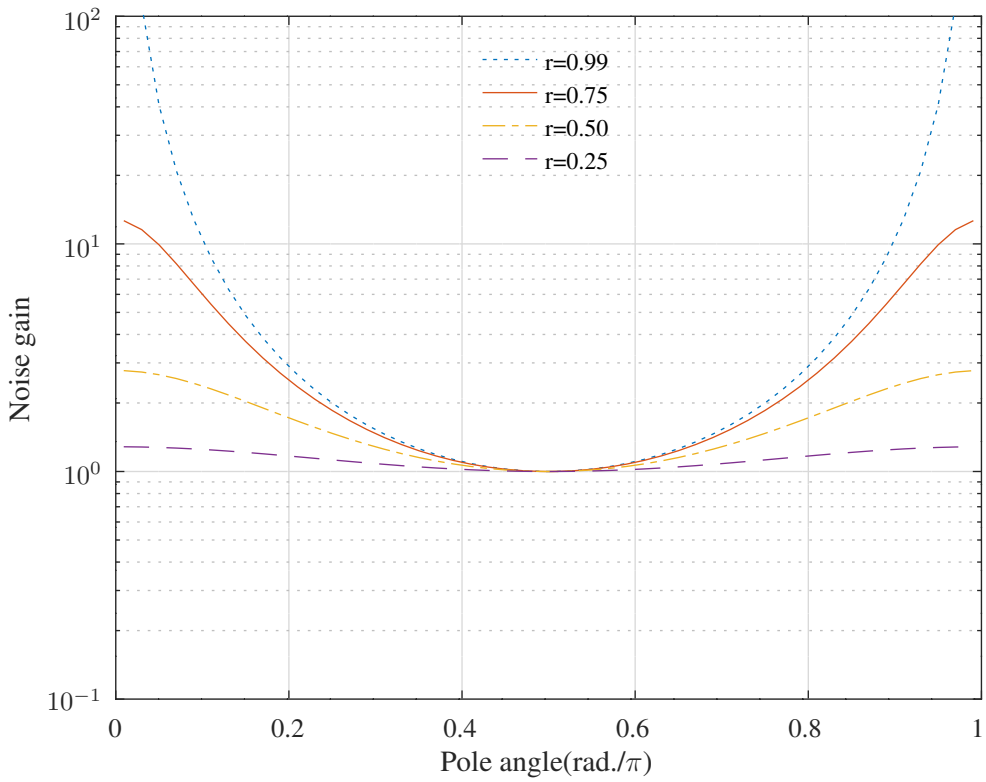



Figure 4.21: Noise gain of the scaled and retimed second-order direct-form all-pass filter section with complex conjugate poles

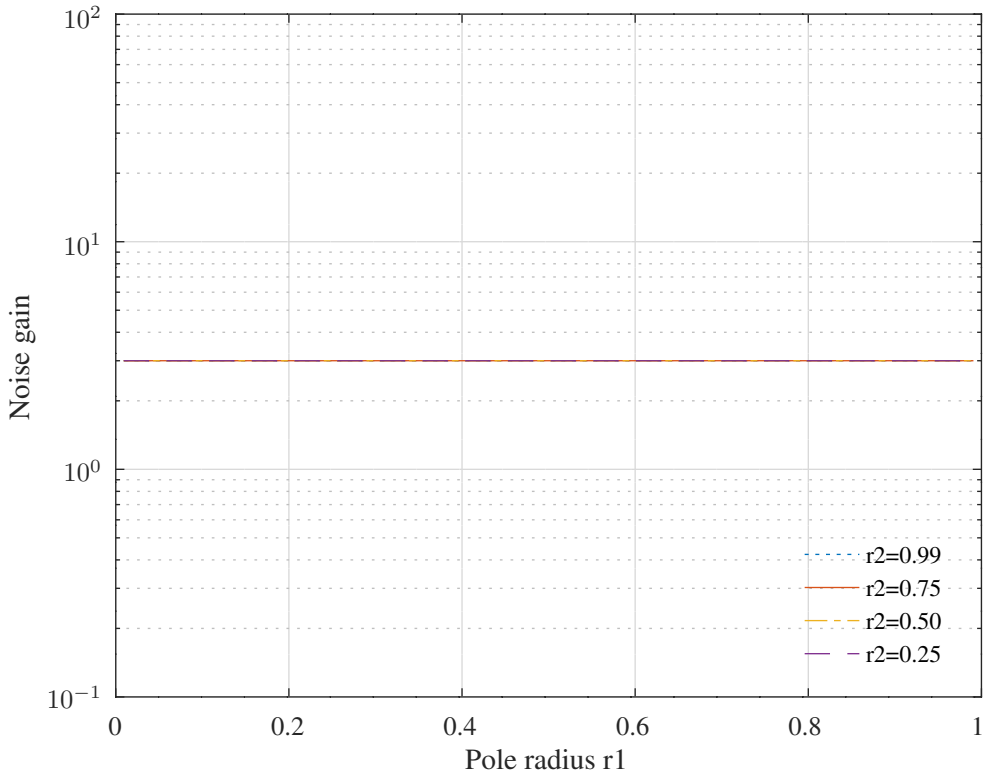



Figure 4.22: Noise gain of the scaled and retimed second-order Gray-and-Markel all-pass filter section with real poles

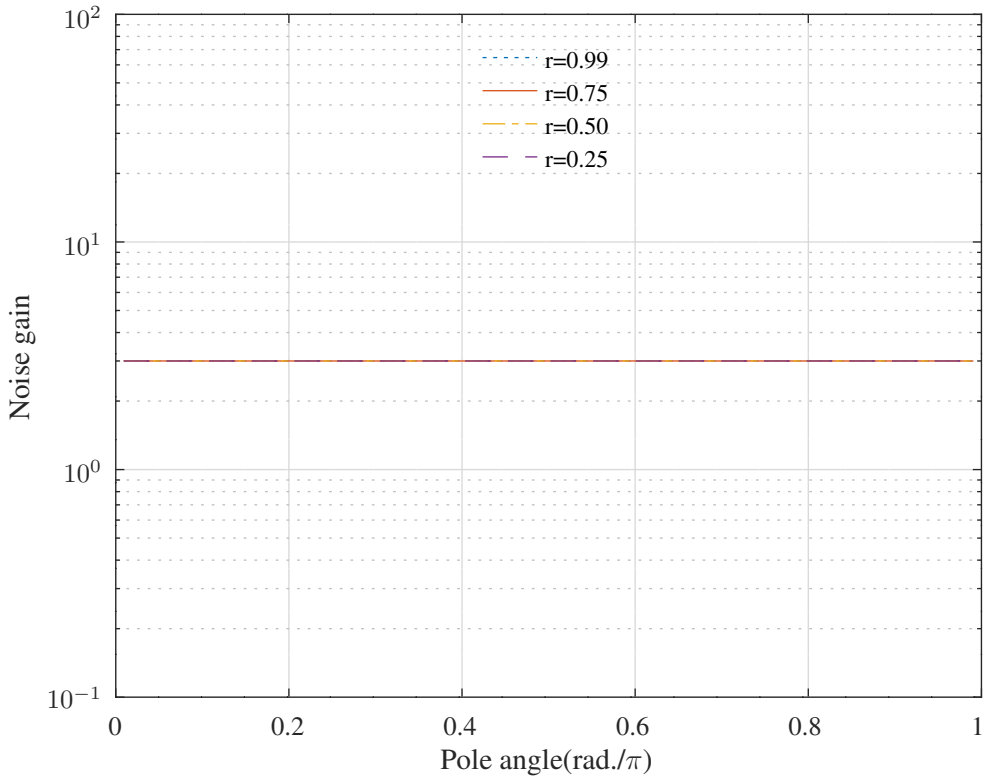



Figure 4.23: Noise gain of the scaled and retimed second-order Gray-and-Markel all-pass filter section with complex conjugate poles

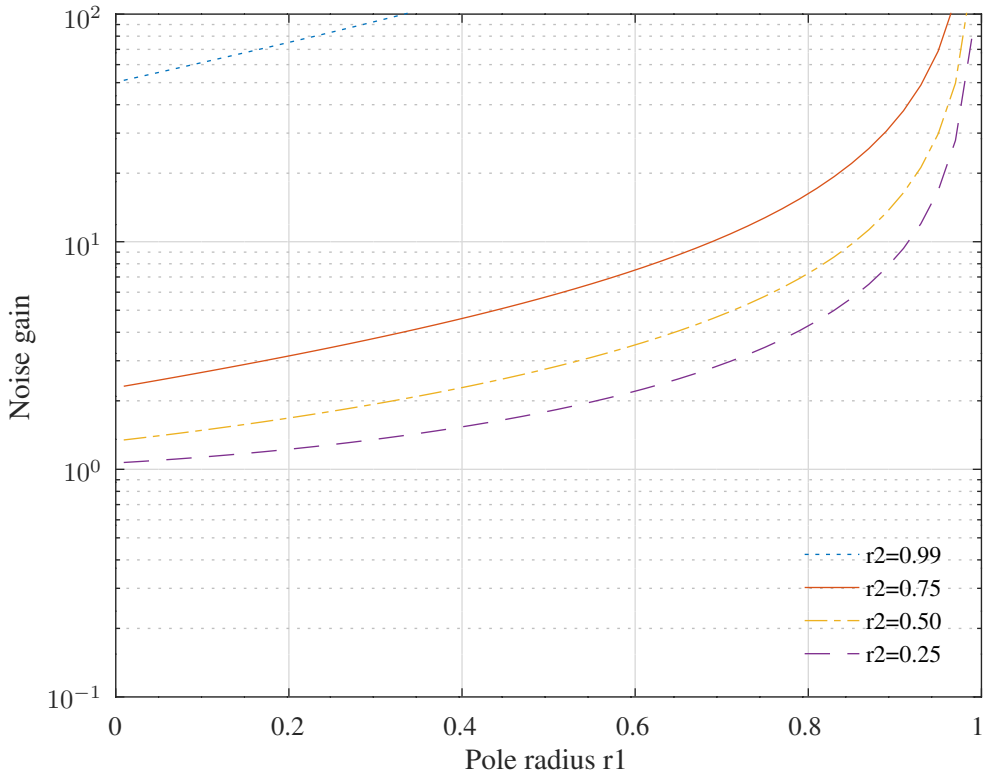



Figure 4.24: Noise gain of the scaled and retimed second-order Ansari-and-Lui all-pass filter section with real poles

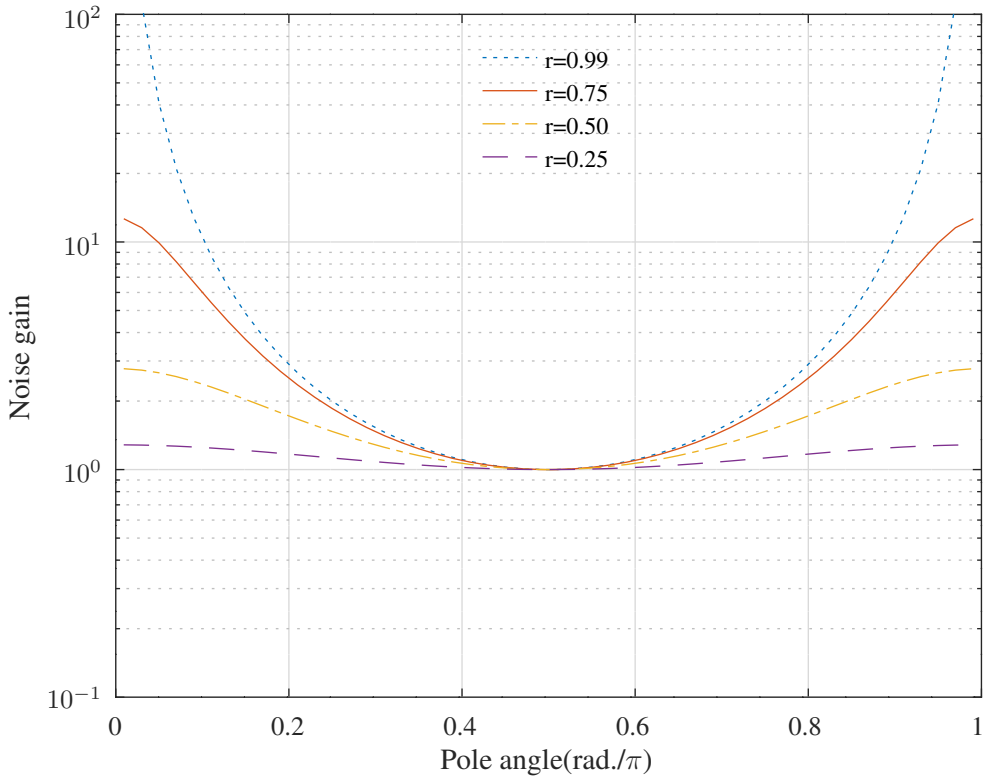



Figure 4.25: Noise gain of the scaled and retimed second-order Ansari-and-Lui all-pass filter section with complex conjugate poles

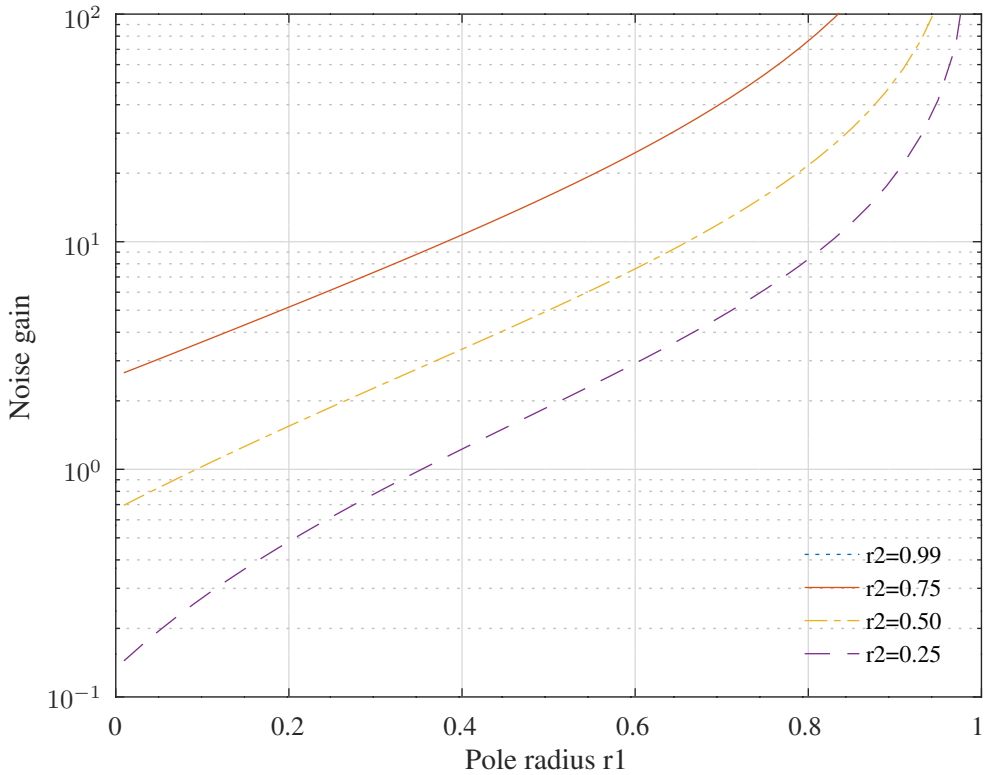



Figure 4.26: Noise gain of the scaled and retimed second-order Mitra-and-Hirano type 2d all-pass filter section with real poles

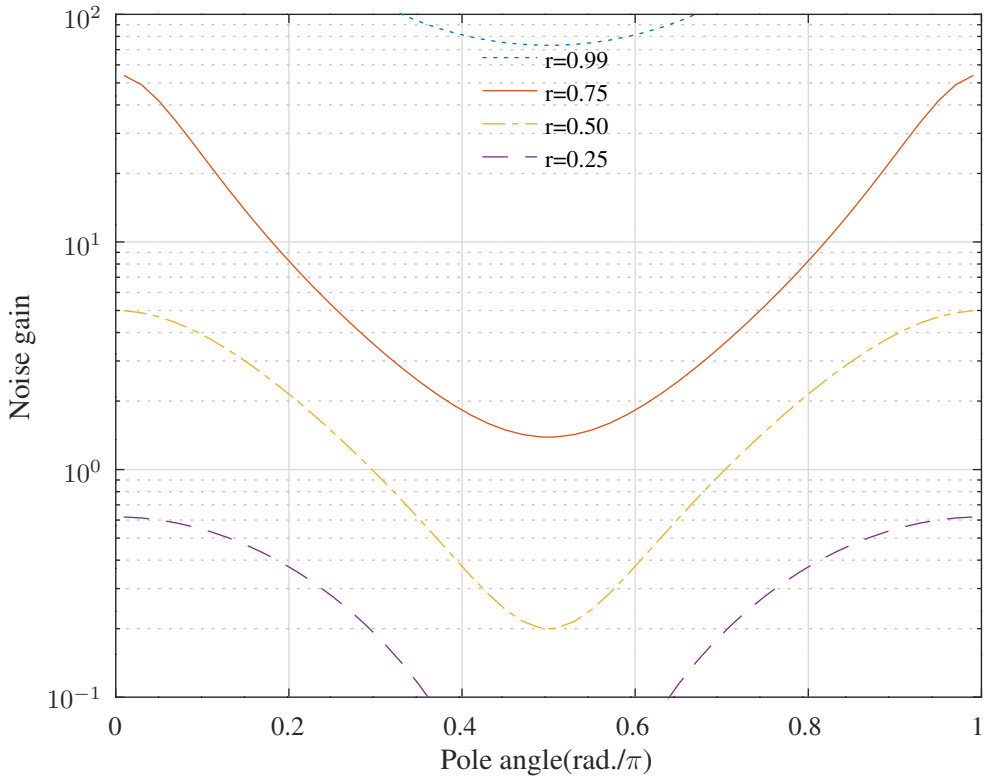



Figure 4.27: Noise gain of the scaled and retimed second-order Mitra-and-Hirano type 2d all-pass filter section with complex conjugate poles

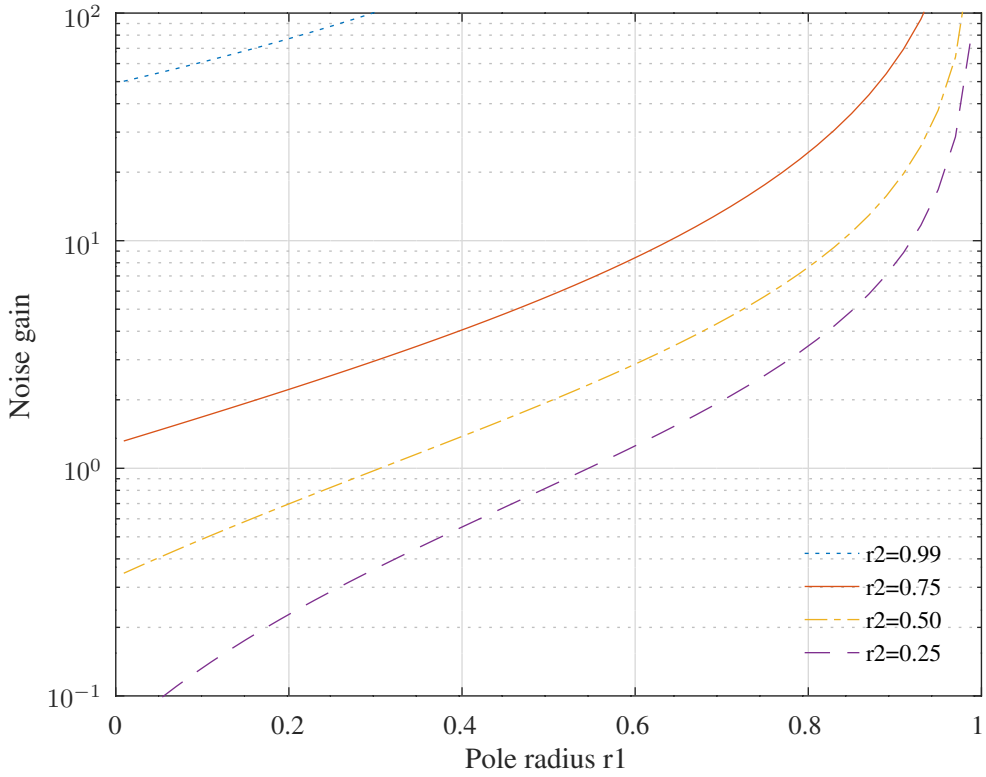



Figure 4.28: Noise gain of the scaled second-order Mitra-and-Hirano type 3d all-pass filter section with real poles

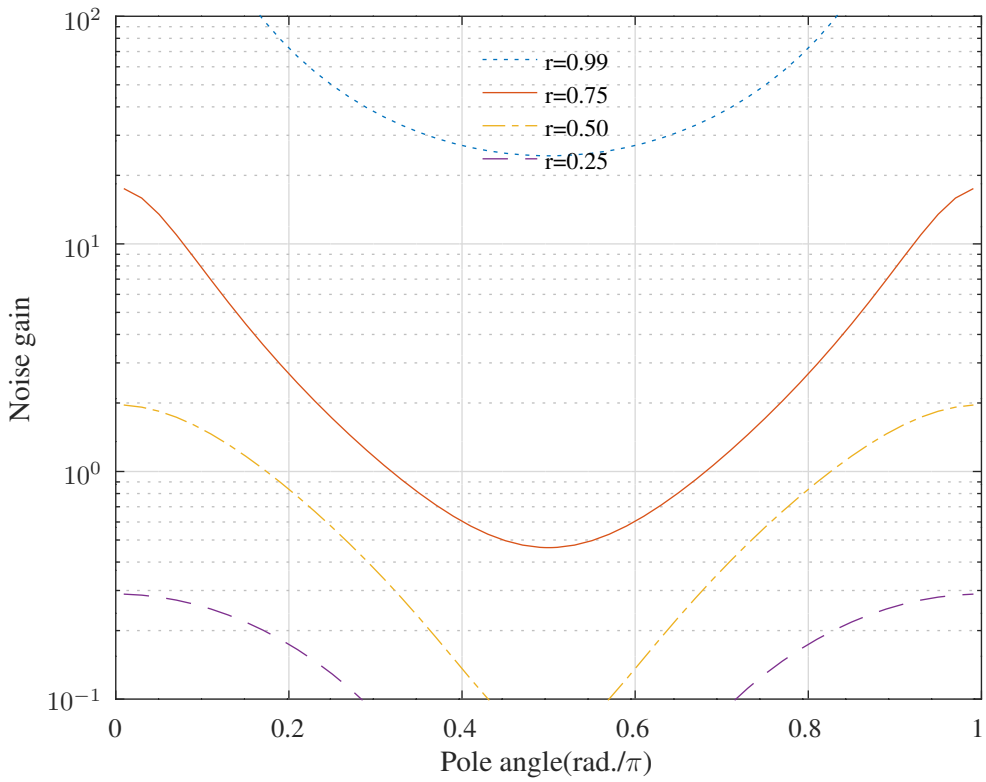



Figure 4.29: Noise gain of the scaled second-order Mitra-and-Hirano type 3d all-pass filter section with complex conjugate poles

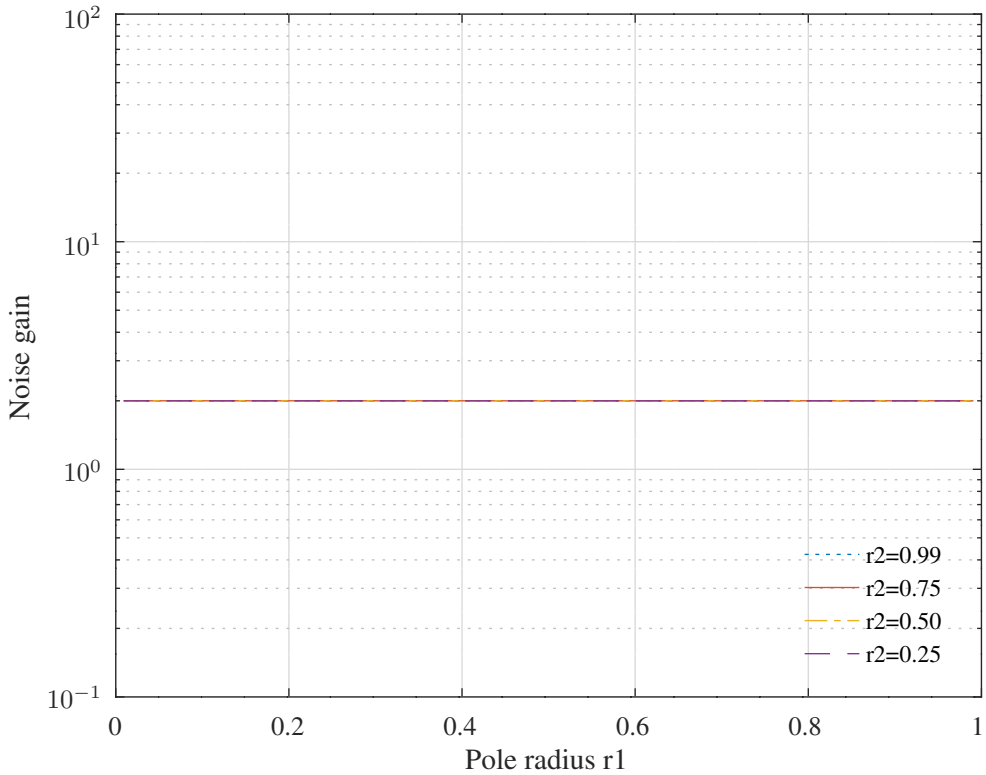



Figure 4.30: Noise gain of the scaled second-order Stoyanov low-sensitivity near $z = 1$ all-pass filter section with real poles

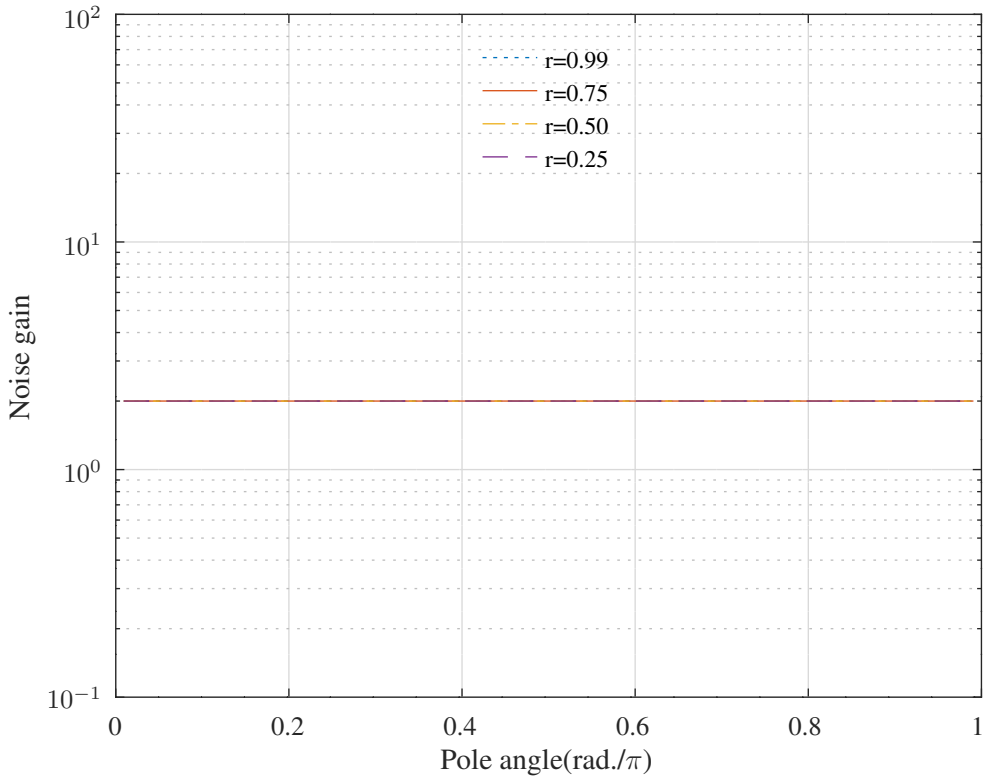



Figure 4.31: Noise gain of the scaled second-order Stoyanov low-sensitivity near $z = 1$ all-pass filter section with complex conjugate poles

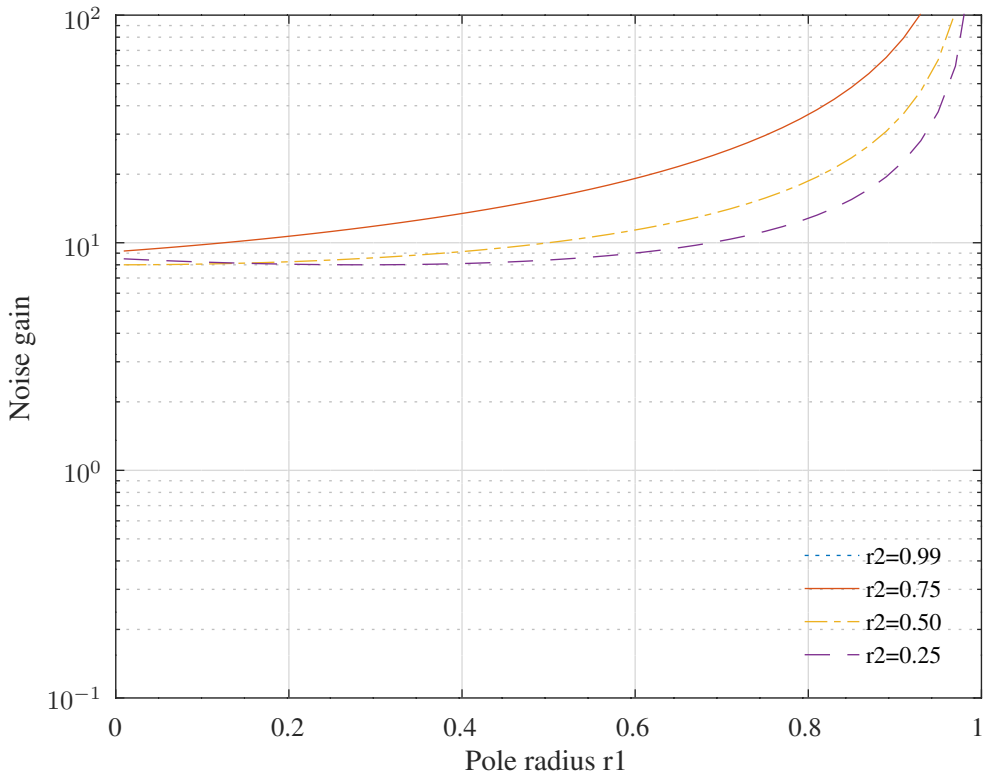



Figure 4.32: Noise gain of the scaled and retimed second-order Ivanova-and-Stoyanov low-sensitivity near $z = 0$ all-pass filter section with real poles

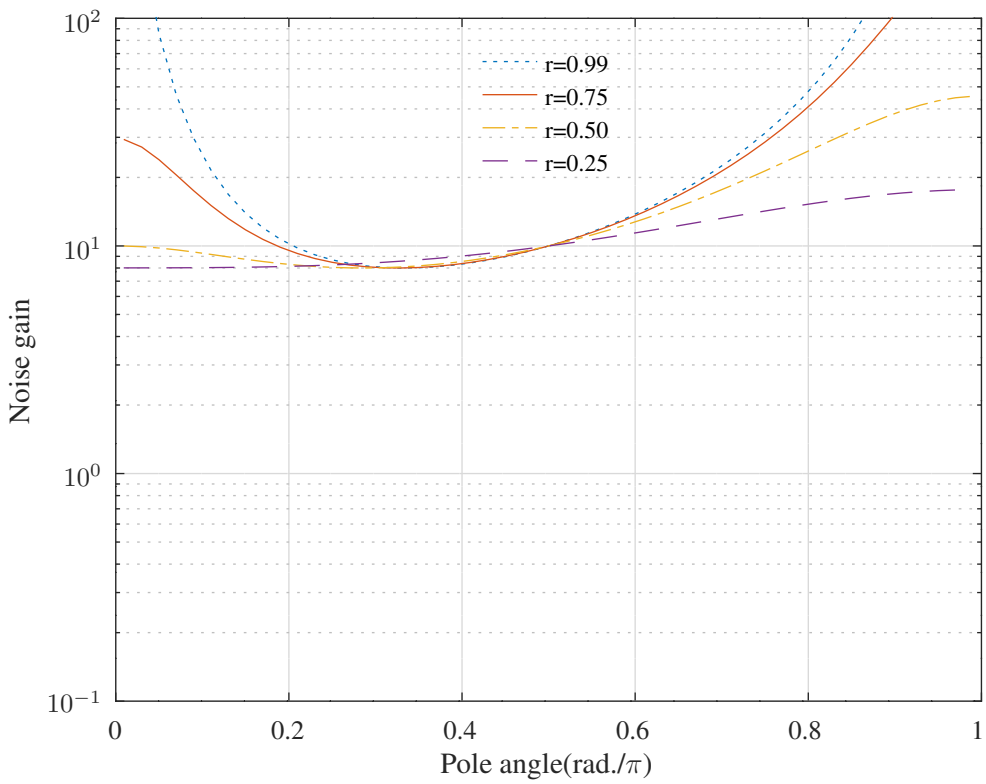



Figure 4.33: Noise gain of the scaled and retimed second-order Ivanova-and Stoyanov low-sensitivity near $z = 0$ all-pass filter section with complex conjugate poles

# Chapter 5

# Filter synthesis by the Schur decomposition

This chapter follows *Parhi* [98, Chapter 12].

## 5.1 The Schur algorithm

From *Parhi* [98, 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
>
> $$\{\Phi_i(z), i = N, N-1, \ldots, 0\}$$
>
> 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.

### 5.1.1 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:

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

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

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

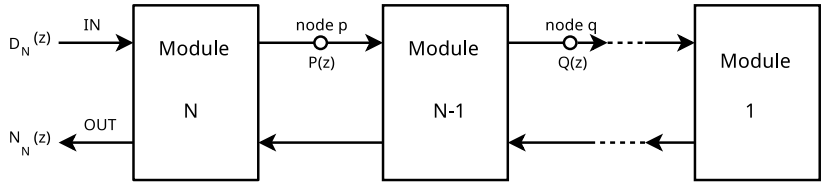



Figure 5.1: A filter structure implementing $H(z) = \frac{N_N(z)}{P_N(z)}$ (after *Parhi* [98, Fig. 12.1])

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:

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

For $\Phi_N(z)$ to be a Schur polynomial, $|k_i| < 1$ for each polynomial in the set $\{\Phi_N(z), \Phi_{N-1}(z), \ldots, \Phi_1(z)\}$. *Parhi* [98, 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 $2 \times 2$ submatrices formed by the first column and each succeeding column in the matrix:

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

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* abitrary precision floating point library [101, 164] written by *Fousse et al.* [1]. The mantissa precision is set to 256 bits.

### 5.1.2 Orthonormality of Schur Polynomials

The filter structure shown in Figure 5.1 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:

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

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

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

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:

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

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 transpose. 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 $\alpha$, $\beta$ and $\gamma$:

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

3. Positive norm:

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

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

Some identities:

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

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

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

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:

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

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:

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

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* [98, Appendix D]. The reverse Schur polynomials are not orthonormal. However:

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

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

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

also form an orthonormal basis. This basis can be used to synthesise an alternative lattice structure. *Parhi* [98, 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.

### 5.1.3 Polynomial Expansion Algorithm

The Schur polynomials in the set $\{\Phi_N(z), \ldots, \Phi_0(z)\}$ 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 5.1 shows the Schur expansion of an arbitrary polynomial in a Schur basis.

**Algorithm 5.1** Schur polynomial expansion (see *Parhi* [98, 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** $i = m, m-1, \ldots, 0$ **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* abitrary precision floating point library written by *Fousse et al.* [1]. The mantissa precision is set to 256 bits.

### 5.1.4 Power calculation using the Schur algorithm

For the lattice filter structure in Figure 5.1, 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

$$P_p = \sum_i c_i^2$$

## 5.2 Derivation of Digital Lattice Filters

The Schur polynomials are obtained by the degree reduction procedure

$$\Phi_{i-1}(z) = \frac{z^{-1}\{\Phi_i(z) - k_i \Phi_i^*(z)\}}{s_i} \tag{5.1}$$

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 $\Lambda_i$, and if $s_i = 1 - k_i^2$ is chosen, by $\Psi_i(z)$.

Note that since

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

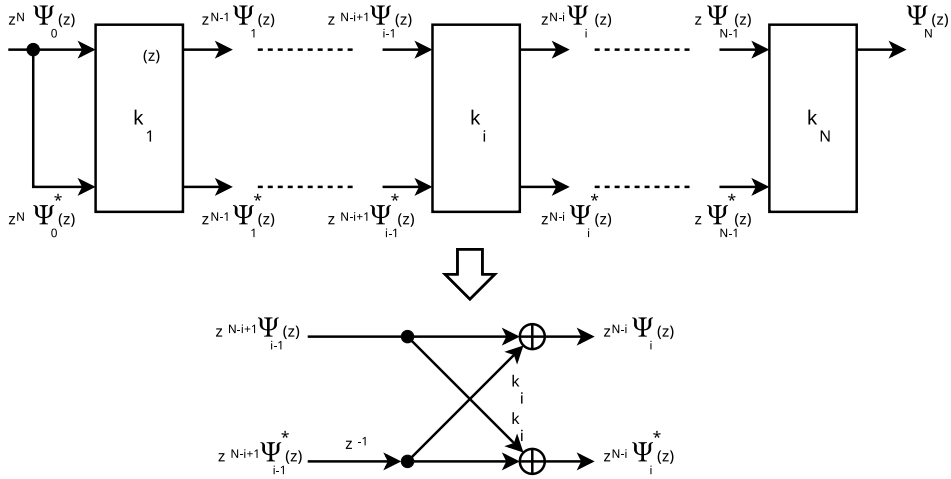



Figure 5.2: FIR filter structure (after *Parhi* [98, Fig. 12.8])

by rearranging

$$\Phi_i(z) = \frac{s_i}{1-k_i^2}\left\{z\Phi_{i-1}(z) + k_i\Phi_{i-1}^*(z)\right\}$$
$$\Phi_i^*(z) = \frac{s_i}{1-k_i^2}\left\{zk_i\Phi_{i-1}(z) + \Phi_{i-1}^*(z)\right\}$$

### 5.2.1 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

$$\Psi_{N-1}(z) = \frac{z^{-1}\left\{\Psi_N(z) - k_N\Psi_N^*(z)\right\}}{1-k_N^2} \tag{5.2}$$

The reverse Schur polynomial is

$$\Psi_{N-1}^*(z) = z^{N-1}\Psi_{N-1}(z^{-1}) \tag{5.3}$$
$$= \frac{\Psi_N^*(z) - k_N\Psi_N(z)}{1-k_N^2} \tag{5.4}$$

so

$$\begin{aligned}\Psi_N^*(z) &= \left(1-k_N^2\right)\Psi_{N-1}^*(z) + k_N\Psi_N(z)\\ &= \Psi_{N-1}^*(z) + k_N\left\{\Psi_N - k_N\Psi_{N-1}^*(z)\right\}\end{aligned} \tag{5.5}$$

Substituting Equation 5.5 into Equation 5.2

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

so

$$\Psi_N(z) = z\Psi_{N-1}(z) + k_N\Psi_{N-1}^*(z) \tag{5.6}$$

and, applying the definition of the reverse polynomial

$$\Psi_N^*(z) = zk_N\Psi_{N-1}(z) + \Psi_{N-1}^*(z) \tag{5.7}$$

Equations 5.6 and 5.7 represent the FIR filter shown in Figure 5.2. This structure has twice the number of multipliers of the direct form structure. However, the structure is useful for implementing adaptive filters. Equations 5.2 and 5.4 represent the IIR filter section shown in Figure 5.3. 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

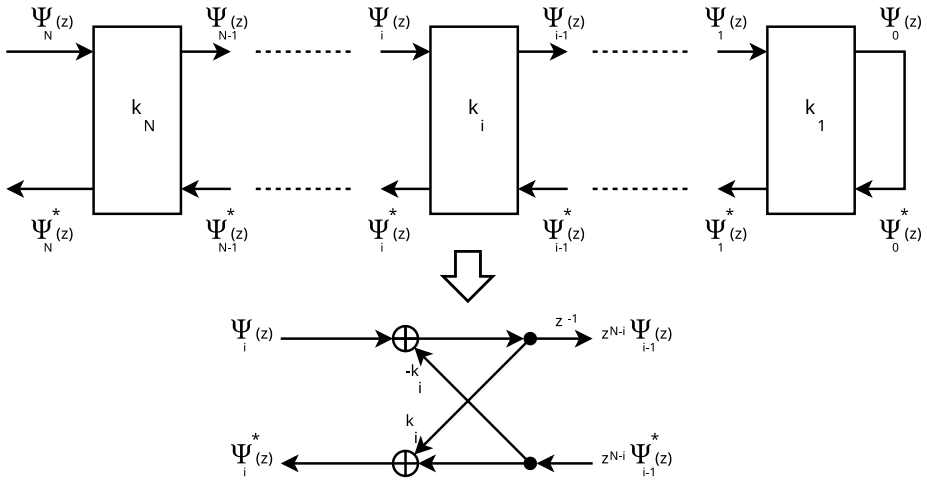



Figure 5.3: All-pass and all-pole filter structure (after *Parhi* [98, Fig. 12.5])

$$\Psi_N(z) = \Phi_N(z)$$
$$\Psi_i(z) = \frac{\Phi_i(z)}{\sqrt{\left(1-k_N^2\right)\left(1-k_{N-1}^2\right)\cdots\left(1-k_{i+1}^2\right)}}, \quad 0 \le i < N$$

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 $\Psi_i$ are orthogonal but not orthonormal since

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

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.

## 5.3 Derivation of the One-Multiplier IIR Lattice Filter

If $s_i = 1 - \epsilon_i k_i$ in Equation 5.1 then

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

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

$$\Lambda_{N-1}(z) = \frac{z^{-1}\left\{\Lambda_N(z) - k_N\Lambda_N^*(z)\right\}}{1-\epsilon_N k_N}$$
$$\Lambda_{N-1}^*(z) = \frac{\Lambda_N^*(z) - k_N\Lambda_N(z)}{1-\epsilon_N k_N}$$

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

$$\Lambda_N^*(z) = k_N\Lambda_N(z) + \left(1-\epsilon_N k_N\right)\Lambda_{N-1}^*(z) \tag{5.8}$$
$$\Lambda_{N-1}(z) = z^{-1}\left\{\left(1+\epsilon_N k_N\right)\Lambda_N - k_N\Lambda_{N-1}^*(z)\right\} \tag{5.9}$$

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

$$\Lambda_N(z) = \Phi_N(z)$$

$$\Lambda_i(z) = \Phi_i(z)\sqrt{\frac{(1+\epsilon_N k_N)(1+\epsilon_{N-1}k_{N-1})\cdots(1+\epsilon_{i+1}k_{i+1})}{(1-\epsilon_N k_N)(1-\epsilon_{N-1}k_{N-1})\cdots(1-\epsilon_{i+1}k_{i+1})}}$$

where $0 \le i < N$. 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 $\Lambda_i$ are orthogonal but not orthonormal since

$$\langle \Lambda_i(z), \Lambda_i(z)\rangle = \langle \Lambda_i^*(z), \Lambda_i^*(z)\rangle \tag{5.10}$$

$$= \frac{(1+\epsilon_N k_N)(1+\epsilon_{N-1}k_{N-1})\cdots(1+\epsilon_{i+1}k_{i+1})}{(1-\epsilon_N k_N)(1-\epsilon_{N-1}k_{N-1})\cdots(1-\epsilon_{i+1}k_{i+1})} \tag{5.11}$$

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 $m = l-1, l-2, \ldots, 0$ and again for $m = l+1, l+2, \ldots, N$. 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 5.2 shows the method used to assign the sign parameters in the Octave function *schurOneMscale*.

**Algorithm 5.2** One-multiplier lattice sign assignment [4, Page 496]

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

$$Q_m = \frac{\langle \Lambda_m(z), \Lambda_m(z)\rangle}{\langle \Lambda_l(z), \Lambda_l(z)\rangle}$$

$$q_m = \frac{1+|k_m|}{1-|k_m|}$$

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

$$\frac{Q_m}{Q_{m+1}} = \begin{cases} q_m & if\ \epsilon_m = sgn(k_m) \\ 1/q_m & if\ \epsilon_m = -sgn(k_m) \end{cases} \tag{5.12}$$

Now assign the $\epsilon_m$:

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

Since the polynomials $\{\Lambda_N(z), \Lambda_{N-1}(z), \ldots, \Lambda_0(z)\}$ 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 5.4. The Octave function *tf2schurOneMlattice* calculates the coefficients of the one-multiplier Schur lattice from the transfer function.

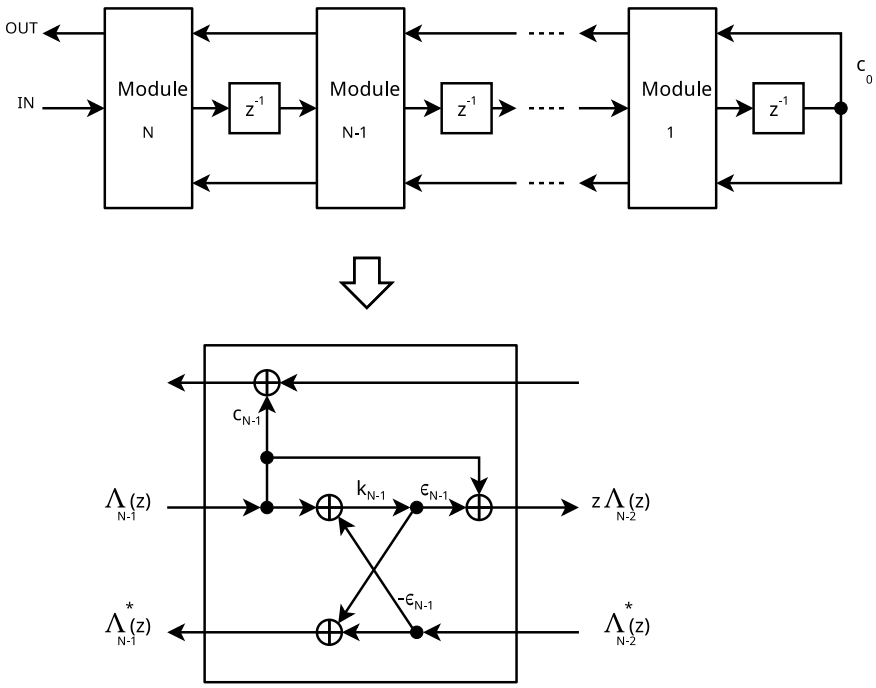



Figure 5.4: One-multiplier lattice structure (after *Parhi* [98, Fig. 12.11])

## 5.4 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

$$\Phi_N^*(z) = \sqrt{1-k_N^2}\,\Phi_{N-1}^*(z) + k_N \Phi_N(z) \tag{5.13}$$

$$\Phi_{N-1}(z) = z^{-1}\left\{\sqrt{1-k_N^2}\,\Phi_N(z) - k_N \Phi_{N-1}^*(z)\right\} \tag{5.14}$$

$$\Phi_N(z) = \frac{1}{\sqrt{1-k_N^2}}\left\{z\Phi_{N-1}(z) + k_N \Phi_{N-1}^*(z)\right\} \tag{5.15}$$

Figure 5.5 shows an implementation of Equations 5.13 and 5.15.

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

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

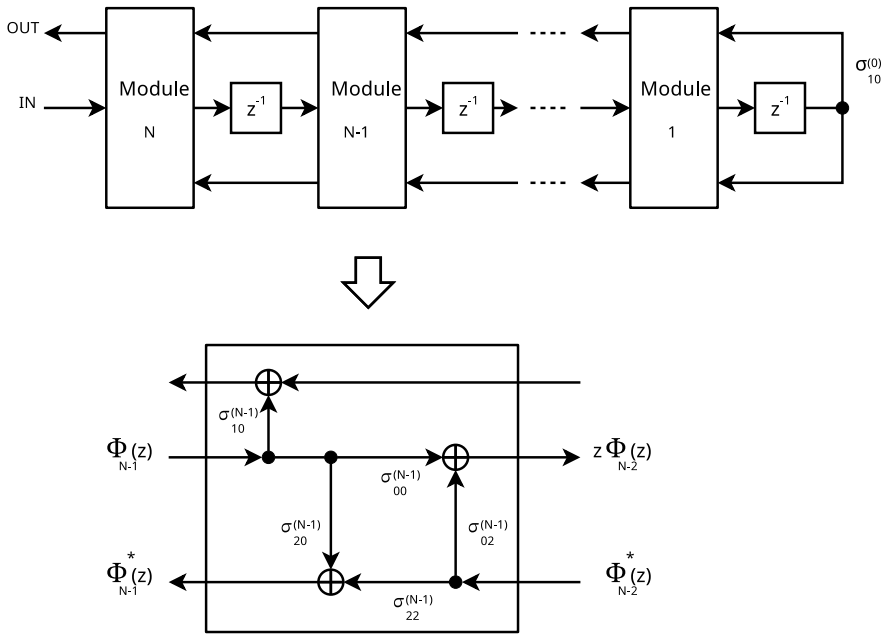



Figure 5.5: Normalised lattice filter (after *Parhi* [98, Fig. 12.20])

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

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.

## 5.5 Derivation of the Scaled Normalised Lattice Filter

We can introduce a delay at each node in the normalised lattice by making the transformation $z \to z^2$ and retiming so that each node corresponds to a state in the state variable description. Figure 5.6 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 scaled[a]. 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:

$$\sigma_{10}^{(N)} = c_N \tag{5.16}$$

$$\sigma_{11}^{(N)} = \sqrt{\sum_{j=0}^{N} c_j^2} \tag{5.17}$$

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

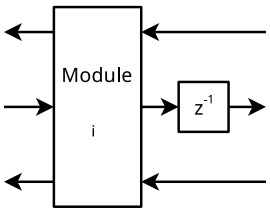
$$\sigma_{10}^{(i)} = \frac{c_i}{\sqrt{\sum_{j=0}^{i} c_j^2}} \tag{5.18}$$

$$\sigma_{11}^{(i)} = \frac{\sqrt{\sum_{j=0}^{i-1} c_j^2}}{\sqrt{\sum_{j=0}^{i} c_j^2}} \tag{5.19}$$

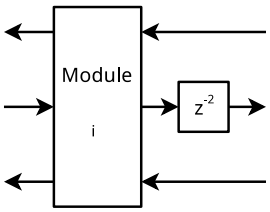
3. Any module $N$ to 1:

$$\sigma_{20}^{(i)} = -\sigma_{02}^{(i)} = k_i \tag{5.20}$$

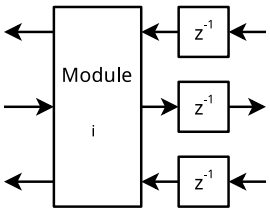
$$\sigma_{00}^{(i)} = \sigma_{22}^{(i)} = \sqrt{1 - k_i^2} \tag{5.21}$$




(a) Original lattice section




(b) After slow-down




(c) After slow-down and re-timing

Figure 5.6: Slowed and retimed lattice section

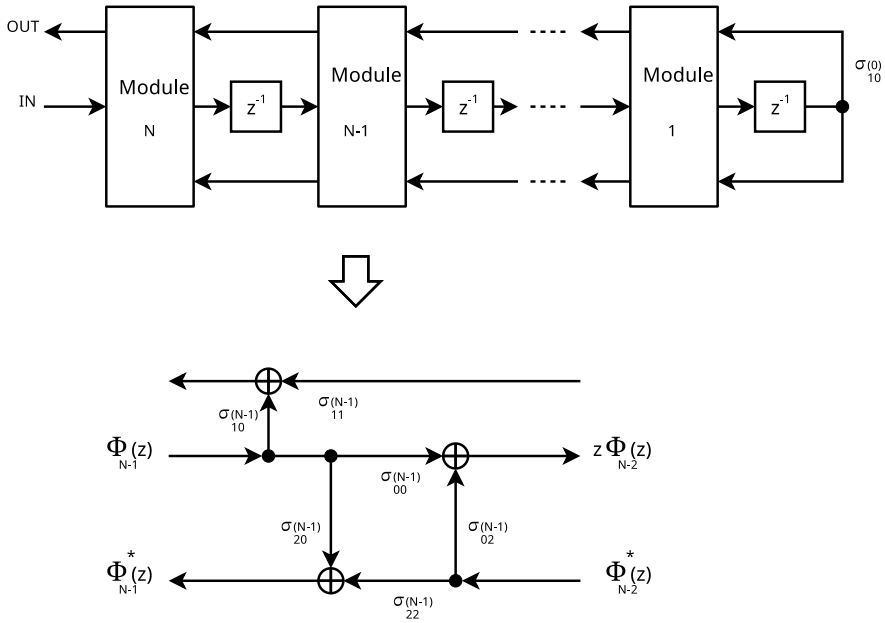



Figure 5.7: Normalised-scaled lattice filter (after *Parhi* [98, Fig. 12.19])

The structure of an N-th order normalised-scaled lattice filter is shown in Figure 5.7. Note that $\sigma_{10}^{(0)} = \operatorname{sign} c_0$. The Octave function *schurNSscale* implements the scaling of the $\sigma_{10}$ and $\sigma_{11}$ 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.

### 5.5.1 Example: synthesis of a 3rd order Butterworth lattice filter

*Parhi* [98, Example 12.6.1] uses as an example a third order Butterworth low-pass filter with cutoff at angular frequency $0.1\pi$ (where the sampling frequency is normalised to $2\pi$). The squared magnitude of the $n$-th order Butterworth response is defined to be

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

The poles of the response are evenly distributed around the unit circle. For stability, we choose the poles in the left-hand $s$-plane, $\lambda_k = \omega_c e^{\imath\theta_k}$, $\theta_k = \frac{\pi}{2}\left(1 + \frac{(2k-1)}{n}\right)$ with $1 \le k \le n$ 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: $\lambda_1 = -\Omega_c\left[\left(1 - \imath\sqrt{3}\right)/2\right]$ so:

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

Choose the bi-linear transformation from the $s$-plane to the $z$-plane so that the transfer function of the digital filter is $H(z) = \hat{H}(\frac{z-1}{z+1})$. If the cutoff frequency of the digital filter $H(z)$ is $\theta_c = \omega_c t_0$ (where $\omega_c$ 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 $z = e^{\imath\omega t_0}$ into the bi-linear transformation). For the third order Butterworth filter:

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

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:

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

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

[a] The roundoff-noise performance of the transformed filter is the same as that of the original filter

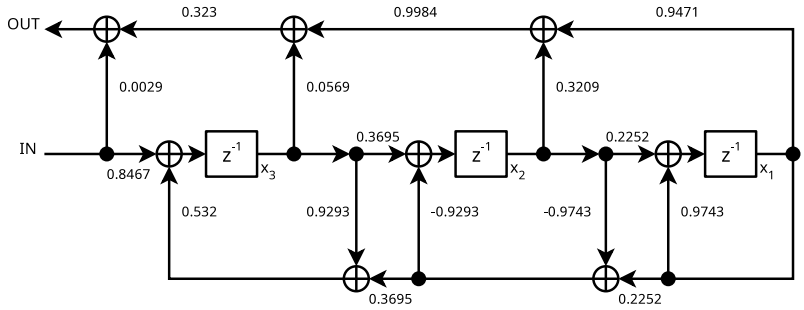



Figure 5.8: Butterworth 3rd order normalised-scaled lattice filter example (after *Parhi* [98, Fig. 12.20])

## 5.6 State Variable Descriptions for Schur Lattice Filters

### 5.6.1 State variable description of the Schur FIR lattice filter

Figure 5.2 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 5.3.

**Algorithm 5.3** Construction of a state variable description of the Schur FIR lattice filter

$y_0 = u$
$y^*_0 = u$
**for** $n = 1, \ldots, N$ **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 1.12.2, 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.

$$\begin{bmatrix} y_0 \\ y^*_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ y_1 \\ y^*_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & k_1 & 0 & \cdots & 0 \\ k_1 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & & & & \cdots & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y^*_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

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

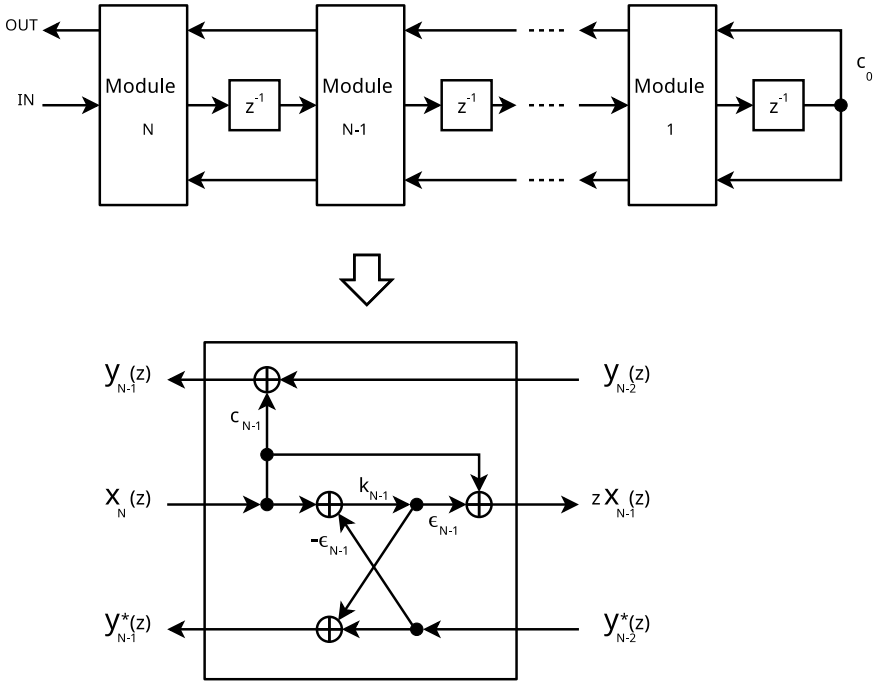



Figure 5.9: State variable description of the Schur one-multiplier lattice filter

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

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

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

### 5.6.2 State variable description of the one-multiplier IIR lattice filter

In Figure 5.9, Figure 5.4 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 Algorithm 5.4.

**Algorithm 5.4** Construction of a state variable description of the Schur one-multiplier lattice filter

$y^*_0 = x_1$
**for** $n = 1, \ldots, N-1$ **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**
$x_N = -k_N y^*_{N-1} + (1 + k_N \epsilon_N)\, u$
$y^* = (1 - k_N \epsilon_N)\, y^*_{N-1} + k_N u$
$y = c_0 x_1 + c_1 x_2 + \cdots + c_{N-1} x_N + c_N u$

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.

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

$$\begin{bmatrix} x_1' \\ y_1^* \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} -k_1 & (1+k_1\epsilon_1) & 0 & \cdots & \cdots & 0 \\ (1-k_1\epsilon_1) & k_1 & 0 & & & \vdots \\ 0 & 0 & 1 & & & \\ \vdots & & & \ddots & & \vdots \\ 0 & & & & 1 & 0 \\ 0 & & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_0^* \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x_1' \\ x_2' \\ y_2^* \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & -k_2 & (1+k_2\epsilon_2) & 0 & & \vdots \\ 0 & (1-k_2\epsilon_2) & k_2 & 0 & & \\ \vdots & & & \ddots & & \vdots \\ 0 & & & & 1 & 0 \\ 0 & \cdots & & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1' \\ y_1^* \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_{N-1}' \\ y_{N-1}^* \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & & & \cdots & 0 \\ 0 & 1 & & & & & \vdots \\ \vdots & & \ddots & & & & \vdots \\ 0 & & & -k_{N-1} & (1+k_{N-1}\epsilon_{N-1}) & & 0 \\ 0 & & & (1-k_{N-1}\epsilon_{N-1}) & k_{N-1} & & 0 \\ 0 & \cdots & \cdots & 0 & 0 & & 1 \end{bmatrix} \begin{bmatrix} x_1' \\ \vdots \\ x_{N-2}' \\ y_{N-2}^* \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ \vdots \\ x_N' \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 1 & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ 0 & & 1 & 0 & & 0 \\ 0 & & 0 & -k_N & & (1+k_N\epsilon_N) \\ 0 & \cdots & 0 & (1-k_N\epsilon_N) & & k_N \end{bmatrix} \begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_{N-1}' \\ y_{N-1}^* \\ u \end{bmatrix}$$

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).

### 5.6.3 State variable description of a retimed one-multiplier Schur lattice filter

Figure 5.10 shows a second order segment of the retimed Schur one-multiplier lattice filter in Figure 5.24b.

Construction of a state variable description of the retimed Schur one-multiplier lattice filter is summarised in Algorithm 5.5.

As in Section 5.6.2, the state variable description can be expressed as a series of matrix multiplications. For an even order filter:

$$\begin{bmatrix} y_0 \\ y_0^* \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{3\frac{N}{2}} \\ u \end{bmatrix} = \begin{bmatrix} c_0 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & & & \vdots \\ & & \ddots & & \vdots \\ 0 & & & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{3\frac{N}{2}} \\ u \end{bmatrix}$$

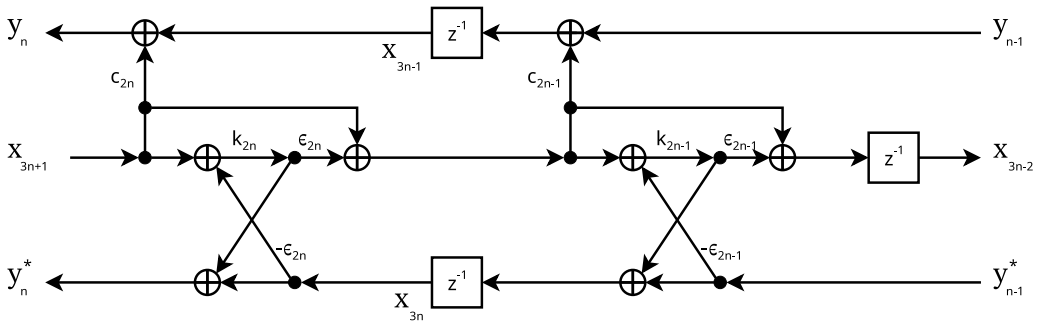



Figure 5.10: Second order segment of the retimed Schur one-multiplier lattice filter

**Algorithm 5.5** Construction of a state variable description of the retimed Schur one-multiplier lattice filter

Given $\{k_1, k_2, \ldots, k_N\}$, $\{\epsilon_1, \epsilon_2 \ldots, \epsilon_N\}$ and $\{c_0, c_1, \ldots, c_N\}$:

$y_0 = c_0 x_1$
$y_0^* = x_1$
**for** $n = 1, \ldots, \lceil \frac{N}{2} \rceil - 1$ **do**
$x'_{3n-2} = -k_{2n-1} y^*_{n-1} - (1 + k_{2n-1}\epsilon_{2n-1})\, k_{2n} x_{3n} + (1 + k_{2n-1}\epsilon_{2n-1})\,(1 + k_{2n}\epsilon_{2n})\, x_{3n+1}$
$x'_{3n-1} = y_{n-1} - c_{2n-1} k_{2n} x_{3n} + c_{2n-1}\,(1 + k_{2n}\epsilon_{2n})\, x_{3n+1}$
$x'_{3n} = (1 - k_{2n-1}\epsilon_{2n-1})\, y^*_{n-1} - k_{2n-1} k_{2n} x_{3n} + k_{2n-1}\,(1 + k_{2n}\epsilon_{2n})\, x_{3n+1}$
$y_n = x_{3n-1} + c_{2n} x_{3n+1}$
$y^*_n = (1 - k_{2n}\epsilon_{2n})\, x_{3n} + k_{2n} x_{3n+1}$
**end for**
**if** N is odd **then**
$x'_{3\lceil \frac{N}{2} \rceil - 2} = -k_N y^*_{\lceil \frac{N}{2} \rceil - 1} + (1 + k_N \epsilon_N)\, u$
$y = y_{\lceil \frac{N}{2} \rceil - 1} + c_N u$
$y^* = (1 - k_N \epsilon_N)\, y^*_{\lceil \frac{N}{2} \rceil - 1} + k_N u$
**else**
$x'_{3\frac{N}{2} - 2} = -k_{N-1} y^*_{\frac{N}{2} - 1} - (1 + k_{N-1}\epsilon_{N-1})\, k_N x_{3\frac{N}{2}} + (1 + k_{N-1}\epsilon_{N-1})\,(1 + k_N \epsilon_N)\, u$
$x'_{3\frac{N}{2} - 1} = y_{\frac{N}{2} - 1} - c_{N-1} k_N x_{3\frac{N}{2}} + c_{N-1}\,(1 + k_N \epsilon_N)\, u$
$x'_{3\frac{N}{2}} = (1 - k_{N-1}\epsilon_{N-1})\, y^*_{\frac{N}{2} - 1} - k_{N-1} k_N x_{3\frac{N}{2}} + k_{N-1}\,(1 + k_N \epsilon_N)\, u$
$y = x_{3\frac{N}{2} - 1} + c_N u$
$y^* = (1 - k_N \epsilon_N)\, x_{3\frac{N}{2}} + k_N u$
**end if**

$$
\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ y_1 \\ y_1^* \\ x_4 \\ \vdots \\ x_{3\frac{N}{2}} \\ u \end{bmatrix}
=
\begin{bmatrix}
0 & -k_1 & 0 & 0 & -(1+k_1\epsilon_1)\,k_2 & (1+k_1\epsilon_1)\,(1+k_2\epsilon_2) & \cdots & 0 \\
1 & 0 & 0 & 0 & -c_1k_2 & c_1\,(1+k_2\epsilon_2) & \cdots & 0 \\
0 & (1-k_1\epsilon_1) & 0 & 0 & -k_1k_2 & k_1\,(1+k_2\epsilon_2) & \cdots & 0 \\
0 & 0 & 0 & 1 & 0 & c_2 & \cdots & 0 \\
0 & 0 & 0 & 0 & (1-k_2\epsilon_2) & k_2 & \cdots & 0 \\
 & & & & & & \ddots & \vdots \\
0 & \cdots & & & & & \cdots\ 1 & 0 \\
0 & \cdots & & & & & \cdots\ 0 & 1
\end{bmatrix}
\begin{bmatrix} y_0 \\ y_0^* \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{3\frac{N}{2}} \\ u \end{bmatrix}
$$

$$
\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3\frac{N}{2}} \\ y \\ y^* \end{bmatrix}
=
\begin{bmatrix}
1 & \cdots & & & & & \cdots & 0 \\
\vdots & \ddots & & & & & & \vdots \\
0 & \cdots & 0 & -k_{N-1} & 0 & 0 & -(1+k_{N-1}\epsilon_{N-1})\,k_N & (1+k_{N-1}\epsilon_{N-1})\,(1+k_N\epsilon_N) \\
0 & \cdots & 1 & 0 & 0 & 0 & -c_{N-1}k_N & c_{N-1}\,(1+k_N\epsilon_N) \\
0 & \cdots & 0 & 1-k_{N-1}\epsilon_{N-1} & 0 & 0 & -k_{N-1}k_N & k_{N-1}\,(1+k_N\epsilon_N) \\
0 & \cdots & 0 & 0 & 0 & 1 & 0 & c_N \\
0 & \cdots & 0 & 0 & 0 & 0 & (1-k_N\epsilon_N) & k_N
\end{bmatrix}
\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3\frac{N}{2}-3} \\ y_{\frac{N}{2}-1} \\ y^*_{\frac{N}{2}-1} \\ x_{3\frac{N}{2}-2} \\ x_{3\frac{N}{2}-1} \\ x_{3\frac{N}{2}} \\ u \end{bmatrix}
$$

For an odd order filter, the final matrix multiplication is:

$$
\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3\lceil\frac{N}{2}\rceil-2} \\ y \\ y^* \end{bmatrix}
=
\begin{bmatrix}
1 & \cdots & & & \cdots & 0 \\
\vdots & \ddots & & & & \vdots \\
0 & \cdots & 0 & -k_N & 0 & (1+k_N\epsilon_N) \\
0 & \cdots & 1 & 0 & 0 & c_N \\
0 & \cdots & 0 & (1-k_N\epsilon_N) & 0 & k_N
\end{bmatrix}
\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3\lceil\frac{N}{2}\rceil-3} \\ y_{\lceil\frac{N}{2}\rceil-1} \\ y^*_{\lceil\frac{N}{2}\rceil-1} \\ x_{3\lceil\frac{N}{2}\rceil-2} \\ u \end{bmatrix}
$$

The Octave function *schurOneMlatticeRetimed2Abcd* returns the state variable description of a retimed Schur one-multiplier lattice filter (including the all-pass filter $Cap$ and $Dap$ output matrixes).

### 5.6.4 State variable description of the scaled-normalised IIR lattice filter

In Figure 5.11, Figure 5.7 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 5.6 [b].

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.

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

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

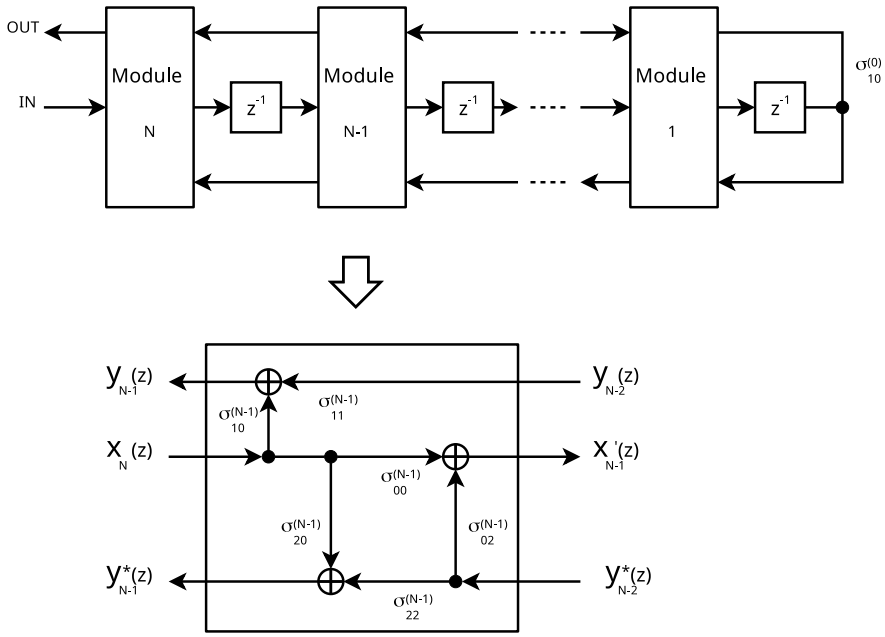



Figure 5.11: State variable description of the Schur Scaled-Normalised lattice filter

**Algorithm 5.6** Construction of a state variable description of the Scaled-Normalised Lattice

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

$$
\begin{bmatrix} x'_1 \\ y^*_1 \\ y_1 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}
=
\begin{bmatrix}
\sigma^{(1)}_{02} & 0 & \sigma^{(1)}_{00} & 0 & 0 & \cdots & 0 \\
\sigma^{(1)}_{22} & 0 & \sigma^{(1)}_{20} & 0 & & & \vdots \\
0 & \sigma^{(1)}_{11} & \sigma^{(1)}_{10} & 0 & & & \\
0 & 0 & 0 & 1 & & & \\
\vdots & & & & \ddots & & \vdots \\
0 & & & & & 1 & 0 \\
0 & \cdots & & & \cdots & 0 & 1
\end{bmatrix}
\begin{bmatrix} y^*_0 \\ y_0 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}
$$

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

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

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

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).

## 5.7 Roundoff Noise Calculation in Schur Lattice Filters

In this section I rely on the analysis of round-off noise presented by *Roberts* and *Mullis* [168], [28], [29].

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.

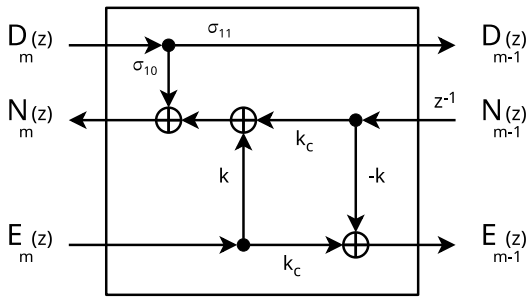



Figure 5.12: Transposed normalised-scaled lattice filter module (after *Parhi* [98, Fig. 12.24])

### 5.7.1 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 5.12 shows module $m$ of $N, \ldots, 1$ 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

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

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, $D_N(z) = 0$, $N_N(z) = z^N D\left(z^{-1}\right)$ 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 5.5.1. 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 5.8).

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 5.13 shows the amplitude response of the Schur normalised-scaled lattice implementation of the Butterworth filter and all-pass filter determined from the cross-correlation of the filter input and outputs.

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 =  0.1825
varyd =  0.1819
```

and for the Schur lattice all-pass filter

```
est_varyapd =  0.5000
varyapd =  0.4913
```

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

```
est_varyoptd =  0.1225
varyoptd =  0.1215
```

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

```
est_varydird =  5.8317
varydird =  1.8148
```

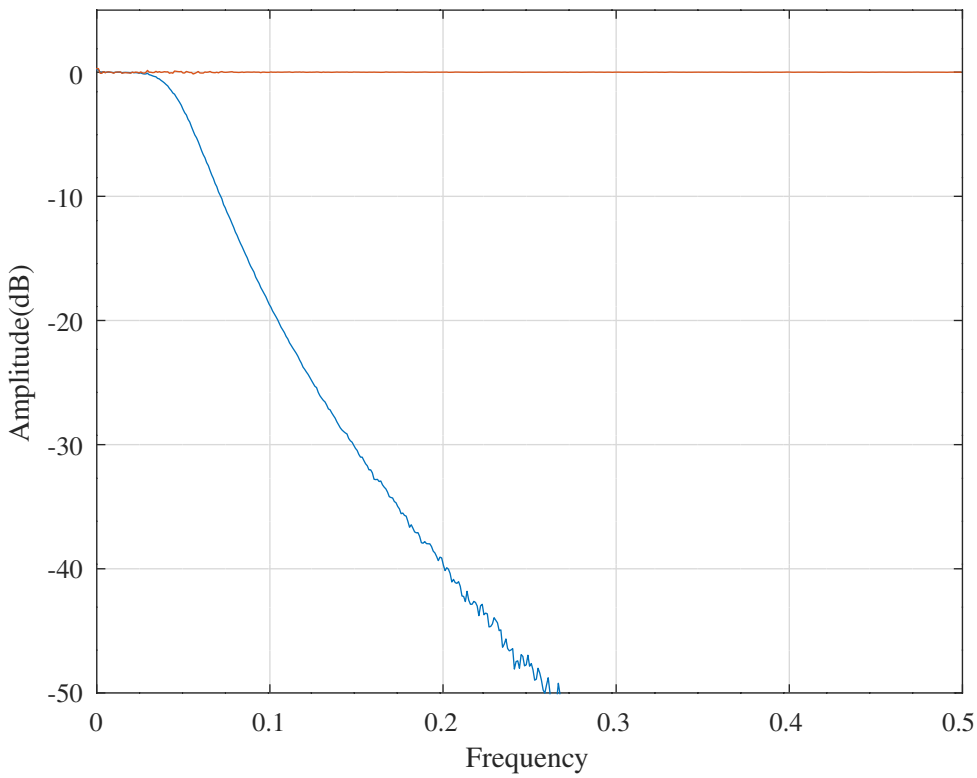



Figure 5.13: Amplitude response of the 3rd order Butterworth filter implemented with a Schur normalised-scaled lattice structure

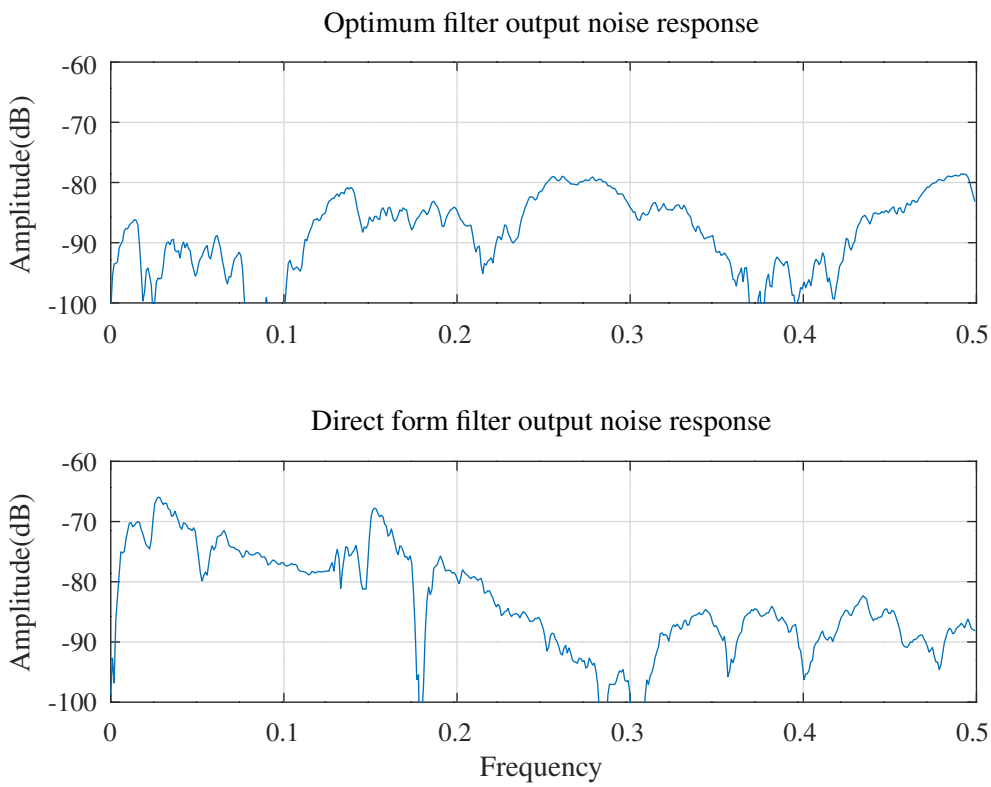



Figure 5.14: Comparison of the amplitude response of the output noise of the 3rd order Butterworth filter when implemented with a scaled direct-form and globally optimised state variable structure

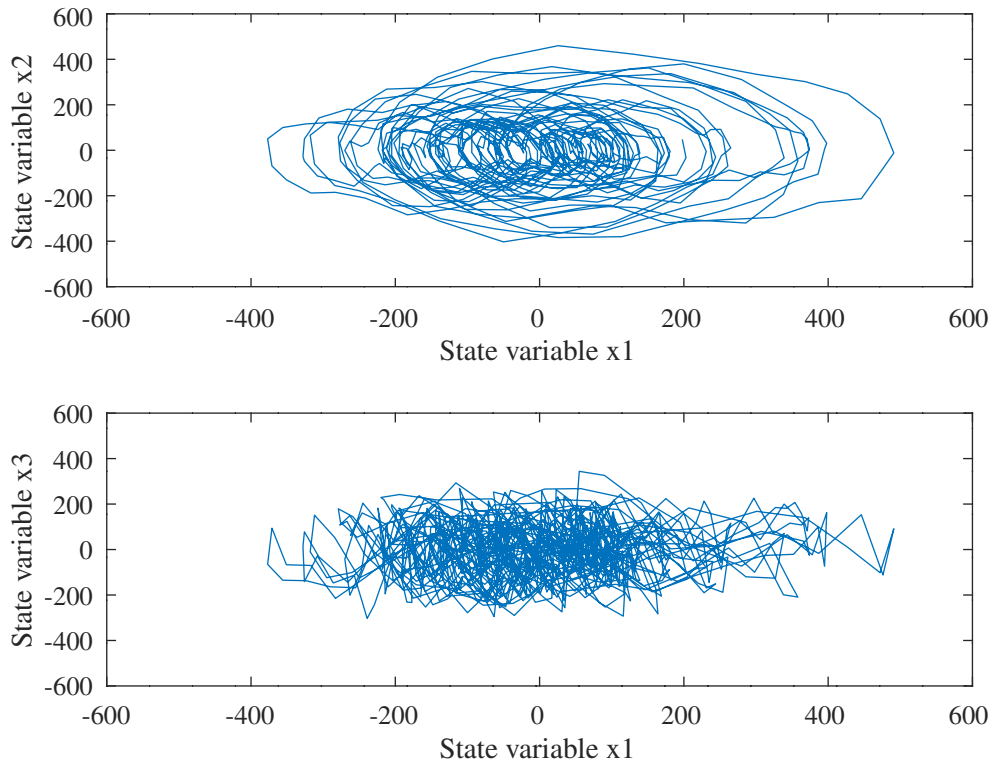



Figure 5.15: Internal states in the 3rd order Butterworth filter implemented in a normalised-scaled Schur lattice structure with a random noise input

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 5.14 compares the response of the output roundoff noise of the scaled direct-form implementation to that of the globally optimised filter.

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

```
stdxx =  131.21   129.39   127.97
```

The standard deviations of the internal states of the globally-optimum and scaled direct-form filters are similar. Figures 5.15, 5.16 and 5.17 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 5.18 shows part of the state trajectories for the Schur lattice implementation of the Butterworth filter in response to a sine wave input.

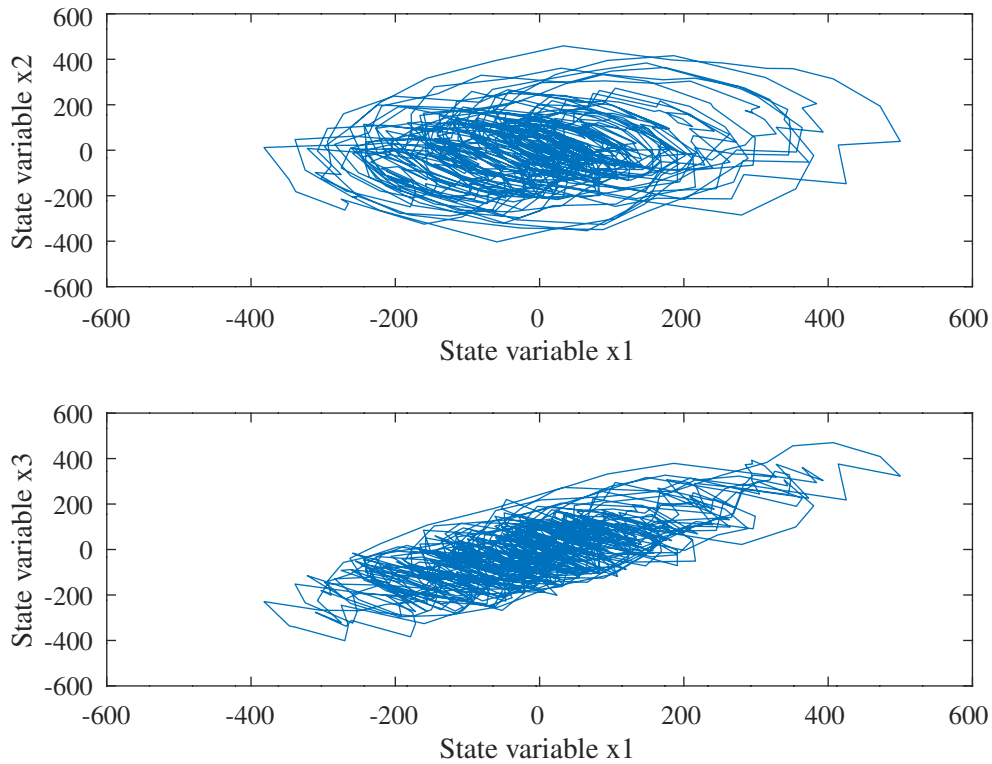



Figure 5.16: Internal states in the 3rd order Butterworth filter implemented in the globally optimised state variable structure with a random noise input

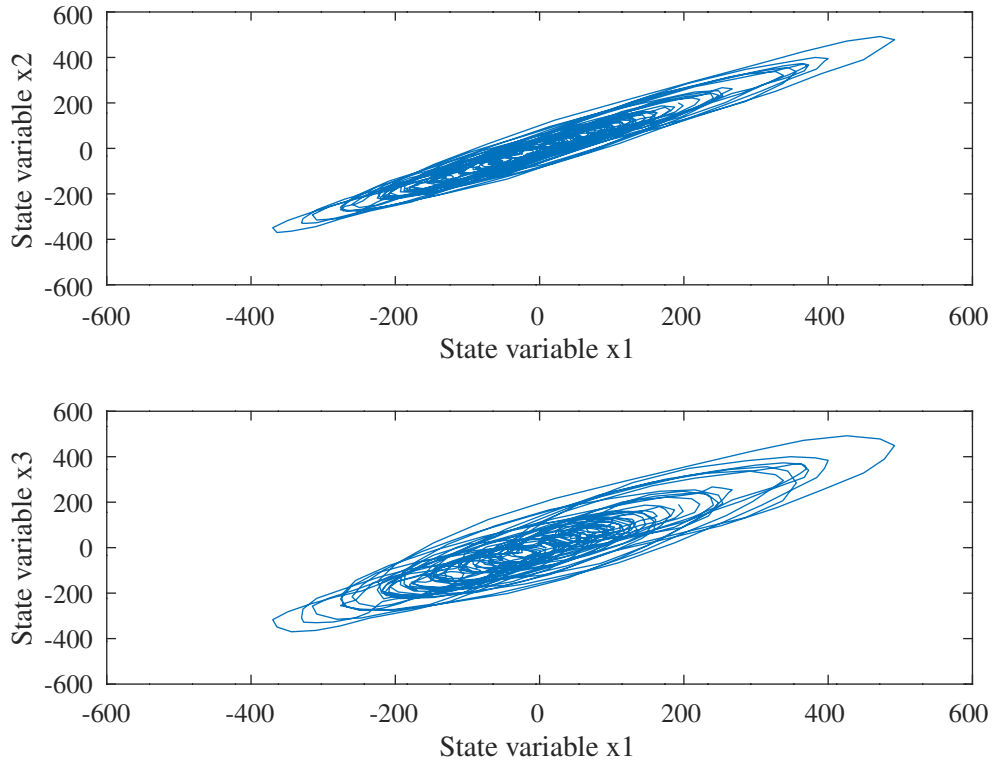



Figure 5.17: Internal states in the 3rd order Butterworth filter implemented in a direct-form structure with a random noise input

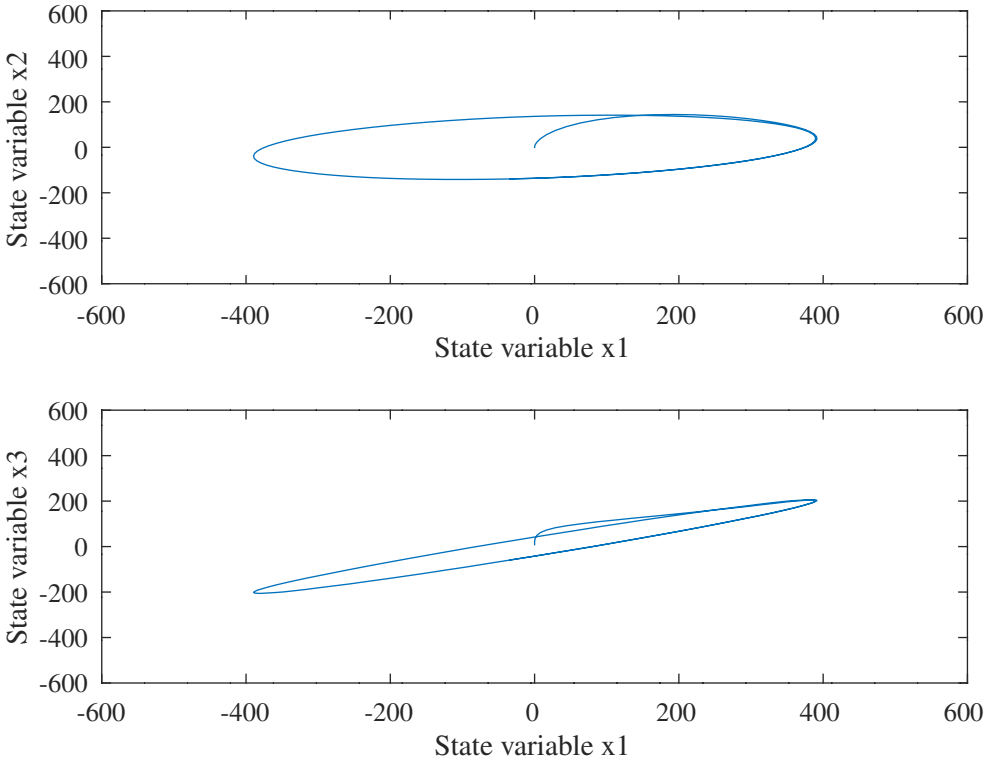



Figure 5.18: Internal states in the 3rd order Butterworth filter implemented with a Schur lattice structure with a sine wave input

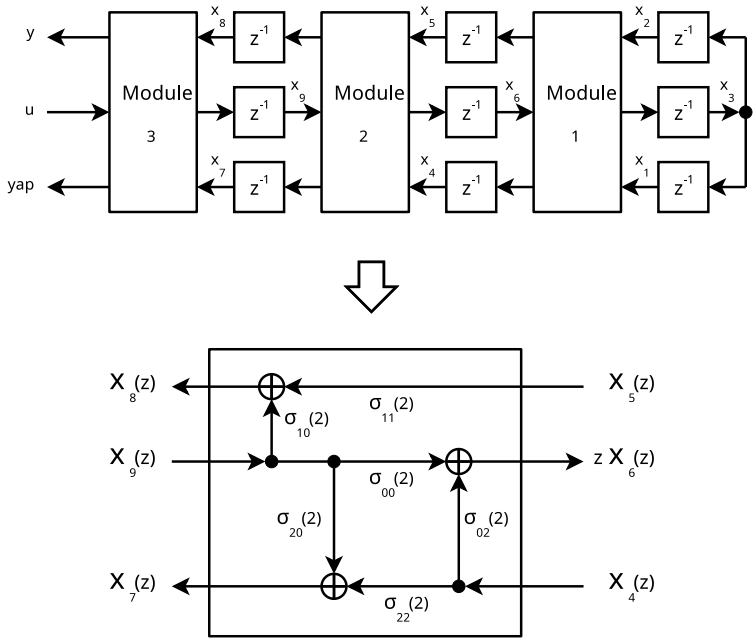



Figure 5.19: Slowed and retimed normalised-scaled lattice for the 3rd order Butterworth example

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

*Parhi* [98, 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 5.19. 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 Octave function *schurNSlatticeRetimedNoiseGain* converts the Schur normalised-scaled lattice implementation to a retimed state variable form used to calculate the noise gains. 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-digits[c]. The *oct*-file *bin2SD.cc* implements a C++ version of *bin2SD*. The filter amplitude responses for the corresponding Butterworth and all-pass filters with 10-bit 2-signed-digit coefficients are shown in Figure 5.20. The response demonstrates a drawback of the normalised-scaled lattice filters: they are not *structurally loss-less* in the sense of Vaidyanathan et al. [152, 149] (see Appendix M.2). The normalised-scaled lattice does not preserve the all-pass characteristic when its coefficients are truncated.

[c] *Parhi* [98, 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.

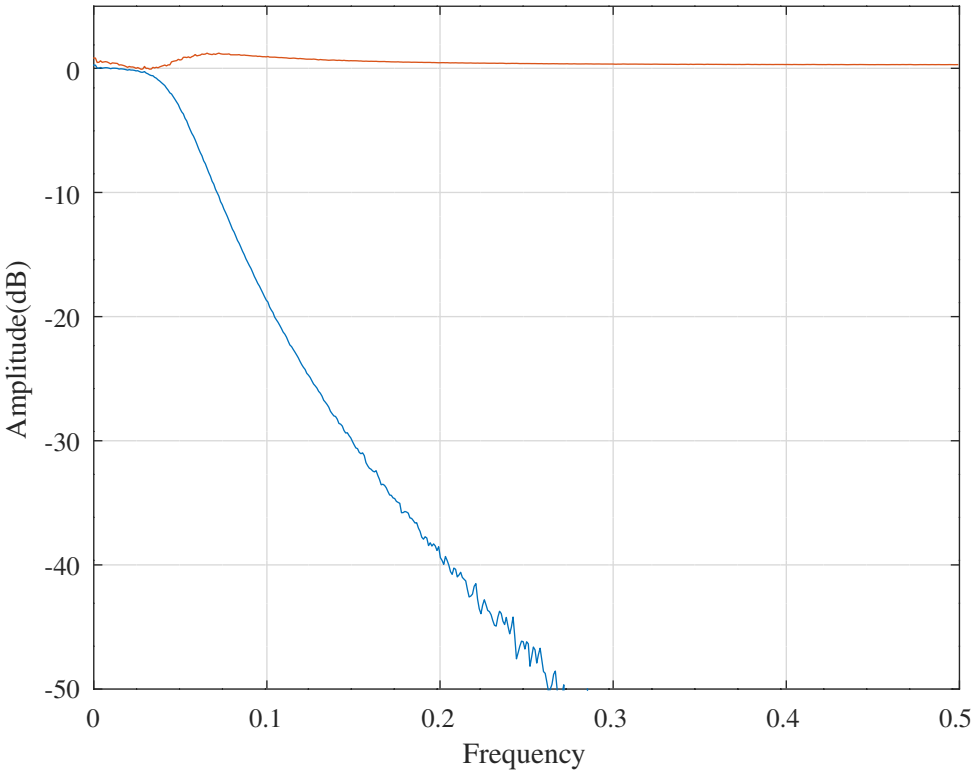



Figure 5.20: Amplitude response of the 3rd order Butterworth filter implemented with a normalised-scaled lattice structure and 10-bit 2 signed-digit coefficients

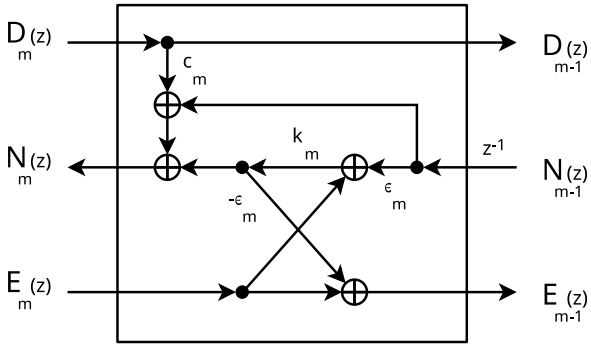



Figure 5.21: Transposed one multiplier lattice filter module

### 5.7.2 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 5.21 shows module $m$ of $N, \ldots, 1$ of the transposed graph of a one multiplier lattice filter. The transposed graph of the 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 5.11 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:

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

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 $\ell_2$ 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 =  0.1654
```

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

```
est_varyapd = 0.50000
varyapd = 0.4913
```

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

```
stdxf =   131.21   129.39   127.96
```

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 5.22. 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 *schurOneMlatticeRetimedNoiseGain*. The Octave function *schurOneMlatticeFilter* calculates the upper, Butterworth output, edge of Figure 5.4 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 *schurOneMlatticeRetimedNoiseGain* illustrates that the retiming method of calculating noise gain is much more flexible than the transposed filter method. In addition, as noted in Section 5.7.1, 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 *schurOneMlatticeRetimedNoiseGain* function can calculate round-off noise gain for three different one multiplier lattice filter implementations by selecting the states included through the *filterStr* argument:

- For "**schur**" *schurOneMlatticeFilter* calculates the Butterworth output (the upper row of Figure 5.22) 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

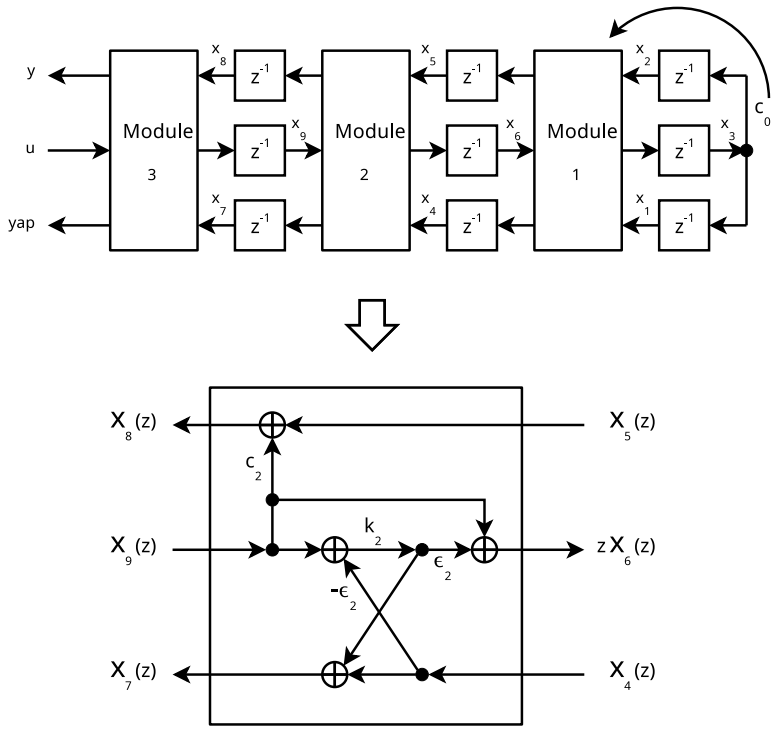



Figure 5.22: Slowed and retimed single-multipler lattice for the 3rd order Butterworth example

In both these cases, states $x_1$ and $x_2$ in Figure 5.22 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 *schurOneMlatticeRetimedNoiseGain* 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 5.7.1) 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 =  0.1752
varyd =  0.1740
```

and for the allpass filter:

```
est_varyapd =  0.5000
varyapd =  0.4931
```

The standard deviations of the internal states of the filter are

```
stdxf = 137.07   129.01   127.79
```

Figure 5.23 shows the amplitude responses 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*.

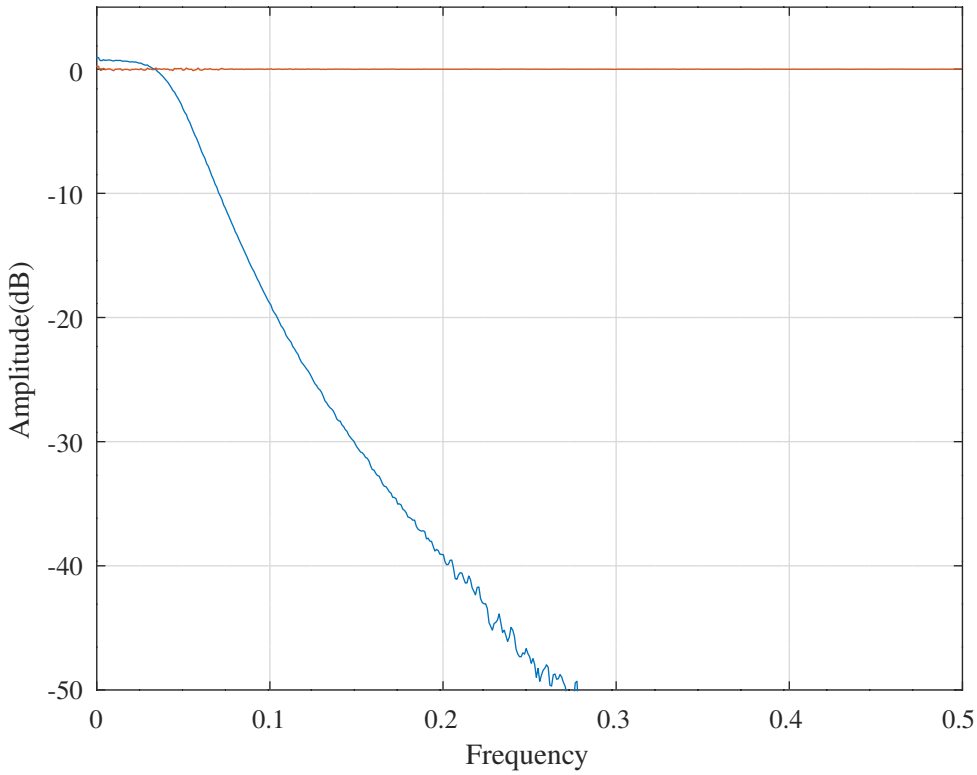



Figure 5.23: Amplitude responses of the slowed and retimed single-multiplier lattice 3rd order Butterworth filter implemented in Schur lattice form with 10-bit 3-signed-digit truncated coefficients

### 5.7.3 Frequency transformations and Schur normalised-scaled lattice filter round-off noise

Section 2.3 shows the frequency transformation of a state variable filter. Section 3.8 shows some results, given by *Mullis* and *Roberts* [28, Section III], concerning the noise gain of a frequency transformed state variable filter. *Koshita et al.* [191] point out that the normalised-scaled Schur lattice all-pass filter has controllability and observability Grammians $K = W = I$. It follows from Equation 3.4 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 3.5). 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 2.4 shown in Figure 2.6. Table 5.1 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).

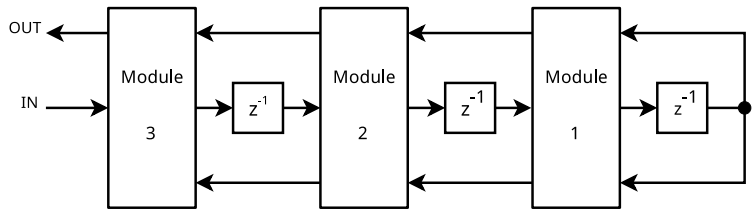
## 5.8 Examples of retiming Schur lattice filters

### 5.8.1 Retiming a 3-rd order lattice filter

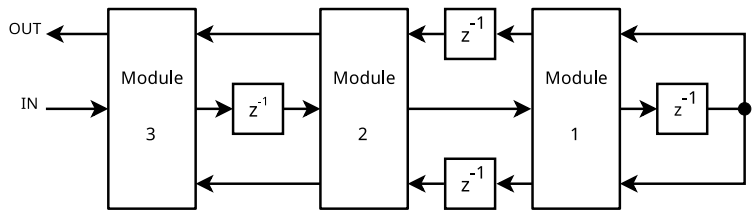
Figure 5.24a shows a representation 3rd order lattice filter and Figure 5.24b shows the filter after retiming by moving the second filter delay to the upper and lower branches of the signal flow graph. The total delay in each loop of the signal flow graph is

| | 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 5.1: 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.




(a) Original filter




(b) Filter after retiming

Figure 5.24: Example of retiming a 3rd order lattice filter

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[d]. Appendix 5.6.3 derives the state variable description of the retimed one-multipler Schur lattice filter.

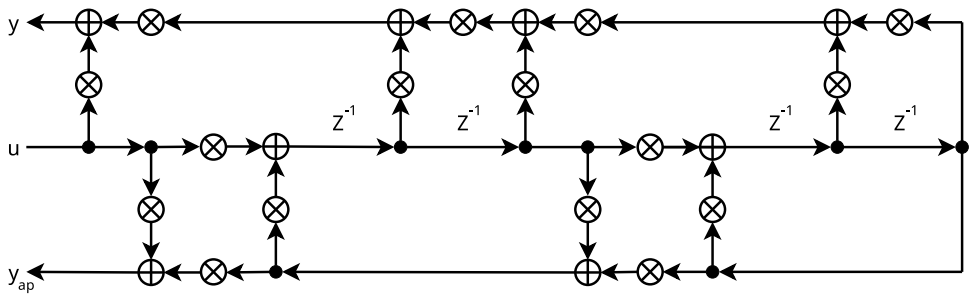
### 5.8.2 Retiming a 4-th order Schur normalised-scaled lattice filter

The normalised-scaled lattice filter of order N shown in Figure 5.7 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 1.12.1 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* [98, Chapter 4] discusses formal methods for retiming a signal flow graph. Figure 5.25a 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 5.25b 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.

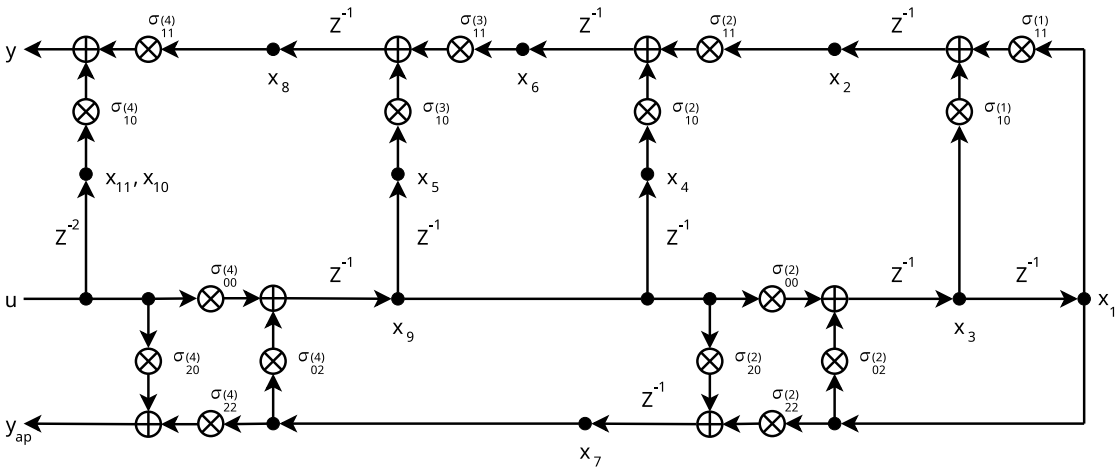
Following the notation of Figure 5.11, the state variable equations for the filter shown in Figure 5.25b are:

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

[d]The tapped output could be retimed by pipelining the arithmetic operations in a tree structure. The all-pass lattice ouput calculation is recursive and cannot be retimed in that way.




(a) In original form




(b) After retiming

Figure 5.25: Signal flow graph of a 4th order normalised-scaled Schur lattice filter with a denominator polynomial having coefficients only in $z^{-2}$

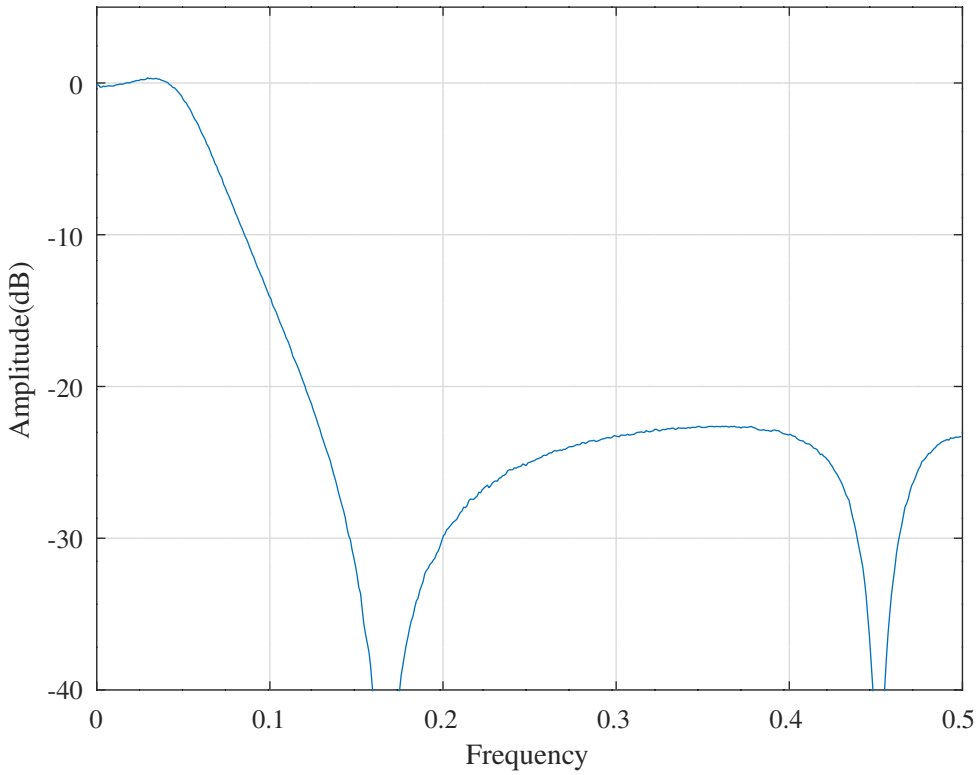


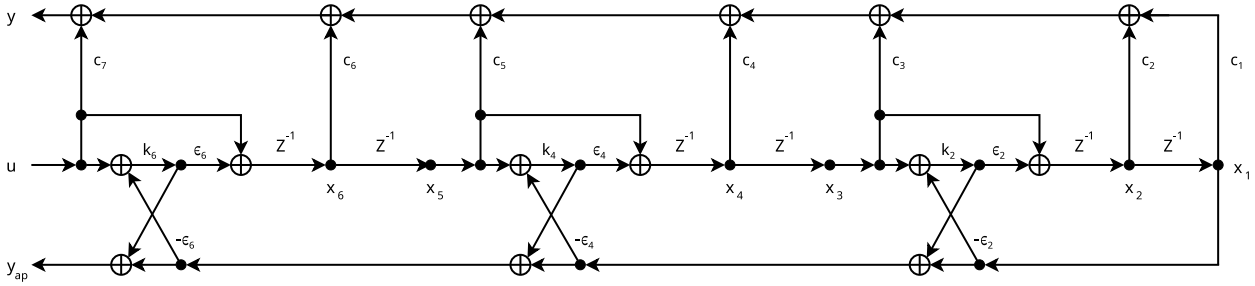

Figure 5.26: Simulated amplitude response of the retimed 4th order normalised-scaled Schur lattice filter

$$
\begin{aligned}
x_8(k+1) &= \sigma_{10}^{(3)} x_5(k) + \sigma_{11}^{(3)} x_6(k) \\
x_9(k+1) &= \sigma_{02}^{(4)} x_7(k) + \sigma_{00}^{(4)} u(k) \\
x_{10}(k+1) &= x_{11}(k) \\
x_{11}(k+1) &= u(k) \\
y(k) &= \sigma_{11}^{(4)} x_8(k) + \sigma_{10}^{(4)} x_{10}(k) \\
y_{ap}(k) &= \sigma_{22}^{(4)} x_7(k) + \sigma_{20}^{(4)} u(k)
\end{aligned}
$$

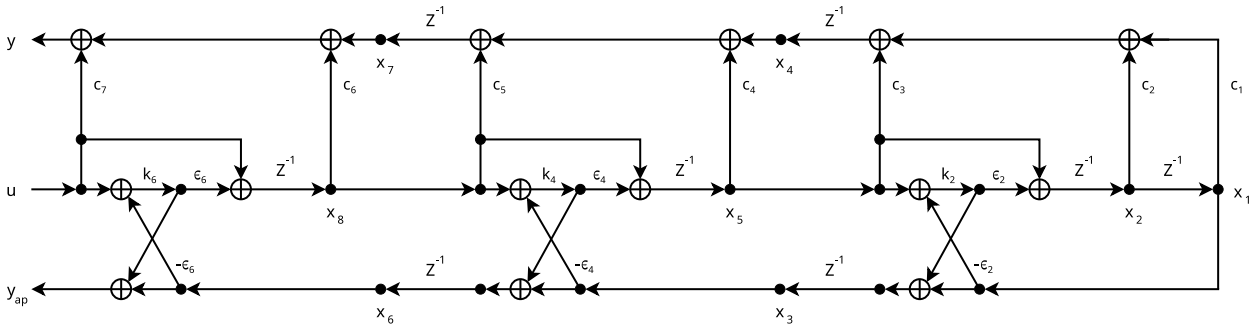
The Octave script *schur_retimed_test.m* designs a 4-th order low-pass filter with cutoff frequency $0.05 f_S$ and denominator coefficients in $z^{-2}$. The script minimises the amplitude response error with the Octave *fminunc* function. The barrier function of *Tarczynski et al.* [10] is added to the 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 5.26 shows the simulated amplitude response of the retimed Schur lattice filter.

### 5.8.3 Retiming a 6th-order Schur one-multiplier lattice filter

Figure 5.27 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.




(a) Signal flow graph of a 6th order one-multiplier Schur lattice filter with a denominator polynomial having coefficients in $z^{-2}$




(b) Signal flow graph of a 6th order one-multiplier Schur lattice filter with a denominator polynomial having coefficients in $z^{-2}$ after retiming

Figure 5.27: Original and retimed signal flow graphs of a 6th order one-multiplier Schur lattice filter with a denominator polynomial having coefficients in $z^{-2}$. The number of delays around each loop of the signal flow graph is unchanged.

### 5.8.4 Frequency transformations of retimed Schur lattice filters

An implementation of a rational transfer function having demominator 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.* [10] 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 5.28 shows the response of the low-pass prototype filter.

Figure 5.29a 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 5.29b 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 $z^{-2}$.

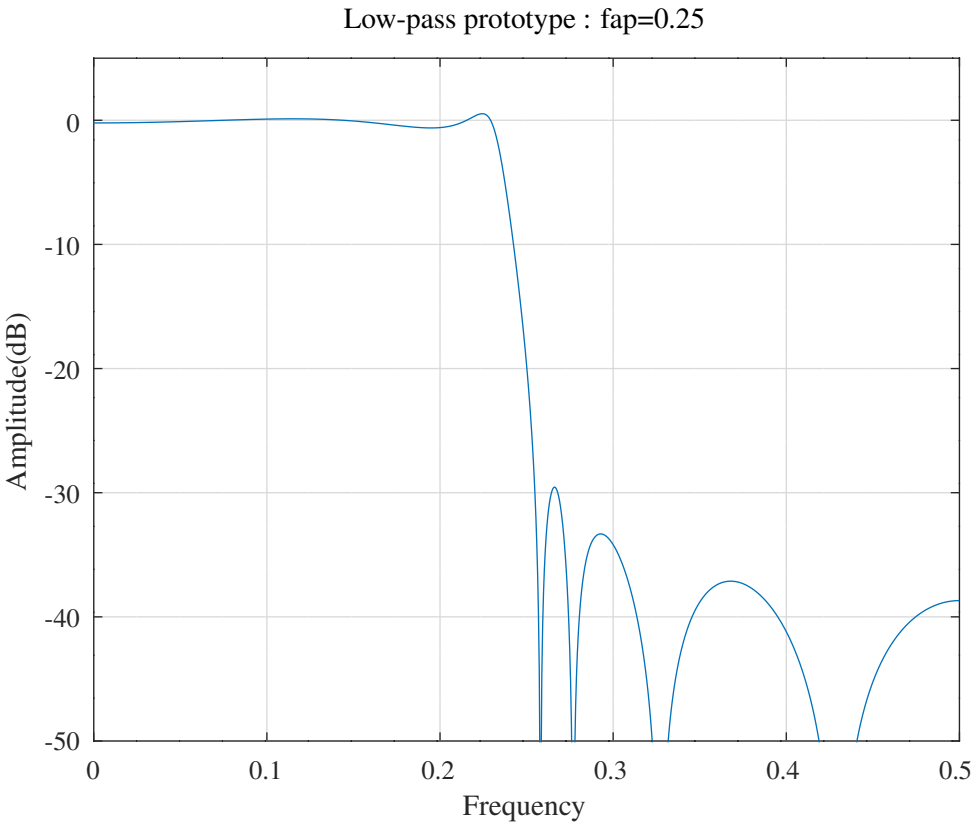
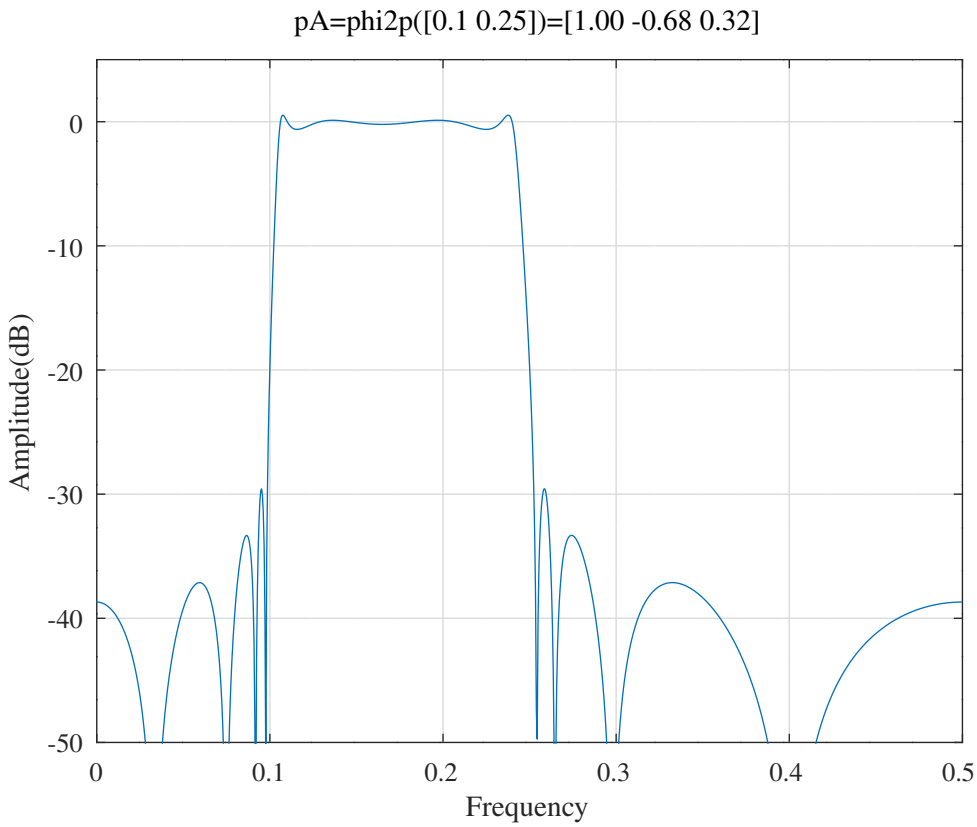
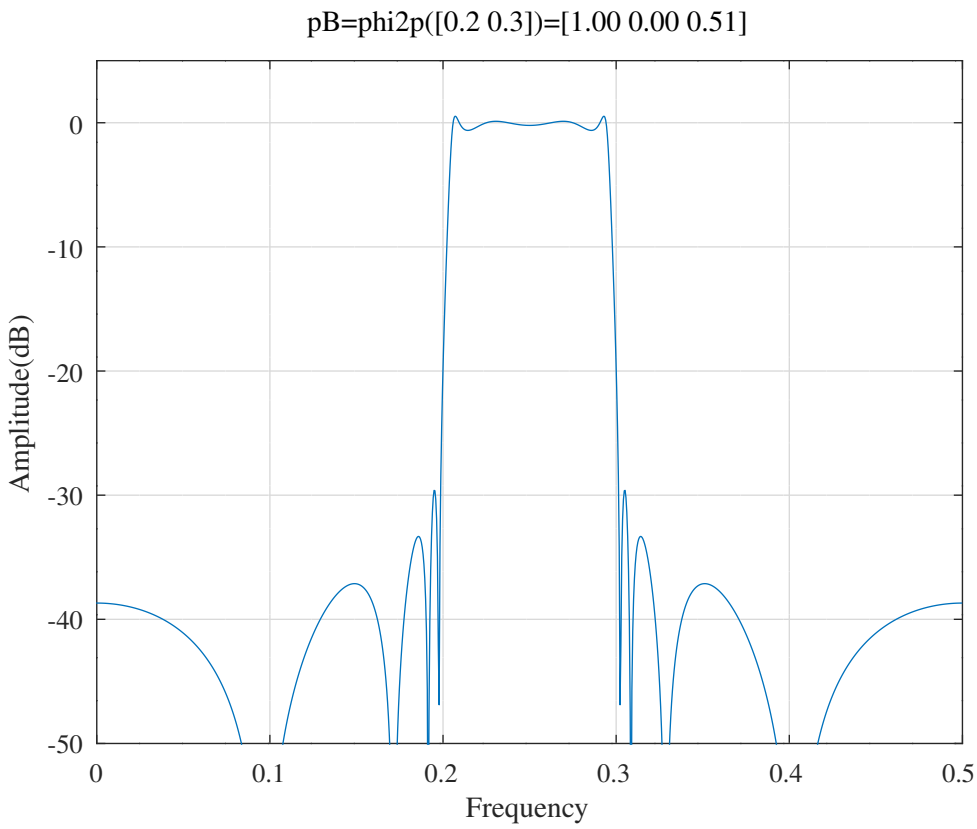



Figure 5.28: Amplitude response of a low-pass filter prototype having denominator polynomial coefficients in $z^{-2}$




(a) Amplitude response of a band-pass filter with frequency transformation [0.1 0.25]




(b) Amplitude response of a band-pass filter with frequency transformation [0.2 0.3]

Figure 5.29: Frequency transformations of a low-pass filter prototype having denominator polynomial coefficients in powers of $z^{-2}$ only

## 5.9 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 $i = N, N-1, \cdots 1$
3. Expand the numerator $N_N(z)$ in the Schur polynomial basis by the polynomial expansion algorithm
4. Synthesise the filter by Equations 5.16, 5.17, 5.18, 5.19, 5.20 and 5.21

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 $i = N, N-1, \cdots 1$
3. Find the $k_i$ with greatest magnitude and recursively compute the sign parameters by Algorithm 5.2. 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 5.8 and 5.9

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 6

# Orthogonal state variable filters

Chapter 5 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* [168, Section 10.4] for decomposing an arbitrary transfer function into $2 \times 2$ block diagonal coordinate rotations that can be "*manipulated to acheive either high-speed, parallel computation using a sparsely connected array of processor modules or low-speed single processor realizations*"[a].

## 6.1 Definition of orthogonal state variable filters

If, for matrix $O$, $OO^\top = I$, 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 complex conjugate transpose matrix operator, and $\mathcal{U}$ is the group of *unitary* matrixes. An all pass filter has magnitude response $\left|H\left(e^{\imath\theta}\right)\right| = 1$ for all real $\theta$. The $m \times m$ matrix transfer function $H(z)$ of $m$ inputs and $m$ outputs and complex elements is *all-pass* if $H\left(e^{\imath\theta}\right)$ is unitary:

$$H\left(e^{\imath\theta}\right) \in \mathcal{U}(m) \ \forall \text{ real } \ \theta$$

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. $AA^\top + BB^\top = I_{(n\times n)}$
2. $H\left(e^{\imath\theta}\right) = D + C\left(e^{\imath\theta}I - A\right)^{-1}B \ \in \mathcal{U}(m) \ \forall$ 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

$$F \in \mathcal{O}(m+n) \Rightarrow F \in \mathcal{F}(m,n)$$
$$F \in \mathcal{F}_0(m,n) \Rightarrow F \in \mathcal{O}(m+n)$$

[a] In Appendix M.2, I review the method presented by *Vaidyanathan and Mitra* [152, 149] for decomposing an odd-order transfer function into the sum of two all-pass filters

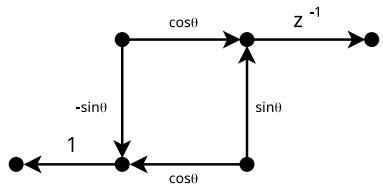



Figure 6.1: Lattice section

For any orthogonal matrix $F \in \mathcal{O}(N)$, there are $N$ possible filters, one each in $\mathcal{F}(m, N-n)$, $1 \leq m \leq N$. 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 $T \in \mathcal{O}(N)$,

$$\begin{aligned} F' &= \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} \\ &= \left[\begin{array}{c|c} T^\top AT & T^\top B \\ \hline CT & D \end{array}\right] \\ &= \begin{bmatrix} T & 0 \\ 0 & I \end{bmatrix}^\top \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} T & 0 \\ 0 & I \end{bmatrix} \end{aligned}$$

are also members of $\mathcal{F}$ and that $F'$ represents an alternative realisation of $H(z) = D + C(zI - A)^{-1}B$.

## 6.2 The Lattice Orthogonal All-Pass Filter Section

Figure 6.1 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

$$H(z) = \begin{bmatrix} z^{-1}\cos\theta & z^{-1}\sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

## 6.3 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 $\mu_i^2$ 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 $\mu_i$ are all equal. Recall that the second order modes of the globally optimised filter are invariant under a frequency transformation.

## 6.4 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

$$K = AA^\top + BB^\top = I$$

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

$$\begin{aligned} G &= \begin{bmatrix} A & B \\ C_1 & D_1 \end{bmatrix} F^\top F \\ &= \begin{bmatrix} I & 0 \\ C_1 A^\top + D_1 B^\top & C_1 C^\top + D_1 D^\top \end{bmatrix} F \\ &= G_0 F \end{aligned}$$

where

$$\begin{aligned} \begin{bmatrix} \Gamma & \delta \end{bmatrix} &= \begin{bmatrix} C_1 & D_1 \end{bmatrix} F^\top \\ G_0 &= \begin{bmatrix} I & 0 \\ \Gamma & \delta \end{bmatrix} \end{aligned}$$

$G_0$ can be simplified by finding the transformation

$$T = \begin{bmatrix} T_0 & 0 \\ 0 & 1 \end{bmatrix} \in \mathcal{O}(n+1)$$

so that

$$\Gamma T_0 = \begin{bmatrix} 0 & \gamma \end{bmatrix}$$

where $\gamma$ 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 $\Gamma$

$$\begin{bmatrix} \gamma_i & \gamma_{i+1} \end{bmatrix} \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix} = \begin{bmatrix} 0 & \gamma'_{i+1} \end{bmatrix}$$

where $\theta_i$ is given by

$$\tan\theta_i = \frac{\gamma_i}{\gamma_{i+1}}$$

Now transform $G$ by $T$

$$\begin{aligned} G' &= T^{-1} G T \\ &= T^{-1} (G_0 F) T \\ &= \left(T^{-1} G_0 T\right) \left(T^{-1} F T\right) \\ &= G'_0 F' \end{aligned}$$

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 6.2 is a representation of the corresponding implementation of $G(z)$ (see [168, Figure 10.4.5]).

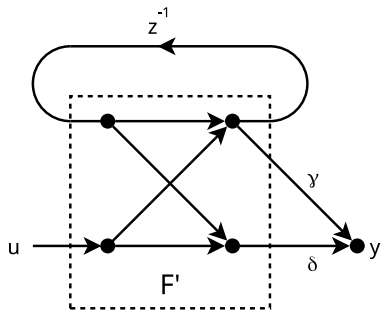



Figure 6.2: Signal flow graph of an arbitrary transfer function constructed from a two-input two-output all-pass filter

*Roberts* and *Mullis* [168, pp. 460-461] describe the following procedure for factoring a two-input, two-output all-pass filter, $F$, into the product of $2 \times 2$ 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 $G_0'$[b]:

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 $F \in \mathcal{O}(6)$:

$$F' = \begin{bmatrix} X & X & X & X & X & X \\ X & X & X & X & X & X \\ X & X & X & X & X & X \\ 0_6 & X & X & X & X & X \\ 0_4 & 0_5 & X & X & X & X \\ 0_1 & 0_2 & 0_3 & X & X & X \end{bmatrix} = T_6^\top T_5^\top T_4^\top T_3^\top T_2^\top T_1^\top F T_1 T_2 T_3 T_4 T_5 T_6$$

   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 $\Gamma$. In the example:

$$\begin{bmatrix} I_{4\times4} & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}^\top \begin{bmatrix} X & X & X & X & X & X \\ X & X & X & X & X & X \\ X & X & X & X & X & X \\ F_{4,1} & X & X & X & X & X \\ F_{5,1} & F_{5,2} & X & X & X & X \\ F_{6,1} & F_{6,2} & F_{6,3} & X & X & X \end{bmatrix} \begin{bmatrix} I_{4\times4} & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} X & X & X & X & X & X \\ X & X & X & X & X & X \\ X & X & X & X & X & X \\ F'_{4,1} & X & X & X & X & X \\ F'_{5,1} & F'_{5,2} & X & X & X & X \\ 0 & F'_{6,2} & F'_{6,3} & X & X & X \end{bmatrix}$$

2. Factor $F'$ with $q = 2n - 1$ coordinate rotations that zero the sub-diagonal elements

$$F'' = F' F_1 \cdots F_q$$

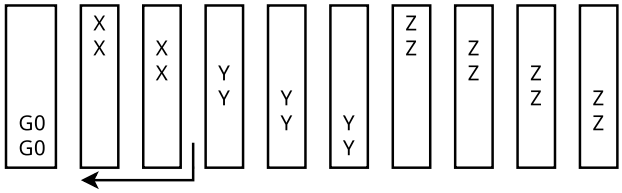
   These are *not* similarity transformations. In fact, $F''$ will be diagonal with elements $\pm 1$[c]. The diagonal elements can be included in the rotations so that:

$$F' = F_q^\top \cdots F_1^\top$$

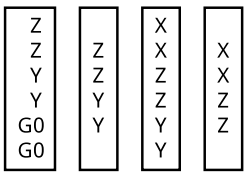
   For example:

$$F'' = \begin{bmatrix} X & X & X & X & X & X \\ 0_9 & X & X & X & X & X \\ 0_4 & 0_8 & X & X & X & X \\ 0 & 0_3 & 0_7 & X & X & X \\ 0 & 0 & 0_2 & 0_6 & X & X \\ 0 & 0 & 0 & 0_1 & 0_5 & X \end{bmatrix} = F' F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8 F_9$$

[b]This procedure is known as the *upper Hessenberg reduction* [49, Section 7.4.2].
[c]This is a consquence of $F''$ being upper-triangular and $F'' \in \mathcal{O}(n+1)$.




(a) Orthogonal filter as a depth-10 pipeline with a single rotation element




(b) Orthogonal filter as a depth-4 pipeline

Figure 6.3: Orthogonal structures for a 5-th order filter

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* [49, Theorem 2.5.2] provides the required similarity transform[d]. Alternatively, the Schur decomposition of a transfer function (reviewed in Chapter 5, based on *Parhi* [98, Chapter 12]) realises an orthogonal lattice filter without requiring the calculation of $K$ to find an initial similarity transformation[e]. 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 $2 \times 2$ 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 $0.05 f_S$. The orthogonal decomposition of the filter contains 17 non-trivial $2 \times 2$ 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.

### 6.4.1 An example of structural variations

The decomposition of the filter transfer function into $2 \times 2$ 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 6.3a. The $2 \times 2$ 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 6.3a 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 6.3a can be combined into a single similarity transform, $T$, and applied to $G_0 F$ to form a new filter factorisation:

$$F' = T^{-1} F T$$

This is equivalent to a circular shift of the order of the orthogonal factor matrixes. Also, the individual factors commute if the $2 \times 2$ sub-matrixes on the diagonal have no common rows. Hence the filter matrix can be compacted as shown in Figure 6.3b, which represents a sequence of $2, 3, 2, 2$ rotation elements in a depth-4 pipeline.

*Roberts* and *Mullis* [168, pp. 462-467] show other examples of orthogonal filter factorisation.

[d]The SVD of $K$ gives $U^\top K V = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$ where $U$ and $V$ are orthogonal $n \times n$ matrixes (ie: $U^\top U = I$)

[e]In fact the Schur decomposition of the transfer function is equivalent to the orthogonal decomposition of the state transition matrix presented here.

# Chapter 7

# Feedforward and feedback of state quantisation error in state variable filters

Chapter 3 reviewed the optimisation of a state variable digital filter for minimum roundoff noise. The resulting filter has $\mathcal{O}\left(N^2\right)$ non-zero coefficients. Section 4 showed how a filter could be implemented as a cascade of second-order sections with section optimal roundoff noise performance. Chapters 5 and 6 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 [59, 213, 40] in order to improve the roundoff noise performance of the filter with a small increase in complexity. *Mullis* and *Roberts* [30] 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* [40] . *Lu* and *Hinamoto* [213] 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* [40] .

## 7.1 Problem formulation

Figure 7.1 [213, 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 $\delta$ and the feed-forward error transfer function is $\eta$.

As shown in Section 1.5 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 [40] 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 $\tilde{x}(k)$ and the ouptput $\tilde{y}(k)$ all have a fractional $B+B_0$ bit representation and the input $u(k)$ is a $B$ bit fraction. The quantiser $Q\left[\tilde{x}(k)\right]$ *rounds* the $B+B_0$ fraction $\tilde{x}(k)$ 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:

$$e(k) = \tilde{x}(k) - Q\left[\tilde{x}(k)\right]$$

The roundoff residue sequence, $e(k)$, is modelled as a zero-mean noise process with covariance

$$\begin{aligned} \sigma_e &= E\left\{e(k)\, e^{\top}(k)\right\} \\ &= \frac{q^2}{12} I \end{aligned}$$

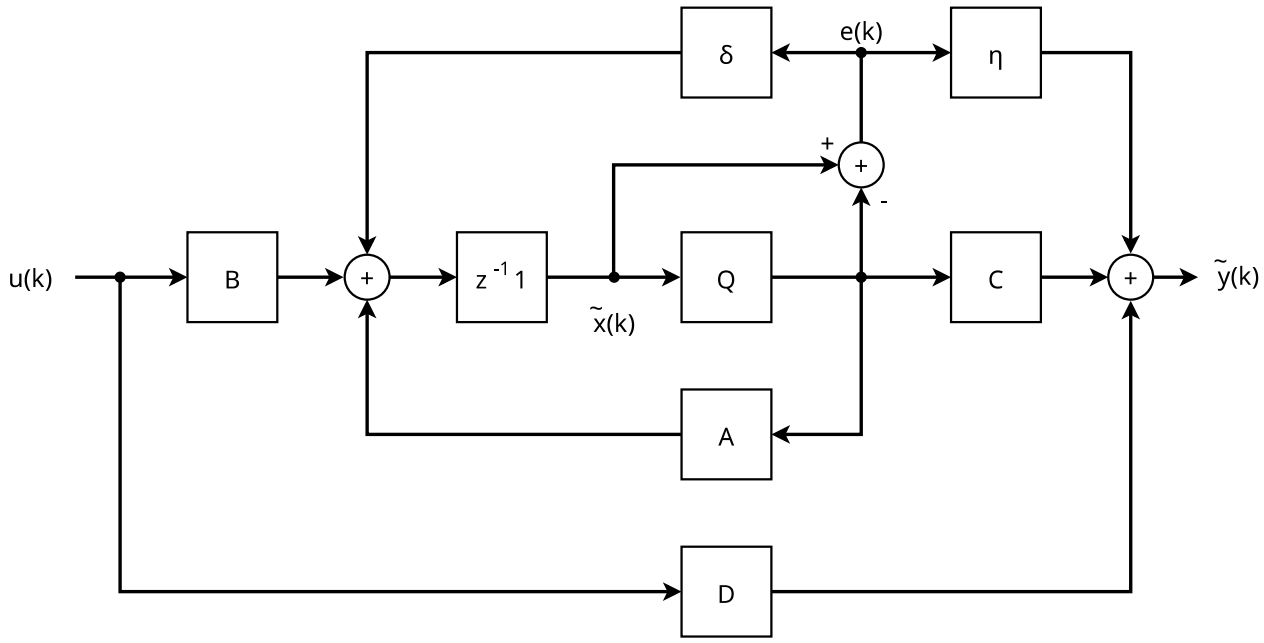



Figure 7.1: Error feedback and feed-forward in state variable digital filters, (*Lu* and *Hinamoto* [213, Figure 2])

where $q = 2^{-B}$. When the state variables are quantised, the state variable equations for the error feedback/feed-forward filter of Figure 7.1 are

$$\begin{aligned}\tilde{x}(k+1) &= AQ[\tilde{x}(k)] + Bu(k) + \delta e(k)\\ \tilde{y}(k) &= CQ[\tilde{x}(k)] + Du(k) + \eta e(k)\end{aligned}$$

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 1.9, 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 3.3.3 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

$$K = AKA^\top + BB^\top + q^2(A-\delta)(A-\delta)^\top$$

If $q^2 \ll 1$ then

$$K \approx AKA^\top + BB^\top$$

For optimum $l_2$-norm scaling $\operatorname{diag}\{K\} = I$. As shown in Section 1.11, 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 3.4, the output roundoff noise variance due to truncation of each state is $\sigma_e^2\sigma^2$, where

$$\begin{aligned}\sigma^2 &= \operatorname{trace}\left\{(A-\delta)(A-\delta)^\top W + (C-\eta)(C-\eta)^\top\right\}\\ W &= AWA^\top + CC^\top\end{aligned}$$

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, $\operatorname{diag}\{K\} = I$.

Given an initial realisation, $\{A_0, B_0, C_0, D\}$, of $H(z)$, Williamson [40, Section III] defines the *residue matrix* as

$$P_0 = (I-A)^\top W_0 + W_0(I-A)$$

where

$$W_0 = A_0 W_0 A_0^\top + C_0 C_0^\top$$

He shows that the eigenvalues, $\left\{\mu_i^2\right\}$ of $K_0 W_0$, and $\left\{\rho_i^2\right\}$ of $K_0 P_0$, are invariant under a similarity transformation, $T$, and are positive. In Section 6.3, the eigenvalues $\left\{\mu_i^2\right\}$ are referred to as the *second-order modes* of $H(z)$. Williamson calls $\left\{\rho_i^2\right\}$ the *residue modes*. Further, he calls a realisation $\{A_0, B_0, C_0, D\}$ *input balanced* if $K_0 = I$ and $W_0 = M^2 = \operatorname{diag}\left\{\mu_1^2, \ldots, \mu_N^2\right\}$, where $N$ is the order of $H(z)$[a]. An input balanced structure can be transformed to an *internally balanced* structure with $K_1 = W_1 = M$ by means of the similarity transformation $T_1 = M^{-\frac{1}{2}}$.

## 7.2 Minimisation of round-off noise with $\delta = I$ and $\eta = 0$

Williamson [40, Section V] considers the "problem of finding the optimal transformation, $T$, and the optimal integer residue feedback matrixes $\delta_T$ and $\eta_T$, which together minimise the output round-off noise". He proves the theorem shown as Algorithm 7.1 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 7.1** Minimisation of round off noise in error feedback/feedforward state variable filters.
(See *Williamson* [40, 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 $\left\{A_T = T^{-1} A_0 T, B_T = T^{-1} B_0, C_T = T^\top C_0, D\right\}$. 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

$$\begin{aligned} T &= R_1 \Pi R_0^\top \\ \Pi &= \operatorname{diag}\{\pi_1, \ldots, \pi_N\} \end{aligned}$$

where

$$\pi_m^2 = \frac{1}{\rho_m} \frac{\sum_{k=1}^{N} \rho_k}{N}$$

and for unitary matrixes $R_0$ and $R_1$

$$\begin{aligned} \operatorname{diag}\left\{R_0 \Pi^{-1} R_0^\top\right\} &= I \\ R_1^\top P_0 R_1 \quad &\text{is} \quad \text{diagonal} \end{aligned}$$

where

$$P_0 = (I - A_0)^\top M^2 + M^2 (I - A_0)$$

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];
```

[a] If $K = I$, then the filter is *orthogonal*. See Part 6. An SVD transformation of the corresponding $W$ matrix gives the input balanced structure.

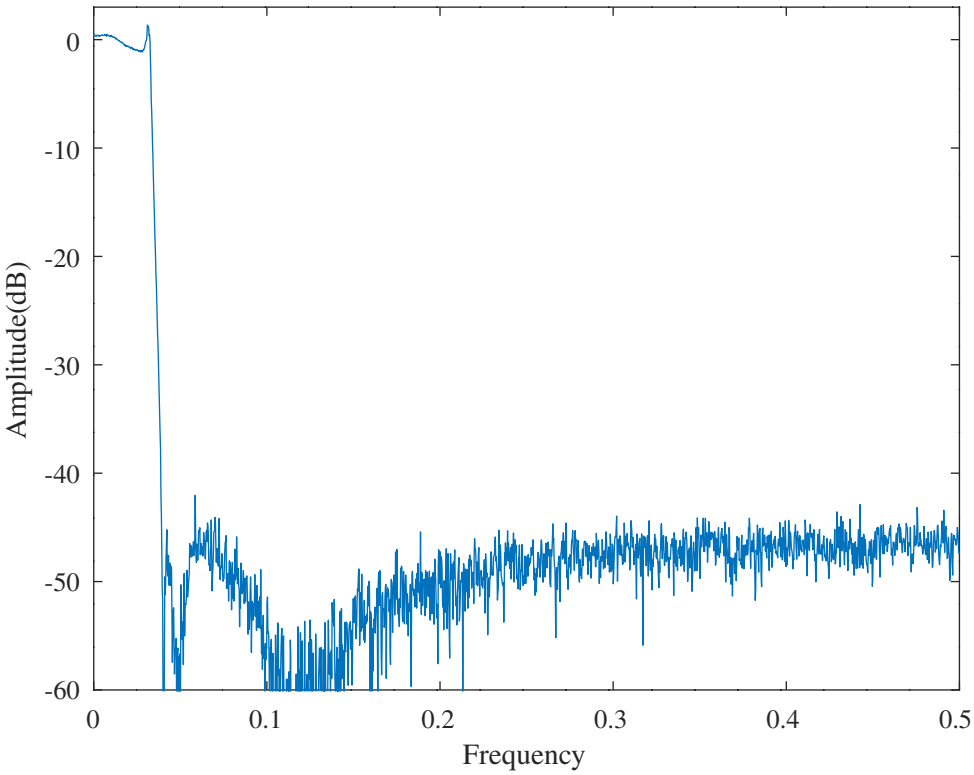



Figure 7.2: Simulated response of a 6-th order filter with 8 bit coefficients and a 2 bit mantissa (for error feedback) in the state variables. (After *Williamson* [40, Example 6.3]).

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:

```
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 7.2.

# Part II

# Constrained optimisation of the IIR filter frequency response

# Chapter 8

# 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

$$H(z) = \frac{b(z)}{a(z)} = \frac{\sum_{l=0}^{m} b_l z^{-l}}{1 + \sum_{l=1}^{n} a_l z^{-l}} \tag{8.1}$$

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 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 function[a].

## 8.1 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

$$x = \left[K,\ R_{0,1}, \ldots, R_{0,U},\ R_{p,1}, \ldots, R_{p,V},\ r_{0,1}, \ldots, r_{0,\frac{M}{2}},\ \phi_{0,1}, \ldots, \phi_{0,\frac{M}{2}},\ r_{p,1}, \ldots, r_{p,\frac{Q}{2}},\ \phi_{p,1}, \ldots, \phi_{p,\frac{Q}{2}}\right] \tag{8.2}$$

This vector represents the poles and zeros of a filter with gain factor $K$, the radiuses of $U$ real zeros and $V$ real poles and the radiuses 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 G shows the amplitude, phase and group delay responses and their gradients in terms of the coefficient vector.

[a] An alternative to Equation 1 is to linearise the cost function, $\mathcal{E}_H$, by:

$$\hat{\mathcal{E}}_H = \sum_{j=1}^{NB} \int_{f_{l,j}}^{f_{u,j}} \frac{W_j(f)}{\left|\hat{D}(f)\right|^2} |N(f) - D(f) H_d(f)|^2 df$$

where $\hat{D}$ is the previous estimate of $D$. See, for example, *Dumitrescu and Niemistö* [20]

The weighted squared-magnitude of the response error, $\mathcal{E}_A$, is:

$$\mathcal{E}_A(x) = \sum_{j=1}^{NB} \int_{f_{l,j}}^{f_{u,j}} W_{Aj}(f) \left[A(x,f) - A_D(x,f)\right]^2 df$$

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 $\lambda_i$ are the Lagrange multipliers (or *dual variables*) for those constraints. The *Karush-Kuhn-Tucker* conditions for the minimum are:

$$\nabla_x \left\{ \mathcal{E}(x) - \sum_{i \in \mathcal{A}(x)} \lambda_i g_i(x) \right\} = 0 \tag{8.3}$$

$$g_i(x) \geq 0 \tag{8.4}$$

$$\lambda_i \geq 0 \tag{8.5}$$

$$\langle \lambda_i, g_i(x) \rangle = 0 \tag{8.6}$$

The gradient of the magnitude error is:

$$\nabla_x \mathcal{E}_A(x) = \sum_{j=1}^{NB} 2 \int_{f_{l,j}}^{f_{u,j}} W_{Aj}(f) \left[A(x,f) - A_D(x,f)\right] \nabla_x A(x,f) \, df$$

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:

$$\Lambda(\lambda_i) = \inf_{x \in \mathcal{D}} \left\{ \mathcal{E}(x) - \sum_{i \in \mathcal{A}(x)} \lambda_i g_i(x) \right\}$$

### 8.1.1 Solution of the constrained quasi-Newton optimisation problem

Express the *Karush-Kuhn-Tucker* conditions (Equations 8.3 and 8.4) at $x^k$ in terms of a linearised version of the gradient of the error and linearised constraints (see Appendix K.1):

$$\nabla_x \mathcal{E}(x^k) + \nabla_x^2 \mathcal{E}(x^k)(x - x^k) - \sum_{i \in \mathcal{A}(x)} \lambda_i \nabla_x g_i(x^k) = 0 \tag{8.7}$$

$$g_i(x^k) + \nabla_x g_i(x^k)(x - x^k) \geq 0$$

where the Hessian of the squared-magnitude response error, $\mathcal{E}_A$, is:

$$\nabla_x^2 \mathcal{E}_A(x) = \sum_{j=1}^{NB} 2 \int_{f_{l,j}}^{f_{u,j}} W_{Aj}(f) \left\{ \left[\nabla_x A(x,f)\right]^2 + \left[A(x,f) - A_D(x,f)\right] \nabla_x^2 A(x,f) \right\} df$$

and the Hessian of the group delay error, $\mathcal{E}_T$, is similar.

Appendix K.6 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

$$\mathcal{W}_k d^k - \mathcal{B}_k \lambda^k = -\nabla_x \mathcal{E}(x^k) \tag{8.8}$$

$$-\mathcal{B}_k^\top d^k = g\left(x^k\right)$$

where $\mathcal{W}_k$ is a positive-definite approximation to the true Hessian matrix derived using the *Broyden-Fletcher-Goldfarb-Shanno* formula (see Appendix K.7.1) 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 $\lambda^k$ and the direction vector $d^k$:

$$\begin{aligned}\lambda^k &= -\left(\mathcal{B}_k^\top \mathcal{W}_k^{-1}\mathcal{B}_k\right)^{-1}\left[g\left(x^k\right) - \mathcal{B}_k^\top \mathcal{W}_k^{-1}\nabla_x \mathcal{E}\left(x^k\right)\right]\\ d^k &= -\mathcal{W}_k^{-1}\left[\nabla_x \mathcal{E}\left(x^k\right) - \mathcal{B}_k\lambda^k\right]\end{aligned}$$

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. $\tau^k$ is found by a line search for the minimum of the Lagrangian function

$$\mathcal{L}\left(\tau^k\right) = \mathcal{E}\left(x^k + \tau^k d^k\right) - \sum_{i\in\mathcal{A}(x^k)} \lambda_i^k g_i\left(x^k + \tau^k d^k\right)$$

subject to the constraints. The iteration finishes when the *Karush-Kuhn-Tucker* conditions are satisfied.

### 8.1.2 Choice of active constraints

*Selesnick, Lang and Burrus* [75] 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 $S = \{f_1, \ldots, f_r\}$ 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 $S_L = \{f_1, \ldots, f_q\}$ and $S_U = \{f_{q+1}, \ldots, f_r\}$. To minimise $\mathcal{E}_A(x,f)$ subject to these constraints, form the Lagrangian

$$\mathcal{L}(x,\lambda) = \mathcal{E}_A(x,f) - \sum_{i=1}^{q}\lambda_i\left[A(x,f) - L(f)\right] - \sum_{i=q+1}^{r}\lambda_i\left[U(f) - A(x,f)\right]$$

At the minimum of $\mathcal{E}_A(x,f)$ the gradient $\nabla_x\mathcal{L}(x,\lambda) = 0$ and

$$\nabla_x\mathcal{E}_A(x,f) - \sum_{i=1}^{q}\lambda_i\nabla_x A(x,f) + \sum_{i=q+1}^{r}\lambda_i\nabla_x A(x,f) = 0$$

$$\begin{aligned}A(x,f_i) &= L(f_i) \quad \text{for} \quad 1 \le i \le q\\ A(x,f_i) &= U(f_i) \quad \text{for} \quad q+1 \le i \le r\end{aligned}$$

According to the *Karush-Kuhn-Tucker* conditions, when the Lagrange multipliers $\lambda_1, \ldots, \lambda_r$ are all non-negative, then the solution to these equations minimises $\epsilon(x,f)$ subject to the inequality constraints

$$A(x,f_i) \ge L(f_i) \quad \text{for} \quad 1 \le i \le q$$

**Algorithm 8.1** Exchange algorithm for multiband FIR filters. (*Selesnick, Lang and Burrus*. See [76, 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 $\omega_i$, for which the Lagrange multiplier $\lambda_i$ 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 $\omega$, 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 $\omega$, 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.

$$A(x, f_i) \leq U(f_i) \quad \text{for} \quad q+1 \leq i \leq r$$

*Selesnick et al.* [76, p.498] modify the algorithm of *Selesnick et al.* [75] so that it can be used in the design of multiband FIR filters, as shown in Algorithm 8.1. 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.* [75, 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 $\omega_i$ 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, in the past, published implementations of PCLS filter design in the files *cl2lp.m* and *cl2bp.m* that do not appear to be currently available [188] but are included in the source files for this project.

### 8.1.3 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* [81, p.2855]. The linearised magnitude response is:

$$\hat{A}(x) = A\left(x^k\right) + \nabla_x A\left(x^k\right)^\top \left(x - x^k\right)$$

The frequency response magnitude inequality constraints are, in the pass band

$$A_D - \hat{A}(x) \geq 0$$
$$\hat{A}(x) - [A_D - \Delta_{A_P}] \geq 0$$

and in the stop band

$$\Delta_{A_S} - \hat{A}(x) \geq 0$$

where $\Delta_{A_p}$ is the pass band ripple for the desired gain $A_D$ and $\Delta_{A_S}$ is the stop band ripple. After linearising about $x^k$ these constraints become

$$\begin{aligned} A_D - A\left(x^k\right) - \nabla_x A\left(x^k\right)^\top \left(x - x^k\right) &\geq 0 \\ A\left(x^k\right) + \nabla_x A\left(x^k\right)^\top \left(x - x^k\right) - \left[A_D - \Delta_{A_P}\right] &\geq 0 \\ \Delta_{A_S} - A\left(x^k\right) - \nabla_x A\left(x^k\right)^\top \left(x - x^k\right) &\geq 0 \end{aligned}$$

Similarly, the linearised group-delay is

$$\hat{T}(x) = T\left(x^k\right) + \nabla_x T\left(x^k\right)^\top \left(x - x^k\right)$$

and the group-delay inequality constraints in the pass band are

$$\begin{aligned} \left[T_D + \Delta_D\right] - \hat{T}(x) &\geq 0 \\ \hat{T}(x) - \left[T_D - \Delta_T\right] &\geq 0 \end{aligned}$$

where $\Delta_T = T_{\text{max}} - T_D = T_D - T_{\text{min}}$ 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

$$\begin{aligned} \left[T_D + \Delta_T\right] - T\left(x^k\right) - \nabla_x T\left(x^k\right)^\top \left(x - x^k\right) &\geq 0 \\ T\left(x^k\right) + \nabla_x T\left(x^k\right)^\top \left(x - x^k\right) - \left[T_D - \Delta_T\right] &\geq 0 \end{aligned}$$

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.

### 8.1.4 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* [116] 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 derivatives 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.* [10] 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* [118] describes a method for finding successive coefficient vectors based on Rouché's theorem. *Dumitrescu and Niemistø* [20] 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* [212] express the denominator polynomial as a product of second-order sections and derive a linear inequality stability constraint on the denominator coefficients. *Lu* [219] 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.

### 8.1.5 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 $\omega_p$ is

$$\begin{aligned} H(\omega) &= \sum_{k=-\infty}^{\infty} h_k e^{-\imath k\omega} \\ &= \begin{cases} 1, & |\omega| < \omega_p \\ 0, & \omega_p < |\omega| < \pi \end{cases} \end{aligned}$$

The coefficients of the impulse response are

$$\begin{aligned} h_k &= \frac{1}{2\pi} \int_{-\omega_p}^{\omega_p} e^{\imath k\omega} d\omega \\ &= \frac{1}{\pi k} \sin k\omega_p \\ &= \frac{\omega_p}{\pi} \operatorname{sinc} k\omega_p \end{aligned}$$

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

$$W_k = \begin{cases} \left[\alpha + (1-\alpha)\cos\frac{2\pi k}{N}\right], & |k| \le N \\ 0, & |k| > N \end{cases}$$

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 $f_S = 1$.

### Tarzcynski's method of unconstrained optimisation of an IIR initial filter

An arbitrary filter can be refined by unconstrained optimisation with a *"barrier"* function (see Appendix K.8.2). *Tarczynski et al.* [10], propose minimising the error with the following, so-called, *WISE* barrier function:

$$(1-\lambda)\,\mathcal{E}_H + \lambda \sum_{t=T+1}^{T+M} f^2(t) \tag{8.9}$$

where $\mathcal{E}_H$ is the filter response error defined in Equation 1, $\lambda$, $T$ 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 $\lambda$, $T$ and $M$. Typically, $\lambda \in \left[10^{-10}, 10^{-3}\right]$, $T \in [100, 500]$ and $M = RK$. The barrier function (the second part of Equation 8.9) is intended to be small when the filter is stable and increase rapidly otherwise. *Roberts and Mullis* [168, 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:

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -d_{N_d} & -d_{N_d-1} & \cdots & -d_1 \end{bmatrix} \\ B &= \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix}^\top \\ C &= \begin{bmatrix} -d_{N_d} & \cdots & -d_1 \end{bmatrix} \\ D &= 1 \end{aligned}$$

Alternatively, for the filter transfer function, $H(z)$, given by Equation 8.1, *Tarczynski et al.* [10, 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 $|p_1| \geq |p_2| \geq \ldots \geq |p_{N_d}|$, then the state variable 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 variable model is (for $R = 1$):

$$\begin{aligned} A &= \begin{bmatrix} p_1\rho^{-1} & 0 & 0 & \cdots & 0 \\ 1 & p_1\rho^{-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & p_{N_d}\rho^{-1} \end{bmatrix} \\ B &= \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \end{bmatrix}^\top \\ C &= \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \\ D &= 0 \end{aligned}$$

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* [49, Algorithm 11.2.2] show an algorithm, reproduced as Algorithm 8.2, for efficiently computing the powers of a matrix by cumulative products of the binary powers of the matrix. This algorithm requires at most $2 \times \text{floor}\left[\log_2(s)\right]$ matrix multiplies. If $s$ is a power of 2, then only $\log_2(s)$ matrix multiplies are needed.

**Algorithm 8.2** 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 a.l* Example 2 [10] with $nN = 24$, $nD = 2$ and $R = 2$ by unconstrained minimisation of Equation 8.9 with *fminunc*. The resulting response is shown in Figure 8.1 and the pole-zero plot of the filter is shown in Figure 8.2.

The Octave function *xInitHd* uses the WISE barrier function to design an initial filter in polynomial form and then calls the Octave *qroots* function to convert that polynomial into *gain-pole-zero* form. As noted in the Introduction, finding the roots of a polynomial is a difficult problem in numerical analysis and the output of the *qroots* function depends on the CPU architecture, operating system, library versions, compiler version and Octave version.

### Surma-aho and Saramäki method of designing an IIR initial filter

Appendix M.4.3 describes the method of *Surma-aho* and *Saramäki* [96] for designing an initial low-pass filter implemented as either a single filter or as the parallel combination of two all-pass filters. Unfortunately, this method requires root-finding of intermediate polynomials. I find the method of *Tarczynski et al.* to be more useful.

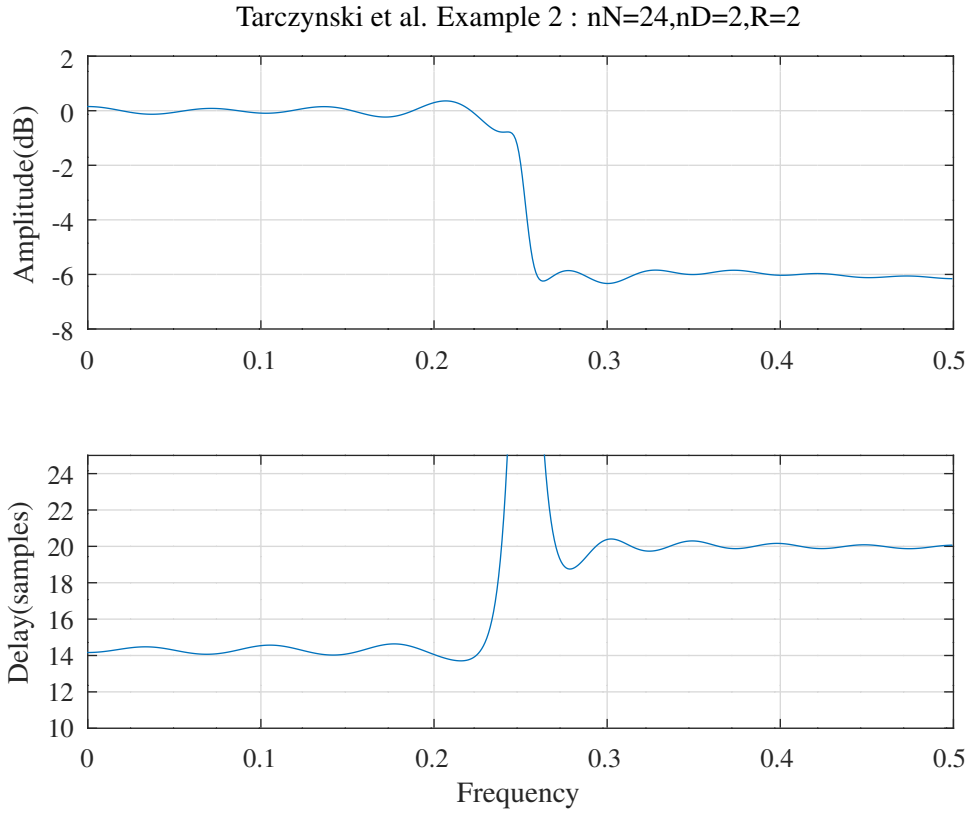



Figure 8.1: Tarczynski et al. Example 2:Response for nN=24, nD=2 and R=2 with

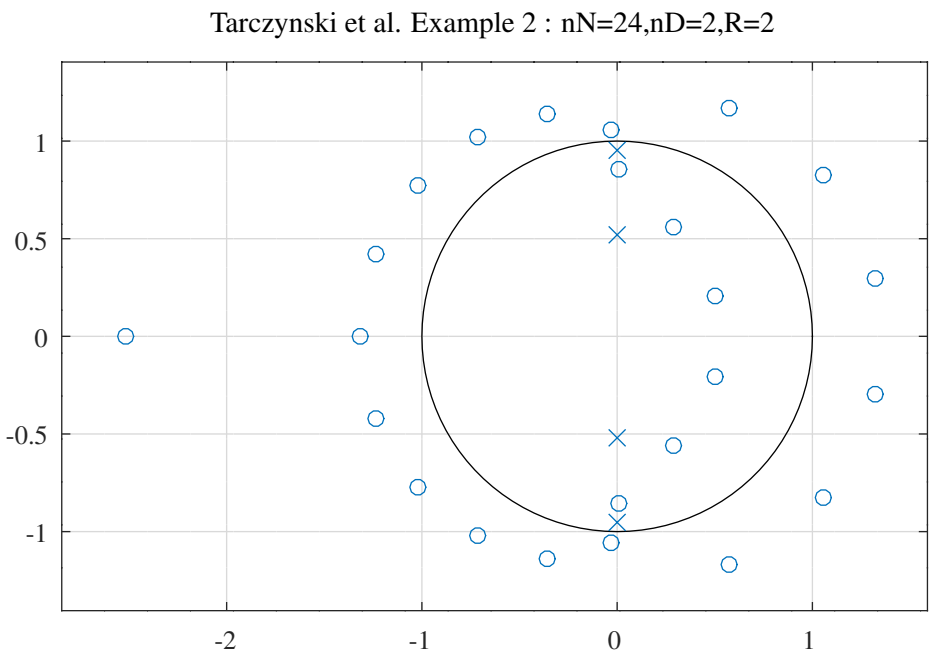



Figure 8.2: Tarczynski et al. Example 2:Pole-zero plot for nN=24, nD=2 and R=2

## 8.2 Examples of IIR filter design with SQP and constrained pole and zero locations

### 8.2.1 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 8.1.1.
- iteratively apply constraints with the peak-constrained-least-square-error (PCLS) algorithm described in Section 8.1.2.

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.

**!!! WARNING !!! The *iirA*, *iirP* and *iirT* functions do not attempt to handle the discontinuity and non-differentiability of the properties of a zero at $z = 1$ with $\omega = 0$.** In general, I have found that this does not cause difficulties but may result in a sub-optimal result. The differentiator examples of Section 8.2.8 and Section 8.2.9 assume a zero at $z = 1$ and design a correction filter that provides the desired combined response.

#### 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 8.1.5 and *Tarczynski et al.* [10]. The coefficient constraints are defined in the Octave function *xConstraints*. In fact I only constrain the pole radiuses 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 K.10 works through the derivation of the *Goldfard-Idnani* algorithm [33] 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 radiuses are constrained to $|r| < rho < 1$. The other parts of the coefficient vector (scale factor, zero radiuses and pole and zero angles) are not constrained. The *vS* argument to *iir_sqp_mmse* specifies the indexes into the frequency vectors $\omega_a$ 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. A useful future improvement is to add a constraint on the slope of the amplitude response in the transition band. For a lowpass filter: $\frac{\partial A}{\partial \omega} < 0$.

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 K.9.2 and Appendix K.7.1 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 comined 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* [76, Fig.4,p.499] (reproduced above as Algorithm 8.1), 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 [86, Section IV].

Figure 8.3 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.

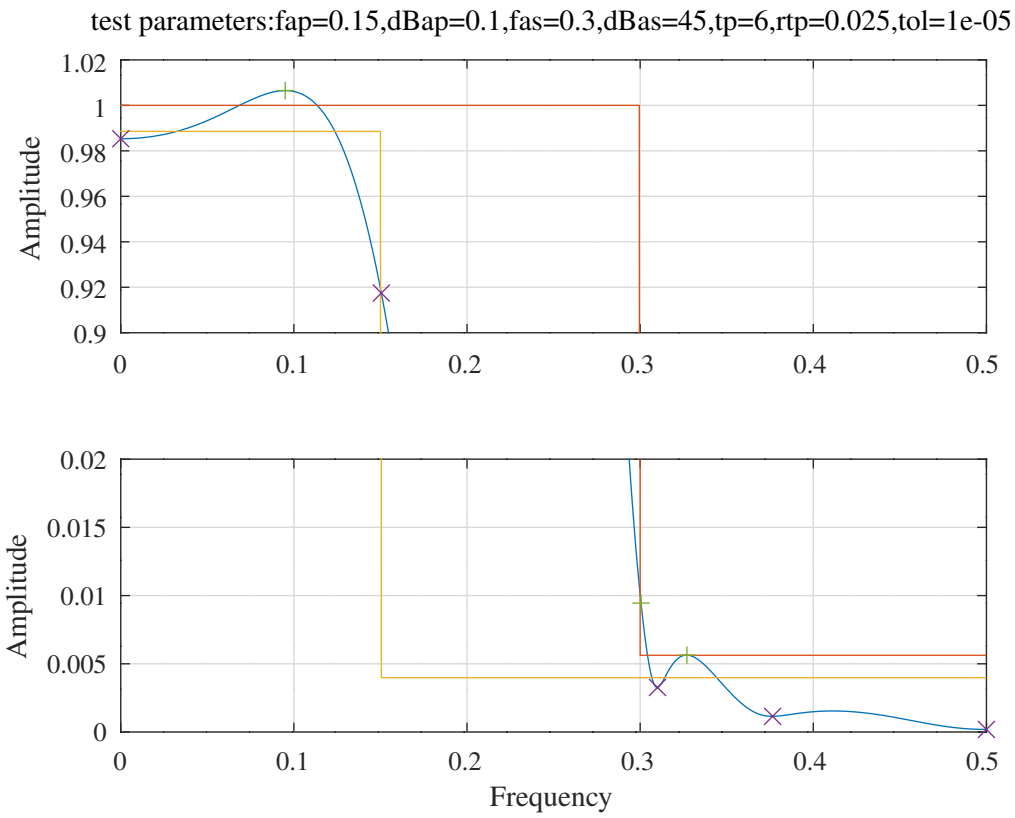



Figure 8.3: Example of failed constraints for a low pass filter

### 8.2.2 Tarczynski et al. Example 2

Figure 8.1 shows the response of a filter designed with the "WISE" method of *Tarczynski et al.* [10]. The coefficients of the numerator and denominator polynomials of the filter are:

```
N0 = [   0.0055315897,   0.0168955333,   0.0074742723,  -0.0015222109, ...
        -0.0019754772,   0.0069414003,   0.0033975862,  -0.0102843593, ...
        -0.0055112323,   0.0171245164,   0.0104436381,  -0.0353405894, ...
        -0.0284862390,   0.1348459416,   0.4155094295,   0.6323670625, ...
         0.6374912396,   0.4464475766,   0.1789030168,  -0.0679324192, ...
         0.2506269955,  -0.3305101375,   0.2960012935,  -0.1721597698, ...
         0.0604549583 ]';


D0 = [   1.0000000000,   1.1782065652,   0.2453764111 ]';
```

In the above listing, the first line shows the gain, and subsequent lines show, respectively, 2 real zeros, 2 real poles, the zero radiuses 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, ...
         0.6034298000,   0.8306859000,   1.0533570000,   1.1899991000, ...
         1.2418447000,   1.2794257000,   1.3011204000,   1.3091281000, ...
         1.3386509000,   1.3538295000, ...
         0.9197805000,   1.5328472000,   1.6013209000,   1.8790233000, ...
         2.1870677000,   2.5029369000,   1.1173639000,   2.8187821000, ...
         0.6631062000,   0.2204128000 ]';
```

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 = [   0.0017320719, ...
        -1.3848301093,   0.4125509029,   0.4125509029, ...
        -0.5535583491,   0.0111284883, ...
         0.5862376536,   0.8555459647,   1.2340323025,   1.3457959128, ...
         1.3766498387,   1.3890052592,   1.3904153941,   1.5201431901, ...
         1.5763473382,   1.5841438418, ...
         0.9389116881,   1.5229001580,   1.6851391399,   1.9749880919, ...
         2.2541329622,   2.5431581836,   2.8379025736,   1.0191948043, ...
         0.2063733702,   0.6493642166 ]';
```

The corresponding transfer function numerator and denominator polynomials are, respectively:

```
N1 = [   0.0017320719,   0.0010567346,  -0.0020670973,  -0.0023919018, ...
         0.0048179403,   0.0072124023,  -0.0064617736,  -0.0182992204, ...
         0.0002677316,   0.0271697651,   0.0051420244,  -0.0541760808, ...
        -0.0295752284,   0.1678144148,   0.4345163431,   0.5316776331, ...
         0.3558168730,   0.0911637847,  -0.0022310380,   0.1178985225, ...
        -0.2444893149,   0.2464467626,  -0.1425491230,   0.0288515415 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,   0.5424298608,   0.0000000000, ...
        -0.0061602676 ]';
```

Figure 8.4 shows the response of the optimised filter, Figure 8.5 shows the pass-band response and Figure 8.6 shows the corresponding pole-zero plot.

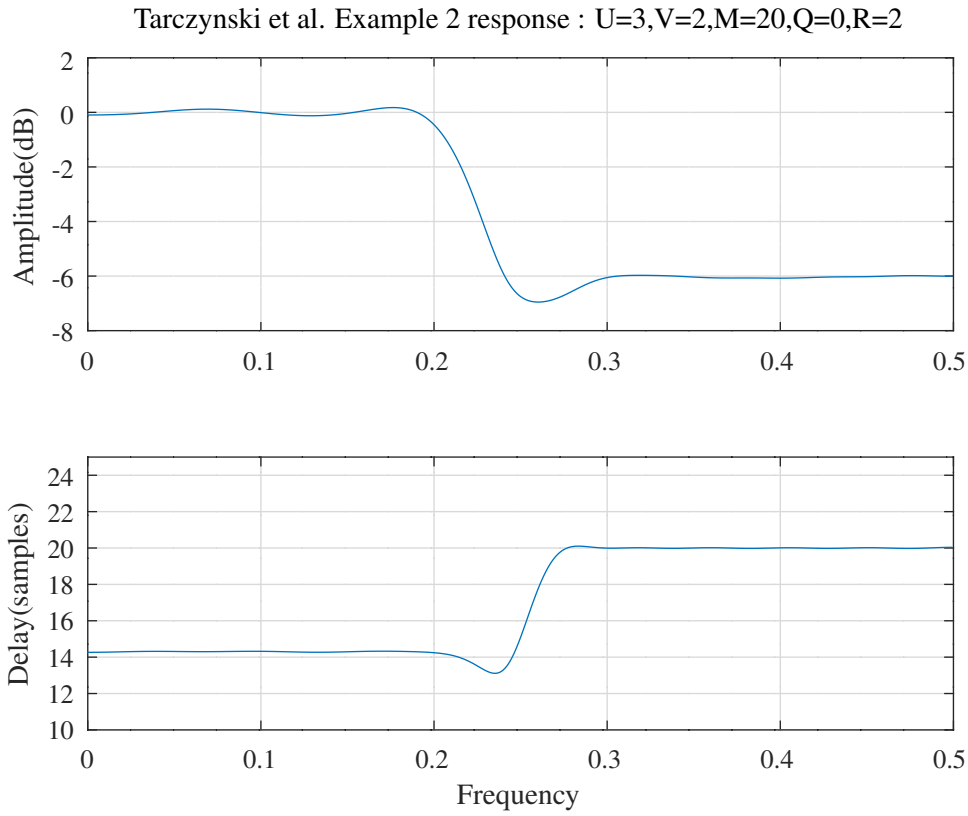



Figure 8.4: Tarczynski et al. Example 2 response after MMSE optimisation

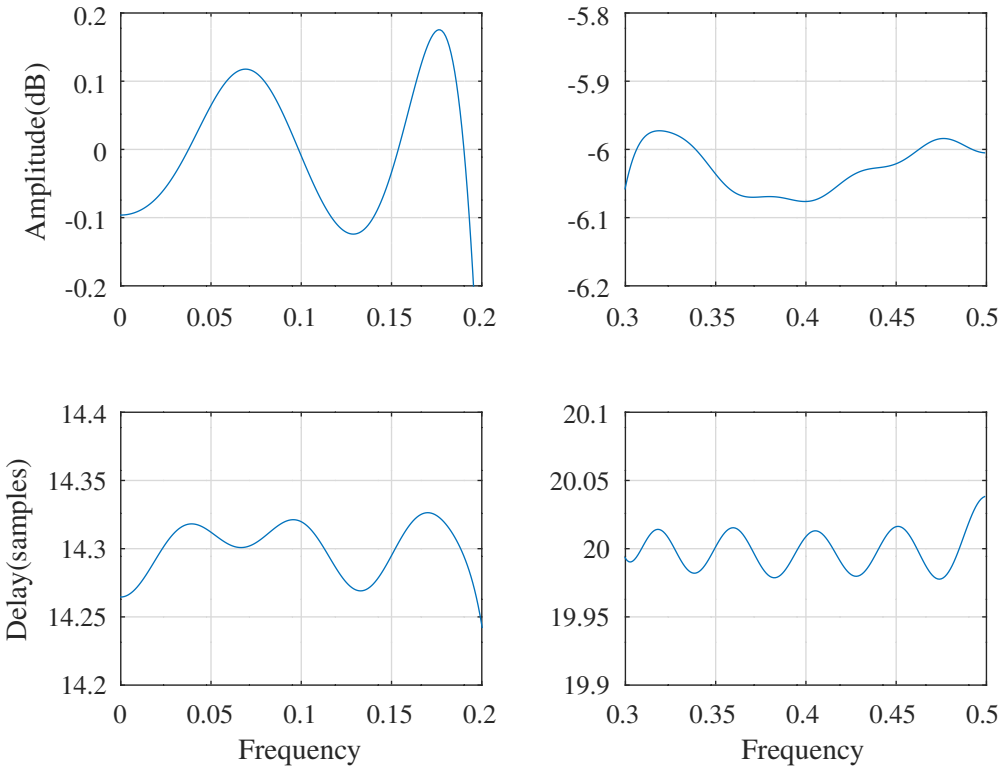



Figure 8.5: Tarczynski et al. Example 2 pass-band response after MMSE optimisation

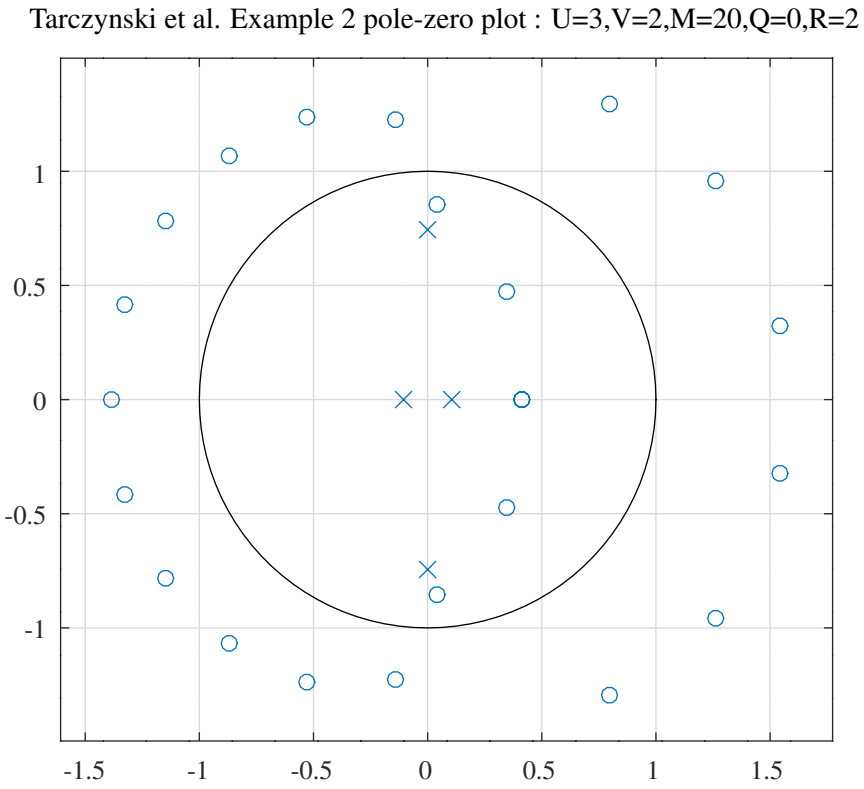



Figure 8.6: Tarczynski et al. Example 2 pole-zero plot after MMSE optimisation

### 8.2.3 Deczky's Example 3

*Sullivan and Adams* [81, 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
ctol=0.0002 % Tolerance on constraints
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=0.004 % Pass band group delay peak-to-peak ripple
Wtp_mmse1=0.125 % Pass band group delay weight(MMSE pass 1)
Wtp_mmse2=0.5 % Pass band group delay weight(MMSE pass 2)
Wtp_pcls=2 % Pass band group delay weight(PCLS 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" [81, 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 8.7. The corresponding pole-zero plot is shown in Figure 8.8.

#### MMSE optimisation of Deczky's Example 3

With weights $W_{ap} = W_{as} = 1$ and $W_{tp} = 0.125$ the first MMSE optimisation pass gives the response shown in Figure 8.9. After some experimentation, the second iteration, with $W_{tp} = 0.5$, gives the response shown in Figure 8.10 with pass-band details shown in Figure 8.11 and the pole-zero plot shown in Figure 8.12.

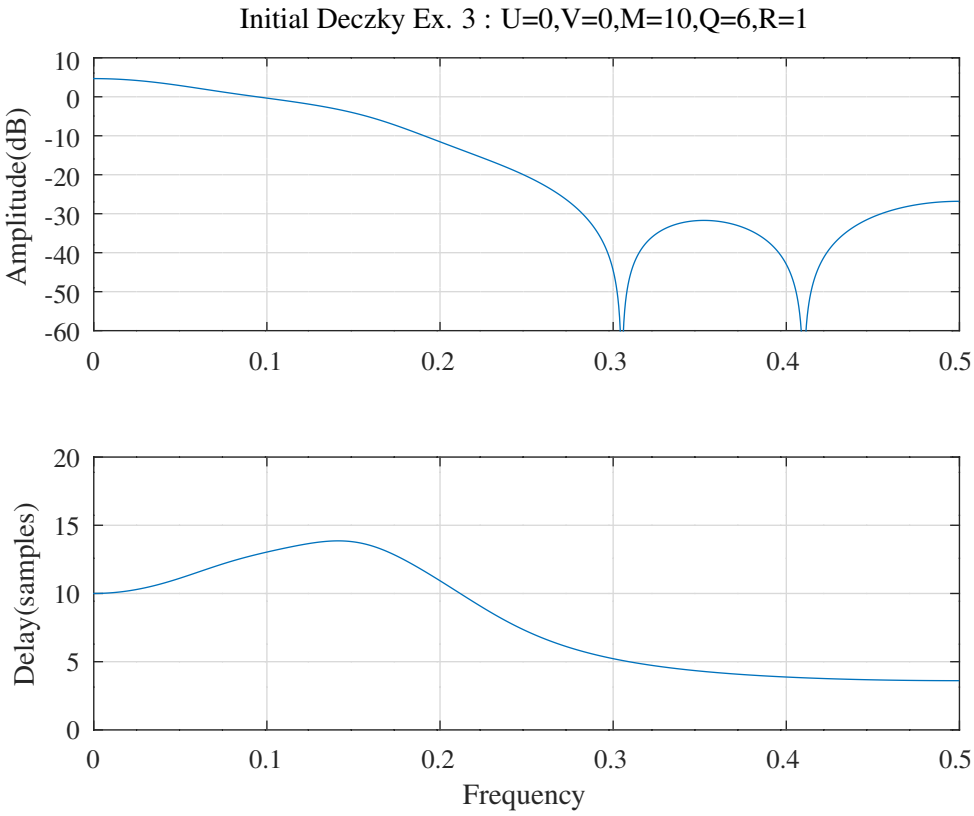



Figure 8.7: Deczky Example 3:Response for initial coefficients

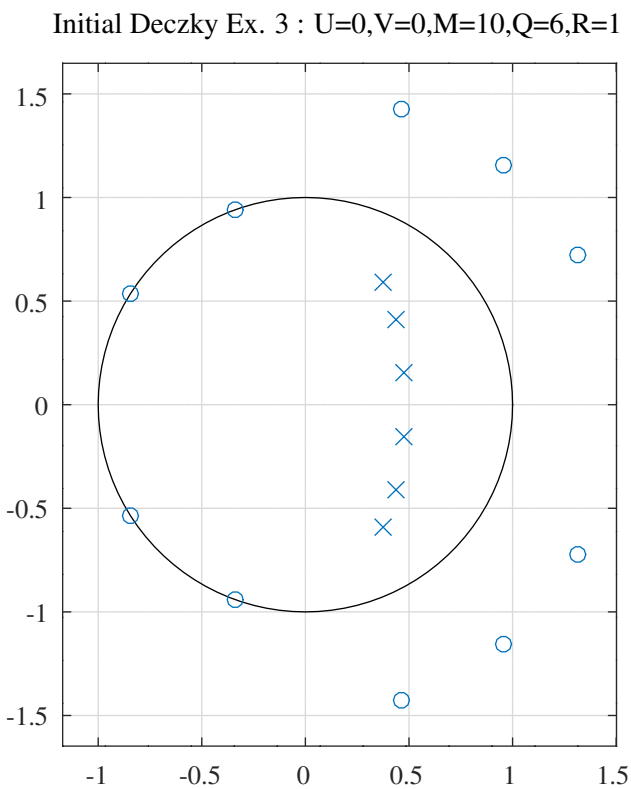



Figure 8.8: Deczky Example 3:Pole-zero plot for initial coefficients

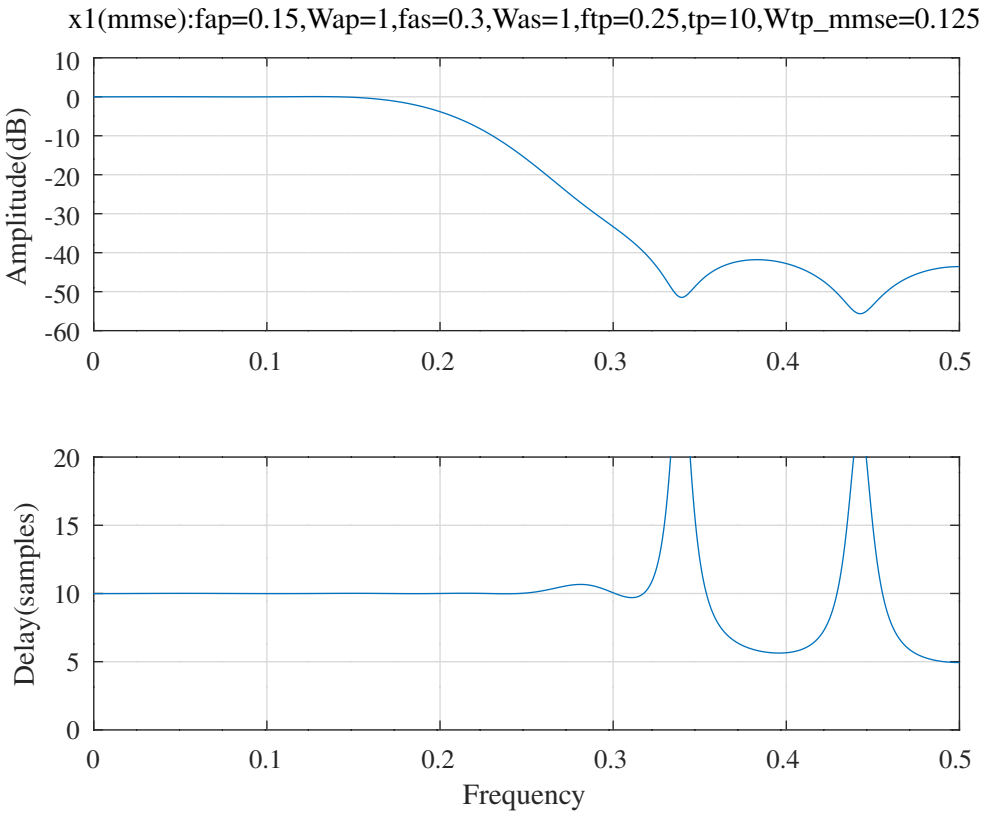



Figure 8.9: Deczky Example 3 MMSE optimised response after pass 1

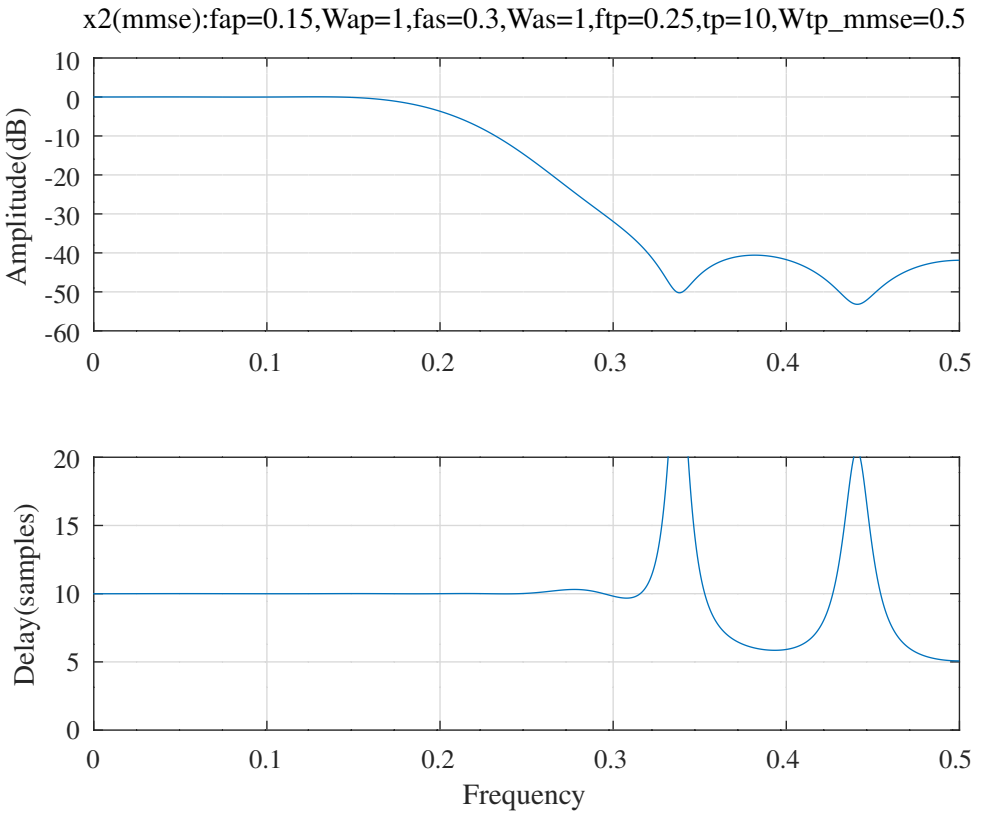



Figure 8.10: Deczky Example 3 MMSE optimised response after pass 2

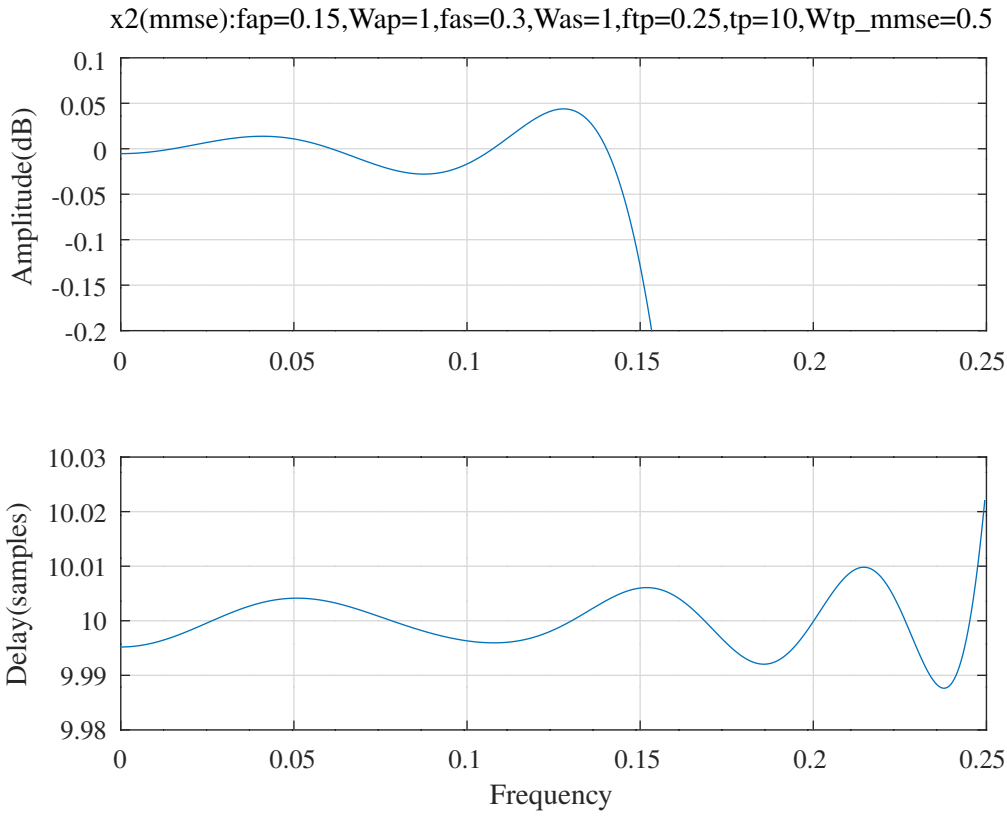



Figure 8.11: Deczky Example 3 MMSE optimised passband response after pass 2

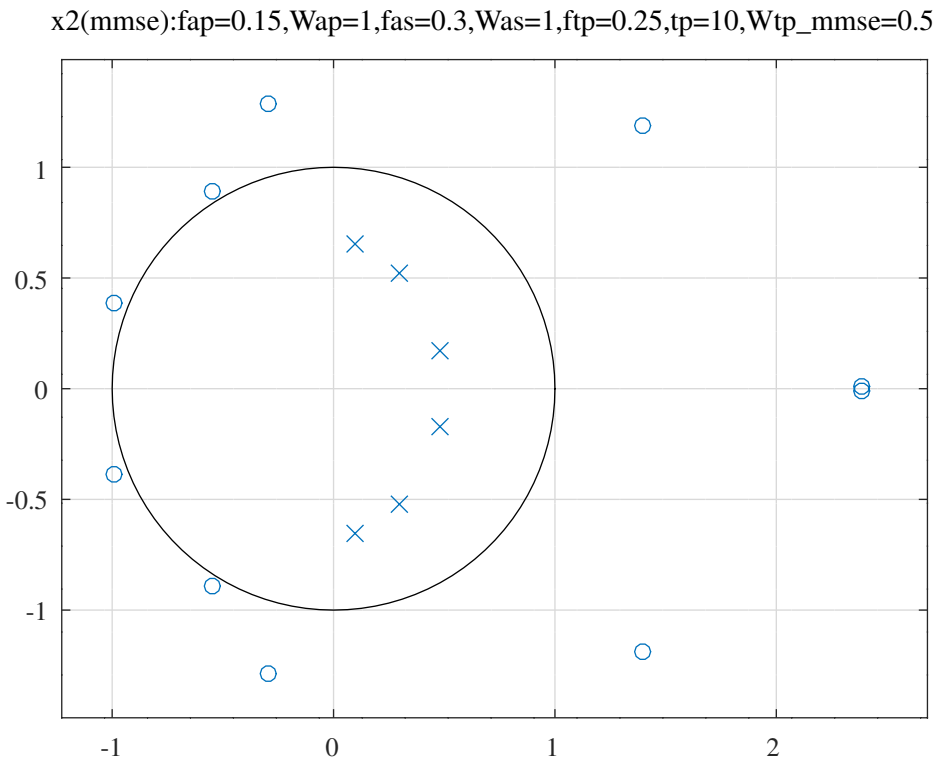



Figure 8.12: Deczky Example 3 MMSE optimised pole-zero plot after pass 2

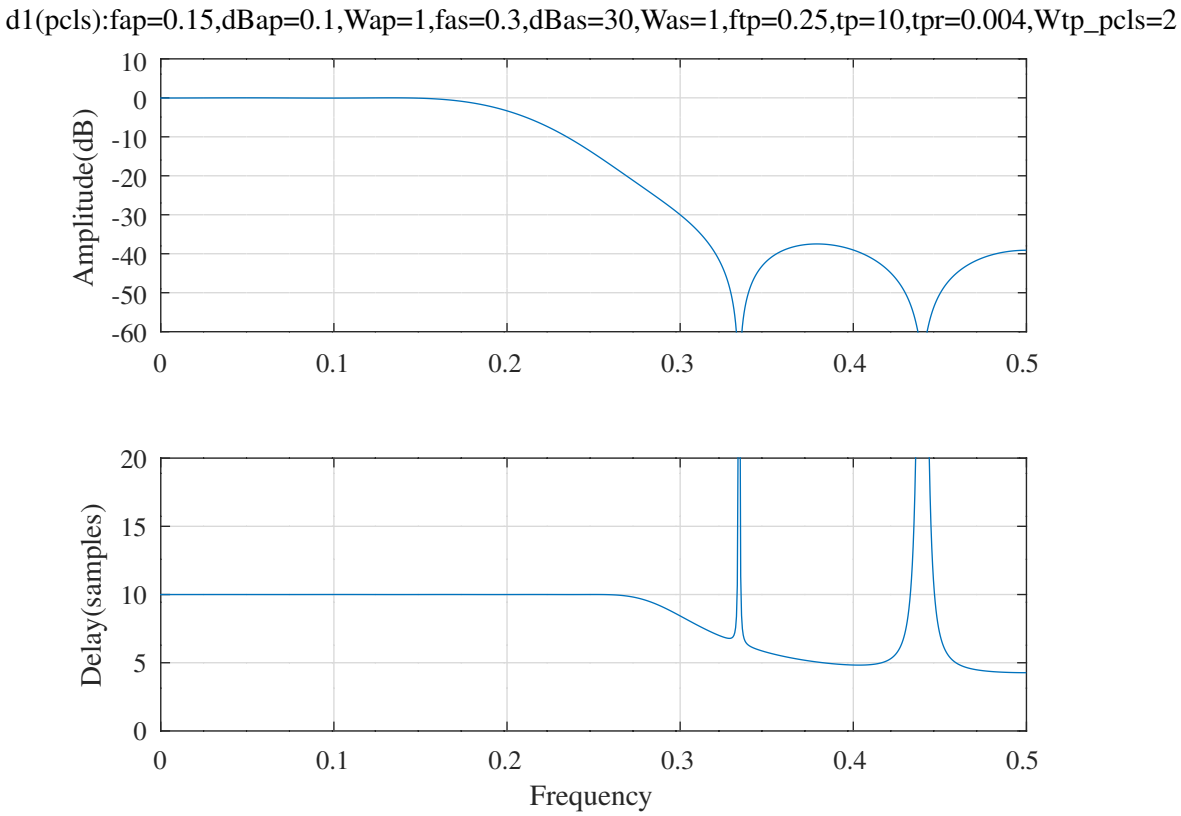



Figure 8.13: Deczky Example 3 PCLS optimised response

**PCLS optimisation of Deczky's Example 3**

The test script *deczky3_sqp_test.m* now switches to PCLS 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 $tpr = 0.004$ samples of peak-to-peak group delay ripple. The resulting amplitude and delay responses are shown in Figure 8.13 with pass-band detail shown in Figure 8.14. The corresponding pole-zero plot is shown in Figure 8.15. The optimised filter vector is, in the gain, zeros and poles form of Equation 8.2:

```
Ud1=0,Vd1=0,Md1=10,Qd1=6,Rd1=1
d1 = [   0.0030704976, ...
         1.0001739287,   1.0118446097,   1.3647214586,   1.7749833724, ...
         2.2790153173, ...
         2.0987301339,   2.7642360110,   1.7523094587,   0.7235434506, ...
         0.0028126273, ...
         0.5155405535,   0.5992052376,   0.6425952043, ...
         0.3549279331,   1.0886898987,   1.4319416748 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0030704976,  -0.0117811692,   0.0100287988,  -0.0022760865, ...
         0.0139082930,  -0.0089302922,  -0.0347952200,  -0.0001878953, ...
         0.0971260026,   0.1329671048,   0.0958425975 ]';
```

and

```
D1 = [   1.0000000000,  -1.7003360437,   1.8457746531,  -1.3301796507, ...
         0.6632811601,  -0.2212965274,   0.0394050451 ]';
```

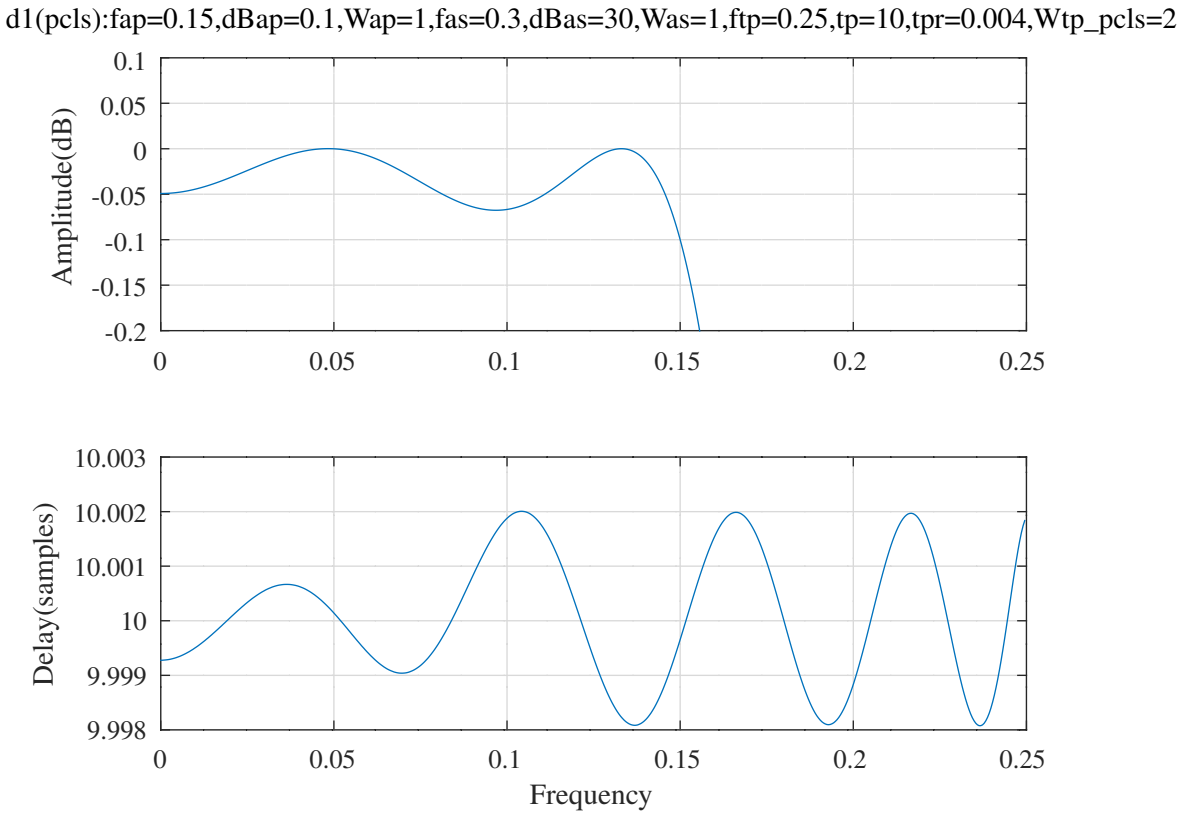



Figure 8.14: Deczky Example 3 PCLS optimised passband response

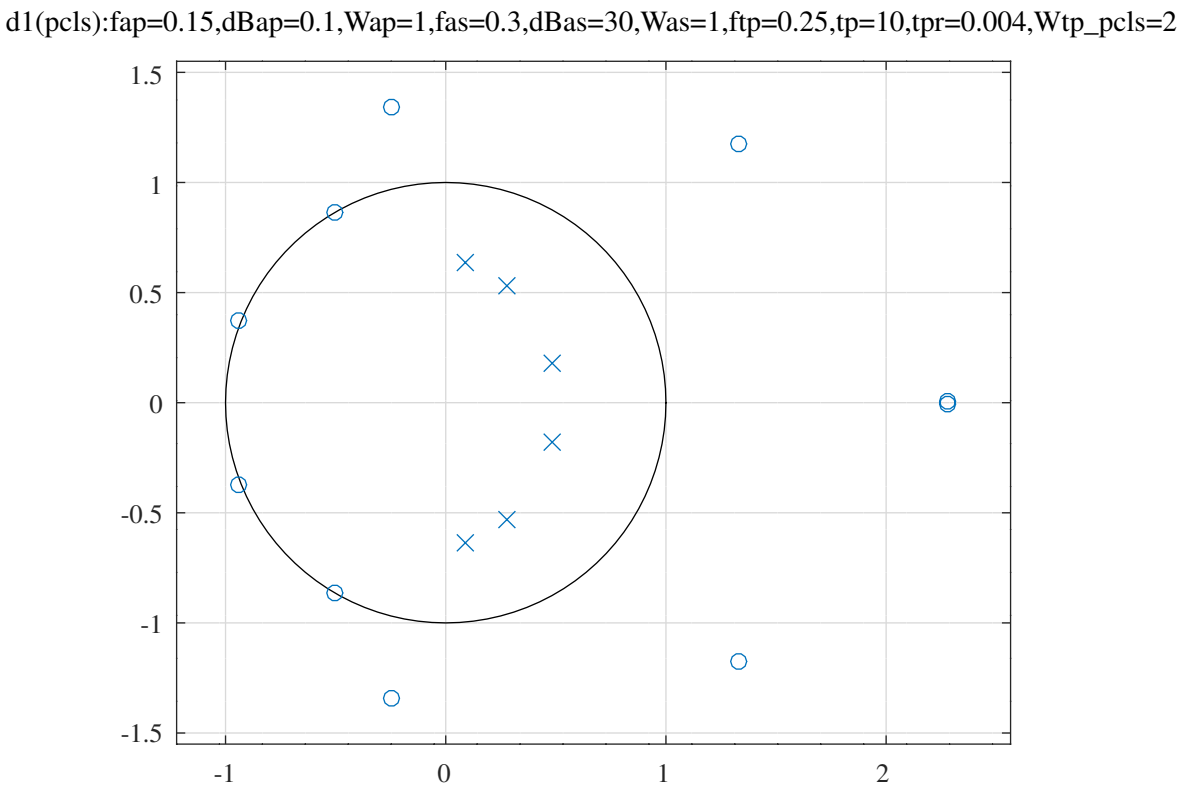



Figure 8.15: Deczky Example 3 PCLS optimised pole-zero plot

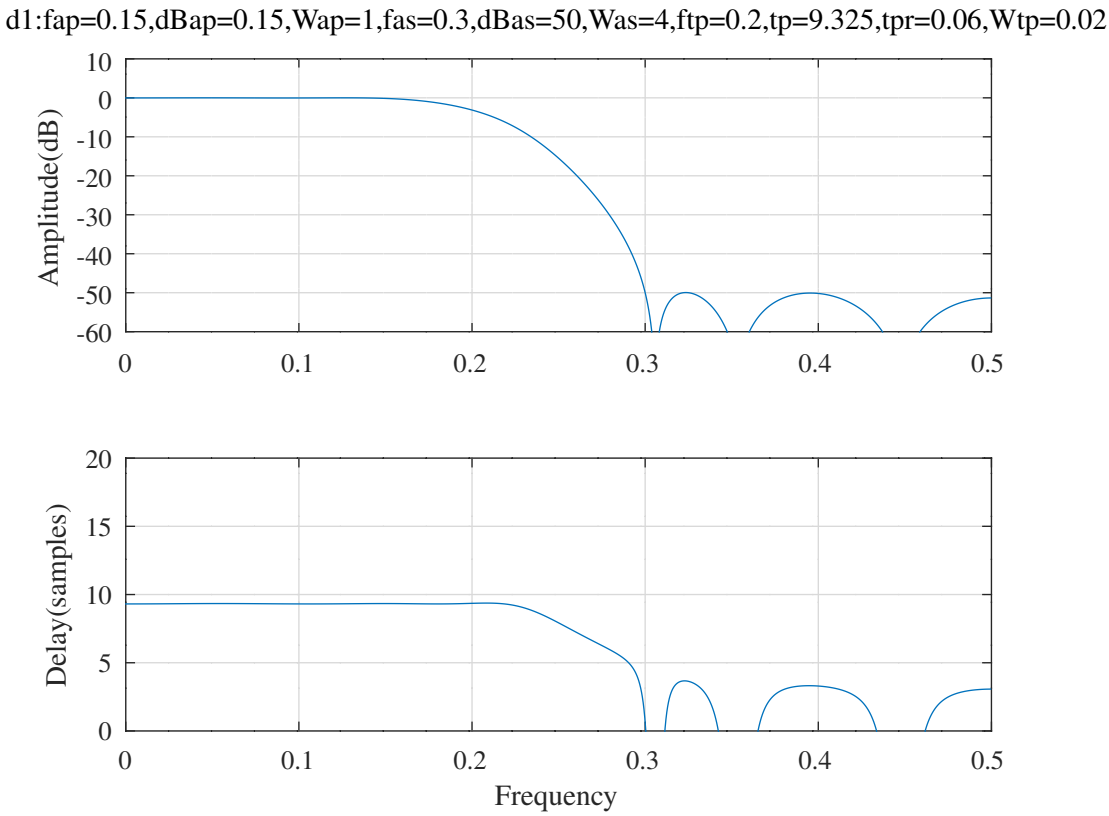



Figure 8.16: Deczky Example 3a PCLS optimised response

**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 filter specification:

```
n=1000 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-05 % Tolerance on constraints
fap=0.15 % Pass band amplitude response edge
dBap=0.15 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.2 % Pass band group delay response edge
tp=9.325 % Nominal filter group delay
tpr=0.06 % 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=50 % Stop band minimum attenuation
Was=4 % 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 resulting amplitude and delay responses are shown in Figure 8.16 with pass-band detail shown in Figure 8.17. The corresponding pole-zero plot is shown in Figure 8.18. The PCLS optimised filter vector is, in gain-zero-pole form:

```
Ud1=0,Vd1=0,Md1=10,Qd1=6,Rd1=1
d1 = [   0.0019791532, ...
         0.9688731542,   0.9759568563,   0.9938308954,   1.9485396413, ...
         2.5951292185, ...
         2.8123808088,   2.2211380993,   1.9210175287,   0.7332407417, ...
         0.0322179580, ...
         0.5153725948,   0.6113373842,   0.7286063751, ...
         0.3571156768,   1.0458032556,   1.4323505163 ]';
```

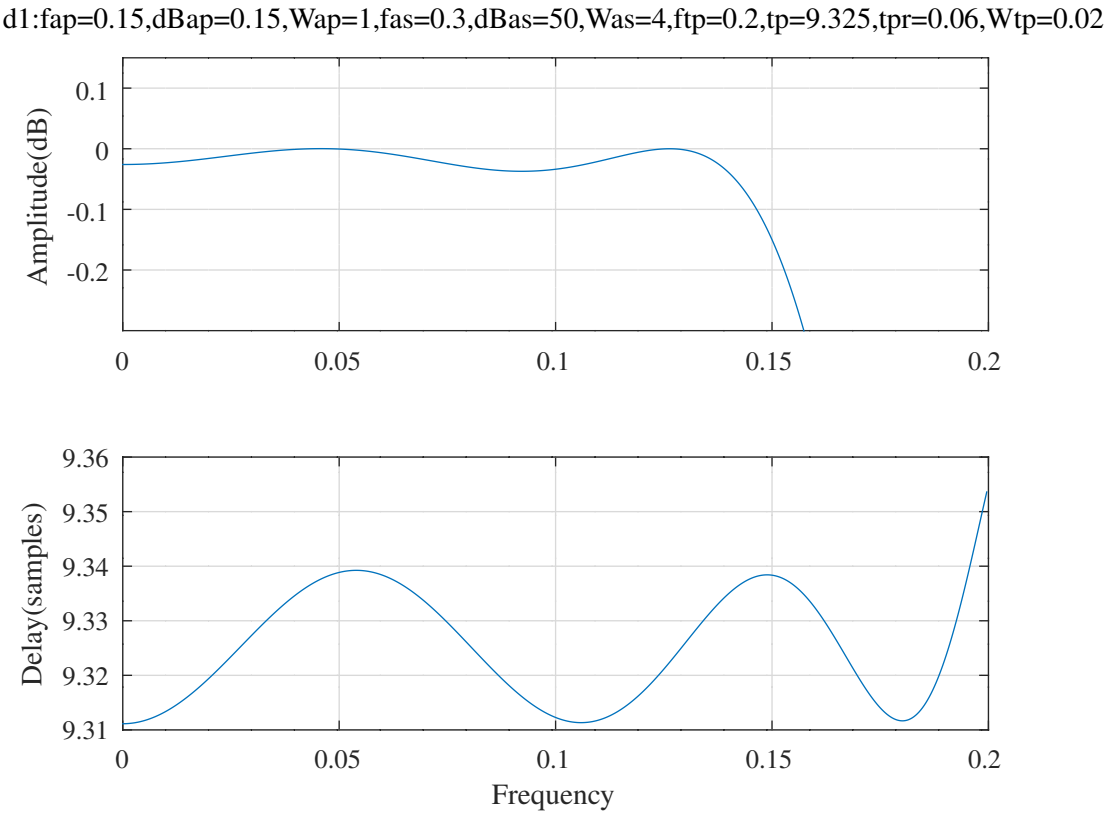



Figure 8.17: Deczky Example 3a PCLS optimised passband response

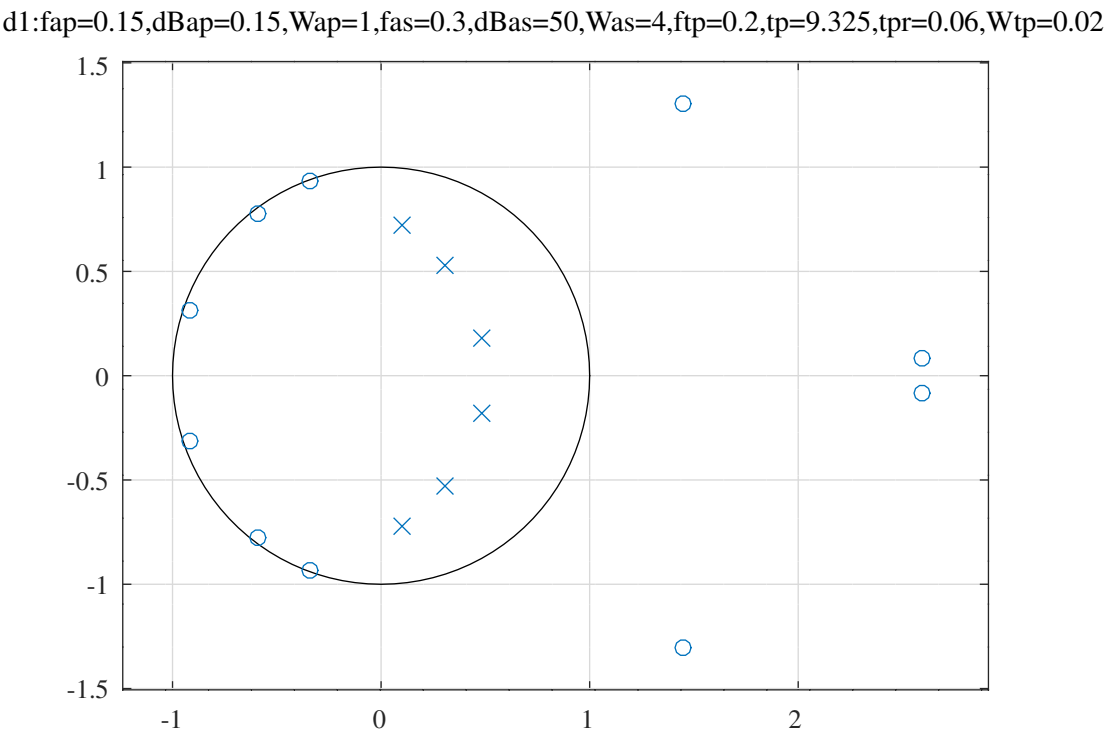



Figure 8.18: Deczky Example 3a PCLS optimised pole-zero plot

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0019791532,  -0.0086799055,   0.0054779039,   0.0128134726, ...
        -0.0010978817,  -0.0316416527,  -0.0145822692,   0.0632776781, ...
         0.1251975624,   0.1051041390,   0.0446926335 ]';
```

and

```
D1 = [   1.0000000000,  -1.7796287457,   2.0794557784,  -1.6092616136, ...
         0.8581557206,  -0.2979716997,   0.0526975573 ]';
```

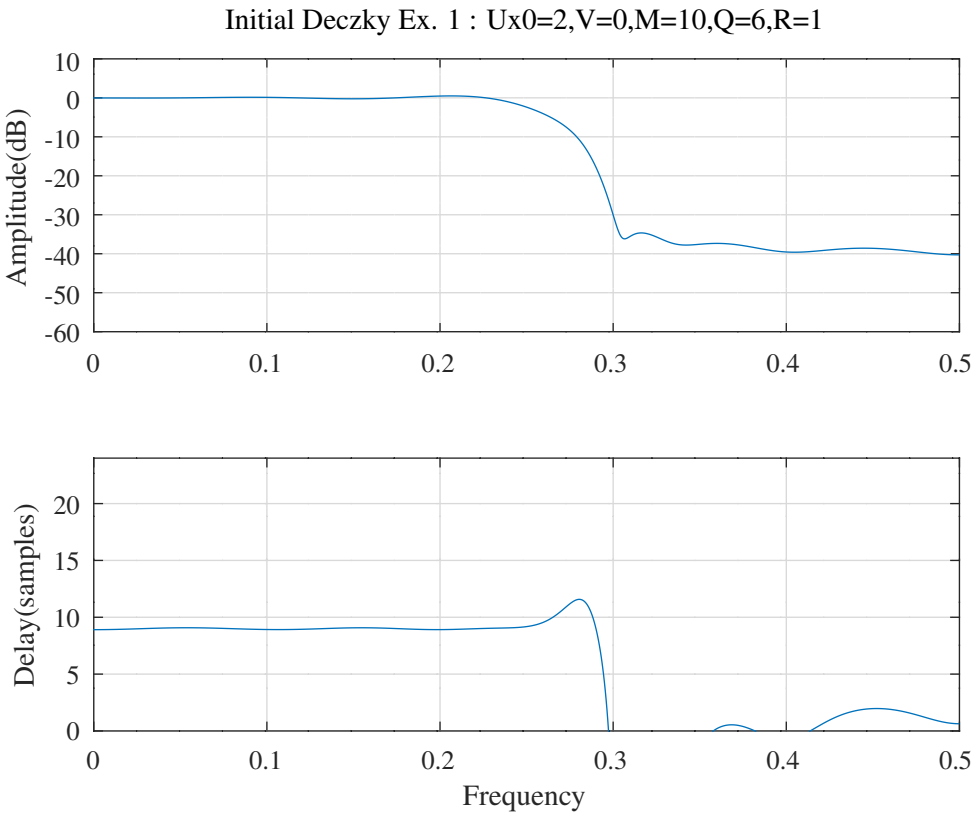



Figure 8.19: Deczky Example 1:Response for initial coefficients

### 8.2.4 Deczky's Example 1

The following example IIR filter design specification is similar to *Deczky*'s Example 1 [3]. The pass-band is $[0.0, 0.25]$ with $1dB$ maximum amplitude ripple, the pass-band delay is 8 samples with 2 sample maximum peak-to-peak ripple and the stop band is $[0.3, 0.5)$ with at least $36dB$ attenuation. The PCLS optimisation in this example includes a constraint on the derivative of the amplitude response in the transition band: $\frac{\partial A(\omega)}{\partial \omega} < 0$. The Octave script *deczky1_sqp_test.m* implements this example. The filter specification defined in that file is:

```
n=400 % Frequency points across the band
tol=0.0002 % Tolerance on relative coefficient update size
ctol=2e-06 % Tolerance on constraints
fap=0.25 % Pass band amplitude response edge
dBap=1 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
Wat=0.01 % Transition band weight
ftp=0.25 % Pass band group delay response edge
tp=8 % Nominal filter group delay
tpr=2 % Pass band group delay peak-to-peak ripple
Wtp=0.01 % Pass band group delay weight
fas=0.3 % Stop band amplitude response edge
dBas=36 % Stop band minimum attenuation
Was=1 % Stop band amplitude weight
Ux0=2 % Number of real zeros
Vx0=0 % Number of real poles
Mx0=10 % Number of complex zeros
Qx0=6 % Number of complex poles
Rx0=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 8.19.

#### MMSE optimisation of Deczky's Example 1

The frequency response after MMSE optimisation is shown in Figure 8.20 and the pole-zero plot is shown in Figure 8.21.

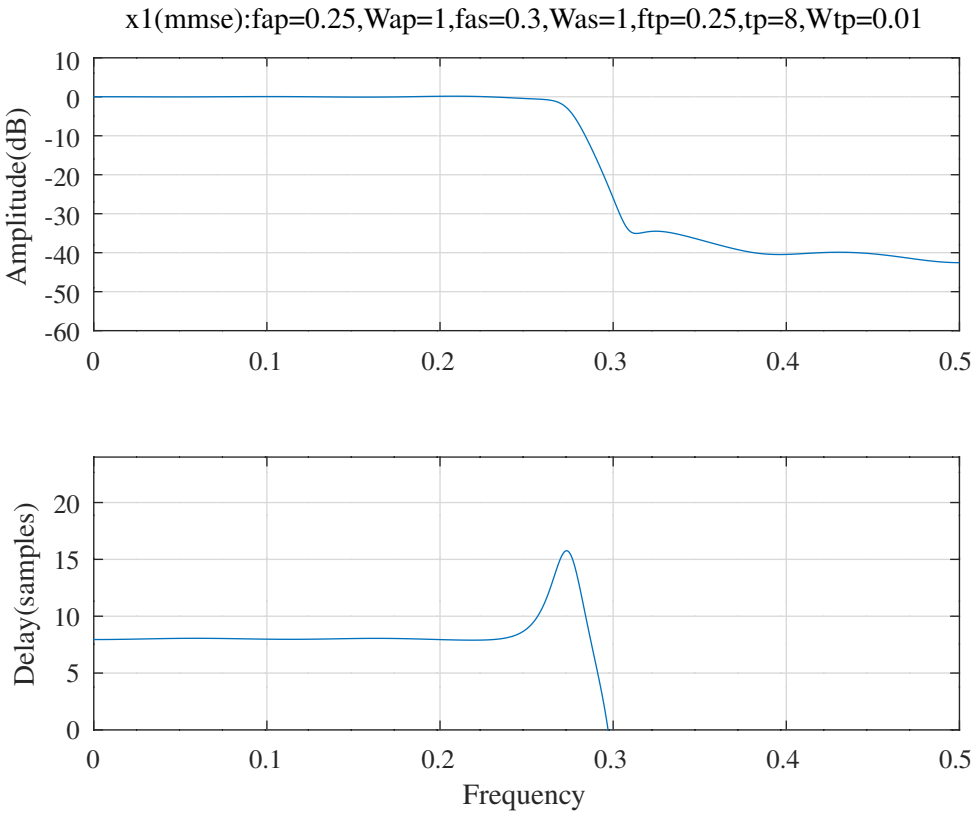



Figure 8.20: Deczky Example 1 MMSE optimised response

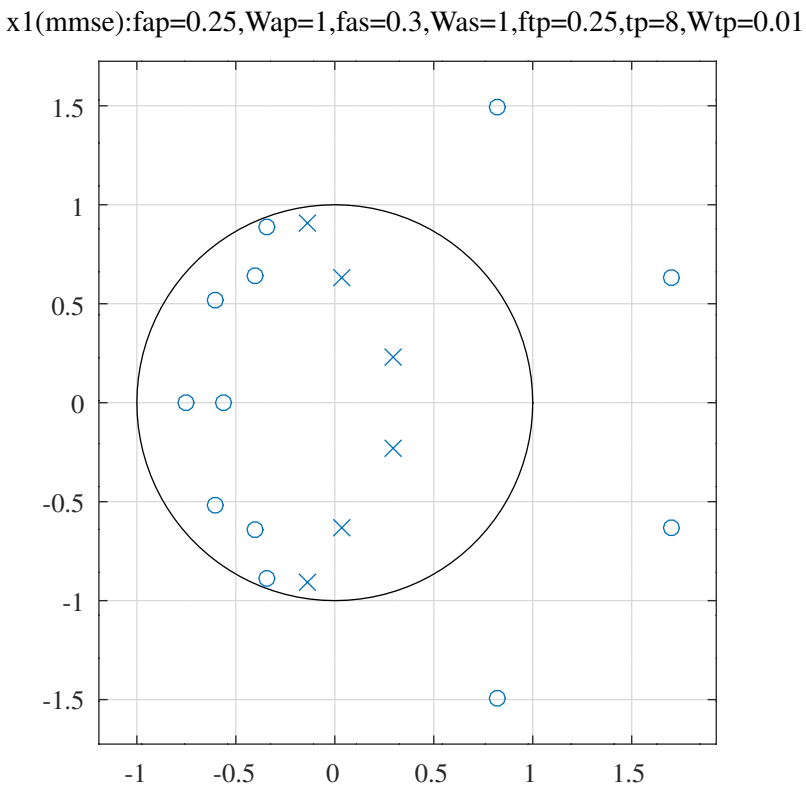



Figure 8.21: Deczky Example 1 MMSE optimised pole-zero plot

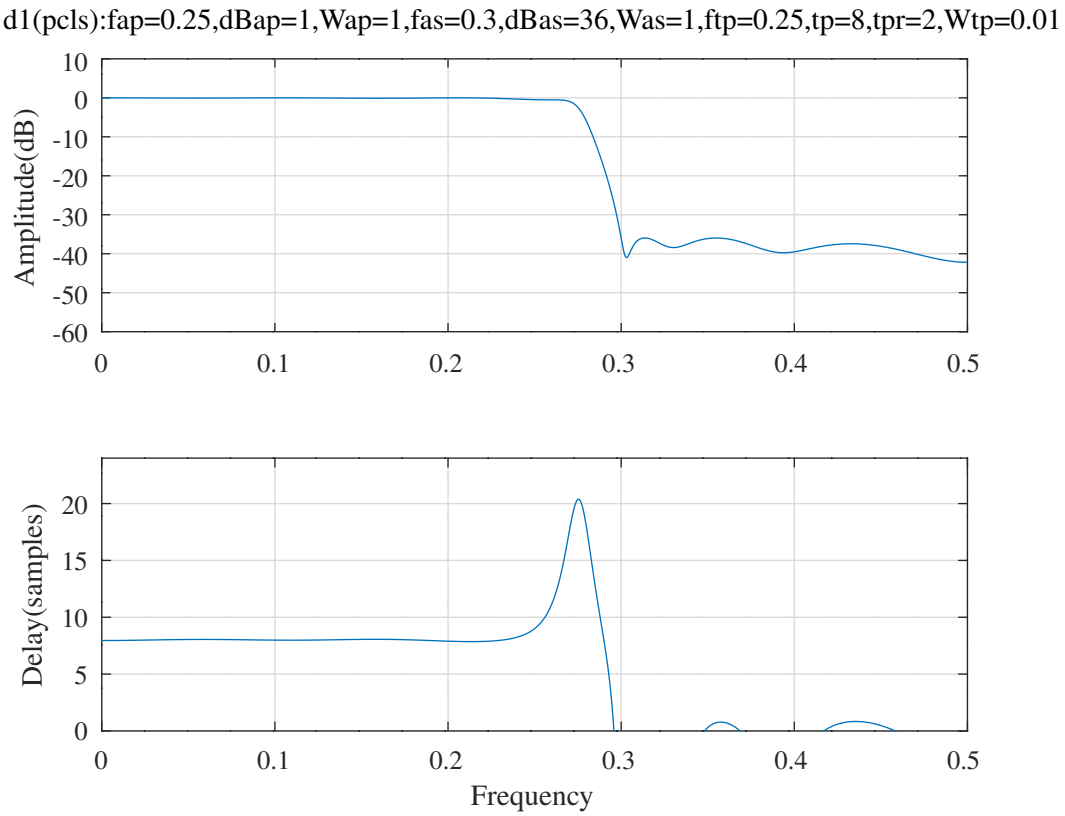



Figure 8.22: Deczky Example 1 PCLS optimised response

**PCLS optimisation of Deczky's Example 1**

The frequency response after PCLS optimisation is shown in Figure 8.22, the pass-band detail is shown in Figure 8.23 and the pole-zero plot is shown in Figure 8.24. The optimised filter vector is, in gain, zeros and poles form:

```
Ud1=2,Vd1=0,Md1=10,Qd1=6,Rd1=1
d1 = [   0.0155799430, ...
        -0.7523036858,  -0.7518665708, ...
         0.8703222369,   0.9220339906,   0.9840007279,   1.7221325146, ...
         1.8264537434, ...
         2.4570170382,   2.0641445282,   1.9010029088,   1.0642566229, ...
         0.3580108247, ...
         0.2896776015,   0.6000679611,   0.9347762276, ...
         0.8019832955,   1.6827767399,   1.7324084130 ]';
```

and the corresponding transfer function numerator and denominator polynomials are, respectively:

```
N1 = [   0.0155799430,  -0.0113462677,  -0.0025824969,   0.0115591764, ...
         0.0144223301,  -0.0362073247,  -0.0193822455,   0.2045861594, ...
         0.5293124063,   0.6802092311,   0.5259670795,   0.2466265127, ...
         0.0543619693 ]';
```

and

```
D1 = [   1.0000000000,   0.0321241645,   1.1829451956,  -0.2512736417, ...
         0.3307275280,  -0.1078193164,   0.0264025595 ]';
```

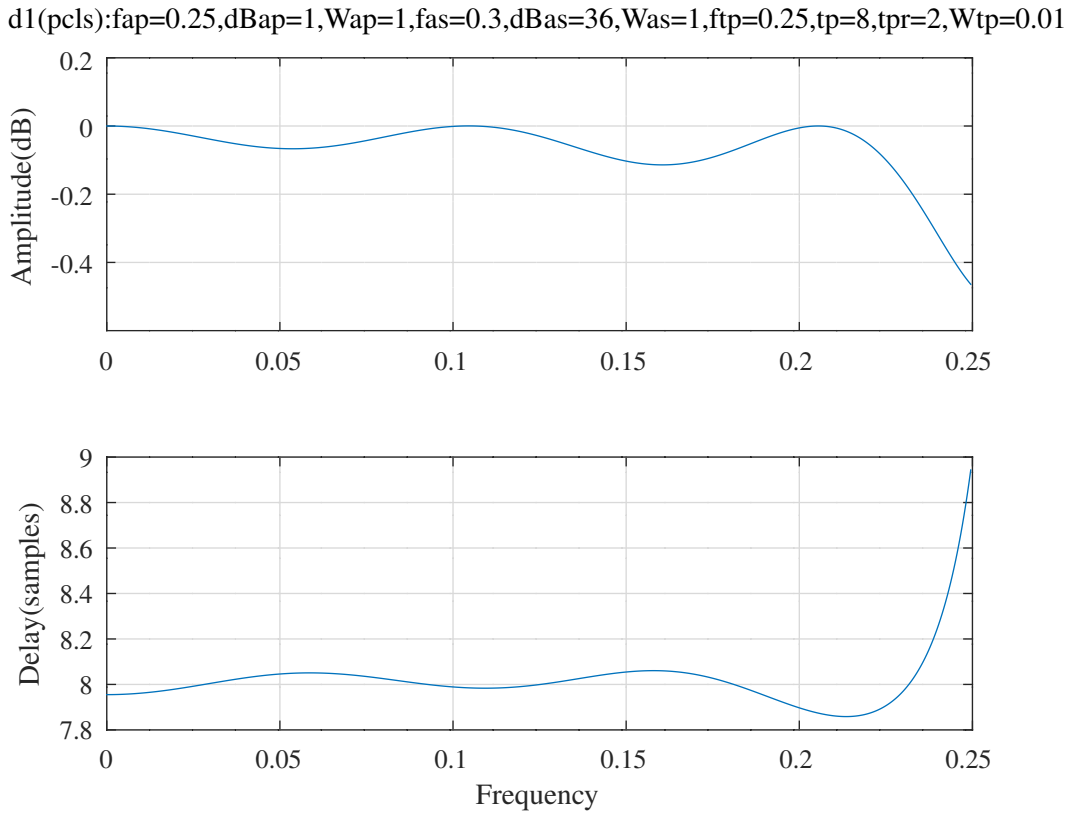



Figure 8.23: Deczky Example 1 PCLS optimised passband response

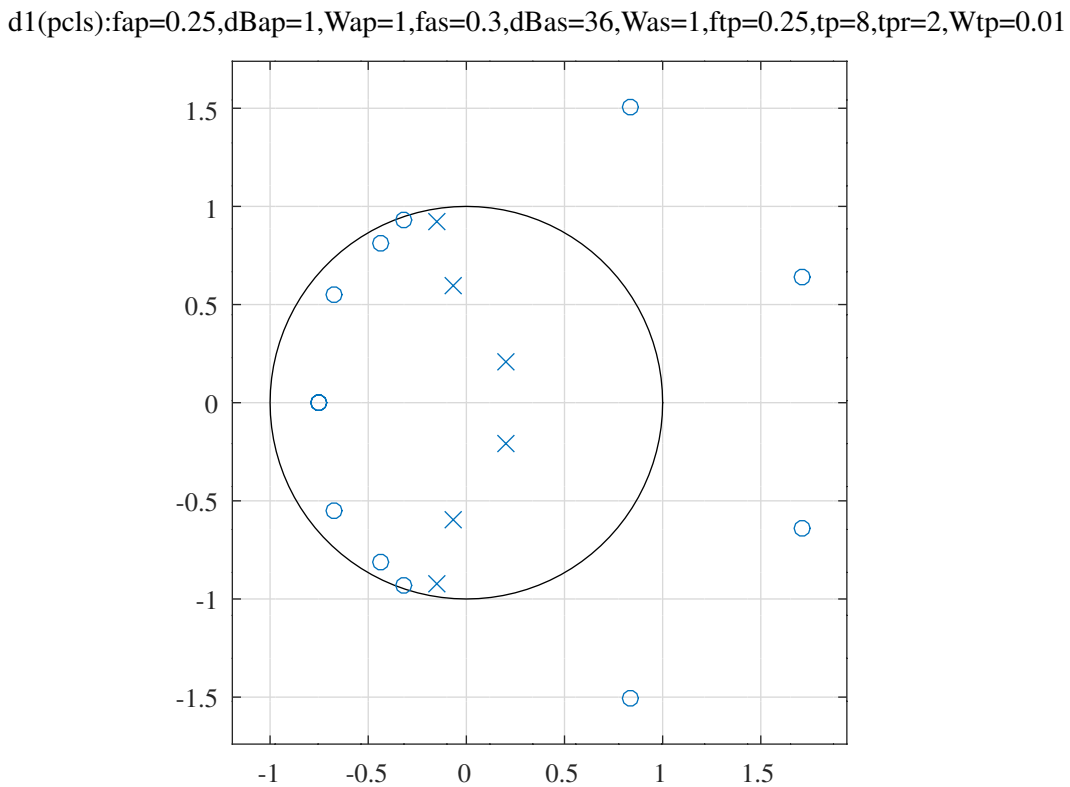



Figure 8.24: Deczky Example 1 PCLS optimised pole-zero plot

### 8.2.5 Low-pass R=2 decimation filter

The second example design is a low-pass $R = 2$ decimation filter with compensation for the zero-order hold amplitude response:

$$
\begin{aligned}
U &= 0 \\
V &= 0 \\
M &= 10 \\
Q &= 6 \\
R &= 2 \\
A(f) &= \begin{cases} \frac{\pi f}{\sin \pi f} & 0 < f < 0.10 \\ 0 & 0.25 < f < 0.5 \end{cases} \\
T(f) &= 8 \quad 0 < f < 0.125
\end{aligned}
$$

The Octave script *decimator_R2_test.m* implements this example. 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=2 % Denominator polynomial decimation factor
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=0.0004 % Tolerance on PCLS relative coef. update
ctol=1e-06 % Tolerance on constraints
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=8 % Nominal filter group delay
tpr=0.008 % Pass band group delay peak-to-peak ripple
Wtp=0.25 % Pass band group delay weight
fas=0.25 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=3 % Stop band amplitude weight
```

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=[ 0.001, ...
     [1,1,1,1,1], ...
     (7:11)*pi/12, ...
     0.7*[1,1,1], ...
     (1:3)*pi/8]';
```

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. In the above listing, the first line shows the gain, and subsequent lines show, respectively, 0 real zeros, 0 real poles, the zero radiuses for 5 conjugate pairs of zeros, the zero angles for 5 conjugate pairs of zeros, the pole radiuses for 3 conjugate quadruples of poles and the pole angles for 3 conjugate quadruples of poles. The resulting initial response is shown in Figure 8.25.

#### MMSE optimisation of the low-pass R=2 decimator

After some experimentation and iteration, weights $Wap = 1$, $Was = 4$ and $Wtp = 0.25$ give the response shown in Figure 8.26 with the pass-band detail shown in Figure 8.27. The corresponding pole-zero plot is shown in Figure 8.28.

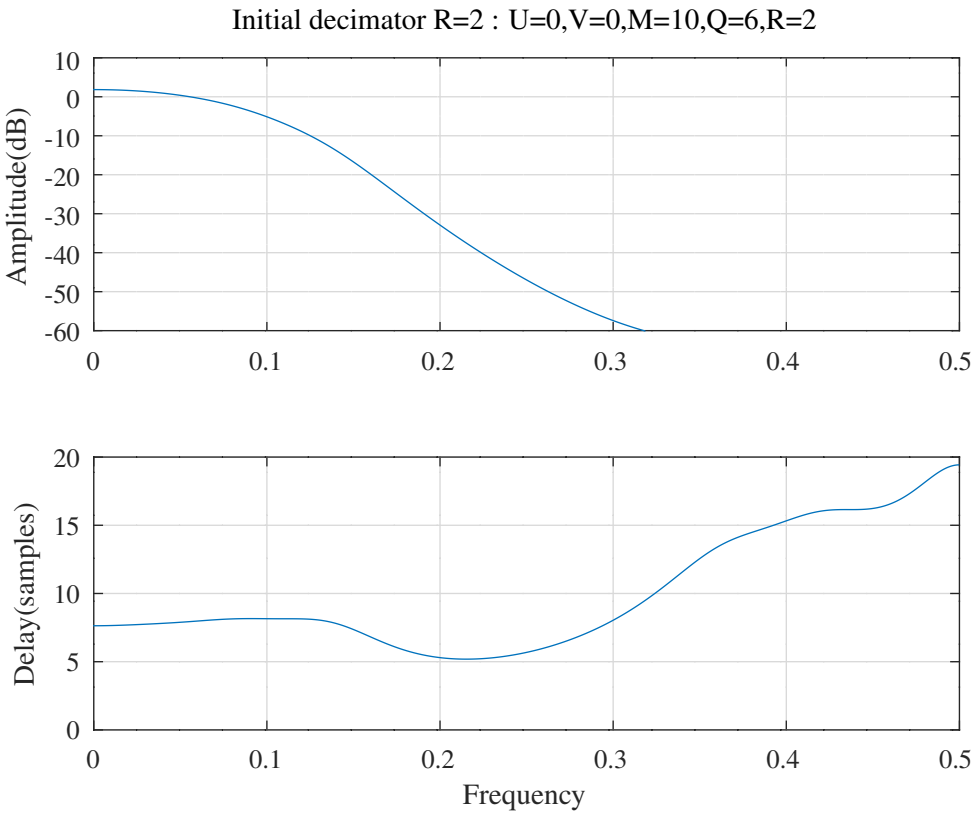



Figure 8.25: Low-pass decimator R=2:Response for initial coefficients

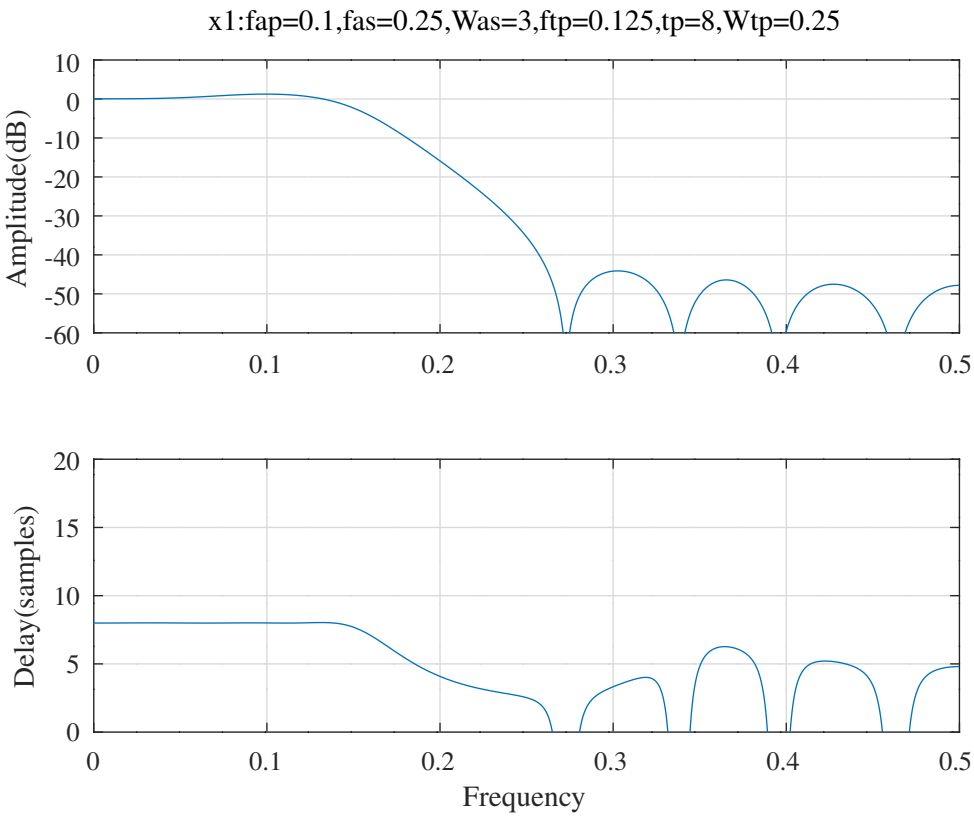



Figure 8.26: Low-pass decimator R=2 MMSE optimised response

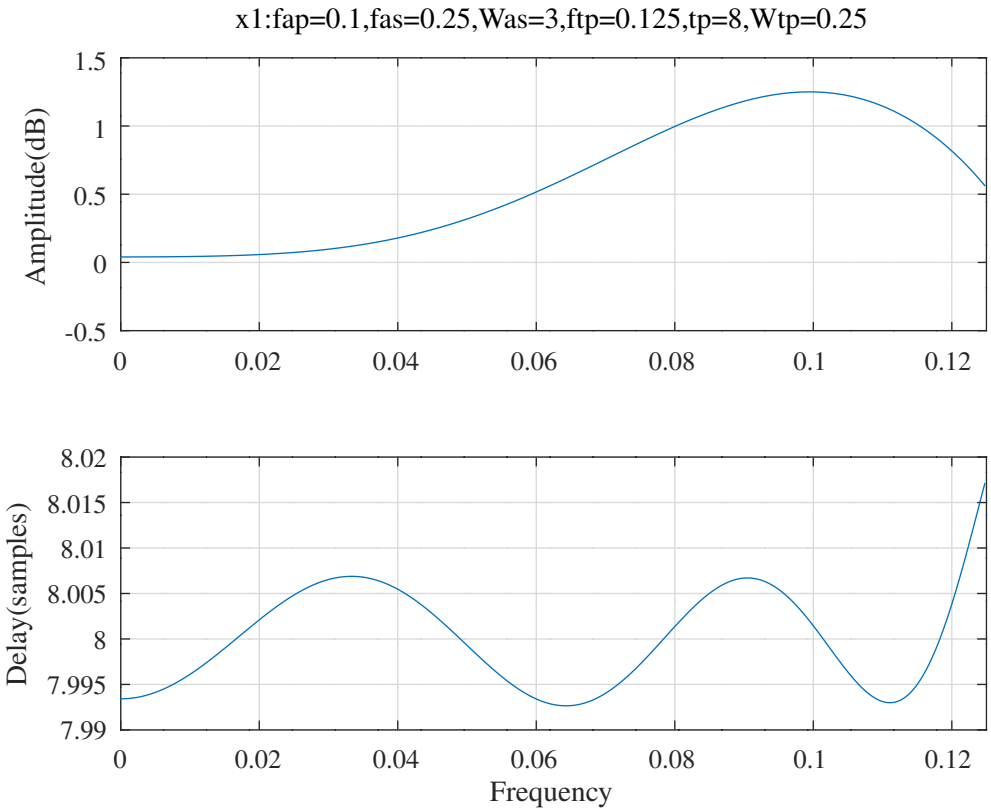



Figure 8.27: Low-pass decimator R=2 MMSE optimised passband response

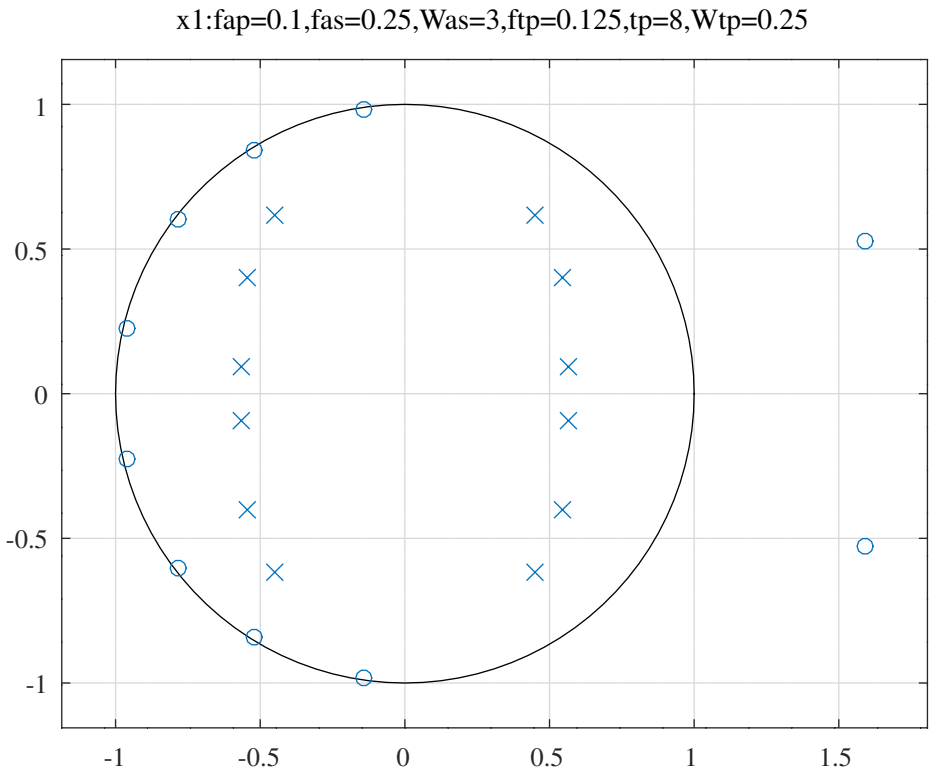



Figure 8.28: Low-pass decimator R=2 MMSE optimised pole-zero plot

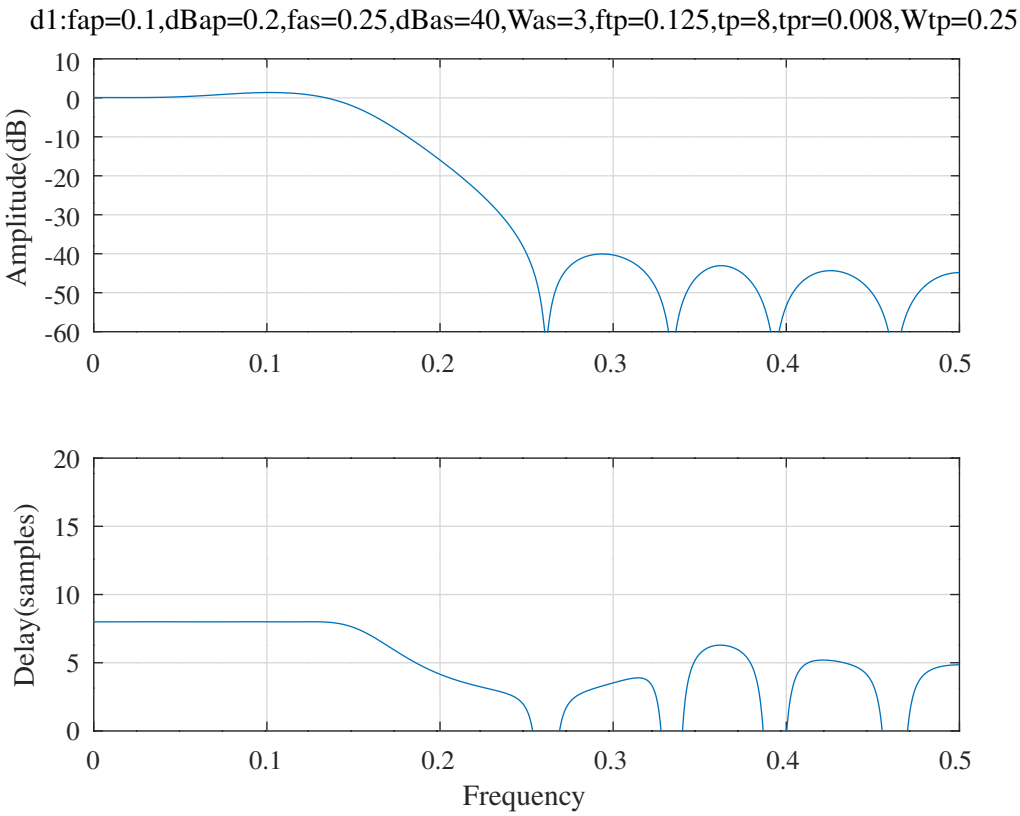



Figure 8.29: Low-pass decimator R=2 PCLS optimised response

**PCLS optimisation of the low-pass R=2 decimator**

Starting with the MMSE filter of Figure 8.26, the weights $Wap = 1$, $Was = 3$ and $Wtp = 0.25$ and the constraints $dBap = 0.2$, $dBas = 40$, $tp = 8$ and $tpr = 0.008$ give the response shown in Figure 8.29 with pass-band detail shown in Figure 8.30. The corresponding pole-zero plot is shown in Figure 8.31. The estimate of the optimised filter vector is, in the gain, zeros and poles form of Equation 8.2:

```
Ud1=0,Vd1=0,Md1=10,Qd1=6,Rd1=2
d1 = [   0.0160456528, ...
         0.9876270483,   0.9883670285,   0.9908605837,   0.9926785794, ...
         1.6353372851, ...
         2.4715642627,   2.9071695652,   2.0990333511,   1.6418081781, ...
         0.3255108385, ...
         0.2897932913,   0.4476466285,   0.5686156022, ...
         0.1249186305,   1.2775236376,   1.8743905024 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0160456528,   0.0242551209,  -0.0099621816,  -0.0688503637, ...
        -0.0736937769,   0.0253849891,   0.1796988010,   0.2782245719, ...
         0.2526368868,   0.1407064512,   0.0395582365 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,  -0.4939102543,   0.0000000000, ...
         0.4730265221,   0.0000000000,  -0.2593078670,   0.0000000000, ...
         0.1103267545,   0.0000000000,  -0.0385650984,   0.0000000000, ...
         0.0054410765 ]';
```

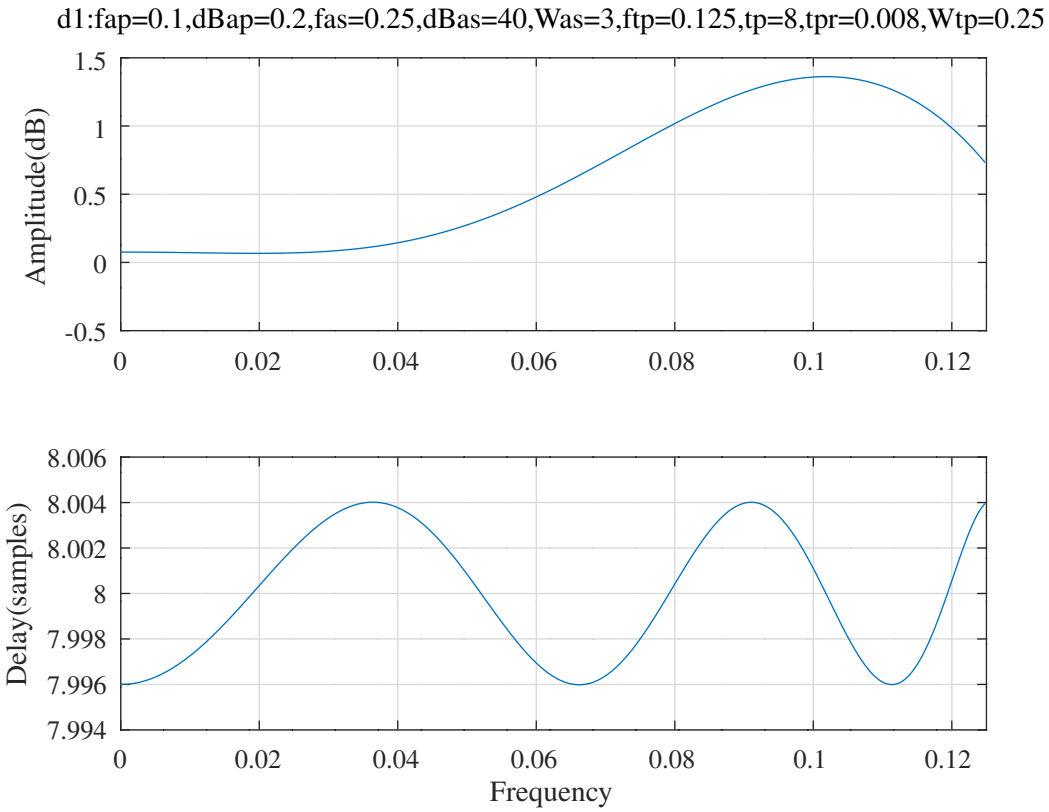



Figure 8.30: Low-pass decimator R=2 PCLS optimised passband response

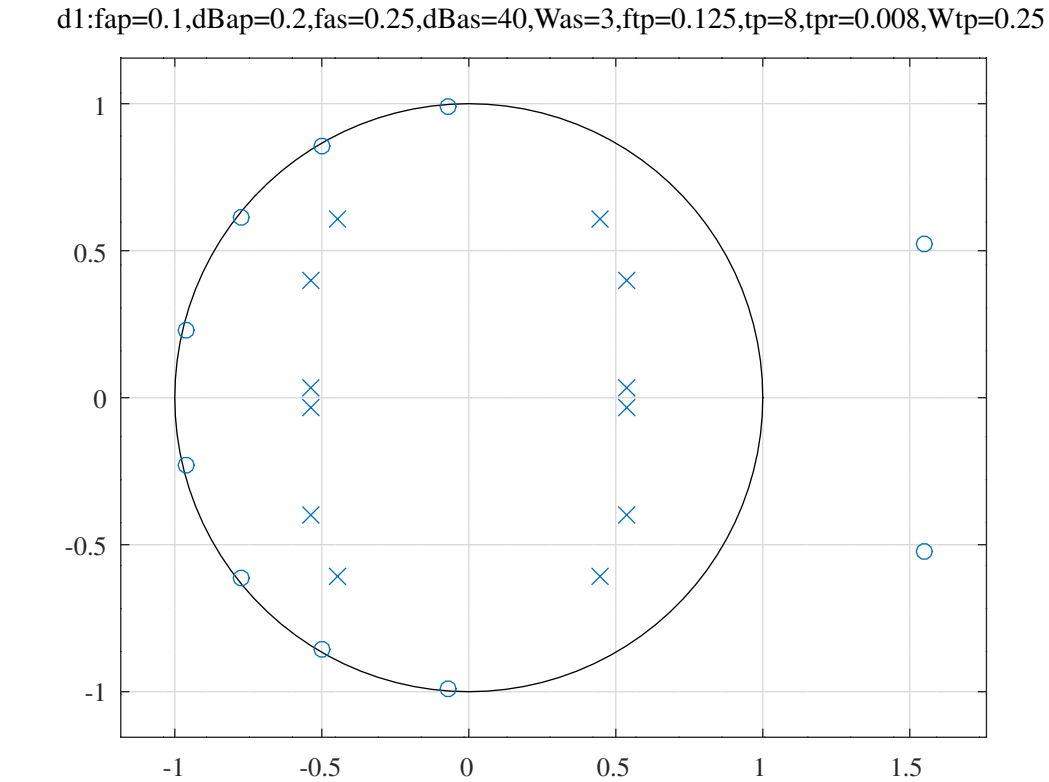



Figure 8.31: Low-pass decimator R=2 PCLS optimised pole-zero plot

**An alternative implementation of the low-pass R=2 decimator**

The Octave script *decimator_R2_alternate_test.m* designs an alternative implementation of an $R = 2$ decimation low-pass filter with a flat pass-band amplitude response. The alternative filter specification is:

```
U=0 % Number of real zeros
V=0 % Number of real poles
M=12 % Number of complex zeros
Q=6 % Number of complex poles
R=2 % Denominator polynomial decimation factor
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=0.0002 % Tolerance on PCLS relative coef. update
ctol=1e-07 % Tolerance on constraints
fap=0.1 % Pass band amplitude response edge
dBap=0.35 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.125 % Pass band group delay response edge
tp=10 % Nominal filter group delay
tpr=0.02 % Pass band group delay peak-to-peak ripple
Wtp=1 % Pass band group delay weight
fas=0.25 % Stop band amplitude response edge
dBas=50 % Stop band minimum attenuation
Was=1.25 % Stop band amplitude weight
```

Figure 8.32 shows the response of the resulting filter. The corresponding pole-zero plot is shown in Figure 8.33.

The gain, zeros and poles of the optimised filter are:

```
Ud1=0,Vd1=0,Md1=12,Qd1=6,Rd1=2
d1 = [   0.0005443183, ...
         1.0068133386,   1.0493374153,   1.0607618300,   1.0886216898, ...
         1.6621311854,   3.3908786751, ...
         1.6276936863,   2.0310362428,   2.4262174162,   2.8918881576, ...
         0.3195020905,   3.1362760228, ...
         0.3293172112,   0.4472305721,   0.5706669338, ...
         0.2719547623,   1.2108468518,   1.7847720136 ]';
```

The numerator and denominator polynomials of the PCLS optimised filter are:

```
N1 = [   0.0005443183,   0.0045632593,   0.0119218709,   0.0058623309, ...
        -0.0226466042,  -0.0484568963,  -0.0250172007,   0.0617081177, ...
         0.1641621325,   0.2113513446,   0.1751289594,   0.0929134217, ...
         0.0257351082 ]';


D1 = [   1.0000000000,   0.0000000000,  -0.7071230319,   0.0000000000, ...
         0.6038896840,   0.0000000000,  -0.3470702073,   0.0000000000, ...
         0.1482042985,   0.0000000000,  -0.0471946641,   0.0000000000, ...
         0.0070641066 ]';
```

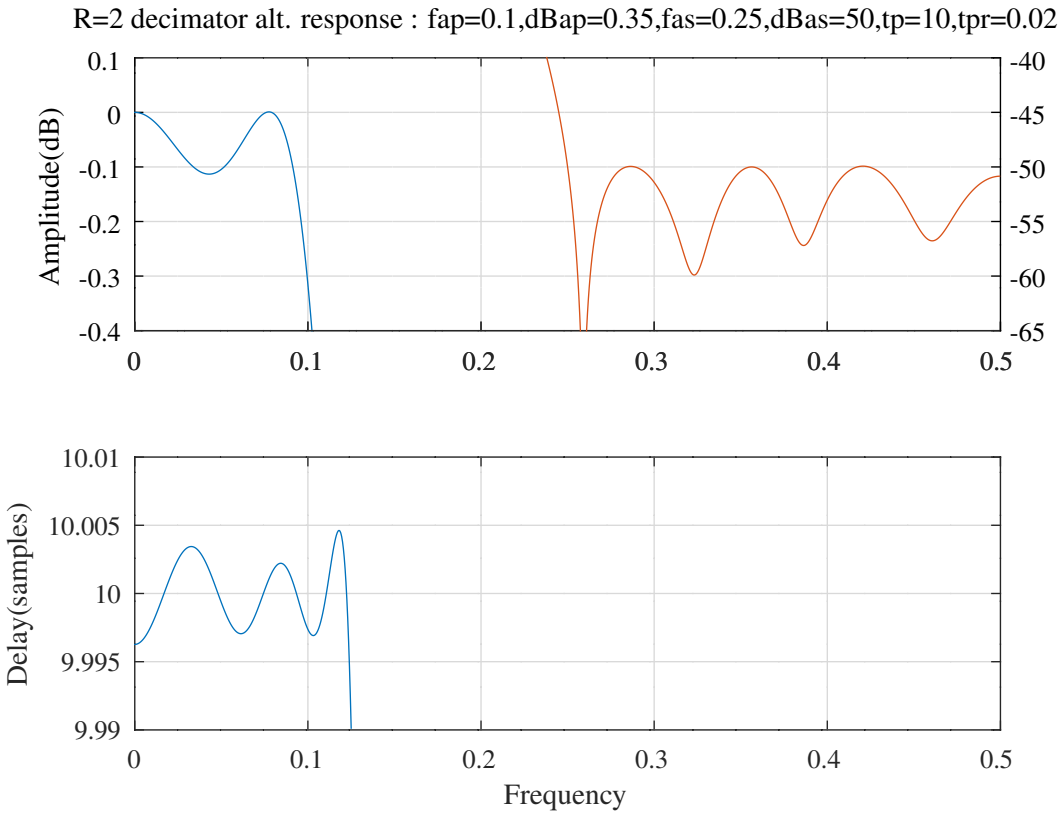



Figure 8.32: Alternative low-pass decimator R=2 PCLS optimised response

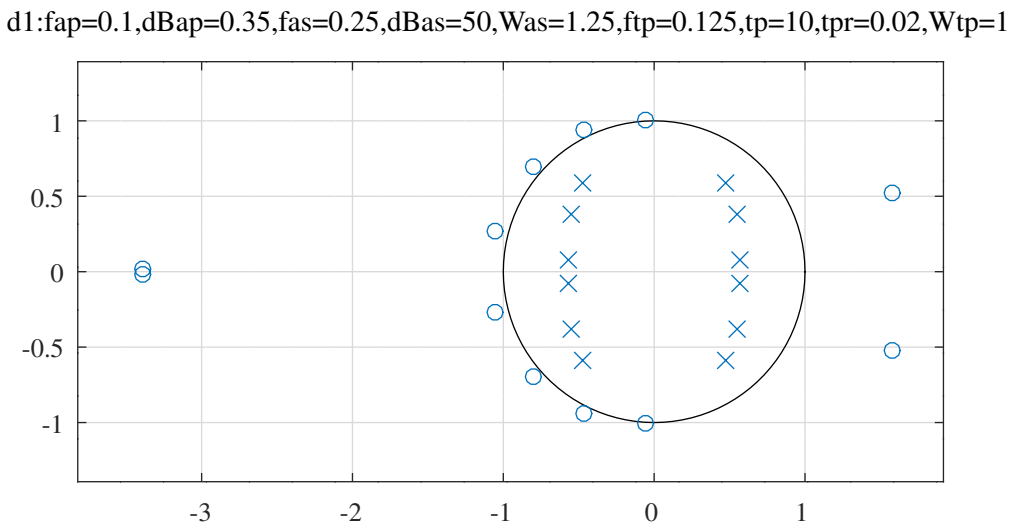



Figure 8.33: Alternative low-pass decimator R=2 PCLS optimised pole-zero plot

**An interpolated low-pass R=2 decimator**

*Lyons* [177] describes the design of interpolated low-pass FIR filters. The Octave script *decimator_R2_interpolated_test.m* first designs an interpolated $R = 2$ IIR low-pass filter. The un-interpolated filter specification is:

```
P=3 % Interpolation factor
U=0 % Number of real zeros
V=0 % Number of real poles
M=12 % Number of complex zeros
Q=6 % Number of complex poles
R=2 % Denominator polynomial decimation factor
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
ctol=1e-05 % Tolerance on constraints
fap=0.09 % Pass band amplitude response edge
dBap=0.25 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.12 % Pass band group delay response edge
tp=10 % Nominal filter group delay
tpr=0.2 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
fas=0.18 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=10 % Stop band amplitude weight
```

The gain, zeros and poles of the optimised filter are:

```
Ud1=0,Vd1=0,Md1=12,Qd1=6,Rd1=2
d1 = [   0.0127982334, ...
         0.9408972367,   0.9760750897,   0.9797669922,   0.9800474487, ...
         0.9897057355,   1.5100713950, ...
         1.5971047835,   2.9344277459,   2.5501370034,   2.2232754680, ...
         1.1715733543,   0.2814683759, ...
         0.4949040911,   0.5993584343,   0.7055873476, ...
         0.3808663872,   1.1669938395,   1.7213905846 ]';
```

The numerator and denominator polynomials of the PCLS optimised un-interpolated filter are:

```
N1 = [   0.0127982334,   0.0141544347,  -0.0093698724,  -0.0311157812, ...
        -0.0285109379,  -0.0084527799,   0.0294315478,   0.0814767890, ...
         0.1189739976,   0.1243687058,   0.1045039039,   0.0652884707, ...
         0.0222304278 ];

D1 = [   1.0000000000,   0.0000000000,  -1.1781662100,   0.0000000000, ...
         1.2405500857,   0.0000000000,  -0.9178721231,   0.0000000000, ...
         0.5099284874,   0.0000000000,  -0.2031418559,   0.0000000000, ...
         0.0438043156 ];
```

Figure 8.34 shows the amplitude and delay responses of the prototype IIR filter.

Next, an anti-aliasing filter is designed with the *remez* function from the Octave-Forge *signal* package using the actual pass-band edge, $0.093$ , stop-band edge, $0.179$ , and interpolation factor, $P$. The coefficients of the FIR anti-aliasing filter are:

```
b = [  -0.0123510030,  -0.0226223862,   0.0106253797,   0.1158827378, ...
        0.2483715491,   0.3099785029,   0.2483715491,   0.1158827378, ...
        0.0106253797,  -0.0226223862,  -0.0123510030 ];
```

The interpolated, anti-aliased IIR filter has a nominal delay of $35$ samples and $25$ multipliers. Figure 8.35 shows the responses of the IIR filter interpolated by the factor, $P$, and the FIR anti-aliasing filter. Figure 8.36 shows the amplitude response of the interpolated and anti-aliased IIR filter. Figure 8.37 shows the pass-band amplitude and group delay responses of the interpolated and anti-aliased IIR filter. Figure 8.38 compares the amplitude response of the interpolated and anti-aliased IIR filter with that of three equi-ripple FIR filters designed with the *remez* function from the Octave-Forge *signal* package: firstly an interpolated and anti-aliased FIR filter having the same frequency specifications as the IIR filter with a delay of $35$ samples and $17$ distinct mulipliers; secondly, a direct-form FIR filter with $25$ distinct multipliers; thirdly, a direct form FIR filter with $35$ samples delay.

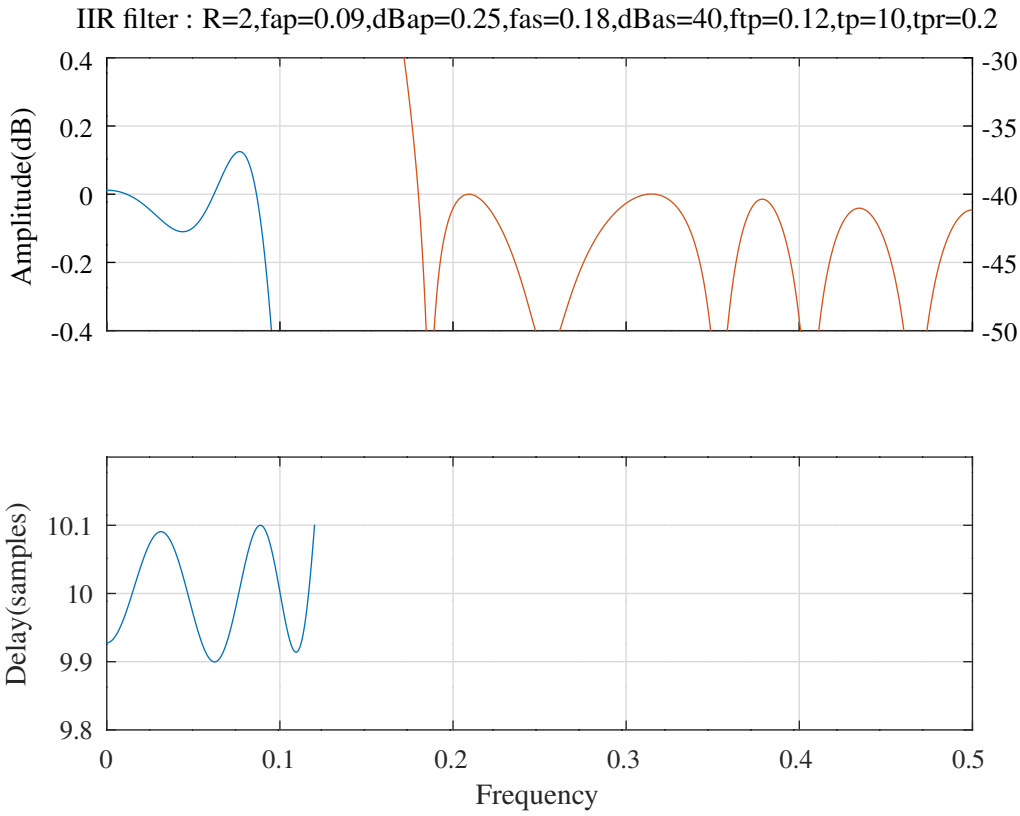



Figure 8.34: Amplitude and delay responses of PCLS optimised prototype R=2 IIR filter

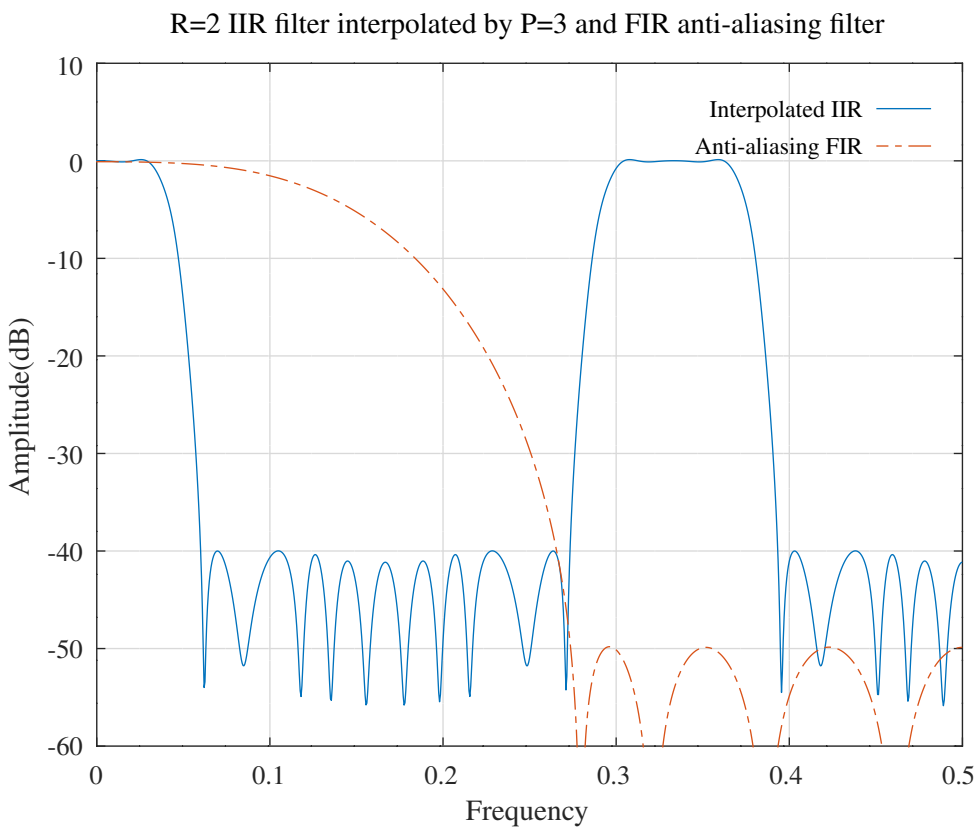



Figure 8.35: Responses of an interpolated low-pass PCLS optimised R=2 IIR filter and FIR anti-aliasing filter.

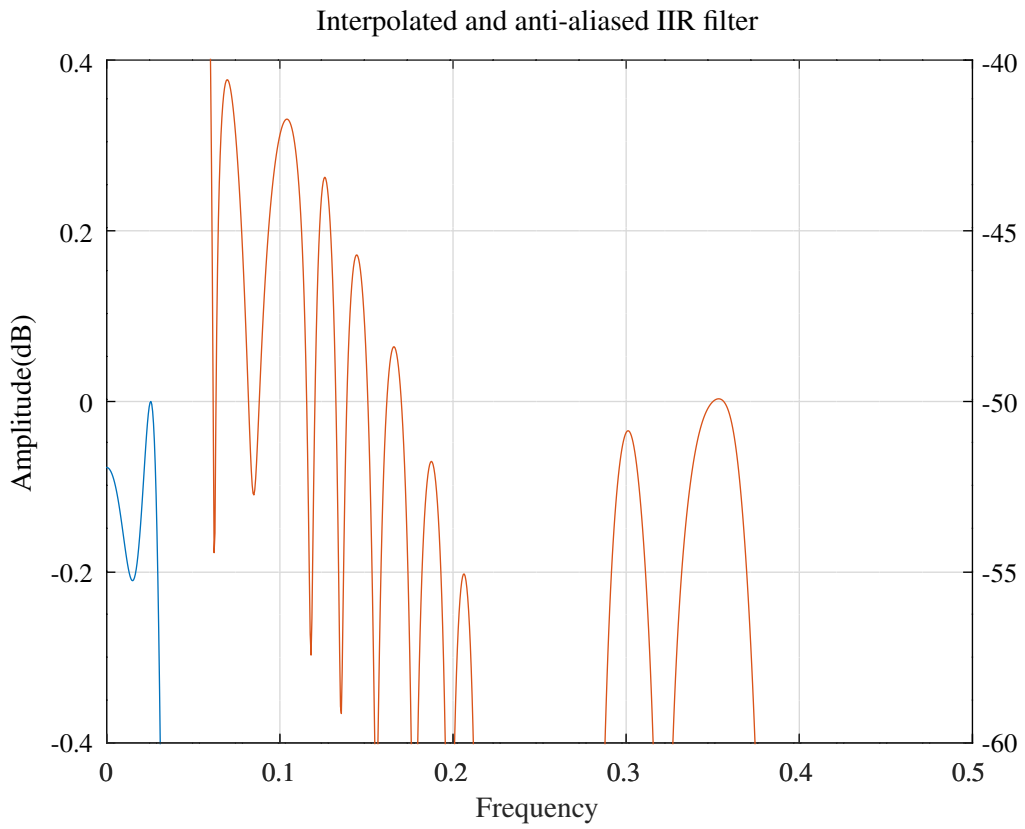



Figure 8.36: Amplitude response of an interpolated anti-aliased low-pass IIR filter.

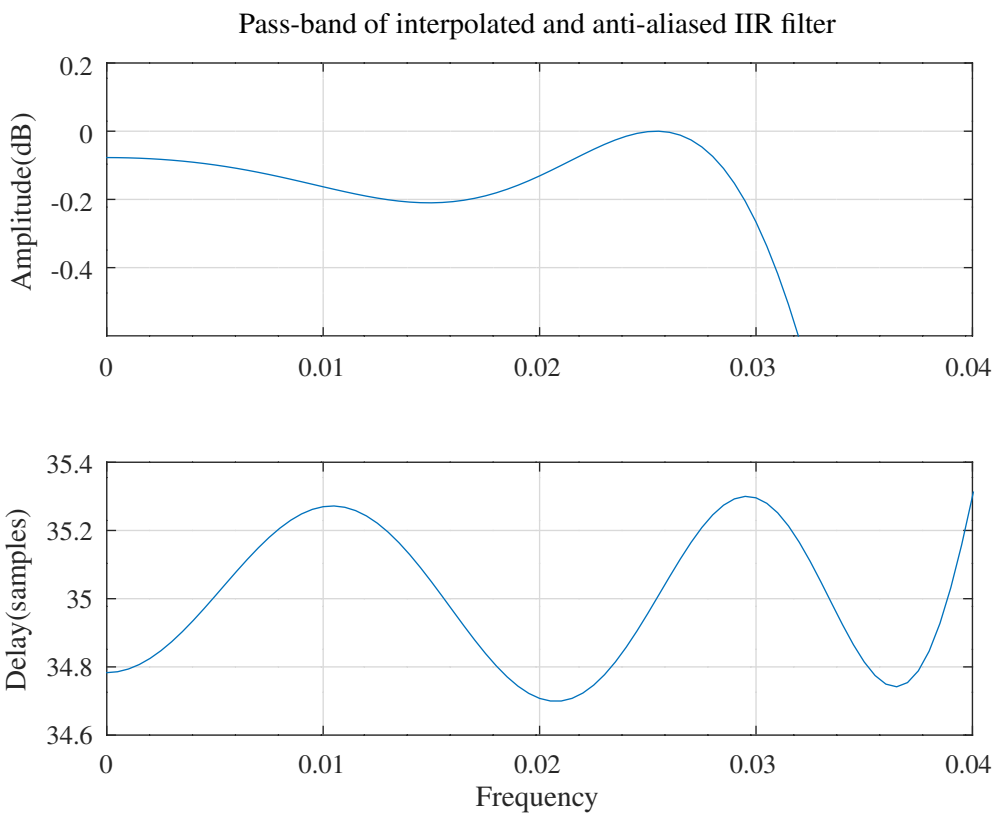
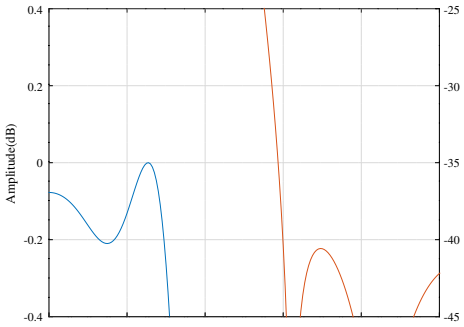
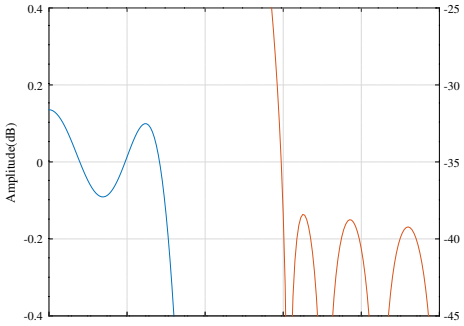


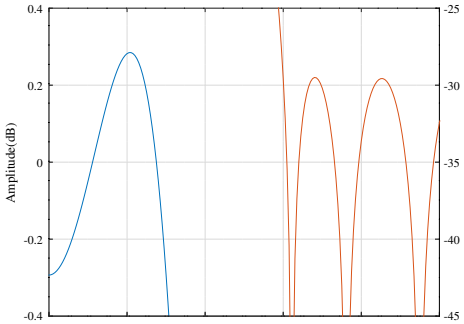

Figure 8.37: Pass-band amplitude and group delay responses of an interpolated anti-aliased low-pass IIR filter.



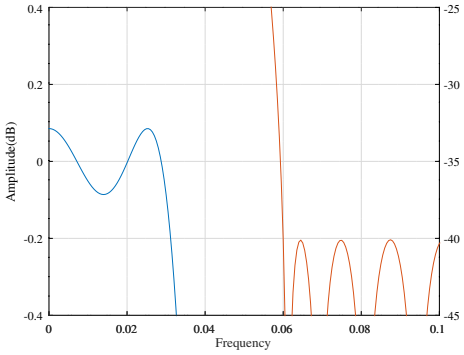

(a) Response of an interpolated IIR filter with 25 multipliers and 34 samples nominal delay




(b) Response of an equivalent interpolated anti-aliased symmetric FIR filter with 17 distinct multipliers and 35 samples delay




(c) Response of a symmetric direct-form FIR filter with 25 distinct multipliers




(d) Response of a symmetric direct-form FIR filter with 35 samples nominal delay

Figure 8.38: Comparison of the amplitude response of the interpolated and anti-aliased IIR filter with that of three FIR filters designed with the *remez* function : firstly an interpolated and anti-aliased FIR filter having the same frequency specifications as the IIR filter; secondly, a direct-form FIR filter having the same number of distinct multipliers; thirdly, a direct form FIR filter having the same nominal delay.

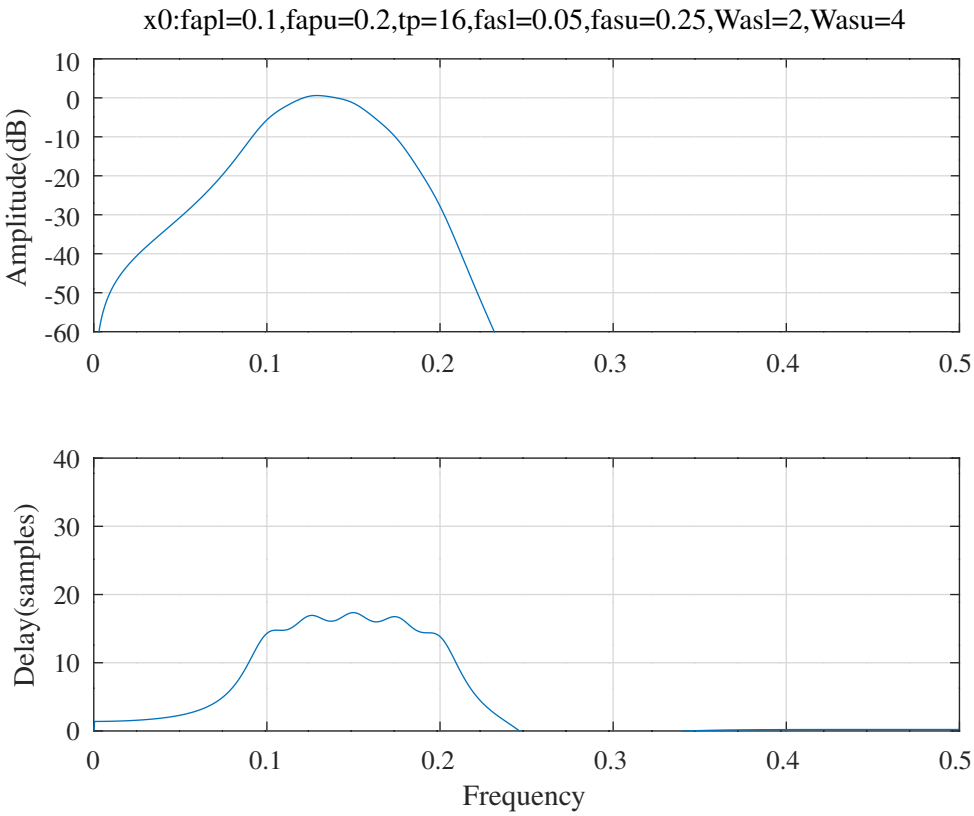



Figure 8.39: IIR band-pass R=2 decimation filter : response of the initial filter

### 8.2.6 Band-pass R=2 decimation filter

This 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 radiuses for 9 conjugate pairs of zeros, the zero angles for 9 conjugate pairs of zeros, the pole radiuses 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 8.39.

#### 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 $Wasu = 4$, respectively, and group-delay response pass-band weight $Wtp = 1$. The response of the MMSE-optimised filter is shown in Figure 8.40 with pass-band detail shown in Figure 8.41. The corresponding pole-zero plot is shown in Figure 8.42.

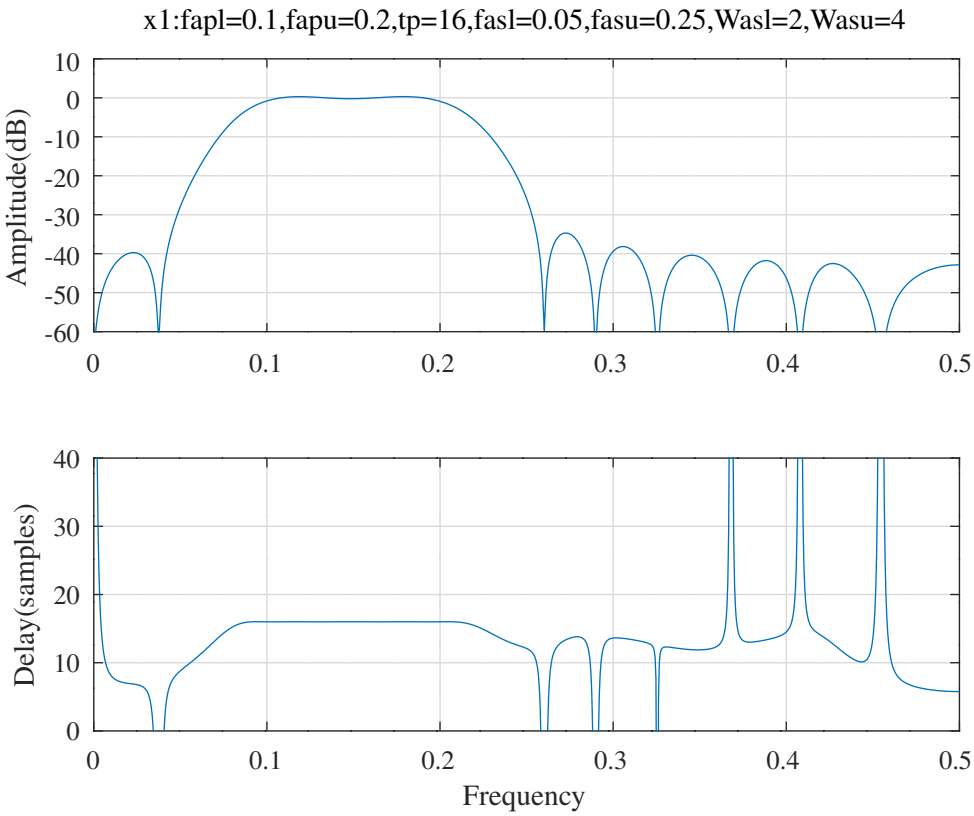



Figure 8.40: IIR band-pass R=2 decimation filter with delay tp=16 samples : response of the filter after MMSE optimisation

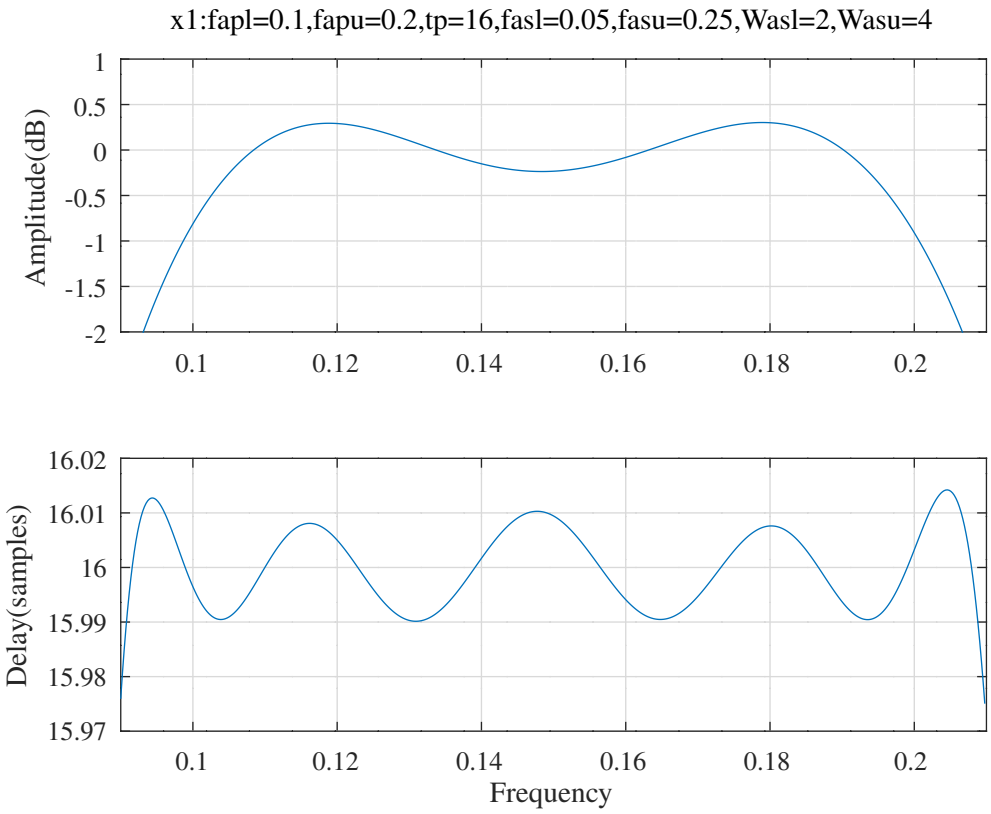



Figure 8.41: IIR band-pass R=2 decimation filter with delay tp=16 samples : passband response of the filter after MMSE optimisation

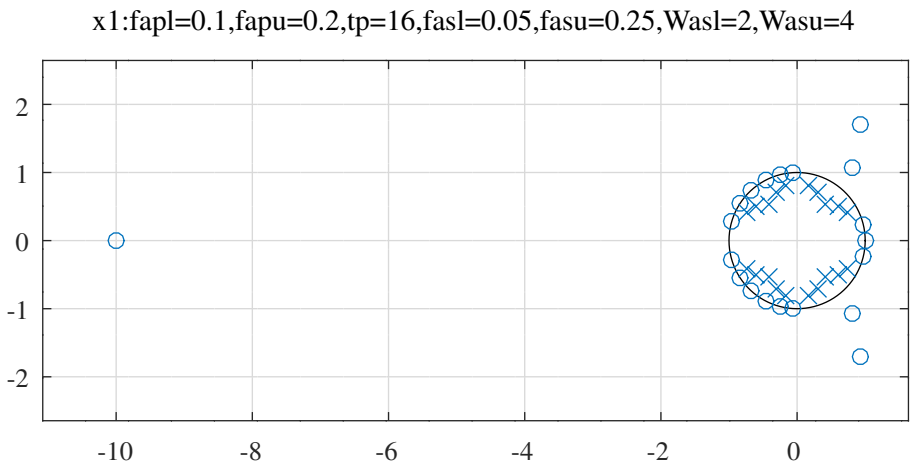



Figure 8.42: IIR band-pass R=2 decimation filter with delay tp=16 samples:pole-zero plot of the filter after MMSE optimisation

**PCLS optimisation of the band-pass R=2 decimator**

Next, the filter designed in Section 8.2.6 is passed through *iir_slb* for PCLS optimisation with $Wtp = 1$, $tpr = 0.08$ samples, $dBap = 1$ and $dBas = 35$. The response of the resulting filter is shown in Figure 8.43 with pass-band detail shown in Figure 8.44. The corresponding pole-zero plot is shown in Figure 8.45. The estimate of the optimised filter vector is, in the gain, zeros and poles form of Equation 8.2:

```
Ud1=2,Vd1=0,Md1=18,Qd1=10,Rd1=2
d1 = [   0.0119781334, ...
        -0.9501854759,   0.9700299850, ...
         0.9804478987,   0.9805466967,   0.9820770988,   0.9824927194, ...
         0.9838266649,   0.9929262261,   0.9947340877,   1.3537452074, ...
         1.4193911791, ...
         2.6756962667,   2.1929278437,   2.4486798999,   1.9388563897, ...
         1.7375703254,   0.2813703050,   1.5933065259,   0.8426117409, ...
         1.0914178045, ...
         0.5675646316,   0.6204192829,   0.6463715109,   0.7112138524, ...
         0.7464164827, ...
         1.9315692456,   2.4222482137,   1.3926168065,   2.7966923629, ...
         1.0540496576 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0119781334,   0.0053372581,   0.0225953299,   0.0226057078, ...
         0.0476583039,   0.0341407987,   0.0294625008,   0.0000430755, ...
        -0.0024678492,  -0.0310804121,  -0.0679675076,  -0.1023386019, ...
        -0.0694596255,   0.0369652616,   0.1362499644,   0.1556848981, ...
         0.0622425193,  -0.0405480060,  -0.0988351108,  -0.0710911985, ...
        -0.0331193158 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,   1.7060993914,   0.0000000000, ...
         1.9330569490,   0.0000000000,   1.9381867906,   0.0000000000, ...
         1.7150005810,   0.0000000000,   1.2590350700,   0.0000000000, ...
         0.8059003363,   0.0000000000,   0.4345691412,   0.0000000000, ...
         0.1970293344,   0.0000000000,   0.0648657750,   0.0000000000, ...
         0.0145991810 ]';
```

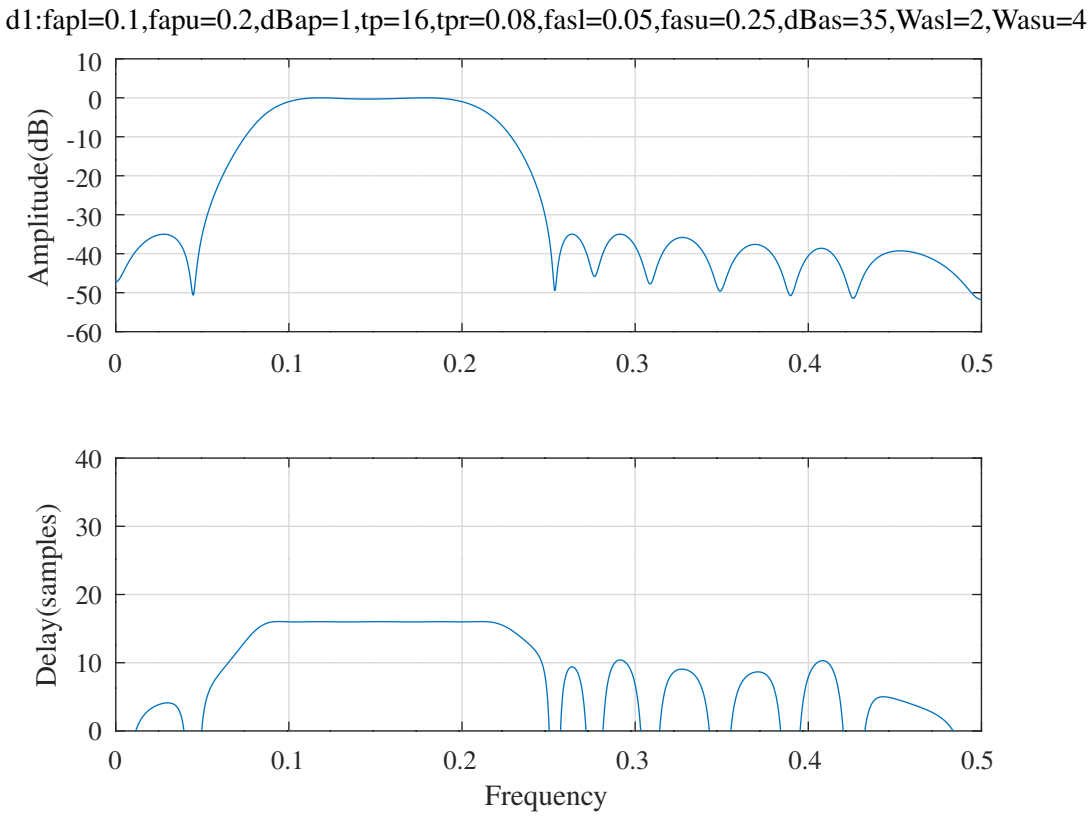



Figure 8.43: IIR band-pass R=2 decimation filter with delay tp=16 samples : response of the filter after PCLS optimisation

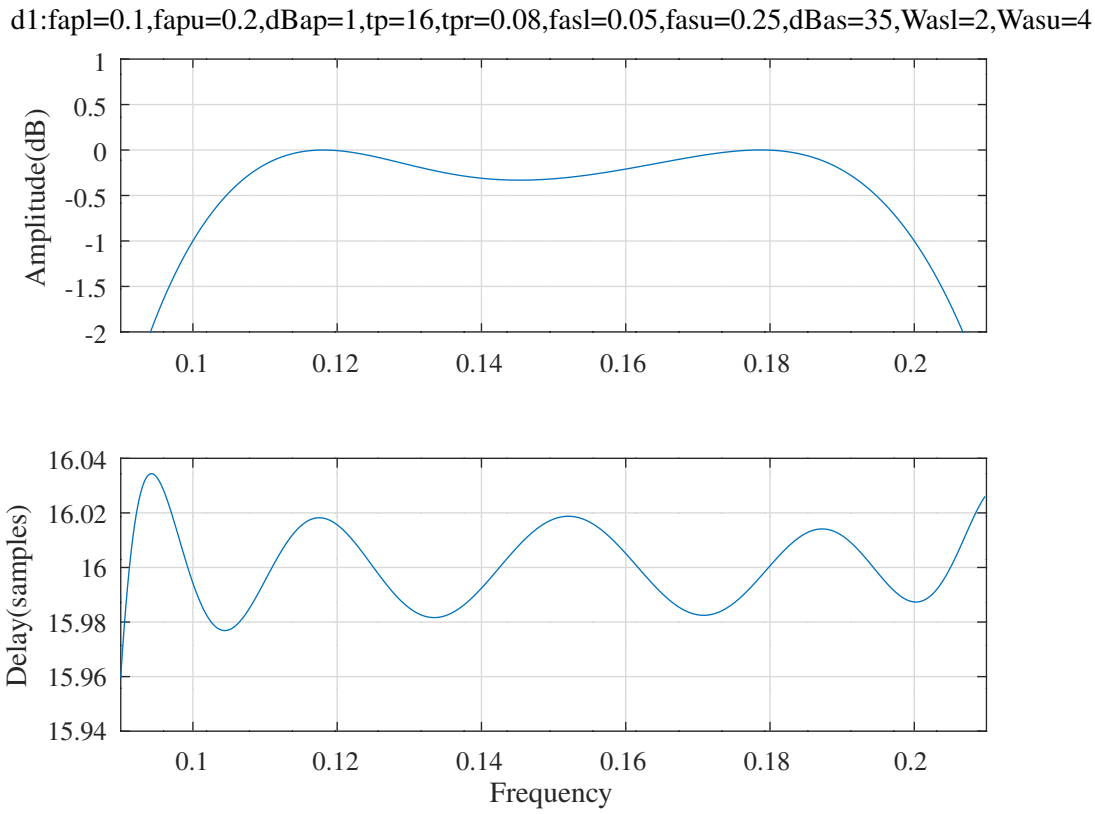



Figure 8.44: IIR band-pass R=2 decimation filter with delay tp=16 samples : passband response of the filter after PCLS optimisation

d1:fapl=0.1,fapu=0.2,dBap=1,tp=16,tpr=0.08,fasl=0.05,fasu=0.25,dBas=35,Wasl=2,Wasu=4

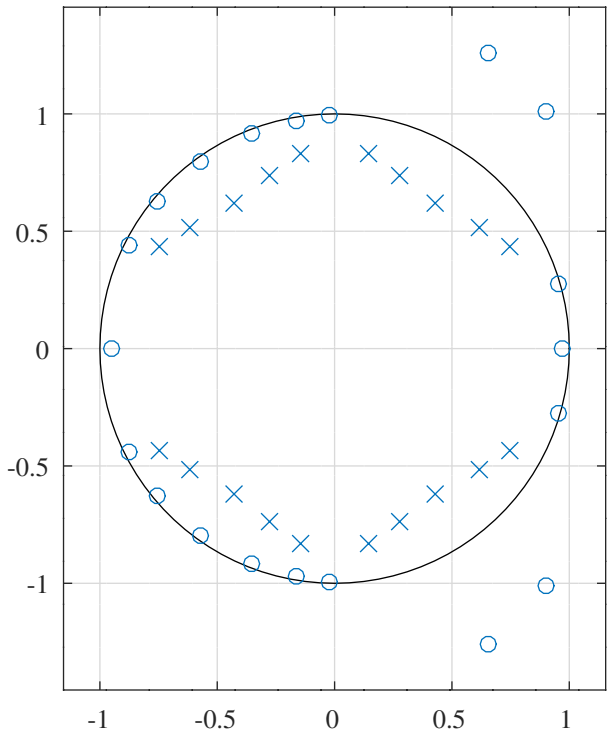



Figure 8.45: IIR band-pass R=2 decimation filter with delay tp=16 samples:pole-zero plot of the filter after PCLS optimisation

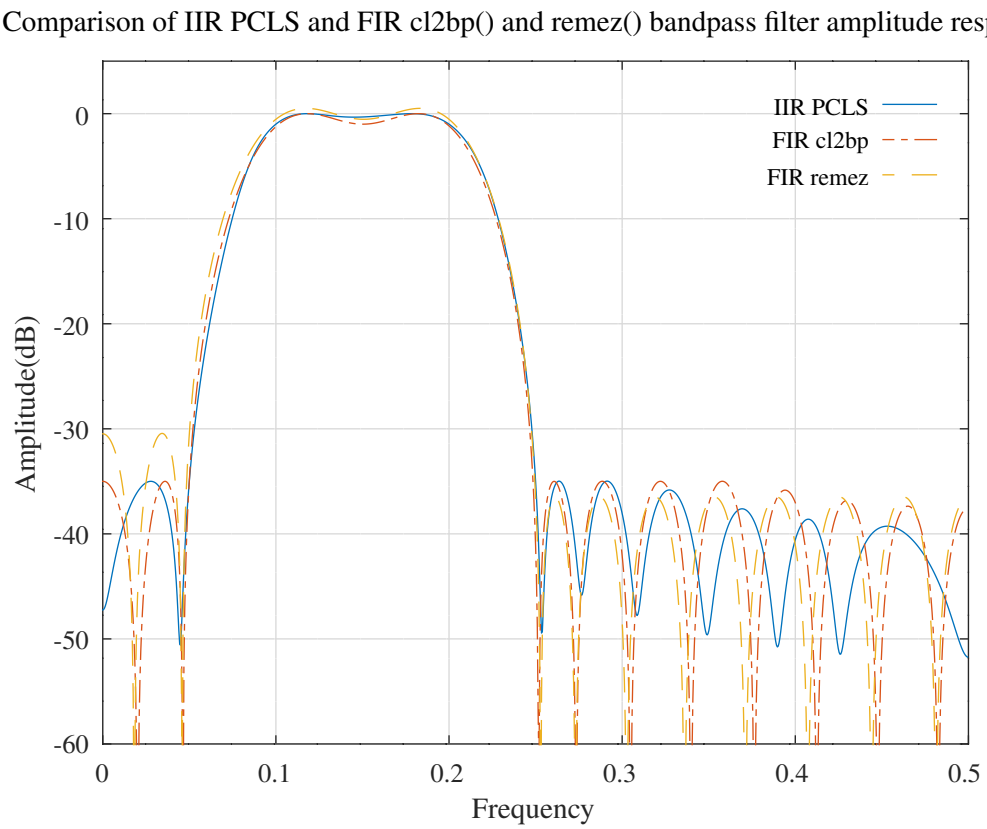



Figure 8.46: Comparison of the magnitude responses of the bandpass PCLS IIR filter and FIR filters designed by *cl2bp* and *remez*

**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 8.46.

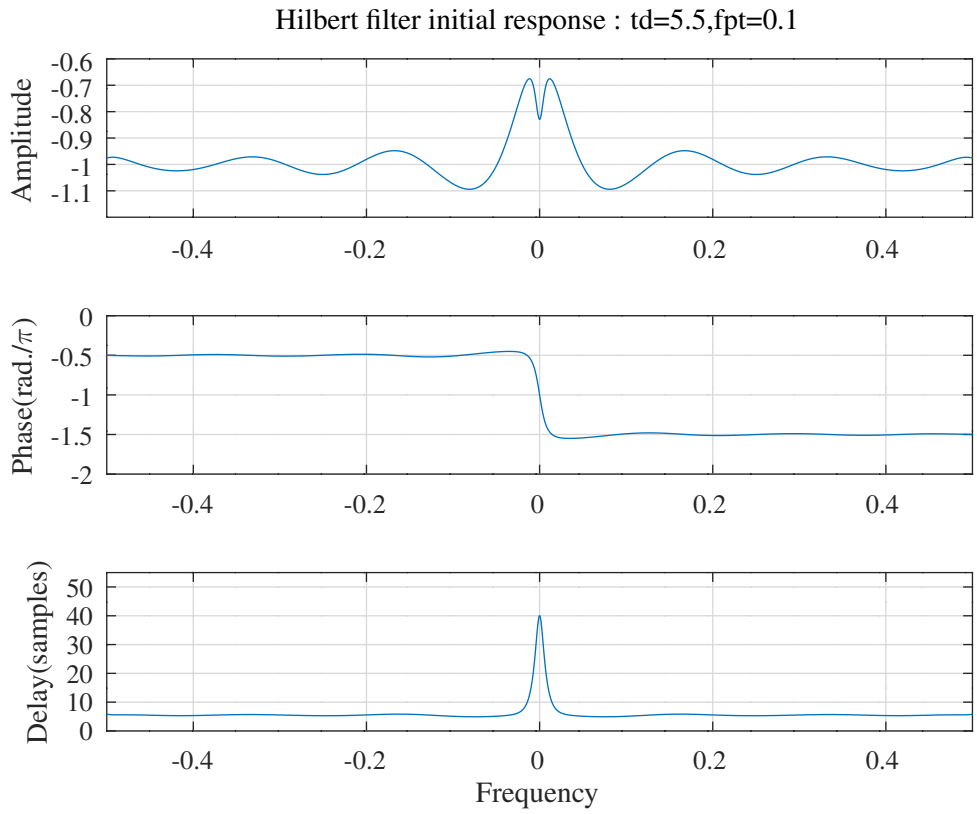



Figure 8.47: Response of the initial Hilbert filter designed by *tarczynski_hilbert_test.m* with $R = 2$ and $t_d = 5.5$. The phase response shown is adjusted for the nominal delay.

### 8.2.7 Hilbert transform R=2 decimation filter

A Hilbert transform filter[12, p.118] has the transfer function

$$H_d(\omega) = -\imath sign(\omega)\, e^{-\imath\omega t_d}$$

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 = [  -0.0579058120, ...
        -1.8392427728,  -0.9615144922,  -0.6828634516,  -0.1324634860, ...
         0.4144388376,   0.8764715432,   1.0642619866, ...
        -0.2223215228,   0.9249475556, ...
         1.6409692530,   1.7962310331, ...
         1.0002172846,   2.0743531485, ...
         0.2452036668,   0.3860245935, ...
         1.6839869509,   0.1456530004 ]';
```

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 8.47. 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 $f_{pt} = \pm 0.075$. 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 $\pi$, is shown adjusted for the group-delay by adding $\omega t_d$ where $\omega$ is the angular frequency vector for which the response is calculated.

#### MMSE optimisation of the Hilbert transform R=2 decimator

Figure 8.48 shows the response after MMSE optimisation of the initial filter. The phase response shown is adjusted for the nominal delay. The relative weights of the amplitude, group delay and phase were $W_{ap} = 1$, $W_{tp} = 0.0001$ and $W_{pp} =$

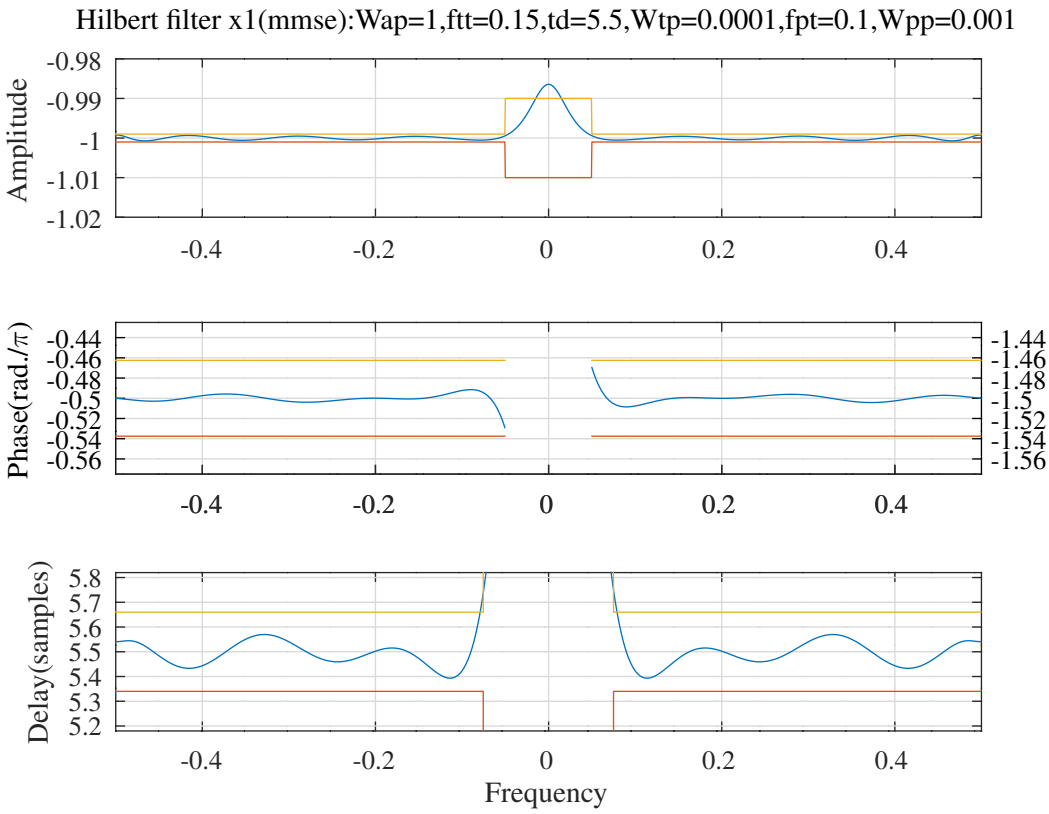



Figure 8.48: Response of the Hilbert filter after MMSE optimisation. The phase response shown is adjusted for the nominal delay.

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*.

**PCLS optimisation of the Hilbert transform R=2 decimator**

Figure 8.49 shows the response after PCLS optimisation of the MMSE optimised filter of Figure 8.48. Figure 8.50 shows the pole-zero plot for the PCLS optimised filter. The phase response shown is adjusted for the nominal delay. 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 = [  -0.0105518106, ...
        -2.4416134956,  -0.8241677452,  -0.2388348172,  -0.2105433247, ...
         0.3816574850,   0.3982341321,   1.2113520766, ...
         0.1971514470,   0.6779471602, ...
         2.3111631658,   2.4486225067, ...
         1.0241103092,   2.0425175572, ...
         0.0455698177,   0.1724892687, ...
         0.4574376346,   2.1019084401 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively:

```
N1 = [  -0.0105518106,  -0.0163169434,  -0.0307338630,  -0.0657011421, ...
        -0.1330870088,  -0.5266104343,   0.8094400709,   0.7103751349, ...
        -0.3784675704,  -0.1093789070,   0.0284822475,   0.0062957311 ]';
```

and

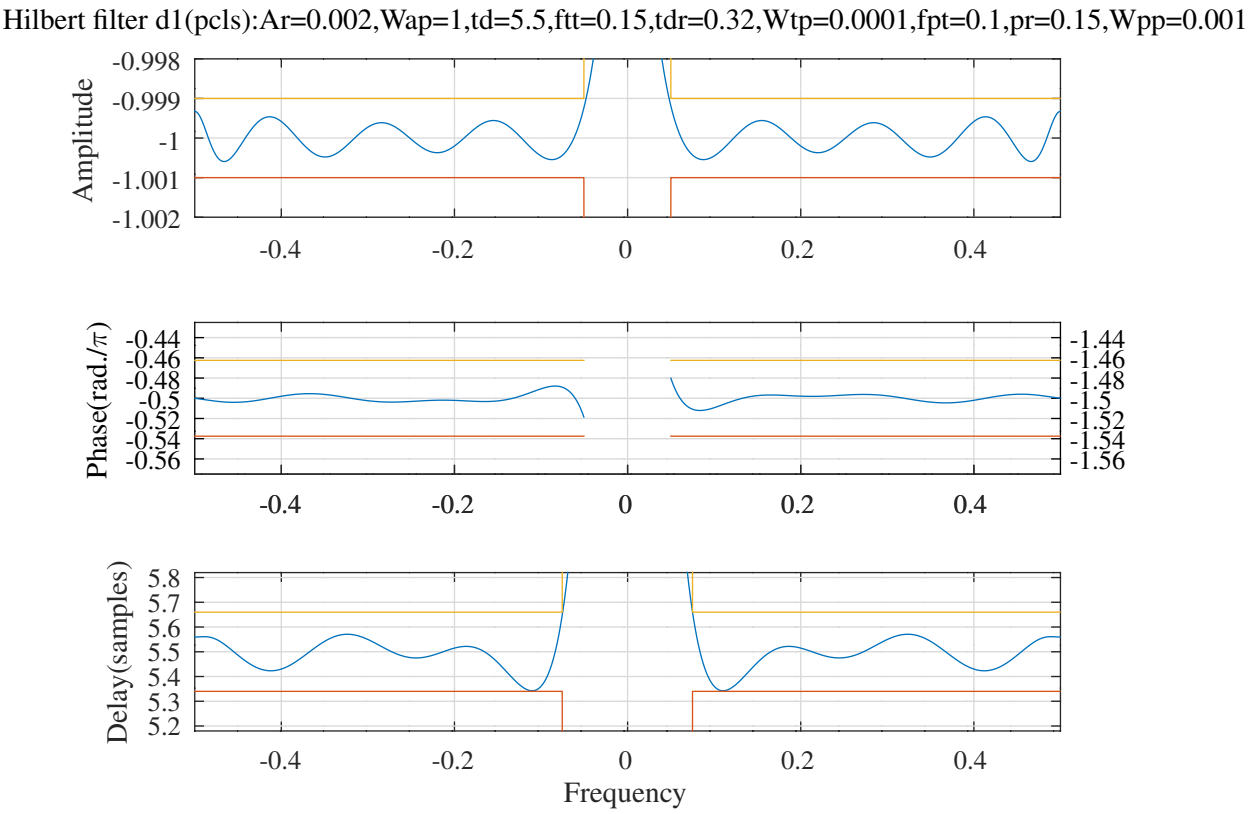



Figure 8.49: Response of the Hilbert filter after PCLS optimisation. The phase response shown is adjusted for the nominal delay.

```
D1 = [   1.0000000000,   0.0000000000,  -0.7821388481,   0.0000000000, ...
         0.0698509896,   0.0000000000,  -0.0049958626,   0.0000000000, ...
         0.0042178304,   0.0000000000,  -0.0003307402,   0.0000000000, ...
         0.0000082580 ]';
```

Hilbert filter d1(pcls):Ar=0.002,Wap=1,td=5.5,ftt=0.15,tdr=0.32,Wtp=0.0001,fpt=0.1,pr=0.15,Wpp=0.001

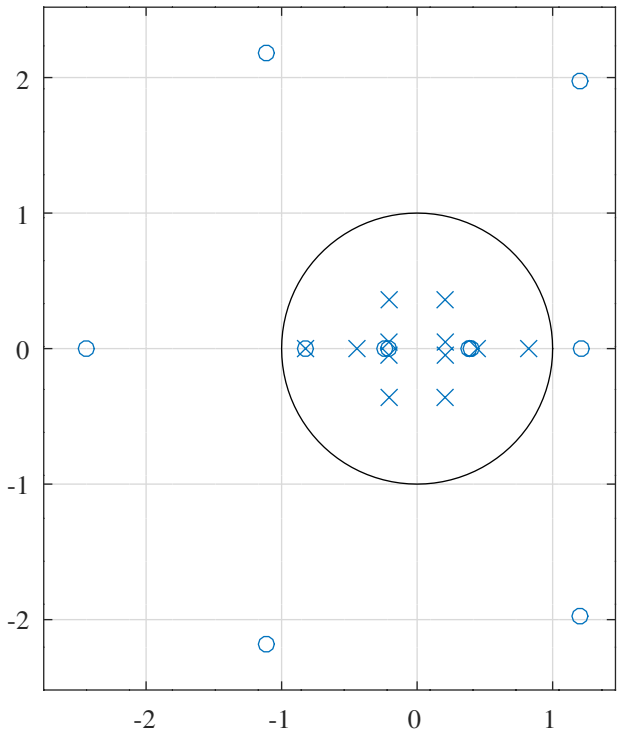



Figure 8.50: Pole-zero plot of the Hilbert filter after PCLS optimisation.

### 8.2.8 R=2 differentiator filter

A differentiator filter has transfer function $H_d(\omega) = \frac{\imath\omega}{\pi} e^{-\imath\omega t_d}$. The Octave script *iir_sqp_slb_differentiator_test.m* designs a differentiator filter with the specification:

```
tol=0.0001 % Tolerance on coef. update
ctol=5e-06 % Tolerance on constraints
U=4 % Number of real zeros
V=2 % Number of real poles
M=4 % Number of complex zeros
Q=2 % Number of complex poles
R=2 % Multiplicity of real and complex poles
n=1000 % Frequency points across the band
ft1=0.39 % Amplitude pass band first upper edge
Ar1=0.004 % Amplitude first peak-to-peak ripple
ft2=0.455 % Amplitude pass band second upper edge
Ar2=0.01 % Amplitude second peak-to-peak ripple
Wap=1 % Amplitude pass band weight
td=5.5 % Pass band group delay
tdr=0.0089 % Pass band group delay peak-to-peak ripple
Wtp=0.01 % Pass band group delay weight
pr=0.00 % Phase pass band peak-to-peak ripple(rad./$\pi$))
Wpp=0.01 % Phase pass band weight
```

The filter is designed with the effect of a fixed zero at $z = 1$ removed from the desired response. That zero is added back after optimisation is complete.

#### Initial filter for the R=2 differentiator filter

An initial filter was designed by the method of *Tarczynski et al* with the Octave script *tarczynski_differentiator_test.m*. The initial filter coefficients, in gain-pole-zero form, are:

```
Ux0=4,Vx0=2,Mx0=4,Qx0=2,Rx0=2
x0 = [   0.0032220974, ...
        -0.2657599176,   0.3525812573,   1.0067973881,   2.4623084816, ...
        -0.1903203150,   0.2967913957, ...
         2.5879786919,   2.6254494011, ...
         2.3918211297,   1.2321633117, ...
         0.2195422577, ...
         1.5304416148 ]';
```

Figure 8.51 shows the initial filter amplitude, phase and group delay responses. The phase response is adjusted for the nominal group delay. Figure 8.52 shows the pole-zero plot of the initial filter.

#### MMSE optimisation of the R=2 differentiator filter

Figure 8.53 shows the amplitude response error and the phase and group delay responses after MMSE optimisation of the initial filter. A zero near $z = 1$ is removed from the initial filter and the desired responses are adjusted to allow for a zero at exactly $z = 1$. The phase response is adjusted for the nominal group delay.

#### PCLS optimisation of the R=2 differentiator filter

Figure 8.54 shows the amplitude response error and the phase and group delay responses after PCLS optimised filter. The zero at $z = 1$ was added back after PCLS optimisation. The phase response is adjusted for the nominal group delay. Figure 8.55 shows the pole-zero plot for the PCLS optimised $R = 2$ differentiator filter. Note the double poles on the real and imaginary axes. The PCLS optimised filter coefficients are, in gain-pole-zero form:

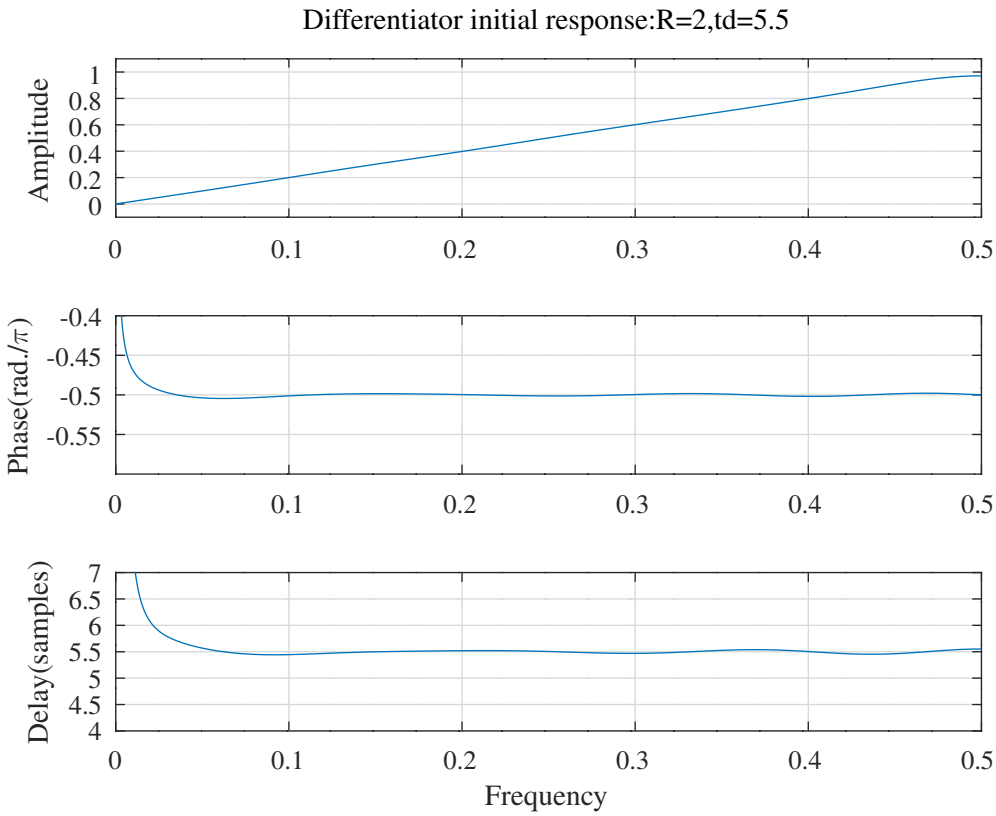



Figure 8.51: Amplitude, phase and group delay responses of the initial $R = 2$ differentiator filter. The phase response shown is adjusted for the nominal group delay.

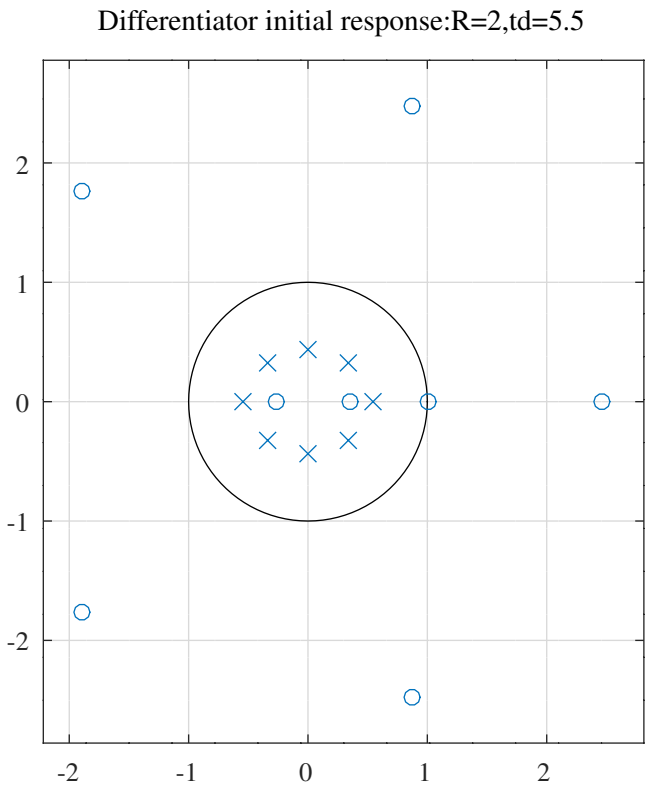



Figure 8.52: Pole-zero plot of the initial $R = 2$ differentiator filter.

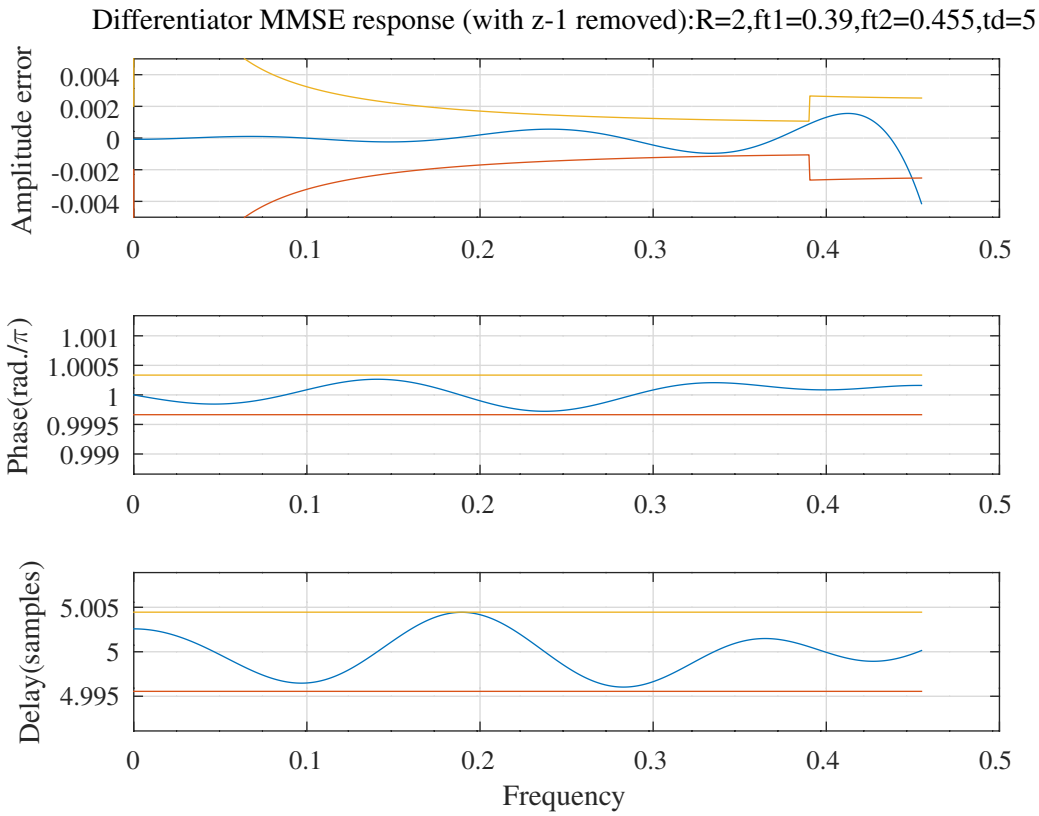



Figure 8.53: Amplitude response error and phase and group delay responses of the $R = 2$ differentiator filter, without a zero at $z = 1$, after MMSE optimisation. The phase response shown is adjusted for the nominal group delay.

```
Ud1=4,Vd1=2,Md1=4,Qd1=2,Rd1=2
d1 = [   0.0007338704, ...
        -0.2754788378,   0.3626894900,   1.0000000000,   3.7390954380, ...
        -0.0323822097,  -0.0307545513, ...
         3.2100460414,   3.6026809184, ...
         2.3099467492,   1.1253873860, ...
         0.0944237527, ...
         0.0000682279 ]';
```

The numerator and denominator polynomials are:

```
N1 = [   0.0007338704,  -0.0026460498,   0.0058848396,  -0.0134558511, ...
         0.0417984977,  -0.4011592477,   0.3976635854,   0.0078480196, ...
        -0.0366676641 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,  -0.1257107439,   0.0000000000, ...
        -0.0020114744,   0.0000000000,   0.0003748443,   0.0000000000, ...
         0.0000088793 ]';
```

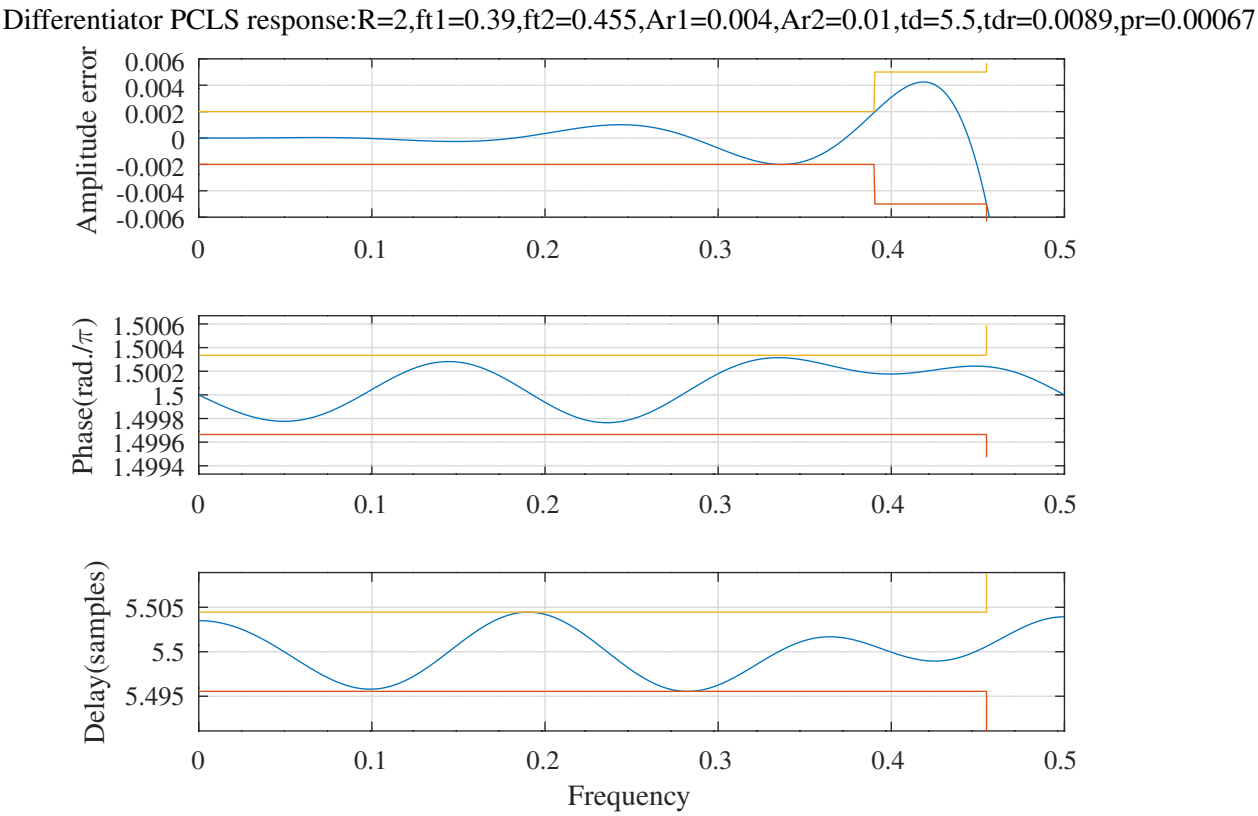



Figure 8.54: Amplitude response error and phase and group delay responses of the $R = 2$ differentiator filter after PCLS optimisation. The phase response is adjusted for the nominal group delay.

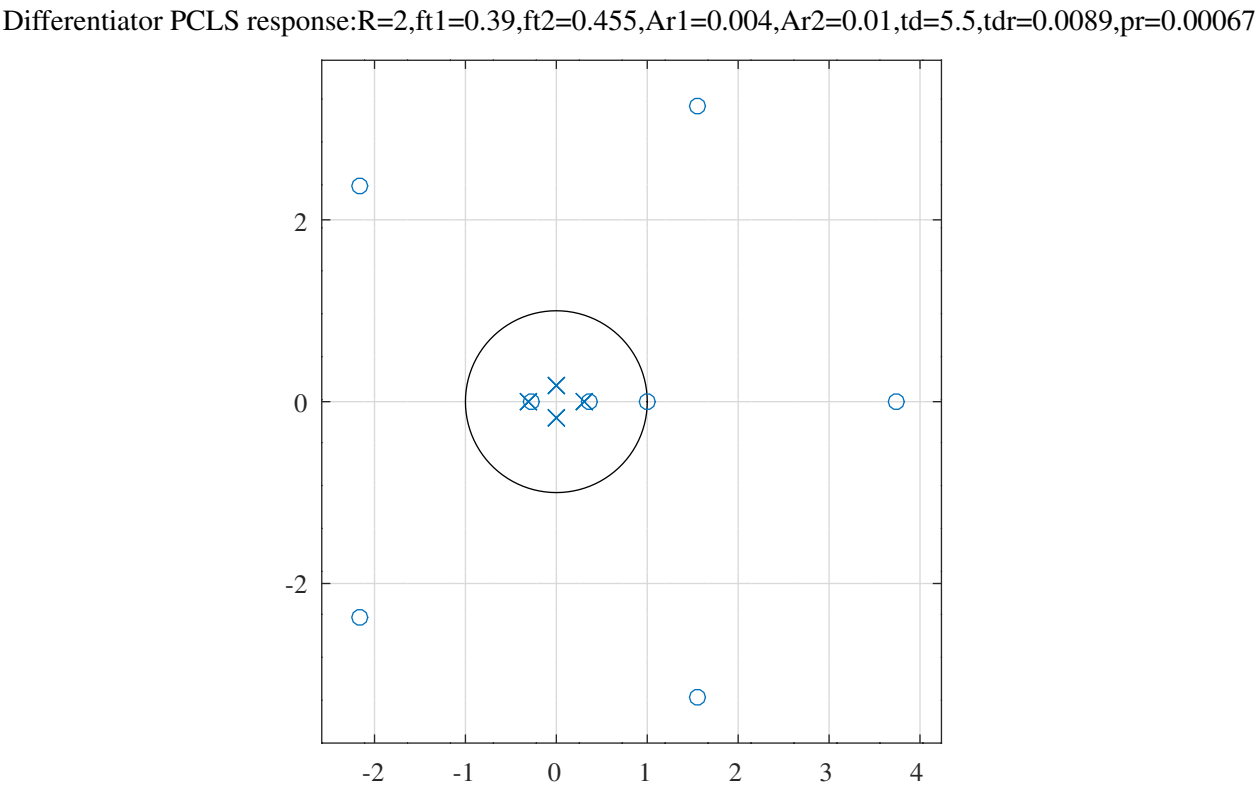



Figure 8.55: Pole-zero plot of the $R = 2$ differentiator filter after PCLS optimisation.

### 8.2.9 Low-pass differentiator filter

The Octave script *iir_sqp_slb_lowpass_differentiator_test.m* designs an IIR low-pass differentiator filter with the specification:

```
tol=0.0001 % Tolerance on coef. update
ctol=1e-07 % Tolerance on constraints
n=1000 % Frequency points across the band
fap=0.19 % Amplitude pass band upper edge
Arp=0.02 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Ars=0.01 % Amplitude stop band peak-to-peak ripple
Was=1 % Amplitude stop band weight
td=9 % Pass band group delay
tdr=0.03 % Pass band group delay peak-to-peak ripple
Wtp=0.02 % Pass band group delay weight
pr=0.0006 % Phase pass band peak-to-peak ripple(rad./$\pi$))
Wpp=0.02 % Phase pass band weight
```

The filter is designed with the effect of a fixed zero at $z = 1$ removed from the desired response. That zero is added back after optimisation is complete.

An initial filter was designed by the method of *Tarczynski et al* with the Octave script *tarczynski_lowpass_differentiator_test.m*. The inital filter numerator and denominator polynomials are:

```
N0 = [   0.0019869482,  -0.0025114882,  -0.0020553712,   0.0041191154, ...
         0.0022003604,  -0.0014834352,  -0.0082577359,  -0.0103465638, ...
        -0.0019667799,   0.0056739372,   0.0086764971,   0.0043468272 ]';

D0 = [   1.0000000000,  -1.6623907440,   1.4074516901,   0.2868406039, ...
        -1.9455756366,   2.0502062177,  -0.7184083864,  -0.6350459089, ...
         1.0240200241,  -0.6664781243,   0.2344048284,  -0.0378429241 ]';
```

After conversion to gain-zero-pole form and removal of the zero at $z = 1$, the amplitude, group delay and phase response errors of the initial filter are shown in Figure 8.56.

Figure 8.57 shows the response errors of the low-pass differentiator after PCLS optimisation. Figure 8.58 shows the pole-zero plot of the low-pass differentiator filter after PCLS optimisation. The PCLS optimised filter coefficients are, in gain-zero-pole form:

```
Ud1=3,Vd1=1,Md1=8,Qd1=10,Rd1=1
d1 = [   0.0006299703, ...
        -2.8746772513,   1.0000000000,   1.7971927779, ...
         0.5343783927, ...
         0.7730498546,   0.9093454863,   0.9908668629,   1.7372223476, ...
         2.3401628097,   1.7160591221,   1.5814128058,   0.7191724079, ...
         0.6023986003,   0.6227119592,   0.7243015321,   0.9290959042, ...
         0.9687500000, ...
         0.4018269110,   0.8749048685,   1.1646404734,   1.4173825047, ...
         1.5247607168 ]';
```

and, in transfer function form, the numerator and denominator polynomials are, respectively

```
N1 = [   0.0006299703,  -0.0007412410,  -0.0026207784,   0.0079143653, ...
        -0.0100885977,   0.0112707022,  -0.0153109758,   0.0048318311, ...
        -0.0079829172,   0.0070226755,   0.0003093598,   0.0047656059 ]';
```

and

```
D1 = [   1.0000000000,  -3.3870480730,   7.5286879188, -11.9521453513, ...
        14.6777954632, -14.3103781217,  11.1593595851,  -6.9047907990, ...
         3.3060908461,  -1.1633789142,   0.2716651500,  -0.0319576282 ]';
```

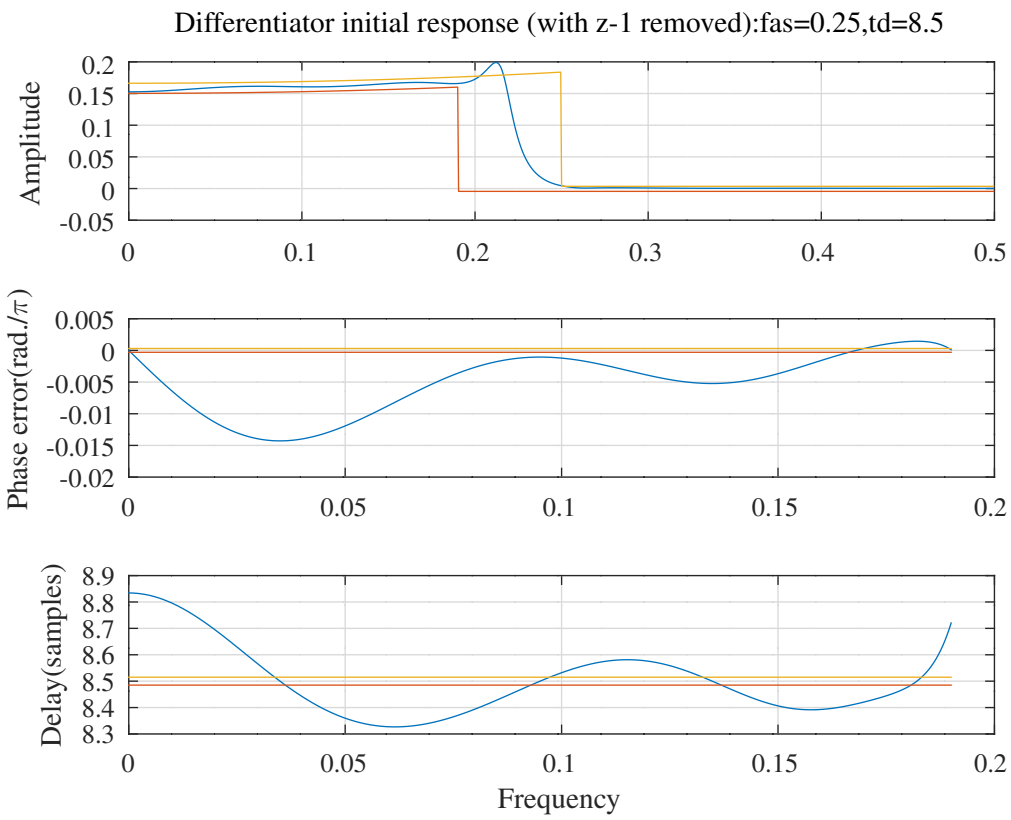



Figure 8.56: Response errors of the initial low-pass differentiator filter. The zero at $z = 1$ is removed and the phase response shown is adjusted for the nominal delay.

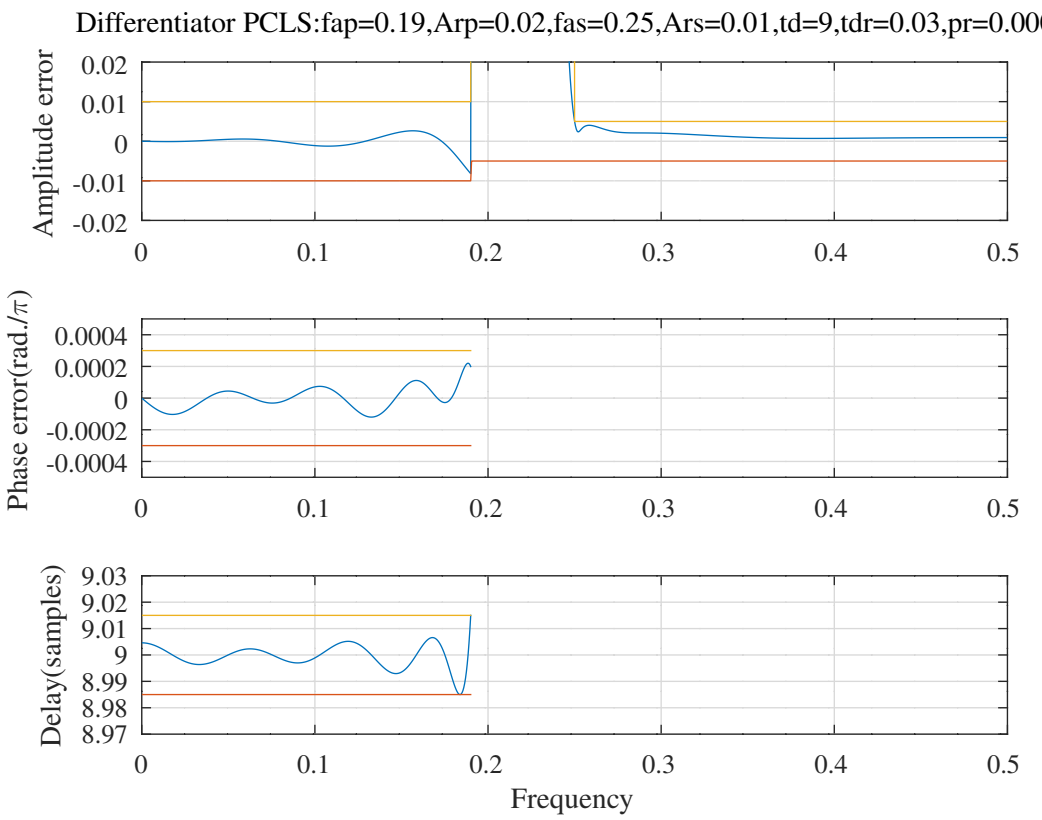



Figure 8.57: Response errors of the $R = 1$ low-pass differentiator filter after PCLS optimisation. The phase response shown is adjusted for the nominal delay.

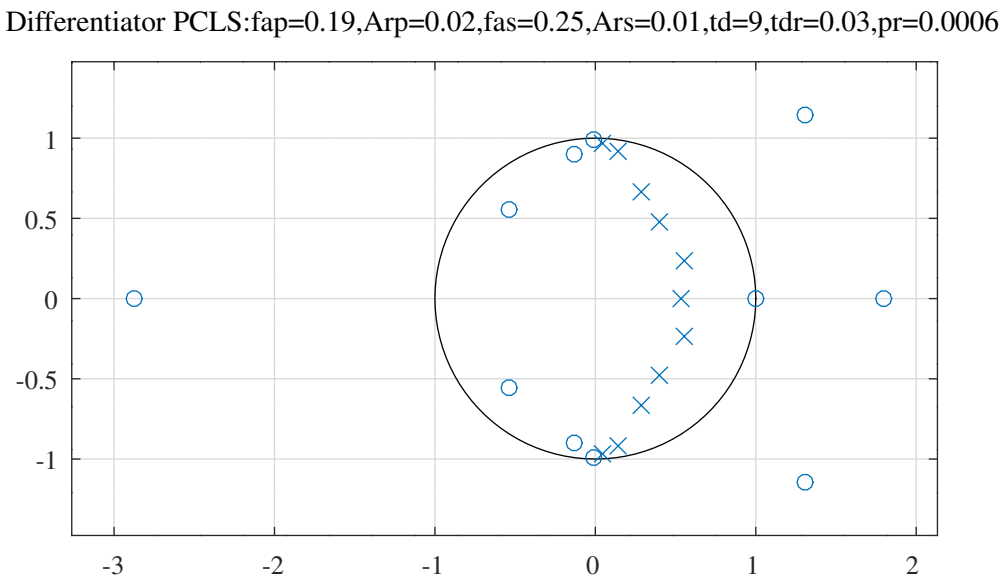



Figure 8.58: Pole-zero plot of the $R = 1$ low-pass differentiator filter after PCLS optimisation.

### 8.2.10 Pink noise filter

Pink noise has equal noise power per octave. A ideal pink noise filter has the transfer function $H_d(\omega) = \frac{K}{\sqrt{\omega}} e^{-\imath\omega t_d}$ 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_pcls=2e-05 % Tolerance on coef. update (PCLS pass)
ctol=2e-05 % Tolerance on constraints
fat=0.005 % Amplitude transition band width
AdBr=0.2 % Relative amplitude peak-to-peak ripple (dB)
Wap=10 % Amplitude weight
ftt=0.025 % Group delay transition band width
tp=4.77 % Nominal filter group delay
tpr=0.02 % 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 from $f = 0$ to $ftt = 0.025$ 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 = [   0.0255776792, ...
        -1.5626015368,  -0.1395126076,   0.7620284406, ...
         0.8665385044, ...
         0.6575701956,   0.6858167930,   1.5126492949,   1.5916883959, ...
         2.5461293871,   1.3955409796,   0.8496676072,   1.9998926521, ...
         0.5304440873,   0.5481152693,   0.6085420259,   0.6215289813, ...
         0.6724036776, ...
         0.4676243898,   1.5734381101,   2.6881937981,   2.4389647012, ...
         1.3509243598 ]';
```

Figure 8.59 compares the amplitude response of the initial filter to the desired response. Figure 8.60 shows the amplitude and group delay responses of the filter after MMSE and PCLS optimisation. Figure 8.61 shows the errors of the amplitude and group delay responses of the filter after MMSE and PCLS optimisation. Figure 8.62 shows the pole-zero plot of the filter.

The PCLS optimised filter coefficients are, in gain-pole-zero form

```
Ud1=3,Vd1=1,Md1=8,Qd1=10,Rd1=1
d1 = [   0.0008367854, ...
        -2.7692029682,   0.7592390468,   0.8792893734, ...
         0.9611515965, ...
         0.5006813041,   0.9710174468,   3.1183691894,   3.2110419306, ...
         0.0000002906,   0.0298039746,   0.6946535263,   1.8874490840, ...
         0.3346426995,   0.3483303196,   0.5109819435,   0.7590127871, ...
         0.9687500000, ...
         2.4095991920,   1.2407315830,   0.3499620731,   0.0000001496, ...
         0.0248996913 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively:

```
N1 = [   0.0008367854,  -0.0038522565,   0.0095828029,  -0.0185870169, ...
         0.0420800612,   0.1380477090,  -0.9214075629,   1.8736064450, ...
        -1.9139954321,   1.0666468181,  -0.3096161756,   0.0366608948 ]';
```

and

```
D1 = [   1.0000000000,  -5.1040419316,  10.9344922022, -12.7151163261, ...
         8.6707651484,  -3.5274214236,   0.8952213706,  -0.2162437649, ...
         0.0933006449,  -0.0417051979,   0.0125944359,  -0.0018436135 ]';
```

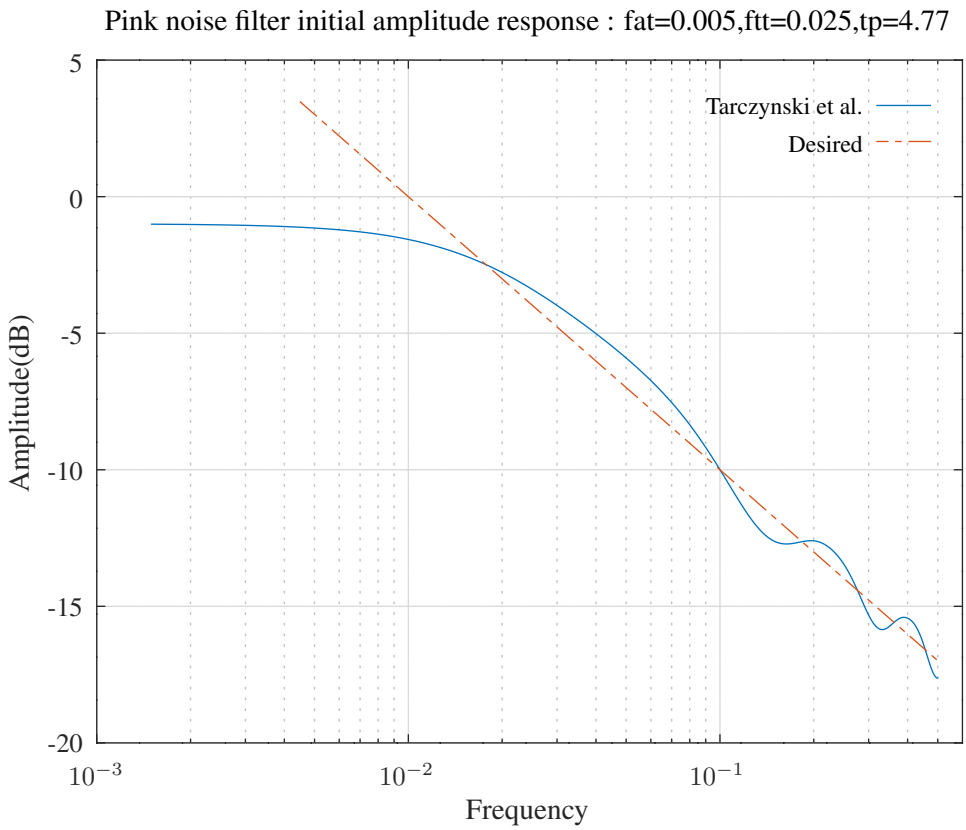



Figure 8.59: Response of the initial pink noise filter designed by *tarczynski_pink_test.m*

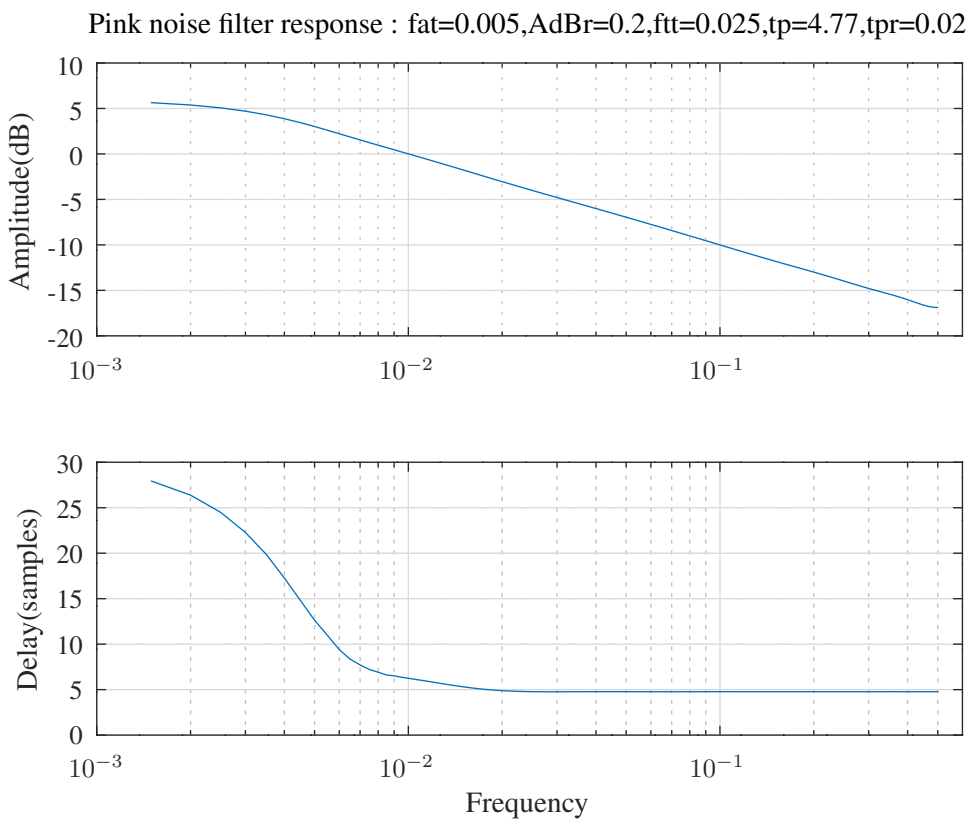



Figure 8.60: Amplitude and group-delay responses of the pink noise filter after PCLS optimisation

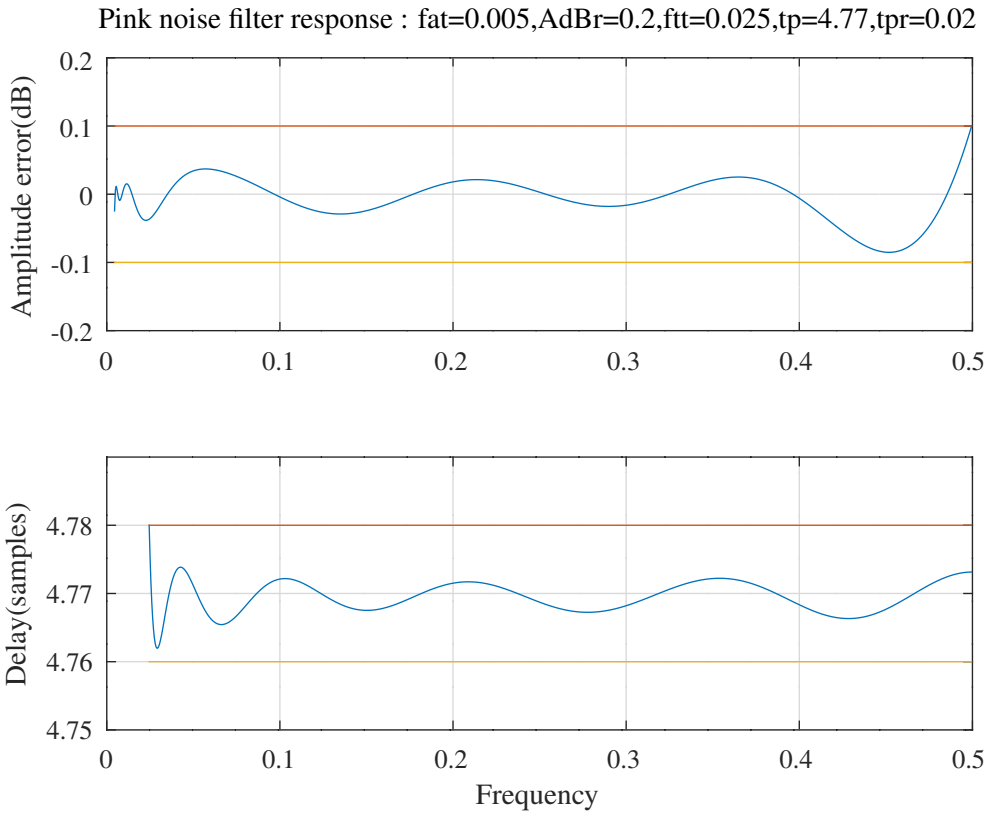



Figure 8.61: Errors of the amplitude and group-delay responses of the pink noise filter after PCLS optimisation

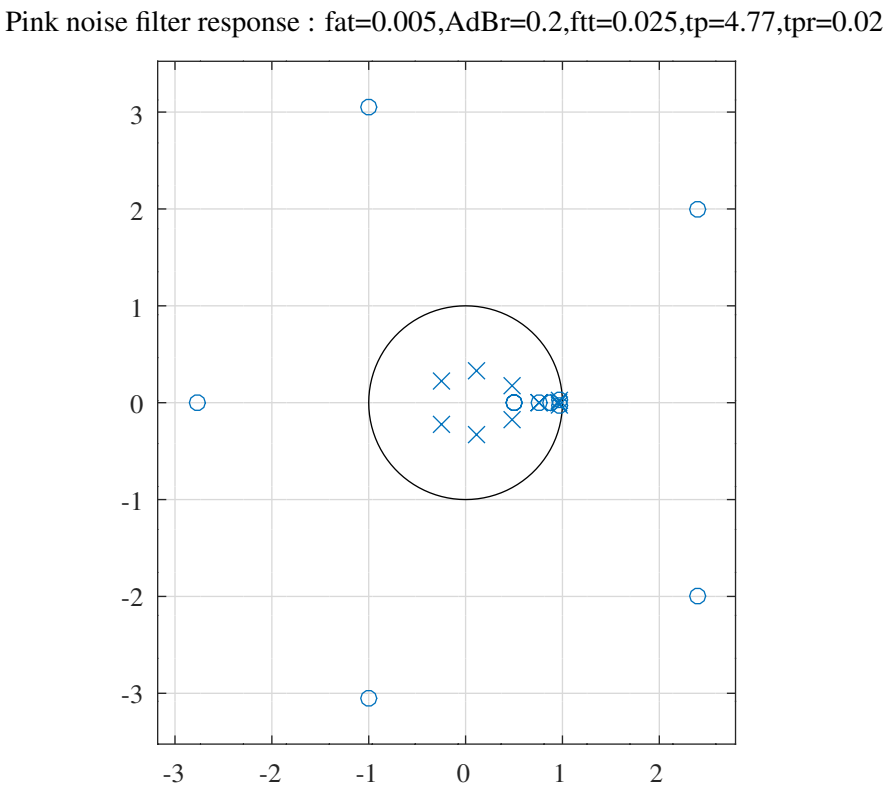



Figure 8.62: Pole-zero plot of the pink noise filter after PCLS optimisation

### 8.2.11 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 $|z| \leq 1$[b] 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=0.0001 % Tolerance on relative coefficient update size
ctol=1e-06 % Tolerance on constraints
fap=0.15 % Pass band amplitude response edge
dBap=0.05 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
fas=0.25 % Stop band amplitude response edge
dBas=50 % Stop band minimum attenuation
Was=2 % 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=[ 0.004; ...
    -127/128;-127/128; ...
     0.6; ...
     127/128*ones(4,1); ...
     pi*(9:12)'/16; ...
     0.6;0.6; ...
     2*pi/3;pi/2 ];
```

The inital filter is optimised with the PCLS algorithm. The PCLS optimised pole and zero radiuses are contrained to be less than $\frac{255}{256}$. Figure 8.63 shows the response of the PCLS optimised filter. Figure 8.64 shows the pass-band detail of the response of the PCLS optimised filter. Figure 8.65 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 = [   0.0109265326, ...
        -0.9960937500,  -0.9960937500, ...
         0.1331336923, ...
         0.9960937500,   0.9960937500,   0.9960937500,   0.9960937500, ...
         1.6326159532,   1.8780210315,   2.1203847714,   2.4383327048, ...
         0.3448692677,   0.7591836747, ...
         1.7292389673,   2.1371565168 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively

```
N1 = [   0.0109265326,   0.0576684981,   0.1641107349,   0.3199508233, ...
         0.4650951980,   0.5239189716,   0.4614687386,   0.3149808079, ...
         0.1603017564,   0.0558908046,   0.0105071399 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,   0.7903950732,   0.0000000000, ...
         0.6610036461,   0.0000000000,   0.0552490469,   0.0000000000, ...
         0.0472984422,   0.0000000000,  -0.0091262148 ]';
```

[b] If the inverse filter is required then the zeros must be within the unit circle, $|z| \leq \rho < 1$.

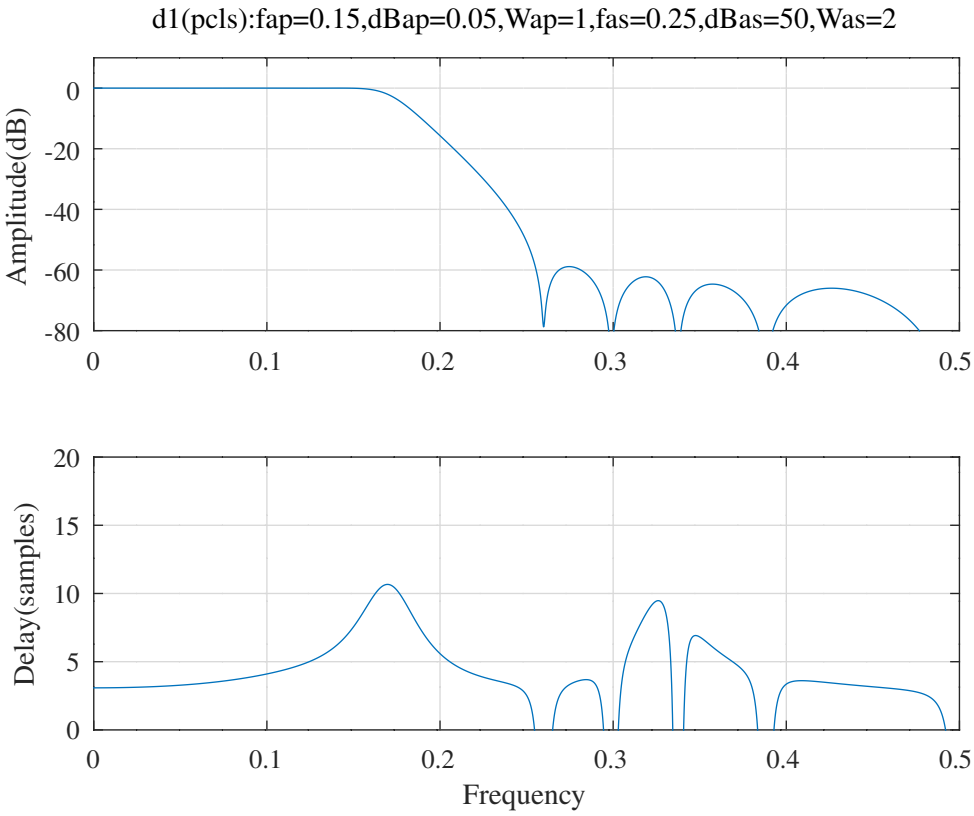



Figure 8.63: Response of the R=2 minimum phase filter after PCLS optimisation

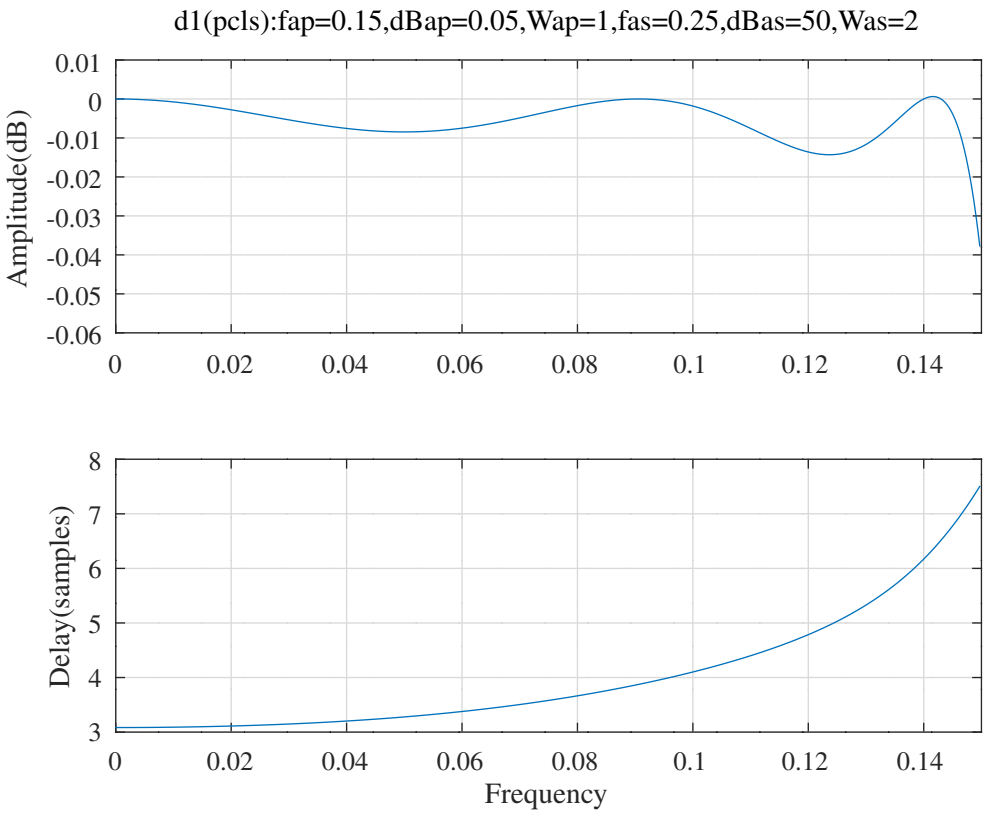



Figure 8.64: Passband response of the R=2 minimum phase filter after PCLS optimisation

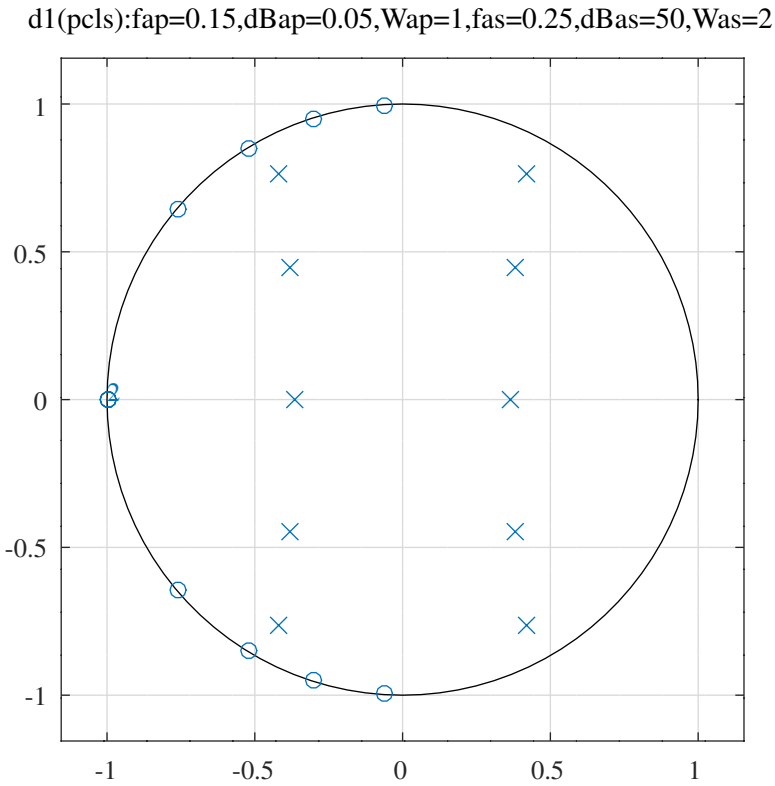



Figure 8.65: Pole-zero plot of the R=2 minimum phase filter after PCLS optimisation

### 8.2.12 Non-linear phase FIR low-pass filter

The Octave script *iir_sqp_slb_fir_lowpass_test.m* designs an FIR low-pass filter that has approximately linear phase in the pass-band. The impulse response is *not* symmetric. The filter specification is:

```
U=0 % Number of real zeros
V=0 % Number of real poles
M=60 % Number of complex zeros
Q=0 % Number of complex poles
R=1 % Denominator polynomial decimation factor
n=1000 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-05 % Tolerance on constraints
fap=0.15 % Pass band amplitude response edge
dBap=0.2 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band amplitude weight
Wat=0.01 % Transition band amplitude weight
ftp=0.15 % Pass band group-delay response edge
td=15 % Pass band group-delay
tdr=0.3 % Pass band amplitude peak-to-peak ripple
Wtp=0.05 % Pass band group-delay weight
fas=0.2 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=50 % Stop band amplitude weight
```

Figure 8.66 shows the response of the PCLS optimised filter. Figure 8.67 shows the pass-band detail of the response of the PCLS optimised filter. The coefficients of the PCLS optimised minimum phase FIR bandpass filter are, in gain-pole-zero form:

```
Ud1=0,Vd1=0,Md1=60,Qd1=0,Rd1=1
d1 = [   0.0005324559, ...
         0.4397616766,   0.8273378250,   0.8337259742,   0.8357796520, ...
         0.8413596063,   0.8534180769,   0.8651179066,   0.8929405815, ...
         0.9929876825,   0.9932067147,   0.9940588880,   0.9942160251, ...
         0.9952692283,   0.9966996456,   0.9978939723,   0.9982000478, ...
         1.0001639773,   1.0017963087,   1.0038536454,   1.0062443170, ...
         1.0095443342,   1.0124051427,   1.0174757734,   1.0229432596, ...
         1.0487497079,   1.1962531891,   1.2574105079,   1.2773085708, ...
         1.4055079108,   1.4391638242, ...
         0.0000818472,   0.0751903261,   0.2259629977,   0.3769608154, ...
         0.5230983818,   0.6652825081,   0.8090692055,   0.9523254793, ...
         1.4143884367,   1.5079418104,   1.6062519515,   1.3308898599, ...
         1.7070993964,   1.8095646589,   1.2709278169,   1.9131942655, ...
         2.0177484818,   2.1233526394,   2.2300350925,   2.3378971001, ...
         2.4473058062,   2.5592274347,   2.6758130293,   2.8012576017, ...
         2.9468981260,   3.1413019855,   3.1415853689,   3.1415925370, ...
         0.6633283451,   0.2188657165 ]';
```

and in transfer function form the FIR polynomial is:

```
N1 = [   0.0005324559,   0.0028087091,   0.0054402539,   0.0033744046, ...
        -0.0046193497,  -0.0112440454,  -0.0063367345,   0.0108600263, ...
         0.0234011508,   0.0096907719,  -0.0289690222,  -0.0541332998, ...
        -0.0130595483,   0.1102012011,   0.2626491003,   0.3478955391, ...
         0.3000206547,   0.1430646333,  -0.0213908373,  -0.0940568692, ...
        -0.0561142749,   0.0247443989,   0.0627637105,   0.0307463467, ...
        -0.0266529394,  -0.0477306111,  -0.0168733901,   0.0269607132, ...
         0.0371074011,   0.0079609777,  -0.0249244238,  -0.0268401781, ...
         0.0000239736,   0.0236945016,   0.0196797142,  -0.0043308813, ...
        -0.0206780747,  -0.0134045183,   0.0066607414,   0.0169118403, ...
         0.0082710167,  -0.0074167870,  -0.0129294828,  -0.0044022184, ...
         0.0069525088,   0.0090580900,   0.0015868609,  -0.0059705347, ...
        -0.0059765251,  -0.0000926916,   0.0044721539,   0.0035206121, ...
        -0.0005700753,  -0.0029456046,  -0.0017914741,   0.0006766692, ...
         0.0016679159,   0.0007803396,  -0.0003804114,  -0.0005326493, ...
         0.0001880815 ]';
```

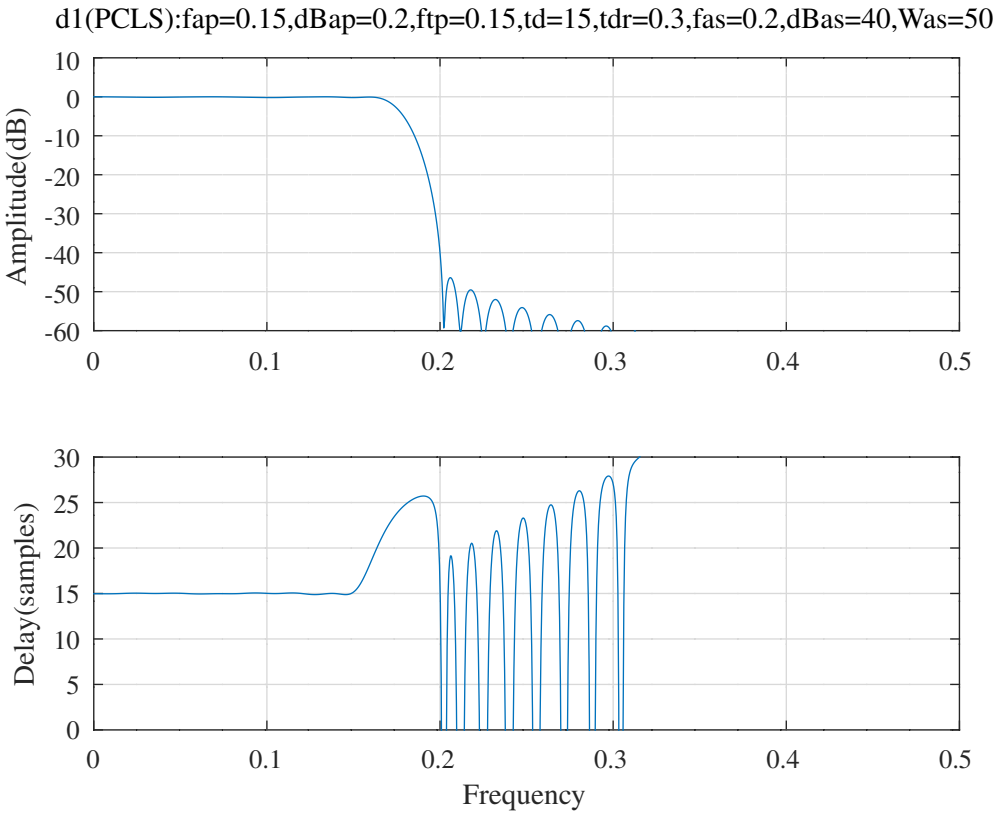



Figure 8.66: Response of the non-linear phase FIR low-pass filter after SQP PCLS optimisation

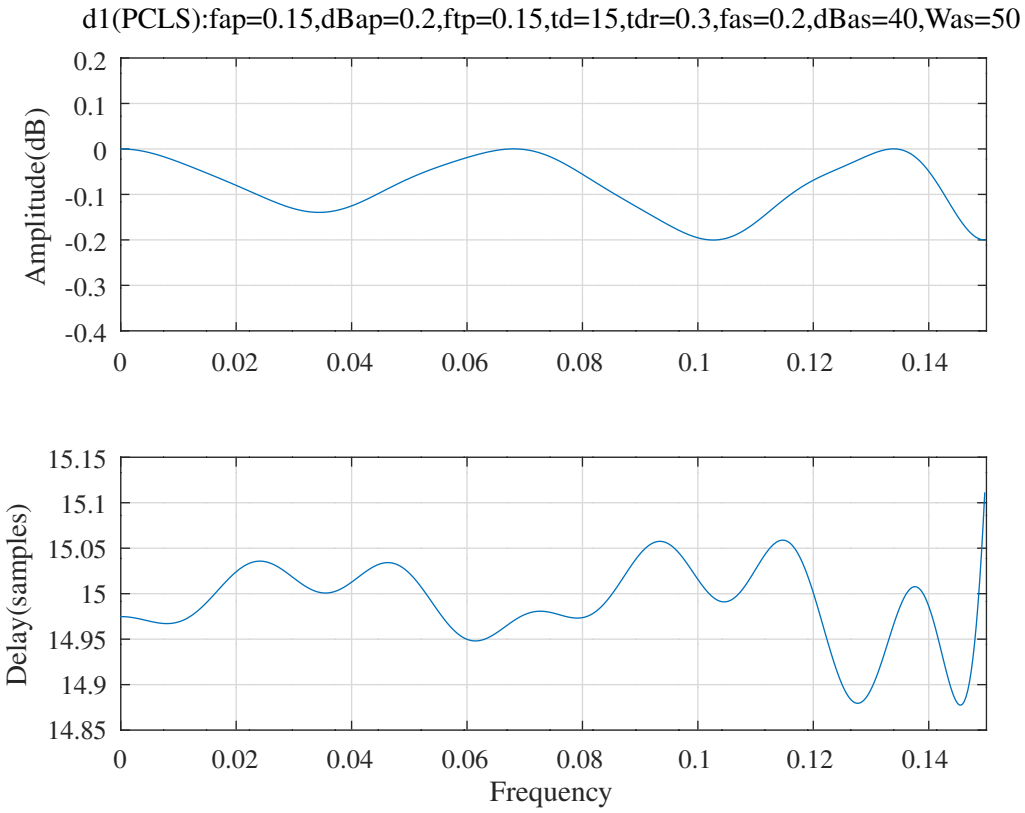



Figure 8.67: Pass-band response of the non-linear phase FIR low-pass filter after SQP PCLS optimisation

### 8.2.13 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=8,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 radiuses are constrained by $|z| < \frac{31}{32}$. The inital filter is first MMSE optimised and then optimised with the PCLS algorithm. The radiuses of the zeros of the filter are constrained to $|z| \leq 1$. Figure 8.68 shows the response of the PCLS optimised filter. Figure 8.69 shows the pass-band detail of the response of the PCLS optimised filter. Figure 8.70 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 = [   0.0179091263, ...
         0.9687500000,   0.9687500000, ...
         0.4070764056,   0.8511982553,   0.8555133528,   0.9687500000, ...
         0.9687500000,   0.9687500000,   0.9687500000,   0.9687500000, ...
         0.9687500000,   0.9687500000,   0.9687500000,   0.9687500000, ...
         0.9687500000,   0.9687500000, ...
         3.1365798355,   0.7978118979,   1.0532833102,   1.8809248876, ...
         2.7455719979,   0.2396194338,   0.2971109856,   1.6694428110, ...
         2.0793063810,   2.2871802908,   3.0632867167,   3.1228351190, ...
         2.5076893141,   1.5867741656 ]';
```

and in transfer function form the FIR polynomial is:

```
Nd1 = [   0.0179091263,   0.0600437111,   0.0669433379,  -0.0231028769, ...
         -0.1631114413,  -0.2018081476,  -0.0495400228,   0.1786554928, ...
          0.2614552768,   0.1221840488,  -0.0802141813,  -0.1476967264, ...
         -0.0703183938,   0.0049621768,  -0.0086811942,  -0.0461982090, ...
         -0.0201858875,   0.0468110934,   0.0703713932,   0.0304282070, ...
         -0.0132356186,  -0.0153738518,   0.0029406949,   0.0020141883, ...
         -0.0170262505,  -0.0244047107,  -0.0095653611,   0.0091122026, ...
          0.0122887431,   0.0051340730,   0.0007345519 ]';
```

The script also calculates the FIR filter with complementary amplitude response. The combined response is shown in Figure 8.71. Figure 8.72 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 = [   0.0384265071, ...
        -0.6713032668,  -0.6713032668, ...
         0.5000066490,   0.7158648819,   0.7201962664,   0.7375682194, ...
         0.7434167215,   0.7634299094,   0.9914622385,   0.9999999995, ...
         1.0104597246,   1.2541457073,   1.3173859985,   1.3444913492, ...
         1.3493381930,   1.3666033553, ...
         0.0000032646,   1.6012960814,   2.8834765700,   2.4472483716, ...
         2.0220348637,   0.1227311725,   1.1656843007,   0.9141739737, ...
         0.7052428690,   0.3516775350,   1.5094872474,   1.8094385757, ...
         2.2339680879,   2.6630555546 ]';
```

and in transfer function form the FIR polynomial is:

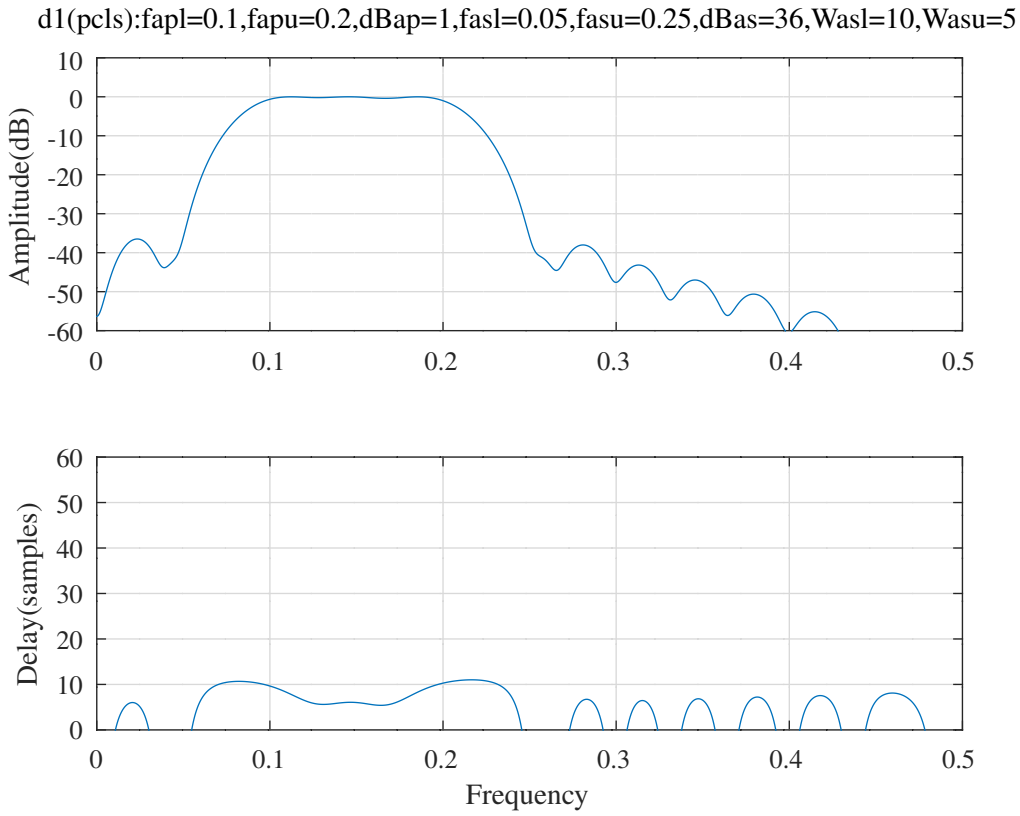



Figure 8.68: Response of the minimum phase FIR bandpass filter after PCLS optimisation

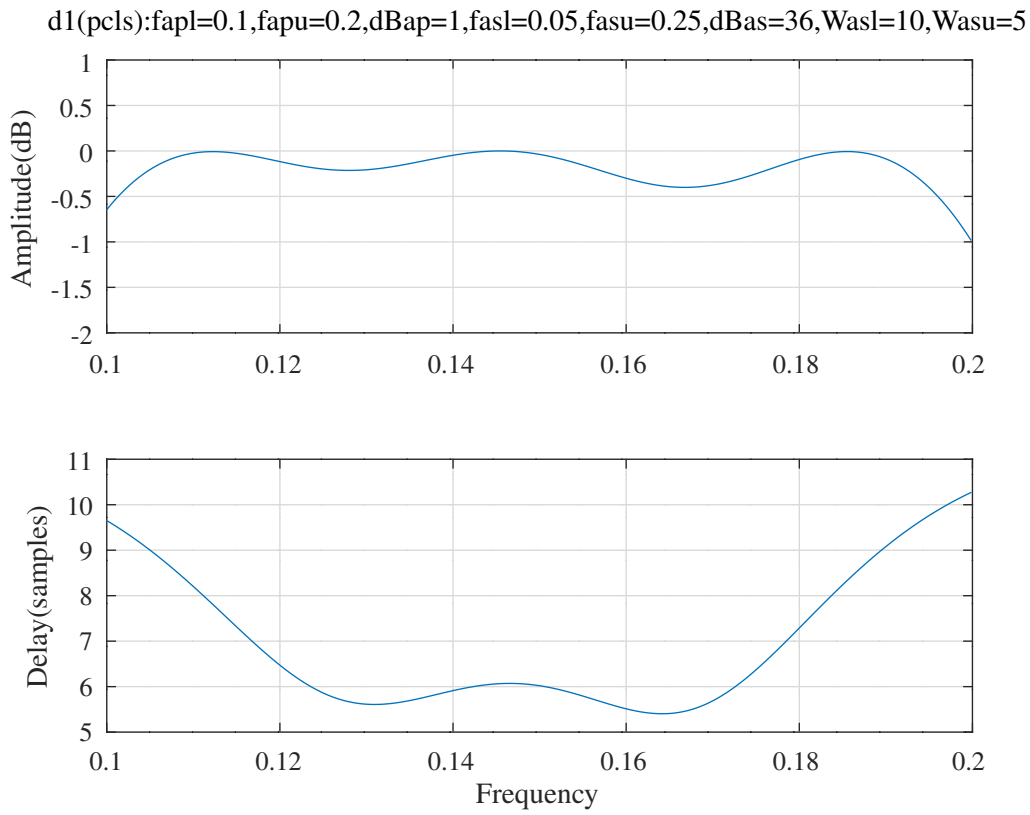



Figure 8.69: Passband response of the minimum phase FIR bandpass filter after PCLS optimisation

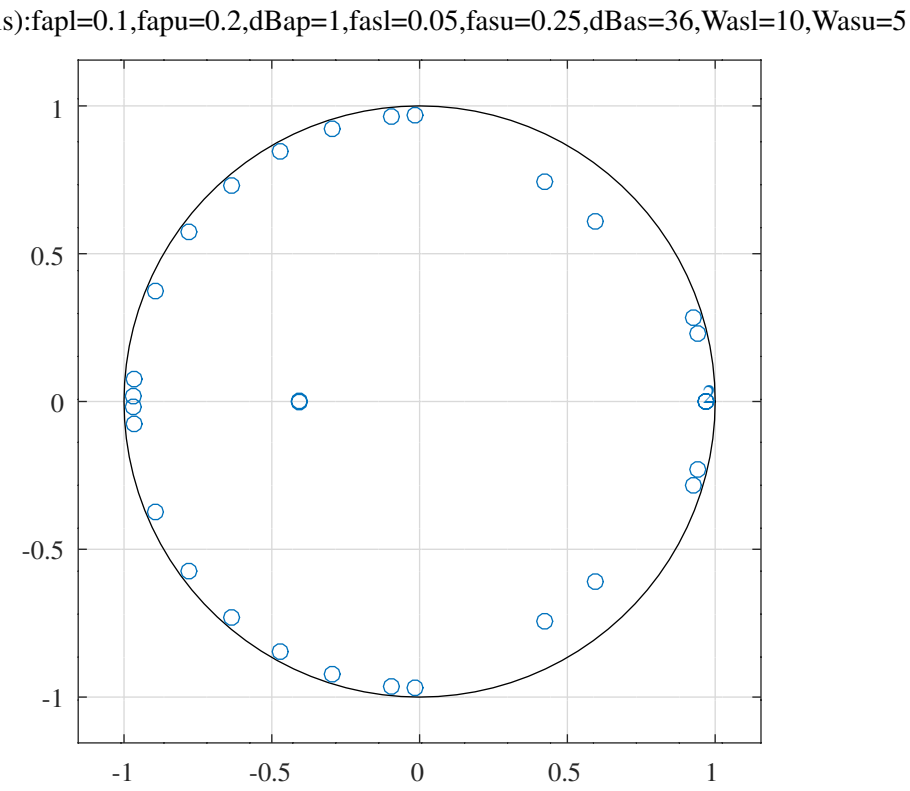



Figure 8.70: Pole-zero plot of the minimum phase FIR bandpass filter after PCLS optimisation

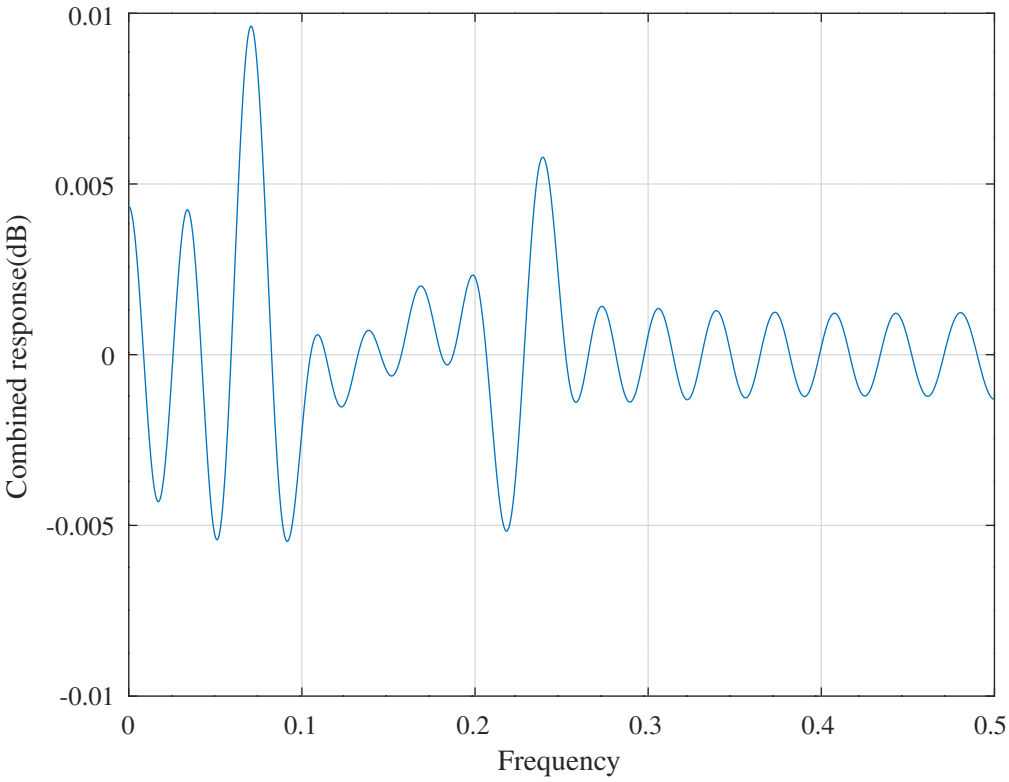



Figure 8.71: Combined response of the PCLS optimised minimum phase FIR bandpass filter and the complementary FIR filter

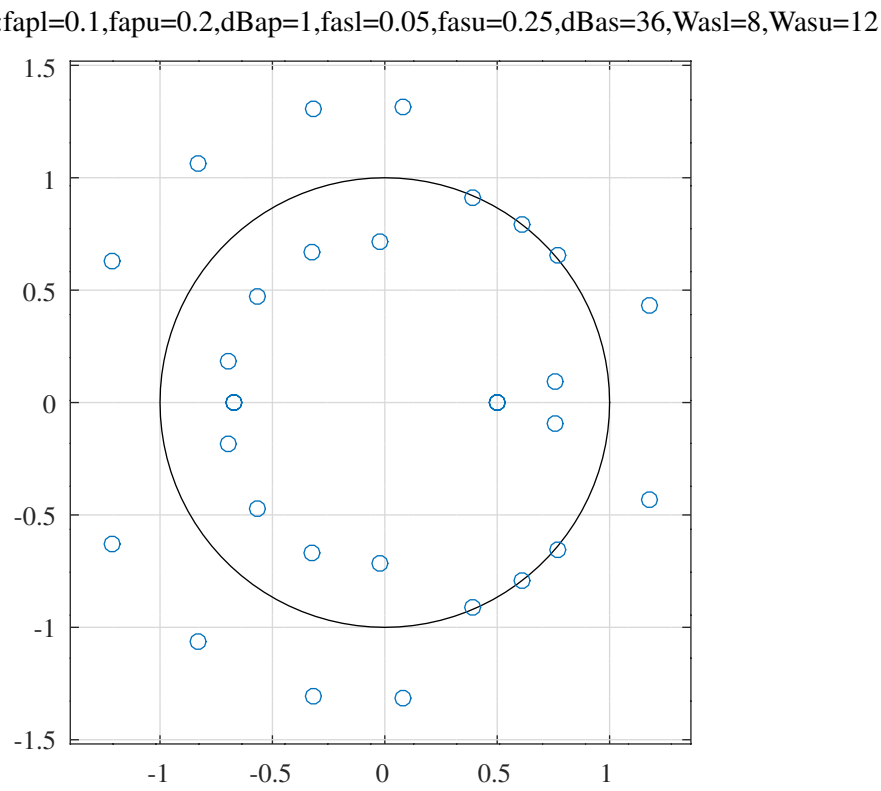



Figure 8.72: Pole-zero plot of the complementary FIR filter

```
Nc1 = [   0.0384265071,   0.0273312657,   0.0597243956,  -0.0206003541, ...
          0.0074162664,  -0.0714831921,   0.1213486206,   0.0071930435, ...
          0.2350911818,  -0.0814549847,   0.3369863771,  -0.0816334736, ...
          0.4960359384,   0.1104568289,  -0.2627229484,   0.3622847178, ...
          0.0994740870,  -0.2151842642,  -0.1146291932,   0.0116571258, ...
         -0.0119188743,  -0.0975031199,  -0.0764519014,   0.0107248470, ...
          0.0714579946,   0.0523546574,   0.0155943541,  -0.0133313046, ...
         -0.0190542848,  -0.0004873269,   0.0033958327 ]';
```

Alternatively, given an FIR filter of order $N$, $H(z)$, with the desired magnitude response, $|H(e^{\imath\omega})| \leq 1$, a more accurate complementary minimum-phase FIR spectral factor of $1 - z^{-N} H\left(z^{-1}\right) H(z) = 1 - |H(z)|^2$ can be derived from the real cepstrum method of *Mian* and *Nainer* [54] or directly by *Orchard* and *Willson*'s Newton-Raphson solution [63]. See Appendix N.1.4.

# Chapter 9

# IIR filter design using Second Order Cone Programming

## 9.1 Second Order Cone Programming

Following *Alizadeh* and *Goldfarb* [45], 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

$$\begin{aligned} \textbf{minimise} \quad & \boldsymbol{f}^\top \boldsymbol{x} \\ \textbf{subject to} \quad & \|A_i \boldsymbol{x} + b_i\| \leq c_i^\top \boldsymbol{x} + h_i \quad i = 1, \ldots, N \end{aligned}$$

where $\|u\| = \sqrt{u^\top u}$ is the Euclidean norm of the vector $u$, $\boldsymbol{x} \in \mathbb{R}^{p\times 1}$, $\boldsymbol{f} \in \mathbb{R}^{p\times 1}$, $A_i \in \mathbb{R}^{(p-1)\times p}$, $b_i \in \mathbb{R}^{(p-1)\times 1}$, $c_i \in \mathbb{R}^{p\times 1}$ and $h_i \in \mathbb{R}$. Each constraint is equivalent to:

$$\begin{bmatrix} c_i^\top \\ A_i \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} h_i \\ b_i \end{bmatrix} \in \mathcal{C}_i$$

where $\mathcal{C}_i$ is a cone in $\mathbb{R}^p$

$$\mathcal{C}_i = \left\{ \begin{bmatrix} t \\ u \end{bmatrix} \; : \; u \in \mathbb{R}^{(p-1)\times 1}, \; t \geq 0, \; \|u\| \leq t \right\}$$

## 9.2 Design of IIR filters with SOCP

Equation 2 defined the mini-max optimisation problem for the filter. Rewrite this in terms of an upper bound, $\epsilon$, as

$$\begin{aligned} \textbf{minimise} \quad & \epsilon \\ \textbf{subject to} \quad & \|W(\omega_i)\left[H(\boldsymbol{x}, \omega_i) - H_d(\omega_i)\right]\| \leq \epsilon \quad \text{for } \omega_i \in \Omega \\ & H(\boldsymbol{x}) \text{ is stable} \end{aligned}$$

where $\boldsymbol{x}$ is the vector of coefficients, $\epsilon$ is an auxilliary optimisation variable, $W(\omega)$ is a weighting factor, $H(\boldsymbol{x}, \omega)$ is the frequency response with coefficients $\boldsymbol{x}$ and $H_d(\omega)$ is the desired frequency response. $H(\boldsymbol{x}, \omega)$ and $H_d(\omega)$ usually have complex values in the pass band. The optimisation problem may be formulated so that $H(\boldsymbol{x}, \omega)$ 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 $\nabla_x H(\boldsymbol{x}_k, \omega)$ is known and the step-size, $\|\boldsymbol{x} - \boldsymbol{x}_k\|$, is small, then a useful *first-order* approximation to $H(\boldsymbol{x}, \omega)$ is:

$$H(\boldsymbol{x}, \omega) \approx H(\boldsymbol{x}_k, \omega) + \nabla_x H(\boldsymbol{x}_k, \omega)^\top (\boldsymbol{x} - \boldsymbol{x}_k)$$

so that

$$\begin{aligned} W(\omega)\left|H(\boldsymbol{x}, \omega) - H_d(\omega)\right| &\approx W(\omega)\left|\nabla_x H(\boldsymbol{x}_k, \omega)^\top (\boldsymbol{x} - \boldsymbol{x}_k) + H(\boldsymbol{x}_k, \omega) - H_d(\omega)\right| \\ &= W(\omega)\left|\nabla_x H(\boldsymbol{x}_k, \omega)^\top (\boldsymbol{x} - \boldsymbol{x}_k) + \boldsymbol{e}_k\right| \end{aligned}$$

where $\boldsymbol{e}_k = H(\boldsymbol{x}_k, \omega) - H_d(\omega)$.

Similarly, a *second-order* approximation to $H(\boldsymbol{x}, \omega)$ is:

$$H(\boldsymbol{x}, \omega) \approx H(\boldsymbol{x}_k, \omega) + \nabla_x H(\boldsymbol{x}_k, \omega)^\top (\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}_k)^\top \nabla_{xx}^2 H(\boldsymbol{x}_k, \omega) (\boldsymbol{x} - \boldsymbol{x}_k)$$

Unfortunately the IIR filter Hessian matrix, $\nabla_{xx}^2 H(\boldsymbol{x}_k, \omega)$, is unlikely to be positive definite.

Lu and Hinamoto [212] add linear constraints on the maximum filter response in the transition band, $\Omega_t$:

$$\|\nabla_x H(\boldsymbol{x}_k, \omega_t)^\top (\boldsymbol{x} - \boldsymbol{x}_k) + H(\boldsymbol{x}_k, \omega_t)\| \leq \gamma \quad \text{for } \omega_t \in \Omega_t$$

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

$$\begin{aligned}
\textbf{minimise} \quad & \epsilon \\
\textbf{subject to} \quad & \|W(\omega_i) \left[\nabla_x H(\boldsymbol{x}_k, \omega_i)^\top (\boldsymbol{x} - \boldsymbol{x}_k) + \boldsymbol{e}_k\right]\| \leq \epsilon \quad \text{for } \omega_i \in \Omega \\
& \|\nabla_x H(\boldsymbol{x}_k, \omega_t)^\top (\boldsymbol{x} - \boldsymbol{x}_k) + H(\boldsymbol{x}_k, \omega_t)\| \leq \gamma \quad \text{for } \omega_t \in \Omega_t \\
& \|\boldsymbol{x} - \boldsymbol{x}_k\| \leq \beta \\
& H(\boldsymbol{x}) \text{ is stable}
\end{aligned}$$

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, $(\boldsymbol{x} - \boldsymbol{x}_k)$, is satisfactory or the iteration limit is exceeded. Typically, $\epsilon$ 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 of two auxilliary variables $\epsilon_A$ and $\epsilon_T$.

The complex frequency response can be written in terms of amplitude and phase as

$$\begin{aligned}
H(\boldsymbol{x}_k, \omega_i) &= H_a(\boldsymbol{x}_k, \omega_i) e^{\imath H_p(\boldsymbol{x}_k, \omega_i)} \\
&= H_a(\boldsymbol{x}_k, \omega_i) [\cos H_p(\boldsymbol{x}_k, \omega_i) + \imath \sin H_p(\boldsymbol{x}_k, \omega_i)]
\end{aligned}$$

so that, at each $\omega_i$, the gradient of the complex frequency response is

$$\begin{aligned}
\nabla_x H(\boldsymbol{x}_k, \omega_i) = & [\cos H_p(\boldsymbol{x}_k, \omega_i) \nabla_x H_a(\boldsymbol{x}_k, \omega_i) - H_a(\boldsymbol{x}_k, \omega_i) \sin H_p(\boldsymbol{x}_k, \omega_i) \nabla_x H_p(\boldsymbol{x}_k, \omega_i)] \quad \ldots \\
& + \imath [\sin H_p(\boldsymbol{x}_k, \omega_i) \nabla_x H_a(\boldsymbol{x}_k, \omega_i) + H_a(\boldsymbol{x}_k, \omega_i) \cos H_p(\boldsymbol{x}_k, \omega_i) \nabla_x H_p(\boldsymbol{x}_k, \omega_i)]
\end{aligned}$$

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

$$\begin{aligned}
|H(\boldsymbol{x}_k, \omega_i)|^2 &= H(\boldsymbol{x}_k, \omega_i) H(\boldsymbol{x}_k, \omega_i)^\dagger \\
\nabla_x |H(\boldsymbol{x}_k, \omega_i)|^2 &= (\nabla_x H(\boldsymbol{x}_k, \omega_i)) H(\boldsymbol{x}_k, \omega_i)^\dagger + H(\boldsymbol{x}_k, \omega_i) (\nabla_x H(\boldsymbol{x}_k, \omega_i))^\dagger \\
&= 2\Re\nabla_x H(\boldsymbol{x}_k, \omega_i) \Re H(\boldsymbol{x}_k, \omega_i) + 2\Im\nabla_x H(\boldsymbol{x}_k, \omega_i) \Im H(\boldsymbol{x}_k, \omega_i)
\end{aligned}$$

where $\dagger$ represents the complex conjugate transpose. 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

$$\begin{aligned}
|H(\boldsymbol{x}_k, \omega_i)|^2 + (\boldsymbol{x} - \boldsymbol{x}_k)^\top \nabla_x |H(\boldsymbol{x}_k, \omega_i)|^2 &\geq H_{d,l}^2(\omega_i) \\
|H(\boldsymbol{x}_k, \omega_i)|^2 + (\boldsymbol{x} - \boldsymbol{x}_k)^\top \nabla_x |H(\boldsymbol{x}_k, \omega_i)|^2 &\leq H_{d,u}^2(\omega_i)
\end{aligned}$$

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

$$\begin{aligned}
\phi_H(\boldsymbol{x}_k, \omega_i) &= \arctan \frac{\Im H(\boldsymbol{x}_k, \omega_i)}{\Re H(\boldsymbol{x}_k, \omega_i)} \\
\nabla_x \phi_H(\boldsymbol{x}_k, \omega_i) &= \frac{\Re H(\boldsymbol{x}_k, \omega_i) \Im\nabla_x H(\boldsymbol{x}_k, \omega_i) - \Im H(\boldsymbol{x}_k, \omega_i) \Re\nabla_x H(\boldsymbol{x}_k, \omega_i)}{|H(\boldsymbol{x}_k, \omega_i)|^2}
\end{aligned}$$

## 9.3 Using the *SeDuMi* SOCP solver

In the following I use the *SeDuMi* (*Se*lf-*Du*al-*Mi*nimisation) SOCP solver originally written by *Jos Sturm* [193]. *Lu* [217, Section III] provides an example of expressing an optimisation problem in the form accepted by SeDuMi. In *Lu*'s notation the problem is:

$$\textbf{minimise} \quad \boldsymbol{b}^\top \boldsymbol{x} \tag{9.1a}$$

$$\textbf{subject to} \quad \|\boldsymbol{A}_i^\top \boldsymbol{x} + \boldsymbol{c}_i\| \leq \boldsymbol{b}_i^\top \boldsymbol{x} + d_i \quad \text{for } i = 1, \ldots, q \tag{9.1b}$$

$$\boldsymbol{D}^\top \boldsymbol{x} + \boldsymbol{f} \geq \boldsymbol{0} \tag{9.1c}$$

where $\boldsymbol{x} \in \mathbb{R}^{m\times 1}$, $\boldsymbol{b} \in \mathbb{R}^{m\times 1}$, $\boldsymbol{A}_i \in \mathbb{R}^{m\times(n_i-1)}$ [a], $\boldsymbol{c}_i \in \mathbb{R}^{(n_i-1)\times 1}$, $\boldsymbol{b}_i \in \mathbb{R}^{m\times 1}$, $d_i \in \mathbb{R}$ for $1 \leq i \leq q$, $\boldsymbol{D} \in \mathbb{R}^{m\times p}$ and $\boldsymbol{f} \in \mathbb{R}^{p\times 1}$. The problem is cast into SeDuMi format by defining

$$\begin{aligned}
\boldsymbol{A}_t &= \begin{bmatrix} -\boldsymbol{D} & \boldsymbol{A}_t^{(1)} \ldots & \boldsymbol{A}_t^{(q)} \end{bmatrix} \\
\boldsymbol{A}_t^{(i)} &= -\begin{bmatrix} \boldsymbol{b}_i & \boldsymbol{A}_i \end{bmatrix} \\
\boldsymbol{b}_t &= -\boldsymbol{b} \\
\boldsymbol{c}_t &= \begin{bmatrix} \boldsymbol{f}; & \boldsymbol{c}_t^{(1)}; & \ldots & \boldsymbol{c}_t^{(q)} \end{bmatrix} \\
\boldsymbol{c}_t^{(i)} &= \begin{bmatrix} d_i; & \boldsymbol{c}_i \end{bmatrix}
\end{aligned} \tag{9.2}$$

The Octave commands to solve the SOCP problem with SeDuMi are

```
K.l = 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 9.1c and $\boldsymbol{q}$ is a vector giving the dimensions of the $q$ sets of conic constraints in Equation 9.1b, $\boldsymbol{q} = \begin{bmatrix} n_1 & n_2 & \ldots & n_q \end{bmatrix}$. The Octave script *Lu_remarks_example_4_test.m* shows *Lu*'s Example 4 [217, Section III], an SOCP problem with linear and quadratic constraints.

## 9.4 An example of SOCP 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 8.2.3, with MMSE optimisation of the weighted response error by the *SeDuMi* SOCP solver. As in Part 8, 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 8.2.3, 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

$$\mathbf{x} = \begin{bmatrix} \epsilon \\ \beta \\ \boldsymbol{\delta} \end{bmatrix}$$

where $\epsilon$ is the MMSE error, $\boldsymbol{\delta}$ is the step direction from coefficent vector $\boldsymbol{x}_k$, and $\beta$ is the constraint on the coefficient step-size, $\|\boldsymbol{\delta}\| \leq \beta$.

In a similar fashion to the MMSE error used in Part 8, 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

$$\boldsymbol{A}_i^\top = \begin{bmatrix} \boldsymbol{0} & W_a(\omega_s)\nabla_x A(\boldsymbol{x}_k, \omega_a) \\ \boldsymbol{0} & W_t(\omega_t)\nabla_x T(\boldsymbol{x}_k, \omega_t) \\ \boldsymbol{0} & W_s(\omega_s)\nabla_x S(\boldsymbol{x}_k, \omega_s) \end{bmatrix}$$

[a] $n_i - 1$ to allow for the column $b_i$ in the matrix $\boldsymbol{A}_t^{(i)}$ and $d_i$ in the column vector $\boldsymbol{c}_t^{(i)}$

$$
\begin{aligned}
\boldsymbol{b}_i^\top &= \begin{bmatrix} 1 & 0 & \mathbf{0} \end{bmatrix} \\
\boldsymbol{c}_i &= \begin{bmatrix} W_a(\omega_a) & [A(\boldsymbol{x}_k, \omega_a) - A_d(\omega_a)] \\ W_t(\omega_t) & [T(\boldsymbol{x}_k, \omega_t) - T_d(\omega_t)] \\ W_s(\omega_s) & [S(\boldsymbol{x}_k, \omega_s) - S_d(\omega_s)] \end{bmatrix} \\
d_i &= 0
\end{aligned}
$$

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, $\boldsymbol{x}_u$ and $\boldsymbol{x}_l$ respectively, ensuring stability with

$$
\begin{aligned}
-(\boldsymbol{x} - \boldsymbol{x}_k) - (\boldsymbol{x}_k - \boldsymbol{x}_u) &\geq 0 \\
(\boldsymbol{x} - \boldsymbol{x}_k) - (\boldsymbol{x}_l - \boldsymbol{x}_k) &\geq 0
\end{aligned}
$$

The stability constraint is a linear constraint on the coefficients

$$
\begin{aligned}
\boldsymbol{D}^\top &= \begin{bmatrix} \mathbf{0} & -\boldsymbol{I} \\ \mathbf{0} & \boldsymbol{I} \end{bmatrix} \\
\boldsymbol{f} &= \begin{bmatrix} -(\boldsymbol{x}_k - \boldsymbol{x}_u) \\ -(\boldsymbol{x}_l - \boldsymbol{x}_k) \end{bmatrix}
\end{aligned}
$$

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

$$
\begin{aligned}
A_{du}(\boldsymbol{x}_k, \omega_{au}) - \left[A(\boldsymbol{x}_k, \omega_{au}) + \nabla_x A(\boldsymbol{x}_k, \omega_{au})^\top \boldsymbol{\delta}\right] &\geq 0 \\
-A_{dl}(\boldsymbol{x}_k, \omega_{al}) + \left[A(\boldsymbol{x}_k, \omega_{al}) + \nabla_x A(\boldsymbol{x}_k, \omega_{al})^\top \boldsymbol{\delta}\right] &\geq 0
\end{aligned}
$$

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 8.1.

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 8.2.3. The filter specification is

```
n=500 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-05 % Tolerance on constraints
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 % 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 9.1 with pass-band details shown in Figure 9.2 and pole-zero plot shown in Figure 9.3.

The estimate of the SOCP PCLS optimised filter vector is, in the gain, zeros and poles form of Equation 8.2:

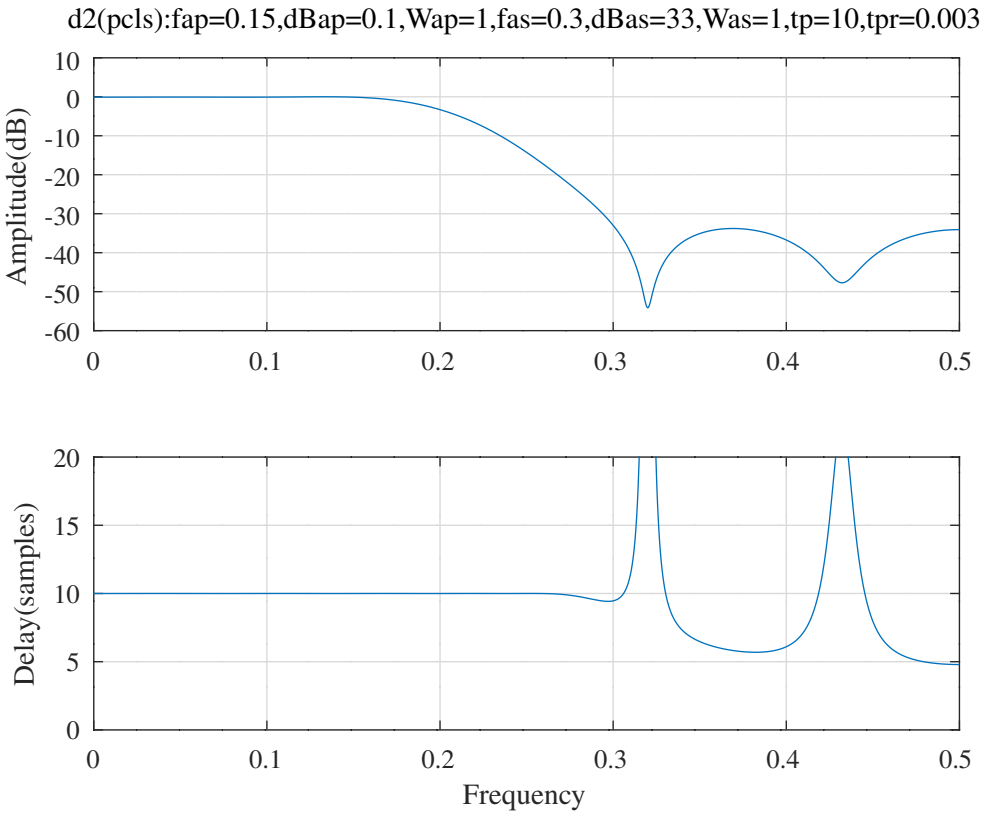



Figure 9.1: Deczky Example 3 PCLS:Response after SOCP optimisation

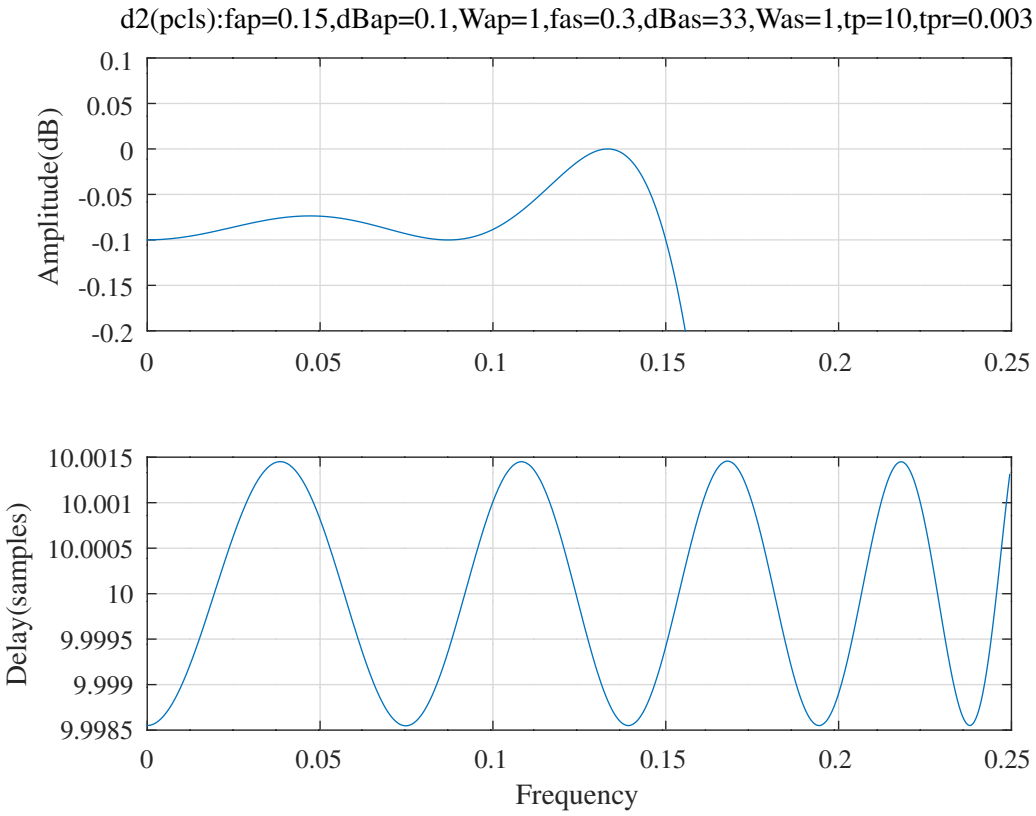



Figure 9.2: Deczky Example 3 PCLS:Passband response after SOCP optimisation

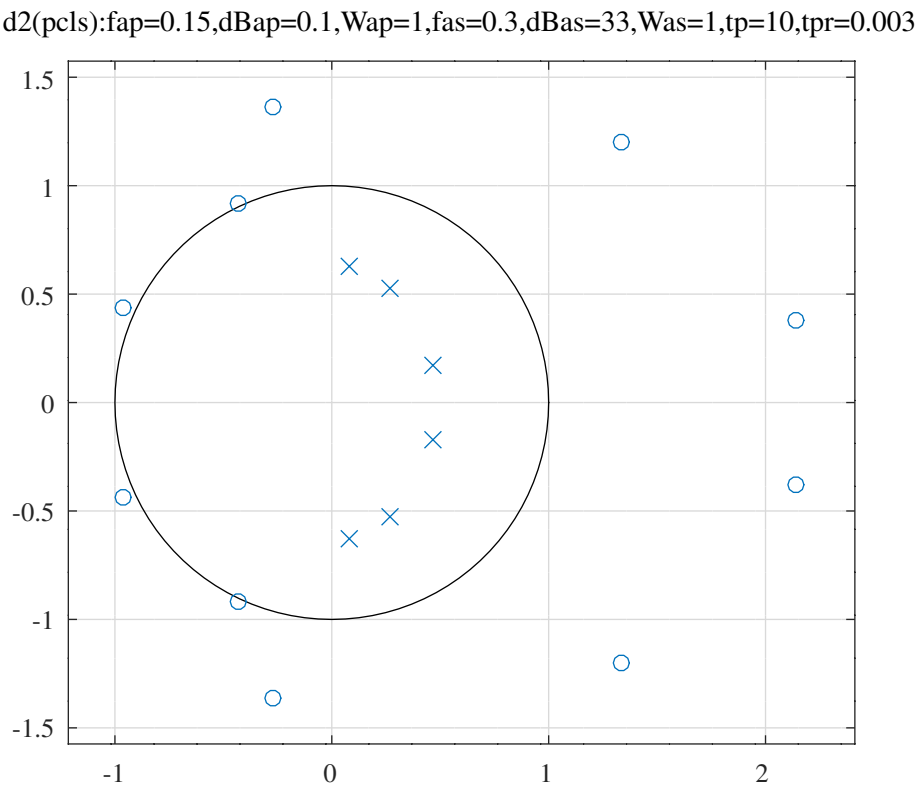



Figure 9.3: Deczky Example 3 PCLS:Pole-zero plot after SOCP optimisation

```
Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [   0.0034352390, ...
         1.0142167915,   1.0568423581,   1.3897353151,   1.7955610908, ...
         2.1734118934, ...
         2.0101283605,   2.7152949030,   1.7674544856,   0.7324611712, ...
         0.1753051231, ...
         0.4968946423,   0.5911774015,   0.6332304070, ...
         0.3523987661,   1.0993086491,   1.4431872258 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N2 = [   0.0034352390,  -0.0124353690,   0.0119557516,  -0.0064774001, ...
         0.0193257438,  -0.0112886817,  -0.0345263836,  -0.0055423202, ...
         0.1036105933,   0.1287076613,   0.1160880770 ]';
```

and

```
D2 = [   1.0000000000,  -1.6309320112,   1.7351691218,  -1.2247748845, ...
         0.6001977822,  -0.1977871000,   0.0346009321 ]';
```

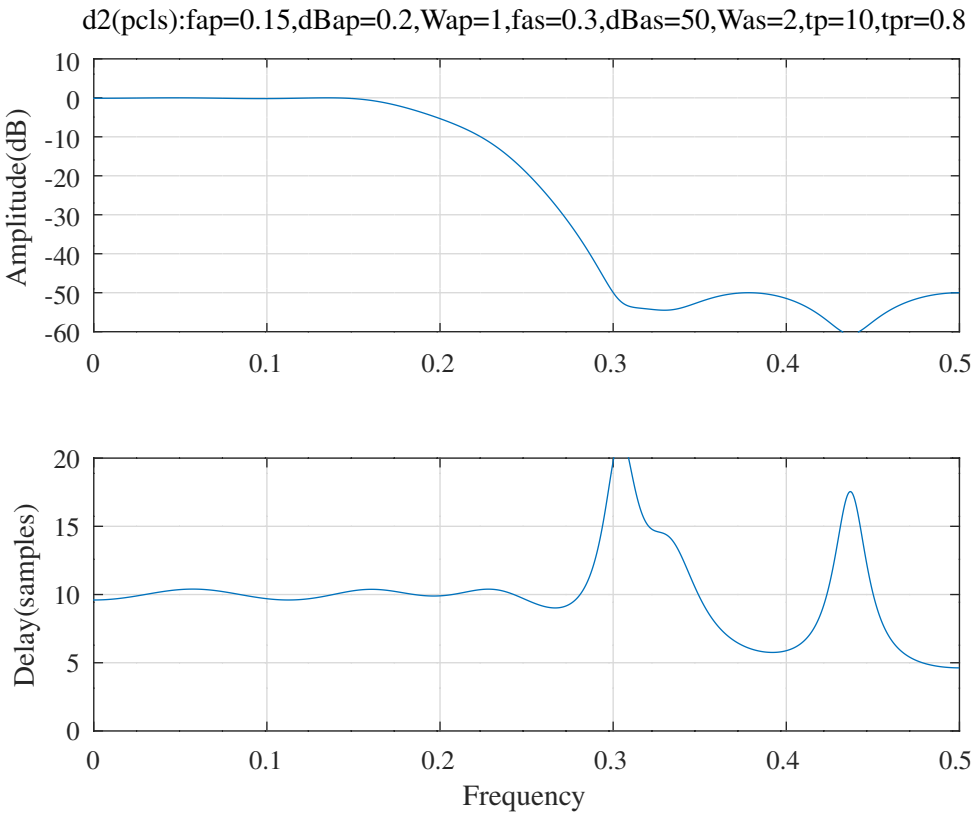



Figure 9.4: Deczky Example 3 PCLS:Response after SOCP optimisation, alternative specification

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
ctol=2e-05 % Tolerance on constraints
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=0.8 % 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=2 % 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 9.4 with pass-band details shown in Figure 9.5 and pole-zero plot shown in Figure 9.6. The estimate of the SOCP PCLS optimised filter vector is, in the gain, zeros and poles form of Equation 8.2:

```
Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [   0.0024085237, ...
         1.0757207923,   1.0770594716,   1.1394947756,   1.6132256567, ...
         2.1416452439, ...
         1.9101686254,   2.7459482411,   2.0914338105,   0.3770820547, ...
         1.0592438580, ...
         0.6699523174,   0.7374149832,   0.7784073816, ...
         0.3559820006,   1.0082977483,   1.4625220543 ]';
```

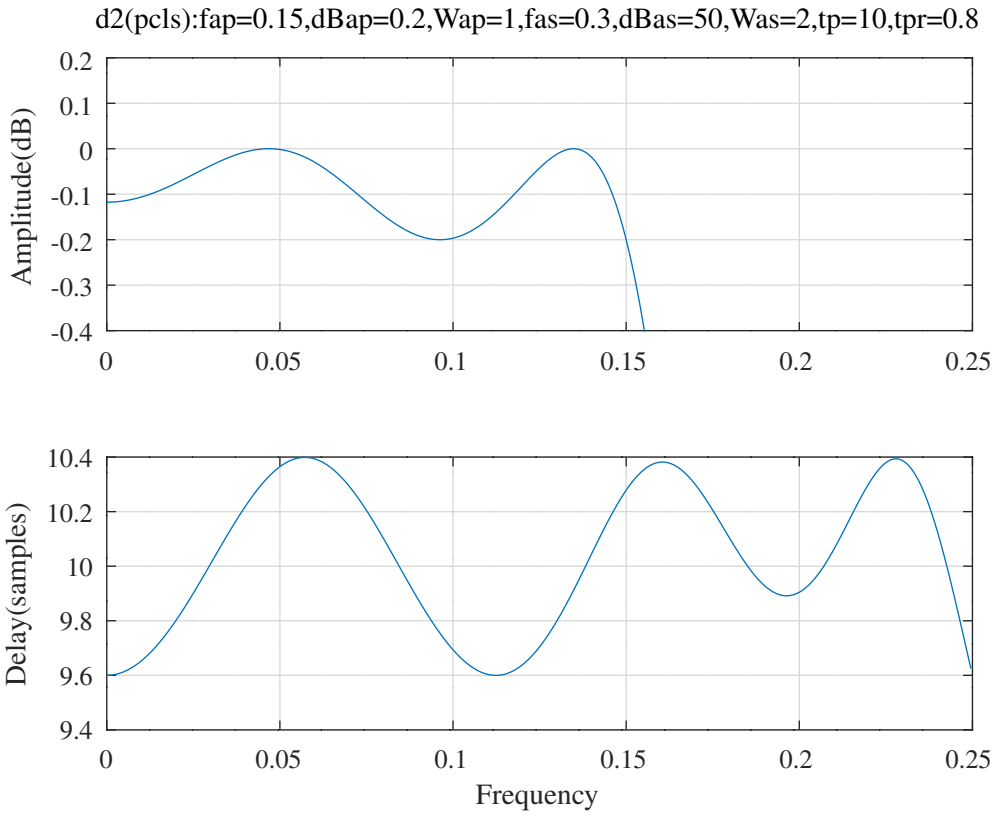



Figure 9.5: Deczky Example 3 PCLS:Passband response after SOCP optimisation, alternative specification

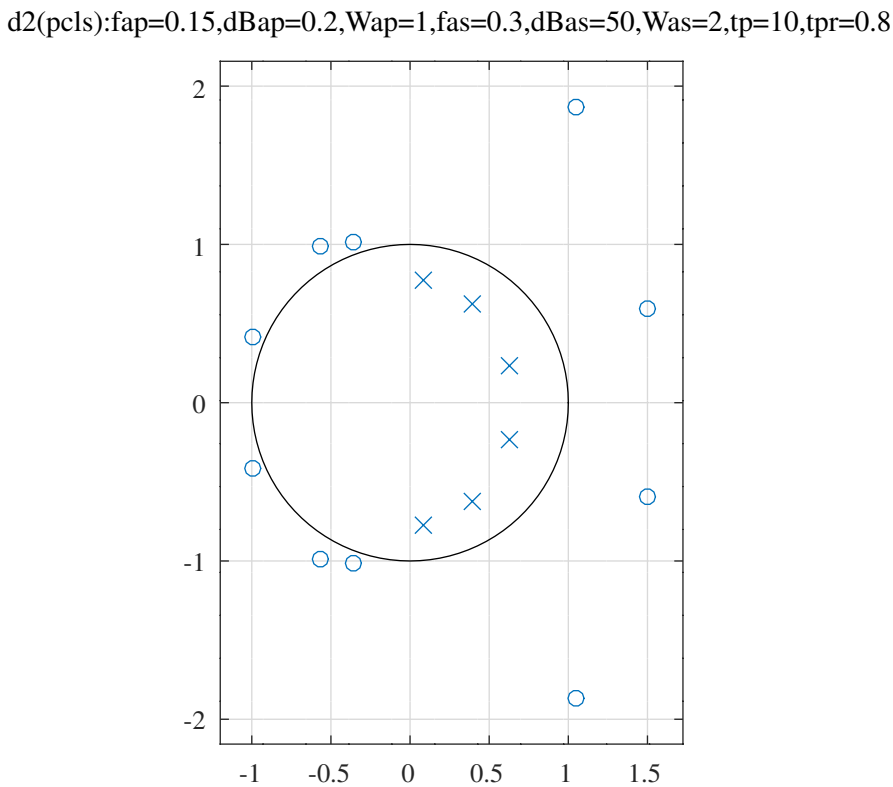



Figure 9.6: Deczky Example 3 PCLS:Pole-zero plot after SOCP optimisation, alternative specification

In Section 8.1.1, I expressed the filter design problem as a *sequential quadratic programming*, or SQP, problem:

$$\textbf{minimise} \quad q(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{a}^\top \boldsymbol{x} + \beta$$

where $\boldsymbol{Q}$ 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 $q(\boldsymbol{x})$:

$$q(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{Q}^{\frac{1}{2}}\boldsymbol{x} + \boldsymbol{Q}^{-\frac{1}{2}}\boldsymbol{a}\|^2 + \beta - \frac{1}{2}\boldsymbol{a}^\top \boldsymbol{Q}^{-1}\boldsymbol{a}$$

See *Alizadeh* and *Goldfarb* [45, 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 9.4, but with MMSE optimisation of the weighted response error, $\mathcal{E}(\boldsymbol{x})$, represented as $\|\boldsymbol{Q}^{\frac{1}{2}}\boldsymbol{x} + \boldsymbol{Q}^{-\frac{1}{2}}\nabla_x \mathcal{E}(\boldsymbol{x})\|$, where $\boldsymbol{Q}$ is initialised with the diagonal of $\nabla^2_{xx}\mathcal{E}(\boldsymbol{x})$ and updated with the BFGS Hessian matrix update algorithm shown in Appendix K.7.1.

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
ctol=1e-06 % Tolerance on constraints
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 9.7 with pass-band details shown in Figure 9.8 and pole-zero plot shown in Figure 9.9.

The estimate of the SOCP PCLS optimised filter vector is, in the gain, zeros and poles form of Equation 8.2:

```
Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [   0.0028603111, ...
         1.1153479370,   1.1326276048,   1.1339470248,   1.8249200328, ...
         2.1304862453, ...
         2.7293066409,   2.0494174580,   1.8375912877,   0.7412308781, ...
         0.2300938324, ...
         0.5590032117,   0.6426477932,   0.7101803655, ...
         0.3514326070,   1.0470003164,   1.4478415767 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N2 = [   0.0028603111,  -0.0090270543,   0.0046356931,   0.0042885497, ...
         0.0125876510,  -0.0167148545,  -0.0287899470,   0.0082897177, ...
         0.0946954276,   0.1063564366,   0.0887235618 ]';
```

and

```
D2 = [   1.0000000000,  -1.8667421650,   2.1994797792,  -1.7319712394, ...
         0.9458082016,  -0.3424437920,   0.0650896550 ]';
```

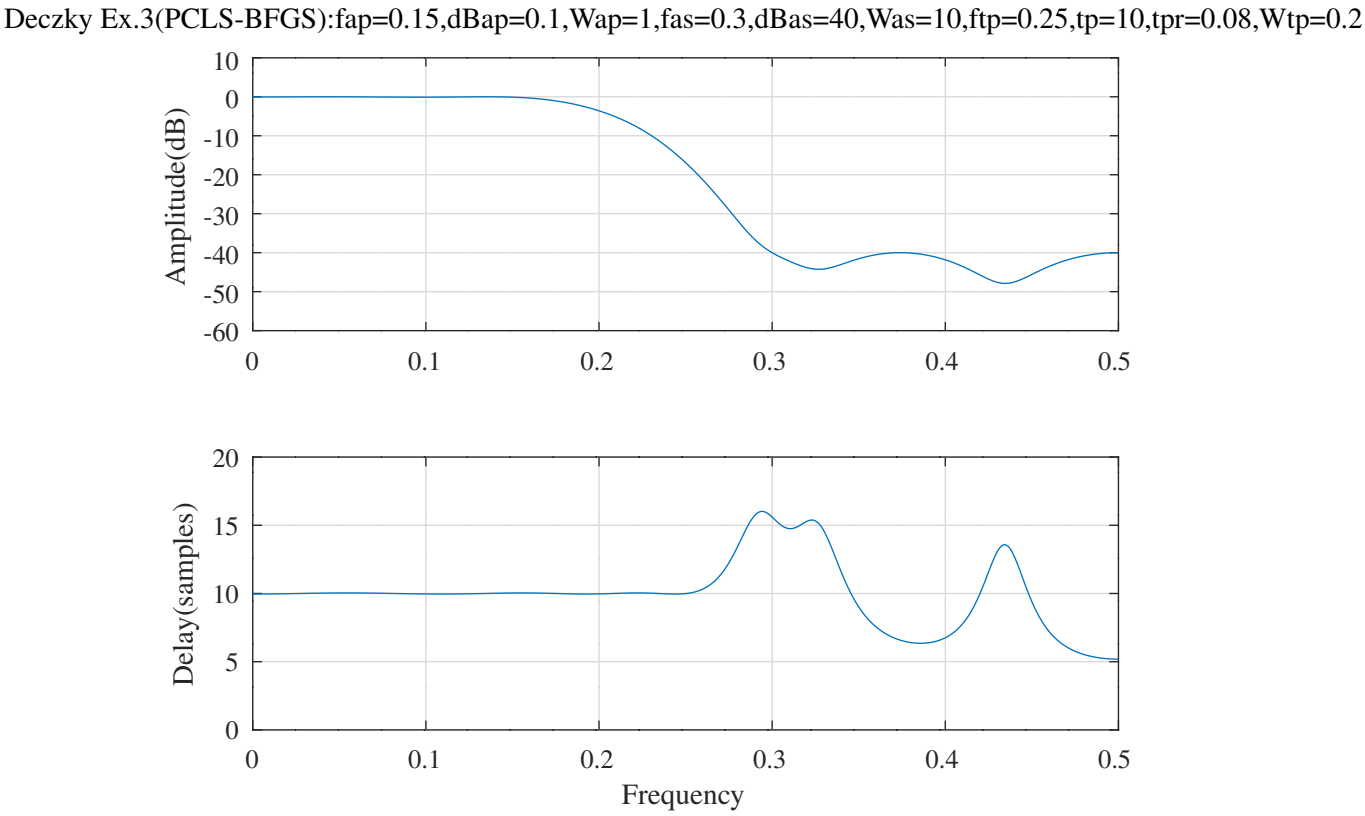



Figure 9.7: Deczky Example 3 PCLS:Response after SOCP optimisation with BFGS update of the Hessian

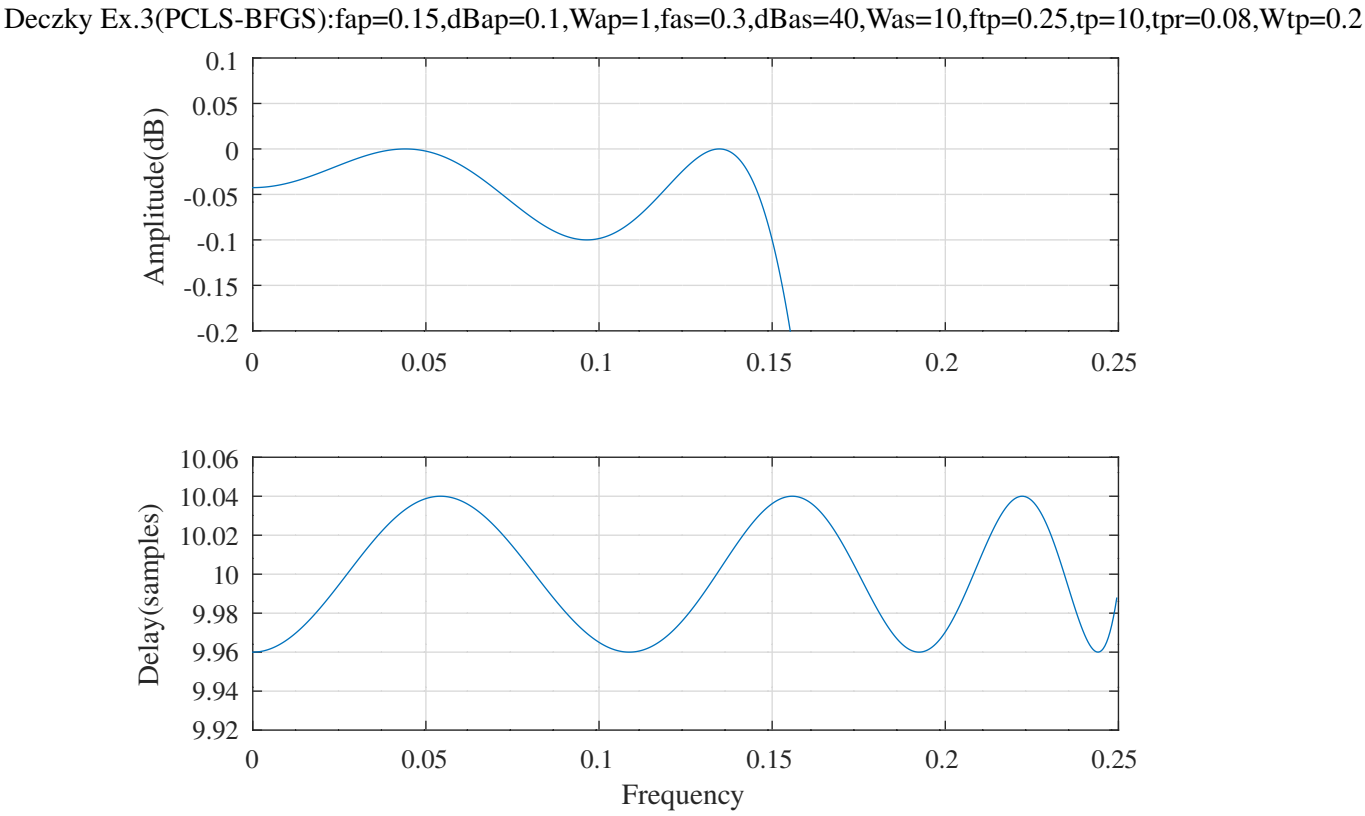



Figure 9.8: Deczky Example 3 PCLS:Passband response after SOCP optimisation with BFGS update of the Hessian

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

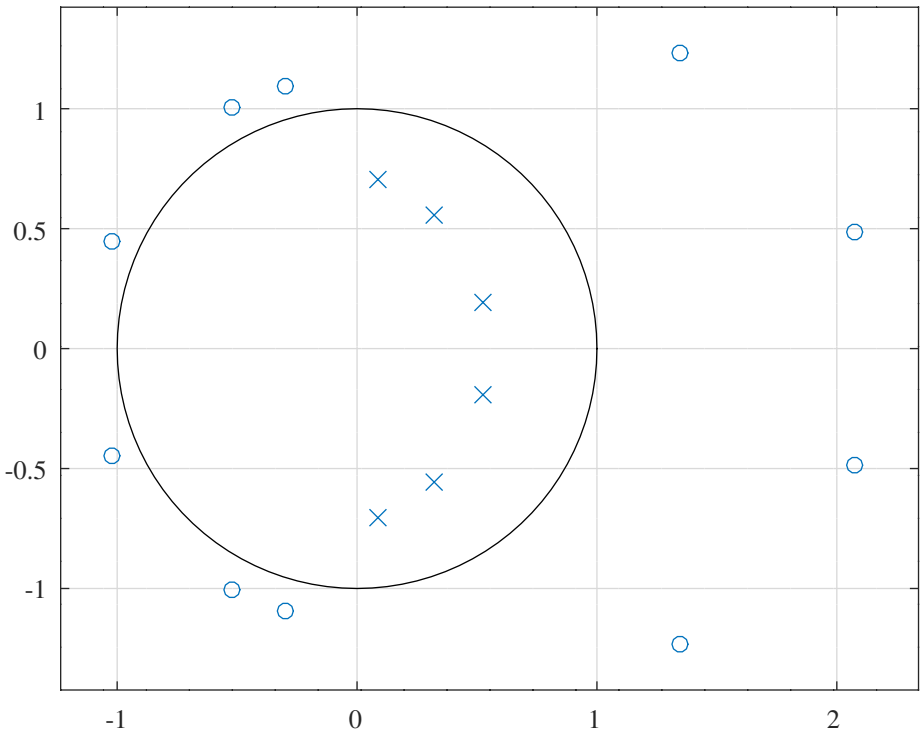



Figure 9.9: Deczky Example 3 PCLS:Pole-zero plot after SOCP optimisation with BFGS update of the Hessian

## 9.5 SOCP design of a non-linear phase FIR low-pass filter

The Octave script *iir_socp_slb_fir_lowpass_test.m* designs an FIR low-pass filter that has approximately linear phase in the pass-band. The impulse response is *not* symmetric. The filter specification is:

```
U=2 % Number of real zeros
V=0 % Number of real poles
M=54 % Number of complex zeros
Q=0 % Number of complex poles
R=1 % Denominator polynomial decimation factor
n=1000 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-06 % Tolerance on constraints
fap=0.15 % Pass band amplitude response edge
dBap=0.5 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band amplitude weight
Wat=0.0001 % Transition band amplitude weight
ftp=0.15 % Pass band group-delay response edge
td=15 % Pass band group-delay
tdr=0.3 % Pass band amplitude peak-to-peak ripple
Wtp=0.01 % Pass band group-delay weight
fas=0.2 % Stop band amplitude response edge
dBas=50 % Stop band minimum attenuation
Was=1 % Stop band amplitude weight
```

Figure 9.10 shows the response of the PCLS optimised filter. Figure 9.11 shows the pass-band detail of the response of 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=54,Qd1=0,Rd1=1
d1 = [  0.00183851, ...
       -1.16374161,  0.86741438, ...
        0.86794029,  0.87007424,  0.87081646,  0.87271561, ...
        0.87279406,  0.88806778,  0.93790938,  0.94529525, ...
        0.95353382,  0.95607409,  0.96403099,  0.96949828, ...
        0.97335195,  0.97615667,  0.97827229,  0.97992797, ...
        0.98129199,  0.98250069,  0.98369363,  0.98449733, ...
        0.98507458,  0.98695650,  0.98986785,  0.99331319, ...
        0.99866275,  1.33714281,  1.35104526, ...
        0.47289025,  0.31206720,  0.15947848,  0.78063678, ...
        0.62941578,  0.94160146,  1.49199731,  1.59212830, ...
        1.40442980,  1.70032378,  1.80330111,  1.90661195, ...
        2.01087499,  2.11604038,  2.22192837,  2.32836587, ...
        2.43519154,  2.54227325,  2.64948570,  1.32286715, ...
        2.75673309,  2.86397589,  2.97139021,  3.07844211, ...
        1.26478671,  0.66461325,  0.22194853 ]';
```

and in transfer function form the FIR polynomial is:

```
N1 = [  0.00183851,  0.00669621,  0.00984563,  0.00489654, ...
       -0.00706682, -0.01450199, -0.00581601,  0.01532410, ...
        0.02709525,  0.00816303, -0.03460744, -0.05781442, ...
       -0.01074537,  0.11514996,  0.26190160,  0.33750110, ...
        0.28616903,  0.13701569, -0.01506487, -0.08306145, ...
       -0.05276943,  0.01654800,  0.05191133,  0.02912864, ...
       -0.01716363, -0.03740210, -0.01656447,  0.01784173, ...
        0.03003792,  0.01244799, -0.01178596, -0.01724316, ...
       -0.00219777,  0.01372972,  0.01331001, -0.00085226, ...
       -0.01215527, -0.00939889,  0.00244786,  0.00990545, ...
        0.00616402, -0.00308752, -0.00761772, -0.00380996, ...
        0.00294169,  0.00541355,  0.00212080, -0.00244454, ...
       -0.00359291, -0.00106225,  0.00182173,  0.00225987, ...
        0.00044622, -0.00150145, -0.00205694, -0.00138083, ...
       -0.00045085 ]';
```

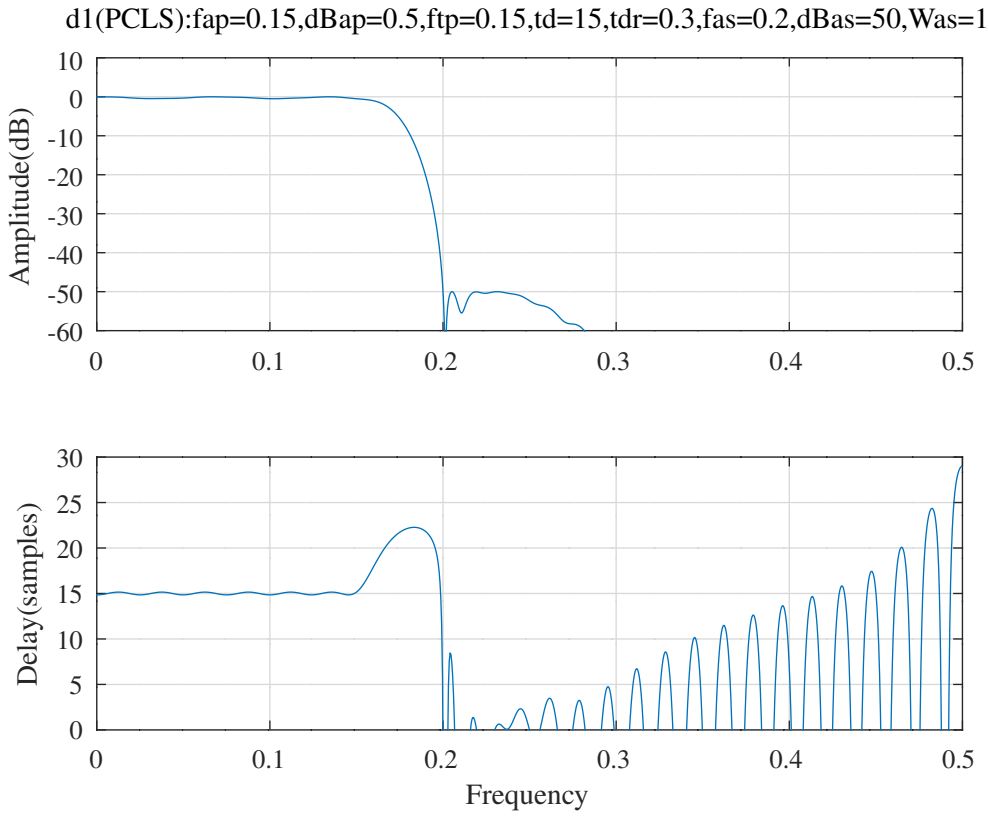



Figure 9.10: Response of the non-linear phase FIR low-pass filter after SOCP PCLS optimisation

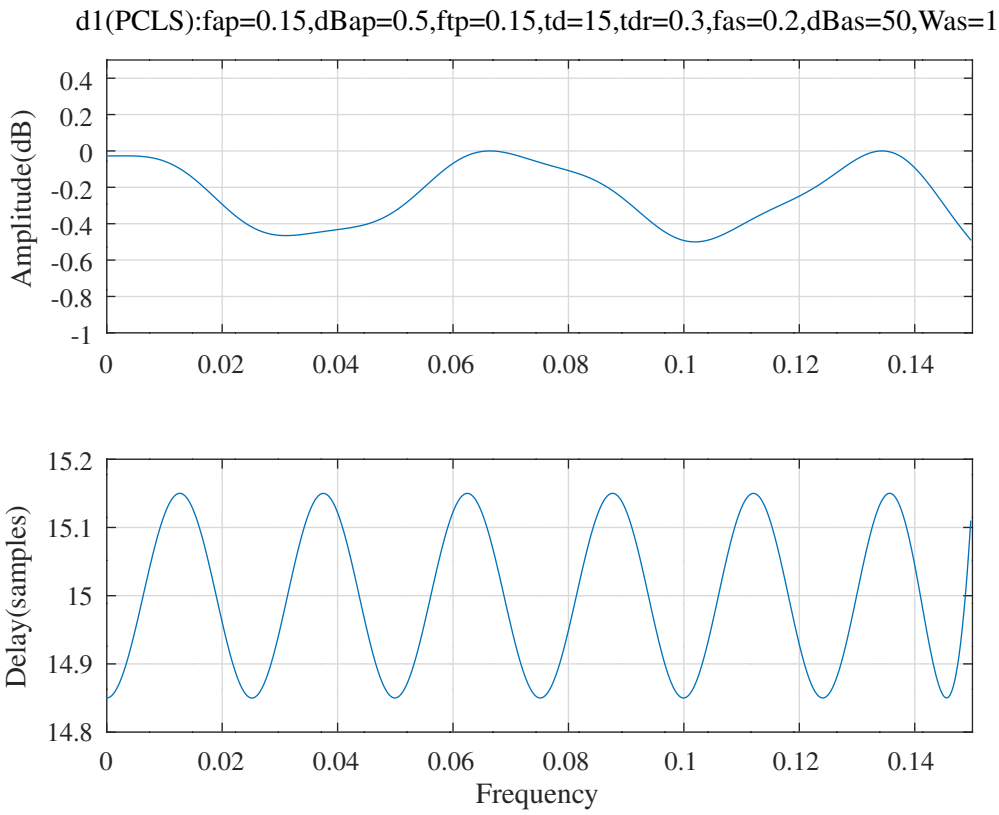



Figure 9.11: Pass-band response of the non-linear phase FIR low-pass filter after SOCP PCLS optimisation

The Octave script *iir_socp_slb_fir_lowpass_alternate_test.m* designs an alternative minimum-phase FIR low-pass filter that has approximately linear phase in the pass-band. The filter specification is:

```
U=2 % Number of real zeros
V=0 % Number of real poles
M=28 % Number of complex zeros
Q=0 % Number of complex poles
R=1 % Denominator polynomial decimation factor
n=500 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-06 % Tolerance on constraints
fap=0.15 % Pass band amplitude response edge
dBap=3 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band amplitude weight
Wat=1e-06 % Transition band amplitude weight
ftp=0.14 % Pass band group-delay response edge
td=10 % Pass band group-delay
tdr=0.6 % Pass band amplitude peak-to-peak ripple
Wtp=0.25 % Pass band group-delay weight
fas=0.2 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=10 % Stop band amplitude weight
```

Figure 9.12 shows the response of the PCLS optimised filter. Figure 9.13 shows the pass-band detail of the response of the PCLS optimised filter. The coefficients of the PCLS optimised alternative minimum phase FIR bandpass filter are, in gain-pole-zero form:

```
Ud1=2,Vd1=0,Md1=28,Qd1=0,Rd1=1
d1 = [  0.00681003, ...
       -0.96180167,  2.23811567, ...
        0.79806375,  0.80483819,  0.80647914,  0.94358388, ...
        0.94519296,  0.94677232,  0.94771620,  0.94986533, ...
        0.95006793,  0.95599483,  0.96348243,  0.97994706, ...
        0.99656829,  1.67869267, ...
        0.16045020,  0.47500129,  0.81349574,  2.54423808, ...
        2.34406358,  2.74377934,  2.14458990,  2.94177249, ...
        1.93894795,  1.74561483,  1.55778836,  1.38974285, ...
        1.27394369,  0.40169843 ]';
```

and in transfer function form the FIR polynomial is:

```
N1 = [  0.00681003, -0.00618991, -0.01820610, -0.01586388, ...
        0.00818548,  0.04390017,  0.05259741, -0.00253328, ...
       -0.12057279, -0.24809988, -0.31085280, -0.26689832, ...
       -0.14161853, -0.00985159,  0.05960123,  0.05184022, ...
        0.00504794, -0.02856534, -0.02611915, -0.00249580, ...
        0.01366247,  0.01014745, -0.00315715, -0.00952879, ...
       -0.00389927,  0.00539110,  0.00765574,  0.00190038, ...
       -0.00540024, -0.00765331, -0.00467777 ]';
```

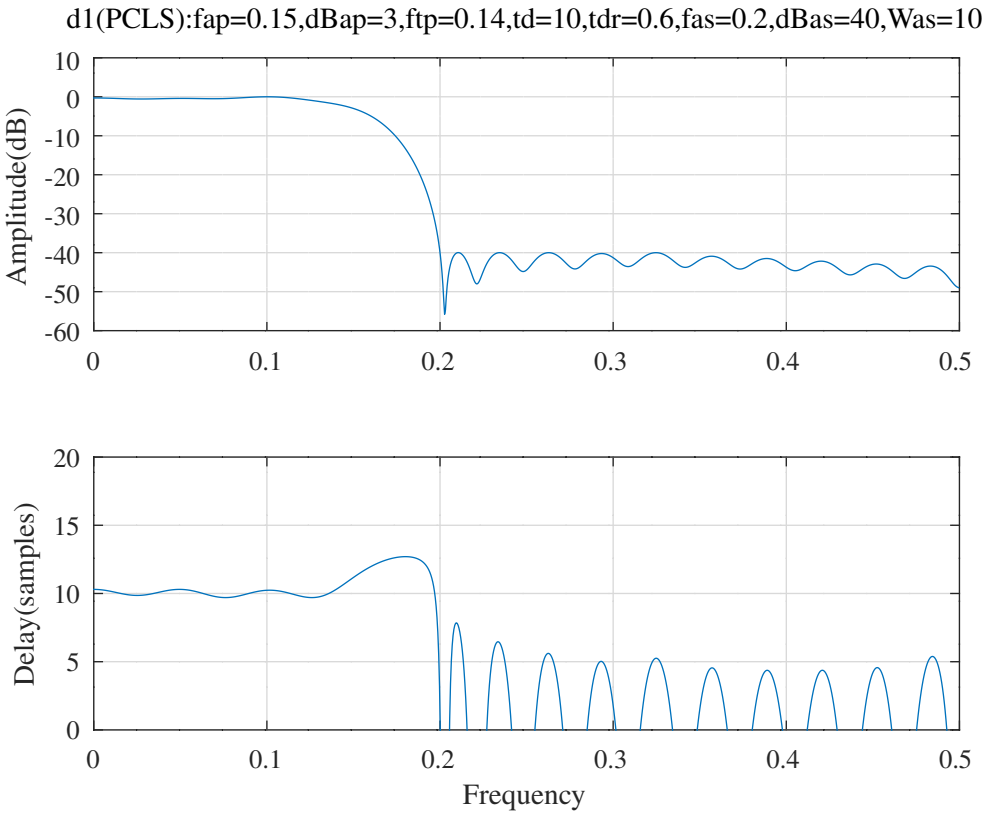



Figure 9.12: Response of the alternative non-linear phase FIR low-pass filter after SOCP PCLS optimisation

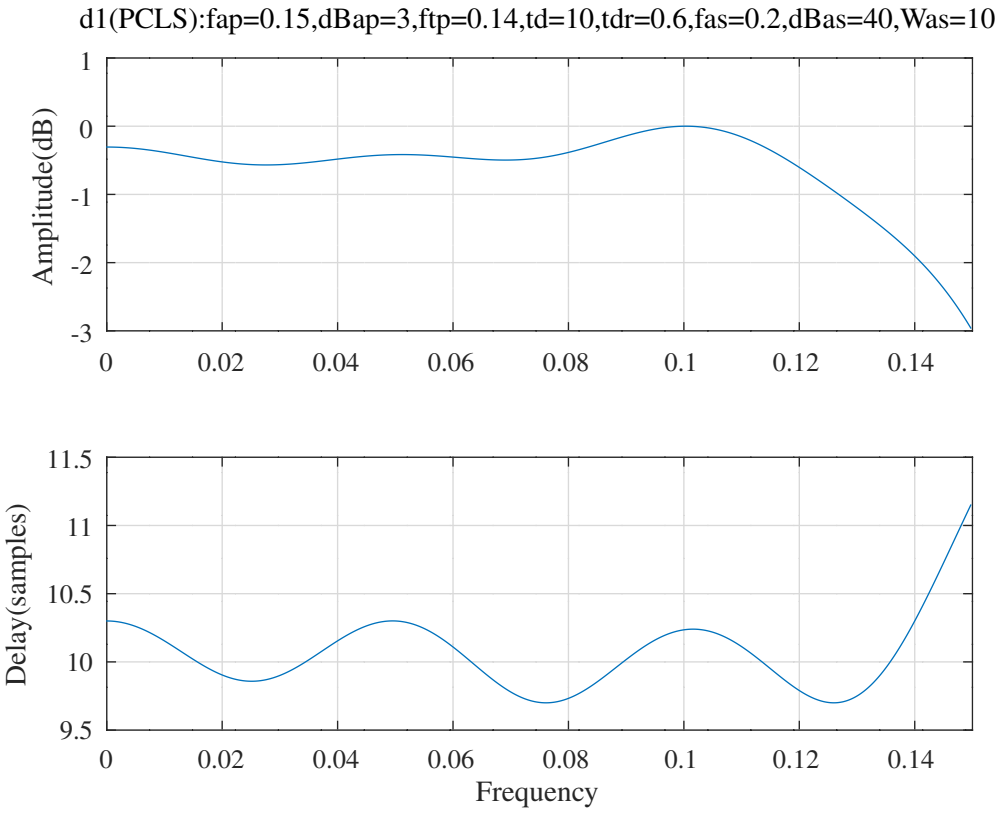



Figure 9.13: Pass-band response of the alternative non-linear phase FIR low-pass filter after SOCP PCLS optimisation

## 9.6 Comparison of FIR and IIR low-pass filters having approximately flat pass-band group delay with symmetric FIR filters

The Octave script *compare_fir_iir_socp_slb_lowpass_test.m* compares the performance of FIR and IIR implementations of a low-pass filter having 31 coefficients and a pass-band group delay of approximately 10 samples designed by the *SeDuMi* solver with symmetric FIR low-pass filters having 21 and 31 coefficients designed by the Octave *remez* function. The low-pass filter pass-band and stop-band edges are 0.15 and 0.2, respectively, and the allowed pass-band ripple is $3dB$. The IIR filter specification is:

```
U=1 % Number of real zeros
V=1 % Number of real poles
M=14 % Number of complex zeros
Q=14 % Number of complex poles
R=1 % Denominator polynomial decimation factor
n=1000 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-06 % Tolerance on constraints
rho=0.992188 % Constraint on pole radius
fap=0.15 % Pass band amplitude response edge
dBap=1 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band amplitude weight
Wat=0.001 % Transition band amplitude weight
ftp=0.15 % Pass band group-delay response edge
td=10 % Pass band group-delay
tdr=0.2 % Pass band amplitude peak-to-peak ripple
Wtp=0.1 % Pass band group-delay weight
Wtt=0.001 % Transition band group-delay weight
fas=0.2 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=10 % Stop band amplitude weight
```

The *gain-zero-pole* FIR filter specification is:

```
U=2 % Number of real zeros
V=0 % Number of real poles
M=28 % Number of complex zeros
Q=0 % Number of complex poles
R=1 % Denominator polynomial decimation factor
n=1000 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-06 % Tolerance on constraints
fap=0.15 % Pass band amplitude response edge
dBap=3 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band amplitude weight
ftp=0.15 % Pass band group-delay response edge
td=10 % Pass band group-delay
tdr=0.8 % Pass band amplitude peak-to-peak ripple
Wtp=0.002 % Pass band group-delay weight
fas=0.2 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=50 % Stop band amplitude weight
```

The symmetric FIR filters were designed with:

```
d10=remez(20,[0 0.15 0.2 0.5]*2,[1 1 0 0],[1 7]);
```

and

```
d15=remez(30,[0 0.15 0.2 0.5]*2,[1 1 0 0],[1 50]);
```

Figure 9.14 compares the amplitude responses of the four filters and Figure 9.15 compares the delay responses (the flat delay of 15 samples of the order 30 symmetric FIR filter is not shown).

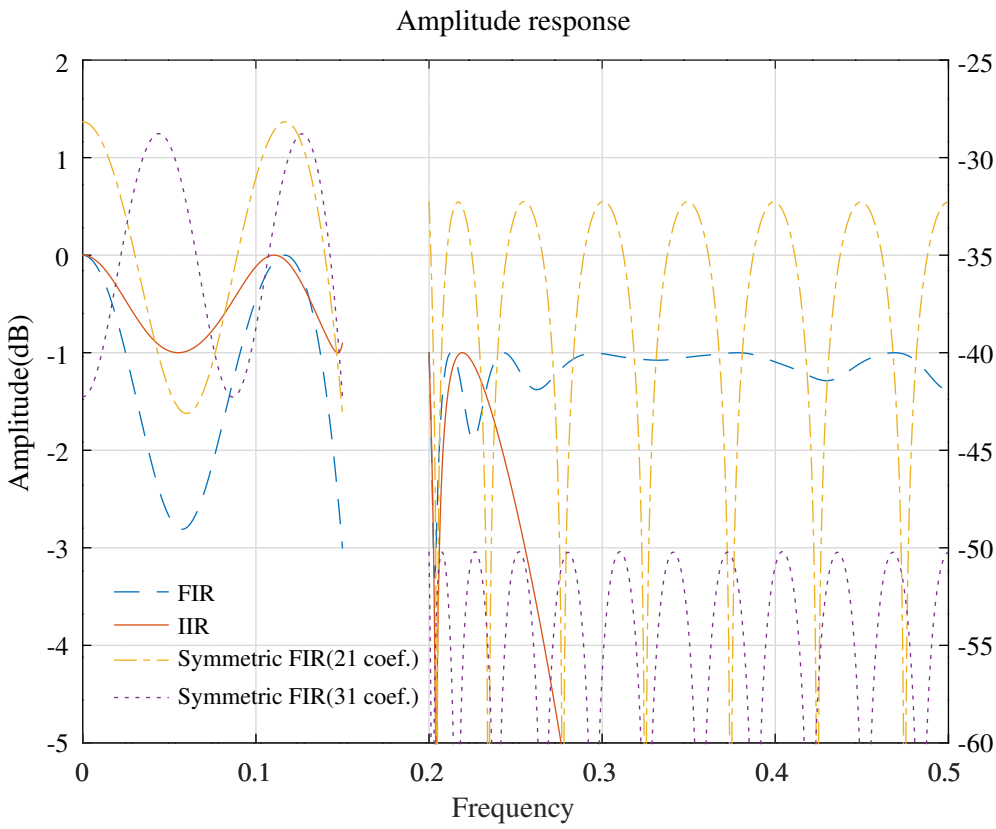



Figure 9.14: Comparison of FIR and IIR low-pass filters:Amplitude responses

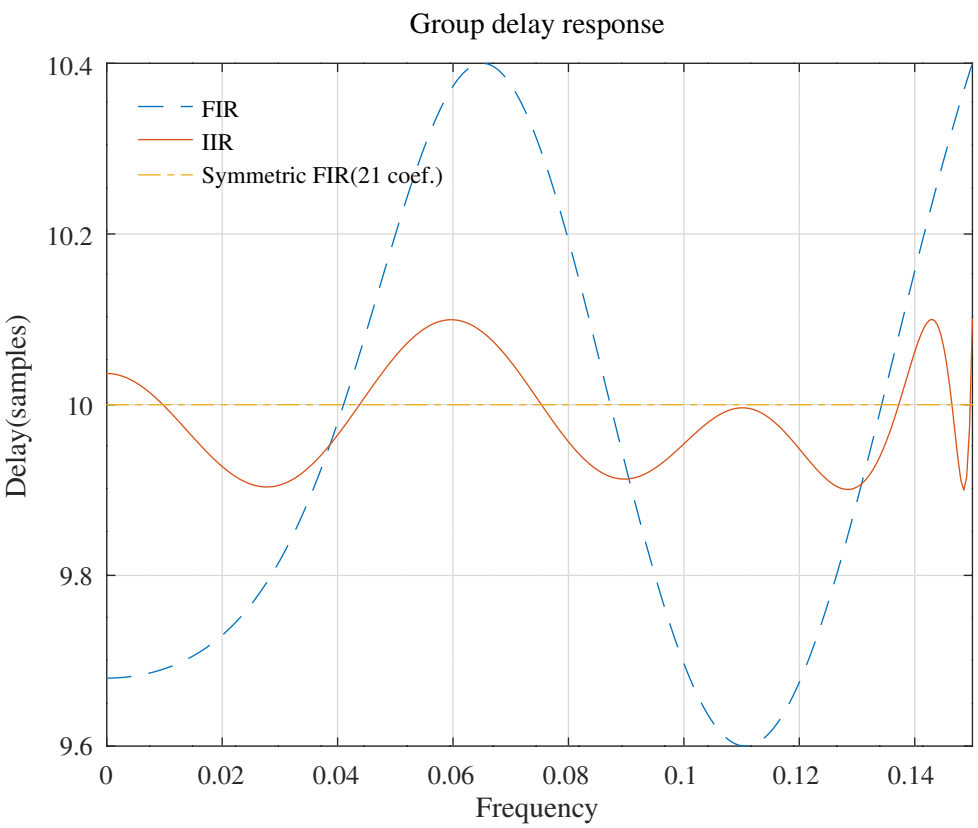



Figure 9.15: Comparison of FIR and IIR low-pass filters:Group delay responses

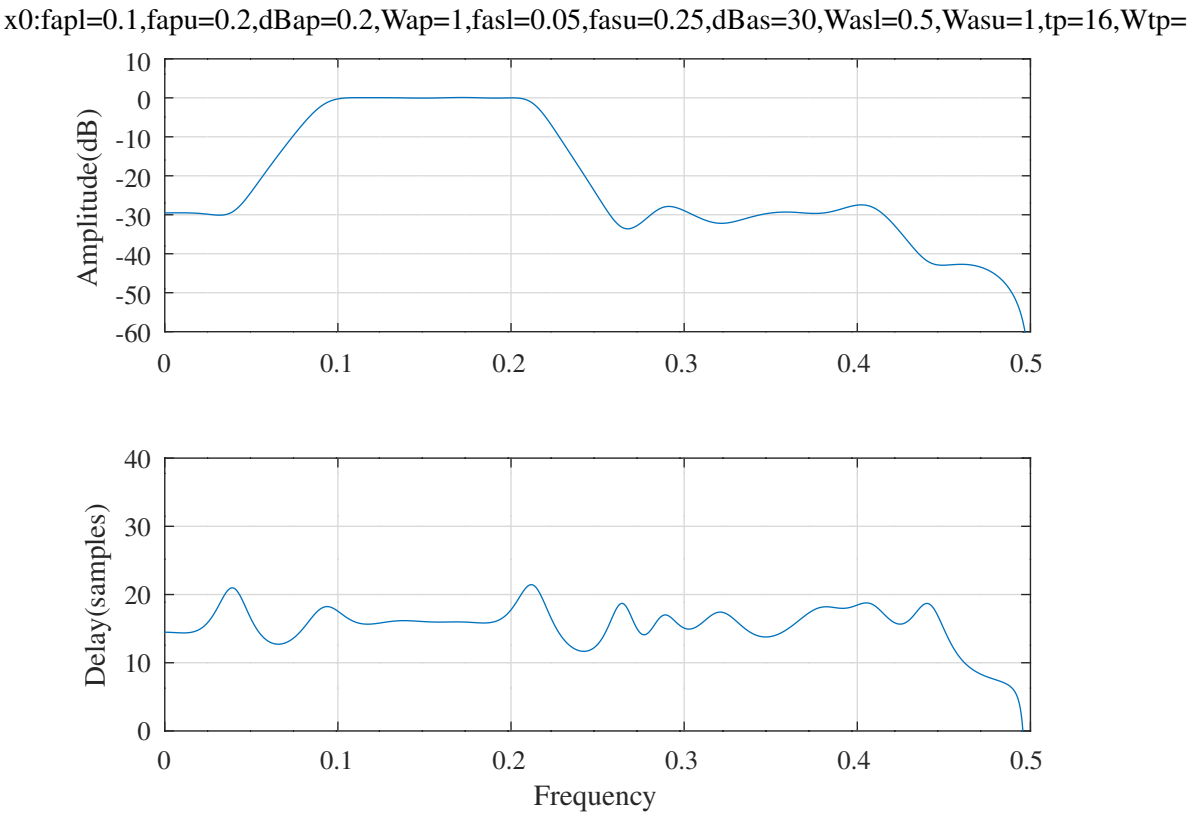



Figure 9.16: IIR band-pass R=2 decimation filter : response of the initial filter calculated with the WISE method of *Tarcynzki et al.*

## 9.7 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 8.2.6. After some experimentation, the filter specification is:

```
n=500 % Frequency points across the band
tol=0.0001 % Tolerance on relative coefficient update size
ctol=0.0001 % Tolerance on constraints
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=0.2 % Pass band amplitude peak-to-peak ripple
dBas=30 % Stop band amplitude peak-to-peak ripple
Wasl=0.5 % Lower stop band weight
Wap=1 % Pass band weight
Wasu=1 % Upper stop band weight
ftpl=0.09 % Pass band group delay response lower edge
ftpu=0.21 % Pass band group delay response upper edge
tp=16 % Nominal filter group delay
tpr=0.032 % Pass band group delay peak-to-peak ripple
Wtp=1 % Pass band group delay weight
U=2 % Number of real zeros
V=2 % Number of real poles
M=18 % Number of complex zeros
Q=8 % 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 9.16. After PCLS SOCP optimimisation, the response of the resulting filter is shown in Figure 9.17 with pass-band

detail shown in Figure 9.18. The corresponding pole-zero plot is shown in Figure 9.19. The estimate of the optimised filter vector is, in the gain, zeros and poles form of Equation 8.2:

```
Ud1=2,Vd1=2,Md1=18,Qd1=8,Rd1=2
d1 = [   0.0163061564, ...
        -1.0642007959,   0.9743301667, ...
        -0.6781850251,  -0.5472992260, ...
         0.9917467397,   0.9926851198,   1.0001611212,   1.0030549524, ...
         1.0734655894,   1.0830114950,   1.1205558252,   1.2829654040, ...
         1.2903179219, ...
         1.9272058003,   0.2806196143,   1.7180771221,   1.5900734378, ...
         2.4751347334,   2.1911788196,   2.6644259872,   0.7779719524, ...
         1.1194500892, ...
         0.6035307759,   0.6143246310,   0.6319038121,   0.7281844175, ...
         1.9135840355,   1.2865743040,   2.6025854213,   1.0060693680 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0163061564,   0.0194128772,   0.0314672510,   0.0301273458, ...
         0.0514102810,   0.0427589464,   0.0393053668,  -0.0007941423, ...
        -0.0096551207,  -0.0363761262,  -0.0604890757,  -0.1058050286, ...
        -0.1054034693,   0.0014820789,   0.1251443973,   0.2141167043, ...
         0.1213507122,   0.0016006900,  -0.1363178413,  -0.1035421953, ...
        -0.0767057714 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,   1.5918642281,   0.0000000000, ...
         1.5248725643,   0.0000000000,   1.4799534204,   0.0000000000, ...
         1.2746646990,   0.0000000000,   0.9144913686,   0.0000000000, ...
         0.5686124990,   0.0000000000,   0.3016010251,   0.0000000000, ...
         0.1350285582,   0.0000000000,   0.0513041233,   0.0000000000, ...
         0.0108032006 ]';
```

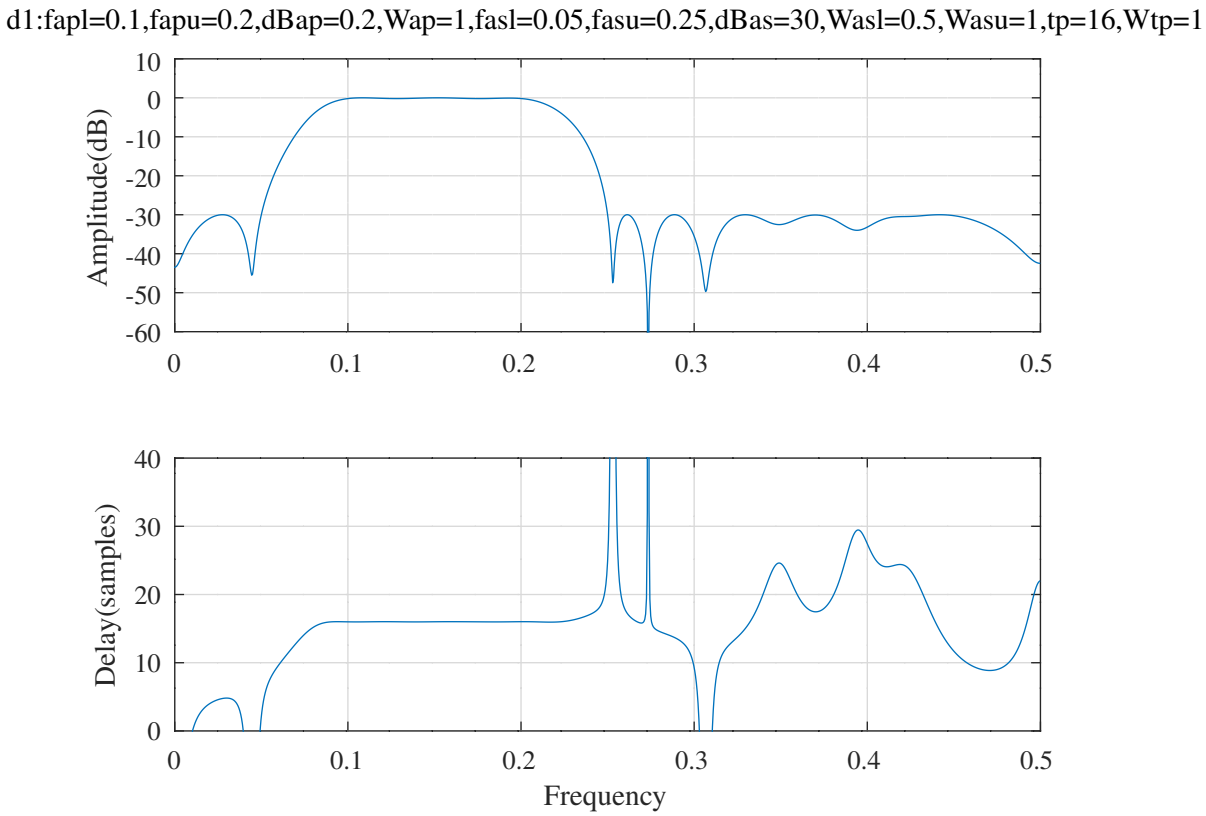



Figure 9.17: IIR band-pass R=2 decimation filter with delay tp=16 samples : response of the filter after PCLS SOCP optimisation

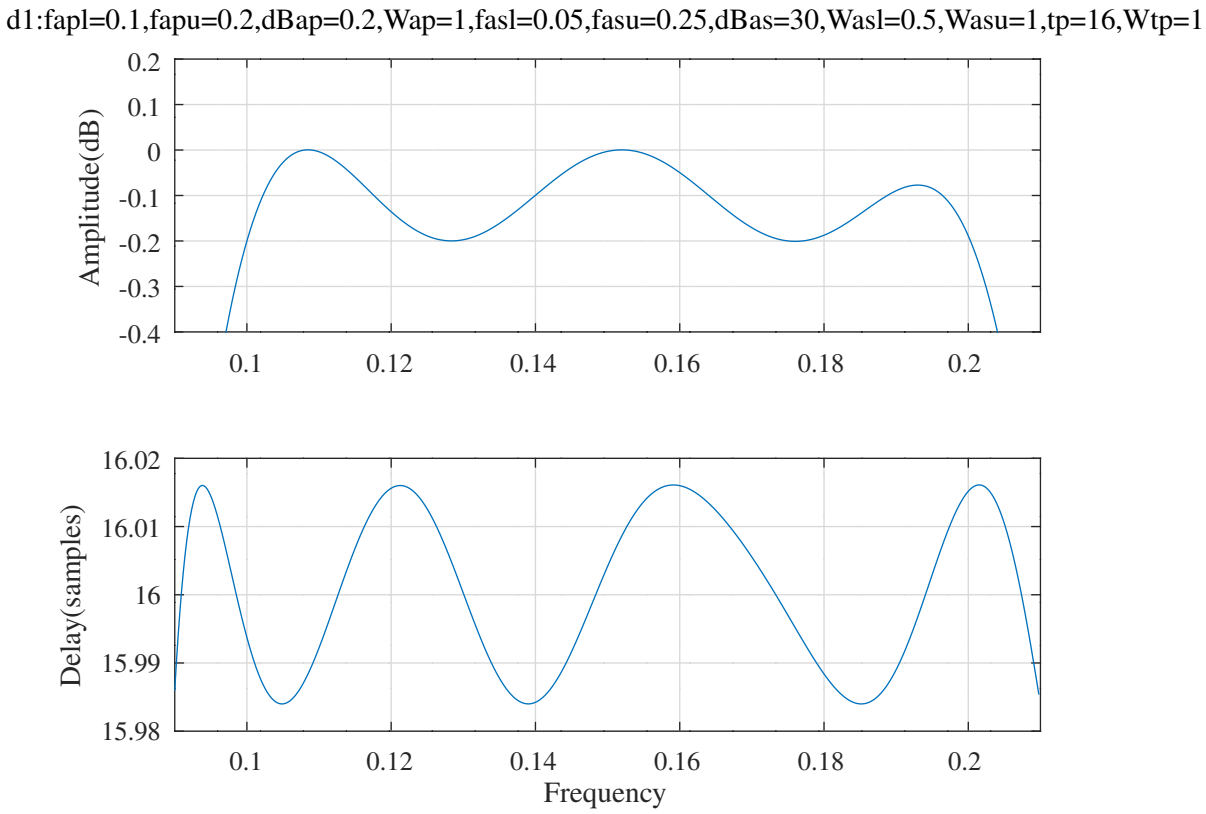



Figure 9.18: IIR band-pass R=2 decimation filter with delay tp=16 samples : passband response of the filter after PCLS SOCP optimisation

d1:fapl=0.1,fapu=0.2,dBap=0.2,Wap=1,fasl=0.05,fasu=0.25,dBas=30,Wasl=0.5,Wasu=1,tp=16,Wtp=1

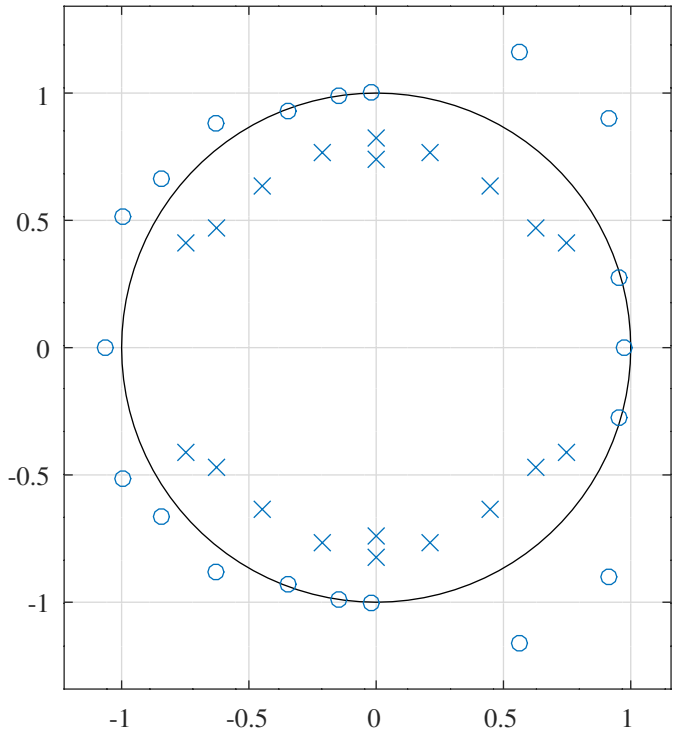



Figure 9.19: IIR band-pass R=2 decimation filter with delay tp=16 samples : pole-zero plot of the filter after PCLS SOCP optimisation

## 9.8 SOCP MMSE design of a multi-band-pass filter expressed in *gain-zero-pole* format with *SeDuMi*

The Octave script *iir_socp_slb_multiband_test.m* implements the design of a multi-band-pass IIR filter expressed in *gain-pole-zero* format with SOCP and PCLS optimisation using *SeDuMi*. The specification of the filter is:

```
tol=0.0001 % Tolerance on combined response
mtol=0.0001 % Tolerance on MMSE update
ptol=0.0001 % Tolerance on PCLS update
ctol=0.0001 % Tolerance on constraints
maxiter=500 % SOCP iteration limit
npoints=500 % Frequency points across the band
nplot=500 % Frequency points plotted across the band
rho=0.999900 % Constraint on pole radius
fas1u=0.05 % Amplitude stop band 1 upper edge
fap1l=0.075 % Amplitude pass band 1 lower edge
fap1u=0.1 % Amplitude pass band 1 upper edge
fas2l=0.125 % Amplitude stop band 2 lower edge
fas2u=0.15 % Amplitude stop band 2 upper edge
fap2l=0.175 % Amplitude pass band 2 lower edge
fap2u=0.225 % Amplitude pass band 2 upper edge
fas3l=0.25 % Amplitude stop band 3 lower edge
dBas1=20 % Amplitude stop band 1 attenuation
dBap1=1 % Amplitude pass band 1 peak-to-peak ripple
dBas2=20 % Amplitude stop band 2 attenuation
dBap2=1 % Amplitude pass band 2 peak-to-peak ripple
dBas3=20 % Amplitude stop band 3 attenuation
Was1=1 % Amplitude stop band 1 weight
Wap1=1 % Amplitude pass band 1 weight
Was2=1 % Amplitude stop band 2 weight
Wap2=1 % Amplitude pass band 2 weight
Was3=1 % Amplitude stop band 3 weight
ftp1l=0.08 % Delay pass band 1 lower edge
ftp1u=0.095 % Delay pass band 1 upper edge
ftp2l=0.185 % Delay pass band 2 lower edge
ftp2u=0.215 % Delay pass band 2 upper edge
tp1=30 % Nominal pass band 1 filter group delay
tp2=15 % Nominal pass band 2 filter group delay
tpr1=10 % Delay pass band 1 peak-to-peak ripple
tpr2=10 % Delay pass band 2 peak-to-peak ripple
Wtp1=0.0001 % Delay pass band 1 weight
Wtp2=0.0001 % Delay pass band 2 weight
```

The initial filter is designed by frequency transformation of a prototype elliptic filter. The SOCP PCLS optimised filter is:

```
Ux2=2,Vx2=0,Mx2=18,Qx2=20,Rx2=1
x2 = [   0.0548096341, ...
        -0.9998255551,   0.9998746732, ...
         0.9992931030,   0.9995165132,   0.9995765713,   0.9996007852, ...
         0.9997193048,   0.9997835163,   0.9998031135,   0.9998921594, ...
         1.0007620164, ...
         0.6358430968,   1.0680775917,   1.0922840531,   0.4605553429, ...
         1.4802266753,   0.6473490797,   0.8290481931,   1.4249686718, ...
         0.4419922953, ...
         0.8987391970,   0.9470306474,   0.9579908292,   0.9760327548, ...
         0.9783069711,   0.9860579203,   0.9951413404,   0.9960146004, ...
         0.9973218184,   0.9980163637, ...
         1.2176572518,   0.5574798955,   1.3900415881,   1.1110224649, ...
         0.4784099214,   0.6232628828,   1.4138766159,   0.4693335882, ...
         1.0986855896,   0.6306833796 ]';
```

Figure 9.20 shows the amplitude and delay response of the PCLS SOCP optimised filter. Figure 9.21 shows the pass-band response of the filter.

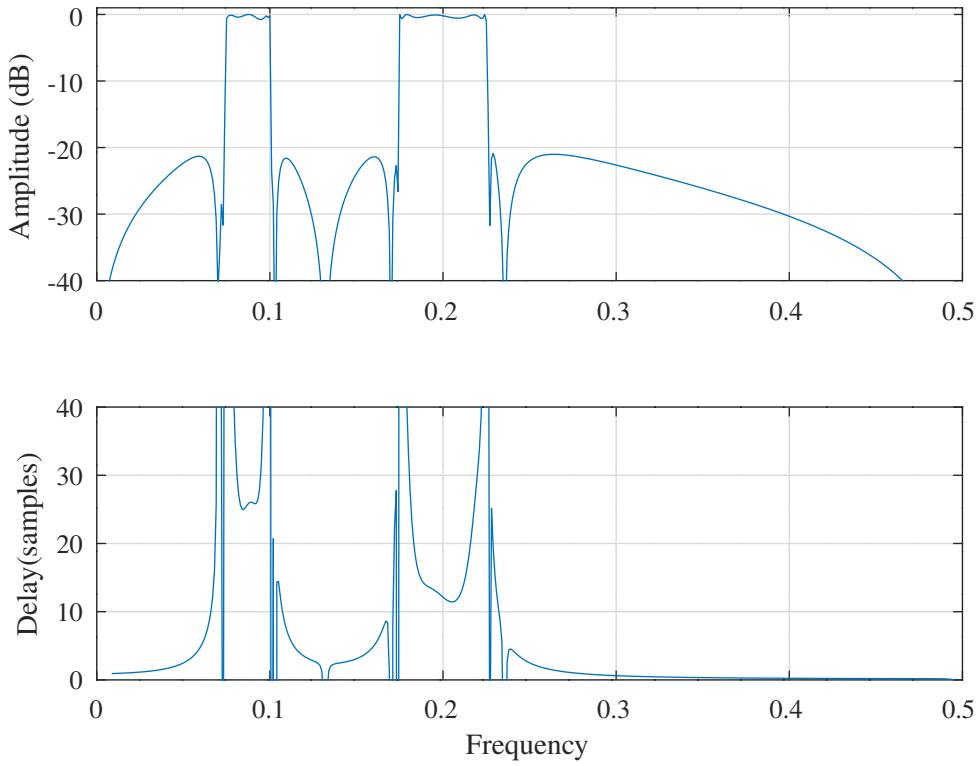



Figure 9.20: IIR multi-band filter : response after SOCP PCLS optimisation

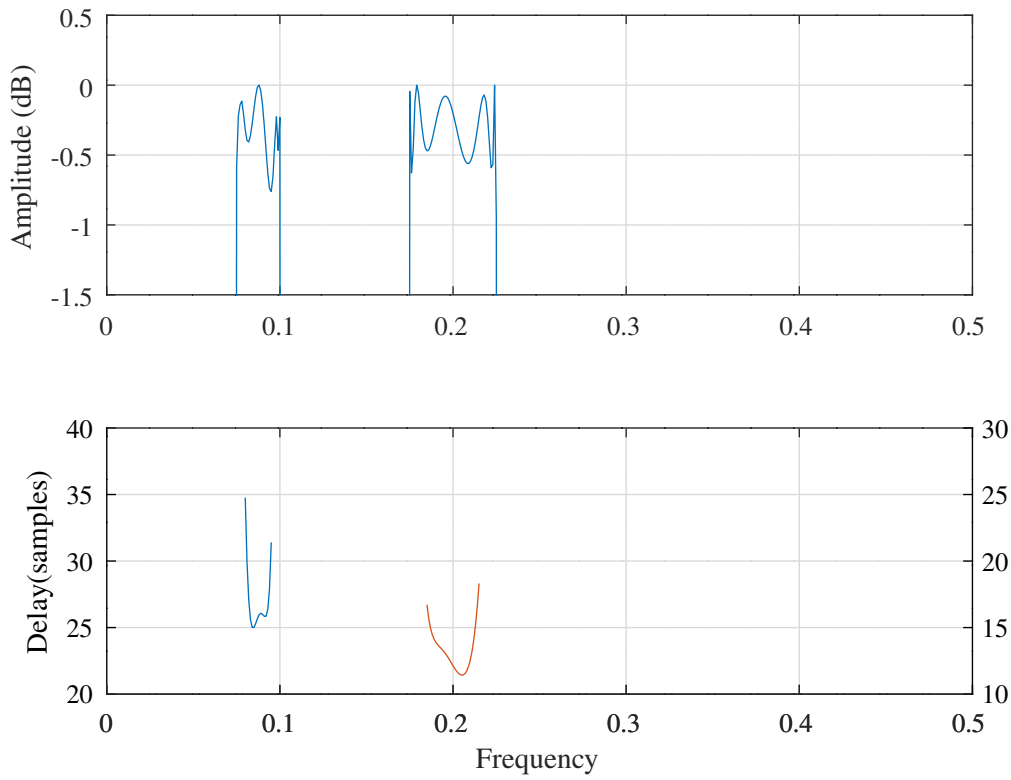



Figure 9.21: IIR multi-band filter : pass-band response after SOCP PCLS optimisation

# Chapter 10

# IIR filter design with a pre-defined structure

In this chapter I describe the design of IIR filters with a predefined structure.

## 10.1 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* [211]).

### 10.1.1 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

$$
\boldsymbol{d} = \begin{bmatrix} d_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_L \end{bmatrix} \qquad \text{where } \boldsymbol{d}_i = \begin{bmatrix} d_{i1} \\ d_{i2} \end{bmatrix}
$$

$$
\boldsymbol{C} = \begin{bmatrix} c_1 & & & \boldsymbol{0} \\ & c_2 & & \\ & & \ddots & \\ \boldsymbol{0} & & & c_2 \end{bmatrix} \qquad \text{where } c_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ and } c_2 = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 0 & 1 \end{bmatrix}
$$

$$
\boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_2 \end{bmatrix} \qquad \text{where } e_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } e_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}
$$

The stability constraint on the filter denominator polynomial is

$$
\boldsymbol{C}\boldsymbol{d} + (1-\tau)\,\boldsymbol{e} \geq \boldsymbol{0}
$$

Suppose the coefficient vector, $\boldsymbol{d}$, is perturbed by $\boldsymbol{\delta}$. Then the stability constraint becomes

$$
\boldsymbol{C}\boldsymbol{\delta} + \boldsymbol{h} \geq \boldsymbol{0}
$$

where $\boldsymbol{h} = \boldsymbol{C}\boldsymbol{d} + (1-\tau)\,\boldsymbol{e}$.

### 10.1.2 Linear constraints on limit-cycle oscillations in second-order filter sections

Section 3.2 states that second-order section limit-cycle oscillations can be suppressed by adding the constraint:

$$
|d_1| + |d_2| \leq 1
$$

so that, in addition to the stability constraints shown in the previous section:

$$
\begin{aligned}
d_1 - d_2 &> -1+\tau \\
-d_1 - d_2 &> -1+\tau
\end{aligned}
$$

and

$$
e_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}
$$

$$
c_2 = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & -1 \\ 0 & 1 \end{bmatrix}
$$

### 10.1.3 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)

$$d(z) = \begin{cases} \left(1 + d_0 z^{-1}\right) \prod_{i=1}^{\frac{r-1}{2}} \left(1 + d_{i1} z^{-1} + d_{i2} z^{-2}\right), & \text{if } r \text{ odd} \\ \prod_{i=1}^{\frac{r}{2}} \left(1 + d_{i1} z^{-1} + d_{i2} z^{-2}\right), & \text{if } r \text{ even} \end{cases}$$

Define the two filter coefficient vectors as

$$\boldsymbol{a} = \begin{bmatrix} a_0 & a_1 & \dots & a_n \end{bmatrix}^\top$$

and

$$\boldsymbol{d} = \begin{bmatrix} d_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_L \end{bmatrix}$$

$$\boldsymbol{d}_i = \begin{bmatrix} d_{i1} \\ d_{i2} \end{bmatrix}, \quad \text{for } 1 \le i \le L$$

$$L = \begin{cases} \frac{r-1}{2}, & \text{if } r \text{ odd} \\ \frac{r}{2}, & \text{if } r \text{ even} \end{cases}$$

The frequency response of the filter is

$$H(\omega) = \frac{\boldsymbol{a}^\top \boldsymbol{v}(\omega)}{d(\omega)}$$

where

$$\begin{aligned} \boldsymbol{v}(\omega) &= \boldsymbol{c}(\omega) - \imath \boldsymbol{s}(\omega) \\ \boldsymbol{c}(\omega) &= \begin{bmatrix} 1 & \dots & \cos n\omega \end{bmatrix}^\top \\ \boldsymbol{s}(\omega) &= \begin{bmatrix} 0 & \dots & -\sin n\omega \end{bmatrix}^\top \\ v_1(\omega) &= \cos\omega - \imath \sin\omega \\ \boldsymbol{v}_2(\omega) &= \begin{bmatrix} \cos\omega \\ \cos 2\omega \end{bmatrix} - \imath \begin{bmatrix} \sin\omega \\ \sin 2\omega \end{bmatrix} \end{aligned}$$

and

$$d(\omega) = \begin{cases} \left[1 + d_0 v_1(\omega)\right] \prod_{i=1}^{L} \left[1 + \boldsymbol{d}_i^\top \boldsymbol{v}_2(\omega)\right], & \text{if } r \text{ odd} \\ \prod_{i=1}^{L} \left[1 + \boldsymbol{d}_i^\top \boldsymbol{v}_2(\omega)\right], & \text{if } r \text{ even} \end{cases}$$

The gradients of $H(\omega)$ with respect to the coefficients are

$$\begin{aligned} \frac{\partial H(\omega)}{\partial \boldsymbol{a}} &= \frac{\boldsymbol{v}(\omega)}{d(\omega)} \\ \frac{\partial H(\omega)}{\partial d_0} &= -H(\omega) \frac{v_1(\omega)}{1 + d_0 v_1(\omega)} \\ \frac{\partial H(\omega)}{\partial \boldsymbol{d}_i} &= -H(\omega) \frac{\boldsymbol{v}_2(\omega)}{1 + \boldsymbol{d}_i^\top \boldsymbol{v}_2(\omega)} \quad \text{for } 1 \le i \le L \end{aligned}$$

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

$$
\begin{aligned}
\boldsymbol{A}_i^\top &= W(\omega_i) \begin{bmatrix} 0 & \Re\nabla_x H(\omega_i)^\top \\ 0 & \Im\nabla_x H(\omega_i)^\top \end{bmatrix} \\
\boldsymbol{b}_i^\top &= \begin{bmatrix} 1 & \mathbf{0} \end{bmatrix} \\
\boldsymbol{c}_i &= W(\omega_i) \begin{bmatrix} 0 & \Re\left[H(\omega_i) - H_d(\omega_i)\right] \\ 0 & \Im\left[H(\omega_i) - H_d(\omega_i)\right] \end{bmatrix} \\
d_i &= 0
\end{aligned}
$$

where $x$ represents the vector of coefficients and the desired low pass filter frequency response is

$$
H_d(\omega) = \begin{cases} e^{-\imath\omega t_d} & \text{if } 0 \le \omega \le \omega_{pass} \\ 10^{-\frac{dBstop}{20}} & \text{if } \omega_{pass} < \omega \end{cases}
$$

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. The script does not enforce the limit-cycle constraints. After SOCP MMSE optimisation, the numerator and denominator polynomials are respectively:

```
a = [  -0.0021092647,   0.0004069329,   0.0076883395,   0.0051931733, ...
       -0.0115231077,  -0.0210543663,   0.0019486082,   0.0417050319, ...
        0.0578424346,   0.0390044901,   0.0123776721 ]';
```

and

```
d = [   1.0000000000,  -2.4277939777,   3.0446749930,  -2.3351332888, ...
        1.1383562669,  -0.3371599412,   0.0468622433 ]';
```

The overall frequency response of the MMSE SOCP design is shown in Figure 10.1 with pass-band details shown in Figure 10.2 and pole-zero plot shown in Figure 10.3.

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 10.4, the passband responses are shown in Figure 10.5 and the pole-zero plot is shown in Figure 10.6.

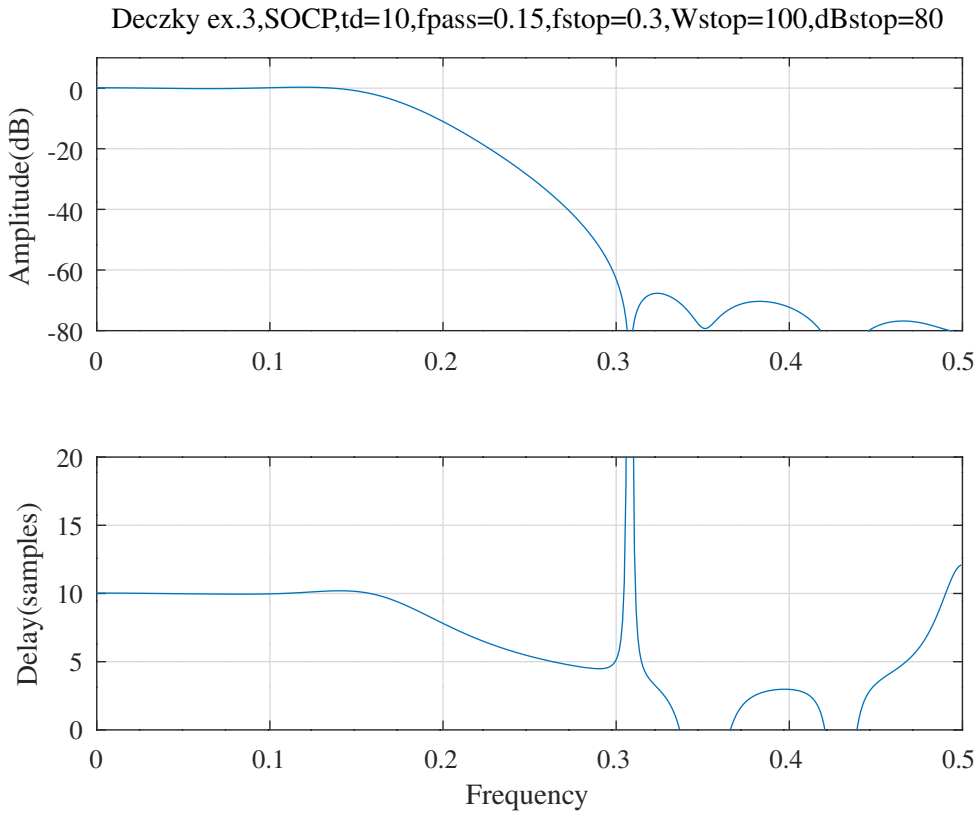



Figure 10.1: Deczky Example 3:Response with 2nd order sections and MMSE SOCP optimisation

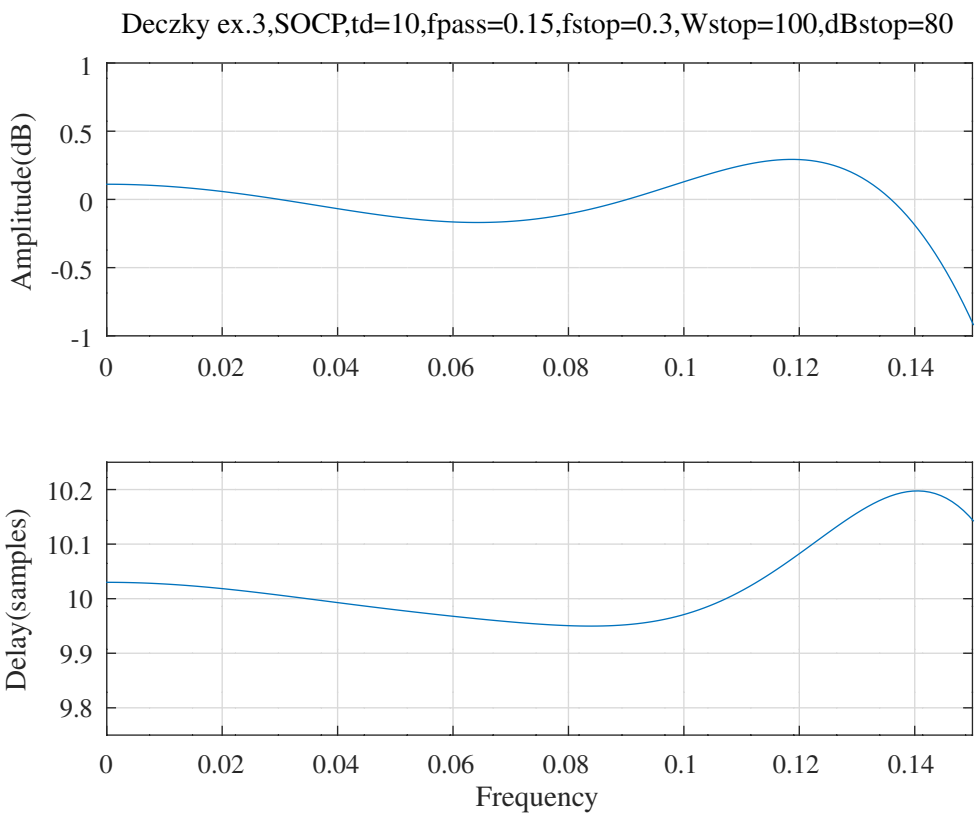



Figure 10.2: Deczky Example 3:Passband response with 2nd order sections and MMSE SOCP optimisation

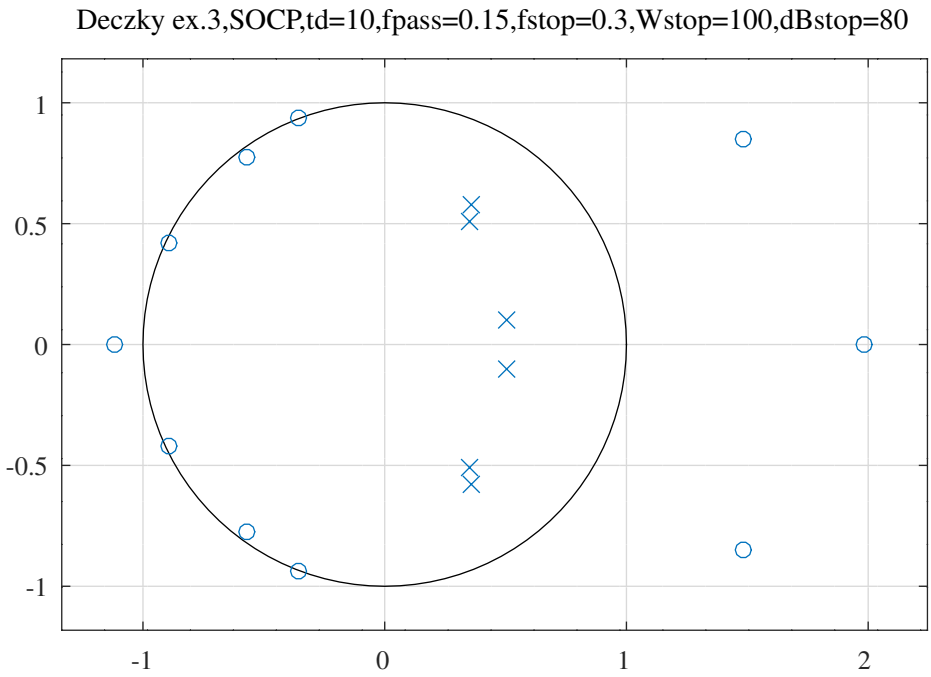



Figure 10.3: Deczky Example 3:Pole-zero plot with 2nd order sections and MMSE SOCP optimisation

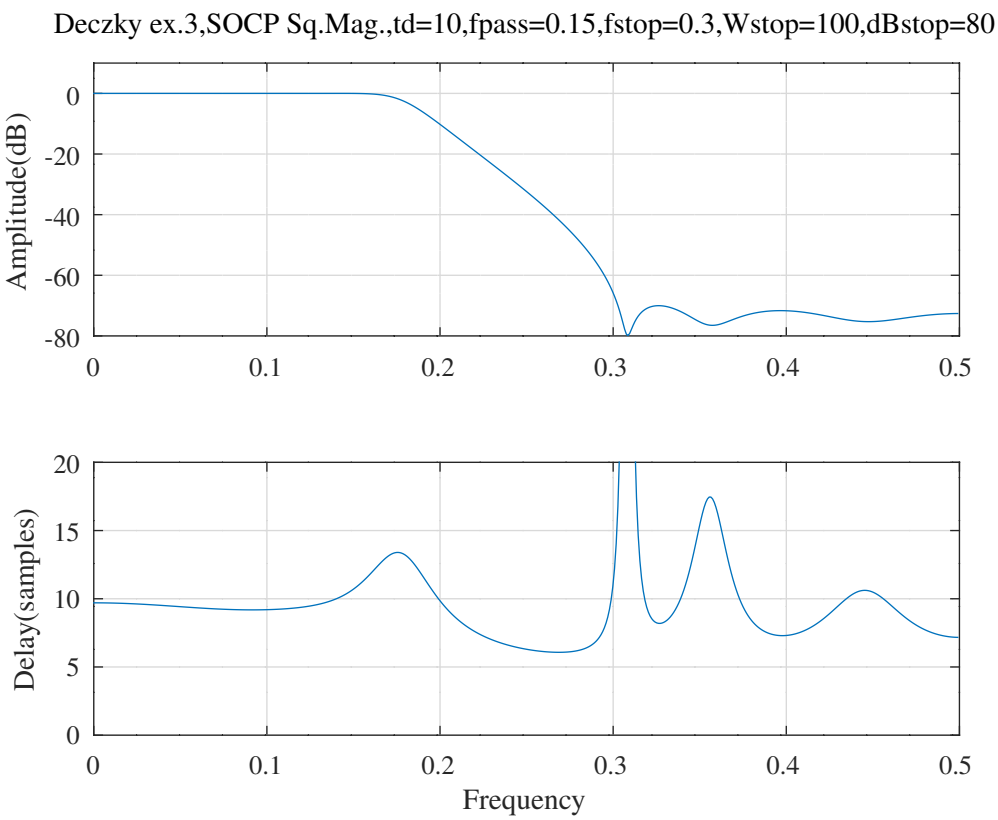



Figure 10.4: Deczky Example 3:Response with 2nd order sections and MMSE SOCP optimisation of the squared-magnitude response

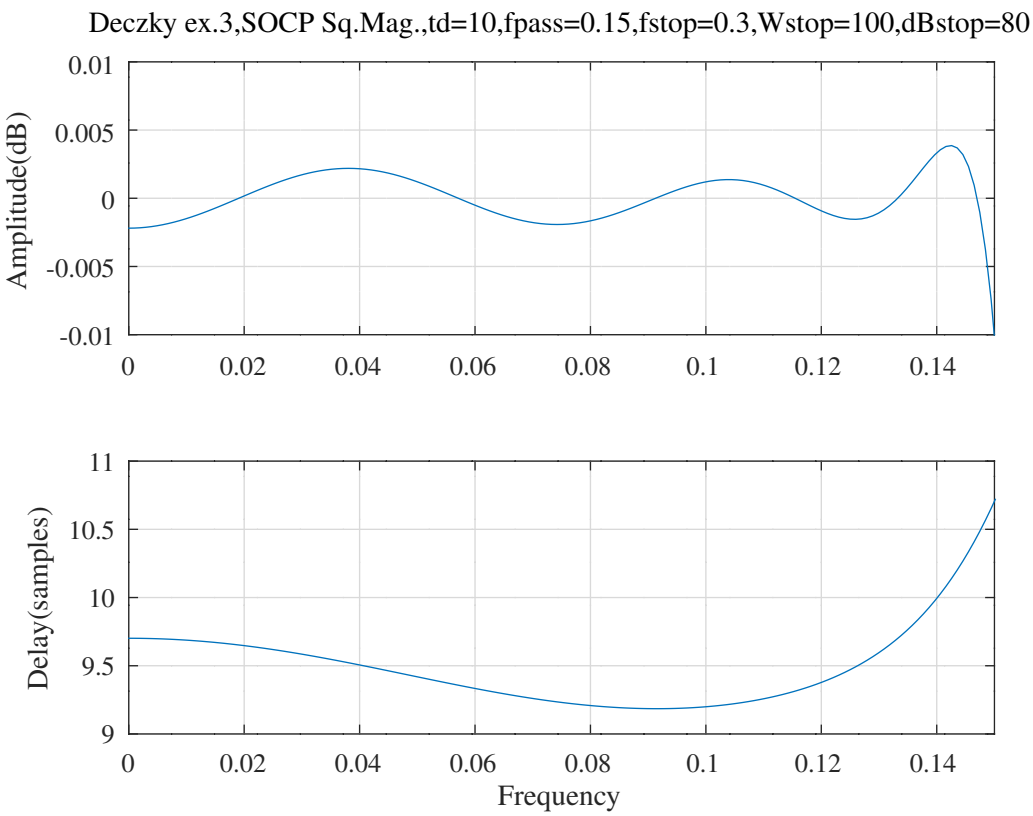



Figure 10.5: Deczky Example 3:Passband response with 2nd order sections and MMSE SOCP optimisation of the squared-magnitude response

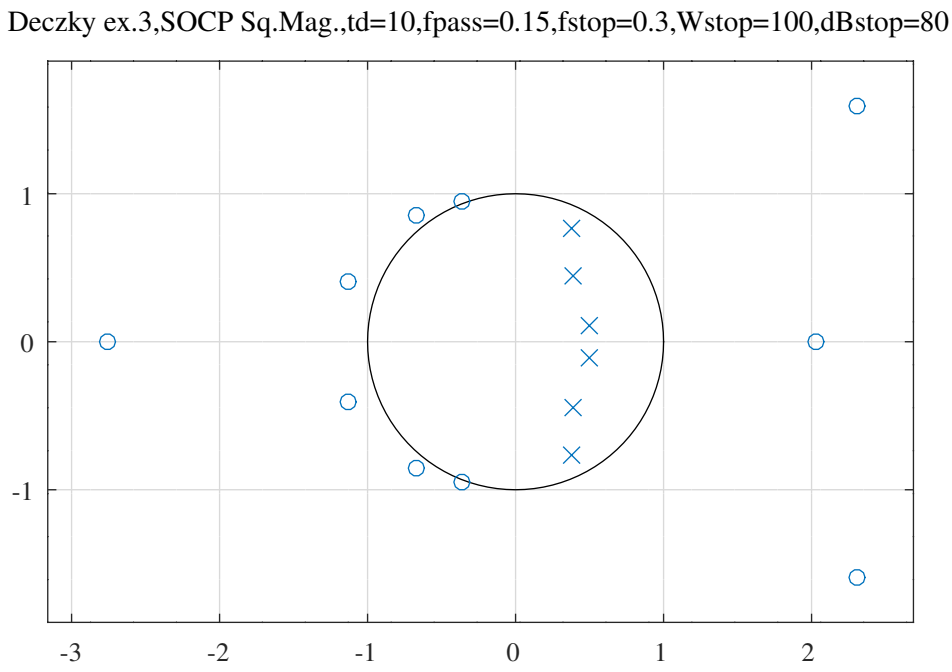



Figure 10.6: Deczky Example 3:Pole-zero plot with 2nd order sections and MMSE SOCP optimisation of the squared-magnitude response

### 10.1.4 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.

## 10.2 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 $B(z)$

$$H(z) = \frac{A(z) + B(z)}{2} \tag{10.1}$$

*Vaidyanathan et al.* [152] 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.

### 10.2.1 Design of an IIR filter as the sum of two all-pass filters each composed of second-order sections

Following *Lu* and *Hinamoto* [211], the transfer functions of the $A(z)$ and $B(z)$ filters are expressed as the product of second-order sections. As shown in Section 10.1.1, 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

$$\begin{aligned} a(z) &= \left(1 + a_0 z^{-1}\right) \prod_{k=1}^{\frac{m-1}{2}} 1 + a_{k1} z^{-1} + a_{k2} z^{-2} \\ A(z) &= z^{-m} \frac{a\left(z^{-1}\right)}{a(z)} \end{aligned}$$

and

$$\begin{aligned} b(z) &= \prod_{k=1}^{\frac{n}{2}} 1 + b_{k1} z^{-1} + b_{k2} z^{-2} \\ B(z) &= z^{-n} \frac{b\left(z^{-1}\right)}{b(z)} \end{aligned}$$

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:

$$\begin{aligned} A(\omega) &= e^{-\imath m \omega} \frac{a(-\omega)}{a(\omega)} \\ a(\omega) &= (1 + a_0 v_1) \prod_{k=1}^{\frac{m-1}{2}} 1 + \boldsymbol{v}_2 \boldsymbol{a}_k \end{aligned}$$

where $\boldsymbol{a}_k = \begin{bmatrix} a_{k1} \\ a_{k2} \end{bmatrix}$, $v_1 = \cos\omega - \imath \sin\omega$, $\boldsymbol{v}_2 = \begin{bmatrix} \cos\omega & \cos 2\omega \end{bmatrix} - \imath \begin{bmatrix} \sin\omega & \sin 2\omega \end{bmatrix}$ and $A(\omega)$ is understood to mean $A(e^{\imath\omega})$.

The gradients of $A(z)$ with respect to its coefficients $a_0$, $a_{k1}$ and $a_{k2}$ are

$$\begin{aligned} \frac{\partial A(z)}{\partial a_0} &= A(z) \frac{z - z^{-1}}{1 + a_0^2 + a_0\left(z + z^{-1}\right)} \\ \frac{\partial A(z)}{\partial a_{k1}} &= A(z) \frac{\left(z - z^{-1}\right)(1 - a_{k2})}{1 + a_{k1}^2 + a_{k2}^2 + a_{k1}(1 + a_{k2})\left(z + z^{-1}\right) + a_{i2}\left(z^2 + z^{-2}\right)} \\ \frac{\partial A(z)}{\partial a_{k2}} &= A(z) \frac{\left(z - z^{-1}\right)\left(a_{k1} + z + z^{-1}\right)}{1 + a_{k1}^2 + a_{k2}^2 + a_{k1}(1 + a_{k2})\left(z + z^{-1}\right) + a_{k2}\left(z^2 + z^{-2}\right)} \end{aligned}$$

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 9.4, the parallel allpass filter design problem can be expressed in SOCP form as

**minimise** $\epsilon,\ \beta$

$$
\begin{aligned}
\textbf{subject to} \quad & \left\| W(\omega_i) \begin{bmatrix} \Re\nabla H(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i)^\top \\ \Im\nabla H(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i)^\top \end{bmatrix} \boldsymbol{\delta} + W(\omega_i) \begin{bmatrix} \Re H(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i) - \Re H_d(\omega_i) \\ \Im H(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i) - \Im H_d(\omega_i) \end{bmatrix} \right\| \leq \epsilon \\
& \|\boldsymbol{\delta}\| \leq \beta \\
& \boldsymbol{C}\boldsymbol{\delta} + \boldsymbol{h} \geq \boldsymbol{0} \\
\textbf{where} \quad & \boldsymbol{\delta} = \begin{bmatrix} \boldsymbol{a} - \boldsymbol{a}_k \\ \boldsymbol{b} - \boldsymbol{b}_k \end{bmatrix} \\
& H(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i) = 0.5\left[A(\boldsymbol{a}_k, \omega_i) + B(\boldsymbol{b}_k, \omega_i)\right] \\
& \nabla H(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i) = 0.5\left[\nabla_a A(\boldsymbol{a}_k, \omega_i) + \nabla_b B(\boldsymbol{b}_k, \omega_i)\right]
\end{aligned}
$$

A quadratic constraint on the squared-magnitude response at the stop-band frequencies is

$$
\left\| \nabla H^2(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i)^\top \boldsymbol{\delta} + H^2(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i) \right\| \leq |H_d(\omega_i)|^2
$$

where

$$
\begin{aligned}
H^2(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i) &= \|H(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i)\|^2 \\
&= 0.25\left[\Re A(\boldsymbol{a}_k, \omega_i) + \Re B(\boldsymbol{b}_k, \omega_i)\right]^2 + 0.25\left[\Im A(\boldsymbol{a}_k, \omega_i) + \Im B(\boldsymbol{b}_k, \omega_i)\right]^2 \\
\nabla H^2(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i) &= 0.5\left[\Re A(\boldsymbol{a}_k, \omega_i) + \Re B(\boldsymbol{b}_k, \omega_i)\right]\left[\Re\nabla_a A(\boldsymbol{a}_k, \omega_i) + \Re\nabla_b B(\boldsymbol{b}_k, \omega_i)\right] + \ldots \\
&\quad 0.5\left[\Im A(\boldsymbol{a}_k, \omega_i) + \Im B(\boldsymbol{b}_k, \omega_i)\right]\left[\Im\nabla_a A(\boldsymbol{a}_k, \omega_i) + \Im\nabla_b B(\boldsymbol{b}_k, \omega_i)\right]
\end{aligned}
$$

**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:

$$
H_d(\omega) = \begin{cases} e^{-\imath\omega t_d} & \text{if } 0 \leq \omega \leq \omega_{pass} \\ 0 & \text{if } \omega_{pass} < \omega \end{cases}
$$

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=1000 % 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 10.7.

After optimisation with the *SeDuMi* SOCP solver the response is shown in Figure 10.8 with passband detail shown in Figure 10.9 and pole-zero plot shown in Figure 10.10. The second-order section coefficients are

```
a1 = [   0.7274966236,   1.7414347737,   0.7577922953,  -0.1991323756, ...
        -0.2701513245,  -1.1972910365,   0.4944284184,  -0.7065290013, ...
         0.4927964400,   0.3053499809,   0.1510424928 ]';
```

and

```
b1 = [  -0.2338728631,  -0.0957553116,  -0.6933902429,   0.1395230401, ...
         1.6472467047,   0.6733639460,  -0.4924446261,   0.0281709049, ...
        -0.9184014862,   0.8657744917,   0.8966691726,   0.0503654151 ]';
```

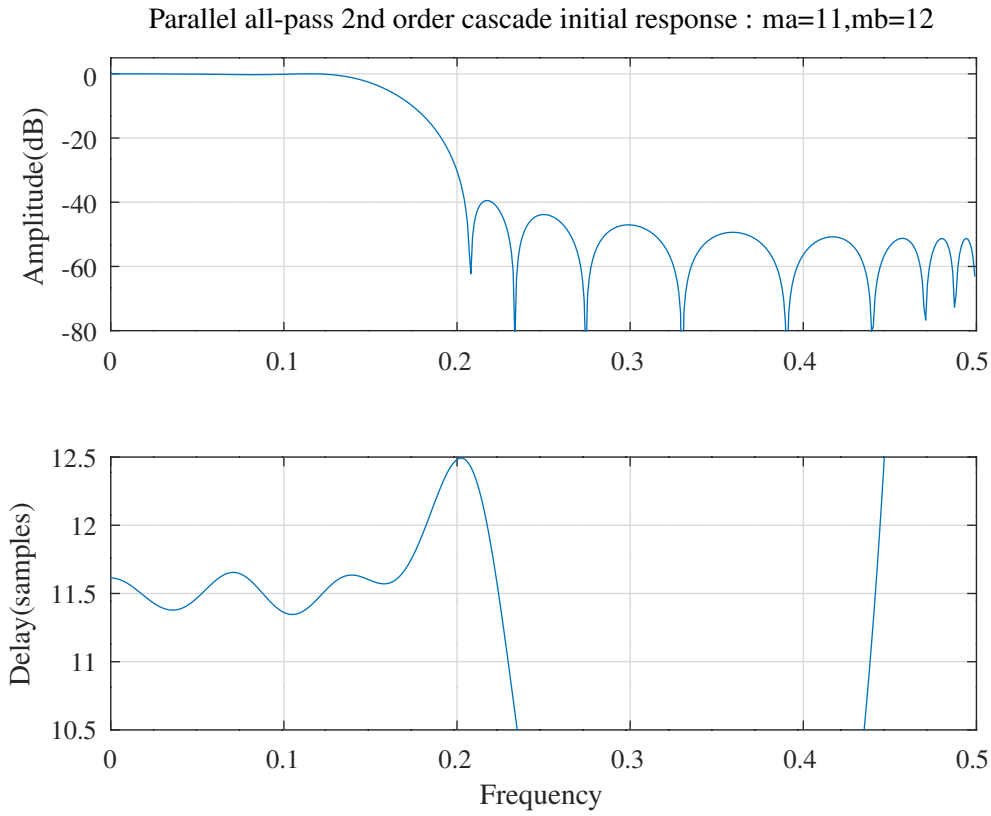



Figure 10.7: Parallel 2nd order cascaded allpass filters : initial response found with the WISE barrier function

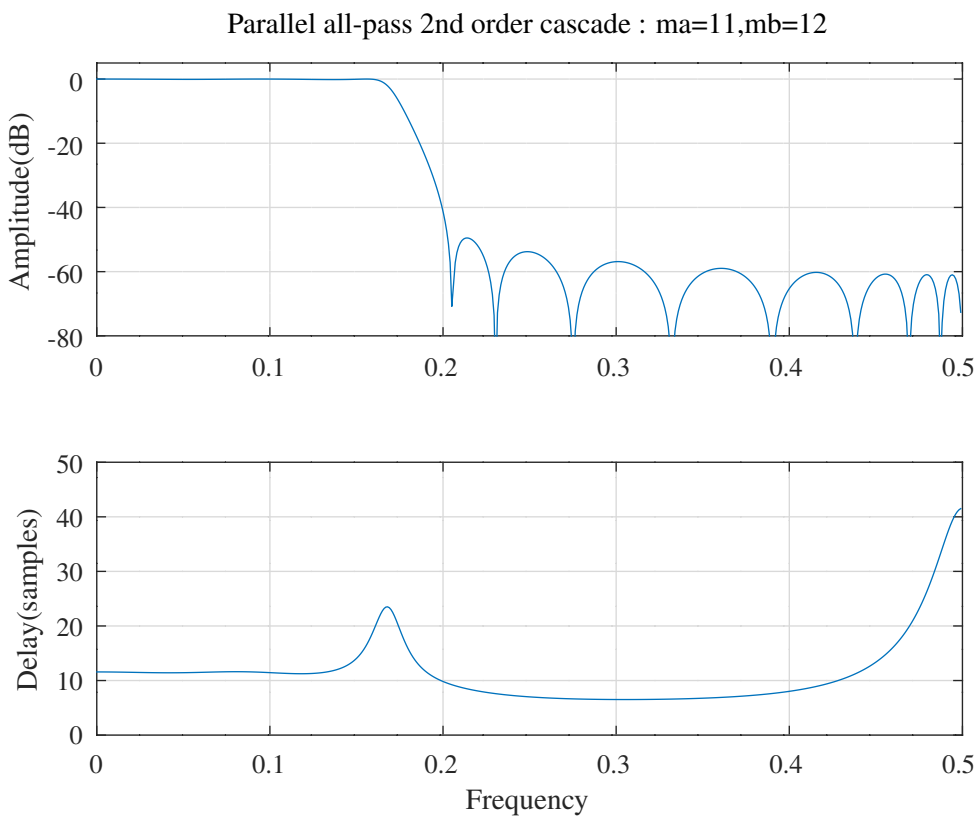



Figure 10.8: Parallel 2nd order cascade allpass filters : response after SOCP optimisation

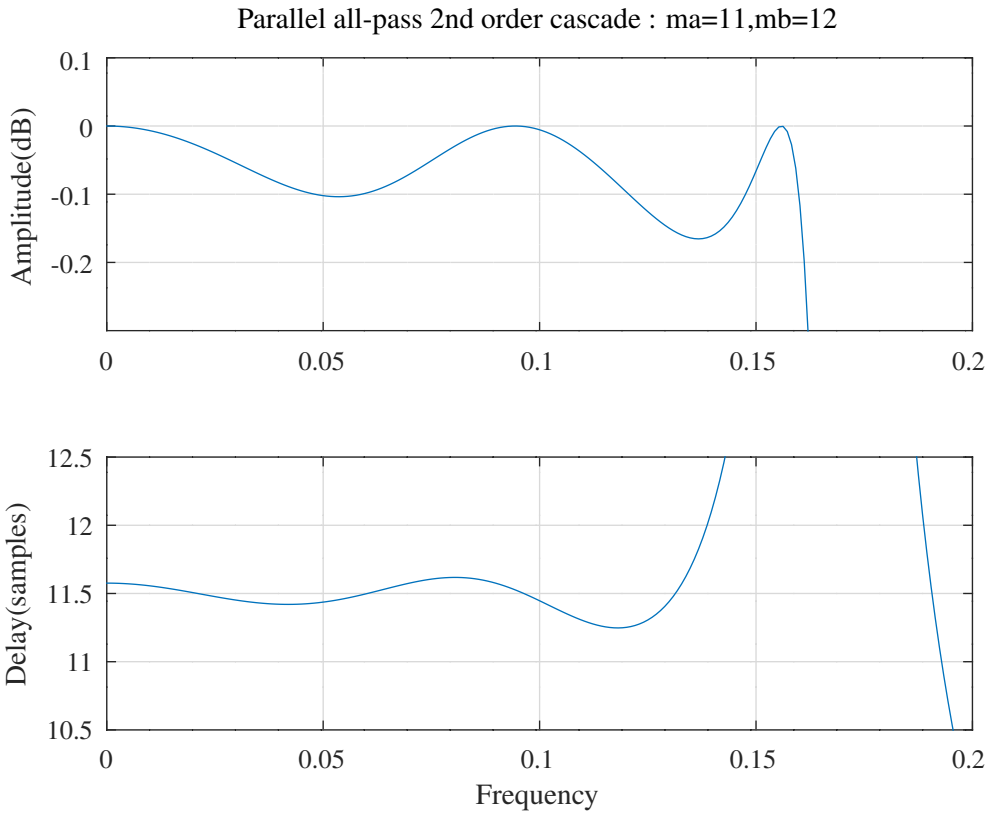



Figure 10.9: Parallel 2nd order cascade allpass filters : pass-band response after SOCP optimisation

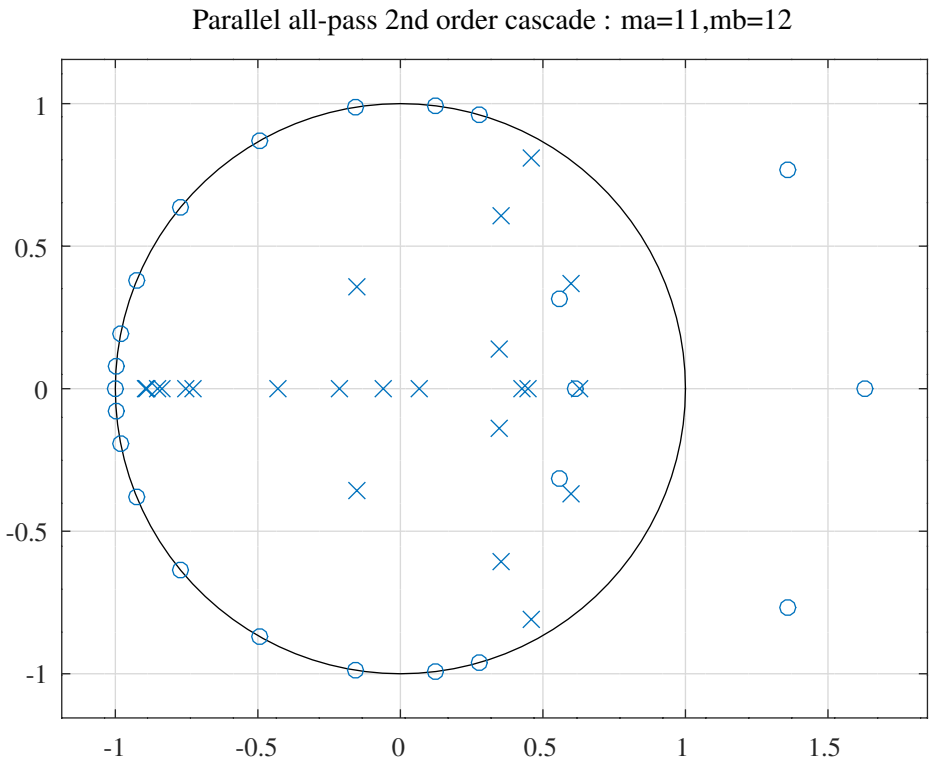



Figure 10.10: Parallel 2nd order cascade allpass filters:pole-zero plot after SOCP optimisation

The corresponding allpass filter denominator polynomials are

```
Da1 = [   1.0000000000,   0.6713289648,  -0.9624642774,  -0.0205090853, ...
          0.6636959866,  -0.2066002005,  -0.1583200654,   0.1062711306, ...
          0.0273661698,  -0.0095827504,  -0.0141193656,  -0.0054809823 ]';
```

and

```
Db1 = [   1.0000000000,   0.2058066589,  -0.8868227515,   0.8603694642, ...
          0.4039996097,  -0.7950488522,   0.0461450119,   0.2086790005, ...
         -0.0579370475,  -0.0068530605,   0.0030400794,   0.0000090523, ...
         -0.0000110509 ]';
```

**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=1000 % 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 10.11.

After optimisation with the *SeDuMi* SOCP solver the response is shown in Figure 10.12 and the pole-zero plot is shown in Figure 10.13. The second-order section coefficients are

```
a1 = [  -0.6215129062,  -1.1437023644,   0.8978793124,  -1.2137641955, ...
         0.6285174820 ]';
```

and

```
b1 = [  -1.2392776008,   0.4611629193,  -1.1414280376,   0.9700349049, ...
        -1.1726228280,   0.7877985049 ]';
```

The corresponding allpass filter denominator polynomials are

```
Da1 = [   1.0000000000,  -2.9789794661,   4.3797776676,  -3.6201008807, ...
          1.6884325922,  -0.3507401463 ]';
```

and

```
Db1 = [   1.0000000000,  -3.5533284664,   6.4252123073,  -6.9410275634, ...
          4.7161270188,  -1.8862967781,   0.3524170357 ]';
```

For comparison, Figures 10.14 and 10.15 show the response and pole-zero plot of an elliptic filter designed with a similar specification:

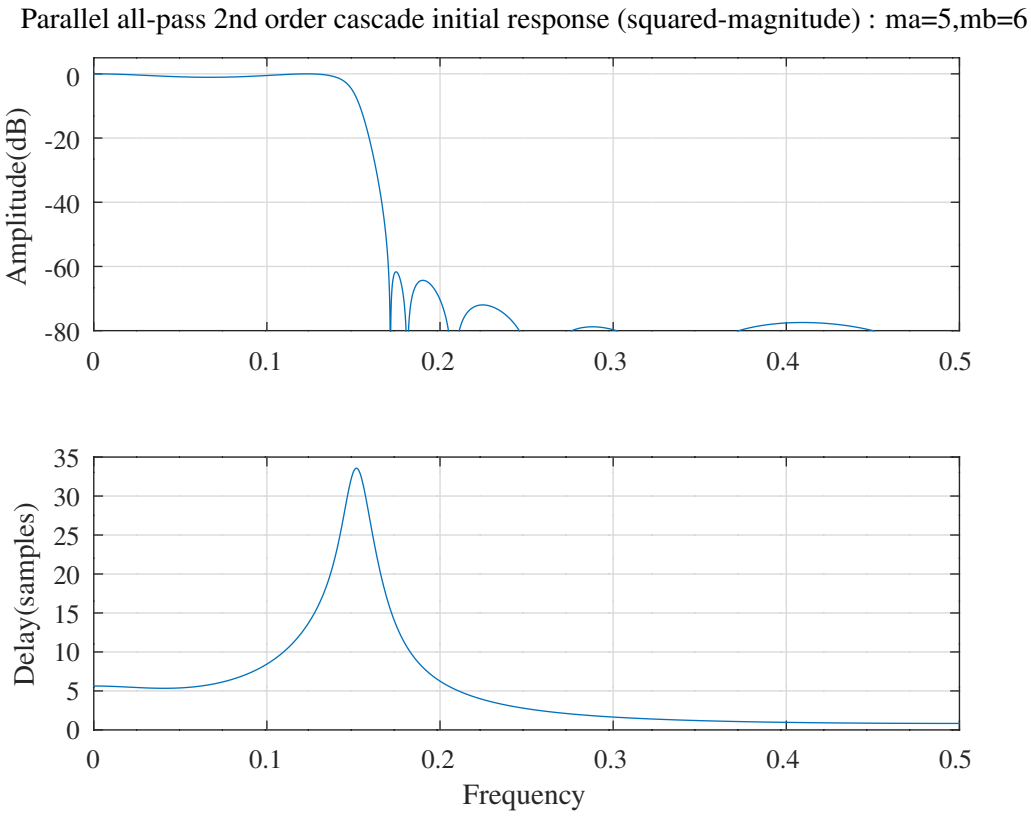



Figure 10.11: Parallel 2nd order allpass cascade filters (squared magnitude):Initial response found with the WISE barrier function

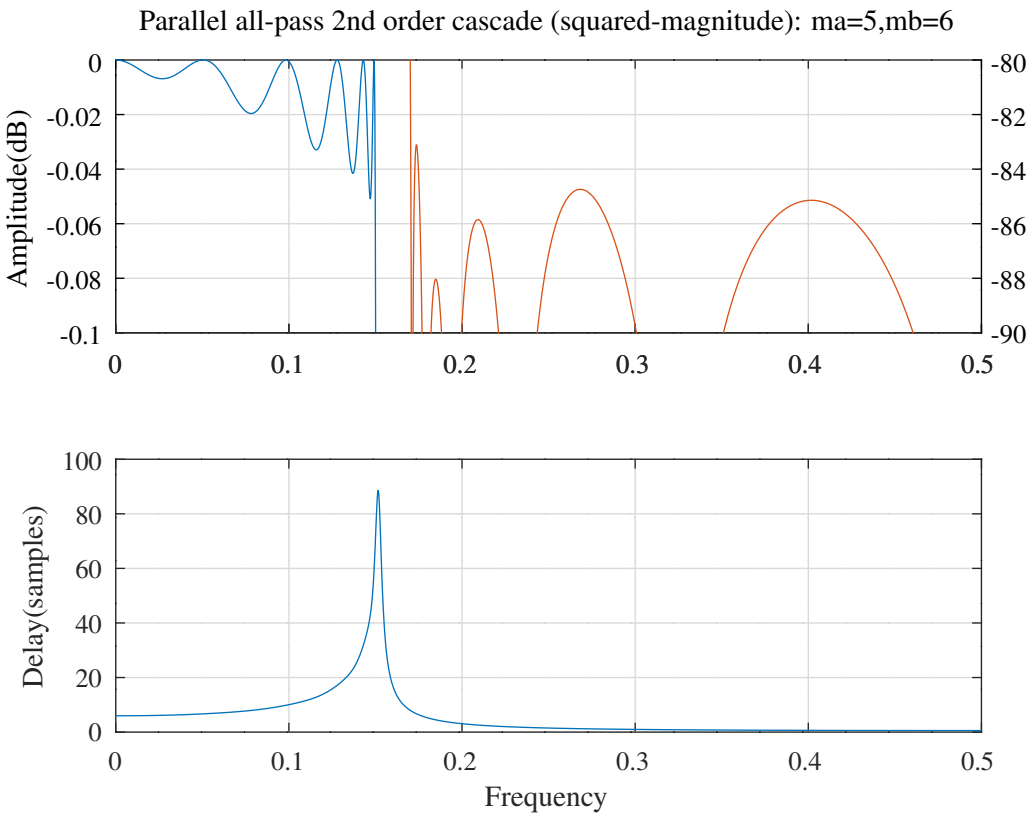



Figure 10.12: Parallel 2nd order cascade allpass filters (squared magnitude) : response after SOCP optimisation

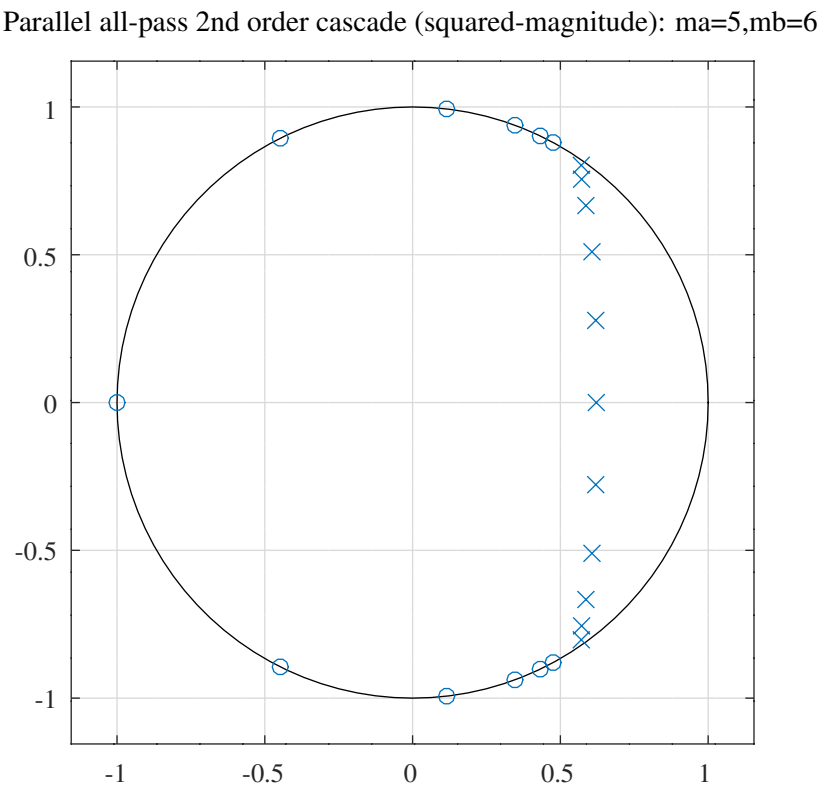



Figure 10.13: Parallel 2nd order cascade allpass filters (squared magnitude):pole-zero plot after SOCP optimisation

```
ma=5,mb=6,fap=0.15,dBap=0.02,fas=0.17,dBas=84
[Nellip,Dellip]=ellip(ma+mb,dBap,dBas,fap*2);
```

The corresponding filter polynomials are:

```
Nellip = [   0.0009293269,  -0.0007898645,   0.0031439811,  -0.0008194769, ...
             0.0035413538,   0.0013979326,   0.0013979326,   0.0035413538, ...
            -0.0008194769,   0.0031439811,  -0.0007898645,   0.0009293269 ];
```

and

```
Dellip = [   1.0000000000,  -6.4891487546,  21.1744373739, -44.7352568821, ...
            67.2938395531, -75.1847257433,  63.4160244544, -40.2860479200, ...
            18.8696899055,  -6.2066204107,   1.2918999342,  -0.1292850044 ];
```

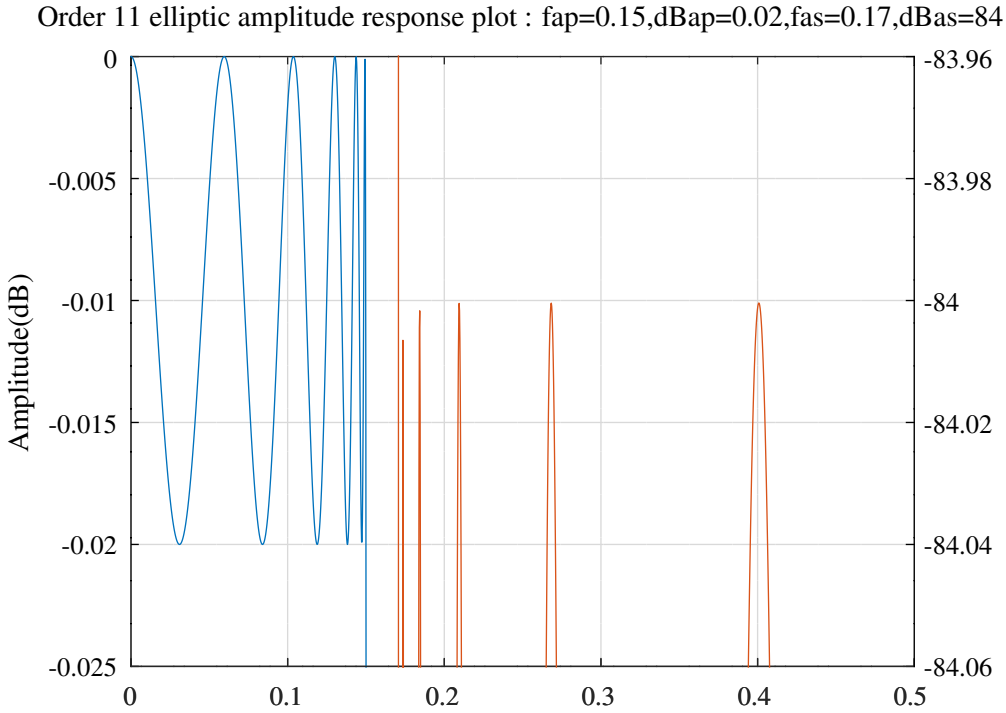



Figure 10.14: Response of an elliptic filter with fap=0.15, dBap=0.02, dBas=84

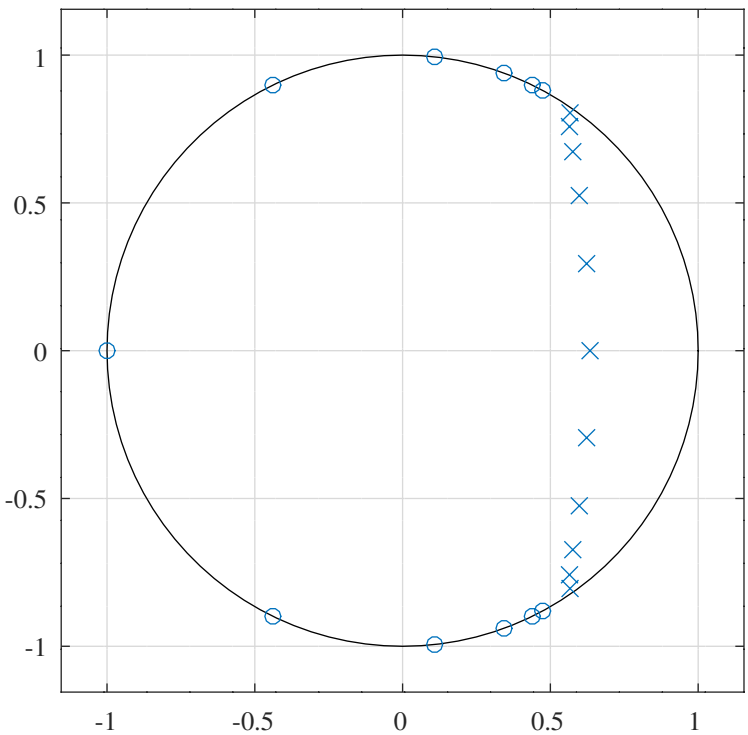



Figure 10.15: Pole-zero plot of an elliptic filter with fap=0.15, dBap=0.02, dBas=84

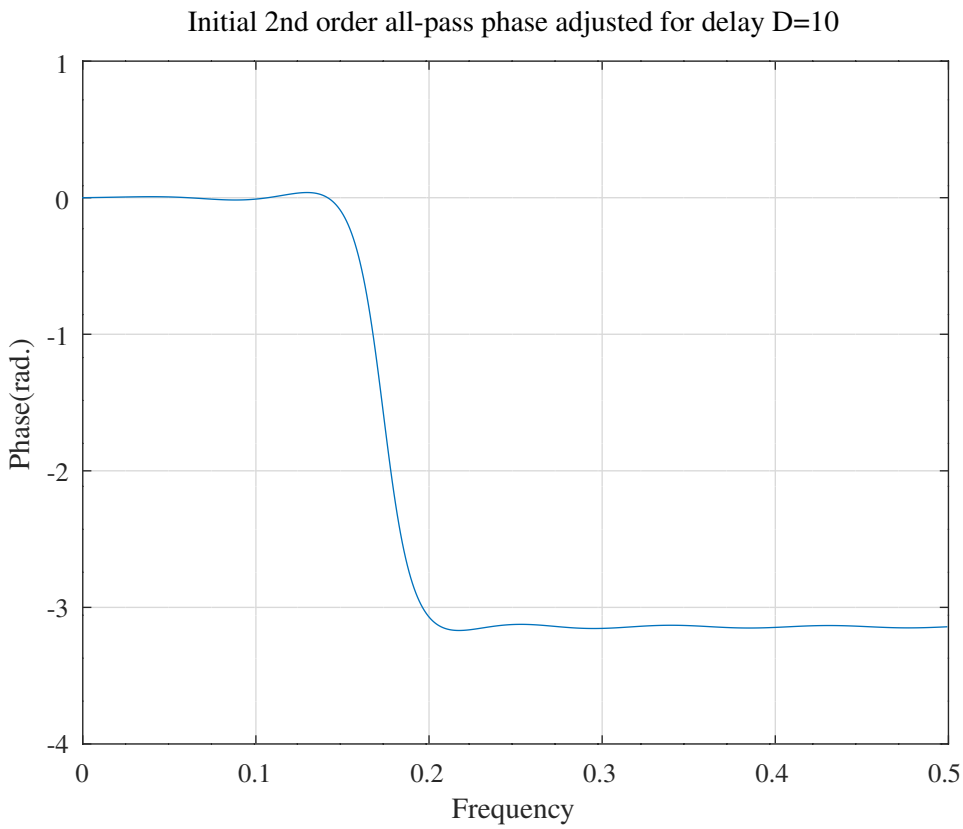



Figure 10.16: Parallel all-pass 2nd order cascade filter and delay:phase response of the initial all-pass filter adjusted for the group delay of the fixed delay branch

### 10.2.2 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 10.2.1. 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 $\pi$ radians in the stop band. The phase response of the initial allpass filter is shown in Figure 10.16. The phase has been adjusted by $\omega D$, where $D$ is the number of samples in the pure delay branch.

The allpass filter polynomial found for the optimised delay case is:

```
Da1 = [   1.0000000000,  -0.4940632914,   0.3737009597,   0.1807528323, ...
          0.0198375901,  -0.0547182792,  -0.0506744693,  -0.0144919076, ...
          0.0113201529,   0.0138227844,   0.0045376389,  -0.0041454492 ]';
```

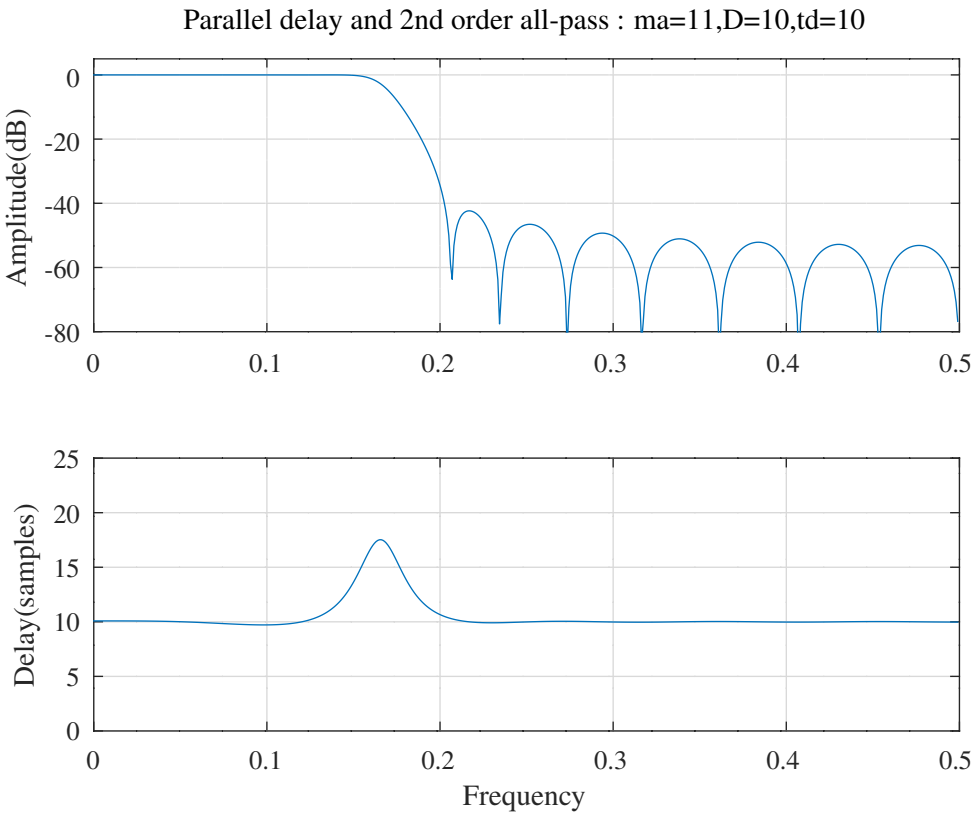



Figure 10.17: Parallel all-pass 2nd order cascade filter and delay:Overall response optimised for flat pass band group delay

with the overall filter response shown in Figure 10.17.

The allpass filter polynomial found for the optimised squared-magnitude case is:

```
Da1sqm = [   1.0000000000,  -0.5292155427,   0.3637414507,   0.1968593384, ...
             0.0368342348,  -0.0661347505,  -0.0989859298,  -0.0823773854, ...
            -0.0478345365,  -0.0194755176,  -0.0051849901,  -0.0007373687 ]';
```

with the overall filter response shown in Figure 10.18. At the filter band edges the change in phase of the allpass branch produces a corresponding transient in the group delay response.

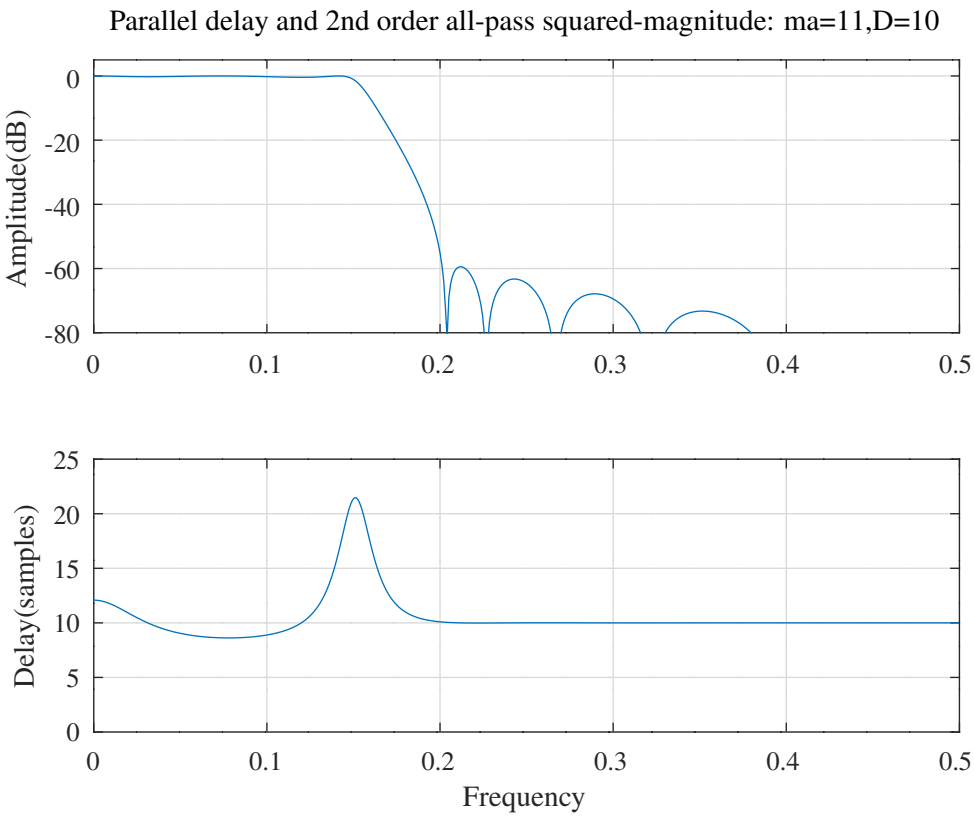



Figure 10.18: Parallel all-pass 2nd order cascade filter and delay:Overall response optimised for squared-magnitude

### 10.2.3 Design of an IIR filter as the sum of two all-pass filters each represented in pole-zero form

Rewriting Equation 10.1 in terms of the phase responses of the component all-pass filters gives

$$H(z) = \frac{e^{\imath\phi_1(z)} + e^{\imath\phi_2(z)}}{2}$$

where $\phi_1(z) = \phi_{A_1}(z)$ and $\phi_2(z) = \phi_{A_2}(z)$. The squared magnitude response and phase of the frequency response, $H(\omega)$, are:

$$|H(\omega)|^2 = \frac{1 + \cos(\phi_1(\omega) - \phi_2(\omega))}{2}$$
$$\phi_H(\omega) = \frac{\phi_1(\omega) + \phi_2(\omega)}{2}$$

The group delay response is

$$T(\omega) = -\frac{1}{2}\left[\frac{\partial\phi_1(\omega)}{\partial\omega} + \frac{\partial\phi_2(\omega)}{\partial\omega}\right]$$

The partial derivatives of the squared-magnitude, phase and group delay with respect to the real and complex conjugate pole radiuses of filters $A_1$ and $A_2$ (for convenience all represented here by $\boldsymbol{r}$), are:

$$\frac{\partial|H(\omega)|^2}{\partial\boldsymbol{r}} = -\frac{1}{2}\sin(\phi_1(\omega) - \phi_2(\omega))\left\{\frac{\partial\phi_1(\omega)}{\partial\boldsymbol{r}} - \frac{\partial\phi_2(\omega)}{\partial\boldsymbol{r}}\right\}$$
$$\frac{\partial\phi_H(\omega)}{\partial\boldsymbol{r}} = \frac{1}{2}\left\{\frac{\partial\phi_1(\omega)}{\partial\boldsymbol{r}} + \frac{\partial\phi_2(\omega)}{\partial\boldsymbol{r}}\right\}$$
$$\frac{\partial T(\omega)}{\partial\boldsymbol{r}} = -\frac{1}{2}\left\{\frac{\partial^2\phi_1(\omega)}{\partial\boldsymbol{r}\partial\omega} + \frac{\partial^2\phi_2(\omega)}{\partial\boldsymbol{r}\partial\omega}\right\}$$

The partial derivatives with respect to the filter pole angles, $\theta$, are similar. Appendix I.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 8.1.2. 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-10 % Tolerance on constraints
n=2000 % Frequency points across the band
ma=5 % Allpass model filter A denominator order
K=10 % Scale factor
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
```

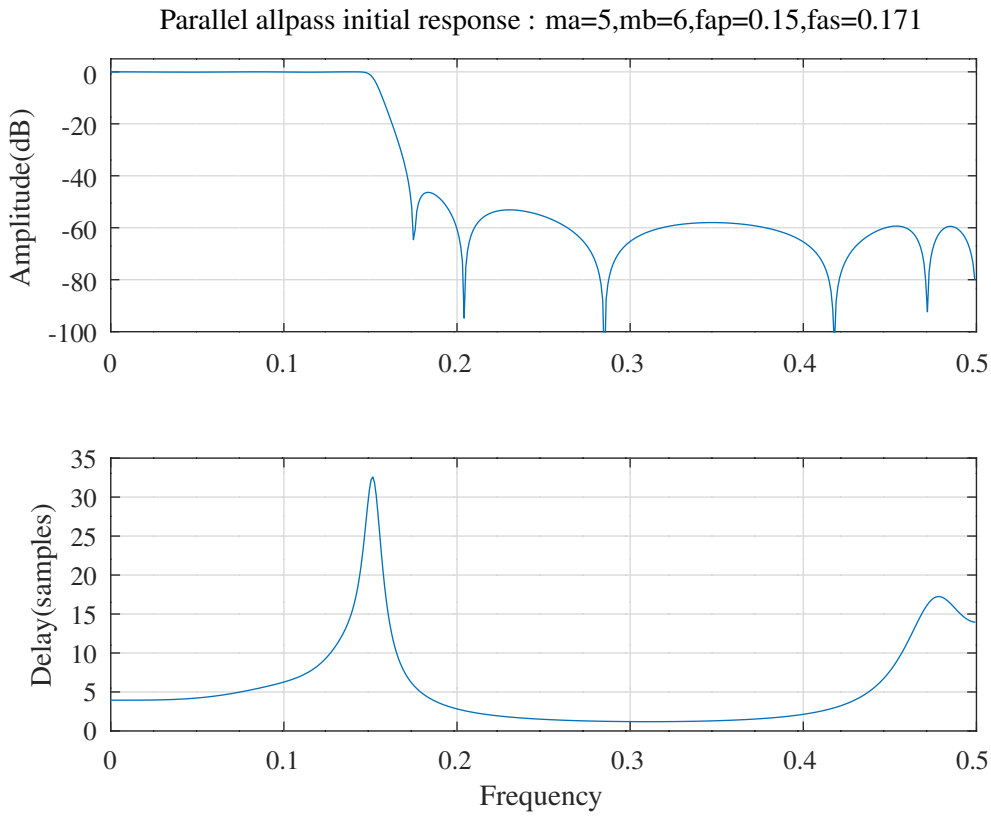



Figure 10.19: Parallel all-pass filters:Initial response found with the WISE barrier function

```
fap=0.15 % Pass band amplitude response edge
dBap=0.020000 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas=0.171 % Stop band amplitude response edge
dBas=84.020000 % Stop band amplitude response ripple
Was_mmse=10000 % Stop band amplitude response weight(MMSE)
Was_pcls=1e-06 % Stop band amplitude response weight(PCLS)
rho=0.999000 % Constraint on allpass pole radius
```

Compare this filter with the elliptic filter shown in Figure 10.14. The response of that filter is slightly wider than the specification. The response of the filter shown here is intended to match that slight extra width in the transition band.

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 8.1.5. The response of the initial filter is shown in Figure 10.19. The pole-zero plot of the initial filter is shown in Figure 10.20.

The script first runs an MMSE pass on the initial filter. The squared-amplitude error weighting function increases linearly near the band edges. The resulting MMSE optimised response is shown in Figure 10.21.

The PCLS pass does not attempt to minimise the stop-band response minimum mean-squared squared-amplitude error. Instead the stop-band squared-amplitude response is controlled by the PCLS constraints. In this case, the desired stop-band attenuation is more than $80dB$ and the filter response is scaled to make the stop-band squared-amplitude greater than the machine precision. The default SeDuMi *pars.eps* is of the same order as the desired stop-band attenuation and must be reduced to avoid the PCLS algorithm cycling between solutions. The denominator polynomials of the two all-pass filters are:

```
Da1 = [   1.0000000000,  -2.9458870286,   4.3150780689,  -3.5613969661, ...
          1.6660588928,  -0.3494643032 ]';
```

and

```
Db1 = [   1.0000000000,  -3.5168083250,   6.3336261743,  -6.8250850574, ...
          4.6401372835,  -1.8616131086,   0.3513270327 ]';
```

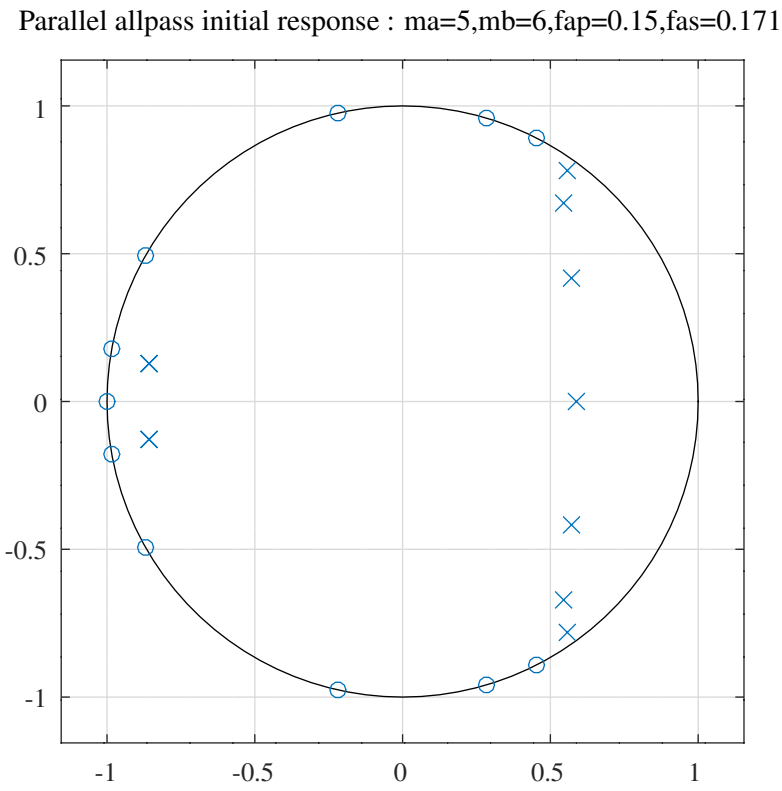



Figure 10.20: Parallel all-pass filters:Pole zero plot of the initial response found with the WISE barrier function

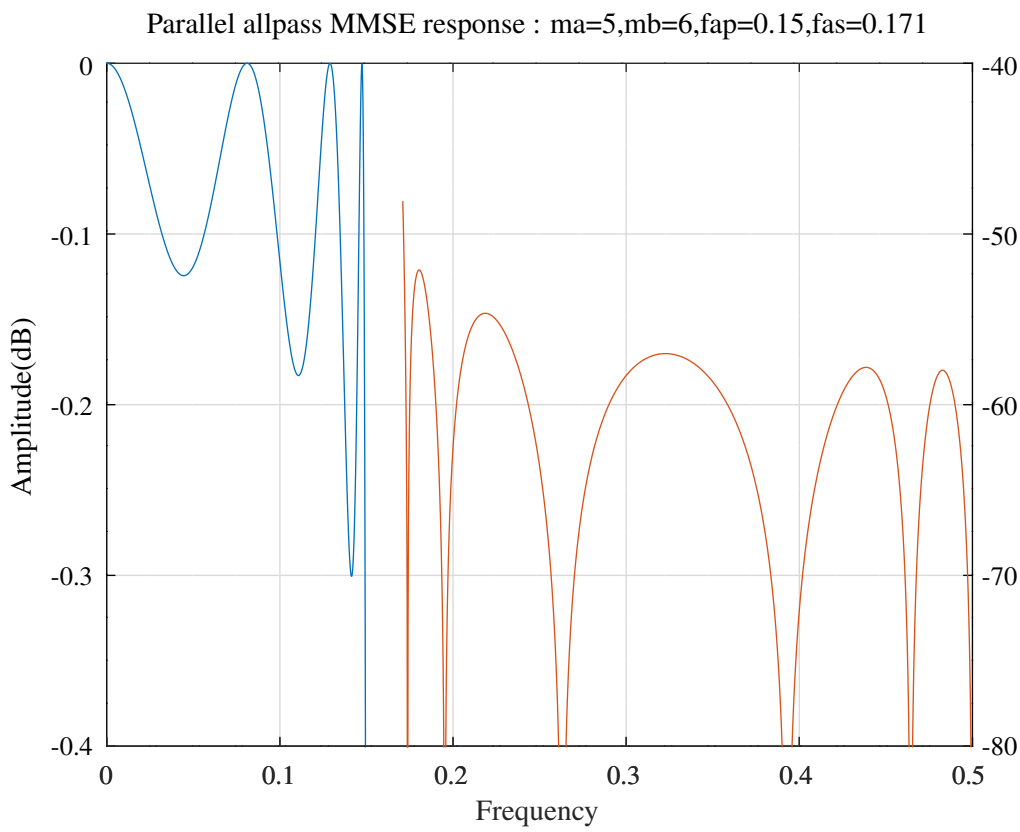



Figure 10.21: Parallel all-pass filters:Response found with MMSE SOCP optimisation

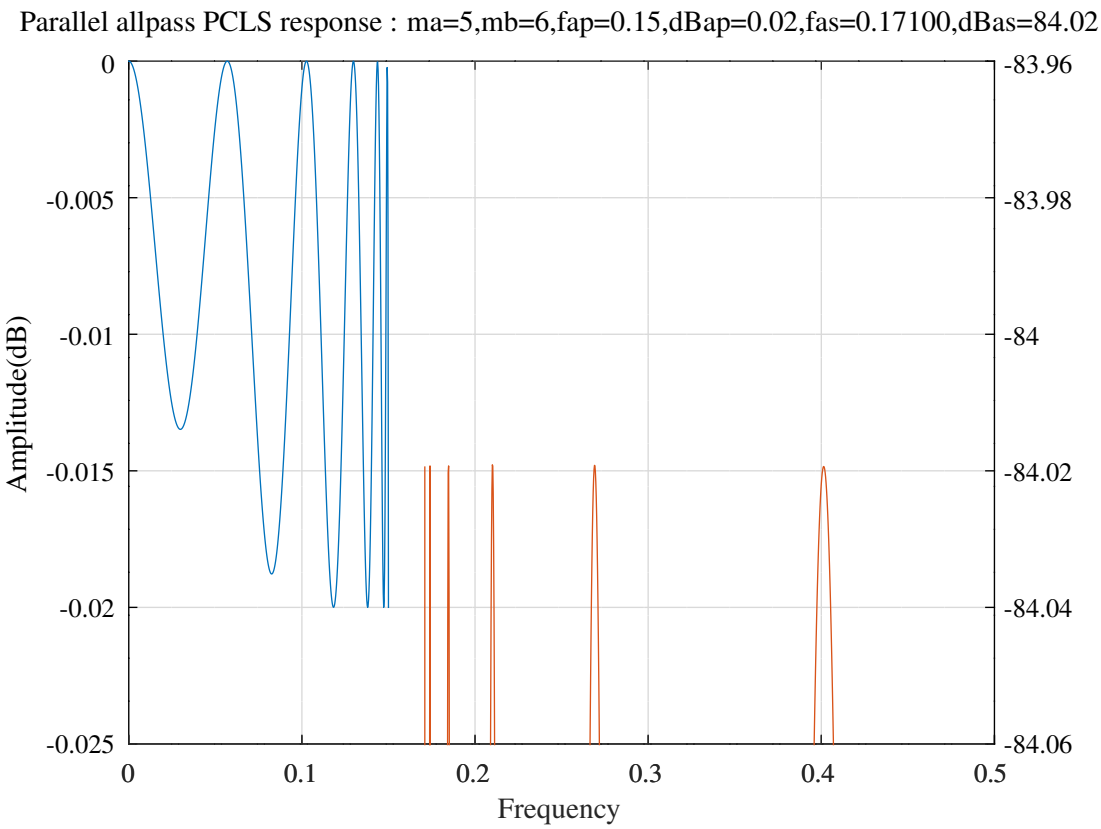



Figure 10.22: Parallel all-pass filters:Response after PCLS SOCP optimisation

The corresponding filter numerator and denominator polynomials of the overall filter are:

```
Nab1 = [   0.0009313647,  -0.0007624967,   0.0031298791,  -0.0007399225, ...
           0.0035413024,   0.0014617085,   0.0014617085,   0.0035413024, ...
          -0.0007399225,   0.0031298791,  -0.0007624967,   0.0009313647 ]';
```

and

```
Dab1 = [   1.0000000000,  -6.4626953535,  21.0088242697, -44.2199316898, ...
          66.2669676160, -73.7469393212,  61.9460139691, -39.1777164015, ...
          18.2618124296,  -5.9743344454,   1.2358988549,  -0.1227762567 ]';
```

The PCLS filter response is shown in Figure 10.22 and the pole-zero plot is shown in Figure 10.23. The pole-zero plots of the allpass filters are shown in Figure 10.24 and Figure 10.25.

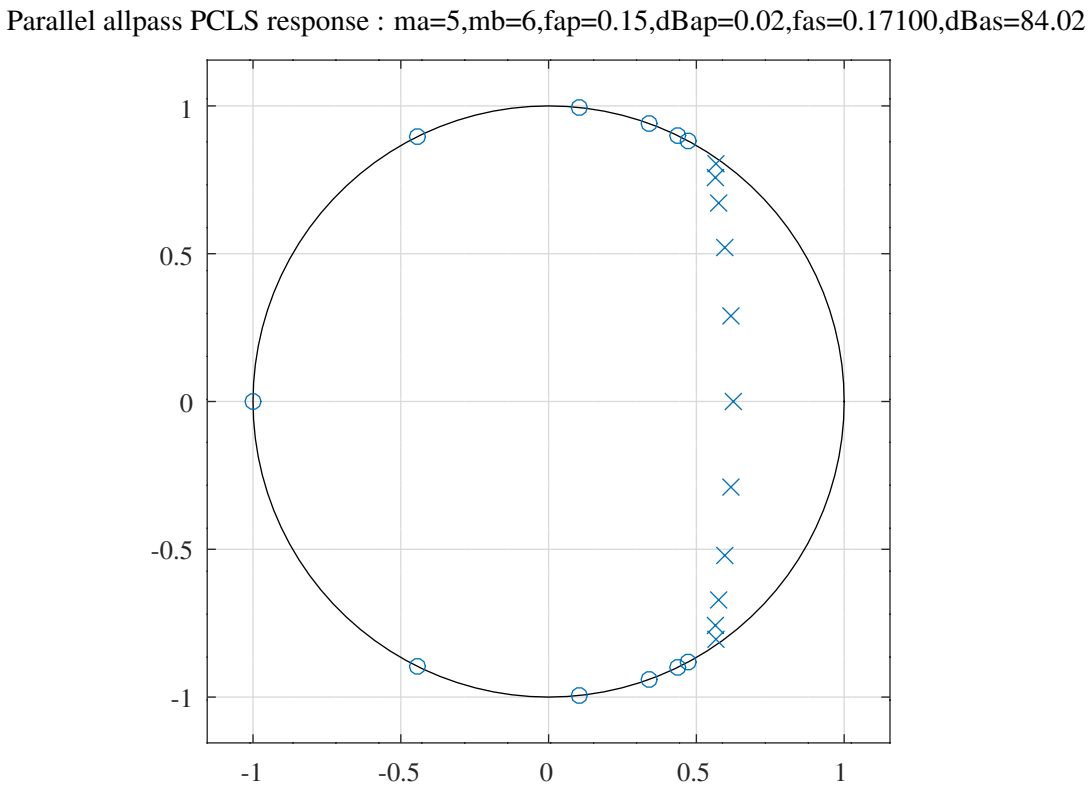



Figure 10.23: Parallel all-pass filters:Pole-zero plot after PCLS SOCP optimisation

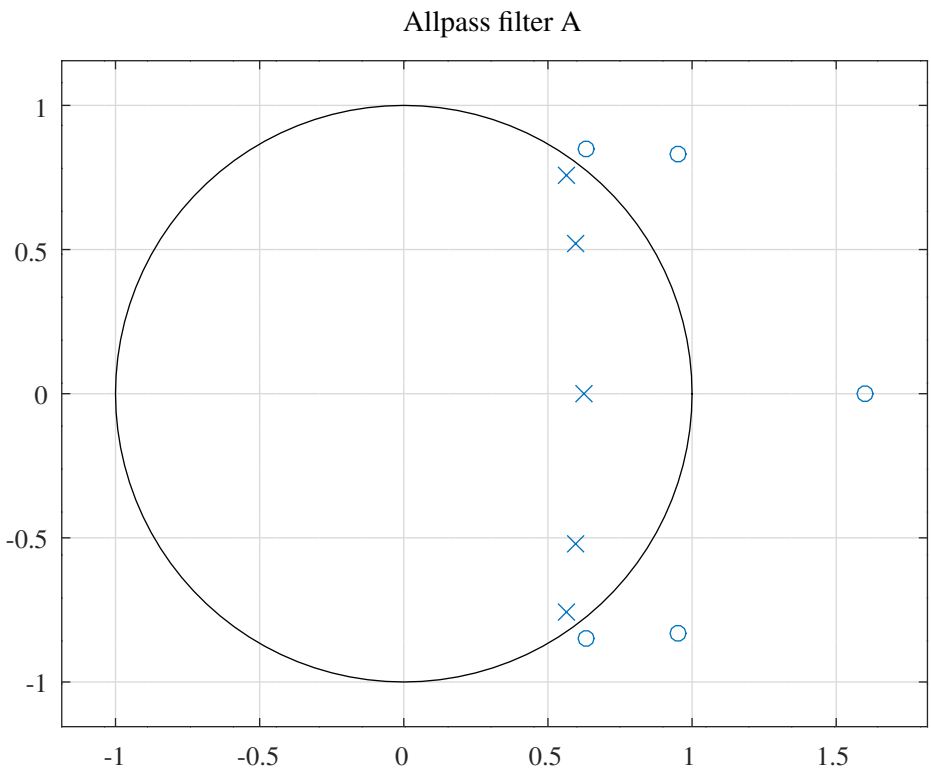



Figure 10.24: Parallel all-pass filters:Pole-zero plot of the A allpass filter branch after PCLS SOCP optimisation

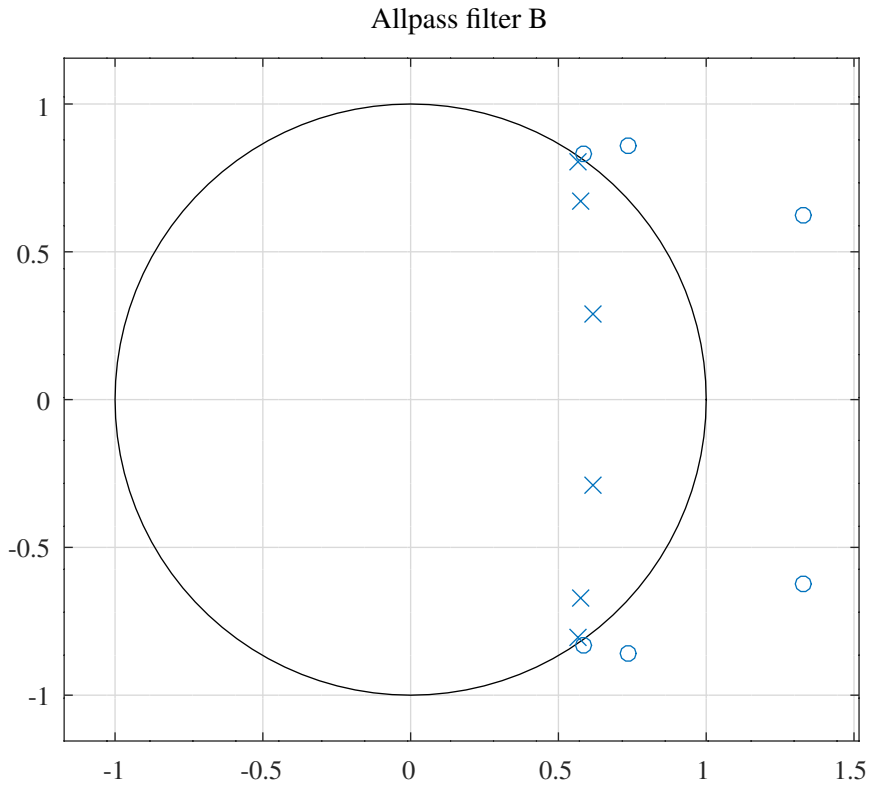



Figure 10.25: Parallel all-pass filters:Pole-zero plot of the B allpass filter branch after PCLS SOCP optimisation

**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 8.1.2. 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=1000 % 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 8.1.5. The response of the initial filter is shown in Figure 10.26. The final filter response is shown in Figure 10.27 with passband detail shown in Figure 10.28.The pole-zero plots of the branch allpass filters are shown in Figures 10.29 and 10.30. The denominator polynomials of the two all-pass filters are

```
Da1 = [   1.0000000000,  -0.0624623803,  -0.0987507617,   0.2454359371, ...
         -0.4989263856,  -0.0471158893,   0.2552491131,  -0.1122120859, ...
          0.0976544253,   0.0565493548,  -0.0821647744,   0.0195203198 ]';
```

and

```
Db1 = [   1.0000000000,  -0.5897574080,   0.3150914969,   0.5150660170, ...
         -0.5594870646,   0.3325510346,   0.1597362410,  -0.3412605504, ...
          0.2018379612,  -0.0196506879,  -0.1023746838,   0.0800099213, ...
         -0.0426851505 ]';
```

The corresponding filter numerator and denominator polynomials of the overall filter are

```
Nab1 = [  -0.0115824153,  -0.0055004451,   0.0040001116,   0.0084176120, ...
           0.0244405452,   0.0094118676,  -0.0229220791,  -0.0292373835, ...
          -0.0131350057,   0.0516192817,   0.1475425400,   0.2036577979, ...
           0.2036577979,   0.1475425400,   0.0516192817,  -0.0131350057, ...
          -0.0292373835,  -0.0229220791,   0.0094118676,   0.0244405452, ...
           0.0084176120,   0.0040001116,  -0.0055004451,  -0.0115824153 ]';
```

and

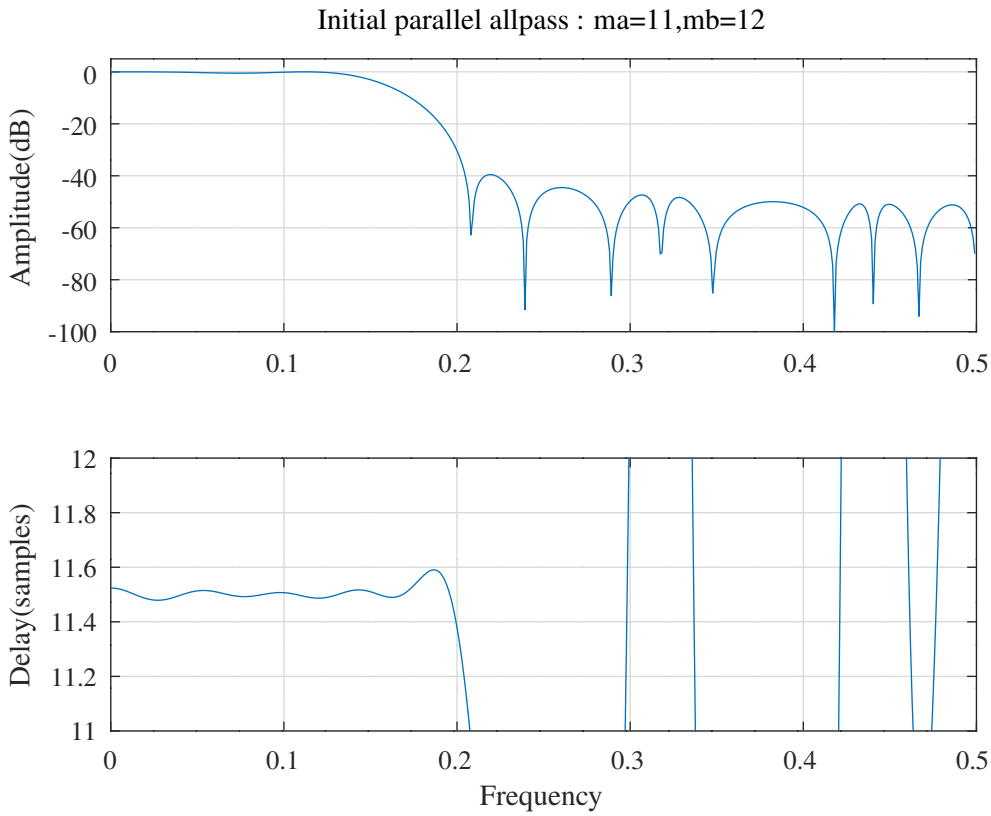



Figure 10.26: Parallel all-pass filters with flat pass-band delay:Initial response found with the WISE barrier function

```
Dab1 = [   1.0000000000,  -0.6522197883,   0.2531783866,   0.7990595826, ...
          -1.2664488871,   0.6410991864,   0.4464583925,  -1.0559689398, ...
           0.7881340317,  -0.0038409652,  -0.5855518792,   0.5843799434, ...
          -0.1864747809,  -0.1704307177,   0.2560971897,  -0.1357745045, ...
          -0.0126198768,   0.0745738156,  -0.0542274872,   0.0123684262, ...
           0.0083841207,  -0.0109862014,   0.0050690350,  -0.0008332278 ]';
```

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 10.23, and the pole-zero plot for the elliptic filter example, Figure 10.15.

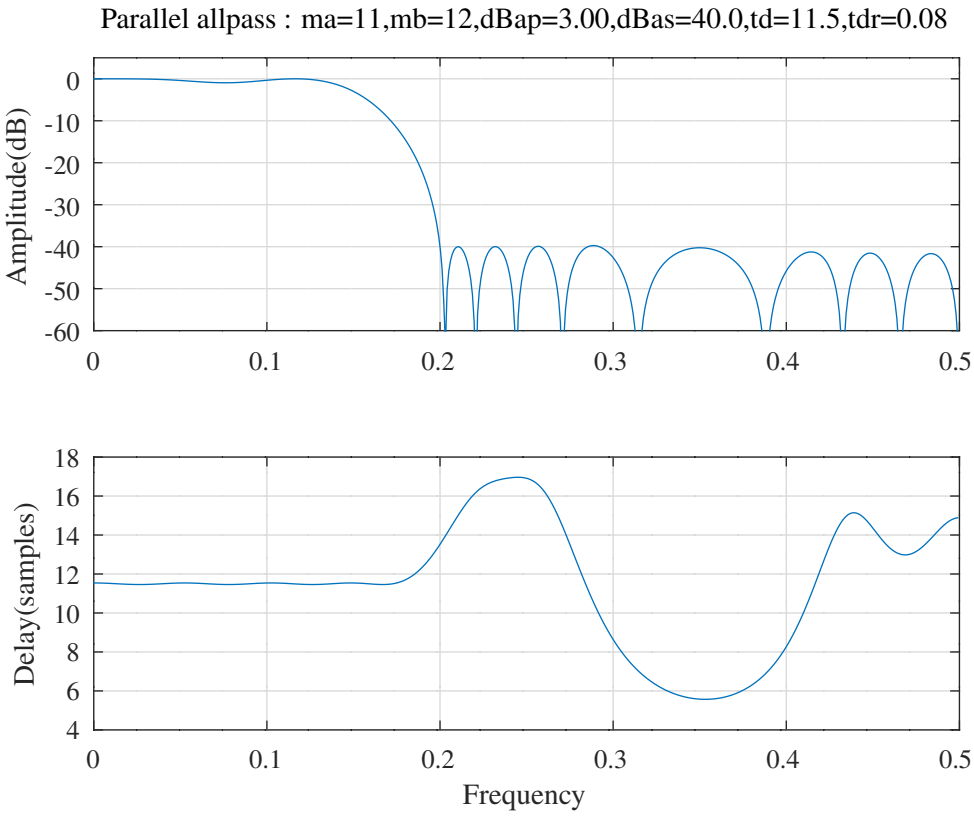



Figure 10.27: Parallel all-pass filters with flat pass-band delay:Response after PCLS SOCP optimisation

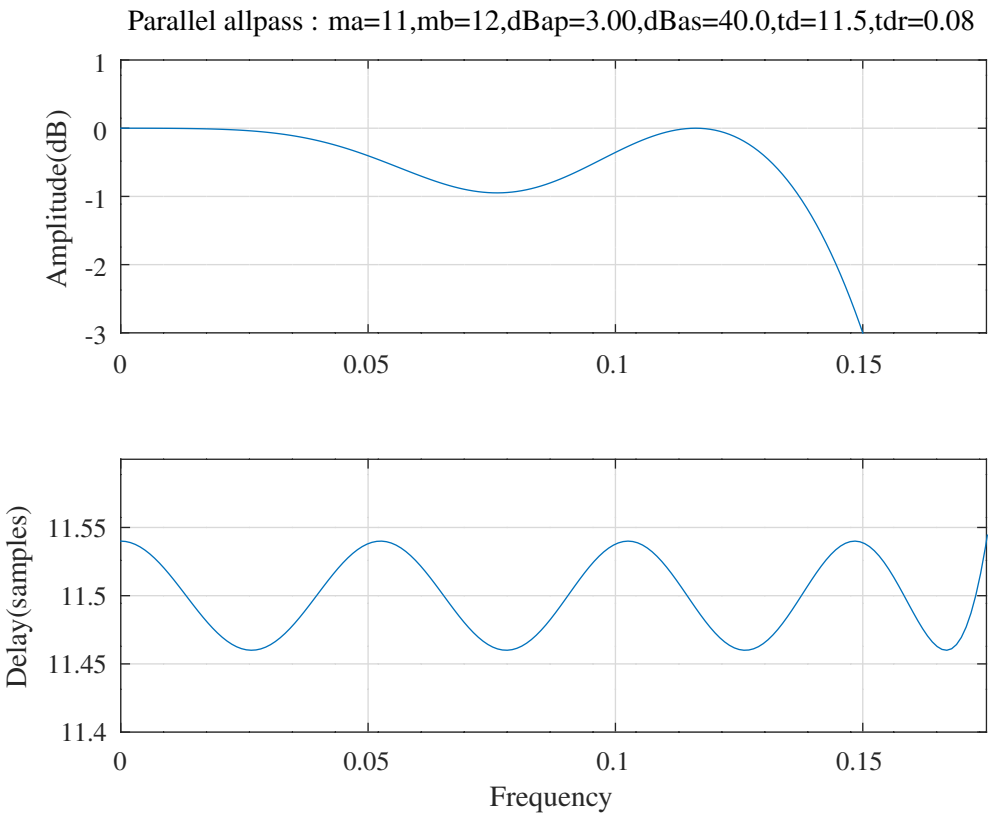



Figure 10.28: Parallel all-pass filters with flat pass-band delay:Pass-band response after PCLS SOCP optimisation

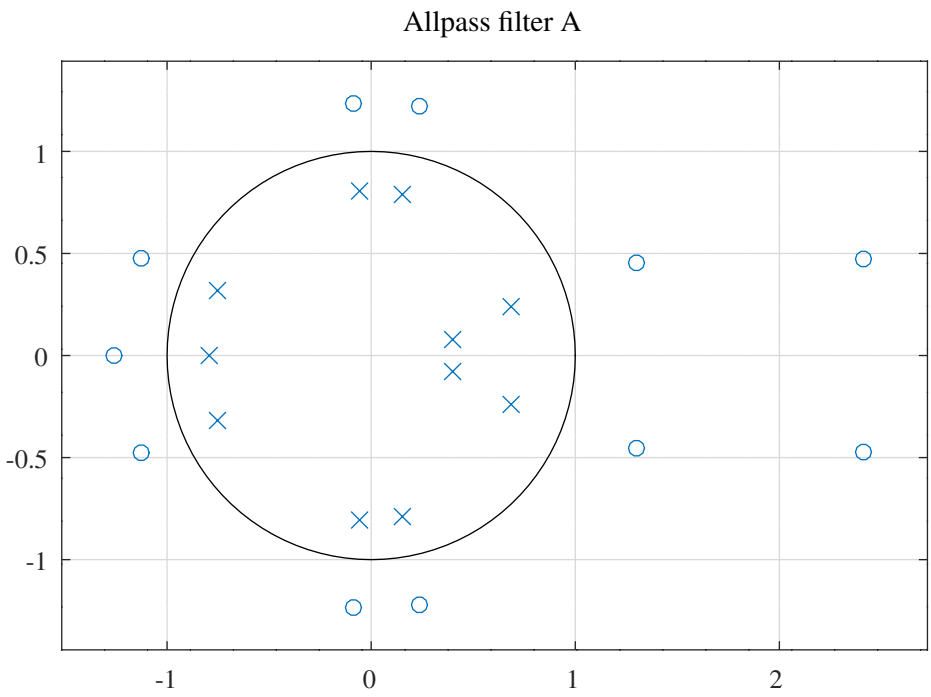



Figure 10.29: Parallel all-pass filters:Pole-zero plot of the A allpass filter branch after PCLS SOCP optimisation

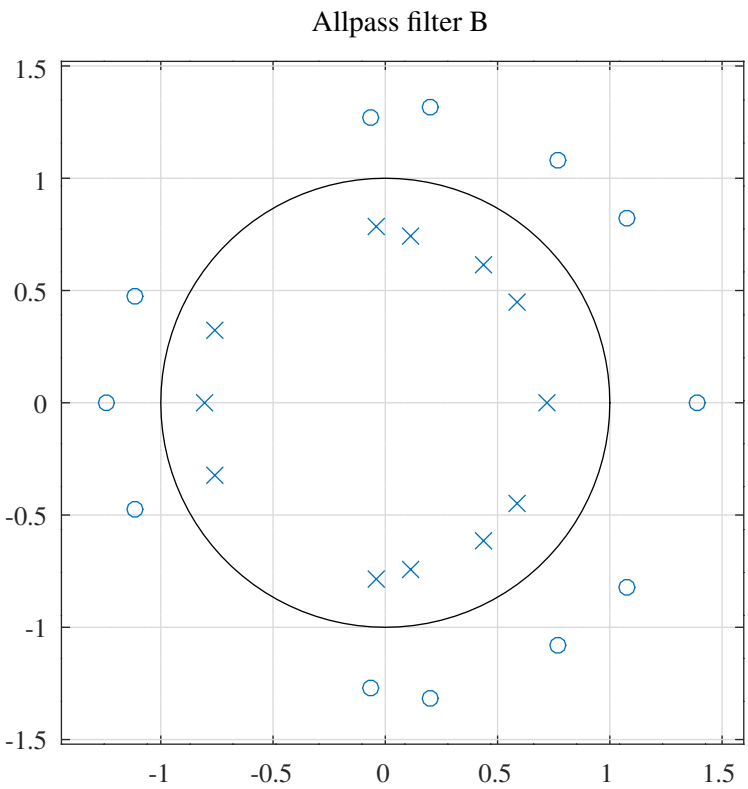



Figure 10.30: Parallel all-pass filters with flat pass-band delay:Pole-zero plot of the B allpass filter branch after PCLS SOCP optimisation

**Design of an IIR filter as the difference of two all-pass filters each represented in pole-zero form with constraints on the group delay**

If the filter of Equation 10.1 is rewritten as the difference of two all-pass filters:

$$H(z) = \frac{e^{\imath\phi_1(z)} - e^{\imath\phi_2(z)}}{2}$$

then the squared magnitude response and phase of the frequency response, $H(\omega)$, are, with simple trigonometry

$$|H(\omega)|^2 = \frac{1 - \cos(\phi_1(\omega) - \phi_2(\omega))}{2}$$
$$\phi_H(\omega) = \frac{\phi_1(\omega) + \phi_2(\omega)}{2} + \frac{\pi}{2}$$

Clearly, the group delay response is unchanged:

$$T(\omega) = -\frac{1}{2}\left[\frac{\partial\phi_1(\omega)}{\partial\omega} + \frac{\partial\phi_2(\omega)}{\partial\omega}\right]$$

The Octave script *parallel_allpass_socp_slb_bandpass_test.m* uses the SeDuMi SOCP solver to design a band-pass filter consisting of the difference of two all-pass filters. The script calls the Octave function *parallel_allpass_slb* to perform the PCLS design of the filter in terms of the all-pass filter pole locations with constraints on the filter group-delay. The PCLS algorithm of *Selesnick*, *Lang* and *Burrus* was reviewed in Section 8.1.2. 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
difference=1 % Use difference of all-pass filters
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-07 % Tolerance on constraints
rho=0.999000 % Constraint on allpass pole radius
n=1000 % Frequency points across the band
ma=10 % Allpass model filter A denominator order
Va=0 % 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=10 % Allpass model filter B denominator order
Vb=0 % 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
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
dBap=2.000000 % Pass band amplitude response ripple(dB)
Wap=1 % Pass band amplitude response weight
Watl=0.01 % Lower transition band amplitude response weight
Watu=0.01 % Upper transition band amplitude response weight
fasl=0.05 % Stop band amplitude response lower edge
fasu=0.25 % Stop band amplitude response upper edge
dBas=50.000000 % Stop band amplitude response ripple(dB)
Wasl=10000 % Lower stop band amplitude response weight
Wasu=10000 % Upper stop band amplitude response weight
ftpl=0.09 % Pass band group-delay response lower edge
ftpu=0.21 % Pass band group-delay response upper edge
td=16.000000 % Pass band nominal group-delay response (samples)
tdr=0.080000 % Pass band group-delay response ripple(samples)
Wtp=1 % Pass band group-delay response weight
```

The original initial filter was designed by the Octave script *tarczynski_parallel_allpass_bandpass_test.m* using the *WISE* method described in Section 8.1.5. The response of the initial filter is shown in Figure 10.31.

The denominator polynomials of the all-pass filters in the final band-pass filter are:

```
Da1 = [   1.0000000000,  -2.8742837201,   5.2338605551,  -6.5181914021, ...
          6.0655036371,  -4.2287488199,   2.1669300660,  -0.7445027723, ...
          0.1429665316,   0.0067112965,   0.0000741227 ]';
```

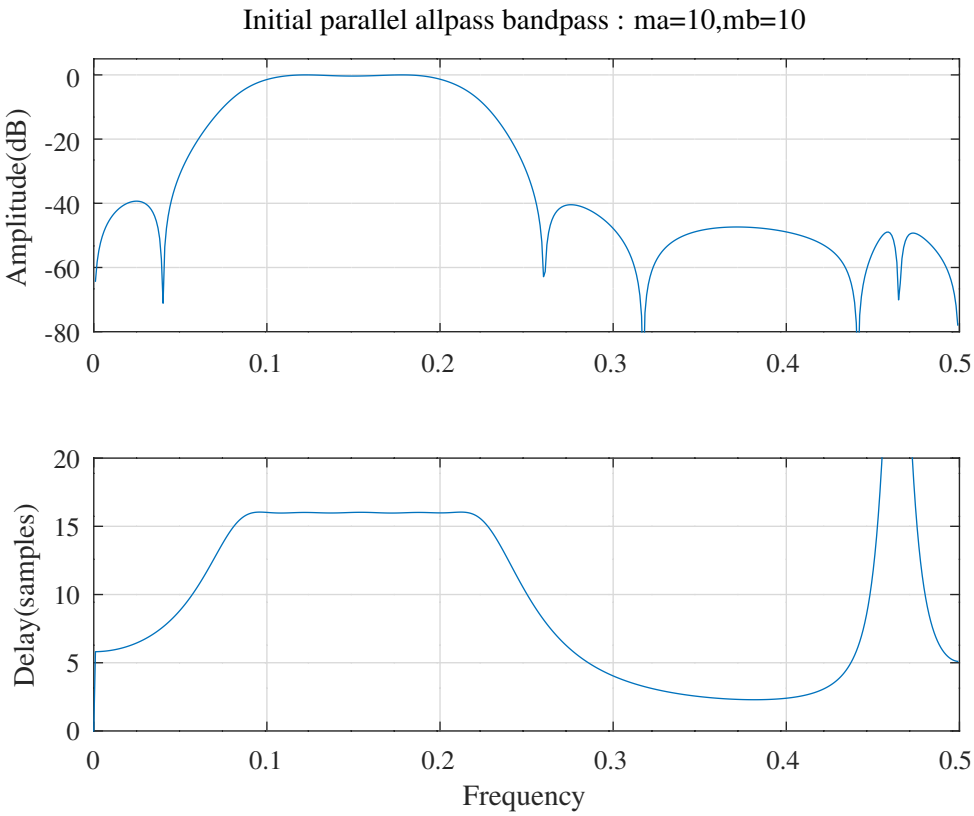



Figure 10.31: Band-pass filter implemented as the difference of two all-pass filters:Initial response found with the WISE barrier function

```
Db1 = [   1.0000000000,  -3.4842183042,   6.5101793566,  -8.1598482266, ...
          7.5476840654,  -5.1780922137,   2.5645709730,  -0.8349265309, ...
          0.1515412213,  -0.0055904333,   0.0000659383 ]';
```

The final filter response is shown in Figure 10.32 and Figure 10.33. The phase responses of the parallel all-pass filters are compared in Figure 10.34.

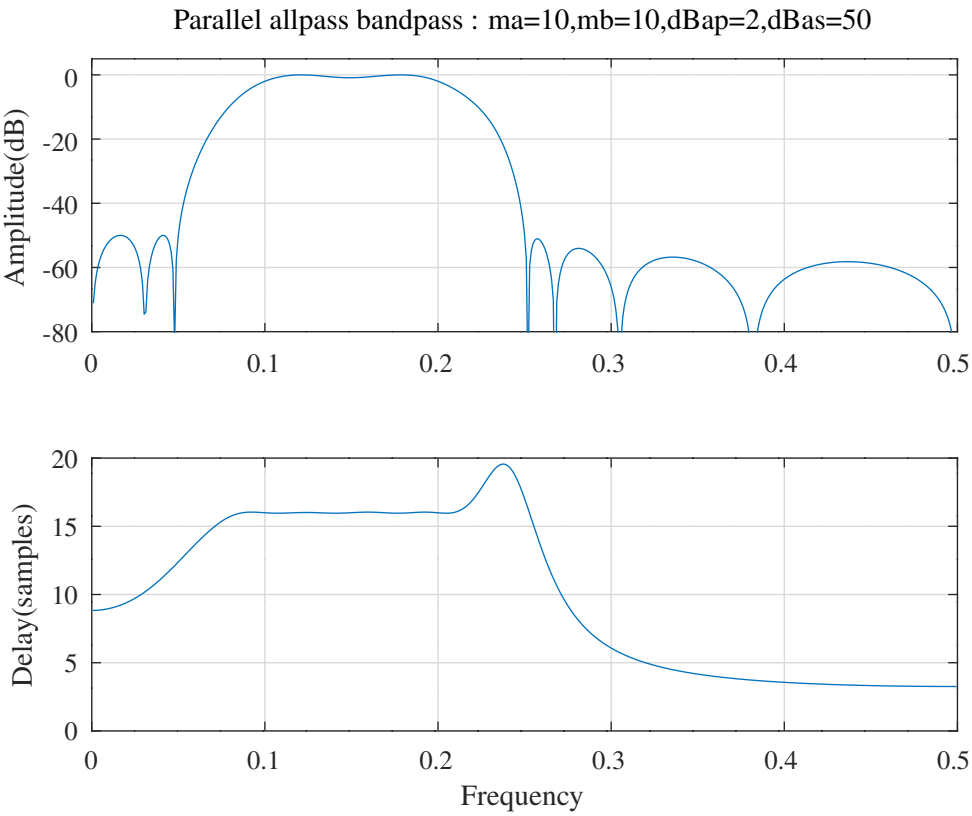



Figure 10.32: Band-pass filter implemented as the difference of two all-pass filters:Response after PCLS SOCP optimisation

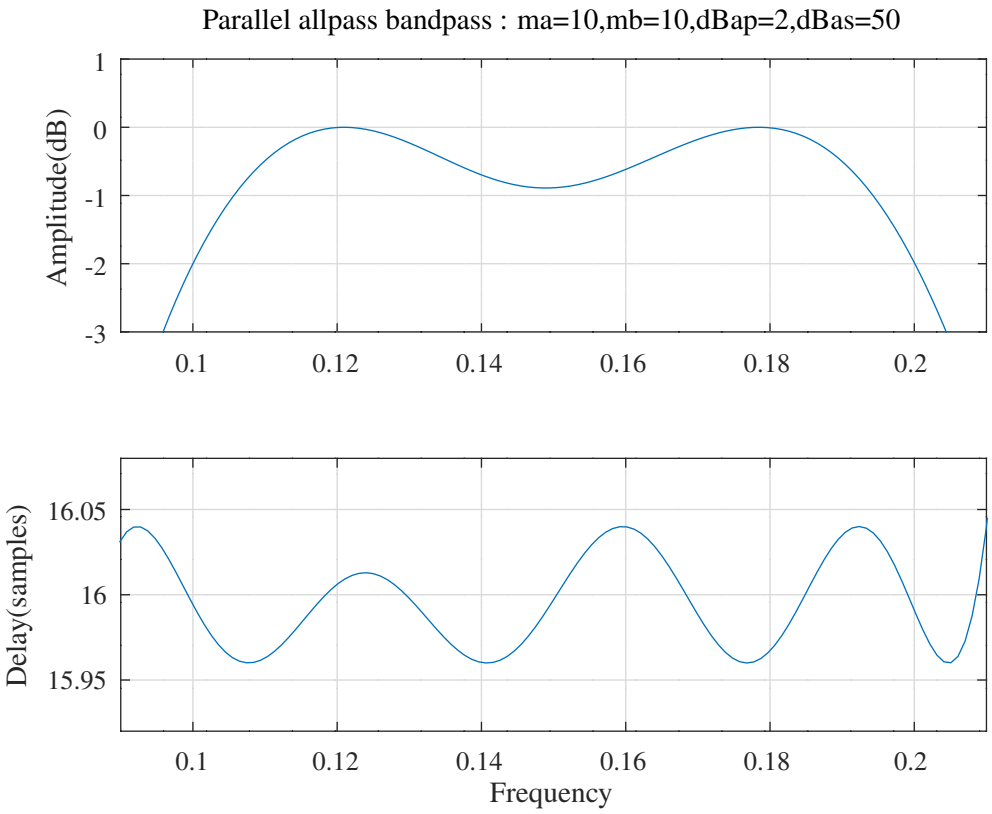



Figure 10.33: Band-pass filter implemented as the difference of two all-pass filters :Pass-band response after PCLS SOCP optimisation.

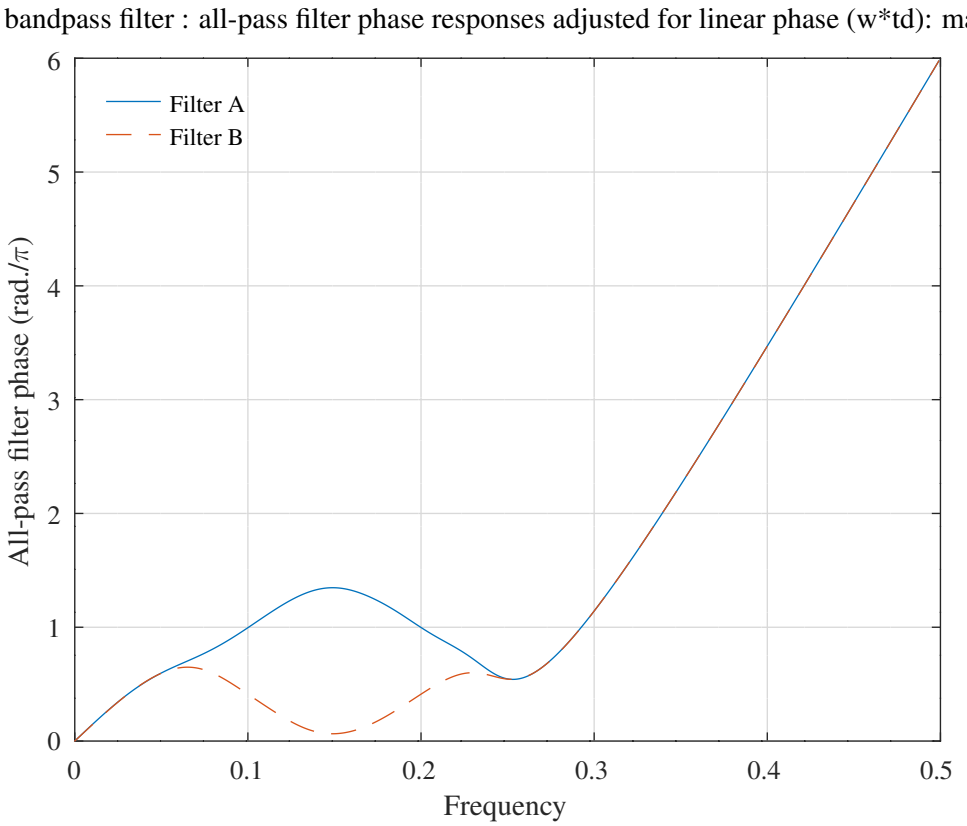



Figure 10.34: Band-pass filter implemented as the difference of two all-pass filters :Comparison of all-pass filter phase responses after PCLS SOCP optimisation.

**Design of an IIR filter as the difference of two all-pass filters each represented in pole-zero form with constraints on the group delay and phase**

The Octave script *parallel_allpass_socp_slb_bandpass_hilbert_test.m* uses the SeDuMi SOCP solver to design a band-pass Hilbert filter consisting of the difference of two all-pass filters. The script calls the Octave function *parallel_allpass_slb* to perform the PCLS design of the filter in terms of the all-pass filter pole locations with constraints on the filter group-delay and phase. The filter specification is

```
polyphase=0 % Use polyphase combination
difference=1 % Use difference of all-pass filters
tol=1e-05 % Tolerance on coefficient update vector
ctol=1e-08 % Tolerance on constraints
rho=0.999 % Constraint on allpass pole radius
n=1000 % Frequency points across the band
ma=10 % Allpass model filter A denominator order
Va=0 % 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=10 % Allpass model filter B denominator order
Vb=0 % 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
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
dBap=0.04 % Pass band amplitude response ripple(dB)
Wap=1 % Pass band amplitude response weight
Watl=0.001 % Lower transition band amplitude response weight
Watu=0.001 % Upper transition band amplitude response weight
fasl=0.05 % Stop band amplitude response lower edge
fasu=0.25 % Stop band amplitude response upper edge
dBas=40 % Stop band amplitude response ripple(dB)
Wasl=200 % Lower stop band amplitude response weight
Wasu=200 % Upper stop band amplitude response weight
ftpl=0.12 % Pass band group-delay response lower edge
ftpu=0.18 % Pass band group-delay response upper edge
td=16 % Pass band nominal group-delay response (samples)
tdr=0.01 % Pass band group-delay response ripple(samples)
Wtp=10 % Pass band group-delay response weight
fppl=0.12 % Pass band phase response lower edge
fppu=0.18 % Pass band phase response upper edge
pd=1.5 % Pass band initial phase response (rad./pi)
pdr=0.0001 % Pass band phase response ripple(rad./pi)
Wpp=100 % Pass band phase response weight
```

The initial filter was designed by the Octave script *tarczynski_parallel_allpass_bandpass_hilbert_test.m* using the *WISE* method described in Section 8.1.5.

The denominator polynomials of the all-pass filters in the final band-pass filter are:

```
Da1 = [   1.0000000000,  -1.5181766376,   1.4711131135,   0.2171882520, ...
         -1.4544277184,   1.7696826986,  -0.6674113860,  -0.2080648270, ...
          0.5333339102,  -0.2872647848,   0.0895292465 ]';
```

```
Db1 = [   1.0000000000,  -2.1834645679,   1.9912560274,   0.2036956776, ...
         -2.1396408902,   2.2714713048,  -0.7976980572,  -0.3788993806, ...
          0.6642283127,  -0.3426265833,   0.0922993192 ]';
```

The final filter response is shown in Figure 10.35 and Figure 10.36. The phase response shown is adjusted for the nominal delay.

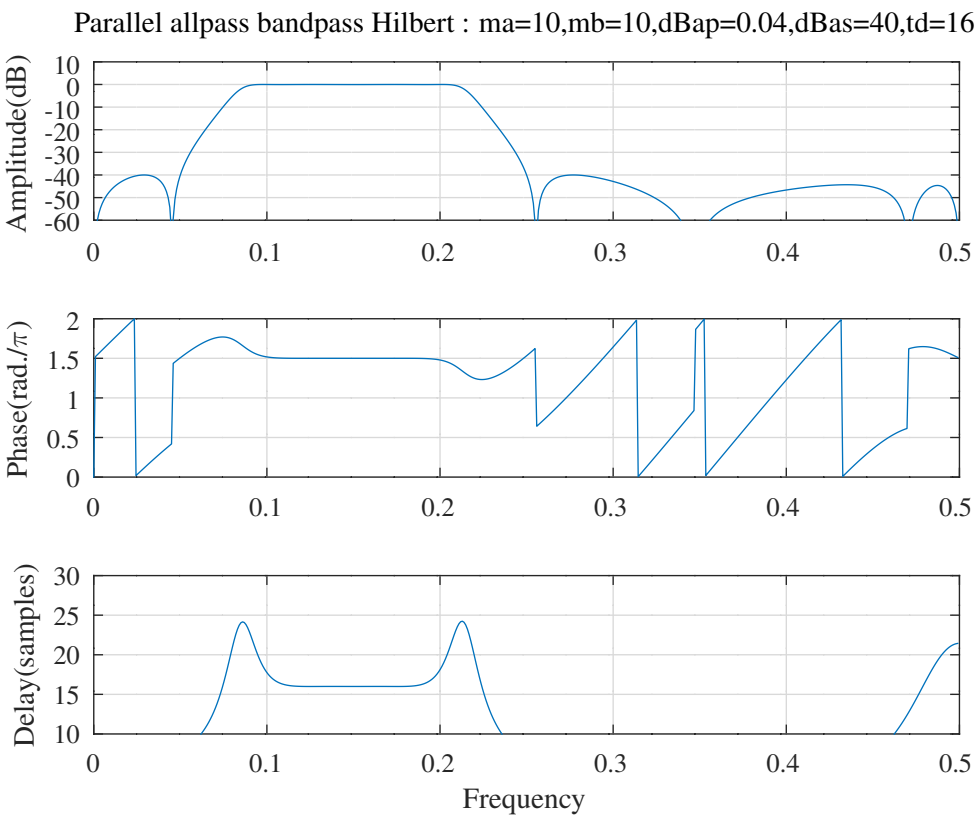



Figure 10.35: Band-pass Hilbert filter implemented as the difference of two all-pass filters:Response after PCLS SOCP optimisation. The phase response shown is adjusted for the nominal delay.

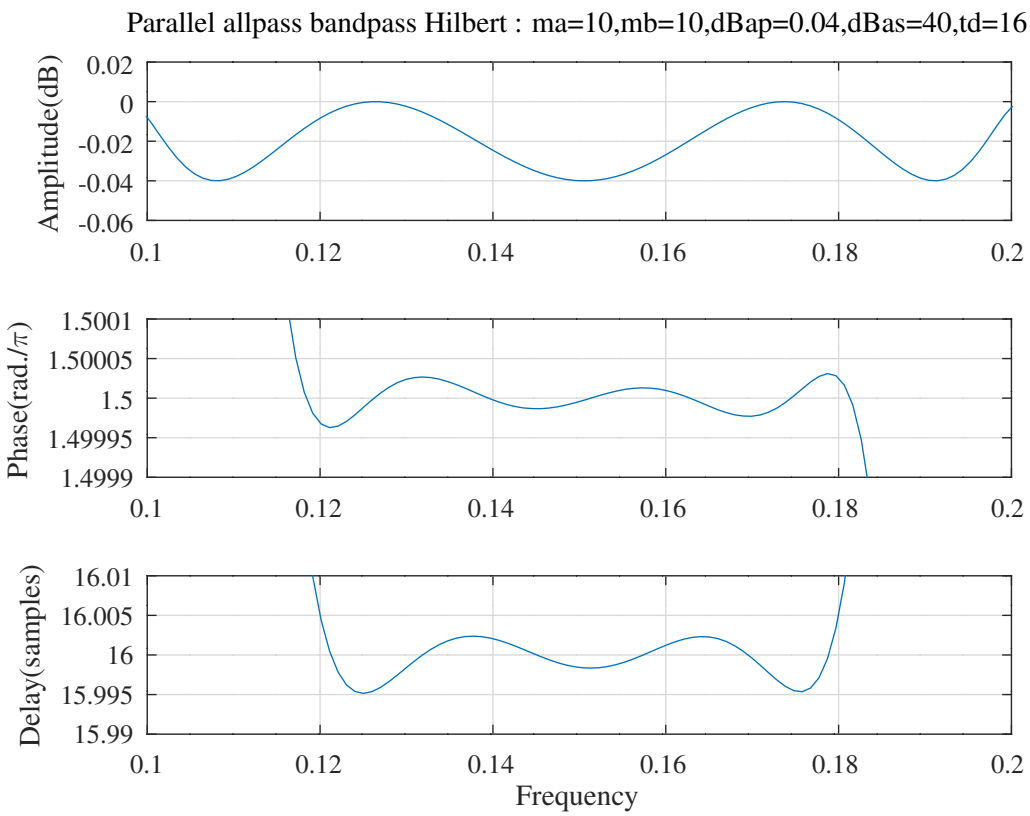



Figure 10.36: Band-pass Hilbert filter implemented as the difference of two all-pass filters:Pass-band response after PCLS SOCP optimisation. The phase response shown is adjusted for the nominal delay.

### 10.2.4 Design of an IIR filter as the sum of a delay and an all-pass filter represented in pole-zero form

*Kunold* [69] 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 8.1.2. 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
parallel_allpass_delay_flat_delay=0
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 8.1.5. The response of the initial filter is shown in Figure 10.37 and the pole-zero plot of the initial filter is shown in Figure 10.38.

The final filter response is shown in Figure 10.39 with pass-band and stop-band details shown in Figure 10.40. The pole-zero plot of the all-pass branch of the filter is shown in Figure 10.41. The denominator polynomial of the final all-pass filter is

```
Da1 = [   1.0000000000,  -0.5306436742,   0.3600382311,   0.1923578941, ...
          0.0374581441,  -0.0531783005,  -0.0703135828,  -0.0408241865, ...
         -0.0025374765,   0.0196080436,   0.0216225925,   0.0129983991, ...
          0.0044921326 ]';
```

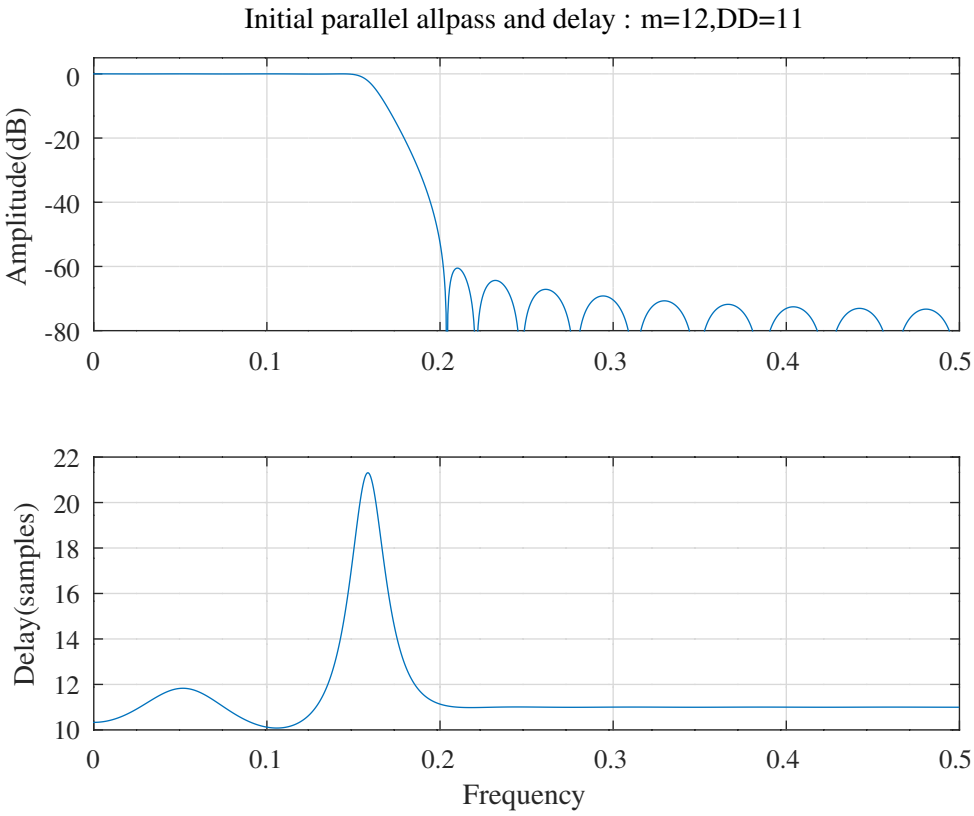



Figure 10.37: Parallel delay and all-pass filter:Initial response found with the WISE barrier function

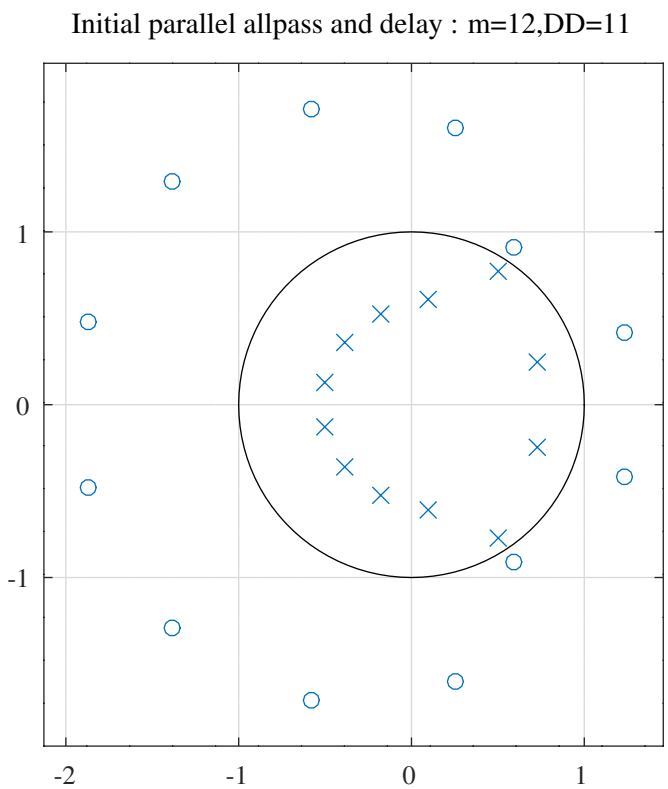



Figure 10.38: Parallel delay and all-pass filter:Pole zero plot of the initial filter found with the WISE barrier function

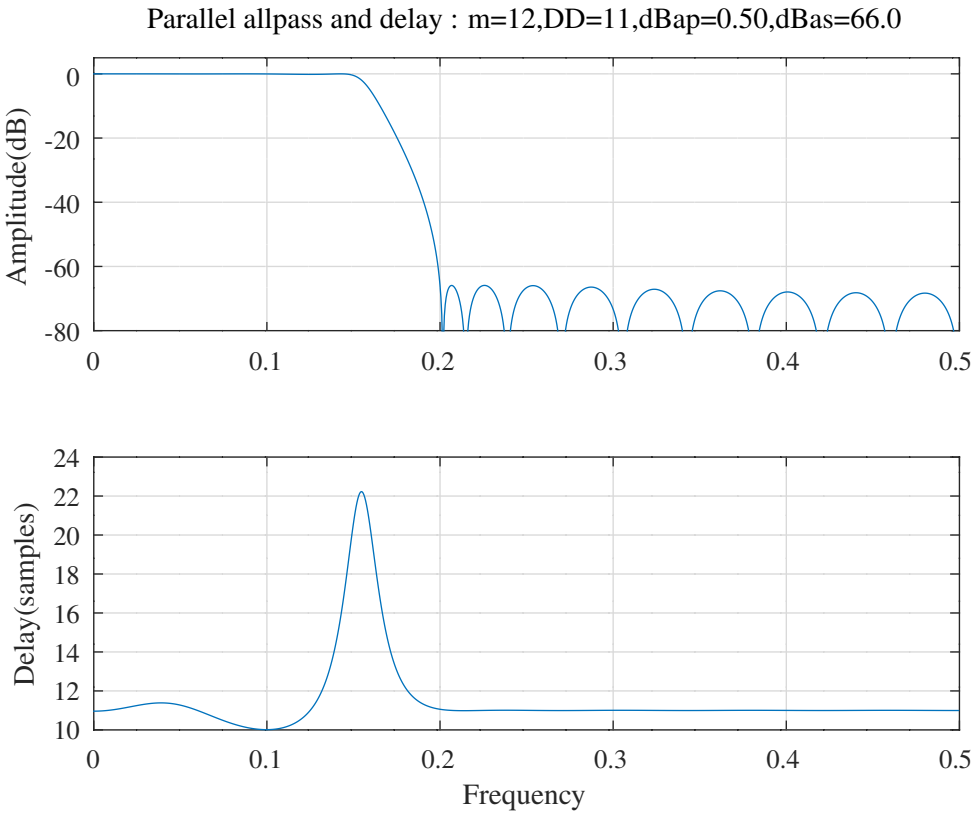



Figure 10.39: Parallel delay and all-pass filter:Response after PCLS SOCP optimisation

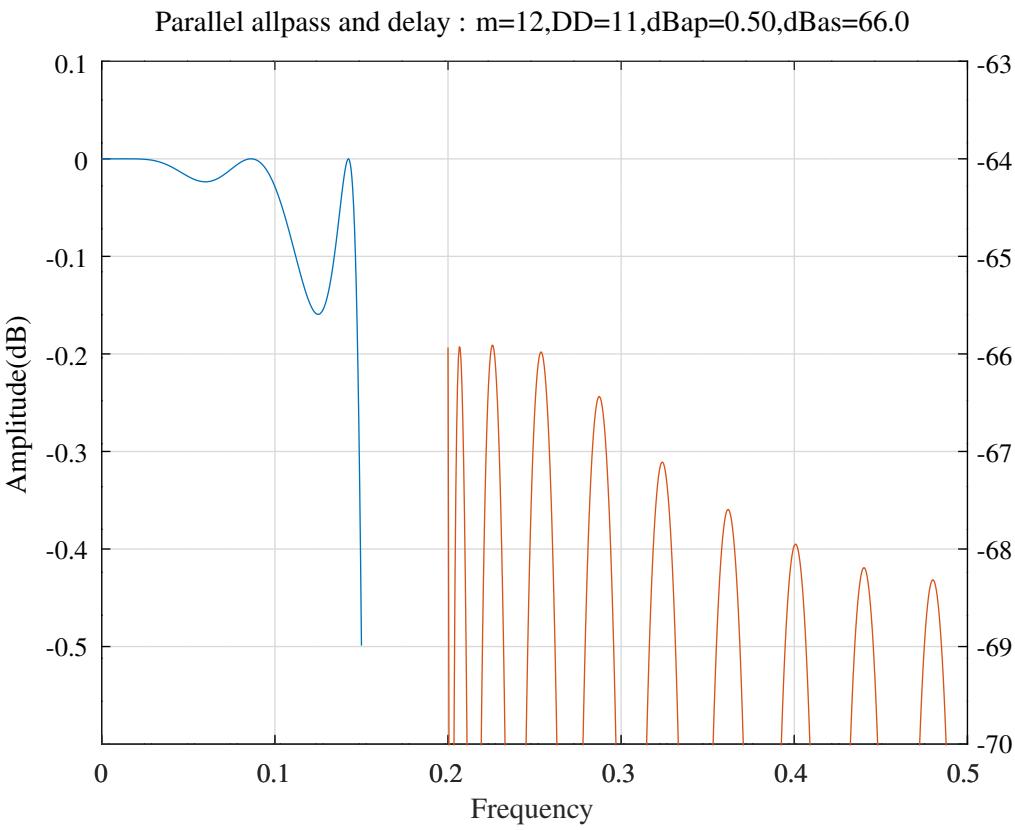



Figure 10.40: Parallel delay and all-pass filter:Detail of the pass-band and stop-band responses after PCLS SOCP optimisation

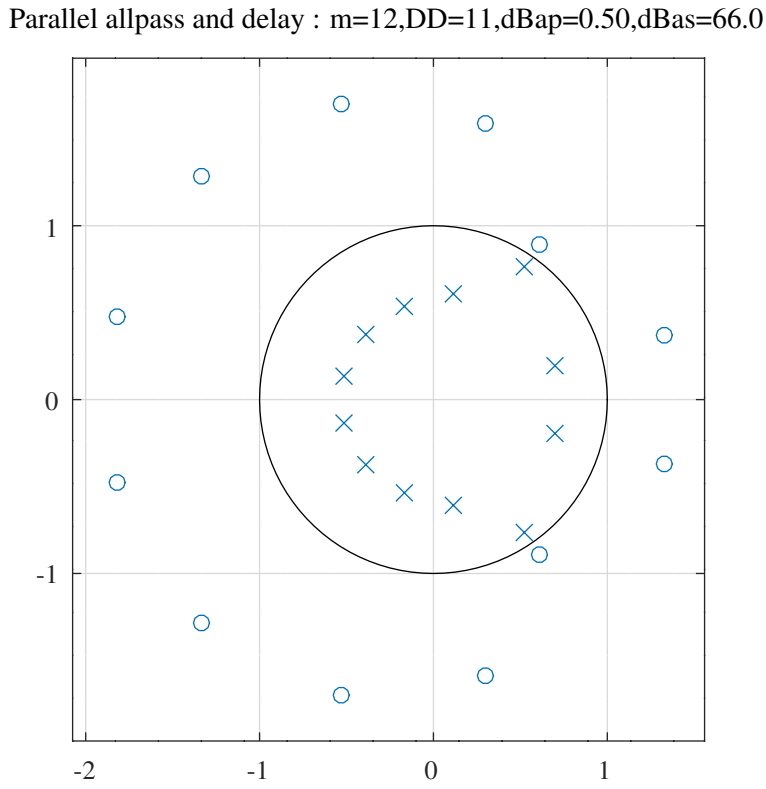



Figure 10.41: Parallel delay and all-pass filter:Pole-zero plot of the all-pass filter branch after PCLS SOCP optimisation

**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_sqp_slb_test.m* repeats the design example of Section 10.2.4 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=1e-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=0.040000 % 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=50 % Stop band amplitude response weight
rho=0.990000 % Constraint on allpass pole radius
dmax=0.005000 % Constraint on coefficent step-size
```

There are no group delay constraints.

The final filter response is shown in Figure 10.42 with pass-band and stop-band details shown in Figure 10.43. Figure 10.44 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,  -0.4845742265,   0.3792252718,   0.1790209825, ...
          0.0164202491,  -0.0580633367,  -0.0440054242,  -0.0054551080, ...
          0.0255811275,   0.0302915179,   0.0075732601,  -0.0005135438, ...
          0.0000173380 ]';
```

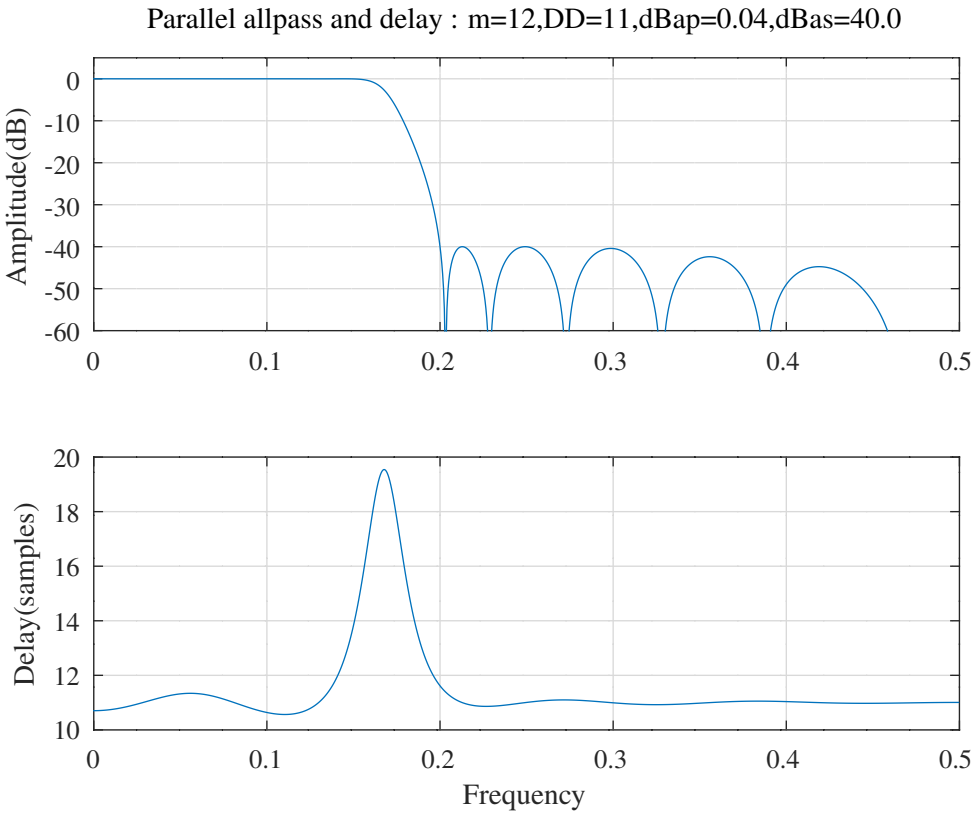



Figure 10.42: Parallel delay and all-pass filter:Response after PCLS SQP optimisation

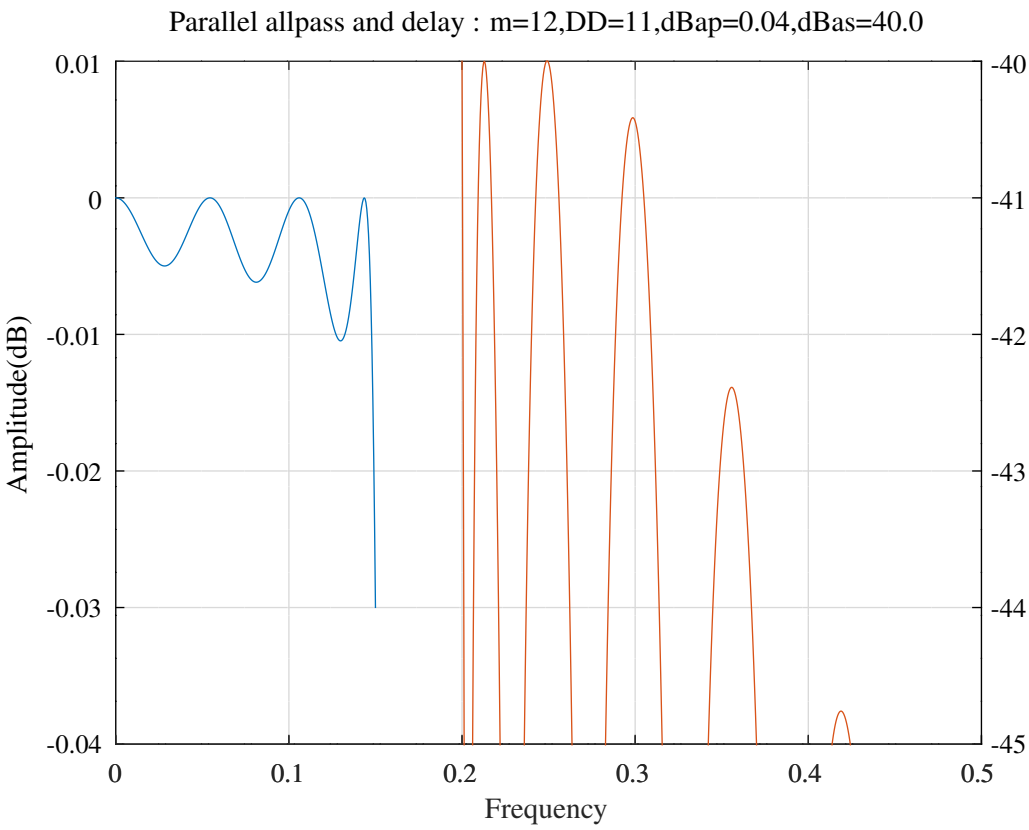



Figure 10.43: Parallel delay and all-pass filter:Detail of the pass-band and stop-band responses after PCLS SQP optimisation

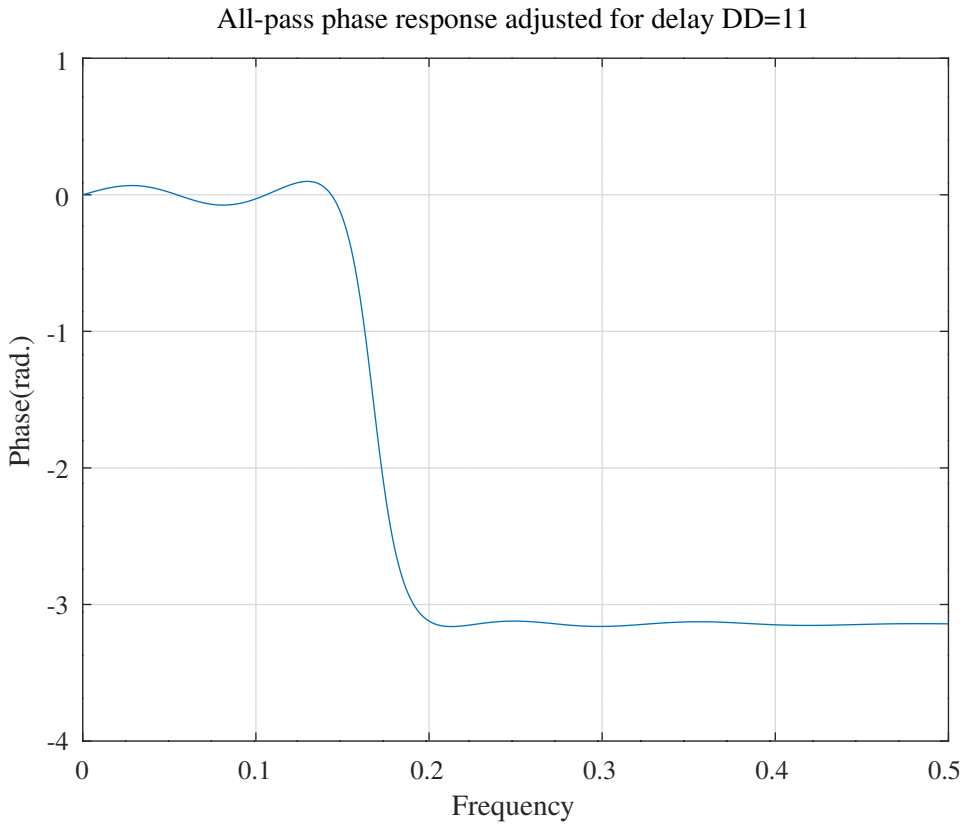



Figure 10.44: Parallel delay and all-pass filter:phase response of the all-pass filter branch after PCLS SQP optimisation. The response has been adjusted for the group delay of the fixed delay branch

### 10.2.5 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 10.1 can be represented in "polyphase" form [123, 124] as

$$H(z) = \frac{1}{R}\sum_{k=0}^{R-1} z^{-k} A_k\left(z^R\right) \tag{10.2}$$

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 8.1.2. 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
K=100 % Scale factor
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=81.000000 % Stop band amplitude response ripple
Was=0.01 % Stop band amplitude response weight
rho=0.999000 % Constraint on allpass pole radius
```

As in Section 10.2.3, 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 8.1.5. In this case the script has $flat_delay = false$. The response of the initial filter is shown in Figure 10.45. After PCLS optimisation with the *SeDuMi* SOCP solver the response is shown in Figure 10.46 and Figure 10.47. The denominator polynomials of the two all-pass filters are

```
Da1 = [   1.0000000000,   0.0000000000,   1.2164012841,   0.0000000000, ...
         -0.0253192766,  -0.0000000000,  -0.1381459585,  -0.0000000000, ...
          0.2411366542,   0.0000000000,   0.1176198179,   0.0000000000, ...
         -0.0128970875,  -0.0000000000,  -0.0369876356,  -0.0000000000, ...
          0.0028020016,   0.0000000000,   0.0167165626,   0.0000000000, ...
          0.0021457208,   0.0000000000,  -0.0005754968 ]';
```

and

```
Db1 = [   1.0000000000,   0.0000000000,   1.7161808466,   0.0000000000, ...
          0.4579440125,   0.0000000000,  -0.2403589334,  -0.0000000000, ...
          0.2121915366,   0.0000000000,   0.2336557866,   0.0000000000, ...
          0.0209117105,   0.0000000000,  -0.0472097462,  -0.0000000000, ...
         -0.0128412583,  -0.0000000000,   0.0213015188,   0.0000000000, ...
          0.0087646285,   0.0000000000,  -0.0005004266 ]';
```

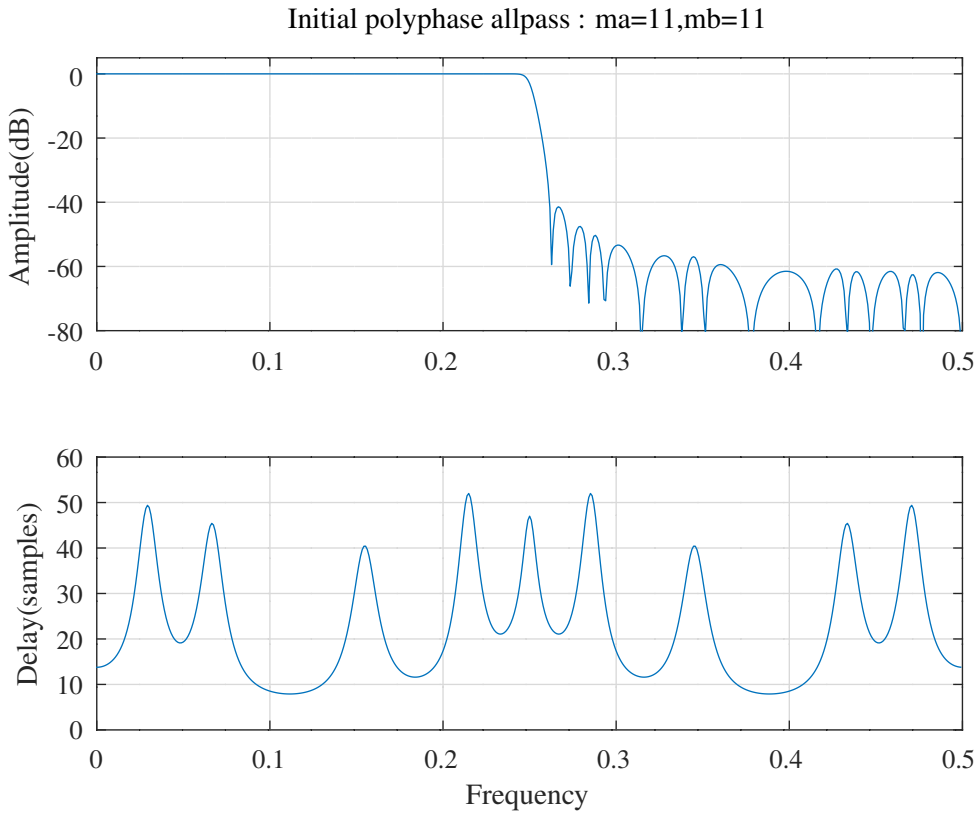



Figure 10.45: Polyphase combination of allpass filters:Initial response found with the WISE barrier function

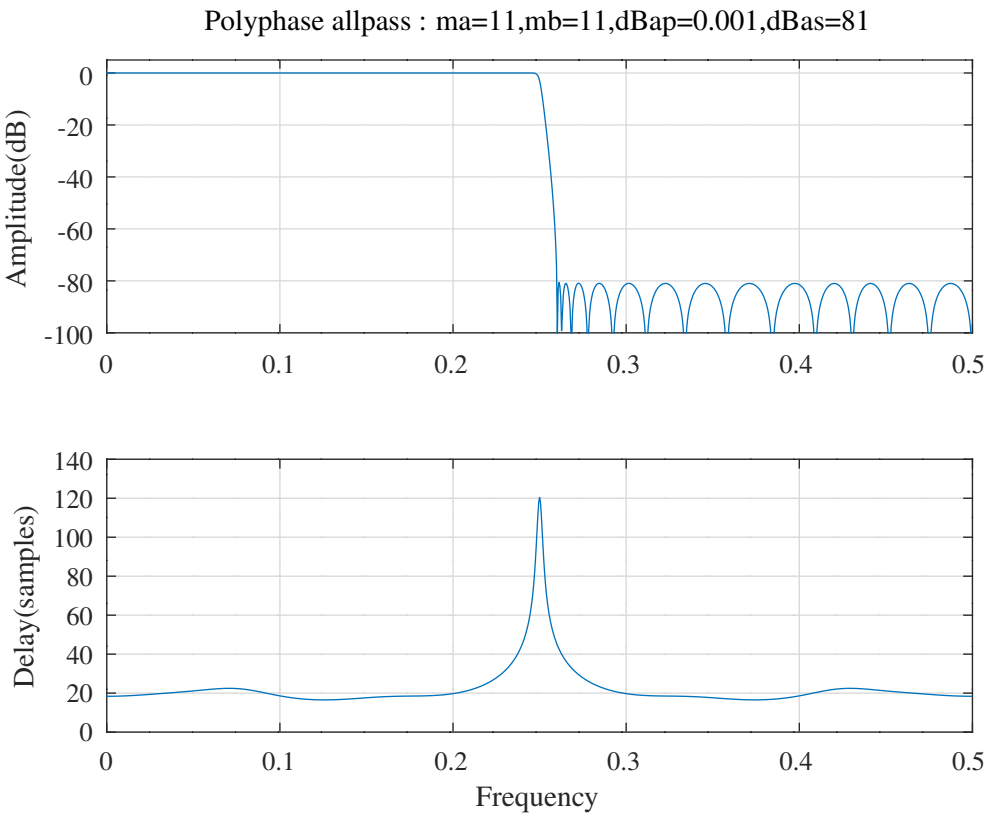



Figure 10.46: Polyphase combination of two allpass filters:Response after PCLS SOCP optimisation

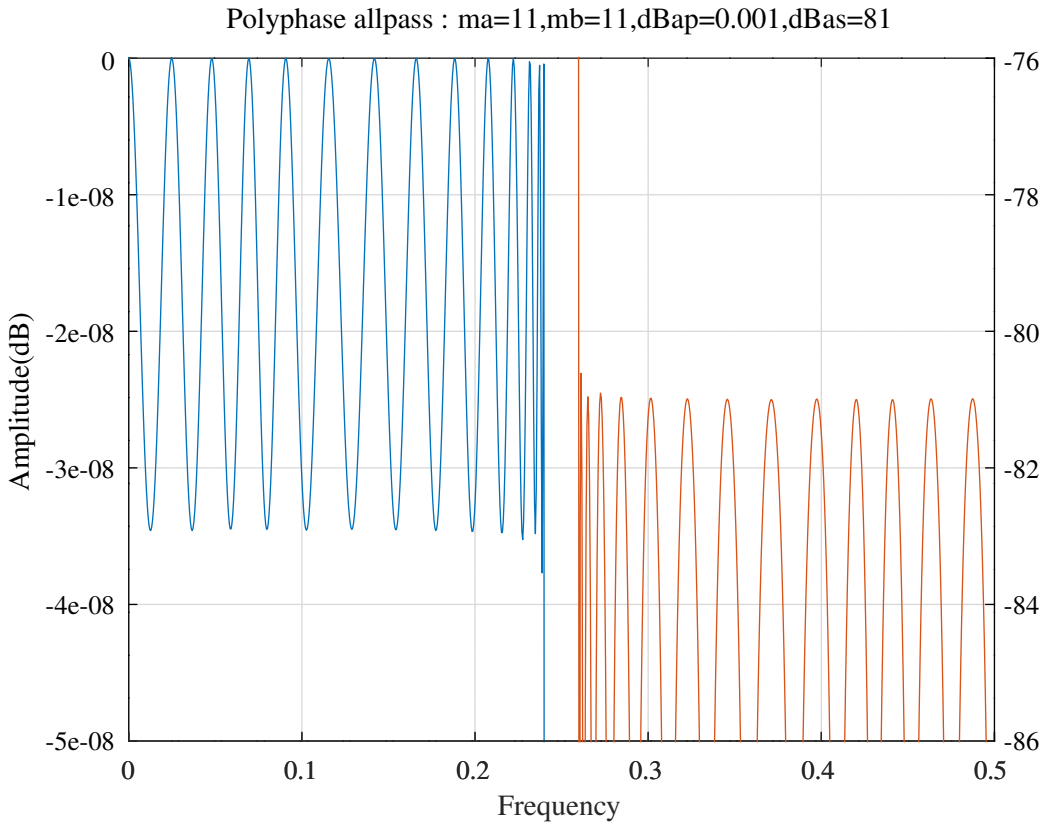



Figure 10.47: Polyphase combination of two allpass filters:Detail of response after PCLS SOCP optimisation

**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 8.1.2. 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=0.000010 % 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 8.1.5. In this case the script has $flat_delay = true$. The response of the initial filter is shown in Figure 10.48. After PCLS optimisation with the *SeDuMi* SOCP solver the response is shown in Figure 10.49. The pass band response is shown in Figure 10.50. The denominator polynomials of the two all-pass filters are

```
Da1 = [   1.0000000000,   0.0000000000,  -0.0123811097,  -0.0000000000, ...
          0.0058250714,   0.0000000000,  -0.0035119236,  -0.0000000000, ...
          0.0023351823,   0.0000000000,  -0.0016730756,  -0.0000000000, ...
          0.0012484635,   0.0000000000,  -0.0009228432,  -0.0000000000, ...
          0.0005589445,   0.0000000000,  -0.0002735016,  -0.0000000000, ...
         -0.0000008515,  -0.0000000000,  -0.0001632399 ]';
```

and

```
Db1 = [   1.0000000000,   0.0000000000,   0.4842649364,   0.0000000000, ...
         -0.1203538026,  -0.0000000000,   0.0572934660,   0.0000000000, ...
         -0.0330156690,  -0.0000000000,   0.0206100732,   0.0000000000, ...
         -0.0133194246,  -0.0000000000,   0.0087149706,   0.0000000000, ...
         -0.0057780813,  -0.0000000000,   0.0037286679,   0.0000000000, ...
         -0.0022918096,  -0.0000000000,   0.0019192365 ]';
```

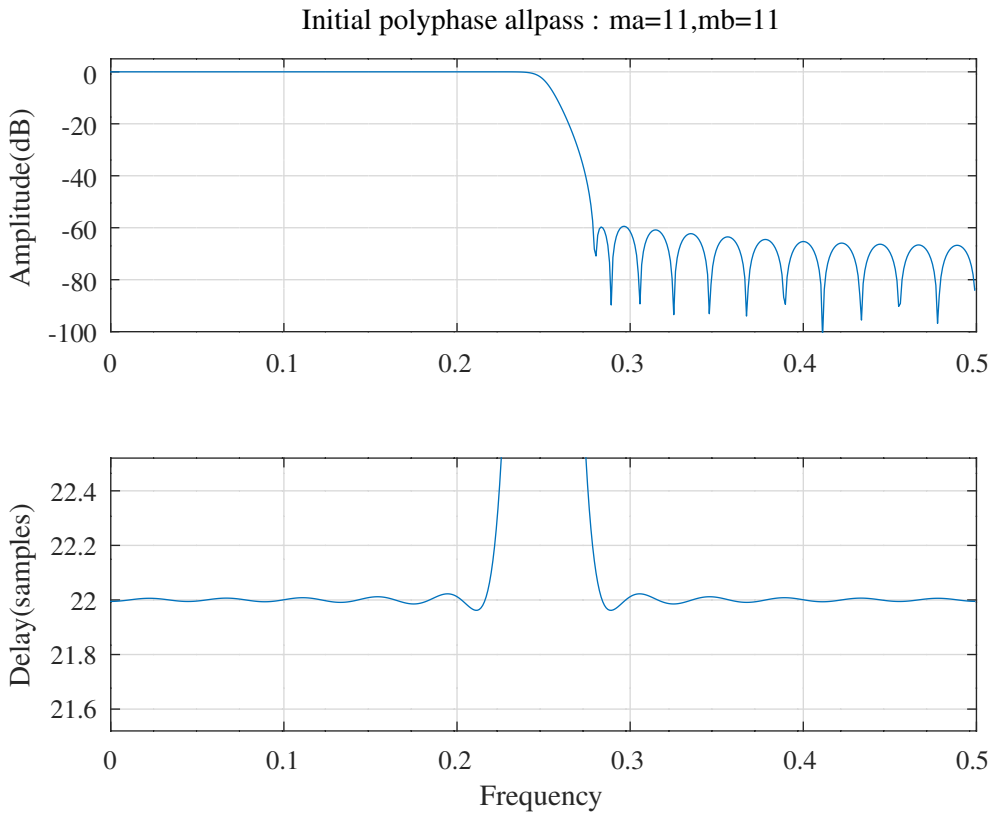



Figure 10.48: Polyphase combination of all-pass filters with flat pass-band delay:Initial response found with the WISE barrier function

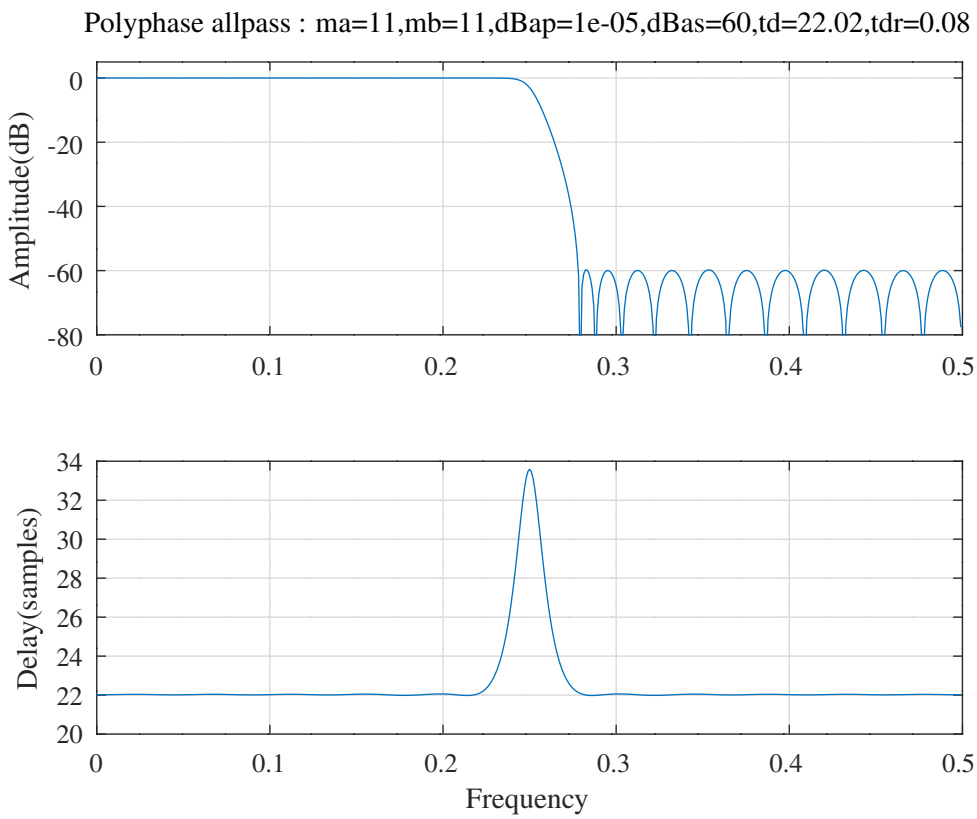



Figure 10.49: Polyphase combination of two all-pass filters with flat pass-band delay:Response after PCLS SOCP optimisation

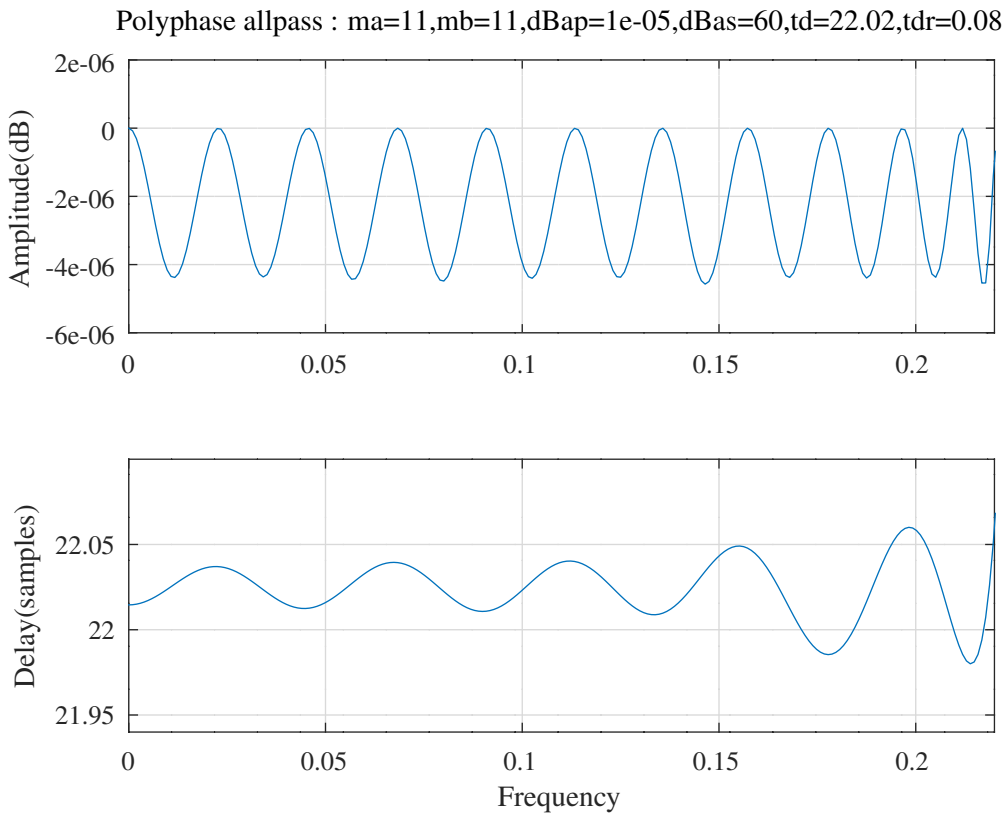



Figure 10.50: Polyphase combination of two all-pass filters with flat pass-band delay:Pass-band response after PCLS SOCP optimisation

## 10.3 Design of an IIR Schur lattice filter

There is a large literature describing adaptive IIR lattice filters. For example, *Regalia* [144] 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: $|k_i| < 1$ (see, for example, *Vaidyanathan* and *Mitra* [150]). This section describes the PCLS design of the squared magnitude, phase and group delay responses of one-multplier 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 5 reviews the Schur decomposition of an IIR transfer function into an IIR tapped-lattice structure. Algorithm 5.4 shows the factorisation of the state variable description of the *one-multiplier* lattice filter. Algorithm 5.6 shows the factorisation of the state variable description of the *normalised-scaled* lattice filter. Section 1.9.4 reviews the sensitivities of the complex transfer function with respect to the state variable coefficients. Appendix J 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 5.4 or Algorithm 5.6 is lower Hessenberg. The matrix resolvent, $(zI - A)^{-1}$ is calculated by direct matrix inversion rather than *Leverrier's* algorithm (see Algorithm 1.6). *Xu Zhong* [221] 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 Chapter 5. 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.

### 10.3.1 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 8.2.3 and 9.4:

```
tol=1e-06 % Tolerance on coefficient update vector
ctol=1e-06 % Tolerance on constraints
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 8.2.3. Figures 10.51 and 10.52 show the overall and passband response after SOCP PCLS optimisation. Figure 10.53 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:

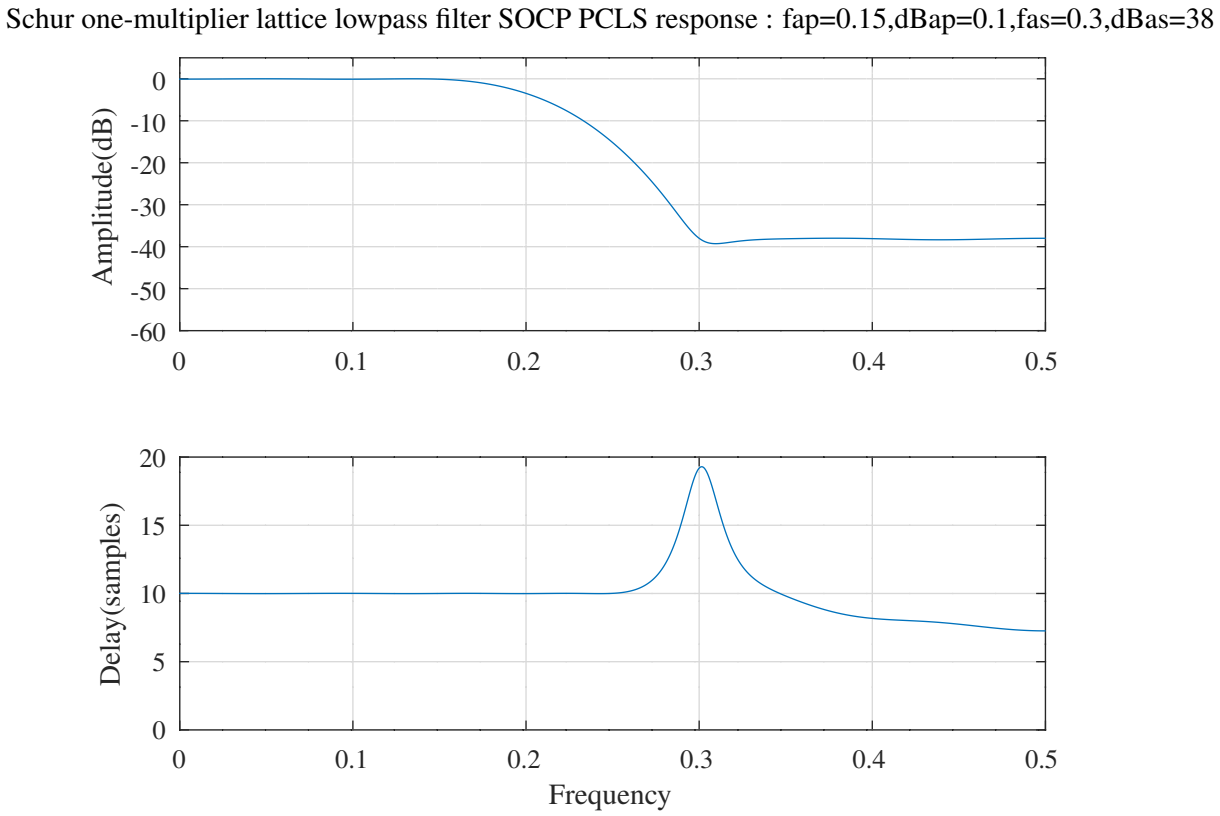



Figure 10.51: Schur one-multiplier lattice lowpass filter : response after PCLS SOCP optimisation

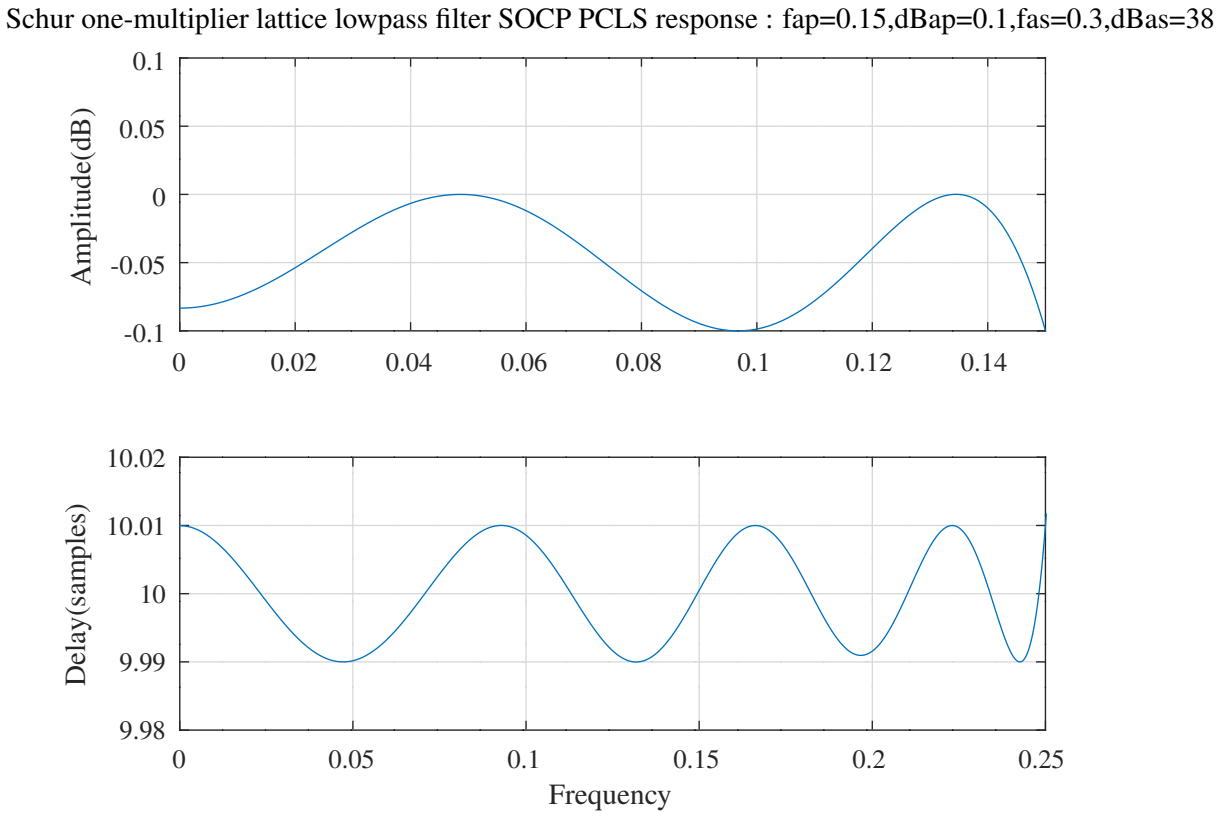



Figure 10.52: Schur one-multiplier lattice lowpass filter : passband response after PCLS SOCP optimisation

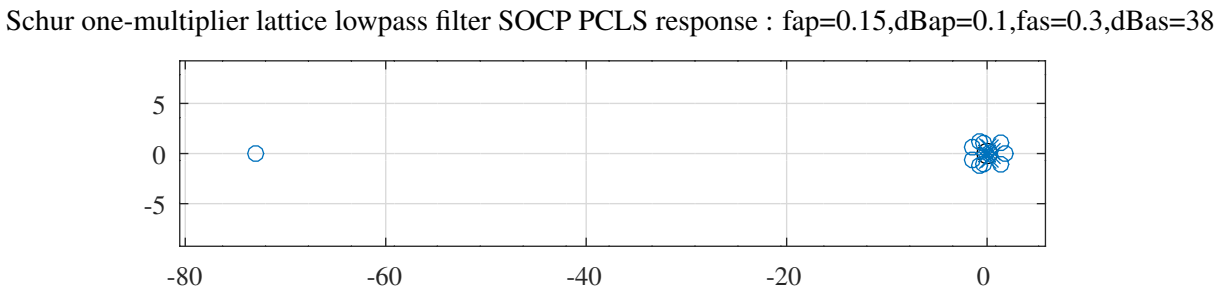



Figure 10.53: Schur one-multiplier lattice lowpass filter:pole-zero plot after PCLS SOCP optimisation

```
k2 = [  -0.6632177318,   0.6513053465,  -0.5324612331,   0.3544152955, ...
        -0.1741362165,   0.0474983102,  -0.0000000000,   0.0000000000, ...
         0.0000000000,  -0.0000000000 ];


epsilon2 = [ -1, -1, -1, -1, ...
              1, -1,  1, -1, ...
             -1,  1 ];


p2 = [   1.0208141982,   0.4593532119,   0.9996266060,   0.5521433830, ...
         0.7997443499,   0.9535779758,   1.0000000000,   1.0000000000, ...
         1.0000000000,   1.0000000000 ];


c2 = [   0.4475005430,   0.8865977587,   0.0987173923,  -0.1374209710, ...
        -0.0510584679,   0.0175849734,   0.0222767287,   0.0003721339, ...
        -0.0078400155,  -0.0034992589,  -0.0000464786 ];
```

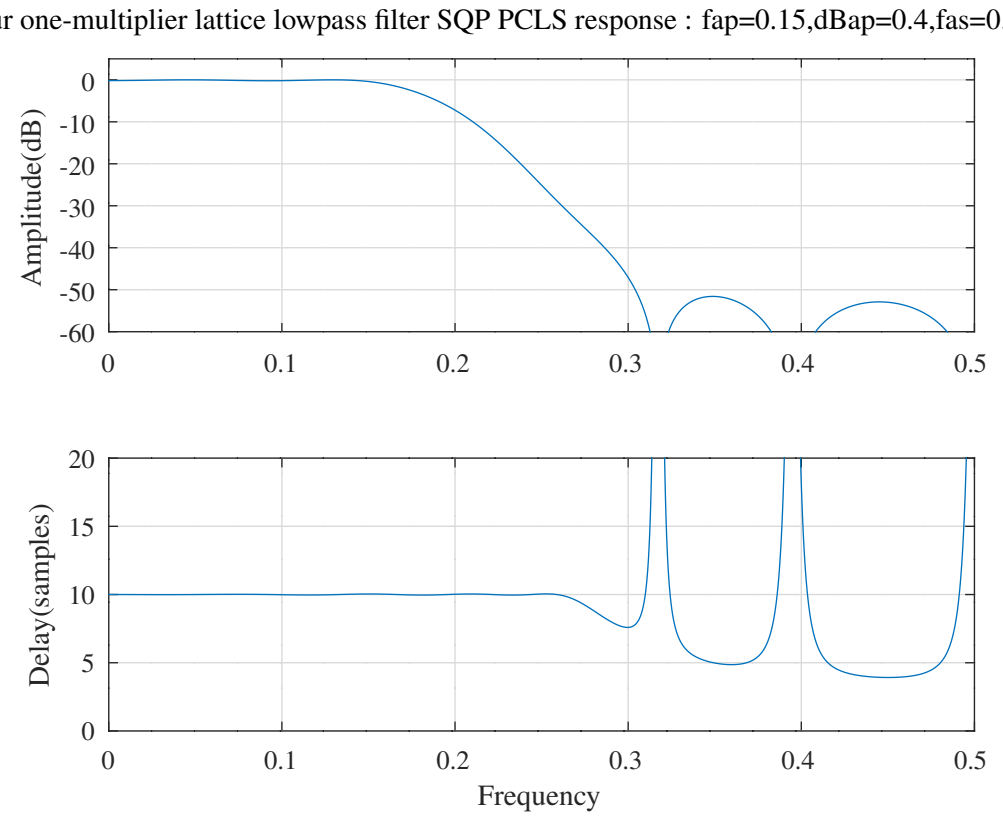



Figure 10.54: Schur one-multiplier lattice lowpass filter : response after SQP PCLS optimisation

### 10.3.2 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
ctol=4e-05 % Tolerance on constraints
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 8.2.3. As for the examples of Chapter 8 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 J.

Figures 10.54 and 10.55 show the overall and passband response of the filter after SQP PCLS optimisation. Figure 10.56 shows the pole-zero plot of the filter after SQP PCLS optimisation.

The SQP PCLS optimised Schur one-multiplier all-pass lattice and numerator tap coefficients of the low-pass filter are:

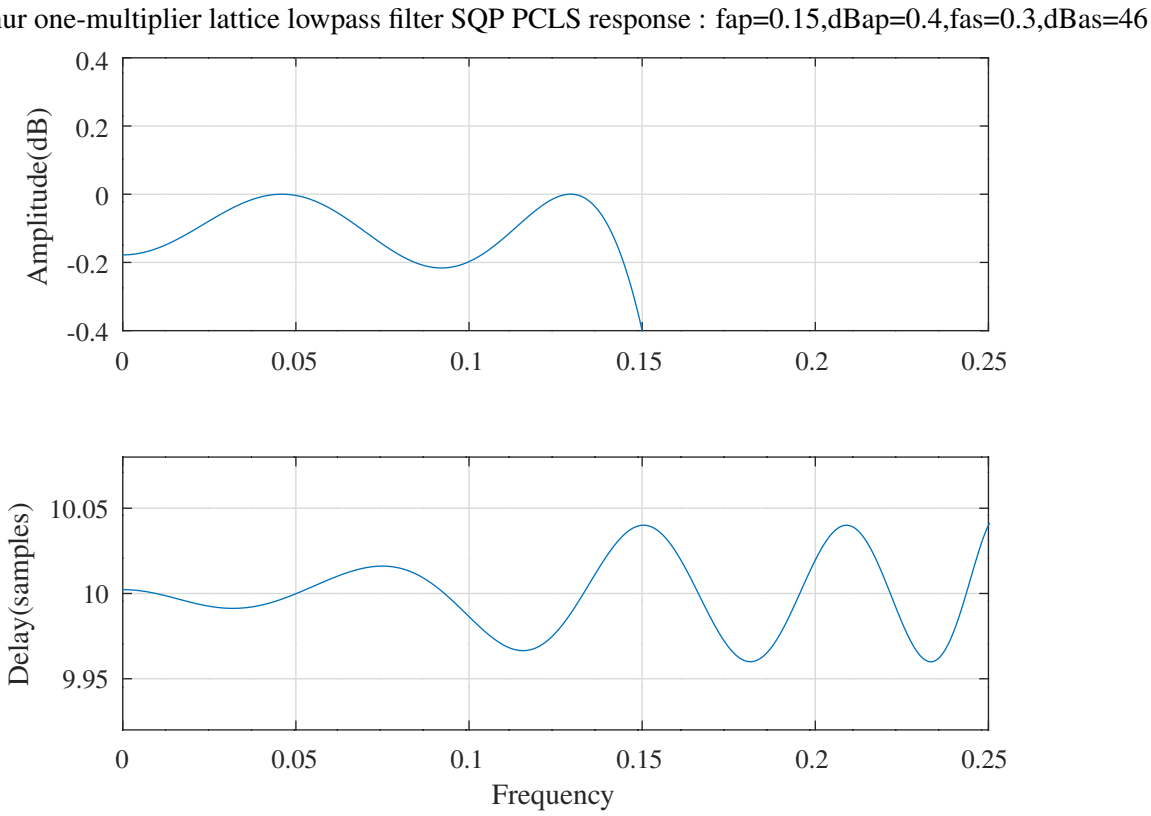



Figure 10.55: Schur one-multiplier lattice lowpass filter : passband response after SQP PCLS optimisation

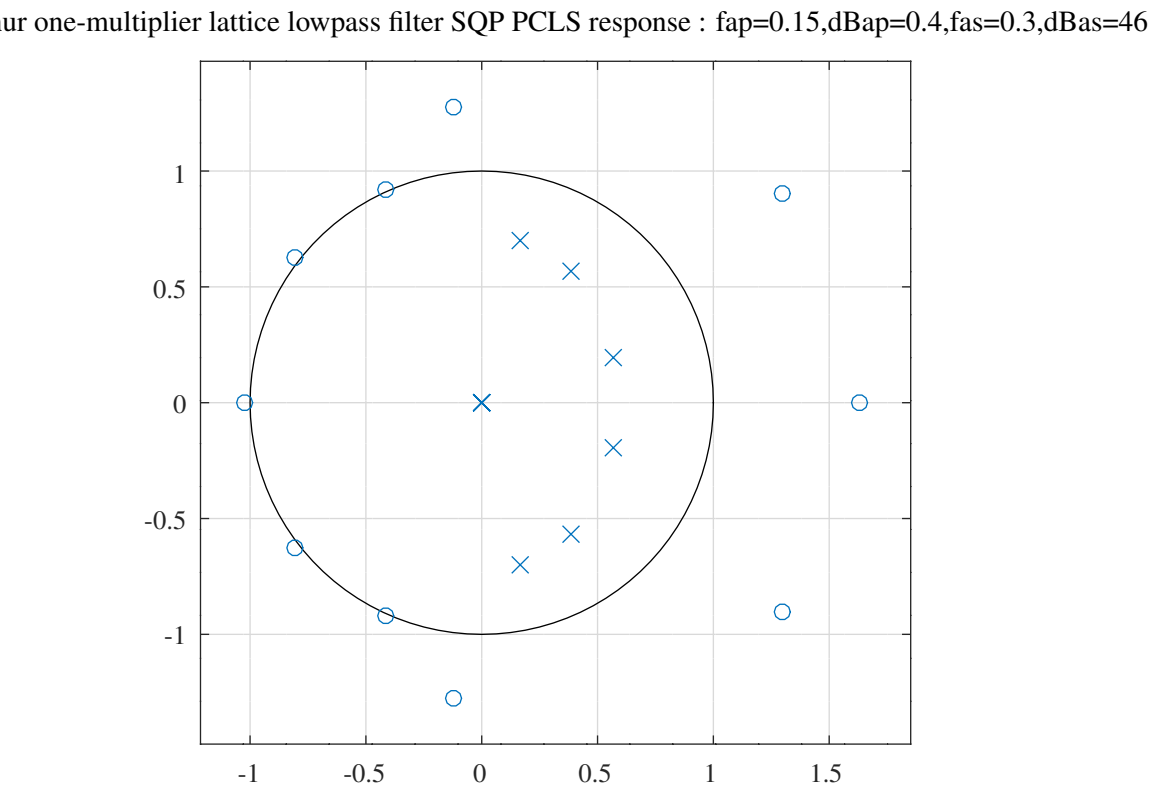



Figure 10.56: Schur one-multiplier lattice lowpass filter:pole-zero plot after SQP PCLS optimisation

```
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 ];
```

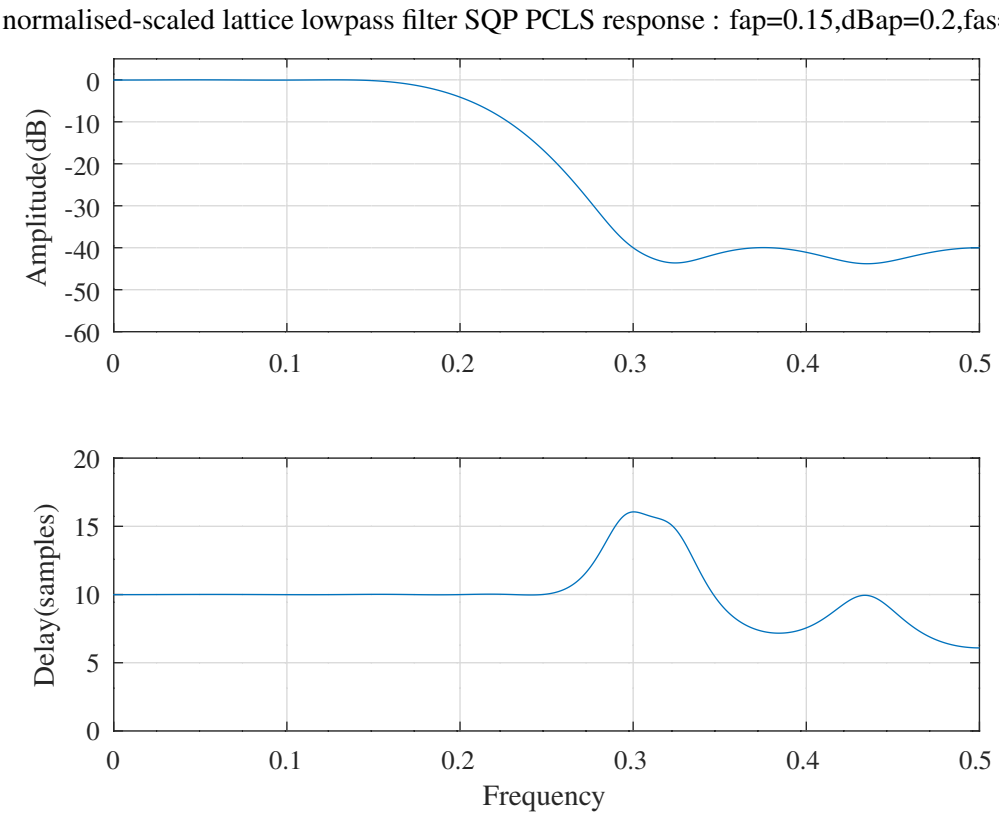



Figure 10.57: Schur normalised-scaled lattice lowpass filter : response after SQP PCLS optimisation

### 10.3.3 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=0.0001 % Tolerance on coefficient update vector for MMSE
ctol=1e-06 % Tolerance on constraints
n=800 % 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=0.999900 % 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=10 % Nominal pass band filter group delay
tpr=0.08 % Delay pass band peak-to-peak ripple
Wtp=0.1 % Delay pass band weight
fas=0.3 % Amplitude stop band edge
dBas=40 % amplitude stop band peak-to-peak ripple
Was=1000 % Amplitude stop band weight
```

The initial filter is the "IPZS-1" of Section 8.2.3. As for the examples of Chapter 8 the SQP BFGS update is initialised by the diagonal of the Hessian matrix of the squared error.

Figures 10.57 and 10.58 show the overall and passband response of the filter after SQP PCLS optimisation. Figure 10.59 shows the pole-zero plot of the filter 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 = [   1.0279396548,   0.1935260157,  -0.0916241858,  -0.0653808722, ...
            0.0142448015,   0.0129142652,   0.0024737017,  -0.0069440305, ...
           -0.0051571508,   0.0012782823 ];
```

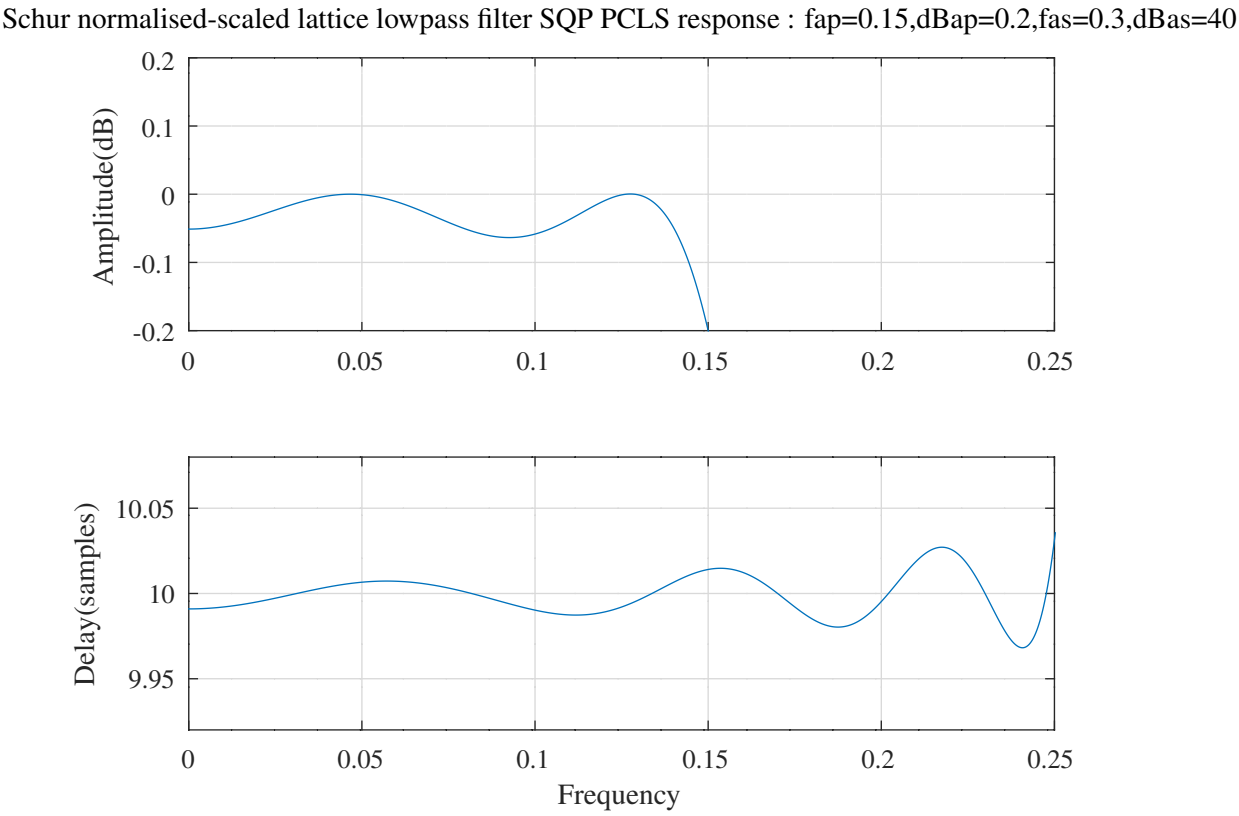



Figure 10.58: Schur normalised-scaled lattice lowpass filter : passband response after SQP PCLS optimisation

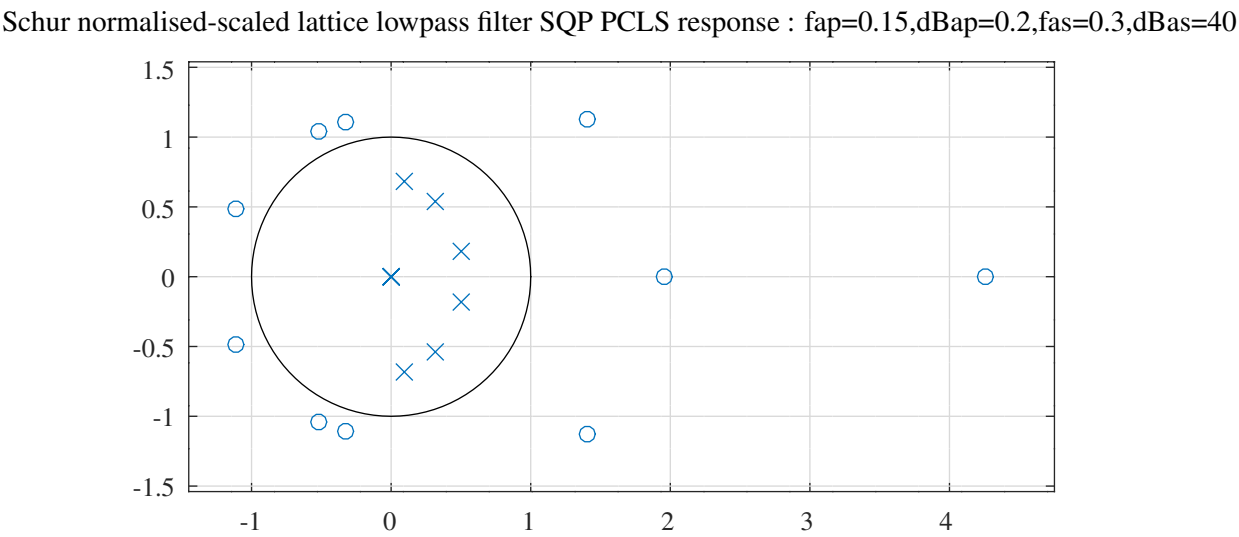



Figure 10.59: Schur normalised-scaled lattice lowpass filter:pole-zero plot after SQP PCLS optimisation

```
s11_2 = [   0.8040534037,   0.9316660442,   0.9941967593,   1.0261479817, ...
            0.9249817439,   1.1144246585,   1.3100056198,   1.3019852097, ...
            1.3079771599,   0.8138627835 ];


s20_2 = [  -0.6889239260,   0.4752667630,  -0.4961994741,   0.3970745806, ...
           -0.1697955137,   0.0441000000,   0.0000000000,   0.0000000000, ...
            0.0000000000,   0.0000000000 ];


s00_2 = [   0.9883987249,   0.4926734214,   0.7408301356,   0.8854322120, ...
            0.9273987541,   0.4059507633,   1.0000000000,   1.0000000000, ...
            1.0000000000,   1.0000000000 ];


s02_2 = [   0.6857383075,  -0.5440075919,   0.6572742436,  -0.5692084825, ...
            0.2760979095,  -0.3613674206,  -0.0000000000,  -0.0000000000, ...
           -0.0000000000,  -0.0000000000 ];


s22_2 = [   0.7436318730,   0.7278805905,   0.6313750134,   0.9450728519, ...
            0.2005128923,   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) = [     0.0889,     0.0488,     0.1119,     0.1415, ...
                0.1496,     0.1658,     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 $s_{22} = s_{00}$. In neither case does this function enforce normalised-scaling with the relations $s_{02} = \sqrt{1 - s_{00}^2}$ and $s_{22} = \sqrt{1 - s_{20}^2}$. For simplicity, I have assumed that all relationships between coefficients are linear.

### 10.3.4 Design of an IIR low-pass filter with parallel Schur one-multiplier all-pass lattice filters using SOCP

The Octave script *schurOneMPAlattice_socp_slb_lowpass_test.m* implements the design of a low-pass IIR filter composed of the sum of two parallel one-multiplier Schur lattice all-pass filters with SOCP and PCLS optimisation. The specification of the filter is:

```
tol=1e-07 % Tolerance on coefficient update vector
ctol=1e-07 % Tolerance on constraints
n=400 % Frequency points across the band
m1=11 % Allpass model filter 1 denominator order
m2=12 % Allpass model filter 2 denominator order
rho=0.992188 % Constraint on allpass coefficients
fap=0.125 % Amplitude pass band edge
dBap=0.1 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=0 % Amplitude transition band weight
fas=0.25 % Amplitude stop band edge
dBas=60 % amplitude stop band peak-to-peak ripple
Was=100 % Amplitude stop band weight
ftp=0.175 % Delay pass band edge
td=11.5 % Nominal pass band filter group delay
tdr=0.08 % Delay pass band peak-to-peak ripple
Wtp=2 % Delay pass band weight
```

The initial parallel all-pass filters were designed by the Octave script *tarczynski_parallel_allpass_test.m* (with $flat_delay = true$). These initial filters are those used in Section 10.2.3.

Figures 10.60 and 10.61 show the overall and passband responses of the filter after SOCP PCLS optimisation and Figure 10.62 shows the pole-zero plot of the filter. Figure 10.63 shows the coefficient sensitivity responses of the filter.

The low-pass filter SOCP PCLS optimised Schur one-multiplier all-pass lattice coefficients are:

```
A1k = [   0.7709956651,  -0.0881301024,  -0.2677269107,  -0.0637813383, ...
         -0.0591232245,   0.2444717094,  -0.1442401630,  -0.0042745945, ...
          0.1646508935,  -0.1595515041,   0.0536905119 ];


A1epsilon = [  1,  1,  1,  1, ...
               1, -1,  1,  1, ...
               1,  1, -1 ];


A2k = [   0.3877973627,  -0.2735916308,   0.1868050615,   0.1639492949, ...
         -0.0462216184,   0.0417084103,  -0.2010053855,   0.1801845652, ...
          0.0055412612,  -0.1785124736,   0.1505008762,  -0.0547047632 ];


A2epsilon = [  1,  1,  1, -1, ...
               1, -1, -1, -1, ...
              -1, -1, -1,  1 ];
```

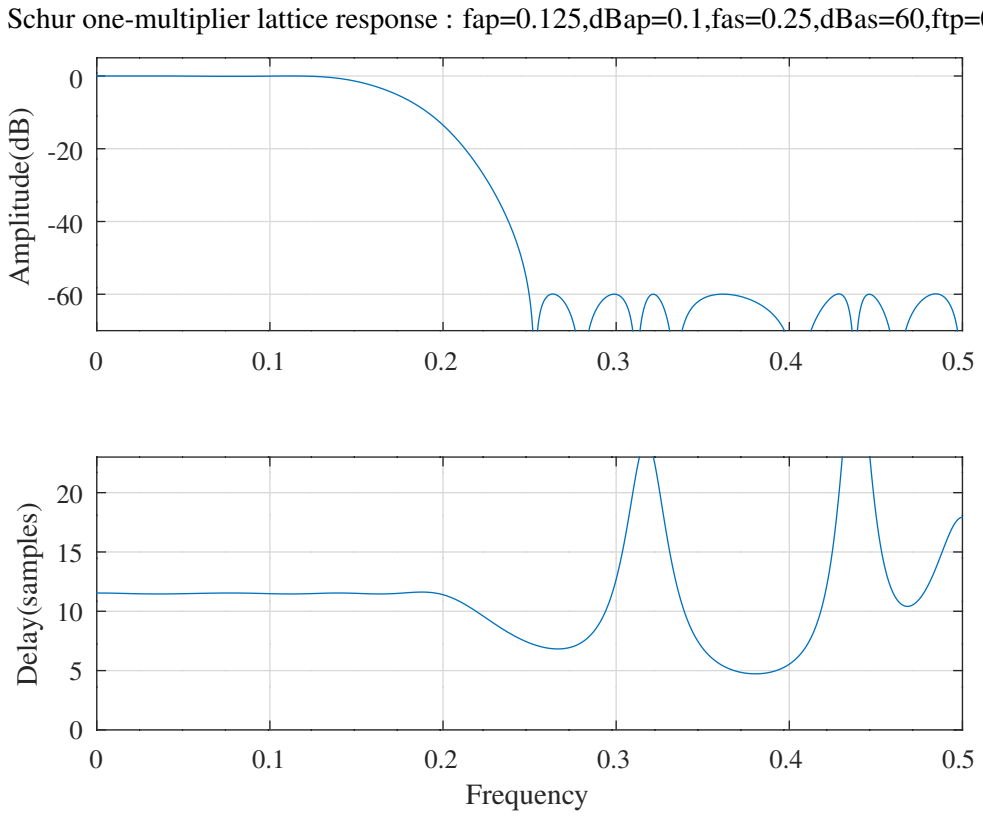



Figure 10.60: Parallel Schur one-multiplier all-pass lattice low-pass filter : response after SOCP PCLS optimisation

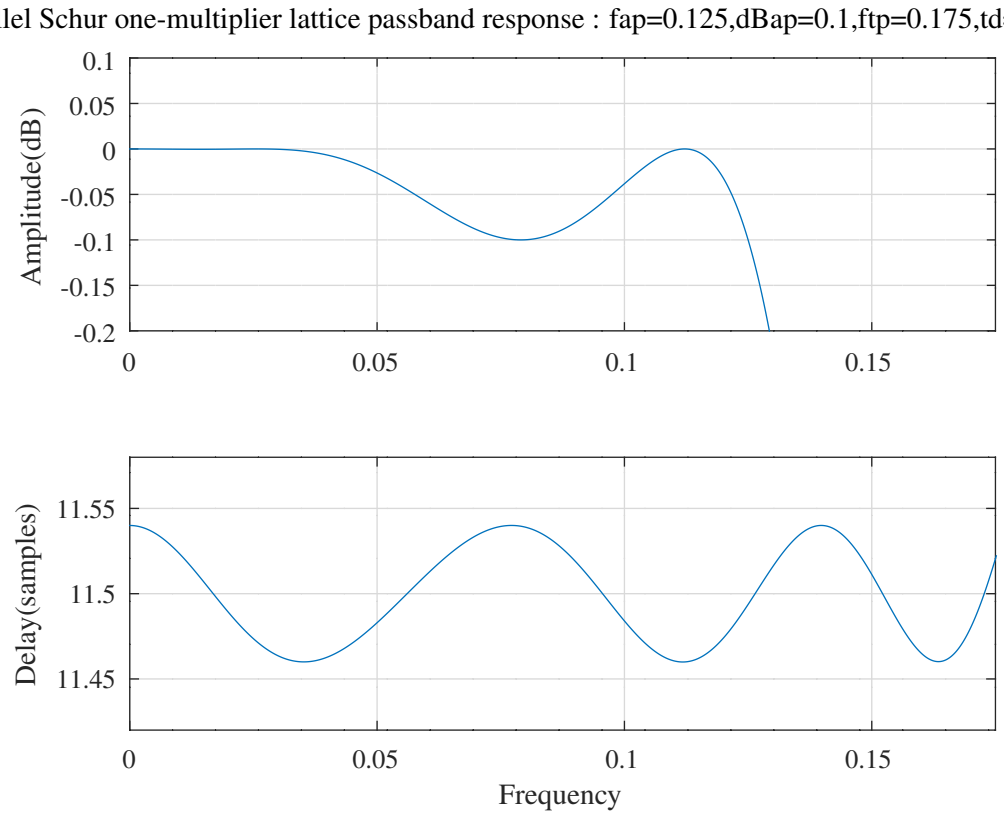



Figure 10.61: Parallel Schur one-multiplier all-pass lattice low-pass filter : passband response after SOCP PCLS optimisation

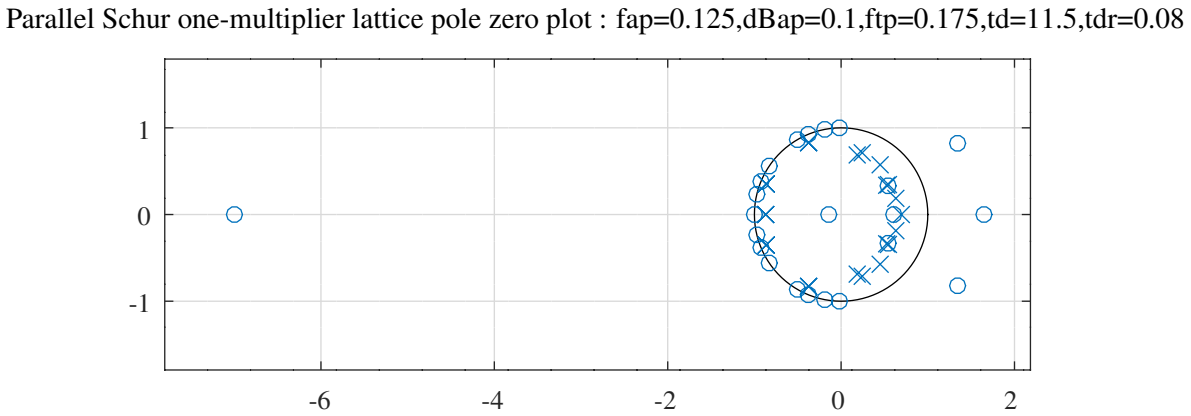



Figure 10.62: Parallel Schur one-multiplier all-pass lattice low-pass filter : pole-zero plot after SOCP PCLS optimisation

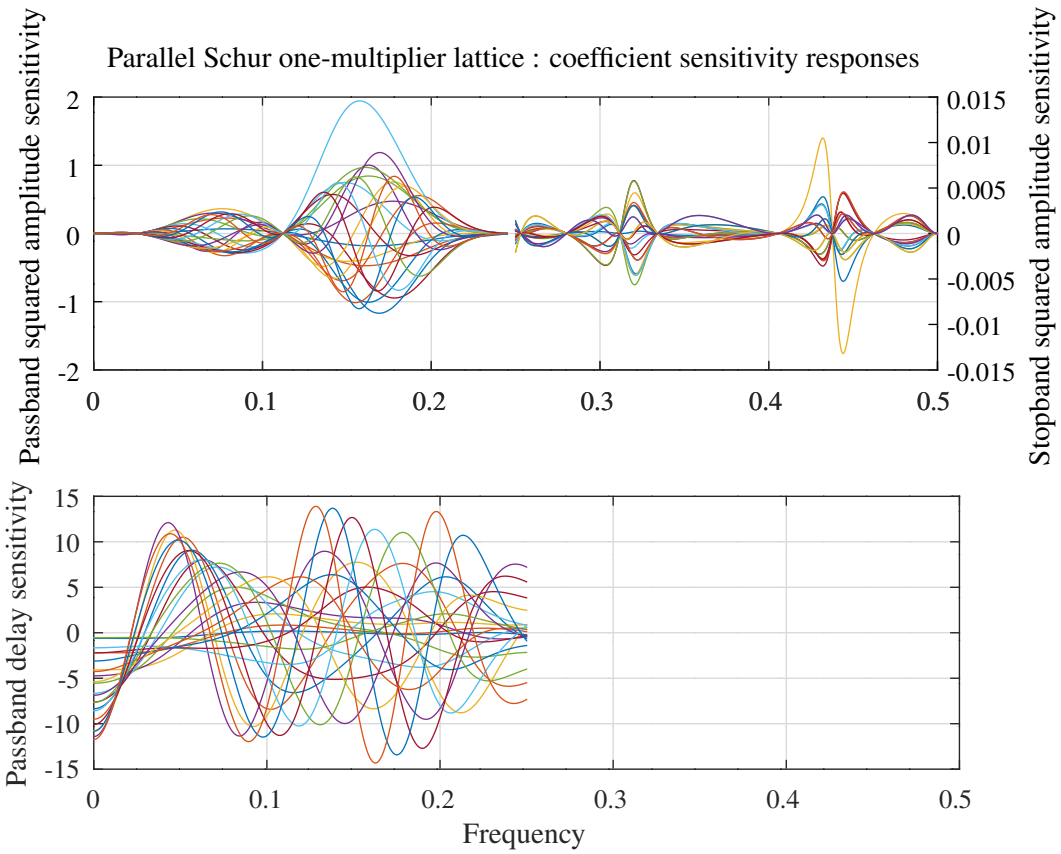



Figure 10.63: Parallel Schur one-multiplier all-pass lattice low-pass filter : coefficient sensitivity responses after SOCP PCLS optimisation

### 10.3.5 Design of a parallel IIR Schur one-multiplier all-pass lattice band-pass filter using SOCP

The Octave script *schurOneMPAlattice_socp_slb_bandpass_test.m* implements the design of a band-pass IIR filter composed of the difference of two parallel one-multiplier Schur lattice all-pass filters with SOCP and PCLS optimisation. The specification of the filter is:

```
m1=10 % Allpass model filter 1 denominator order
m2=10 % Allpass model filter 2 denominator order
difference=1 % Use difference of all-pass filters
tol=0.0001 % Tolerance on coefficient update vector
ctol=5e-08 % Tolerance on constraints
rho=0.992188 % Constraint on allpass pole radius
n=1000 % Frequency points across the band
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
dBap=2.000000 % Pass band amplitude response ripple(dB)
Wap=1 % Pass band amplitude response weight
Watl=0.1 % Lower transition band amplitude response weight
Watu=0.1 % Upper transition band amplitude response weight
fasl=0.05 % Stop band amplitude response lower edge
fasu=0.25 % Stop band amplitude response upper edge
dBas=53.000000 % Stop band amplitude response ripple(dB)
Wasl=10000 % Lower stop band amplitude response weight
Wasu=10000 % Upper stop band amplitude response weight
ftpl=0.09 % Pass band group-delay response lower edge
ftpu=0.21 % Pass band group-delay response upper edge
td=16.000000 % Pass band nominal group-delay response(samples)
tdr=0.080000 % Pass band group-delay response ripple(samples)
Wtp=0.1 % Pass band group-delay response weight
```

The initial parallel all-pass filters were designed by the Octave script *tarczynski_parallel_allpass_bandpass_test.m*. These initial filters are those used in Section 10.2.3. Figures 10.64 and 10.65 show the overall and passband responses of the filter after SOCP PCLS optimisation. The band-pass filter SOCP PCLS optimised Schur one-multiplier all-pass lattice coefficients are:

```
A1k = [  -0.4413268217,   0.8242662607,   0.0643746790,  -0.3295019297, ...
          0.5443935951,  -0.1871394049,  -0.1895702406,   0.4406169247, ...
         -0.2842081663,   0.1653497952 ];


A1epsilon = [  1,  1, -1,  1, ...
              -1, -1,  1,  1, ...
               1, -1 ];


A2k = [  -0.7665280708,   0.8657045338,   0.0326024218,  -0.3757652566, ...
          0.5644013261,  -0.1013601046,  -0.1435194111,   0.4316424464, ...
         -0.3119686474,   0.1522026272 ];


A2epsilon = [  1,  1, -1,  1, ...
              -1,  1,  1,  1, ...
               1,  1 ];
```

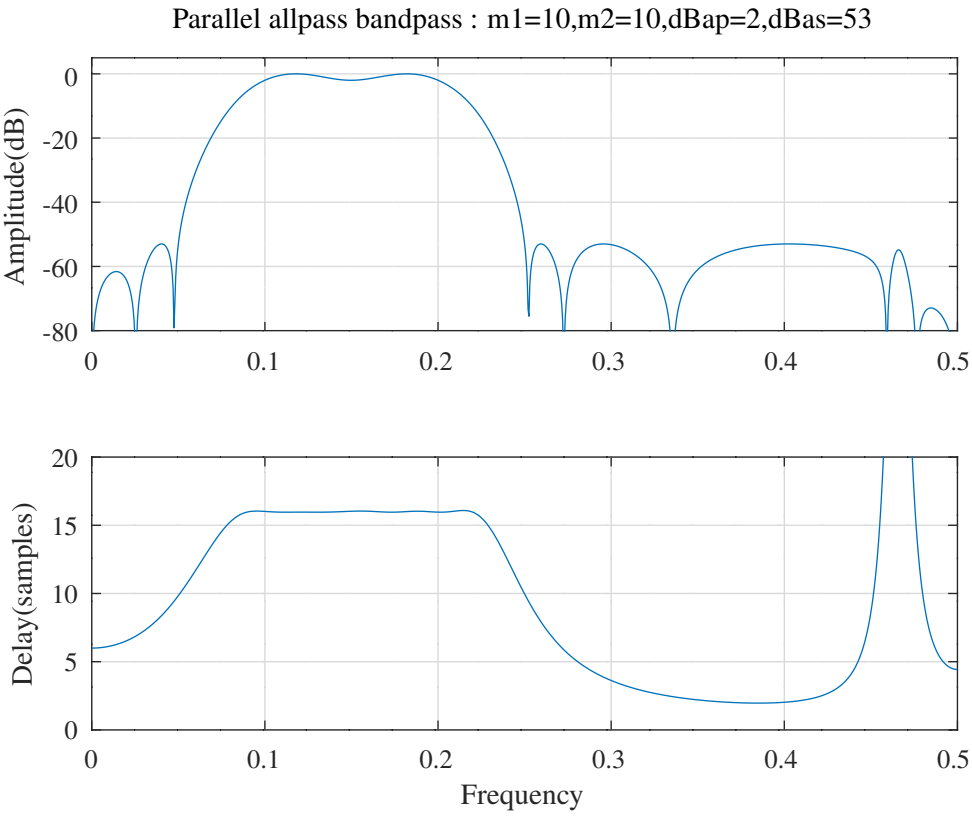



Figure 10.64: Parallel Schur one-multiplier all-pass lattice band-pass filter : response after SOCP PCLS optimisation

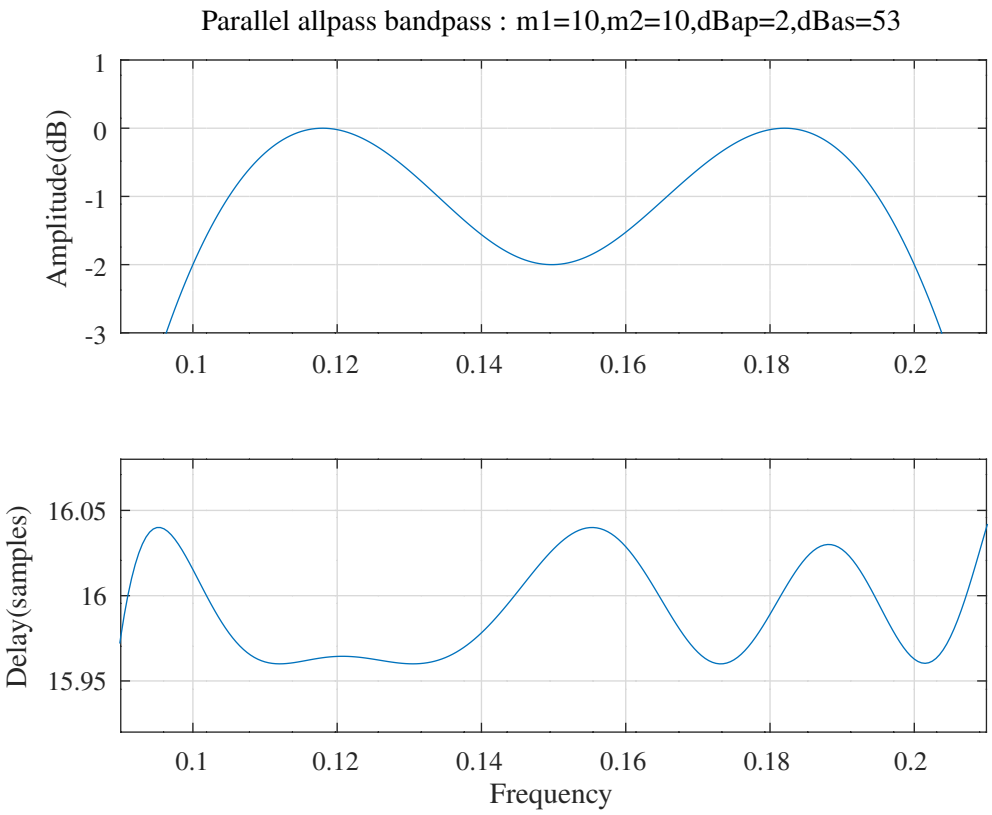



Figure 10.65: Parallel Schur one-multiplier all-pass lattice band-pass filter : passband response after SOCP PCLS optimisation

### 10.3.6 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=1e-05 % Tolerance on coef. update for PCLS
ctol=1e-05 % Tolerance on constraints
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
ftpu=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=1 % Delay pass band weight(MMSE)
Wtp_pcls=1 % 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=100000 % Ampl. lower stop band weight(MMSE)
Wasu_mmse=400000 % Ampl. upper stop band weight(MMSE)
Wasl_pcls=100000 % Ampl. lower stop band weight(PCLS)
Wasu_pcls=400000 % 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 8.2.6. The denominator polynomial of the filter has coefficients in $z^2$ only.

Figure 10.66 shows the overall response of the band-pass filter after SQP MMSE optimisation. Figures 10.67 and 10.68 show the overall and passband response of the band-pass filter after SQP PCLS optimisation. Figure 10.69 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,   0.6672955640,   0.0000000000,   0.4964341949, ...
         0.0000000000,   0.3462544804,   0.0000000000,   0.4174442880, ...
         0.0000000000,   0.2972266682,   0.0000000000,   0.2512374722, ...
         0.0000000000,   0.1512063085,   0.0000000000,   0.1021208736, ...
         0.0000000000,   0.0362871687,   0.0000000000,   0.0150432836 ];


epsilon2 = [  0,  1,  0, -1, ...
              0,  1,  0, -1, ...
              0,  1,  0, -1, ...
              0, -1,  0,  1, ...
              0, -1,  0, -1 ];


p2 = [   1.1347075754,   1.1347075754,   0.5068820974,   0.5068820974, ...
         0.8737911640,   0.8737911640,   0.6089034442,   0.6089034442, ...
         0.9498011634,   0.9498011634,   0.6990888887,   0.6990888887, ...
         0.9037123550,   0.9037123550,   1.0524603142,   1.0524603142, ...
         0.9499484805,   0.9499484805,   0.9850681834,   0.9850681834 ];


c2 = [   0.0704221417,  -0.0128119538,  -0.2992871552,  -0.4821902862, ...
        -0.1624798794,   0.1224163794,   0.3957922392,   0.3003821423, ...
         0.0171930791,  -0.0825256371,  -0.0795022560,  -0.0125415926, ...
        -0.0099075145,  -0.0352745191,  -0.0255813129,   0.0048306421, ...
         0.0246321541,   0.0165314320,   0.0027523861,   0.0012900332, ...
         0.0058051915 ];
```

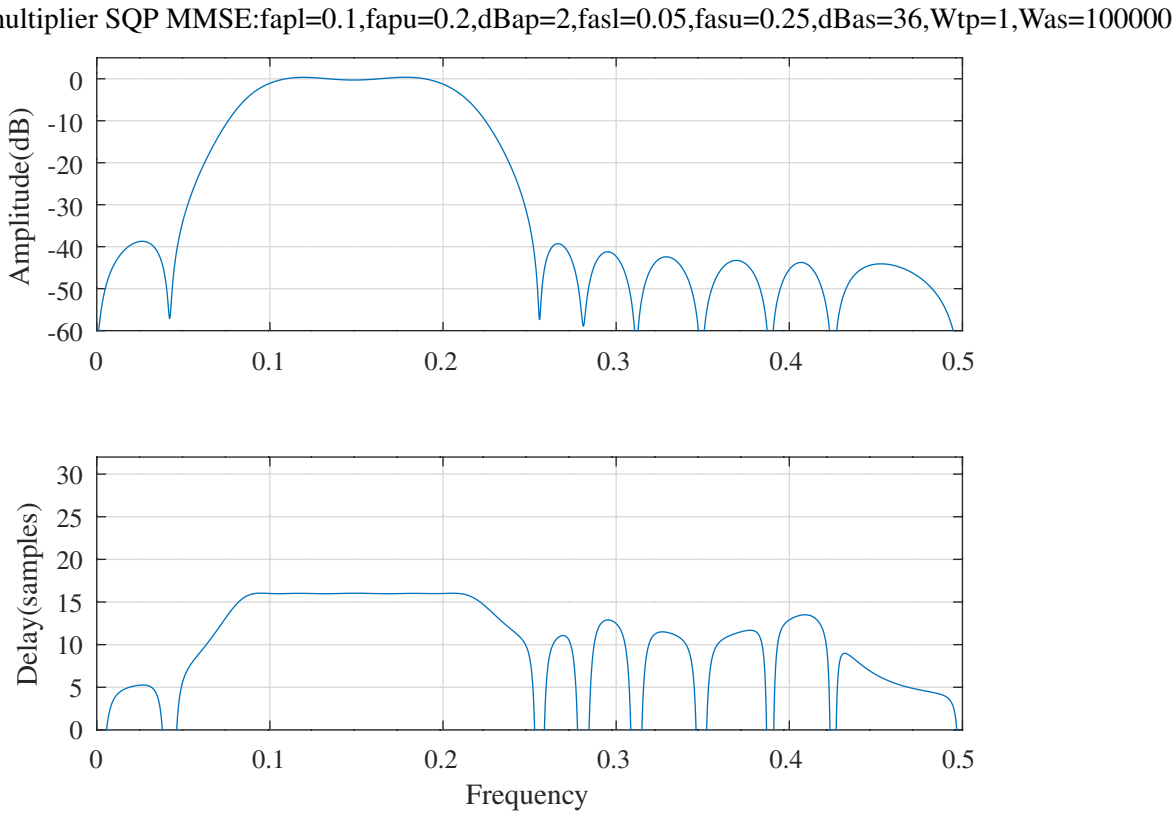



Figure 10.66: Schur one-multiplier lattice band-pass filter : response after SQP MMSE optimisation

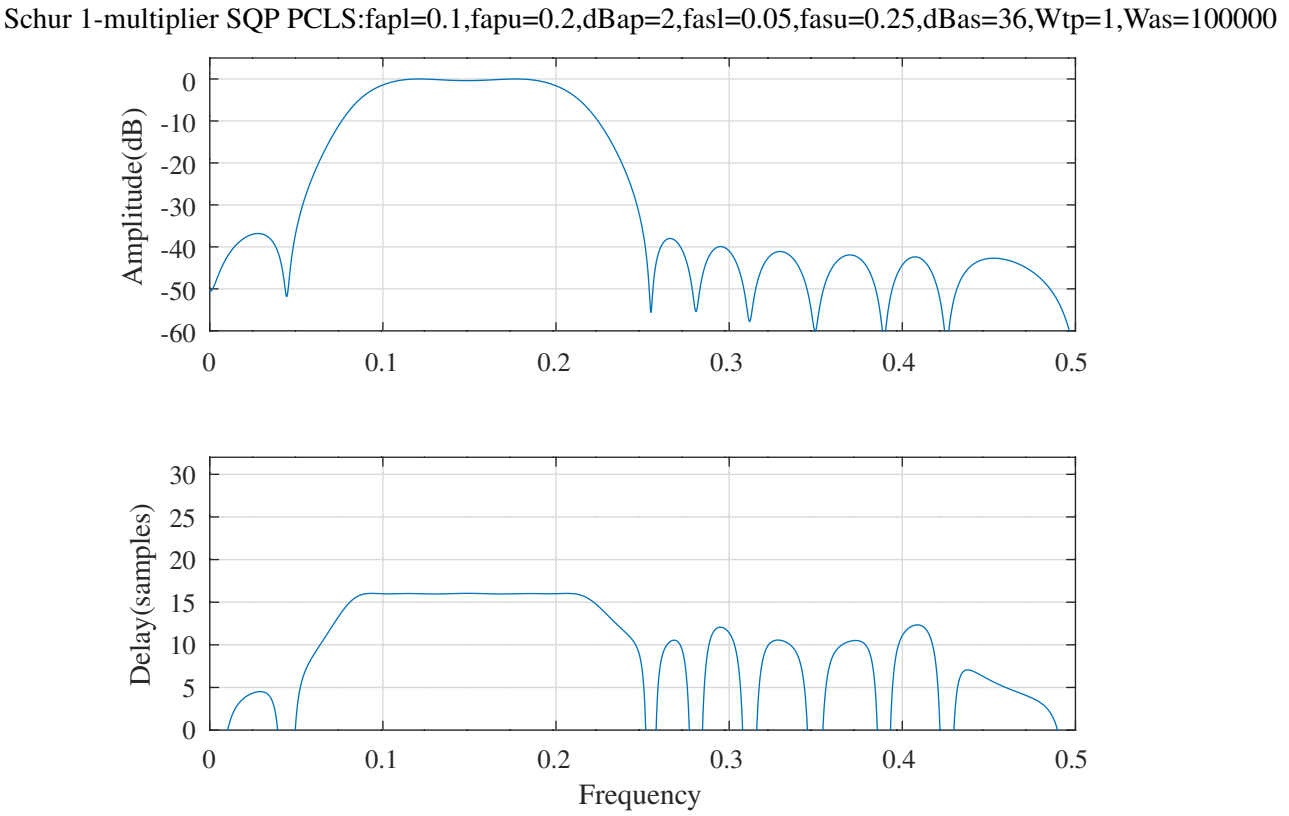



Figure 10.67: Schur one-multiplier lattice band-pass filter : response after SQP PCLS optimisation

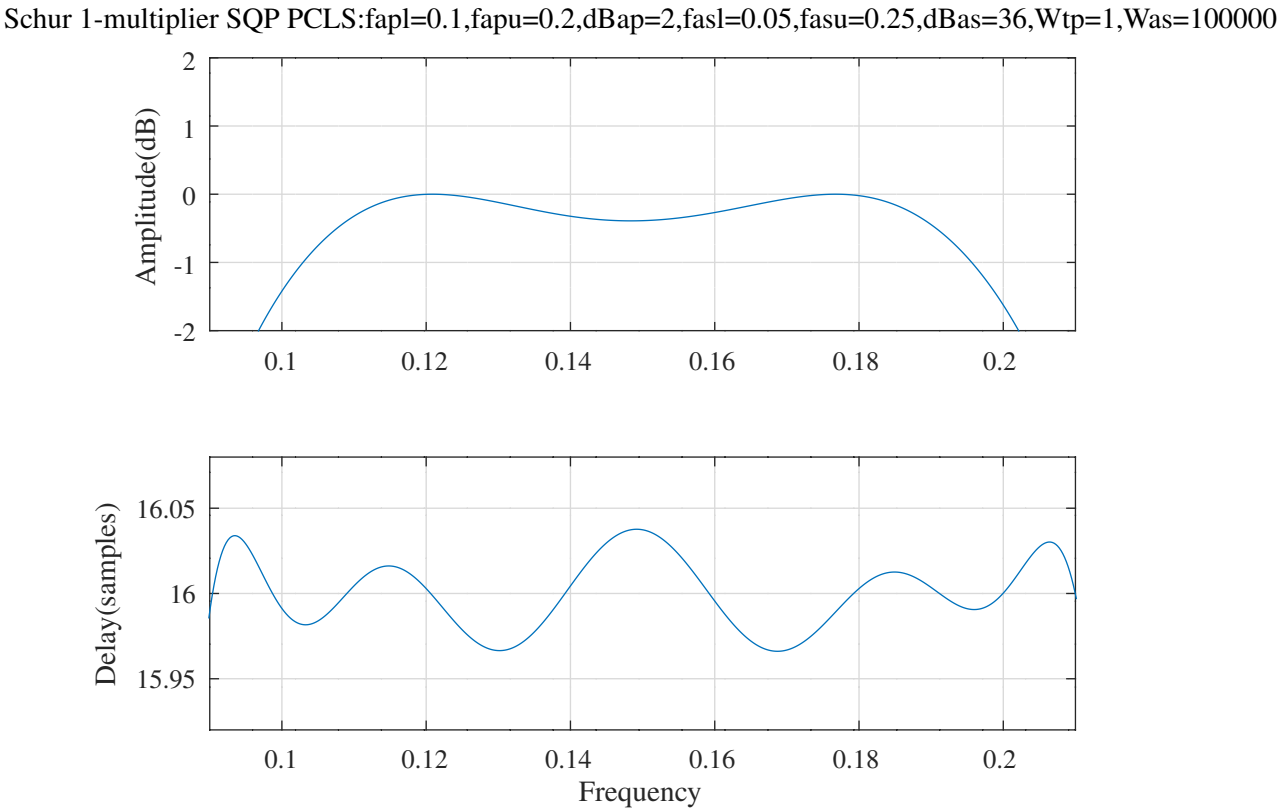



Figure 10.68: Schur one-multiplier lattice band-pass filter : passband response after SQP PCLS optimisation

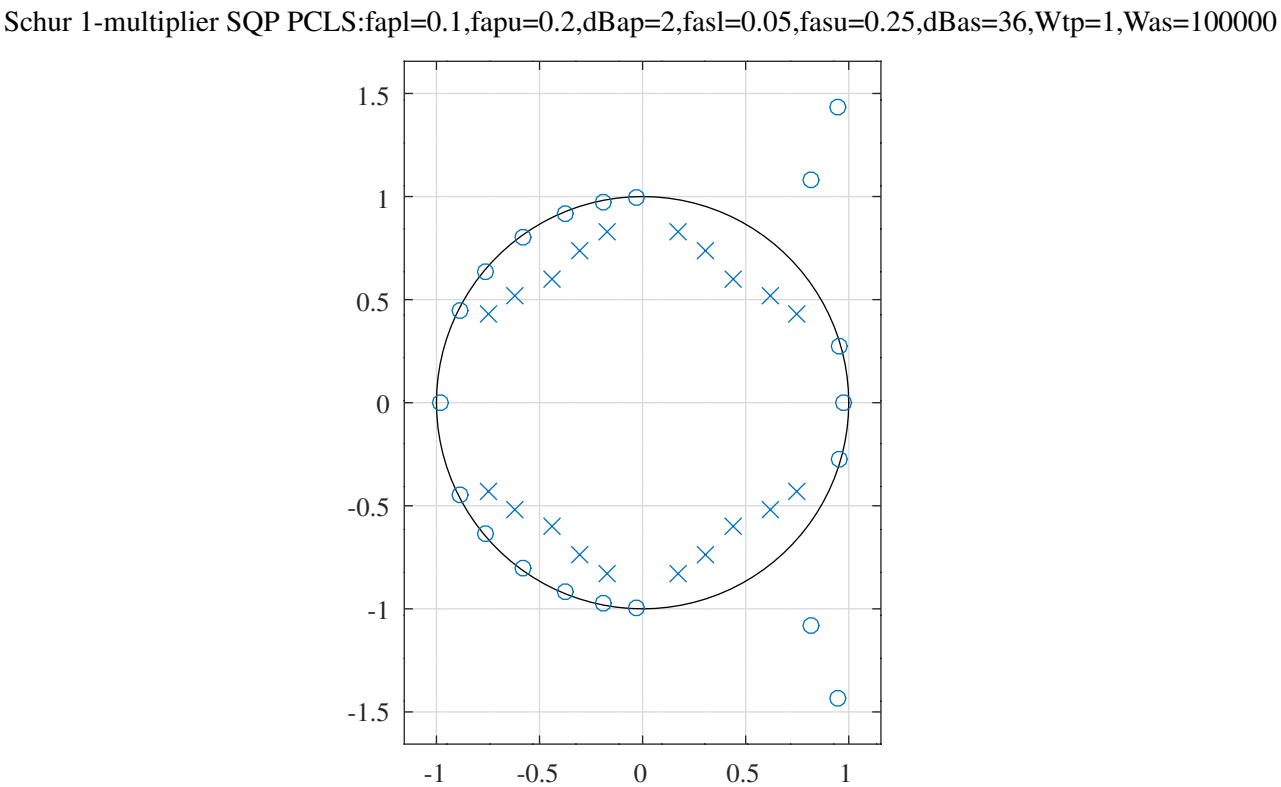



Figure 10.69: Schur one-multiplier lattice band-pass filter:pole-zero plot after SQP PCLS optimisation

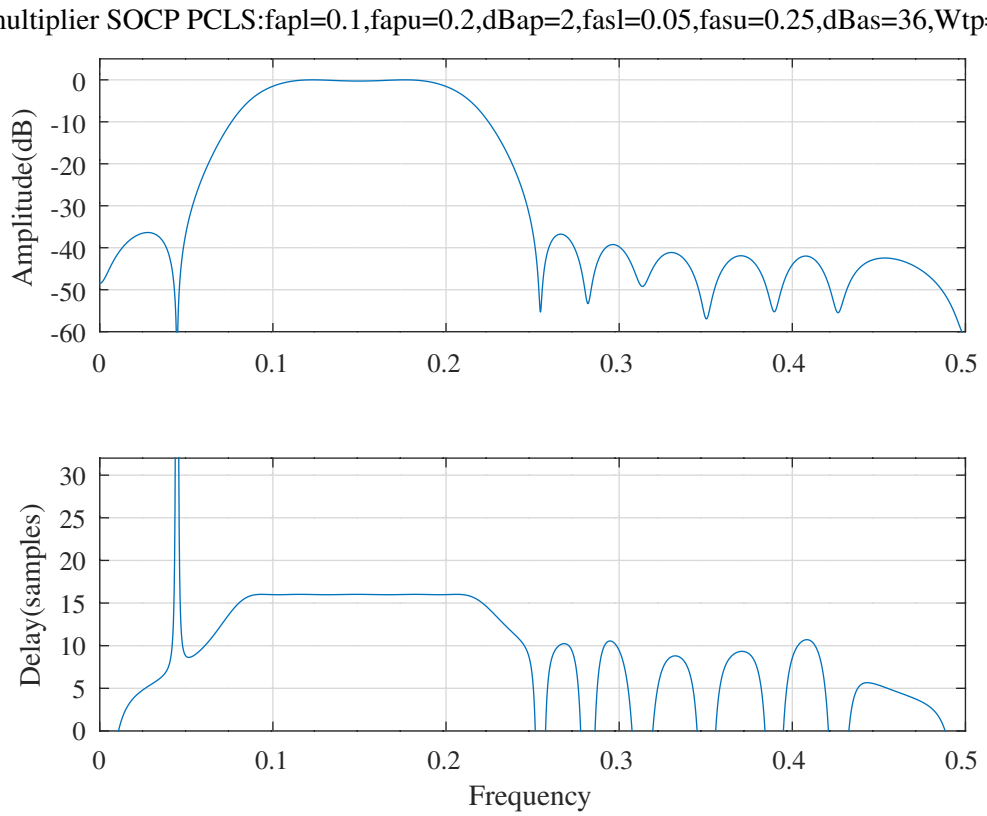



Figure 10.70: Schur one-multiplier lattice band-pass filter : response after SOCP PCLS optimisation

The Octave script *schurOneMlattice_socp_slb_bandpass_test.m* runs a PCLS SOCP pass starting with the $k2$, $epsilon2$, $p2$ and $c2$ found above and the constraint $tpr = tp/400$. Figures 10.70 and 10.71 show the overall and passband response of the band-pass filter after SOCP PCLS optimisation.

The SOCP PCLS optimised Schur one-multiplier allpass lattice and numerator tap coefficients of the band-pass filter are:

```
k3 = [   0.0000000000,   0.6606880654,   0.0000000000,   0.4941519607, ...
         0.0000000000,   0.3428225897,   0.0000000000,   0.4122471209, ...
         0.0000000000,   0.2866612230,   0.0000000000,   0.2442437414, ...
         0.0000000000,   0.1437048239,   0.0000000000,   0.0969655072, ...
         0.0000000000,   0.0332013284,   0.0000000000,   0.0142748957 ];


epsilon3 = [  0,  1,  0, -1, ...
              0,  1,  0, -1, ...
              0,  1,  0, -1, ...
              0, -1,  0,  1, ...
              0, -1,  0, -1 ];


p3 = [   1.1299629201,   1.1299629201,   0.5107633510,   0.5107633510, ...
         0.8778232446,   0.8778232446,   0.6140999624,   0.6140999624, ...
         0.9519125162,   0.9519125162,   0.7087822734,   0.7087822734, ...
         0.9094413647,   0.9094413647,   1.0510416622,   1.0510416622, ...
         0.9536205800,   0.9536205800,   0.9858255516,   0.9858255516 ];


c3 = [   0.0720225098,  -0.0121370667,  -0.2985655612,  -0.4829959396, ...
        -0.1634197124,   0.1208005840,   0.3919384761,   0.2962671586, ...
         0.0154030969,  -0.0830746748,  -0.0776318106,  -0.0109900562, ...
        -0.0090269506,  -0.0339610266,  -0.0245173699,   0.0059251362, ...
         0.0244517007,   0.0150950869,   0.0015193045,   0.0014569594, ...
         0.0066575132 ];
```

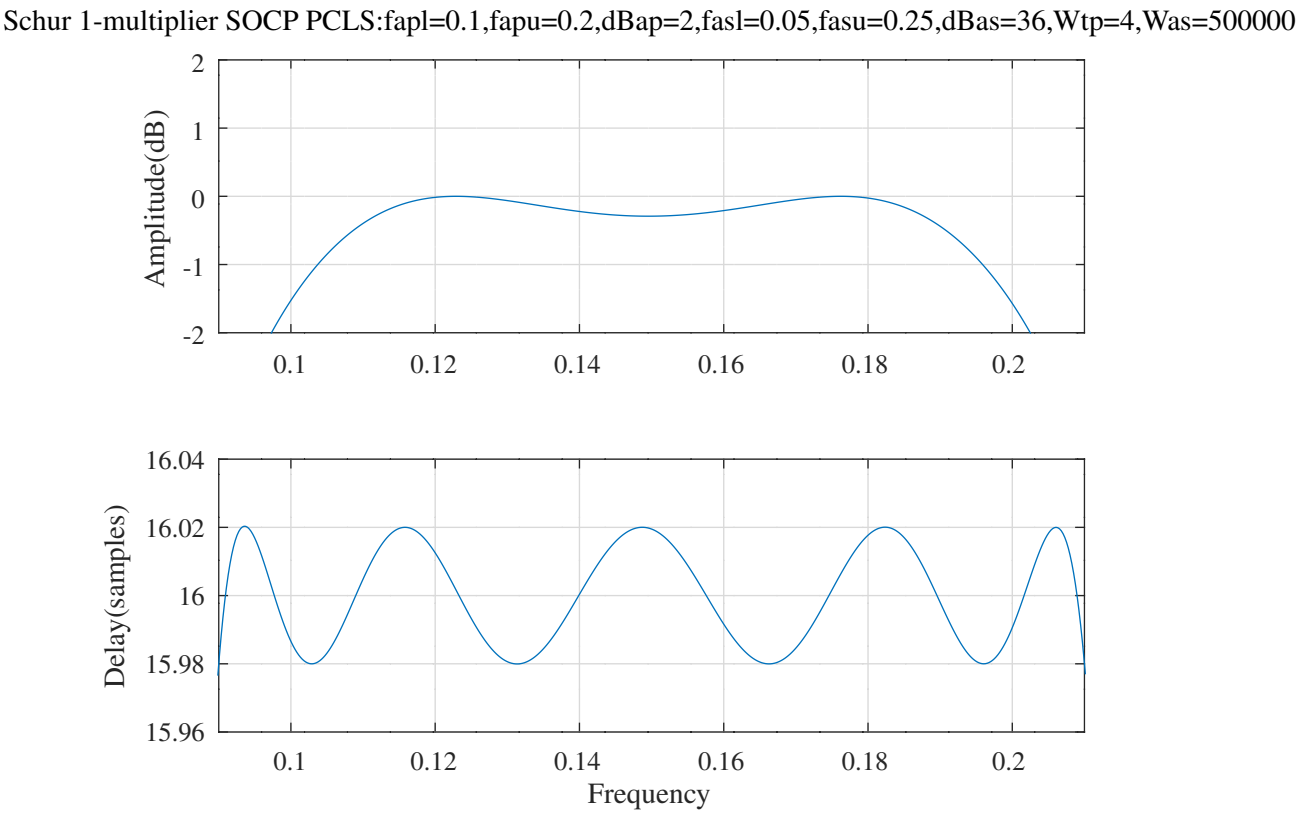



Figure 10.71: Schur one-multiplier lattice band-pass filter : passband response after SOCP PCLS optimisation

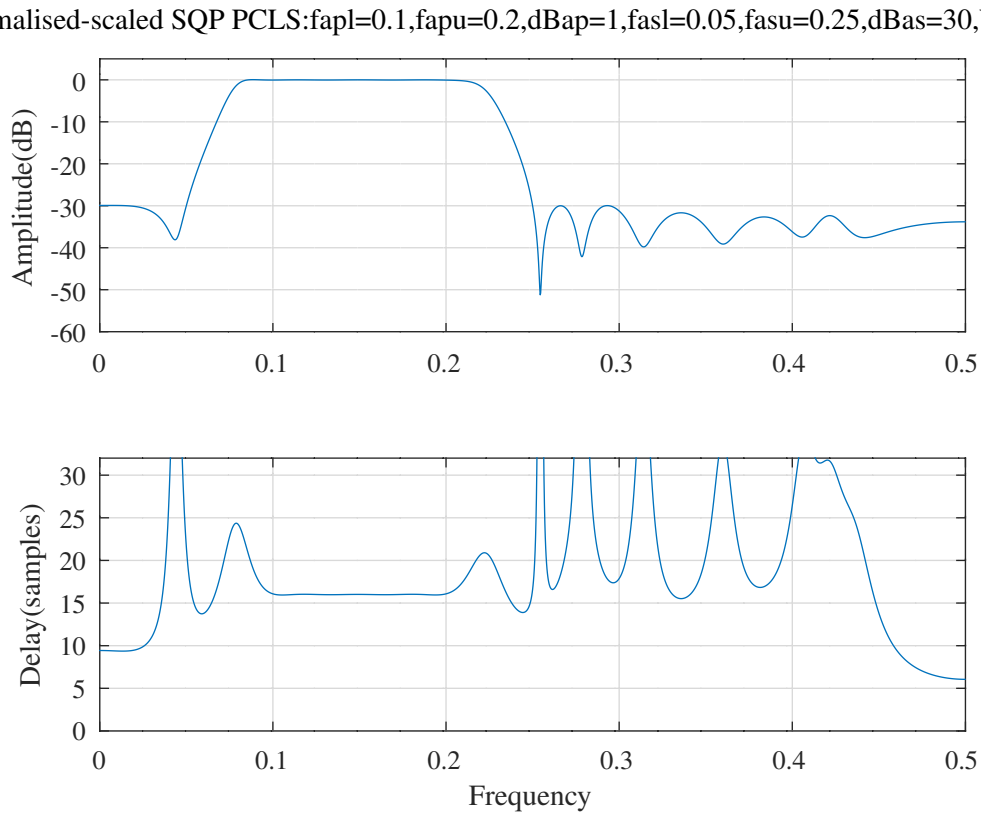



Figure 10.72: Schur normalised-scaled lattice band-pass filter : response after SQP PCLS optimisation

### 10.3.7 Design of an IIR normalised-scaled Schur lattice band-pass filter using SQP

The Octave script *schurNSlattice_sqp_slb_bandpass_test.m* implements the design of a band-pass IIR normalised-scaled Schur lattice filter with SQP and PCLS. The specification of the filter is:

```
tol=0.0005 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=500 % Frequency points across the band
dmax=0.050000 % Constraint on norm of coefficient SQP step size
rho=0.999500 % Constraint on allpass coefficients
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=1 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
ftpl=0.1 % Delay pass band lower edge
ftpu=0.2 % Delay pass band upper edge
tp=16 % Nominal passband filter group delay
tpr=0.16 % Delay pass band peak-to-peak ripple
Wtp=0.1 % Delay pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
dBas=30 % Amplitude stop band peak-to-peak ripple
Wasl=10 % Ampl. lower stop band weight
Wasu=20 % Ampl. upper stop band weight
```

The initial filter is that of the Octave script *iir_sqp_slb_bandpass_test.m*, as shown in Section 8.2.6. The denominator polynomial of the filter has coefficients in $z^2$ only.

Figures 10.72 and 10.73 show the overall and passband response of the band-pass filter after SQP PCLS optimisation. Figure 10.74 shows the pole-zero plot of the band-pass filter after SQP PCLS optimisation.

The SQP PCLS optimised Schur normalised-scaled allpass lattice and numerator tap coefficients of the band-pass filter are:

```
s00_2 = [   1.0000000000,   0.6439903342,   1.0000000000,   0.5628274523, ...
```

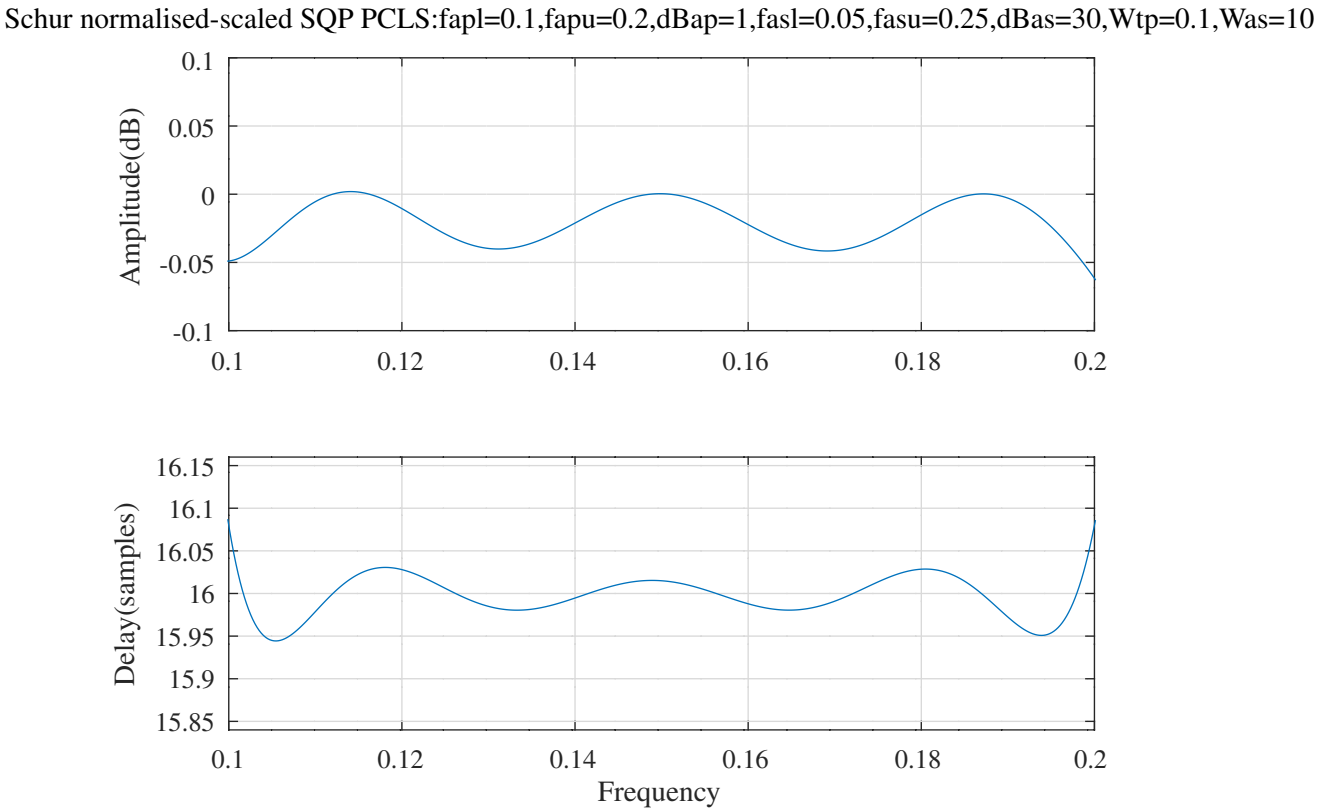



Figure 10.73: Schur normalised-scaled lattice band-pass filter : passband response after SQP PCLS optimisation

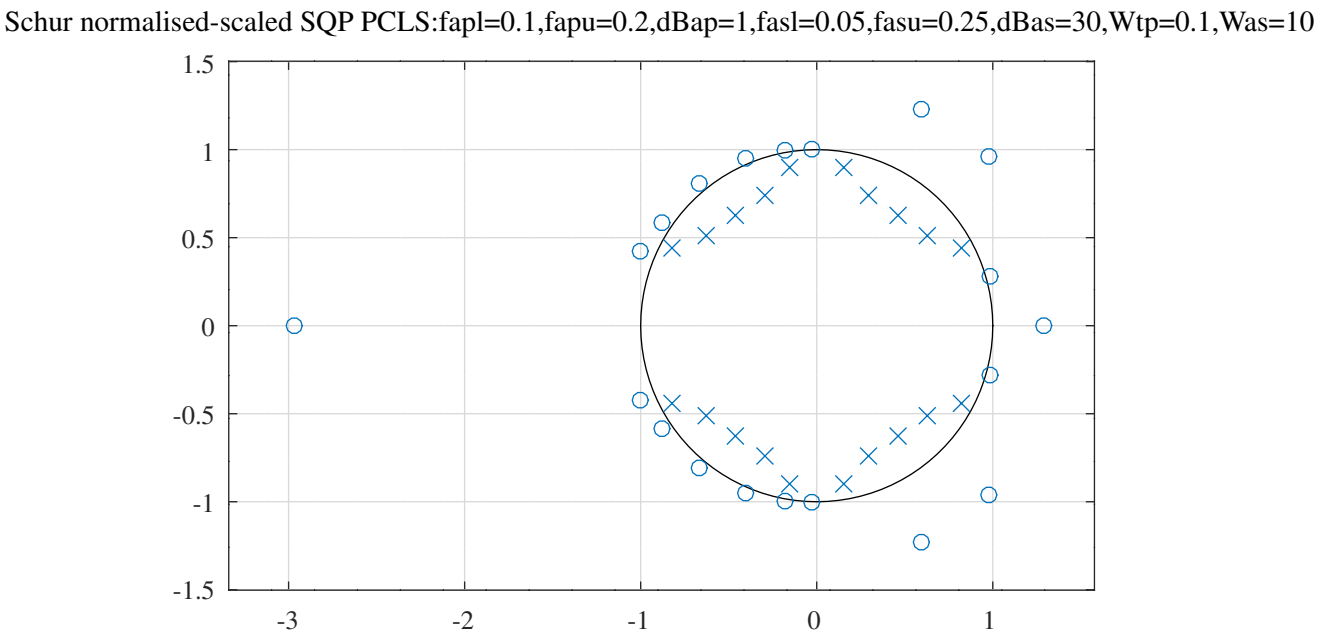



Figure 10.74: Schur normalised-scaled lattice band-pass filter:pole-zero plot after SQP PCLS optimisation

```
            1.0000000000,   0.6302257235,   1.0000000000,   0.9408796863, ...
            1.0000000000,   0.9995000000,   1.0000000000,   0.9833424566, ...
            1.0000000000,   0.6334137245,   1.0000000000,   0.9995000000, ...
            1.0000000000,   0.5851065461,   1.0000000000,   0.4652669135 ];


s02_2 = [  -0.0000000000,  -0.9961709211,  -0.0000000000,  -0.5526613224, ...
           -0.0000000000,  -0.1915872102,  -0.0000000000,  -0.5348203616, ...
           -0.0000000000,  -0.1077532145,  -0.0000000000,  -0.7235143011, ...
           -0.0000000000,  -0.4342404272,  -0.0000000000,  -0.3969491302, ...
           -0.0000000000,  -0.1181655975,  -0.0000000000,  -0.3495541241 ];


s22_2 = [   1.0000000000,   0.6598661199,   1.0000000000,   0.5971500059, ...
            1.0000000000,   0.8696260047,   1.0000000000,   0.9995000000, ...
            1.0000000000,   0.9951059199,   1.0000000000,   0.9995000000, ...
            1.0000000000,   0.9995000000,   1.0000000000,   0.7824300935, ...
            1.0000000000,   0.9995000000,   1.0000000000,   0.9925820455 ];


s20_2 = [   0.0000000000,  -0.0103649490,   0.0000000000,   0.4562501142, ...
            0.0000000000,   0.3013742707,   0.0000000000,   0.3161227445, ...
            0.0000000000,   0.1976081866,   0.0000000000,   0.1526246705, ...
            0.0000000000,   0.4817726007,   0.0000000000,   0.9995000000, ...
            0.0000000000,  -0.4893175223,   0.0000000000,   0.1215766546 ];


s10_2 = [   1.3507071547,  -1.1820186779,  -1.6437276195,  -0.6380884326, ...
            0.9272982801,   0.9861605953,   0.6154409027,  -0.1139088806, ...
           -0.4659695764,  -0.2674966421,   0.0903048044,  -0.7035690227, ...
           -0.5433156193,  -0.3103331374,   0.2207050416,   0.1445594437, ...
           -0.3011949729,  -0.0550808850,   0.1208662286,   0.0034907888 ];


s11_2 = [   2.2304566168,   0.4194937020,   0.8585195304,   1.2143640329, ...
            1.4247109681,   0.9355494463,   1.0127998365,   1.1176643194, ...
            1.4575950818,   1.3025932727,   3.0164777518,   0.9601971808, ...
            0.5184476060,   1.5323890850,   1.5734520136,   1.2372118495, ...
            1.4810941494,   0.8150901337,   0.6003197163,   0.1793359191 ];
```

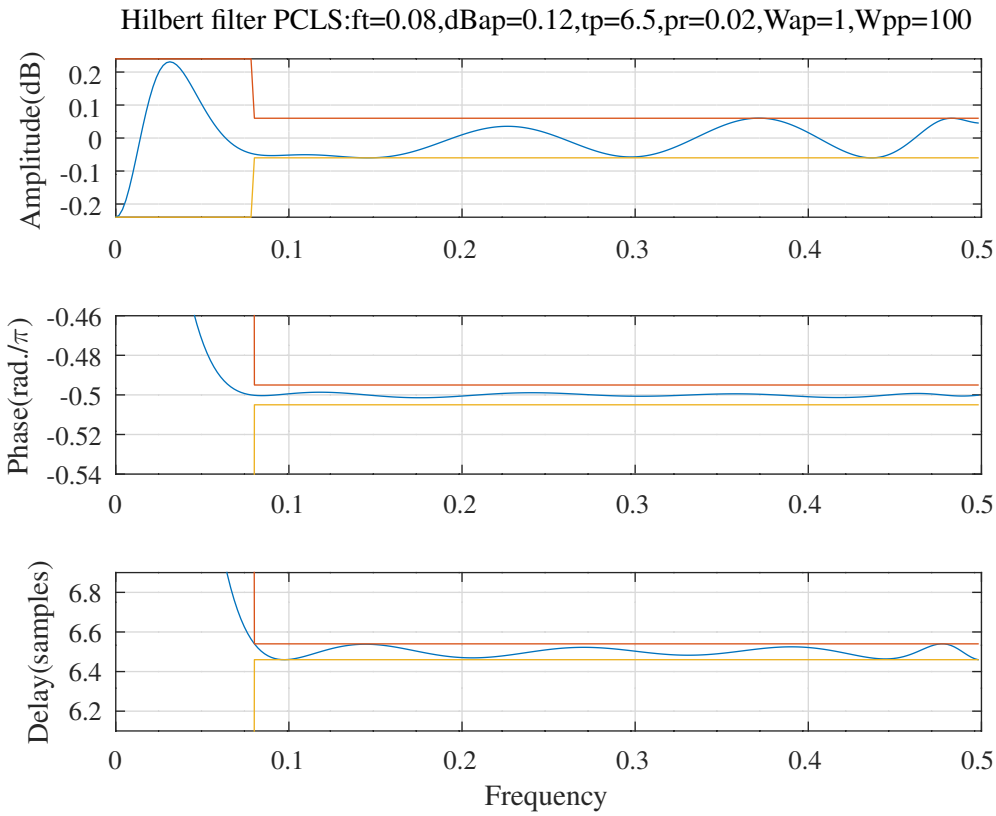



Figure 10.75: Schur one-multiplier lattice Hilbert filter : response after SOCP PCLS optimisation

### 10.3.8 Design of an IIR one-multiplier Schur lattice Hilbert filter using SOCP

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. The specification of the filter is:

```
tol=0.0001 % Tolerance on coefficient update vector
ctol=0.0001 % Tolerance on constraints
n=256 % Frequency points across the band
rho=0.999900 % Constraint on lattice coefficient magnitudes
ft=0.08 % Transition band width [0,ft]
dBap=0.12 % Amplitude pass band peak-to-peak ripple
Wat=0.001 % Amplitude transition band weight
Wap_mmse=1 % Amplitude pass band weight for MMSE
Wap_pcls=1 % Amplitude pass band weight for PCLS
tp=6.5 % Nominal pass band filter group delay
pr=0.02 pi/2 % Phase pass band peak-to-peak ripple
Wpt=0.001 % Phase transition band weight
Wpp=100 % Phase pass band 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 8.2.7. The denominator polynomial of the filter has coefficients in $z^2$ only. Figure 10.75 shows the response of the Hilbert filter after SOCP 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 $\pi$ radians. Figure 10.76 shows the pole-zero plot of the Hilbert filter after SOCP PCLS optimisation.

The SOCP PCLS optimised Schur one-multiplier all-pass lattice and numerator tap coefficients of the Hilbert filter are:

```
k2 = [   0.0000000000,  -0.8889925323,   0.0000000000,   0.4146297654, ...
         0.0000000000,  -0.0222284252,   0.0000000000,   0.0027602150, ...
        -0.0000000000,   0.0027380888,  -0.0000000000,  -0.0006160879 ];


epsilon2 = [  0, -1,  0, -1, ...
              0,  1,  0, -1, ...
              0, -1,  0,  1 ];
```

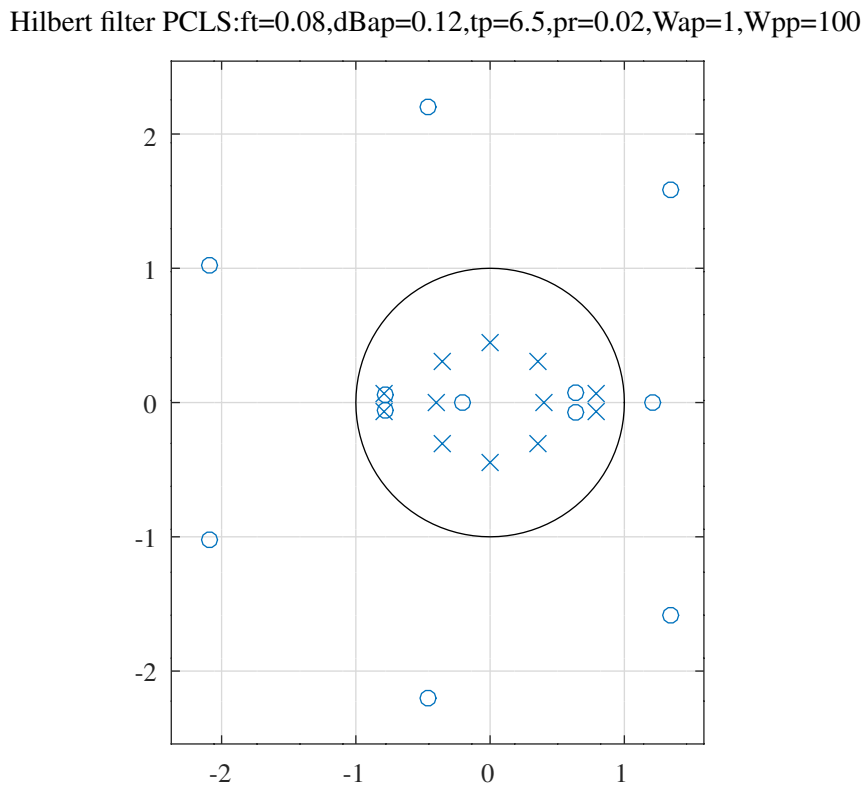



Figure 10.76: Schur one-multiplier lattice Hilbert filter:pole-zero plot after SOCP PCLS optimisation

```
p2 = [   2.5794209272,   2.5794209272,   0.6252924663,   0.6252924663, ...
         0.9720514995,   0.9720514995,   0.9939042494,   0.9939042494, ...
         0.9966514354,   0.9966514354,   0.9993841018,   0.9993841018 ];


c2 = [   0.0434675760,   0.0533736333,   0.2293089733,   0.2934949306, ...
         0.2692924433,   0.7093497276,  -0.5854091195,  -0.1568278375, ...
        -0.0729023818,  -0.0376923287,  -0.0206625621,  -0.0118584743, ...
        -0.0069806675 ];
```

The corresponding denominator and numerator transfer function polynomials are:

```
d2 = [   1.0000000000,  -0.0000000000,  -1.2668673487,  -0.0000000000, ...
         0.4437357548,  -0.0000000000,  -0.0244948185,  -0.0000000000, ...
        -0.0009819763,  -0.0000000000,   0.0035185894,  -0.0000000000, ...
        -0.0006160879 ];


n2 = [  -0.0069806675,  -0.0118511684,  -0.0118062523,  -0.0225521272, ...
        -0.0495949859,  -0.1135381127,  -0.4987797275,   0.8704260200, ...
         0.9670432491,  -0.7679738447,  -0.4543183259,   0.1980821631, ...
         0.0519329524 ];
```

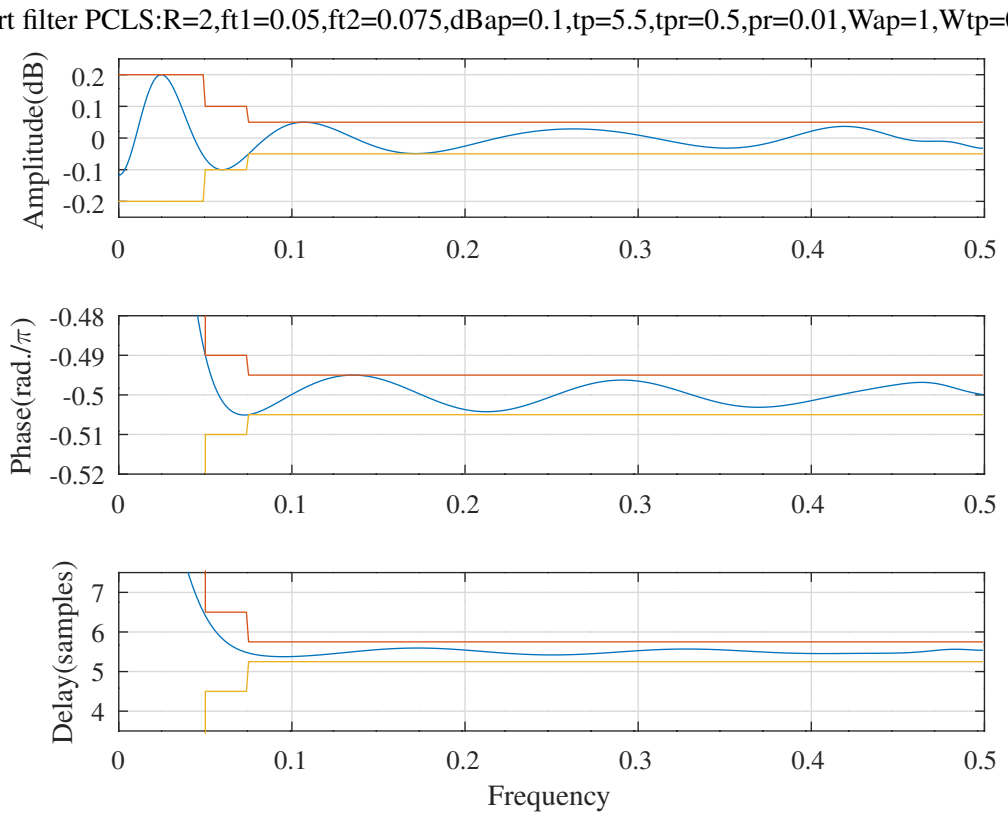



Figure 10.77: Schur one-multiplier lattice Hilbert filter : response after SQP PCLS optimisation

### 10.3.9 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
ctol=0.0001 % Tolerance on constraints
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 8.2.7. The denominator polynomial of the filter has coefficients in $z^2$ only. Figure 10.77 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 $\pi$ radians. Figure 10.78 shows the pole-zero plot of the Hilbert filter 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,  -0.9135081371,   0.0000000000,   0.5546428077, ...
         0.0000000000,  -0.0966755724,   0.0000000000,   0.0007462344, ...
         0.0000000000,  -0.0020582170,   0.0000000000,   0.0015840384 ];
```

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

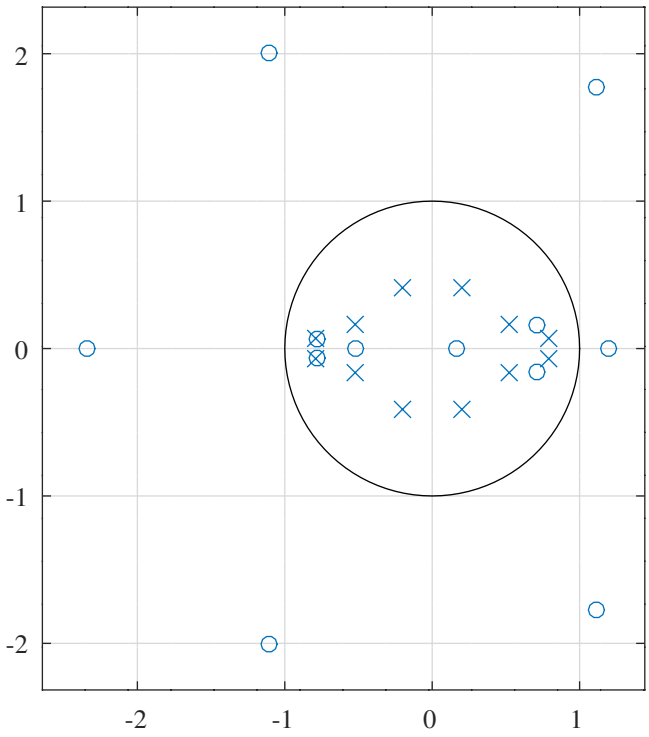



Figure 10.78: Schur one-multiplier lattice Hilbert filter:pole-zero plot after SQP PCLS optimisation

```
epsilon2 = [  0, -1,  0, -1, ...
              0,  1,  0, -1, ...
              0,  1,  0, -1 ];


p2 = [   2.2748006759,   2.2748006759,   0.4836328786,   0.4836328786, ...
         0.9036013965,   0.9036013965,   0.9956211212,   0.9956211212, ...
         0.9963643654,   0.9963643654,   0.9984172142,   0.9984172142 ];


c2 = [   0.0435189929,   0.0508921401,   0.2428519496,   0.2882277487, ...
         0.1877845013,   0.2854412265,   0.6862739758,  -0.5909939643, ...
        -0.1639093080,  -0.0790058477,  -0.0435858016,  -0.0252833113, ...
        -0.0155222724 ];
```

The corresponding denominator and numerator transfer function polynomials are:

```
d2 = [   1.0000000000,  -0.0000000000,  -1.4738762048,  -0.0000000000, ...
         0.6926629907,  -0.0000000000,  -0.0993577339,  -0.0000000000, ...
         0.0048769769,  -0.0000000000,  -0.0043928883,  -0.0000000000, ...
         0.0015840384 ];


n2 = [  -0.0155222724,  -0.0252432615,  -0.0206388521,  -0.0415129857, ...
        -0.1099263712,  -0.4898681276,   0.8953685694,   1.0719004661, ...
        -0.9469562874,  -0.6486747896,   0.3460726371,   0.1328074275, ...
        -0.0279441389 ];
```

### 10.3.10 Design of a parallel IIR Schur one-multiplier all-pass lattice band-pass Hilbert filter using SOCP

The Octave script *schurOneMPAlattice_socp_slb_bandpass_hilbert_test.m* implements the design of a band-pass IIR Hilbert filter composed of the difference of two parallel one-multiplier Schur lattice all-pass filters with SOCP and PCLS optimisation. The specification of the filter is:

```
m1=10 % Allpass model filter 1 denominator order
m2=10 % Allpass model filter 2 denominator order
difference=1 % Use difference of all-pass filters
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
rho=0.999000 % Constraint on all-pass pole radius
n=1000 % Frequency points across the band
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
dBap=0.200000 % Pass band amplitude response ripple(dB)
Wap=1 % Pass band amplitude response weight
Watl=0.001 % Lower transition band amplitude response weight
Watu=0.001 % Upper transition band amplitude response weight
fasl=0.05 % Stop band amplitude response lower edge
fasu=0.25 % Stop band amplitude response upper edge
dBas=30.000000 % Stop band amplitude response ripple(dB)
Wasl=100 % Lower stop band amplitude response weight
Wasu=100 % Upper stop band amplitude response weight
ftpl=0.11 % Pass band group-delay response lower edge
ftpu=0.19 % Pass band group-delay response upper edge
td=16.000000 % Pass band nominal group-delay response(samples)
tdr=0.080000 % Pass band group-delay response ripple(samples)
Wtp=10 % Pass band group-delay response weight
fppl=0.11 % Pass band phase response lower edge
fppu=0.19 % Pass band phase response upper edge
pd=1.500000 % Pass band nominal phase response(rad./pi)
pdr=0.002000 % Pass band phase response ripple(rad./pi)
Wpp=200 % Pass band phase response weight
```

The initial parallel all-pass filters were designed by the Octave script *tarczynski_parallel_allpass_bandpass_hilbert_test.m*. Figures 10.79 and 10.80 show the overall and passband responses of the filter after SOCP PCLS optimisation. The phase response shown is adjusted for the nominal delay. The band-pass filter SOCP PCLS optimised Schur one-multiplier all-pass lattice coefficients are:

```
A1k = [  -0.4135004229,   0.6119694730,   0.4794873904,  -0.5067394002, ...
          0.7072993736,  -0.3010516772,  -0.0646071285,   0.3675203809, ...
         -0.2563697410,   0.0890435079 ];


A1epsilon = [ -1, -1,  1,  1, ...
               1,  1,  1, -1, ...
              -1, -1 ];


A2k = [  -0.7696310526,   0.7216132469,   0.4481643290,  -0.5762702648, ...
          0.7320010792,  -0.2531067642,  -0.0688876567,   0.3754458362, ...
         -0.2352692288,   0.1058925849 ];


A2epsilon = [ -1, -1,  1, -1, ...
              -1,  1, -1,  1, ...
               1, -1 ];
```

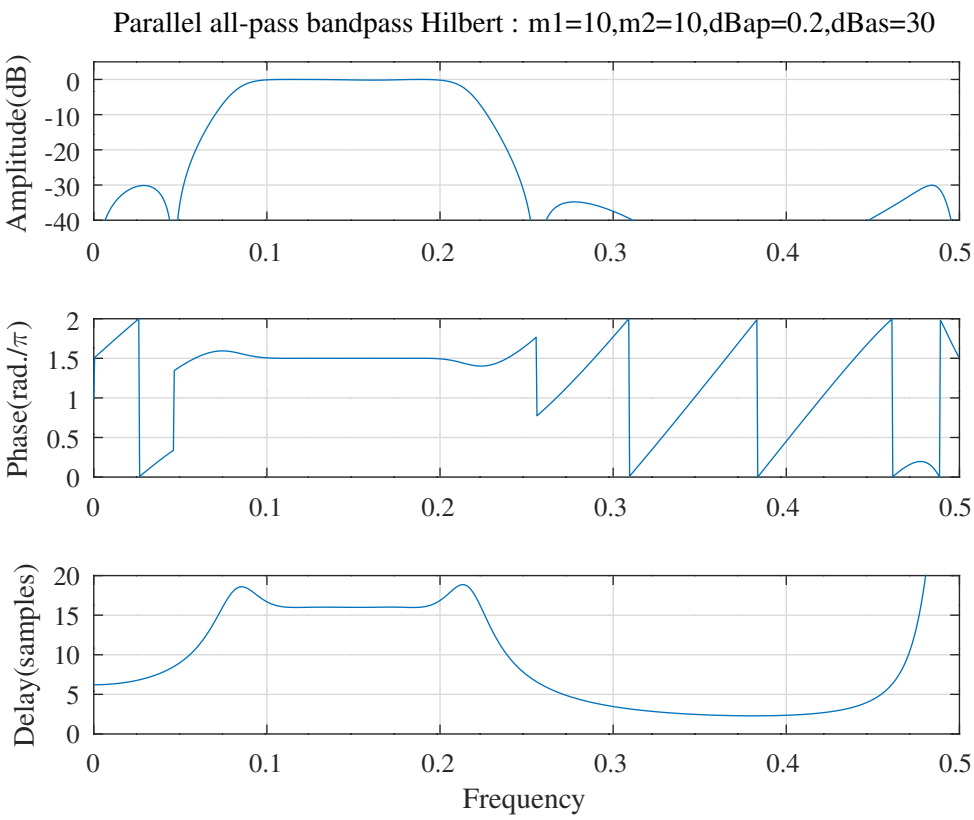



Figure 10.79: Parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter : response after SOCP PCLS optimisation. The phase response shown is adjusted for the nominal delay.

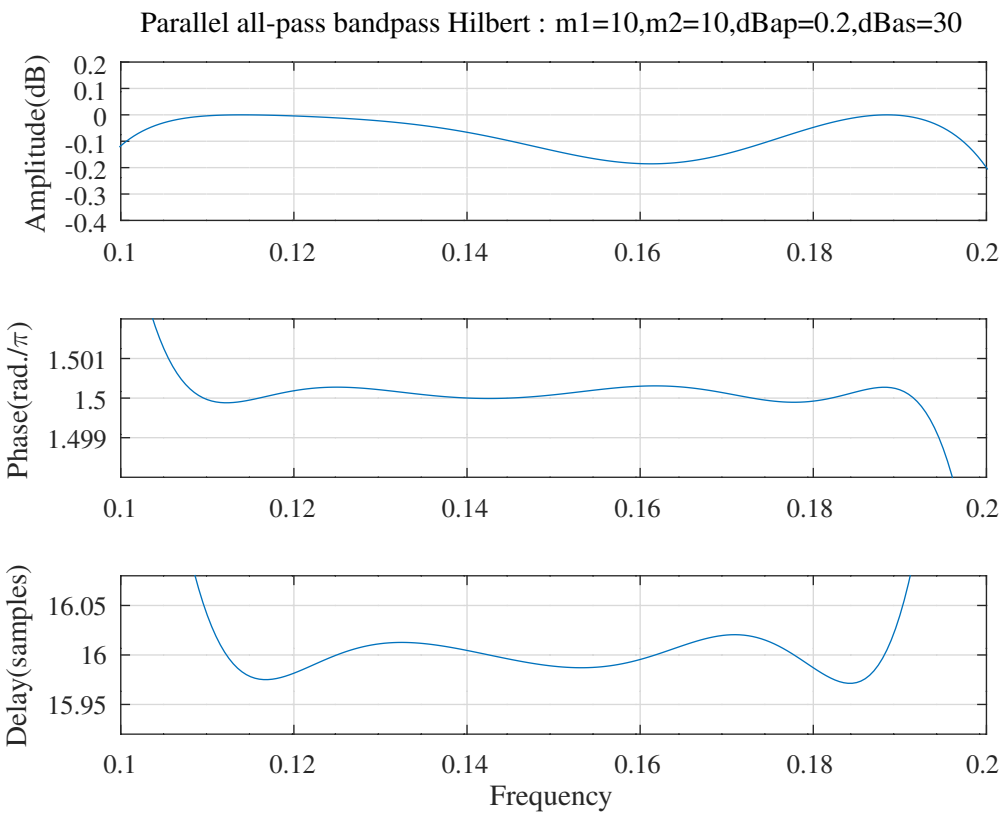



Figure 10.80: Parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter : passband response after SOCP PCLS optimisation. The phase response shown is adjusted for the nominal delay.

### 10.3.11 Design of a parallel IIR Schur one-multiplier all-pass lattice multi-band-pass filter using SOCP

The Octave script *schurOneMPAlattice_socp_slb_lowpass_to_multiband_test.m* implements the design of a multi-band-pass IIR filter composed of the difference of two parallel one-multiplier Schur lattice all-pass filters with SOCP and PCLS optimisation. The specification of the filter is:

```
tol=0.0001 % Tolerance on combined response
mtol=0.0001 % Tolerance on MMSE update
ptol=0.0001 % Tolerance on PCLS update
ctol=0.0001 % Tolerance on constraints
maxiter=10000 % SOCP iteration limit
npoints=1000 % Frequency points across the band
nplot=1000 % Frequency points plotted across the band
rho=0.996094 % Constraint on allpass coefficients
fas1u=0.05 % Amplitude stop band 1 upper edge
fap1l=0.075 % Amplitude pass band 1 lower edge
fap1u=0.1 % Amplitude pass band 1 upper edge
fas2l=0.125 % Amplitude stop band 2 lower edge
fas2u=0.15 % Amplitude stop band 2 upper edge
fap2l=0.175 % Amplitude pass band 2 lower edge
fap2u=0.225 % Amplitude pass band 2 upper edge
fas3l=0.25 % Amplitude stop band 3 lower edge
dBas1=20 % Amplitude stop band 1 attenuation
dBap1=1 % Amplitude pass band 1 peak-to-peak ripple
dBas2=20 % Amplitude stop band 2 attenuation
dBap2=1 % Amplitude pass band 2 peak-to-peak ripple
dBas3=20 % Amplitude stop band 3 attenuation
Was1=1 % Amplitude stop band 1 weight
Wap1=1 % Amplitude pass band 1 weight
Was2=1 % Amplitude stop band 2 weight
Wap2=1 % Amplitude pass band 2 weight
Was3=1 % Amplitude stop band 3 weight
ftp1l=0.08 % Delay pass band 1 lower edge
ftp1u=0.095 % Delay pass band 1 upper edge
ftp2l=0.185 % Delay pass band 2 lower edge
ftp2u=0.215 % Delay pass band 2 upper edge
tp1=23 % Nominal pass band 1 filter group delay
tp2=13 % Nominal pass band 2 filter group delay
tpr1=2 % Delay pass band 1 peak-to-peak ripple
tpr2=2 % Delay pass band 2 peak-to-peak ripple
Wtp1=0.005 % Delay pass band 1 weight
Wtp2=0.005 % Delay pass band 2 weight
```

The initial parallel all-pass filters are designed by the frequency transformation of a prototype elliptic low-pass filter. The parallel all-pass multi-band filter orders are 12 and 8. The SOCP PCLS optimised lattice coefficients are:

```
A1kpcls = [  -0.8284814286,   0.9619285274,  -0.5895615255,   0.8587700995, ...
             -0.7055500672,   0.7605379424,  -0.4239830987,   0.7855591572 ]';


A1epsilon0 = [  1,  1,  1, -1, ...
               -1,  1,  1, -1 ];


A2kpcls = [  -0.8236548196,   0.9115734805,  -0.8137477287,   0.8741940439, ...
             -0.7512126727,   0.9917707622,  -0.2544288989,   0.9046419603, ...
             -0.7502018217,   0.8162944796,  -0.4682769792,   0.7041672063 ]';


A2epsilon0 = [  1,  1,  1, -1, ...
               -1,  1,  1, -1, ...
                1,  1,  1, -1 ];
```

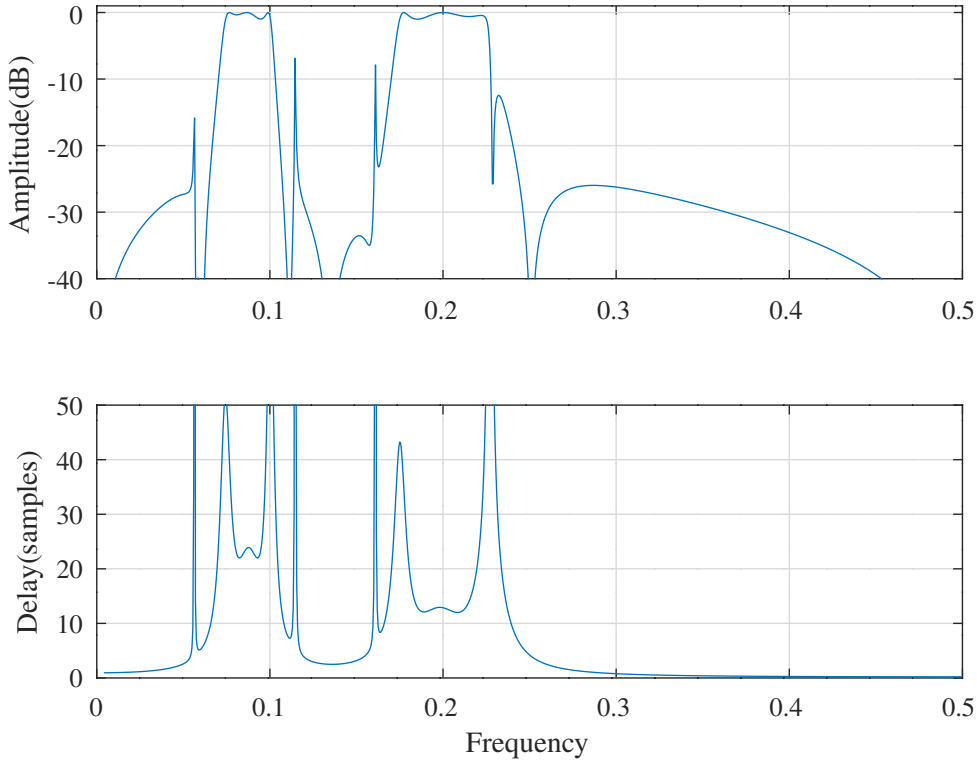



Figure 10.81: Schur one-multiplier parallel all-pass lattice multi-band filter : response after SOCP PCLS optimisation. The initial filter is designed by a low-pass to multi-band frequency transformation.

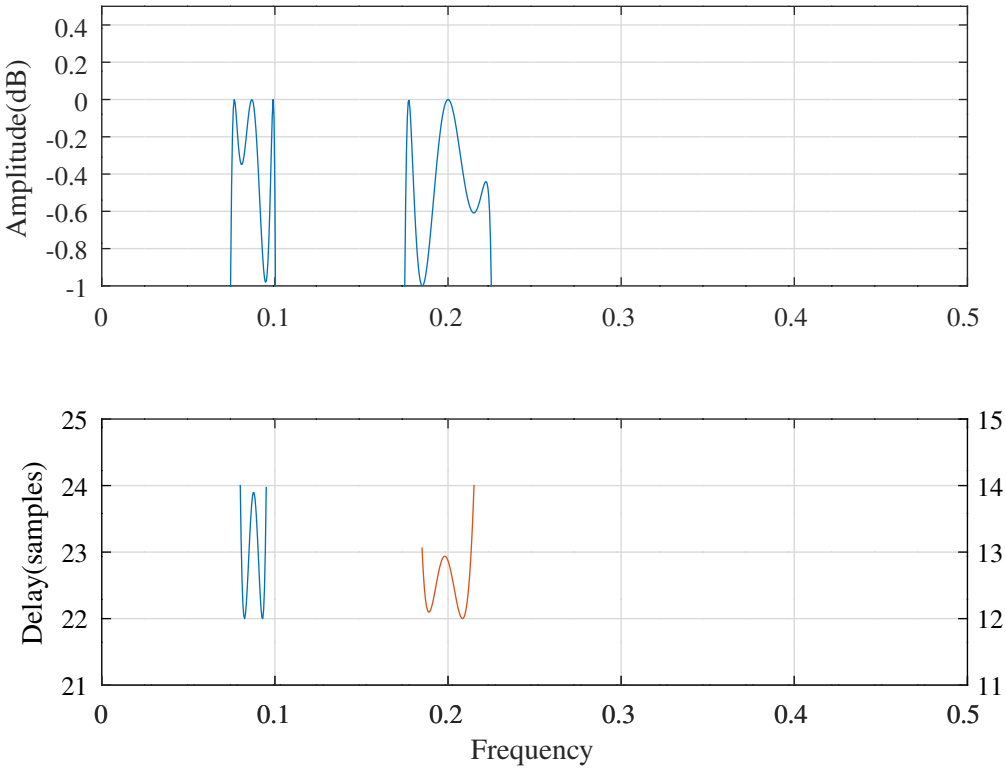



Figure 10.82: Schur one-multiplier parallel all-pass lattice multi-band filter : pass-band response after SOCP PCLS optimisation. The initial filter is designed by a low-pass to multi-band frequency transformation.

Figure 10.81 shows the amplitude and delay response of the PCLS SOCP optimised parallel all-pass filter. Figure 10.82 shows the pass-band response of the filter.

The Octave script *schurOneMPAlattice_socp_slb_multiband_test.m* implements the design of a multi-band-pass IIR filter composed of the difference of two parallel one-multiplier Schur lattice all-pass filters with SOCP and PCLS optimisation. The specification of the filter is:

```
tol=0.0001 % Tolerance on combined response
mtol=0.0001 % Tolerance on MMSE update
ptol=0.0001 % Tolerance on PCLS update
ctol=1e-06 % Tolerance on constraints
maxiter=2000 % SOCP iteration limit
npoints=1000 % Frequency points across the band
nplot=1000 % Frequency points plotted across the band
rho=0.996094 % Constraint on allpass coefficients
ma=14 % Order of the first all-pass filter
mb=14 % Order of the second all-pass filter
fas1u=0.05 % Amplitude stop band 1 upper edge
fap1l=0.08 % Amplitude pass band 1 lower edge
fap1u=0.095 % Amplitude pass band 1 upper edge
fas2l=0.125 % Amplitude stop band 2 lower edge
fas2u=0.15 % Amplitude stop band 2 upper edge
fap2l=0.175 % Amplitude pass band 2 lower edge
fap2u=0.225 % Amplitude pass band 2 upper edge
fas3l=0.25 % Amplitude stop band 3 lower edge
dBas1=30 % Amplitude stop band 1 attenuation
dBap1=1 % Amplitude pass band 1 peak-to-peak ripple
dBas2=20 % Amplitude stop band 2 attenuation
dBap2=1 % Amplitude pass band 2 peak-to-peak ripple
dBas3=30 % Amplitude stop band 3 attenuation
Was1=1 % Amplitude stop band 1 weight
Wap1=2 % Amplitude pass band 1 weight
Was2=2 % Amplitude stop band 2 weight
Wap2=1 % Amplitude pass band 2 weight
Was3=1 % Amplitude stop band 3 weight
ftp1l=0.08 % Delay pass band 1 lower edge
ftp1u=0.095 % Delay pass band 1 upper edge
ftp2l=0.175 % Delay pass band 2 lower edge
ftp2u=0.225 % Delay pass band 2 upper edge
tp1=20 % Nominal pass band 1 filter group delay
tp2=20 % Nominal pass band 2 filter group delay
tpr1=2 % Delay pass band 1 peak-to-peak ripple
tpr2=2 % Delay pass band 2 peak-to-peak ripple
Wtp1=0.01 % Delay pass band 1 weight
Wtp2=0.01 % Delay pass band 2 weight
```

The initial parallel all-pass filters are designed by the Octave script *tarczynski_parallel_allpass_multiband_test.m*. The SOCP PCLS optimised lattice coefficients are:

```
A1kpcls = [   0.3937699976,  -0.1585331104,   0.5419332651,   0.7464239293, ...
             -0.3101811510,   0.2345562806,   0.5290213004,  -0.1119670934, ...
              0.1906766833,   0.3998007792,   0.0121434701,   0.0421709556, ...
              0.2223422696,  -0.0948025457 ]';


A1epsilon0 = [ -1, -1, -1,  1, ...
                1, -1,  1,  1, ...
               -1, -1, -1,  1, ...
                1,  1 ];


A2kpcls = [   0.7372729376,   0.1062352603,   0.2850330805,   0.6316353093, ...
             -0.3639831328,   0.1650744445,   0.4568902653,  -0.1981064709, ...
              0.1468021536,   0.3749287013,  -0.0095159013,   0.0604562239, ...
              0.2722433632,  -0.0327703020 ]';
```

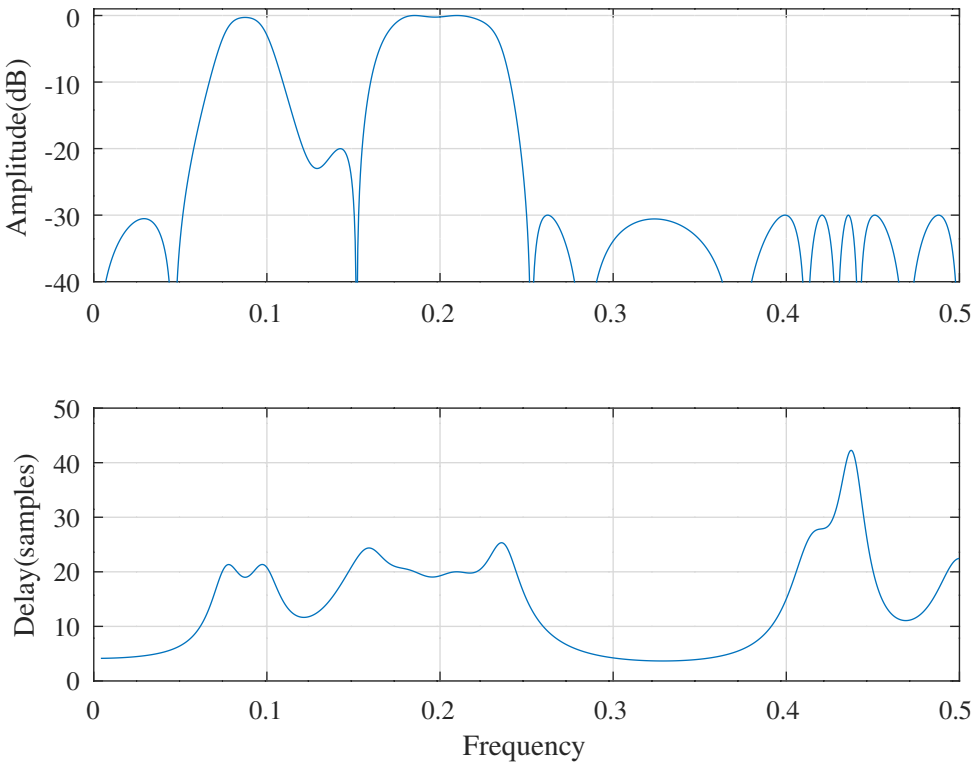



Figure 10.83: Schur one-multiplier parallel all-pass lattice multi-band filter : response after SOCP PCLS optimisation.

```
A2epsilon0 = [  1, -1, -1,  1, ...
                1, -1,  1,  1, ...
               -1, -1,  1, -1, ...
                1,  1 ];
```

Figure 10.83 shows the amplitude and delay response of the PCLS SOCP optimised parallel all-pass filter. Figure 10.84 shows the pass-band response of the filter.

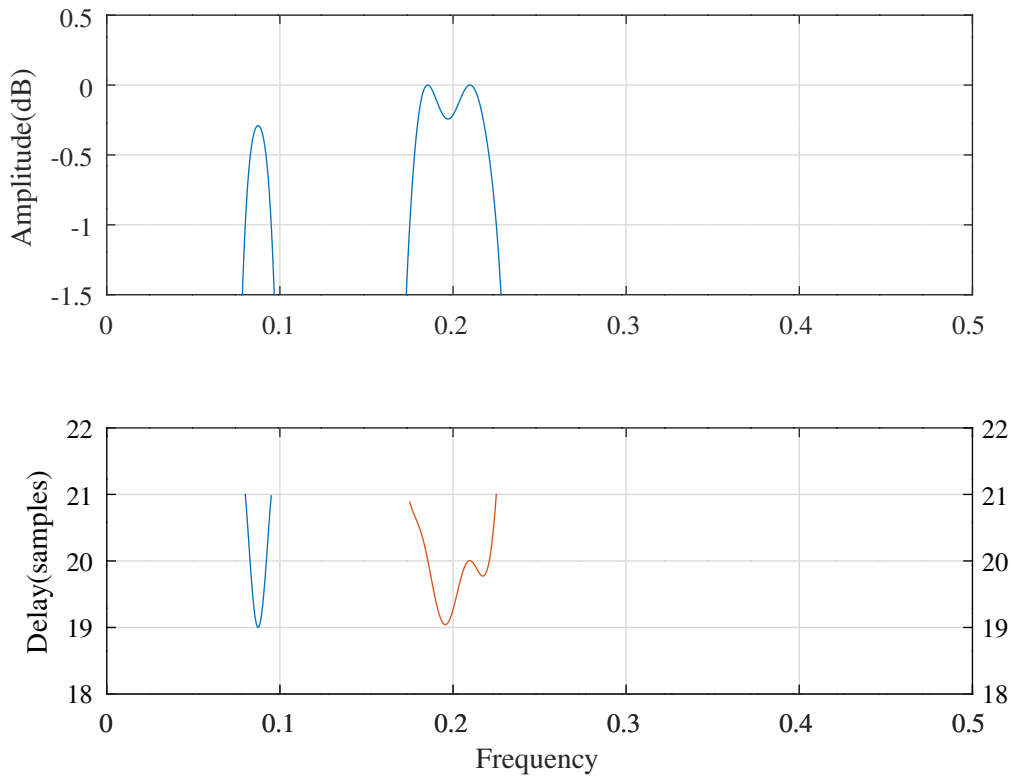



Figure 10.84: Schur one-multiplier parallel all-pass lattice multi-band filter : pass-band response after SOCP PCLS optimisation.

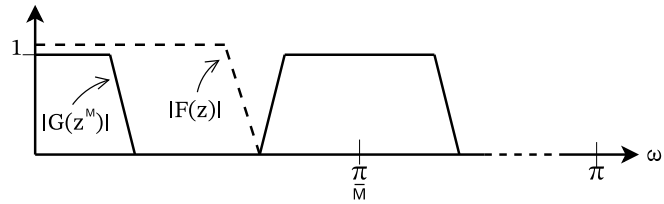



Figure 10.85: Simple frequency response masking filter

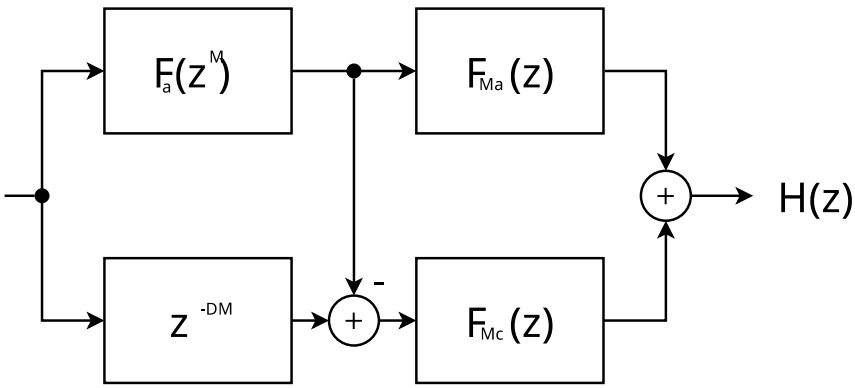



Figure 10.86: Lim's frequency response masking filter structure

## 10.4 Design of IIR filters with a sharp transition band by frequency response masking

### 10.4.1 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 10.85. This simple frequency response masking filter technique is only suitable for designing narrow-band filters. Lim [222] describes the design of digital filters with sharp transition bands with wider bandwidths using FIR model and masking filters. Lu and Hinamoto [212] describe the design of digital filters with an IIR model filter and FIR masking filters using a structure, shown in Figure 10.86, 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 $F_a(z)$.

Johansson and Wanhammar [64] describe design of frequency response masking filters with a model filter consisting of parallel all-pass filters, FIR masking filters and the structure shown in Figure 10.87. Johansson and Wanhammar specify a delay line in one arm of the model filter if approximately linear phase response is desired.

In the following I use Lim's description of frequency response masking for the two cases shown in Figure 10.88 [222, Section III, Figure 4]. Figure 10.88a shows the response of a prototype filter a with frequency response, $F_a(\omega)$, having passband and

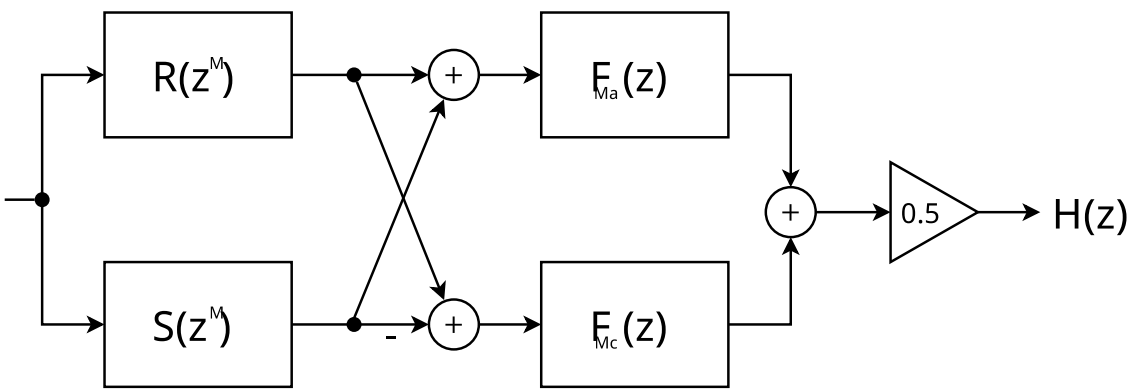



Figure 10.87: Johansson and Wanhammar's frequency response masking filter structure

stopband edge frequencies of $\theta$ and $\phi$ respectively. Figure 10.88b shows the response of the *complementary* filter $F_c(z) = z^{-D} - F_a(z)$. Figure 10.88c shows the responses of the filters $F_a'(z) = F_a\left(z^M\right)$ and $F_c'(z) = F_c\left(z^M\right)$. 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 10.88d, $F_{M_a}$ 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 10.88e, has passband edge $\omega_p = \frac{2m\pi+\theta}{M}$ and stopband edge $\omega_s = \frac{2m\pi+\phi}{M}$. Alternatively, Figures 10.88f and 10.88g 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 10.88a, for Figure 10.88e

$$\begin{aligned} m &= \lfloor \frac{\omega_p M}{2\pi} \rfloor \\ \theta &= \omega_p M - 2m\pi \\ \phi &= \omega_s M - 2m\pi \end{aligned}$$

and for Figure 10.88g

$$\begin{aligned} m &= \lceil \frac{\omega_s M}{2\pi} \rceil \\ \theta &= 2m\pi - \omega_s M \\ \phi &= 2m\pi - \omega_p M \end{aligned}$$

where $\lfloor x \rfloor$ and $\lceil x \rceil$ represent the largest integer less than $x$ and the smallest integer larger than $x$ respectively.

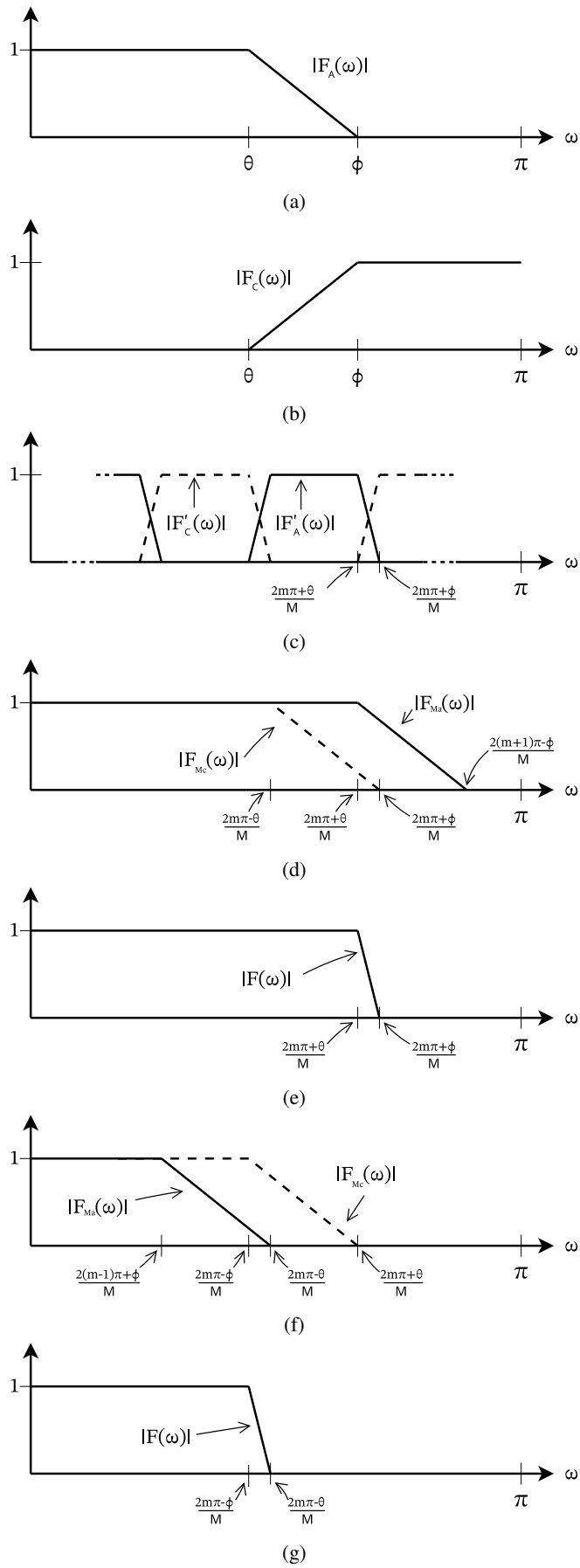



Figure 10.88: Frequency response masking with Lim's filter structure

### 10.4.2 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 [212, Section V] The FRM filter structure is shown in Figure 10.86. 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)

$$d(z) = \begin{cases} \left(1 + d_0 z^{-1}\right) \prod_{k=1}^{\frac{r-1}{2}} \left(1 + d_{k1} z^{-1} + d_{k2} z^{-2}\right), & \text{if } r \text{ odd} \\ \prod_{k=1}^{\frac{r}{2}} \left(1 + d_{k1} z^{-1} + d_{k2} z^{-2}\right), & \text{if } r \text{ even} \end{cases}$$

Define the two model filter coefficient vectors as

$$\boldsymbol{a} = \begin{bmatrix} a_0 & a_1 & \dots & a_n \end{bmatrix}^\top$$

and

$$\boldsymbol{d} = \begin{bmatrix} d_0 \\ \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_L \end{bmatrix}$$

$$\boldsymbol{d}_k = \begin{bmatrix} d_{k1} \\ d_{k2} \end{bmatrix}, \quad \text{for } 1 \le k \le L$$

$$L = \begin{cases} \frac{r-1}{2}, & \text{if } r \text{ odd} \\ \frac{r}{2}, & \text{if } r \text{ even} \end{cases}$$

The masking filters are

$$F_{M_a}(z) = \sum_{k=0}^{n_a-1} p_k z^{-k}$$

$$F_{M_c}(z) = \sum_{k=0}^{n_c-1} q_k z^{-k}$$

$F_{M_a}(z)$ 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 $F_{M_a}$)

$$\begin{aligned} F_{M_a}(\omega) &= \sum_{k=0}^{n_a-1} p_k e^{-\imath\omega k} \\ &= e^{-\imath\omega\frac{n_a-1}{2}} \begin{cases} p_{\frac{n_a-1}{2}} + 2\sum_{k=1}^{\frac{n_a-1}{2}} p_{k+\frac{n_a-1}{2}} \cos k\omega, & \text{if } n_a \text{ odd} \\ 2\sum_{k=1}^{\frac{n_a}{2}} p_{k+\frac{n_a}{2}-1} \cos\left(k - \frac{1}{2}\right)\omega, & \text{if } n_a \text{ even} \end{cases} \end{aligned}$$

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

$$e^{-\imath D_s \omega} H(\boldsymbol{x}, \omega)$$

where

$$H(\boldsymbol{x}, \omega) = \tilde{H}_a(M\omega)\left[\boldsymbol{a}_a^\top \boldsymbol{c}_a(\omega) - \boldsymbol{a}_c^\top \boldsymbol{c}_c(\omega)\right] + \boldsymbol{a}_c^\top \boldsymbol{c}_c(\omega)$$

$$
\begin{aligned}
\tilde{H}_a(\omega) &= e^{\imath D\omega}\frac{a(\omega)}{d(\omega)} = \frac{\boldsymbol{a}^\top \boldsymbol{v}(\omega)}{d(\omega)} \\
\boldsymbol{v}(\omega) &= \boldsymbol{c}(\omega) + \imath \boldsymbol{s}(\omega) \\
\boldsymbol{c}(\omega) &= \begin{bmatrix} \cos D\omega & \dots & \cos(D-n)\omega \end{bmatrix}^\top \\
\boldsymbol{s}(\omega) &= \begin{bmatrix} \sin D\omega & \dots & \sin(D-n)\omega \end{bmatrix}^\top \\
v_1(\omega) &= \cos\omega - \imath \sin\omega \\
\boldsymbol{v}_2(\omega) &= \begin{bmatrix} \cos\omega \\ \cos 2\omega \end{bmatrix} - \imath \begin{bmatrix} \sin\omega \\ \sin 2\omega \end{bmatrix}
\end{aligned}
$$

and

$$
d(\omega) = \begin{cases} [1 + d_0 v_1(\omega)] \prod_{k=1}^{L} \left[1 + \boldsymbol{d}_k^\top \boldsymbol{v}_2(\omega)\right], & \text{if } r \text{ odd} \\ \prod_{k=1}^{L} \left[1 + \boldsymbol{d}_k^\top \boldsymbol{v}_2(\omega)\right], & \text{if } r \text{ even} \end{cases}
$$

$$
\boldsymbol{a}_a = \begin{cases} \begin{bmatrix} p_{\frac{n_a-1}{2}} & p_{\frac{n_a+1}{2}} & \dots & p_{n_a-1} \end{bmatrix}^\top, & \text{if } n_a \text{ odd} \\ \begin{bmatrix} p_{\frac{n_a}{2}} & p_{\frac{n_a}{2}+1} & \dots & p_{n_a-1} \end{bmatrix}^\top, & \text{if } n_a \text{ even} \end{cases}
$$

$$
\boldsymbol{c}_a(\omega) = \begin{cases} \begin{bmatrix} 1 & 2\cos\omega & \dots & 2\cos\frac{(n_a-1)}{2}\omega \end{bmatrix}^\top, & \text{if } n_a \text{ odd} \\ \begin{bmatrix} 2\cos\frac{1}{2}\omega & \dots & 2\cos\frac{(n_a-1)}{2}\omega \end{bmatrix}^\top, & \text{if } n_a \text{ even} \end{cases}
$$

$$
\boldsymbol{a}_c = \begin{cases} \begin{bmatrix} q_{\frac{n_c-1}{2}} & q_{\frac{n_c+1}{2}} & \dots & q_{n_c-1} \end{bmatrix}^\top, & \text{if } n_c \text{ odd} \\ \begin{bmatrix} q_{\frac{n_c}{2}} & q_{\frac{n_c}{2}+1} & \dots & q_{n_c-1} \end{bmatrix}^\top, & \text{if } n_c \text{ even} \end{cases}
$$

$$
\boldsymbol{c}_c(\omega) = \begin{cases} \begin{bmatrix} 1 & 2\cos\omega & \dots & 2\cos\frac{(n_c-1)}{2}\omega \end{bmatrix}^\top, & \text{if } n_c \text{ odd} \\ \begin{bmatrix} 2\cos\frac{1}{2}\omega & \dots & 2\cos\frac{(n_c-1)}{2}\omega \end{bmatrix}^\top, & \text{if } n_c \text{ even} \end{cases}
$$

The gradient of $H(\boldsymbol{x}, \omega)$ is

$$
\frac{\partial H(\boldsymbol{x}, \omega)}{\partial \boldsymbol{x}} = \begin{bmatrix} y(\omega)\frac{\partial \tilde{H}_a(M\omega)}{\partial \boldsymbol{a}} \\ y(\omega)\frac{\partial \tilde{H}_a(M\omega)}{\partial \boldsymbol{d}} \\ \tilde{H}_a(M\omega)\boldsymbol{c}_a(\omega) \\ \left[1 - \tilde{H}_a(M\omega)\right]\boldsymbol{c}_c(\omega) \end{bmatrix}^\top
$$

where

$$
\begin{aligned}
y(\omega) &= \boldsymbol{a}_a^\top \boldsymbol{c}_a(\omega) - \boldsymbol{a}_c^\top \boldsymbol{c}_c(\omega) \\
\frac{\partial \tilde{H}_a(\omega)}{\partial \boldsymbol{a}} &= \frac{\boldsymbol{v}(\omega)}{d(\omega)} \\
\frac{\partial \tilde{H}_a(\omega)}{\partial \boldsymbol{d}} &= \begin{bmatrix} \frac{\partial \tilde{H}_a(\omega)}{\partial d_0} & \frac{\partial \tilde{H}_a(\omega)}{\partial \boldsymbol{d_1}} & \dots & \frac{\partial \tilde{H}_a(\omega)}{\partial \boldsymbol{d_L}} \end{bmatrix}
\end{aligned}
$$

with

$$
\begin{aligned}
\frac{\partial \tilde{H}_a(\omega)}{\partial d_0} &= -\tilde{H}_a(\omega)\frac{v_1(\omega)}{1 + d_0 v_1(\omega)} \\
\frac{\partial \tilde{H}_a(\omega)}{\partial \boldsymbol{d}_k} &= -\tilde{H}_a(\omega)\frac{\boldsymbol{v}_2(\omega)}{1 + \boldsymbol{d}_k^\top \boldsymbol{v}_2(\omega)}, \quad \text{for } 1 \le k \le L
\end{aligned}
$$

The overall parameter vector is

$$
\boldsymbol{x} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{d} \\ \boldsymbol{a}_a \\ \boldsymbol{a}_c \end{bmatrix}
$$

The calculation of the FRM zero-phase frequency response, $H(\boldsymbol{x}, \omega)$, 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.m* designs a FRM filter similar to that of the example shown by Lu and Hinamoto [212, 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 $\boldsymbol{d} = 1$. 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 = [  -0.0666608263,   0.2071661897,  -0.2564968336,   0.0447180016, ...
        0.2504145959,  -0.0229597383,  -0.8471343005,  -0.4399563780, ...
       -0.6442452466,   1.1969861365,  -0.1172138677 ]';
```

The model filter denominator polynomial is

```
d = [   1.0000000000,  -1.1408937575,   1.0817657800,  -1.0259629139, ...
        0.3990556127,   0.0752597882,  -0.2425741035,   0.1874928988, ...
       -0.0788587155,   0.0176300755,  -0.0017083324 ]';
```

The masking filter polynomial is

```
aa = [   0.0037133656,  -0.0092057211,  -0.0029989223,   0.0149290602, ...
        -0.0084167612,  -0.0131473986,   0.0227129384,   0.0020507608, ...
        -0.0214990382,   0.0345952033,  -0.0133732665,  -0.0454769790, ...
         0.0735105603,  -0.0105531357,  -0.1387451849,   0.2788374154, ...
         0.6785032503,   0.2788374154,  -0.1387451849,  -0.0105531357, ...
         0.0735105603,  -0.0454769790,  -0.0133732665,   0.0345952033, ...
        -0.0214990382,   0.0020507608,   0.0227129384,  -0.0131473986, ...
        -0.0084167612,   0.0149290602,  -0.0029989223,  -0.0092057211, ...
         0.0037133656 ]';
```

The complementary masking filter polynomial is

```
ac = [   0.0017221068,  -0.0048977382,  -0.0004522946,   0.0075172466, ...
        -0.0065967666,  -0.0051675524,   0.0136208789,  -0.0029579418, ...
        -0.0089738319,   0.0278743693,  -0.0228523260,  -0.0300227675, ...
         0.0718756074,  -0.0264938903,  -0.1224009983,   0.2864760949, ...
         0.6527271029,   0.2864760949,  -0.1224009983,  -0.0264938903, ...
         0.0718756074,  -0.0300227675,  -0.0228523260,   0.0278743693, ...
        -0.0089738319,  -0.0029579418,   0.0136208789,  -0.0051675524, ...
        -0.0065967666,   0.0075172466,  -0.0004522946,  -0.0048977382, ...
         0.0017221068 ]';
```

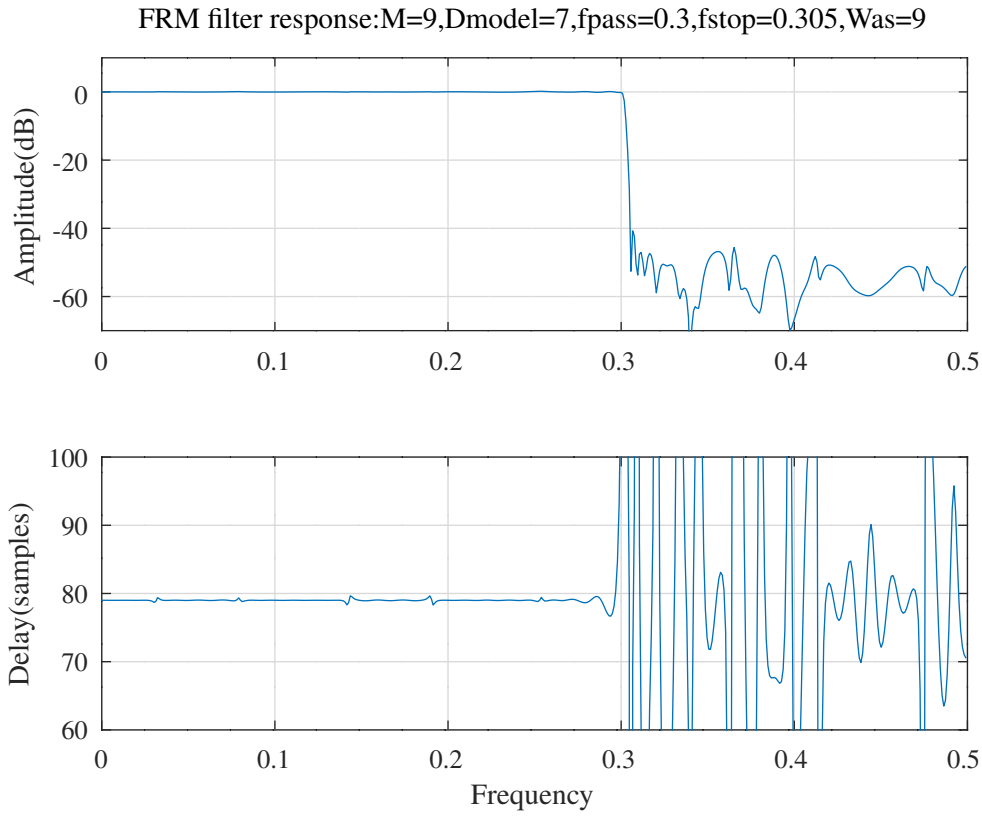



Figure 10.89: Second order cascade FRM filter response

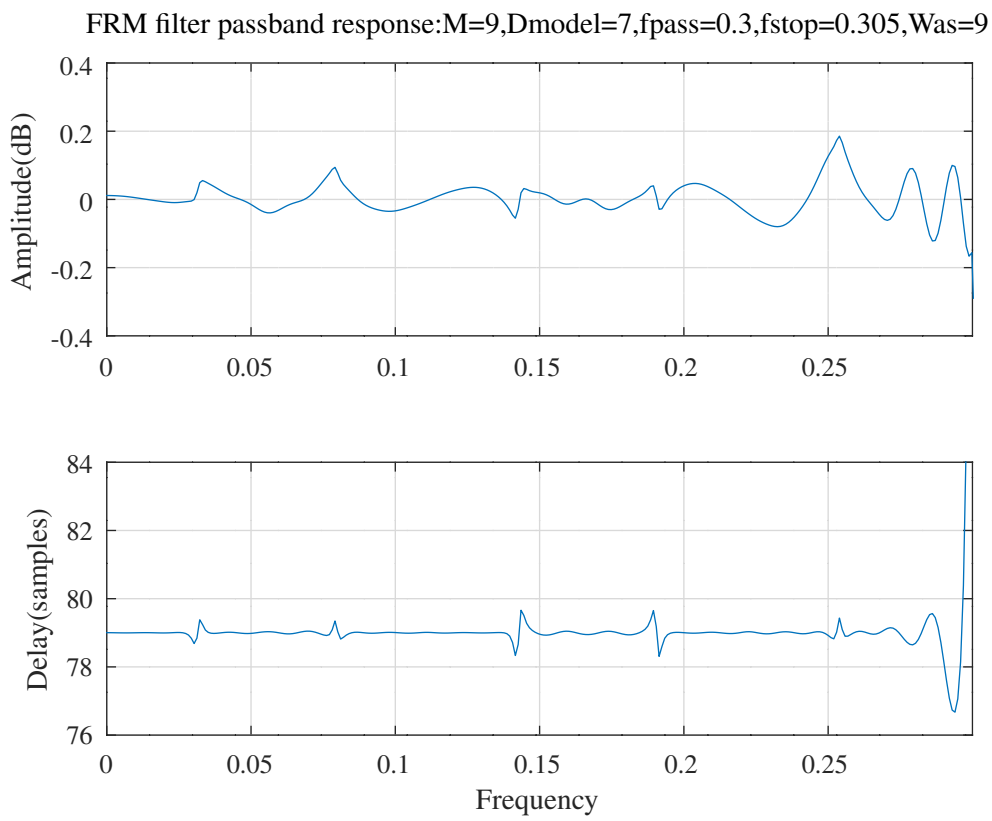



Figure 10.90: Second order cascade FRM filter passband response

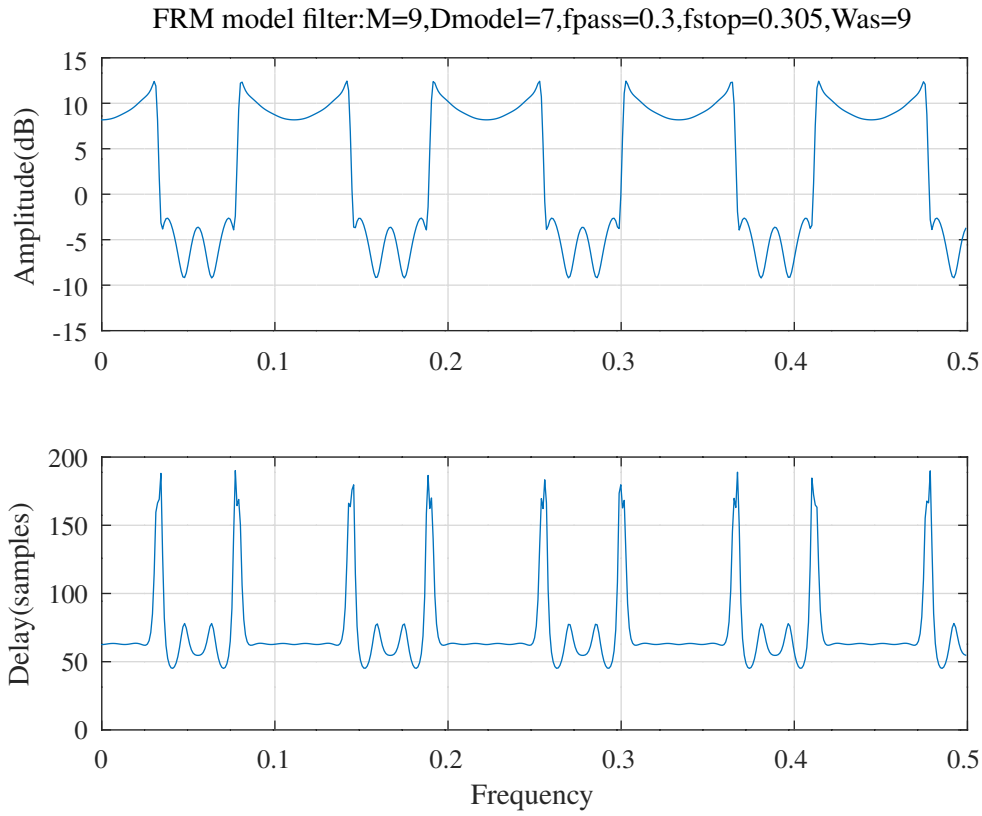



Figure 10.91: Second order cascade FRM model filter response

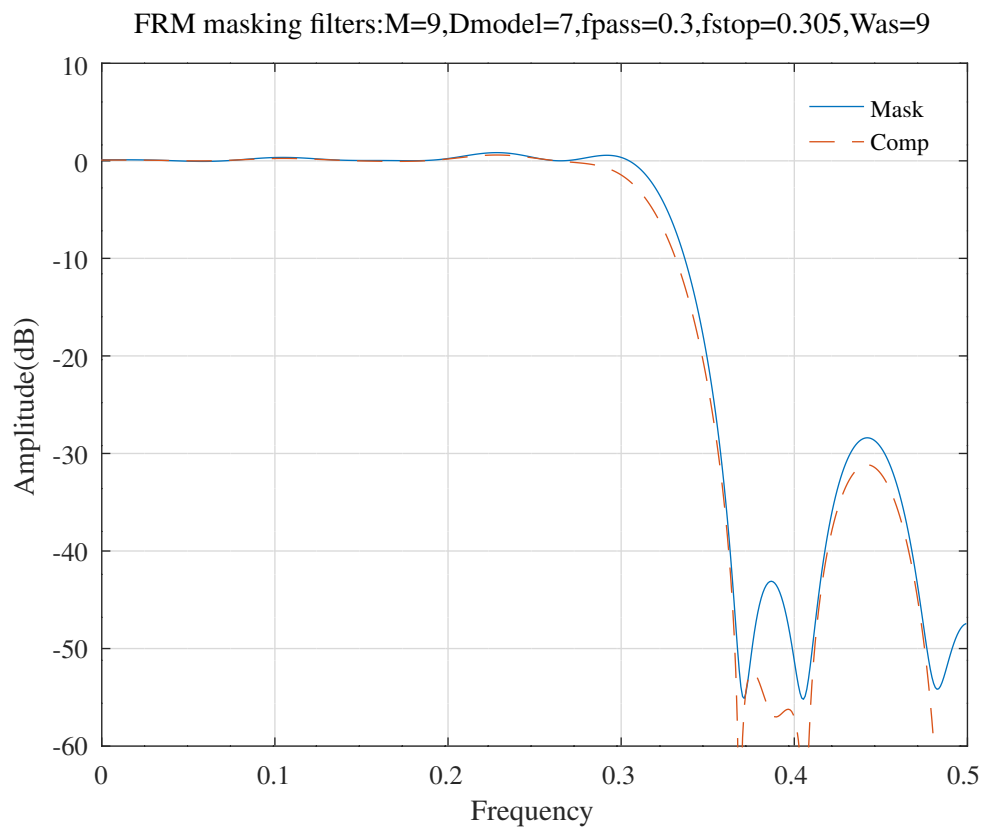



Figure 10.92: Second order cascade FRM masking filter responses

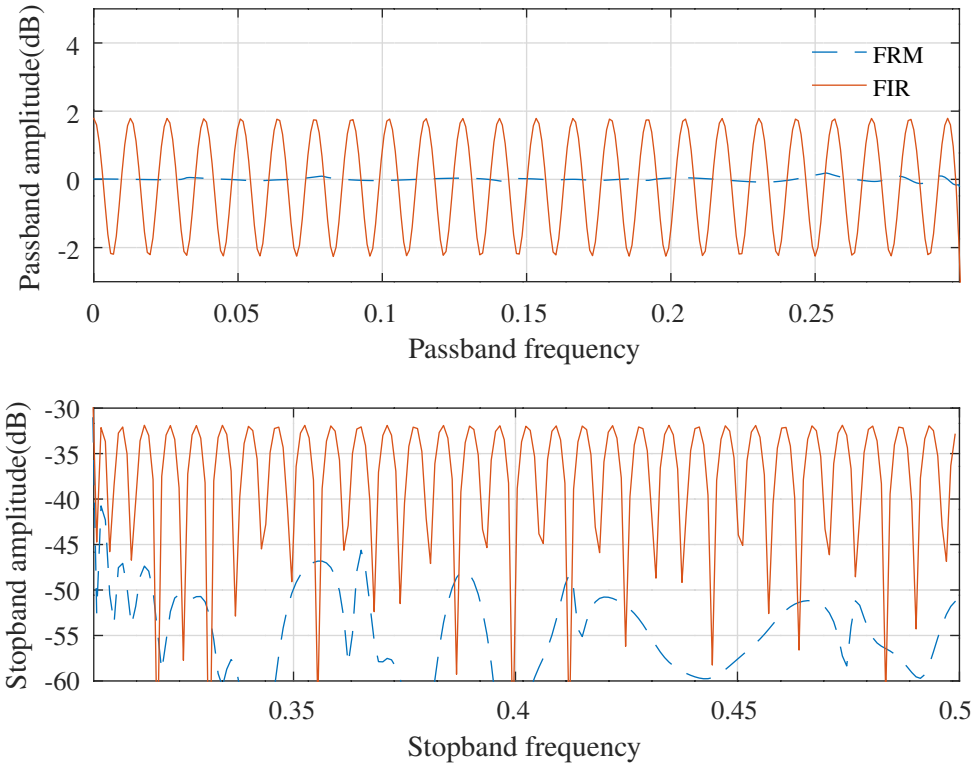



Figure 10.93: Comparison of the second order cascade FRM filter with a linear-phase FIR filter designed with *remez*

Figure 10.89 shows the overall response of the resulting FRM filter. Figure 10.90 shows the passband response of the FRM filter. Figure 10.91 shows the decimated response of the FRM model filter. Figure 10.92 shows the responses of the FRM masking filters. Figure 10.93 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]);
```

### 10.4.3 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 10.86 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 8.1.2.

Using the notation of Section 10.4.2, 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 $R(\omega)\, e^{\imath\phi_R(\omega)}$ is the frequency response of the IIR model filter then the *zero phase* response of the FRM filter is

$$H(\boldsymbol{x}, \omega) = A(\omega)\, R(M\omega)\, e^{\imath\phi_Z(M\omega)} + B(\omega)$$

where

$$\begin{aligned}
\phi_Z(\omega) &= D\omega + \phi_R(\omega) \\
A(\omega) &= \boldsymbol{a}_a^\top \boldsymbol{c}_a(\omega) - \boldsymbol{a}_c^\top \boldsymbol{c}_c(\omega) \\
B(\omega) &= \boldsymbol{a}_c^\top \boldsymbol{c}_c(\omega)
\end{aligned}$$

The squared-magnitude and phase responses of the zero phase response of the FRM filter are

$$\begin{aligned}
|H(\omega)|^2 &= A^2(\omega)\, R^2(M\omega) + B^2(\omega) + 2A(\omega)\, B(\omega)\, R(M\omega) \cos\phi_Z(M\omega) \\
\phi_H(\omega) &= \arctan \frac{A(\omega)\, R(M\omega) \sin\phi_Z(M\omega)}{A(\omega)\, R(M\omega) \cos\phi_Z(M\omega) + B(\omega)}
\end{aligned}$$

The group delay response, $T(\omega)$, of the zero phase response of the FRM filter (ie: the group delay error of the FRM filter, $-\frac{\partial\phi_H}{\partial\omega}$) is given by

$$\begin{aligned}
|H(\omega)|^2 T(\omega) = &- \left(A^2(\omega)\, R^2(M\omega) + A(\omega)\, B(\omega)\, R(M\omega) \cos\phi_Z(M\omega)\right) \frac{\partial\phi_Z(M\omega)}{\partial\omega} \ldots \\
&- A(\omega)\, B(\omega) \sin\phi_Z(M\omega) \frac{\partial R(M\omega)}{\partial\omega} \ldots \\
&+ R(M\omega) \sin\phi_Z(M\omega) \left(A(\omega) \frac{\partial B(\omega)}{\partial\omega} - B(\omega) \frac{\partial A(\omega)}{\partial\omega}\right)
\end{aligned}$$

where

$$\begin{aligned}
\frac{\partial\phi_Z(M\omega)}{\partial\omega} &= DM + \frac{\partial\phi_R(M\omega)}{\partial\omega} \\
\frac{\partial A(\omega)}{\partial\omega} &= \boldsymbol{a}_a^\top \boldsymbol{s}_a(\omega) - \boldsymbol{a}_c^\top \boldsymbol{s}_c(\omega) \\
\frac{\partial B(\omega)}{\partial\omega} &= \boldsymbol{a}_c^\top \boldsymbol{s}_c(\omega) \\
\boldsymbol{s}_a(\omega) &= \begin{cases} -2\left[\begin{array}{cccc} 0 & \sin\omega & \ldots & \frac{(n_a-1)}{2} \sin\frac{(n_a-1)}{2}\omega \end{array}\right]^\top, & \text{if } n_a \text{ odd} \\ -2\left[\begin{array}{ccc} \sin\frac{1}{2}\omega & \ldots & \frac{(n_a-1)}{2} \sin\frac{(n_a-1)}{2}\omega \end{array}\right]^\top, & \text{if } n_a \text{ even} \end{cases} \\
\boldsymbol{s}_c(\omega) &= \begin{cases} -2\left[\begin{array}{cccc} 0 & \sin\omega & \ldots & \frac{(n_c-1)}{2} \sin\frac{(n_c-1)}{2}\omega \end{array}\right]^\top, & \text{if } n_c \text{ odd} \\ -2\left[\begin{array}{ccc} \sin\frac{1}{2}\omega & \ldots & \frac{(n_c-1)}{2} \sin\frac{(n_c-1)}{2}\omega \end{array}\right]^\top, & \text{if } n_c \text{ even} \end{cases}
\end{aligned}$$

The gradients of $|H(\omega)|^2$ with respect to the coefficients are

$$\begin{aligned}
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{r}} &= 2\left(A^2(\omega)\, R(M\omega) + A(\omega)\, B(\omega) \cos\phi_Z(M\omega)\right) \frac{\partial R(M\omega)}{\partial \boldsymbol{r}} \ldots \\
&\quad - 2A(\omega)\, B(\omega)\, R(M\omega) \sin\phi_Z(M\omega) \frac{\partial\phi_R(M\omega)}{\partial \boldsymbol{r}} \\
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{a}} &= 2\left(A(\omega)\, R^2(M\omega) + B(\omega)\, R(M\omega) \cos\phi_Z(M\omega)\right) \frac{\partial A(\omega)}{\partial \boldsymbol{a}} \ldots
\end{aligned}$$

$$+2\left(B\left(\omega\right)+A\left(\omega\right)R\left(M\omega\right)\cos\phi_Z\left(M\omega\right)\right)\frac{\partial B\left(\omega\right)}{\partial \boldsymbol{a}}$$

where $\boldsymbol{r}$ represents the coefficients of the IIR model filter, and $\boldsymbol{a}$ represents the coefficients of both the masking filter, $\boldsymbol{a}_a$, and the complementary masking filter, $\boldsymbol{a}_c$.

The gradients of $T\left(\omega\right)$ with respect to the coefficients are given by

$$\begin{aligned}
&\frac{\partial\left|H\left(\omega\right)\right|^2}{\partial \boldsymbol{r}}T\left(\omega\right)+\left|H\left(\omega\right)\right|^2\frac{\partial T\left(\omega\right)}{\partial \boldsymbol{r}}=\ldots\\
&-\left(A^2\left(\omega\right)R^2\left(M\omega\right)+A\left(\omega\right)B\left(\omega\right)R\left(M\omega\right)\cos\phi_Z\left(M\omega\right)\right)\frac{\partial^2\phi_R\left(M\omega\right)}{\partial\omega\partial \boldsymbol{r}}\ldots\\
&-\left(2A^2\left(\omega\right)R\left(M\omega\right)+A\left(\omega\right)B\left(\omega\right)\cos\phi_Z\left(M\omega\right)\right)\frac{\partial\phi_Z\left(M\omega\right)}{\partial\omega}\frac{\partial R\left(M\omega\right)}{\partial \boldsymbol{r}}\ldots\\
&+A\left(\omega\right)B\left(\omega\right)R\left(M\omega\right)\sin\phi_Z\left(M\omega\right)\frac{\partial\phi_Z\left(M\omega\right)}{\partial\omega}\frac{\partial\phi_R\left(M\omega\right)}{\partial \boldsymbol{r}}\ldots\\
&-A\left(\omega\right)B\left(\omega\right)\cos\phi_Z\left(M\omega\right)\frac{\partial R\left(M\omega\right)}{\partial\omega}\frac{\partial\phi_R\left(M\omega\right)}{\partial \boldsymbol{r}}\ldots\\
&-A\left(\omega\right)B\left(\omega\right)\sin\phi_Z\left(M\omega\right)\frac{\partial^2 R\left(M\omega\right)}{\partial\omega\partial \boldsymbol{r}}\ldots\\
&+\sin\phi_Z\left(M\omega\right)\frac{\partial R\left(M\omega\right)}{\partial \boldsymbol{r}}\left(A\left(\omega\right)\frac{\partial B\left(\omega\right)}{\partial\omega}-B\left(\omega\right)\frac{\partial A\left(\omega\right)}{\partial\omega}\right)\ldots\\
&+R\left(M\omega\right)\cos\phi_Z\left(M\omega\right)\frac{\partial\phi_R\left(M\omega\right)}{\partial \boldsymbol{r}}\left(A\left(\omega\right)\frac{\partial B\left(\omega\right)}{\partial\omega}-B\left(\omega\right)\frac{\partial A\left(\omega\right)}{\partial\omega}\right)
\end{aligned}$$

and

$$\begin{aligned}
&\frac{\partial\left|H\left(\omega\right)\right|^2}{\partial \boldsymbol{a}}T\left(\omega\right)+\left|H\left(\omega\right)\right|^2\frac{\partial T\left(\omega\right)}{\partial \boldsymbol{a}}=\ldots\\
&-\left(2A\left(\omega\right)R^2\left(M\omega\right)+B\left(\omega\right)R\left(M\omega\right)\cos\phi_Z\left(M\omega\right)\right)\frac{\partial\phi_Z\left(M\omega\right)}{\partial\omega}\frac{\partial A\left(\omega\right)}{\partial \boldsymbol{a}}\ldots\\
&-A\left(\omega\right)R\left(M\omega\right)\cos\phi_Z\left(M\omega\right)\frac{\partial\phi_Z\left(M\omega\right)}{\partial\omega}\frac{\partial B\left(\omega\right)}{\partial \boldsymbol{a}}\ldots\\
&-\sin\phi_Z\left(M\omega\right)\frac{\partial R\left(M\omega\right)}{\partial\omega}\left(A\left(\omega\right)\frac{\partial B\left(\omega\right)}{\partial \boldsymbol{a}}+B\left(\omega\right)\frac{\partial A\left(\omega\right)}{\partial \boldsymbol{a}}\right)\ldots\\
&-R\left(M\omega\right)\sin\phi_Z\left(M\omega\right)\left(\frac{\partial A\left(\omega\right)}{\partial\omega}\frac{\partial B\left(\omega\right)}{\partial \boldsymbol{a}}-\frac{\partial B\left(\omega\right)}{\partial\omega}\frac{\partial A\left(\omega\right)}{\partial \boldsymbol{a}}\right)\ldots\\
&+R\left(M\omega\right)\sin\phi_Z\left(M\omega\right)\left(A\left(\omega\right)\frac{\partial^2 B\left(\omega\right)}{\partial\omega\partial \boldsymbol{a}}-B\left(\omega\right)\frac{\partial^2 A\left(\omega\right)}{\partial\omega\partial \boldsymbol{a}}\right)
\end{aligned}$$

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 H 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. To avoid numerical problems, *iir_frm_socp_mmse.m* sets the *SeDuMi* parameter *pars.eps* to $1e-6$ rather than the default $1e-8$. 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=800 % Frequency points across the band
tol=0.001 % Tolerance on relative coefficient update size
ctol=5e-06 % Tolerance on constraints
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=0.2 % Pass band amplitude peak-to-peak ripple
```

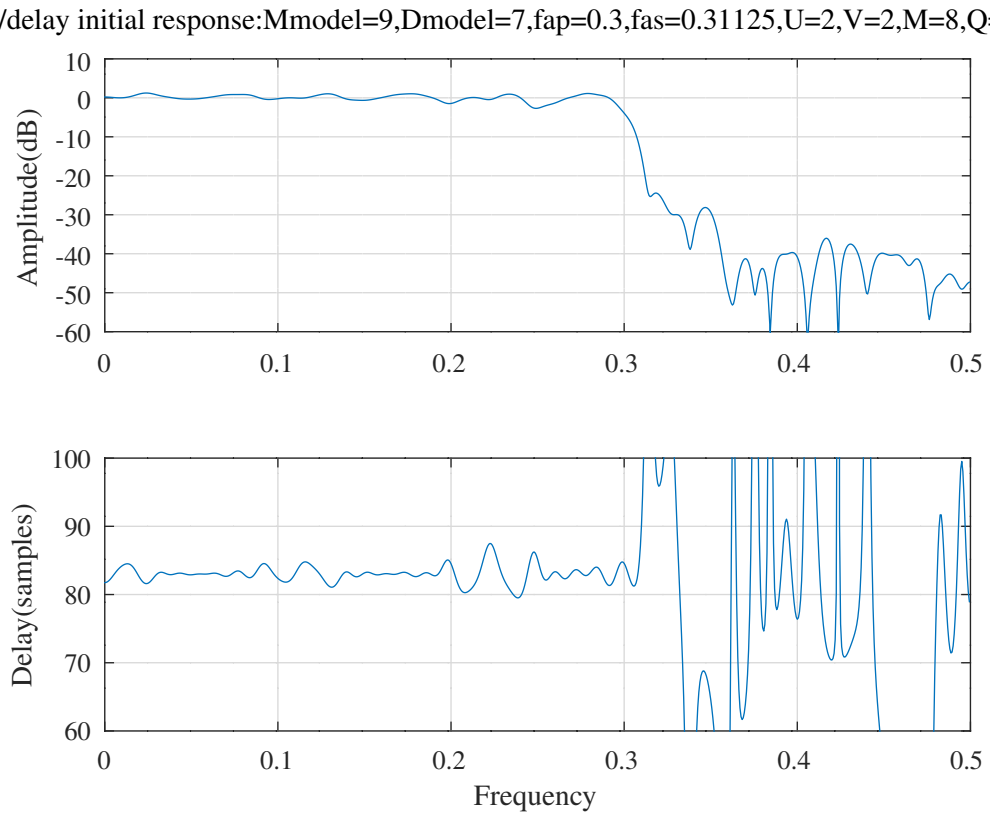



Figure 10.94: FRM gain-pole-zero format IIR model filter with WISE optimisation : initial response

```
Wap=1 % Pass band weight
tpr=1 % Pass band delay peak-to-peak ripple
Wtp=0.001 % Pass band delay weight
Wat=1e-06 % 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=20 % Stop band weight
```

The passband edge frequency is the same as that of the design example given by Lu and Hinamoto [212, 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_frm_iir_test.m* with the WISE method of *Tarczynski et al.* (as shown in Section 8.1.5). Figure 10.94 shows the overall response of the initial FRM filter. After SOCP and PCLS optimisation of the initial response the resulting IIR model filter numerator polynomial is

```
a = [   0.0023700678,  -0.0003397829,  -0.0029417522,  -0.0006355842, ...
        0.0075008764,   0.0007510600,  -0.0156068584,   0.0326937325, ...
       -0.0274110300,   0.0082303676,   0.0054733216 ]';
```

and the IIR model filter denominator polynomial is

```
d = [   1.0000000000,   0.3924339481,   0.7228430286,   0.2906406793, ...
        0.0561994661,  -0.0054228378,  -0.0040699815,  -0.0008255637, ...
       -0.0001066888,  -0.0000112448,  -0.0000007043 ]';
```

The FIR masking filter polynomial is

```
aa = [   0.1393928261,   0.1674620489,  -0.4627033797,   0.3065750259, ...
         0.1312125766,  -0.1983178293,   0.0989691556,   0.2219864884, ...
        -0.2561893787,  -0.0574137450,   0.4176795004,  -0.2832805339, ...
        -0.2370691645,   0.4110746262,   0.0660054646,  -0.4964330163, ...
```

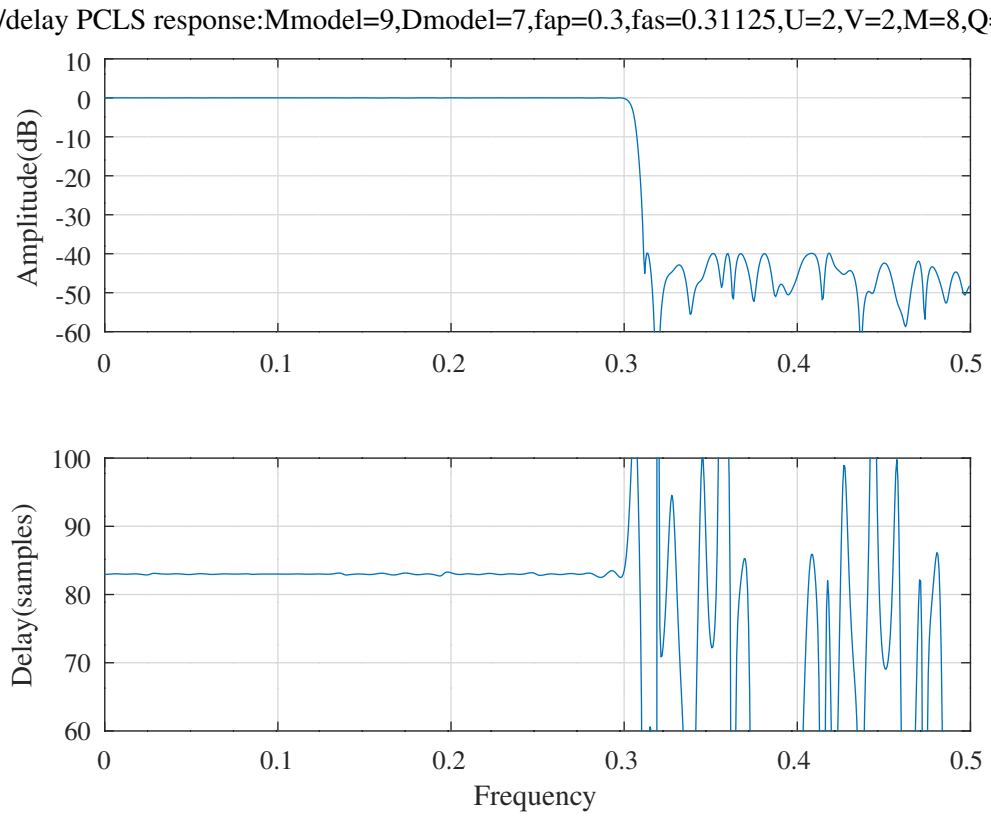



Figure 10.95: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation:overall response

```
 0.3012899387,   0.3717917805,  -0.3296978271,   0.5699938505, ...
 0.2924262915,   0.5699938505,  -0.3296978271,   0.3717917805, ...
 0.3012899387,  -0.4964330163,   0.0660054646,   0.4110746262, ...
-0.2370691645,  -0.2832805339,   0.4176795004,  -0.0574137450, ...
-0.2561893787,   0.2219864884,   0.0989691556,  -0.1983178293, ...
 0.1312125766,   0.3065750259,  -0.4627033797,   0.1674620489, ...
 0.1393928261 ]';
```

The complementary FIR masking filter polynomial is

```
ac = [  -0.0030412571,  -0.0079564018,   0.0153736627,  -0.0066969934, ...
        -0.0080778214,   0.0051399782,   0.0030547944,  -0.0100664723, ...
         0.0014355515,   0.0219452142,   0.0003206176,  -0.0389680458, ...
         0.0380030437,   0.0138465796,  -0.0449024201,   0.0183658387, ...
         0.0617272289,  -0.0747188237,  -0.0835647165,   0.2923059192, ...
         0.6085792457,   0.2923059192,  -0.0835647165,  -0.0747188237, ...
         0.0617272289,   0.0183658387,  -0.0449024201,   0.0138465796, ...
         0.0380030437,  -0.0389680458,   0.0003206176,   0.0219452142, ...
         0.0014355515,  -0.0100664723,   0.0030547944,   0.0051399782, ...
        -0.0080778214,  -0.0066969934,   0.0153736627,  -0.0079564018, ...
        -0.0030412571 ]';
```

Figure 10.95 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 10.96 shows the passband response of the resulting FRM filter. Figure 10.97 shows the responses of the resulting FRM masking filters. Figure 10.98 shows the decimated response of the resulting FRM model filter.

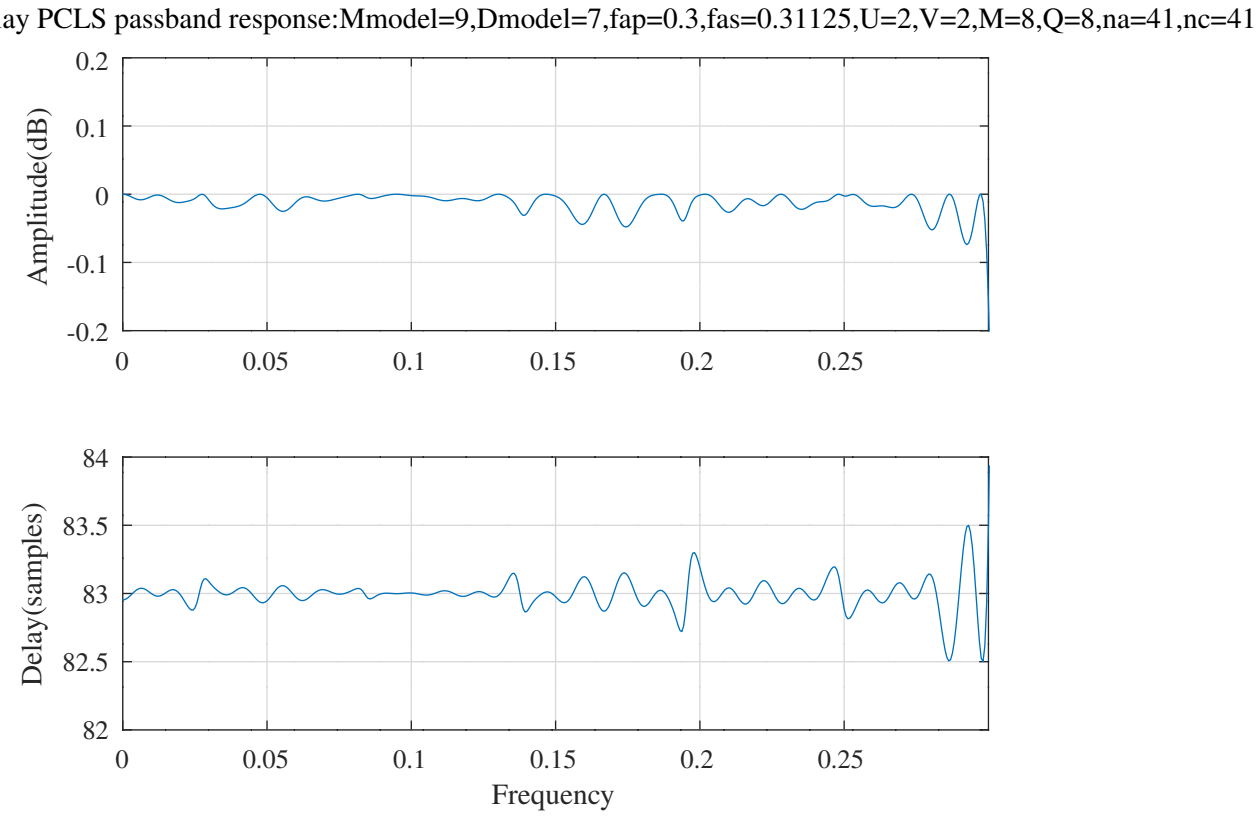



Figure 10.96: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation : passband response

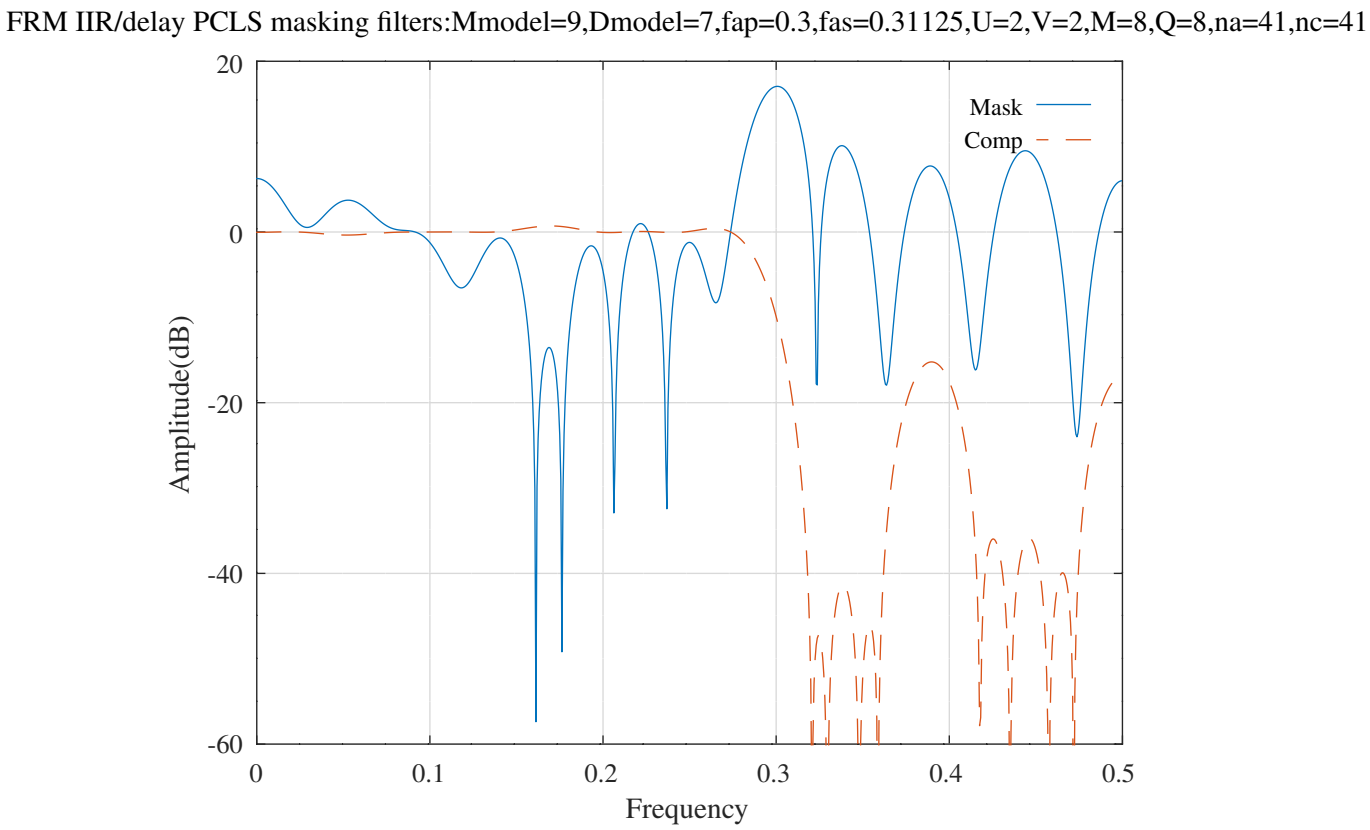



Figure 10.97: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation:masking filter responses

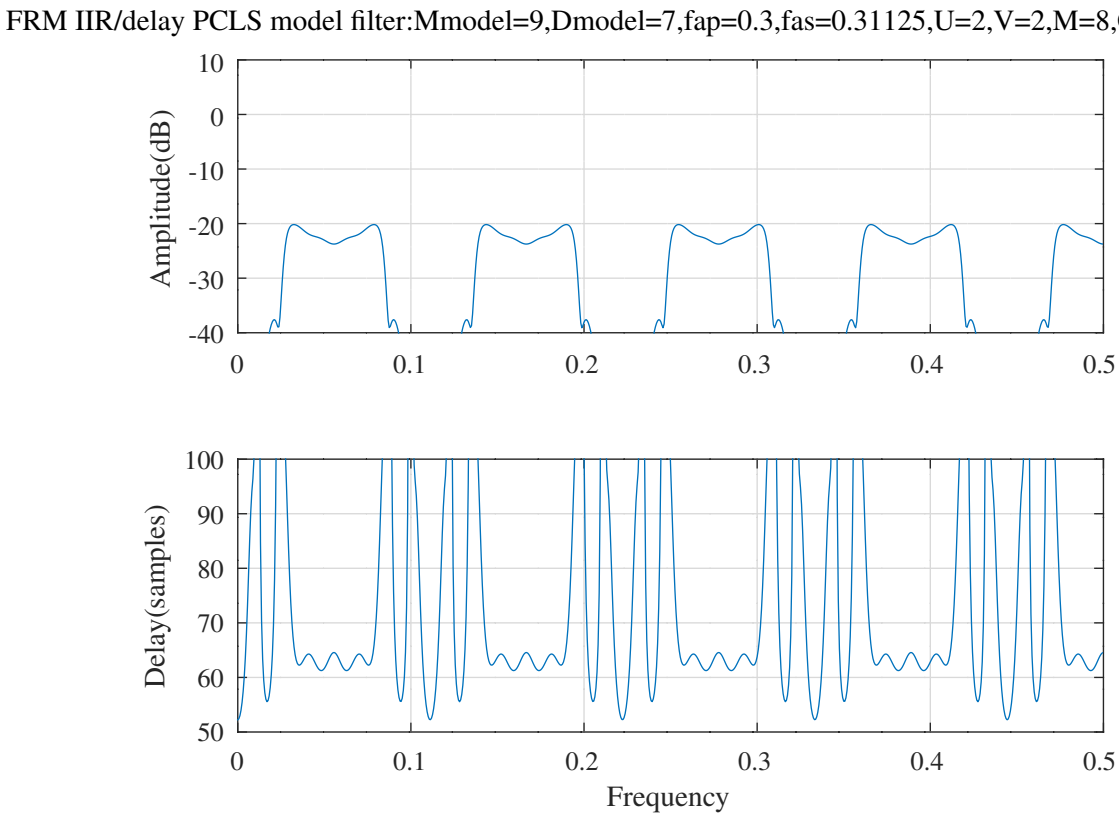



Figure 10.98: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation:model filter response

### 10.4.4 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 10.87, 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 8.1.2.

Using the notation of Section 10.4.2, 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

$$H(\boldsymbol{x}, \omega) = e^{\imath[DM\omega + \phi_R(M\omega)]} A(\omega) + B(\omega)$$

where $\phi_R(\omega)$ is the phase response of the allpass branch of the model filter and:

$$\begin{aligned} A(\omega) &= \frac{\boldsymbol{a}_a^\top \boldsymbol{c}_a(\omega) + \boldsymbol{a}_c^\top \boldsymbol{c}_c(\omega)}{2} \\ B(\omega) &= \frac{\boldsymbol{a}_a^\top \boldsymbol{c}_a(\omega) - \boldsymbol{a}_c^\top \boldsymbol{c}_c(\omega)}{2} \end{aligned}$$

The squared-magnitude and phase responses of the zero phase response of the FRM filter are:

$$\begin{aligned} |H(\omega)|^2 &= A^2(\omega) + B^2(\omega) + 2A(\omega)B(\omega)\cos\phi_Z(M\omega) \\ \phi_H(\omega) &= \arctan\frac{A(\omega)\sin\phi_Z(M\omega)}{A(\omega)\cos\phi_Z(M\omega) + B(\omega)} \end{aligned}$$

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:

$$\begin{aligned} |H(\omega)|^2 T(\omega) = &-\left(A^2(\omega) + A(\omega)B(\omega)\cos\phi_Z(M\omega)\right)\frac{\partial\phi_Z(M\omega)}{\partial\omega}\ldots \\ &- B(\omega)\sin\phi_Z(M\omega)\frac{\partial A(\omega)}{\partial\omega}\ldots \\ &+ A(\omega)\sin\phi_Z(M\omega)\frac{\partial B(\omega)}{\partial\omega} \end{aligned}$$

where:

$$\begin{aligned} \frac{\partial\phi_Z(M\omega)}{\partial\omega} &= DM + \frac{\partial\phi_R(M\omega)}{\partial\omega} \\ \frac{\partial A(\omega)}{\partial\omega} &= \frac{\boldsymbol{a}_a^\top \boldsymbol{s}_a(\omega) + \boldsymbol{a}_c^\top \boldsymbol{s}_c(\omega)}{2} \\ \frac{\partial B(\omega)}{\partial\omega} &= \frac{\boldsymbol{a}_a^\top \boldsymbol{s}_a(\omega) - \boldsymbol{a}_c^\top \boldsymbol{s}_c(\omega)}{2} \\ \boldsymbol{s}_a(\omega) &= -2\begin{bmatrix} 0 & \sin\omega & \ldots & \frac{(n_a-1)}{2}\sin\frac{(n_a-1)}{2}\omega \end{bmatrix}^\top \\ \boldsymbol{s}_c(\omega) &= -2\begin{bmatrix} 0 & \sin\omega & \ldots & \frac{(n_c-1)}{2}\sin\frac{(n_c-1)}{2}\omega \end{bmatrix}^\top \end{aligned}$$

The gradients of $|H(\omega)|^2$ with respect to the coefficients are:

$$\begin{aligned} \frac{\partial|H(\omega)|^2}{\partial\boldsymbol{r}} &= -2A(\omega)B(\omega)\sin\phi_Z(M\omega)\frac{\partial\phi_R(M\omega)}{\partial\boldsymbol{r}} \\ \frac{\partial|H(\omega)|^2}{\partial\boldsymbol{a}} &= 2\left(A(\omega) + B(\omega)\cos\phi_Z(M\omega)\right)\frac{\partial A(\omega)}{\partial\boldsymbol{a}} + 2\left(B(\omega) + A(\omega)\cos\phi_Z(M\omega)\right)\frac{\partial B(\omega)}{\partial\boldsymbol{a}} \end{aligned}$$

where $\boldsymbol{r}$ represents the coefficients of the allpass model filter, and $\boldsymbol{a}$ represents the coefficients of both the masking filter, $\boldsymbol{a}_a$, and the complementary masking filter, $\boldsymbol{a}_c$.

The gradients of $\phi_H(\omega)$ with respect to the coefficients are:

$$|H(\omega)|^2\frac{\partial\phi_H(\omega)}{\partial\boldsymbol{r}} = A(\omega)\left[A(\omega) + B(\omega)\cos\phi_Z(M\omega)\right]\frac{\partial\phi_R(M\omega)}{\partial\boldsymbol{r}}$$

$$|H(\omega)|^2 \frac{\partial \phi_H(\omega)}{\partial \boldsymbol{a}} = \sin\phi_Z(M\omega)\left[B(\omega)\frac{\partial A(\omega)}{\partial \boldsymbol{a}} - A(\omega)\frac{\partial B(\omega)}{\partial \boldsymbol{a}}\right]$$

The gradients of $T(\omega)$ with respect to the coefficients are given by:

$$\begin{aligned}
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{r}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \boldsymbol{r}} = &- \left(A^2(\omega) + A(\omega)B(\omega)\cos\phi_Z(M\omega)\right)\frac{\partial^2 \phi_Z(M\omega)}{\partial \boldsymbol{r}\partial\omega}\ldots \\
&+ A(\omega)B(\omega)\sin\phi_Z(M\omega)\frac{\partial \phi_R(M\omega)}{\partial \boldsymbol{r}}\frac{\partial \phi_Z(M\omega)}{\partial\omega}\ldots \\
&+ A(\omega)\cos\phi_Z(M\omega)\frac{\partial \phi_R(M\omega)}{\partial \boldsymbol{r}}\frac{\partial B(\omega)}{\partial\omega}\ldots \\
&- B(\omega)\cos\phi_Z(M\omega)\frac{\partial \phi_R(M\omega)}{\partial \boldsymbol{r}}\frac{\partial A(\omega)}{\partial\omega} \\
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{a}} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial \boldsymbol{a}} = &- \left(2A(\omega) + B(\omega)\cos\phi_Z(M\omega)\right)\frac{\partial A(\omega)}{\partial \boldsymbol{a}}\frac{\partial \phi_Z(M\omega)}{\partial\omega}\ldots \\
&- A(\omega)\cos\phi_Z(M\omega)\frac{\partial B(\omega)}{\partial \boldsymbol{a}}\frac{\partial \phi_Z(M\omega)}{\partial\omega}\ldots \\
&+ \sin\phi_Z(M\omega)\frac{\partial A(\omega)}{\partial \boldsymbol{a}}\frac{\partial B(\omega)}{\partial\omega} - \sin\phi_Z(M\omega)\frac{\partial B(\omega)}{\partial \boldsymbol{a}}\frac{\partial A(\omega)}{\partial\omega}\ldots \\
&+ A(\omega)\sin\phi_Z(M\omega)\frac{\partial^2 B(\omega)}{\partial \boldsymbol{a}\partial\omega} - B(\omega)\sin\phi_Z(M\omega)\frac{\partial^2 A(\omega)}{\partial \boldsymbol{a}\partial\omega}
\end{aligned}$$

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
ctol=2e-05 % Tolerance on constraints
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=0.02 % Pass band delay weight
fas=0.3105 % Stop band edge
dBas=40 % Stop band attenuation ripple
Was=20 % Stop band weight
rho=0.968750 % Constraint on allpass pole radius
```

The passband edge frequency is are the same as for the design example given by Lu and Hinamoto [212, 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 8.1.5. Figure 10.99 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,  -0.0189470249,   0.4865123105,   0.0190978169, ...
       -0.1093824106,   0.0009756279,   0.0402398870,   0.0125728273, ...
       -0.0251169616,  -0.0020192504,   0.0034131821 ]';
```

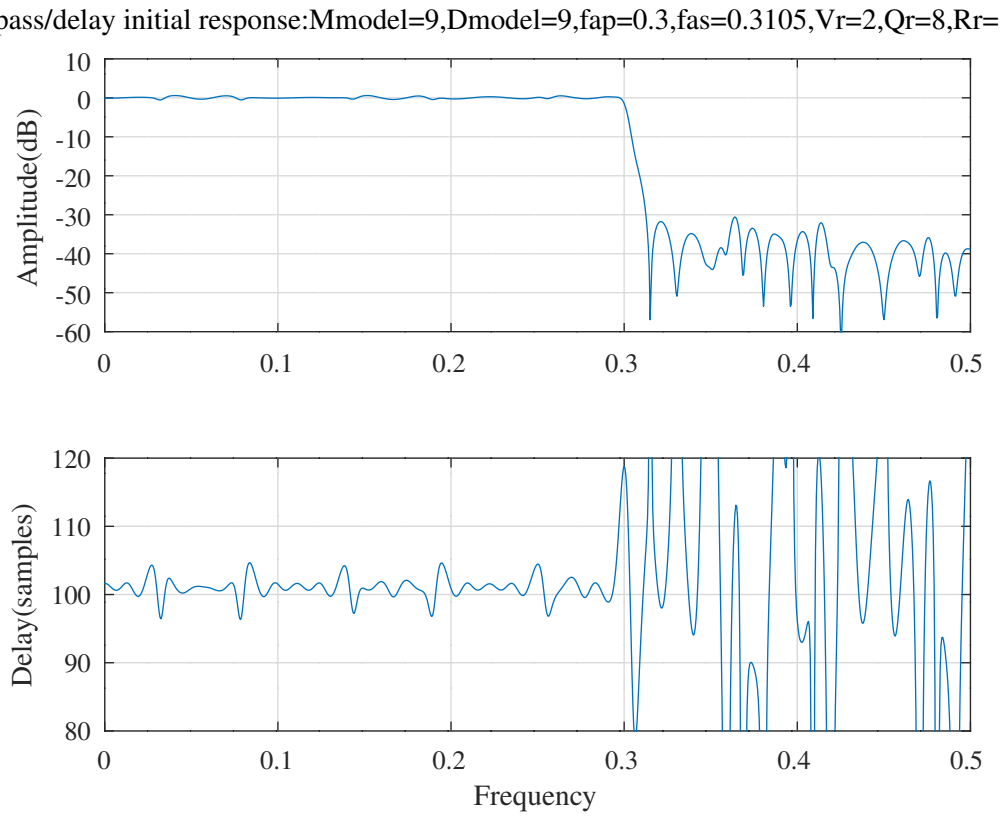



Figure 10.99: FRM gain-pole-zero format allpass model filter with WISE optimisation : initial response

The FIR masking filter polynomial is

```
aa = [  -0.0016821494,  -0.0021239529,   0.0093768963,  -0.0108094884, ...
         0.0013830322,   0.0078746430,  -0.0046354371,  -0.0082329000, ...
         0.0096799602,   0.0088349492,  -0.0098165660,  -0.0201947172, ...
         0.0418016558,  -0.0097556981,  -0.0425037024,   0.0338403070, ...
         0.0504424783,  -0.0846729482,  -0.0577420324,   0.2978996281, ...
         0.5809995029,   0.2978996281,  -0.0577420324,  -0.0846729482, ...
         0.0504424783,   0.0338403070,  -0.0425037024,  -0.0097556981, ...
         0.0418016558,  -0.0201947172,  -0.0098165660,   0.0088349492, ...
         0.0096799602,  -0.0082329000,  -0.0046354371,   0.0078746430, ...
         0.0013830322,  -0.0108094884,   0.0093768963,  -0.0021239529, ...
        -0.0016821494 ]';
```

The complementary FIR masking filter polynomial is

```
ac = [   0.0012366160,   0.0001341499,  -0.0041863689,   0.0085669101, ...
        -0.0080847066,   0.0013665249,   0.0081087355,  -0.0104873788, ...
         0.0021823269,   0.0120414340,  -0.0121849857,  -0.0076371977, ...
         0.0341302026,  -0.0373887400,   0.0035521550,   0.0489376070, ...
        -0.0653912709,   0.0039084335,   0.1302476977,  -0.2662619338, ...
        -0.6753011720,  -0.2662619338,   0.1302476977,   0.0039084335, ...
        -0.0653912709,   0.0489376070,   0.0035521550,  -0.0373887400, ...
         0.0341302026,  -0.0076371977,  -0.0121849857,   0.0120414340, ...
         0.0021823269,  -0.0104873788,   0.0081087355,   0.0013665249, ...
        -0.0080847066,   0.0085669101,  -0.0041863689,   0.0001341499, ...
         0.0012366160 ]';
```

Figure 10.100 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 10.101 shows the passband response of the resulting FRM filter. Figure 10.102 shows the responses of the resulting FRM masking filters. Figure 10.103 shows the response of the resulting FRM model filter.

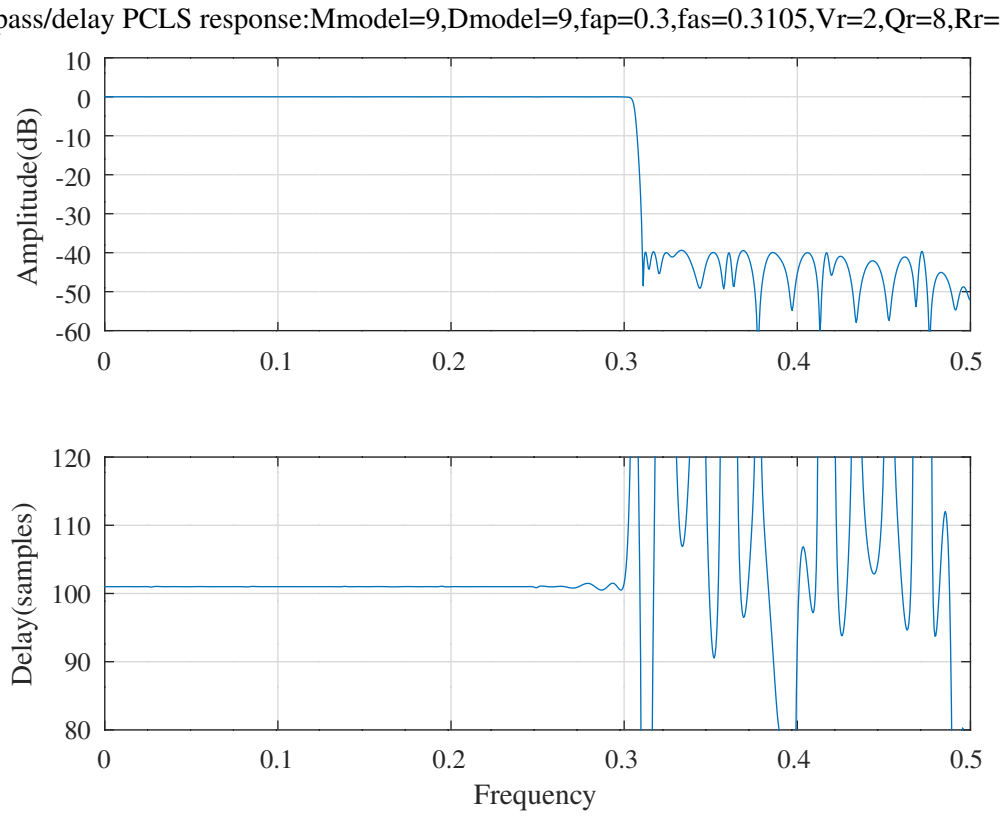



Figure 10.100: FRM gain-pole-zero format allpass model filter with SOCP and PCLS optimisation:overall response

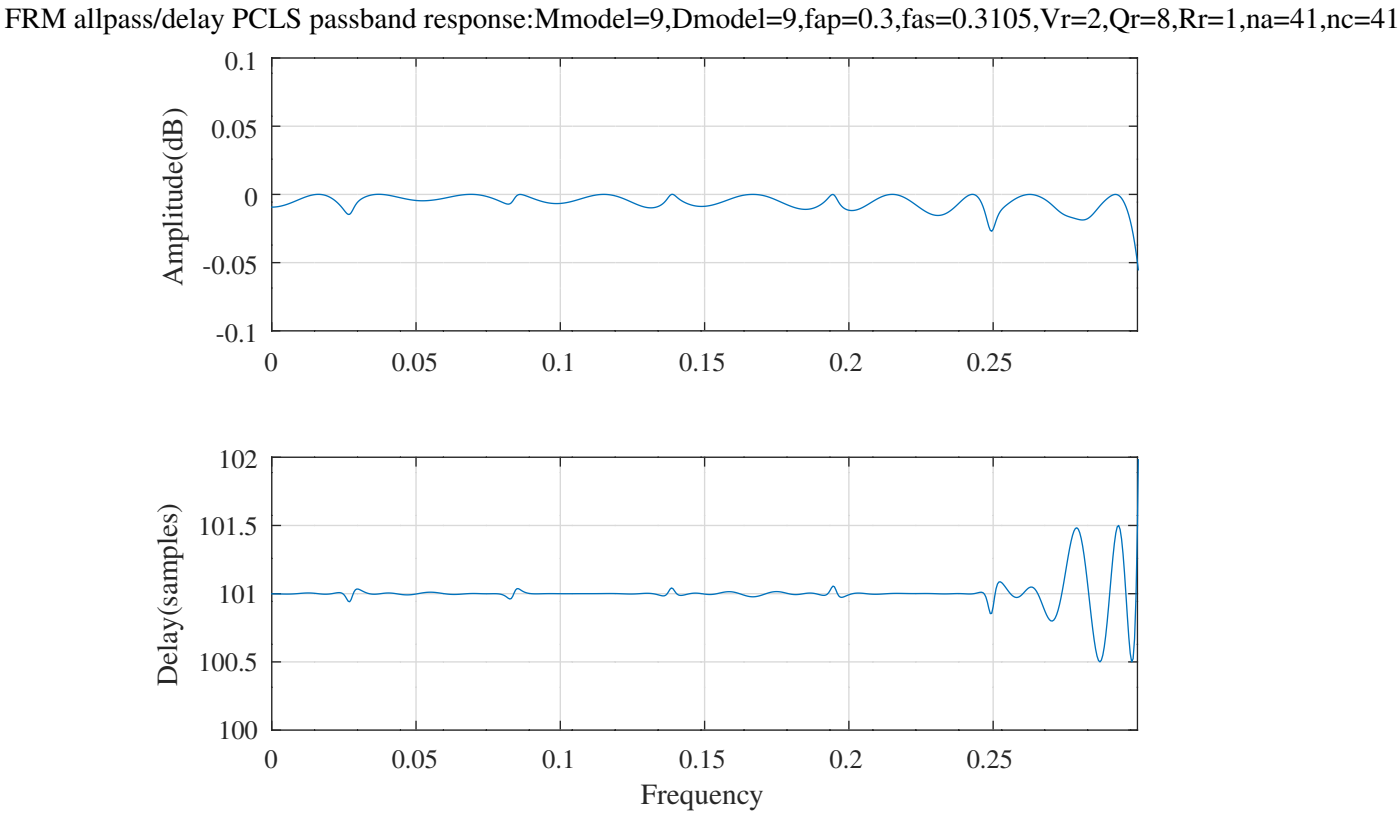



Figure 10.101: FRM gain-pole-zero format allpass model filter : passband response with SOCP and PCLS optimisation

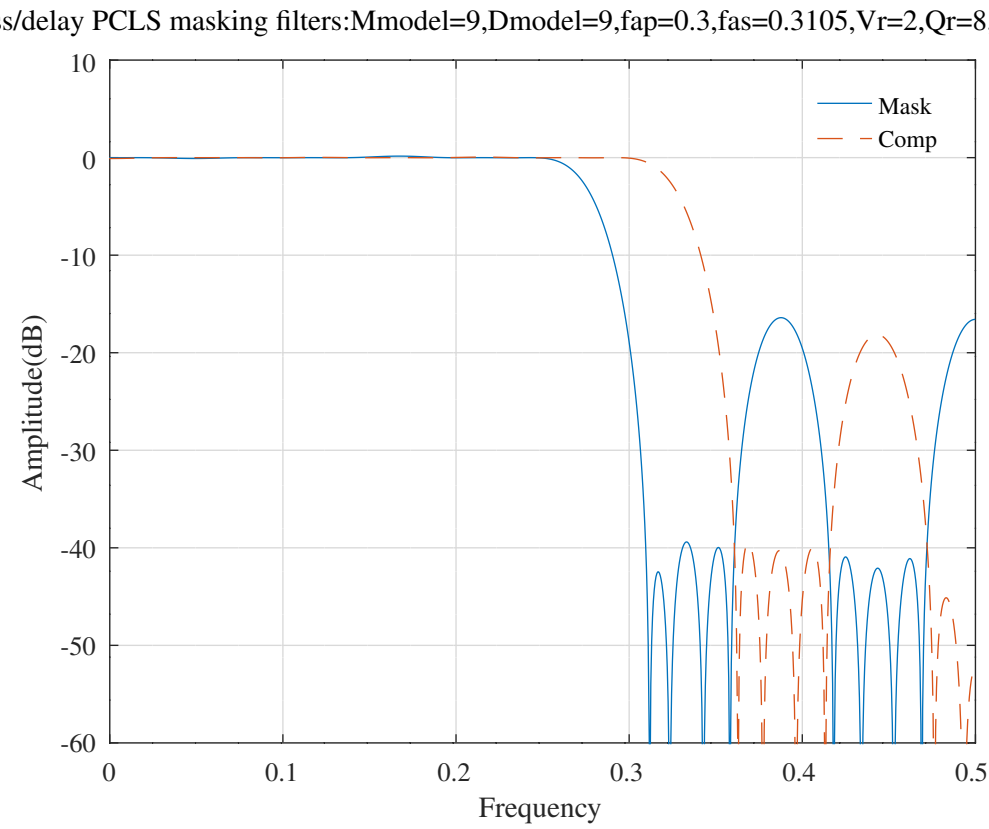



Figure 10.102: FRM gain-pole-zero format allpass model filter with SOCP and PCLS optimisation:masking filter responses

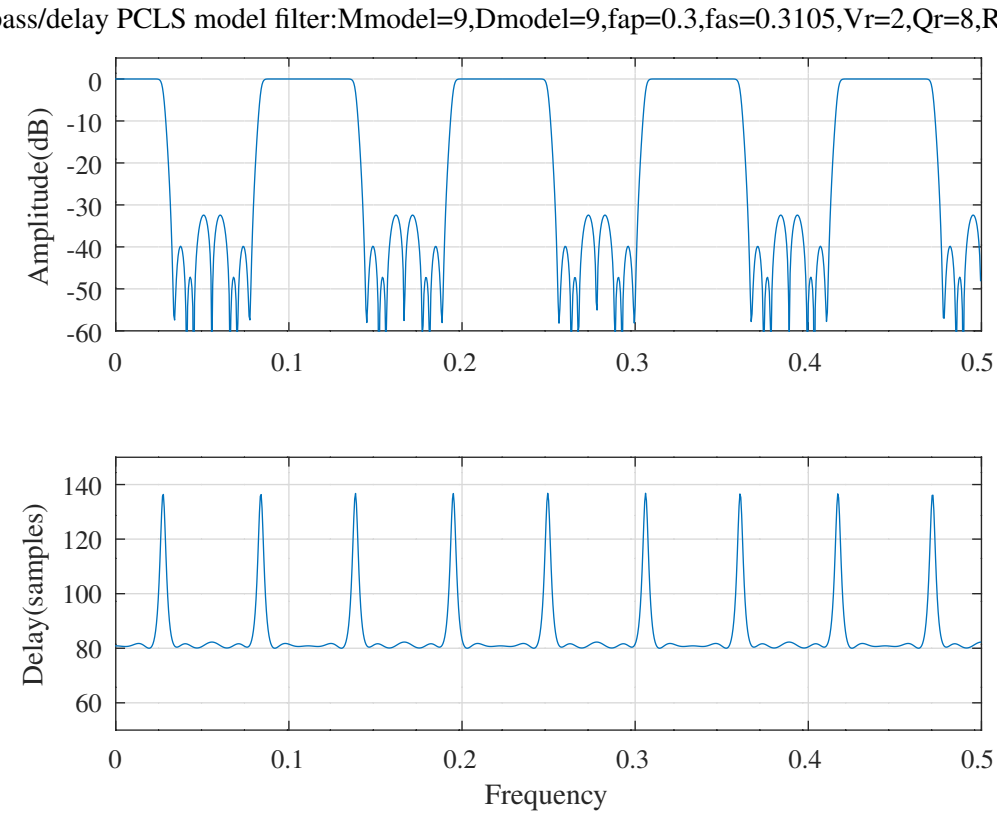



Figure 10.103: FRM gain-pole-zero format allpass model filter with SOCP and PCLS optimisation:model filter response

### 10.4.5 Design of an FRM digital filter with an IIR model filter consisting of parallel allpass filters

Figure 10.87 shows an FRM filter with a model filter consisting of parallel allpass filters. The filter transfer function is

$$H(z) = \frac{1}{2}\left[R\left(z^M\right) + S\left(z^M\right)\right] F_{M_a}(z) + \frac{1}{2}\left[R\left(z^M\right) - S\left(z^M\right)\right] F_{M_c}(z)$$

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

$$H(z) = R\left(z^M\right) A(z) + S\left(z^M\right) B(z)$$

where

$$A(z) = \frac{1}{2}\left[F_{M_a}(z) + F_{M_c}(z)\right] = \sum_{k=0}^{n_m-1} a_k z^{-k}$$
$$B(z) = \frac{1}{2}\left[F_{M_a}(z) - F_{M_c}(z)\right] = \sum_{k=0}^{n_m-1} b_k z^{-k}$$

$n_m$ 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

$$\begin{aligned} H(\omega) &= e^{\imath\phi_R(M\omega)} \sum_{k=0}^{n_m-1} a_k e^{-\imath k\omega} + e^{\imath\phi_S(M\omega)} \sum_{k=0}^{n_m-1} b_k e^{-\imath k\omega} \\ &= \boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b} - \imath\left(\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b}\right) \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{a} &= \left[a_0 \ldots a_{n_m-1}\right]^\top \\ \boldsymbol{b} &= \left[b_0 \ldots b_{n_m-1}\right]^\top \\ \boldsymbol{v} &= \left[0 \ldots n_m - 1\right] \\ \boldsymbol{c}_R &= \cos\left[\boldsymbol{v}\omega - \phi_R(M\omega)\right] \\ \boldsymbol{c}_S &= \cos\left[\boldsymbol{v}\omega - \phi_S(M\omega)\right] \\ \boldsymbol{s}_R &= \sin\left[\boldsymbol{v}\omega - \phi_R(M\omega)\right] \\ \boldsymbol{s}_S &= \sin\left[\boldsymbol{v}\omega - \phi_S(M\omega)\right] \end{aligned}$$

The squared-amplitude and phase responses of $H(z)$ are

$$|H(\omega)|^2 = \left(\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b}\right)^2 + \left(\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b}\right)^2$$
$$\phi_H(\omega) = -\arctan\frac{\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b}}{\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b}}$$

The group delay response is given by

$$|H(\omega)|^2 T(\omega) = \left(\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b}\right)\left(\frac{\partial\boldsymbol{s}_R}{\partial\omega}\boldsymbol{a} + \frac{\partial\boldsymbol{s}_S}{\partial\omega}\boldsymbol{b}\right) - \left(\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b}\right)\left(\frac{\partial\boldsymbol{c}_R}{\partial\omega}\boldsymbol{a} + \frac{\partial\boldsymbol{c}_S}{\partial\omega}\boldsymbol{b}\right)$$

where

$$\frac{\partial\boldsymbol{c}_R}{\partial\omega} = -\left[\boldsymbol{v} - \frac{\partial\phi_R(M\omega)}{\partial\omega}\right] \circ \boldsymbol{s}_R$$

$$
\begin{aligned}
\frac{\partial \boldsymbol{c}_S}{\partial \omega} &= -\left[\boldsymbol{v} - \frac{\partial \phi_S(M\omega)}{\partial \omega}\right] \circ \boldsymbol{s}_S \\
\frac{\partial \boldsymbol{s}_R}{\partial \omega} &= \left[\boldsymbol{v} - \frac{\partial \phi_R(M\omega)}{\partial \omega}\right] \circ \boldsymbol{c}_R \\
\frac{\partial \boldsymbol{s}_S}{\partial \omega} &= \left[\boldsymbol{v} - \frac{\partial \phi_S(M\omega)}{\partial \omega}\right] \circ \boldsymbol{c}_S
\end{aligned}
$$

The $\circ$ symbol represents the *Hadamard* or element-wise product.

The gradients of the squared amplitude response are

$$
\begin{aligned}
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{r}} &= 2(\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b})\frac{\partial \boldsymbol{c}_R}{\partial \boldsymbol{r}}\boldsymbol{a} + 2(\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b})\frac{\partial \boldsymbol{s}_R}{\partial \boldsymbol{r}}\boldsymbol{a} \\
&= 2((\boldsymbol{c}_S\boldsymbol{b})(\boldsymbol{s}_R\boldsymbol{a}) - (\boldsymbol{s}_S\boldsymbol{b})(\boldsymbol{c}_R\boldsymbol{a}))\frac{\partial \phi_R(M\omega)}{\partial \boldsymbol{r}} \\
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{s}} &= 2(\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b})\frac{\partial \boldsymbol{c}_S}{\partial \boldsymbol{s}}\boldsymbol{b} + 2(\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b})\frac{\partial \boldsymbol{s}_S}{\partial \boldsymbol{s}}\boldsymbol{b} \\
&= -2((\boldsymbol{c}_S\boldsymbol{b})(\boldsymbol{s}_R\boldsymbol{a}) - (\boldsymbol{s}_S\boldsymbol{b})(\boldsymbol{c}_R\boldsymbol{a}))\frac{\partial \phi_S(M\omega)}{\partial \boldsymbol{s}} \\
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{a}} &= 2(\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b})\boldsymbol{c}_R + 2(\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b})\boldsymbol{s}_R \\
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{b}} &= 2(\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b})\boldsymbol{c}_S + 2(\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b})\boldsymbol{s}_S
\end{aligned}
$$

The gradients of the group delay response are given by

$$
\begin{aligned}
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{r}}T(\omega) + |H(\omega)|^2\frac{\partial T(\omega)}{\partial \boldsymbol{r}} &= \boldsymbol{s}_R\boldsymbol{a}\left(\frac{\partial \boldsymbol{s}_R}{\partial \omega}\boldsymbol{a} + \frac{\partial \boldsymbol{s}_S}{\partial \omega}\boldsymbol{b}\right)\frac{\partial \phi_R(M\omega)}{\partial \boldsymbol{r}} + (\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b})\frac{\partial^2 \boldsymbol{s}_R}{\partial \boldsymbol{r}\partial \omega}\boldsymbol{a} \\
&\quad + \boldsymbol{c}_R\boldsymbol{a}\left(\frac{\partial \boldsymbol{c}_R}{\partial \omega}\boldsymbol{a} + \frac{\partial \boldsymbol{c}_S}{\partial \omega}\boldsymbol{b}\right)\frac{\partial \phi_R(M\omega)}{\partial \boldsymbol{r}} - (\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b})\frac{\partial^2 \boldsymbol{c}_R}{\partial \boldsymbol{r}\partial \omega}\boldsymbol{a} \\
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{s}}T(\omega) + |H(\omega)|^2\frac{\partial T(\omega)}{\partial \boldsymbol{s}} &= \boldsymbol{s}_S\boldsymbol{b}\left(\frac{\partial \boldsymbol{s}_R}{\partial \omega}\boldsymbol{a} + \frac{\partial \boldsymbol{s}_S}{\partial \omega}\boldsymbol{b}\right)\frac{\partial \phi_S(M\omega)}{\partial \boldsymbol{s}} + (\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b})\frac{\partial^2 \boldsymbol{s}_S}{\partial \boldsymbol{s}\partial \omega}\boldsymbol{b} \\
&\quad + \boldsymbol{c}_S\boldsymbol{b}\left(\frac{\partial \boldsymbol{c}_R}{\partial \omega}\boldsymbol{a} + \frac{\partial \boldsymbol{c}_S}{\partial \omega}\boldsymbol{b}\right)\frac{\partial \phi_S(M\omega)}{\partial \boldsymbol{s}} - (\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b})\frac{\partial^2 \boldsymbol{c}_S}{\partial \boldsymbol{s}\partial \omega}\boldsymbol{b} \\
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{a}}T(\omega) + |H(\omega)|^2\frac{\partial T(\omega)}{\partial \boldsymbol{a}} &= \left(\frac{\partial \boldsymbol{s}_R}{\partial \omega}\boldsymbol{a} + \frac{\partial \boldsymbol{s}_S}{\partial \omega}\boldsymbol{b}\right)\boldsymbol{c}_R + (\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b})\frac{\partial \boldsymbol{s}_R}{\partial \omega} \\
&\quad - \left(\frac{\partial \boldsymbol{c}_R}{\partial \omega}\boldsymbol{a} + \frac{\partial \boldsymbol{c}_S}{\partial \omega}\boldsymbol{b}\right)\boldsymbol{s}_R - (\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b})\frac{\partial \boldsymbol{c}_R}{\partial \omega} \\
\frac{\partial |H(\omega)|^2}{\partial \boldsymbol{b}}T(\omega) + |H(\omega)|^2\frac{\partial T(\omega)}{\partial \boldsymbol{b}} &= \left(\frac{\partial \boldsymbol{s}_R}{\partial \omega}\boldsymbol{a} + \frac{\partial \boldsymbol{s}_S}{\partial \omega}\boldsymbol{b}\right)\boldsymbol{c}_S + (\boldsymbol{c}_R\boldsymbol{a} + \boldsymbol{c}_S\boldsymbol{b})\frac{\partial \boldsymbol{s}_S}{\partial \omega} \\
&\quad - \left(\frac{\partial \boldsymbol{c}_R}{\partial \omega}\boldsymbol{a} + \frac{\partial \boldsymbol{c}_S}{\partial \omega}\boldsymbol{b}\right)\boldsymbol{s}_S - (\boldsymbol{s}_R\boldsymbol{a} + \boldsymbol{s}_S\boldsymbol{b})\frac{\partial \boldsymbol{c}_S}{\partial \omega}
\end{aligned}
$$

where

$$
\begin{aligned}
\frac{\partial^2 \boldsymbol{c}_R}{\partial \boldsymbol{r}\partial \omega}\boldsymbol{a} &= (\boldsymbol{s}_R\boldsymbol{a})\frac{\partial^2 \phi_R(M\omega)}{\partial \boldsymbol{r}\partial \omega} + \left(\frac{\partial \boldsymbol{s}_R}{\partial \omega}\boldsymbol{a}\right)\frac{\partial \phi_R(M\omega)}{\partial \boldsymbol{r}} \\
\frac{\partial^2 \boldsymbol{c}_S}{\partial \boldsymbol{s}\partial \omega}\boldsymbol{b} &= (\boldsymbol{s}_S\boldsymbol{b})\frac{\partial^2 \phi_S(M\omega)}{\partial \boldsymbol{s}\partial \omega} + \left(\frac{\partial \boldsymbol{s}_S}{\partial \omega}\boldsymbol{b}\right)\frac{\partial \phi_S(M\omega)}{\partial \boldsymbol{s}} \\
\frac{\partial^2 \boldsymbol{s}_R}{\partial \boldsymbol{r}\partial \omega}\boldsymbol{a} &= -(\boldsymbol{c}_R\boldsymbol{a})\frac{\partial^2 \phi_R(M\omega)}{\partial \boldsymbol{r}\partial \omega} - \left(\frac{\partial \boldsymbol{c}_R}{\partial \omega}\boldsymbol{a}\right)\frac{\partial \phi_R(M\omega)}{\partial \boldsymbol{r}} \\
\frac{\partial^2 \boldsymbol{s}_S}{\partial \boldsymbol{s}\partial \omega}\boldsymbol{b} &= -(\boldsymbol{c}_S\boldsymbol{b})\frac{\partial^2 \phi_S(M\omega)}{\partial \boldsymbol{s}\partial \omega} - \left(\frac{\partial \boldsymbol{c}_S}{\partial \omega}\boldsymbol{b}\right)\frac{\partial \phi_S(M\omega)}{\partial \boldsymbol{s}}
\end{aligned}
$$

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:

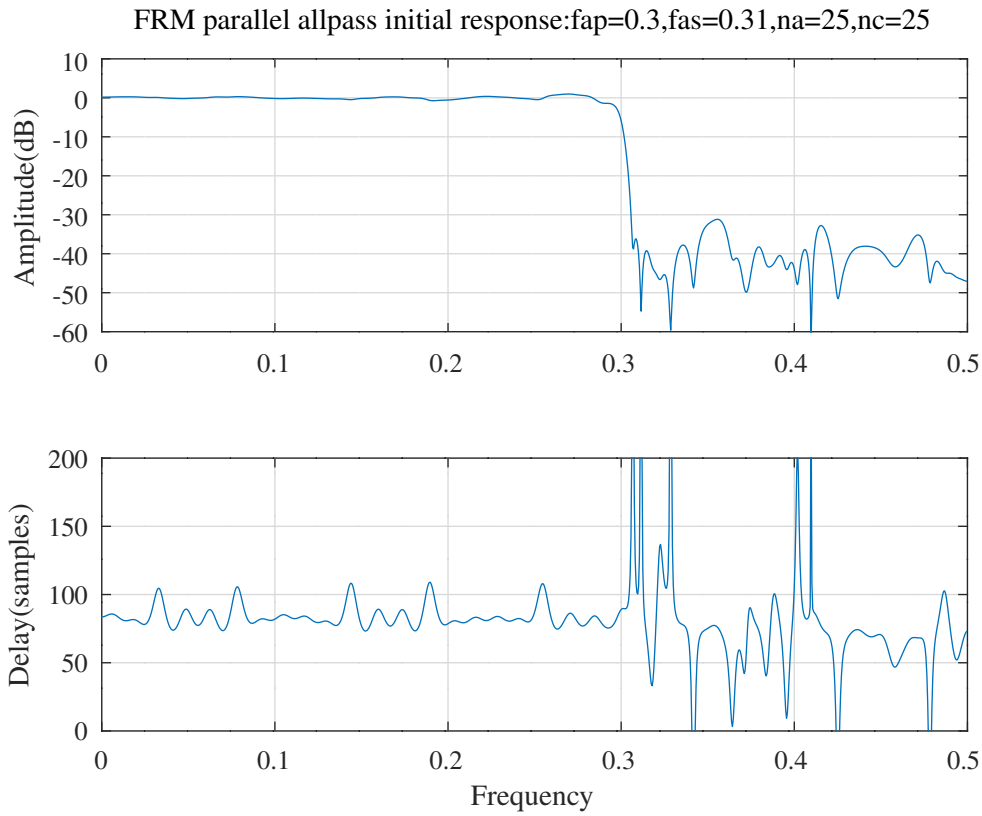



Figure 10.104: FRM filter with gain-pole-zero format parallel allpass model filter : initial response with WISE optimisation

```
tol=0.001 % Tolerance on coefficient update vector
ctol=2e-05 % Tolerance on constraints
n=500 % 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=0.04 % 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=0.31 % Stop band edge
dBas=40 % Stop band attenuation ripple
Was=10 % 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 8.1.5). The response of the initial filter is shown in Figure 10.104. The initial filter was SOCP and PCLS optimised with the Octave function *iir_frm_parallel_allpass_slb*. The resulting model filter parallel allpass filter denominator polynomials are

```
r = [   1.0000000000,  -0.3609323789,   0.6575786320,  -0.2218466618, ...
       -0.0214686384,   0.0518837804,  -0.0103025664,  -0.0036209391, ...
        0.0033005761 ]';
```

and

```
s = [   1.0000000000,  -0.5764912563,   0.2356295634,   0.0329805162, ...
       -0.0732205238,   0.0113745030,   0.0106441729,   0.0027049155 ]';
```

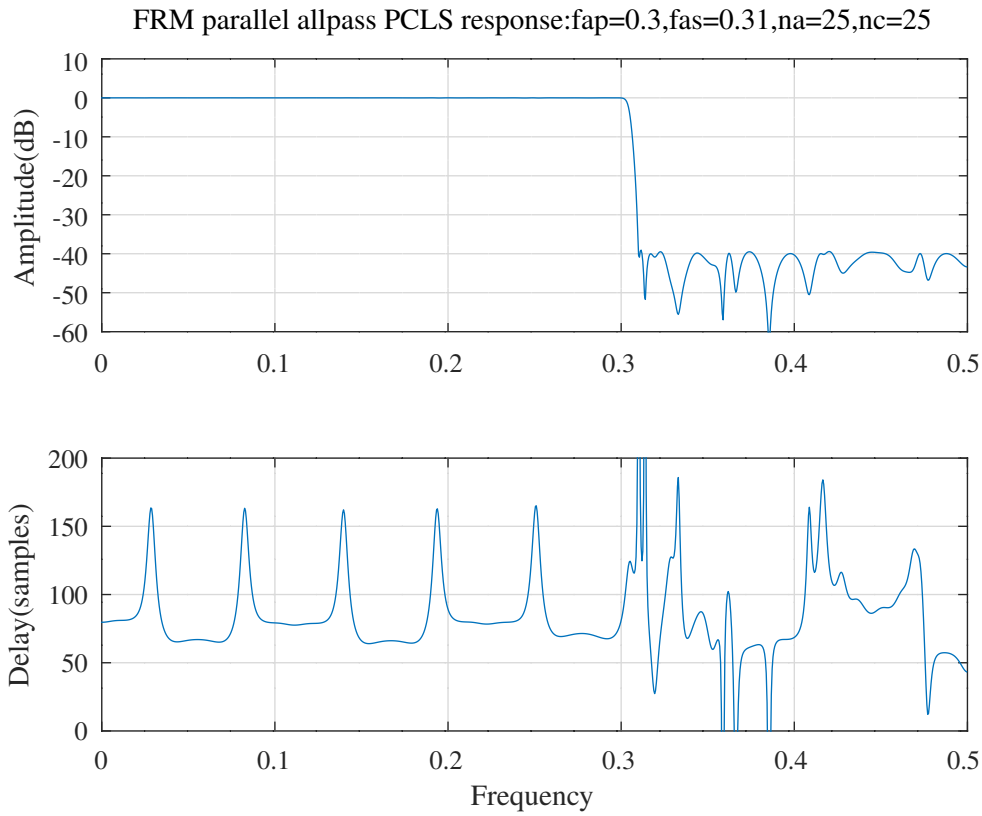



Figure 10.105: FRM filter with gain-pole-zero format parallel allpass model filter:SOCP PCLS optimised overall response

The FIR masking filter polynomial is

```
aa = [   0.0078989030,   0.0160183581,  -0.0266566372,  -0.0403264574, ...
         0.0161681486,   0.0120735110,  -0.0343958748,  -0.0032565973, ...
         0.0036305283,  -0.1458282291,  -0.1332423468,   0.2809612827, ...
         0.5645849007,   0.3076237531,  -0.0238603457,  -0.0058985727, ...
         0.1143791082,   0.0453060785,  -0.0392197866,   0.0018193453, ...
         0.0353008881,   0.0203582813,   0.0132790009,   0.0054332569, ...
         0.0069579142 ]';
```

and the complementary FIR masking filter polynomial is

```
ac = [   0.0101093076,  -0.0228344876,  -0.0125188206,   0.0046804495, ...
        -0.0356563148,  -0.0240334972,   0.0470780681,  -0.0008369752, ...
        -0.0885592761,  -0.0816669578,  -0.0873836137,   0.1827120689, ...
         0.5867422108,   0.3946448820,  -0.1102909779,  -0.0366716860, ...
         0.2188327636,   0.0055201079,  -0.1171704576,   0.0458981948, ...
         0.0686261044,  -0.0291983615,   0.0009643986,   0.0348993849, ...
        -0.0008874613 ]';
```

Figure 10.105 shows the overall response of the resulting SOCP and PCLS optimised FRM filter. Figure 10.106 shows the passband response of the resulting FRM filter. Figure 10.107 shows the impulse response of the resulting FRM masking filters. Figure 10.108 shows the responses of the resulting FRM masking filters. Figure 10.109 shows the response of the resulting FRM model filter[a].

[a]The responses shown are calculated with the allpass filter polynomial derived from the gain-pole-zero form.

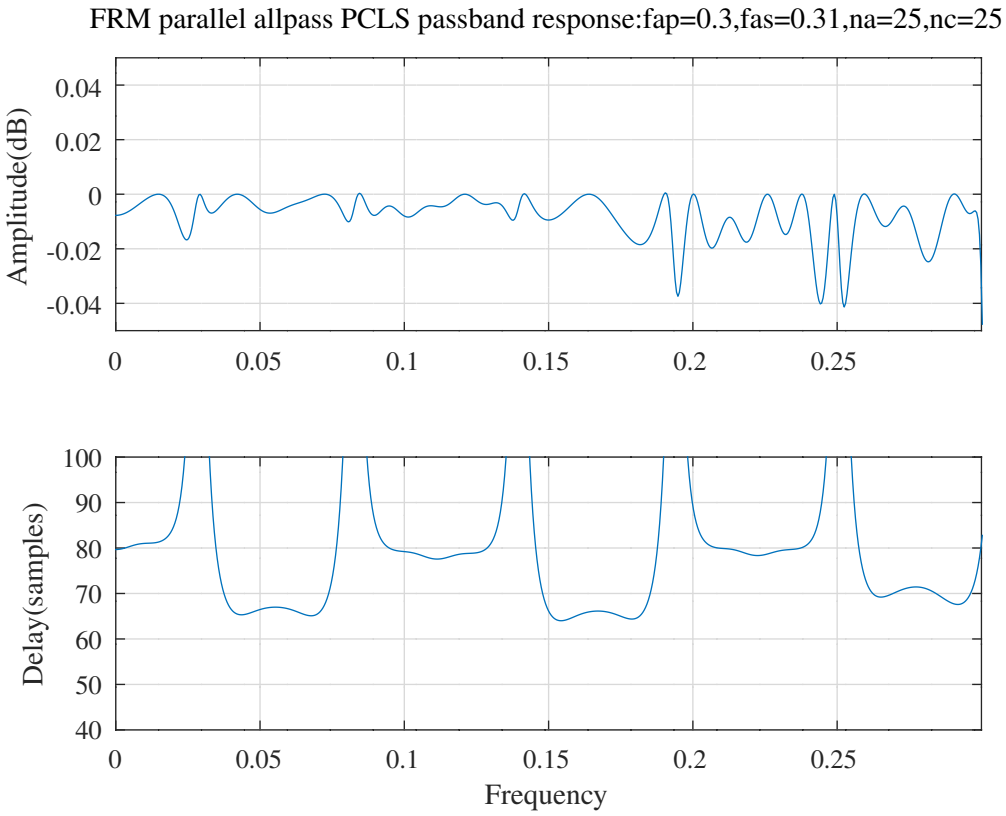



Figure 10.106: FRM filter with gain-pole-zero format parallel allpass model filter:SOCP PCLS optimised passband response

Figure 10.107: FRM filter with gain-pole-zero format parallel allpass model filter:SOCP PCLS optimised FIR masking filters

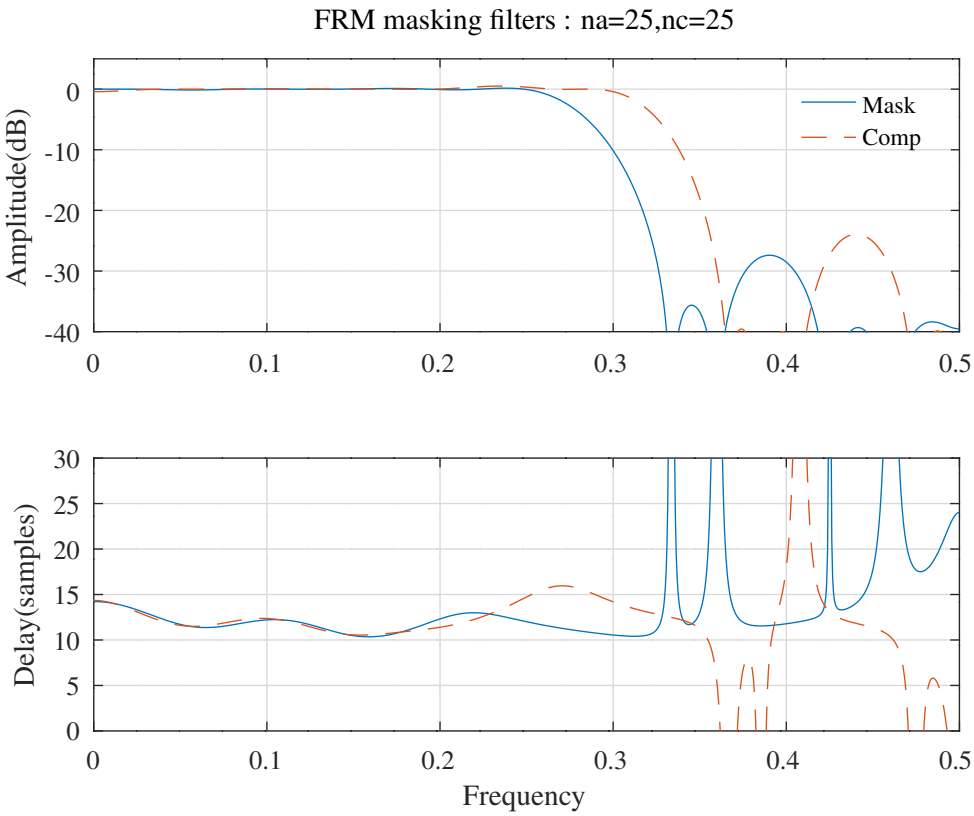



Figure 10.108: FRM filter with gain-pole-zero format parallel allpass model filter:SOCP PCLS optimised FIR masking filter responses

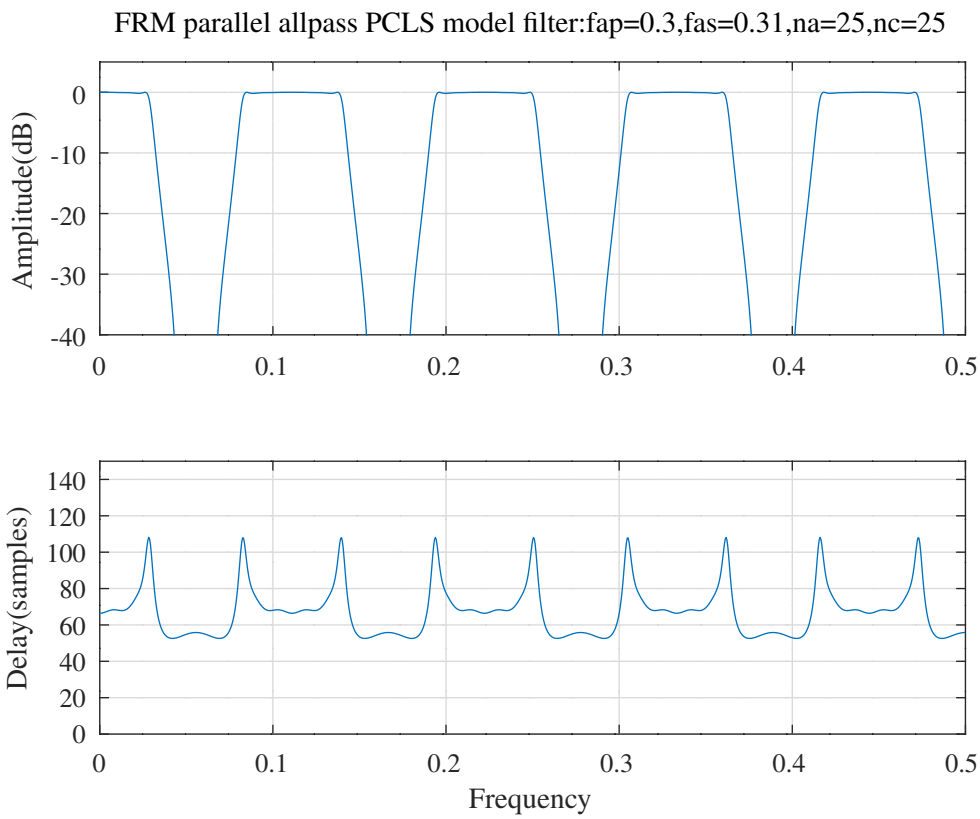



Figure 10.109: FRM filter with gain-pole-zero format parallel allpass model filter:SOCP PCLS optimised model filter response

### 10.4.6 Design of an FRM low-pass digital filter with an all-pass Schur lattice model filter in parallel with a delay using SOCP and PCLS optimisation

This section describes a low-pass FRM filter with the *Johansson* and *Wanhammar* FRM filter structure shown in Figure 10.87 in which the model filter is an all-pass Schur one-multiplier lattice filter in parallel with a delay and the FIR masking filters are symmetric (ie: linear phase) and even order. The squared-magnitude, phase 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 8.1.2. The calculations for this filter are similar to those in Section 10.4.4 with the values and gradients of $\phi_R(\omega)$ calculated by the Octave *schurOneMAPlatticeP.m* and *schurOneMAPlatticeT.m* functions. The Octave function *schurOneMAPlattice_frmEsq.m* returns the FRM filter squared-magnitude, phase and group delay error responses and their gradients. The Octave function *schurOneMAPlattice_frm_socp_mmse* finds the SOCP solution that optimises the coefficients of a filter error response calculated by *schurOneMAPlattice_frmEsq* with the required amplitude, phase and group delay constraints. The Octave function *schurOneMAPlattice_frm_slb* implements the PCLS algorithm for finding the constraint frequencies. The Octave script *schurOneMAPlattice_frm_socp_slb_test.m* designs an FRM filter with the following specification:

```
n=1000 % Frequency points
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
Mmodel=9 % Model filter decimation
Dmodel=9 % Desired model filter passband delay
dmask=20 % FIR masking filter delay
fap=0.3 % Amplitude pass band edge
dBap=0.05 % Pass band amplitude ripple
Wap=1 % Pass band amplitude weight
fas=0.3105 % Amplitude stop band edge
dBas=43 % Stop band amplitude ripple
Was=10 % Stop band amplitude weight
ftp=0.3 % Delay pass band edge
tp=101 % Nominal FRM filter delay
tpr=tp/101 % Peak-to-peak pass band delay ripple
Wtp=0.02 % Pass band delay weight
fpp=0.3 % Phase pass band edge
pp=0*pi % Nominal passband phase (adjusted for delay)
ppr=pi/100 % Peak-to-peak pass band phase ripple
Wpp=0.01 % Pass band phase weight
rho=0.968750 % Constraint on allpass pole radius
```

Both the FIR masking filters are symmetric and 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 8.1.5. The response of the initial filter is shown in Figure 10.99. After SOCP and PCLS optimisation of the initial filter the resulting model filter all-pass Schur lattice filter has coefficients:

```
k1 = [  -0.0121468836,   0.5790957833,   0.0168575877,  -0.1439129895, ...
        -0.0043721524,   0.0562377167,   0.0085302320,  -0.0268776209, ...
        -0.0044278279,   0.0112909020 ]';
```

```
epsilon1 = [  1,  1, -1,  1, ...
              1, -1, -1,  1, ...
              1, -1 ];
```

The distinct coefficients of the FIR masking filter polynomial are

```
u1 = [   0.5758901065,   0.3015888882,  -0.0567294779,  -0.0857731768, ...
         0.0512608568,   0.0331424537,  -0.0413951614,  -0.0099741315, ...
         0.0388904128,  -0.0168137861,  -0.0138670813,   0.0079329199, ...
         0.0103266569,  -0.0088208897,  -0.0040616549,   0.0071820392, ...
         0.0013646935,  -0.0097370361,   0.0076903209,   0.0004045362, ...
        -0.0006503635 ]';
```

The distinct coefficients of the complementary FIR masking filter polynomial are

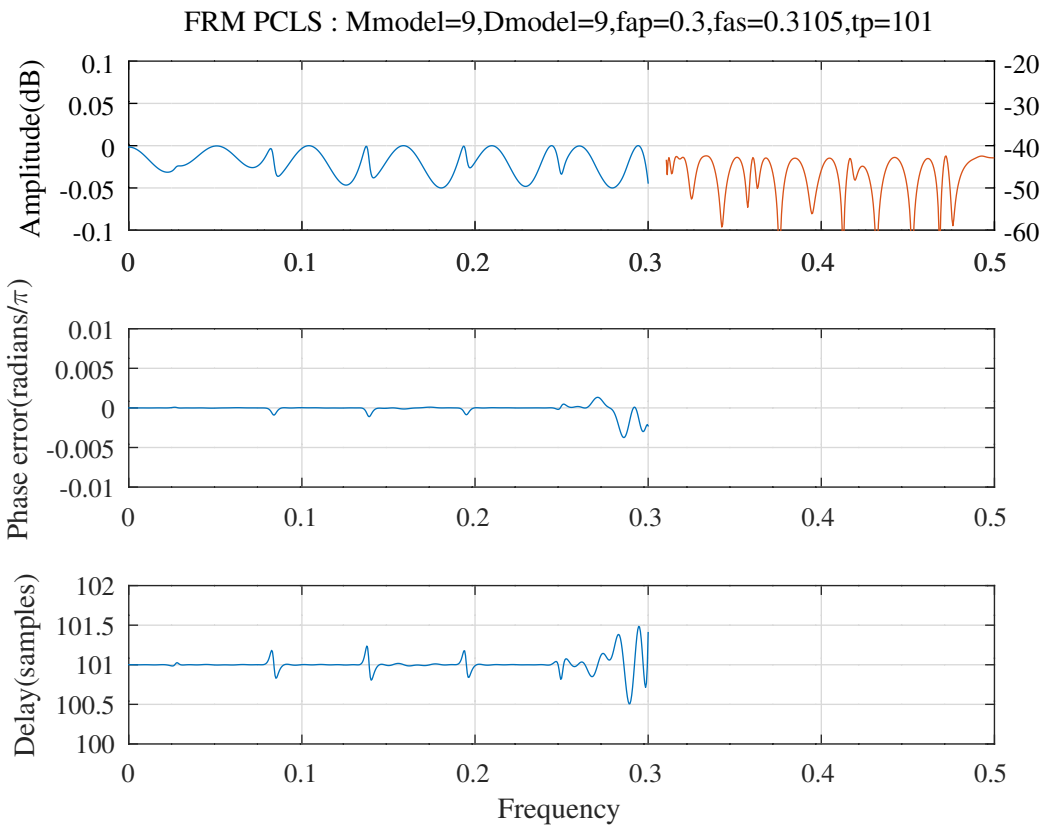



Figure 10.110: FRM filter with Schur one-multiplier lattice filter allpass model filter:amplitude, delay and phase responses after SOCP and PCLS optimisation

```
v1 = [  -0.6685798645,  -0.2724341193,   0.1314647262,   0.0041918831, ...
        -0.0656979387,   0.0481241599,   0.0045975243,  -0.0361540141, ...
         0.0296288853,  -0.0034058258,  -0.0175251927,   0.0125430317, ...
         0.0026158676,  -0.0110444841,   0.0078744807,   0.0018934884, ...
        -0.0072156928,   0.0065669589,  -0.0018514169,  -0.0031492251, ...
         0.0011187234 ]';
```

Figure 10.110 shows the response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 10.111 shows the responses of the resulting FRM masking filters. Figure 10.112 shows the response of the resulting FRM model filter. Figure 10.113 compares the amplitude response of the FRM low-pass filter with that of a linear-phase FIR filter designed with the Octave *remez* function. The FIR and FRM low-pass filters have a similar number of distinct coefficients.

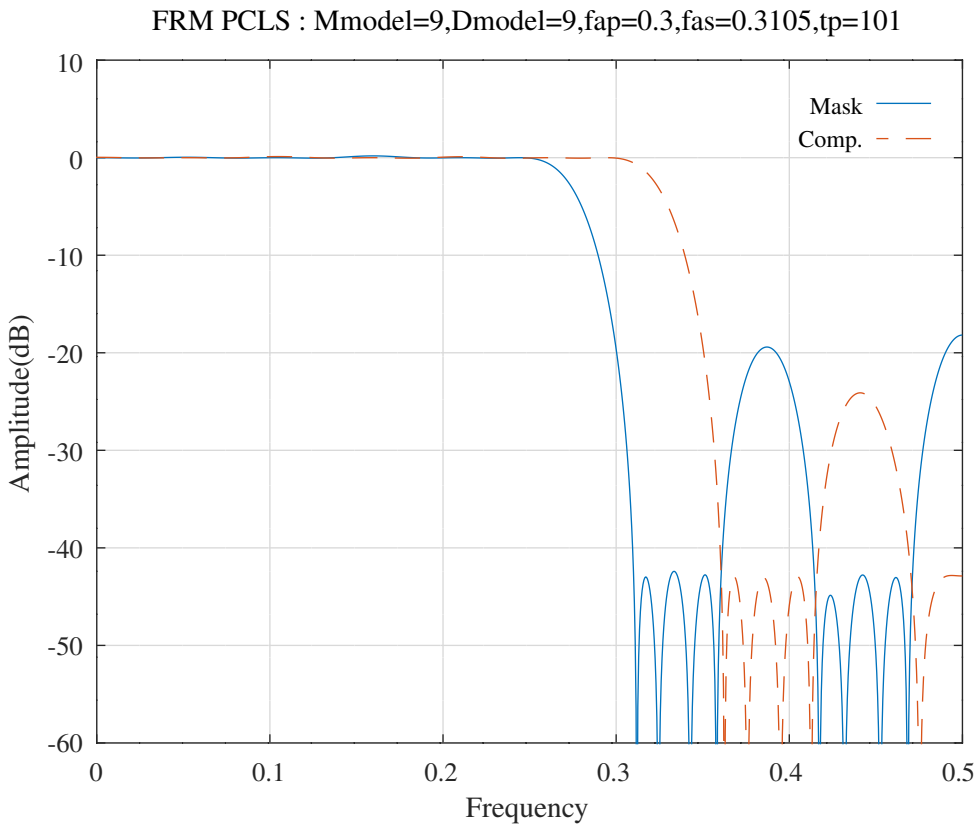



Figure 10.111: FRM filter with Schur one-multiplier lattice allpass model filter:masking filter responses after SOCP and PCLS optimisation

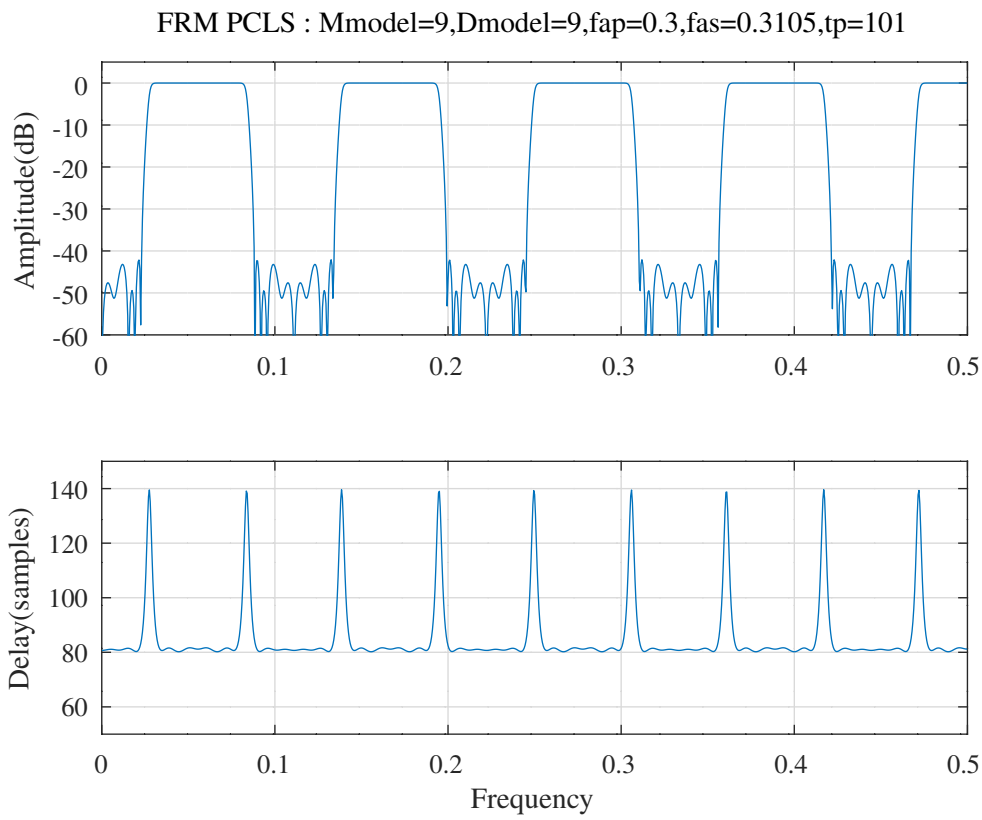



Figure 10.112: FRM filter with Schur one-multiplier lattice allpass model filter:model filter response after SOCP and PCLS optimisation

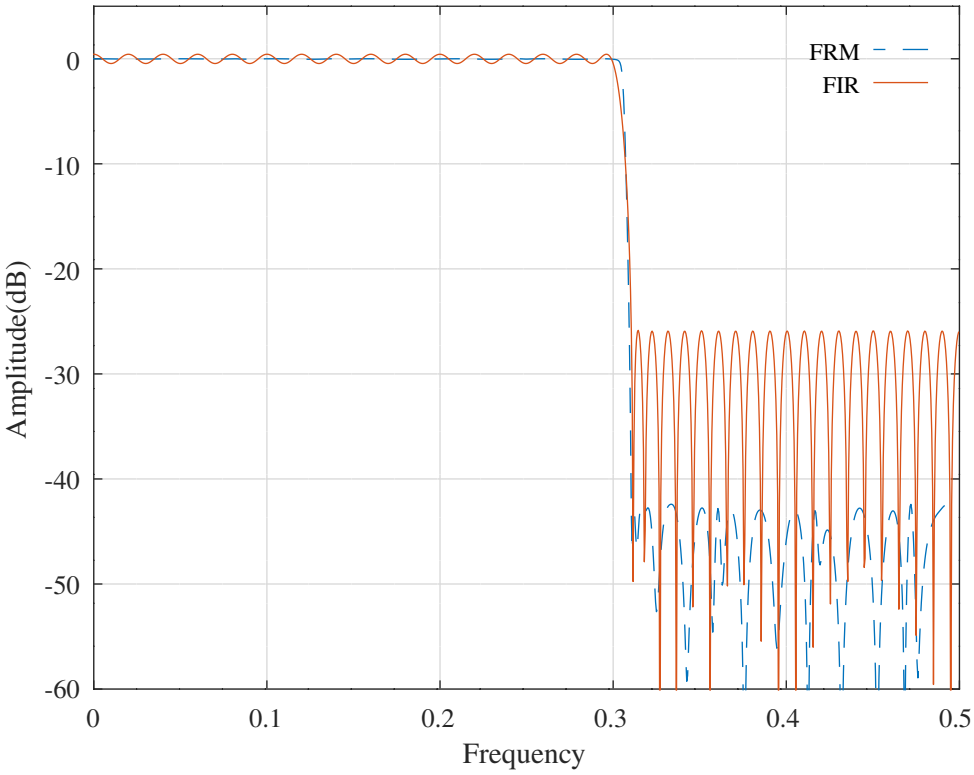



Figure 10.113: Comparison of the amplitude responses of an FRM filter with Schur one-multiplier lattice allpass model filter and a linear-phase FIR filter having a similar number of distinct coefficients.

### 10.4.7 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 10.87. 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 $z^2$, $R\left(z^2\right)$. The coefficients of the denominator polynomial of $R\left(z^2\right)$ 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:

$$\begin{aligned} H(z) &= \frac{1}{2}\left[R\left(z^{2M}\right) + z^{-DM}\right] F_{M_a}(z) + \frac{1}{2}\left[R\left(z^{2M}\right) - z^{-DM}\right] F_{M_c}(z) \\ &= \frac{1}{2} R\left(z^{2M}\right)\left[F_{M_a}(z) + F_{M_c}(z)\right] + \frac{1}{2} z^{-DM}\left[F_{M_a}(z) - F_{M_c}(z)\right] \end{aligned}$$

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 10.4.1 shows the calculation of the band edges of the masking filters required for the FRM filter corresponding to these model filters. *Saramäki* et al. [204, 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:

$$F_{M_a}(z) = U\left(z^2\right) + z^{-1} V\left(z^2\right)$$

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 $F_{M_a}$:

$$\begin{aligned} F_{M_a}(z) + F_{M_c}(z) &= 2U\left(z^2\right) - z^{-d} \\ F_{M_a}(z) - F_{M_c}(z) &= 2z^{-1} V\left(z^2\right) + z^{-d} \end{aligned}$$

so that:

$$H(z) = \frac{1}{2} z^{-DM-d} + \frac{1}{2}\left[R\left(z^{2M}\right)\left(2U\left(z^2\right) - z^{-d}\right) + 2z^{-DM} z^{-1} V\left(z^2\right)\right]$$

It is convenient to express $U\left(z^2\right)$ and $z^{-1}V\left(z^2\right)$ in terms of the coefficients, $u_k$ and $v_k$:

$$\begin{aligned} U\left(z^2\right) &= \sum_{k=0}^{d} u_k z^{-2k} \\ &= u_{\frac{d}{2}} z^{-d} + \sum_{k=0}^{\frac{d}{2}-1} u_k z^{-2k} + \sum_{k=\frac{d}{2}+1}^{d} u_k z^{-2k} \\ &= u_{\frac{d}{2}} z^{-d} + \sum_{k=0}^{\frac{d}{2}-1} u_k \left(z^{-2k} + z^{2k-2d}\right) \\ &= z^{-d}\left[u_{\frac{d}{2}} + \sum_{k=0}^{\frac{d}{2}-1} u_k \left(z^{-2k+d} + z^{2k-d}\right)\right] \end{aligned}$$

and:

$$\begin{aligned} z^{-1} V\left(z^2\right) &= z^{-1} \sum_{k=0}^{d-1} v_k z^{-2k} \\ &= z^{-1} \sum_{k=0}^{\frac{d}{2}-1} v_k z^{-2k} + z^{-1} \sum_{k=\frac{d}{2}}^{d-1} v_k z^{-2k} \\ &= z^{-1} \sum_{k=0}^{\frac{d}{2}-1} v_k \left(z^{-2k} + z^{2k-2d+2}\right) \end{aligned}$$

$$= z^{-d} \sum_{k=0}^{\frac{d}{2}-1} v_k \left(z^{-2k+d-1} + z^{2k-d+1}\right)$$

Rearranging $H(z)$:

$$H(z) = z^{-DM-d} \left[ z^{DM} R\left(z^{2M}\right) \left(-\frac{1}{2} + z^d U\left(z^2\right)\right) + \left(\frac{1}{2} + z^d z^{-1} V\left(z^2\right)\right) \right]$$

so that the zero-phase frequency response of the half-band FRM filter, $H(z)$, is:

$$H(\omega) = e^{\imath[DM\omega + \phi_R(2M\omega)]} A(\omega) + B(\omega)$$

where $\phi_R(2M\omega)$ is the phase response of the all-pass filter $R\left(z^{2M}\right)$ and:

$$A(\omega) = -\frac{1}{2} + u_{\frac{d}{2}} + \sum_{k=0}^{\frac{d}{2}-1} 2u_k \cos\left[\omega(2k-d)\right]$$
$$B(\omega) = \frac{1}{2} + \sum_{k=0}^{\frac{d}{2}-1} 2v_k \cos\left[\omega(2k-d+1)\right]$$

The squared-magnitude and phase responses of the zero phase response of the FRM half-band filter are similar to those shown in Section 10.4.4:

$$|H(\omega)|^2 = A^2(\omega) + B^2(\omega) + 2A(\omega)B(\omega)\cos\phi_Z(M\omega)$$
$$\phi_H(\omega) = \arctan \frac{A(\omega)\sin\phi_Z(M\omega)}{A(\omega)\cos\phi_Z(M\omega) + B(\omega)}$$

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:

$$|H(\omega)|^2 T(\omega) = -\left(A^2(\omega) + A(\omega)B(\omega)\cos\phi_Z(M\omega)\right)\frac{\partial\phi_Z(M\omega)}{\partial\omega} \ldots$$
$$- \sin\phi_Z(M\omega)\left[B(\omega)\frac{\partial A(\omega)}{\partial\omega} - A(\omega)\frac{\partial B(\omega)}{\partial\omega}\right]$$

where:

$$\frac{\partial A(\omega)}{\partial\omega} = -2\sum_{k=0}^{\frac{d}{2}-1}(2k-d)\,u_k \sin\left[\omega(2k-d)\right]$$
$$\frac{\partial B(\omega)}{\partial\omega} = -2\sum_{k=0}^{\frac{d}{2}-1}(2k-d+1)\,v_k \sin\left[\omega(2k-d+1)\right]$$

The gradients of $|H(\omega)|^2$ with respect to the coefficients are:

$$\frac{\partial|H(\omega)|^2}{\partial r_k} = -2A(\omega)B(\omega)\sin\phi_Z(M\omega)\frac{\partial\phi_R(2M\omega)}{\partial r_k}$$
$$\frac{\partial|H(\omega)|^2}{\partial u_k} = 2\left(A(\omega) + B(\omega)\cos\phi_Z(M\omega)\right)\frac{\partial A(\omega)}{\partial u_k}$$
$$\frac{\partial|H(\omega)|^2}{\partial v_k} = 2\left(B(\omega) + A(\omega)\cos\phi_Z(M\omega)\right)\frac{\partial B(\omega)}{\partial v_k}$$

and:

$$\frac{\partial A(\omega)}{\partial u_k} = \begin{cases} 1 & k = \frac{d}{2} \\ 2\cos\left[\omega(2k-d)\right] & otherwise \end{cases}$$
$$\frac{\partial B(\omega)}{\partial v_k} = 2\cos\left[\omega(2k-d+1)\right]$$

The gradients of $T(\omega)$ with respect to the coefficients are given by:

$$\begin{aligned}
\frac{\partial |H(\omega)|^2}{\partial r_k} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial r_k} = & \left(A^2(\omega) + A(\omega) B(\omega) \cos\phi_Z(M\omega)\right) \frac{\partial^2 \phi_R(2M\omega)}{\partial\omega \partial r_k} \ldots \\
& + A(\omega) B(\omega) \sin\phi_Z(M\omega) \frac{\partial \phi_Z(M\omega)}{\partial\omega} \frac{\partial \phi_R(2M\omega)}{\partial r_k} \ldots \\
& - \cos\phi_Z(M\omega) \left[ B(\omega) \frac{\partial A(\omega)}{\partial\omega} - A(\omega) \frac{\partial B(\omega)}{\partial\omega} \right] \frac{\partial \phi_R(2M\omega)}{\partial r_k} \\
\frac{\partial |H(\omega)|^2}{\partial u_k} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial u_k} = & - \left(2A(\omega) + B(\omega) \cos\phi_Z(M\omega)\right) \frac{\partial \phi_Z(M\omega)}{\partial\omega} \frac{\partial A(\omega)}{\partial u_k} \ldots \\
& - \sin\phi_Z(M\omega) \left[ B(\omega) \frac{\partial^2 A(\omega)}{\partial\omega \partial u_k} - \frac{\partial B(\omega)}{\partial\omega} \frac{\partial A(\omega)}{\partial u_k} \right] \\
\frac{\partial |H(\omega)|^2}{\partial v_k} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial v_k} = & - A(\omega) \cos\phi_Z(M\omega) \frac{\partial \phi_Z(M\omega)}{\partial\omega} \frac{\partial B(\omega)}{\partial v_k} \ldots \\
& - \sin\phi_Z(M\omega) \left[ \frac{\partial A(\omega)}{\partial\omega} \frac{\partial B(\omega)}{\partial v_k} - A(\omega) \frac{\partial^2 B(\omega)}{\partial\omega \partial v_k} \right]
\end{aligned}$$

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-05 % Tolerance on coefficient update vector
ctol=1e-07 % Tolerance on constraints
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=0.335 % 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 8.1.5. Figure 10.114 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 = [   0.5529783588,  -0.1247983547,   0.0415763466,  -0.0135409242, ...
         0.0022491861 ]';

epsilon2 = [  1,  1, -1,  1, ...
             -1 ];
```

The FIR masking filter polynomials are:

```
u2 = [  -0.0004669752,   0.0023547510,  -0.0070121550,   0.0129930129, ...
        -0.0309489227,   0.0346492104,  -0.0508887501,   0.0580403399, ...
         0.4385549755 ]';

v2 = [   0.0065985617,  -0.0044784965,   0.0066983590,  -0.0031478118, ...
        -0.0069818388,   0.0306895254,  -0.0815136595,   0.3141373617 ]';
```

Figure 10.115 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 10.116 shows the passband response of the resulting FRM filter. Figure 10.117 shows the responses of the resulting FRM masking filters. Figure 10.118 shows the response of the resulting FRM model filter.

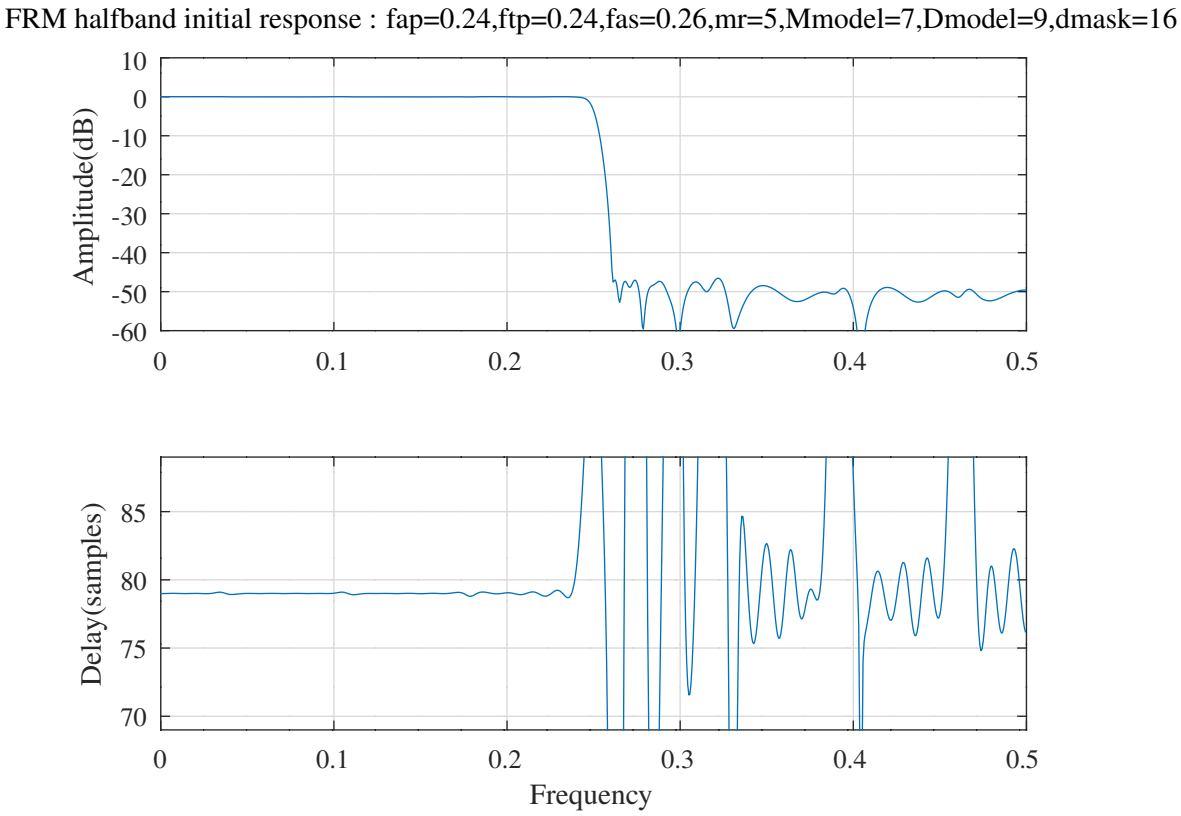



Figure 10.114: FRM half-band filter : initial response

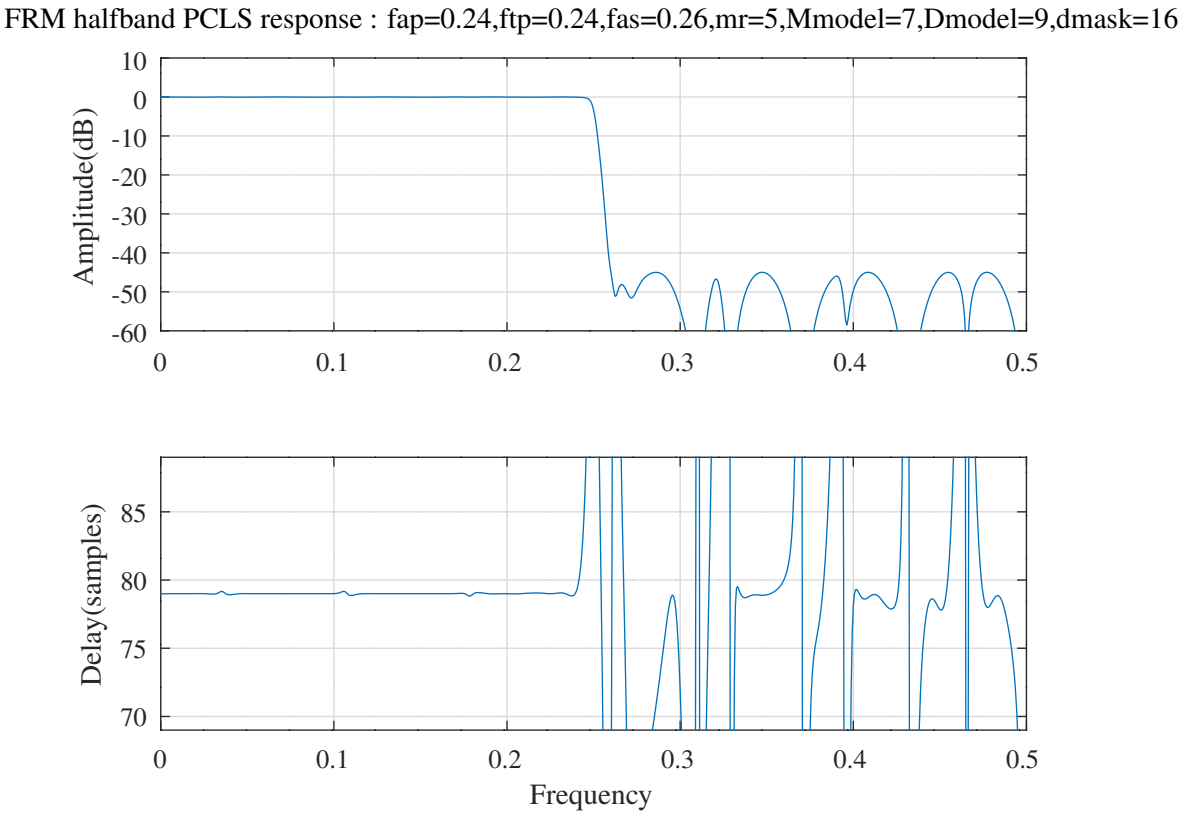



Figure 10.115: FRM half-band filter:overall response after SOCP and PCLS optimisation

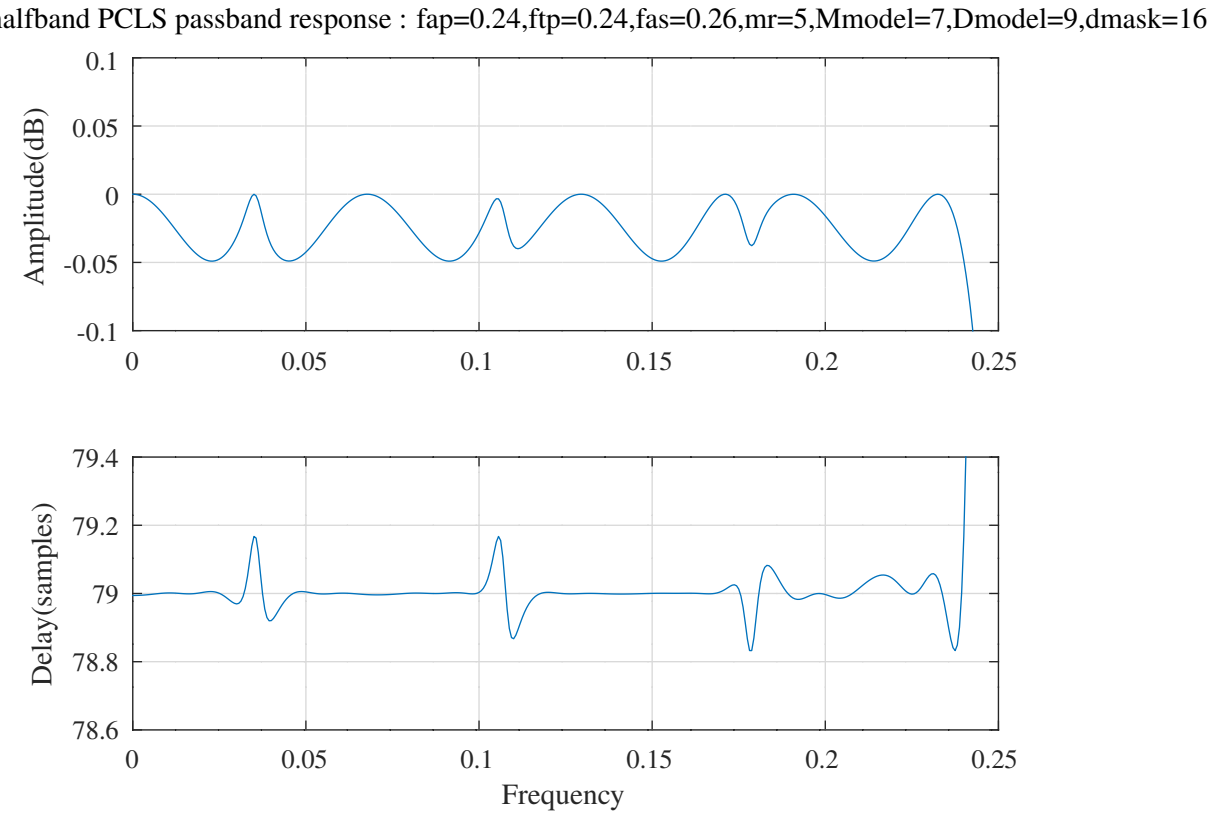



Figure 10.116: FRM half-band filter : passband response after SOCP and PCLS optimisation

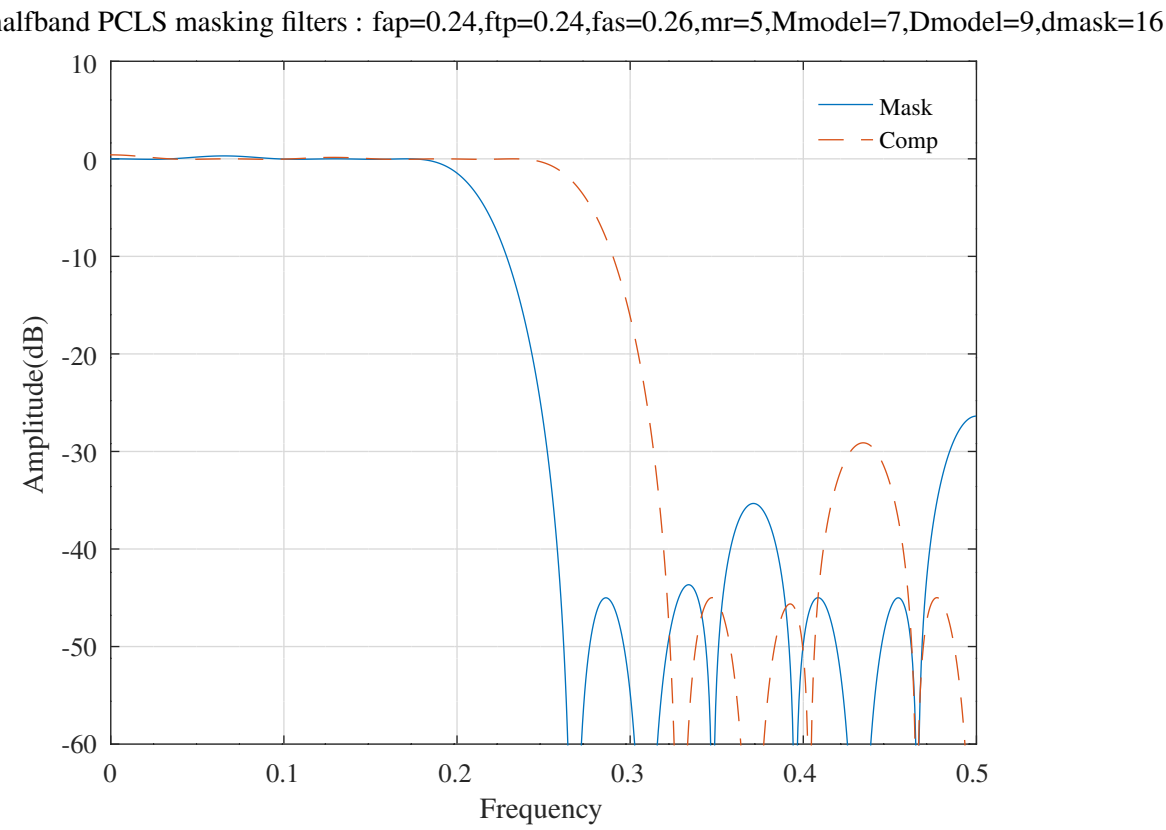



Figure 10.117: FRM half-band filter with SOCP and PCLS optimisation:masking filter responses

FRM halfband PCLS model filter : fap=0.24,ftp=0.24,fas=0.26,mr=5,Mmodel=7,Dmodel=9,dmask=16

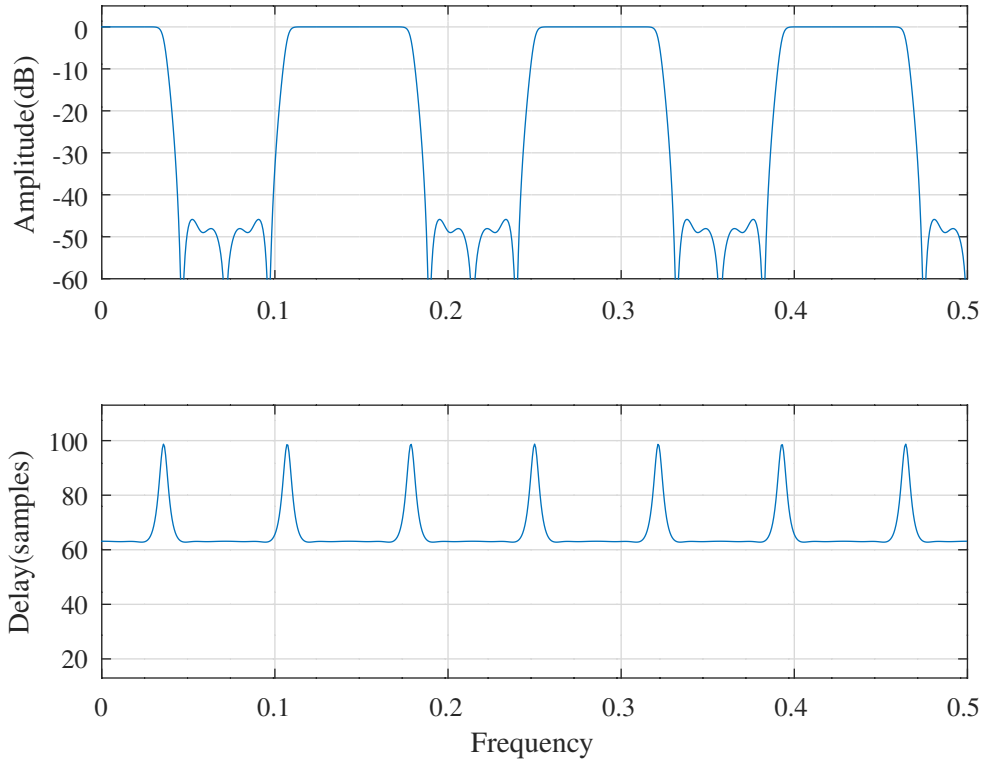



Figure 10.118: FRM half-band filter with SOCP and PCLS optimisation:model filter response

### 10.4.8 Design of an FRM Hilbert digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation

*Milić* et al. [106] point out that shifting the transfer function of the FRM half-band filter, $H(z)$, of Section 10.4.7, 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:

$$H(z) = \frac{1}{2}z^{-N} + \frac{1}{2}Q\left(z^2\right)$$

where $N+1$ is a multiple of 4. The filter, $H_A(z)$, that generates the analytic signal is:

$$\begin{aligned} H_A(z) &= 2\imath H(-\imath z) \\ &= z^{-N} + \imath Q\left(-z^2\right) \end{aligned}$$

In other words, $H_H(z) = Q\left(-z^2\right)$ 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:

$$H_H(z) = R\left(-z^{2M}\right)\left(2U\left(-z^2\right) - z^{-d}\right) + 2z^{-DM-1}V\left(-z^2\right)$$

where:

$$\begin{aligned} U\left(-z^2\right) &= \sum_{k=0}^{d} u_k\left(-z^2\right)^{-k} \\ &= (-1)^{-\frac{d}{2}} u_{\frac{d}{2}} z^{-d} + \sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} u_k z^{-2k} + \sum_{k=\frac{d}{2}+1}^{d} (-1)^{-k} u_k z^{-2k} \\ &= u_{\frac{d}{2}} z^{-d} + \sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} u_k\left(z^{-2k} + z^{2k-2d}\right) \\ &= z^{-d}\left[u_{\frac{d}{2}} + \sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} u_k\left(z^{-2k+d} + z^{2k-d}\right)\right] \end{aligned}$$

and:

$$\begin{aligned} z^{-1}V\left(-z^2\right) &= z^{-1}\sum_{k=0}^{d-1} v_k\left(-z^2\right)^{-k} \\ &= z^{-1}\sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} v_k z^{-2k} + z^{-1}\sum_{k=\frac{d}{2}}^{d-1} (-1)^{-k} v_k z^{-2k} \\ &= z^{-d}\sum_{k=0}^{\frac{d}{2}-1} (-1)^{-k} v_k\left(z^{-2k+d-1} - z^{2k-d+1}\right) \end{aligned}$$

If $r\left(z^2\right)$ is the denominator polynomial of the prototype all-pass filter prototype, $R\left(z^2\right)$, then:

$$\begin{aligned} R\left(-z^2\right) &= \frac{\left(-z^{-2}\right)^{N_{model}} r\left(-z^{-2}\right)}{r\left(-z^2\right)} \\ &= (-1)^{N_{model}} \frac{z^{-2N_{model}} r\left(-z^{-2}\right)}{r\left(-z^2\right)} \end{aligned}$$

where $N_{model}$ is the order of $r(z)$. If $N_{model}$ is odd then the zero frequency gain of $R\left(-z^2\right)$ 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:

$$H_H(\omega) = e^{\imath\left[DM\omega + \phi_{R_H}(2M\omega)\right]} A_H(\omega) + \imath B_H(\omega)$$

where:

$$A_H(\omega) = -1 + 2u'_{\frac{d}{2}} + 4\sum_{k=0}^{\frac{d}{2}-1} u'_k \cos\left[\omega(2k-d)\right]$$

$$B_H(\omega) = -4\sum_{k=0}^{\frac{d}{2}-1} v'_k \sin\left[\omega\left(2k-d+1\right)\right]$$

and $\phi_{R_H}(2M\omega)$ is the phase response of the modified all-pass filter $R_H\left(z^{2M}\right) = R\left(-z^{2M}\right)$.

The squared-magnitude and phase responses of the zero phase response of the FRM Hilbert filter are:

$$\left|H_H(\omega)\right|^2 = A_H^2(\omega) + B_H^2(\omega) + 2A_H(\omega)B_H(\omega)\sin\phi_{Z_H}(M\omega)$$

and:

$$\phi_{H_H}(\omega) = \arctan\frac{A_H(\omega)\sin\phi_{Z_H}(M\omega) + B_H(\omega)}{A_H(\omega)\cos\phi_{Z_H}(M\omega)}$$

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:

$$\begin{aligned}\left|H_H(\omega)\right|^2 T_H(\omega) = &-\left(A_H^2(\omega) + A_H(\omega)B_H(\omega)\sin\phi_{Z_H}(M\omega)\right)\frac{\partial\phi_{Z_H}(M\omega)}{\partial\omega}\ldots\\ &+\cos\phi_{Z_H}(M\omega)\left[B_H(\omega)\frac{\partial A_H(\omega)}{\partial\omega} - A_H(\omega)\frac{\partial B_H(\omega)}{\partial\omega}\right]\end{aligned}$$

where:

$$\begin{aligned}\frac{\partial A_H(\omega)}{\partial\omega} &= -4\sum_{k=0}^{\frac{d}{2}-1}(2k-d)\,u'_k\sin\left[\omega\left(2k-d\right)\right]\\ \frac{\partial B_H(\omega)}{\partial\omega} &= -4\sum_{k=0}^{\frac{d}{2}-1}(2k-d+1)\,v'_k\cos\left[\omega\left(2k-d+1\right)\right]\end{aligned}$$

The gradients of $\left|H_H(\omega)\right|^2$ with respect to the coefficients are:

$$\begin{aligned}\frac{\partial\left|H_H(\omega)\right|^2}{\partial r'_k} &= 2A_H(\omega)B_H(\omega)\cos\phi_{Z_H}(M\omega)\frac{\partial\phi_{R_H}(2M\omega)}{\partial r'_k}\\ \frac{\partial\left|H_H(\omega)\right|^2}{\partial u'_k} &= 2\left(A_H(\omega) + B_H(\omega)\sin\phi_{Z_H}(M\omega)\right)\frac{\partial A_H(\omega)}{\partial u'_k}\\ \frac{\partial\left|H_H(\omega)\right|^2}{\partial v'_k} &= 2\left(B_H(\omega) + A_H(\omega)\sin\phi_{Z_H}(M\omega)\right)\frac{\partial B_H(\omega)}{\partial v'_k}\end{aligned}$$

where:

$$\begin{aligned}\frac{\partial A_H(\omega)}{\partial u'_k} &= \begin{cases}1 & k=\frac{d}{2}\\ 4\cos\left[\omega\left(2k-d\right)\right] & otherwise\end{cases}\\ \frac{\partial B_H(\omega)}{\partial v'_k} &= -4\sin\left[\omega\left(2k-d+1\right)\right]\end{aligned}$$

The gradients of $\phi_{H_H}(\omega)$ with respect to the coefficients are given by:

$$\begin{aligned}\left|H_H(\omega)\right|^2\frac{\partial\phi_{H_H}(\omega)}{\partial r'_k} &= \left(A_H^2(\omega) + A_H(\omega)B_H(\omega)\sin\phi_{Z_H}(M\omega)\right)\frac{\partial\phi_{Z_H}(2M\omega)}{\partial\omega}\\ \left|H_H(\omega)\right|^2\frac{\partial\phi_{H_H}(\omega)}{\partial u'_k} &= -B_H(\omega)\cos\phi_{Z_H}(M\omega)\frac{\partial A_H(\omega)}{\partial u'_k}\\ \left|H_H(\omega)\right|^2\frac{\partial\phi_{H_H}(\omega)}{\partial v'_k} &= A_H(\omega)\cos\phi_{Z_H}(M\omega)\frac{\partial B_H(\omega)}{\partial v'_k}\end{aligned}$$

The gradients of $T_H(\omega)$ with respect to the coefficients are given by:

$$\frac{\partial\left|H_H(\omega)\right|^2}{\partial r'_k}T_H(\omega) + \left|H_H(\omega)\right|^2\frac{\partial T_H(\omega)}{\partial r'_k} = -A_H(\omega)B_H(\omega)\cos\phi_{Z_H}(M\omega)\frac{\partial\phi_{Z_H}(M\omega)}{\partial\omega}\frac{\partial\phi_{R_H}(2M\omega)}{\partial r'_k}\ldots$$

$$
\begin{aligned}
& - \left(A_H^2(\omega) + A_H(\omega) B_H(\omega) \sin\phi_{Z_H}(M\omega)\right) \frac{\partial^2 \phi_{Z_H}(M\omega)}{\partial\omega\partial r'_k} \ldots \\
& - \sin\phi_{Z_H}(M\omega) \left[B_H(\omega) \frac{\partial A_H(\omega)}{\partial\omega} - A_H(\omega) \frac{\partial B_H(\omega)}{\partial\omega}\right] \frac{\partial \phi_{R_H}(2M\omega)}{\partial r'_k} \\
\frac{\partial |H_H(\omega)|^2}{\partial u'_k} T_H(\omega) + |H_H(\omega)|^2 \frac{\partial T_H(\omega)}{\partial u'_k} = & - \left(2A_H(\omega) + B_H(\omega) \sin\phi_{Z_H}(M\omega)\right) \frac{\partial \phi_{Z_H}(M\omega)}{\partial\omega} \frac{\partial A_H(\omega)}{\partial u'_k} \ldots \\
& + \cos\phi_{Z_H}(M\omega) \left[B_H(\omega) \frac{\partial^2 A_H(\omega)}{\partial\omega\partial u'_k} - \frac{\partial B_H(\omega)}{\partial\omega} \frac{\partial A_H(\omega)}{\partial u'_k}\right] \\
\frac{\partial |H_H(\omega)|^2}{\partial v'_k} T_H(\omega) + |H_H(\omega)|^2 \frac{\partial T_H(\omega)}{\partial v'_k} = & - A_H(\omega) \sin\phi_{Z_H}(M\omega) \frac{\partial \phi_{Z_H}(M\omega)}{\partial\omega} \frac{\partial B_H(\omega)}{\partial v'_k} \ldots \\
& + \cos\phi_{Z_H}(M\omega) \left[\frac{\partial A_H(\omega)}{\partial\omega} \frac{\partial B_H(\omega)}{\partial v'_k} - A_H(\omega) \frac{\partial^2 B_H(\omega)}{\partial\omega\partial v'_k}\right]
\end{aligned}
$$

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
ctol=7.5e-05 % Tolerance on constraints
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 8.1.5. Figure 10.119 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 = [  -0.5737808282,  -0.1358103643,  -0.0532675380,  -0.0211130145, ...
        -0.0087769205 ]';


epsilon2 = [ -1,  1,  1,  1, ...
              1 ];
```

and FIR masking filter polynomials:

```
u2 = [  -0.0008974101,  -0.0025340508,  -0.0071186853,  -0.0127944692, ...
        -0.0309461597,  -0.0343350559,  -0.0517868148,  -0.0570167861, ...
         0.4398532317 ]';
```

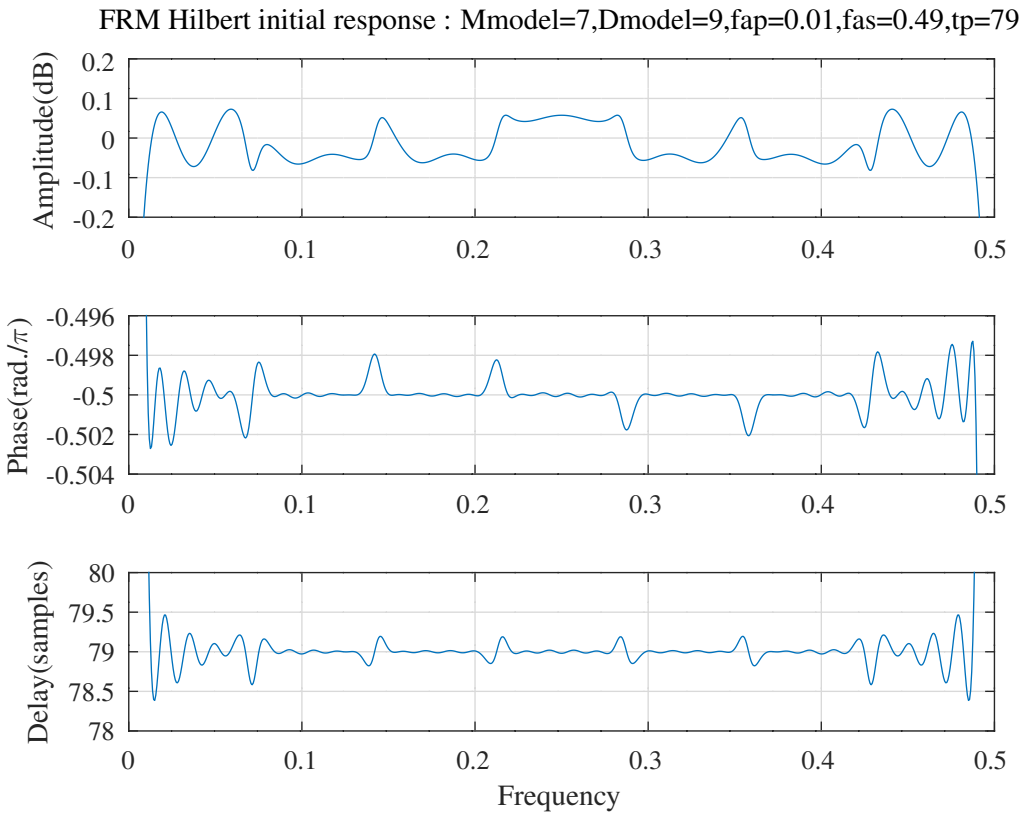



Figure 10.119: FRM Hilbert filter : initial response

```
v2 = [   0.0065334491,   0.0043793157,   0.0072209533,   0.0021032202, ...
        -0.0078549528,  -0.0311872663,  -0.0808294982,  -0.3143674103 ]';
```

Figure 10.120 shows the response of the FRM Hilbert filter after SOCP and PCLS optimisation.

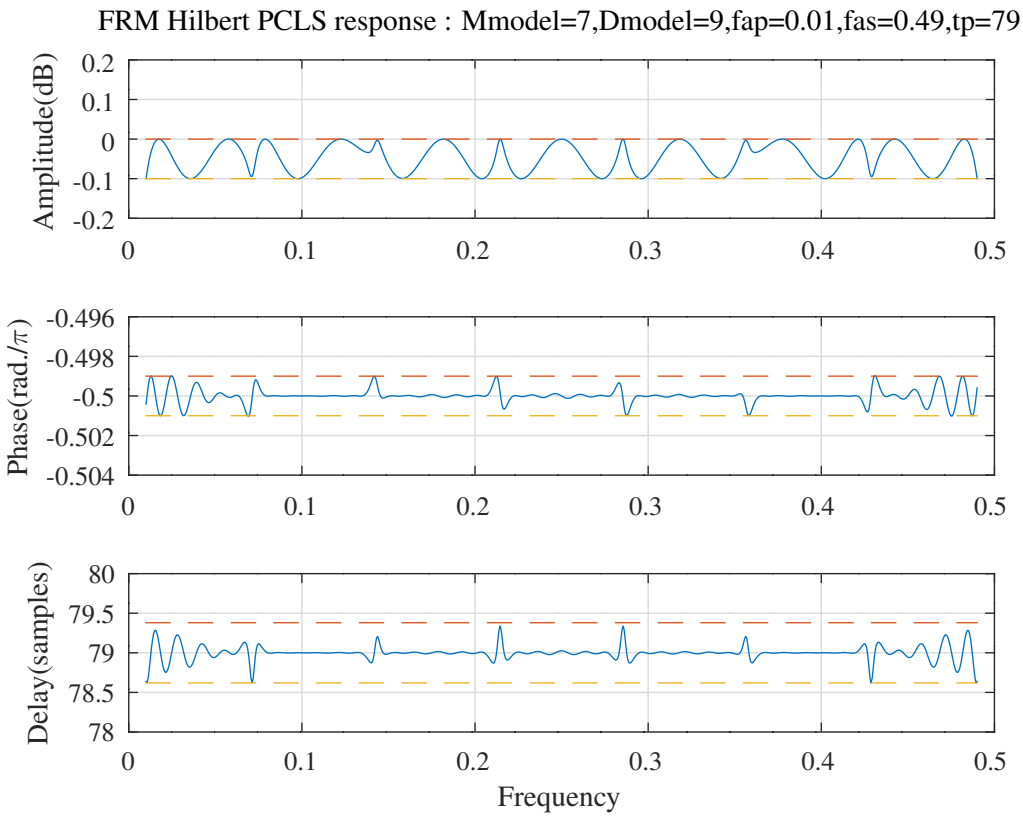



Figure 10.120: FRM Hilbert filter : response after SOCP and PCLS optimisation

# Part III

# Design of IIR filters with integer coefficients

This part describes the results of my experiments in optimising the response of digital filters with integer coefficients expressed in *signed-digit* form. A coefficient approximated with a small number of signed-digits can be implemented with *shift-and-add* operations. Chapter 11 describes the conversion of coefficients to signed-digit form and two heuristics for allocating the number of signed-digits used by each coefficient. The remaining chapters in this part consider methods of searching for the signed-digit approximations to the coefficients that give an acceptable filter response.

# Chapter 11

# Signed-digit representation of filter coefficients

*Hwang* [97, 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:
>
> $$\Sigma_r = \{-\alpha, \ldots, -1, 0, 1, \ldots, \alpha\}$$
>
> where the maximum digit magnitude, $\alpha$, 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 $\Sigma_r$, 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 $\alpha$ should be made tighter as
>
> $$\lceil \frac{r+1}{2} \rceil \leq \alpha \leq r - 1$$

*Parhi* [98, Section 13.6] lists the following properties of the *canonical* binary signed-digit (CSD) representation of a binary signed-digit number $A = a_{W-1}a_{W-2}\cdots a_1 a_0$ 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 miniumum possible number of non-zero bits, thus the name *canonic*
- the CSD representation of a number is unique
- CSD numbers cover the range $\left(\frac{-4}{3}, \frac{4}{3}\right)$, 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 $\frac{W}{3} + \frac{1}{9} + \mathcal{O}\left(2^{-W}\right)$. Hence, on average, the CSD representation contains about two-thirds the number of non-zero bits of the equivalent two's complement representation.

*Parhi* [98, 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 11.1.

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 11.1** Conversion of 2's complement numbers to the canonical signed-digit representation (*Parhi* [98, 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 $A = a_{W-1}a_{W-2}\cdots a_1 a_0$.

$\hat{a}_{-1} = 0$
$\gamma_{-1} = 0$
$\hat{a}_W = \hat{a}_{W-1}$
**for** $k = 0, \ldots, W-1$ **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**

## 11.1 Lim's method for allocating signed-digits to filter coefficients

*Lim et al.* [224] 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 [224, 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:

$$Q_E \approx 0.36\log_2 \lfloor \frac{n_1}{n_2} \rfloor$$

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 $\Delta x_k$ 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.

## 11.2 Ito's method for allocating signed-digits to filter coefficients

*Ito et al.* [178] describe a heuristic for allocating signed-digits to the coefficients of an FIR filter. Suppose $\boldsymbol{x} = \{x_1, \ldots, x_K\}$ 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 $\boldsymbol{x}$. $c(\boldsymbol{x})$ is a cost function for the filter design and $\boldsymbol{e_k}$ 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 11.2 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 11.2** Modified signed-digit allocation heuristic of *Ito et al.* [178]

Initialise $n_k$:
  **for** $k = 1, \ldots, K$ **do**
    **if** $|x_k| < \epsilon$ **then**
      $n_k = 0$
    **else**
      $n_k = 2N$
    **end if**
  **end for**
Allocate $n_k$:
  **for** $r = 2R, \ldots, R$ **do**
    **for** $k = 1, \ldots, K$ **do**
      **if** $n_k \geq 1$ **then**
        $c_k^U = c(\boldsymbol{x} + (\lceil x_k \rceil - x_k)\, \boldsymbol{e_k})$
        $c_k^L = c(\boldsymbol{x} - (x_k - \lfloor x_k \rfloor)\, \boldsymbol{e_k})$
        $c_k = \min\{c_k^L, c_k^U\}$
      **end if**
    **end for**
    $c_{k_{min}} = \min\{c_k\}$
    $n_{k_{min}} - = 1$
  **end for**

## 11.3 Signed-digit allocation of the coefficients of a Schur one-multiplier lattice filter

This section compares the performance of the SQP optimised Schur one-multiplier bandpass filter of Section 10.3.6 with coefficients that are exact, rounded, approximated by two signed-digits and approximated by an average of two and three 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.

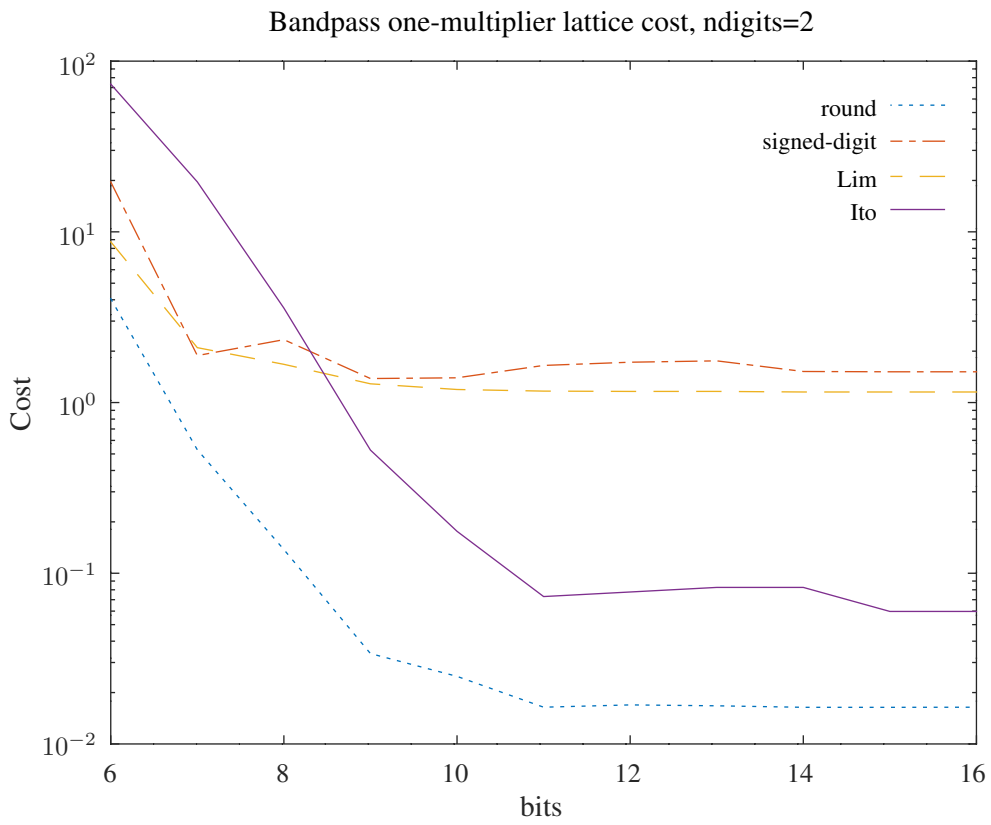



Figure 11.1: Comparison of the cost function for a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

Figure 11.1 compares the cost for each allocation method, Figure 11.2 compares the maximum stopband response in the frequency range $[0.26, 0.5)$, Figure 11.3 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 11.4 compares the estimated filter noise-gain using word-sizes of 6 to 16 bits. Figure 11.5 compares the responses for 10-bit coefficients and Figure 11.6 compares the passband responses for 10-bit 2-signed-digit coefficients.

Figures 11.7, 11.8, 11.9, 11.10, 11.11 and 11.12 show the corresponding results for an allocation of 3 signed-digits to each non-zero 10-bit 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 5.27). 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.

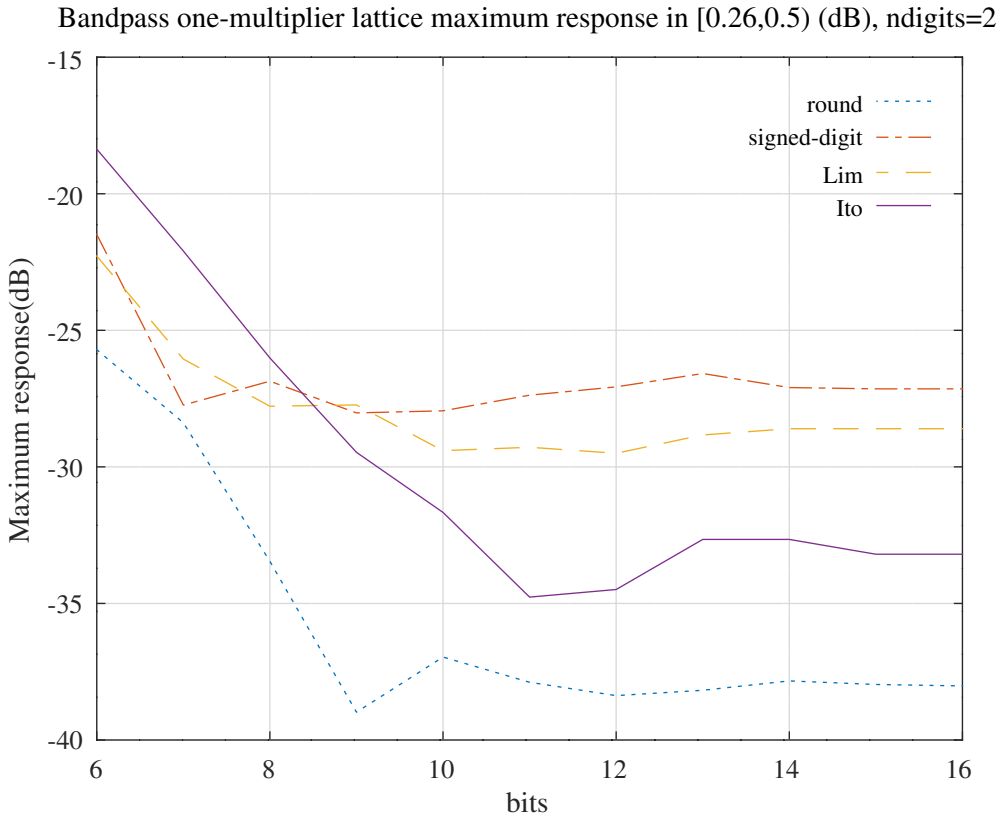



Figure 11.2: Comparison of the maximum stopband response in the region [0.26,0.5) for a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

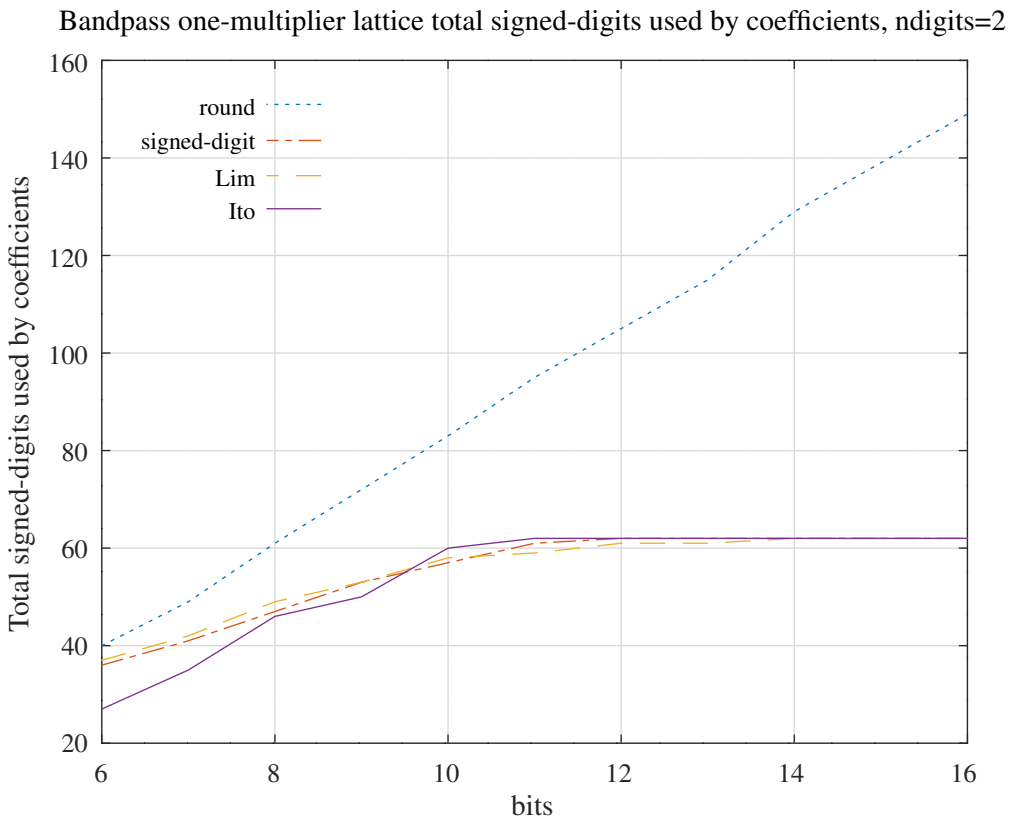



Figure 11.3: Comparison of the total number of signed-digits required to implement the coefficients of a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

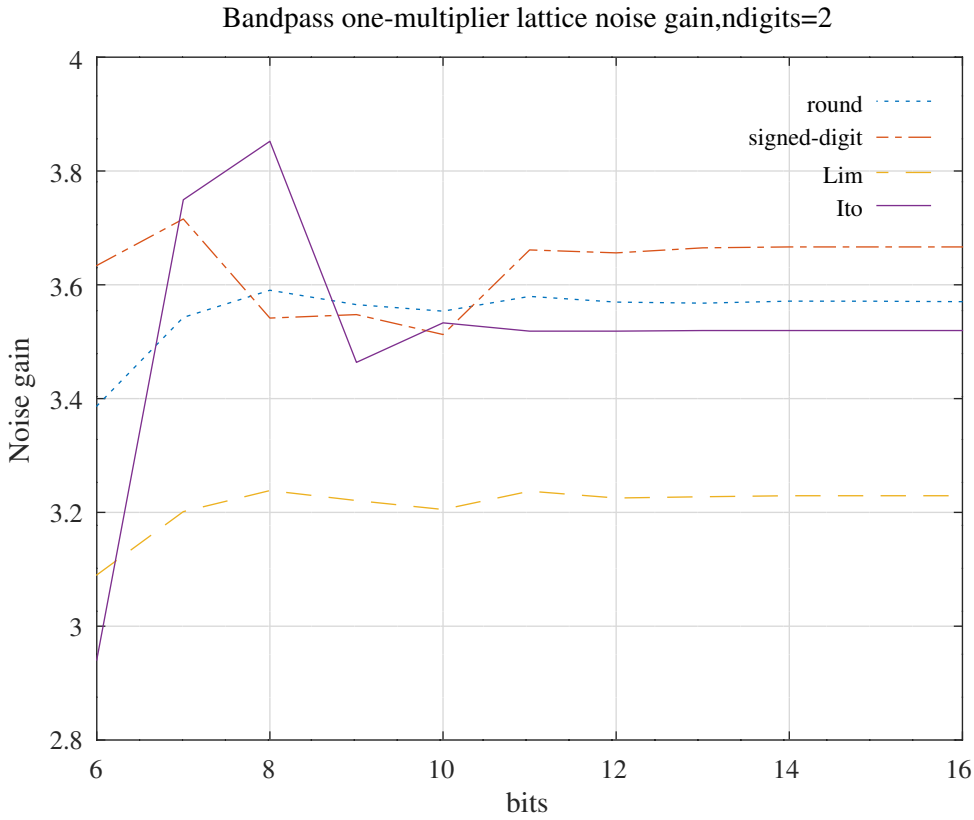



Figure 11.4: Comparison of the estimated noise gain of a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*.

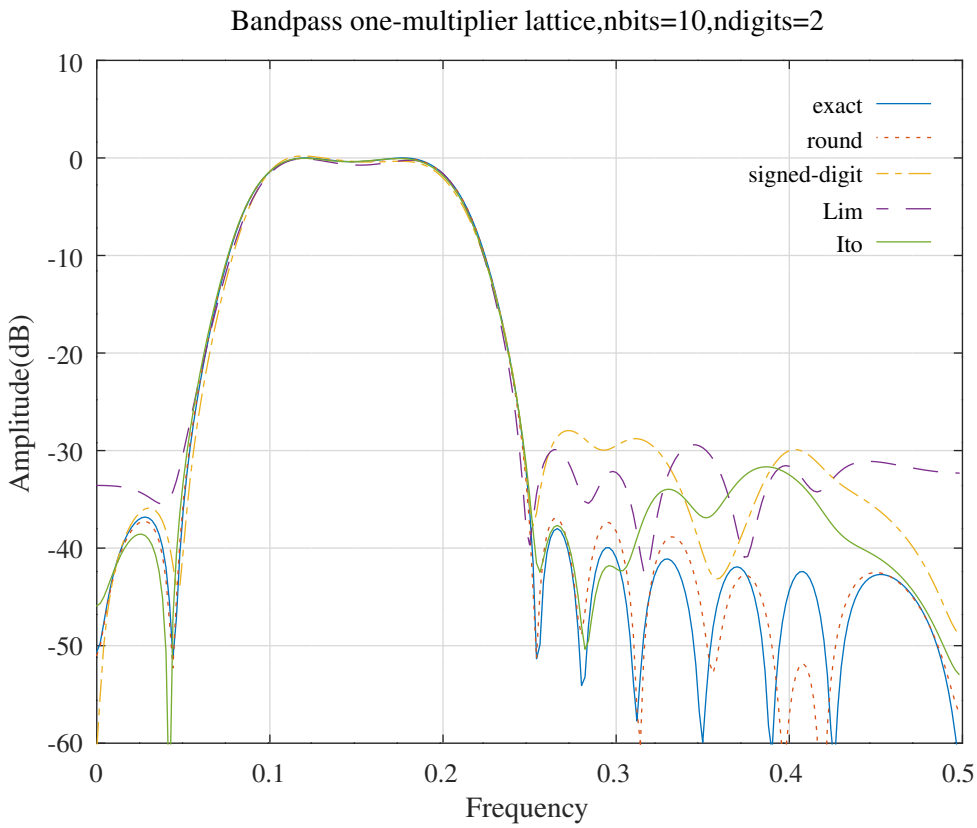



Figure 11.5: Comparison of the responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

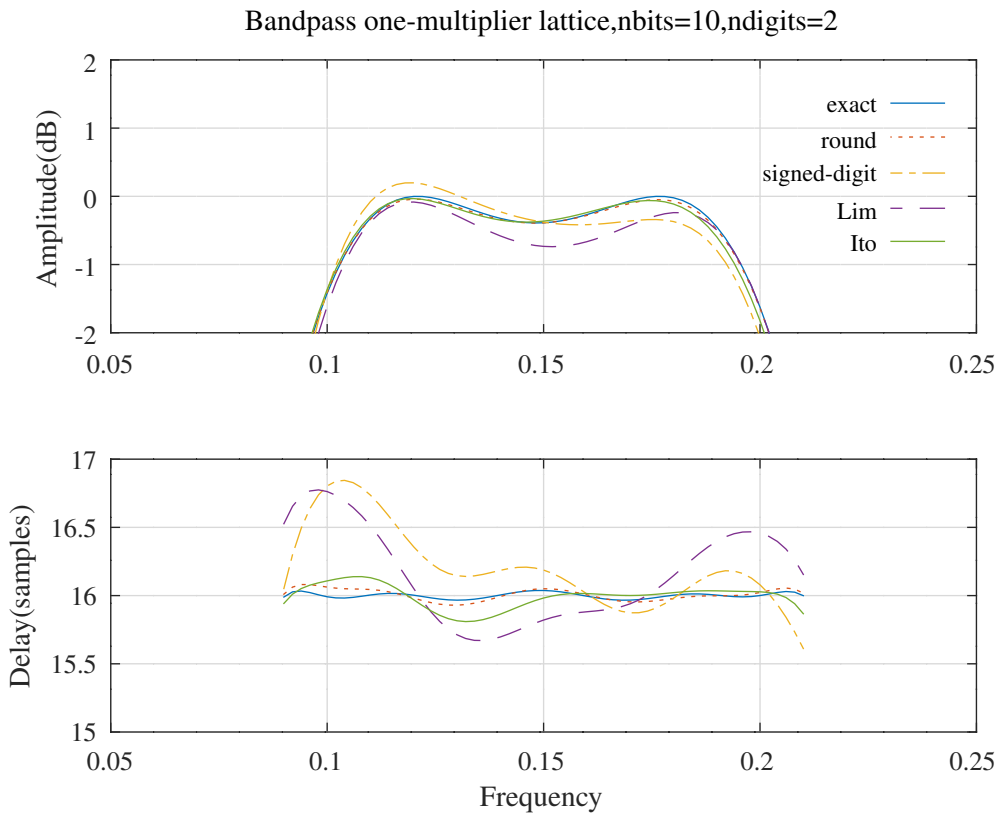



Figure 11.6: Comparison of the passband responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

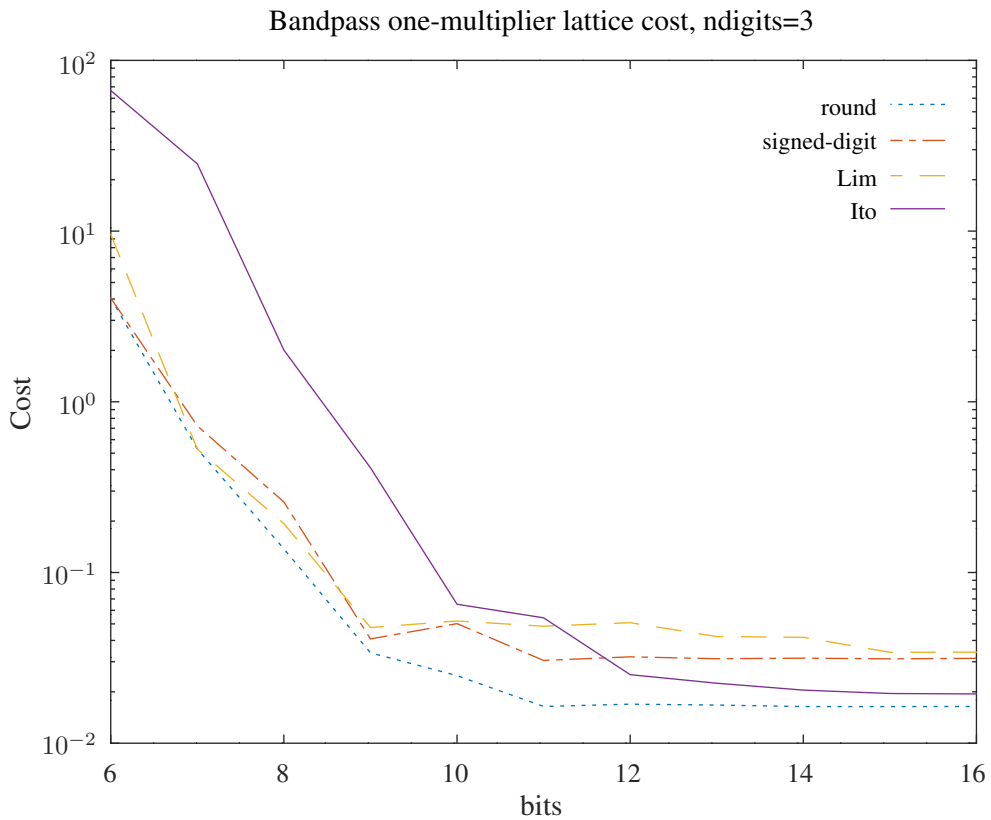



Figure 11.7: Comparison of the cost function for a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

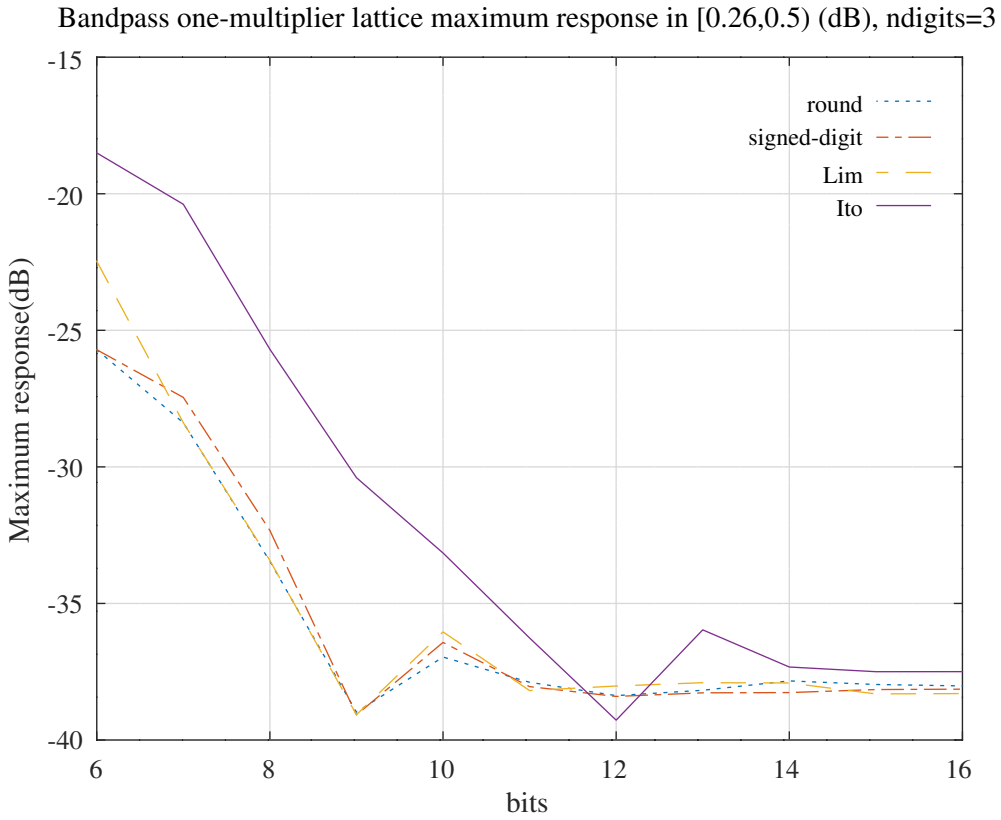



Figure 11.8: Comparison of the maximum stopband response in the region [0.26,0.5) for a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

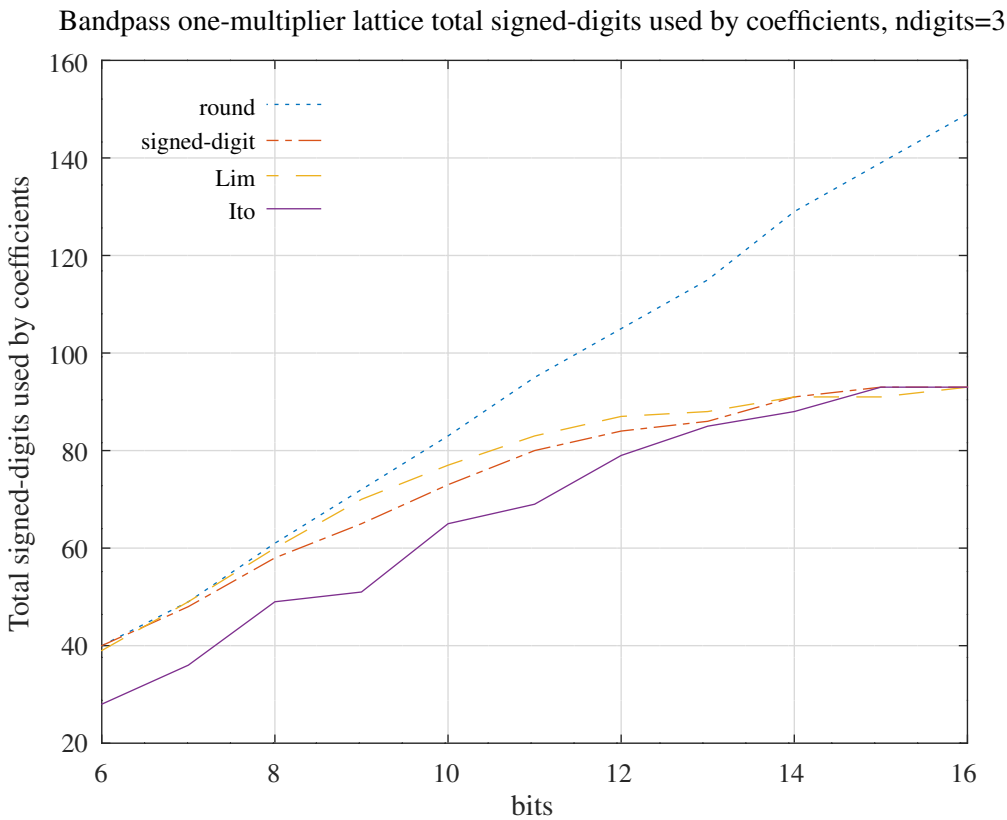



Figure 11.9: Comparison of the total number of signed-digits required to implement the coefficients of a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

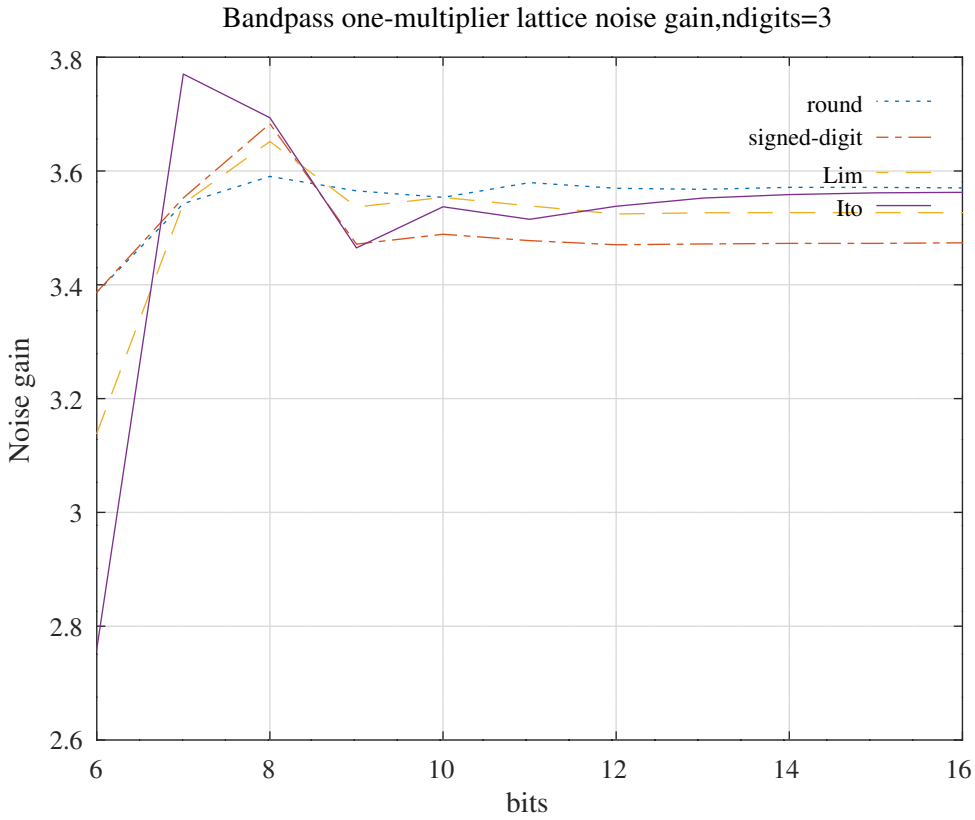



Figure 11.10: Comparison of the estimated noise gain of a Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*.

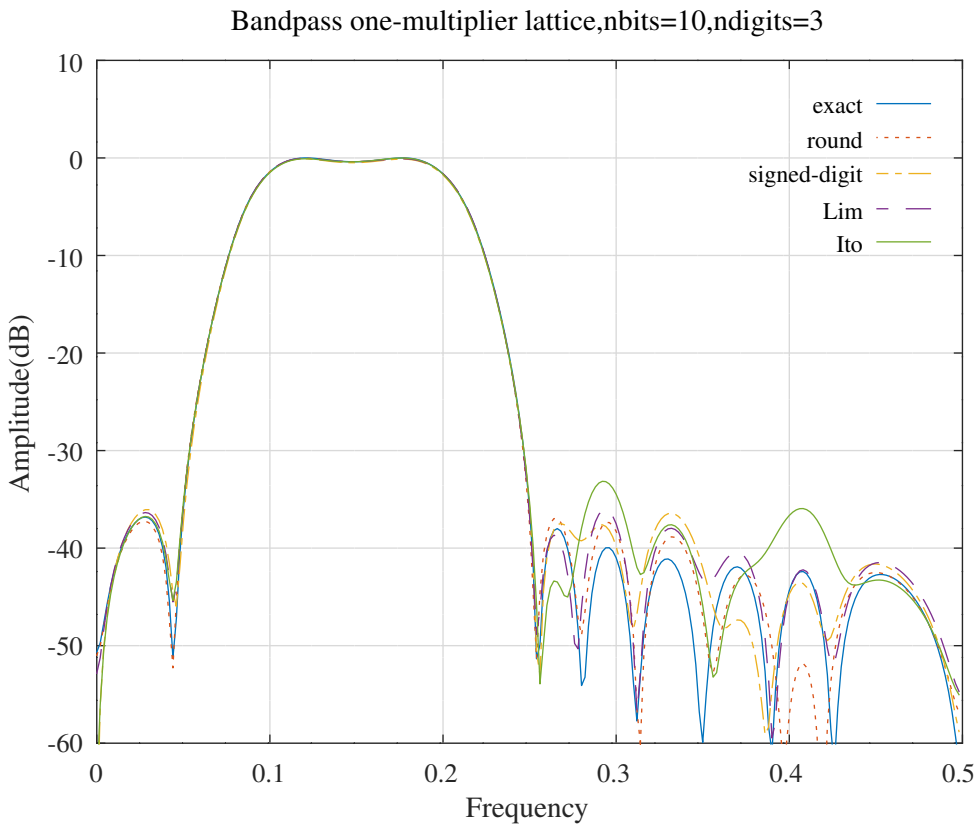



Figure 11.11: Comparison of the responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

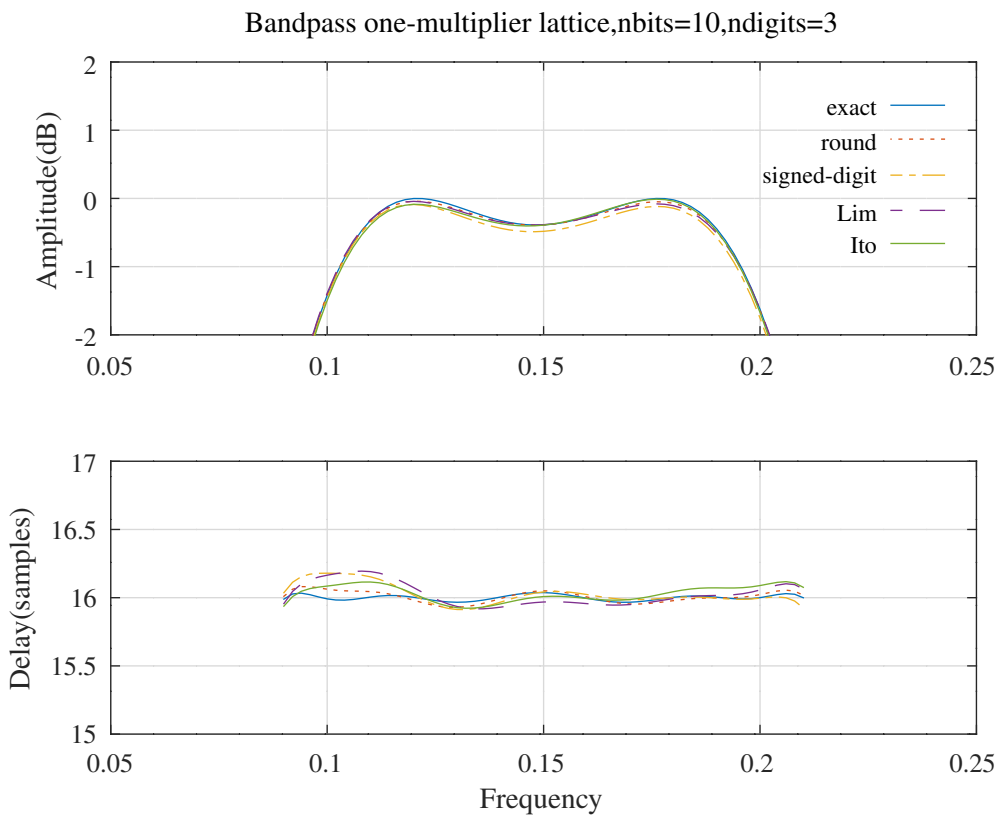



Figure 11.12: Comparison of the passband responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

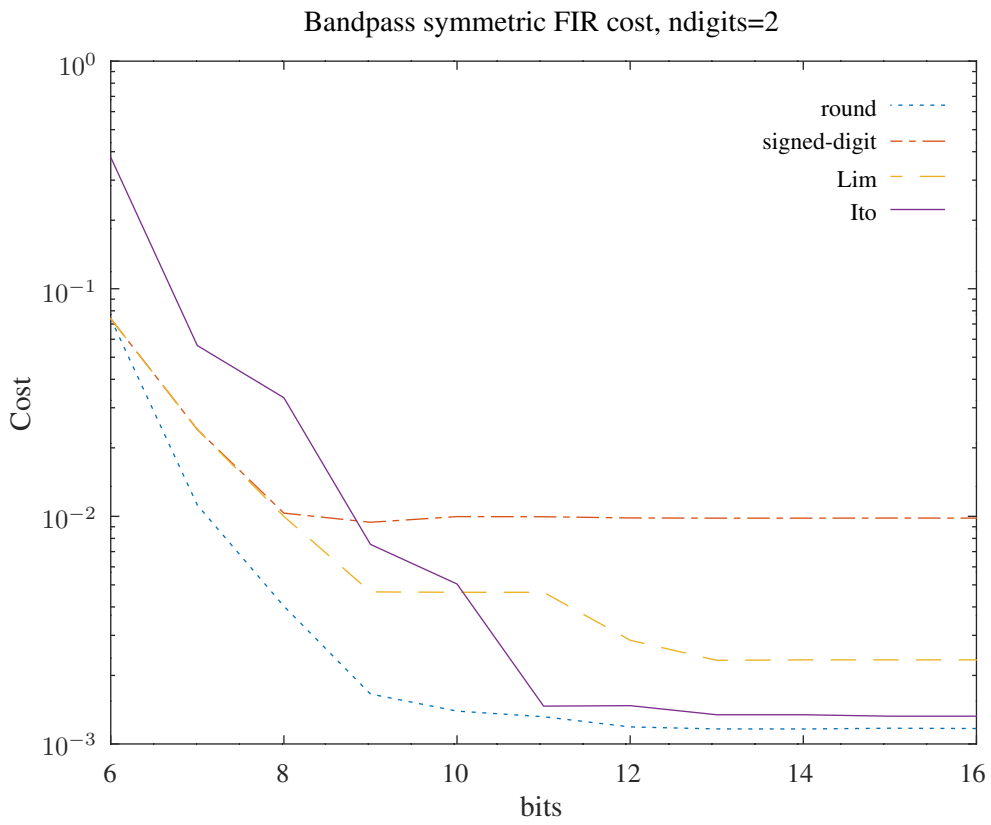



Figure 11.13: Comparison of the cost function for a direct-form even-order symmetric FIR bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

## 11.4 Signed-digit allocation of the coefficients of a symmetric FIR band-pass filter

Appendix N.3 describes the optimisation of even-order symmetric direct-form FIR filters with constraints on the amplitude response by the method of Lagrange multipliers. This section compares the performance of a direct-form even-order symmetric FIR bandpass with coefficients that are exact, rounded, approximated by two signed-digits and approximated by an average of two and three signed-digits allocated by the *Lim* and *Ito* heuristics. The comparison is implemented in the Octave script *directFIRsymmetric_bandpass_allocsd_test.m*. That script calls the Octave function *directFIRsymmetric_slb* to design a symmetric FIR band-pass filter polynomial of order 30 with 16 distinct coefficients. The filter amplitude passband is $[0.1, 0.2]$, the lower amplitude stopband is $[0, 0.05]$ and the upper amplitude stopband is $[0.25, 0.5)$. The heuristic of *Lim et al.* is implemented in the Octave function *directFIRsymmetric_allocsd_Lim*. That function allocates signed digits according to the gradients of the *un-weighted* sum of the magnitude errors. The heuristic of *Ito et al.* is implemented in the Octave function *directFIRsymmetric_allocsd_Ito.m*.

Figure 11.13 compares the cost for each allocation method, Figure 11.14 compares the maximum stopband response in the frequency range $[0.26, 0.5)$, Figure 11.15 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 11.16 compares the responses for 10-bit 2-signed-digit coefficients. Figures 11.17, 11.18, 11.19 and 11.20 show the corresponding results for an allocation of 3 signed-digits to each non-zero 10-bit coefficient.

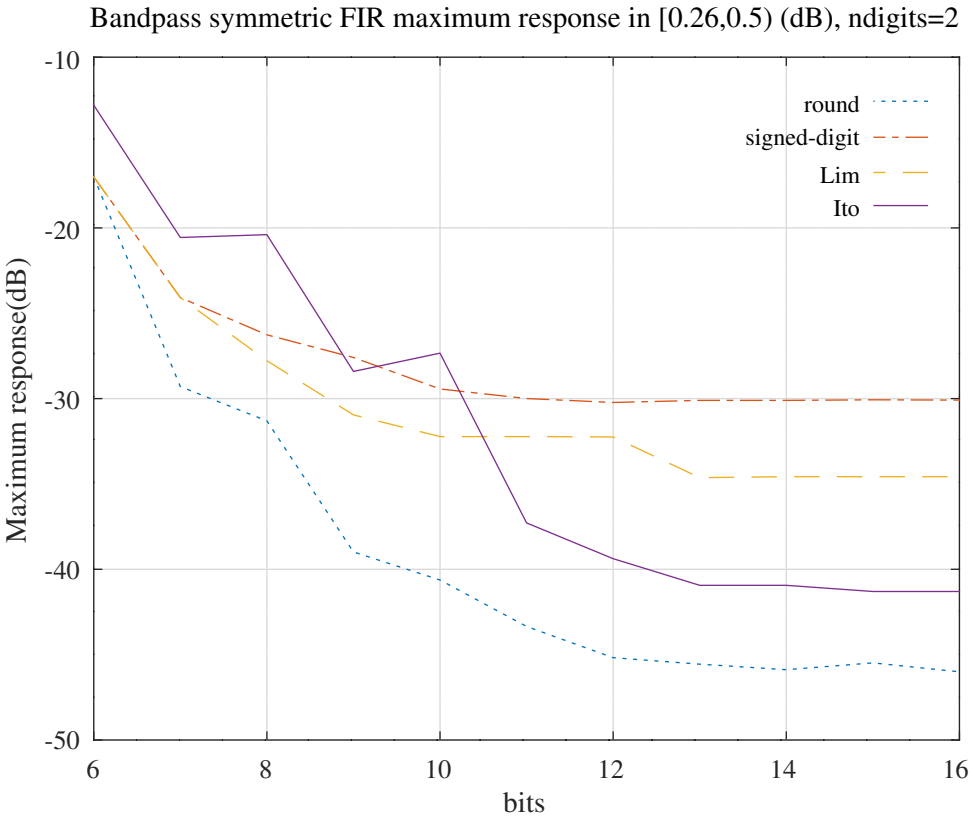



Figure 11.14: Comparison of the maximum stopband response in the region [0.26,0.5) for a direct-form even-order symmetric FIR bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

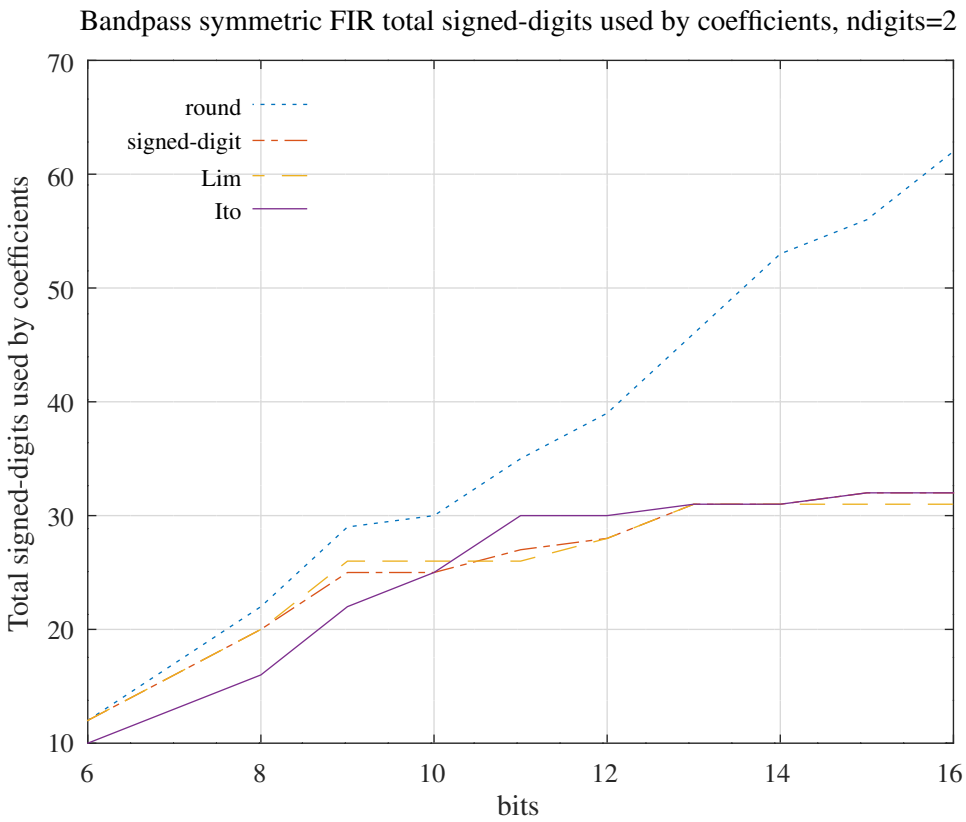



Figure 11.15: Comparison of the total number of signed-digits required to implement the coefficients of a direct-form even-order symmetric FIR bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

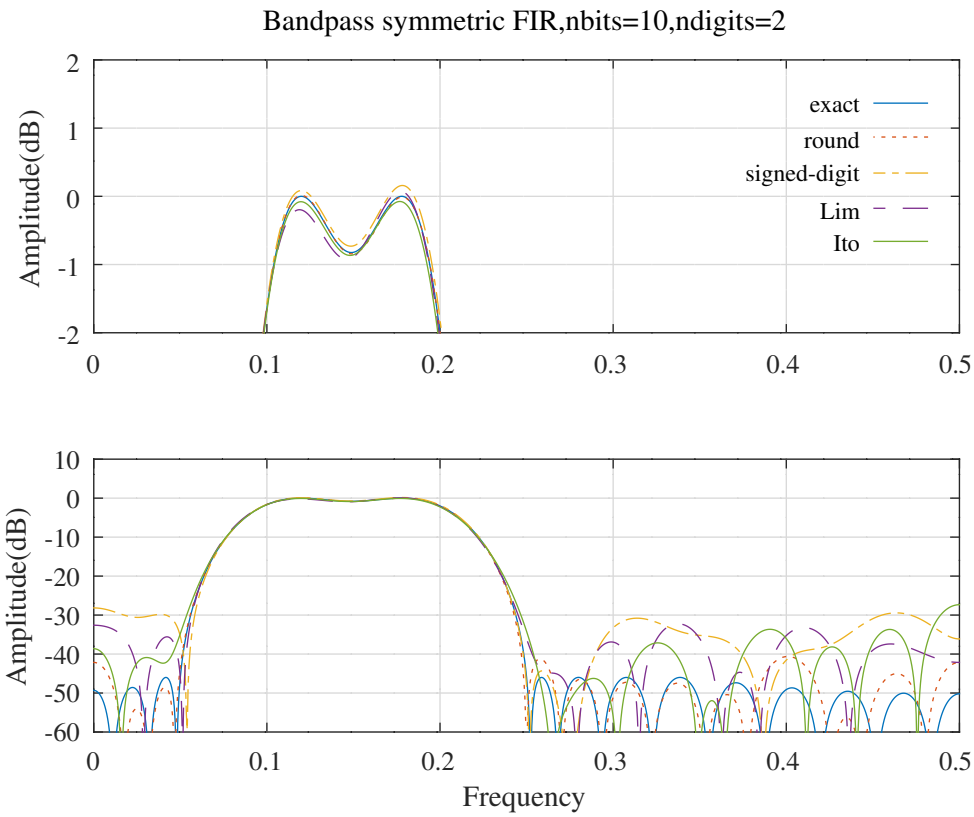



Figure 11.16: Comparison of the responses for a direct-form even-order symmetric FIR bandpass filter with 10-bit integer coefficients found by rounding, approximation by two signed-digits and approximation by an average of two signed digits using the heuristics of *Lim et al.* and *Ito et al.*

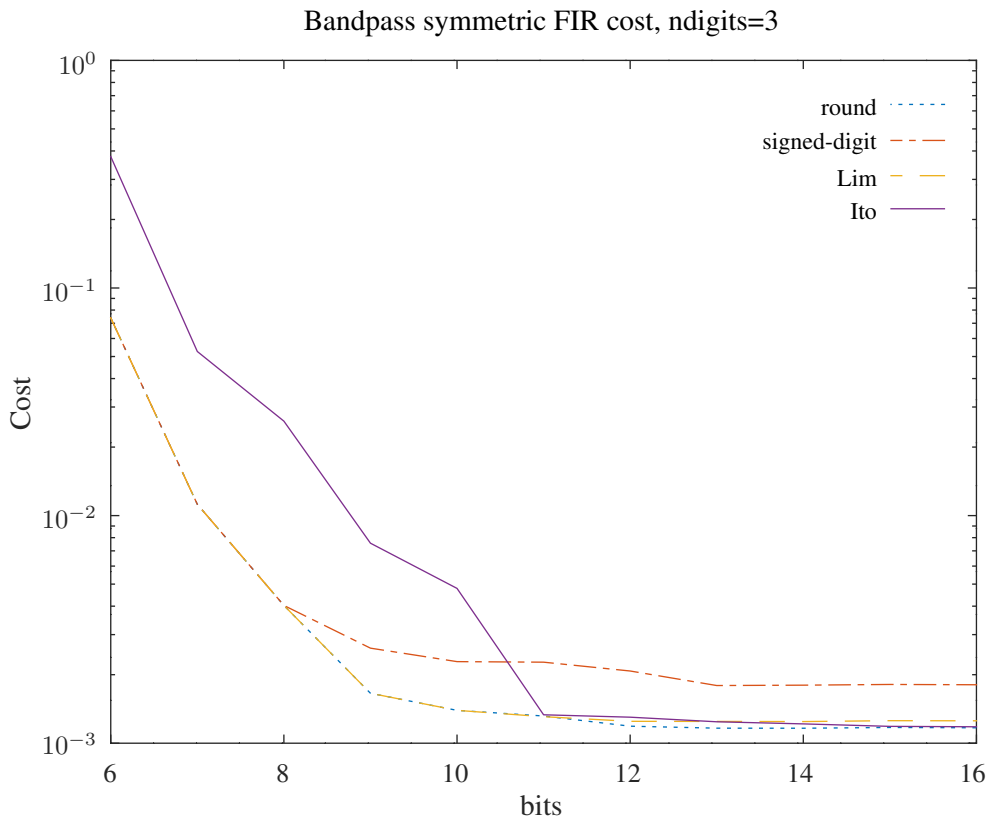



Figure 11.17: Comparison of the cost function for a direct-form even-order symmetric FIR bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

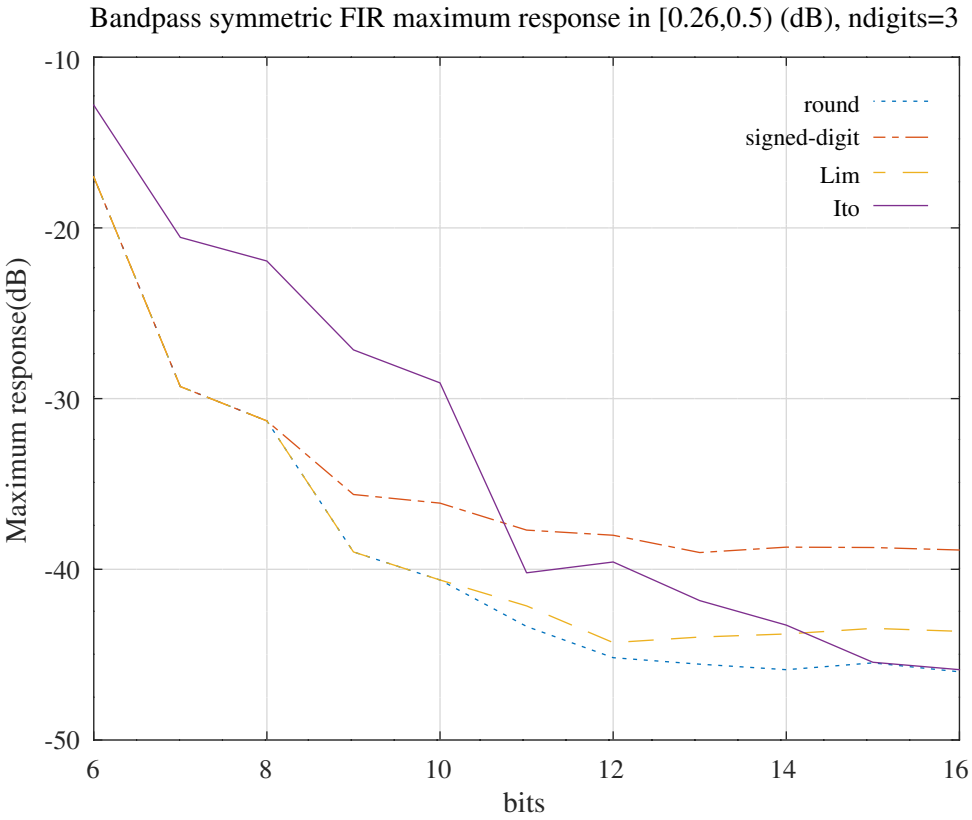



Figure 11.18: Comparison of the maximum stopband response in the region [0.26,0.5) for a direct-form even-order symmetric FIR bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

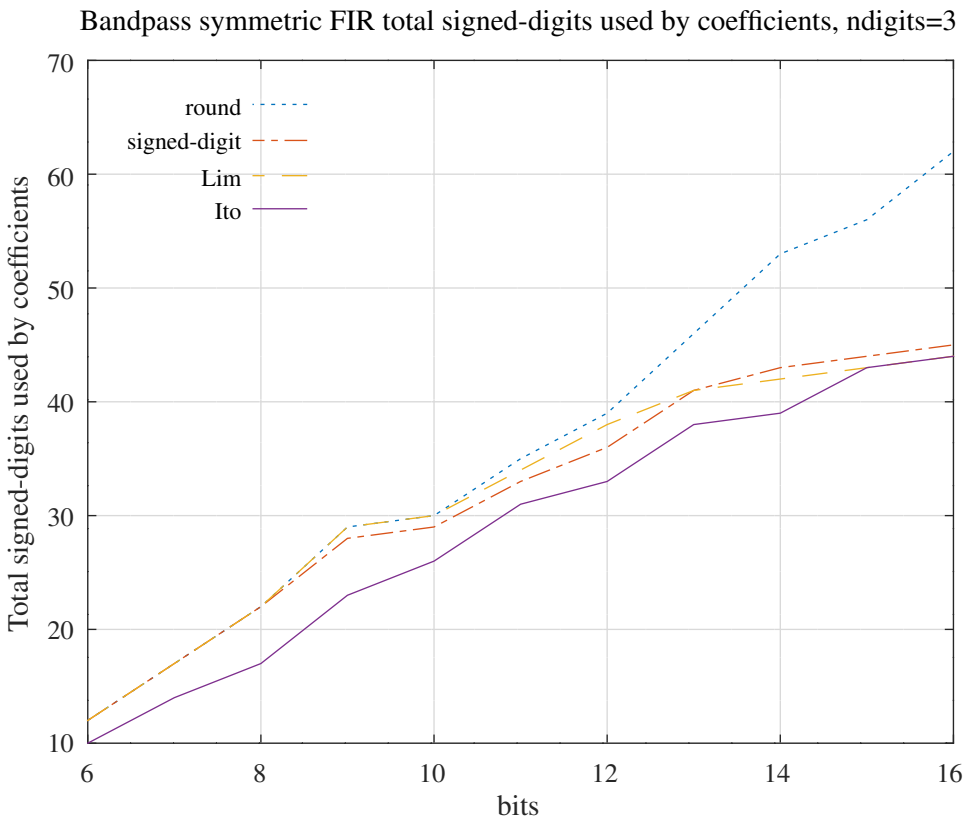



Figure 11.19: Comparison of the total number of signed-digits required to implement the coefficients of a direct-form even-order symmetric FIR bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

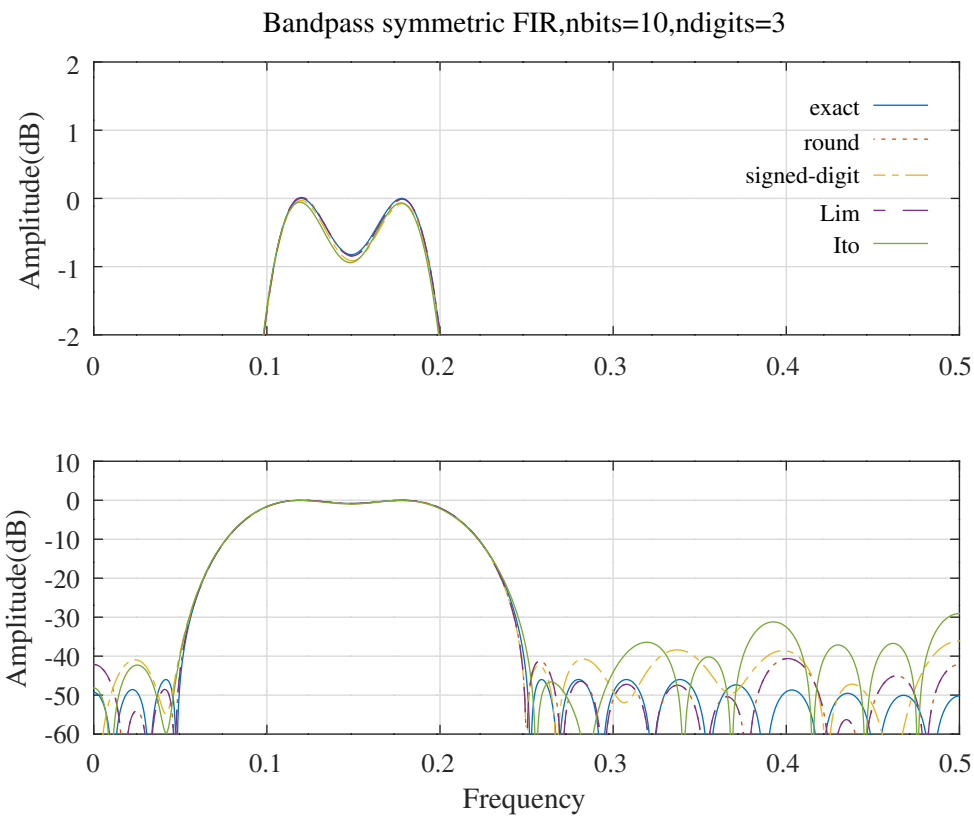



Figure 11.20: Comparison of the responses for a direct-form even-order symmetric FIR bandpass filter with 10-bit integer coefficients found by rounding, approximation by three signed-digits and approximation by an average of three signed digits using the heuristics of *Lim et al.* and *Ito et al.*

# Chapter 12

# Exhaustive search for integer and signed-digit filter coefficients

The Octave script *exhaustive_schurOneMlattice_bandpass_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 10.3.6. 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 a month of processing time on my PC (which has an Intel i7-7700K CPU with 4 cores running at 4.2GHz).

# Chapter 13

# Searching for integer and signed-digit filter coefficients with the *bit-flipping* algorithm

*Krukowski and Kale* [7], describe a "bit-flipping" algorithm for fixed-point filter design, shown in flow-graph form in Figure 13.1 (from [7, 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*.

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 IIR filter is that designed by the Octave script *iir_sqp_slb_bandpass_test.m* with:

```
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=35             % Amplitude stopband attenuation
```

For completeness, I also apply the bit-flipping algorithm to the coefficients of a minimum-phase Schur FIR lattice band-pass filter and a direct-form linear-phase symmetric FIR band-pass filter having the desired frequency bands. The parallel allpass one-multiplier Schur lattice filter has amplitude response constraints of $dBap = 2$ and $dBas = 53$. The Schur FIR lattice has amplitude response constraints of $dBap = 3$ and $dBas = 25$. The direct-form symmetric FIR filter has amplitude response constraints of $dBap = 2$ and $dBas = 46$.

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 8-bit rounded, 8-bit rounding optimised with the bit-flipping algorithm, 8-bit 2-signed-digits and 8-bit 2-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 6 of 8 bits and uses a mask size of 3 bits.

## 13.1 Bit-flipping search for the signed-digit coefficients of a direct-form bandpass IIR filter

The Octave script *bitflip_directIIR_bandpass_test.m* applies the bit-flipping algorithm to a direct-form IIR bandpass filter with the following filter transfer function polynomials:

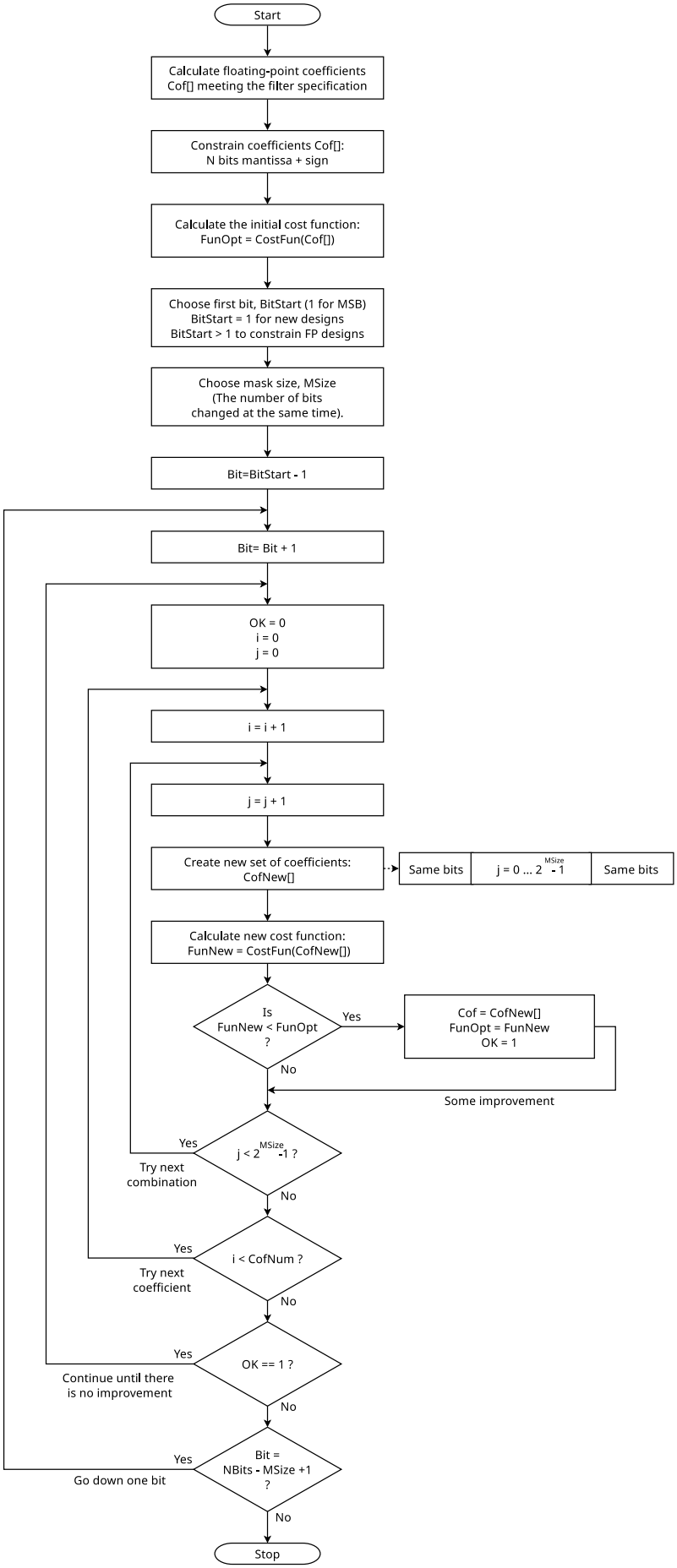



Figure 13.1: The bit-flipping algorithm of *Krukowski and Kale* [7, Figure 2]

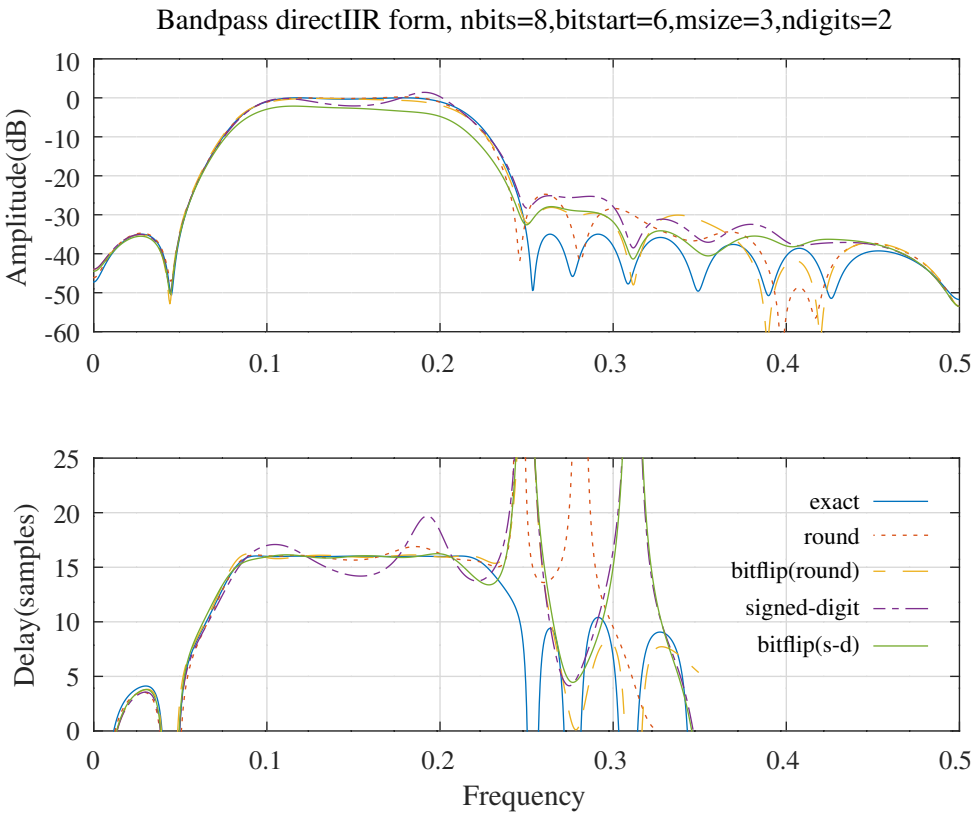



Figure 13.2: Amplitude and group-delay responses of a direct-form IIR band-pass filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

```
n_ex = [   0.0119781334,   0.0053372581,   0.0225953299,   0.0226057078, ...
           0.0476583039,   0.0341407987,   0.0294625008,   0.0000430755, ...
          -0.0024678492,  -0.0310804121,  -0.0679675076,  -0.1023386019, ...
          -0.0694596255,   0.0369652616,   0.1362499644,   0.1556848981, ...
           0.0622425193,  -0.0405480060,  -0.0988351108,  -0.0710911985, ...
          -0.0331193158 ];

d_ex = [   1.0000000000,   0.0000000000,   1.7060993914,   0.0000000000, ...
           1.9330569490,   0.0000000000,   1.9381867906,   0.0000000000, ...
           1.7150005810,   0.0000000000,   1.2590350700,   0.0000000000, ...
           0.8059003363,   0.0000000000,   0.4345691412,   0.0000000000, ...
           0.1970293344,   0.0000000000,   0.0648657750,   0.0000000000, ...
           0.0145991810 ];
```

There is no attempt to ensure that the bit-flipped filter transfer function is stable. The filter responses are shown in Figure 13.2.

Table 13.1 compares the cost result for each test.

| Band-pass direct form IIR | Cost |
|---|---|
| Exact | 0.8898 |
| 8-bit rounded | 4.3285 |
| 8-bit rounded with bitflipping | 2.1902 |
| 8-bit 2-signed-digit | 14.2590 |
| 8-bit 2-signed-digit with bitflipping | 3.4917 |

Table 13.1: Summary of the cost results for the direct form IIR bandpass filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

The *bitflip* optimised 8-bit 2 signed-digit direct-form IIR transfer function coefficients are:

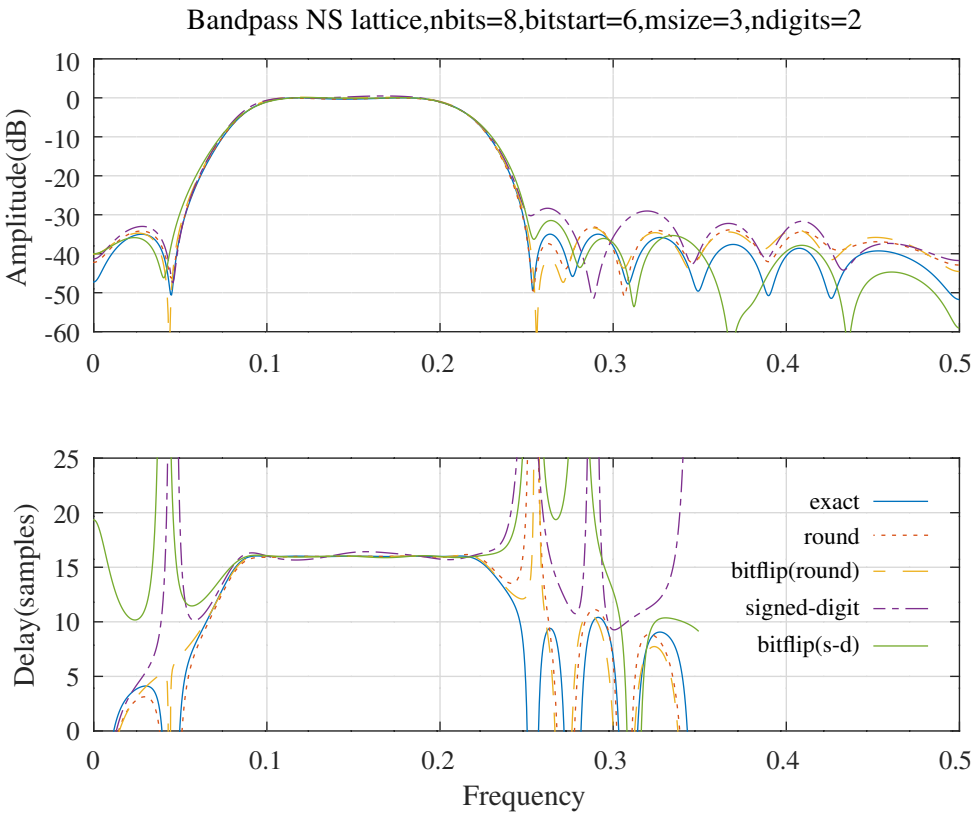



Figure 13.3: Amplitude and group-delay responses of a band-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

```
n_bfsd = [        2,        1,        3,        3, ...
                  6,        4,        4,        0, ...
                  0,       -4,       -9,      -12, ...
                 -9,        5,       17,       20, ...
                  8,       -5,      -12,       -9, ...
                 -4 ]/128;


d_bfsd = [      160,        2,      240,        9, ...
                248,       16,      248,        4, ...
                224,        0,      160,        6, ...
                 96,        5,       56,        0, ...
                 24,        3,        8,        0, ...
                  2 ]/128;
```

A total of 25 adders is required to implement these signed-digit coefficients.

## 13.2 Bit-flipping search for the signed-digit coefficients of a normalised-scaled lattice bandpass IIR filter

The Octave script *bitflip_schurNSlattice_bandpass_test.m* applies the bit-flipping algorithm to the example bandpass filter implemented as a Schur normalised-scaled lattice filter. The filter responses are shown in Figure 13.3.

Table 13.2 compares the cost result for each test.

The *bitflip* optimised 8-bit 2 signed-digit lattice filter coefficients are:

```
s10_bfsd = [       16,     -112,     -112,      -68, ...
                   24,       72,       56,       12, ...
```

| Band-pass Schur normalised-scaled | Cost |
|---|---|
| Exact | 0.8898 |
| 8-bit rounded | 1.4271 |
| 8-bit rounded with bitflipping | 1.1480 |
| 8-bit 2-signed-digit | 3.4061 |
| 8-bit 2-signed-digit with bitflipping | 1.3342 |

Table 13.2: Summary of the cost results for the bandpass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

```
                  -17,      -12,       -2,       -3, ...
                  -10,       -9,        0,        5, ...
                    3,        0,        1,        1 ]'/128;


s11_bfsd = [      124,       80,       65,      112, ...
                  127,      112,      112,      128, ...
                  124,      124,      136,      128, ...
                  136,      132,      128,      128, ...
                  128,      128,      128,       56 ]'/128;


s20_bfsd = [        0,       80,        0,       48, ...
                    0,       60,        0,       48, ...
                    0,        8,        0,       32, ...
                    0,       31,        0,       31, ...
                    0,       63,        0,        2 ]'/128;


s00_bfsd = [      128,       80,      128,      112, ...
                  128,      120,      128,      120, ...
                  128,      120,      128,      126, ...
                  128,      127,      128,      124, ...
                  128,      128,      128,      128 ]'/128;


s02_bfsd = [        0,      -96,        0,      -63, ...
                    0,      -48,        0,      -48, ...
                    0,      -40,        0,      -33, ...
                    0,      -20,        0,      -12, ...
                    0,       -5,        0,       -2 ]'/128;


s22_bfsd = [      128,       80,      128,      120, ...
                  128,      120,      128,      112, ...
                  128,      112,      128,      126, ...
                  128,      112,      128,      120, ...
                  128,      128,      128,      128 ]'/128;
```

A total of 59 adders is required to implement these signed-digit coefficients.

## 13.3 Bit-flipping search for the signed-digit coefficients of a one-multiplier lattice bandpass IIR filter

The Octave script *bitflip_schurOneMlattice_bandpass_test.m* applies the bit-flipping algorithm to the example bandpass filter implemented as a Schur one-multiplier lattice filter. Note that, in this example, the one-multiplier state scaling coefficients are not truncated.

The filter responses are shown in Figure 13.4.

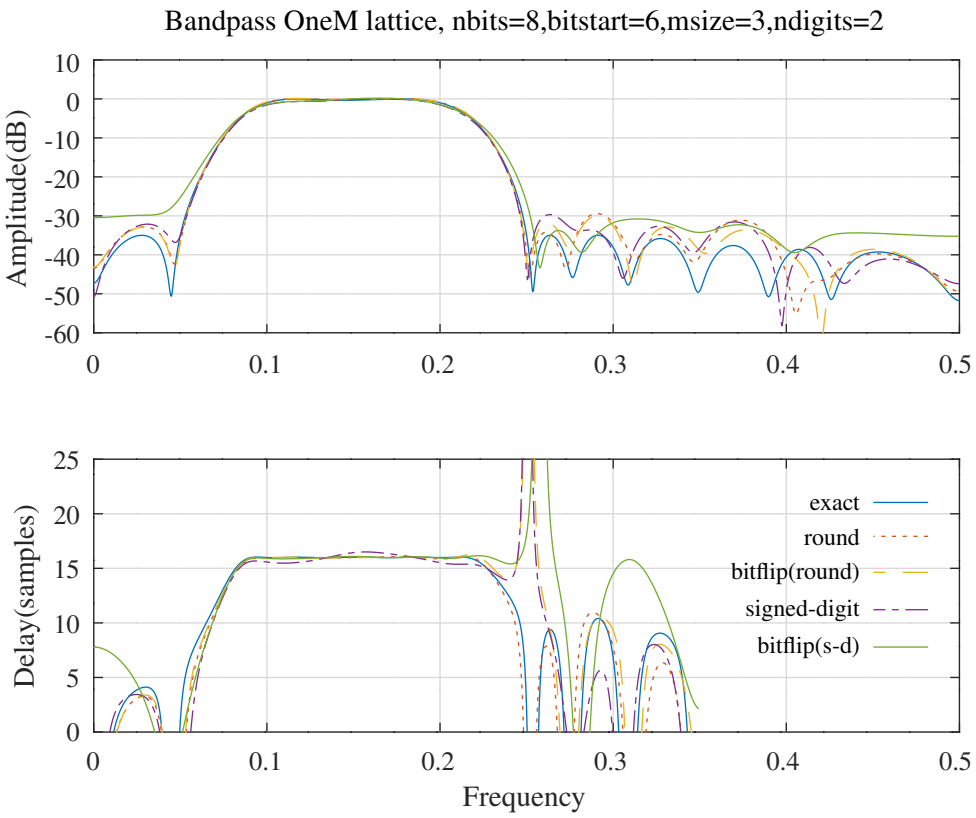



Figure 13.4: Amplitude and group-delay responses of a band-pass filter synthesised as a one-multiplier lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

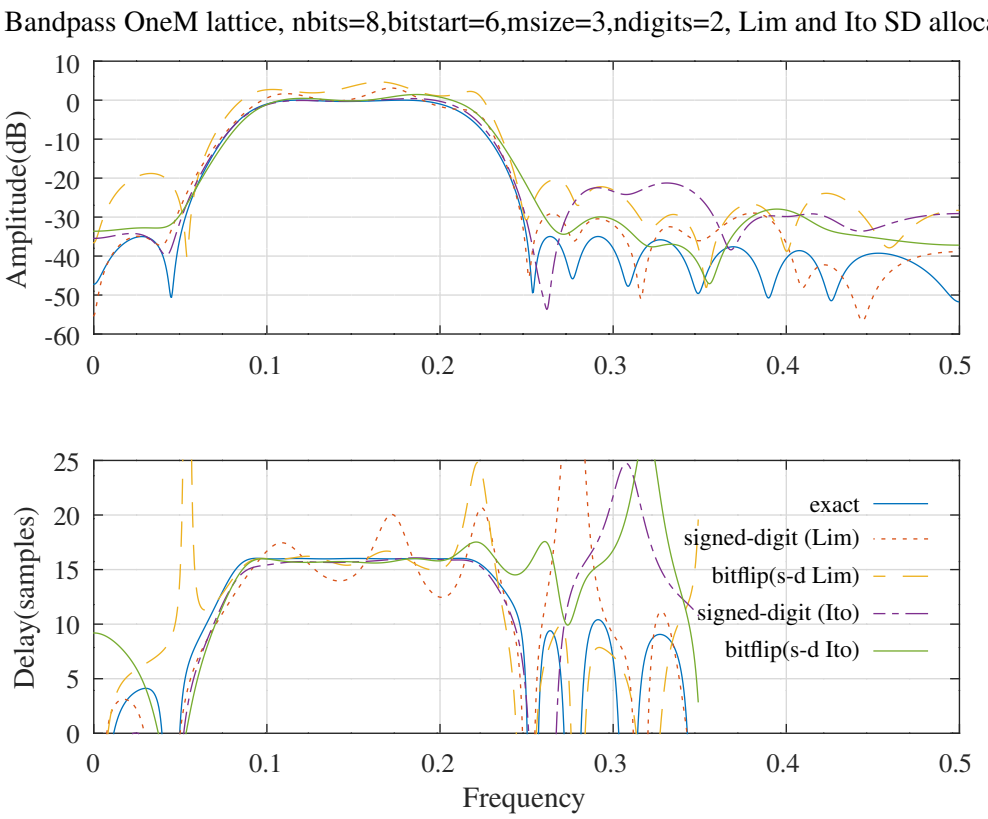



Figure 13.5: Amplitude and group-delay responses of a band-pass filter synthesised as a one-multiplier lattice filter with exact coefficients, 8-bit 2-signed-digit coefficients allocated with Lim's algorithm, 8-bit 2-signed-digit coefficients allocated with Lim's algorithm and optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients allocated with Ito's algorithm and 8-bit 2-signed-digit coefficients allocated with Ito's algorithm and optimised with the bit-flipping algorithm

| Band-pass Schur one-multiplier | Cost |
|---|---|
| Exact | 0.8898 |
| 8-bit rounded | 1.5326 |
| 8-bit rounded with bit-flipping | 1.4454 |
| 8-bit 2-signed-digit | 4.2317 |
| 8-bit 2-signed-digit with bit-flipping | 2.2054 |
| 8-bit 2-signed-digit(Lim alloc.) | 17.9008 |
| 8-bit 2-signed-digit(Lim alloc.) with bit-flipping | 8.5979 |
| 8-bit 2-signed-digit(Ito alloc.) | 5.4671 |
| 8-bit 2-signed-digit(Ito alloc.) with bit-flipping | 3.8194 |

Table 13.3: Summary of the cost results for the bandpass filter synthesised as a one-multiplier lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm. The signed digits are allocated with 2 digits each, Lim's allocation method with an average of 2-signed-digits each and Ito's allocation method with an average of 2-signed-digits each.

The filter responses for the signed-digit coefficients with 2-signed-digits, 2-signed-digits allocated with Lim's algorithm and 2-signed-digits allocated with Ito's algorithm are shown in Figure 13.5.

Table 13.3 compares the cost result for each test.

The *bitflip* optimised 8-bit, rounded, one-multiplier lattice bandpass filter coefficients are:

```
k_bf = [        0,        96,         0,        62, ...
                0,        46,         0,        52, ...
                0,        40,         0,        33, ...
                0,        20,         0,        13, ...
                0,         5,         0,         2 ]/128;


c_bf = [       10,         1,       -39,       -79, ...
              -33,        13,        29,        25, ...
                7,       -12,        -7,        -1, ...
               -2,        -7,        -5,         0, ...
                3,         2,         0,         1, ...
                2 ]/128;
```

A total of 30 adders is required to implement these rounded coefficients.

The *bitflip* optimised 8-bit, 2-signed-digit, one-multiplier lattice bandpass filter coefficients are:

```
k_bfsd = [        0,        96,         0,        60, ...
                  0,        48,         0,        48, ...
                  0,        40,         0,        32, ...
                  0,        20,         0,        12, ...
                  0,         5,         0,         2 ]/128;


c_bfsd = [       10,         1,       -40,       -80, ...
                -33,        15,        30,        24, ...
                  6,       -12,        -7,        -1, ...
                 -2,        -7,        -5,         0, ...
                  0,         1,         0,         1, ...
                  1 ]/128;
```

A total of 20 adders is required to implement these signed-digit coefficients.

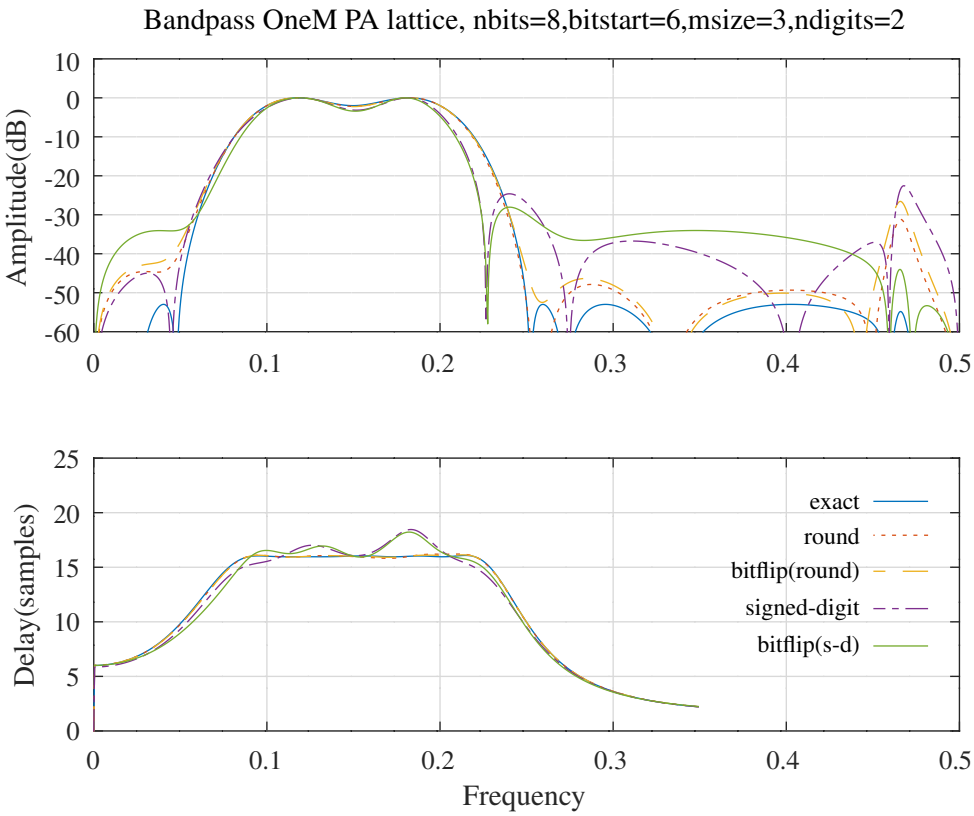



Figure 13.6: Amplitude and group-delay responses of a band-pass filter synthesised as a parallel-allpass one-multiplier lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

## 13.4 Bit-flipping search for the signed-digit coefficients of a one-multiplier parallel-allpass lattice bandpass IIR filter

The Octave script *bitflip_schurOneMPAlattice_bandpass_test.m* applies the bit-flipping algorithm to the example bandpass filter implemented as a Schur parallel-allpass one-multiplier lattice filter. Note that, in this example, the one-multiplier state scaling coefficients are not truncated. The IIR filter in Section 13.3 has a denominator polynomial decimated with $R = 2$. The initial IIR filter in this example is that designed by the Octave script *schurOneMPAlattice_socp_slb_bandpass_test.m*.

The filter responses with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm are shown in Figure 13.6. The filter corresponding passband responses are shown in Figure 13.7.

The filter responses for the signed-digit coefficients with 2-signed-digits, 2-signed-digits allocated with Lim's algorithm and 2-signed-digits allocated with Ito's algorithm are shown in Figure 13.8. The corresponding pass-band responses are shown in Figure 13.9.

Table 13.4 compares the cost result for each test.

The *bitflip* optimised 8-bit, rounded, parallel-allpass one-multiplier lattice bandpass filter coefficients are:

```
A1k_bf = [      -56,      106,        9,      -42, ...
                 70,      -24,      -24,       56, ...
                -36,       21 ]/128;

A2k_bf = [      -98,      111,        4,      -48, ...
                 72,      -13,      -18,       55, ...
                -40,       19 ]/128;
```

A total of 29 adders is required to implement these rounded coefficients.

The *bitflip* optimised 8-bit, 2-signed-digit, parallel-allpass one-multiplier lattice bandpass filter coefficients are:

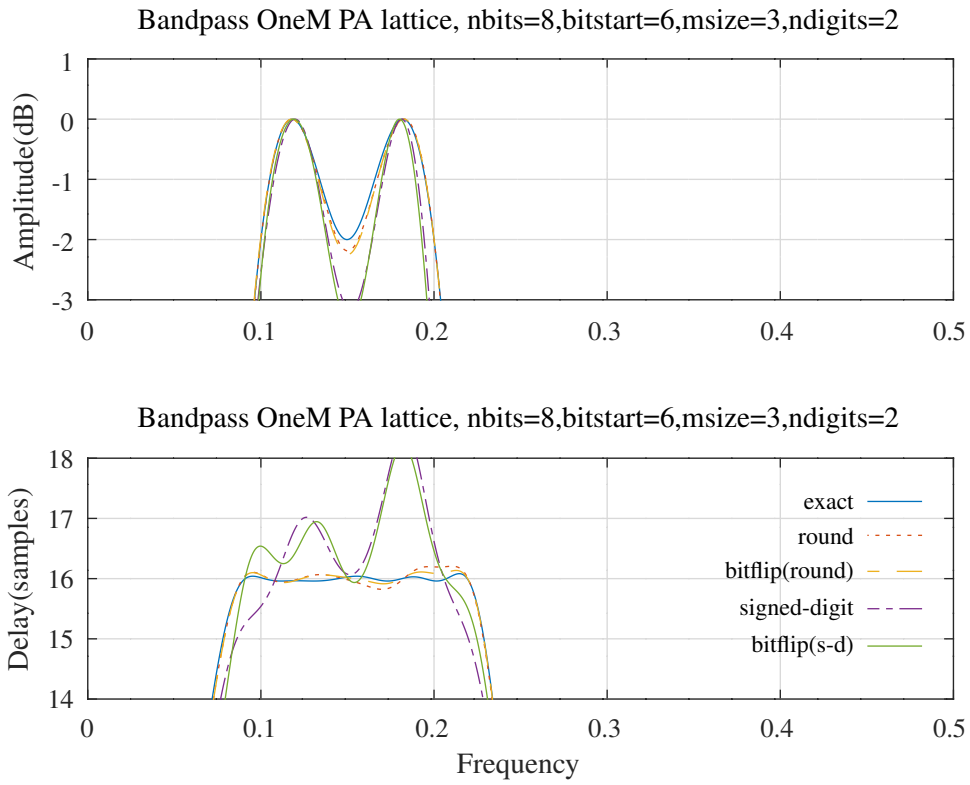



Figure 13.7: Amplitude and group-delay passband responses of a band-pass filter synthesised as a parallel-allpass one-multiplier lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

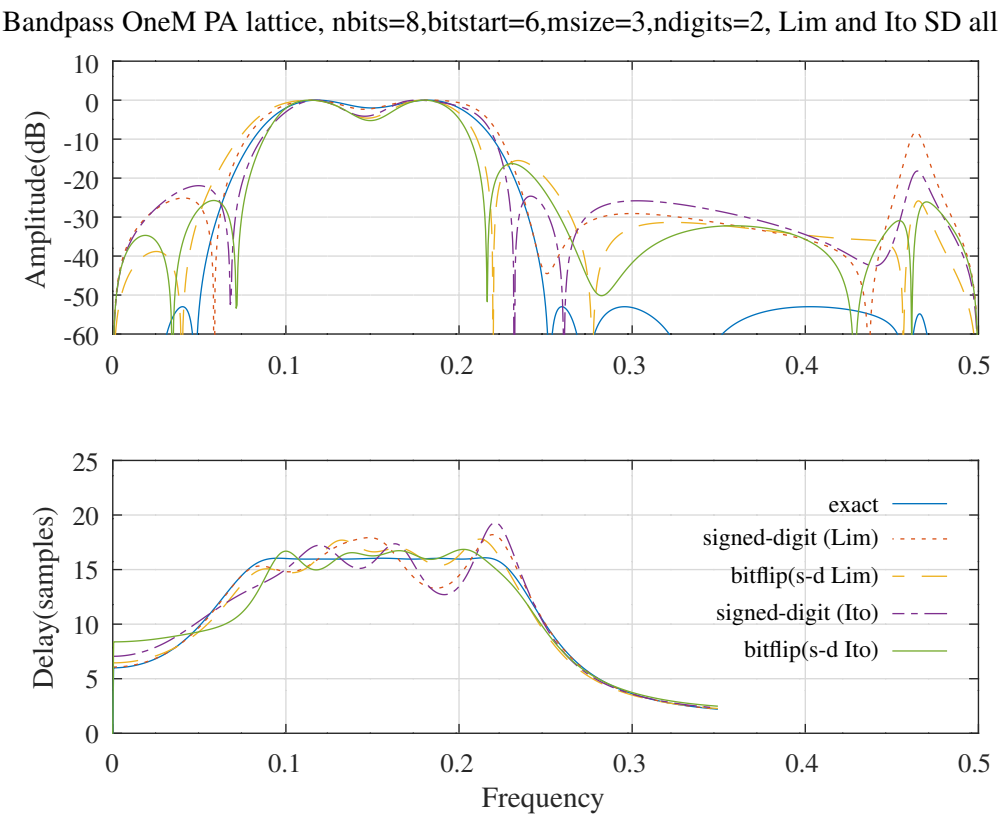



Figure 13.8: Amplitude and group-delay responses of a band-pass filter synthesised as a parallel-allpass one-multiplier lattice filter with exact coefficients, 8-bit 2-signed-digit coefficients allocated with Lim's algorithm, 8-bit 2-signed-digit coefficients allocated with Lim's algorithm and optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients allocated with Ito's algorithm and 8-bit 2-signed-digit coefficients allocated with Ito's algorithm and optimised with the bit-flipping algorithm

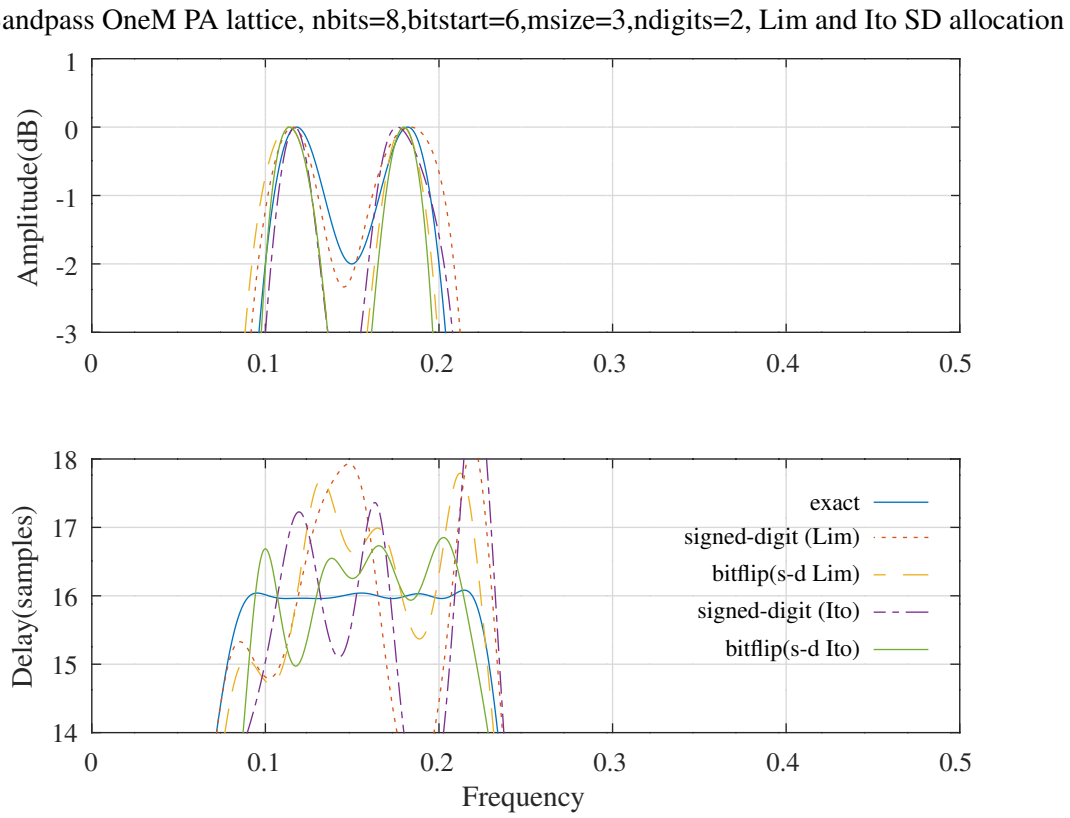



Figure 13.9: Amplitude and group-delay pass-band responses of a band-pass filter synthesised as a parallel-allpass one-multiplier lattice filter with exact coefficients, 8-bit 2-signed-digit coefficients allocated with Lim's algorithm, 8-bit 2-signed-digit coefficients allocated with Lim's algorithm and optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients allocated with Ito's algorithm and 8-bit 2-signed-digit coefficients allocated with Ito's algorithm and optimised with the bit-flipping algorithm

| Band-pass Schur parallel-allpass one-multiplier | Cost |
|---|---|
| Exact | 1.0870 |
| 8-bit rounded | 1.8517 |
| 8-bit rounded with bit-flipping | 1.6093 |
| 8-bit 2-signed-digit | 10.1625 |
| 8-bit 2-signed-digit with bit-flipping | 9.1437 |
| 8-bit 2-signed-digit(Lim alloc.) | 16.8760 |
| 8-bit 2-signed-digit(Lim alloc.) with bit-flipping | 9.8041 |
| 8-bit 2-signed-digit(Ito alloc.) | 15.0589 |
| 8-bit 2-signed-digit(Ito alloc.) with bit-flipping | 6.5797 |

Table 13.4: Summary of the cost results for the bandpass filter synthesised as a parallel-allpass one-multiplier lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm. The signed digits are allocated with 2 digits each, Lim's allocation method with an average of 2-signed-digits each and Ito's allocation method with an average of 2-signed-digits each.

```
A1k_bfsd = [      -56,      112,       10,      -40, ...
                   72,      -24,      -24,       56, ...
                  -36,       20 ]/128;


A2k_bfsd = [      -96,      112,        9,      -48, ...
                   72,      -12,      -18,       56, ...
                  -40,       16 ]/128;
```

A total of 19 adders is required to implement these signed-digit coefficients.

## 13.5 Bit-flipping search for the signed-digit coefficients of a minimum-phase band-pass FIR filter

Section 8.2.13 shows the design of a minimum-phase FIR bandpass filter with an amplitude response that is similar to that of the IIR examples 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 FIR Schur lattice has two real multipliers for each reflection coefficient [227].

The Octave script, *bitflip_schurFIRlattice_bandpass_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 = [   0.1005298996,   0.1293242803,   0.0242501500,  -0.1717872849, ...
        -0.2582567805,  -0.1154275079,   0.1245916592,   0.2242804626, ...
         0.1203731991,  -0.0278730002,  -0.0672739488,  -0.0199185076, ...
         0.0038876245,  -0.0235774874,  -0.0438306579,  -0.0253907727, ...
         0.0267391576 ]';
```

The Schur FIR lattice multiplier coefficients of the initial filter are

```
k_ex = [   0.4673548523,   0.0215846872,  -0.4485631407,  -0.5735447576, ...
          -0.3680742834,   0.1477578747,   0.4893876472,   0.4478270228, ...
           0.0701156912,  -0.3783422054,  -0.5960490321,  -0.7267672416, ...
          -0.7928968758,   0.6674446338,  -0.6400144940,   0.2659821377 ];
```

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 13.10.

Table 13.5 compares the cost results for each truncation method.

| Band-pass Schur FIR | Cost |
|---|---|
| Exact | 3.0422 |
| 8-bit rounded | 3.0646 |
| 8-bit rounded with bitflipping | 2.8440 |
| 8-bit 2-signed-digit | 2.9774 |
| 8-bit 2-signed-digit with bitflipping | 2.8362 |

Table 13.5: Summary of the cost results for the FIR minimum-phase bandpass filter synthesised as a Schur FIR lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

The *bitflip* optimised 8-bit 2-signed-digit Schur FIR lattice coefficients are:

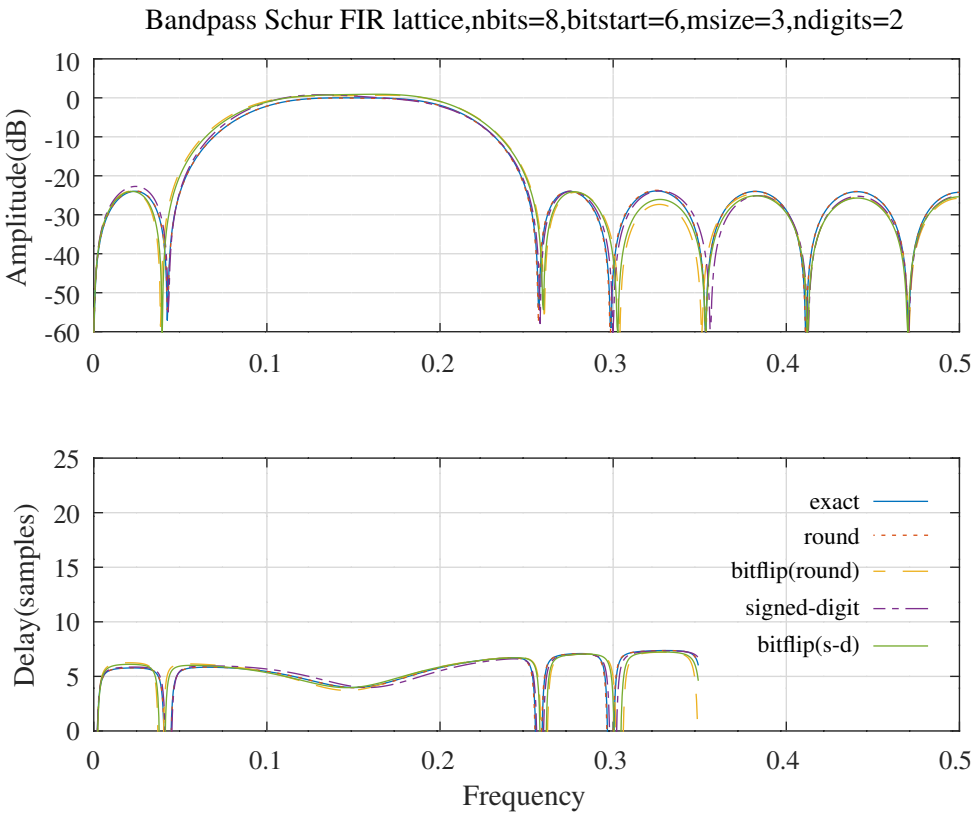



Figure 13.10: Amplitude and group-delay responses of a minimum-phase band-pass filter synthesised as a Schur FIR lattice filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

```
k_bfsd = [        56,         9,       -56,       -72, ...
                 -48,        34,        63,        60, ...
                  28,       -48,       -80,       -96, ...
                 -96,        80,       -80,        24 ]/128;
```

A total of 32 adders is required to implement these signed-digit coefficients.

## 13.6 Bit-flipping search for the signed-digit coefficients of a direct-form symmetric bandpass FIR filter

Appendix N.3 shows the design of a direct-form symmetric FIR bandpass filter with an amplitude response that is similar to that of the IIR examples above. The filter is linear phase and consequently has a flat group delay response. The Octave script *bitflip_directFIRsymmetric_bandpass_test.m* uses the bit-flipping algorithm to optimise the coefficients when truncated to 8-bit integers. The initial filter polynomial is that designed by the Octave script *directFIRsymmetric_slb_bandpass_test.m*. The distinct initial coefficients are:

```
hM_ex = [  -0.0004538174,  -0.0114029873,  -0.0194431345,  -0.0069796479, ...
            0.0215771882,   0.0348545408,   0.0158541332,  -0.0033225166, ...
            0.0154055974,   0.0414424100,  -0.0021970758,  -0.1162784301, ...
           -0.1760013118,  -0.0669604509,   0.1451751014,   0.2540400868 ];
```

The filter order is 30 giving a filter delay of 15 samples. The symmetric direct-form implementation requires 16 multipliers. The initial filter response and the filter responses after coefficient truncation are shown in Figure 13.11. The bit-flipping algorithm does not improve the response obtained with 8-bit rounded coefficients.

The filter responses for the signed-digit coefficients with 2-signed-digits, 2-signed-digits allocated with Lim's algorithm and 2-signed-digits allocated with Ito's algorithm are shown in Figure 13.12.

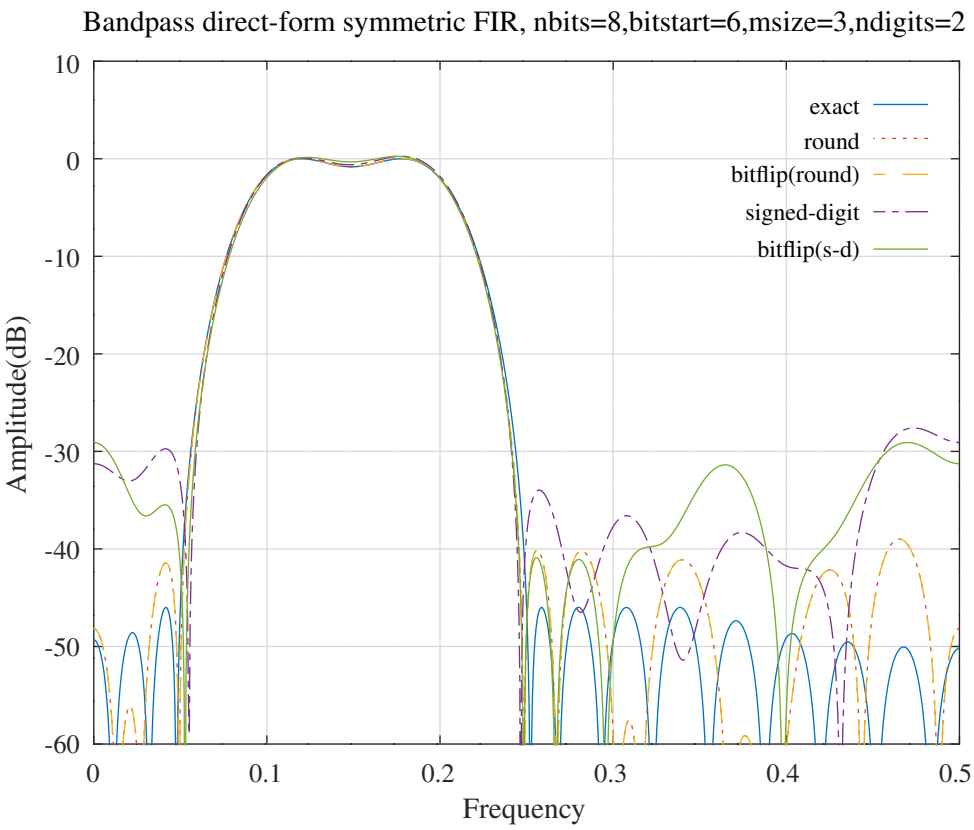



Figure 13.11: Amplitude and group-delay responses of a direct-form symmetric FIR band-pass filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm

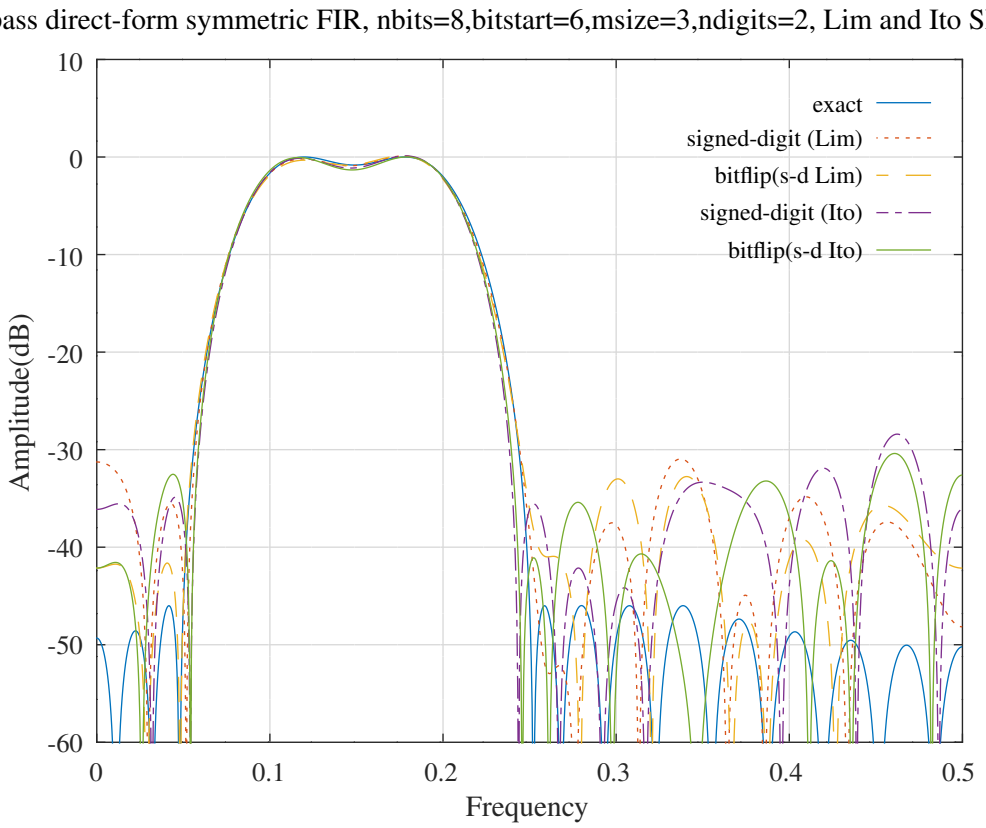



Figure 13.12: Amplitude and group-delay responses of a direct-form symmetric FIR band-pass filter with exact coefficients, 8-bit 2-signed-digit coefficients allocated with Lim's algorithm, 8-bit 2-signed-digit coefficients allocated with Lim's algorithm and optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients allocated with Ito's algorithm and 8-bit 2-signed-digit coefficients allocated with Ito's algorithm and optimised with the bit-flipping algorithm

| Band-pass direct-form symmetric FIR | Cost |
|---|---|
| Exact | 0.0011 |
| 8-bit rounded | 0.0015 |
| 8-bit rounded with bitflipping | 0.0015 |
| 8-bit 2-signed-digit | 0.0082 |
| 8-bit 2-signed-digit with bitflipping | 0.0074 |
| 8-bit 2-signed-digit(Lim alloc.) | 0.0042 |
| 8-bit 2-signed-digit(Lim alloc.) with bit-flipping | 0.0036 |
| 8-bit 2-signed-digit(Ito alloc.) | 0.0062 |
| 8-bit 2-signed-digit(Ito alloc.) with bit-flipping | 0.0047 |

Table 13.6: Summary of the cost results for the direct-form symmetric FIR bandpass filter with exact coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flipping algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flipping algorithm. The signed digits are allocated with 2 digits each, Lim's allocation method with an average of 2-signed-digits each and Ito's allocation method with an average of 2-signed-digits each.

Table 13.6 compares the cost results for each truncation method.

The *bitflip* optimised 8-bit 2-signed-digits (allocated with Lim's algorithm) direct-form symmetric filter coefficients are:

```
hM_bfsdl = [        0,       -3,       -5,        0, ...
                    6,        8,        3,       -1, ...
                    4,       11,        0,      -30, ...
                  -44,      -17,       37,       64 ]/256;
```

The coefficients are scaled to make full use of the 8-bit range. A total of 12 adders is required to implement these signed-digit coefficients.

The *bitflip* optimised 8-bit rounded direct-form symmetric FIR coefficients are:

```
hM_bf = [        0,       -3,       -5,       -2, ...
                 6,        9,        4,       -1, ...
                 4,       11,       -1,      -30, ...
               -45,      -17,       37,       65 ]/256;
```

A total of 14 adders is required to implement these coefficient multiplications in the direct-form symmetric FIR filter structure and 27 adders are required to implement the coefficient multiplications in the transposed (pipelined) direct-form FIR filter structure.

# Chapter 14

# *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 $\mathcal{O}\left(2^K\right)$ comparisons of the corresponding filter approximation error. *Branch-and-bound* [5], [174, 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 14.1 shows a flow diagram of an implementation of the algorithm using a stack. The exact filter coefficients $\boldsymbol{x}$ are approximated by the signed-digit coefficients $\bar{\boldsymbol{x}}$. 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.* [178] 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 the 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.

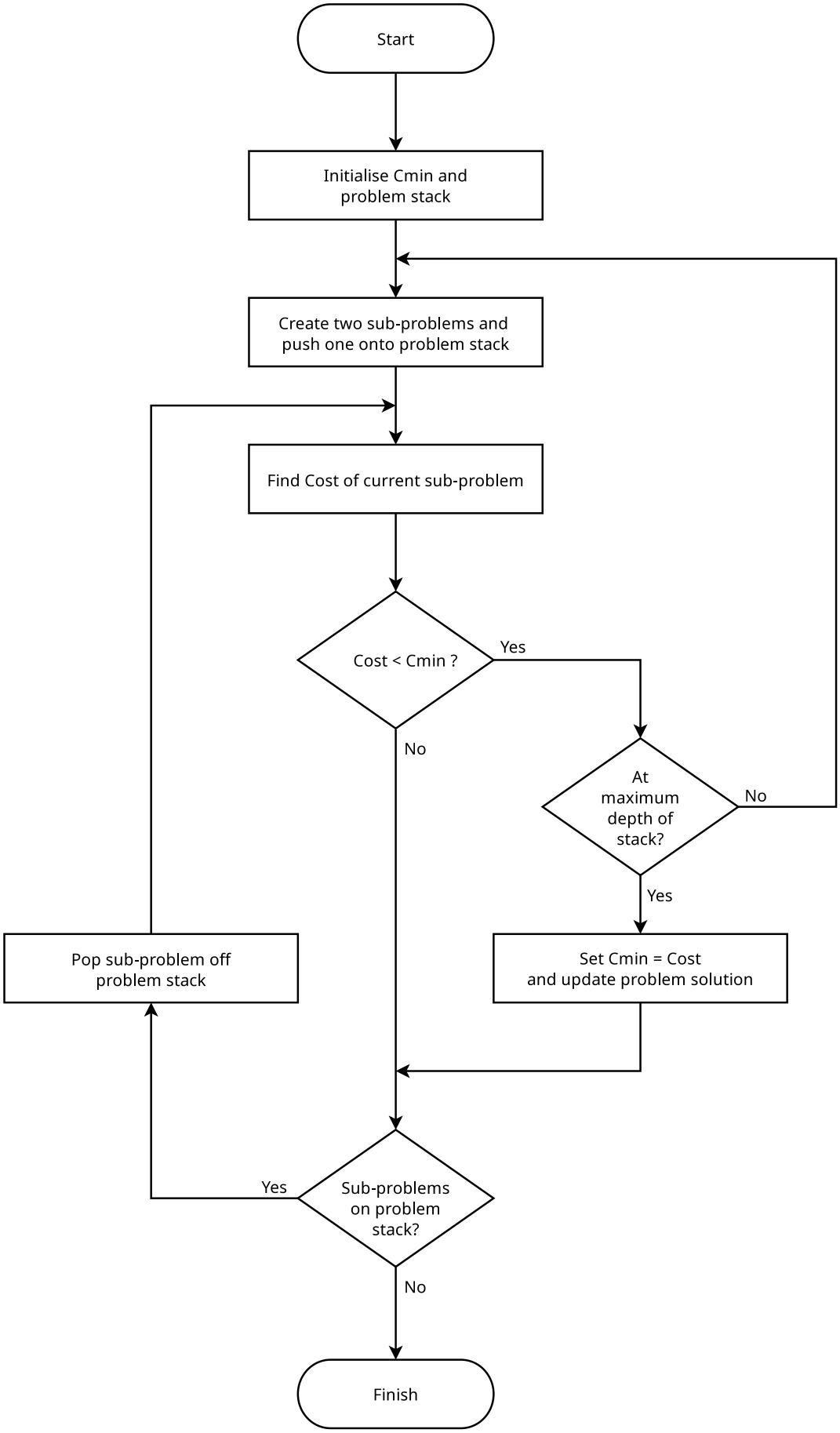



Figure 14.1: The branch-and-bound tree search algorithm.

## 14.1 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a direct-form symmetric bandpass FIR filter

The Octave script *branch_bound_directFIRsymmetric_bandpass_8_nbits_test.m* uses the branch-and-bound algorithm to optimise the 8-bit 3-signed-digit coefficients of a direct-form symmetric FIR bandpass filter.

The initial filter polynomial is that designed by the Octave script *directFIRsymmetric_slb_bandpass_test.m*.

```
hM1 = [  -0.0058181010,   0.0017787857,  -0.0047084625,  -0.0143846688, ...
         -0.0077550125,   0.0219788564,   0.0432578789,   0.0247317110, ...
         -0.0077853817,  -0.0010276677,   0.0304650309,   0.0009925325, ...
         -0.1110651112,  -0.1806101683,  -0.0725659905,   0.1536437055, ...
          0.2719559562 ]';
```

The filter specification is:

```
nbits=8 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
escale=2 % Coefficient scaling for full range
tol=0.0001 % Tolerance on coefficient. update
maxiter=400 % iteration limit
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
Wap=1 % Amplitude pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
Wasl=5 % Amplitude lower stop band weight
Wasu=10 % Amplitude upper stop band weight
```

The initial filter response and the filter responses after coefficient truncation to 8-bit 3-signed-digits and branch-and-bound search are shown in Figure 14.2. Table 14.1 compares the cost results.

The *branch-and-bound* optimised 8-bit 2-signed-digits direct-form symmetric filter coefficients are:

```
hM_min = [        -1,         0,        -2,        -4, ...
                  -2,         5,        11,         6, ...
                  -1,         0,         8,         1, ...
                 -28,       -47,       -19,        39, ...
                  70 ]'/256;
```

The coefficients are scaled to make full use of the 8-bit range. A total of 13 adders is required to implement these signed-digit coefficient multiplications.

| Band-pass direct-form symmetric FIR | Cost | Signed-digits | Additions |
|---|---|---|---|
| Exact | 0.00133 | | |
| 8-bit 3-signed-digit | 0.00141 | 27 | 13 |
| 8-bit 3-signed-digit(branch-and-bound) | 0.00127 | 28 | 13 |

Table 14.1: Summary of the cost results for the direct-form symmetric FIR bandpass filter with exact coefficients, 8-bit 3-signed-digit coefficients and 8-bit 3-signed-digit coefficients optimised with the branch-and-bound algorithm.

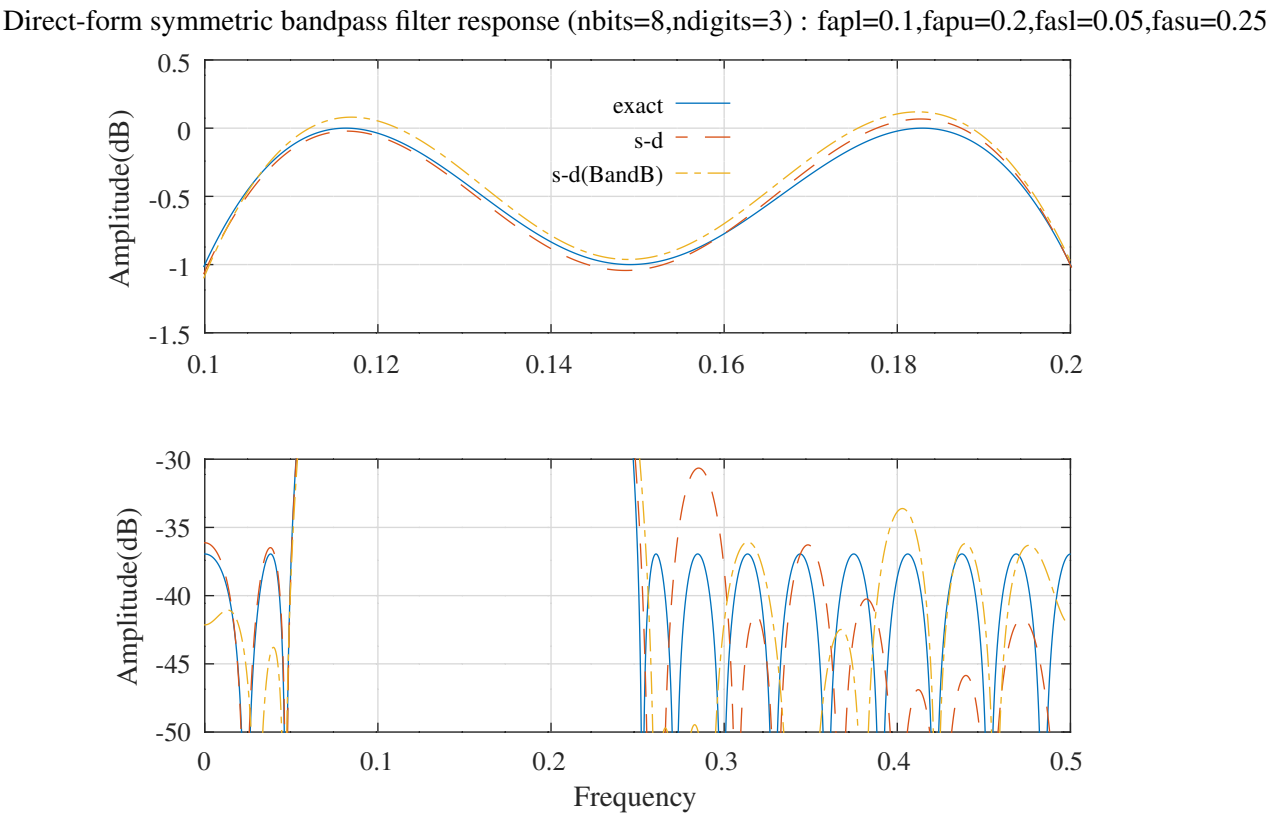



Figure 14.2: Amplitude responses of a direct-form symmetric FIR band-pass filter with exact coefficients, 8-bit 3-signed-digit coefficients and 8-bit 3-signed-digit coefficients optimised with the branch-and-bound algorithm

## 14.2 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a lattice band-pass IIR filter

The Octave script *branch_bound_schurOneMlattice_bandpass_8_nbits_test.m* uses the branch-and-bound heuristic to optimise the response of the SQP optimised band-pass Schur one-multiplier lattice filter of Section 10.3.6 with coefficients truncated to 8 bits and 3 signed-digits. The filter specification is:

```
nbits=8 % 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
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=10 % 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=10000000 % Amplitude upper stop band weight
```

The 31 non-zero 8-bit 3-signed-digit one-multiplier lattice coefficients found by the branch-and-bound search are:

```
k_min = [        0,       88,        0,       63, ...
                 0,       44,        0,       52, ...
                 0,       39,        0,       32, ...
                 0,       20,        0,       14, ...
                 0,        5,        0,        2 ]'/128;


c_min = [       18,       -3,      -76,     -124, ...
               -42,       31,      100,       76, ...
                 4,      -21,      -20,       -3, ...
                -2,       -9,       -7,        1, ...
                 6,        5,        1,        0, ...
                 1 ]'/256;
```

The $c_{min}$ tap coefficients have been scaled to make full use of the range of integers available. I assume that the internal filter state scaling is approximated by bit-shifts. The $\epsilon$ one-multiplier lattice cofficients are not recalculated since that would require rescaling the $c_{min}$ tap coefficients. Figure 14.3 compares the pass-band responses of the filter with exact coefficients, the initial 8-bit 3-signed-digit coefficients and 8-bit 3-signed-digit coefficients found by branch-and-bound search. Figure 14.4 shows the filter stop-band response and Figure 14.5 shows the filter pass-band group delay response. Table 14.2 compares the cost and number of 8 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 structure[a]. Although 3 signed-digits are allocated to each coefficient, many of these signed-digits are not used.

[a] A 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 5.4.

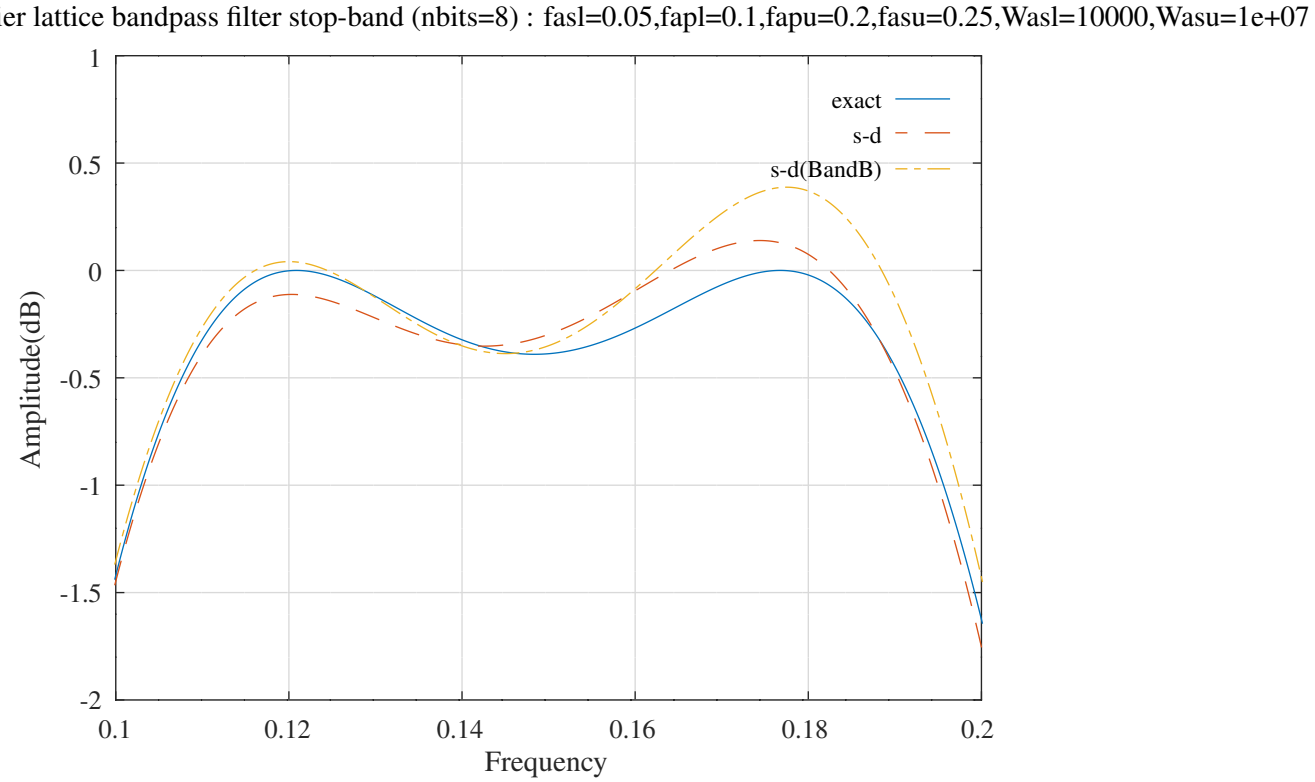



Figure 14.3: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 8-bit 3-signed-digit coefficients found by branch-and-bound search

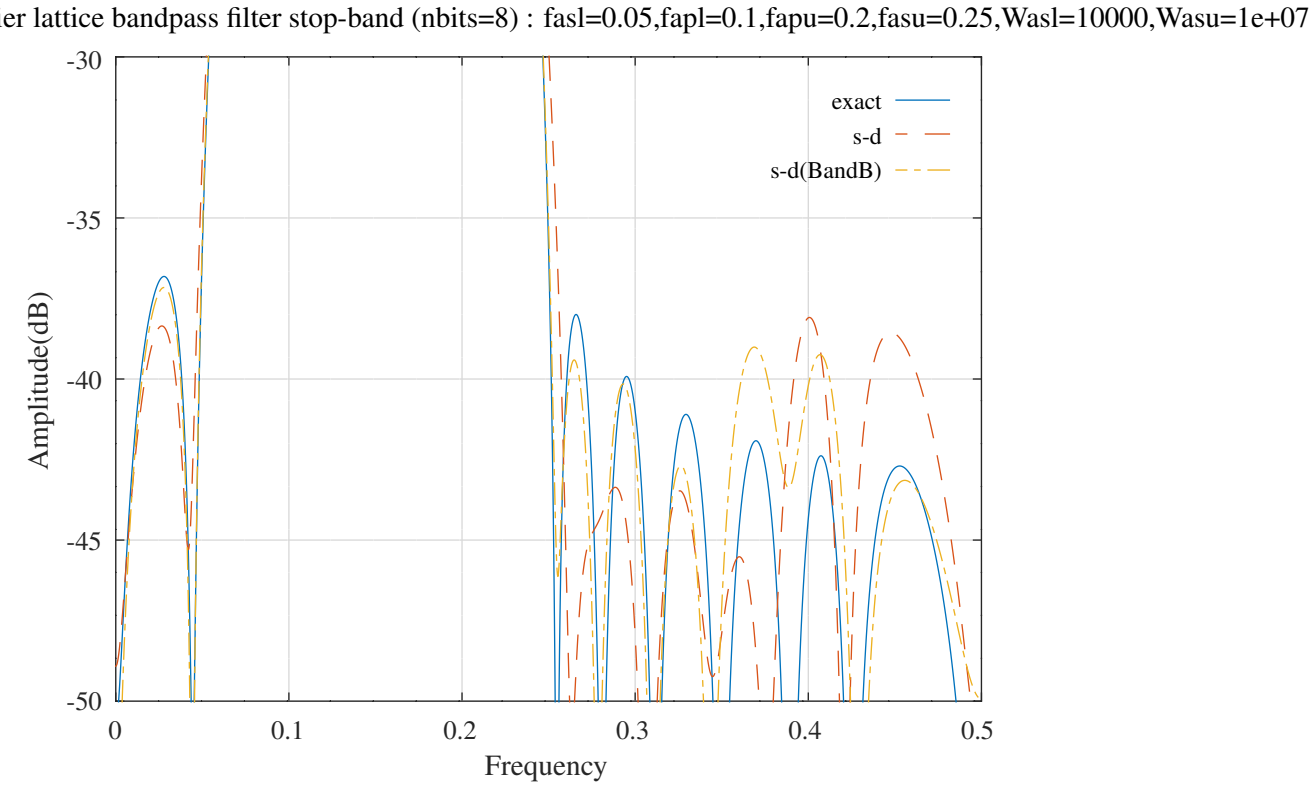



Figure 14.4: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 8-bit 3-signed-digit coefficients found by branch-and-bound search

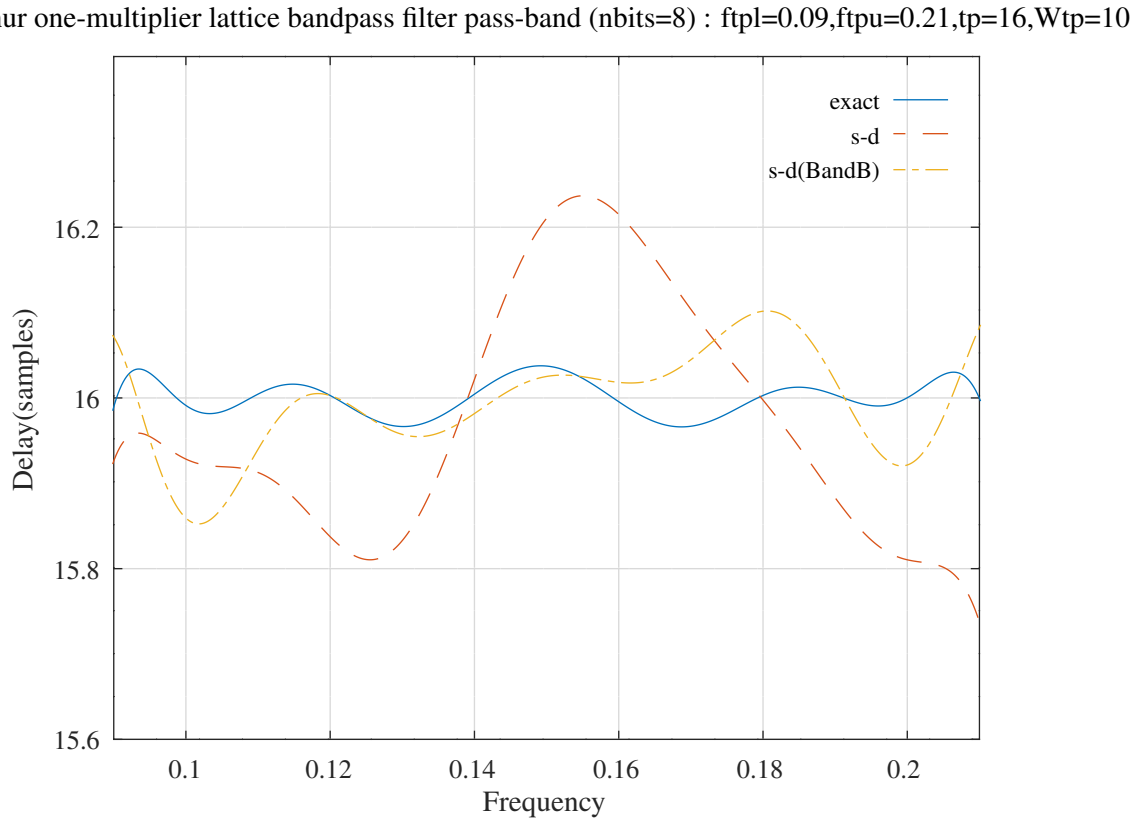



Figure 14.5: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 8-bit 3-signed-digit coefficients found by branch-and-bound search

| | Cost | Signed-digits | Additions |
|---|---|---|---|
| Exact | 0.0598 | | |
| 8-bit 3-signed-digit | 0.4404 | 64 | 34 |
| 8-bit 3-signed-digit(branch-and-bound) | 0.1035 | 62 | 32 |

Table 14.2: Comparison of the cost and number of additions required to implement the coefficient multiplications for a Schur one-multiplier lattice bandpass filter with 8-bit 3-signed-digit coefficients found by branch-and-bound search

## 14.3 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a lattice band-pass IIR filter

The Octave script *branch_bound_schurOneMlattice_bandpass_10_nbits_test.m* uses the branch-and-bound heuristic to optimise the response of the band-pass Schur one-multiplier lattice filter of Section 10.3.6. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
tol=0.0001 % Tolerance on coefficient update
ctol=1e-05 % Tolerance on constraints
maxiter=1000 % 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=0.998047 % 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=0.4 % Delay pass band peak-to-peak ripple
Wtp=5 % 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=500000 % 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 11.2. The average number of non-zero signed-digits per coefficient is close to 2. The tap coefficients are scaled to make full use of the range of integers available. 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:

```
k_min = [        0,      344,        0,      256, ...
                 0,      184,        0,      212, ...
                 0,      156,        0,      128, ...
                 0,       80,        0,       52, ...
                 0,       19,        0,        7 ]'/512;


c_min = [       69,      -16,     -304,     -480, ...
              -156,      128,      400,      304, ...
                16,      -84,      -80,      -16, ...
                -8,      -33,      -24,        4, ...
                24,       16,        3,        1, ...
                 4 ]'/1024;
```

Figure 14.6 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 14.7 shows the filter stop-band response and Figure 14.8 shows the filter pass-band group delay response. Table 14.3 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.

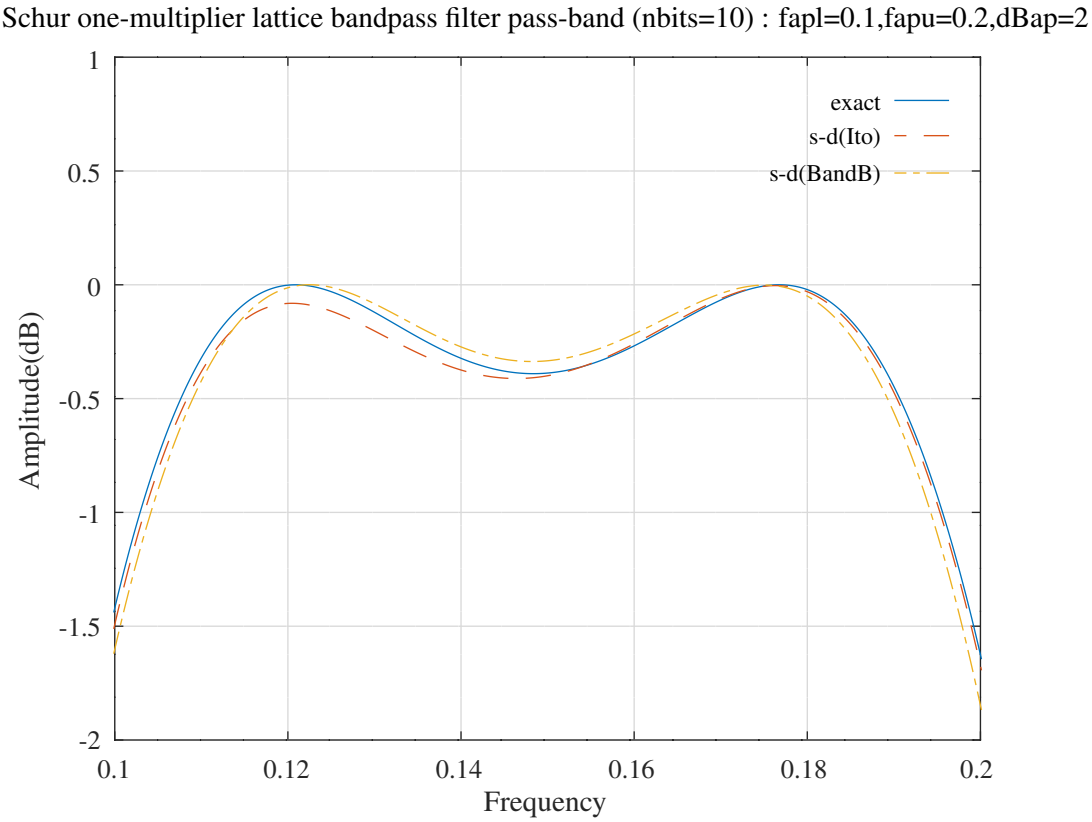



Figure 14.6: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search

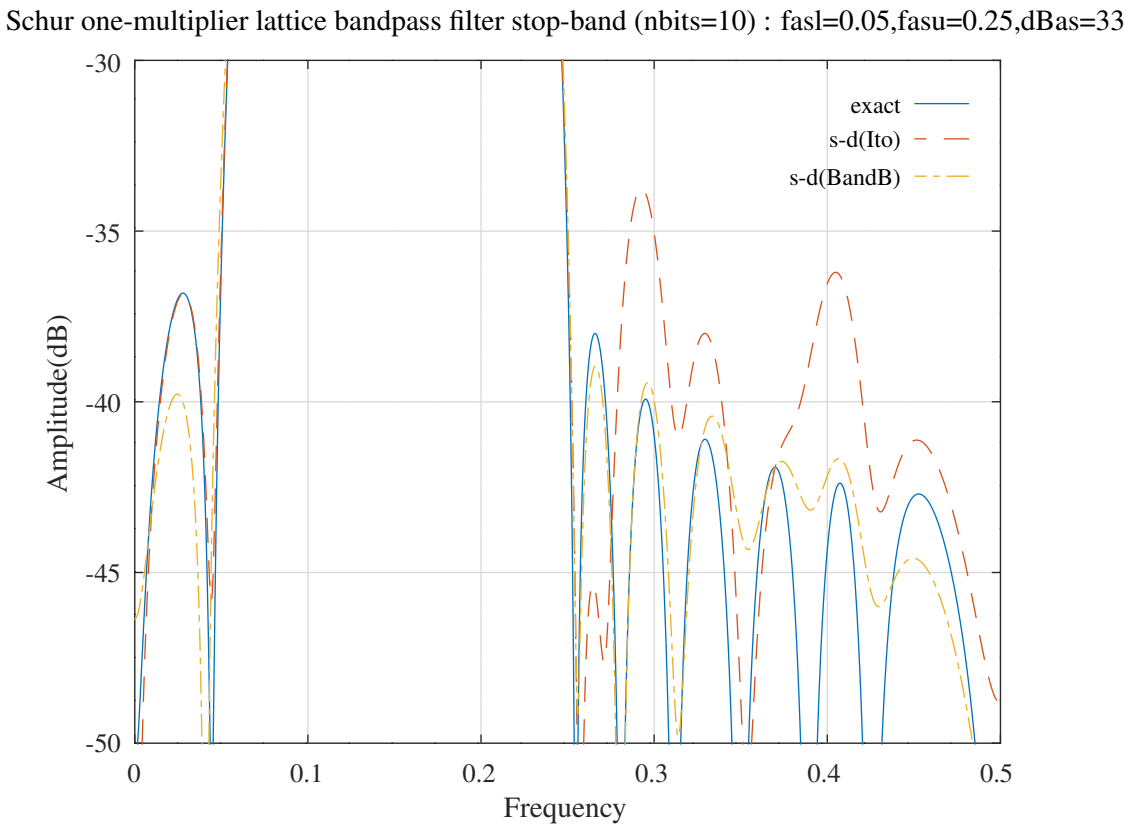



Figure 14.7: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search

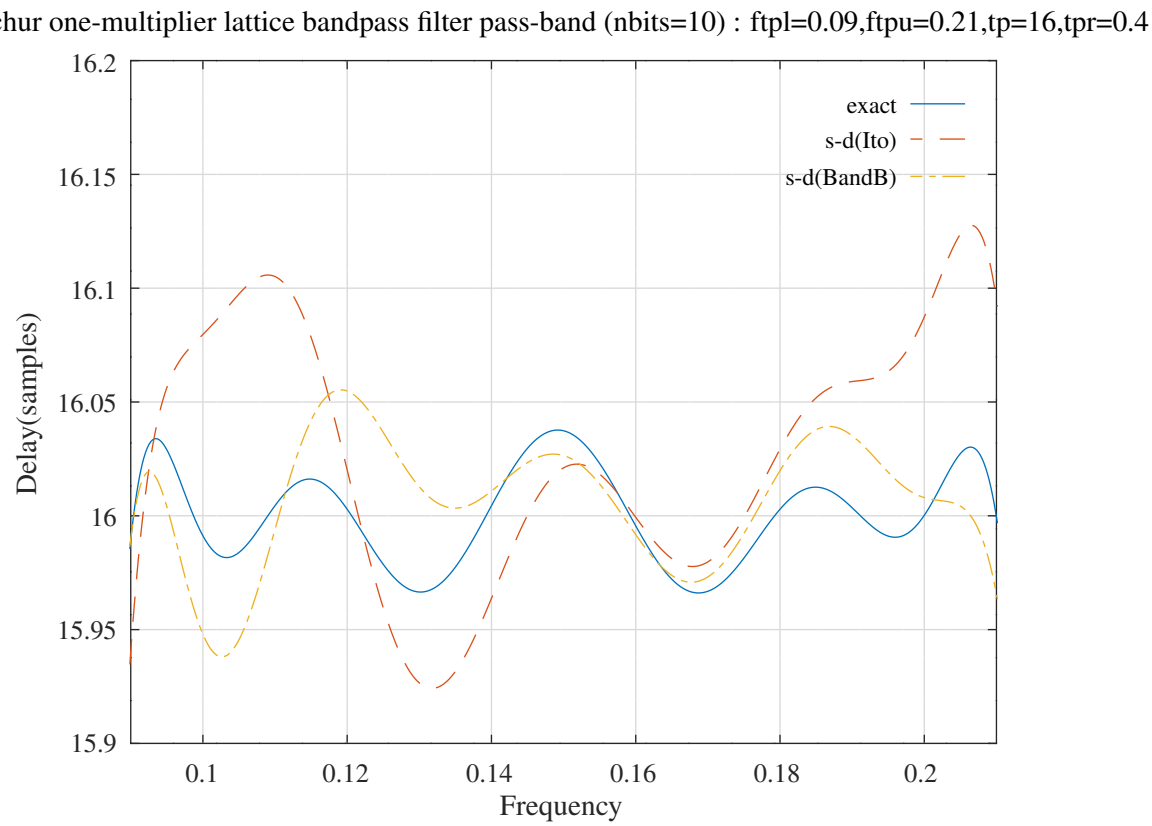



Figure 14.8: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.0164 | | |
| 10-bit 3-signed-digit(Ito) | 0.0578 | 65 | 34 |
| 10-bit 3-signed-digit(branch-and-bound) | 0.0207 | 65 | 34 |

Table 14.3: 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

## 14.4 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass lattice low-pass IIR filter

The Octave script *branch_bound_schurOneMPAlattice_lowpass_12_nbits_test.m* performs branch-and-bound search to optimise the response of the parallel Schur one-multiplier all-pass lattice low-pass filter of Section 10.3.4 with 12-bit integer coefficients. The filter specification is:

```
nbits=12 % Coefficient word length
ndigits=3 % Average number of signed digits per coef.
tol=0.0001 % Tolerance on coefficient update vector
ctol=5e-07 % Tolerance on constraints
n=400 % Frequency points across the band
difference=0 % Use difference of all-pass filters
m1=11 % Allpass model filter 1 denominator order
m2=12 % Allpass model filter 2 denominator order
rho=0.992188 % Constraint on allpass coefficients
fap=0.125 % Amplitude pass band edge
dBap=0.2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=0 % Amplitude transition band weight
fas=0.25 % Amplitude stop band edge
dBas=56 % amplitude stop band peak-to-peak ripple
Was=10000 % Amplitude stop band weight
ftp=0.175 % Delay pass band edge
td=11.5 % Nominal pass band filter group delay
tdr=0.08 % Delay pass band peak-to-peak ripple
Wtp=0.1 % Delay pass band weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2. 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 SOCP PCLS optimised with the filter specification given above. In this case there are no filter tap coefficients and the one-multiplier lattice $\epsilon$ coefficients can be recalculated. The signed-digit lattice coefficients found by the heuristic of *Ito et al.* are:

```
A1k0_sd = [     1576,     -192,     -548,     -128, ...
                -128,      500,     -296,       -8, ...
                 336,     -328,      110 ]'/2048;

A2k0_sd = [      800,     -560,      384,      336, ...
                 -96,       84,     -412,      368, ...
                  12,     -366,      308,     -112 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are:

```
A1k_min = [     1592,     -160,     -543,     -128, ...
                -128,      476,     -290,       16, ...
                 336,     -324,      115 ]'/2048;

A2k_min = [      784,     -592,      384,      352, ...
                 -96,       76,     -398,      368, ...
                  -4,     -360,      304,     -119 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 62 signed-digits and 39 shift-and-add operations.

Figure 14.9 shows the amplitude pass-band response of the filter with 12-bit 3-signed-digit coefficients allocated with the algorithm of *Ito et al.* and branch-and-bound search. Figure 14.10 shows the corresponding filter pass-band group-delay response. Figure 14.11 shows the corresponding filter stop-band amplitude response. Table 14.4 compares the cost and the number of 12 bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the branch-and-bound search.

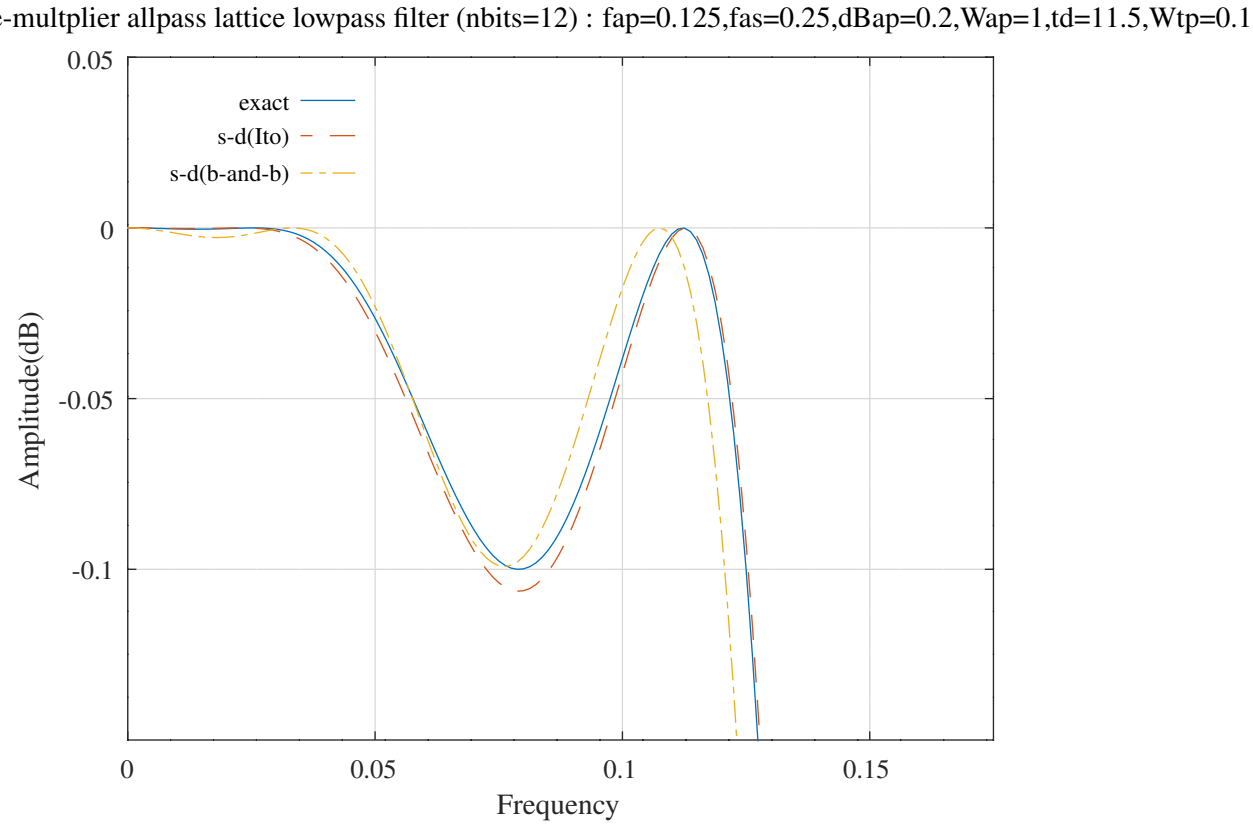



Figure 14.9: Pass band amplitude response for a parallel Schur one-multiplier all-pass lattice low-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

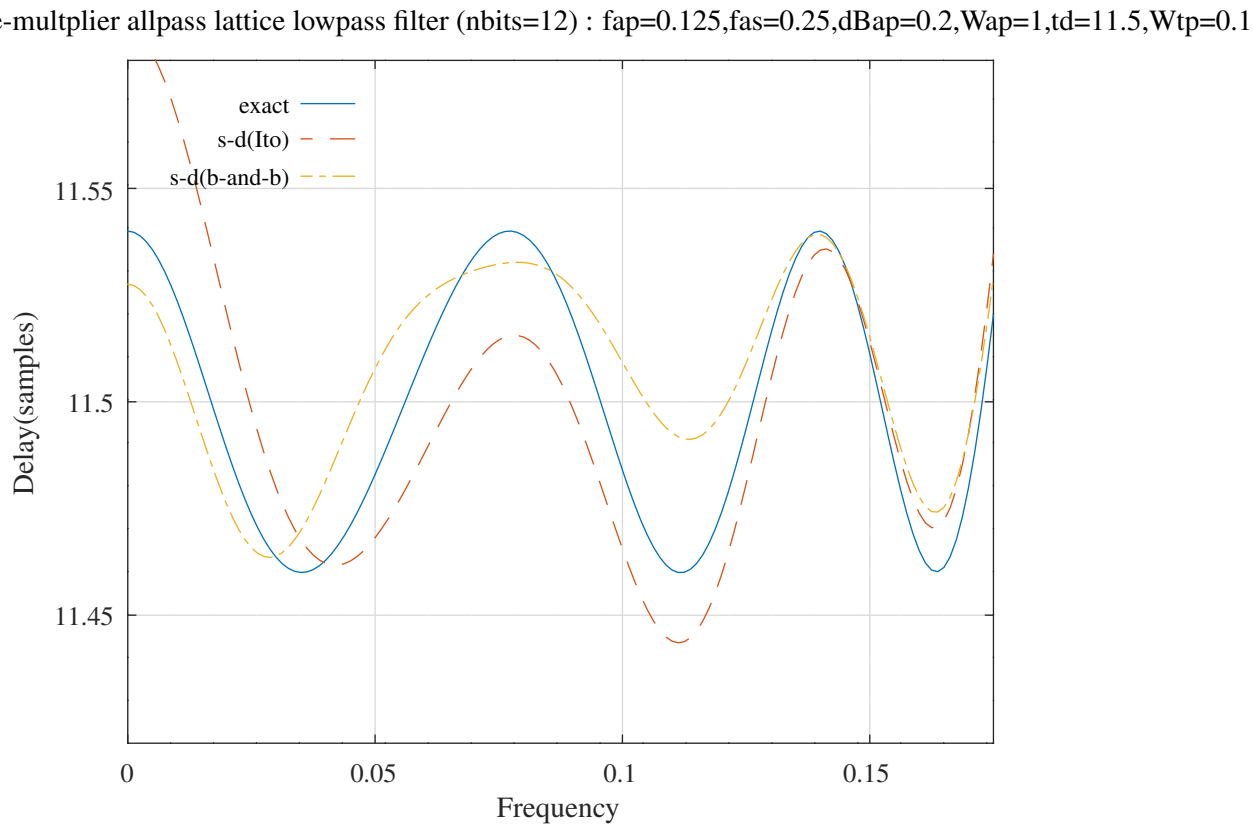



Figure 14.10: Pass band group delay response for a parallel Schur one-multiplier all-pass lattice low-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

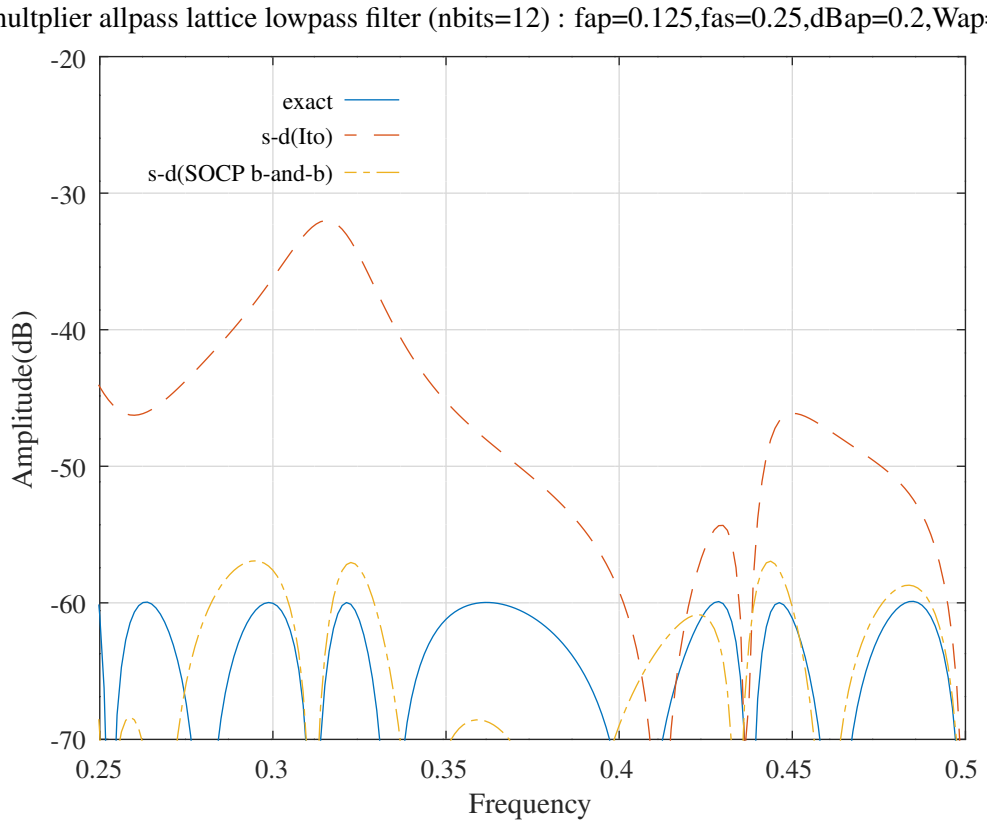



Figure 14.11: Stop band amplitude response for a parallel Schur one-multiplier all-pass lattice low-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.000195 | | |
| 12-bit 3-signed-digit(Ito) | 0.000685 | 62 | 39 |
| 12-bit 3-signed-digit(SOCP b-and-b) | 0.000191 | 62 | 39 |

Table 14.4: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a parallel Schur one-multiplier all-pass lattice low-pass filter with 12 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.

## 14.5 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a parallel all-pass lattice IIR elliptic low-pass filter

The Octave script *branch_bound_schurOneMPAlattice_elliptic_lowpass_8_nbits_test.m* performs branch-and-bound search to optimise the response of a parallel Schur one-multiplier all-pass lattice elliptic approximation low-pass filter with 8-bit integer coefficients. The filter specification is:

```
N=5 % Filter order
nbits=8 % Coefficient word length
ndigits=2 % Average number of signed digits per coef.
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-08 % Tolerance on constraints
n=1000 % Frequency points across the band
difference=0 % Use difference of all-pass filters
rho=0.992188 % Constraint on allpass coefficients
fap=0.04 % Amplitude pass band edge
dBap=0.2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=0 % Amplitude transition band weight
fas=0.053 % Amplitude stop band edge
dBas=20 % amplitude stop band peak-to-peak ripple
Was=10 % Amplitude stop band weight
```

The filter coefficients are truncated to 8 bits and 2 signed-digits. At each branch the script fixes the coefficient with the largest difference between upper and lower 2 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 SOCP PCLS optimised with the filter specification given above.

The 8 bit, 2 signed-digit lattice coefficients are:

```
A1k0_sd = [     -124,      112 ]'/128;

A2k0_sd = [     -124,      127,      -96 ]'/128;
```

The signed-digit lattice coefficients found by the branch-and-bound search are:

```
A1k_min = [     -124,       96 ]'/128;

A2k_min = [     -124,      126,      -96 ]'/128;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 10 signed-digits and 5 shift-and-add operations.

Figure 14.12 shows the amplitude response of the filter with 8-bit, 2-signed-digit coefficients and branch-and-bound search. Figure 14.13 shows the filter pass-band amplitude response. Table 14.5 compares the cost and the number of 8 bit shift-and-add operations required to implement the coefficient multiplications found with the branch-and-bound search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.000661 | | |
| 8-bit 2-signed-digit | 0.008154 | 10 | 5 |
| 8-bit 2-signed-digit(b-and-b) | 0.000046 | 10 | 5 |

Table 14.5: Comparison of the cost and number of 8-bit shift-and-add operations required to implement the coefficient multiplications for a parallel Schur one-multiplier all-pass lattice elliptic approximation low-pass filter with 8 bit, 2 signed-digit integer coefficients found by performing branch-and-bound search.

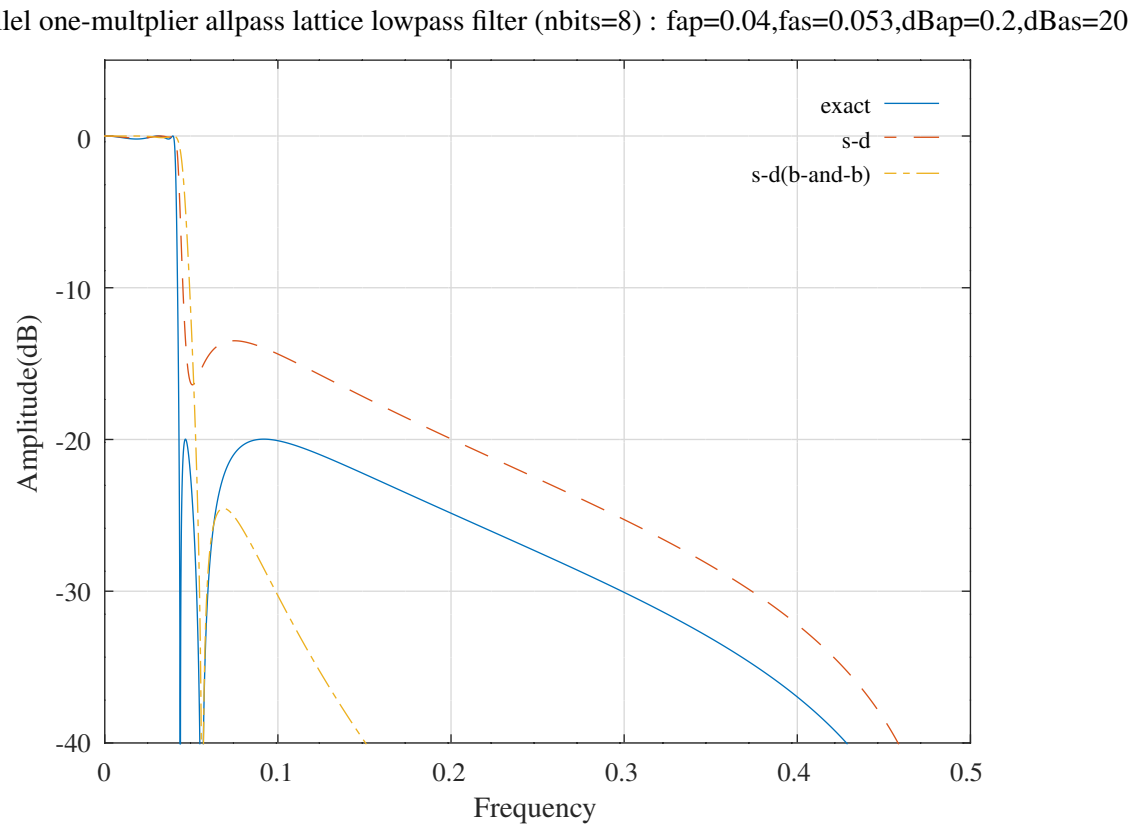



Figure 14.12: Amplitude response for a parallel Schur one-multiplier all-pass lattice elliptic approximation low-pass filter with 8 bit, 2-signed-digits integer coefficients found performing branch-and-bound search.

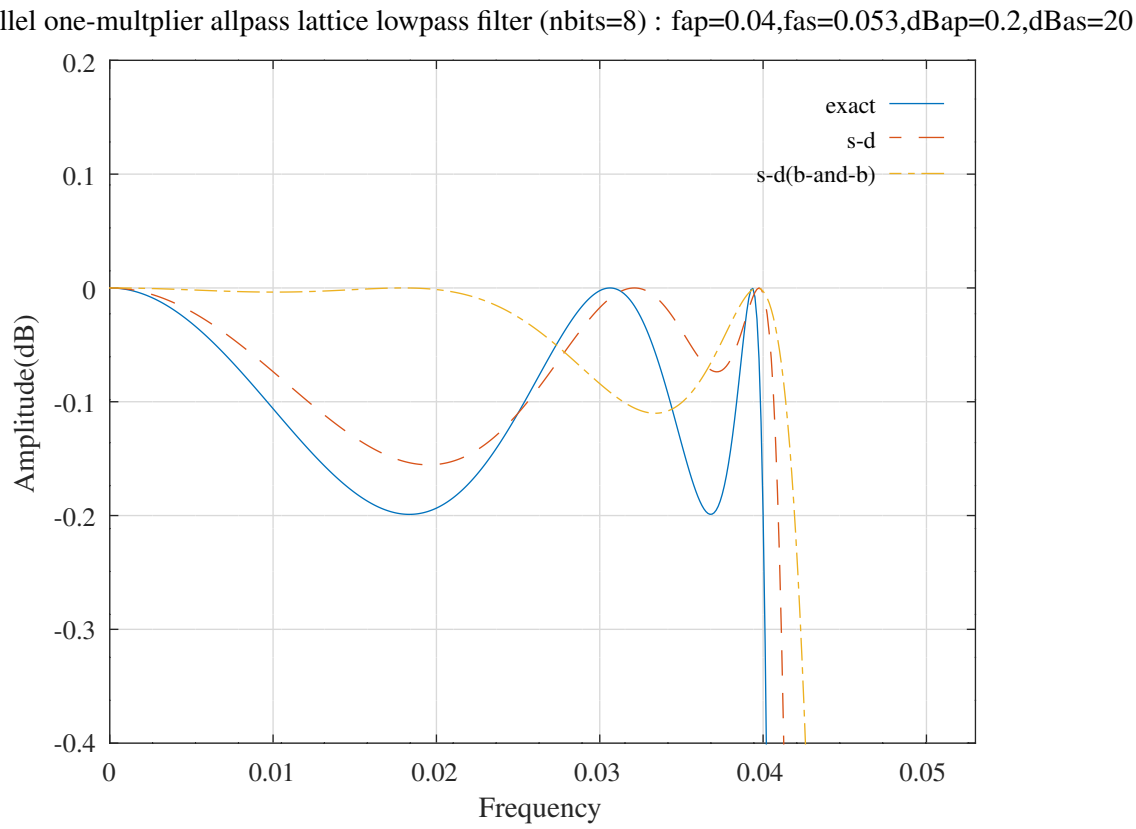



Figure 14.13: Pass band amplitude response for a parallel Schur one-multiplier all-pass lattice elliptic approximation low-pass filter with 8 bit, 2 signed-digit integer coefficients found by performing branch-and-bound search.

## 14.6 Branch-and-bound search for the 16-bit 4-signed-digit coefficients of a parallel all-pass lattice IIR elliptic low-pass filter

The Octave script *branch_bound_schurOneMPAlattice_elliptic_lowpass_16_nbits_test.m* performs branch-and-bound search to optimise the response of the parallel Schur one-multiplier all-pass lattice elliptic low-pass filter with 16-bit integer coefficients. The filter specification is:

```
nbits=16 % Coefficient word length
ndigits=4 % Average number of signed digits per coef.
tol=0.0001 % Tolerance on coefficient update vector
ctol=5e-09 % Tolerance on constraints
n=2000 % Frequency points across the band
difference=0 % Use difference of all-pass filters
rho=0.992188 % Constraint on allpass coefficients
fape=0.05 % Extra amplitude weight pass band edge
fap=0.15 % Amplitude pass band edge
dBap=0.04 % Amplitude pass band peak-to-peak ripple
Wape=0 % Extra amplitude pass band weight
Wap=1 % Amplitude pass band weight
Wat=0.0001 % Amplitude transition band weight
fas=0.171 % Amplitude stop band edge
fase=0.271 % Extra amplitude weight stop band edge
dBas=77 % amplitude stop band peak-to-peak ripple
Was=10000000 % Amplitude stop band weight
Wase=0 % Extra amplitude stop band weight
```

The initial parallel all-pass filters are those for the filter designed by the Octave function *ellip(11,0.02,84,2*0.15)*. The filter coefficients are truncated to 16 bits allocated with an average of 4 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2. The following numbers of signed-digits are allocated to the lattice coefficients:

```
A1k0_allocsd_digits = [  4,  3,  5,  5, ...
                         4,  4 ]';

A2k0_allocsd_digits = [  3,  3,  5,  5, ...
                         3 ]';
```

The signed-digit lattice coefficients found by the heuristic of *Ito et al.* are:

```
A1k0_sd = [   -19712,    32384,   -25792,    28320, ...
              -23520,    11808 ]'/32768;

A2k0_sd = [   -22528,    30736,   -26768,    23872, ...
              -11776 ]'/32768;
```

The signed-digit lattice coefficients found by the branch-and-bound search are:

```
A1k_min = [   -19584,    32384,   -25648,    28180, ...
              -23296,    11328 ]'/32768;

A2k_min = [   -22528,    30722,   -26544,    23684, ...
              -11264 ]'/32768;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 44 signed-digits and 33 shift-and-add operations.

Figure 14.14 shows the amplitude pass-band response of the filter with 16-bit 4-signed-digit coefficients allocated with the algorithm of *Ito et al.* and branch-and-bound search. Figure 14.15 shows the amplitude stop-band response. Table 14.6 compares the cost and the number of 16 bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the branch-and-bound search.

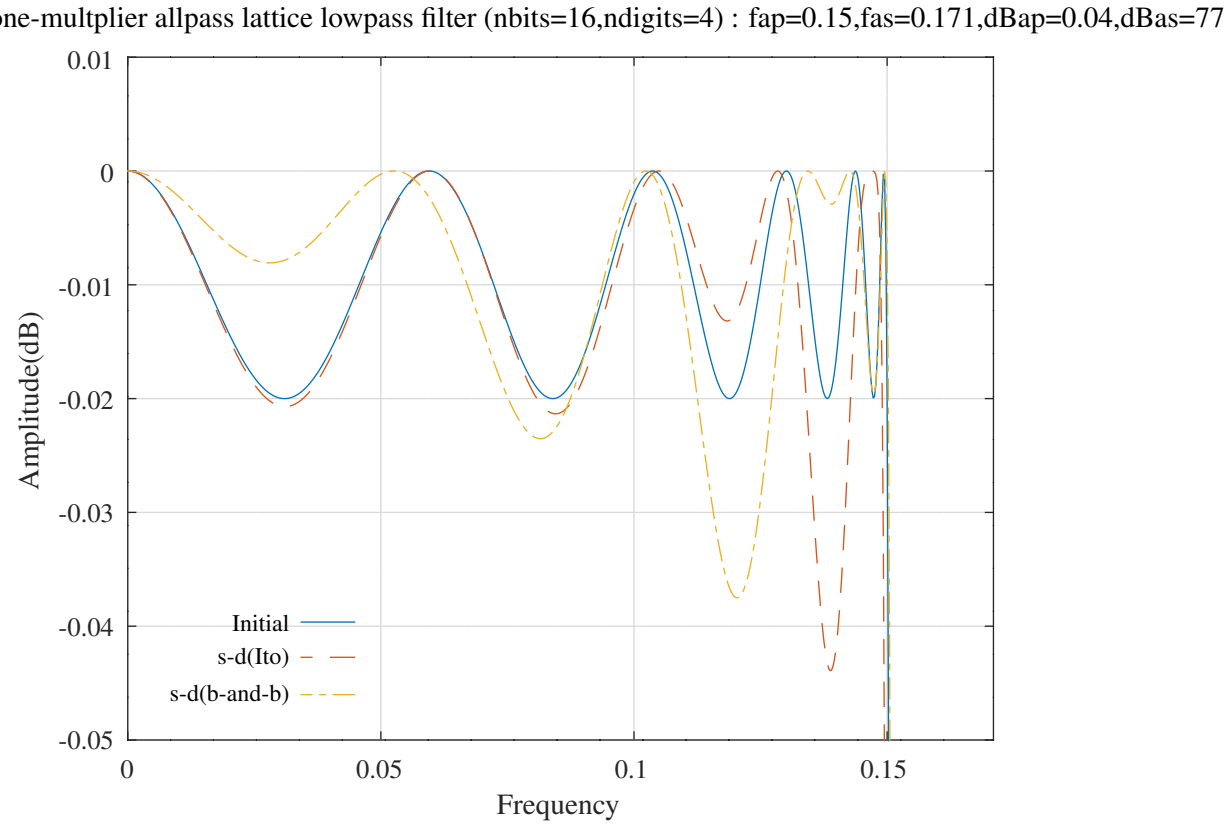



Figure 14.14: Pass band amplitude response of a parallel Schur one-multiplier all-pass lattice elliptic approximation low-pass filter with 16 bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

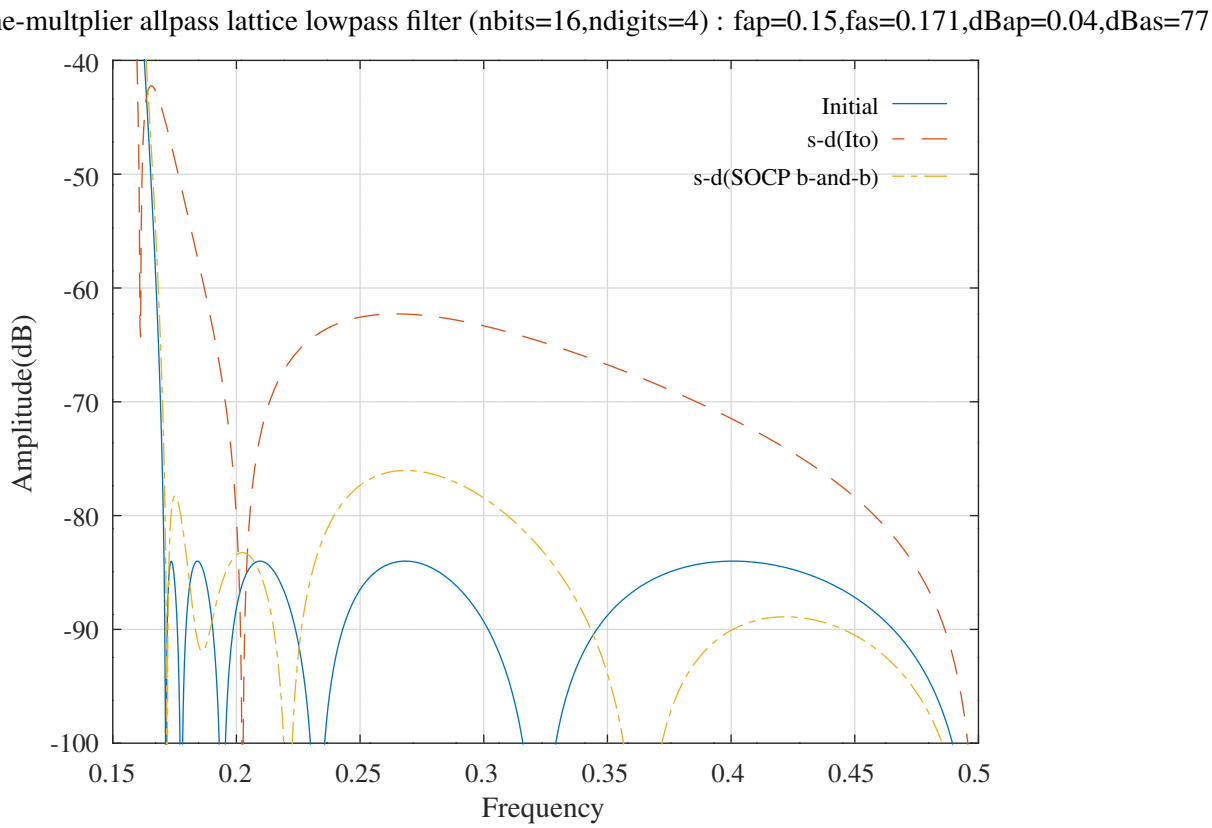



Figure 14.15: Stop band amplitude response of a parallel Schur one-multiplier all-pass lattice elliptic approximation low-pass filter with 16 bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Initial | 7.49e-06 | | |
| 16-bit 4-signed-digit(Ito) | 1.67e-04 | 44 | 33 |
| 16-bit 4-signed-digit(SOCP b-and-b) | 9.46e-06 | 44 | 33 |

Table 14.6: Comparison of the cost and number of 16-bit shift-and-add operations required to implement the coefficient multiplications for a parallel Schur one-multiplier all-pass lattice elliptic approximation low-pass filter with 16 bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

## 14.7 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass IIR filter

The Octave script *branch_bound_schurOneMPAlattice_bandpass_12_nbits_test.m* performs branch-and-bound search to optimise the response of the parallel Schur one-multiplier all-pass lattice band-pass filter of Section 10.3.5 with 12-bit integer coefficients. The filter specification is:

```
use_best_branch_and_bound_found=1
enforce_pcls_constraints_on_final_filter=1
branch_bound_schurOneMPAlattice_bandpass_12_nbits_test_allocsd_Lim=0
branch_bound_schurOneMPAlattice_bandpass_12_nbits_test_allocsd_Ito=1
nbits=12 % Coefficient word length
ndigits=3 % Average number of signed digits per coef.
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-06 % Tolerance on constraints
n=1000 % Frequency points across the band
difference=1 % Use difference of all-pass filters
m1=10 % Allpass model filter 1 denominator order
m2=10 % Allpass model filter 2 denominator order
rho=0.992188 % Constraint on allpass coefficients
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
dBap=2.000000 % Pass band amplitude response ripple(dB)
Wap=0.25 % Pass band amplitude response weight
Watl=0.1 % Lower transition band amplitude response weight
Watu=0.1 % Upper transition band amplitude response weight
fasl=0.05 % Stop band amplitude response lower edge
fasu=0.25 % Stop band amplitude response upper edge
dBas=40.000000 % Stop band amplitude response ripple(dB)
Wasl=10000 % Lower stop band amplitude response weight
Wasu=10000 % Upper stop band amplitude response weight
ftpl=0.09 % Pass band group-delay response lower edge
ftpu=0.21 % Pass band group-delay response upper edge
td=16.000000 % Pass band nominal group-delay response(samples)
tdr=0.200000 % Pass band group-delay response ripple(samples)
Wtp=1 % Pass band group-delay response weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2. 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 SOCP PCLS optimised with the filter specification given above.

The signed-digit lattice coefficients found by the heuristic of *Lim et al.* are:

```
A1k0_sd = [     -904,     1688,      128,     -672, ...
                1115,     -384,     -388,      904, ...
                -584,      340 ]'/2048;


A2k0_sd = [    -1568,     1773,       64,     -768, ...
                1152,     -208,     -288,      880, ...
                -640,      312 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are:

```
A1k_min = [     -988,     1605,      128,     -832, ...
                1205,     -576,     -184,      766, ...
                -504,      257 ]'/2048;


A2k_min = [    -1600,     1760,       32,     -896, ...
                1280,     -432,     -160,      736, ...
                -528,      255 ]'/2048;
```

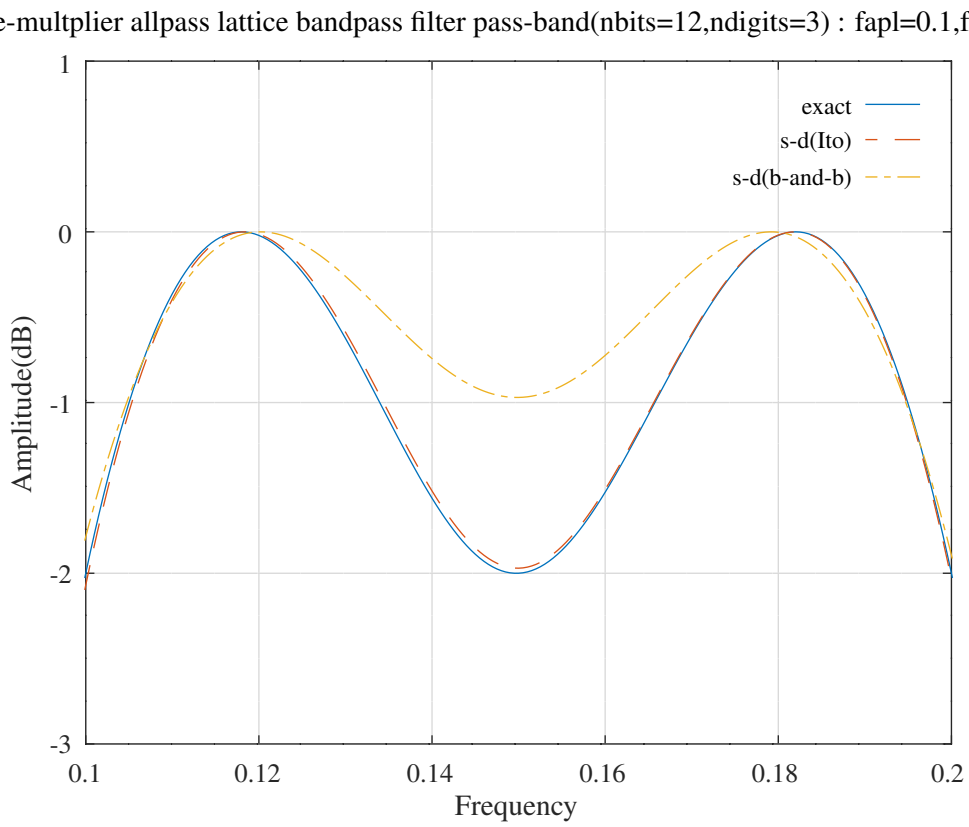



Figure 14.16: Pass band amplitude response for a parallel Schur one-multiplier all-pass lattice band-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 53 signed-digits and 33 shift-and-add operations.

Figure 14.16 shows the amplitude pass-band response of the filter with 12-bit 3-signed-digit coefficients allocated with the algorithm of *Ito et al.* and branch-and-bound search. Figure 14.17 shows the corresponding filter pass-band group-delay response. Figure 14.18 shows the corresponding filter stop-band amplitude response. 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 signed-digit allocation heuristic of *Ito et al.* with the branch-and-bound search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.011087 | | |
| 12-bit 3-signed-digit(Ito) | 0.012160 | 58 | 38 |
| 12-bit 3-signed-digit(SOCP b-and-b) | 0.007570 | 53 | 33 |

Table 14.7: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a parallel Schur one-multiplier all-pass lattice band-pass filter with 12 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.

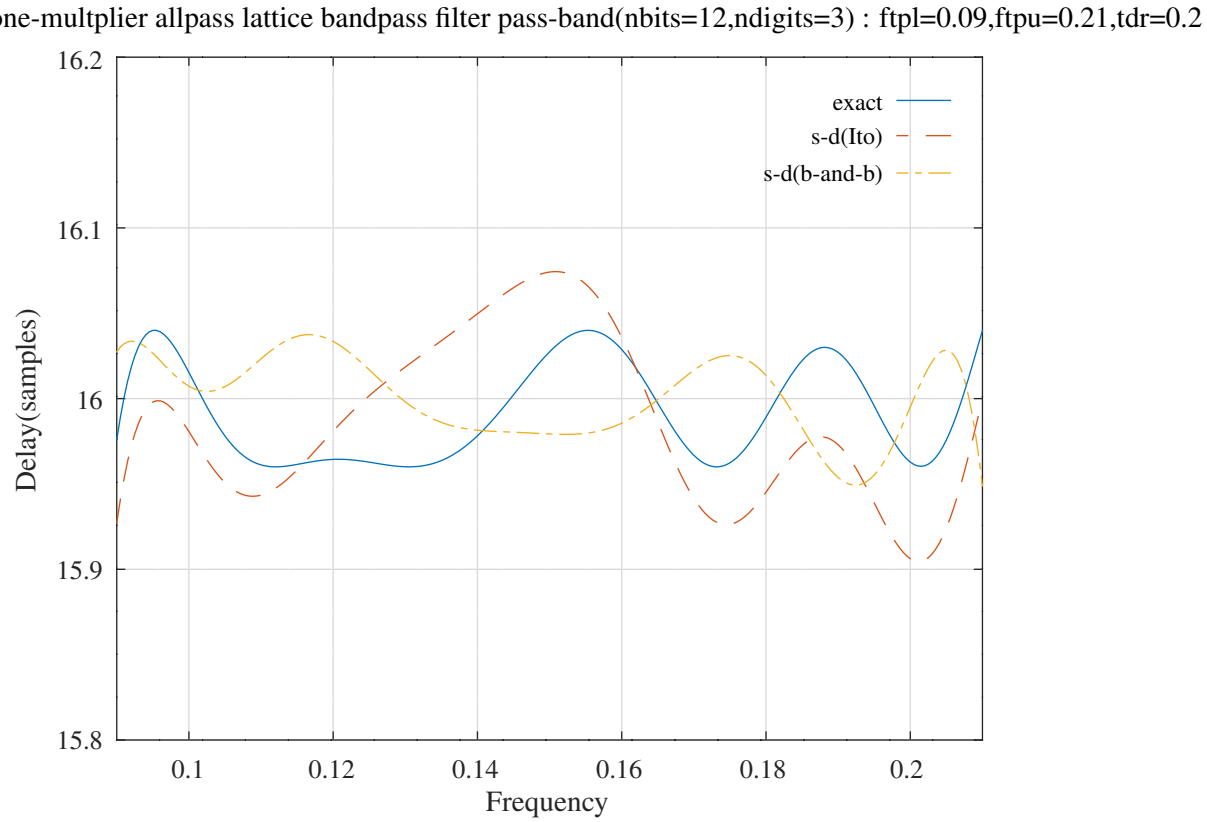



Figure 14.17: Pass band group delay response for a parallel Schur one-multiplier all-pass lattice band-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

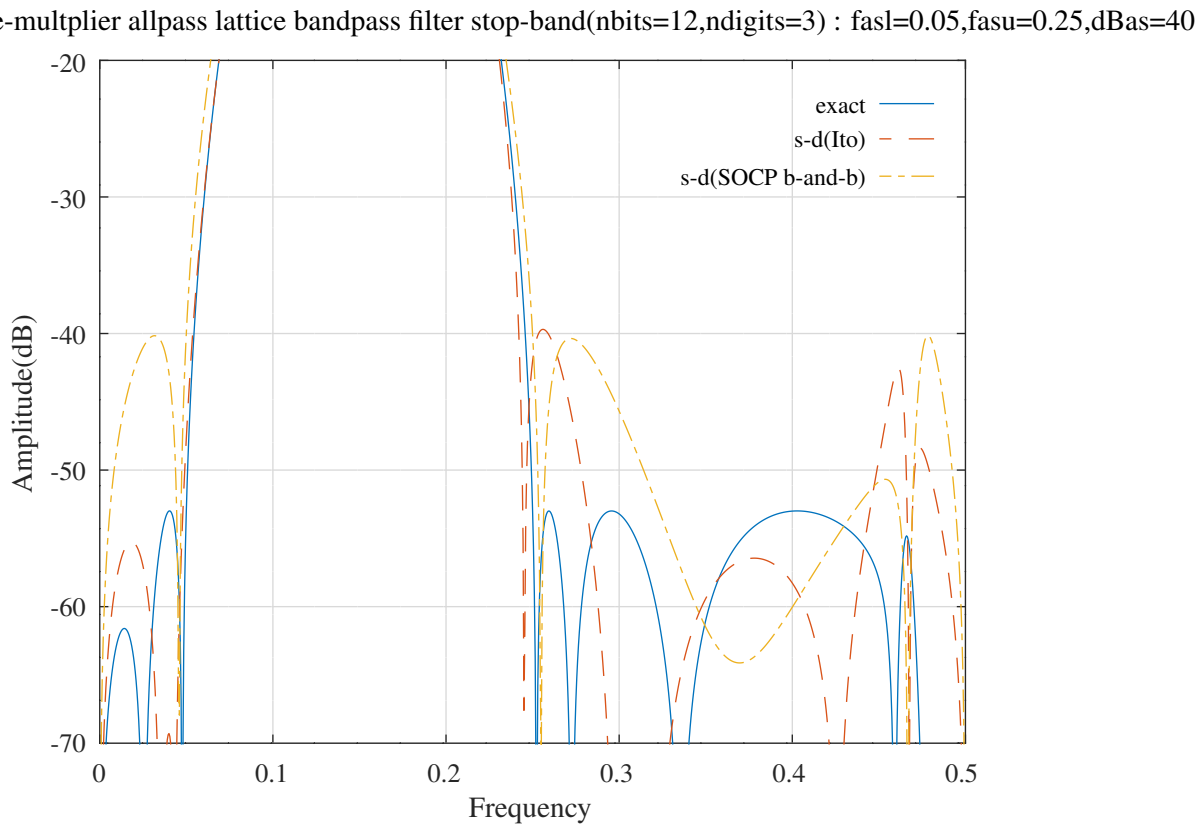



Figure 14.18: Stop band amplitude response for a parallel Schur one-multiplier all-pass lattice band-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

## 14.8 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass Hilbert IIR filter

The Octave script *branch_bound_schurOneMPAlattice_bandpass_hilbert_10_nbits_test.m* performs branch-and-bound search to optimise the response of the parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter of Section 10.3.10 with 10-bit integer coefficients. The filter specification is:

```
use_best_branch_and_bound_found=1
enforce_pcls_constraints_on_final_filter=1
nbits=10 % Coefficient word length
ndigits=3 % Average number of signed digits per coef.
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
difference=1 % Use difference of all-pass filters
m1=10 % Allpass model filter 1 denominator order
m2=10 % Allpass model filter 2 denominator order
rho=0.999000 % Constraint on allpass coefficients
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
dBap=0.200000 % Pass band amplitude response ripple(dB)
Wap=10 % Pass band amplitude response weight
Watl=0.001 % Lower transition band amplitude response weight
Watu=0.001 % Upper transition band amplitude response weight
fasl=0.05 % Stop band amplitude response lower edge
fasu=0.25 % Stop band amplitude response upper edge
dBas=35.000000 % Stop band amplitude response ripple(dB)
Wasl=50000 % Lower stop band amplitude response weight
Wasu=5000 % Upper stop band amplitude response weight
ftpl=0.11 % Pass band group-delay response lower edge
ftpu=0.19 % Pass band group-delay response upper edge
td=16.000000 % Pass band nominal group-delay response(samples)
tdr=0.200000 % Pass band group-delay response ripple(samples)
Wtp=2 % Pass band group-delay response weight
fppl=0.11 % Pass band phase response lower edge
fppu=0.19 % Pass band phase response upper edge
pd=3.500000 % Pass band nominal phase response(rad./pi)
pdr=0.040000 % Pass band phase response ripple(rad./pi)
Wpp=10 % Pass band phase response weight
```

The filter coefficients are truncated to 10 bits with 3 signed-digits. 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 SOCP PCLS optimised with the filter specification given above.

The 12-bit, 3-signed-digit, lattice coefficients found by the branch-and-bound search are:

```
A1k_min = [     -250,      432,     -168,       94, ...
                 328,     -208,      148,      108, ...
                 -70,       44 ]'/512;


A2k_min = [     -416,      452,     -216,       88, ...
                 336,     -194,      152,      108, ...
                 -58,       50 ]'/512;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 60 signed-digits and 40 shift-and-add operations.

Figure 14.19 shows the amplitude pass-band response of the filter with 10-bit, 3-signed-digit coefficients. Figure 14.20 shows the corresponding filter pass-band phase response (in multiples of $\pi$ and adjusted for the nominal delay). Figure 14.21 shows the corresponding filter pass-band group-delay response. Figure 14.22 shows the corresponding filter stop-band amplitude response. Table 14.8 compares the cost and the number of 12 bit shift-and-add operations required to implement the coefficient multiplications found with the branch-and-bound search.

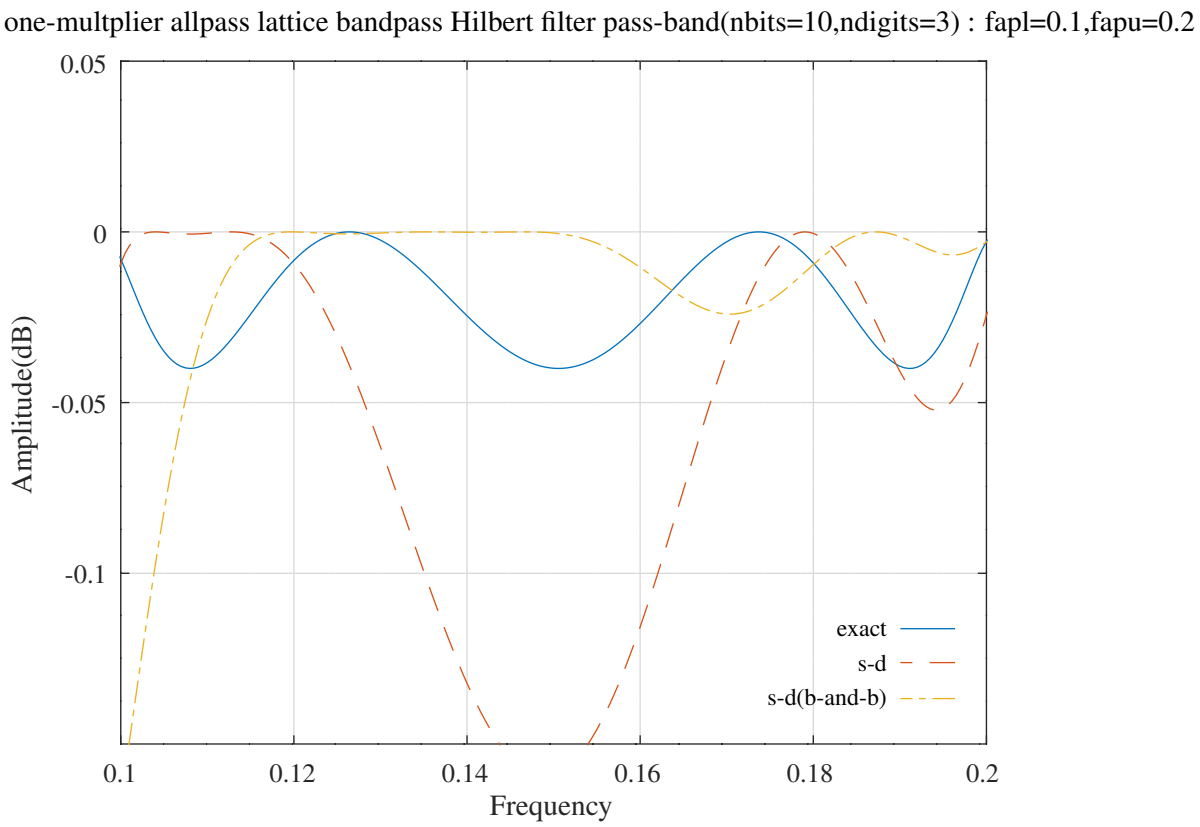



Figure 14.19: Pass band amplitude response for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 10 bit 3-signed-digit coefficients found by branch-and-bound search.

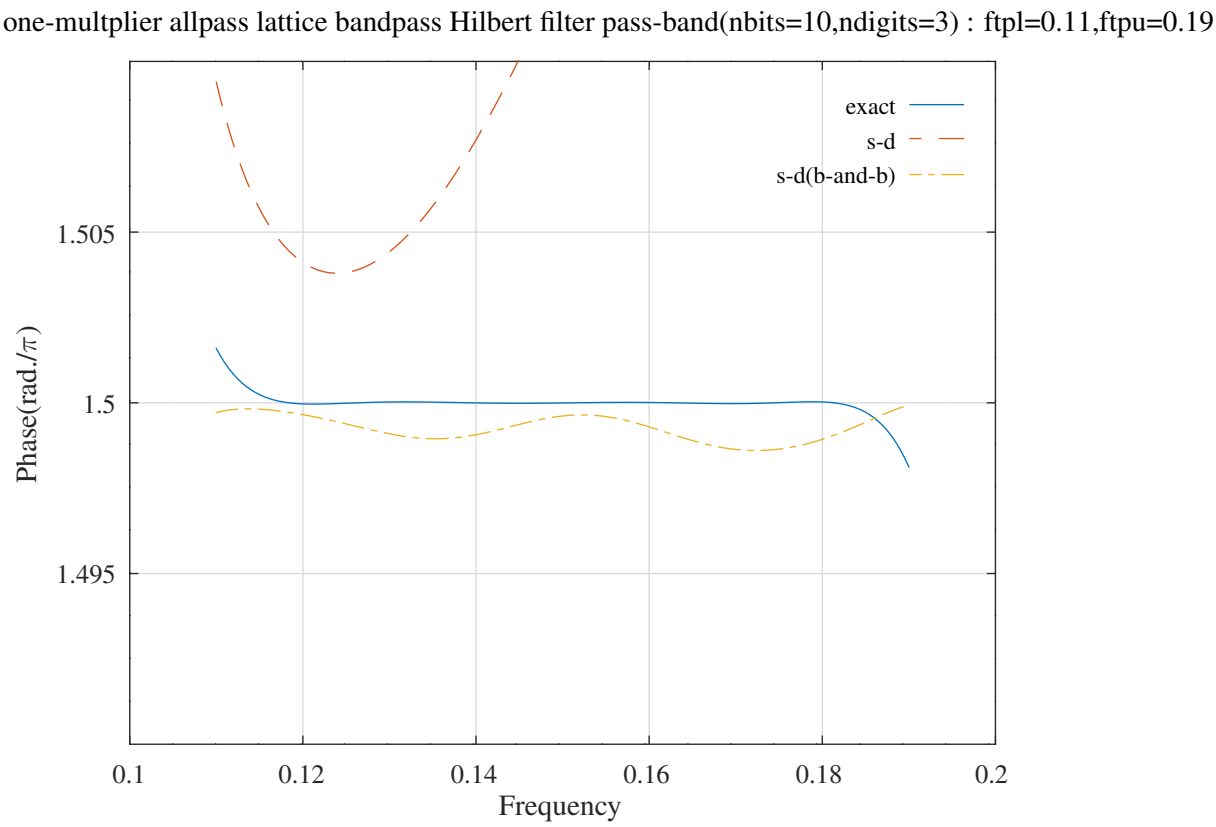



Figure 14.20: Pass band phase response for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 10 bit 3-signed-digit coefficients found by performing branch-and-bound search. The phase response shown is adjusted for the nominal delay.

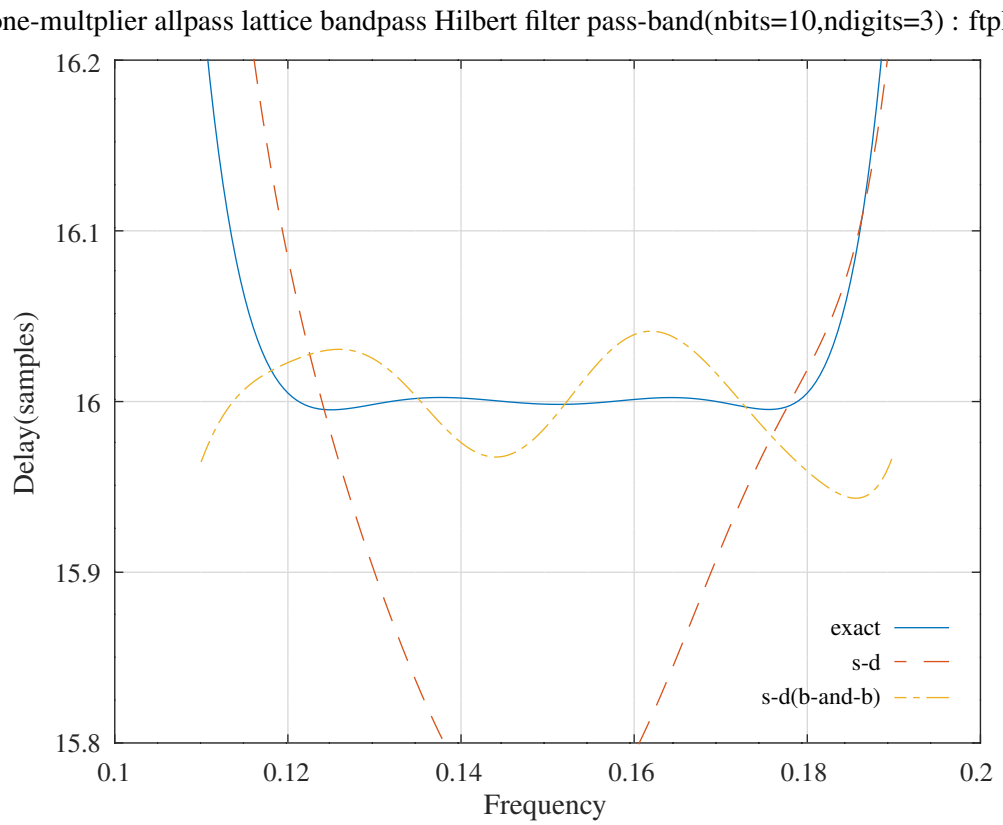



Figure 14.21: Pass band group delay response for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 10 bit 3-signed-digit coefficients found by branch-and-bound search.

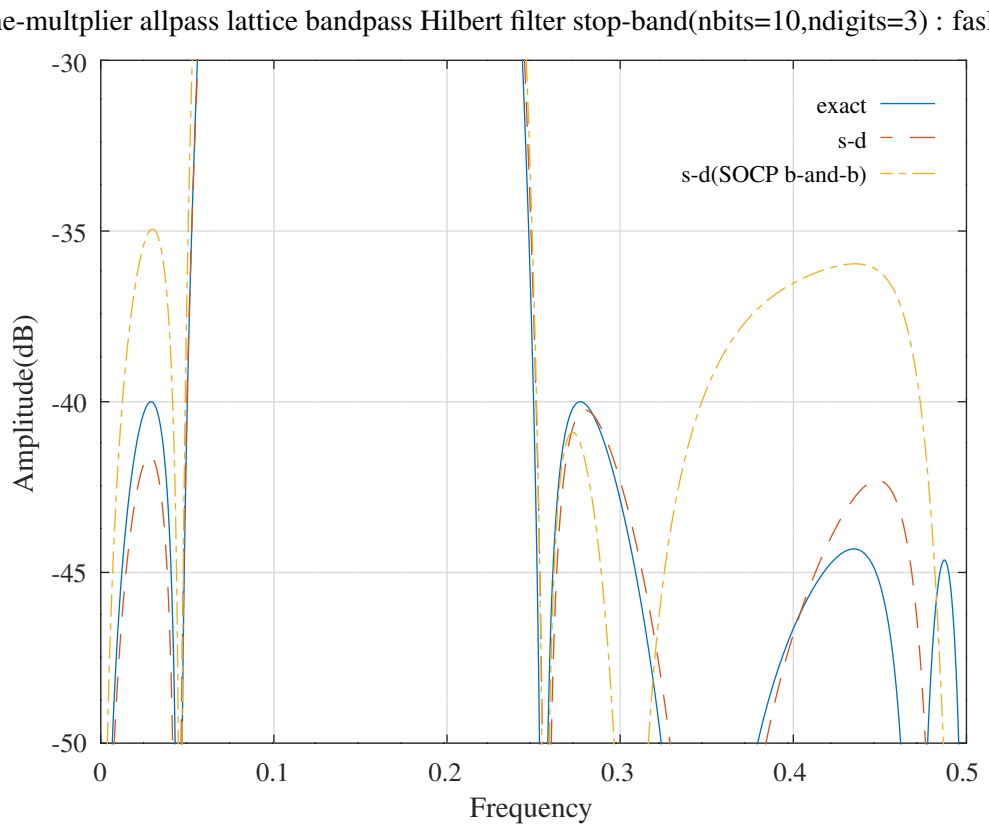



Figure 14.22: Stop band amplitude response for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 10 bit 3-signed-digit coefficients found by branch-and-bound search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.003782 | | |
| 10-bit 3-signed-digit | 0.047160 | 59 | 39 |
| 10-bit 3-signed-digit(SOCP b-and-b) | 0.002236 | 60 | 40 |

Table 14.8: Summary of cost results for the parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 10 bit 3-signed-digit coefficients found by branch-and-bound search.

## 14.9 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass Hilbert IIR filter

The Octave script *branch_bound_schurOneMPAlattice_bandpass_hilbert_12_nbits_test.m* performs branch-and-bound search to optimise the response of the parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter of Section 10.3.10 with 12-bit integer coefficients. The filter specification is:

```
use_best_branch_and_bound_found=1
enforce_pcls_constraints_on_final_filter=1
branch_bound_schurOneMPAlattice_bandpass_hilbert_12_nbits_test_allocsd_Lim=0
branch_bound_schurOneMPAlattice_bandpass_hilbert_12_nbits_test_allocsd_Ito=1
nbits=12 % Coefficient word length
ndigits=3 % Average number of signed digits per coef.
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
difference=1 % Use difference of all-pass filters
m1=10 % Allpass model filter 1 denominator order
m2=10 % Allpass model filter 2 denominator order
rho=0.999000 % Constraint on allpass coefficients
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
dBap=0.050000 % Pass band amplitude response ripple(dB)
Wap=1 % Pass band amplitude response weight
Watl=0.001 % Lower transition band amplitude response weight
Watu=0.001 % Upper transition band amplitude response weight
fasl=0.05 % Stop band amplitude response lower edge
fasu=0.25 % Stop band amplitude response upper edge
dBas=35.000000 % Stop band amplitude response ripple(dB)
Wasl=20000 % Lower stop band amplitude response weight
Wasu=5000 % Upper stop band amplitude response weight
ftpl=0.11 % Pass band group-delay response lower edge
ftpu=0.19 % Pass band group-delay response upper edge
td=16.000000 % Pass band nominal group-delay response(samples)
tdr=0.200000 % Pass band group-delay response ripple(samples)
Wtp=10 % Pass band group-delay response weight
fppl=0.11 % Pass band phase response lower edge
fppu=0.19 % Pass band phase response upper edge
pd=3.500000 % Pass band nominal phase response(rad./pi)
pdr=0.020000 % Pass band phase response ripple(rad./pi)
Wpp=500 % Pass band phase response weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2. At each branch the script fixes the coefficient with the largest difference between upper and lower 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 SOCP PCLS optimised with the filter specification given above.

The 12-bit, average of 3-signed-digit, lattice coefficients allocated with the algorithm of *Ito et al.* are:

```
A1k0_sd = [     -976,     1745,     -736,      384, ...
                1312,     -880,      544,      368, ...
                -312,      184 ]'/2048;

A2k0_sd = [    -1616,     1832,     -836,      352, ...
                1344,     -832,      544,      368, ...
                -288,      192 ]'/2048;
```

The 12-bit, average of 3-signed-digit, lattice coefficients allocated with the algorithm of *Ito et al.* found by the branch-and-bound search are:

```
A1k_min = [     -988,     1728,     -704,      384, ...
                1296,     -888,      576,      352, ...
                -272,      137 ]'/2048;
```

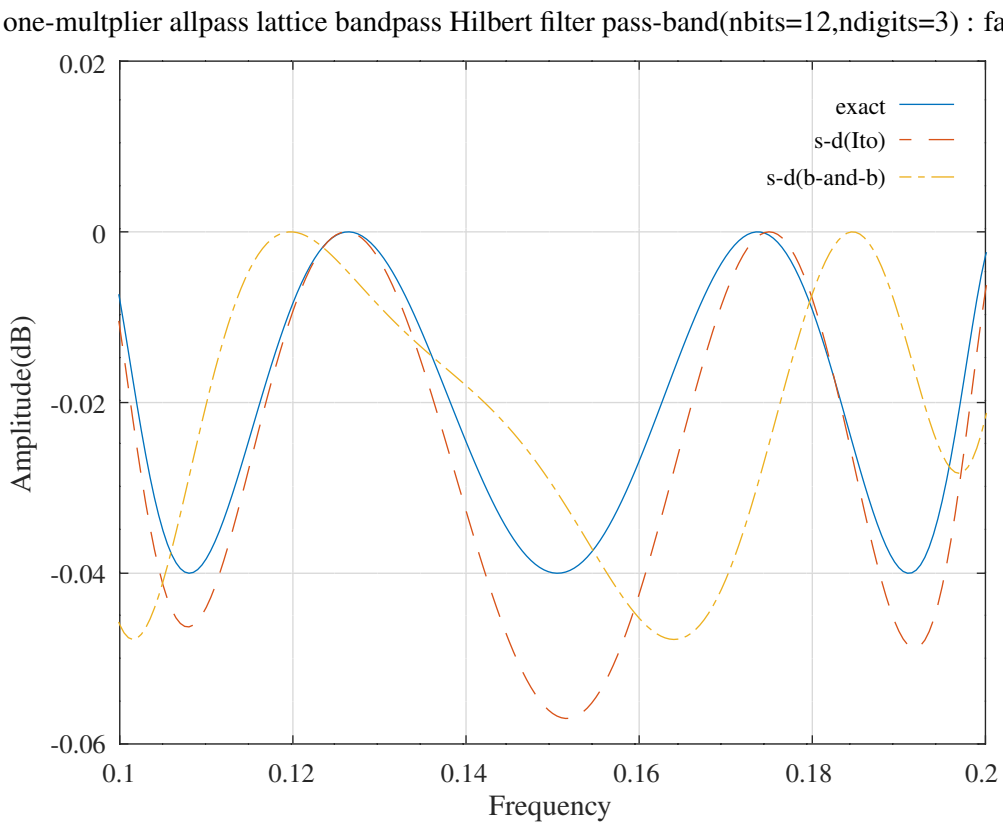



Figure 14.23: Pass band amplitude response for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

```
A2k_min = [     -1632,      1808,      -848,       304, ...
                 1344,      -832,       576,       368, ...
                 -240,       160 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 56 signed-digits and 36 shift-and-add operations.

Figure 14.23 shows the amplitude pass-band response of the filter with 12-bits 3-signed-digit coefficients allocated with the algorithm of *Ito et al.* and branch-and-bound search. Figure 14.24 shows the corresponding filter pass-band phase response (in multiples of $\pi$ and adjusted for the nominal delay). Figure 14.25 shows the corresponding filter pass-band group-delay response. Figure 14.26 shows the corresponding filter stop-band amplitude response. Table 14.9 compares the cost and the number of 12 bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the branch-and-bound search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.017168 | | |
| 12-bit 3-signed-digit(Ito) | 0.112595 | 60 | 40 |
| 12-bit 3-signed-digit(SOCP b-and-b) | 0.000972 | 56 | 36 |

Table 14.9: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 12 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.

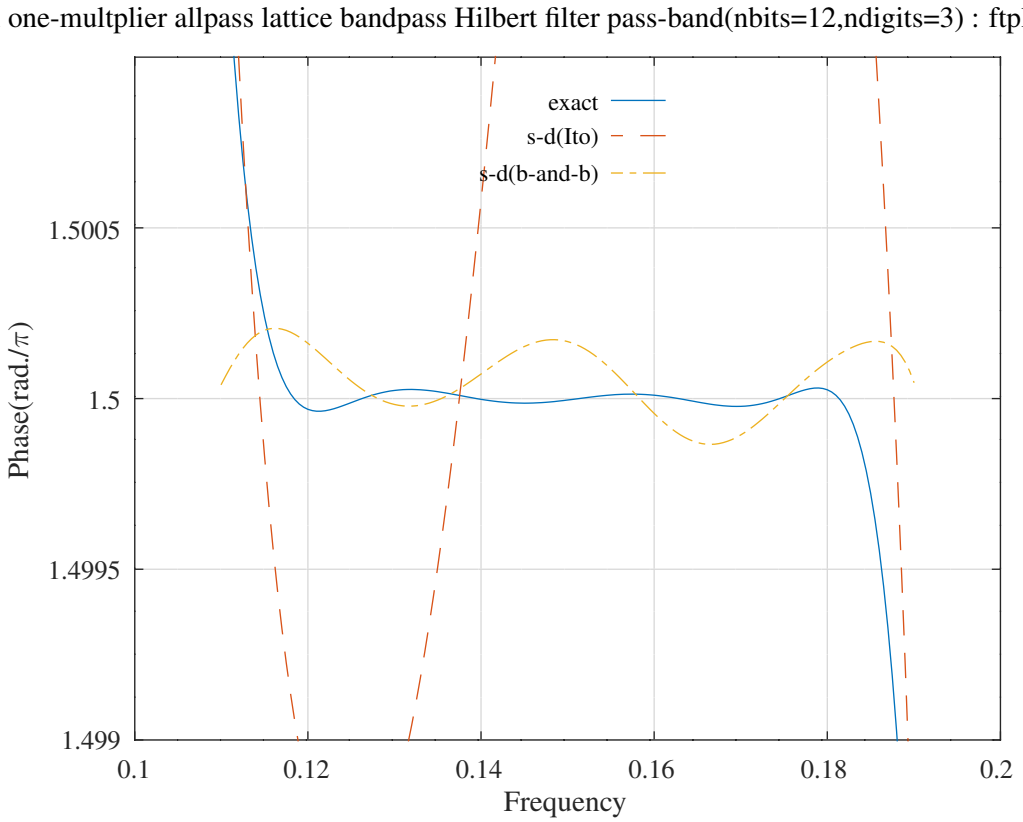



Figure 14.24: Pass band phase response for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search. The phase response shown is adjusted for the nominal delay.

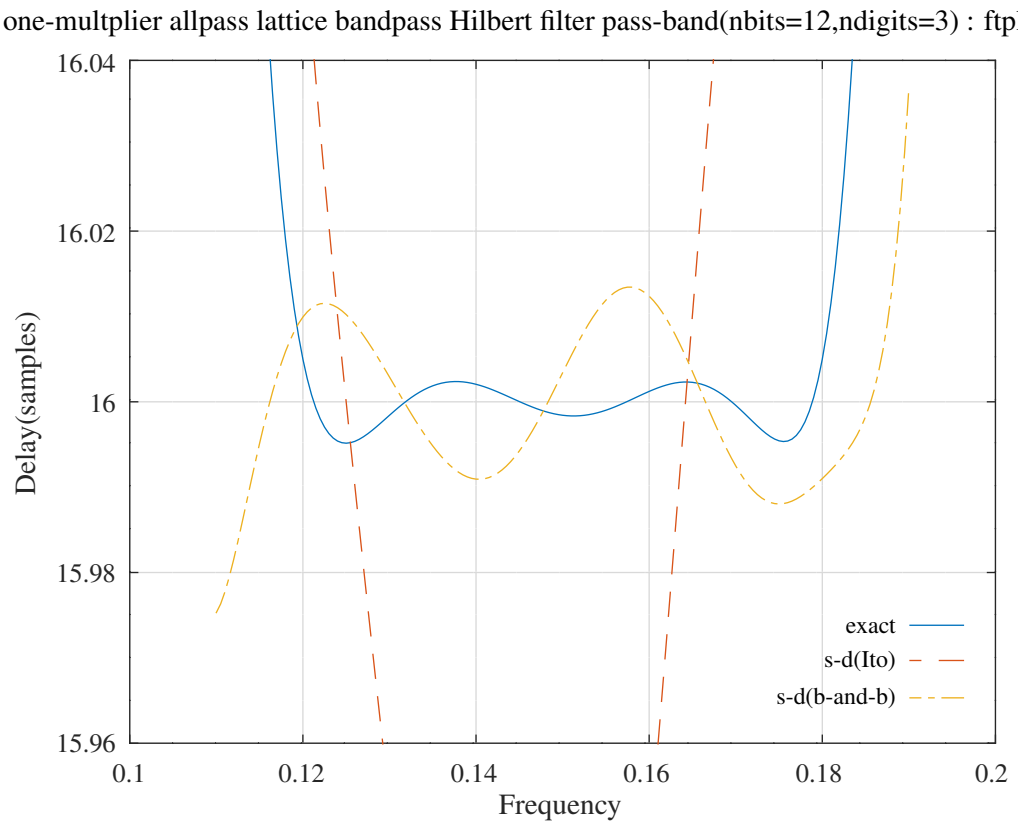



Figure 14.25: Pass band group-delay response for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

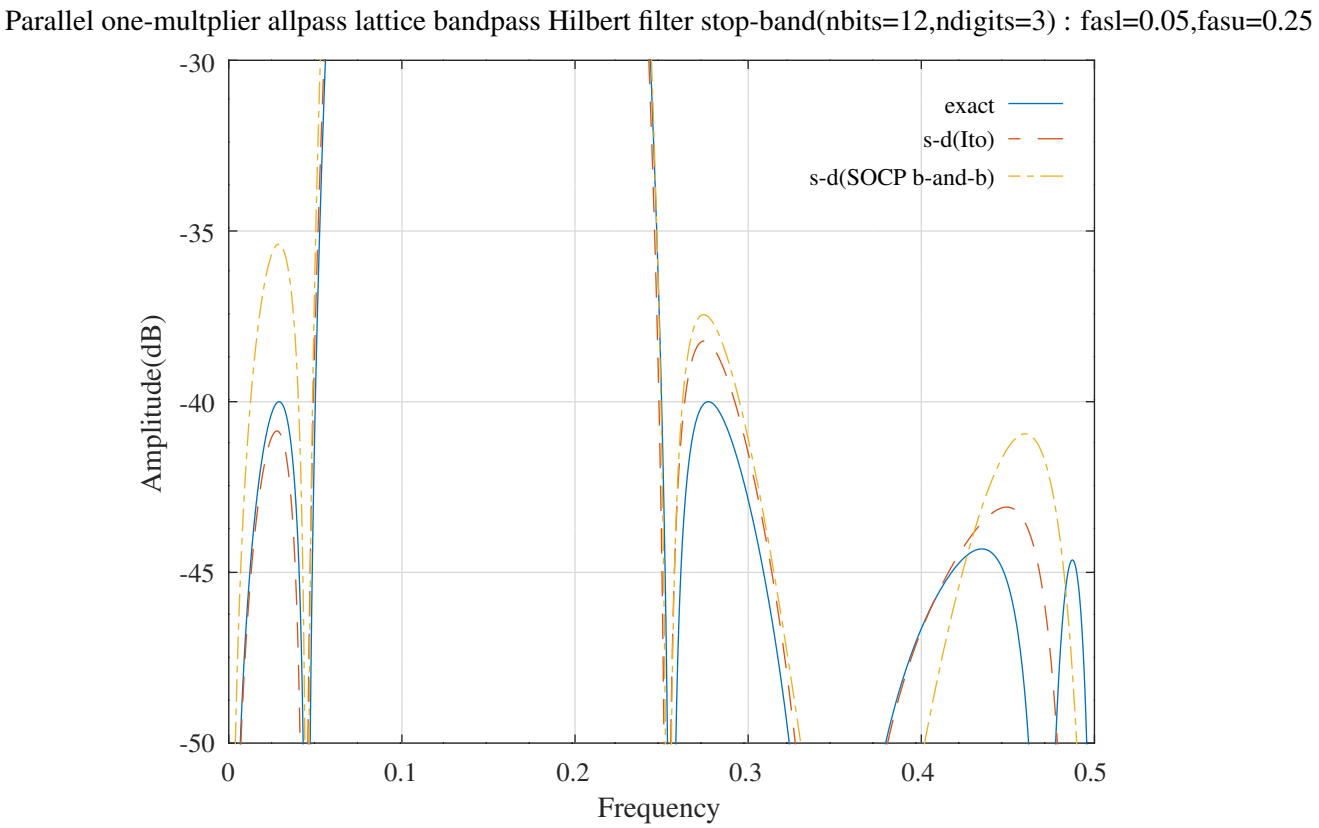



Figure 14.26: Stop band amplitude response for a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing branch-and-bound search.

## 14.10 Branch-and-bound search for the coefficients of an FRM low-pass filter implemented with 12-bits and an average of 3-signed-digits

The Octave script *branch_bound_schurOneMAPlattice_frm_12_nbits_test.m* uses the branch-and-bound heuristic to optimise the response of the FRM low-pass filter of Section 10.4.6 with 12-bit coefficients each having an average of 3 signed-digits allocated by the method of *Lim et al.*. The number of signed-digits allocated to the coefficients of the all-pass lattice filter are:

```
k_allocsd_digits = [  2,  4,  3,  4, ...
                      2,  4,  3,  4, ...
                      2,  4 ]';
```

The number of signed-digits allocated to the distinct coefficients of the FIR masking filter are:

```
u_allocsd_digits = [  5,  4,  3,  4, ...
                      4,  3,  4,  4, ...
                      4,  2,  4,  3, ...
                      3,  3,  2,  2, ...
                      1,  3,  3,  1, ...
                      1 ]';
```

The number of signed-digits allocated to the distinct coefficients of the FIR complementary masking filter are:

```
v_allocsd_digits = [  5,  4,  4,  2, ...
                      4,  3,  3,  4, ...
                      4,  1,  4,  4, ...
                      2,  2,  3,  2, ...
                      2,  2,  2,  2, ...
                      2 ]';
```

The filter specification is:

```
n=1000 % Frequency points across the band
tol=0.0005 % Tolerance on coefficient update vector
ctol=5e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
mr=10 % Allpass model filter denominator order
Mmodel=9 % Model filter FRM decimation factor
Dmodel=9 % Model filter nominal pass band group delay
dmask=20 % FIR masking filter delay
fap=0.3 % Magnitude-squared pass band edge
dBap=1 % Pass band magnitude peak-to-peak ripple
Wap=1 % Pass band magnitude-squared weight
fas=0.3105 % Magnitude-squared stop band edge
dBas=38 % Stop band magnitude minimum attenuation
Was=10 % Stop band magnitude-squared weight
ftp=0.3 % Delay pass band edge
tp=101 % Pass band nominal delay
tpr=tp/126.25 % Pass band delay peak-to-peak ripple
Wtp=1 % Pass band magnitude-squared weight
fpp=0.3 % Phase pass band edge
ppr=0.04*pi % Pass band phase peak-to-peak ripple (rad./pi)
Wpp=0.01 % Phase pass band weight
rho=0.992188 % Constraint on allpass pole radius
```

The filter coefficients found by the branch-and-bound search[b] are:

```
k_min = [      -24,     1188,       35,     -295, ...
                -9,      117,       16,      -53, ...
                -8,       20 ]'/2048;
```

[b]The search was stopped after about 36 hours.

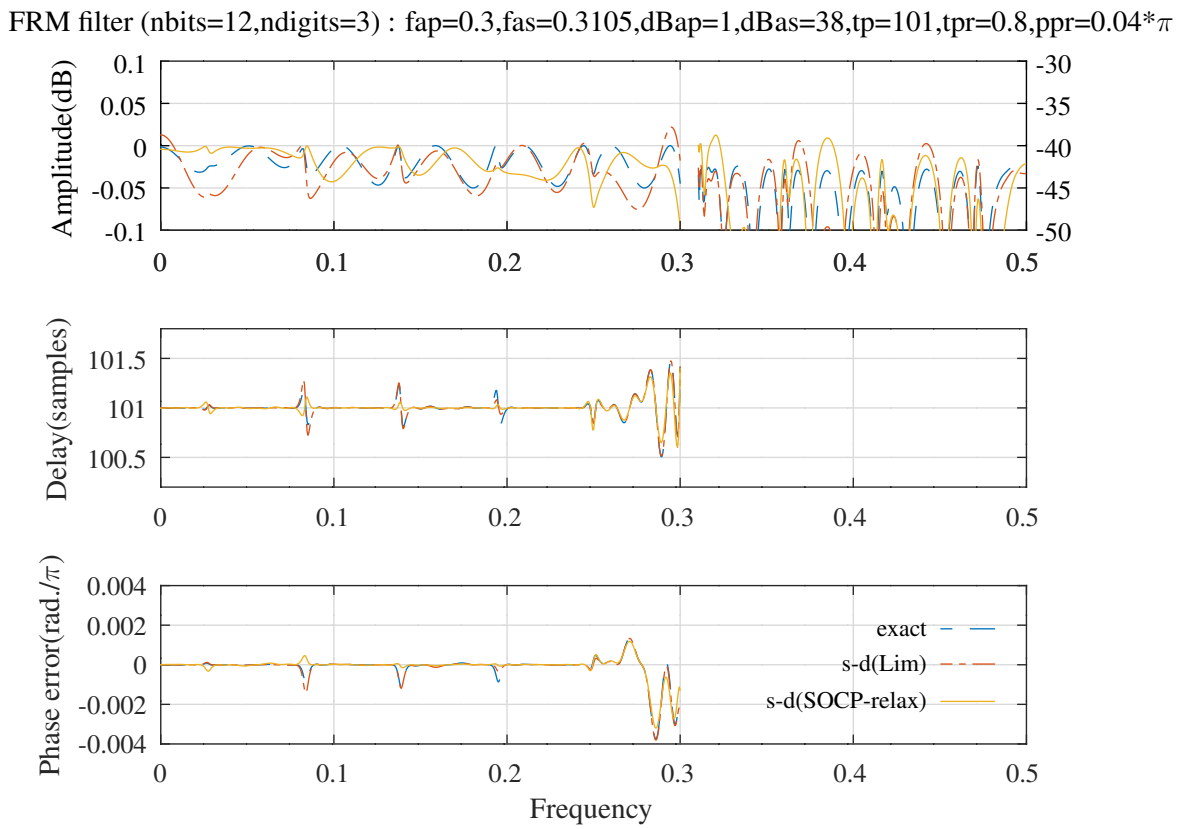



Figure 14.27: Comparison of the amplitude, delay and phase responses of an FRM low-pass filter with exact and 12 bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by branch-and-bound search

```
epsilon_min = [  1,  1, -1,  1, ...
                 1, -1, -1,  1, ...
                 1, -1 ];


u_min = [     1179,      620,     -116,     -172, ...
               107,       65,      -87,      -20, ...
                76,      -34,      -28,       17, ...
                18,      -19,       -5,       14, ...
                 4,      -16,       14,       -2, ...
                -2 ]'/2048;


v_min = [    -1368,     -556,      268,       10, ...
              -133,      100,        9,      -74, ...
                60,       -4,      -35,       28, ...
                 5,      -20,       16,        4, ...
               -16,       14,       -5,       -4, ...
                 5 ]'/2048;
```

Figure 14.27, compares the amplitude, delay and phase responses of the FRM low-pass filter with exact and 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by branch-and-bound search. Table 14.10 compares the cost and the number of 12 bit shift-and-add operations required to implement the coefficient multiplications found by branch-and-bound search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.001723 | | |
| 12-bit 3-signed-digit(Lim) | 0.001813 | 126 | 74 |
| 12-bit 3-signed-digit(branch-and-bound) | 0.001061 | 126 | 74 |

Table 14.10: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for an FRM low-pass filter with 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by branch-and-bound search

## 14.11 Branch-and-bound search for the coefficients of an FRM Hilbert filter implemented with 12-bits and 2-signed-digits

The Octave script *branch_bound_schurOneMAPlattice_frm_hilbert_12_nbits_test.m* uses the branch-and-bound heuristic to optimise the response of the FRM Hilbert filter of Section 10.4.8 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
ctol=1e-05 % Tolerance on constraints
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:

```
k_min = [    -1152,     -264,      -96,      -36, ...
               -12 ]'/2048;


u_min = [        0,       -3,      -16,      -24, ...
               -63,      -72,     -112,     -120, ...
               896 ]'/2048;


v_min = [       14,        9,       18,        8, ...
               -12,      -63,     -160,     -640 ]'/2048;
```

Figures 14.28, 14.29 and 14.30 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 14.31 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 phase responses shown are adjusted for the nominal delay. 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 14.11 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.

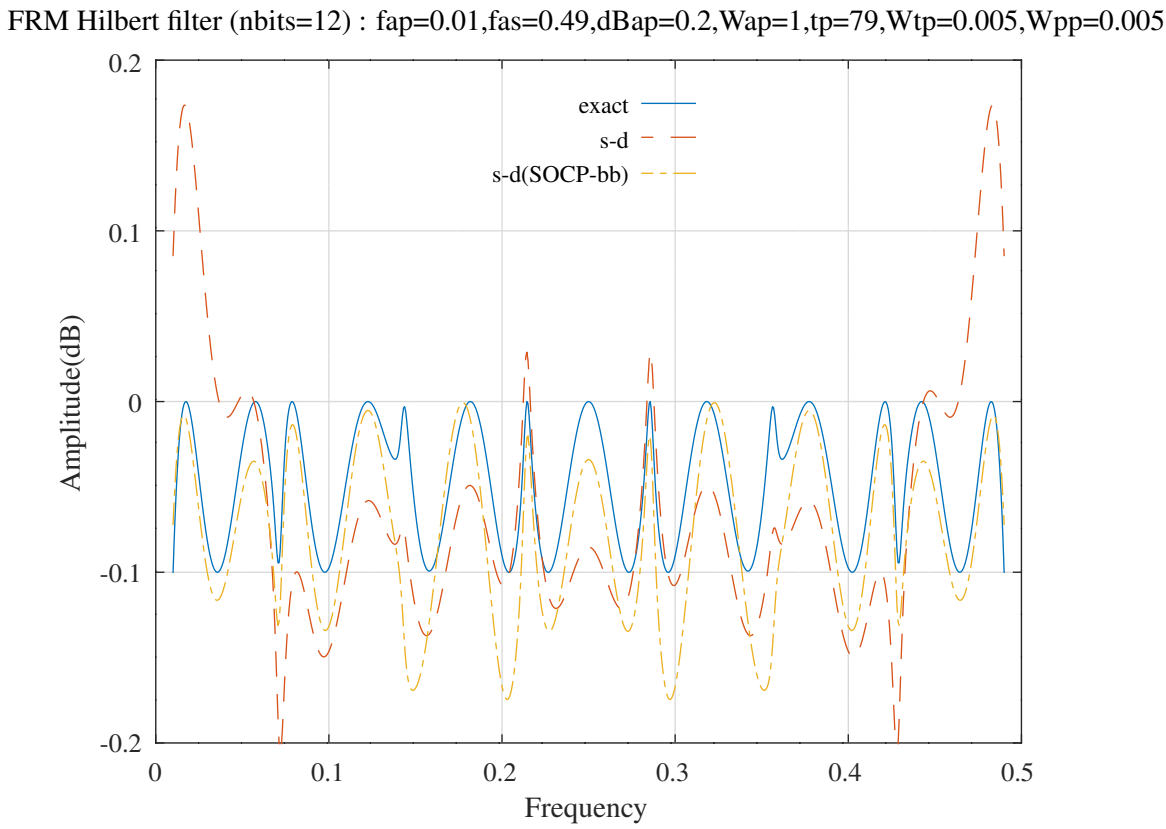



Figure 14.28: Comparison of the amplitude response of an FRM Hilbert filter with exact and 12 bit 2-signed digit integer coefficients found by branch-and-bound search

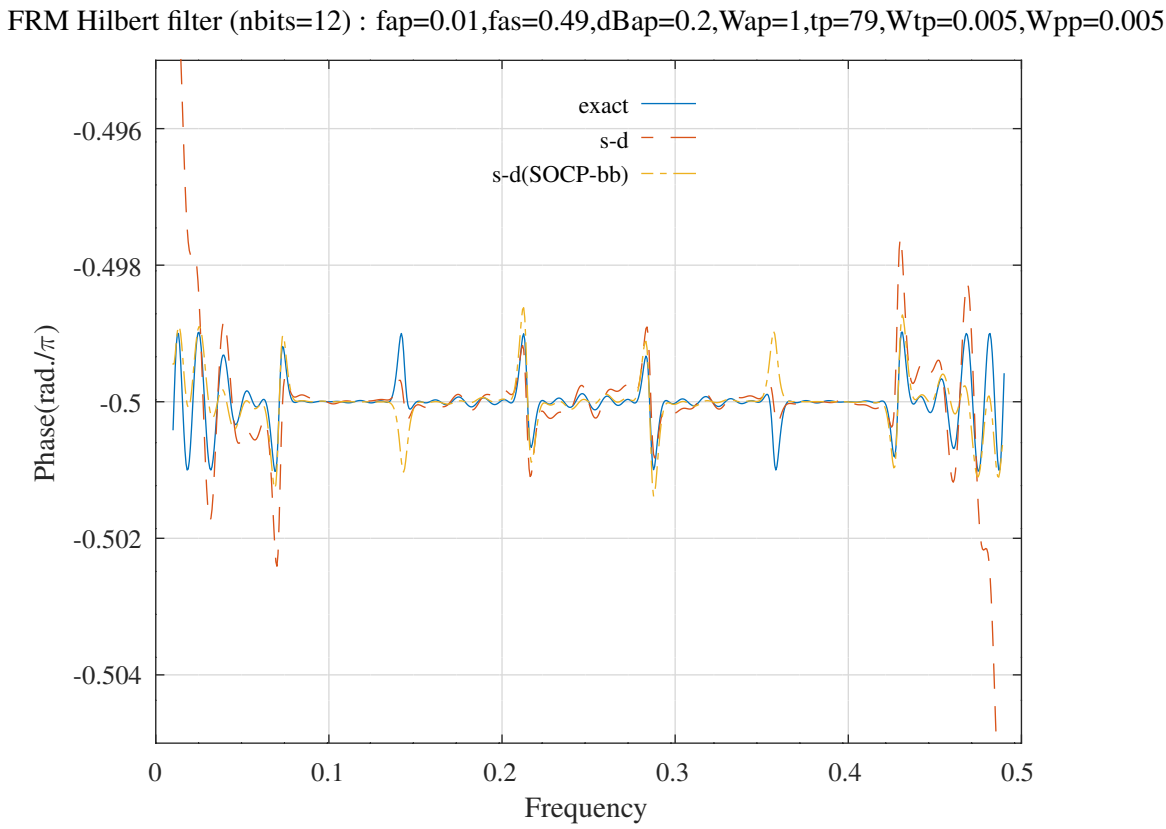



Figure 14.29: Comparison of the phase response of an FRM Hilbert filter with exact and 12 bit 2-signed digit integer coefficients found by branch-and-bound search. The phase response shown is adjusted for the nominal delay.

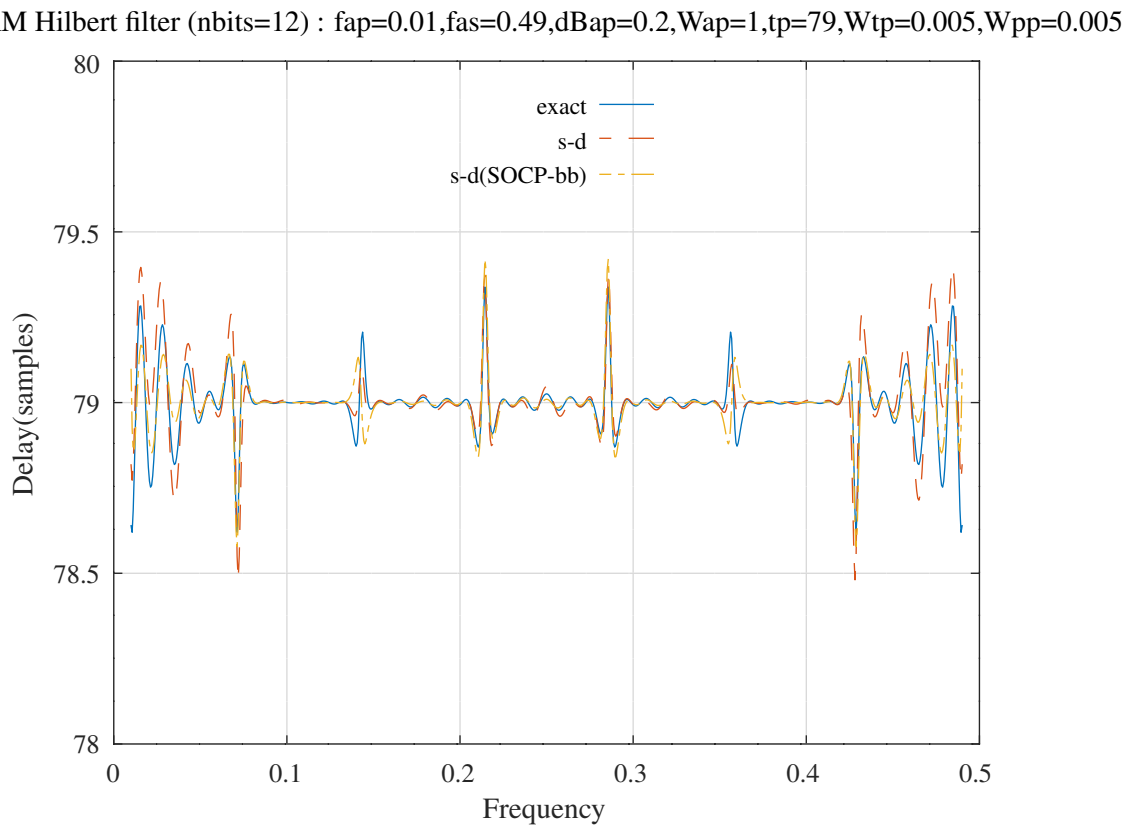



Figure 14.30: Comparison of the delay response of an FRM Hilbert filter with exact and 12 bit 2-signed digit integer coefficients found by branch-and-bound search.

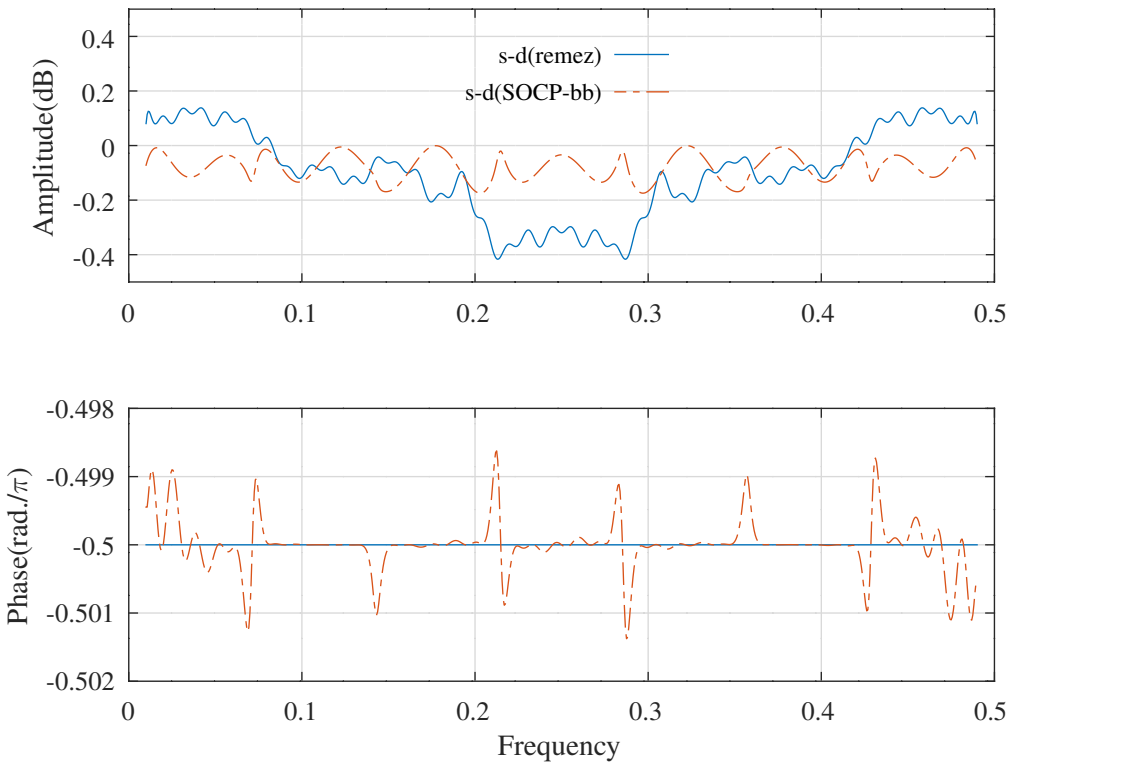



Figure 14.31: Comparison of the amplitude and phase responses of an FIR Hilbert filter with 12-bit 2-signed-digit coefficients and an FRM Hilbert filter with 12 bit 2-signed digit integer coefficients found by branch-and-bound search. The phase response shown is adjusted for the nominal delay.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.000656 | | |
| 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) | 0.005131 | 73 | 33 |

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 12-bit 2-signed-digit coefficients found by branch-and-bound search

## 14.12 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of an FIR Hilbert filter

The Octave script *branch_bound_directFIRhilbert_12_nbits_test.m* uses the branch-and-bound heuristic to optimise the response of a direct-form FIR Hilbert filter with 12-bit 2-signed-digit coefficients having a similar specification to the FIR Hilbert filter of Appendix N.4.2. An average of 2 signed-digits are allocated to each non-zero coefficient with the heuristic of *Ito et al.*. The Octave script *directFIRhilbert_allocsd_test.m* compares the heuristics of *Lim et al.* and *Ito et al.* for coefficient wordlengths from 6 to 16 bits. The direct-form FIR Hilbert filter specification is:

```
M=40 % Number of distinct coefficients
nbits=12 % Coefficient bits
ndigits=2 % Nominal average coefficient signed-digits
tol=0.0001 % Tolerance on coefficient. update
maxiter=400 % iteration limit
fapl=0.01 % Amplitude pass band lower edge
fapu=0.49 % Amplitude pass band upper edge
Wap=1 % Amplitude pass band weight
Was=0 % Amplitude stop band weight
```

The filter has 40 distinct non-zero coefficients and the filter group-delay is 79 samples. The distinct non-zero filter coefficients found by the branch-and-bound search are:

```
hM_min = [        -1,        -1,        -1,        -1, ...
                  -2,        -2,        -2,        -3, ...
                  -4,        -4,        -4,        -4, ...
                  -8,        -8,        -8,        -8, ...
                 -12,       -12,       -16,       -16, ...
                 -18,       -20,       -24,       -28, ...
                 -30,       -32,       -36,       -40, ...
                 -48,       -52,       -60,       -68, ...
                 -80,       -96,      -112,      -140, ...
                -184,      -258,      -432,     -1304 ]'/2048;
```

Figure 14.32, compares the amplitude responses of the FIR Hilbert filter with exact, 12-bit 2-signed-digit coefficients, 12-bit 2-signed-digit coefficients with signed-digits allocated by the heuristic of *Ito et al.* and with the 12-bit 2-signed-digit coefficients found by branch-and-bound search. Table 14.12 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. The FIR filter structure requires an extra 80 additions.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 1.07e-06 | | |
| 12-bit 2-signed-digit | 4.25e-04 | 71 | 31 |
| 12-bit 2-signed-digit(Ito) | 1.88e-05 | 68 | 28 |
| 12-bit 2-signed-digit(branch-and-bound) | 1.71e-05 | 68 | 28 |

Table 14.12: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a direct-form FIR Hilbert filter with 12-bit 2-signed-digit coefficients found by branch-and-bound search

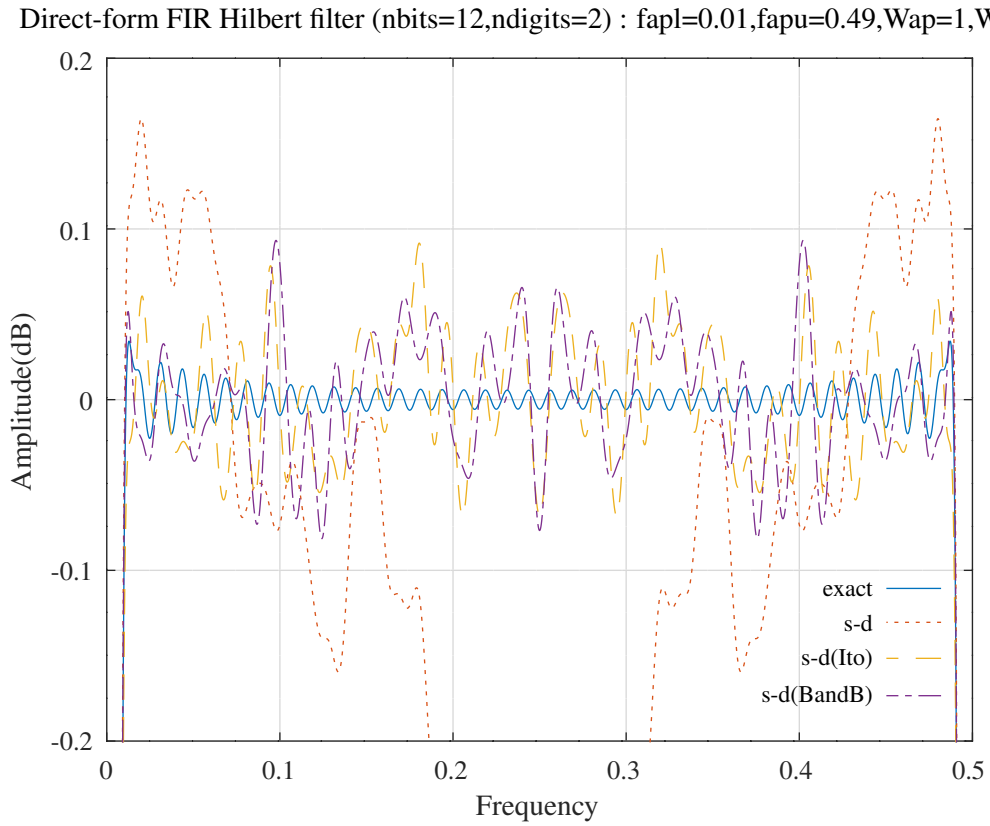



Figure 14.32: Comparison of the amplitude response of a direct-form FIR Hilbert filter with exact and 12-bit 2-signed digit integer coefficients found by branch-and-bound search

## 14.13 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of an FIR Hilbert band-pass filter

The Octave script *branch_bound_directFIRhilbert_bandpass_12_nbits_test.m* uses the branch-and-bound heuristic to optimise the response of a direct-form FIR Hilbert band-pass filter with 12-bit 2-signed-digit coefficients having a similar bandwidth specification to the FIR Hilbert band-pass filter of Appendix N.4.3. An average of 2 signed-digits are allocated to each non-zero coefficient with the heuristic of *Ito et al.* .The direct-form FIR Hilbert filter specification is:

```
M=8 % Number of distinct coefficients
nbits=12 % Coefficient bits
ndigits=2 % Nominal average coefficient signed-digits
tol=0.0001 % Tolerance on coefficient. update
maxiter=400 % iteration limit
fapl=0.16325 % Amplitude pass band lower edge
fapu=0.33675 % Amplitude pass band upper edge
Wap=2 % Amplitude pass band weight
Was=1 % Amplitude stop band weight
```

The filter has 8 distinct non-zero coefficients and the filter group-delay is 15 samples. The distinct non-zero filter coefficients found by the allocation of signed-digits with the heuristic of *Ito et al.* are:

```
hM_sd = [      -20,        8,       64,      -32, ...
              -128,      112,      327,     -868 ]'/2048;
```

and by branch-and-bound search are:

```
hM_min = [      -24,        8,       64,      -32, ...
               -128,      120,      327,     -872 ]'/2048;
```

Figure 14.33, compares the amplitude responses of the FIR Hilbert band-pass filter with exact, 12-bit 2-signed-digit coefficients, 12-bit 2-signed-digit coefficients with signed-digits allocated by the heuristic of *Ito et al.* and with the 12-bit 2-signed-digit coefficients found by branch-and-bound search. Table 14.13 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. The FIR filter structure requires an extra 16 additions.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 4.99e-05 | | |
| 12-bit 2-signed-digit | 5.39e-04 | 16 | 8 |
| 12-bit 2-signed-digit(Ito) | 7.91e-05 | 16 | 8 |
| 12-bit 2-signed-digit(branch-and-bound) | 4.96e-05 | 16 | 8 |

Table 14.13: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a direct-form FIR Hilbert band-pass filter with 12-bit 2-signed-digit coefficients found by branch-and-bound search

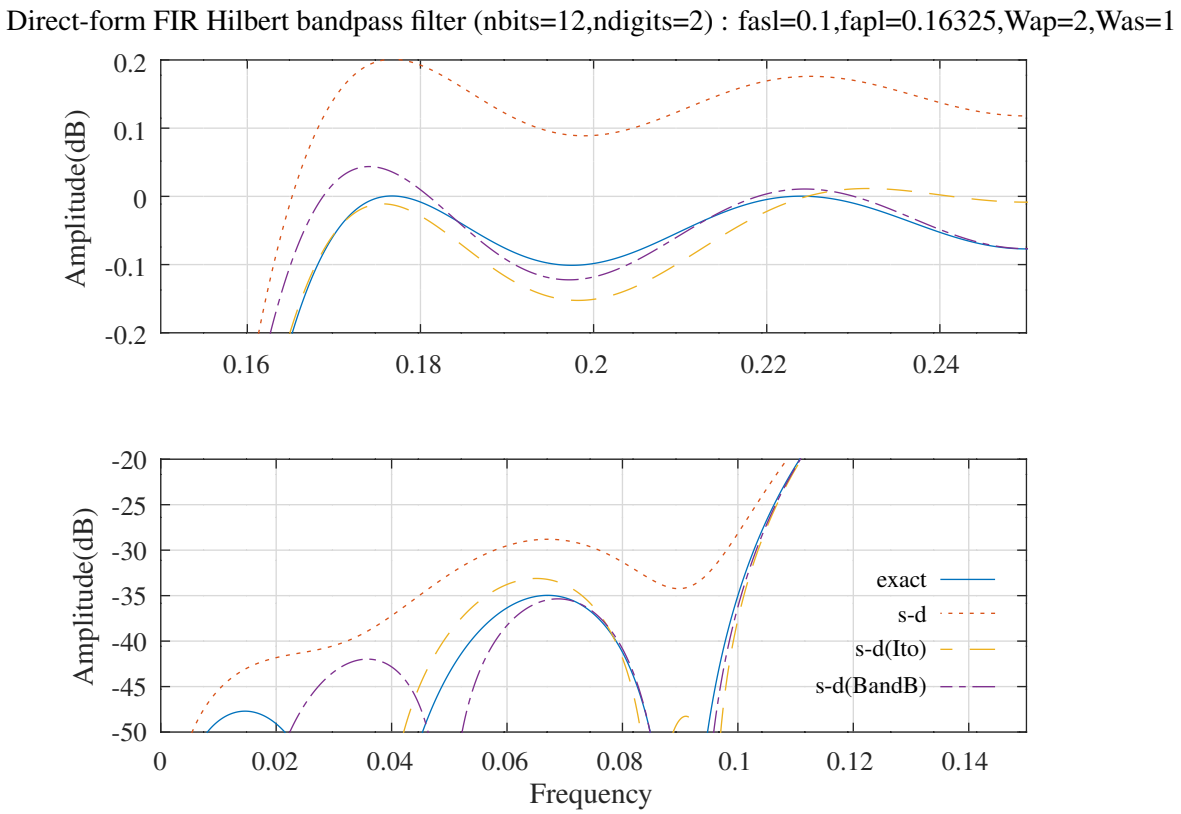



Figure 14.33: Comparison of the amplitude response of a direct-form FIR Hilbert bandpass filter with exact and 12-bit 2-signed digit integer coefficients found by branch-and-bound search

# Chapter 15

# Successive coefficient relaxation search for signed-digit filter coefficients

This section describes the results of experiments in which I fix one coefficient and minimise the objective function over the remaining free coefficients. This is called a *relaxation* of the optimisation. The relaxation is repeated until all the coefficients are fixed.

## 15.1 SOCP-relaxation search for the signed-digit coefficients of a direct-form symmetric bandpass FIR filter

The Octave script *socp_relaxation_directFIRsymmetric_bandpass_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of a direct-form symmetric band-pass FIR filter. The filter specification is:

```
nbits=12 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
tol=1e-05 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
maxiter=5000 % SOCP iteration limit
npoints=1000 % Frequency points across the band
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
fasl=0.05 % Amplitude stop band(1) lower edge
fasu=0.25 % Amplitude stop band(1) upper edge
dBas=43 % 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=37 % Amplitude stop band(2) peak-to-peak ripple
Wasl=40 % Amplitude lower stop band weight
Wasu=40 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2.

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 *directFIRsymmetric_socp_mmse*. The results of this MMSE optimisation are then passed to *directFIRsymmetric_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 distinct direct-form symmetric FIR bandpass filter coefficients found by the SOCP-relaxation search are:

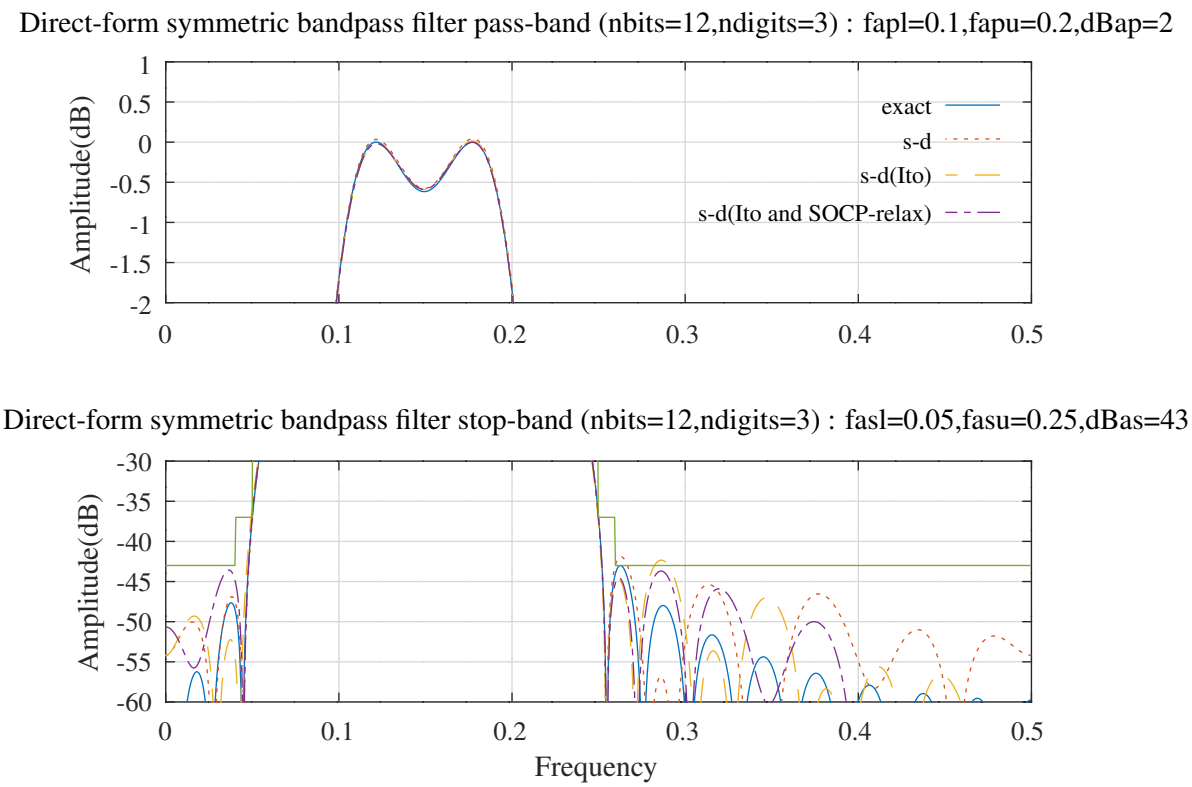



Figure 15.1: Comparison of the amplitude responses for a direct-form symmetric FIR bandpass filter with 12-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SOCP-relaxation search

```
hM_min = [        -1,       -16,       -33,       -15, ...
                  39,        68,        32,        -8, ...
                  28,        82,        -2,      -236, ...
                -366,      -142,       302,       530 ]'/2048;
```

Figure 15.1 compares the responses of the filter with exact, 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and SOCP-relaxation search. Table 15.1 compares the cost and the number of 12 bit shift-and-add operations required to implement the 16 distinct coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the SOCP-relaxation search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.0009 | | |
| 12-bit 3-signed-digit | 0.000971 | 39 | 23 |
| 12-bit 3-signed-digit(Ito) | 0.001018 | 36 | 20 |
| 12-bit 3-signed-digit(SOCP-relax) | 0.001038 | 36 | 20 |

Table 15.1: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a direct-form symmetric FIR bandpass filter with 12-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SOCP-relaxation search

## 15.2 SQP-relaxation search for the signed-digit coefficients of a lattice bandpass IIR filter

The Octave script *sqp_relaxation_schurOneMlattice_bandpass_10_nbits_test.m* performs successive SQP relaxations to optimise the response of the SQP optimised band-pass Schur one-multiplier lattice filter of Section 10.3.6. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
tol=0.0001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
maxiter=2000 % 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=0.998047 % 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=0.2 % Delay pass band peak-to-peak ripple
Wtp=5 % 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=37 % Amplitude stop band(2) peak-to-peak ripple
Wasl=500000 % 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 11.2.

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:

```
k_min = [        0,      332,        0,      256, ...
                 0,      176,        0,      214, ...
                 0,      152,        0,      128, ...
                 0,       76,        0,       52, ...
                 0,       18,        0,        7 ]'/512;

c_min = [       36,       -8,     -156,     -240, ...
               -78,       64,      200,      148, ...
                 8,      -42,      -40,       -8, ...
                -4,      -18,      -13,        1, ...
                12,        8,        2,        0, ...
                 2 ]'/512;
```

Figure 15.2 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 15.3 shows the filter stop-band response and Figure 15.4 shows the filter pass-band group delay response. Table 15.2 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.

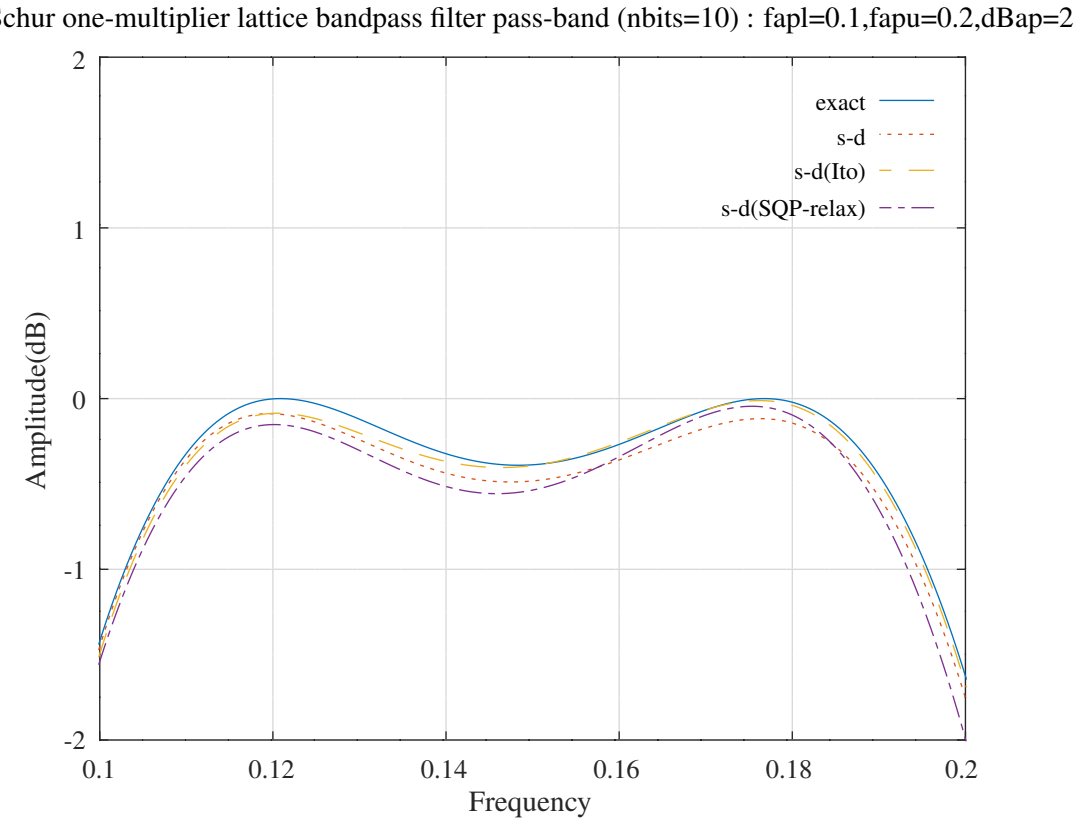



Figure 15.2: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation search

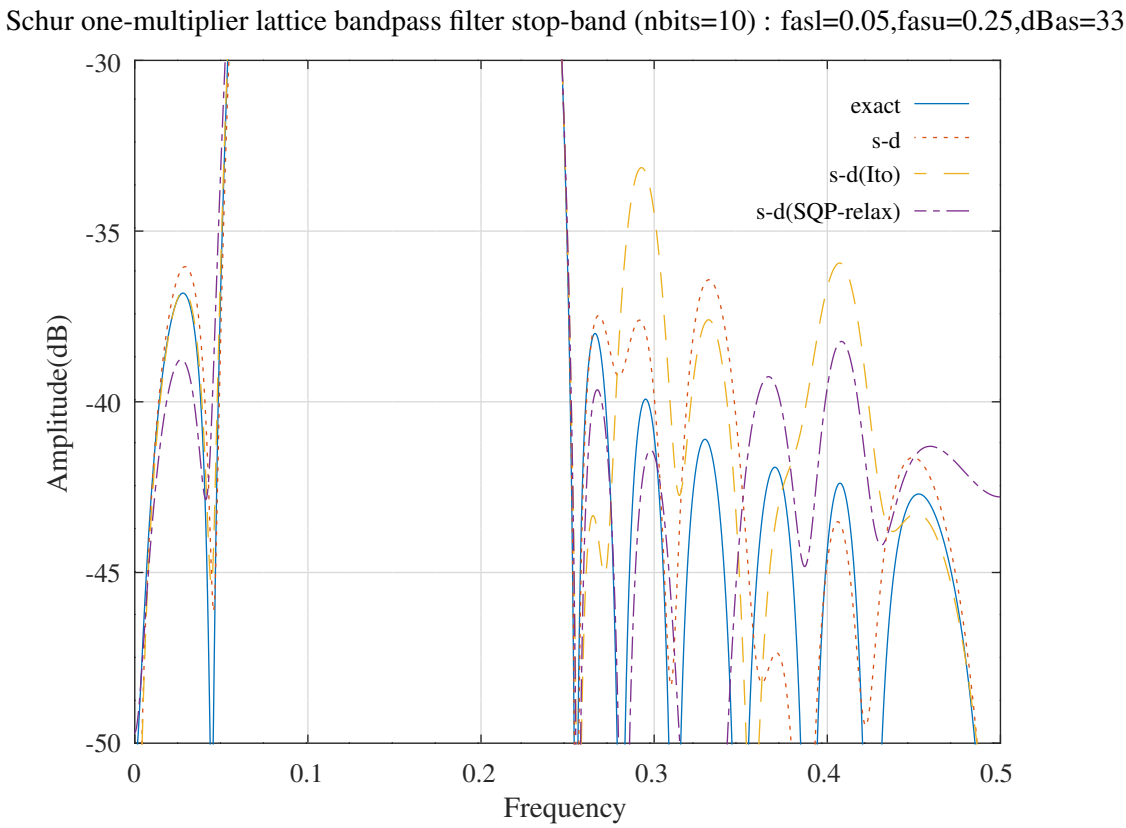



Figure 15.3: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation search

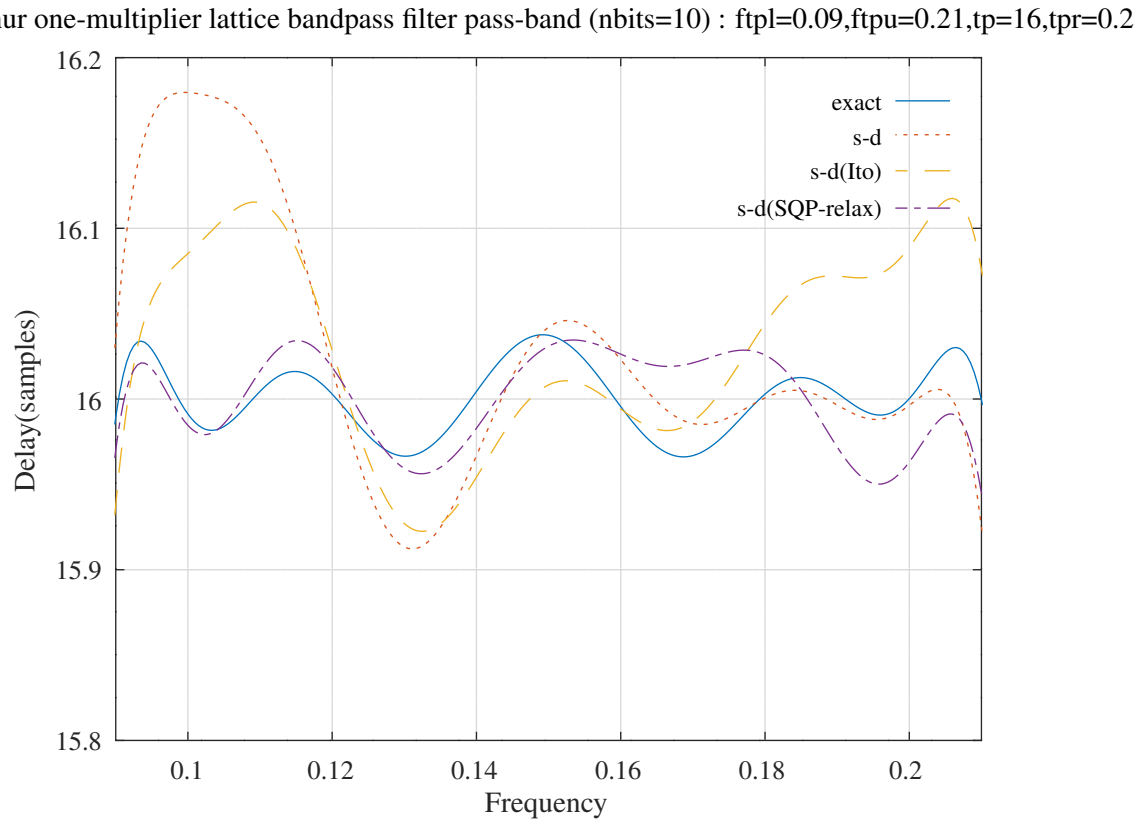



Figure 15.4: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SQP-relaxation search

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.0164 | | |
| 10-bit 3-signed-digit | 0.0502 | 73 | 42 |
| 10-bit 3-signed-digit(Ito) | 0.0652 | 65 | 34 |
| 10-bit 3-signed-digit(SQP-relax) | 0.0258 | 63 | 33 |

Table 15.2: 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

## 15.3 SQP-relaxation search for the signed-digit coefficients of a lattice lowpass IIR filter

The Octave script *sqp_relaxation_schurOneMlattice_lowpass_10_nbits_test.m* performs successive SQP relaxations to optimise the response of the low-pass Schur one-multiplier lattice filter of Section 10.3.2 with 10-bit integer coefficients. The filter specification is:

```
tol=0.0001 % Tolerance on coefficient update vector
ctol=0.0001 % Tolerance on constraints
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=0.2 % Delay pass band peak-to-peak ripple
Wtp=0.5 % Delay pass band weight
fas=0.3 % Amplitude stop band edge
dBas=37 % amplitude stop band peak-to-peak ripple
Was=20000 % Amplitude stop 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 11.1. 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_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:

```
k_min = [      -358,       355,      -303,       215, ...
               -112,        32,         0,         0, ...
                  0,         0 ]'/512;


c_min = [       140,       350,        43,       -44, ...
                -22,         4,        12,         4, ...
                 -4,        -4,         0 ]'/512;
```

Figure 15.5 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 15.6 shows the corresponding filter pass-band response and Figure 15.7 shows the filter pole-zero plot. Table 15.3 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.

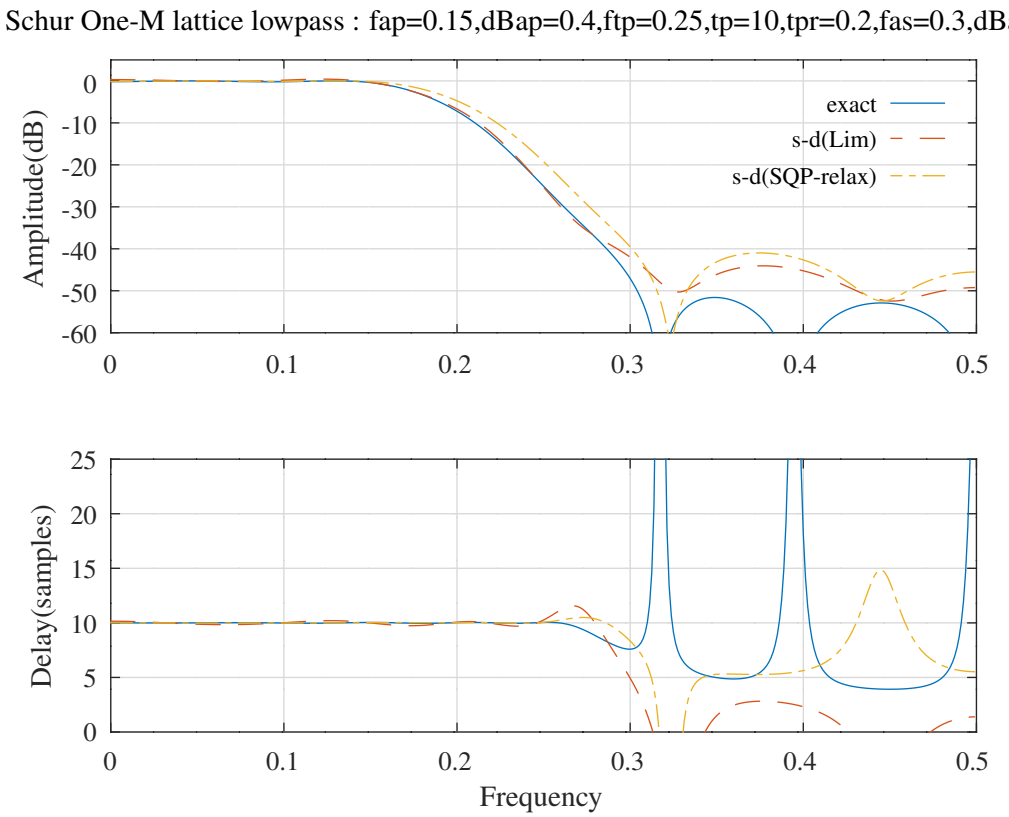



Figure 15.5: Amplitude and group-delay responses for a Schur one-multiplier lattice low-pass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SQP-relaxation search.

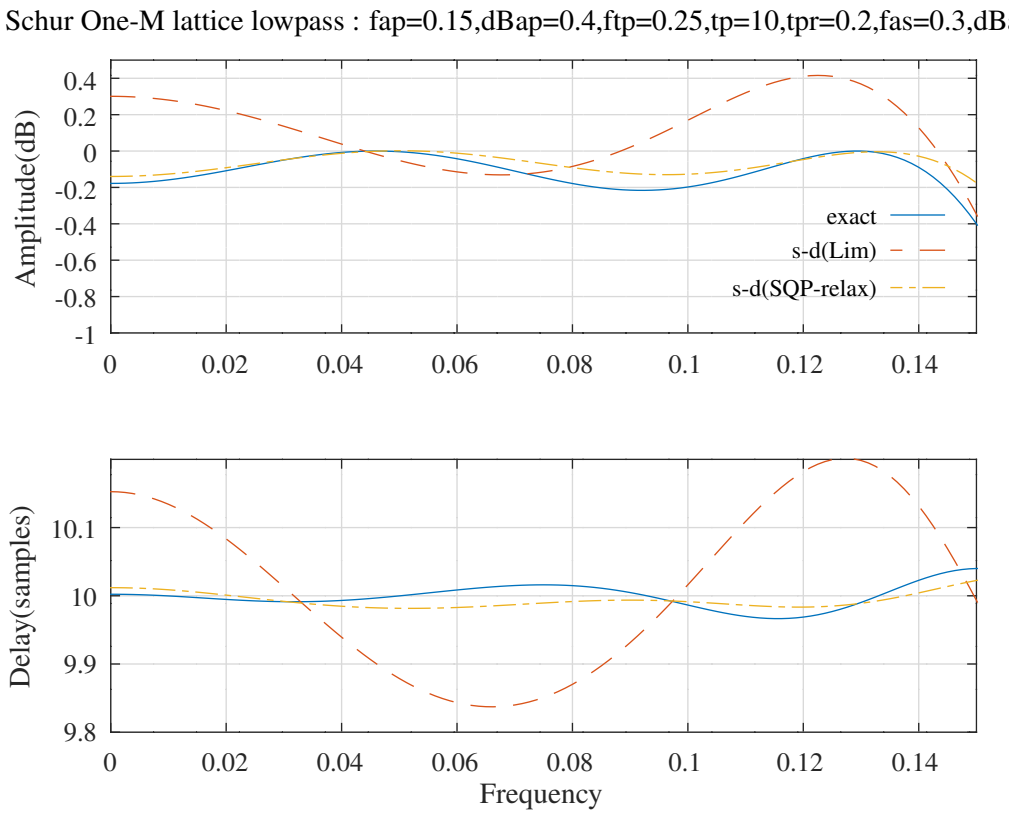



Figure 15.6: Amplitude and group-delay pass-band responses for a Schur one-multiplier lattice low-pass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SQP-relaxation search.

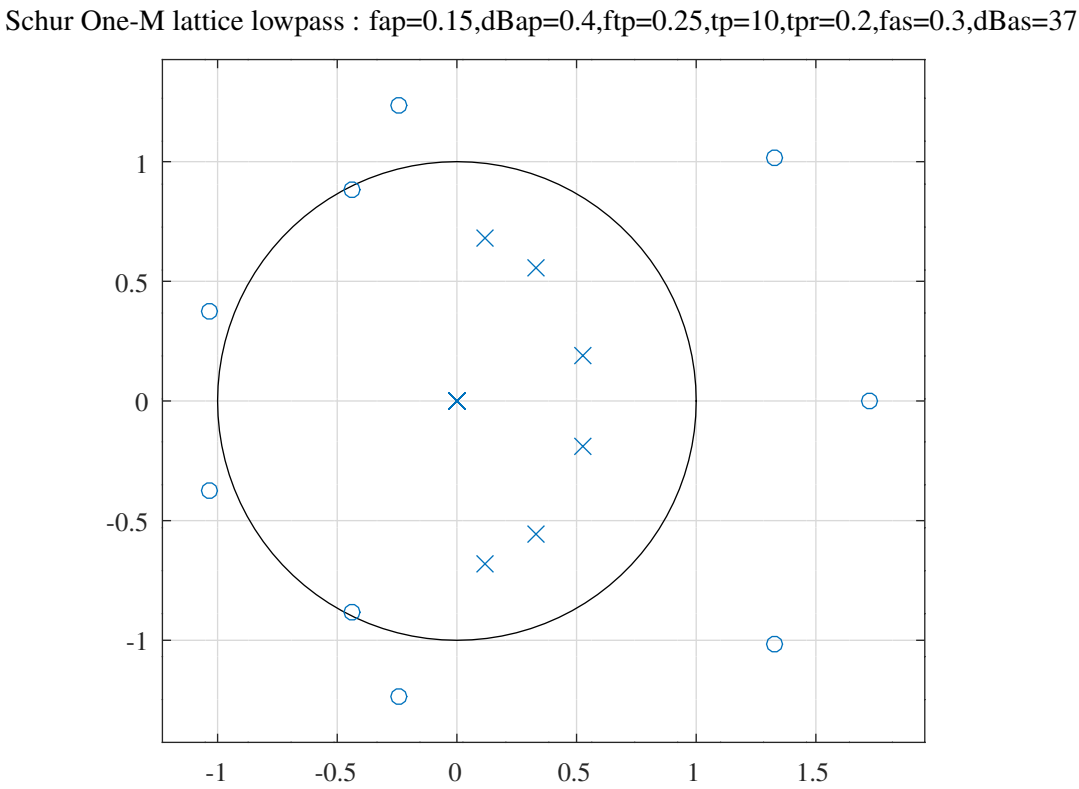



Figure 15.7: Pole-zero plot for a Schur one-multiplier lattice low-pass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SQP-relaxation search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.001253 | | |
| 10-bit 3-signed-digit(Lim) | 0.019346 | 42 | 26 |
| 10-bit 3-signed-digit(SQP-relax) | 0.000629 | 44 | 28 |

Table 15.3: 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.

## 15.4 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice low-pass IIR filter

The Octave script *socp_relaxation_schurOneMPAlattice_lowpass_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of the parallel Schur one-multiplier all-pass lattice low-pass filter of Section 10.3.4 with 12-bit integer coefficients. The filter specification is:

```
nbits=12 % Coefficient word length
ndigits=3 % Average number of signed digits per coef.
tol=0.0001 % Tolerance on coefficient update vector
ctol=5e-07 % Tolerance on constraints
n=1000 % Frequency points across the band
m1=11 % Allpass model filter 1 denominator order
m2=12 % Allpass model filter 2 denominator order
rho=0.992188 % Constraint on allpass coefficients
fap=0.125 % Amplitude pass band edge
dBap=0.2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=0 % Amplitude transition band weight
fas=0.25 % Amplitude stop band edge
dBas=50 % amplitude stop band peak-to-peak ripple
Was=100 % Amplitude stop band weight
ftp=0.175 % Delay pass band edge
td=11.5 % Nominal pass band filter group delay
tdr=0.08 % Delay pass band peak-to-peak ripple
Wtp=2 % Delay pass band weight
```

The filter weights differ from those used in Section 14.4.

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits by the heuristic of *Lim et al.* as shown in Section 11.1. 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 *schurOneMPAlattice_socp_mmse*. The results of this MMSE optimisation are then passed to *schurOneMPAlattice_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 lattice coefficients found by the heuristic of *Lim et al.* are:

```
A1k_sd = [     1576,     -176,     -548,     -131, ...
               -121,      501,     -296,       -9, ...
                336,     -328,      112 ]'/2048;

A2k_sd = [      792,     -560,      383,      336, ...
                -95,       80,     -416,      368, ...
                  8,     -368,      304,     -112 ]'/2048;
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are:

```
A1k_min = [     1576,     -176,     -548,     -134, ...
                -116,      508,     -292,       -5, ...
                 336,     -322,      112 ]'/2048;

A2k_min = [      792,     -556,      380,      335, ...
                 -96,       80,     -416,      368, ...
                   8,     -368,      304,     -112 ]'/2048;
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are implemented with 65 signed-digits and 42 shift-and-add operations.

Figure 15.8 shows the amplitude and group-delay pass-band responses of the filter with 12-bit 3-signed-digit coefficients allocated with the algorithm of *Lim et al.* and SOCP-relaxation search. Figure 15.9 shows the corresponding filter stop-band amplitude response. Table 15.4 compares the cost 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.* with the SOCP-relaxation search.

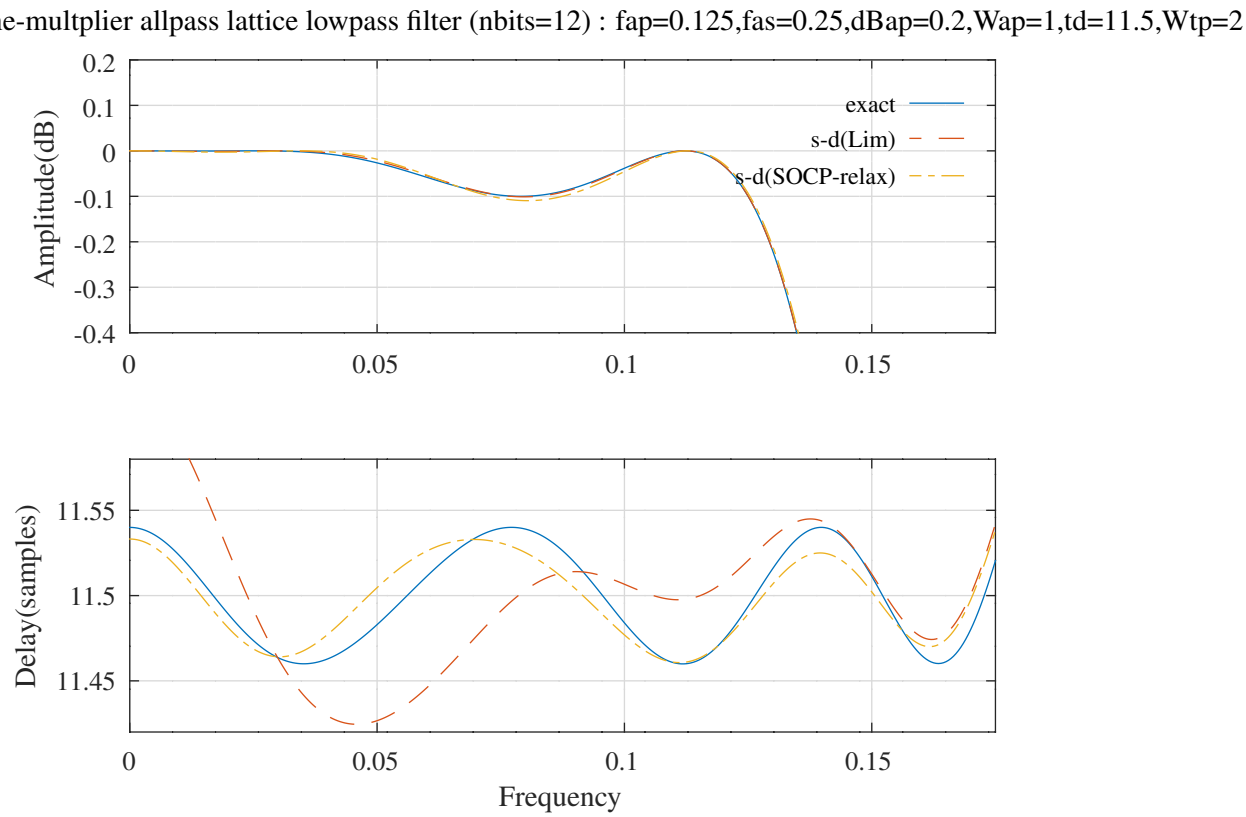



Figure 15.8: Amplitude and group-delay pass-band responses for a parallel Schur one-multiplier all-pass lattice low-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search.

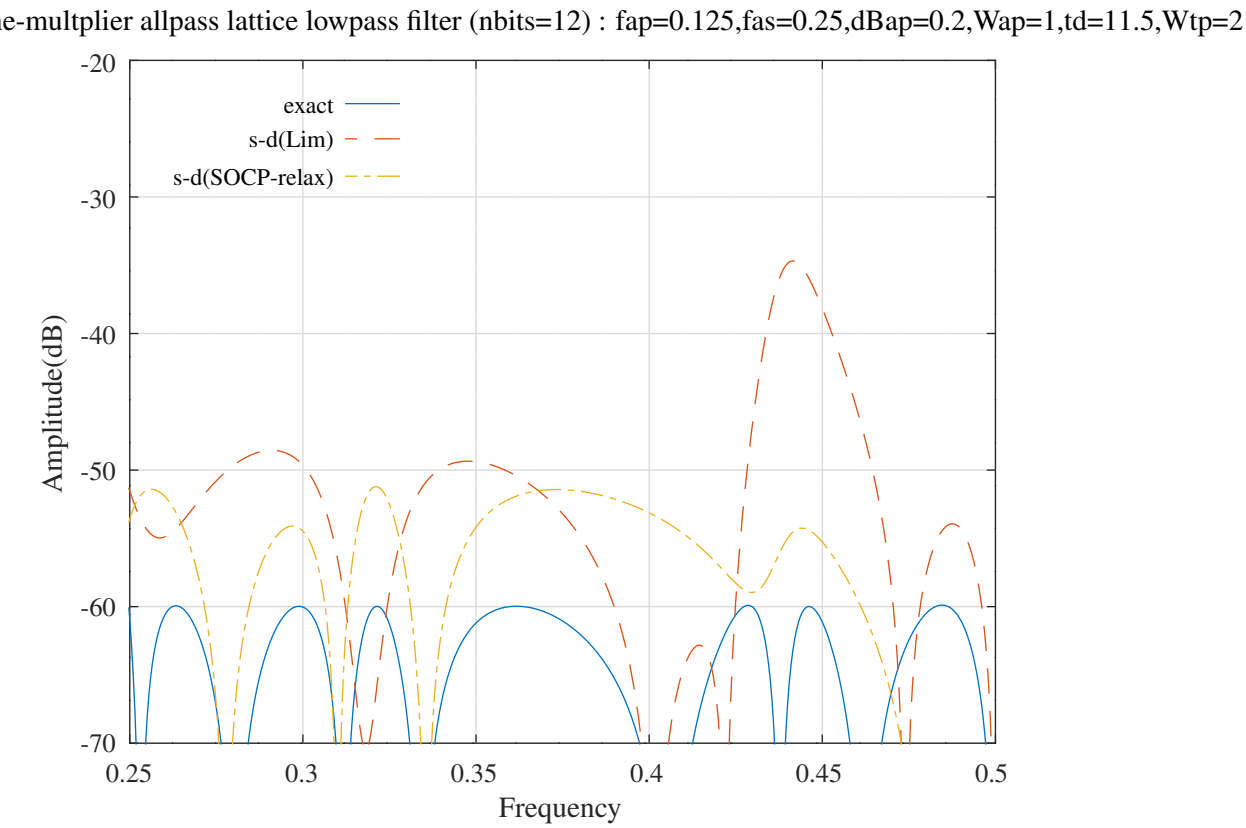



Figure 15.9: Stop band amplitude responses for a parallel Schur one-multiplier all-pass lattice low-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.001848 | | |
| 12-bit 3-signed-digit(Lim) | 0.004636 | 66 | 43 |
| 12-bit 3-signed-digit(SOCP-relax) | 0.001371 | 65 | 42 |

Table 15.4: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a parallel Schur one-multiplier all-pass lattice low-pass filter with 12 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.

## 15.5 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice band-pass IIR filter

The Octave script *socp_relaxation_schurOneMPAlattice_bandpass_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of the parallel Schur one-multiplier all-pass lattice band-pass filter of Section 10.3.5 with 12-bit integer coefficients. The filter specification is:

```
socp_relaxation_schurOneMPAlattice_bandpass_12_nbits_test_allocsd_Lim=1
socp_relaxation_schurOneMPAlattice_bandpass_12_nbits_test_allocsd_Ito=0
nbits=12 % Coefficient word length
ndigits=3 % Average number of signed digits per coef.
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
difference=1 % Use difference of all-pass filters
m1=10 % Allpass model filter 1 denominator order
m2=10 % Allpass model filter 2 denominator order
rho=0.992188 % Constraint on allpass coefficients
fapl=0.1 % Pass band amplitude response lower edge
fapu=0.2 % Pass band amplitude response upper edge
dBap=2.000000 % Pass band amplitude response ripple(dB)
Wap=1 % Pass band amplitude response weight
Watl=0.01 % Lower transition band amplitude response weight
Watu=0.01 % Upper transition band amplitude response weight
fasl=0.05 % Stop band amplitude response lower edge
fasu=0.25 % Stop band amplitude response upper edge
dBas=40.000000 % Stop band amplitude response ripple(dB)
Wasl=1 % Lower stop band amplitude response weight
Wasu=2 % Upper stop band amplitude response weight
ftpl=0.1 % Pass band group-delay response lower edge
ftpu=0.2 % Pass band group-delay response upper edge
td=16.000000 % Pass band nominal group-delay response(samples)
tdr=0.320000 % Pass band group-delay response ripple(samples)
Wtp=1 % Pass band group-delay response weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits by the heuristic of *Lim et al.* as shown in Section 11.1. 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 *schurOneMPAlattice_socp_mmse*. The results of this MMSE optimisation are then passed to *schurOneMPAlattice_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 lattice coefficients found by the heuristic of *Lim et al.* are:

```
A1k_sd = [     -904,     1696,      132,     -672, ...
               1116,     -383,     -384,      904, ...
               -584,      336 ]'/2048;


A2k_sd = [    -1570,     1772,       68,     -770, ...
               1156,     -192,     -256,      896, ...
               -639,      312 ]'/2048;
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are:

```
A1k_min = [     -920,     1648,      144,     -648, ...
                1188,     -368,     -320,      912, ...
                -584,      328 ]'/2048;


A2k_min = [    -1556,     1744,       80,     -736, ...
                1218,     -240,     -256,      896, ...
                -608,      336 ]'/2048;
```

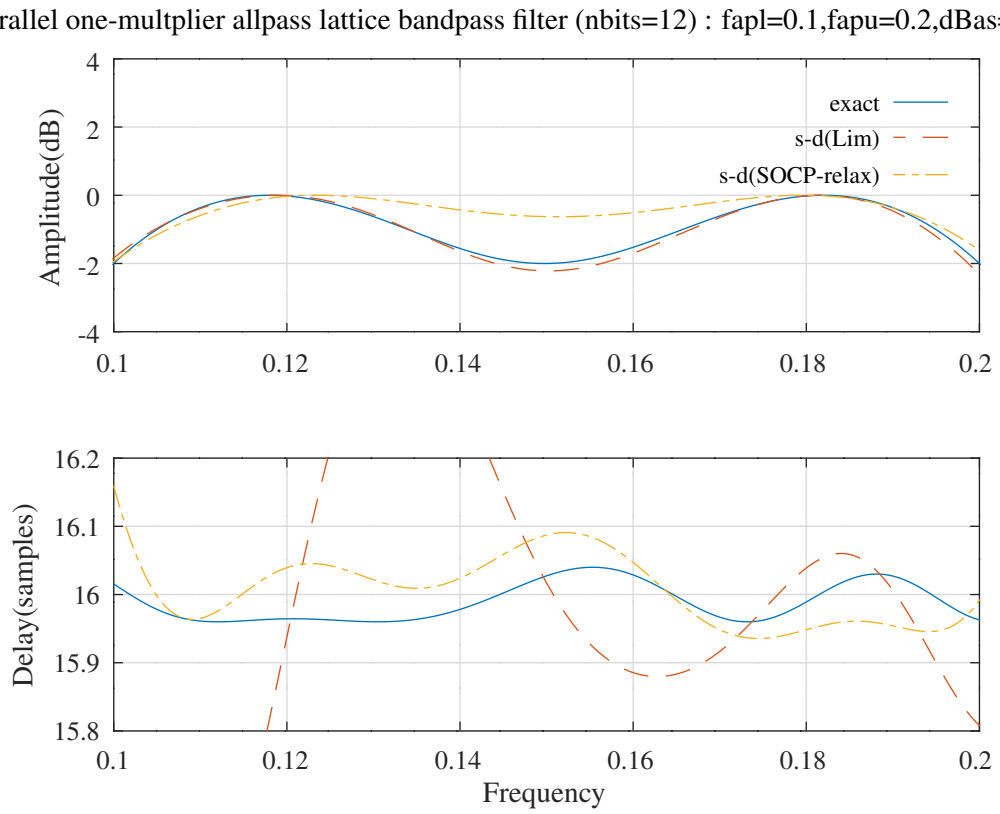



Figure 15.10: Amplitude and group-delay pass-band responses for a parallel Schur one-multiplier all-pass lattice band-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search.

The signed-digit lattice coefficients found by the SOCP-relaxation search are implemented with 59 signed-digits and 39 shift-and-add operations.

Figure 15.10 shows the amplitude and group-delay pass-band responses of the filter with 12-bit 3-signed-digit coefficients allocated with the algorithm of *Lim et al.* and SOCP-relaxation search. Figure 15.11 shows the corresponding filter stop-band amplitude response. Table 15.5 compares the cost 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.* with the SOCP-relaxation search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.029973 | | |
| 12-bit 3-signed-digit(Lim) | 0.076105 | 57 | 37 |
| 12-bit 3-signed-digit(SOCP-relax) | 0.011573 | 59 | 39 |

Table 15.5: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a parallel Schur one-multiplier all-pass lattice band-pass filter with 12 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.

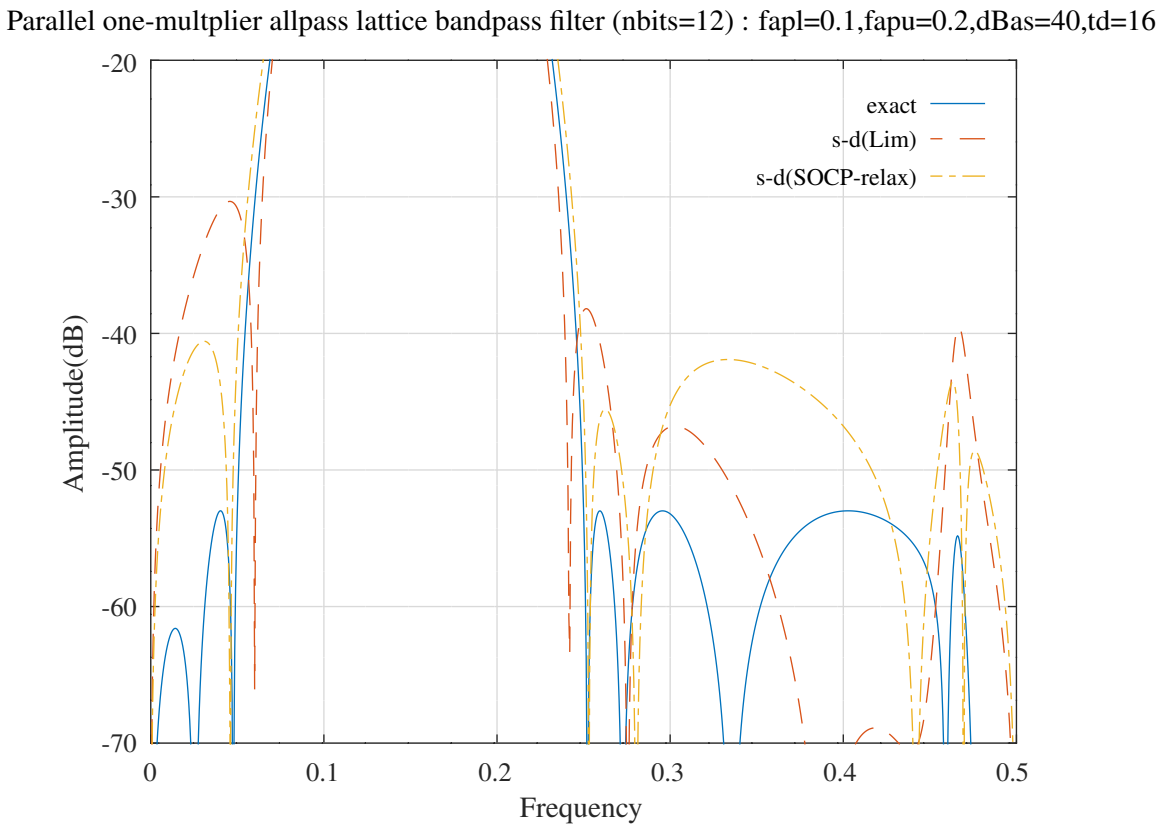



Figure 15.11: Stop band amplitude responses for a parallel Schur one-multiplier all-pass lattice band-pass filter with 12 bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search.

## 15.6 SOCP-relaxation search for the signed-digit coefficients of a lattice Hilbert IIR filter

The Octave script *socp_relaxation_schurOneMlattice_hilbert_10_nbits_test.m* performs successive SOCP relaxations to optimise the response of the band-pass Schur one-multiplier lattice filter of Section 10.3.9 with 10-bit integer coefficients. The filter specification is:

```
tol=1e-08 % Tolerance on coefficient update vector
ctol=1e-08 % Tolerance on constraints
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 11.1. 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 15.2, 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_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:

```
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 ]'/512;
```

The signed-digit filter coefficients found by the SOCP-relaxation search are:

```
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 ]'/512;
```

Figure 15.12 compares the amplitude 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 responses shown are adjusted for the nominal filter delay. Table 15.6 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.

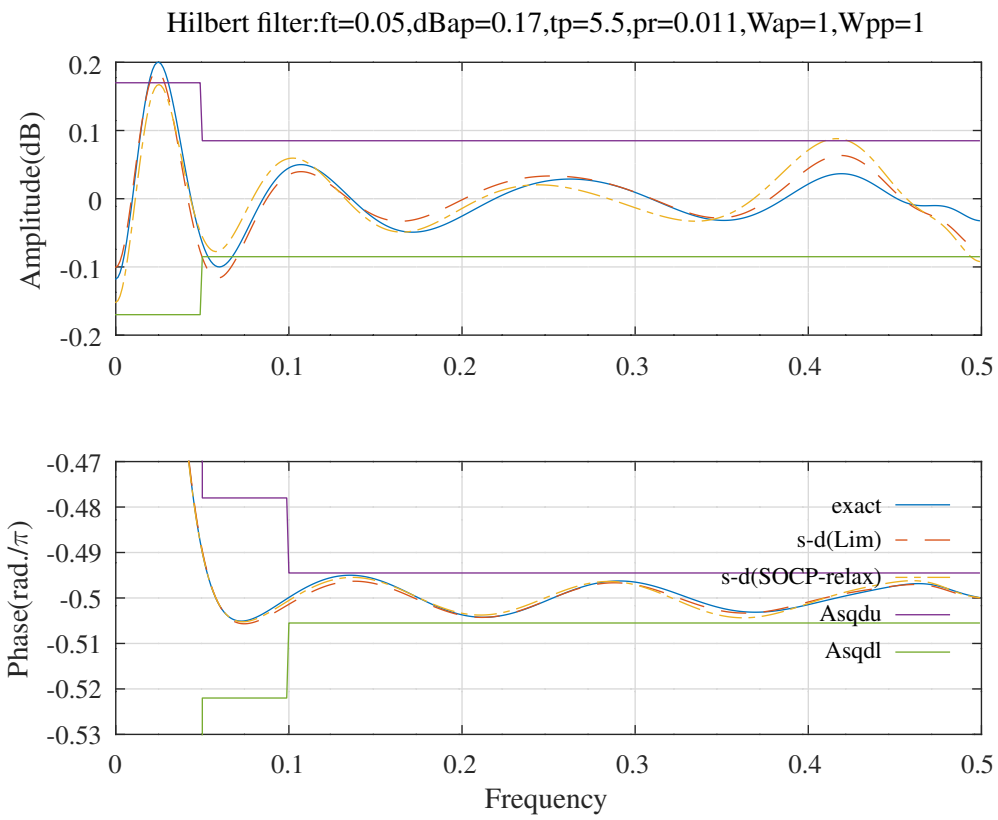



Figure 15.12: Comparison of the amplitude and phase responses for a Schur one-multiplier lattice Hilbert filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The phase responses shown are adjusted for the nominal delay.

| | 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 15.6: 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 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.

## 15.7 SOCP-relaxation search for the signed-digit coefficients of a one-multiplier lattice low-pass differentiator IIR filter

The Octave script *socp_relaxation_schurOneMlattice_lowpass_differentiator_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of an implementation of the low-pass differentiator filter shown in Section 8.2.9 as the series combination of $z-1$ and a Schur one-multiplier lattice filter with 12-bit integer signed-digit coefficients. The filter specification is:

```
nbits=12 % Bits-per-coefficient
ndigits=3 % Average igned-digits-per-coefficient
tol=0.0001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
fap=0.19 % Amplitude pass band upper edge
Arp=0.5 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Ars=0.5 % Amplitude stop band peak-to-peak ripple
Was=1 % Amplitude stop band weight
td=9 % Pass band group delay
tdr=0.08 % Pass band group delay peak-to-peak ripple
Wtp=0.02 % Pass band group delay weight
pr=0.0200 % Phase pass band peak-to-peak ripple(rad./$\pi$))
Wpp=0.02 % Phase pass band weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2. 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_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 signed-digit filter coefficients found by the heuristic of *Ito et al.* are:

```
k0_sd = [     -256,     1920,    -1312,     1392, ...
             -1408,     1400,    -1336,     1160, ...
              -780,      336,      -64 ]'/2048;


c0_sd = [       40,        1,     -512,     -256, ...
               -24,       60,        4,      -16, ...
                -1,        5,        1,        0 ]'/2048;
```

The signed-digit filter coefficients found by the SOCP-relaxation search are:

```
k_min = [     -256,     1920,    -1312,     1392, ...
             -1408,     1400,    -1336,     1156, ...
              -780,      336,      -64 ]'/2048;


c_min = [       40,        1,     -512,     -256, ...
               -24,       60,        4,      -16, ...
                -2,        5,        2,        0 ]'/2048;
```

Figure 15.13 compares the amplitude error, phase and group delay responses of the filter with exact, 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and SOCP-relaxation search. The phase responses shown are adjusted for the nominal filter delay. Table 15.7 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 *Ito et al.* with SOCP-relaxation search.

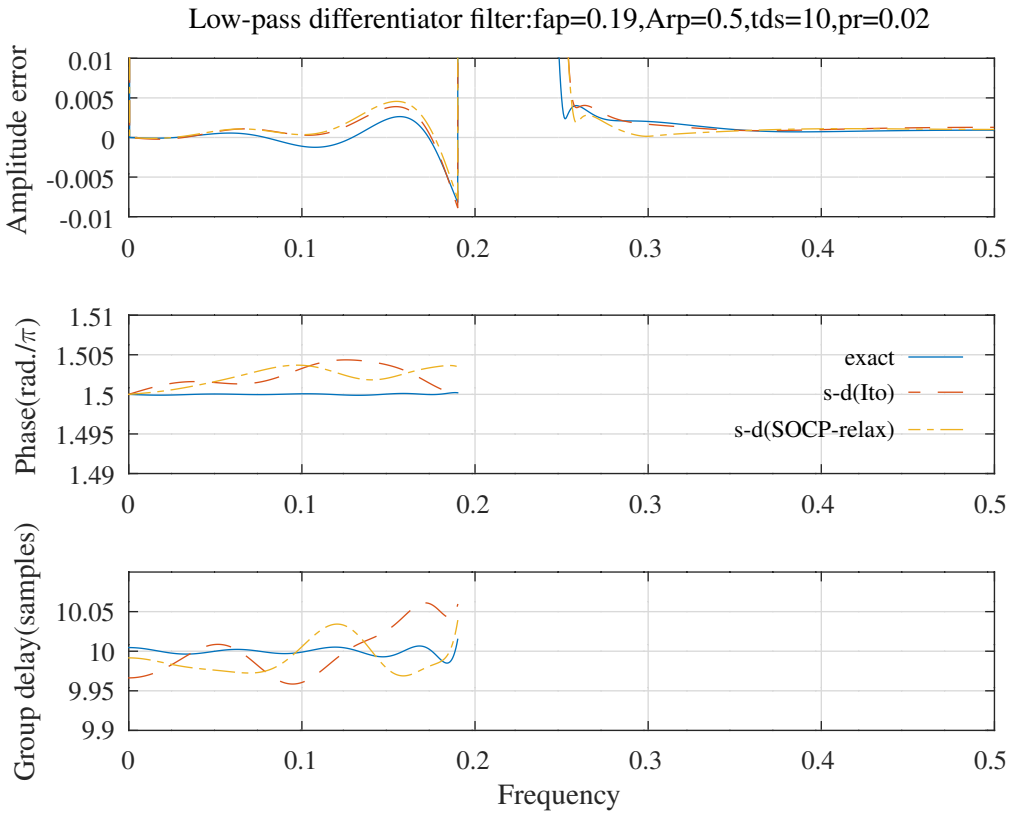



Figure 15.13: Comparison of the amplitude error, phase and group delay responses for a Schur one-multiplier lattice lowpass differentiator filter with 12-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SOCP-relaxation search. The phase responses shown are adjusted for the nominal delay.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 2.0637e-06 | | |
| 12-bit 3-signed-digit(Ito) | 2.40126e-05 | 47 | 25 |
| 12-bit 3-signed-digit(SOCP-relax) | 1.30909e-05 | 47 | 25 |

Table 15.7: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier lattice lowpass differentiator filter with 12-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SOCP-relaxation search.

## 15.8 SOCP relaxation search for the 12-bit 3-signed-digit coefficients of a FRM low-pass filter

The Octave script *socp_relaxation_schurOneMAPlattice_frm_12_nbits_test.m* uses socp-relaxation search to optimise the response of the FRM low-pass filter of Section 10.4.6 with 12-bit coefficients each having an average of 3 signed-digits allocated by the method of *Ito et al.*. The number of signed-digits per coefficient is as shown in Section 14.10. The filter specification is:

```
n=1000 % Frequency points across the band
tol=0.0005 % Tolerance on coefficient update vector
ctol=5e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
mr=10 % Allpass model filter denominator order
Mmodel=9 % Model filter FRM decimation factor
Dmodel=9 % Model filter nominal pass band group delay
dmask=20 % FIR masking filter delay
fap=0.3 % Magnitude-squared pass band edge
dBap=1 % Pass band magnitude peak-to-peak ripple
Wap=1 % Pass band magnitude-squared weight
fas=0.3105 % Magnitude-squared stop band edge
dBas=40 % Stop band magnitude minimum attenuation
Was=1 % Stop band magnitude-squared weight
ftp=0.3 % Delay pass band edge
tp=101 % Pass band nominal delay
tpr=tp/101 % Pass band delay peak-to-peak ripple
Wtp=1 % Pass band magnitude-squared weight
fpp=0.3 % Phase pass band edge
ppr=0.04*pi % Pass band phase peak-to-peak ripple (rad.)
Wpp=0.1 % Phase pass band weight
rho=0.992188 % Constraint on allpass pole radius
```

The filter coefficients found by the socp-relaxation search are:

```
k_min = [      -24,     1186,       35,     -295, ...
                -9,      115,       17,      -55, ...
                -9,       23 ]'/2048;


epsilon_min = [  1,  1, -1,  1, ...
                 1, -1, -1,  1, ...
                 1, -1 ];


u_min = [     1179,      616,     -116,     -176, ...
               105,       68,      -85,      -20, ...
                80,      -34,      -28,       16, ...
                21,      -18,       -8,       15, ...
                 2,      -19,       16,        0, ...
                -1 ]'/2048;


v_min = [    -1368,     -558,      269,        9, ...
              -135,       98,        9,      -74, ...
                61,       -8,      -36,       26, ...
                 6,      -24,       16,        4, ...
               -15,       14,       -4,       -6, ...
                 2 ]'/2048;
```

Figure 15.14, compares the amplitude, phase and delay responses of the FRM low-pass filter with exact and 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by SOCP-relaxation search. The phase response shown is adjusted for the nominal delay. Table 15.8 compares the cost and the number of 12 bit shift-and-add operations required to implement the coefficient multiplications found by SOCP-relaxation search.

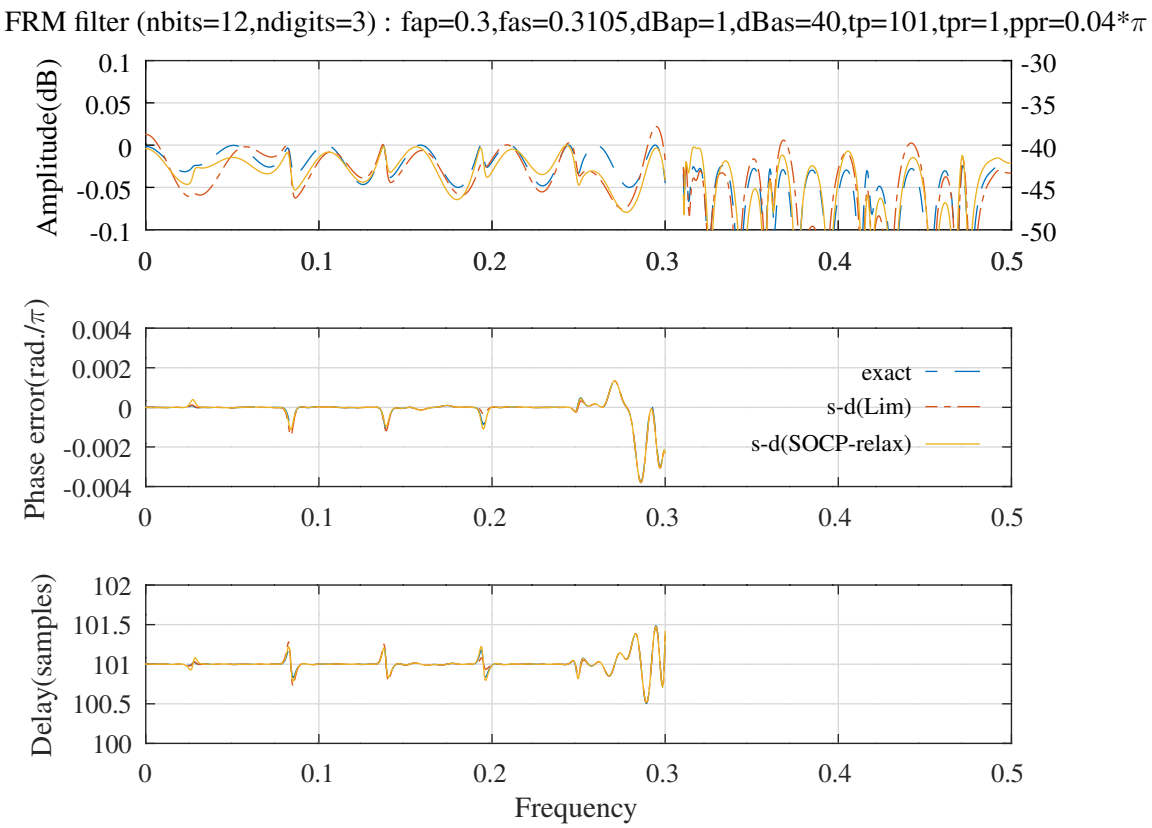



Figure 15.14: Comparison of the amplitude, phase and delay responses of an FRM low-pass filter with exact and 12 bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by SOCP-relaxation search. The phase response shown is adjusted for the nominal delay.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.001723 | | |
| 12-bit 3-signed-digit(Lim) | 0.001813 | 126 | 74 |
| 12-bit 3-signed-digit(SOCP-relax) | 0.001797 | 126 | 75 |

Table 15.8: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for an FRM low-pass filter with 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by SOCP-relaxation search

## 15.9 SOCP-relaxation search for the signed-digit coefficients of an FRM Hilbert IIR filter with an all-pass lattice model filter

The Octave script *socp_relaxation_schurOneMAPlattice_frm_hilbert_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of the FRM Hilbert filter of Section 10.4.8 with 12-bit integer coefficients. The FRM 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
ctol=5e-05 % Tolerance on constraints
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 nominal phase (rad.)
ppr=0.01*pi % 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 11.1.

As in Section 15.2, 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 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:

```
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 ]'/2048;
```

The signed-digit filter coefficients found by the SOCP-relaxation search are:

```
k_min = [    -1152,     -272,     -112,      -32, ...
               -16 ]'/2048;


u_min = [       -1,       -4,      -14,      -24, ...
               -64,      -72,     -104,     -116, ...
               901 ]'/2048;
```

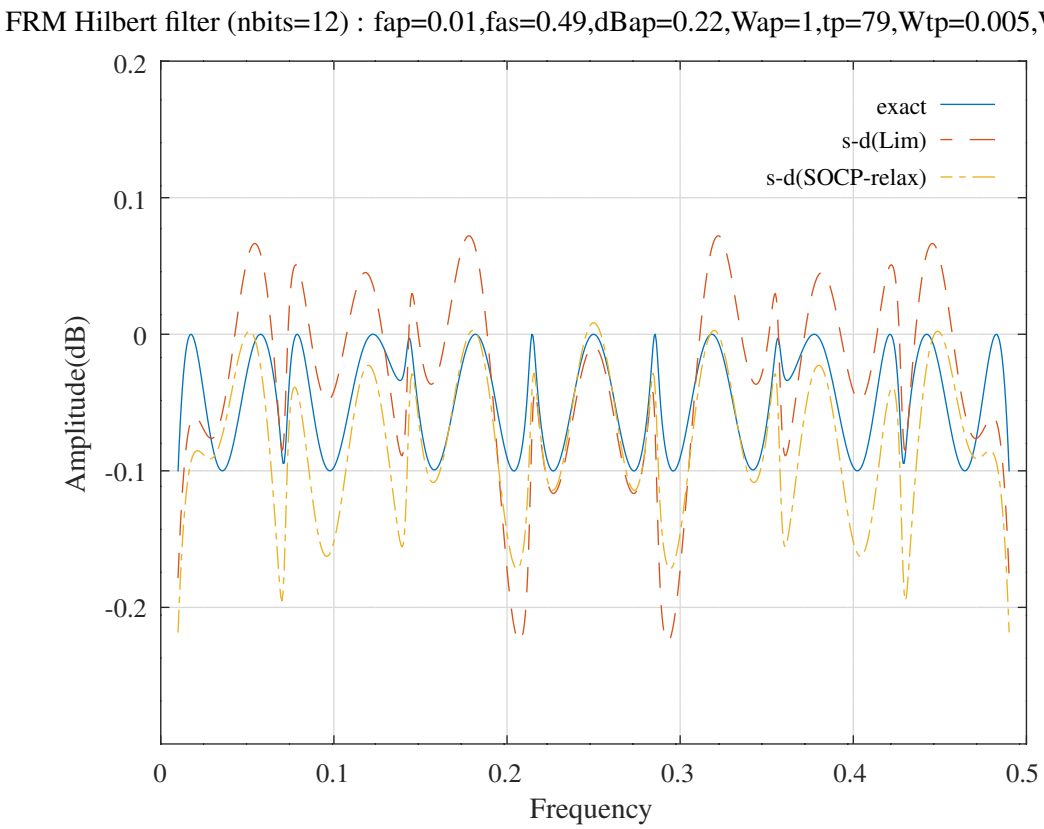



Figure 15.15: Comparison of the amplitude responses for an FRM Hilbert filter with with 12 bit integer coefficients found by allocating an average of 2 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The FRM model filter is implemented as a Schur one-multiplier all-pass lattice filter.

```
v_min = [        16,        10,        14,         4, ...
                -16,       -64,      -164,      -641 ]'/2048;
```

Table 15.9 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 15.15, 15.17 and 15.16 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 shown is adjusted for the nominal delay and normalised to $\pi$ radians.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.000656 | | |
| 12-bit 2-signed-digit(Lim) | 0.001206 | 41 | 19 |
| 12-bit 2-signed-digit(SOCP-relax) | 0.001701 | 41 | 19 |

Table 15.9: 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 filter is implemented as a Schur one-multiplier all-pass lattice filter.

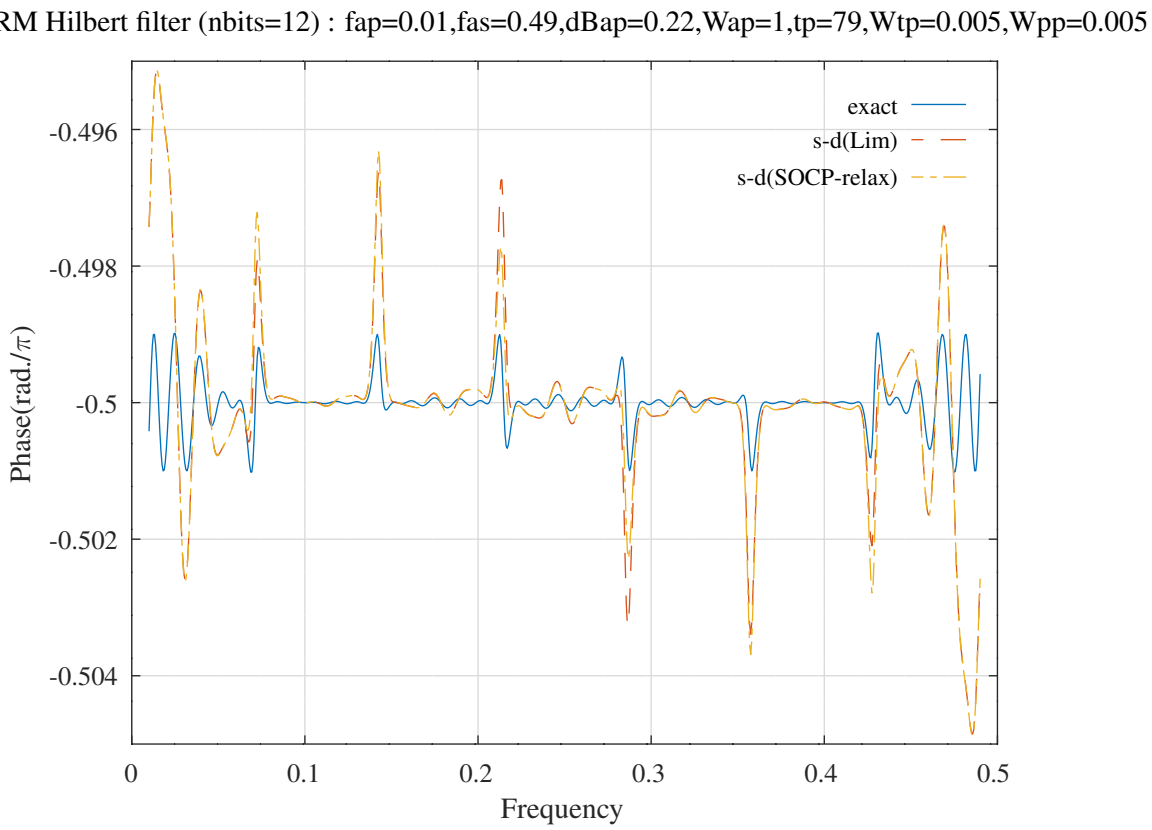



Figure 15.16: Comparison of the phase responses for an FRM Hilbert filter with with 12 bit integer coefficients found by allocating an average of 2 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The phase response shown is adjusted for the nominal delay and normalised to $\pi$ radians. The FRM model filter is implemented as a Schur one-multiplier all-pass lattice filter.

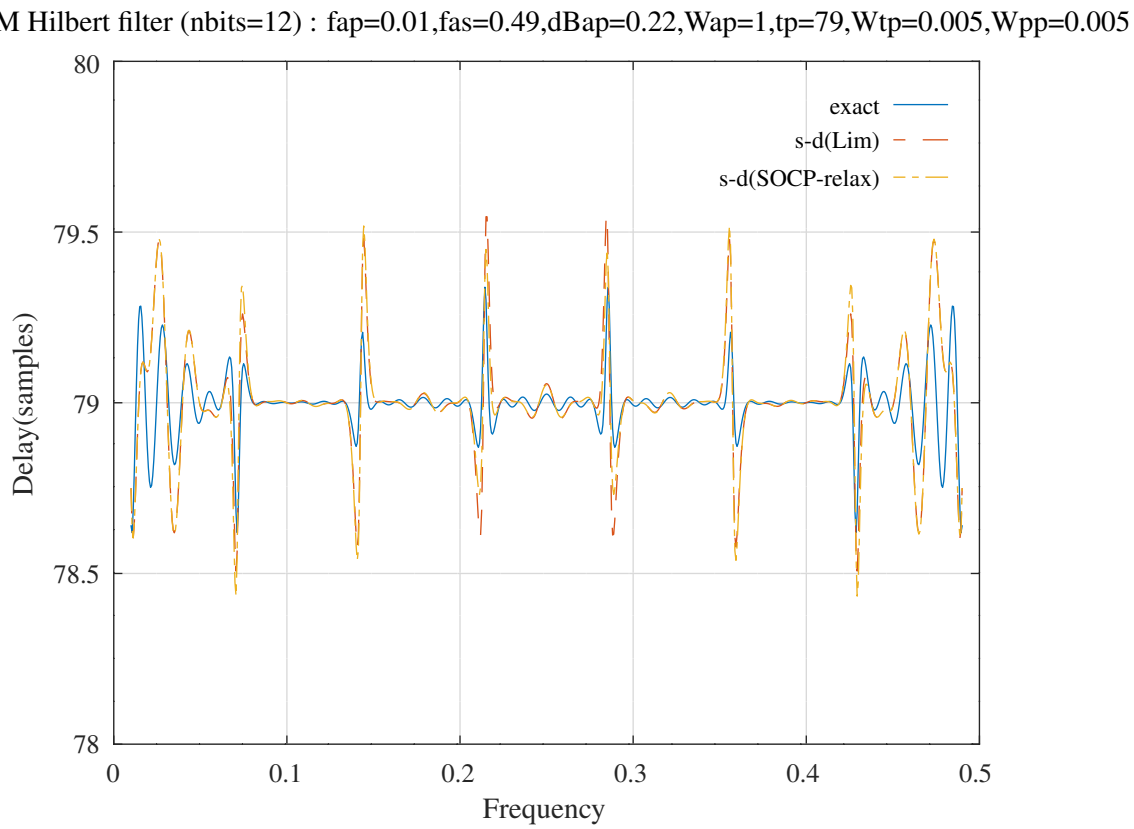



Figure 15.17: Comparison of the delay responses for an FRM Hilbert filter with with 12 bit integer coefficients found by allocating an average of 2 signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The FRM model filter is implemented as a Schur one-multiplier all-pass lattice filter.

## 15.10 SOCP-relaxation search for the signed-digit coefficients of a direct-form FIR Hilbert filter

The Octave script *socp_relaxation_directFIRhilbert_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of a direct-form FIR Hilbert filter having a response similar to that of Section 15.9. The filter specification is:

```
M=40 % Number of distinct coefficients
nbits=12 % Coefficient bits
ndigits=2 % Nominal average coefficient signed-digits
tol=1e-05 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
maxiter=1000 % SOCP iteration limit
npoints=500 % Frequency points across the band
fapl=0.01 % Amplitude pass band lower edge
fapu=0.49 % Amplitude pass band upper edge
dBap=0.1575 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Was=0 % Amplitude stop band weight
```

The filter has $M$ distinct non-zero coefficients, the filter polynomial order is $4M-2$, the filter group-delay is $2M-1$ samples. The filter coefficients are truncated to 12 bits allocated with an average of 2 signed-digits by the heuristic of *Ito et al.*, as shown in Section 11.2. 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 *directFIRhilbert_socp_mmse*. The results of this MMSE optimisation are then passed to *directFIRhilbert_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 12-bit signed-digit filter coefficients found by the heuristic of *Ito et al.* are:

```
hM0_Ito_sd = [       -1,       -1,       -1,       -2, ...
                     -2,       -2,       -4,       -3, ...
                     -4,       -4,       -4,       -8, ...
                     -8,       -8,       -8,       -8, ...
                    -12,      -12,      -16,      -16, ...
                    -18,      -20,      -24,      -24, ...
                    -28,      -32,      -36,      -40, ...
                    -48,      -52,      -60,      -68, ...
                    -80,      -96,     -112,     -140, ...
                   -184,     -258,     -432,    -1304 ]'/2048;
```

The 12-bit signed-digit filter coefficients found by the SOCP-relaxation search are:

```
hM_min = [       -1,       -1,       -1,       -1, ...
                 -2,       -2,       -2,       -3, ...
                 -4,       -4,       -4,       -4, ...
                 -8,       -8,       -8,       -8, ...
                -12,      -12,      -16,      -16, ...
                -18,      -20,      -24,      -28, ...
                -28,      -32,      -36,      -40, ...
                -48,      -52,      -60,      -68, ...
                -79,      -96,     -112,     -142, ...
               -184,     -258,     -432,    -1304 ]'/2048;
```

Figure 15.18 compares the amplitude responses of the filter with exact, 12-bit 2-signed-digit coefficients, 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and SOCP-relaxation search. Table 15.10 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 *Ito et al.* with SOCP-relaxation search. The FIR filter structure requires an extra 80 additions.

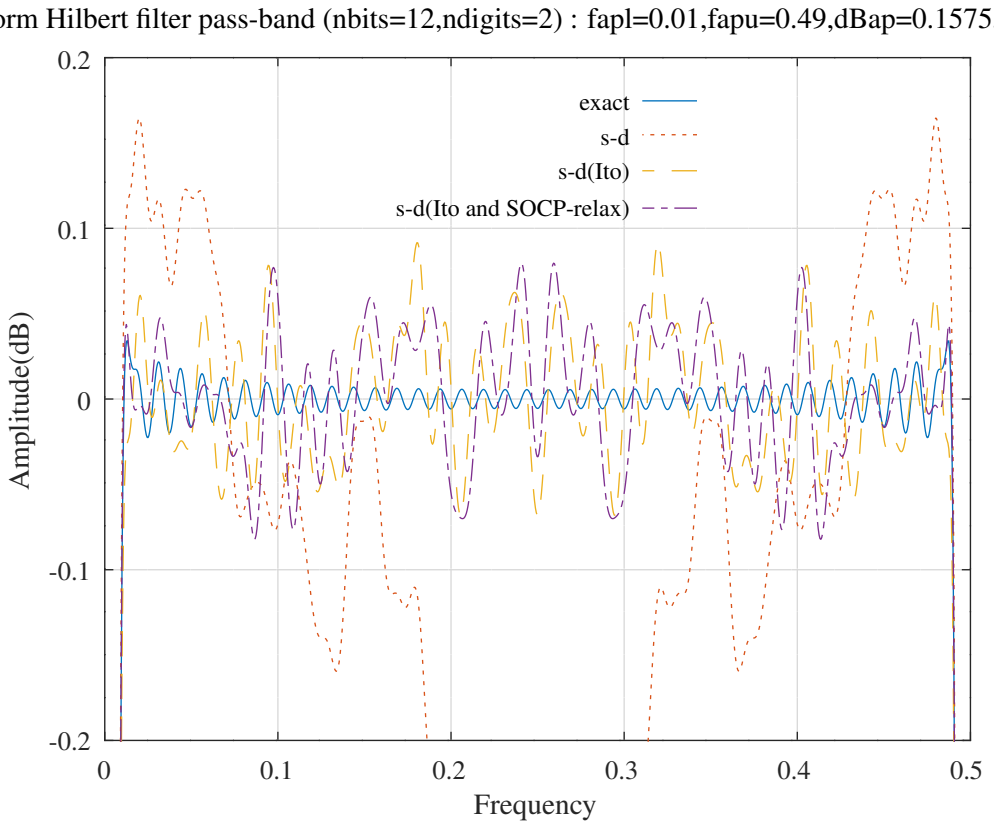



Figure 15.18: Comparison of the amplitude responses for a direct-form FIR Hilbert filter with 12-bit integer coefficients found by allocating an average of 2-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SOCP-relaxation search.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 5.346e-07 | | |
| 12-bit 2-signed-digit | 0.0002085 | 71 | 31 |
| 12-bit 2-signed-digit(Ito) | 9.289e-06 | 68 | 28 |
| 12-bit 2-signed-digit(SOCP-relax) | 9.126e-06 | 69 | 29 |

Table 15.10: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a direct-form FIR Hilbert filter with 12-bit integer coefficients found by allocating an average of 2-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SOCP-relaxation search.

## 15.11 POP relaxation search for the signed-digit coefficients of a one-multiplier lattice bandpass filter

*Lu* [115] and *Lu* and *Hinamoto* [214] point out that the integer programming optimisation of the truncated filter coefficients shown in Equation 16.1 can be rewritten as a *polynomial optimisation problem* (POP) in SOCP form as:

$$\begin{aligned} \textbf{minimise} \quad & \mathcal{E}\left(\boldsymbol{x_k}\right) + \nabla_x \mathcal{E}\left(\boldsymbol{x_k}\right)^\top \Delta \boldsymbol{x} + \frac{1}{2} \Delta \boldsymbol{x}^\top \nabla_{xx}^2 \mathcal{E}\left(\boldsymbol{x_k}\right) \Delta \boldsymbol{x} \\ \textbf{subject to} \quad & \left(\boldsymbol{x_k} + \Delta \boldsymbol{x} - \boldsymbol{x}_u\right)^\top \left(\boldsymbol{x_k} + \Delta \boldsymbol{x} - \boldsymbol{x}_l\right) = \boldsymbol{0} \end{aligned}$$

where $\Delta \boldsymbol{x}$ is the coefficient update vector, $\mathcal{E}$ is the weighted response error at $\boldsymbol{x_k}$ and $\boldsymbol{x}_u$ and $\boldsymbol{x}_u$ are the upper and lower bounds on the truncated coefficients, $\boldsymbol{x_k} + \Delta \boldsymbol{x}$.

*Lasserre* [111] and *Waki et al.* [67] 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* [68], 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_schurOneMlattice_bandpass_10_nbits_test.m* performs SparsePOP optimisation of the truncated coefficients of the SQP optimised band-pass Schur one-multiplier lattice filter described in Section 10.3.6. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
tol=0.0001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
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=0.998047 % 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=0.2 % Delay pass band peak-to-peak ripple
Wtp=5 % 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=40 % Amplitude stop band(2) peak-to-peak ripple
Wasl=500000 % 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 11.2. 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:

```
k_min = [        0,      332,        0,      256, ...
                 0,      176,        0,      214, ...
                 0,      148,        0,      128, ...
                 0,       76,        0,       54, ...
                 0,       18,        0,        9 ]'/512;
```

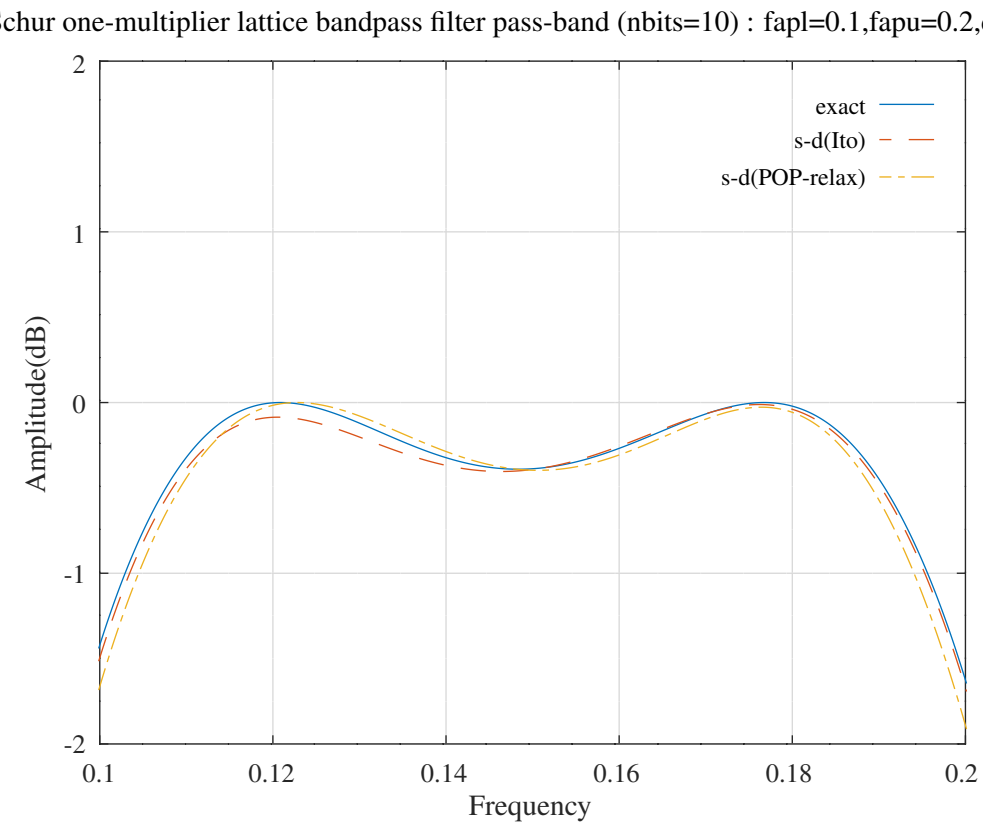



Figure 15.19: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing POP-relaxation search

```
c_min = [        38,        -8,      -156,      -248, ...
                -81,        64,       200,       152, ...
                  8,       -42,       -40,        -8, ...
                 -4,       -16,       -12,         2, ...
                 12,         8,         1,        -1, ...
                  2 ]'/512;
```

Table 15.11 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 15.19 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 15.20 shows the filter stop-band response and Figure 15.21 shows the filter pass-band group delay response.

| | Cost | Signed-digits | Shift-and-adds |
|---|---|---|---|
| Exact | 0.0164 | | |
| 10-bit 3-signed-digit(Ito) | 0.0652 | 65 | 34 |
| 10-bit 3-signed-digit(POP-relax) | 0.0228 | 63 | 32 |

Table 15.11: 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.

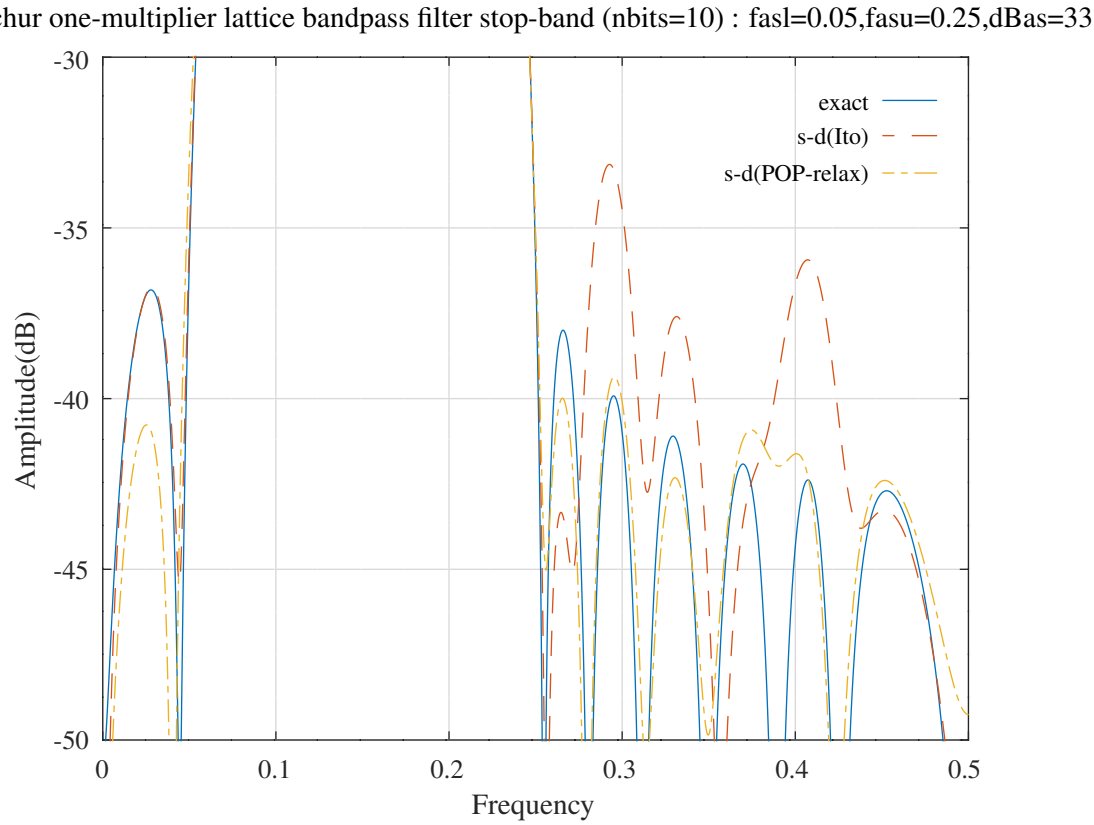



Figure 15.20: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing POP-relaxation search

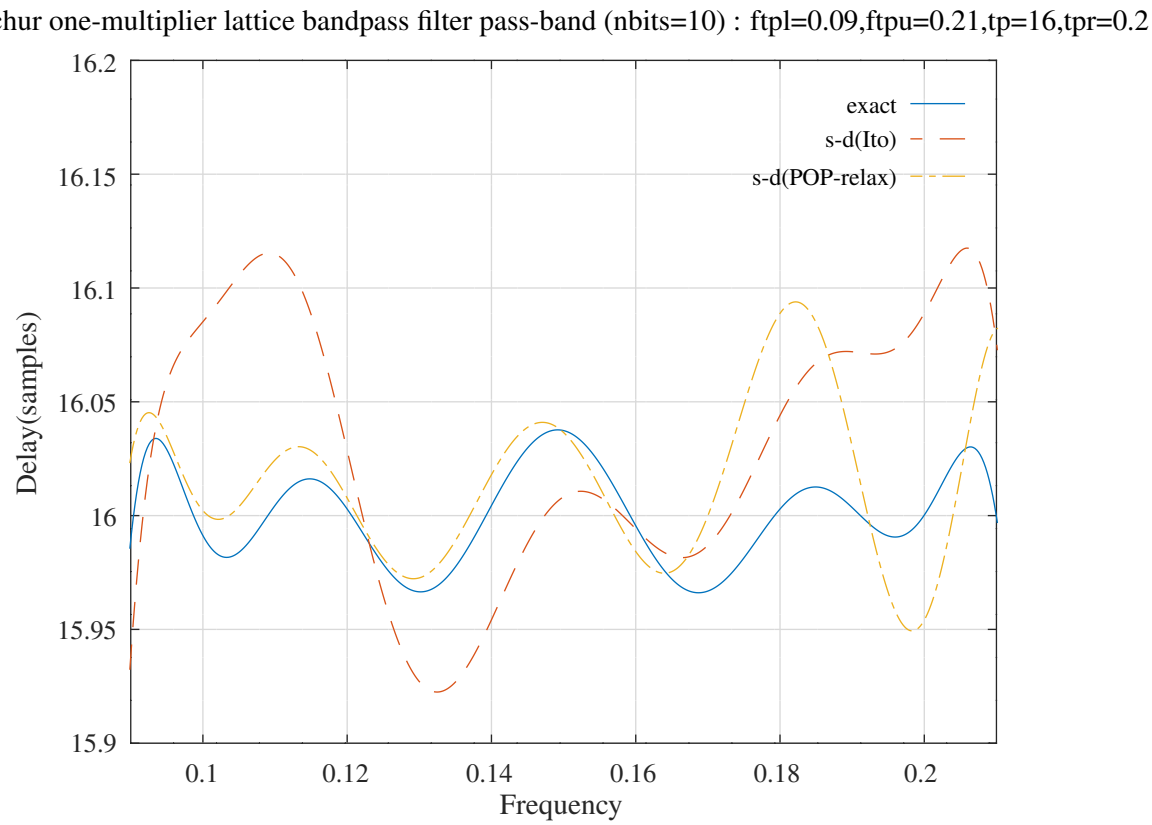



Figure 15.21: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing POP-relaxation search

## 15.12 SOCP-relaxation search for the signed-digit coefficients of a lattice FIR filter

The Gaussian function in the continuous angular frequency domain is:

$$G(\omega) = e^{-\left(\frac{\omega}{\alpha}\right)^2}$$

Appendix L shows that the corresponding time domain Fourier transform pair function is:

$$g(t) = \frac{\alpha}{2\sqrt{\pi}} e^{-\left(\frac{t\alpha}{2}\right)^2}$$

If the symbol interval is $T_S$ then the normalised one-sided half-power bandwidth-symbol time product, $BT_S$, is given by:

$$e^{-\left(\frac{2\pi BT_S}{T_S \alpha}\right)^2} = \frac{1}{\sqrt{2}}$$

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 $k = 0, \ldots, RN_S$. 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 15.22 shows the impulse response of the filter.

The Octave script *socp_relaxation_schurFIRlattice_gaussian_16_nbits_test.m* performs SOCP relaxation to find the 16-bit coefficients of the Gaussian filter implemented as a complementary FIR lattice. (See Appendix N.1 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 11.2. 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
ctol=1e-08 % Tolerance on constraints
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:

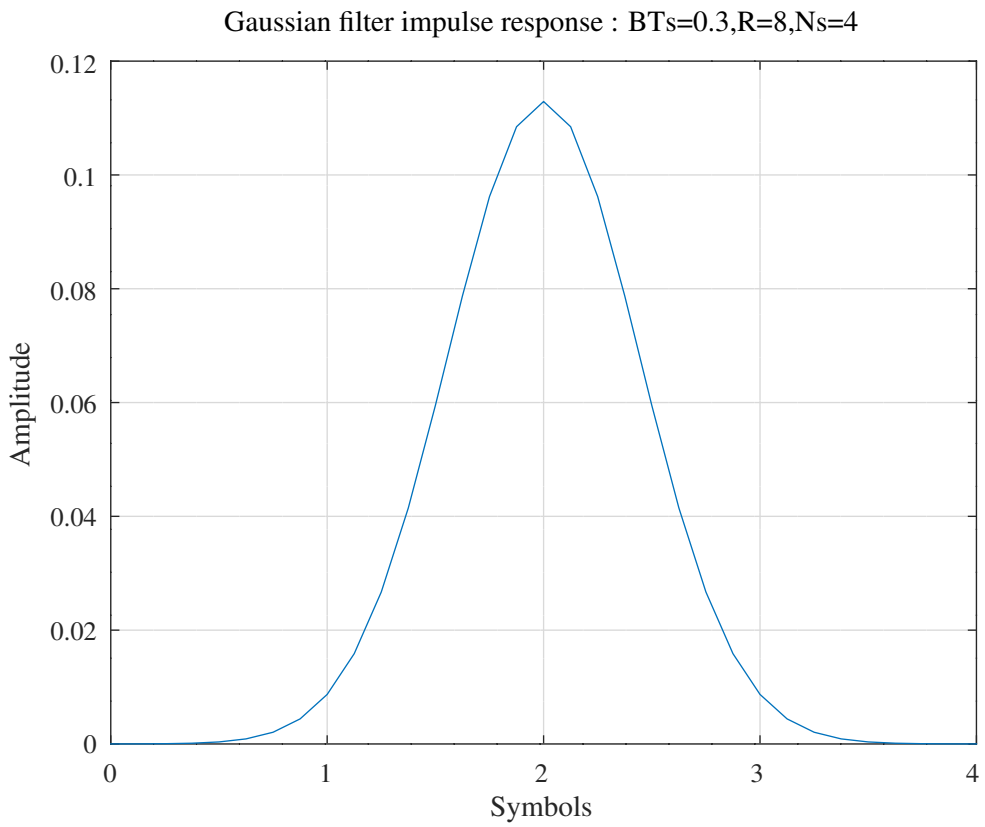



Figure 15.22: Impulse response of a sampled Gaussian filter with half-power bandwidth-symbol-interval product $BT_S = 0.3$,length $N_S = 4$ symbols and $R = 8$ samples per symbol

```
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 ]'/32768;
```

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, ...
           -0.188391850, -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:

```
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 ]'/32768;
```

The 16-bit lattice coefficients with an average of 3-signed-digits optimised by SOCP-relaxation are:

```
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 ]'/32768;
```

Table 15.12 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 multiplier scompared 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 15.12: 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 15.23 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 15.24 compares the amplitude responses in the "pass-band" (up to *dBas*). Figure 15.25 compares the group delay responses in the "pass-band" (up to *dBas*).

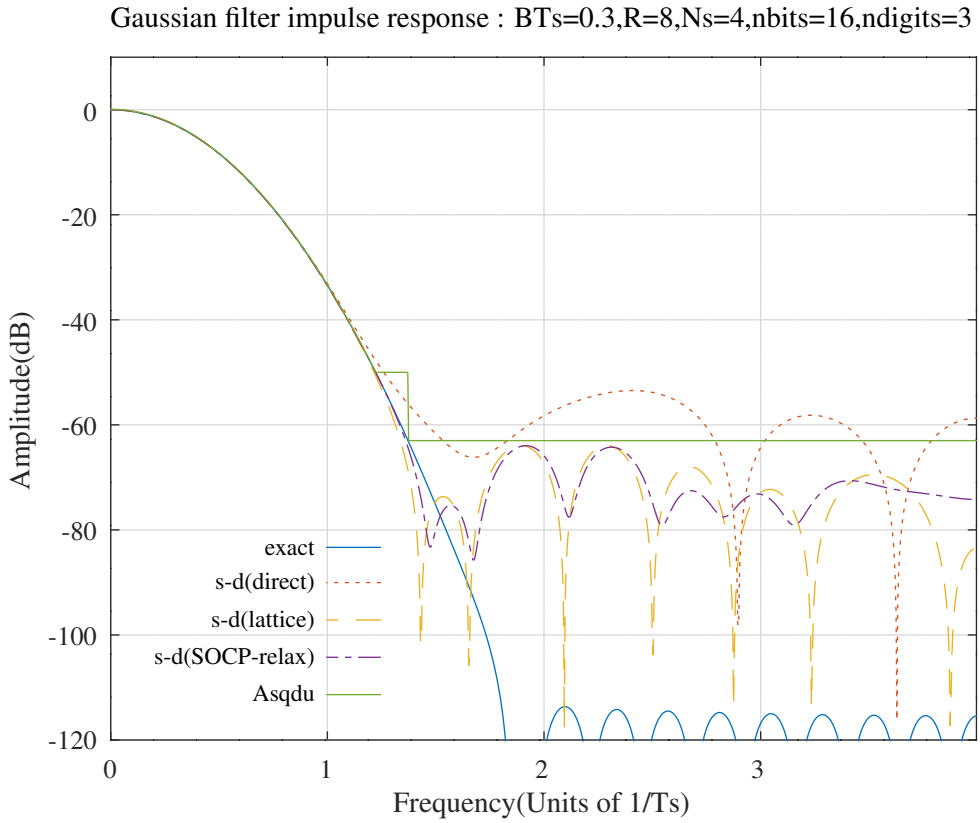



Figure 15.23: Comparison of the amplitude response of the FIR Gaussian filter for exact direct form coefficients, 16-bit 3-signed-digit direct form coefficients, 16-bit coefficients with an average of 3-signed-digits per coefficient and 16-bit coefficients with an average of 3-signed-digits per coefficient optimised by SOCP-relaxation.

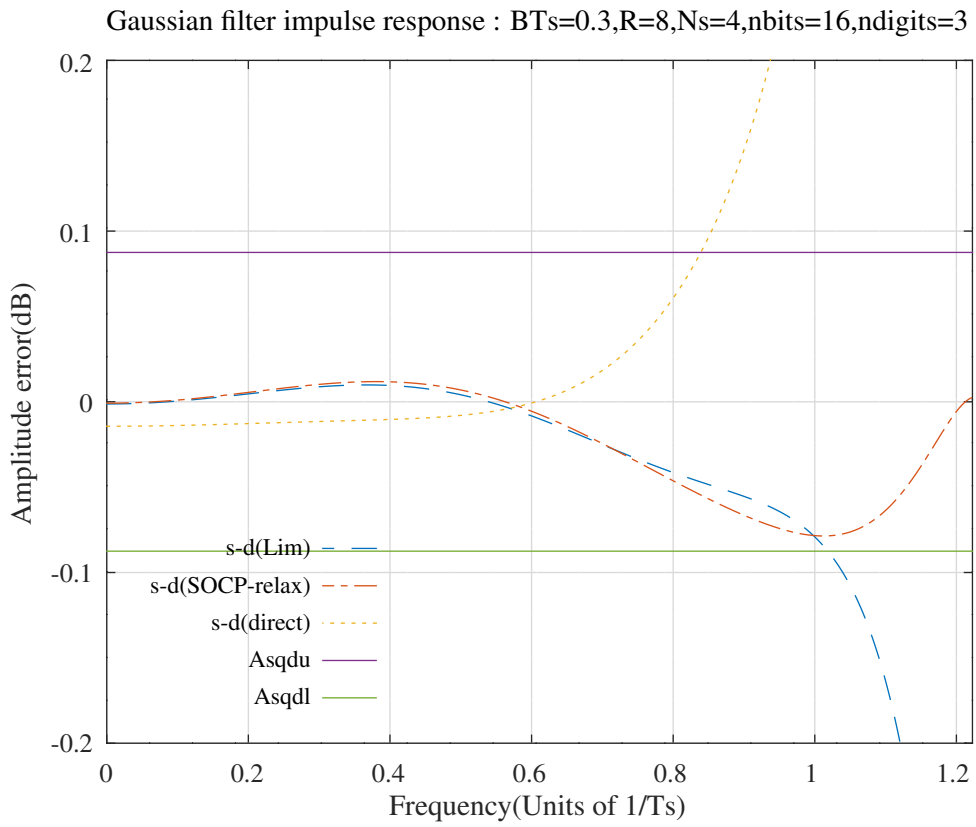



Figure 15.24: Comparison of the pass-band amplitude response of the FIR Gaussian filter for 16-bit coefficients with an average of 3-signed-digits per coefficient and 16-bit coefficients with an average of 3-signed-digits per coefficient optimised by SOCP-relaxation.

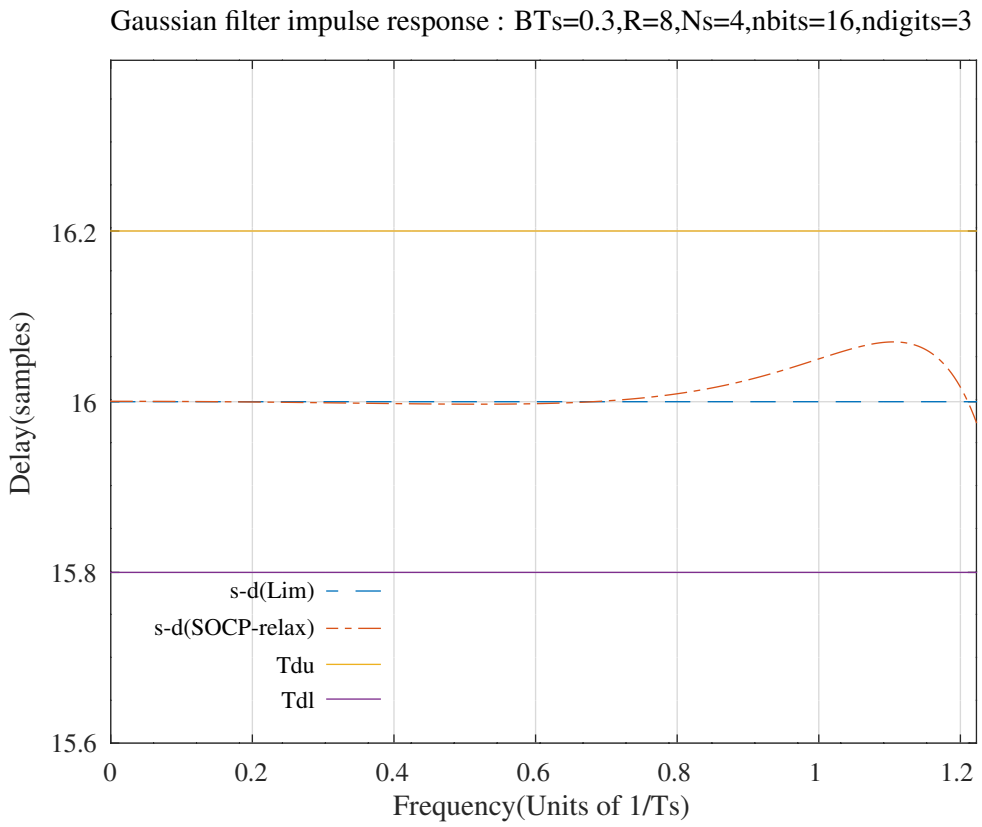



Figure 15.25: Comparison of the "pass-band" (up to *dBas*) group-delay response of the FIR Gaussian filter for 16-bit coefficients with an average of 3-signed-digits per coefficient and 16-bit coefficients with an average of 3-signed-digits per coefficient optimised by SOCP-relaxation.

# Chapter 16

# Semi-definite programming search for integer and signed-digit filter coefficients

This section follows the method introduced by *Lu* [218] and *Ito et al.* [179] to find the signed-digit coefficients of a direct-form FIR filter. The branch-and-bound algorithm of Section 14 and the successive coefficient relaxation methods of Section 15 perform SQP or SOCP optimisation at each coefficient optimisation relaxation step. Alternatively, the optimisation problem can be converted to an overall *integer programming* problem as follows: Given a set of exact coefficients, $\boldsymbol{x}$, and signed-power-of-two upper and lower bounds, $\boldsymbol{u}$ and $\boldsymbol{l}$, corresponding to stable filter implementations such that $u_i \geq x_i \geq l_i$ for each of the $N$ components of $\boldsymbol{x}$, define

$$\hat{\boldsymbol{x}} = \frac{(\boldsymbol{u}+\boldsymbol{l})}{2}$$
$$\boldsymbol{\delta} = \frac{(\boldsymbol{u}-\boldsymbol{l})}{2}$$

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 $y_i = -1$. Define the filter coefficient vector as $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{y} \circ \boldsymbol{\delta}$, where $\circ$ represents the element-by-element product, and approximate the filter amplitude response by $A(\omega) \approx \hat{A}(\omega) + A_\delta(\omega)$[a]. The weighted mean-squared-error of the filter amplitude response is approximately:

$$\mathcal{E} \approx \frac{1}{\pi}\int_0^\pi W(\omega)\left[\hat{A}(\omega) + A_\delta(\omega) - A_d(\omega)\right]^2 d\omega$$

The weighted mean-squared-error of the amplitude response, $\mathcal{E}$, of an IIR filter with signed-digit coefficients, $\hat{\boldsymbol{x}} + (\boldsymbol{y} \circ \boldsymbol{\delta})$, is approximately:

$$\mathcal{E} \approx \mathcal{E}(\hat{\boldsymbol{x}}) + \nabla_x\mathcal{E}(\hat{\boldsymbol{x}})^\top(\boldsymbol{y}\circ\boldsymbol{\delta}) + \frac{1}{2}(\boldsymbol{y}\circ\boldsymbol{\delta})^\top\nabla_x^2\mathcal{E}(\hat{\boldsymbol{x}})(\boldsymbol{y}\circ\boldsymbol{\delta}) \tag{16.1}$$

If $\boldsymbol{Y} = \boldsymbol{y}\boldsymbol{y}^\top$ and $\boldsymbol{\Delta} = \boldsymbol{\delta}\boldsymbol{\delta}^\top$, then mean-squared-error can be expressed in the form[b]:

$$\mathcal{E} \approx \frac{1}{2}\operatorname{trace}\left((\boldsymbol{Q}\circ\boldsymbol{\Delta})\boldsymbol{Y}\right) + (\boldsymbol{q}\circ\boldsymbol{\delta})\boldsymbol{y} + \text{const}$$

The approximate minimum mean-squared-error is found by the integer programming optimisation:

$$\begin{aligned} &\textbf{minimise} && \operatorname{trace}\left(\hat{\boldsymbol{Q}}\boldsymbol{Y}\right) + 2\hat{\boldsymbol{q}}\boldsymbol{y} \\ &\textbf{subject to} && y_i \in \{-1, 1\} \end{aligned}$$

The integer programming constraints are equivalent to:

$$Y_{ii} = 1, \quad \operatorname{rank}\begin{bmatrix} \boldsymbol{Y} & \boldsymbol{y} \\ \boldsymbol{y}^\top & 1 \end{bmatrix} = 1, \quad \begin{bmatrix} \boldsymbol{Y} & \boldsymbol{y} \\ \boldsymbol{y}^\top & 1 \end{bmatrix} \succeq \boldsymbol{0}$$

[a] This is an equality for an FIR filter transfer function.
[b] The *trace* of a matrix is defined in Appendix B.2.

where $\succeq 0$ means that the array is positive semi-definite. *Lu* [218, Section 2.3] suggests a relaxation of the integer programming problem by discarding the rank constraint. The resulting constraints are simplified by defining a symmetric matrix $\hat{\boldsymbol{Y}}$ with ones on the diagonal:

$$\begin{aligned} \textbf{minimise} \quad & \operatorname{trace}\left(\hat{\boldsymbol{Q}}\hat{\boldsymbol{Y}}\right) + 2\hat{\boldsymbol{q}}\boldsymbol{y} \\ \textbf{subject to} \quad & \bar{\boldsymbol{Y}} = \begin{bmatrix} \hat{\boldsymbol{Y}} & \boldsymbol{y} \\ \boldsymbol{y}^\top & 1 \end{bmatrix} \succeq \boldsymbol{0} \end{aligned}$$

Define a length $\frac{N(N+1)}{2}$ augmented variable, $\bar{\boldsymbol{y}}$, consisting of the $N$ elements of $\boldsymbol{y}$ and the $\frac{N(N-1)}{2}$ distinct off-diagonal elements of $\hat{\boldsymbol{Y}}$. The relaxed optimisation problem becomes:

$$\begin{aligned} \textbf{minimise} \quad & \boldsymbol{c}^\top \bar{\boldsymbol{y}} \\ \textbf{subject to} \quad & \bar{\boldsymbol{Y}} \succeq \boldsymbol{0} \end{aligned}$$

*Murumatsu* and *Suzuki* [122, Section 3.3] and *Ito et al.* [179] add *triangle inequality* constraints:

$$\begin{aligned} y_i + y_j + \hat{Y}_{ij} &\geq -1 \\ y_i - y_j - \hat{Y}_{ij} &\geq -1 \\ -y_i - y_j + \hat{Y}_{ij} &\geq -1 \\ -y_i + y_j - \hat{Y}_{ij} &\geq -1 \end{aligned} \tag{16.2}$$

for $0 \leq i < j < N$.

*Lu* [217, Section V] shows how to solve this *semidefinite programming* optimisation problem with the *SeDuMi* [193] SOCP solver by representing it in the form:

$$\begin{aligned} \textbf{minimise} \quad & \boldsymbol{c}^\top \bar{\boldsymbol{y}} \\ \textbf{subject to} \quad & \boldsymbol{A}\bar{\boldsymbol{y}} \geq \boldsymbol{b} \\ & \bar{\boldsymbol{Y}} = \boldsymbol{F}_0 + \bar{y}_1 \boldsymbol{F}_1 + \ldots + \bar{y}_P \boldsymbol{F}_P \succeq \boldsymbol{0} \end{aligned}$$

where $P = \frac{N(N+1)}{2}$, $\bar{\boldsymbol{y}} = [\bar{y}_1 \ldots \bar{y}_P]^\top$ and the $\boldsymbol{F}_k$ are the symmetric matrixes that select the two off-diagonal elements of $\bar{\boldsymbol{Y}}$ corresponding to $\bar{y}_k$. If the SDP solution is $\boldsymbol{y}^*$ and $\hat{\boldsymbol{Y}}^*$, then sign $(\boldsymbol{y}^*)$ is a reasonable solution to the original integer programming problem. *Lu* [218] points out that, given the singular-value decomposition

$$Z^* = \begin{bmatrix} \hat{\boldsymbol{Y}}^* & \boldsymbol{y}^* \\ \boldsymbol{y}^{*\top} & 1 \end{bmatrix} = U \Sigma U^\top$$

an alternative solution is sign $(u_{N+1,l} u_{1\ldots N,l})$, where $u_{1\ldots N,l}$ is the first column vector in $U$ that corresponds to the largest singular value, $\sigma_l$. "*The motivation ... is that a perfect solution* $\boldsymbol{Y}^*$ *... would imply that matrix* $Z^*$ *has rank one, hence* $\sigma_l u_{1\ldots N+1,l} u_{1\ldots N+1,l}^\top$ *is the best rank-one approximation of* $Z^*$ *in the 2-norm sense.*"

## 16.1 SDP optimisation of the signed-digit coefficients of a direct-form symmetric FIR filter

Following Appendix N.3, the components of the amplitude responses of a direct-form symmetic FIR filter with signed-digit coefficients are:

$$\hat{A}(\omega) = \hat{x}_M + 2 \sum_{n=0}^{M-1} \hat{x}_n \cos(M-n)\omega$$

$$A_\delta(\omega) = y_M \delta_M + 2 \sum_{n=0}^{M-1} y_n \delta_n \cos(M-n)\omega$$

The Octave script *sdp_relaxation_directFIRsymmetric_bandpass_10_nbits_test.m* performs SDP optimisation of the response of a direct-form symmetric band-pass FIR filter. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
tol=1e-06 % Tolerance on coef. update
ctol=1e-06 % Tolerance on constraints
npoints=1000 % Frequency points across the band
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=1.5 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
fasl=0.05 % Amplitude stop band(1) lower edge
fasu=0.25 % Amplitude stop band(1) upper edge
dBas=43 % Amplitude stop band(1) peak-to-peak ripple
Wasl=100 % Amplitude lower stop band weight
Wasu=100 % 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 *Lim et al.* as shown in Section 11.1. The Octave function *directFIRsymmetric_sdp_mmsePW* implements SDP optimisation with the triangle inequalities of Equation 16.2. The resulting $\boldsymbol{y}$ values were all very close to $-1$ or 1. Attempts at PCLS optimisation with *directFIRsymmetric_slb* failed.

The distinct direct-form symmetric FIR bandpass filter coefficients found by the SDP optimisation with triangle inequalities are:

```
hM1_sd_sdp = [        0,       -5,      -10,       -5, ...
                     11,       20,       10,       -2, ...
                      5,       20,        0,      -58, ...
                    -91,      -36,       75,      132 ]'/512;
```

Figure 16.1 compares the responses of the filter with exact, 10-bit signed-digit coefficients, 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 SDP optimisation. Table 16.1 compares the cost and the number of 10 bit shift-and-add operations required to implement the 16 distinct coefficient multiplications found by the signed-digit allocation heuristic of *Lim et al.* with the SDP optimisation and triangle inequalities.

| | Cost | Stop-band response | Signed-digits | Shift-and-adds |
|---|---|---|---|---|
| Exact | 0.001901 | -43.0 | | |
| 10-bit 3-signed-digit | 0.005824 | -33.5 | 34 | 20 |
| 10-bit 3-signed-digit(Lim) | 0.002063 | -37.8 | 36 | 22 |
| 10-bit 3-signed-digit(SDP) | 0.001874 | -40.5 | 33 | 19 |

Table 16.1: Comparison of the cost, maximum stop-band response and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for a direct-form symmetric FIR bandpass 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 SDP optimisation with triangle inequalities.

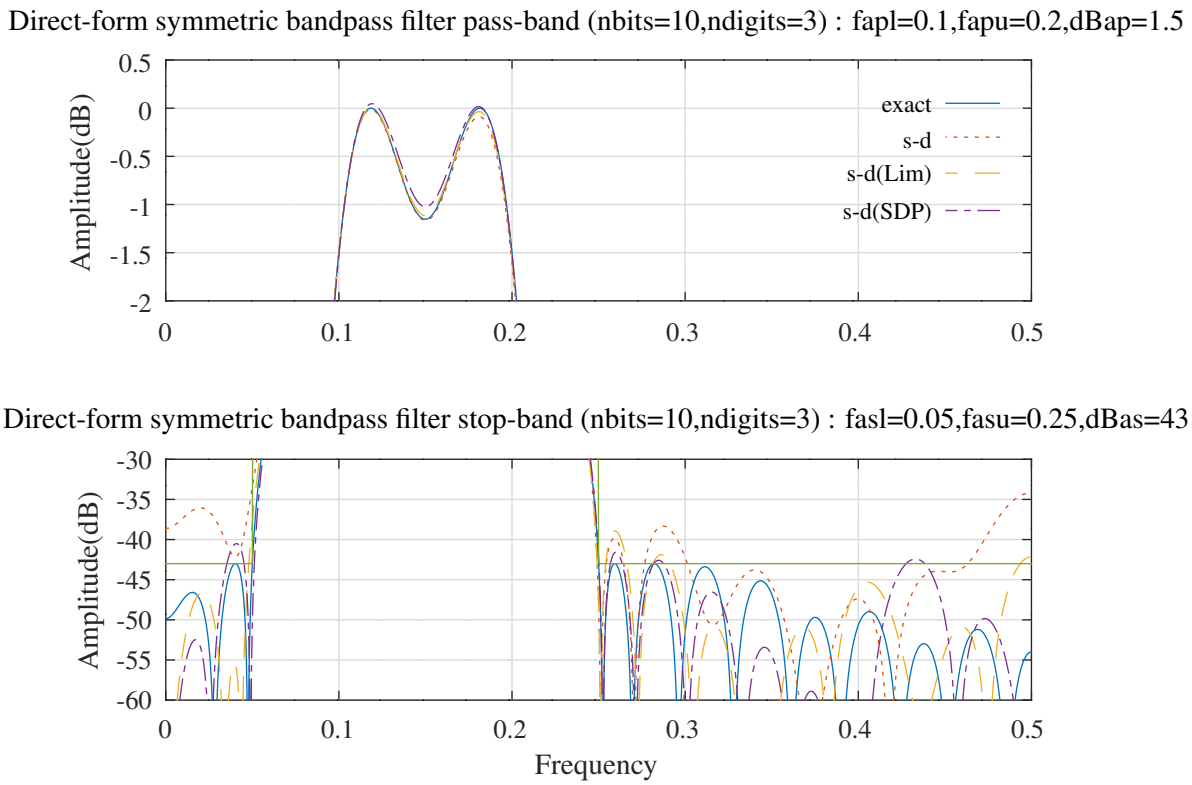



Figure 16.1: Comparison of the amplitude responses for a direct-form symmetric FIR bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP optimisation with triangle inequalities.

## 16.2 SDP optimisation of the signed-digit coefficients of an FIR Hilbert filter

The Octave script *sdp_relaxation_directFIRhilbert_12_nbits_test.m* performs SDP optimisation of the response of a direct-form Hilbert FIR filter. The filter specification is:

```
M=40 % Number of distinct coefficients
nbits=12 % Coefficient bits
ndigits=2 % Nominal average coefficient signed-digits
tol=1e-05 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
npoints=500 % Frequency points across the band
fapl=0.01 % Amplitude pass band lower edge
fapu=0.49 % Amplitude pass band upper edge
dBap=0.135 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Was=0 % Amplitude stop band weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 2 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2. The Octave function *directFIRhilbert_sdp_mmsePW* implements SDP optimisation with the triangle inequalities of Equation 16.2. The resulting $\boldsymbol{y}$ values were all very close to $-1$ or 1.

The distinct direct-form Hilbert FIR filter coefficients found by signed-digit allocation with the heuristic of *Ito et al.* are:

```
hM1_sd_Ito = [       -1,       -1,       -1,       -2, ...
                     -2,       -2,       -2,       -4, ...
                     -4,       -4,       -4,       -8, ...
                     -8,       -8,       -8,      -10, ...
                    -12,      -12,      -16,      -16, ...
                    -18,      -20,      -24,      -24, ...
                    -28,      -32,      -36,      -40, ...
                    -48,      -52,      -60,      -69, ...
                    -80,      -96,     -112,     -140, ...
                   -183,     -258,     -432,    -1304 ]'/2048;
```

and by SDP optimisation with triangle inequalities are:

```
hM1_sd_sdp = [       -1,       -1,       -1,       -1, ...
                     -2,       -2,       -2,       -2, ...
                     -4,       -4,       -4,       -4, ...
                     -8,       -8,       -8,      -10, ...
                    -12,      -12,      -16,      -16, ...
                    -18,      -20,      -24,      -24, ...
                    -28,      -32,      -36,      -40, ...
                    -48,      -52,      -60,      -69, ...
                    -80,      -96,     -112,     -140, ...
                   -183,     -258,     -432,    -1304 ]'/2048;
```

The corresponding FIR filter is:

```
h=[-flipud(kron(hM1_sd_sdp,[1;0]));0;kron(hM1_sd_sdp,[1;0])](2:end-1)
```

Figure 16.2 compares the responses of the filter with exact, 12-bit signed-digit coefficients, 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and SDP optimisation. Table 16.2 compares the cost, maximum pass-band response error and the number of 12 bit shift-and-add operations required to implement the 40 distinct coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* and performing SDP optimisation with triangle inequalities.

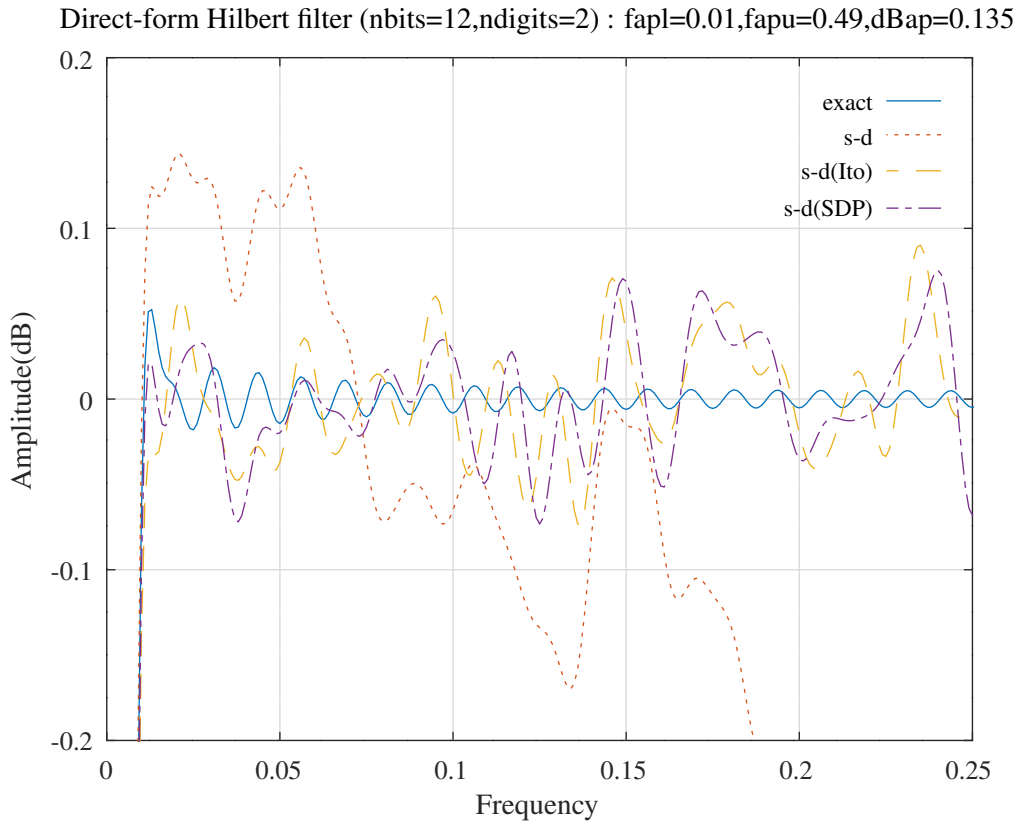



Figure 16.2: Comparison of the amplitude responses for a direct-form Hilbert FIR filter with 12-bit integer coefficients found by allocating an average of 2-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SDP optimisation with triangle inequalities.

| | Cost | Pass-band response | Signed-digits | Shift-and-adds |
|---|---|---|---|---|
| Exact | 5.232e-07 | 0.0675 | | |
| 12-bit 2-signed-digit | 0.0002086 | 0.4029 | 71 | 31 |
| 12-bit 2-signed-digit(Ito) | 7.87e-06 | 0.1384 | 70 | 30 |
| 12-bit 2-signed-digit(SDP) | 6.988e-06 | 0.1013 | 70 | 30 |

Table 16.2: Comparison of the cost, maximum pass-band response error (in dB) and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a direct-form Hilbert FIR filter with 12-bit integer coefficients found by allocating an average of 2-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SDP optimisation with triangle inequalities.

## 16.3 SDP optimisation of the signed-digit coefficients of an FIR Hilbert band-pass filter

The Octave script *sdp_relaxation_directFIRhilbert_bandpass_12_nbits_test.m* performs SDP optimisation of the response of a direct-form Hilbert band-pass FIR filter. The initial filter is that shown in Appendix N.4.3. The filter specification is:

```
M=8 % Number of distinct coefficients
nbits=12 % Coefficient bits
ndigits=2 % Nominal average coefficient signed-digits
tol=1e-05 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
npoints=5000 % Frequency points across the band
fasl=0.1 % Amplitude stop band lower edge
fapl=0.16325 % Amplitude pass band lower edge
fapu=0.33675 % Amplitude pass band upper edge
fasu=0.4 % Amplitude stop band upper edge
Wap=2 % Amplitude pass band weight
dBap=0.1 % Amplitude pass band peak-to-peak ripple
Wat=0.001 % Amplitude transition band weight
Was=1 % Amplitude stop band weight
dBas=35 % Amplitude stop band peak ripple
```

The group-delay of the filter is $2M - 1$ samples. The filter coefficients are truncated to 12 bits allocated with an average of 2 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2. The Octave function *directFIRhilbert_sdp_mmsePW* implements SDP optimisation with the triangle inequalities of Equation 16.2.

The distinct direct-form Hilbert band-pass FIR filter coefficients found by signed-digit allocation with the heuristic of *Ito et al.* are:

```
hM2_sd_Ito = [      -20,        8,       64,      -32, ...
                   -128,      112,      327,     -868 ]'/2048;
```

and by SDP optimisation with triangle inequalities are:

```
hM2_sd_sdp = [      -24,        8,       64,      -32, ...
                   -128,      120,      326,     -872 ]'/2048;
```

Figure 16.3 compares the responses of the filter with exact, 12-bit signed-digit coefficients, 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and 12-bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and SDP optimisation. Figure 16.4 compares the pass-band responses. Table 16.3 compares the cost, maximum stop-band response error and the number of 12 bit shift-and-add operations required to implement the 8 distinct coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the SDP relaxation optimisation and triangle inequalities.

| | Cost | Stop-band response | Signed-digits | Shift-and-adds |
|---|---|---|---|---|
| Exact | 0.0001338 | -34.97 | | |
| 12-bit 2-signed-digit | 0.0006283 | -28.15 | 16 | 8 |
| 12-bit 2-signed-digit(Ito) | 0.0001635 | -33.11 | 16 | 8 |
| 12-bit 2-signed-digit(SDP) | 0.0001392 | -34.89 | 16 | 8 |

Table 16.3: Comparison of the cost, maximum stop-band response error and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a direct-form Hilbert band-pass FIR filter with 12-bit integer coefficients found by allocating an average of 2-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SDP optimisation with triangle inequalities.

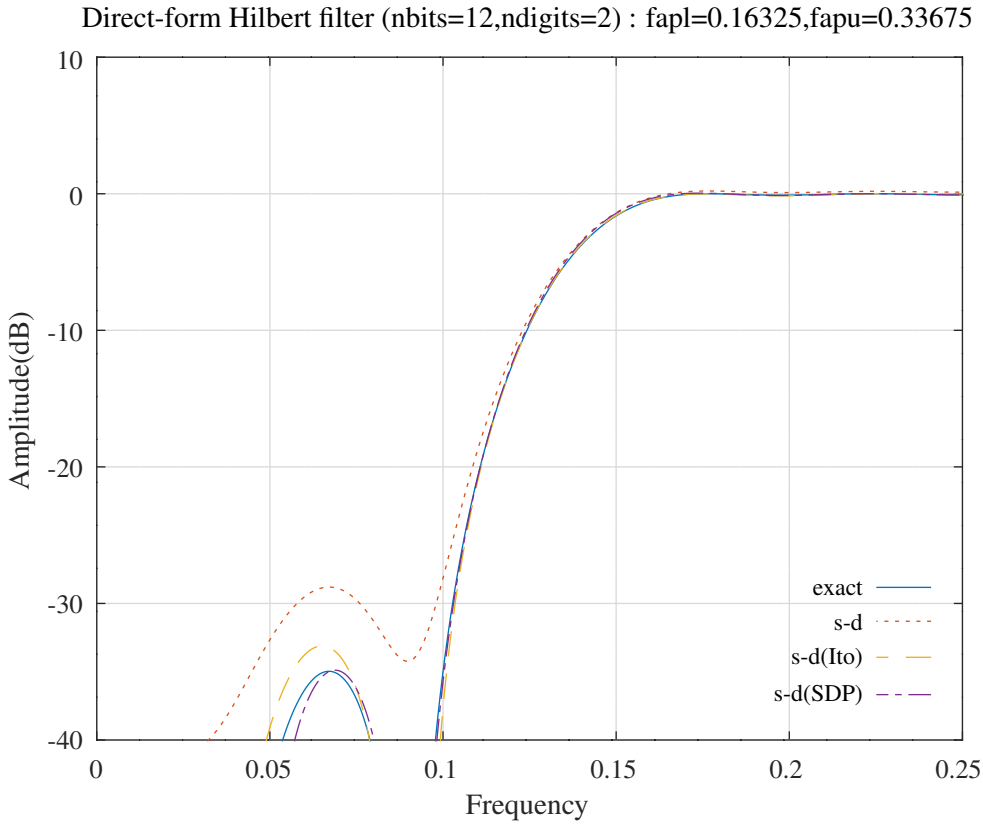



Figure 16.3: Comparison of the amplitude responses for a direct-form Hilbert band-pass FIR filter with 12-bit integer coefficients found by allocating an average of 2-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SDP optimisation with triangle inequalities.

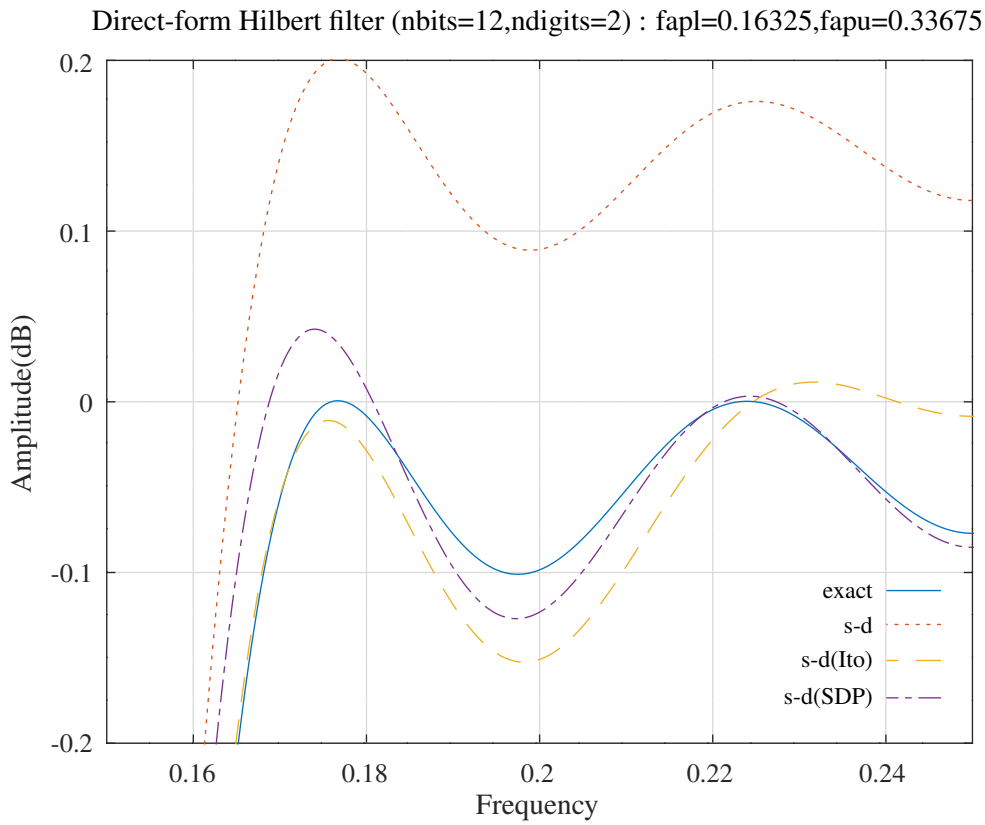



Figure 16.4: Comparison of the pass-band amplitude responses for a direct-form Hilbert band-pass FIR filter with 12-bit integer coefficients found by allocating an average of 2-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SDP optimisation with triangle inequalities.

## 16.4 SDP-relaxation search for the signed-digit coefficients of a lattice bandpass IIR filter

The Octave script *sdp_relaxation_schurOneMlattice_bandpass_10_nbits_test.m* performs successive SDP relaxations to optimise the response of the SQP optimised band-pass Schur one-multiplier lattice filter of Section 10.3.6. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
tol=0.0001 % Tolerance on coef. update
ctol=0.0001 % Tolerance on constraints
n=500 % Frequency points across the band
Nk=20 % Filter order
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
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
dBas=35 % Amplitude stop band peak-to-peak ripple
Wasl=500000 % Amplitude lower stop band weight
Wasu=1000000 % Amplitude upper stop band weight
ftpl=0.1 % Pass band delay lower edge
ftpu=0.2 % Pass band delay upper edge
tp=16 % Nominal pass band filter group delay
tpr=0.32 % Delay pass band peak-to-peak ripple
Wtp=2 % Delay 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 11.1.

The script compares the filter designed by a single call to the Octave function *schurOneMlattice_sdp_mmse* with that designed by successively fixing the value of the remaining active coefficients to the SDP coefficient corresponding to that with the largest difference between the upper and lower signed-digit approximations as in Section 15.2. 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. Then the active coefficients are optimised with *schurOneMlattice_sdp_mmse*, the set of active coefficients is updated and the remaining active coefficients are optimised with the Octave function *schurOneMlattice_socp_mmse* and the *Selesnick-Lang-Burrus* constraint algorithm implemented in the Octave function *schurOneMlattice_slb*. 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 a single "global" SDP optimisation are:

```
k0_sd_sdp = [        0,      340,        0,      255, ...
                     0,      177,        0,      212, ...
                     0,      156,        0,      129, ...
                     0,       76,        0,       52, ...
                     0,       18,        0,        7 ]'/512;

c0_sd_sdp = [       36,       -6,     -153,     -247, ...
                   -82,       63,      202,      153, ...
                     8,      -42,      -41,       -7, ...
                    -6,      -19,      -13,        4, ...
                    13,        9,        1,        0, ...
                     2 ]'/512;
```

The filter coefficients found by the SDP-relaxation search are:

```
k0_sd_min = [        0,      340,        0,      255, ...
                     0,      177,        0,      216, ...
                     0,      156,        0,      132, ...
                     0,       79,        0,       52, ...
                     0,       17,        0,        6 ]'/512;
```

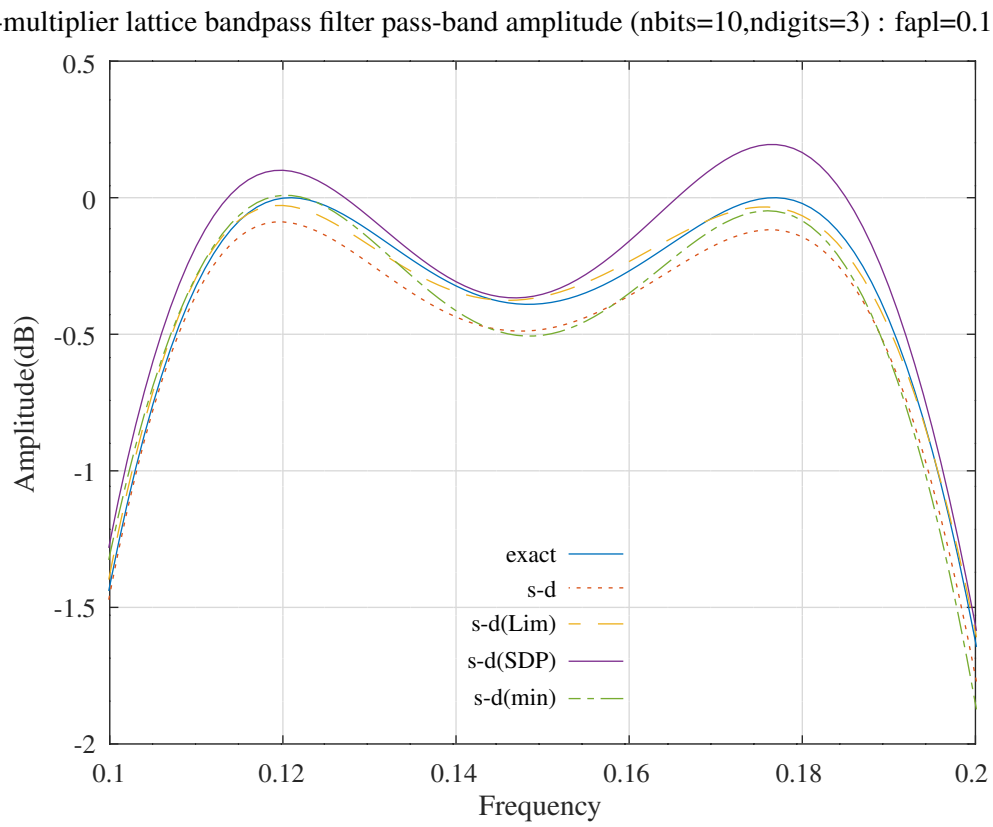



Figure 16.5: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search

```
c0_sd_min = [        36,        -7,      -154,      -246, ...
                    -82,        63,       201,       154, ...
                     11,       -40,       -41,        -8, ...
                     -5,       -17,       -12,         2, ...
                     12,        10,         2,         0, ...
                      2 ]'/512;
```

Figure 16.5 compares the pass-band 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 SDP-relaxation search. Figure 16.6 shows the filter stop-band response and Figure 16.7 shows the filter pass-band group delay response. The "global" unconstrained SDP optimisation gives a poor delay response. Application of the *Selesnick-Lang-Burrus* PCLS algorithm to that "global" signed-digit SDP optimisation failed by entering a loop of constraints. Figure 16.8 shows the history of the difference of the successive relaxed signed-digit coefficients from the initial exact values. Table 16.4 compares the cost and the number of 10 bit shift-and-add operations required to implement the 20 coefficient multiplications found by the signed-digit allocation heuristic of *Lim et al.* with the SDP-relaxation search. A further 51 additions are required by the lattice filter structure.

| | Cost | Stop-band response | Signed-digits | Shift-and-adds |
|---|---|---|---|---|
| Exact | 0.014314 | -36.0 | | |
| 10-bit 3-signed-digit | 0.032112 | -35.8 | 73 | 42 |
| 10-bit 3-signed-digit(Lim) | 0.027798 | -35.2 | 78 | 47 |
| 10-bit 3-signed-digit(SDP) | 0.047554 | -33.0 | 78 | 48 |
| 10-bit 3-signed-digit(min) | 0.019727 | -34.5 | 74 | 44 |

Table 16.4: 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 *Lim et al.* and performing SDP-relaxation search

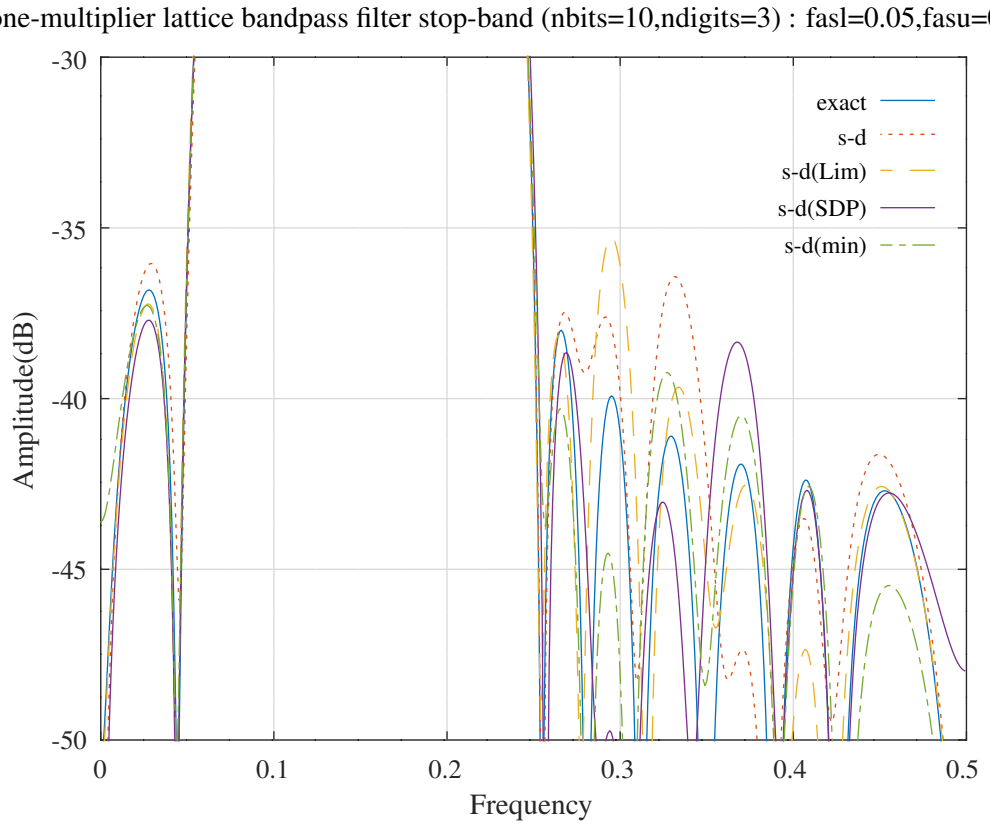



Figure 16.6: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search

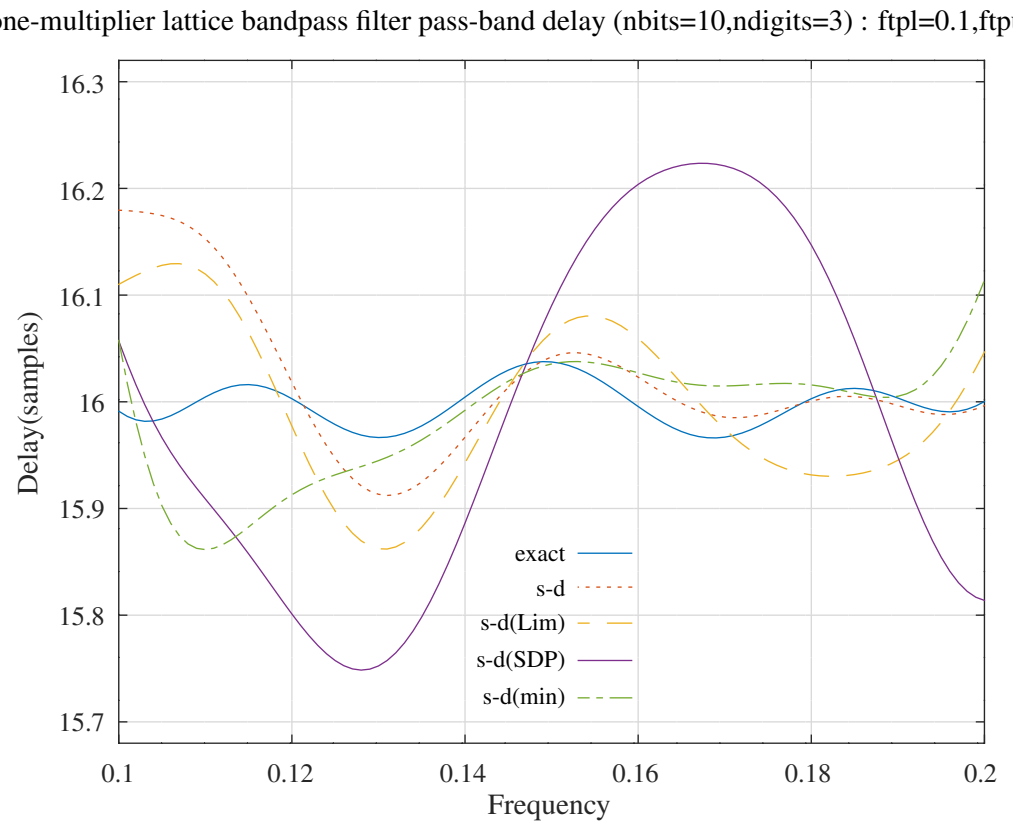



Figure 16.7: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 10-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search

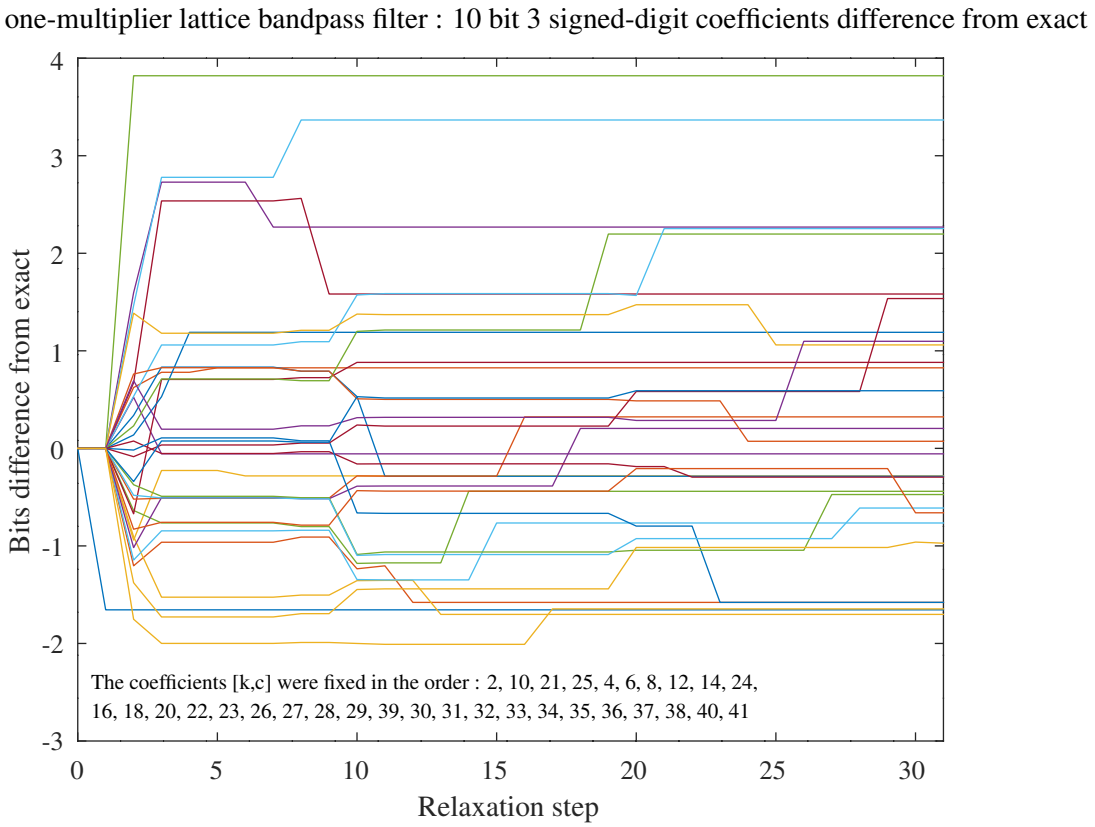



Figure 16.8: History of the difference of the successive SDP relaxed signed-digit coefficients from the initial exact values.

## 16.5 SDP-relaxation search for the signed-digit coefficients of a parallel allpass lattice elliptic low-pass IIR filter

The Octave script *sdp_relaxation_schurOneMPAlattice_elliptic_lowpass_16_nbits_test.m* performs successive SDP relaxations to optimise the response of an elliptic low-pass filter implemented with 16-bit coefficients. The filter specification is:

```
nbits=16 % Coefficient bits
ndigits=4 % Nominal average coefficient signed-digits
tol=1e-05 % Tolerance on coef. update
ctol=1e-08 % Tolerance on constraints
n=1000 % Frequency points across the band
difference=0 % Use difference of all-pass filters
rho=0.992188 % Constraint on allpass coefficients
m1=5 % All-pass filter 1 order
m2=4 % All-pass filter 2 order
fap=0.15 % Amplitude pass band edge
dBap=0.06 % Amplitude pass band peak-to-peak ripple
Wape=0 % Extra amplitude pass band weight
Wap=1 % Amplitude pass band weight
fas=0.171 % Amplitude stop band edge
dBas=76 % Amplitude stop band peak-to-peak ripple
Was=10000000 % Amplitude stop band weight
Wase=0 % Extra amplitude stop band weight
```

The filter specification is similar to that for the filter designed with branch-and-bound search shown in Section 14.6. The initial parallel all-pass filters are those for the filter designed by the Octave function *ellip(11,0.02,84,2*0.15)*. The filter coefficients are truncated to 16-bit coefficients with an average of 4 signed-digits allocated by the heuristic of *Ito et al.*, as shown in Section 11.2. The following numbers of signed-digits are allocated to the lattice coefficients:

```
A1k0_allocsd_digits = [  4,  3,  5,  5, ...
                         4,  4 ]';


A2k0_allocsd_digits = [  3,  3,  5,  5, ...
                         3 ]';
```

The script compares the filter designed by a single call to the Octave function *schurOneMPAlattice_sdp_mmse* with that designed by successively fixing the value of the remaining active coefficients to the SDP coefficient corresponding to that with the largest difference between the upper and lower signed-digit approximations as in Section 15.2. 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. Then the active coefficients are optimised with *schurOneMPAlattice_sdp_mmse*, the set of active coefficients is updated and the remaining active coefficients are optimised with the Octave function *schurOneMPAlattice_socp_mmse* and the *Selesnick-Lang-Burrus* constraint algorithm implemented in the Octave function *schurOneMPAlattice_slb*. 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 SDP-relaxation search are:

```
A1k0_sd_min = [   -19584,    32384,   -25728,    28352, ...
                  -23616,    11808 ]'/32768;


A2k0_sd_min = [   -22560,    30680,   -26880,    23936, ...
                  -11760 ]'/32768;
```

Figures 16.9 and 16.10 compare the pass-band and stop-band responses of the filter with exact, 16-bit coefficients with an average of 4 signed-digits allocated with the algorithm of *Ito et al.*, signed-digit coefficients found with "global" SDP approximation and those found by SDP-relaxation search. Figure 16.11 shows the history of the difference of the successive relaxed signed-digit coefficients from the initial exact values. Table 16.5 compares the cost and the number of 16 bit shift-and-add operations required to implement the 11 coefficient multiplications.

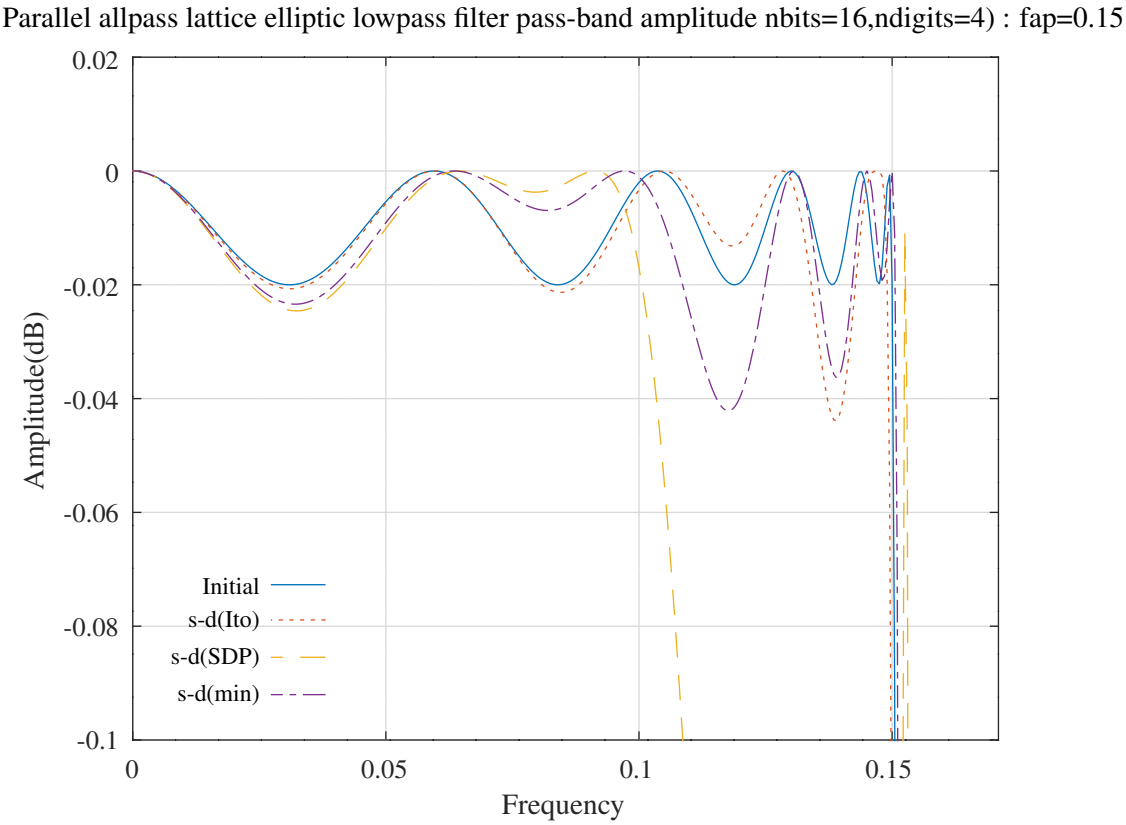



Figure 16.9: Comparison of the pass-band amplitude responses for a parallel all-pass Schur one-multiplier lattice elliptic low-pass filter with 16-bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SDP relaxation search

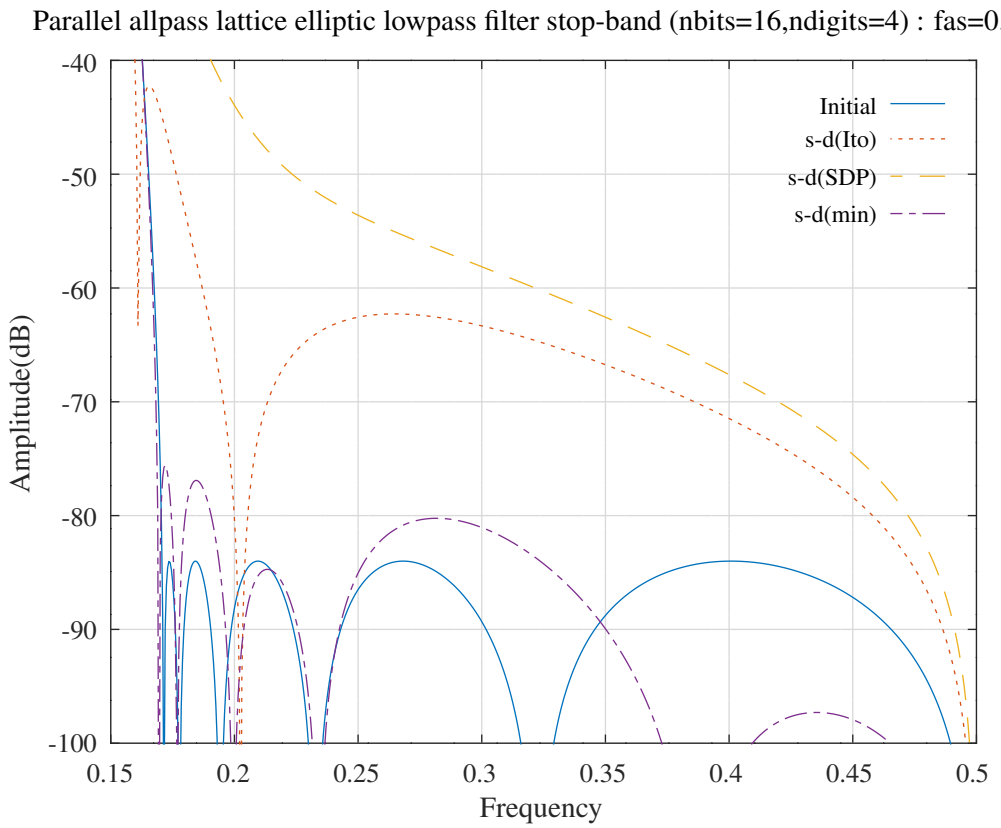



Figure 16.10: Comparison of the stop-band amplitude responses for a parallel all-pass Schur one-multiplier lattice elliptic low-pass filter with 16-bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SDP relaxation search

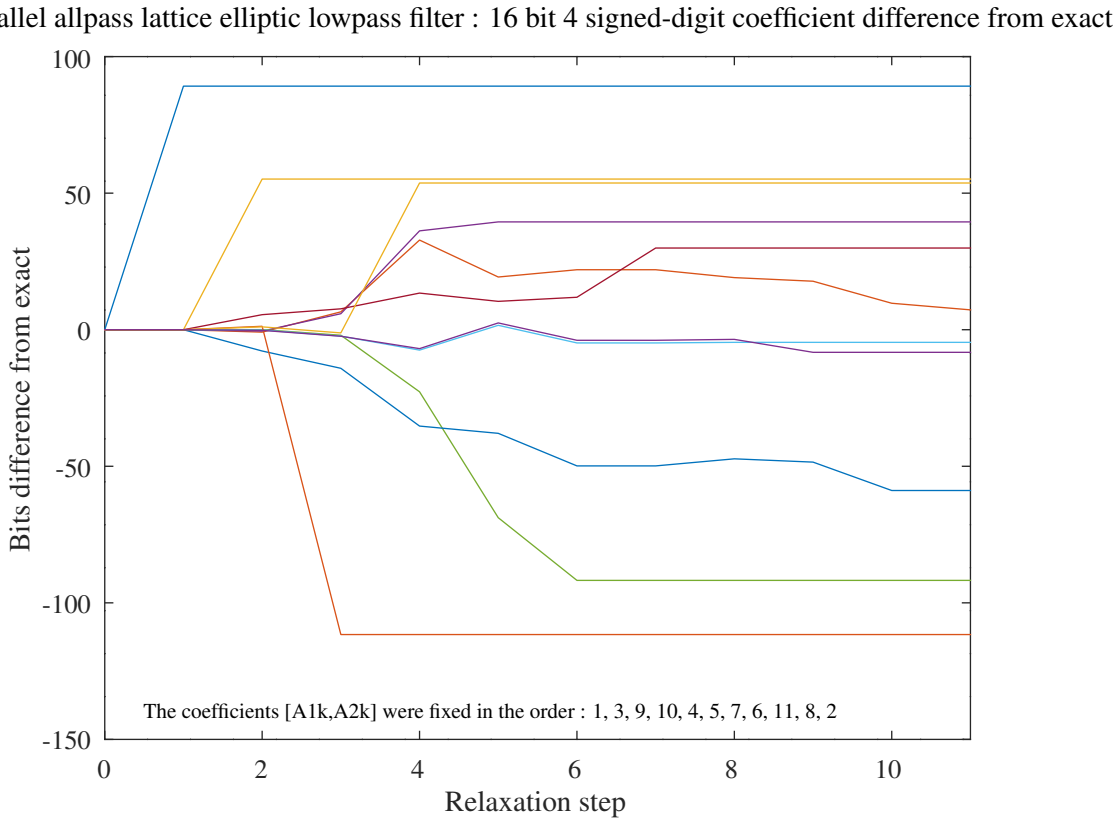



Figure 16.11: History of the difference of the successive SDP relaxed signed-digit coefficients from the initial exact values

| | Cost | Stop-band response | Signed-digits | Shift-and-adds |
|---|---|---|---|---|
| Initial | 7.52e-06 | -84.0 | | |
| 16-bit 4-signed-digit(Ito) | 1.76e-04 | -45.2 | 44 | 33 |
| 16-bit 4-signed-digit(SDP) | 6.00e-01 | -27.5 | 44 | 33 |
| 16-bit 4-signed-digit(min) | 1.40e-05 | -75.6 | 43 | 32 |

Table 16.5: Comparison of the cost, stop-band attenuation and number of 16-bit shift-and-add operations required to implement the coefficient multiplications for a parallel all-pass Schur one-multiplier lattice elliptic-lowpass filter with 16-bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SDP relaxation search

## 16.6 SDP-relaxation search for the signed-digit coefficients of a parallel allpass lattice bandpass Hilbert IIR filter

The Octave script *sdp_relaxation_schurOneMPAlattice_bandpass_hilbert_13_nbits_test.m* performs successive SDP relaxations to optimise the response of the band-pass Hilbert filter designed by the script *parallel_allpass_socp_slb_bandpass_hilbert_test.m* described in Section 10.2.3 implemented with 13 bit coefficients each having an average of 3 signed-digits allocated by the heuristic of *Lim et al.* as shown in Section 11.1. The filter specification is:

```
nbits=13 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
tol=0.0001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
ma=10 % All-pass filter a order
mb=10 % All-pass filter b order
fapl=0.12 % Amplitude pass band lower edge
fapu=0.18 % Amplitude pass band upper edge
dBap=0.1 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
dBas=35 % Amplitude stop band peak-to-peak ripple
Wasl=2000 % Amplitude lower stop band weight
Wasu=2000 % Amplitude upper stop band weight
ftpl=0.12 % Pass band delay lower edge
ftpu=0.18 % Pass band delay upper edge
td=16 % Nominal pass band filter group delay
tdr=0.4 % Delay pass band peak-to-peak ripple
Wtp=2 % Delay pass band weight
fppl=0.12 % Pass band phase response lower edge
fppu=0.18 % Pass band phase response upper edge
pd=3.5 % Pass band initial phase response (rad./pi)
pdr=0.03 % Pass band phase response ripple(rad./pi)
Wpp=500 % Pass band phase response weight
```

The script compares the filter designed by a single call to the Octave function *schurOneMPAlattice_sdp_mmse* with that designed by successively fixing the value of the remaining active coefficients to the SDP coefficient corresponding to that with the largest difference between the upper and lower signed-digit approximations as in Section 15.2. 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. Then the active coefficients are optimised with *schurOneMPAlattice_sdp_mmse*, the set of active coefficients is updated and the remaining active coefficients are optimised with the Octave function *schurOneMPAlattice_socp_mmse* and the *Selesnick-Lang-Burrus* constraint algorithm implemented in the Octave function *schurOneMPAlattice_slb*. 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 a single "global" SDP optimisation are:

```
A1k0_sd_sdp = [    -1730,     2558,     1952,    -2179, ...
                    2816,    -1152,     -256,     1568, ...
                    -960,      384 ]'/4096;
```

```
A2k0_sd_sdp = [    -3114,     3000,     1887,    -2352, ...
                    2928,    -1044,     -240,     1536, ...
                    -968,      448 ]'/4096;
```

The filter coefficients found by the SDP-relaxation search are:

```
A1k0_sd_min = [    -1792,     2528,     1952,    -2177, ...
                    2816,    -1152,     -256,     1568, ...
                    -960,      448 ]'/4096;
```

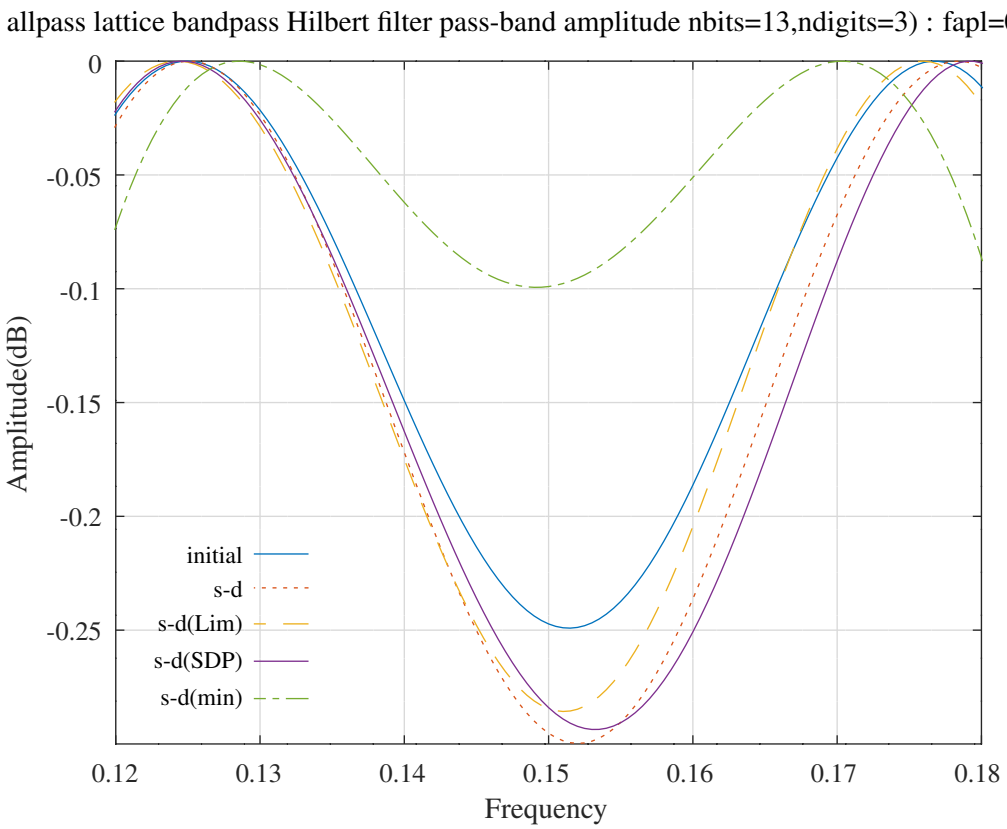



Figure 16.12: Comparison of the pass-band amplitude responses for a parallel all-pass Schur one-multiplier lattice bandpass Hilbert filter with 13-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search

```
A2k0_sd_min = [    -3107,     3016,     1886,    -2303, ...
                    2912,     -991,     -192,     1536, ...
                    -928,      448 ]'/4096;
```

Figures 16.12, Figure 16.13 and Figure 16.14 compare the pass band responses of the filter with exact, 13-bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and 13-bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and SDP-relaxation search. The pass band phase response shown is adjusted for the nominal delay. Likewise, Figure 16.15 shows the filter stop band response. Table 16.6 compares the cost and the number of 13 bit shift-and-add operations required to implement the 20 coefficient multiplications found by the signed-digit allocation heuristic of *Lim et al.* with the SDP-relaxation search.

| | Cost | Stop-band response | Signed-digits | Shift-and-adds |
|---|---|---|---|---|
| Exact | 0.001400 | -35.2 | | |
| 13-bit 3-signed-digit | 0.129012 | -28.5 | 60 | 40 |
| 13-bit 3-signed-digit(Lim) | 0.092708 | -33.5 | 59 | 39 |
| 13-bit 3-signed-digit(SDP) | 0.339782 | -27.5 | 60 | 40 |
| 13-bit 3-signed-digit(min) | 0.001890 | -35.0 | 56 | 36 |

Table 16.6: Comparison of the cost and number of 13-bit shift-and-add operations required to implement the coefficient multiplications for a parallel all-pass Schur one-multiplier lattice bandpass Hilbert filter with 13-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search

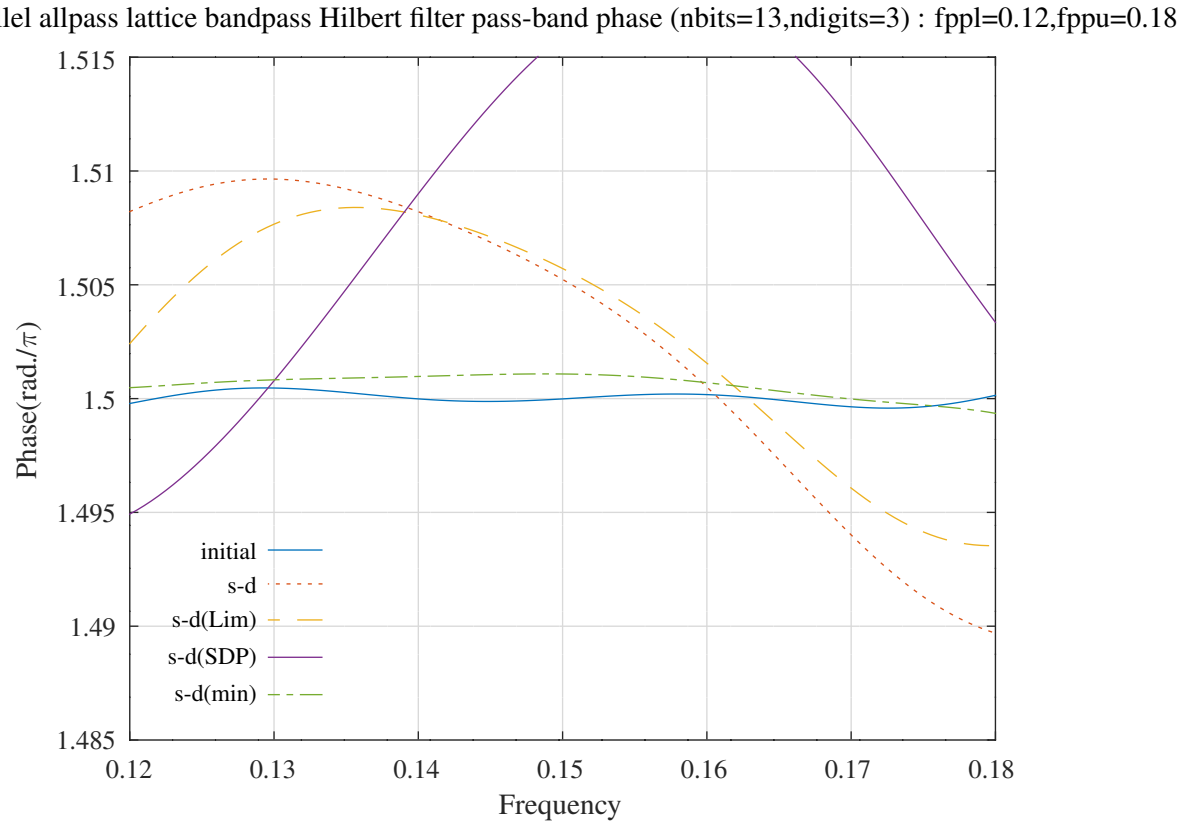



Figure 16.13: Comparison of the pass-band phase responses for a parallel all-pass Schur one-multiplier lattice bandpass Hilbert filter with 13-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search. The phase responses shown are adjusted for the nominal delay.

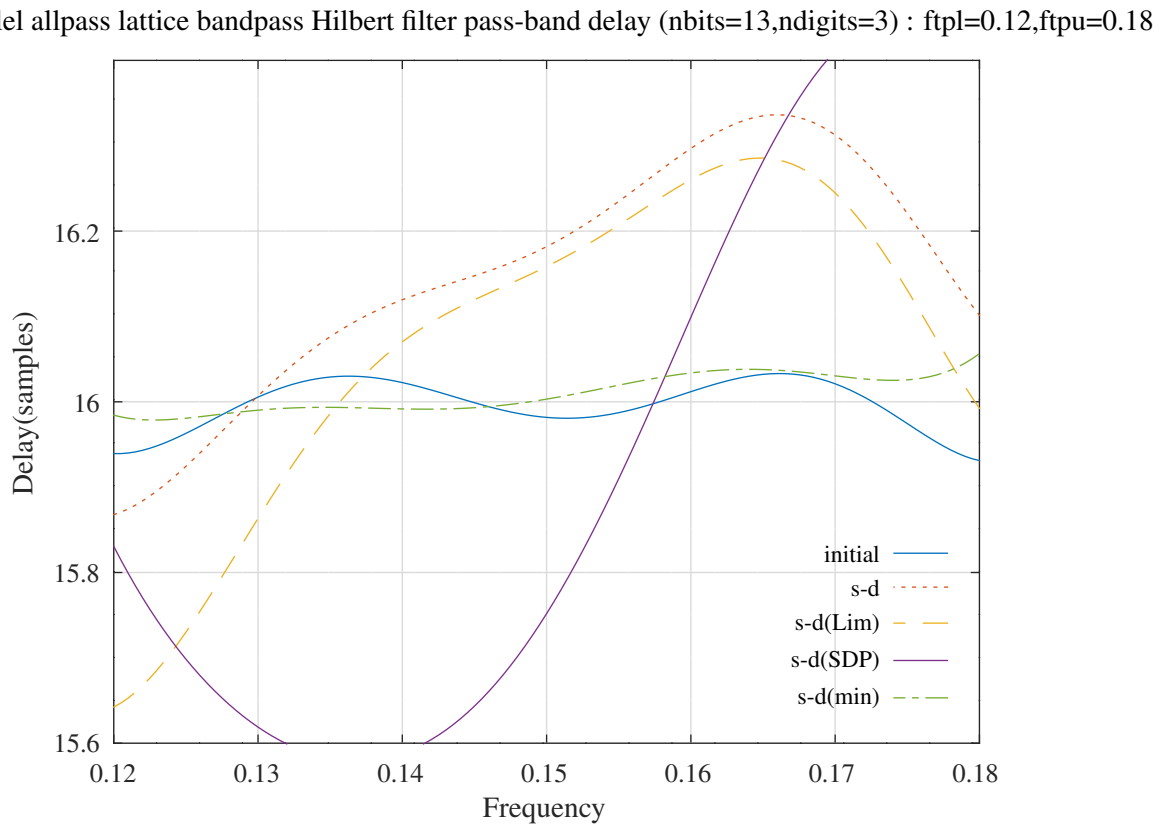



Figure 16.14: Comparison of the pass-band group delay responses for a parallel all-pass Schur one-multiplier lattice bandpass Hilbert filter with 13-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search

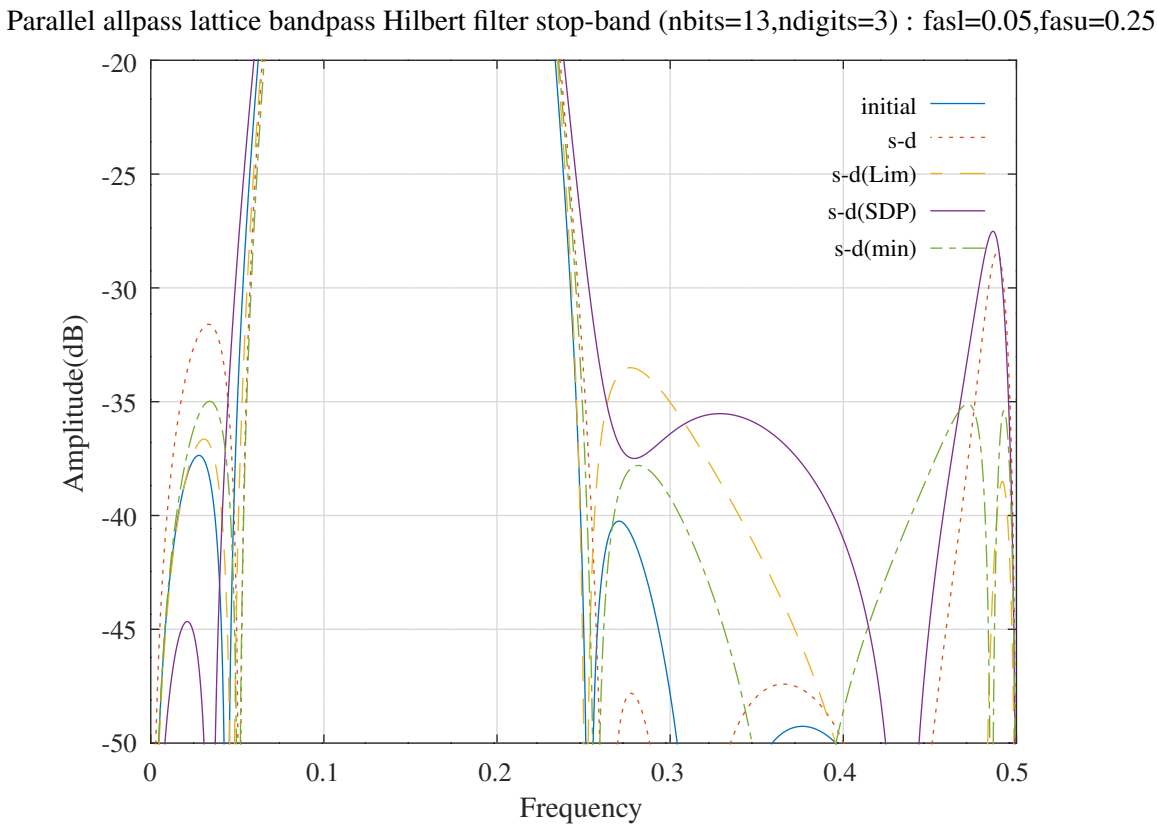



Figure 16.15: Comparison of the stop-band amplitude responses for a parallel all-pass Schur one-multiplier lattice bandpass Hilbert filter with 13-bit integer coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search

# Chapter 17

# 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 11 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:

$$A(f) = \begin{cases} 1 & f <= 0.125 \\ 0 & f > 0.150 \end{cases}$$

The response errors are weighted as follows:

$$W_a(f) = \begin{cases} 1 & f <= 0.125 \\ 0.1 & 0.125 < f < 0.150 \\ 10 & f >= 0.150 \end{cases}$$

The prototype filter is a 5-th order elliptic filter with pass-band edge at $fpass = 0.125$, $1dB$ 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.

## 17.1 Searching with the bit-flipping algorithm

The Octave script *bitflip_schurNSlattice_lowpass_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 17.1 and 17.2 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_schurOneMlattice_lowpass_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 17.3 and 17.4 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 M.2 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_schurNSPAlattice_lowpass_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 17.5 and 17.6 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_schurOneMPAlattice_lowpass_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 17.7 and 17.8 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_lowpass_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 4). The script optimises the truncated coefficients with the bit-flipping algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 17.9 and 17.10 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 17.1 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 | $\infty$ |
| Signed-digit with bit-flipping | 0.8744 | 2.1571 | 1.3824 | 3.1746 | 1.1334 |

Table 17.1: 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 ]';
```

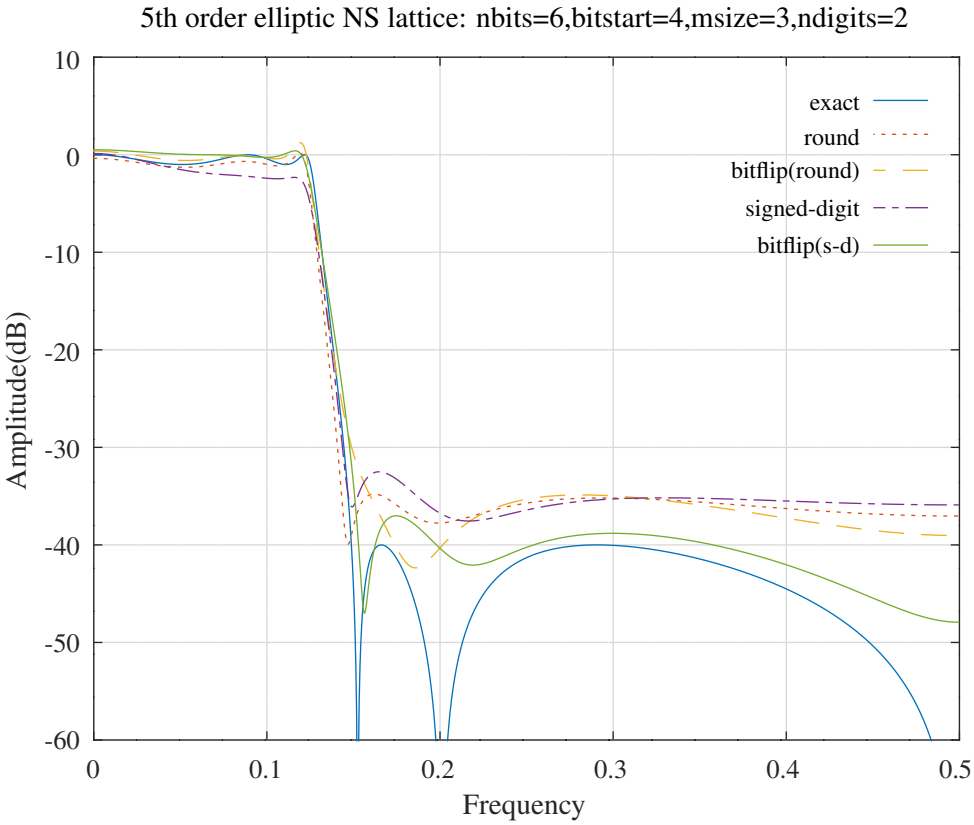



Figure 17.1: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

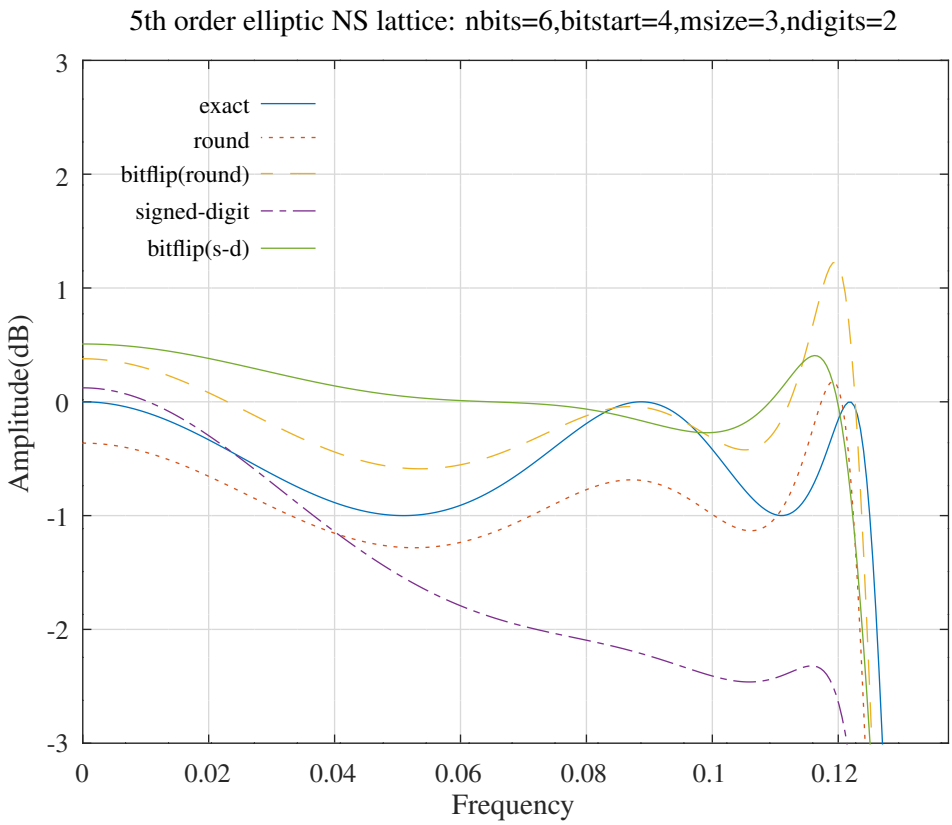



Figure 17.2: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

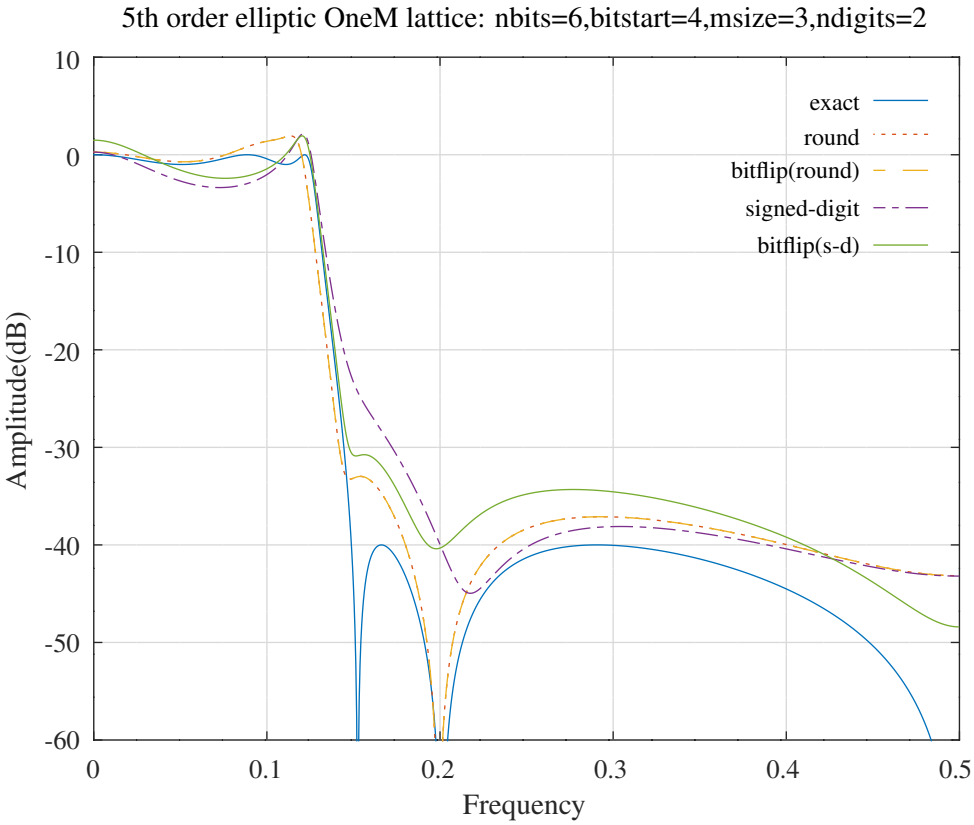



Figure 17.3: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

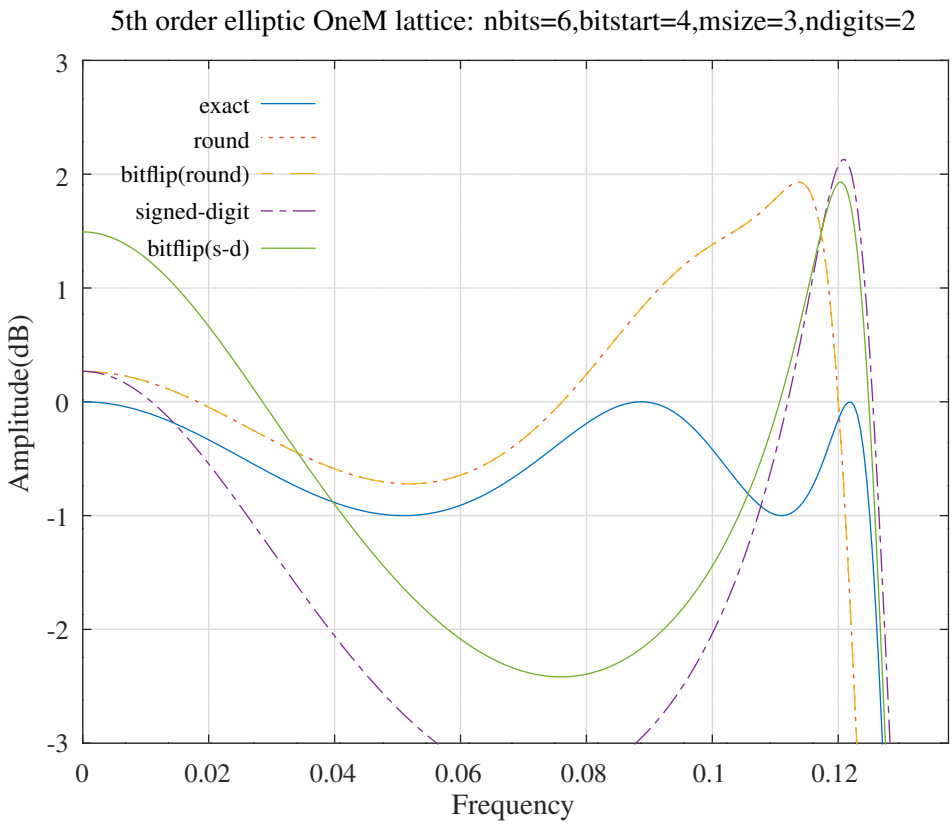



Figure 17.4: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

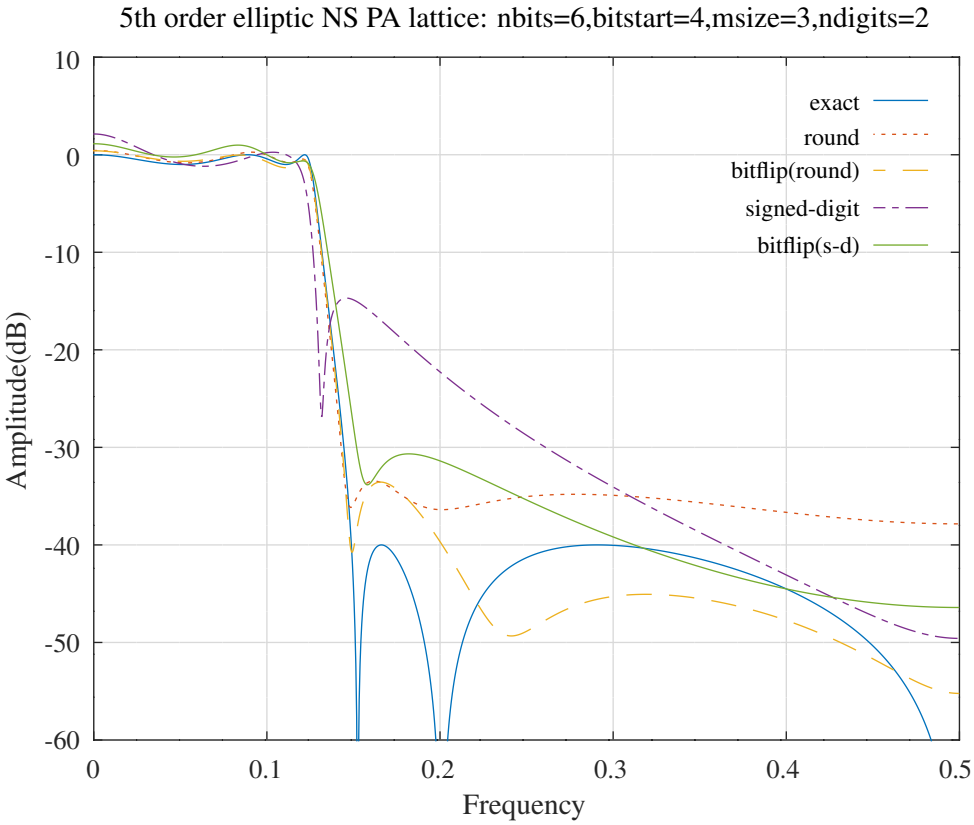



Figure 17.5: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

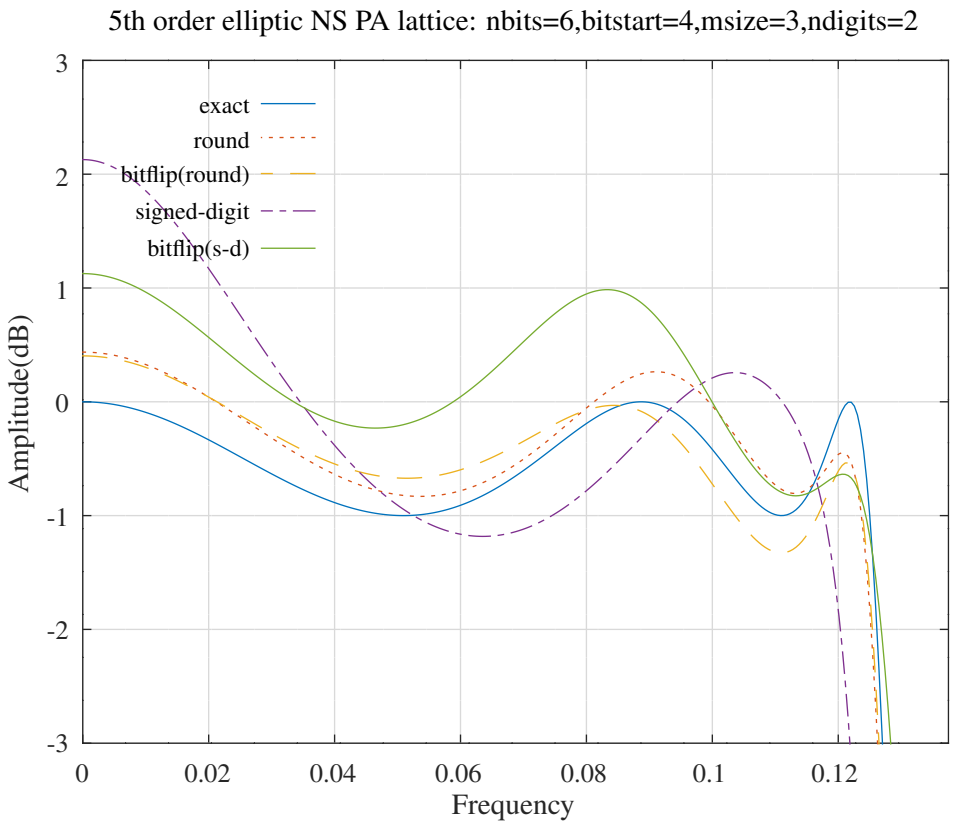



Figure 17.6: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

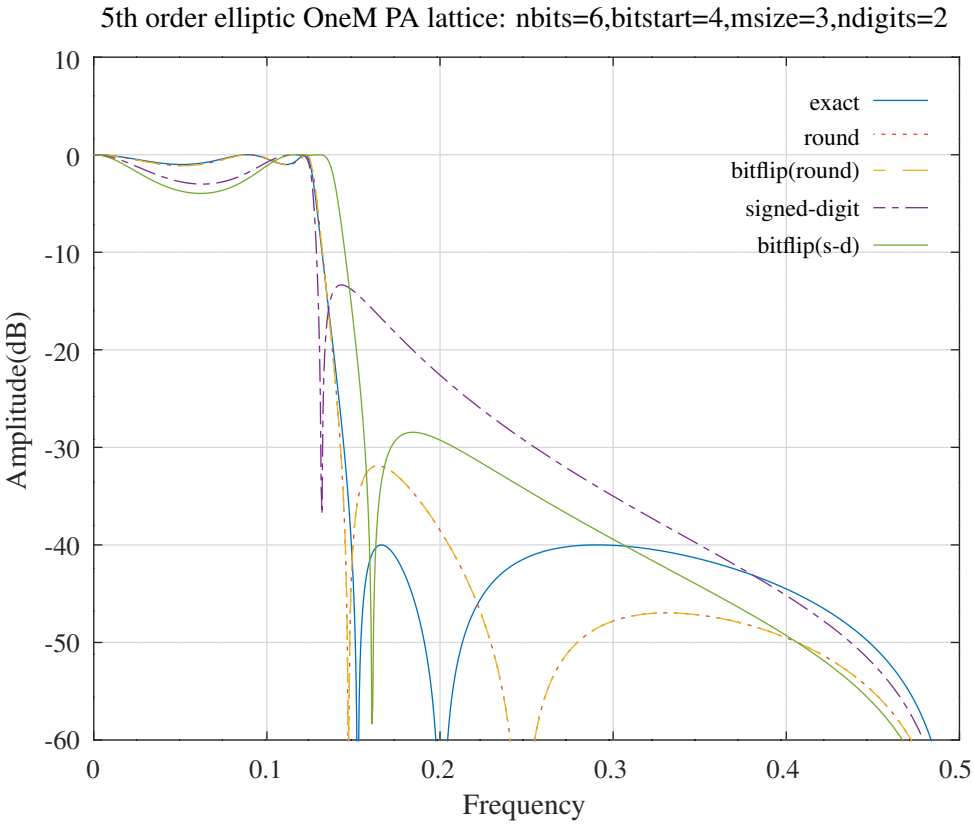



Figure 17.7: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

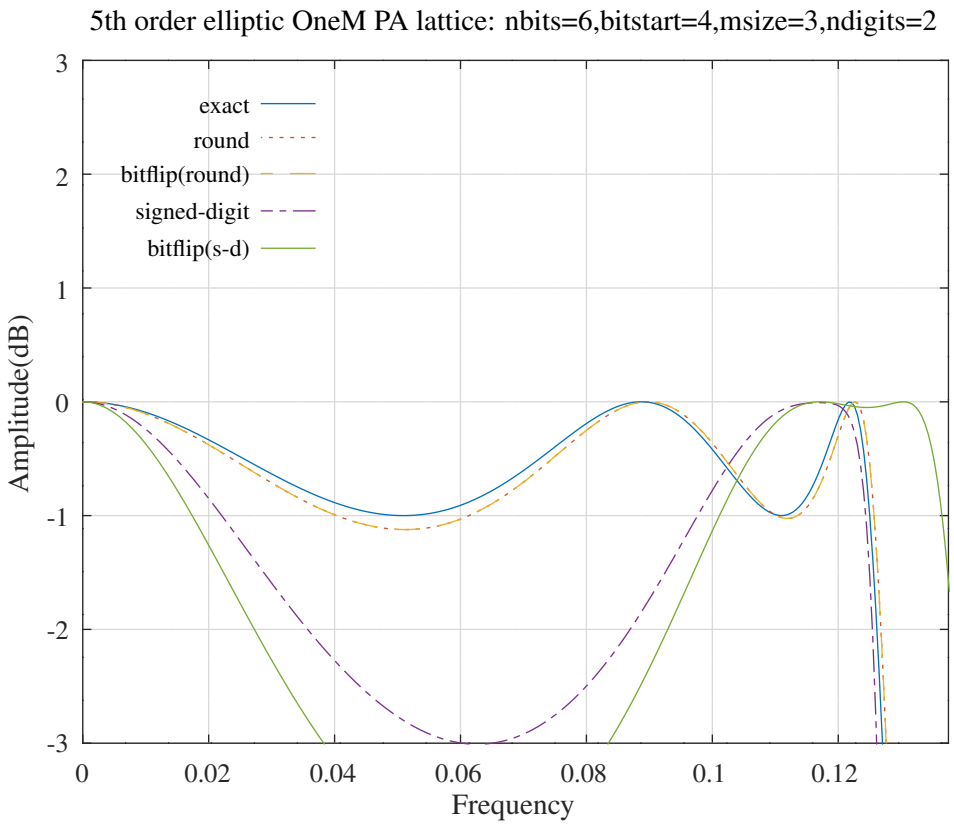



Figure 17.8: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

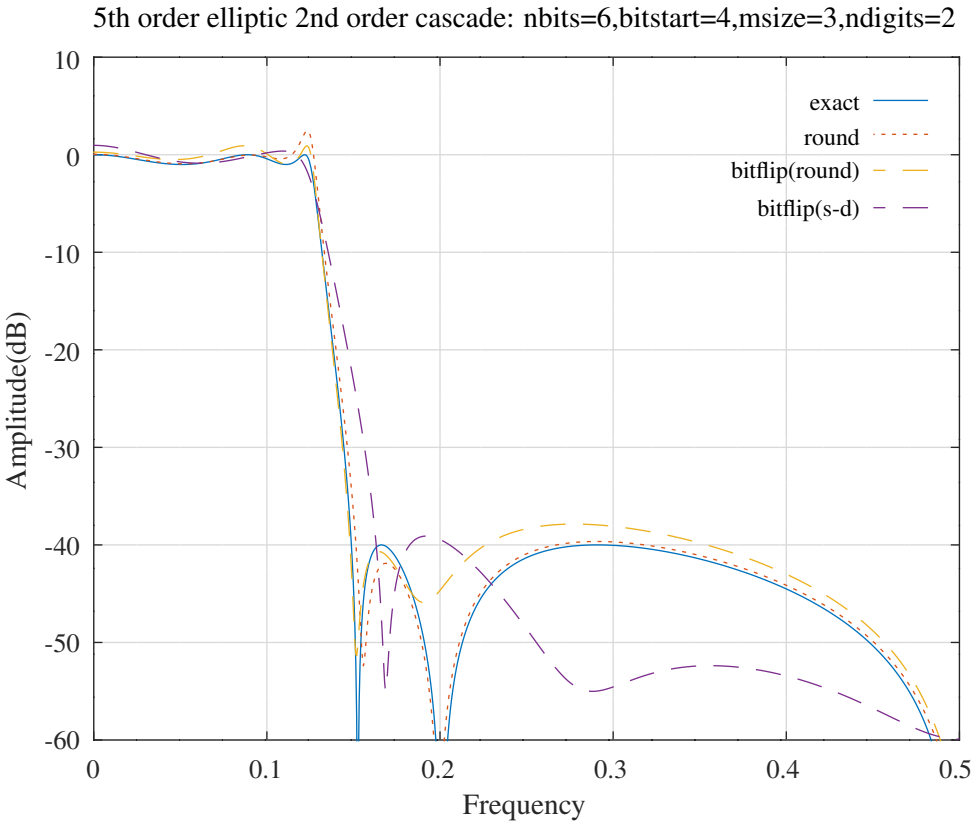



Figure 17.9: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

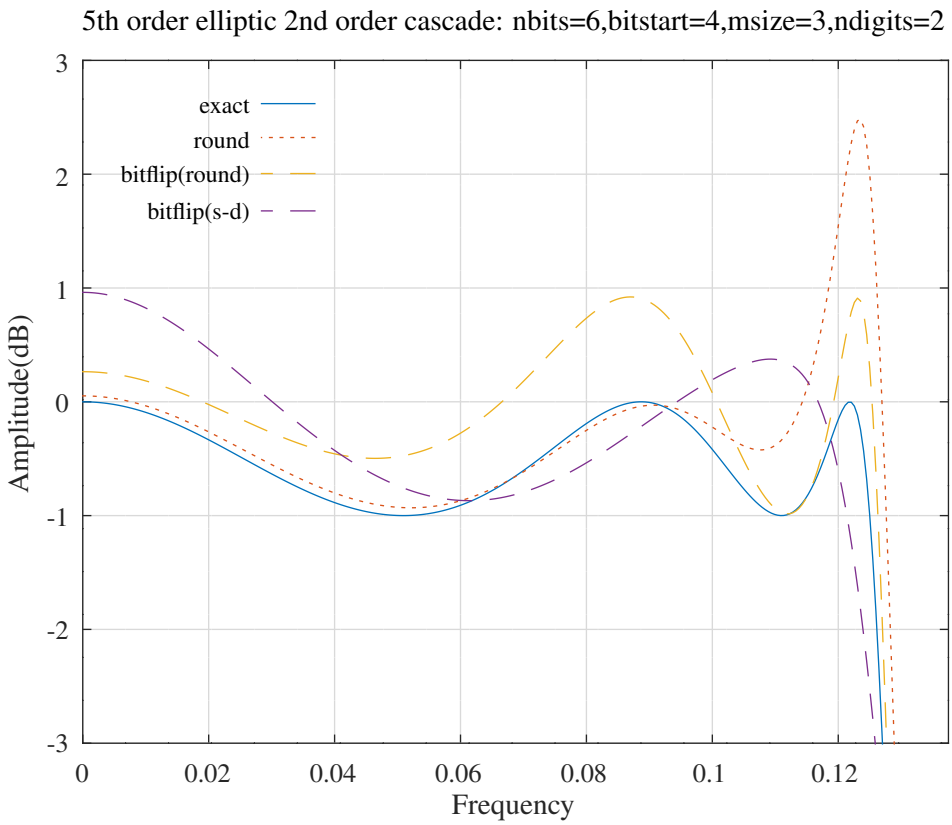



Figure 17.10: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flipping algorithm and 6 bit 2 signed-digit coefficients optimised with the bit-flipping algorithm

## 17.2 Searching with the *Nelder-Mead* simplex algorithm

The Octave script *simplex_schurNSlattice_lowpass_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 [137]. Figures 17.11 and 17.12 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_schurOneMlattice_lowpass_test.m* implements the 5th order elliptic filter as a one-multiplier lattice and optimises the truncated coefficients with the simplex algorithm. Figures 17.13 and 17.14 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_schurNSPAlattice_lowpass_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 17.15 and 17.16 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_schurOneMPAlattice_lowpass_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 17.17 and 17.18 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_svcasc_lowpass_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 4). The script optimises the truncated coefficients with the simplex algorithm. Figures 17.19 and 17.20 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 17.2 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 | $\infty$ |
| Signed-digit with simplex | 1.6581 | 2.3918 | 3.2559 | 3.1746 | 3.2134 |

Table 17.2: 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

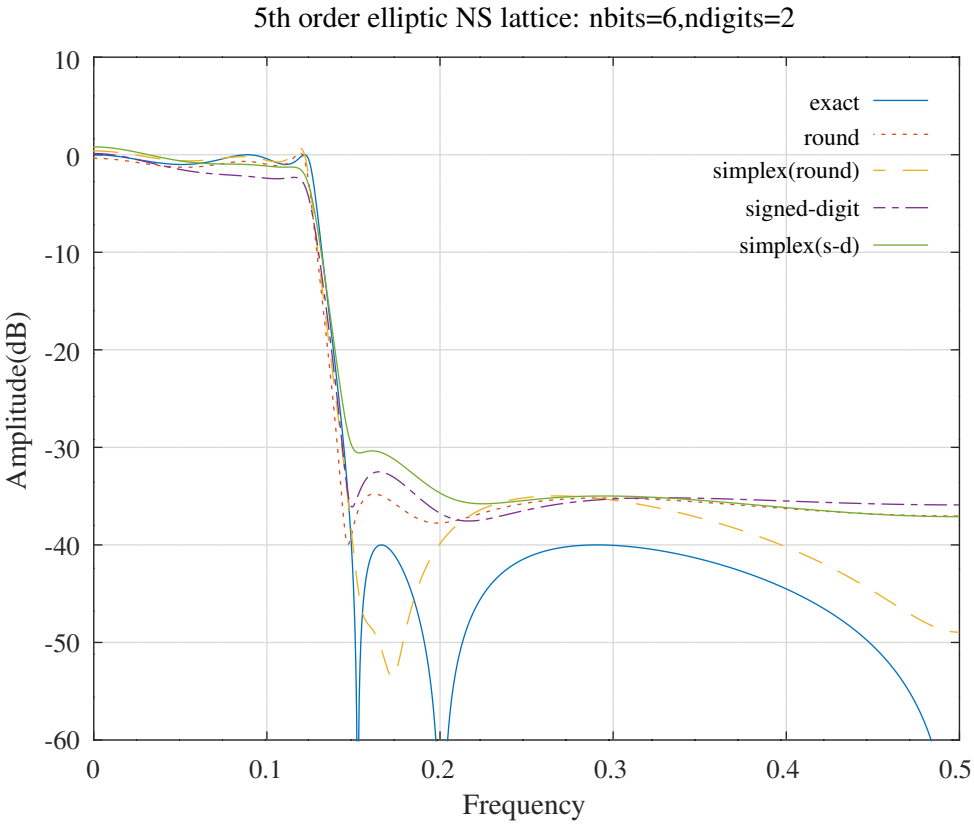



Figure 17.11: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

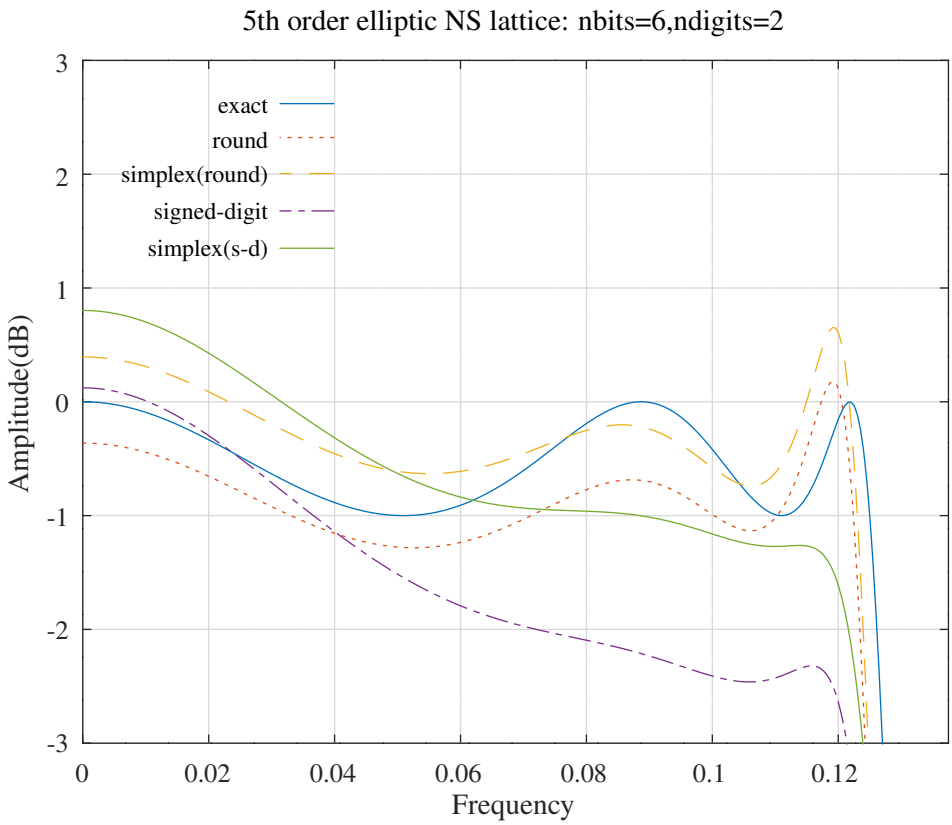



Figure 17.12: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

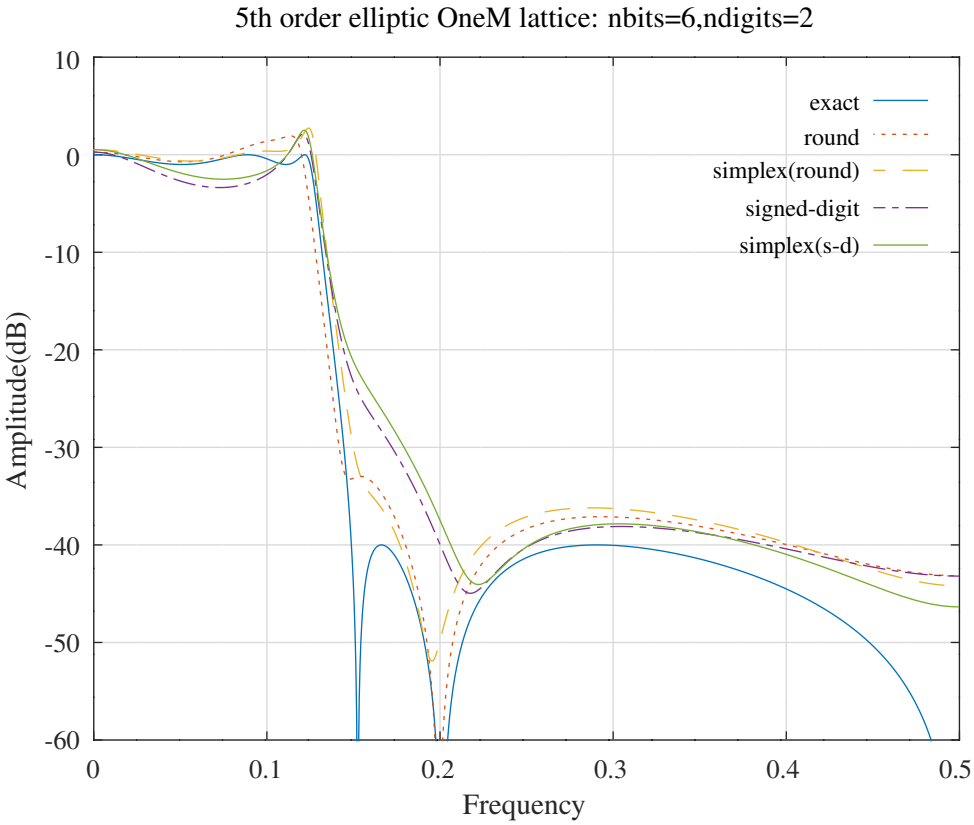



Figure 17.13: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

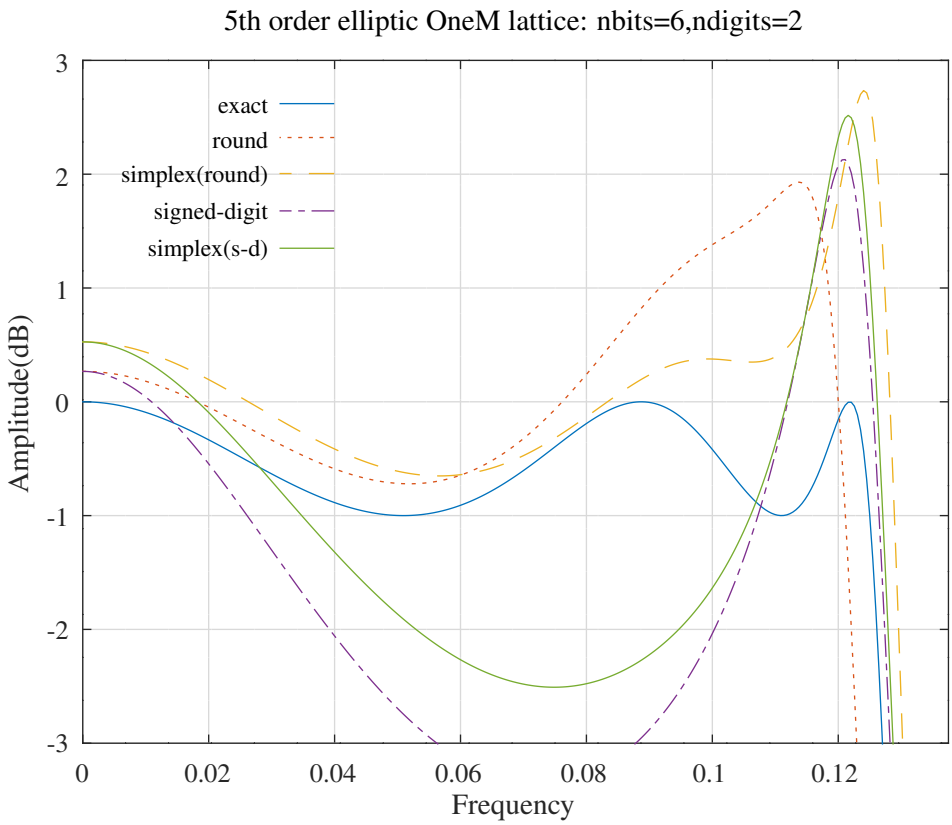



Figure 17.14: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

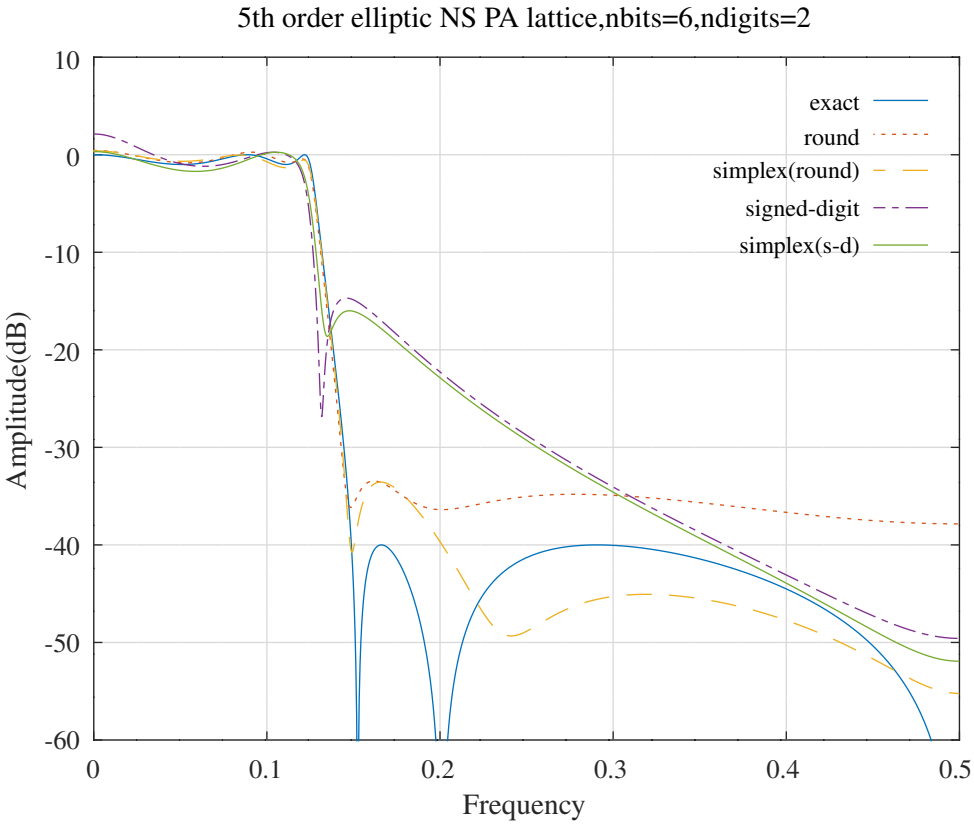



Figure 17.15: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

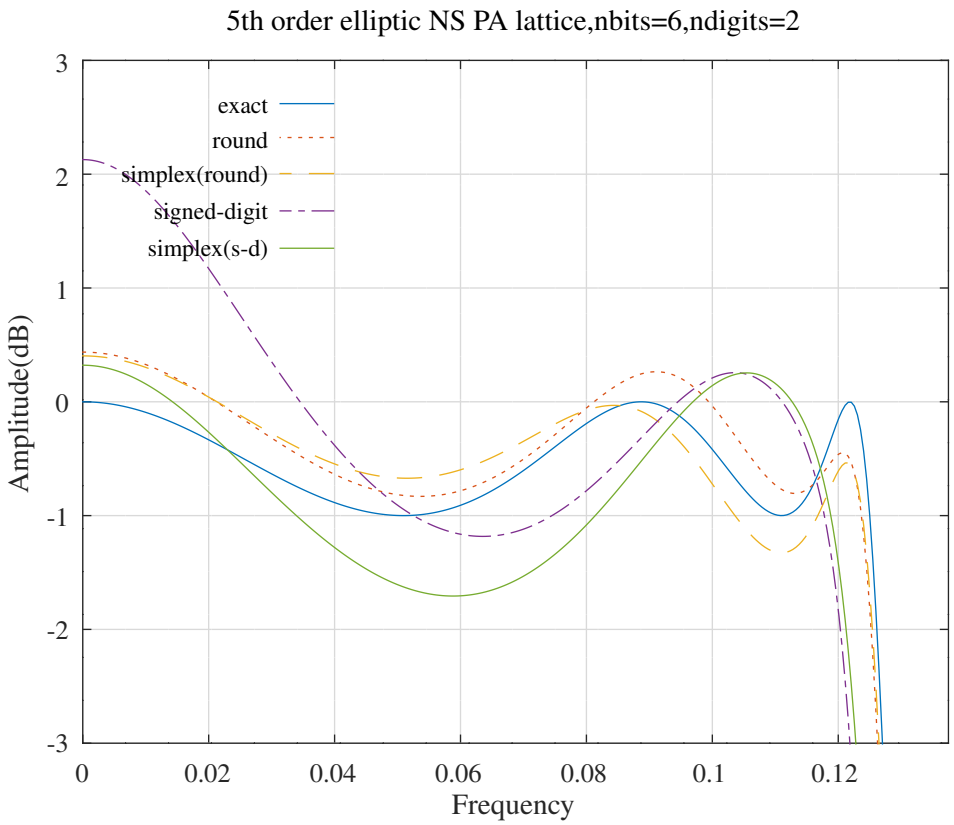



Figure 17.16: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

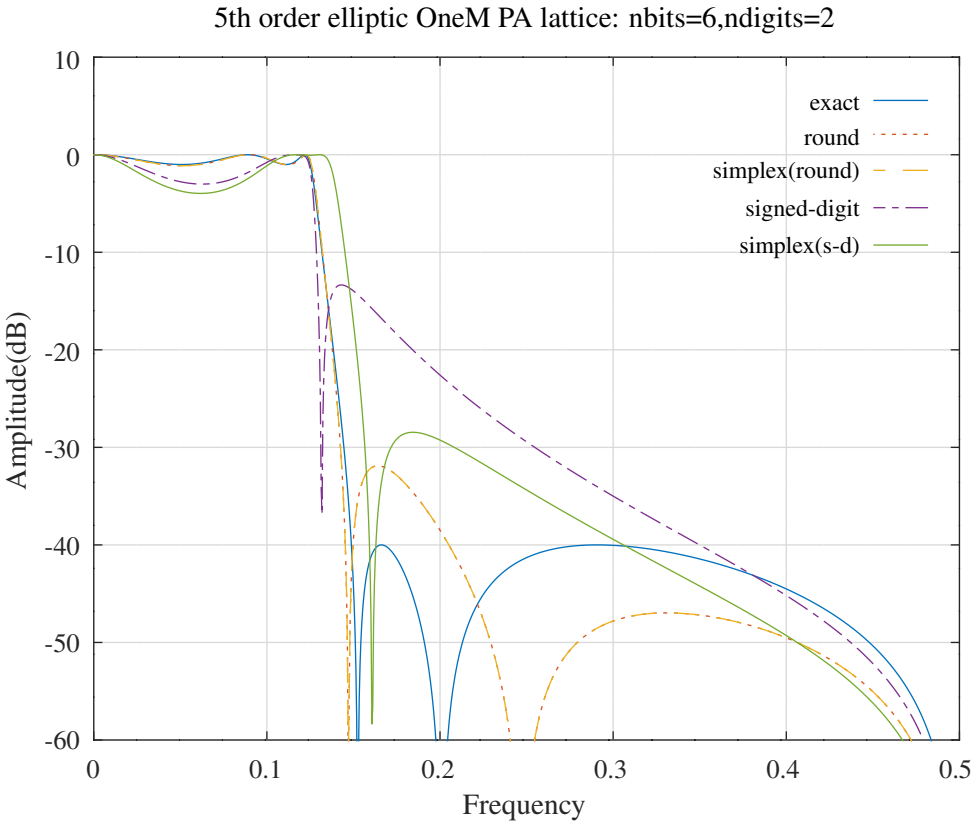



Figure 17.17: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

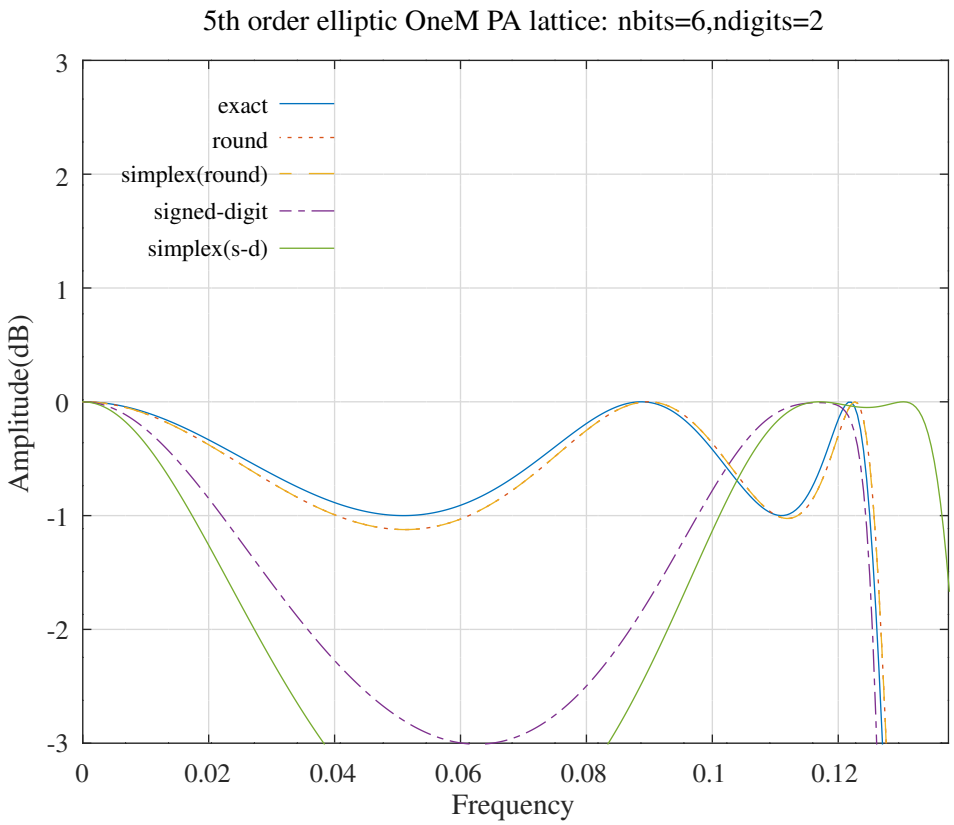



Figure 17.18: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

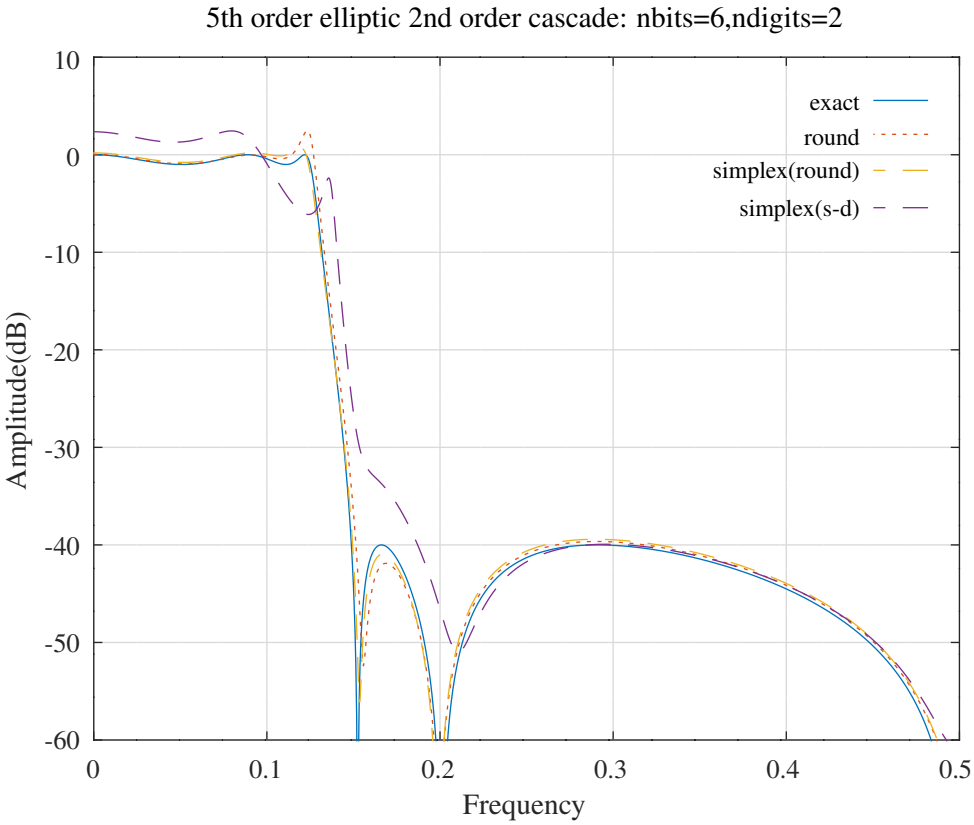



Figure 17.19: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

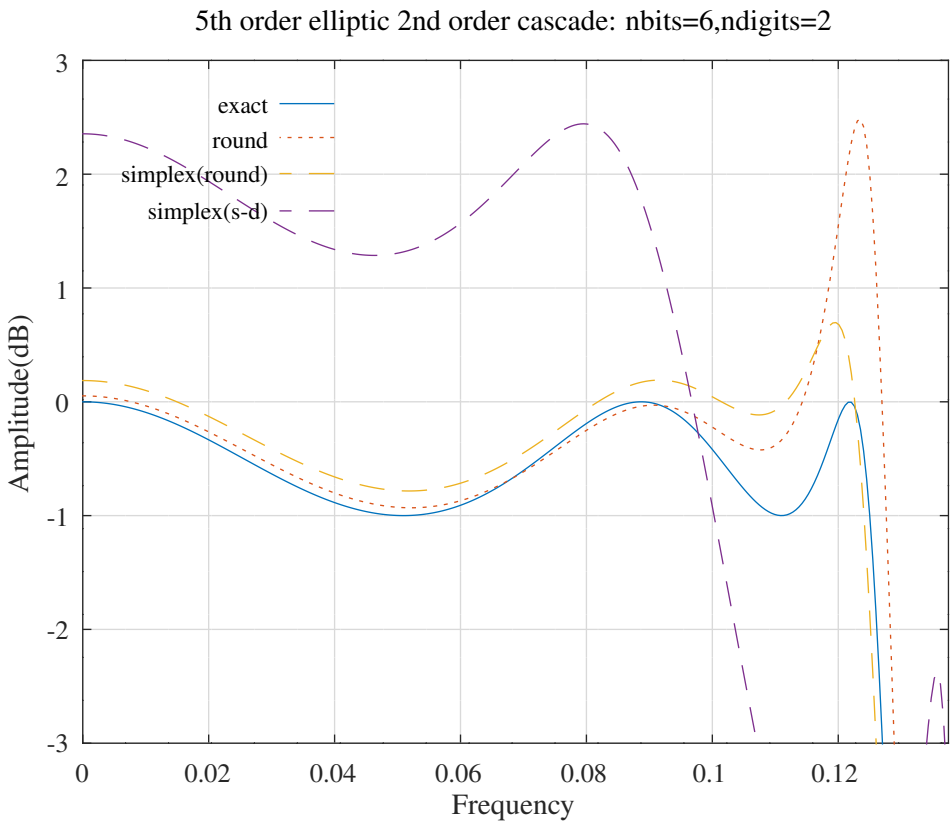



Figure 17.20: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simplex algorithm and 6 bit 2 signed-digit coefficients optimised with the simplex algorithm

## 17.3 Searching with the *simulated annealing* algorithm

This section shows the results of searching for the coefficients of a 5-th order low-pass filter with the Octave-Forge *optim* package [137] implementation of the simulated annealing algorithm, *nonlin_min* [138, Demonstartion 2][a]. Unfortunately, the minimum cost found by *nonlin_min* may vary by a factor of 2 or more from run to run. In this section I show the best results from 20 runs of each test.

The Octave script *samin_schurNSlattice_lowpass_test.m* implements the prototype elliptic filter as a normalised-scaled lattice and optimises the truncated coefficients with the *nonlin_min* simulated annealing algorithm from the Octave-Forge *optim* package [137]. Figures 17.21 and 17.22 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_schurOneMlattice_lowpass_test.m* implements the prototype elliptic filter as a one-multiplier lattice and optimises the truncated coefficients with the simulated annealing algorithm. Figures 17.23 and 17.24 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_schurNSPAlattice_lowpass_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 17.25 and 17.26 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_schurOneMPAlattice_lowpass_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 17.27 and 17.28 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_svcasc_lowpass_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 4). The script optimises the truncated coefficients with the simulated annealing algorithm. Figures 17.29 and 17.30 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 17.3 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 | 0.8420 | 1.4617 | 0.8295 | 0.8803 | 0.6781 |
| Signed-digit | 2.3373 | 2.7157 | 3.6059 | 3.9013 | $\infty$ |
| Signed-digit with samin | 0.8309 | 1.7673 | 1.0841 | 1.6706 | $\infty$ |

Table 17.3: 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:

[a] The *samin* function has been deprecated.

```
s10_sasd = [  0.968750,  0.750000,  0.375000,  0.156250,  0.031250 ]';
s11_sasd = [  0.156250,  0.750000,  0.875000,  1.000000,  0.500000 ]';
s20_sasd = [ -0.750000,  0.968750, -0.937500,  0.750000, -0.312500 ]';
s00_sasd = [  0.625000,  0.281250,  0.468750,  0.562500,  0.937500 ]';
s02_sasd = [  0.750000, -1.000000,  0.875000, -0.750000,  0.468750 ]';
s22_sasd = [  0.625000,  0.250000,  0.531250,  0.500000,  0.875000 ]';
```

The lowest cost 6 bit rounded coefficients of the parallel-allpass one-multiplier lattice filter found with the simulated annealing algorithm are[b].

```
A1k_sa = [ -0.812500,  0.625000 ];
A2k_sa = [ -0.781250,  0.906250, -0.593750 ];
```

The lowest cost 6 bit rounded coefficients of the cascade of 2nd order sections found with the simulated annealing algorithm are:

```
a11_sa = [  0.750000,  0.687500,  0.000000 ];
a12_sa = [ -0.687500, -0.375000,  1.000000 ];
a21_sa = [  0.656250,  0.531250,  0.000000 ];
a22_sa = [  0.625000,  0.687500,  0.687500 ];
b1_sa  = [  0.406250,  0.656250,  0.000000 ];
b2_sa  = [  0.125000,  0.062500,  1.000000 ];
c1_sa  = [  0.375000,  0.062500,  0.000000 ];
c2_sa  = [  0.937500,  0.406250,  0.406250 ];
dd_sa  = [  0.250000,  0.281250,  0.218750 ];
```

[b]These coefficient multiplications can be implemented with a total of 14 signed-digits and 9 adders

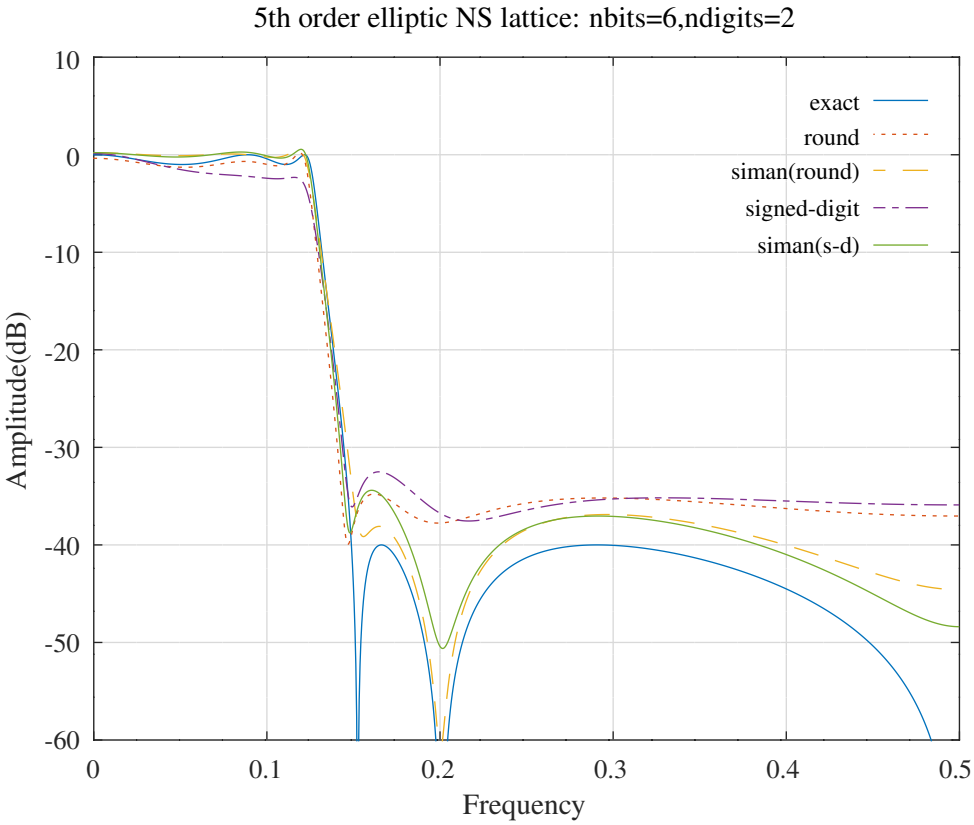



Figure 17.21: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

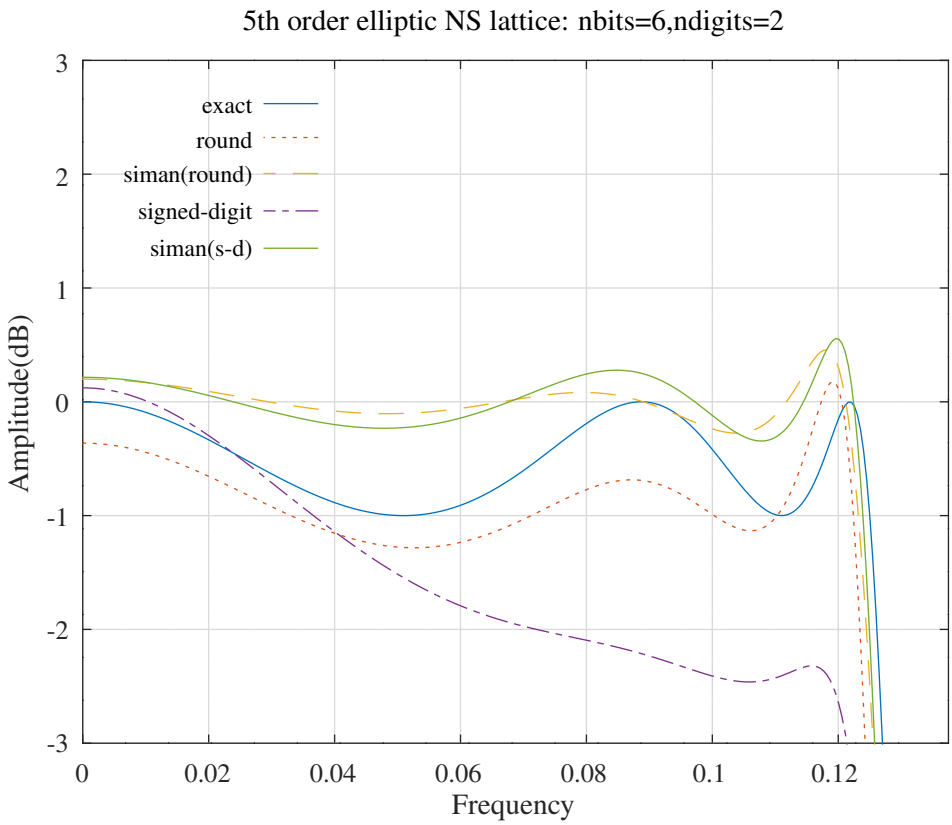



Figure 17.22: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

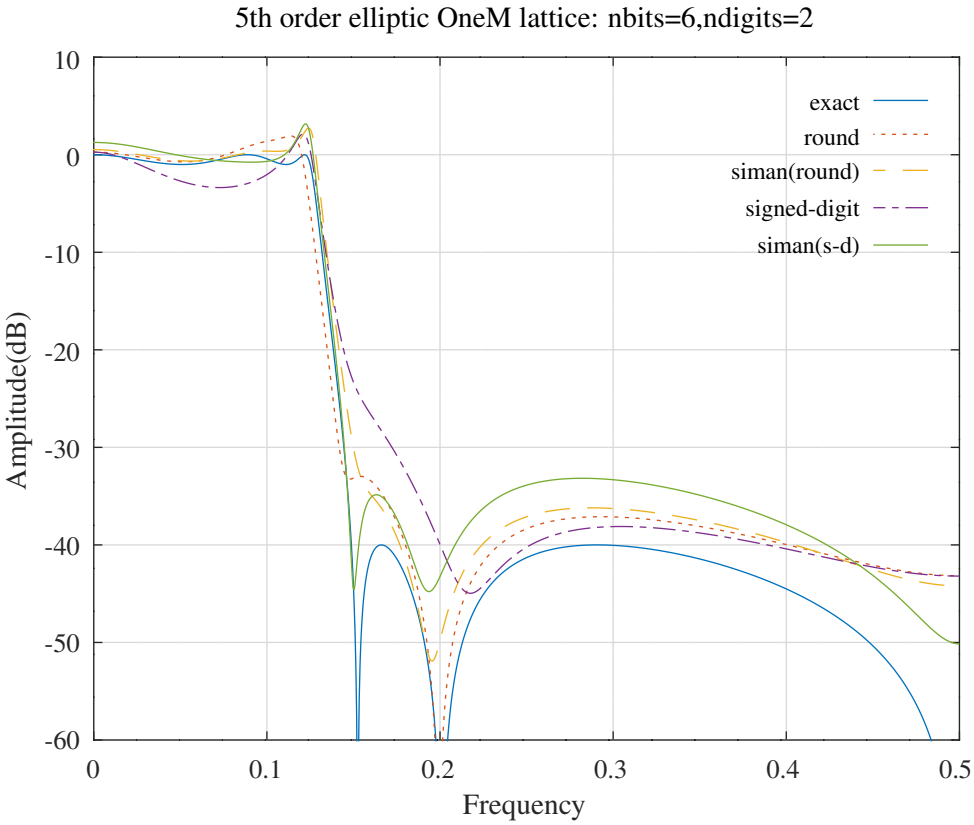



Figure 17.23: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

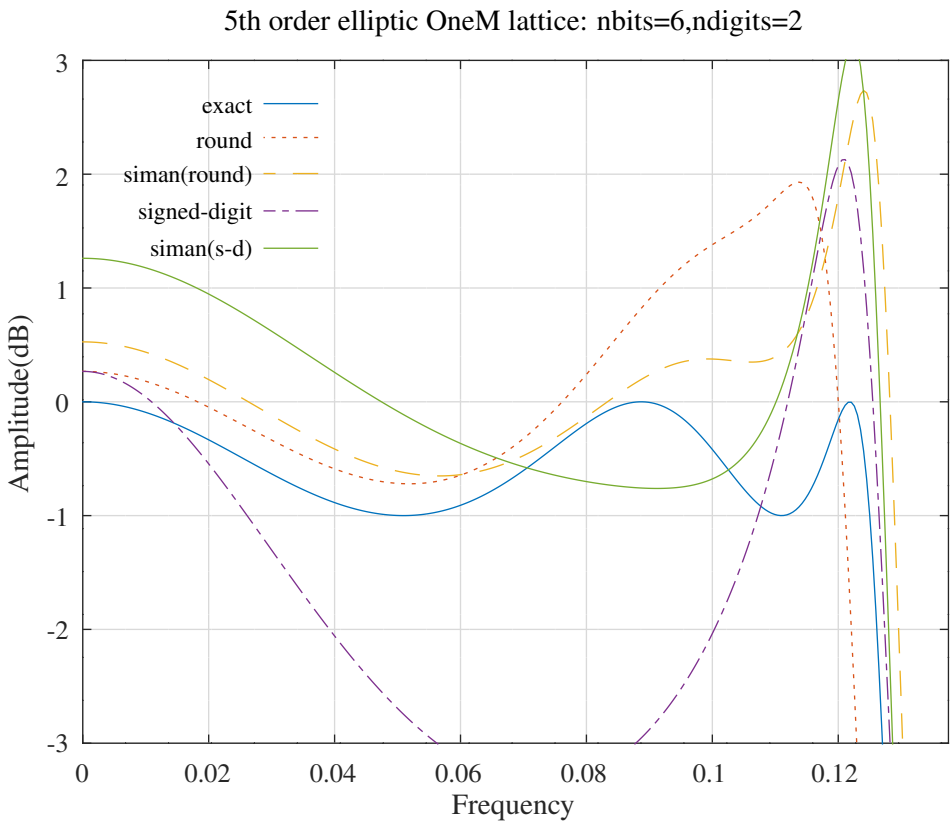



Figure 17.24: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

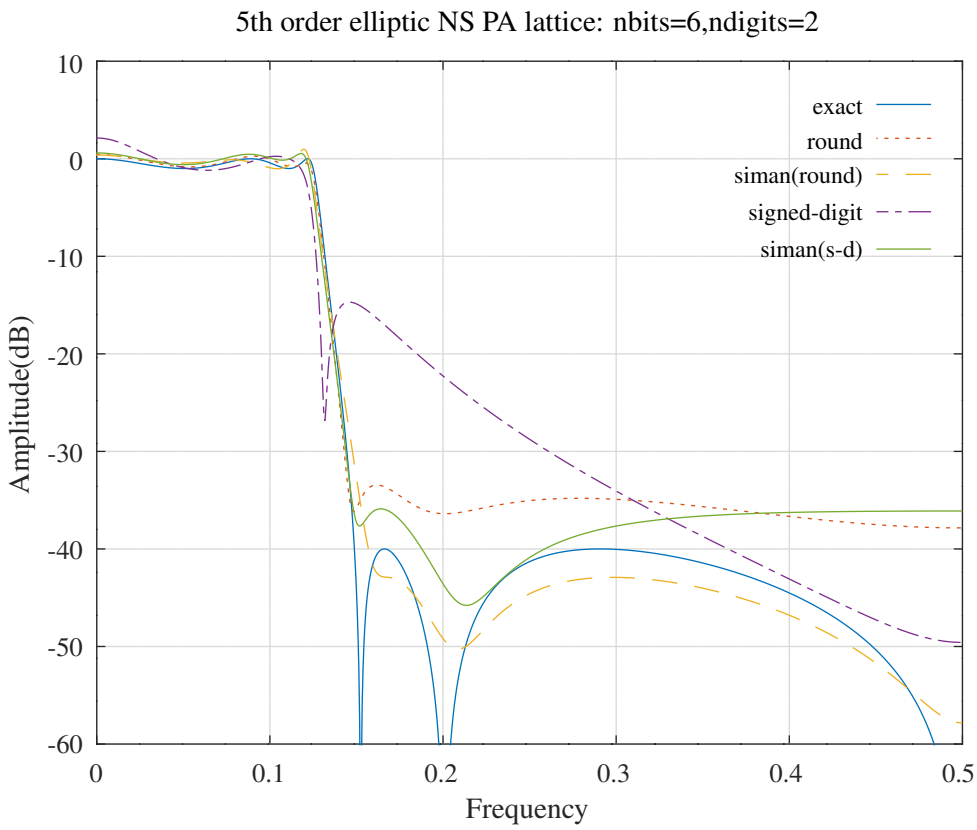



Figure 17.25: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

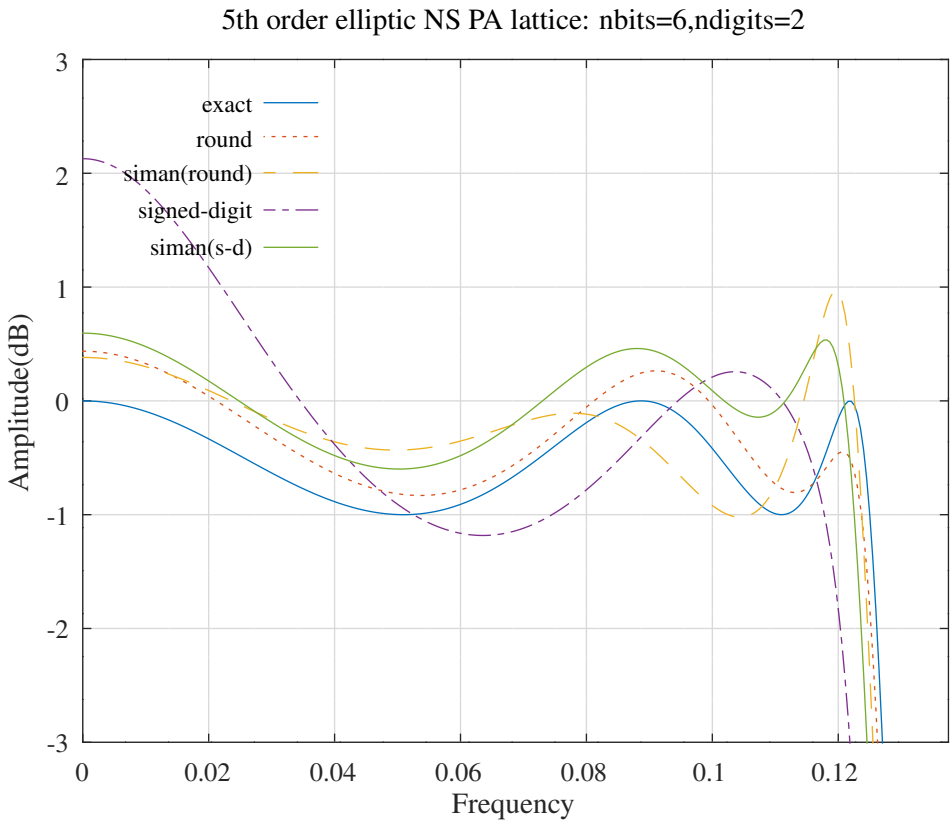



Figure 17.26: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

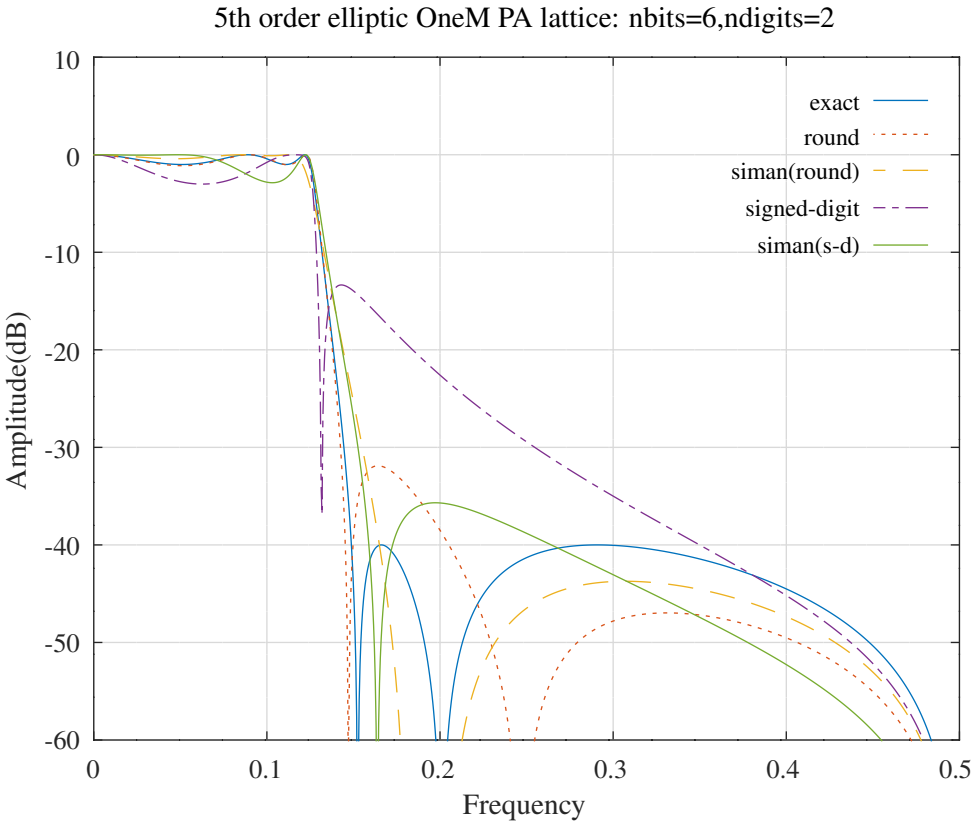



Figure 17.27: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

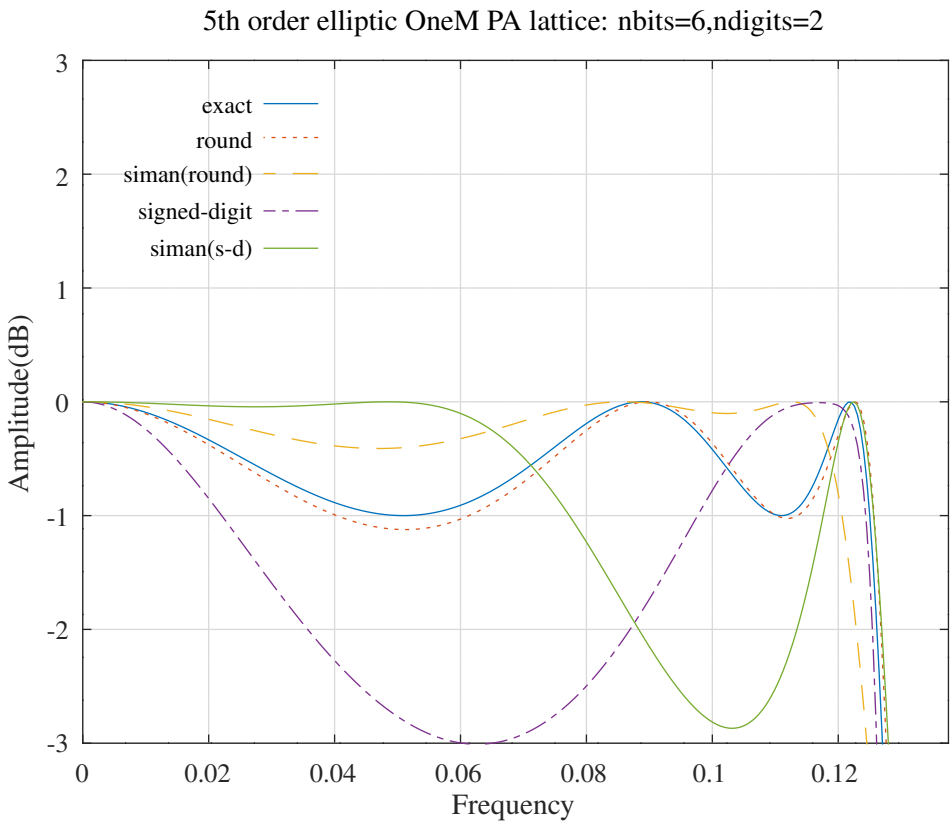



Figure 17.28: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

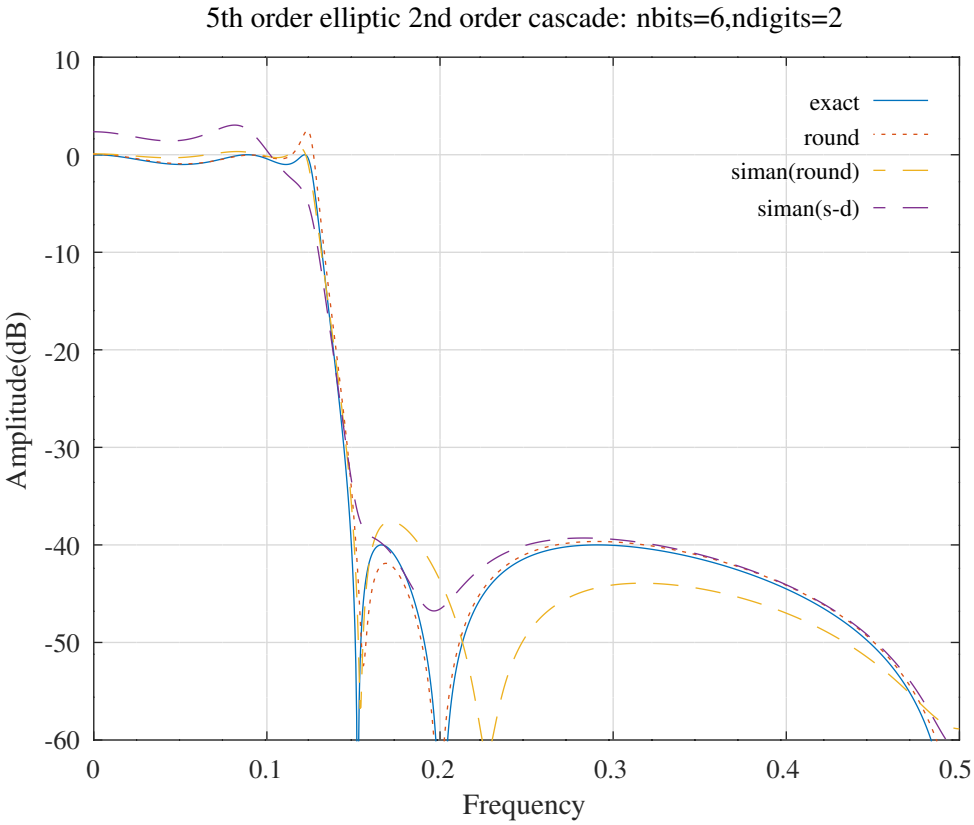



Figure 17.29: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

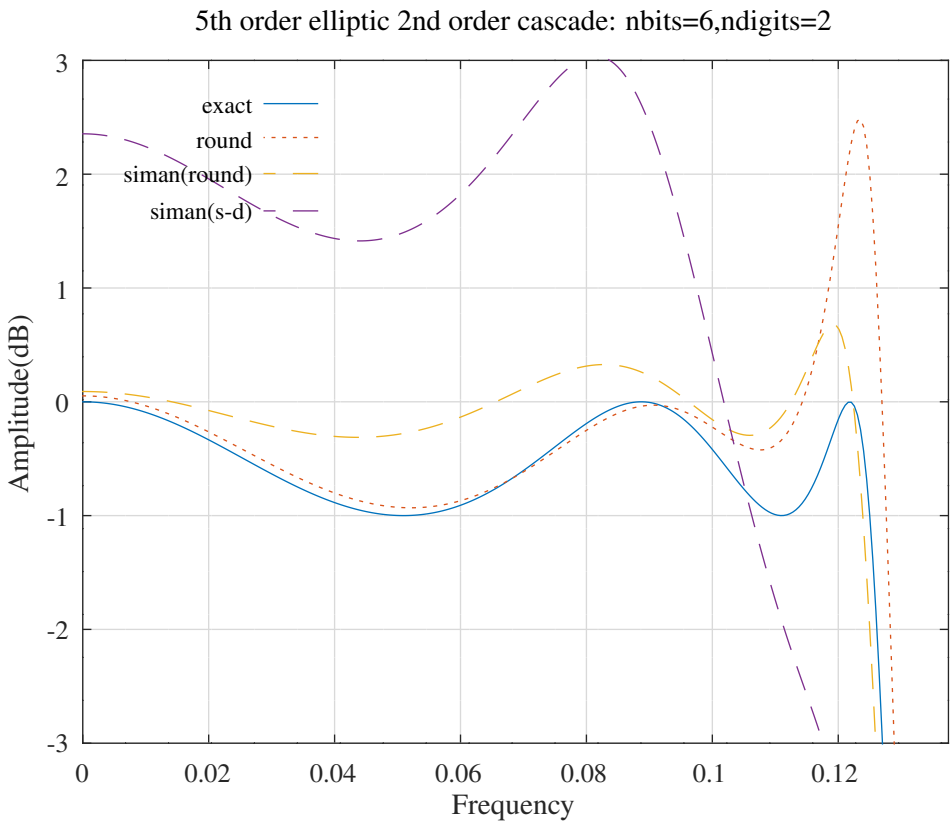



Figure 17.30: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the simulated annealing algorithm and 6 bit 2 signed-digit coefficients optimised with the simulated annealing algorithm

## 17.4 Searching with the *differential evolution* algorithm

This section shows the results of searching for the coefficients of a 5-th order low-pass filter with the Octave-Forge *optim* package [137] implementation of the *differential evolution* algorithm of *Storn* and *Price* [175], *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 20 runs of each test. These tests use the default control settings for *de_min*. The *de_min* function includes 12 possible optimisation strategies. The default strategy is *DEGL/SAW/bin*.

The Octave script *de_min_schurNSlattice_lowpass_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 [137]. Figures 17.31 and 17.32 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_schurOneMlattice_lowpass_test.m* implements the prototype elliptic filter as a one-multiplier lattice and optimises the truncated coefficients with the differential evolution algorithm. Figures 17.33 and 17.34 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_schurNSPAlattice_lowpass_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 17.35 and 17.36 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_schurOneMPAlattice_lowpass_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 17.37 and 17.38 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_lowpass_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 4). The script optimises the truncated coefficients with the differential evolution algorithm. Figures 17.39 and 17.40 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 17.4 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 | 0.7797 | 1.2765 | 0.7874 | 0.8803 | 0.6572 |
| Signed-digit | 2.3373 | 2.7157 | 3.6059 | 3.9013 | $\infty$ |
| Signed-digit with de_min | 0.7768 | 1.9147 | 1.1879 | 1.3505 | 0.6995 |

Table 17.4: 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

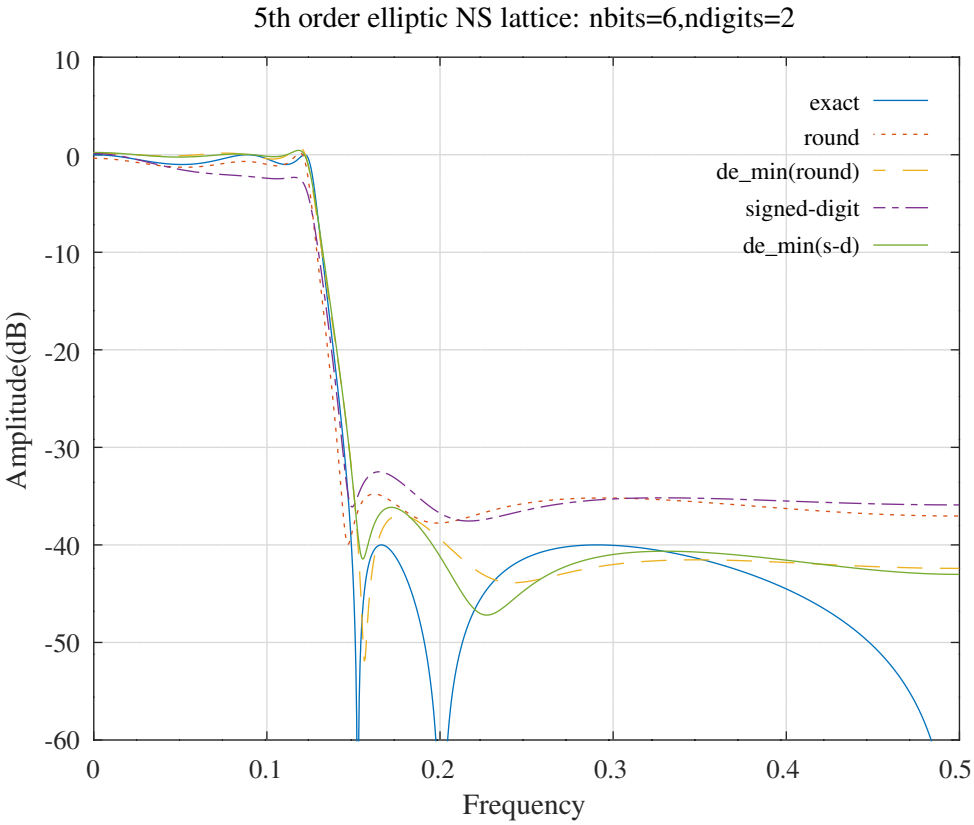



Figure 17.31: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

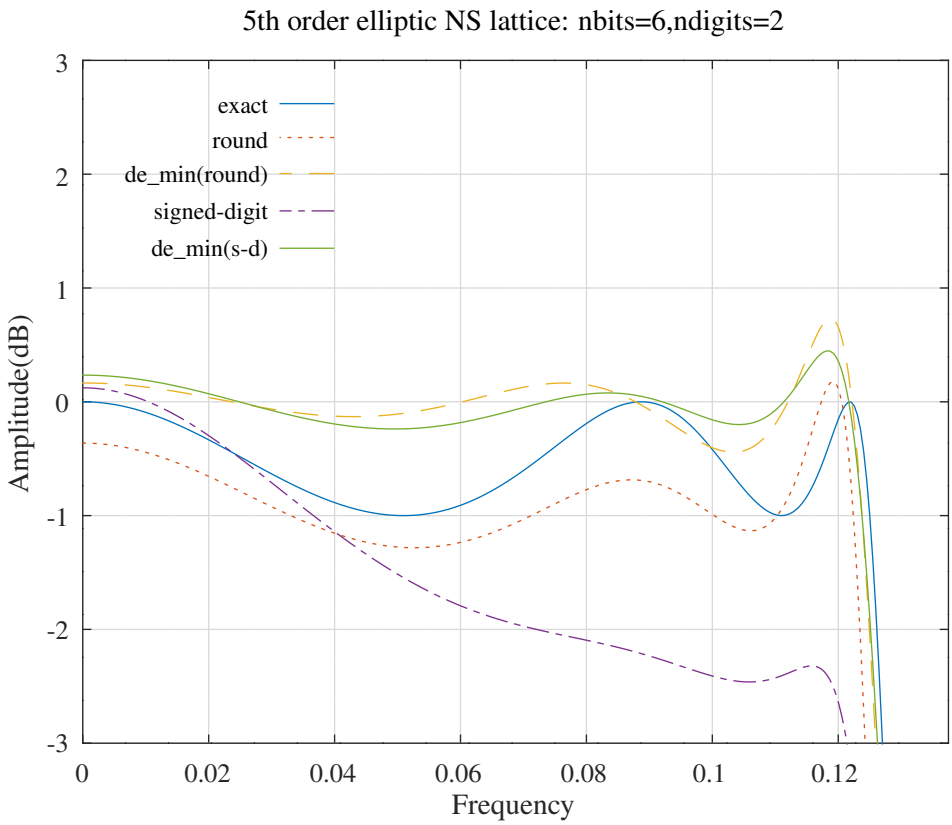



Figure 17.32: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

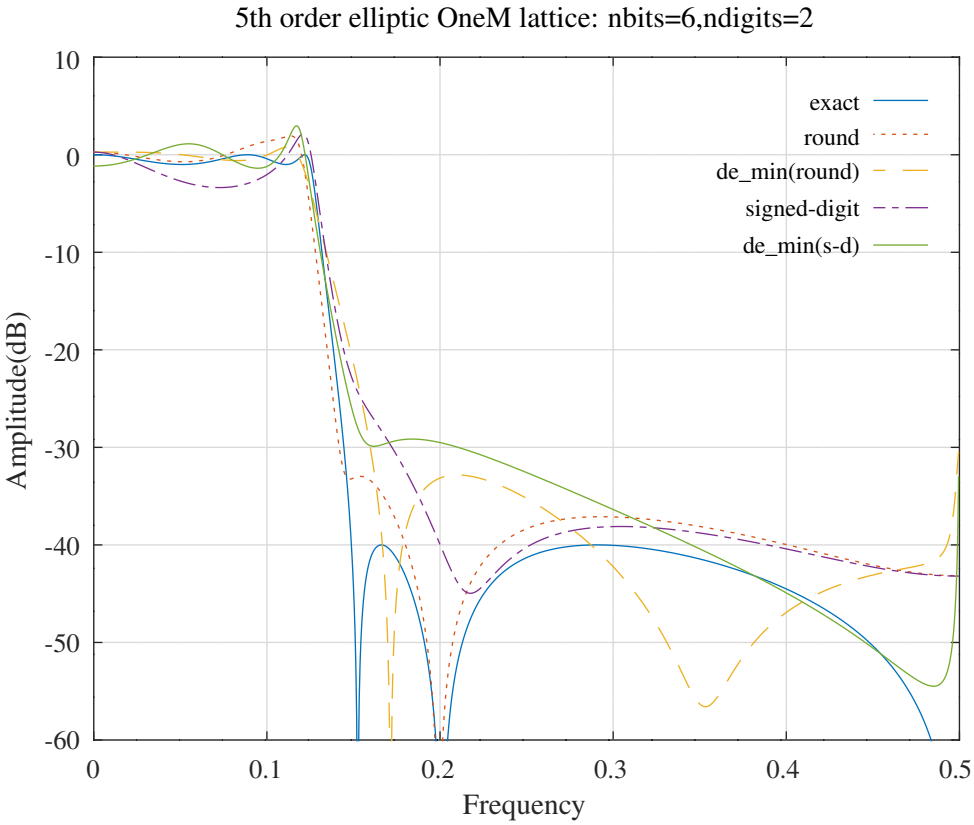



Figure 17.33: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

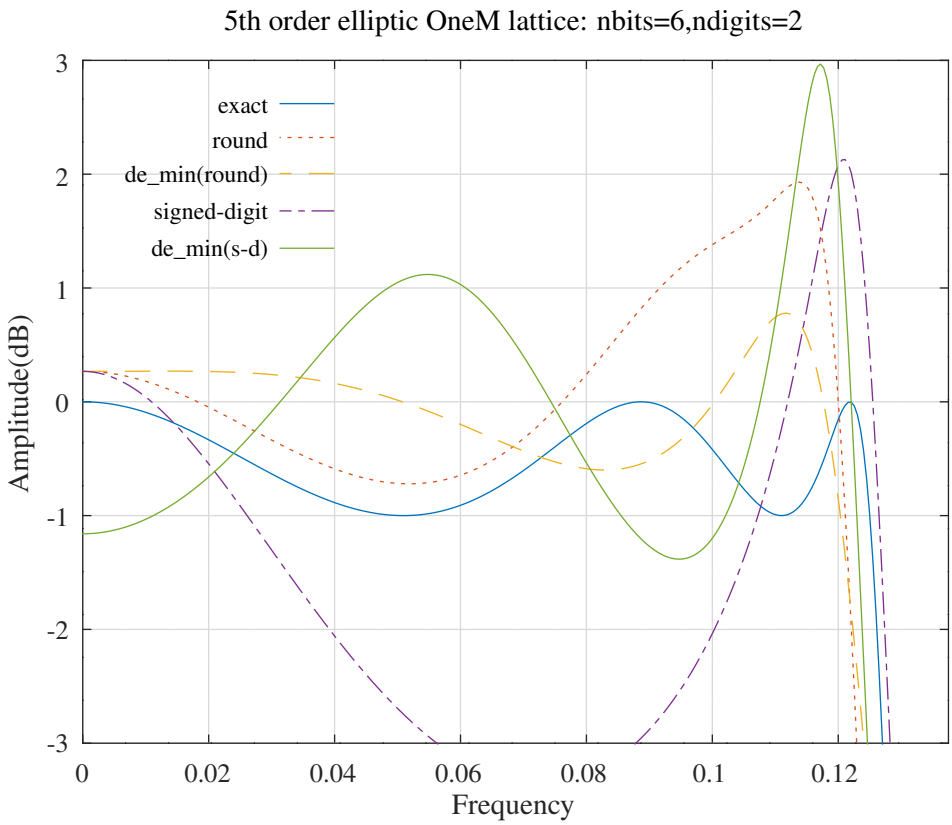



Figure 17.34: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

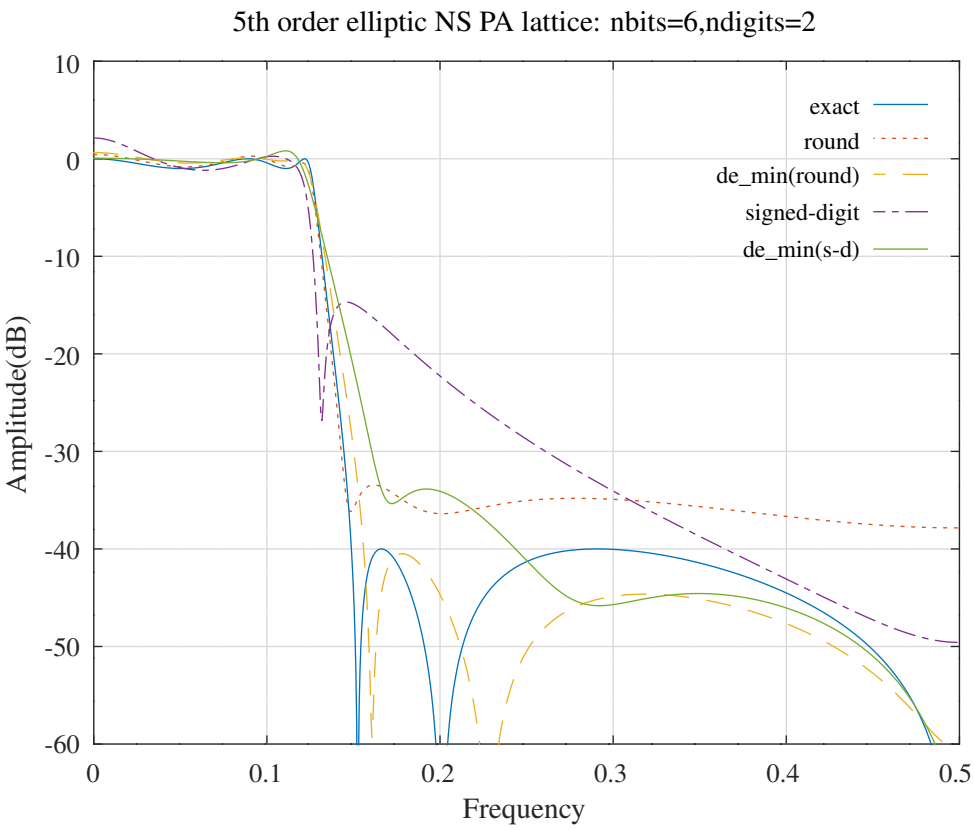



Figure 17.35: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

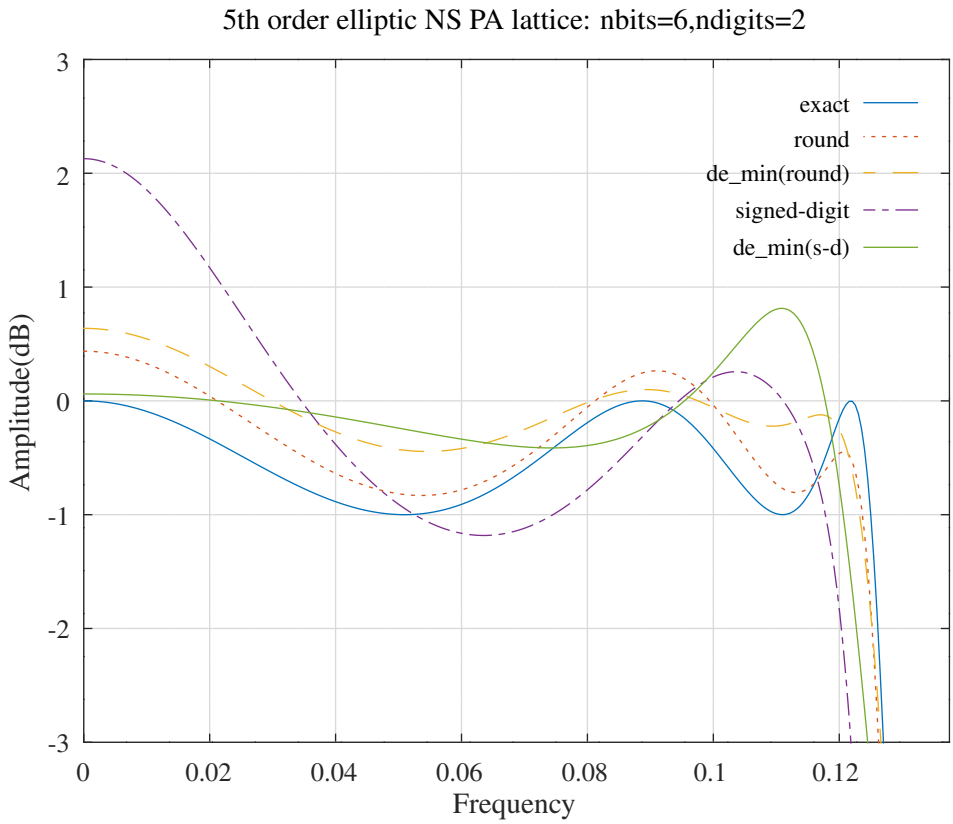



Figure 17.36: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

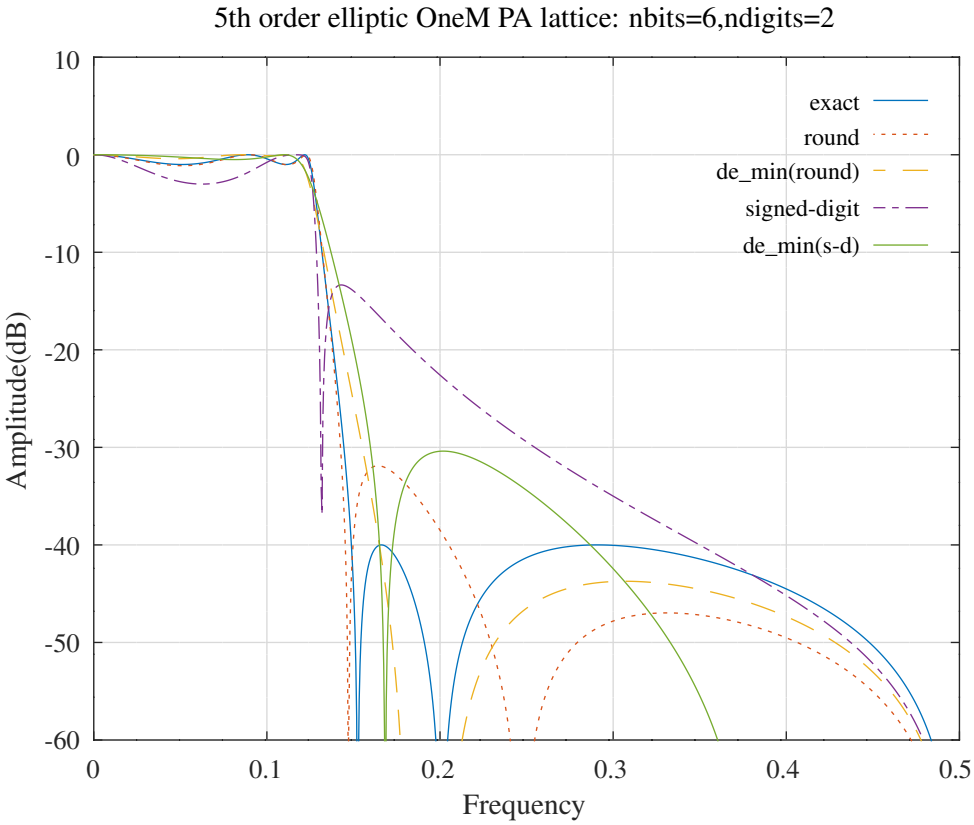



Figure 17.37: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

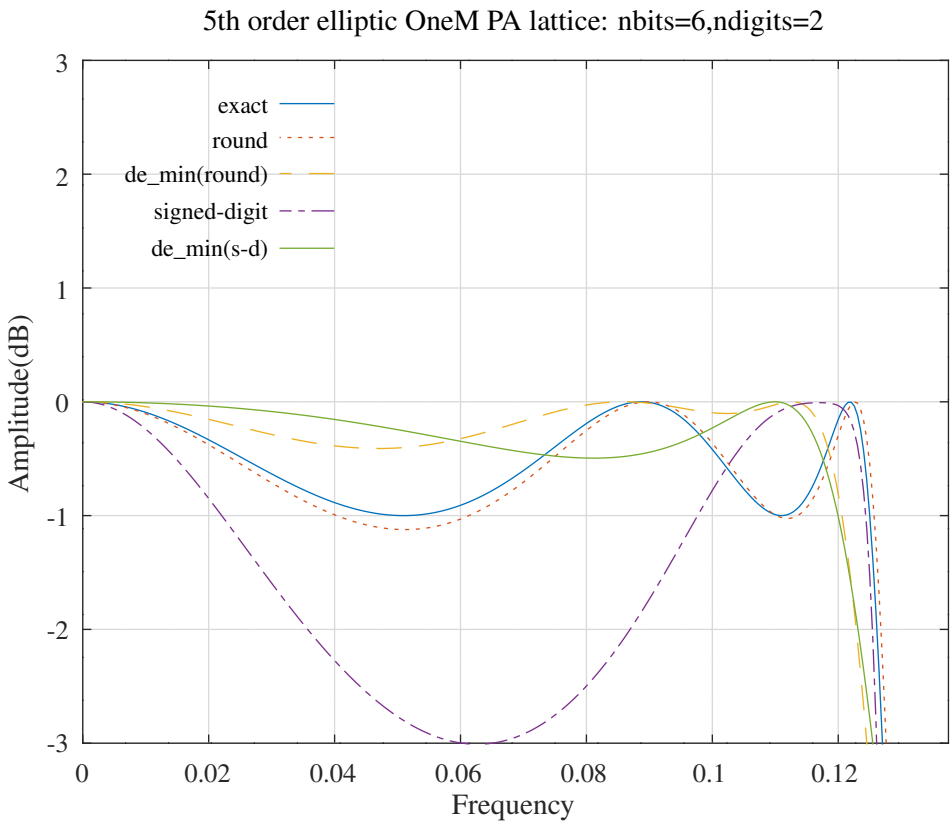



Figure 17.38: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

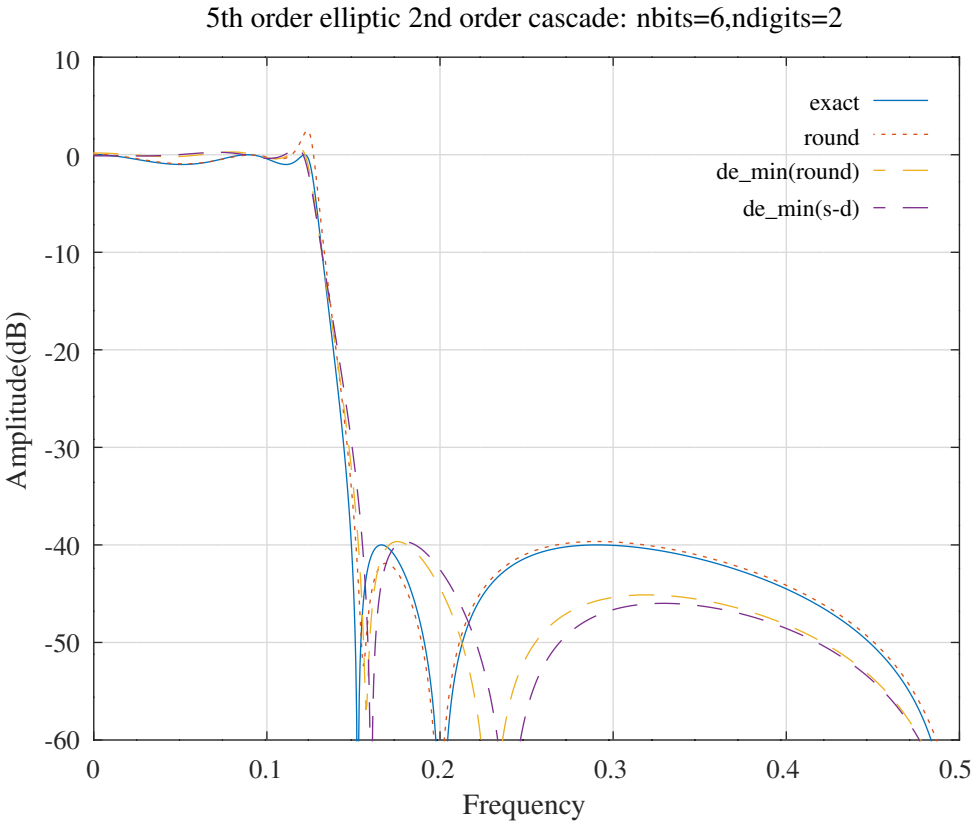



Figure 17.39: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

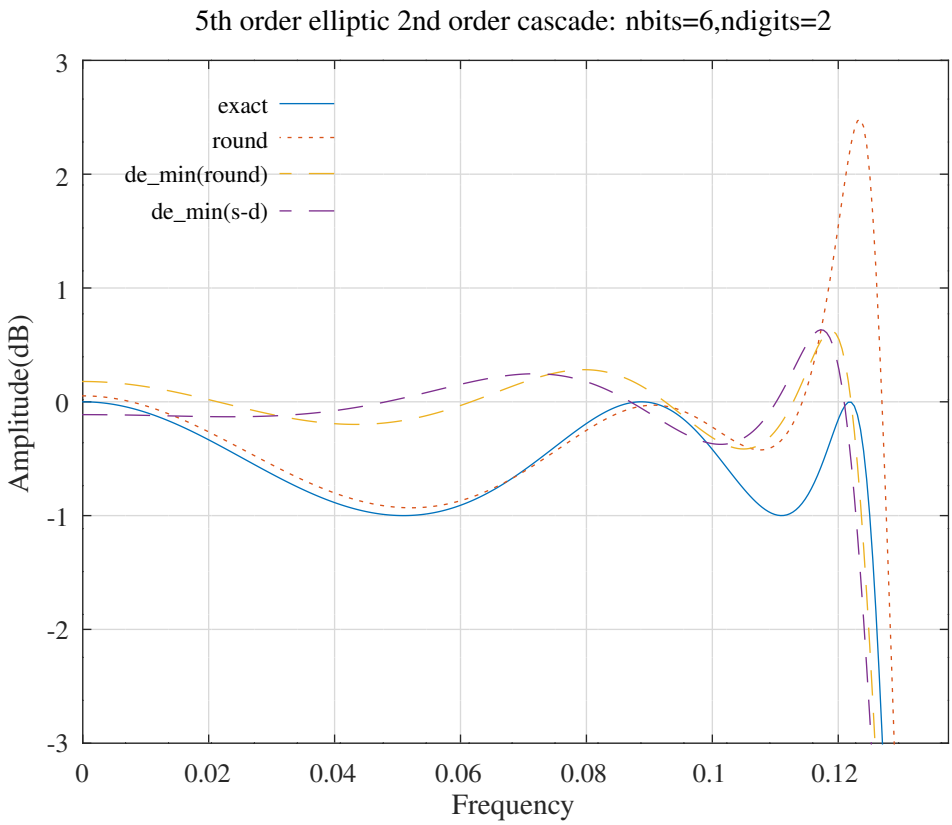



Figure 17.40: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with exact coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the differential evolution algorithm and 6 bit 2 signed-digit coefficients optimised with the differential evolution algorithm

## 17.5 Summary of the search algorithm comparison

Table 17.5 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 | 0.8420 | 1.4617 | 0.8295 | 0.8803 | 0.6781 |
| Rounded with de_min | 0.7797 | 1.2765 | 0.7874 | 0.8803 | 0.6572 |
| Signed-digit | 2.3373 | 2.7157 | 3.6059 | 3.9013 | $\infty$ |
| 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 | 0.8309 | 1.7673 | 1.0841 | 1.6706 | $\infty$ |
| Signed-digit with de_min | 0.7768 | 1.9147 | 1.1879 | 1.3505 | 0.6995 |

Table 17.5: 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 17.6 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 17.6: 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 assuming symmetry is not required.

# Part IV

# Appendixes

# Appendix A

# Review of Complex Variables

This chapter summarises *Kreyszig* [43, 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 $z = x + \imath y$ as $w = f(z) = u(x, y) + \imath v(x, y)$ where $u$ and $v$ are real valued functions of the real variables $x$ and $y$ and, of course, $\imath = \sqrt{-1}$.

## A.2 Limit

A function $f(z)$ is said to have a limit $\ell$ as $z$ approaches $w = f(z) = u(x, y) + \imath v(x, y)$ 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 $\epsilon$ we can find a real positive $\delta$ such that, for all $z \neq z_0$ in the disk $|z - z_0| < \delta$ , $|f(z) - \ell| < \epsilon$. We write:

$$\lim_{z \to z_0} f(z) = \ell$$

A function $f(z)$ is *continuous* at $z = z_0$ if $f(z_0)$ is defined and

$$\lim_{z \to z_0} f(z) = f(z_0)$$

A function $f(z)$ is *differentiable* at $z = z_0$ if the limit

$$f'(z) = \lim_{z \to z_0} \frac{f(z_0 + \Delta z) - f(z_0)}{\Delta z}$$

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, $f(z) = x - \imath y$ is not differentiable. If $\Delta z = \Delta x + \imath \Delta y$ then:

$$\frac{f(z + \Delta z) - f(z)}{\Delta z} = \frac{\Delta x - \imath \Delta y}{\Delta x + \imath \Delta y}$$

If $\Delta z$ approaches $z$ with $\Delta y = 0$ then the derivative is $-1$ but if $\Delta z$ approaches $z$ with $\Delta x = 0$ then the derivative is $1$. Hence the limit approaches different values along different paths to $z = x + \imath y$.

## A.3 The Cauchy-Riemann Equations

Suppose $f(z) = u(x, y) + \imath v(x, y)$ is defined and continuous within a neighbourhood of an arbitrary fixed point $z$ and differentiable at $z$ so that $f'(z)$ exists. Set $\Delta z = \Delta x + \imath \Delta y$. On a path for which $\Delta z$ approaches $z$ with $\Delta y = 0$, then:

$$f'(z) = \lim_{\Delta x \to 0} \frac{u(x + \Delta x, y) - u(x, y)}{\Delta x} + \imath \lim_{\Delta x \to 0} \frac{v(x + \Delta x, y) - v(x, y)}{\Delta x}$$

$$= \frac{\partial u}{\partial x} + \imath \frac{\partial v}{\partial x}$$

Similarly, for a path on which $\Delta z$ approaches $z$ with $\Delta x = 0$

$$f'(z) = \frac{\partial v}{\partial y} - \imath \frac{\partial u}{\partial y}$$

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 $f(z) = u(x, y) + \imath v(x, y)$ 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 $z(t) = x(t) + \imath y(t)$ 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 $t_0 = a \leq t_1 \leq \cdots \leq t_n = b$ corresponding to $z_0 = z(t_0), z_1, \ldots, z_n = z(t_n)$. For each sub-interval of $C$ choose an arbitrary point, $\zeta_i$, 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 $\zeta_i = \varepsilon_i + \imath \eta_i$ and $\Delta z_i = \Delta x_i + \imath \Delta y_i$ and let $f(z) = u(x, y) + \imath v(x, y)$. Then $S_n$ consists of four real sums

$$\begin{aligned} S_n &= \sum_{i=1}^{n} (u + \imath v)(\Delta x_i + \imath \Delta y_i) \\ &= \sum_{i=1}^{n} u \Delta x_i - \sum_{i=1}^{n} v \Delta y_i + \imath \left[ \sum_{i=1}^{n} u \Delta y_i + \sum_{i=1}^{n} v \Delta x_i \right] \end{aligned}$$

and the *line integral* of $f(z)$ along $C$ is defined as the limit of the sums $S_n$ and

$$\begin{aligned} \int_C f(z)\, dz &= \lim_{n \to \infty} S_n \\ &= \int_C u dx - \int_C v dy + \imath \left[ \int_C u dy + \int_C v dx \right] \\ &= \int_a^b u \dot{x} dt - \int_a^b v \dot{y} dt + \imath \left[ \int_a^b u \dot{x} dt + \int_a^b v \dot{y} dt \right] \\ &= \int_c f[z(t)]\, \dot{z} dt \end{aligned}$$

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 $D$

$$\oint_C f(z) = 0$$

PROOF: Cauchy made the additional assumption that $f'(z)$ is continuous and applied Green's theorem. Write

$$\oint_C f(z) = \int_C udx - vdy + \imath \int_C udy + vdx$$

$f(z)$ is analytic so $f'(z)$ exists. For the real part, using the Cauchy-Riemann equations, by Green's theorem

$$\begin{aligned}\int_C udx - vdy &= \iint_R \left[-\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right] dxdy \\ &= 0\end{aligned}$$

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 $z_0$

$$\oint_C \frac{f(z)}{z - z_0} dz = 2\pi\imath f(z_0)$$

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

$$\int_c (z - z_0)^m dz = \begin{cases} 2\pi\imath & m = -1 \\ 0 & m \neq -1, \, integral \end{cases}$$

found by setting $z(t) = z_0 + \rho e^{\imath t}$ 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 $\rho$ of $K$ smaller than $\delta$ 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 $D$

$$f^{(n)}(z_0) = \frac{n!}{2\pi\imath} \oint_c \frac{f(z)}{(z-z_0)^{n+1}} dz, \; n = 1, 2, \cdots$$

PROOF: For $f'(z_0)$

$$f'(z_0) = \lim_{\Delta z \to 0} \frac{f(z_0 + \Delta z) f(z_0)}{\Delta z}$$

Applying Cauchy's Integral formula

$$\begin{aligned} f'(z_0) &= \lim_{\Delta z \to 0} \frac{1}{2\pi\imath \Delta z} \left[ \int_C \frac{f(z)}{z - (z_0 + \Delta z)} dz - \int_C \frac{f(z)}{z - z_0} dz \right] \\ &= \frac{1}{2\pi\imath} \int_C \frac{f(z)}{(z-z_0)^2} dz + \lim_{\Delta z \to 0} \frac{\Delta z}{2\pi\imath} \int_C \frac{f(z)}{(z - z_0 - \Delta z)(z-z_0)^2} dz \end{aligned}$$

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 $L$

$$\left| \frac{\Delta z}{2\pi\imath} \int_C \frac{f(z)}{(z - z_0 - \Delta z)(z-z_0)^2} dz \right| < \frac{|\Delta z|}{2\pi} \frac{M}{\frac{1}{2} d d^2} L$$

As $\Delta z$ 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

$$\begin{aligned} b_n &= \frac{1}{2\pi\imath} \oint_C \frac{f(z^*)}{(z^* - a)^{n+1}} dz^* \\ c_n &= \frac{1}{2\pi\imath} \oint_C (z^* - a)^{n-1} f(z^*) \, dz^* \end{aligned}$$

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 *Kreyszig* [43, 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

$$f(z) = \sum_{n=0}^{\infty} b_n (z-a)^n + \frac{c_1}{(z-a)} + \frac{c_2}{(z-a)^2} + \cdots$$

which converges on the domain $0 < |z-a| < R$. Consequently

$$\oint_C f(z)\, dz = 2\pi\imath c_1$$

$c_1$ 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

$$\oint_C \frac{f'(z)}{f(z)} dz = 2\pi\imath (k-m)$$

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

$$f'(z) = -m(z - z_P)^{-m-1} h(z) + (z - z_N)^{-m} h'(z)$$

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

# Review of selected results from linear algebra

This appendix collects some results from linear algebra used in convex optimization. See, for example, *Golub and Van Loan* [49], *Boyd* and *Vandenberghe* [181, Appendixes A, B and C], *Antoniou* and *Lu* [2, Appendix A], *Gallier* [53] and many, many other on-line resources.

## B.1 Norm of a matrix

The vector norms are:

$$\begin{aligned} \|x\|_p &= \left(|x_1|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}} \qquad p \geq 1 \\ \|x\|_\infty &= \max_{1 \leq i \leq n} |x_i| \end{aligned}$$

$\|x\|$ is assumed to mean $\|x\|_2$.

The matrix norm, $\|A\|_{\alpha,\beta}$, is:

$$\|A\|_{\alpha,\beta} = \sup_{x \neq 0} \frac{\|Ax\|_\beta}{\|x\|_\alpha}$$

*Golub and Van Loan* [49, Section 2.3.1, Section 2.2.1] define the *Frobenius norm* (also called the *Euclidean norm*) of a matrix, $A$, as

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

The Octave manual defines the Frobenius norm of a matrix, $A$, as:

```
norm(A,"fro") = sqrt(sum(diag(A'*A))
```

## B.2 Trace of a matrix

The elements of the product of two matrixes, $\boldsymbol{R}$ and $\boldsymbol{S}$, of equal size, are:

$$[RS]_{kl} = \sum_m R_{km} S_{ml}$$

The *trace* of a matrix is the sum of the diagonal elements:

$$\text{trace}\,(\boldsymbol{T}) = \sum_n T_{nn}$$

For a column vector, $\boldsymbol{y}$, and a matrix, $\boldsymbol{Q}$:

$$\begin{aligned}\boldsymbol{y}^\top \boldsymbol{Q}\boldsymbol{y} &= \sum_k \sum_l Q_{kl} y_k y_l \\ &= \sum_k \sum_l Q_{kl} Y_{kl} \\ &= \operatorname{trace}(\boldsymbol{Q}\boldsymbol{Y})\end{aligned}$$

where $\boldsymbol{Y} = \boldsymbol{y}\boldsymbol{y}^\top$ is symmetric.

## B.3 Rank, range, span and null-space of a matrix

*Golub and Van Loan* [49, Section 2.1.2] define the *range* and *null-space* or *kernel* of a matrix, $B$, as

$$\begin{aligned}\operatorname{range}(B) &= \{y \in \mathbb{R}^n \mid y = Bx\ ,\ x \in \mathbb{R}^n\} \\ \operatorname{null}(B) &= \{x \in \mathbb{R}^n \mid Bx = 0\}\end{aligned}$$

If $B = [b_1, \ldots, b_n]$ is a column vector partitioning then the *span* of $B$ is the set of all linear combinations of those column vectors

$$\operatorname{span}\{b_1, \ldots, b_n\} = \left\{ \sum_{j=1}^n \beta_j b_j \mid \beta_j \in \mathbb{R} \right\}$$

*Golub and Van Loan* [49, Theorem 2.5.2] show that if $A$ is a real $m$-by-$n$ matrix then there exist orthogonal matrixes:

$$\begin{aligned}U &= [u_1, \ldots, u_m] \in \mathbb{R}^{m \times m} \\ V &= [v_1, \ldots, v_n] \in \mathbb{R}^{n \times n}\end{aligned}$$

such that:

$$U^\top A V = \operatorname{diag}[\sigma_1, \ldots, \sigma_p] \in \mathbb{R}^{m \times n} \quad p = \min(m, n)$$

where $\sigma_1 \geq \sigma_1 \geq \ldots \geq \sigma_p \geq 0$. The $\sigma_k$ are called the *singular values* of $A$. Define $r$ by:

$$\sigma_1 \geq \sigma_1 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots \sigma_p = 0$$

then, given the singular value decomposition (SVD) of a matrix, $A$:

$$\begin{aligned}\operatorname{rank}(A) &= r \\ \operatorname{null}(A) &= \operatorname{span}\{v_{r+1}, \ldots, v_n\} \\ \operatorname{range}(A) &= \operatorname{span}\{u_1, \ldots, u_r\} \\ A &= \sum_{k=1}^r \sigma_k u_k v_k^\top = U \Sigma V^\top\end{aligned}$$

## B.4 Matrix determinants

### B.4.1 Definitions

The following definitions of the determinant of a matrix are equivalent:

1. The determinant is a function of a square matrix $A \in \mathbb{R}^{n \times n}$, $\det : \{A \mapsto \mathbb{R}\}$, with the following properties:
   - row replacement on row $r_k$ (ie: $r_k = r_k + m \times r_l$) does not change $\det A$
   - scaling a row by a scalar $p$ multiplies $\det A$ by $p$
   - swapping adjacent rows of $A$ multiplies $\det A$ by $-1$
   - the determinant of the identity matrix $I_n$ is 1

2. Suppose a square matrix, $A$, is reduced to row-echelon form, $B$, by Gaussian elimination. Then:

$$\det A = -1^s \times \frac{\text{product of the diagonal entries of } B}{\text{product of the scaling factors used}}$$

where $s$ is the number of row swaps performed. This is often the most efficient means of calculating $\det A$.

3. The parallelpiped determined by $n$ vectors $r_1, \ldots, r_n \in \mathbb{R}^n$ is the set $P = \{p_1 r_1 + \ldots + p_n r_n \mid 0 \leq p_1, \ldots, p_n \leq 1\}$. If $A$ is the matrix formed from the rows, $r_k$, then the volume of the parallelpiped is $|\det A|$. If the unit cube is transformed by the affine transformation corresponding to matrix $A$ and then by that for matrix $B$ it follows that $\det AB = \det A \times \det B$ and that $\det A^{-1} = [\det A]^{-1}$.

4. The $(k, l)$-*minor* of $A \in \mathbb{R}^{n \times n}$, $M_{kl}$, is the determinant of the $(n-1) \times (n-1)$ matrix formed by deleting row $k$ and column $l$ of $A$. The *cofactor* matrix of $A$ is $C = \left((-1)^{k+l} M_{kl}\right)$. The *adjugate* matrix of $A$ is $\operatorname{adj} A = C^\top$. The adjugate matrix is defined so that $A \operatorname{adj} A = \operatorname{adj} AA = \det AI_n$ and $A^{-1} = [\det A]^{-1} \operatorname{adj} A$. If the entries of $A$ are $a_{kl}$ and the entries of $C$ are $c_{kl} = (-1)^{k+l} M_{kl}$, then the *cofactor expansion alongthe k'th row* is $\det A = \sum_{l=1}^n a_{kl} C_{kl}$. The *cofactor expansion along the l'th column* is similar.

### B.4.2 Matrix exponential

The *matrix exponential* is given by the power series:

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k$$

*Jacobi*'s formula for the derivative of the determinant of a matrix, $A$, is:

$$\frac{d}{dt} \det A(t) = \operatorname{trace}\left(\operatorname{adj} A(t) \frac{d}{dt} A(t)\right)$$

The following corollory is obtained by substituting $A(t) = e^{tB}$:

$$\det e^{tB} = e^{\operatorname{trace} tB}$$

## B.5 Positive-definite matrixes

A matrix, $Q \in \mathbb{R}^{n \times n}$, is positive-definite if $x^\top Q x > 0$ for all nonzero $x \in \mathbb{R}^n$.

A complex matrix can be expressed in terms of the real and imaginary parts as $Z = X + \imath Y$. The matrix $Z$ is positive-definite if, for all complex vectors $z$, $z^\dagger Z z$ is real and greater than 0 ($\dagger$ represents the complex conjugate transpose operation). If $z = x + \imath y$, then:

$$\begin{aligned} z^\dagger Z z &= \left(x^\top - \imath y^\top\right)(X + \imath Y)(x + \imath y) \\ &= \left(x^\top X - \imath y^\top X + \imath x^\top Y + y^\top Y\right)(x + \imath y) \\ &= x^\top X x - \imath y^\top X x + \imath x^\top Y x + y^\top Y x + \imath x^\top y + y^\top X y - x^\top Y y + \imath y^\top Y y \\ &= \left(x^\top X x + y^\top X y\right) + \imath\left(x^\top Y x + y^\top Y y\right) \end{aligned}$$

$Z \succ 0$ if-and-only-if $x^\top Y x + y^\top Y y = 0$ and $\begin{bmatrix} X & Y \\ -Y & X \end{bmatrix} \succ 0$. Alternatively, $Z \succ 0$ if-and-only-if $Z$ is *Hermitian*, that is $Z^\top = \bar{Z}$, where $\bar{Z}$ means the element-wise conjugate of $Z$.

## B.6 Schur complement

This section follows *Gallier* [53]. See also *Jönsson* [209, Section 2.1]. The *Schur* complement is often used to convert non-linear matrix inequalities into linear matrix inequalities.

### B.6.1 Schur complement of a matrix

The *Schur complement* appears in the decomposition of a matrix $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ when solving the linear system:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c \\ d \end{bmatrix}$$

If $D$ is invertible:

$$y = D^{-1}(d - Cx)$$

Substituting:

$$\begin{aligned} Ax + BD^{-1}(d - Cx) &= c \\ \left(A - BD^{-1}C\right)x &= c - BD^{-1}d \end{aligned}$$

If $A - BD^{-1}C$ is invertible, then:

$$\begin{aligned} x &= \left(A - BD^{-1}C\right)^{-1} c - \left(A - BD^{-1}C\right)^{-1} BD^{-1}d \\ y &= D^{-1}\left(d - C\left(A - BD^{-1}C\right)^{-1}\left(c - BD^{-1}d\right)\right) \end{aligned}$$

so that:

$$\begin{aligned} \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} \left(A - BD^{-1}C\right)^{-1} & -\left(A - BD^{-1}C\right)^{-1} BD^{-1} \\ -D^{-1}C\left(A - BD^{-1}C\right)^{-1} & D^{-1} + D^{-1}C\left(A - BD^{-1}C\right)^{-1} BD^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \left(A - BD^{-1}C\right)^{-1} & 0 \\ -D^{-1}C\left(A - BD^{-1}C\right)^{-1} & D^{-1} \end{bmatrix} \begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix} \\ &= \begin{bmatrix} I & 0 \\ -D^{-1}C & I \end{bmatrix} \begin{bmatrix} \left(A - BD^{-1}C\right)^{-1} & 0 \\ 0 & D^{-1} \end{bmatrix} \begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix} \end{aligned}$$

Since:

$$\begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} I & BD^{-1} \\ 0 & I \end{bmatrix}$$

then:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & BD^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} A - BD^{-1}C & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ D^{-1}C & I \end{bmatrix}$$

$A - BD^{-1}C$ is called the *Schur complement of $D$ in $M$*[a].

Alternatively, if $A$ and $D - CA^{-1}B$ are invertible, and with $x = A^{-1}(c - By)$:

$$\begin{aligned} \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} A^{-1} + A^{-1}B\left(D - CA^{-1}B\right)^{-1} CA^{-1} & -A^{-1}B\left(D - CA^{-1}B\right)^{-1} \\ -\left(D - CA^{-1}B\right)^{-1} CA^{-1} & \left(D - CA^{-1}B\right)^{-1} \end{bmatrix} \\ &= \begin{bmatrix} A^{-1} & -A^{-1}B\left(D - CA^{-1}B\right)^{-1} \\ 0 & \left(D - CA^{-1}B\right)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \\ &= \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} \begin{bmatrix} A^{-1} & 0 \\ 0 & \left(D - CA^{-1}B\right)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \\ \begin{bmatrix} A & B \\ C & D \end{bmatrix} &= \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ 0 & I \end{bmatrix} \end{aligned}$$

$D - CA^{-1}B$ is the *Schur complement of $A$ in $M$*.

Comparing the two decompositions of $M$:

$$\left(A - BD^{-1}C\right)^{-1} = A^{-1} + A^{-1}B\left(D - CA^{-1}B\right)^{-1} CA^{-1}$$

so that:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} \left(A - BD^{-1}C\right)^{-1} & -A^{-1}B\left(D - CA^{-1}B\right)^{-1} \\ -\left(D - CA^{-1}B\right)^{-1} CA^{-1} & \left(D - CA^{-1}B\right)^{-1} \end{bmatrix}$$

[a] A corollary is: $\det \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \det\left[A - BD^{-1}C\right] \times \det D$.

### B.6.2 Schur complement of a positive definite matrix

If $M$ is symmetric ($A$ and $D$ are symmetric and $C = B^\top$) then if $D$ is invertible:

1. $M \succ 0 \Leftrightarrow D \succeq 0$ and $A - BD^{-1}B^\top \succ 0$
2. If $D \succ 0$, then $M \succeq 0 \Leftrightarrow A - BD^{-1}B^\top \succeq 0$

Similarly, if $A$ is invertible, then:

1. $M \succ 0 \Leftrightarrow A \succeq 0$ and $D - B^\top A^{-1}B \succ 0$
2. If $A \succ 0$, then $M \succeq 0 \Leftrightarrow D - B^\top A^{-1}B \succeq 0$

These results follow from:

1. a block diagonal matrix, $M$, is positive definite, $M \succ 0$, $\Leftrightarrow$ each diagonal block is positive definite
2. if matrix $T$ is symmetric and matrix $N$ is invertible then $T \succ 0 \Leftrightarrow NTN^\top \succ 0$
3. the block decomposition of $M$ with $\begin{bmatrix} A & B \\ B^\top & D \end{bmatrix} \succ 0 \Leftrightarrow \begin{bmatrix} D & B^\top \\ B & A \end{bmatrix} \succ 0$

Given the quadratic constraint:

$$(Ax + b)^\top (Ax + b) \leq c^\top x + d$$

the equivalent positive semi-definite condition:

$$\begin{bmatrix} I & (Ax + b) \\ (Ax + b)^\top & c^\top x + d \end{bmatrix} \succeq 0$$

follows from the properties of the *Schur complement*.

## B.7 Convex vector spaces

See, for example, *Jönsson* [209, Section 2].

### B.7.1 Definitions on convex sets

A set, $\mathcal{C}$, in a linear vector space is *convex* if $\alpha x_1 + (1-\alpha) x_2 \in \mathcal{C}$ for all $x_1, x_2 \in \mathcal{C}$ and $\alpha \in [0,1]$.

A set, $\mathcal{C}$, is a *convex cone* if it is convex and $\alpha x \in \mathcal{C}$ for all $x \in \mathcal{C}$ and all $\alpha > 0$.

A *convex polytope*, $\mathcal{C}$, with vertices at $x_1, \ldots, x_n \in \mathbb{R}^m$ is defined as the convex hull of those points:

$$\mathcal{C} = \operatorname{cohull}\{x_1, \ldots, x_n\} = \left\{ \sum_{k=0}^{n} \alpha_k x_k : \alpha_k \geq 0,\ \sum_{1}^{n} \alpha_k = 1 \right\}$$

An *ellipsoid*, $\mathcal{E}$, with centre $m$, can be defined by:

$$\begin{aligned} \mathcal{E} &= \left\{ x : (x-m)^\top Q^{-1} (x-m) \leq 1 \right\} \\ &= \left\{ x : x^\top P x + 2x^\top b + c \leq 0 \right\} \end{aligned}$$

where $Q$ is positive definite and symmetric. In the second characterisation $P = Q^{-1}$, $b = -Q^{-1}m$ and $c = m^\top Q^{-1} m - 1$. The size, shape and location of the ellipsoid are determined by $m$ and $Q$. The orientation of $\mathcal{E}$ is determined by the eigenvectors of $Q$ and the lengths of the semi-axes of $\mathcal{E}$ are defined by the eigenvalues of $Q$. The volume of an ellipsoid is:

$$V(\mathcal{E}) = k(n) \det(Q)^{\frac{1}{2}} = k(n) \det\left( \left( b^\top P^{-1} b - c \right) P^{-1} \right)^{\frac{1}{2}}$$

where $k(n)$ is a dimension-dependent constraint. A linear transformation of the ellipsoid, $\mathcal{E}$, changes the centre, shape and location:

$$\begin{aligned} \tilde{\mathcal{E}} &= A\mathcal{E} + b \\ &= \{ y : y = Ax + b : x \in \mathcal{E} \} \\ &= \left\{ y : (y - b - Am)^\top \left( AQA^\top \right)^{-1} (y - b - Am) \leq 1 \right\} \end{aligned}$$

### B.7.2 The separating hyperplane theorem

The *separating hyperplane theorem* is an important result in convex optimisation. I follow the finite dimensional proof of *Jönsson* [209, Section 3]. Also see *Rockafellar* [176, Section 11].

*Jönsson* makes the following definitions relating to a *linear vector space*, $V$, over the real numbers, $\mathbb{R}$:

- a *linear functional* on $V$ is a function $f : V \mapsto \mathbb{R}$, which is linear, i.e. $f(\alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 f(v_1) + \alpha_2 f(v_2)$ for all $v_1, v_2 \in V$ and $\alpha_1, \alpha_2 \in \mathbb{R}$
- the continuous linear functions on $V$ also make a vector space, called the *dual* of $V$, denoted $V^*$
- in applications considering a finite dimensional Hilbert space over $\mathbb{R}$, the dual space is $V^* = V$, and the linear functional $v^* = \langle v, v^* \rangle$ is the inner product
- the *affine hull* of $S$ is defined as the set of all linear combinations of the form $\sum \alpha_k x_k$ where $x_k \in S$ and $\sum \alpha_k = 1$
- the *relative interior* of a set $S \subset \mathbb{R}$, denoted by $\operatorname{ri} S$, is the set of points of $S$ which are interior relative to $\operatorname{aff} S$. This means that for any $x \in S$, there exists an $\varepsilon > 0$ such that all $y \in \operatorname{aff} S$ with $|x - y| < \varepsilon$ are also members of $S$
- a *hyperplane* is an affine subset of $V$ with maximal dimension. In other words, if $V$ has dimension $n$, then every hyperplane is a translation of an $n-1$ dimensional subspace of $V$. A hyperplane can be represented as $H = \{x \in V : \langle v, z \rangle = c\}$ for $z \in V^*$ and $c \in \mathbb{R}$. The sets $H = \{x \in V : \langle v, z \rangle \geq c\}$ and $H = \{x \in V : \langle v, z \rangle \leq c\}$ are closed half-spaces of $V$

*Jönsson* states the following lemma:

*The separation lemma:* Let $C \subset V$ be a relatively open convex set (ie: $\operatorname{ri} C = C$) in a finite dimensional vector space, $V$, and let $V_1$ be a linear subspace of $V$ such that $C \cap V_1 = \emptyset$. Then there exists a hyperplane containing $V_1$ such that one of the open half-spaces associated with the hyperplane contains $C$, ie: $\exists z \in V^\star \setminus 0$ such that $\langle x, z \rangle > 0 \; \forall x \in C$ and $\langle x, z \rangle \leq 0 \; \forall x \in V_1$.

*Jönsson* applies this lemma to prove the following theorem:

*Theorem:* Let $C_1$ be an open convex cone and $C_2$ be a convex set. If these sets are disjoint, $C_1 \cap C_2 = \emptyset$, then there exists a separating hyperplane, ie: $\exists z \in V^\star \setminus 0$ such that $\langle x_1, z \rangle > 0 \; \forall x_1 \in C_1$ and $\langle x_2, z \rangle \leq 0 \; \forall x_2 \in C_2$.

### B.7.3 The S-procedure

*Jönsson* [209, Section 4][b] describes the S-procedure as follows: let $\sigma_k : V \mapsto \mathbb{R}$, $k = 0, \ldots, N$, be real valued functionals on a linear vector space $V$ and consider the following two conditions:

$S_1 : \sigma_0 \geq \; \forall \, y \in V$ such that $\sigma_k(y) \geq 0, \; k = 1, \ldots, N$

$S_2 : \exists \, \tau_k \geq 0, \; k = 1, \ldots, N$ such that $\sigma_0(y) - \sum_{k=1}^{N} \tau_k \sigma_k(y) \geq 0$

Clearly $S_2 \Rightarrow S_1$. The S-procedure is the method of verifying that $S_1 \Rightarrow S_2$.

Suppose that the functionals are quadratic, $\sigma_k(y) = y^\top Q_k y + 2 s_k^\top y + r_k, \; k = 0, \ldots, N$, where $Q_k = Q_k^\top \in \mathbb{R}^{m \times m}$, $s_k \in \mathbb{R}^m$ and $r_k \in \mathbb{R}$. In general, $\sigma_0$ is not convex and, likewise, the set of constraints, $\{y \in \mathbb{R}^m : \sigma_k(y) \geq 0, \; k = 1, \ldots, N\}$ is not convex. On the other hand, $S_2$ corresponds to a linear matrix inequality:

$$\begin{aligned} S_2 &\Leftrightarrow \exists \, \tau_k \geq 0 \text{ such that } \sigma_0(y) - \sum_{k=1}^{N} \tau_k \sigma_k(y) \geq 0, \; \forall \, y \in \mathbb{R}^m \\ &\Leftrightarrow \exists \, \tau_k \geq 0 \text{ such that } \begin{bmatrix} Q_0 & s_0 \\ s_0^\top & r_0 \end{bmatrix} - \sum_{k=1}^{N} \tau_k \begin{bmatrix} Q_k & s_k \\ s_k^\top & r_k \end{bmatrix} \succeq 0 \end{aligned}$$

If the conditions $S_1$ and $S_2$ are equivalent, then the S-procedure is said to be *lossless*. The constraints $\sigma_k(y) \geq 0$ for $k = 1, \ldots, N$ are said to be *regular* if $\exists \, y^\star \in V$ such that $\sigma_k(y^\star) \geq 0$.

*Jönsson* [209] gives the following example of the application of the S-procedure to a quadratic programming problem with one quadratic constraint:

$$\begin{aligned} \textbf{minimise} \quad & y^\top Q_0 y + 2 s_0^\top y + r_0 \\ \textbf{subject to} \quad & y^\top Q_1 y + 2 s_1^\top y + r_{k1} \geq 0 \end{aligned}$$

A lower bound is obtained by the semi-definite relaxation:

$$\begin{aligned} \textbf{minimise} \quad & \operatorname{trace} Q_0 Y + 2 s_0^\top y + r_0 \\ \textbf{subject to} \quad & \operatorname{trace} Q_1 Y + 2 s_1^\top y + r_1 \\ & \begin{bmatrix} Y & y \\ y^\top & 1 \end{bmatrix} \succeq 0 \end{aligned}$$

The dual problem of the semi-definite relaxation is:

$$\begin{aligned} \textbf{maximise} \quad & \gamma \\ \textbf{subject to} \quad & \begin{bmatrix} Q_0 - \lambda Q_1 & s_0 - \lambda s_1 \\ s_0^\top - \lambda s_1^\top & r_0 - \lambda r_1 - \gamma \end{bmatrix} \succeq 0 \\ & \lambda \geq 0 \\ & \gamma \in \mathbb{R} \end{aligned}$$

[b] See also, for example, *Pólik* and *Terlaky* [70].

### B.7.4 The *log det A* penalty function

This $\log\det A$ function is widely used as a penalty function in convex optimisation. *VanAntwerp* and *Braatz* [77, Appendix] show that the function $-\log\det A$ is convex if $A = A^\top \succ 0$. Their proof uses the following lemma:

*Lemma:* A function $f(x)$ is convex in $x \in S$ if-and-only-if $f(t) = f(x_0 + th)$ is convex in t for all $x_0$, $h$ and $t$ such that $x_0 + th \in S$ and $x_0 \in S$.

The proof begins by defining $S = \{A | A = A^\top \succ 0\}$. The lemma implies that $-\log\det A$ is convex in $A$ on $A = A^\top \succ 0$ if-and-only-if $-\log\det(A_0 + tH)$ is convex in $t$ for all $A_0 = A_0^\top \succ 0$ and $H$ which satisfy $A_0 + tH = (A_0 + tH)^\top \succ 0$. The latter condition is equivalent to $I + tA_0^{-\frac{1}{2}} H A_0^{-\frac{1}{2}} \succ 0$. Also, if $\lambda_k$ are the eigenvalues[c] of $A_0^{-\frac{1}{2}} H A_0^{-\frac{1}{2}}$, then $1 + t\lambda_k$ are the eigenvalues of $I + tA_0^{-\frac{1}{2}} H A_0^{-\frac{1}{2}}$. Combining these:

$$\begin{aligned} -\log\det(A_0 + tH) &= -\log\det A_0 - \log\det\left(I + tA_0^{-\frac{1}{2}} H A_0^{-\frac{1}{2}}\right) \\ &= -\log\det A_0 - \sum_k \log(1 + t\lambda_k) \end{aligned}$$

The first and second derivatives of the summand are:

$$\begin{aligned} -\frac{d}{dt}\log(1 + t\lambda_k) &= -\frac{\lambda_k}{1 + t\lambda_k} \\ -\frac{d^2}{dt^2}\log(1 + t\lambda_k) &= \frac{\lambda_k^2}{(1 + t\lambda_k)^2} \end{aligned}$$

The second derivative is positive implying that $-\log\det(A_0 + tH)$ is convex in $t$ for all $t$.

[c] The eigenvalues, $\lambda_k$, of $A$ are the solutions of $\det(\lambda I - A)$. Also $\det A = \prod_k \lambda_k$.

# Appendix C

# Review of Chebyshev's polynomials

This appendix gathers together results for *Chebyshev's polynomials of the first and second kind*.

An FIR filter approximation function is generally written as:

$$H(z) = \sum_{k=0}^{N} h_k z^{-k}$$

If $N = 2M$ and the filter is symmetric, $h_k = h_{N-k}$, then the amplitude response is:

$$A(\omega) = a_0 + \sum_{k=1}^{M} 2a_k \cos k\omega$$

where $a_k = h_{k+M}$. The *Chebyshev polynomials of the first kind* are:

$$T_k(\cos\omega) = \cos k\omega \tag{C.1}$$

The Octave function *chebyshevT(k)* returns the coefficients of the $k$'th Chebyshev polynomial of the first kind. Figure C.1 plots the first 7 Chebyshev polynomials of the first kind.

Similarly, the *Chebyshev polynomials of the second kind* are:

$$U_k(\cos\omega)\sin\omega = \sin(k+1)\omega$$

The *Chebyshev polynomials of the second kind* are related to the *Dirichlet kernel* [a], $D_k(x)$:

$$\begin{aligned} 2\pi D_k(x) &= 1 + 2\sum_{l=1}^{k} \cos lx \\ &= \sum_{l=-k}^{k} e^{\imath lx} \\ &= \frac{\sin(2k+1)\frac{x}{2}}{\sin\frac{x}{2}} \\ &= U_{2k}\left(\cos\frac{x}{2}\right) \end{aligned}$$

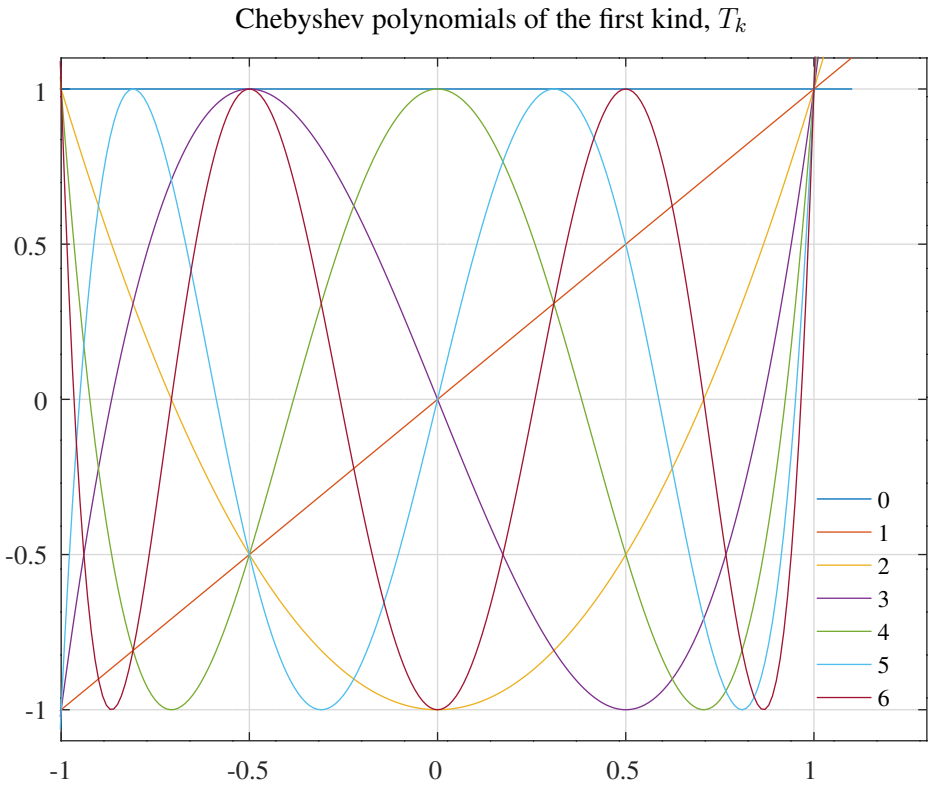
The Octave function *chebyshevU(k)* returns the coefficients of the $k$'th Chebyshev polynomial of the second kind. Figure C.2 plots the first 7 Chebyshev polynomials of the second kind.

[a] The convolution of the $k$'th Dirichlet kernel, $D_k(x)$, with a function, $f$, of period $2\pi$, is the $k$'th degree Fourier series approximation to $f$:

$$(D \star f)(x) = \int_{-\pi}^{\pi} f(y) D_k(x-y)\, dy = \sum_{l=-k}^{k} \hat{f}(l) e^{\imath lx}$$

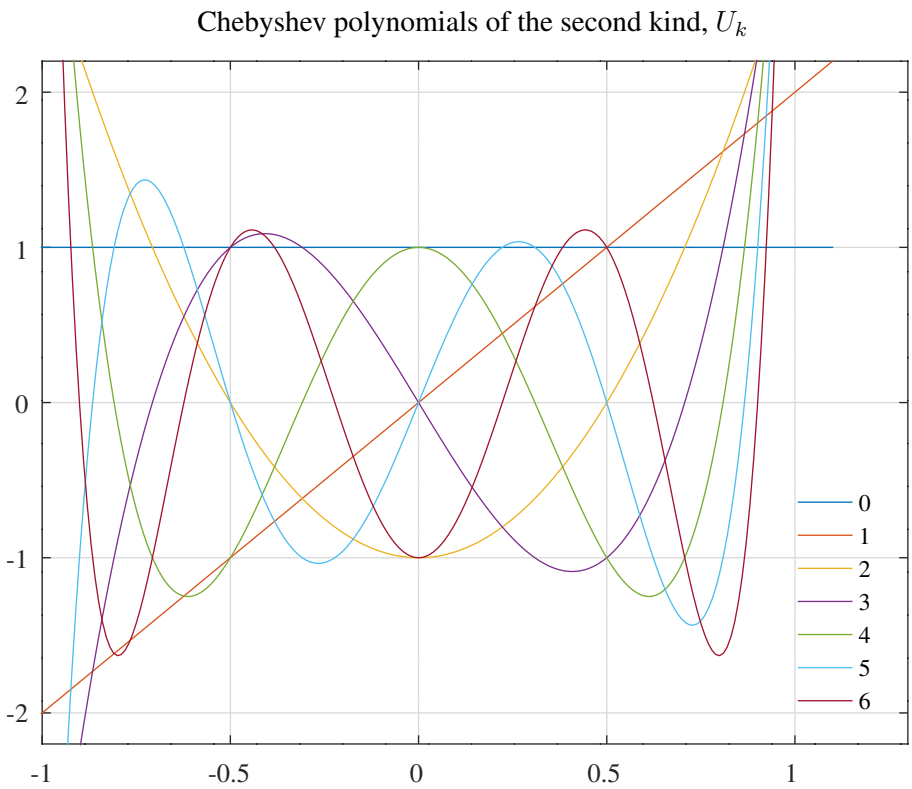
where:

$$\hat{f}(l) = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x) e^{-\imath lx} dx$$




]

Figure C.1: Chebyshev polynomials of the first kind




]

Figure C.2: Chebyshev polynomials of the second kind

## C.1 Recurrence relations

The Chebyshev polynomials of the first kind have the recurrence relation:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{k+1}(x) &= 2xT_k(x) - T_{k-1}(x) \end{aligned} \tag{C.2}$$

The recurrence follows from the trigonometric definition:

$$\begin{aligned} T_{k+1}(\cos\omega) &= \cos\omega\, T_k(\cos\omega) - \sin\omega \sin k\omega \\ T_{k-1}(\cos\omega) &= \cos\omega\, T_k(\cos\omega) + \sin\omega \sin k\omega \end{aligned}$$

The Chebyshev polynomials of the second kind have the recurrence relation:

$$\begin{aligned} U_0(x) &= 1 \\ U_1(x) &= 2x \\ U_{k+1}(x) &= 2xU_k(x) - U_{k-1}(x) \end{aligned}$$

The recurrence follows from the trigonometric definition:

$$\begin{aligned} U_{k+1}(\cos\omega)\sin\omega &= \cos\omega \sin(k+1)\omega + \sin\omega\cos(k+1)\omega \\ U_{k-1}(\cos\omega)\sin\omega &= \cos\omega \sin(k+1)\omega - \sin\omega\cos(k+1)\omega \end{aligned}$$

Again, by substitution of the trigonometric definitions, the Chebyshev polynomials of the first and second kinds are related by the recurrence relations:

$$\begin{aligned} T_{k+1}(x) &= \frac{1}{2}\left(U_{k+1}(x) - U_{k-1}(x)\right) \\ T_k(x) &= U_k(x) - xU_{k-1}(x) \end{aligned}$$

## C.2 Differentiation and integration of the Chebyshev polynomials

Differentiating the trigonometric definition of $T_k(\cos\omega)$ with respect to $\omega$ gives:

$$\frac{dT_k(\cos\omega)}{d\omega} = -k\sin k\omega$$

Substituting $x = \cos\omega$:

$$\begin{aligned} \frac{dT_k(x)}{dx} = \frac{dT_k(\cos\omega)}{d\omega}\frac{d\omega}{dx} &= k\frac{\sin k\omega}{\sin\omega} \\ &= kU_{k-1}(x) \end{aligned}$$

Consequently:

$$T_{k+1}(x) = (k+1)\int U_k(x)\,dx$$

Similarly:

$$\begin{aligned} \frac{dU_k(x)}{dx} = \frac{dU_k(\cos\omega)}{d\omega}\frac{d\omega}{dx} &= \frac{-1}{\sin\omega}\frac{d}{d\omega}\frac{\sin(k+1)\omega}{\sin\omega} \\ &= \frac{(k+1)\,T_{k+1}(x) - xU_k(x)}{x^2-1} \end{aligned}$$

Finally, integrate $T_k(x)$ with the recurrence relation:

$$\begin{aligned} \int T_k(x)\,dx &= \frac{1}{2}\left[\int U_k(x)\,dx - \int U_{k-2}(x)\,dx\right] \\ &= \frac{1}{2}\left[\frac{T_{k+1}(x)}{k+1} - \frac{T_{k-1}(x)}{k-1}\right] \end{aligned}$$

## C.3 Chebyshev differential equations

The Chebyshev polynomials of the first kind satisfy the second order differential equation:

$$\left(1-x^2\right)y''-xy'+k^2y=0$$

since, from the previous section:

$$\begin{aligned}\frac{dT_k(x)}{dx}-kU_{k-1}(x)&=0\\ \frac{d^2T_k(x)}{dx^2}-k\frac{dU_{k-1}(x)}{dx}&=0\\ \left(1-x^2\right)\frac{d^2T_k(x)}{dx^2}-kxU_{k-1}(x)+k^2T_k(x)&=0\\ \left(1-x^2\right)\frac{d^2T_k(x)}{dx^2}-x\frac{dT_k(x)}{dx}+k^2T_k(x)&=0\end{aligned}$$

The Chebyshev polynomials of the second kind satisfy the second order differential equation:

$$\left(1-x^2\right)y''-3xy'+k(k+2)y=0$$

since, similarly:

$$\begin{aligned}\left(1-x^2\right)\frac{dU_k(x)}{dx}-xU_k(x)+(k+1)T_{k+1}(x)&=0\\ \left(1-x^2\right)\frac{d^2U_k(x)}{dx^2}-3x\frac{dU_k(x)}{dx}-U_k(x)+(k+1)^2U_k(x)&=0\\ \left(1-x^2\right)\frac{d^2U_k(x)}{dx^2}-3x\frac{dU_k(x)}{dx}+k(k+2)U_k(x)&=0\end{aligned}$$

## C.4 Orthogonality of the Chebyshev polynomials

The Chebyshev polynomials of the first kind are orthogonal at the *Chebyshev nodes*, $x_k$:

$$\sum_{k=0}^{n-1}T_l(x_k)T_m(x_k)=\begin{cases}0 & \text{if } l\neq m\\ n & \text{if } l=m=0\\ \frac{n}{2} & \text{if } l=m\neq 0\end{cases}\tag{C.3}$$

where $n\geq\max(l,m)$ and $x_k$ are the $n$ zeros of $T_n(x)$ in the interval $[-1,1]$:

$$x_k=\cos\pi\frac{2k+1}{2n}\quad\text{for } k=0,1,\ldots,n-1$$

## C.5 Approximation of functions by Clenshaw's recurrence

The *Clenshaw* algorithm [210, Sections 5.4 and 5.8] calculates the weighted sum:

$$S(x)=\sum_{k=0}^{n}a_k\phi_k(x)$$

where $\phi_k$, $k=0,1,\ldots$, is a sequence of functions that satisfy the recurrence relation:

$$\phi_{k+1}(x)=\alpha_k(x)\phi_k(x)+\beta_k(x)\phi_{k-1}(x)$$

If the coefficients, $a_k$, are known in advance then the *Clenshaw recurrence* calculates auxilliary values, $b_k(x)$, as follows:

$$\begin{aligned}b_{n+1}(x)&=0\\ b_n(x)&=0\\ b_k(x)&=a_k+b_{k+1}(x)\alpha_k(x)+b_{k+2}(x)\beta_{k+1}(x)\end{aligned}$$

By induction:

$$\begin{aligned} b_k\phi_k + b_{k+1}\beta_k\phi_{k-1} &= a_k\phi_k + b_{k+1}\alpha_k\phi_k + b_{k+2}\beta_{k+1}\phi_k + b_{k+1}\beta_k\phi_{k-1} \\ &= a_k\phi_k + b_{k+1}\phi_{k+1} + b_{k+2}\beta_{k+1}\phi_k \end{aligned}$$

so that:

$$S(x) = a_0\phi_0(x) + b_1(x)\phi_1(x) + b_2(x)\beta_1(x)\phi_0(x)$$

The Chebyshev approximation to an arbitrary function, $f(x)$, on the interval $[-1, 1]$[b] is:

$$f(x) \approx -\frac{a_0}{2} + \sum_{k=0}^{n-1} a_k T_k(x)$$

With the orthogonality of the Chebyshev polynomials at the Chebyshev nodes:

$$\begin{aligned} \sum_{l=0}^{n-1} f(x_k) T_l(x_k) &= \sum_{l=0}^{n-1} \left[ -\frac{a_0}{2} + \sum_{k=0}^{n-1} a_k T_k(x_k) \right] T_l(x_k) \\ &= -\sum_{l=0}^{n-1} \frac{a_0}{2} T_l(x_k) + \sum_{l=0}^{n-1}\sum_{k=0}^{n-1} a_k T_k(x_k) T_l(x_k) \\ &= \frac{n}{2} a_k \end{aligned}$$

and:

$$a_k = \frac{2}{n} \sum_{l=0}^{n-1} f\left(\cos \pi \frac{2k+1}{2n}\right) \cos \pi l \frac{2k+1}{2n}$$

Now I can use Clenshaw's recurrence to approximate $f(x)$ with the Chebyshev polynomials of the first kind, for which, from Equation C.2, $\alpha(x) = 2x$ and $\beta(x) = -1$:

$$\begin{aligned} b_{n+1} &= 0 \\ b_n &= 0 \\ b_k &= a_k + 2xb_{k+1} - b_{k+2} \quad k = n-1, n-2, \ldots, 1 \\ b_0 &= 2a_0 + 2xb_1 - b_2 \\ f(x) &\approx \frac{1}{2}(b_0 - b_2) \end{aligned}$$

The Octave script *clenshaw_gaussian_test.m* calculates the Chebychev polynomial expansion coefficients of an approximation to the Gaussian function, $e^{-2x^2}$. The exact approximation coefficients are:

```
ak= = [  0.9315190686,  0.0000000000, -0.4158196500,  0.0000000000, ...
         0.0998610074,  0.0000000000, -0.0161108841, -0.0000000000 ];
```

Figure C.3 shows the Chebyshev approximation and error with exact coefficients. The approximation is close to mini-max. The approximation coefficients rounded to 8 -bits are:

```
ak_fixed_point= = [  119,    0,  -53,    0, ...
                      13,    0,   -2,    0 ];
```

Figure C.4 shows the Chebyshev approximation and error with coefficients rounded to 8 -bits and the Clenshaw recurrence calculated with a 10 -bit accumulator.

[b] Use the following change of variables for a function defined on the interval $[a, b]$:

$$y \equiv \frac{x - \frac{1}{2}(b+a)}{\frac{1}{2}(b-a)}$$

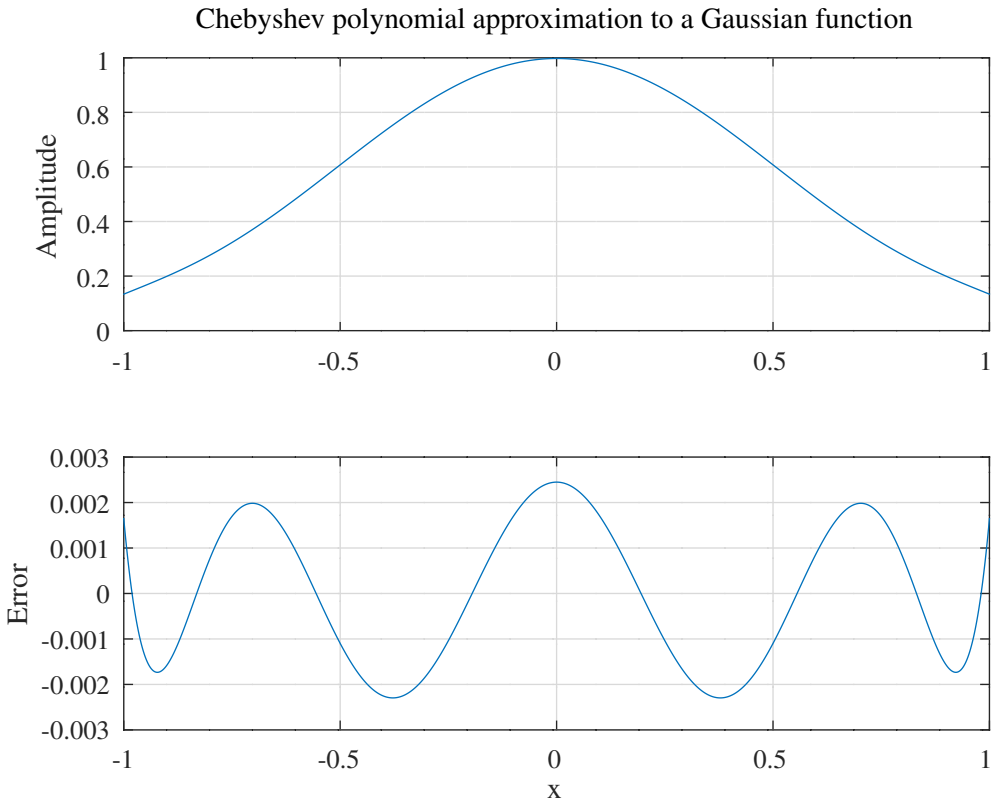



Figure C.3: Chebychev polynomial approximation to a Gaussian

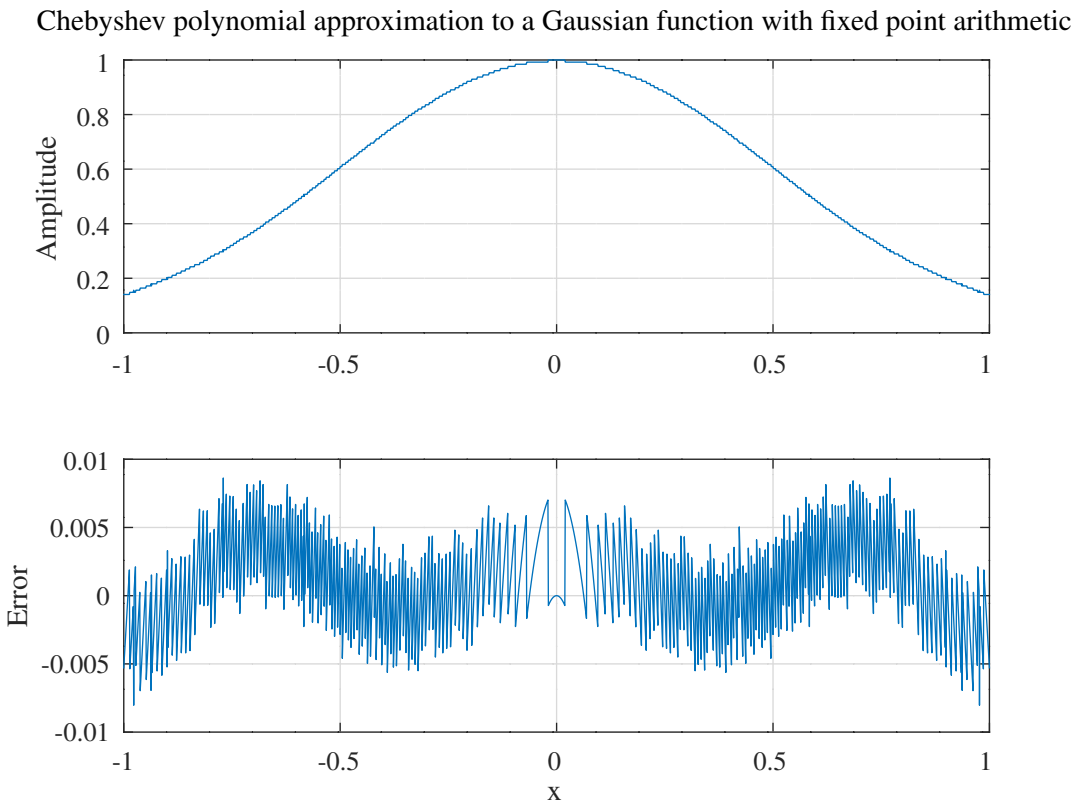



Figure C.4: Chebychev polynomial approximation to a Gaussian with fixed-point coefficients and arithmetic

# Appendix D

# Review of Legendre's elliptic integrals and Jacobi's elliptic functions

This appendix reviews the notation for *Legendre's elliptic integrals* and *Jacobi's elliptic functions*. See *Whittaker* and *Watson* [42, Chapters 20, 21 and 22] and the *NIST Digital Library of MathematicalFunctions* [46, Chapters 19, 20 and 22][a]. *Vlček* and *Unbehauen* [126, Appendix] provide a brief review of the elliptic function theory required for IIR filter design.

## D.1 Doubly periodic functions

*Whittaker* and *Watson* [42, Section 20.1] give the following definition of elliptic functions as doubly-periodic functions:

> Let $\omega_1$ and $\omega_2$ be any two numbers whose ratio is not purely real. A function which satisfies the equations
>
> $$f(z+2\omega_1) = f(z)$$
> $$f(z+2\omega_2) = f(z)$$
>
> for all values of $z$ for which $f(z)$ exists, is called a *doubly periodic* function of $z$, with periods $2\omega_1$ and $2\omega_2$. A doubly-periodic function which is analytic (except at poles), and which has no singularities other than poles in the finite part of the plane, is called an *elliptic function*.

A *cell* of an elliptic function, $f(z)$ is a parallelogram $z, z+2\omega_1, z+2\omega_1+2\omega_2, z+2\omega_2$ for which none of the poles of $f(z)$ lie on the edges of the cell. If $c$ is constant, then the number of roots of $f(z) = c$ that lie within a cell is the *order* of the elliptic function and is equal to the number of poles of $f(z)$ within the cell [42, Section 20.13].

*Whittaker* and *Watson* introduce the general properties of elliptic functions by considering the *Weierstrass* function:

$$\wp(z) = \frac{1}{z^2} + \sum_{\omega \in \mathbb{L}\setminus\{0\}} \left( \frac{1}{(z-\omega)^2} - \frac{1}{\omega^2} \right)$$

where the set of points $\omega = 2m\omega_1 + 2n\omega_2$ with $m, n \in \mathbb{Z}$, constitutes a *lattice*, $\mathbb{L}$. *Whittaker* and *Watson* [42, Section 20.51] show that "any elliptic function can be expressed in terms of the Weierstrassian elliptic functions $\wp(z)$ and $\wp'(z)$ with the same periods, the expression being rational in $\wp(z)$ and linear in $\wp'(z)$". $\wp'(z)$ represents the derivative of $\wp(z)$ with-respect-to $z$.

## D.2 Legendre's elliptic integrals

$F(\phi, \kappa)$ is the *incomplete elliptic integral of the first kind* [46, Equation 19.2.4]

$$F(\phi, \kappa) = \int_0^{\phi} \frac{d\theta}{\sqrt{1-\kappa^2 \sin^2\theta}}$$

[a] The "NIST Digital Library of Mathematical Functions" is the successor to the "Handbook of Mathematical Functions" by *Abramowitz* and *Stegun* [117]. Unfortunately, pages are missing from scans of *Abramowitz* and *Stegun* available on the WWW.

$$= \int_0^{\sin\phi} \frac{dt}{\sqrt{1-t^2}\sqrt{1-\kappa^2 t^2}}$$

where $\kappa$ is the *elliptic modulus*. $\kappa' = \sqrt{1-\kappa^2}$ is the *complementary elliptic modulus*.

$K(\kappa) = F\left(\frac{\pi}{2}, \kappa\right)$ is the *complete elliptic integral of the first kind* [46, Equation 19.2.8]. $K'(\kappa) = K(\kappa')$.

$E(\phi, \kappa)$ is the *incomplete elliptic integral of the second kind* [46, Equation 19.2.5]:

$$\begin{aligned} E(\phi, \kappa) &= \int_0^{\phi} \sqrt{1-\kappa^2 \sin^2\theta} d\theta \\ &= \int_0^{\sin\phi} \frac{\sqrt{1-\kappa^2 t^2}}{\sqrt{1-t^2}} dt \end{aligned}$$

$E(\kappa) = E\left(\frac{\pi}{2}, \kappa\right)$ is the *complete elliptic integral of the second kind*.

$\Pi(\phi, \eta, \kappa)$ is the *incomplete elliptic integral of the third kind* [46, Equation 19.2.7]:

$$\begin{aligned} \Pi(\phi, \eta, \kappa) &= \int_0^{\phi} \frac{d\theta}{\sqrt{1-\kappa^2 \sin^2\theta}\left(1-\eta \sin^2\theta\right)} \\ &= \int_0^{\sin\phi} \frac{dt}{\sqrt{1-t^2}\sqrt{1-\kappa^2 t^2}\left(1-\eta t^2\right)} \end{aligned}$$

$\Pi(\eta, \kappa) = \Pi\left(\frac{\pi}{2}, \eta, \kappa\right)$ is the *complete elliptic integral of the third kind*.

## D.3 Computation of Legendre's elliptic integrals

*Carlson* [15] and [46, Section 19.15] describes a method of computing Legendre's elliptic integrals. He defines the functions:

$$\begin{aligned} R_F(x, y, z) &= \frac{1}{2} \int_0^{\infty} \left[(t+x)(t+y)(t+z)\right]^{-\frac{1}{2}} dt \\ R_C(x, y) &= R_F(x, y, y) \\ R_J(x, y, z, \rho) &= \frac{3}{2} \int_0^{\infty} \left[(t+x)(t+y)(t+z)\right]^{-\frac{1}{2}} (t+\rho)^{-1} dt \\ R_D(x, y, z) &= R_J(x, y, z, z) \end{aligned}$$

and states that [15, Section 4]:

$$\begin{aligned} F(\phi, \kappa) &= (\sin\phi) R_F\left(\cos^2\phi, 1-\kappa^2 \sin^2\phi, 1\right) \\ E(\phi, \kappa) &= (\sin\phi) R_F\left(\cos^2\phi, 1-\kappa^2 \sin^2\phi, 1\right) - \frac{1}{3}\kappa^2 (\sin\phi)^3 R_D\left(\cos^2\phi, 1-\kappa^2 \sin^2\phi, 1\right) \\ \Pi(\phi, \eta, \kappa) &= \sin(\phi) R_F\left(\cos^2\phi, 1-\kappa^2 \sin^2\phi, 1\right) + \frac{\eta}{3} R_J\left(\cos^2\phi, 1-\kappa^2 \sin^2\phi, 1, 1-\eta \sin^2\phi\right) \end{aligned}$$

**Note that for $\Pi(\phi, \eta, \kappa)$ the sign of the terms in $\eta$ is the reverse of that shown by *Carlson* [15, Equation 4.3].**

The complete elliptic integrals are:

$$\begin{aligned} K(\kappa) &= R_F\left(0, 1-\kappa^2, 1\right) \\ E(\kappa) &= R_F\left(0, 1-\kappa^2, 1\right) - \frac{1}{3}\kappa^2 R_D\left(0, 1-\kappa^2, 1\right) \\ \Pi(\eta, \kappa) &= R_F\left(0, 1-\kappa^2, 1\right) + \frac{\eta}{3}\kappa^2 R_J\left(0, 1-\kappa^2, 1, 1-\eta\right) \end{aligned}$$

The computation of $R_F$ uses the, so-called, *duplication formula* [46, Equation 19.26.18]:

$$R_F(x, y, z) = R_F\left(\frac{x+\lambda}{4}, \frac{y+\lambda}{4}, \frac{z+\lambda}{4}\right)$$

where $\lambda = \sqrt{xy} + \sqrt{yz} + \sqrt{zx}$.

Algorithm D.1 [15, Algorithm 1] computes Carlson's $R_F$ function, Algorithm D.2 [15, Algorithm 2] computes Carlson's $R_C$ function, Algorithm D.3 [15, Algorithm 3] computes Carlson's $R_J$ function, and Algorithm D.4 [15, Algorithm 4] computes Carlson's $R_D$ function.

**Algorithm D.1** Carlson's algorithm for computing the $R_F$ function [15, Algorithm 1]

**Require:** $x_0 \geq 0$, $y_0 > 0$ and $z_0 > 0$

**for** $n = 0, \ldots$ **do**

$\lambda_n = (x_n y_n)^{\frac{1}{2}} + (x_n z_n)^{\frac{1}{2}} + (y_n z_n)^{\frac{1}{2}}$

$\mu_n = (x_n + y_n + z_n)/3$

$X_n = 1 - (x_n/\mu_n)$, $Y_n = 1 - (y_n/\mu_n)$, $Z_n = 1 - (z_n/\mu_n)$

$x_{n+1} = (x_n + \lambda_n)/4$, $y_{n+1} = (y_n + \lambda_n)/4$, $z_{n+1} = (z_n + \lambda_n)/4$

$s_n^{(m)} = (X_n^m + Y_n^m + Z_n^m)/2m$, $m = 2, 3$

$\varepsilon_n = \max\{|X_n|, |Y_n|, |Z_n|\}$ $(\varepsilon_n \sim \mathcal{O}(4^{-n}))$

**if** $\varepsilon_n < 1$ **then**

$R_F(x_0, y_0, z_0) = \mu_n^{-\frac{1}{2}} \left[1 + \frac{1}{5} s_n^{(2)} + \frac{1}{7} s_n^{(3)} + \frac{1}{6} \left(s_n^{(2)}\right)^2 + \frac{3}{11} s_n^{(2)} s_n^{(3)} + r_n\right]$

$|r_n| < \frac{\varepsilon_n^6}{4(1-\varepsilon_n)}$, $r_n \sim \frac{5}{26} \left(s_n^{(2)}\right)^3 + \frac{3}{26} \left(s_n^{(3)}\right)^2$

**end if**

**end for**

**Algorithm D.2** Carlson's algorithm for computing the $R_C$ function [15, Algorithm 2]

**Require:** $x_0 \geq 0$ and $y_0 > 0$

**for** $n = 0, \ldots$ **do**

$\lambda_n = 2(x_n y_n)^{\frac{1}{2}} + y_n$

$\mu_n = (x_n + 2y_n)/3$

$s_n = (y_n - x_n)/3\mu_n$,

$x_{n+1} = (x_n + \lambda_n)/4$, $y_{n+1} = (y_n + \lambda_n)/4$

**if** $|s_n| < \frac{1}{2}$ **then**

$R_c(x_0, y_0) = \mu_n^{-\frac{1}{2}} \left[1 + \frac{3}{10} s_n^2 + \frac{1}{7} s_n^3 + \frac{3}{8} s_n^4 + \frac{9}{22} s_n^5 + r_n\right]$

$|r_n| < \frac{16|s_n|^6}{1-2|s_n|}$, $r_n \sim \frac{159}{208} s_n^6$

**end if**

**end for**

**Algorithm D.3** Carlson's algorithm for computing the $R_J$ function [15, Algorithm 3]

**Require:** $x_0 \geq 0$, $y_0 > 0$, $z_0 > 0$ and $\rho_0 > 0$

**for** $n = 0, \ldots$ **do**

$\lambda_n = (x_n y_n)^{\frac{1}{2}} + (x_n z_n)^{\frac{1}{2}} + (y_n z_n)^{\frac{1}{2}}$

$\mu_n = (x_n + y_n + z_n + 2\rho_n)/5$

$X_n = 1 - (x_n/\mu_n)$, $Y_n = 1 - (y_n/\mu_n)$, $Z_n = 1 - (z_n/\mu_n)$, $P_n = 1 - (\rho_n/\mu_n)$

$x_{n+1} = (x_n + \lambda_n)/4$, $y_{n+1} = (y_n + \lambda_n)/4$, $z_{n+1} = (z_n + \lambda_n)/4$, $\rho_{n+1} = (\rho_n + \lambda_n)/4$

$s_n^{(m)} = (X_n^m + Y_n^m + Z_n^m + 2P_n^m)/2m$, $m = 2, 3, 4, 5$

$\alpha_n = \left[\rho_n \left(x^{\frac{1}{2}} + y^{\frac{1}{2}} + z^{\frac{1}{2}}\right) + (x_n y_n z_n)^{\frac{1}{2}}\right]^2$, $\beta = \rho_n (\rho_n + \lambda_n)^2$

$\varepsilon_n = \max\{|X_n|, |Y_n|, |Z_n|, |P_n|\}$ $(\varepsilon_n \sim \mathcal{O}(4^{-n}))$

**if** $\varepsilon_n < 1$ **then**

$R_J(x_0, y_0, z_0, \rho_0) = 3\sum_{m=0}^{n-1} 4^{-m} R_C(\alpha_m, \beta_m) + \ldots$

$4^{-n} \mu_n^{-\frac{3}{2}} \left[1 + \frac{3}{7} s_n^{(2)} + \frac{1}{3} s_n^{(3)} + \frac{3}{22} \left(s_n^{(2)}\right)^2 + \frac{3}{11} s_n^{(4)} + \frac{3}{13} s_n^{(2)} s_n^{(3)} + \frac{3}{13} s_n^{(5)} + r_n\right]$

$|r_n| < \frac{3\varepsilon_n^6}{(1-\varepsilon_n)^{\frac{3}{2}}}$

$r_n \sim -\frac{1}{10} \left(s_n^{(2)}\right)^3 + \frac{3}{10} \left(s_n^{(3)}\right)^2 + \frac{3}{5} s_n^{(2)} s_n^{(4)}$

**end if**

**end for**

**Algorithm D.4** Carlson's algorithm for computing the $R_D$ function [15, Algorithm 4]

**Require:** $x_0 \geq 0$, $y_0 > 0$ and $z_0 > 0$

**for** $n = 0, \ldots$ **do**

$\lambda_n = (x_n y_n)^{\frac{1}{2}} + (x_n z_n)^{\frac{1}{2}} + (y_n z_n)^{\frac{1}{2}}$

$\mu_n = (x_n + y_n + 3z_n)/5$

$X_n = 1 - (x_n/\mu_n)$, $Y_n = 1 - (y_n/\mu_n)$, $Z_n = 1 - (z_n/\mu_n)$

$x_{n+1} = (x_n + \lambda_n)/4$, $y_{n+1} = (y_n + \lambda_n)/4$, $z_{n+1} = (z_n + \lambda_n)/4$

$s_n^{(m)} = (X_n^m + Y_n^m + 3Z_n^m)/2m$, $m = 2, 3, 4, 5$

$\varepsilon_n = \max\{|X_n|, |Y_n|, |Z_n|\}$ $(\varepsilon_n \sim \mathcal{O}(4^{-n}))$

**if** $\varepsilon_n < 1$ **then**

$$R_D(x_0, y_0, z_0) = 3\sum_{m=0}^{n-1} \frac{4^{-m}}{z_m^{\frac{1}{2}}(z_m + \lambda_m)}$$
$$+4^{-n}\mu_n^{-\frac{3}{2}}\left[1 + \frac{3}{7}s_n^{(2)} + \frac{1}{3}s_n^{(3)} + \frac{3}{22}\left(s_n^{(2)}\right)^2 + \frac{3}{11}s_n^{(4)} + \frac{3}{13}s_n^{(2)}s_n^{(3)} + \frac{3}{13}s_n^{(5)} + r_n\right]$$

$$|r_n| < \frac{3\varepsilon_n^6}{(1-\varepsilon_n)^{\frac{3}{2}}},\ r_n \sim -\frac{1}{10}\left(s_n^{(2)}\right)^3 + \frac{3}{10}\left(s_n^{(3)}\right)^2 + \frac{3}{5}s_n^{(2)}s_n^{(4)}$$

**end if**

**end for**

## D.4 Jacobi's theta functions

Jacobi's elliptic functions can be defined as ratios of Jacobi's *theta* functions [46, Section 20.2]. These are defined by the Fourier series:

$$\theta_1(z, q) = 2\sum_{n=0}^{\infty}(-1)^n q^{\left(n+\frac{1}{2}\right)^2}\sin((2n+1)z)$$
$$\theta_2(z, q) = 2\sum_{n=0}^{\infty} q^{\left(n+\frac{1}{2}\right)^2}\cos((2n+1)z)$$
$$\theta_3(z, q) = 1 + 2\sum_{n=1}^{\infty} q^{n^2}\cos(2nz)$$
$$\theta_4(z, q) = 1 + 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos(2nz)$$

where $q = e^{\imath\pi\tau}$ is called the *nome*, with $\Im\tau > 0$ so that $0 < |q| < 1$. For most applications, the unit cell is a rectangle so that $\Re\tau = 0$ and $0 < q < 1$.

Jacobi's *Eta* and *Theta* functions [46, Section 20.1] are, respectively:

$$H(z, q) = \theta_1\left(\frac{z}{\theta_3^2(0, q)}, q\right)$$
$$\Theta(z, q) = \theta_4\left(\frac{z}{\theta_3^2(0, q)}, q\right)$$

## D.5 Computation of Jacobi's theta functions

The Fourier series of Jacobi's theta functions usually converge rapidly because of the doubly-exponential factors $q^{n^2}$. The following transformations[b] are used to compute Jacobi's theta functions when $|q|$ is close to 1 with $\tau' = -\frac{1}{\tau}$ [46, Section

[b] See [46, Section 23.15(i)].Suppose $\mathcal{A}$ denotes a bilinear transformation on $\tau$:

$$\mathcal{A}\tau = \frac{a\tau + b}{c\tau + d}$$

where $a$, $b$, $c$ and $d$ are integers and $ad - bc = 1$. A *modular function* $f(\tau)$ is a function of $\tau$ that is meromorphic in the half-plane $\Im\tau > 0$, and has the property that for all $\mathcal{A}$:

$$f(\mathcal{A}\tau) = c_{\mathcal{A}}(c\tau + d)^l f(\tau)$$

20.7(viii)]:

$$(-\imath\tau)^{\frac{1}{2}}\theta_1(z,\tau) = -\imath\exp\left(\frac{\imath\tau' z^2}{\pi}\right)\theta_1(z\tau',\tau')$$
$$(-\imath\tau)^{\frac{1}{2}}\theta_2(z,\tau) = \exp\left(\frac{\imath\tau' z^2}{\pi}\right)\theta_4(z\tau',\tau')$$
$$(-\imath\tau)^{\frac{1}{2}}\theta_3(z,\tau) = \exp\left(\frac{\imath\tau' z^2}{\pi}\right)\theta_3(z\tau',\tau')$$
$$(-\imath\tau)^{\frac{1}{2}}\theta_4(z,\tau) = \exp\left(\frac{\imath\tau' z^2}{\pi}\right)\theta_2(z\tau',\tau')$$

## D.6 Jacobi's elliptic functions

See [46, Section 22.2]. The nome, $q$, is, in terms of the modulus, $\kappa$:

$$q = e^{-\pi\frac{K'(\kappa)}{K(\kappa)}}$$

The elliptic modulus, $\kappa$, is, in terms of the nome, $q$:

$$\kappa = \frac{\theta_2^2(0,q)}{\theta_3^2(0,q)}$$
$$\kappa' = \frac{\theta_4^2(0,q)}{\theta_3^2(0,q)}$$
$$K(\kappa) = \frac{\pi}{2}\theta_3^2(0,q)$$

If $\zeta = \frac{\pi z}{2K(\kappa)}$, then:

$$\mathrm{sn}(z,\kappa) = \frac{\theta_3(0,q)}{\theta_2(0,q)}\frac{\theta_1(\zeta,q)}{\theta_4(\zeta,q)} = \frac{1}{\mathrm{ns}(z,\kappa)}$$
$$\mathrm{cn}(z,\kappa) = \frac{\theta_4(0,q)}{\theta_2(0,q)}\frac{\theta_2(\zeta,q)}{\theta_4(\zeta,q)} = \frac{1}{\mathrm{nc}(z,\kappa)}$$
$$\mathrm{dn}(z,\kappa) = \frac{\theta_4(0,q)}{\theta_3(0,q)}\frac{\theta_3(\zeta,q)}{\theta_4(\zeta,q)} = \frac{1}{\mathrm{nd}(z,\kappa)}$$
$$\mathrm{sd}(z,\kappa) = \frac{\theta_3^2(0,q)}{\theta_2(0,q)\,\theta_4(0,q)}\frac{\theta_1(\zeta,q)}{\theta_3(\zeta,q)} = \frac{1}{\mathrm{ds}(z,\kappa)}$$
$$\mathrm{cd}(z,\kappa) = \frac{\theta_3(0,q)}{\theta_2(0,q)}\frac{\theta_2(\zeta,q)}{\theta_3(\zeta,q)} = \frac{1}{\mathrm{dc}(z,\kappa)}$$
$$\mathrm{sc}(z,\kappa) = \frac{\theta_3(0,q)}{\theta_4(0,q)}\frac{\theta_1(\zeta,q)}{\theta_2(\zeta,q)} = \frac{1}{\mathrm{cs}(z,\kappa)}$$

From [46, Section 22.2]:

> As a function of $z$ with fixed $\kappa$, each of the 12 Jacobian elliptic functions is doubly periodic, having two periods whose ratio is not real. Each is meromorphic[c] in $z$ for fixed $\kappa$, with simple poles and zeros, and each is meromorphic in $\kappa$ for fixed $z$. For $k \in [0,1]$, all functions are real for $z \in \mathbb{R}$.

The Jacobian elliptic functions have order 2; there are two simple poles in each cell. Table D.1 [46, Tables 22.4.1 and 22.4.2] summarises the periods, pole locations and zero locations of the Jacobian elliptic functions.

*Glaisher's* notation for the Jacobian elliptic functions is [46, Equation 22.2.10]:

$$pq(z,\kappa) = \frac{pr(z,\kappa)}{qr(z,\kappa)} = \frac{1}{qp(z,\kappa)}$$

where $p$, $q$ and $r$ are any three of the letters $s$, $c$, $d$ and $n$. Figure D.1 [46, Figure 22.4.2] shows a mnemonic for the nomenclature of the Jacobian elliptic functions. If $p$ and $q$ are any two distinct letters from the set $s$, $c$, $d$ and $n$ which appear in counter-clockwise orientation at all corners of all lattice unit cells. Then:

where $c_{\mathcal{A}}$ is a constant depending on $\mathcal{A}$, and $l$, called the level, is an integer or half an odd integer. If, as a function of q, $f(\tau)$ is analytic at $q = 0$, then $f(\tau)$ is called a *modular form*.

[c] See Appendix A.10.

| Periods | z-poles | | | | z-zeros | | | |
|---|---|---|---|---|---|---|---|---|
| | $\imath K'$ | $K+\imath K'$ | K | 0 | 0 | K | $K+\imath K'$ | $\imath K'$ |
| $4K, 2\imath K'$ | sn | cd | dc | ns | sn | cd | dc | ns |
| $4K, 2K+2\imath K'$ | cn | sd | nc | ds | sd | cn | ds | nc |
| $2K, 4\imath K'$ | dn | nd | sc | cs | sc | cs | dn | nd |

Table D.1: Periods, pole locations and zero locations of the Jacobian elliptic functions

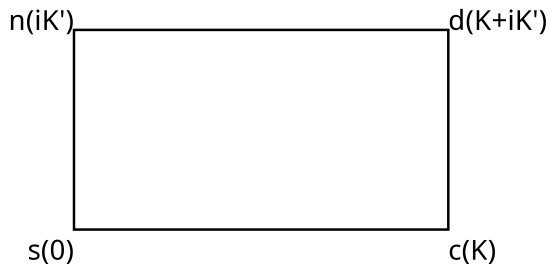



Figure D.1: Jacobian elliptic filter z-plane unit cell

- in any lattice unit cell $pq\,(z,\kappa)$ has a simple zero at $z=p$ and a simple pole at $z=q$
- the difference between $p$ and the nearest $q$ is a half-period of $pq\,(z,\kappa)$

The half-period will be plus or minus a member of the triple $K$, $\imath K'$, $K+\imath K'$; the other two members of this triple are quarter-periods of $pq\,(z,\kappa)$.

## D.7 Inverses of Jacobi's elliptic functions

The inverses of Jacobi's $\mathrm{sn}$, $\mathrm{cn}$ and $\mathrm{cd}$ elliptic functions are, respectively [46, Section 22.15(ii)]:

$$\begin{aligned}
\mathrm{arcsn}\,(x,\kappa) &= \int_0^x \frac{dt}{\sqrt{(1-t^2)\,(1-\kappa^2t^2)}}\,, \quad -1\le x\le 1\\
\mathrm{arccn}\,(x,\kappa) &= \int_x^1 \frac{dt}{\sqrt{(1-t^2)\,(\kappa'^2+\kappa^2t^2)}}\,, \quad -1\le x\le 1\\
\mathrm{arcdn}\,(x,\kappa) &= \int_x^1 \frac{dt}{\sqrt{(1-t^2)\,(\kappa'^2+\kappa^2t^2)}}\,, \quad -1\le x\le 1\\
\mathrm{arcdn}\,(x,\kappa) &= \int_x^1 \frac{dt}{\sqrt{(1-t^2)\,(t^2-\kappa'^2)}}\,, \quad \kappa'\le x\le 1\\
\mathrm{arccd}\,(x,\kappa) &= \int_x^1 \frac{dt}{\sqrt{(1-t^2)\,(1-\kappa^2t^2)}}\,, \quad -1\le x\le 1
\end{aligned}$$

*Carlson* [16, Section 2.1] shows how to calculate the inverse Jacobian elliptic functions with the $R_F$ function. For example,

$$\mathrm{arcsn}\,(z,\kappa) = zR_F\left(1, 1-z^2, 1-\kappa^2z^2\right)$$

## D.8 Elementary identities for the elliptic integrals and elliptic functions

Note that $\mathrm{arcsn}\,(x,\kappa) = F\,(\arcsin x,\kappa)$ so that:

$$x = \int_0^{\mathrm{sn}(x,\kappa)} \frac{dt}{\sqrt{(1-t^2)\,(1-\kappa^2t^2)}}\,, \quad -1\le x\le 1\,, \quad 0\le\kappa\le 1$$

The identities:

$$\begin{aligned}\operatorname{sn}(x,\kappa) &= \sin\phi \\ \operatorname{cn}(x,\kappa) &= \cos\phi \\ \operatorname{dn}(x,\kappa) &= \sqrt{1-\kappa^2\operatorname{sn}^2(x,\kappa)}\end{aligned}$$

give (omitting the $x$ and $\kappa$ arguments):

$$\begin{aligned}\operatorname{sn}^2+\operatorname{cn}^2 &= 1 \\ \operatorname{dn}^2 &= 1-\kappa^2\operatorname{sn}^2 = \kappa'^2+\kappa^2\operatorname{cn}^2 = \operatorname{cn}^2+\kappa'^2\operatorname{sn}^2\end{aligned}$$

The identities:

$$\begin{aligned}\operatorname{sn}(x,\kappa) &= \sin\phi \\ \operatorname{cn}(x,\kappa) &= \cos\phi \\ \operatorname{dn}(x,\kappa) &= \sqrt{1-\kappa^2\operatorname{sn}^2(x,\kappa)}\end{aligned}$$

give (omitting the $x$ and $\kappa$ arguments):

$$\begin{aligned}\operatorname{sn}^2+\operatorname{cn}^2 &= 1 \\ \operatorname{dn}^2 &= 1-\kappa^2\operatorname{sn}^2 = \kappa'^2+\kappa^2\operatorname{cn}^2 = \operatorname{cn}^2+\kappa'^2\operatorname{sn}^2\end{aligned}$$

The half-argument identities for the $\operatorname{sn}$ and $\operatorname{cn}$ functions are [46, Equations 22.6.19 and 22.6.20]:

$$\begin{aligned}\operatorname{sn}^2\left(\frac{z}{2},\kappa\right) &= \frac{1-\operatorname{cn}(z,\kappa)}{1+\operatorname{dn}(z,\kappa)} \\ \operatorname{cn}^2\left(\frac{z}{2},\kappa\right) &= \frac{-\kappa'^2+\operatorname{dn}(z,\kappa)+\kappa^2\operatorname{cn}(z,\kappa)}{\kappa^2(1+\operatorname{cn}(z,\kappa))}\end{aligned}$$

For many more such identities, see, for example, the DLMF [46, Section 22.6].

## D.9 Related functions

Jacobi's *amplitude* function [46, Equation 22.16.1] is:

$$\operatorname{am}(x,\kappa) = \arcsin(\operatorname{sn}(x,\kappa))$$

If $-K \le x \le K$ then the following four equations are equivalent [46, Equations 22.16.10 to 22.16.13]:

$$\begin{aligned}x &= F(\phi,\kappa) \\ \operatorname{am}(x,\kappa) &= \phi \\ \operatorname{sn}(x,\kappa) &= \sin\phi = \sin(\operatorname{am}(x,\kappa)) \\ \operatorname{cn}(x,\kappa) &= \cos\phi = \cos(\operatorname{am}(x,\kappa))\end{aligned}$$

Jacobi's *Epsilon* function [46, Equations 22.16.14 and 22.16.31] is, for $-K \le x \le K$:

$$\begin{aligned}\mathcal{E}(x,\kappa) &= \int_0^{\operatorname{sn}(x,\kappa)} \sqrt{\frac{1-\kappa^2t^2}{1-t^2}}\,dt \\ &= E(\operatorname{am}(x,\kappa),\kappa)\end{aligned}$$

where $-K < x < K$.

Jacobi's *Zeta* function [46, Equation 22.16.32] is:

$$Z(x,\kappa) = E(\arcsin(\operatorname{sn}(x,\kappa)),\kappa) - \frac{E(\kappa)}{K(\kappa)}x$$

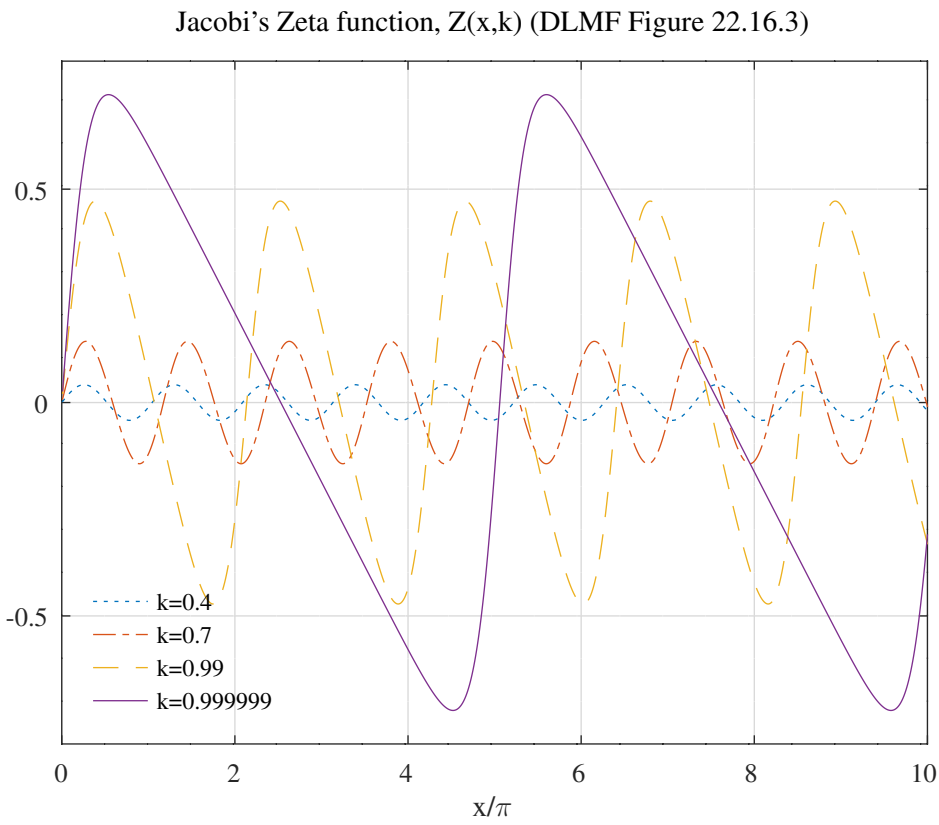



Figure D.2: Jacobi's Zeta function as shown in Figure 22.16.3 of the NIST Digital Library of Mathematical Functions [46]

Alternatively [46, Equation 22.16.30]:

$$Z(z,\kappa) = \frac{1}{\Theta(z,\kappa)}\frac{d\Theta(z,\kappa)}{dz} = \frac{1}{\theta_3^2(0,q)\,\theta_4(\xi,q)}\frac{d\theta_4(\xi,q)}{d\xi}$$

where $\xi = \frac{z}{\theta_3^2(0,q)}$ and $q$ is the *nome* corresponding to $\kappa$.

The *Zeta* function is periodic in $2K$ [46, 22.16.34]:

$$Z(x+2K,\kappa) = Z(x,\kappa)$$

Also [46, 22.16.33]:

$$Z(x+K,\kappa) = Z(x,\kappa) - \kappa^2 \operatorname{sn}(x,\kappa)\operatorname{cd}(x,\kappa)$$

The *Boost* C++ library [26] and *Mathematica* [216] implementations of Jacobi's *Zeta* function use:

$$Z(\phi,\kappa) = E(\phi,\kappa) - \frac{E(\kappa)}{K(\kappa)}F(\phi,\kappa)$$

This definition obscures the periodicity in $K$. Figure 22.16.3 of the DLMF [46], shows the values of Jacobi's *Zeta* function for several elliptic moduluses, $\kappa$. Confusingly, this figure expresses the $x$ argument as a multiple of $\pi$. The Octave script *jacobi_Zeta_test.m* reproduces Figure 22.16.3, as shown in Figure D.2.

## D.10 Octave implementations

Table D.2 lists the Octave functions that implement the elliptic functions and integrals referred to in this Appendix. *ellipj* and *ellipke* are Octave built-in functions.

| Function | Octave implementation |
|---|---|
| $\mathrm{sn}\,(z,\kappa)$, $\mathrm{cn}\,(z,\kappa)$, $\mathrm{dn}\,(z,\kappa)$ | *ellipj* |
| $K\,(\kappa)$, $E\,(\kappa)$ | *ellipke* |
| $\mathrm{arcsn}\,(z,\kappa)$ | *arcsn* |
| $\mathrm{arcsc}\,(z,\kappa)$ | *arcsc* |
| $\mathrm{arccs}\,(z,\kappa)$ | *arccs* |
| $F\,(\phi,\kappa)$ | *elliptic_F* |
| $E\,(\phi,\kappa)$ | *elliptic_E* |
| $\Pi\,(\phi,\eta,\kappa)$ | *elliptic_Pi* |
| $R_F\,(x,y,z)$ | *carlson_RF* |
| $R_C\,(x,y)$ | *carlson_RC* |
| $R_J\,(x,y,z,\rho)$ | *carlson_RJ* |
| $R_D\,(x,y,z)$ | *carlson_RD* |
| $\theta_1\,(z,q)$ | *jacobi_theta1* |
| $\theta_2\,(z,q)$ | *jacobi_theta2* |
| $\theta_3\,(z,q)$ | *jacobi_theta3* |
| $\theta_4\,(z,q)$ | *jacobi_theta4* |
| $H\,(z,\kappa)$ | *jacobi_Eta* |
| $\Theta\,(z,\kappa)$ | *jacobi_Theta* |
| $Z\,(x,\kappa)$ | *jacobi_Zeta* |

Table D.2: Octave implementations of elliptic and related functions

# Appendix E

# Review of *Lanczos* tridiagonalisation of an unsymmetric matrix

See *Golub and Van Loan* [49, 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:

$$\begin{aligned} T &= \begin{bmatrix} t_1 & \cdots & t_N \end{bmatrix} \\ {T^{-1}}^\top = P &= \begin{bmatrix} p_1 & \cdots & p_N \end{bmatrix} \end{aligned}$$

Comparing columns in $AT = T\mathcal{A}$ and $A^\top P = P\mathcal{A}^\top$, for $k \in [1, N-1]$:

$$\begin{aligned} At_k &= \beta_{k-1}t_{k-1} + \alpha_{k-1}t_k + \gamma_k t_{k+1} && \text{with } \beta_0 t_0 \equiv 0 \\ A^\top p_k &= \gamma_{k-1}p_{k-1} + \alpha_{k-1}p_k + \beta_k p_{k+1} && \text{with } \gamma_0 p_0 \equiv 0 \end{aligned}$$

In addition, since $T$ is orthogonal and $P^\top T = I_{N\times N}$:

$$\begin{aligned} \alpha_{k-1} &= p_k^\top A t_k \\ \beta_k p_{k+1} \equiv r_k &= \left(A - \alpha_{k-1}I\right)^\top p_k - \gamma_{k-1}p_{k-1} \\ \gamma_k t_{k+1} \equiv s_k &= \left(A - \alpha_{k-1}I\right) t_k - \beta_{k-1}t_{k-1} \end{aligned}$$

Note that:

$$1 = p_{k+1}^\top t_{k+1} = \left(\frac{r_k}{\beta_k}\right)^\top \left(\frac{s_k}{\gamma_k}\right) \tag{E.1}$$

and so if $\gamma_k$ is specified in advance:

$$\beta_k = \frac{r_k^\top s_k}{\gamma_k} \tag{E.2}$$

A common choice is $\gamma_k = \|s_k\|_2$[a]. Algorithm E.1 calculates the *Lanczos tridiagonalisation* of an unsymmetric matrix.

[a] Recall that $\|s_k\|_2 = \left(s_k^\top s_k\right)^{\frac{1}{2}}$.

**Algorithm E.1** *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 $r_0^\top s_0 \neq 0$
$t_0 = 0$
$p_0 = 0.$
$k = 0$
**while** $r_k \neq 0 \wedge s_k \neq 0 \wedge r_k^\top s_k \neq 0$ **do**
  $\gamma_k = \|s_k\|_2$
  $\beta_k = \frac{r_k^\top s_k}{\gamma_k}$
  $t_{k+1} = \frac{s_k}{\gamma_k}$
  $p_{k+1} = \frac{t_k}{\beta_k}$
  $k = k + 1$
  $\alpha_{k-1} = p_k^\top A t_k$
  $r_k = (A - \alpha_{k-1} I)^\top p_k - \gamma_{k-1} p_{k-1}$
  $s_k = (A - \alpha_{k-1} I)\, t_k - \beta_{k-1} t_{k-1}$
**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 E.1 terminates:

$$\begin{aligned} A^\top \begin{bmatrix} p_1 & \cdots & p_k \end{bmatrix} &= \begin{bmatrix} p_1 & \cdots & p_k \end{bmatrix} \mathcal{A}_k^\top + r_k e_k^\top \\ A \begin{bmatrix} t_1 & \cdots & t_k \end{bmatrix} &= \begin{bmatrix} t_1 & \cdots & t_k \end{bmatrix} \mathcal{A}_k + s_k e_k^\top \end{aligned}$$

If $r_k = 0$ when the iteration terminates then $\operatorname{span}\left\{ \begin{matrix} p_1 & \cdots & p_k \end{matrix} \right\}$ is a subspace for $A$. If $s_k = 0$ when the iteration terminates then $\operatorname{span}\left\{ \begin{matrix} t_1 & \cdots & t_k \end{matrix} \right\}$ is a subspace for $A$. If neither of these conditions is true and $r_k^\top s_k = 0$, then the tridiagonalisation process has failed. *Golub and Van Loan* [49, Section 9.4.4] give a brief review of the *Look-Ahead* technique for solving this problem. The Octave function *lanczos_tridiag* implements Algorithm E.1.

# Appendix F

# Review of Lagrange Interpolation

This chapter follows the review article of *Berrut* and *Trefethen* [83].

## F.1 The barycentric Lagrange polynomial

Let $n+1$ distinct interpolation points or *nodes*, $x_j$, be given, together with corresponding values, $f_j$. The interpolation problem is to find a polynomial such that $p(x_j) = f_j$. The *Lagrange* solution is:

$$\begin{aligned} p(x) &= \sum_{j=0}^{n} f_j l_j(x) \\ l_j &= \frac{\prod_{k=0,k\neq j}^{n}(x-x_k)}{\prod_{k=0,k\neq j}^{n}(x_j-x_k)} \end{aligned} \tag{F.1}$$

The *Lagrange polynomial* $l_j$ corresponding to $x_j$ has the property:

$$l_j(x_k) = \begin{cases} 1, & j=k \\ 0, & otherwise \end{cases}$$

If $l(x) = (x-x_0)\ldots(x-x_n)$, and the *barycentric weights* are defined as:

$$w_j = \frac{1}{\prod_{k\neq j}(x_j-x_k)} = \frac{1}{l'(x_j)} \tag{F.2}$$

then $l_j$ can be rewritten as:

$$l_j = l(x)\frac{w_j}{(x-x_j)} \tag{F.3}$$

so that:

$$p(x) = l(x)\sum_{j=0}^{n}\frac{w_j}{x-x_j}f_j \tag{F.4}$$

If all $f_j = 1$,

$$1 = \sum_{j=0}^{n} l_j(x) = l(x)\sum_{j=0}^{n}\frac{w_j}{x-x_j}$$

giving the *barycentric formula* for $p(x)$:

$$p(x) = \frac{\sum_{j=0}^{n}\frac{w_j}{x-x_j}f_j}{\sum_{j=0}^{n}\frac{w_j}{x-x_j}} \tag{F.5}$$

## F.2 Node distributions

Explicit formulas for the weights, $w_j$, are available for certain sets of nodes, $x_j$. If the nodes are equidistant, with spacing $h = \frac{2}{n}$, in the interval $[-1, 1]$, then:

$$w_j = \frac{(-1)^{n-j} \binom{n}{j}}{h^n n!}$$

After cancelling factors independent of $j$:

$$w_j = (-1)^j \binom{n}{j}$$

Clearly, for large $n$, the weights vary by exponentially large factors and, consequently, interpolation in equally spaced points is *ill-conditioned*[a]. Unless $n$ is small, it is better to use node distributions that are clustered at the end-points of the interval. The simplest such distributions are the *Chebyshev nodes*, formed by projecting equally spaced points on the unit circle down to the interval $[-1, 1]$. The *Chebyshev nodes of the first kind* are:

$$x_j = \cos \frac{(2j+1)\pi}{2n+2}$$

with weights:

$$w_j = (-1)^j \sin \frac{(2j+1)\pi}{2n+2}$$

The *Chebyshev nodes of the second kind* are:

$$x_j = \cos \frac{j\pi}{n}$$

with weights:

$$w_j = (-1)^j \delta_j$$
$$\delta_j = \begin{cases} \frac{1}{2}, & j = 0 \text{ or } j = n \\ 1, & \text{otherwise} \end{cases}$$

*Higham* [134] has shown that Equation F.4 is unconditionally numerically stable and that Equation F.5 is stable if the interpolation points are clustered appropriately.

The Octave script *lagrange_interp_test.m* exercises the implementation of Lagrange interpolation in the Octave function *lagrange_interp*[b]. Figure F.1 attempts to reproduce *Berrut* and *Trefethen*'s Figure 5.1 for 21 Chebyshev nodes of the second kind. Figure F.2 repeats the interpolation experiment with linearly spaced nodes.

[a] This is called the *Runge phenomenon*

[b] The interpolation is much more accurate if the function calculates the weights by the product in Equation F.2 rather than Equation F.3

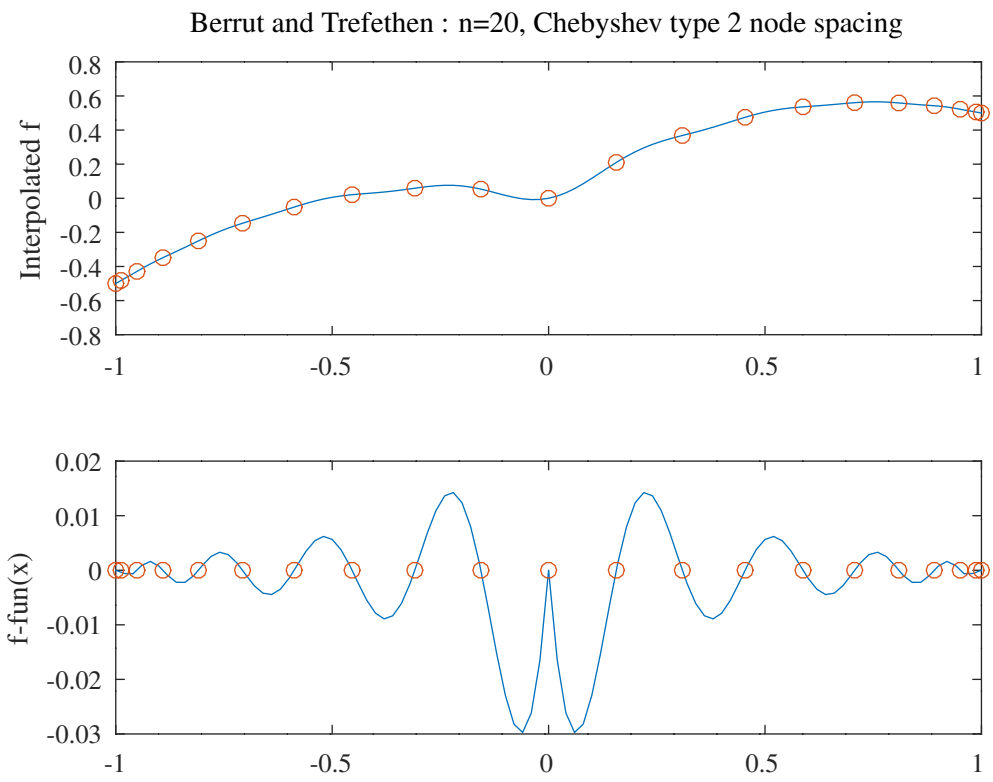



Figure F.1: Lagrange interpolation of the function $fun\,(x) = |x| + \frac{x}{2} - x^2$ with 21 Chebyshev nodes of the second kind and 101 interpolated values on $[-1, 1]$. The marks indicate the values, $f_k$, being interpolated.

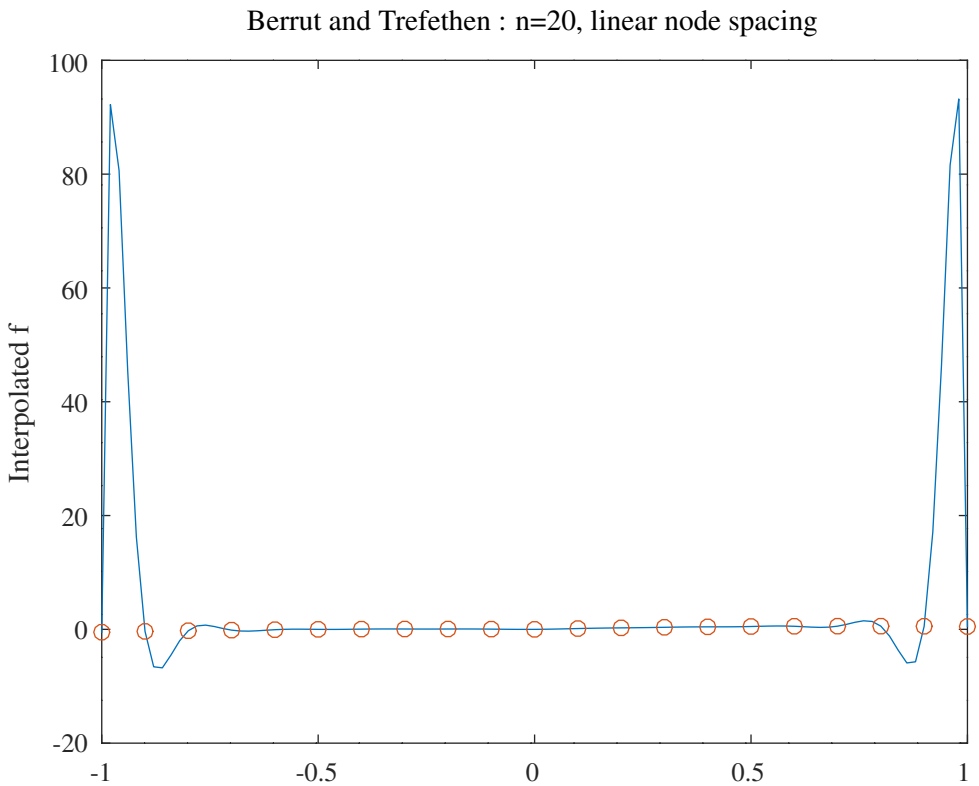



Figure F.2: Lagrange interpolation of the function $fun\,(x) = |x| + \frac{x}{2} - x^2$ with 21 linearly spaced nodes and 101 interpolated values on $[-1, 1]$. The marks indicate the values, $f_k$, being interpolated.

# Appendix G

# 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 coefficent vector
- symbolic differentiation by a symbolic algebra package. For example, *Maxima* [215] or the Octave-Forge *symbolic* calculation toolbox [141]
- 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 [14, *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 formulas[a].

*Richards* [116] shows expressions for the magnitude and group-delay of an IIR filter with an integer decimation factor, $R$, and transfer function

$$H(z) = \frac{N(z)}{D(z)} = K\frac{\sum_{j=0}^{J} n_j z^{-j}}{1+\sum_{l=1}^{L} d_l z^{-Rl}} \tag{G.1}$$

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 G.1 as:

$$H(z) = K\frac{z^{-J}}{z^{-RL}}\frac{\sum_{j=0}^{J} n_j z^{J-j}}{\left[z^{RL}+\sum_{l=1}^{L} d_l z^{R(L-l)}\right]}$$

Suppose there is a zero at $v = re^{\imath\theta}$, then the magnitude response due to this zero is

$$\begin{aligned}\left|\left(e^{\imath\omega} - re^{\imath\theta}\right)\right| &= \left|(\cos\omega - r\cos\theta) + \imath(\sin\omega - r\sin\theta)\right| \\ &= \left\{\cos^2\omega - 2r\cos\omega\cos\theta + r^2\cos^2\theta + \sin^2\omega - 2r\sin\omega\sin\theta + r^2\sin^2\theta\right\}^{\frac{1}{2}}\end{aligned}$$

[a] In 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.

$$= \left\{1 - 2r\cos(\omega - \theta) + r^2\right\}^{\frac{1}{2}}$$

and the phase due to this zero is

$$\arg\left\{e^{\imath\omega} - re^{\imath\theta}\right\} = \arctan\left\{\frac{\sin\omega - r\sin\theta}{\cos\omega - r\cos\theta}\right\}$$

so that the group delay due to this zero is

$$\begin{aligned}
-\frac{d}{d\omega}\arg\left\{e^{\imath\omega} - re^{\imath\theta}\right\} &= -\left[1 + \left(\frac{\sin\omega - r\sin\theta}{\cos\omega - r\cos\theta}\right)^2\right]^{-1}\left[\frac{(\cos\omega - r\cos\theta)\cos\omega + (\sin\omega - r\sin\theta)\sin\omega}{(\cos\omega - r\cos\theta)^2}\right] \\
&= -\frac{1 - r\cos\theta\cos\omega - r\sin\theta\sin\omega}{(\cos\omega - r\cos\theta)^2 + (\sin\omega - r\sin\theta)^2} \\
&= -\frac{1 - r\cos(\omega - \theta)}{1 - 2r\cos(\omega - \theta) + r^2}
\end{aligned}$$

An alternative derivation of the group delay, $T(\omega)$, requires that the amplitude response, $A(\omega)$, be positive:

$$\begin{aligned}
H(e^{\imath\omega}) &= A(\omega)\, e^{\imath\Theta(\omega)} \\
\ln H(e^{\imath\omega}) &= \ln A(\omega) + \imath\Theta(\omega) \\
T(\omega) = -\frac{d}{d\omega}\Theta(\omega) &= -\Im\left\{\frac{H'(e^{\imath\omega})}{H(e^{\imath\omega})}\right\}
\end{aligned}$$

In the following, the filter transfer function, $H(z)$, has 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.

## G.1 IIR filter responses

### G.1.1 IIR filter amplitude response

The amplitude response of $H(z)$ is:

$$\begin{aligned}
A(\omega) = K \times \frac{\prod_{j=1}^{U}\left\{1 - 2R_{0j}\cos\omega + R_{0j}^2\right\}^{\frac{1}{2}}}{\prod_{j=1}^{V}\prod_{i=0}^{R-1}\left\{1 - 2R_{pj}^s\cos(\omega - s2\pi i) + R_{pj}^{2s}\right\}^{\frac{1}{2}}} \times \cdots \\
\cdots \frac{\prod_{j=1}^{\frac{M}{2}}\left\{1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right\}^{\frac{1}{2}}}{\prod_{j=1}^{\frac{Q}{2}}\prod_{i=0}^{R-1}\left\{1 - 2r_{pj}^s\cos(\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}\right\}^{\frac{1}{2}}} \times \cdots \\
\cdots \frac{\prod_{j=1}^{\frac{M}{2}}\left\{1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right\}^{\frac{1}{2}}}{\prod_{j=1}^{\frac{Q}{2}}\prod_{i=0}^{R-1}\left\{1 - 2r_{pj}^s\cos(\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}\right\}^{\frac{1}{2}}}
\end{aligned}$$

The sign of $K$ is included in the amplitude response so that $\frac{\partial A(\omega)}{\partial K}$ is well defined at $K = 0$.

### G.1.2 IIR filter phase response

The phase response of $H(z)$ is:

$$
\begin{aligned}
P(\omega) = {} & [(V+Q)R-(U+M)]\,\omega + \cdots \\
& \cdots \sum_{j=1}^{U} \arctan\left(\frac{\sin\omega}{\cos\omega - R_{0j}}\right) - \cdots \\
& \cdots \sum_{j=1}^{V} \left\{ \sum_{i=0}^{R-1} \arctan\left(\frac{\sin\omega - R_{pj}^{s}\sin(s2\pi i)}{\cos\omega - R_{pj}^{s}\cos(s2\pi i)}\right)\right\} + \cdots \\
& \cdots \sum_{j=1}^{\frac{M}{2}} \arctan\left(\frac{\sin 2\omega - 2r_{0j}\cos\theta_{0j}\sin\omega}{\cos 2\omega - 2r_{0j}\cos\theta_{0j}\cos\omega + r_{0j}^{2}}\right) - \cdots \\
& \cdots \sum_{j=1}^{\frac{Q}{2}} \left\{ \sum_{i=0}^{R-1} \arctan\left(\frac{\sin 2\omega - 2r_{pj}^{s}\cos[s(\theta_{pj}+2\pi i)]\sin\omega}{\cos 2\omega - 2r_{pj}^{s}\cos[s(\theta_{pj}+2\pi i)]\cos\omega + r_{pj}^{2s}}\right)\right\}
\end{aligned}
$$

As noted above, the phase response does not include the sign of $K$.

### G.1.3 IIR filter group-delay response

The group-delay response of $H(z)$ is:

$$
\begin{aligned}
T(\omega) = {} & -[(V+Q)R-(U+M)] - \cdots \\
& \cdots \sum_{j=1}^{U} \frac{1 - R_{0j}\cos\omega}{1 - 2R_{0j}\cos\omega + R_{0j}^{2}} + \cdots \\
& \cdots \sum_{j=1}^{V} \left\{ \sum_{i=0}^{R-1} \frac{1 - R_{pj}^{s}\cos(\omega - s2\pi i)}{1 - 2R_{pj}^{s}\cos(\omega - s2\pi i) + R_{pj}^{2s}}\right\} - \cdots \\
& \cdots \sum_{j=1}^{\frac{M}{2}} \left\{ \frac{1 - r_{0j}\cos(\omega - \theta_{0j})}{1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^{2}} + \frac{1 - r_{0j}\cos(\omega + \theta_{0j})}{1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^{2}}\right\} + \cdots \\
& \cdots \sum_{j=1}^{\frac{Q}{2}} \left\{ \sum_{i=0}^{R-1} \frac{1 - r_{pj}^{s}\cos(\omega - s(\theta_{pj}+2\pi i))}{1 - 2r_{pj}^{s}\cos(\omega - s(\theta_{pj}+2\pi i)) + r_{pj}^{2s}}\right\} + \cdots \\
& \cdots \sum_{j=1}^{\frac{Q}{2}} \left\{ \sum_{i=0}^{R-1} \frac{1 - r_{pj}^{s}\cos(\omega + s(\theta_{pj}+2\pi i))}{1 - 2r_{pj}^{s}\cos(\omega + s(\theta_{pj}+2\pi i)) + r_{pj}^{2s}}\right\}
\end{aligned}
$$

## G.2 Partial derivatives of the IIR filter responses

### G.2.1 Partial derivatives of the IIR filter amplitude response

The partial derivatives of the magnitude response with respect to the coefficients are:

$$
\begin{aligned}
\frac{\partial A(\omega)}{\partial K} &= \frac{A(\omega)}{K} \\
\frac{\partial A(\omega)}{\partial R_{0j}} &= A(\omega)\left\{\frac{R_{0j} - \cos\omega}{1 - 2R_{0j}\cos\omega + R_{0j}^{2}}\right\} \\
\frac{\partial A(\omega)}{\partial R_{pj}} &= -sR_{pj}^{s-1}A(\omega)\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^{s} - \cos(\omega - s2\pi i)}{1 - 2R_{pj}^{s}\cos(\omega - s2\pi i) + R_{pj}^{2s}}\right\} \\
\frac{\partial A(\omega)}{\partial r_{0j}} &= A(\omega)\left\{\frac{r_{0j} - \cos(\omega - \theta_{0j})}{1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^{2}} + \frac{r_{0j} - \cos(\omega + \theta_{0j})}{1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^{2}}\right\}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial A(\omega)}{\partial \theta_{0j}} &= A(\omega)\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}-\frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\} \\
\frac{\partial A(\omega)}{\partial r_{pj}} &= -sr_{pj}^{s-1}A(\omega)\sum_{i=0}^{R-1}\left\{\frac{r_{pj}^s-\cos(\omega-s(\theta_{pj}+2\pi i))}{1-2r_{pj}^s\cos(\omega-s(\theta_{pj}+2\pi i))+r_{pj}^{2s}}+\cdots\right. \\
&\quad \left.\cdots\frac{r_{pj}^s-\cos(\omega+s(\theta_{pj}+2\pi i))}{1-2r_{pj}^s\cos(\omega+s(\theta_{pj}+2\pi i))+r_{pj}^{2s}}\right\} \\
\frac{\partial A(\omega)}{\partial \theta_{pj}} &= -sr_{pj}^{s}A(\omega)\sum_{i=0}^{R-1}\left\{\frac{\sin(\omega+s(\theta_{pj}+2\pi i))}{1-2r_{pj}^s\cos(\omega+s(\theta_{pj}+2\pi i))+r_{pj}^{2s}}-\cdots\right. \\
&\quad \left.\cdots\frac{\sin(\omega-s(\theta_{pj}+2\pi i))}{1-2r_{pj}^s\cos(\omega-s(\theta_{pj}+2\pi i))+r_{pj}^{2s}}\right\}
\end{aligned}
$$

### G.2.2 Partial derivatives of the IIR filter phase response

The partial derivatives of the phase response 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

$$
\begin{aligned}
P_{0N} &= \sin 2\omega - 2r_{0j}\cos\theta_{0j}\sin\omega \\
\frac{\partial P_{0N}}{\partial r_{0j}} &= -2\cos\theta_{0j}\sin\omega \\
\frac{\partial P_{0N}}{\partial \theta_{0j}} &= 2r_{0j}\sin\theta_{0j}\sin\omega \\
P_{0D} &= \cos 2\omega - 2r_{0j}\cos\theta_{0j}\cos\omega + r_{0j}^2 \\
\frac{\partial P_{0D}}{\partial r_{0j}} &= -2\cos\theta_{0j}\cos\omega + 2r_{0j} \\
\frac{\partial P_{0D}}{\partial \theta_{0j}} &= 2r_{0j}\sin\theta_{0j}\cos\omega
\end{aligned}
$$

then

$$
\begin{aligned}
\frac{\partial P(\omega)}{\partial r_{0j}} &= \frac{P_{0D}\frac{\partial P_{0N}}{\partial r_{0j}} - P_{0N}\frac{\partial P_{0D}}{\partial r_{0j}}}{P_{0N}^2+P_{0D}^2} \\
\frac{\partial P(\omega)}{\partial \theta_{0j}} &= \frac{P_{0D}\frac{\partial P_{0N}}{\partial \theta_{0j}} - P_{0N}\frac{\partial P_{0D}}{\partial \theta_{0j}}}{P_{0N}^2+P_{0D}^2}
\end{aligned}
$$

For convenience, write

$$
\begin{aligned}
P_{pN} &= \sin 2\omega - 2r_{pj}^s\cos[s(\theta_{pj}+2\pi i)]\sin\omega \\
\frac{\partial P_{pN}}{\partial r_{pj}} &= -2sr_{pj}^{s-1}\cos[s(\theta_{pj}+2\pi i)]\sin\omega \\
\frac{\partial P_{pN}}{\partial \theta_{pj}} &= 2sr_{pj}^{s}\sin[s(\theta_{pj}+2\pi i)]\sin\omega \\
P_{pD} &= \cos 2\omega - 2r_{pj}^s\cos[s(\theta_{pj}+2\pi i)]\cos\omega + r_{pj}^{2s} \\
\frac{\partial P_{pD}}{\partial r_{pj}} &= -2sr_{pj}^{s-1}\cos[s(\theta_{pj}+2\pi i)]\cos\omega + 2sr_{pj}^{2s-1} \\
\frac{\partial P_{pD}}{\partial \theta_{pj}} &= 2sr_{pj}^{s}\sin[s(\theta_{pj}+2\pi i)]\cos\omega
\end{aligned}
$$

then

$$\frac{\partial P(\omega)}{\partial r_{pj}} = -\sum_{i=0}^{R-1}\left\{\frac{P_{pD}\frac{\partial P_{pN}}{\partial r_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial r_{pj}}}{P_{pN}^2 + P_{pD}^2}\right\}$$

$$\frac{\partial P(\omega)}{\partial \theta_{pj}} = -\sum_{i=0}^{R-1}\left\{\frac{P_{pD}\frac{\partial P_{pN}}{\partial \theta_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial \theta_{pj}}}{P_{pN}^2 + P_{pD}^2}\right\}$$

### G.2.3 Partial derivatives of the IIR filter group-delay response

The partial derivatives of the group delay are

$$\frac{\partial T(\omega)}{\partial K} = 0$$

$$\frac{\partial T(\omega)}{\partial R_{0j}} = \frac{2R_{0j} - \left(R_{0j}^2 + 1\right)\cos\omega}{\left[1 - 2R_{0j}\cos\omega + R_{0j}^2\right]^2}$$

$$\frac{\partial T(\omega)}{\partial R_{pj}} = sR_{pj}^{s-1}\sum_{i=0}^{R-1}\frac{\left(R_{pj}^{2s} + 1\right)\cos(\omega - s2\pi i) - 2R_{pj}^s}{\left[1 - 2R_{pj}^s\cos(\omega - s2\pi i) + R_{pj}^{2s}\right]^2}$$

$$\frac{\partial T(\omega)}{\partial r_{0j}} = \frac{2r_{0j} - \left(r_{0j}^2 + 1\right)\cos(\omega - \theta_{0j})}{\left[1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right]^2} + \frac{2r_{0j} - \left(r_{0j}^2 + 1\right)\cos(\omega + \theta_{0j})}{\left[1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right]^2}$$

$$\frac{\partial T(\omega)}{\partial \theta_{0j}} = \frac{r_{0j}(r_{0j}^2 - 1)\sin(\omega - \theta_{0j})}{\left[1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right]^2} - \frac{r_{0j}(r_{0j}^2 - 1)\sin(\omega + \theta_{0j})}{\left[1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right]^2}$$

$$\frac{\partial T(\omega)}{\partial r_{pj}} = sr_{pj}^{s-1}\sum_{i=0}^{R-1}\left\{\frac{\left(r_{pj}^{2s} + 1\right)\cos(\omega - s(\theta_{pj} + 2\pi i)) - 2r_{pj}^s}{\left[1 - 2r_{pj}^s\cos(\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}\right]^2} + \cdots\right.$$
$$\left.\cdots\frac{\left(r_{pj}^{2s} + 1\right)\cos(\omega + s(\theta_{pj} + 2\pi i)) - 2r_{pj}^s}{\left[1 - 2r_{pj}^s\cos(\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}\right]^2}\right\}$$

$$\frac{\partial T(\omega)}{\partial \theta_{pj}} = sr_{pj}^s\left(1 - r_{pj}^{2s}\right)\sum_{i=0}^{R-1}\left\{\frac{\sin(\omega - s(\theta_{pj} + 2\pi i))}{\left[1 - 2r_{pj}^s\cos(\omega - s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}\right]^2} - \cdots\right.$$
$$\left.\cdots\frac{\sin(\omega + s(\theta_{pj} + 2\pi i))}{\left[1 - 2r_{pj}^s\cos(\omega + s(\theta_{pj} + 2\pi i)) + r_{pj}^{2s}\right]^2}\right\}$$

## G.3 Second partial derivatives of the IIR filter response

### G.3.1 Second partial derivatives of the IIR filter amplitude response

The second partial derivatives of the magnitude response are (with $j \neq k$, and otherwise assumed to vary across all indexes):

$$\frac{\partial^2 A(\omega)}{\partial K^2} = 0$$

$$\frac{\partial^2 A(\omega)}{\partial K \partial R_{0j}} = \frac{1}{K}\frac{\partial A(\omega)}{\partial R_{0j}}$$

$$\frac{\partial^2 A(\omega)}{\partial K \partial R_{pj}} = \frac{1}{K}\frac{\partial A(\omega)}{\partial R_{pj}}$$

$$\frac{\partial^2 A(\omega)}{\partial K \partial r_{0j}} = \frac{1}{K}\frac{\partial A(\omega)}{\partial r_{0j}}$$

$$\frac{\partial^2 A(\omega)}{\partial K \partial \theta_{0j}} = \frac{1}{K}\frac{\partial A(\omega)}{\partial \theta_{0j}}$$

$$\frac{\partial^2 A(\omega)}{\partial K \partial r_{pj}} = \frac{1}{K}\frac{\partial A(\omega)}{\partial r_{pj}}$$

$$\frac{\partial^2 A(\omega)}{\partial K \partial \theta_{pj}} = \frac{1}{K}\frac{\partial A(\omega)}{\partial \theta_{pj}}$$

$$\frac{\partial^2 A(\omega)}{\partial R_{0j}^2} = A(\omega)\frac{\sin^2\omega}{\left[1-2R_{0j}\cos\omega+R_{0j}^2\right]^2}$$

$$\frac{\partial^2 A(\omega)}{\partial R_{0k}\partial R_{0j}} = \frac{\partial A(\omega)}{\partial R_{0k}}\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial R_{pj}\partial R_{0j}} = \frac{\partial A(\omega)}{\partial R_{pj}}\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{0j}\partial R_{0j}} = \frac{\partial A(\omega)}{\partial r_{0j}}\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{0j}\partial R_{0j}} = \frac{\partial A(\omega)}{\partial \theta_{0j}}\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}\partial R_{0j}} = \frac{\partial A(\omega)}{\partial r_{pj}}\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{pj}\partial R_{0j}} = \frac{\partial A(\omega)}{\partial \theta_{pj}}\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2}\right\}$$

$$\begin{aligned}\frac{\partial^2 A(\omega)}{\partial R_{pj}^2} &= -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial R_{pj}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}+\cdots\\ &\cdots sR_{pj}^{s-2}A(\omega)\sum_{i=0}^{R-1}\left\{\frac{R_{pj}^{3s}+(s-3)R_{pj}^{2s}\cos(\omega-s2\pi i)}{\left[1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}\right]^2}+\cdots\right.\\ &\left.\cdots\frac{\left(2\cos^2(\omega-s2\pi i)-2s+1\right)R_{pj}^s+(s-1)\cos(\omega-s2\pi i)}{\left[1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}\right]^2}\right\}\end{aligned}$$

$$\frac{\partial^2 A(\omega)}{\partial R_{pk}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial R_{pk}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{0j}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial r_{0j}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{0j}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial \theta_{0j}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial r_{pj}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{pj}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial \theta_{pj}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\begin{aligned}\frac{\partial^2 A(\omega)}{\partial r_{0j}^2} &= \frac{\partial A(\omega)}{\partial r_{0j}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}+\cdots\\ &\cdots A(\omega)\left\{\frac{2\sin^2(\omega-\theta_{0j})}{\left[1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2\right]^2}-\frac{1}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\cdots\right.\\ &\left.\cdots\frac{2\sin^2(\omega+\theta_{0j})}{\left[1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2\right]^2}-\frac{1}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}\end{aligned}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{0k}\partial r_{0j}} = \frac{\partial A(\omega)}{\partial r_{0k}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}$$

$$\begin{aligned}\frac{\partial^2 A(\omega)}{\partial \theta_{0j}\partial r_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{0j}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}+\cdots\\ &\cdots A(\omega)\left\{\frac{\left(1-r_{0j}^2\right)\sin(\omega+\theta_{0j})}{\left[1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2\right]^2}-\frac{\left(1-r_{0j}^2\right)\sin(\omega-\theta_{0j})}{\left[1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2\right]^2}\right\}\end{aligned}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{0k}\partial r_{0j}} = \frac{\partial A(\omega)}{\partial\theta_{0k}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}\partial r_{0j}} = \frac{\partial A(\omega)}{\partial r_{pj}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{pj}\partial r_{0j}} = \frac{\partial A(\omega)}{\partial\theta_{pj}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{0j}^2} = \frac{\partial A(\omega)}{\partial\theta_{0j}}\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}-\frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\}+\cdots$$

$$\cdots A(\omega)\left\{\frac{\left(r_{0j}+r_{0j}^3\right)\cos(\omega+\theta_{0j})-2r_{0j}^2}{\left[1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2\right]^2}+\frac{\left(r_{0j}+r_{0j}^3\right)\cos(\omega-\theta_{0j})-2r_{0j}^2}{\left[1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2\right]^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{0k}\partial\theta_{0j}} = \frac{\partial A(\omega)}{\partial\theta_{0k}}\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}-\frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}\partial\theta_{0j}} = \frac{\partial A(\omega)}{\partial r_{pj}}\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}-\frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{pj}\partial\theta_{0j}} = \frac{\partial A(\omega)}{\partial\theta_{pj}}\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}-\frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\}$$

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

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}^2} = -s\frac{\partial A(\omega)}{\partial r_{pj}}\sum_{i=0}^{R-1}\left\{\frac{N_n}{D_n}+\frac{N_p}{D_p}\right\}-\cdots$$

$$\cdots sA(\omega)\sum_{i=0}^{R-1}\left\{\frac{(2s-1)r_{pj}^{2s-2}-(s-1)r_{pj}^{s-2}\cos(\omega-s(\theta_{pj}+2\pi i))}{D_n}-\frac{2sN_n^2}{D_n^2}+\cdots\right.$$

$$\left.\cdots\frac{(2s-1)r_{pj}^{2s-2}-(s-1)r_{pj}^{s-2}\cos(\omega+s(\theta_{pj}+2\pi i))}{D_p}-\frac{2sN_p^2}{D_p^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pk}\partial r_{pj}} = -s\frac{\partial A(\omega)}{\partial r_{pk}}\sum_{i=0}^{R-1}\left\{\frac{N_n}{D_n}+\frac{N_p}{D_p}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{pj}\partial r_{pj}} = -s\frac{\partial A(\omega)}{\partial\theta_{pj}}\sum_{i=0}^{R-1}\left\{\frac{N_n}{D_n}+\frac{N_p}{D_p}\right\}-\cdots$$

$$\cdots sA(\omega)\sum_{i=0}^{R-1}\left\{sr_{pj}^{s-1}\sin(\omega+s(\theta_{pj}+2\pi i))\left[\frac{1}{D_p}-\frac{2N_pr_{pj}}{D_p^2}\right]-\cdots\right.$$

$$\left.\cdots sr_{pj}^{s-1}\sin(\omega-s(\theta_{pj}+2\pi i))\left[\frac{1}{D_n}-\frac{2N_nr_{pj}}{D_n^2}\right]\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{pk}\partial r_{pj}} = -s\frac{\partial A(\omega)}{\partial\theta_{pk}}\sum_{i=0}^{R-1}\left\{\frac{N_n}{D_n}+\frac{N_p}{D_p}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{pj}^2} = -s\frac{\partial A(\omega)}{\partial\theta_{pj}}\sum_{i=0}^{R-1}\left\{\frac{r_{pj}^s\sin(\omega+s(\theta_{pj}+2\pi i))}{D_p}-\frac{r_{pj}^s\sin(\omega-s(\theta_{pj}+2\pi i))}{D_n}\right\}-\cdots$$

$$\cdots sA(\omega)\sum_{i=0}^{R-1}\left\{\frac{sr_{pj}^s\cos(\omega+s(\theta_{pj}+2\pi i))}{D_p}-\frac{2sr_{pj}^{2s}\sin^2(\omega+s(\theta_{pj}+2\pi i))}{D_p^2}+\cdots\right.$$

$$\left.\cdots\frac{sr_{pj}^s\cos(\omega-s(\theta_{pj}+2\pi i))}{D_n}-\frac{2sr_{pj}^{2s}\sin^2(\omega-s(\theta_{pj}+2\pi i))}{D_n^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{pk}\partial\theta_{pj}} = -s\frac{\partial A(\omega)}{\partial\theta_{pk}}\sum_{i=0}^{R-1}\left\{\frac{r_{pj}^s\sin(\omega+s(\theta_{pj}+2\pi i))}{D_p} - \frac{r_{pj}^s\sin(\omega-s(\theta_{pj}+2\pi i))}{D_n}\right\}$$

### G.3.2 Second partial derivatives of the IIR filter phase response

The non-zero second partial derivatives of the phase response are:

$$\begin{aligned}
\frac{\partial^2 P(\omega)}{\partial R_{0j}^2} =& \frac{\partial P(\omega)}{\partial R_{0j}}\frac{2\cos\omega - 2R_{0j}}{1-2R_{0j}\cos\omega + R_{0j}^2} \\
\frac{\partial^2 P(\omega)}{\partial R_{pj}^2} =& -s(s-1)R_{pj}^{s-2}\sum_{i=0}^{R-1}\frac{\sin(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\cdots \\
& -2s^2R_{pj}^{2s-2}\sum_{i=0}^{R-1}\frac{\sin(\omega-s2\pi i)\left[\cos(\omega-s2\pi i)-R_{pj}^s\right]}{\left[1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}\right]^2}
\end{aligned}$$

For convenience, write:

$$\begin{aligned}
\frac{\partial^2 P_{0N}(\omega)}{\partial r_{0j}^2} &= 0 \\
\frac{\partial^2 P_{0N}(\omega)}{\partial r_{0j}\theta_{0j}} &= 2\sin\theta_{0j}\sin\omega \\
\frac{\partial^2 P_{0N}(\omega)}{\partial \theta_{0j}^2} &= 2r_{0j}\cos\theta_{0j}\sin\omega \\
\frac{\partial^2 P_{0D}(\omega)}{\partial r_{0j}^2} &= 2 \\
\frac{\partial^2 P_{0D}(\omega)}{\partial r_{0j}\theta_{0j}} &= 2\sin\theta_{0j}\cos\omega \\
\frac{\partial^2 P_{0D}(\omega)}{\partial \theta_{0j}^2} &= 2r_{0j}\cos\theta_{0j}\cos\omega
\end{aligned}$$

then:

$$\begin{aligned}
\frac{\partial^2 P(\omega)}{\partial r_{0j}^2} =& \frac{\frac{\partial P_{0D}}{\partial r_{0j}}\frac{\partial P_{0N}}{\partial r_{0j}} + P_{0D}\frac{\partial^2 P_{0N}}{\partial r_{0j}^2} - \frac{\partial P_{0N}}{\partial r_{0j}}\frac{\partial P_{0D}}{\partial r_{0j}} - P_{0N}\frac{\partial^2 P_{0D}}{\partial r_{0j}^2}}{P_{0N}^2+P_{0D}^2}\cdots \\
& -\frac{\left[P_{0D}\frac{\partial P_{0N}}{\partial r_{0j}} - P_{0N}\frac{\partial P_{0D}}{\partial r_{0j}}\right]\left[2P_{0N}\frac{\partial P_{0N}}{\partial r_{0j}} + 2P_{0D}\frac{\partial P_{0D}}{\partial r_{0j}}\right]}{\left[P_{0N}^2+P_{0D}^2\right]^2} \\
\frac{\partial^2 P(\omega)}{\partial r_{0j}\partial\theta_{0j}} =& \frac{\frac{\partial P_{0D}}{\partial \theta_{0j}}\frac{\partial P_{0N}}{\partial r_{0j}} + P_{0D}\frac{\partial^2 P_{0N}}{\partial r_{0j}\partial\theta_{0j}} - \frac{\partial P_{0N}}{\partial \theta_{0j}}\frac{\partial P_{0D}}{\partial r_{0j}} - P_{0N}\frac{\partial^2 P_{0D}}{\partial r_{0j}\partial\theta_{0j}}}{P_{0N}^2+P_{0D}^2}\cdots \\
& -\frac{\left[P_{0D}\frac{\partial P_{0N}}{\partial r_{0j}} - P_{0N}\frac{\partial P_{0D}}{\partial r_{0j}}\right]\left[2P_{0N}\frac{\partial P_{0N}}{\partial \theta_{0j}} + 2P_{0D}\frac{\partial P_{0D}}{\partial \theta_{0j}}\right]}{\left[P_{0N}^2+P_{0D}^2\right]^2} \\
\frac{\partial^2 P(\omega)}{\partial \theta_{0j}^2} =& \frac{\frac{\partial P_{0D}}{\partial \theta_{0j}}\frac{\partial P_{0N}}{\partial \theta_{0j}} + P_{0D}\frac{\partial^2 P_{0N}}{\partial \theta_{0j}^2} - \frac{\partial P_{0N}}{\partial \theta_{0j}}\frac{\partial P_{0D}}{\partial \theta_{0j}} - P_{0N}\frac{\partial^2 P_{0D}}{\partial \theta_{0j}^2}}{P_{0N}^2+P_{0D}^2}\cdots \\
& -\frac{\left[P_{0D}\frac{\partial P_{0N}}{\partial \theta_{0j}} - P_{0N}\frac{\partial P_{0D}}{\partial \theta_{0j}}\right]\left[2P_{0N}\frac{\partial P_{0N}}{\partial \theta_{0j}} + 2P_{0D}\frac{\partial P_{0D}}{\partial \theta_{0j}}\right]}{\left[P_{0N}^2+P_{0D}^2\right]^2}
\end{aligned}$$

For convenience, write:

$$\begin{aligned}
\frac{\partial^2 P_{pN}(\omega)}{\partial r_{pj}^2} &= -2s(s-1)r_{pj}^{s-2}\cos[s(\theta_{pj}+2\pi i)]\sin\omega \\
\frac{\partial^2 P_{pN}(\omega)}{\partial r_{pj}\theta_{pj}} &= 2s^2r_{pj}^{s-1}\sin[s(\theta_{pj}+2\pi i)]\sin\omega \\
\frac{\partial^2 P_{pN}(\omega)}{\partial \theta_{pj}^2} &= 2s^2r_{pj}^{s}\cos[s(\theta_{pj}+2\pi i)]\sin\omega
\end{aligned}$$

$$\frac{\partial^2 P_{pD}(\omega)}{\partial r_{pj}^2} = -2s(s-1)\, r_{pj}^{s-2} \cos\left[s(\theta_{pj} + 2\pi i)\right] \cos\omega + 2s(2s-1)\, r_{pj}^{2s-2}$$
$$\frac{\partial^2 P_{pD}(\omega)}{\partial r_{pj}\theta_{pj}} = 2s^2 r_{pj}^{s-1} \sin\left[s(\theta_{pj} + 2\pi i)\right] \cos\omega$$
$$\frac{\partial^2 P_{pD}(\omega)}{\partial \theta_{pj}^2} = 2s^2 r_{pj}^{s} \cos\left[s(\theta_{pj} + 2\pi i)\right] \cos\omega$$

then:

$$\frac{\partial^2 P(\omega)}{\partial r_{pj}^2} = -\sum_{i=0}^{R-1} \left\{ \frac{\frac{\partial P_{pD}}{\partial r_{pj}}\frac{\partial P_{pN}}{\partial r_{pj}} + P_{pD}\frac{\partial^2 P_{pN}}{\partial r_{pj}^2} - \frac{\partial P_{pN}}{\partial r_{pj}}\frac{\partial P_{pD}}{\partial r_{pj}} - P_{pN}\frac{\partial^2 P_{pD}}{\partial r_{pj}^2}}{P_{pN}^2 + P_{pD}^2} \cdots \right.$$
$$\left. - \frac{\left[P_{pD}\frac{\partial P_{pN}}{\partial r_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial r_{pj}}\right]\left[2P_{pN}\frac{\partial P_{pN}}{\partial r_{pj}} + 2P_{pD}\frac{\partial P_{pD}}{\partial r_{pj}}\right]}{\left[P_{pN}^2 + P_{pD}^2\right]^2} \right\}$$
$$\frac{\partial^2 P(\omega)}{\partial r_{pj}\partial \theta_{pj}} = -\sum_{i=0}^{R-1} \left\{ \frac{\frac{\partial P_{pD}}{\partial \theta_{pj}}\frac{\partial P_{pN}}{\partial r_{pj}} + P_{pD}\frac{\partial^2 P_{pN}}{\partial r_{pj}\partial \theta_{pj}} - \frac{\partial P_{pN}}{\partial \theta_{pj}}\frac{\partial P_{pD}}{\partial r_{pj}} - P_{pN}\frac{\partial^2 P_{pD}}{\partial r_{pj}\partial \theta_{pj}}}{P_{pN}^2 + P_{pD}^2} \cdots \right.$$
$$\left. - \frac{\left[P_{pD}\frac{\partial P_{pN}}{\partial r_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial r_{pj}}\right]\left[2P_{pN}\frac{\partial P_{pN}}{\partial \theta_{pj}} + 2P_{pD}\frac{\partial P_{pD}}{\partial \theta_{pj}}\right]}{\left[P_{pN}^2 + P_{pD}^2\right]^2} \right\}$$
$$\frac{\partial^2 P(\omega)}{\partial \theta_{pj}^2} = -\sum_{i=0}^{R-1} \left\{ \frac{\frac{\partial P_{pD}}{\partial \theta_{pj}}\frac{\partial P_{pN}}{\partial \theta_{pj}} + P_{pD}\frac{\partial^2 P_{pN}}{\partial \theta_{pj}^2} - \frac{\partial P_{pN}}{\partial \theta_{pj}}\frac{\partial P_{pD}}{\partial \theta_{pj}} - P_{pN}\frac{\partial^2 P_{pD}}{\partial \theta_{pj}^2}}{P_{pN}^2 + P_{pD}^2} \cdots \right.$$
$$\left. - \frac{\left[P_{pD}\frac{\partial P_{pN}}{\partial \theta_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial \theta_{pj}}\right]\left[2P_{pN}\frac{\partial P_{pN}}{\partial \theta_{pj}} + 2P_{pD}\frac{\partial P_{pD}}{\partial \theta_{pj}}\right]}{\left[P_{pN}^2 + P_{pD}^2\right]^2} \right\}$$

### G.3.3 Second partial derivatives of the IIR filter group-delay response

The non-zero second partial derivatives of the group-delay response are:

$$\frac{\partial^2 T(\omega)}{\partial R_{0j}^2} = \frac{2\left(R_{0j}^3 \cos\omega - 3R_{0j}^2 + 3R_{0j}\cos\omega - \cos 2\omega\right)}{\left[1 - 2R_{0j}\cos\omega + R_{0j}^2\right]^3}$$
$$\frac{\partial^2 T(\omega)}{\partial R_{pj}^2} = -sR_{pj}^{s-2} \sum_{i=0}^{R-1} \left\{ \frac{(s+1)\cos(\omega - s2\pi i)\, R_{pj}^{4s} + 2\left[(s-1)\cos^2(\omega - s2\pi i) - (2s+1)\right] R_{pj}^{3s}}{\left[1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}\right]^3} + \cdots \right.$$
$$\left. \cdots \frac{6\cos(\omega - s2\pi i)\, R_{pj}^{2s} + 2\left[(2s-1) - (s+1)\cos^2(\omega - s2\pi i)\right] R_{pj}^s - (s-1)\cos(\omega - s2\pi i)}{\left[1 - 2R_{pj}^s \cos(\omega - s2\pi i) + R_{pj}^{2s}\right]^3} \right\}$$
$$\frac{\partial^2 T(\omega)}{\partial r_{0j}^2} = \frac{2\left[r_{0j}^3 \cos(\omega - \theta_{0j}) - 3r_{0j}^2 + 3r_{0j}\cos(\omega - \theta_{0j}) - \cos 2(\omega - \theta_{0j})\right]}{\left[1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right]^3} + \cdots$$
$$\frac{2\left[r_{0j}^3 \cos(\omega + \theta_{0j}) - 3r_{0j}^2 + 3r_{0j}\cos(\omega + \theta_{0j}) - \cos 2(\omega + \theta_{0j})\right]}{\left[1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right]^3}$$
$$\frac{\partial^2 T(\omega)}{\partial \theta_{0j}\partial r_{0j}} = \frac{\sin(\omega + \theta_{0j})\left[r_{0j}^4 + 2r_{0j}^3\cos(\omega + \theta_{0j}) - 6r_{0j}^2 + 2r_{0j}\cos(\omega + \theta_{0j}) + 1\right]}{\left[1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right]^3} - \cdots$$
$$\frac{\sin(\omega - \theta_{0j})\left[r_{0j}^4 + 2r_{0j}^3\cos(\omega - \theta_{0j}) - 6r_{0j}^2 + 2r_{0j}\cos(\omega - \theta_{0j}) + 1\right]}{\left[1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right]^3}$$
$$\frac{\partial^2 T(\omega)}{\partial \theta_{0j}^2} = -\frac{r_{0j}\left(r_{0j}^2 - 1\right)\left[r_{0j}^2\cos(\omega - \theta_{0j}) - 2r_{0j}\left(1 + \sin^2(\omega - \theta_{0j})\right) + \cos(\omega - \theta_{0j})\right]}{\left[1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right]^3} - \cdots$$
$$\frac{r_{0j}\left(r_{0j}^2 - 1\right)\left[r_{0j}^2\cos(\omega + \theta_{0j}) - 2r_{0j}\left(1 + \sin^2(\omega + \theta_{0j})\right) + \cos(\omega + \theta_{0j})\right]}{\left[1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right]^3}$$

$$
\begin{aligned}
\frac{\partial^2 T(\omega)}{\partial r_{pj}^2} = -s \sum_{i=1}^{R-1} r_{pj}^{s-2} \Bigg\{ & \frac{2r_{pj}^s \left(s\left(r_{pj}^{2s}-1\right) - \left(r_{pj}^{2s}+1\right)\right) \cos^2\left(\omega + s\left(\theta_{pj}+2\pi i\right)\right)}{D_p^3} + \cdots \\
\cdots & \frac{\left(s\left(r_{pj}^{4s}-1\right) + \left(r_{pj}^{4s}+6r_{pj}^{2s}+1\right)\right) \cos\left(\omega + s\left(\theta_{pj}+2\pi i\right)\right) - 4sr_{pj}^s\left(r_{pj}^{2s}-1\right) - 2r_{pj}^s\left(r_{pj}^{2s}+1\right)}{D_p^3} + \cdots \\
\cdots & \frac{2r_{pj}^s \left(s\left(r_{pj}^{2s}-1\right) - \left(r_{pj}^{2s}+1\right)\right) \cos^2\left(\omega - s\left(\theta_{pj}+2\pi i\right)\right)}{D_n^3} + \cdots \\
\cdots & \frac{\left(s\left(r_{pj}^{4s}-1\right) + \left(r_{pj}^{4s}+6r_{pj}^{2s}+1\right)\right) \cos\left(\omega - s\left(\theta_{pj}+2\pi i\right)\right) - 4sr_{pj}^s\left(r_{pj}^{2s}-1\right) - 2r_{pj}^s\left(r_{pj}^{2s}+1\right)}{D_n^3} \Bigg\}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 T(\omega)}{\partial \theta_{pj} \partial r_{pj}} = s^2 \sum_{i=1}^{R-1} r_{pj}^{s-1} \Bigg\{ & \frac{\left[2r_{pj}^s\left(r_{pj}^{2s}+1\right)\cos\left(\omega - s\left(\theta_{pj}+2\pi i\right)\right) + \left(r_{pj}^{4s}-6r_{pj}^{2s}+1\right)\right]\sin\left(\omega - s\left(\theta_{pj}+2\pi i\right)\right)}{D_n^3} - \cdots \\
\cdots & \frac{\left[2r_{pj}^s\left(r_{pj}^{2s}+1\right)\cos\left(\omega + s\left(\theta_{pj}+2\pi i\right)\right) + \left(r_{pj}^{4s}-6r_{pj}^{2s}+1\right)\right]\sin\left(\omega + s\left(\theta_{pj}+2\pi i\right)\right)}{D_p^3} \Bigg\}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 T(\omega)}{\partial \theta_{pj}^2} = s^2 \sum_{i=1}^{R-1} r_{pj}^s \left(r_{pj}^{2s}-1\right) \Bigg\{ & \frac{2r_{pj}^s \cos^2\left(\omega + s\left(\theta_{pj}+2\pi i\right)\right) + \left(r_{pj}^{2s}+1\right)\cos\left(\omega + s\left(\theta_{pj}+2\pi i\right)\right) - 4r_{pj}^s}{D_p^3} + \cdots \\
\cdots & \frac{2r_{pj}^s \cos^2\left(\omega - s\left(\theta_{pj}+2\pi i\right)\right) + \left(r_{pj}^{2s}+1\right)\cos\left(\omega - s\left(\theta_{pj}+2\pi i\right)\right) - 4r_{pj}^s}{D_n^3} \Bigg\}
\end{aligned}
$$

The results for group delay were calculated with the assistance of the *Maxima* [215] 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, partial derivatives and Hessian of the phase response of an IIR filter and is exercised by the test script *iirP_test.m*.

**!!! WARNING !!! The *iirA*, *iirP* and *iirT* functions do not attempt to handle the discontinuity and non-differentiability of the properties of a zero at $z = 1$.**

The Octave function *iirA_parallel.m* uses the *parcellfun* function included in the Octave-Forge *parallel* package [139] 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 H

# Gradient of the IIR filter amplitude response with respect to frequency

This section describes the derivatives with respect to angular frequency of the amplitude response, $A(\omega)$, of an IIR filter expressed in gain-pole-zero form. This section derives formulas for $\frac{\partial A(\omega)}{\partial \omega}$, $\frac{\partial^2 A(\omega)}{\partial \omega^2}$ 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:

$$A^2(\omega) = K^2 \frac{\prod_{j=1}^{U} \left\{1 - 2R_{0j}\cos\omega + R_{0j}^2\right\} \prod_{j=1}^{\frac{M}{2}} \left\{\left(1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right)\left(1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right)\right\}}{\prod_{j=1}^{V} \left\{1 - 2R_{pj}\cos\omega + R_{pj}^2\right\} \prod_{j=1}^{\frac{Q}{2}} \left\{\left(1 - 2r_{pj}\cos(\omega - \theta_{pj}) + r_{pj}^2\right)\left(1 - 2r_{pj}\cos(\omega + \theta_{pj}) + r_{pj}^2\right)\right\}}$$

The gradient of the amplitude response with respect to angular frequency is given by:

$$\begin{aligned}
\frac{1}{A(\omega)}\frac{\partial A(\omega)}{\partial \omega} = & \sum_{j=1}^{U} \frac{R_{0j}\sin\omega}{1 - 2R_{0j}\cos\omega + R_{0j}^2} - \sum_{j=1}^{V} \frac{R_{pj}\sin\omega}{1 - 2R_{pj}\cos\omega + R_{pj}^2} \cdots \\
& + \sum_{j=1}^{\frac{M}{2}} \frac{r_{0j}\sin(\omega - \theta_{0j})}{1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2} - \sum_{j=1}^{\frac{Q}{2}} \frac{r_{pj}\sin(\omega - \theta_{pj})}{1 - 2r_{pj}\cos(\omega - \theta_{pj}) + r_{pj}^2} \cdots \\
& + \sum_{j=1}^{\frac{M}{2}} \frac{r_{0j}\sin(\omega + \theta_{0j})}{1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2} - \sum_{j=1}^{\frac{Q}{2}} \frac{r_{pj}\sin(\omega + \theta_{pj})}{1 - 2r_{pj}\cos(\omega + \theta_{pj}) + r_{pj}^2}
\end{aligned}$$

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{\left(1 - R_{0j}^2\right)\sin\omega}{\left(1 - 2R_{0j}\cos\omega + R_{0j}^2\right)^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{\left(1 - R_{pj}^2\right)\sin\omega}{\left(1 - 2R_{pj}\cos\omega + R_{pj}^2\right)^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{\left(1 - r_{0j}^2\right)\sin(\omega - \theta_{0j})}{\left(1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right)^2} + \frac{\left(1 - r_{0j}^2\right)\sin(\omega + \theta_{0j})}{\left(1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right)^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}\left(1 + r_{0j}^2\right)\cos(\omega - \theta_{0j}) - 2r_{0j}^2}{\left(1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right)^2} + \frac{r_{0j}\left(1 + r_{0j}^2\right)\cos(\omega + \theta_{0j}) - 2r_{0j}^2}{\left(1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right)^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{\left(1 - r_{pj}^2\right)\sin(\omega - \theta_{pj})}{\left(1 - 2r_{pj}\cos(\omega - \theta_{pj}) + r_{pj}^2\right)^2} - \frac{\left(1 - r_{pj}^2\right)\sin(\omega + \theta_{pj})}{\left(1 - 2r_{pj}\cos(\omega + \theta_{pj}) + r_{pj}^2\right)^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}\left(1 + r_{pj}^2\right)\cos(\omega - \theta_{pj}) - 2r_{pj}^2}{\left(1 - 2r_{pj}\cos(\omega - \theta_{pj}) + r_{pj}^2\right)^2} - \frac{r_{pj}\left(1 + r_{pj}^2\right)\cos(\omega + \theta_{pj}) - 2r_{pj}^2}{\left(1 - 2r_{pj}\cos(\omega + \theta_{pj}) + r_{pj}^2\right)^2}
\end{aligned}$$

The second derivative of the amplitude response with respect to angular frequency is given by:

$$
\begin{aligned}
-\left[\frac{1}{A(\omega)}\frac{\partial A(\omega)}{\partial \omega}\right]^2 + \frac{1}{A(\omega)}\frac{\partial^2 A(\omega)}{\partial \omega^2} = & \sum_{j=1}^{U}\frac{R_{0j}\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2} - \sum_{j=1}^{U}\frac{2R_{0j}^2\sin^2\omega}{\left[1-2R_{0j}\cos\omega+R_{0j}^2\right]^2}\cdots \\
& -\sum_{j=1}^{V}\frac{R_{pj}\cos\omega}{1-2R_{pj}\cos\omega+R_{pj}^2} + \sum_{j=1}^{V}\frac{2R_{pj}^2\sin^2\omega}{\left[1-2R_{pj}\cos\omega+R_{pj}^2\right]^2}\cdots \\
& +\sum_{j=1}^{\frac{M}{2}}\frac{r_{0j}\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2} - \sum_{j=1}^{\frac{M}{2}}\frac{2r_{0j}^2\sin^2(\omega-\theta_{0j})}{\left[1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2\right]^2}\cdots \\
& -\sum_{j=1}^{\frac{Q}{2}}\frac{r_{pj}\cos(\omega-\theta_{pj})}{1-2r_{pj}\cos(\omega-\theta_{pj})+r_{pj}^2} + \sum_{j=1}^{\frac{Q}{2}}\frac{2r_{pj}^2\sin^2(\omega-\theta_{pj})}{\left[1-2r_{pj}\cos(\omega-\theta_{pj})+r_{pj}^2\right]^2}\cdots \\
& +\sum_{j=1}^{\frac{M}{2}}\frac{r_{0j}\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2} - \sum_{j=1}^{\frac{M}{2}}\frac{2r_{0j}^2\sin^2(\omega+\theta_{0j})}{\left[1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2\right]^2}\cdots \\
& -\sum_{j=1}^{\frac{Q}{2}}\frac{r_{pj}\cos(\omega+\theta_{pj})}{1-2r_{pj}\cos(\omega+\theta_{pj})+r_{pj}^2} + \sum_{j=1}^{\frac{Q}{2}}\frac{2r_{pj}^2\sin^2(\omega+\theta_{pj})}{\left[1-2r_{pj}\cos(\omega+\theta_{pj})+r_{pj}^2\right]^2}
\end{aligned}
$$

# Appendix I

# Allpass filter frequency response

## I.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

$$D(z) = \left\{ \prod_{k=1}^{V} z - R_{pk} \right\} \left\{ \prod_{k=1}^{\frac{Q}{2}} \left( z - r_{pk} e^{\imath \theta_{pk}} \right) \left( z - r_{pk} e^{-\imath \theta_{pk}} \right) \right\}$$

where $R_{pk}$ are the real zeros and $r_{pk} e^{\pm \imath \theta_{pk}}$ are the complex conjugate zeros of of $D(z)$.

The all-pass filter transfer function with decimation factor, $R$, is:

$$\begin{aligned} A(z) &= z^{-R(V+Q)} \frac{D\left(z^{-R}\right)}{D\left(z^{R}\right)} \\ &= \left\{ \prod_{k=1}^{V} \frac{z^{-R} - R_{pk}}{1 - R_{pk} z^{-R}} \right\} \left\{ \prod_{k=1}^{\frac{Q}{2}} \frac{z^{-R} - r_{pk} e^{\imath \theta_{pk}}}{1 - r_{pk} e^{-\imath \theta_{pk}} z^{-R}} \right\} \left\{ \prod_{k=1}^{\frac{Q}{2}} \frac{z^{-R} - r_{pk} e^{-\imath \theta_{pk}}}{1 - r_{pk} e^{\imath \theta_{pk}} z^{-R}} \right\} \end{aligned}$$

The squared-magnitude response of $A(z)$ is $\left|A\left(e^{\imath \omega}\right)\right|^2 = 1$. The phase response of $A(z)$ is

$$\begin{aligned} P(\omega) = &- \sum_{k=1}^{V} \left\{ \arctan \frac{R_{pk} \sin R\omega}{1 - R_{pk} \cos R\omega} + \arctan \frac{\sin R\omega}{\cos R\omega - R_{pk}} \right\} \cdots \\ &- \sum_{k=1}^{\frac{Q}{2}} \left\{ \arctan \frac{r_{pk} \sin\left(R\omega + \theta_{pk}\right)}{1 - r_{pk} \cos\left(R\omega + \theta_{pk}\right)} - \arctan \frac{\sin R\omega + r_{pk} \sin \theta_{pk}}{\cos R\omega - r_{pk} \cos \theta_{pk}} \right\} \cdots \\ &- \sum_{k=1}^{\frac{Q}{2}} \left\{ \arctan \frac{r_{pk} \sin\left(R\omega - \theta_{pk}\right)}{1 - r_{pk} \cos\left(R\omega - \theta_{pk}\right)} - \arctan \frac{\sin R\omega - r_{pk} \sin \theta_{pk}}{\cos R\omega - r_{pk} \cos \theta_{pk}} \right\} \end{aligned}$$

The partial derivatives of the phase response are:

$$\begin{aligned} \frac{\partial P(\omega)}{\partial R_{pk}} &= -\frac{2 \sin R\omega}{R_{pk}^2 - 2 R_{pk} \cos R\omega + 1} \\ \frac{\partial P(\omega)}{\partial r_{pk}} &= -\frac{2 \sin\left(R\omega - \theta_{pk}\right)}{r_{pk}^2 - 2 r_{pk} \cos\left(R\omega - \theta_{pk}\right) + 1} - \frac{2 \sin\left(R\omega + \theta_{pk}\right)}{r_{pk}^2 - 2 r_{pk} \cos\left(R\omega + \theta_{pk}\right) + 1} \\ \frac{\partial P(\omega)}{\partial \theta_{pk}} &= -\frac{2 r_{pk}^2 - 2 r_{pk} \cos\left(R\omega - \theta_{pk}\right)}{r_{pk}^2 - 2 r_{pk} \cos\left(R\omega - \theta_{pk}\right) + 1} + \frac{2 r_{pk}^2 - 2 r_{pk} \cos\left(R\omega + \theta_{pk}\right)}{r_{pk}^2 - 2 r_{pk} \cos\left(R\omega + \theta_{pk}\right) + 1} \end{aligned}$$

The second partial derivatives of the phase response (on the diagonal of the Hessian matrix only) are:

$$\frac{\partial^2 P(\omega)}{\partial R_{pk}^2} = \frac{4\left[R_{pk} - \cos R\omega\right] \sin R\omega}{\left[R_{pk}^2 - 2 R_{pk} \cos R\omega + 1\right]^2}$$

$$\frac{\partial^2 P(\omega)}{\partial r_{pk}^2} = \frac{4\left[r_{pk} - \cos(R\omega - \theta_{pk})\right]\sin(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^2} + \frac{4\left[r_{pk} - \cos(R\omega + \theta_{pk})\right]\sin(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^2}$$

$$\begin{aligned}\frac{\partial^2 P(\omega)}{\partial \theta_{pk}^2} = & \frac{2r_{pk}\sin(R\omega - \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1} - \frac{4\left[r_{pk}^2 - r_{pk}\cos(R\omega - \theta_{pk})\right] r_{pk}\sin(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^2} \cdots \\ & + \frac{2r_{pk}\sin(R\omega + \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1} - \frac{4\left[r_{pk}^2 - r_{pk}\cos(R\omega + \theta_{pk})\right] r_{pk}\sin(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^2}\end{aligned}$$

The second partial derivatives of the phase response (on the diagonal of the Hessian matrix only) are:

$$\frac{\partial^2 P(\omega)}{\partial R_{pk}^2} = \frac{4\left[R_{pk} - \cos R\omega\right]\sin R\omega}{\left[R_{pk}^2 - 2R_{pk}\cos R\omega + 1\right]^2}$$

$$\frac{\partial^2 P(\omega)}{\partial r_{pk}^2} = \frac{4\left[r_{pk} - \cos(R\omega - \theta_{pk})\right]\sin(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^2} + \frac{4\left[r_{pk} - \cos(R\omega + \theta_{pk})\right]\sin(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^2}$$

$$\begin{aligned}\frac{\partial^2 P(\omega)}{\partial \theta_{pk}^2} = & \frac{2r_{pk}\sin(R\omega - \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1} - \frac{4\left[r_{pk}^2 - r_{pk}\cos(R\omega - \theta_{pk})\right] r_{pk}\sin(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^2} \cdots \\ & + \frac{2r_{pk}\sin(R\omega + \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1} - \frac{4\left[r_{pk}^2 - r_{pk}\cos(R\omega + \theta_{pk})\right] r_{pk}\sin(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^2}\end{aligned}$$

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*.

## I.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 I.1. The group delay of the allpass filter is:

$$\begin{aligned}T(\omega) = & -R\sum_{k=1}^{V}\frac{R_{pk}^2 - 1}{R_{pk}^2 - 2R_{pk}\cos R\omega + 1}\cdots \\ & -R\sum_{k=1}^{\frac{Q}{2}}\frac{r_{pk}^2 - 1}{r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1}\cdots \\ & -R\sum_{k=1}^{\frac{Q}{2}}\frac{r_{pk}^2 - 1}{r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1}\end{aligned}$$

The partial derivatives of the group delay with respect to the coefficients are:

$$\frac{\partial T(\omega)}{\partial R_{pk}} = 2R\frac{\left(R_{pk}^2 + 1\right)\cos R\omega - 2R_{pk}}{\left[R_{pk}^2 - 2R_{pk}\cos R\omega + 1\right]^2}$$

$$\frac{\partial T(\omega)}{\partial r_{pk}} = 2R\frac{\left(r_{pk}^2 + 1\right)\cos(R\omega + \theta_{pk}) - 2r_{pk}}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^2} + 2R\frac{\left(r_{pk}^2 + 1\right)\cos(R\omega - \theta_{pk}) - 2r_{pk}}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^2}$$

$$\frac{\partial T(\omega)}{\partial \theta_{pk}} = 2R\frac{r_{pk}\left(r_{pk}^2 - 1\right)\sin(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^2} - 2R\frac{r_{pk}\left(r_{pk}^2 - 1\right)\sin(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^2}$$

The second partial derivatives of the group delay response (on the diagonal of the Hessian matrix only) are:

$$\frac{\partial^2 T(\omega)}{\partial R_{pk}^2} = -4R\frac{R_{pk}^3\cos R\omega - 3R_{pk}^2 + 3R_{pk}\cos R\omega + 1 - 2\cos^2 R\omega}{\left[R_{pk}^2 - 2R_{pk}\cos R\omega + 1\right]^3}$$

$$
\begin{aligned}
\frac{\partial^2 T(\omega)}{\partial r_{pk}^2} = & -4R\frac{r_{pk}^3\cos(R\omega+\theta_{pk}) - 3r_{pk}^2 + 3r_{pk}\cos(R\omega+\theta_{pk}) + 1 - 2\cos^2(R\omega+\theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega+\theta_{pk}) + 1\right]^3}\cdots \\
& -4R\frac{r_{pk}^3\cos(R\omega-\theta_{pk}) - 3r_{pk}^2 + 3r_{pk}\cos(R\omega-\theta_{pk}) + 1 - 2\cos^2(R\omega-\theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega-\theta_{pk}) + 1\right]^3} \\
\frac{\partial^2 T(\omega)}{\partial \theta_{pk}^2} = & \ 2R\frac{r_{pk}^5\cos(R\omega+\theta_{pk}) - 2r_{pk}^2\left(r_{pk}^2-1\right)\left[1+\sin^2(R\omega+\theta_{pk})\right] - r_{pk}\cos(R\omega+\theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega+\theta_{pk}) + 1\right]^3}\cdots \\
& +2R\frac{r_{pk}^5\cos(R\omega-\theta_{pk}) - 2r_{pk}^2\left(r_{pk}^2-1\right)\left[1+\sin^2(R\omega-\theta_{pk})\right] - r_{pk}\cos(R\omega-\theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega-\theta_{pk}) + 1\right]^3}
\end{aligned}
$$

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 J

# Gradients of the state variable filter frequency response

Section 1.9.4 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 $\alpha$, $\beta$, $\gamma$ and $\delta$ 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).

## J.1 Gradients of the state variable filter complex frequency response

If the components of the 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 $R = (e^{\imath\omega} I - A)^{-1}$, the gradients of the complex frequency response are:

$$\begin{aligned}
\frac{dH}{dx} &= \frac{dC}{dx}RB + CR\frac{dA}{dx}RB + CR\frac{dB}{dx} + \frac{dD}{dx} \\
\frac{d^2H}{dxd\omega} &= -\imath e^{\imath\omega}\left[\frac{dC}{dx}RRB + CR\frac{dA}{dx}RRB + CRR\frac{dA}{dx}RB + CRR\frac{dB}{dx}\right] \\
\frac{d^2H}{dx^2} &= \frac{d^2C}{dx^2}RB + 2\frac{dC}{dx}R\frac{dA}{dx}RB + 2\frac{dC}{dx}R\frac{dB}{dx} + 2CR\frac{dA}{dx}R\frac{dA}{dx}RB + CR\frac{d^2A}{dx^2}RB + 2CR\frac{dA}{dx}R\frac{dB}{dx} + CR\frac{d^2B}{dx^2} + \frac{d^2D}{dx^2}
\end{aligned}$$

and

$$\begin{aligned}
\frac{d^3H}{dx^2d\omega} = -\imath e^{\imath\omega}\Bigg[&\frac{d^2C}{dx^2}RRB + 2\frac{dC}{dx}R\frac{dA}{dx}RRB + 2\frac{dC}{dx}RR\frac{dA}{dx}RB + 2\frac{dC}{dx}RR\frac{dB}{dx} + \cdots \\
&2CR\frac{dA}{dx}R\frac{dA}{dx}RRB + CR\frac{d^2A}{dx^2}RRB + 2CR\frac{dA}{dx}RR\frac{dA}{dx}RB + 2CR\frac{dA}{dx}RR\frac{dB}{dx} + \cdots \\
&2CRR\frac{dA}{dx}R\frac{dA}{dx}RB + CRR\frac{d^2A}{dx^2}RB + 2CRR\frac{dA}{dx}R\frac{dB}{dx} + CRR\frac{d^2B}{dx^2}\Bigg]
\end{aligned}$$

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 $\frac{d^2A}{dx^2} = \frac{d^2B}{dx^2} = \frac{d^2C}{dx^2} = \frac{d^2D}{dx^2} = 0$.

## J.2 State variable filter squared-magnitude response

The squared-magnitude response of a state variable filter is:

$$|H(z)|^2 = \Re H(z)^2 + \Im H(z)^2$$

The gradients of the squared-magnitude response of the filter with respect to the state variable coefficients, $\alpha$, $\beta$, etc., are:

$$\frac{\partial}{\partial x}|H(z)|^2 = 2\left[\Im H(z)\,\Im\frac{\partial H(z)}{\partial x} + \Re H(z)\,\Re\frac{\partial H(z)}{\partial x}\right]$$

where $x$ represents the components of the state variable matrixes. The diagonal of the Hessian matrix of the squared-magnitude (that is the second derivatives of the squared-magnitude) is:

$$\frac{\partial^2 |H(z)|^2}{\partial x^2} = 2\left[\left|\frac{\partial H(z)}{\partial x}\right|^2 + \Im H(z)\,\Im\frac{\partial^2 H(z)}{\partial x^2} + \Re H(z)\,\Re\frac{\partial^2 H(z)}{\partial x^2}\right]$$

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

$$\frac{\partial^2 H(z)}{\partial \beta^2} = \frac{\partial^2 H(z)}{\partial \gamma^2} = \frac{\partial^2 H(z)}{\partial \delta^2} = 0$$

## J.3 State variable filter phase response

The phase response, $P(z)$, of the state variable filter is:

$$P(z) = \arctan\frac{\Im H(z)}{\Re H(z)}$$

The gradients of the phase response of the filter with respect to the state variable coefficients, $\alpha$, $\beta$, etc., are given by:

$$|H(z)|^2\frac{\partial P(z)}{\partial x} = \Re H(z)\,\Im\frac{\partial H(z)}{\partial x} - \Im H(z)\,\Re\frac{\partial H(z)}{\partial x}$$

where $x$ represents the components of the state variable matrixes. The diagonal of the Hessian matrix of the phase (that is the second derivatives of the phase) is given by:

$$\frac{\partial |H(z)|^2}{\partial x}\frac{\partial P(z)}{\partial x} + |H(z)|^2\frac{\partial^2 P(z)}{\partial x^2} = \Re H(z)\,\Im\frac{\partial^2 H(z)}{\partial x^2} - \Im H(z)\,\Re\frac{\partial^2 H(z)}{\partial x^2}$$

## J.4 State variable filter group-delay response

The group delay response, $T(\omega)$, of the state variable filter is

$$T(\omega) = -\frac{\partial}{\partial \omega}\arctan\frac{\Im H(\omega)}{\Re H(\omega)}$$

so that

$$|H(\omega)|^2 T(\omega) = -\left[\Re H(\omega)\,\Im\frac{\partial H(\omega)}{\partial \omega} - \Im H(\omega)\,\Re\frac{\partial H(\omega)}{\partial \omega}\right]$$

where

$$\frac{\partial H(\omega)}{\partial \omega} = -\imath e^{\imath\omega}C\left(e^{\imath\omega}I-A\right)^{-2}B$$

The gradients of the group delay response with respect to the state variable coefficients, $\alpha$, $\beta$, etc., are given by:

$$2\left[\Re H(\omega)\,\Re\frac{\partial H(\omega)}{\partial \alpha} + \Im H(\omega)\,\Im\frac{\partial H(\omega)}{\partial \alpha}\right]T(\omega) + |H(\omega)|^2\frac{\partial T(\omega)}{\partial \alpha} = \cdots$$
$$-\left[\Re\frac{\partial H(\omega)}{\partial \alpha}\Im\frac{\partial H(\omega)}{\partial \omega} + \Re H(\omega)\,\Im\frac{\partial^2 H(\omega)}{\partial \alpha\partial \omega} - \Im\frac{\partial H(\omega)}{\partial \alpha}\Re\frac{\partial H(\omega)}{\partial \omega} - \Im H(\omega)\,\Re\frac{\partial^2 H(\omega)}{\partial \alpha\partial \omega}\right]$$

etc. where

$$\frac{\partial^2 H(\omega)}{\partial \alpha\partial \omega} = -\imath e^{\imath\omega}C\left(e^{\imath\omega}I-A\right)^{-1}\left[\left(e^{\imath\omega}I-A\right)^{-1}\frac{\partial A}{\partial \alpha} + \frac{\partial A}{\partial \alpha}\left(e^{\imath\omega}I-A\right)^{-1}\right]\left(e^{\imath\omega}I-A\right)^{-1}B$$

$$\frac{\partial^2 H(\omega)}{\partial\beta\partial\omega} = -\imath e^{\imath\omega} C \left(e^{\imath\omega} I - A\right)^{-2}$$
$$\frac{\partial^2 H(\omega)}{\partial\gamma\partial\omega} = -\imath e^{\imath\omega} \left(e^{\imath\omega} I - A\right)^{-2} B$$
$$\frac{\partial^2 H(\omega)}{\partial\delta\partial\omega} = 0$$

The diagonal of the Hessian matrix of the group-delay (that is the second derivatives of the group-delay) is given by:

$$\begin{aligned}\frac{\partial^2 |H|^2}{\partial x^2} T + 2\frac{\partial |H|^2}{\partial x}\frac{\partial T}{\partial x} + |H|^2 \frac{\partial^2 T}{\partial x^2} = &- \Re\frac{\partial^2 H}{\partial x^2}\Im\frac{\partial H}{\partial\omega} - \Re\frac{\partial H}{\partial x}\Im\frac{\partial^2 H}{\partial x\partial\omega} - \Re\frac{\partial H}{\partial x}\Im\frac{\partial^2 H}{\partial x\partial\omega} - \Re H\Im\frac{\partial^3 H}{\partial x^2\partial\omega}\cdots \\ &+ \Im\frac{\partial^2 H}{\partial x^2}\Re\frac{\partial H}{\partial\omega} + \Im\frac{\partial H}{\partial x}\Re\frac{\partial^2 H}{\partial x\partial\omega} + \Im\frac{\partial H}{\partial x}\Re\frac{\partial^2 H}{\partial x\partial\omega} + \Im H\Re\frac{\partial^3 H}{\partial x^2\partial\omega}\end{aligned}$$

where $x$ represents the components of the state variable matrixes and:

$$\begin{aligned}\frac{\partial^3 H(\omega)}{\partial\alpha^2\partial\omega} = -\imath e^{\imath\omega} C \left(e^{\imath\omega} I - A\right)^{-1} \Bigg[ & 2\frac{\partial A}{\partial\alpha}\left(e^{\imath\omega} I - A\right)^{-2}\frac{\partial A}{\partial\alpha} + \cdots \\ & 2\left(e^{\imath\omega} I - A\right)^{-1}\frac{\partial A}{\partial\alpha}\left(e^{\imath\omega} I - A\right)^{-1}\frac{\partial A}{\partial\alpha} + \cdots \\ & 2\frac{\partial A}{\partial\alpha}\left(e^{\imath\omega} I - A\right)^{-1}\frac{\partial A}{\partial\alpha}\left(e^{\imath\omega} I - A\right)^{-1} + \cdots \\ & \left(e^{\imath\omega} I - A\right)^{-1}\frac{\partial^2 A}{\partial\alpha^2} + \frac{\partial^2 A}{\partial\alpha^2}\left(e^{\imath\omega} I - A\right)^{-1}\Bigg]\left(e^{\imath\omega} I - A\right)^{-1} B\end{aligned}$$

# Appendix K

# Constrained non-linear optimisation

The following largely follows *Ruszczynski* [9] with some contributions concerning heuristics from *Powell* [119] and *Nocedal* and *Wright* [92].

## K.1 Newton's method for a quadratic function

Consider a quadratic approximation, $f^k(x)$, to a function $f(x)$ evaluated at a point $x^k$

$$f^k(x) = f\left(x^k\right) + \nabla_x f\left(x^k\right)\left(x - x^k\right) + \frac{1}{2}\left(x - x^k\right)^\top \nabla^2_{xx} f\left(x^k\right)\left(x - x^k\right)$$

where $\nabla^2_{xx} f\left(x^k\right)$ is a positive-definite matrix[a] (the Hessian matrix). At an optimum point $\nabla_x f^k(x) = 0$ so, approximating the gradient with the first two terms of $f^k(x)$:

$$\nabla_x f\left(x^k\right) + \nabla^2_{xx} f\left(x^k\right)\left(x - x^k\right) = 0$$

Accordingly, given a pair $\left(x^k, f\left(x^k\right)\right)$, *Newton's method of tangents* gives the next approximation to the location of the optimum point as

$$x^{k+1} = x^k - \tau^k \left[\nabla^2_{xx} f\left(x^k\right)\right]^{-1} \nabla_x f\left(x^k\right)$$

where $\tau^k$ is a stepsize.

## K.2 Lagrange multipliers

Consider the problem

$$\begin{array}{rl} \textbf{minimise} & f(x) \\ \textbf{subject to} & g(x) = c \end{array}$$

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 *Nocedal* and *Wright* [92, p. 309] for a justification). The method of *Lagrange multipliers* defines an auxilliary function (or *Lagrangian*):

$$\mathcal{L}(x, \lambda) = f(x) - \lambda\left(g(x) - c\right)$$

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 $\lambda$ are the *Lagrange multipliers*.

[a] $Q \in \mathbb{R}^{n \times n}$ is positive-definite if $x^\top Q x > 0$ for all nonzero $x \in \mathbb{R}^n$

## K.3 The dual problem

Define the *dual function* as:

$$q(\lambda) = \min_x \mathcal{L}(x, \lambda)$$

The *dual problem* is:

$$\begin{array}{rl} \textbf{maximise} & q(\lambda) \\ \textbf{subject to} & \lambda \geq 0 \end{array}$$

*Nocedal* and *Wright* [92, Example 12.12] give an example of the dual problem of a quadratic programming problem:

$$\begin{array}{rl} \textbf{minimise} & a^\top x + \frac{1}{2} x^\top H x \\ \textbf{subject to} & Gx - b \geq 0 \end{array}$$

where $H$ is a symmetric positive-definite matrix. The Lagrangian function of the dual problem is:

$$q(\lambda) = a^\top x + \frac{1}{2} x^\top H x - \lambda^\top (Gx - b)$$

Since $H$ is positive definite and the Lagrangian is a strictly *convex*[b] quadratic function, the minimum occurs at $\nabla q(\lambda) = 0$:

$$a + Hx - G^\top \lambda = 0$$

Substituting for $x$:

$$q(\lambda) = -\frac{1}{2} \left(G^\top \lambda - a\right)^\top H^{-1} \left(G^\top \lambda - a\right) + b^\top \lambda$$

*Bertsekas* [36, Proposition 3.4.2] gives the following duality theorem:

1. If the primal problem has an optimal solution then the dual problem 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, $\tilde{\lambda} \geq 0$, $\tilde{\lambda}_j = 0$ over the set of active constraints and $\left(\tilde{x}, \tilde{\lambda}\right)$ is a solution of the dual problem.

A *primal-dual* or *interior-point* method solves the primal and dual problems simultaneously. At each iteration both sets of constraints are satisfied.

## K.4 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:

$$\begin{array}{rll} \textbf{minimise} & f(x), & x \in \mathbb{R}^n \\ \textbf{subject to} & g_i(x) \geq 0, & i = 1, \ldots, m \\ \textbf{and} & h_j(x) = 0, & j = 1, \ldots, p \end{array}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ , $g_i : \mathbb{R}^n \mapsto \mathbb{R}$ and $h_j : \mathbb{R}^n \mapsto \mathbb{R}$ are continuous differentiable functions. Define the Lagrangian of the problem as:

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \sum_{i=1}^{m} \lambda_i g_i(x) - \sum_{j=1}^{p} \mu_j h_j(x) \triangleq f(x) - \langle \lambda, g \rangle - \langle \mu, h \rangle$$

[b] A function, $f(x)$, is convex if $f(\alpha x + (1-\alpha) y) \leq \alpha f(x) + (1-\alpha) f(y)$, $\forall \alpha \in [0, 1]$

Define a saddle point of the Lagrangian as:

$$\left(\tilde{x}, \tilde{\lambda}, \tilde{\mu}\right) \quad \textbf{such that} \quad \mathcal{L}\left(\tilde{x}, \tilde{\lambda}, \tilde{\mu}\right) = \max_{\lambda,\mu \geq 0} \min_{x} \mathcal{L}\left(x, \lambda, \mu\right)$$

so that

$$\min_{x} \mathcal{L}\left(x, \lambda, \mu\right) \leq \max_{\lambda,\mu \geq 0} \min_{x} \mathcal{L}\left(x, \lambda, \mu\right) \leq \max_{\lambda,\mu \geq 0} \mathcal{L}\left(x, \lambda, \mu\right)$$

It can be shown that each such saddle point satisfies the *Karush-Kuhn-Tucker* conditions:

$$\begin{aligned} \nabla_x \mathcal{L}\left(\tilde{x}, \tilde{\lambda}, \tilde{\mu}\right) = \nabla_x f\left(\tilde{x}\right) - \left\langle \lambda, \nabla_x g\left(\tilde{x}\right)\right\rangle - \left\langle \mu, \nabla_x h\left(\tilde{x}\right)\right\rangle &= 0 \\ g_i\left(\tilde{x}\right) &\geq 0 \\ h_j\left(\tilde{x}\right) &= 0 \\ \tilde{\lambda}_i &\geq 0 \\ \tilde{\mu}_j &\geq 0 \\ \left\langle \lambda, g\left(\tilde{x}\right)\right\rangle &= 0 \end{aligned}$$

For the *cone* [c] $C = \mathbb{R}^m_+ \times \mathbb{R}^p$, then these conditions can be expressed

$$\begin{aligned} \nabla_x \mathcal{L}\left(\tilde{x}, \tilde{\lambda}, \tilde{\mu}\right) &= 0 \\ \begin{bmatrix} g\left(\tilde{x}\right) \\ h\left(\tilde{x}\right) \end{bmatrix} &\in N_C\left(\tilde{\lambda}, \tilde{\mu}\right) \end{aligned}$$

Furthermore, these conditions are necessary and sufficient for $\tilde{x}$ to be a minimiser of $f\left(x\right)$. In other words:

$$\boxed{\begin{array}{c} \tilde{x} \text{ is a minimiser} \\ \Uparrow \\ \left(\tilde{x}, \tilde{\lambda}, \tilde{\mu}\right) \text{ is a saddle point of } \mathcal{L}\left(x, \lambda, \mu\right) \\ \Downarrow \\ \left(\tilde{x}, \tilde{\lambda}, \tilde{\mu}\right) \text{ satisfy Karush-Kuhn-Tucker} \\ \Uparrow \\ f \text{ and } g \text{ are concave and } h \text{ is affine} \end{array}}$$

For any point $x$, the set of inequality constraints, $\mathcal{K}$, can be partitioned into a set of *active* constraints

$$\mathcal{A}\left(x\right) = \left\{i \mid g_i\left(x\right) = 0\right\}$$

and the corresponding set of *inactive* constraints, for which $i \notin \mathcal{A}\left(x\right)$. Following *Bertsekas* [36, 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".

## K.5 Constrained optimisation using Newton's method

Consider the non-linear optimization problem of Section K.4. Motivated by Newton's method, at a given point $\left(\overline{x}, \overline{\lambda}, \overline{\mu}\right)$ construct an approximation to the *Karush-Kuhn-Tucker* conditions that is linearised at $\bar{x}$:

$$\begin{aligned} \nabla_x \mathcal{L}\left(\overline{x}, \overline{\lambda}, \overline{\mu}\right) + \nabla^2_{xx} \mathcal{L}\left(\overline{x}, \overline{\lambda}, \overline{\mu}\right)\left(x - \overline{x}\right) &= 0 \\ \begin{bmatrix} g\left(\overline{x}\right) + \nabla_x g\left(\overline{x}\right)\left(x - \overline{x}\right) \\ h\left(\overline{x}\right) + \nabla_x h\left(\overline{x}\right)\left(x - \overline{x}\right) \end{bmatrix} &\in N_C\left(\lambda, \mu\right) \end{aligned}$$

These are the necessary conditions of optimality for *sequential quadratic programming*, shown in Algorithm K.1.

[c] *From* Ruszczynski *[9, 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 [9, p.28], the set $K^\circ \triangleq \left\{y \in \mathbb{R}^n : \left\langle y, x\right\rangle \leq 0 \; \forall x \in K\right\}$ is called the* polar cone *of $K$. From [9, p.37], for a convex closed set $X \subset \mathbb{R}^n$ and a point $x \in X$, the set $N_X\left(x\right) \triangleq \left[cone\left(X - x\right)\right]^\circ$ is called the* normal cone *to $X$ at $x$.*

**Algorithm K.1** The sequential quadratic programming method

At iteration $k$, given the current approximation of the solution $x^k$ and multipliers $\left(\lambda^k, \mu^k\right)$ solve the tangent quadratic programming problem:

$$\begin{aligned} \textbf{minimise} \quad & \left\langle \nabla_x f\left(x^k\right), d^k \right\rangle + \frac{1}{2}\left\langle {d^k}^\top, \nabla^2_{xx}\mathcal{L}\left(x^k, \lambda^k, \mu^k\right), d^k \right\rangle \\ \textbf{subject to} \quad & g\left(x^k\right) + \nabla_x g\left(x^k\right) d^k \leq 0 \\ & h\left(x^k\right) + \nabla_x h\left(x^k\right) d^k = 0 \end{aligned}$$

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.

## K.6 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:

$$\begin{aligned} \nabla_x f(x) - \sum_{i \in \mathcal{A}(x)} \lambda_i \nabla_x g_i(x) &= 0 \\ g_i(x) &= 0 \end{aligned}$$

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 $\left(x^k, \lambda^k\right)$, define the matrices:

$$\begin{aligned} \mathcal{H}_k &= \nabla^2_{xx}\mathcal{L}\left(x^k, \lambda^k\right) \\ \mathcal{B}_k &= \left\{\nabla_x g_i\left(x^k\right)\right\} \end{aligned}$$

where $i \in \mathcal{A}\left(x^k\right)$ and the columns of $\mathcal{B}_k$ correspond to the gradient vectors. When the above equations are linearised, Newton's method takes the form:

$$\begin{aligned} \left[\nabla_x f\left(x^k\right) - \mathcal{B}_k \lambda^{k+1}\right] + \mathcal{H}_k d^k &= 0 \\ g_i\left(x^k\right) + \mathcal{B}_k^\top d^k &= 0 \end{aligned}$$

where $\lambda^{k+1}$ and $g\left(x^k\right)$ are the vectors with coordinates $\lambda_i$ and $g_i\left(x^k\right)$, for $i \in \mathcal{A}\left(x^k\right)$. This system can be simplified to:

$$\begin{aligned} \mathcal{H}_k d^k - \mathcal{B}_k \lambda^{k+1} &= -\nabla_x f\left(x^k\right) \\ -\mathcal{B}_k^\top d^k &= g\left(x^k\right) \end{aligned}$$

which can be solved for the direction $d^k$ and the multipliers $\lambda^{k+1}$:

$$\lambda^{k+1} = -\left(\mathcal{B}_k^\top \mathcal{H}_k^{-1} \mathcal{B}_k\right)^{-1}\left[g\left(x^k\right) - \mathcal{B}_k^\top \mathcal{H}_k^{-1} \nabla_x f\left(x^k\right)\right] \tag{K.1}$$

$$d^k = -\mathcal{H}_k^{-1}\left[\nabla_x f\left(x^k\right) - \mathcal{B}_k \lambda^{k+1}\right] \tag{K.2}$$

## K.7 Quasi-Newton methods

Consider a modified version of the tangent quadratic programming problem

$$\textbf{minimise} \quad \nabla_x f\left(x^k\right) d^k + \frac{1}{2}{d^k}^\top \mathcal{W}_k d^k$$

$$\textbf{subject to} \quad \begin{aligned} g\left(x^k\right) + \nabla_x g\left(x^k\right) d^k &\leq 0 \\ h\left(x^k\right) + \nabla_x h\left(x^k\right) d^k &= 0 \end{aligned}$$

where the Hessian of the Lagrangian $\nabla^2_{xx}\mathcal{L}\left(x^k, \lambda^k, \mu^k\right)$ 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 $\mathcal{W}_k$.

### K.7.1 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

$$\gamma_k = \nabla_x \mathcal{L}\left(x^k + \delta_k, \lambda^k, \mu^k\right) - \nabla_x \mathcal{L}\left(x^k, \lambda^k, \mu^k\right)$$

When there are no constraints it is possible to choose the step-length, $\tau^k$, so that the scalar product $\delta_k^\top \gamma_k$ is positive. On the other hand, when there are constraints, it can happen that $\delta_k^\top \gamma_k$ is negative for all non-zero values of $\tau^k$. In this case the usual methods for updating $\mathcal{W}_k$ would fail to make the update positive definite. *Powell* [119, p.148] proposes updating $\mathcal{W}_k$ as follows. First replace $\gamma_k$ with

$$\eta_k = \theta_k \gamma_k + (1 - \theta_k)\, \mathcal{W}_k \delta_k \ , \quad 0 \leq \theta_k \leq 1$$

where

$$\theta_k = \begin{cases} 1 & \delta_k^\top \gamma_k \geq 0.2 \delta_k^\top \mathcal{W}_k \delta_k \\ \frac{0.8 \delta_k^\top \mathcal{W}_k \delta_k}{\delta_k^\top \mathcal{W}_k \delta_k - \delta_k^\top \gamma_k} & \delta_k^\top \gamma_k < 0.2 \delta_k^\top \mathcal{W}_k \delta_k \end{cases} \tag{K.3}$$

The factor of 0.2 was chosen empirically. Now update $\mathcal{W}_k$ with the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) formula

$$W_{k+1} = \mathcal{W}_k - \frac{\mathcal{W}_k \delta_k \left(\mathcal{W}_k \delta_k\right)^\top}{\delta_k^\top \mathcal{W}_k \delta_k} + \frac{\eta_k \eta_k^\top}{\delta_k^\top \eta_k}$$

and update $\mathcal{W}_k^{-1}$ with the *Sherman-Morrison* formula

$$W_{k+1}^{-1} = \mathcal{W}_k^{-1} + \left[1 + \frac{\eta_k^\top \mathcal{W}_k^{-1} \eta_k}{\delta_k^\top \eta_k}\right] \left(\frac{\delta_k \delta_k^\top}{\delta_k^\top \eta_k}\right) - \left[\frac{\mathcal{W}_k^{-1} \eta_k \delta_k^\top + \delta_k \eta_k^\top \mathcal{W}_k^{-1}}{\delta_k^\top \eta_k}\right]$$

Note that $\eta$ and $\delta$ are column vectors and $\delta\eta^\top$ is a *dyadic* or *Kronecker* product also written $\delta \otimes \eta$. *Nocedal* and *Wright* [92, Procedure 18.2] refer to Equation K.3 as *damped BFGS updating*.

### K.7.2 A modified Cholesky factorisation of the Hessian

*Bertsekas* [36, Appendix D] describes implementation of Newton's method by Cholesky factorisation of an approximation to the Hessian, $\nabla^2_{xx}\mathcal{L}(x^k) + \Delta_k$, that is positive definite.

Algorithm K.2 shows the Cholesky factorisation of a positive definite symmetric matrix, $W$, as $LL^\top$ 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 $y \in \mathbb{R}^i$, $y \neq 0$

$$y^\top W_i y = \begin{bmatrix} y^\top & 0 \end{bmatrix} W \begin{bmatrix} y \\ 0 \end{bmatrix} > 0$$

The scalar $\lambda_{ii}$ can be seen to be well-defined by setting $b = W_{i-1}^\top \beta_i$ and recalling that $W_i$ is positive definite:

$$0 < \begin{bmatrix} b^\top & -1 \end{bmatrix} W_i \begin{bmatrix} b \\ -1 \end{bmatrix} = b^\top W_{i-1} b - 2b^\top \beta_i + w_{ii}$$

**Algorithm K.2** Cholesky factorisation of an $n \times n$ positive-definite symmetric matrix, $W$

$L_1 = \sqrt{w_{1,1}}$
$W_1 = L_1 L_1^\top$
**for** $i = 2, \ldots, n$ **do**
$W_i = \begin{bmatrix} W_{i-1} & \beta_i \\ \beta_i^\top & w_{i,i} \end{bmatrix}$ where $\beta_i = \begin{bmatrix} w_{1,i} \\ \vdots \\ w_{i-1,i} \end{bmatrix}$
$W_i = L_i L_i^\top$ where $L_i = \begin{bmatrix} L_{i-1} & 0 \\ l_i^\top & \lambda_{ii} \end{bmatrix}$, $l_i = L_{i-1}^{-1}\beta_i$ and $\lambda_{i,i} = \sqrt{w_{i,i} - l_i^\top l_i}$
**end for**
$W = L_n L_n^\top$

$$
\begin{aligned}
&= w_{ii} - b^\top \beta_i \\
&= w_{ii} - \beta_i^\top W_{i-1}^{-1} \beta_i \\
&= w_{ii} - \left(L_{i-1}^{-1}\beta_i\right)^\top \left(L_{i-1}^{-1}\beta_i\right) \\
&= w_{ii} - l_i^\top l_i
\end{aligned}
$$

Note that matrix inversion of an arbitrary matrix is an $\mathcal{O}\left(n^3\right)$ operation whereas matrix inversion of a triangular matrix via backward or forward substitution is an $\mathcal{O}\left(n^2\right)$ operation[d].

We wish to add a diagonal correction, $\Delta_k$, to the Hessian matrix such that the resulting matrix is positive definite whilst simultaneously factoring $\nabla_{xx}^2 \mathcal{L}\left(x^k\right) + \Delta_k$. Firstly, fix two positive scalars $\mu_1$ and $\mu_2$, where $\mu_1 < \mu_2$. The first column of the factor $L$ is given by

$$
\begin{aligned}
l_{11} &= \begin{cases} \sqrt{w_{11}} & \mu_1 < w_{11} \\ \sqrt{\mu_2} & otherwise \end{cases} \\
l_{i1} &= \frac{w_{i1}}{l_{11}} \quad i = 2, \ldots, n
\end{aligned}
$$

Given columns $1, 2, \ldots, j-1$ of $L$ the elements of the $j$-th column are

$$
\begin{aligned}
l_{jj} &= \begin{cases} \sqrt{w_{jj} - \sum_{m=1}^{j-1} l_{jm}^2} & \mu_1 < w_{jj} - \sum_{m=1}^{j-1} l_{jm}^2 \\ \sqrt{\mu_2} & otherwise \end{cases} \\
l_{ij} &= \frac{w_{ij} - \sum_{m=1}^{j-1} l_{jm} l_{im}}{l_{jj}} \quad i = j+1, \ldots, n
\end{aligned}
$$

This scheme can be used in a modified Newton's method, where at the $k$-th iteration, we add a diagonal correction, $\Delta^k$, to the Hessian, $\nabla_{xx}^2 \mathcal{L}\left(x^k\right)$, and simultaneously obtain a Cholesky factorisation of $\nabla_{xx}^2 \mathcal{L}\left(x^k\right) + \Delta^k$. The direction vector, $d^k$, is found by solving the triangular systems

$$
\begin{aligned}
L_k y &= -\nabla_x f\left(x^k\right) \\
L_k^\top d^k &= y
\end{aligned}
$$

The next point is found by

$$
x^{k+1} = x^k + \tau^k d^k
$$

where $\tau^k$ is a step size. The scalars $\mu_1$ and $\mu_2$ are selected as follows: at each iteration we find the maximum absolute value of the Hessian diagonal elements

$$
w^k = \max \ element \left\{\nabla_{xx}^2 \mathcal{L}\left(x^k\right)\right\}
$$

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 $r_2 = \frac{r_2}{5}$.

[d]The $LDL^\top$ factorisation of a symmetric matrix avoids taking the square-root. See [49, Algorithm 4.1.2].

### K.7.3 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* [49, 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 $U = [u_1, \ldots, u_m] \in \mathbb{R}^{m \times m}$ and $V = [v_1, \ldots, v_n] \in \mathbb{R}^{n \times n}$ such that

$$U^\top B V = \operatorname{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m \times n}$$

where $[u_1, \ldots, u_m]$ is a column partitioning of $U$, $p = \min(m, n)$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$. From *Golub and Van Loan* [49, Section 5.5.4], the *pseudo-inverse* of $B$ is defined to be $B^* = V \Sigma U^\top$ where

$$\Sigma = \operatorname{diag}\left(\frac{1}{\sigma_1}, \cdots, \frac{1}{\sigma_r}, 0, \ldots, 0\right) \in \mathbb{R}^{n \times m}$$

and $\operatorname{rank}(B) = r$. It is the unique minimal Frobenius norm[e] solution to the problem:

$$\min_{X \in \mathbb{R}^{n \times m}} \|BX - I_m\|_F$$

If $\operatorname{rank}(B) = n$, then $B^* = \left(B^\top B\right)^{-1} B^\top$, while if $m = n = \operatorname{rank}(B)$, 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*:

$$\begin{aligned} BXB &= B \\ XBX &= X \\ (BX)^\top &= BX \\ (XB)^\top &= XB \end{aligned}$$

These conditions amount to the requirement that $BB^*$ and $B^*B$ be orthogonal projections onto $\operatorname{range}(B)$ and $\operatorname{range}\left(B^\top\right)$ respectively[f].

*Wright* [183] 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 $\left(\tilde{x}, \tilde{\lambda}\right)$ 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 $w^\top \nabla_{xx} \mathcal{L}\left(\tilde{x}, \tilde{\lambda}\right) w \geq \sigma \|w\|^2$ for all $\tilde{\lambda}$ such that $\left(\tilde{x}, \tilde{\lambda}\right)$ satisfies the *Karush-Kuhn-Tucker* conditions and all $w \in \operatorname{null}\left(\nabla g_{\mathcal{A}}(\tilde{x})\right)$
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* [183] considers the linearised SQP method described in Section K.5. 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

$$\min_x \max_{\lambda_{\mathcal{A}} \geq 0} \langle \nabla f(\bar{x}), x - \bar{x} \rangle - \left\langle \lambda_{\mathcal{A}}, g(\bar{x}) + \nabla_x^\top g(\bar{x})(x - \bar{x}) \right\rangle + \frac{1}{2} \left\langle x - \bar{x}, \left[\nabla_{xx}^2 \mathcal{L}\left(\bar{x}, \bar{\lambda}_{\mathcal{A}}\right)\right](x - \bar{x}) \right\rangle - \frac{1}{2} \nu \|\lambda_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}\|^2$$

where $\nu$ is defined as

$$\nu(x, \lambda_{\mathcal{A}}) = \| \left(\nabla_x \mathcal{L}(x, \lambda_{\mathcal{A}}), g_{\mathcal{A}}(x), \langle \lambda_{\mathcal{A}}, g_{\mathcal{A}}(x) \rangle\right) \|$$

The optimality conditions for a candidate solution $\left(\tilde{x}, \tilde{\lambda}_{\mathcal{A}}\right)$ are likewise similar

$$\nabla_x f(\bar{x}) - \nabla g_{\mathcal{A}}(\bar{x}) \tilde{\lambda}_{\mathcal{A}} + \nabla_{xx} \mathcal{L}\left(\bar{x}, \bar{\lambda}_{\mathcal{A}}\right)(\tilde{x} - \bar{x}) = 0$$

[e] See Appendix B.1.
[f] See Appendix B.3.

$$
\begin{aligned}
g_{\mathcal{A}}(\bar{x}) + \nabla_x^{\top} g_{\mathcal{A}}(\bar{x})(\tilde{x} - \bar{x}) - \nu\left(\tilde{\lambda}_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}\right) &\geq 0 \\
\tilde{\lambda}_{\mathcal{A}} &\geq 0 \\
\left\langle \tilde{\lambda}_{\mathcal{A}}, g_{\mathcal{A}}(\bar{x}) + \nabla g_{\mathcal{A}}^{\top}(\bar{x})(\tilde{x} - \bar{x}) - \nu\left(\tilde{\lambda}_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}\right)\right\rangle &= 0
\end{aligned}
$$

*Wright* shows that for $\left(\bar{x}, \bar{\lambda}_{\mathcal{A}}\right)$ sufficiently close to a primal-dual solution $(x, \lambda_{\mathcal{A}})$ there is a unique solution $\left(\tilde{x}, \tilde{\lambda}_{\mathcal{A}}\right)$ for which $\| \left(\tilde{x} - \bar{x}, \tilde{\lambda}_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}\right) \| = \mathcal{O}(\nu)$. This solution satisfies the following linear system

$$
\begin{bmatrix} \nabla_{xx}\mathcal{L}\left(\bar{x}, \bar{\lambda}_{\mathcal{A}}\right) & -\nabla_x g_{\mathcal{A}}(\bar{x}) \\ -\nabla_x g_{\mathcal{A}}^{\top}(\bar{x}) & \nu I \end{bmatrix} \begin{bmatrix} x - \bar{x} \\ \lambda_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}} \end{bmatrix} = \begin{bmatrix} -\left(\nabla_x f(\bar{x}) - \nabla_x g_{\mathcal{A}}(\bar{x})\bar{\lambda}_{\mathcal{A}}\right) \\ g_{\mathcal{A}}(\bar{x}) \end{bmatrix}
$$
$$
\lambda_{\mathcal{N}} = 0
$$

### K.7.4 Bertsekas' modification to the Hessian

*Bertsekas* [36, p. 461] describes a modification to the Hessian that can ensure that the Hessian is positive definite and invertible. From the usual Newton system

$$
\begin{aligned}
\mathcal{H}_k d^k - \mathcal{B}_k \lambda^{k+1} &= -\nabla_x f\left(x^k\right) \\
-\mathcal{B}_k^{\top} d^k &= g\left(x^k\right)
\end{aligned}
$$

define a new matrix:

$$
\bar{\mathcal{H}_k} = \mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^{\top}
$$

where the scalar, $c^k$, is chosen so that $\bar{\mathcal{H}_k}$ is positive definite. *Bertsekas* points out that

$$
{d^k}^{\top} \mathcal{B}_k \mathcal{B}_k^{\top} d^k = \| g\left(x^k\right) \|^2
$$

is a constant for *equality* constraints. Consequently, the modified optimisation problem

$$
\begin{aligned}
\textbf{minimise} \quad & \nabla_x f\left(x^k\right) d^k + \frac{1}{2} {d^k}^{\top} \left(\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^{\top}\right) d^k \\
\textbf{subject to} \quad & -\mathcal{B}_k^{\top} d^k = g\left(x^k\right)
\end{aligned}
$$

has a larger minimum value but the location of that minimum is unchanged.

Given the scalar, $c^k$, write:

$$
c^k \mathcal{B}_k \mathcal{B}_k^{\top} d^k = -c^k \mathcal{B}_k g\left(x^k\right)
$$

Adding:

$$
\left(\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^{\top}\right) d^k - \mathcal{B}_k \lambda^{k+1} = -\left[\nabla_x f\left(x^k\right) + c^k \mathcal{B}_k g\left(x^k\right)\right]
$$

As in Section K.6:

$$
\begin{aligned}
\lambda^{k+1} &= -\left(\mathcal{B}_k^{\top}\left(\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^{\top}\right)^{-1} \mathcal{B}_k\right)^{-1} \left[g\left(x^k\right) - \mathcal{B}_k^{\top}\left(\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^{\top}\right)^{-1}\left(\nabla_x f\left(x^k\right) + c^k \mathcal{B}_k g\left(x^k\right)\right)\right] \\
d^k &= -\left(\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^{\top}\right)^{-1} \left[\left(\nabla_x f\left(x^k\right) + c^k \mathcal{B}_k g\left(x^k\right)\right) - \mathcal{B}_k \lambda^{k+1}\right]
\end{aligned}
$$

## K.8 Penalty and barrier methods

From *Bertsekas* [36, 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.

### K.8.1 Penalty functions

In the unconstrained case $\tau^k$ 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* [9, 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 $\varrho$ is larger than the "max" norm of the Lagrange multipliers $\left(\hat{\lambda}, \hat{\mu}\right)$ at the solution. Similarly, *Powell* [119, 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 $\tau^k$ satisfies

$$\Psi\left(x^k + \tau^k d^k, \Omega, \Upsilon\right) \leq \Psi\left(x^k, \Omega, \Upsilon\right)$$

$\Upsilon_i$ and $\Omega_k$ are defined below. This condition can be fulfilled if the function

$$\Phi\left(\tau^k\right) = \Psi\left(x^k + \tau^k d^k, \Omega, \Upsilon\right)$$

decreases initially when $\tau^k$ is made positive. *Powell* [119] asserts that this happens if $\mathcal{W}_k$ is positive definite and if

$$\begin{aligned}\Omega_i &\geq |\lambda_i| \quad i = 1, \ldots, m\\ \Upsilon_i &\geq |\mu_i| \quad i = 1, \ldots, p\end{aligned}$$

*Powell* [119, p.151] suggests $\Omega_i = |\lambda_i|$ for the first iteration and subsequently

$$\Omega_i = \max\left(|\lambda_i|, \frac{1}{2}\left[\overline{\Omega}_i + |\lambda_i|\right]\right)$$

where $\overline{\Omega}_i$ is the value of $\Omega_i$ used on the previous iteration. Similarly for $\Upsilon_i$. The line-search procedure given by *Powell* [119, p.152] is obscure but it appears to be a variation of the "two-slope" test that is also described by *Ruszczynski* [9, p. 216] and *Bersekas* [36, p. 28]. The test assumes that $\Phi'(0) < 0$. *Powell* [119, p.152] suggests that the line-search terminate when

$$\Phi(\hat{\tau}_k) \leq \Phi(0) + 0.1\tau^k\Phi'(0)$$

*Bertsekas* [36, 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

$$\begin{aligned}\textbf{minimise} \quad & f(x)\\ \textbf{subject to} \quad & h_i(x) = 0\\ & g_i(x) - z_i^2 = 0\end{aligned}$$

The Quadratic Penalty Function method (*Bertsekas* [36, Section 4.2.1] gives a series of unconstrained minimisations of the form

$$\min_{x,z}\bar{\mathcal{L}}_{c^k}\left(x, z, \mu^k, \lambda^k\right) = f(x) + \left\{\frac{c^k}{2}\|h(x)\|^2 - \left(\mu^k\right)^\top h(x)\right\} + \sum_{i=1}^{m}\left\{\frac{c^k}{2}\left|g_i(x) - z_i^2\right|^2 - \lambda_i^k\left(g_i(x) - z_i^2\right)\right\}$$

where $c^k < c^{k+1}$ and $c^k \to \infty$. Minimising the sum with respect to $z$ is equivalent to

$$\mathcal{L}_{c^k}\left(x, v, \mu^k, \lambda^k\right) = f(x) + \left\{\frac{c^k}{2}\|h(x)\| - \left(\mu^k\right)^\top h(x)\right\} + \min_{v_i \geq 0}\sum_{i=1}^{m}\left\{\frac{c^k}{2}\left|g_i(x) - v_i\right|^2 - \lambda_i^k\left(g_i(x) - v_i\right)\right\}$$

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 $\lambda_i + c^k\left(g_i(x) - \bar{v}_i\right)$ is zero) then

$$v_i^* = \max\left\{0, g_i(x) + \frac{\lambda_i^k}{c^k}\right\}$$

and

$$g_i(x) - v_i^* = \begin{cases} -\frac{\lambda_i^k}{c^k} & g_i(x) > -\frac{\lambda_i^k}{c^k} \\ g_i(x) & g_i(x) \leq -\frac{\lambda_i^k}{c^k} \end{cases}$$
$$= \min\left(g_i(x), -\frac{\lambda_i^k}{c^k}\right)$$

The Lagrangian summation term becomes

$$\sum_{i=1}^m \lambda_i^k \min\left(g_i(x), -\frac{\lambda_i^k}{c^k}\right) + \frac{c^k}{2}\left|\min\left(g_i(x), -\frac{\lambda_i^k}{c^k}\right)\right|^2 = \begin{cases} \frac{1}{2c^k}\sum_{i=1}^m \left[\left(\lambda_i^k + c^k g_i(x)\right)^2 - \left(\lambda_i^k\right)^2\right] & g_i(x) \leq -\frac{\lambda_i^k}{c^k} \\ -\frac{1}{2c^k}\sum_{i=1}^m \left(\lambda_i^k\right)^2 & g_i(x) > -\frac{\lambda_i^k}{c^k} \end{cases}$$

Substituting into the Lagrangian

$$\mathcal{L}_{c^k}\left(x, \mu^k, \lambda^k\right) = f(x) + \left\{\frac{c^k}{2}\|h(x)\| - \left(\mu^k\right)^\top h(x)\right\} + \frac{1}{2c^k}\sum_{i=1}^m \left\{\min\left[0, \left(\lambda_i^k + c^k g_i(x)\right)\right]^2 - \lambda_i^2\right\}$$

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

$$\lambda^{k+1} = \min\left(0, \lambda^k + c^k g(x)\right)$$

Unfortunately, difficulties arise because the Hessian matrix of $\mathcal{L}_{c^k}$ is discontinuous at the $x$ for which $g_i(x) = -\frac{\lambda_i^k}{c^k}$.

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)

$$\mathcal{L}_{c^k}\left(x, \lambda^k\right) = f(x) + \sum_{i=1}^m \frac{\lambda_j^k}{c_j^k}\psi\left(c_j^k g_j(x)\right)$$

with gradient

$$\nabla\mathcal{L}_{c^k}\left(x, \lambda^k\right) = \nabla f(x) - \sum_{i=1}^m \lambda_j^k g(x)\exp\left(-c_j^k g_j(x)\right)$$

The $\left\{c_j^k\right\}$ are a positive penalty parameter sequence for each $j$. The $\lambda^k$ are updated by

$$\lambda^{k+1} = \lambda^k \exp\left(c^k g(x)\right)$$

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 $\left\{w^k\right\}$ is a positive sequence with $w^k \leq w^{k+1}$

### K.8.2 Barrier functions

*Bertsekas* [36, Section 4.1], describes adding a so-called *barrier* function, $B(x)$, to the cost function. This function is continuous and goes to $\infty$ 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

$$S = \{x \in X \mid g_j(x) \geq 0,\ j = 1, \ldots, m\}$$

The two most common barrier functions are

$$B(x) = -\sum_{i=1}^{m} \ln\{g_i(x)\}$$
$$B(x) = \sum_{i=0}^{m} \frac{1}{g_i(x)}$$

The barrier method consists of finding

$$x^k \in \arg\min_{x \in S} \left\{ f(x) + \epsilon^k B(x),\ k = 0, 1, \ldots \right\}$$

where the sequence $\left\{\epsilon^k\right\}$ is defined by

$$0 < \epsilon^{k+1} < \epsilon^k \quad k = 0, 1, \ldots \quad \epsilon^k \to 0$$

## K.9 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.

### K.9.1 Line search with the Golden-Section

The *Golden Section* (see *Ruszczynski* [9, p.213] and *Bertsekas* [36, 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

$$\beta = \alpha + (1 - q)(\delta - \alpha)$$
$$\gamma = \delta - (1 - q)(\delta - \alpha)$$

where

$$q = \frac{-1 + \sqrt{5}}{2}$$
$$1 - q = q^2$$

Note that

$$\frac{\beta - \alpha}{\delta - \alpha} = 1 - q$$
$$\frac{\gamma - \alpha}{\delta - \alpha} = q$$
$$\frac{\gamma - \beta}{\delta - \beta} = 1 - q$$
$$etc.$$

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 [9, p.215]. First, search for a point $\beta > 0$ such that $\Phi(\beta) \geq \Phi(0)$, then quadratic interpolation gives

$$\tau^k = \frac{-\Phi'(0)\beta^2}{2\left[\Phi(\beta) - \Phi(0) - \Phi'(0)\beta\right]}$$

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 $\beta$ with $\tau^k$ 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

$$\gamma_k = \frac{\Phi(\alpha_k)\left[\beta_k^2 - (\tau^k)^2\right] + \Phi(\tau^k)\left[\alpha_k^2 - \beta_k^2\right] + \Phi(\beta_k)\left[(\tau^k)^2 - \alpha_k^2\right]}{2\left(\Phi(\alpha_k)\left[\beta_k - \tau^k\right] + \Phi(\tau^k)\left[\alpha_k - \beta_k\right] + \Phi(\beta_k)\left[\tau^k - \alpha_k\right]\right)}$$

The initial three points can be found by scanning the values $\Phi(0), \Phi(\tau_0), \ldots$ or $\Phi(0), \Phi(-\tau_0), \ldots$ depending on whether $\Phi(\tau_o) < \Phi(0)$.

### K.9.2 Inexact step-size selection

The previous sub-section describes a brute-force search for the best step-size. *Bertsekas* [36, 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 K.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, $\mathcal{W}_k$:

$$\left\langle \nabla_x f(x^k), d^k \right\rangle = -\left\langle \nabla_x f(x^k), \mathcal{W}_k^{-1} \nabla_x f(x^k) \right\rangle < 0$$

and $d^k$ is a descent direction.

*Bertsekas* [36, p. 29], recommends the use of the *Armijo* rule. Given an initial step size $\sigma > 0$ and $\beta \in (0, 1)$, choose $\tau^k$ to be the largest in $\{\sigma, \sigma\beta, \sigma\beta^2, \ldots\}$ so that

$$f(x^k + \tau^k d^k) \leq f(x^k) + c_1 \tau^k \left\langle \nabla_x f(x^k), d^k \right\rangle \tag{W1}$$

The Armijo rule reduces the step size geometrically. An implementation of line search using the *Armijo* rule is shown in Algorithm K.3.

**Algorithm K.3** Line search using the Armijo rule

```
β ∈ [0.1, 0.5]
σ ∈ [10^-5, 10^-1]
while f(x^k) − f(x^k + β^m d^k) ≥ −c_1 σ β^m ⟨∇_x f(x^k), d^k⟩ do
    m = m + 1
end while
τ = σ β^m
```

*Kim et al.* [131] claim improved efficiency with the following modification to the *Armijo* rule:

$$f(x^k + \tau^k d^k) \leq f(x^k) + c_1 \tau^k \left[\left\langle \nabla_x f(x^k), d^k \right\rangle + \frac{1}{2}\tau^k \left\langle d^k, \mathcal{W}_k d^k \right\rangle\right] \tag{K.4}$$

To rule out unacceptably short steps, the *Wolfe* step-size rules add the following to the *Armijo* rule (or $W1$):

$$\left\langle \nabla_x f(x^k + \tau^k d^k), d^k \right\rangle \geq c_2 \left\langle \nabla_x f(x^k), d^k \right\rangle \tag{W2}$$

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

$$\left|\left\langle \nabla_x f(x^k + \tau^k d^k), d^k \right\rangle\right| \leq \left|c_2 \left\langle \nabla_x f(x^k) \right\rangle, d^k\right| \tag{K.5}$$

**Algorithm K.4** Line search using Wolfe conditions

$R = 1, r = 0.5, \tau = 1, a = 0, b = \infty$
**while** $a \neq b$ **do**
  **if** $W1(\tau)$ **then**
    $a = \tau$
  **else**
    $b = \tau$
  **end if**
  **if** $W1(\tau) \wedge W2(\tau)$ **then**
    $b = \tau$
  **else**
    **if** $b = \infty$ **then**
      $\tau = \max(\tau, Ra)$
    **else**
      $\tau = \max((1-r)a + rb\ ,\ \min(\tau, ra + (1-r)b))$
    **end if**
  **end if**
**end while**

This stronger *Wolfe* condition does not allow the gradient to be too positive, excluding points that are far from the minimum. *Christianson* [17] gives Algorithm K.4 for line search using the Wolfe conditions.

The *Goldstein* conditions for step-size selection are similar to the *Wolfe* conditions:

$$f\left(x^k + \tau^k d^k\right) < f\left(x^k\right) + c_1 \tau^k \left\langle \nabla_x f\left(x^k\right), d^k \right\rangle \tag{G1}$$

$$f\left(x^k + \tau^k d^k\right) > f\left(x^k\right) + c_2 \tau^k \left\langle \nabla_x f\left(x^k\right), d^k \right\rangle \tag{G2}$$

where $0 < c_1 < c_2 < 1$. Usually $c_1 < 0.5$ and $c_2 = 1 - c_1$. *Christianson* gives Algorithm K.5 for line search using the *Goldstein* conditions. The first iteration in Algorithm K.5 must terminate because $f(x)$ is differentiable at $x^k$. The second iteration must

**Algorithm K.5** Line search using the Goldstein conditions

$\tau = 1, R = 2$
**while** $\neg G1(\tau)$ **do**
  $\tau = \frac{\tau}{R}$
**end while**
**while** $\neg G2(\tau)$ **do**
  $\tau = \tau R$
**end while**

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.

### K.9.3 Lanczos step-size selection

*Toh* [100] 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 $\mathcal{W}_k = L_k^\top L_k$. Let

$$\mathcal{B}_k = -{L_k^{-1}}^\top \Delta \mathcal{W}_k L_k^{-1}$$

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 $\lambda_1$ is the maximum eigenvalue of $\mathcal{B}_k$ and $\rho_k$ provides an upper bound on the corresponding step-size $\tau^k d^k$. *Toh* suggests that the required computation can be reduced by finding $\lambda_1$ 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 [49, Section 8.5] and have an extensive discussion of *Lanczos* methods [49, Chapter 9].

## K.10 Initial solution with the Goldfarb-Idnani algorithm

*Goldfarb* and *Idnani* [33] observe that the origin in the space of dual variables (ie: Lagrange multipliers) is always a feasible solution of the dual problem. In Algorithm K.6 I reproduce their dual algorithm for solving the tangent quadratic problem with linear constraints.

Some explanation of the notation is required. The problem statement is:

$$\begin{aligned} \textbf{minimise} \quad & f(x) = a^\top x + \frac{1}{2} x^\top H x \\ \textbf{subject to} \quad & g(x) \equiv G^\top x - b \geq 0 \end{aligned}$$

This is equivalent to the linearised optimisation problem defined in Section K.5. 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 $\mathcal{A} \subset \mathcal{K}$. 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

$$\begin{aligned} B^* &= \left(B^\top H^{-1} B\right)^{-1} B^\top H^{-1} \\ E &= H^{-1}\left(I - BB^*\right) \end{aligned}$$

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 [33], if $\hat{x}$ is a point in the $(n-q)$ dimensional manifold $\mathcal{M} = \left\{x \in \mathbb{R}^n \mid n_i^\top x = b_i, i \in \mathcal{A}\right\}$ 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

$$\begin{aligned} E\nabla f(\tilde{x}) &= H^{-1}\left(I - BB^*\right) B\lambda \\ &= H^{-1}\left(B - B\right)\lambda \\ &= 0 \end{aligned}$$

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 $E$:

$$\begin{aligned} & Ew = 0 \Leftrightarrow w = B\alpha \\ & E \textit{ is positive semidefinite} \\ & EWE = E \\ & B^* WE = 0 \\ & EE^+ = E^+ \end{aligned}$$

When justifying Algorithm K.6, *Goldfarb* and *Idnani* [33, 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

$$\begin{aligned} & g_p(x) < 0 \\ & g_i(x) = 0 \; \forall i \in \mathcal{A} \end{aligned}$$

**Algorithm K.6** The *Goldfarb-Idnani* dual algorithm [33]

0. *Find the unconstrained minimum of $f(x)$, then:*
   Set $x \leftarrow H^{-1}a$, $f \leftarrow \frac{1}{2}a^\top x$, $E \leftarrow H^{-1}$, $\mathcal{A} \leftarrow \emptyset$, $q \leftarrow 0$

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, $\tau_1$:*
   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

$$\tau_1 \leftarrow \min_{r_j > 0,\ j=1,\ldots,q} \left\{ \frac{\lambda_j^+(x)}{r_j} \right\} = \frac{\lambda_l^+(x)}{r_l}$$

   In *Step 2c* below, element $k \in \mathcal{K}$ corresponds to the $l$-th element in $\mathcal{A}$.

   ii. *Full step length, $\tau_2$:*
   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 $\tau_2 \leftarrow -\frac{g_p(x)}{d^\top n^+}$.

   iii. *Step length, $\tau$:*
   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 $x \leftarrow x + \tau d$, $f \leftarrow f + \tau d^\top n^+ \left(\frac{1}{2}\tau + \lambda_{q+1}^+\right)$, $\lambda^+ \leftarrow \lambda^+ + \tau \begin{pmatrix} -r \\ 1 \end{pmatrix}$.
   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*.

$$E^{+}\nabla_{x}f\left(x\right)=0$$
$$\lambda^{+}\left(x\right)\triangleq\left(B^{+}\right)^{*}\nabla_{x}f\left(x\right)\geq 0$$

*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 $g_{p}\left(x\right)\geq 0$ in $P\left(\mathcal{A}^{+}\right)$ by $g_{p}\left(x\right)\geq g_{p}\left(\hat{x}\right)$ i.e. by a parallel constraint which passes through $\hat{x}$. Let $x^{*}$ be the point on the manifold $\mathcal{M}^{+}=\left\{x\in\mathbb{R}^{n}\mid n_{i}^{\top}x=b_{i},i\in\mathcal{A}^{+}\right\}$ 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

$$E^{+}\nabla_{x}f\left(\tilde{x}\right)=0 \tag{K.6}$$
$$g_{i}\left(\tilde{x}\right)=0\,\forall i\in\mathcal{A} \tag{K.7}$$
$$\lambda^{+}\left(\tilde{x}\right)\triangleq\left(B^{+}\right)^{*}\nabla_{x}f\left(\tilde{x}\right)=\lambda^{+}\left(x\right)+\tau\begin{bmatrix}-r\\1\end{bmatrix} \tag{K.8}$$

where $r=B^{*}n^{+}$and $g_{p}\left(\tilde{x}\right)=g_{p}\left(x\right)+\tau d^{\top}n^{+}$.

*Proof:*The lemma follows from the properties of the V-triple $(x,\mathcal{A},p)$ and

$$\nabla_{x}f\left(\tilde{x}\right)=\nabla_{x}f\left(x\right)+\tau Hd \tag{K.9}$$

where

$$\begin{aligned}Hd&=HEn^{+}\\&=\left(I-BB^{*}\right)n^{+}\\&=n^{+}-Br\\&=\begin{bmatrix}B&n^{+}\end{bmatrix}\begin{bmatrix}-r\\1\end{bmatrix}\\&=B^{+}\begin{bmatrix}-r\\1\end{bmatrix}\end{aligned}$$

Recalling that $EB=H^{-1}\left(B-BB*B\right)=0$, the results follow from multiplying Equation K.9 on the left by $E^{+}$ and $B^{+}$.

It follows from this lemma that the point $\tilde{x}=x+\tau_{2}d$, where $\tau_{2}=-\frac{g_{p}(x)}{d^{\top}n^{+}}$, minimises the quadratic function $f\left(x\right)$ over $\mathcal{M}^{+}$ (since then $g_{p}\left(\tilde{x}\right)=0$). If $\lambda^{+}\left(\tilde{x}\right)\geq 0$ as well, then $\tilde{x}$ is an optimal solution to $P\left(\mathcal{A}^{+}\right)$ and $(\tilde{x},\mathcal{A}^{+})$ is an S-pair. If not, then Equation K.8 implies that there is a smallest value $\tau_{1}$ of $\tau$, $\tau_{1}<\tau_{2}$, such that some component of $\lambda^{+}\left(\tilde{x}\left(\tau\right)\right)<0$ for $\tau>\tau_{1}$. If the constraint, say $k\in\mathcal{A}$, corresponding to this component is dropped from the active set, then $\left(\tilde{x}\left(\tau_{1}\right),\mathcal{A}^{-},p\right)$, 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 $\tau=\min\left\{\tau_{1},\tau_{2}\right\}$ where

$$\tau_{1}=\min\left\{\min_{r_{j}>0\,,\,1\leq j\leq q}\left\{\frac{\lambda_{j}^{+}\left(x\right)}{r_{j}^{+}\left(x\right)}\right\},\infty\right\}$$
$$\tau_{2}=-\frac{g_{p}\left(x\right)}{d^{\top}n^{+}}$$

then

$$g_{P}\left(\tilde{x}\right)\geq g_{p}\left(x\right)$$
$$f\left(\tilde{x}\right)-f\left(x\right)=\tau d^{\top}n^{+}\left(\frac{1}{2}\tau+\lambda_{q+1}^{+}\left(x\right)\right)\geq 0$$

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,

$$d^\top n^+ = n^{+T} E n^+ = n^{+T} E H E n^+ = d^\top H d > 0$$

and $\tau \geq 0$. Hence, from Lemma 1, $g_P(\tilde{x}) \geq g_p(x)$. Also, from Taylor's theorem

$$f(\tilde{x}) - f(x) = \tau d^\top \nabla_x f(x) + \frac{1}{2}\tau^2 d^\top H d$$

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

$$d^\top \nabla_x f(x) = n^{+T} E \nabla_x f(x) = d^\top n^+ \lambda_{q+1}^+(x) \geq 0$$

The result follows by substitution. Moreover, as long as $\tau > 0$, $f(\tilde{x}) > f(x)$. From the definition of $\tau$ 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 $\left(\tilde{x}, \tilde{\mathcal{A}} \cup \{p\}\right)$ 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

$$\frac{\lambda_l(x)}{r_l} = \min_{r_j > 0\,,\, j=1,\ldots,q} \left\{ \frac{\lambda_j(x)}{r_j} \right\}, \quad l = j(k)$$

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 $n^{+T} d = r^\top B^\top d > 0$ and $B^\top d \geq 0$ 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* [33, Section 4] base their implementation on the *Cholesky* factorisation

$$H = LL^\top$$

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,

$$\begin{aligned}B^* &= \left(B^\top {L^{-1}}^\top L^{-1} B\right)^{-1} B^\top {L^{-1}}^\top L^{-1} \\ &= \left(C^\top C\right)^{-1} C^\top L^{-1} \\ &= R^{-1} {R^{-1}}^\top C^\top L^{-1} \\ &= R^{-1} Q_1^\top L^{-1} \\ &= R^{-1} J_1^\top\end{aligned}$$

and

$$\begin{aligned}E &= {L^{-1}}^\top L^{-1} - {L^{-1}}^\top C \left(C^\top C\right)^{-1} C^\top L^{-1} \\ &= {L^{-1}}^\top L^{-1} - {L^{-1}}^\top C R^{-1} {R^{-1}}^\top C^\top L^{-1} \\ &= {L^{-1}}^\top Q Q^\top L^{-1} - {L^{-1}}^\top Q_1 Q_1^\top L^{-1} \\ &= {L^{-1}}^\top Q_2 Q_2^\top L^{-1} \\ &= J_2 J_2^\top\end{aligned}$$

where

$$\begin{aligned}C^\top C &= \begin{bmatrix} R^\top & 0 \end{bmatrix} Q^\top Q \begin{bmatrix} R \\ 0 \end{bmatrix} \\ &= R^\top R\end{aligned}$$

and

$$\begin{aligned}CR^{-1} &= \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} R^{-1} \\ &= Q_1\end{aligned}$$

and

$$\begin{aligned}J &= \begin{bmatrix} J_1 & J_2 \end{bmatrix} \\ &= \begin{bmatrix} {L^{-1}}^\top Q_1 & {L^{-1}}^\top Q_2 \end{bmatrix} \\ &= {L^{-1}}^\top Q\end{aligned}$$

In the dual algorithm the vectors $d = En^+$ and $r = B^* n^+$ are required. Compute the intermediate vector

$$v = J^\top n^+$$

$$= \begin{bmatrix} J_1^\top \\ J_2^\top \end{bmatrix} n^+$$
$$= \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

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.

## K.11 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:

$$\begin{aligned} \textbf{minimise} \quad & f(x) = x_1^4 + x_2^4 + x_3^4 + x_1 x_2 + x_1 x_3 + x_2 x_3 + x_1 + x_2 + x_3 + 5 \\ \textbf{subject to} \quad & g(x) = \left[ x_2 - 1; -x_1 - \sqrt{2}; -x_3 - 0.5 \right] \geq 0 \end{aligned}$$

*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:
At x =  [ -5.74793e-01 3.34409e-01 -5.74793e-01 ]
f(x) = 4.361595, 0.021994 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, 0.022086 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, 0.019655 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:
At x =  [ -5.74793e-01 3.34408e-01 -5.74792e-01 ]
f(x) = 4.361595, 0.022508 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 =  [ -5.74878e-01 3.34439e-01 -5.74804e-01 ]
f(x) = 4.361595, 0.074893 secs
LINESEARCH goldensection 94 iterations 916 f(x) calls
LINESEARCH goldensection f(x)= 4.361595 x=[ -0.574878 0.334439 -0.574804  ]

Testing quadratic linesearch:
At x =  [ -5.74868e-01 3.34438e-01 -5.74777e-01 ]
f(x) = 4.361595, 0.022610 secs
LINESEARCH quadratic 104 iterations 105 f(x) calls
LINESEARCH quadratic f(x)= 4.361595 x=[ -0.574868 0.334438 -0.574777  ]
```

*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)ii): 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 -0.526902 ] 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 -0.526930 ] 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 -0.526902 ] 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 -0.526930 ] 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 [ -1.414192 0.999986 -0.527111 ] 1 19 296 275
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 -0.527094 ] 1 9 56 45
SQP eye bfgs goldensection [ -1.414194 0.999995 -0.527054 ] 1 33 681 646
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 [ -1.414180 0.999986 -0.527180 ] 1 18 289 269
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 -0.527094 ] 1 9 56 45
SQP eye eye goldensection [ -1.414194 0.999995 -0.527054 ] 1 33 681 646
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 10 4
SQP none exact armijo_kim [ -1.414214 1.000000 -0.527104 ] 1 4 10 4
SQP none exact goldstein [ -1.414214 1.000000 -0.527104 ] 1 4 15 9
SQP none exact goldensection [ -1.414191 0.999985 -0.527110 ] 1 17 243 224
SQP none bfgs nosearch [ -1.414214 1.000000 -0.527085 ] 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 -0.527085 ] 1 8 18 8
SQP none bfgs armijo_kim [ -1.414214 1.000000 -0.527085 ] 1 8 18 8
SQP none bfgs goldstein [ -1.414214 1.000000 -0.527170 ] 1 7 35 26
SQP none bfgs goldensection [ -1.414213 1.000000 -0.526831 ] 1 25 319 292
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 [ -1.414178 0.999985 -0.527182 ] 1 16 236 218
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 -0.527094 ] 1 9 56 45
SQP none eye goldensection [ -1.414194 0.999995 -0.527054 ] 1 33 681 646
```

# Appendix L

# Fourier transform of the Gaussian function

See *NIST Digital Library of Mathematical Functions* [46, Sections 7.2.1 and 1.14.1]

## L.1 Preliminary results

The Gaussian function is $e^{-t^2}$.

### L.1.1 Integral of the Gaussian function

*Poisson* introduced this method of integrating the Gaussian function:

$$\begin{aligned}\left[\int_{-\infty}^{\infty} e^{-t^2} dt\right]^2 &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-x^2-y^2} dxdy \\ &= \int_0^{2\pi}\int_0^{\infty} e^{-r^2} rdrd\theta \\ &= 2\pi\int_0^{\infty} e^{-r^2} rdr \\ &= \pi\int_0^{\infty} e^{-s} ds \\ &= \pi\left[-e^{-s}\right]_0^{\infty} \\ &= \pi\end{aligned}$$

So:

$$\int_0^{\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{2}$$

### L.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:

$$G(\omega) = \mathcal{F}g(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} g(t) e^{\imath\omega t} dt$$

and the inverse Fourier transform is:

$$g(t) = \mathcal{F}^{-1}G(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} G(\omega) e^{-\imath\omega t} d\omega$$

The Fourier transform of $\frac{dg(t)}{dt}$ is:

$$\begin{aligned}
\mathcal{F}\frac{dg\left(t\right)}{dt} &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\frac{dg\left(t\right)}{dt}e^{\imath\omega t}dt \\
&= \frac{1}{\sqrt{2\pi}}\left\{\left[g\left(t\right)e^{\imath\omega t}\right]_{-\infty}^{\infty} - \imath\omega\int_{-\infty}^{\infty}g\left(t\right)e^{\imath\omega t}dt\right\} \\
&= -\imath\omega\mathcal{F}g\left(t\right)
\end{aligned}$$

Similarly:

$$\begin{aligned}
\mathcal{F}^{-1}\frac{dG\left(\omega\right)}{d\omega} &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\frac{dG\left(\omega\right)}{d\omega}e^{-\imath\omega t}d\omega \\
&= \frac{1}{\sqrt{2\pi}}\left\{\left[G\left(\omega\right)e^{-\imath\omega t}\right]_{-\infty}^{\infty} + \imath t\int_{-\infty}^{\infty}G\left(\omega\right)e^{-\imath\omega t}dt\right\} \\
&= \imath t\mathcal{F}^{-1}G\left(\omega\right)
\end{aligned}$$

## L.2 Derivation of the Fourier transform of the Gaussian function in the frequency domain

The Gaussian function in the frequency domain is:

$$G\left(\omega\right) = e^{-\left(\frac{\omega}{\alpha}\right)^2}$$

Differentiating both sides:

$$\frac{dG\left(\omega\right)}{d\omega} = -\frac{2\omega}{\alpha^2}G\left(\omega\right)$$

Taking the Fourier transform of both sides:

$$\imath tg\left(t\right) = -\frac{2\imath}{\alpha^2}\frac{dg\left(t\right)}{dt}$$

Rearranging and integrating both sides:

$$\begin{aligned}
\int_0^t -\frac{\tau\alpha^2}{2}d\tau &= \int_0^t \frac{\frac{dg(\tau)}{d\tau}}{g\left(\tau\right)}d\tau \\
\ln g\left(t\right) - \ln g\left(0\right) &= -\left(\frac{t\alpha}{2}\right)^2 \\
g\left(t\right) &= g\left(0\right)e^{-\left(\frac{t\alpha}{2}\right)^2}
\end{aligned}$$

where:

$$\begin{aligned}
g\left(0\right) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}e^{-\left(\frac{\omega}{\alpha}\right)^2}d\omega \\
&= \frac{2\alpha}{\sqrt{2\pi}}\int_0^{\infty}e^{-\omega^2}d\omega \\
&= \frac{\alpha}{\sqrt{2}}
\end{aligned}$$

# Appendix M

# Design of IIR digital filter transfer functions

This chapter reviews methods of constructing IIR digital filter transfer functions. The first section reviews the analogue-to-digital transformation of $s$-plane Butterworth filters to the $z$-plane. The subsequent sections review methods of designing *equi-ripple* amplitude response IIR filters.

## M.1 Design of discrete time filters with the bilinear transform

Various methods can be used to transform an analog filter 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 $\imath\omega$ axis maps to the $z$-plane unit circle $z = e^{\imath\Omega T}$, 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

$$H\left(e^{\imath\Omega T}\right) = \hat{H}\left(\imath \tan\frac{\Omega T}{2}\right)$$

The design frequencies of the analog prototype filter on the $s$-plane $\imath\omega$ axis must be "pre-warped" to the $z$-plane unit circle by the mapping $\tan\frac{\Omega T}{2} \to \omega$. In other words, the discrete-time filter behaves at frequency $\Omega$ 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

### M.1.1 Design of Butterworth IIR filters

The Butterworth filter transfer function is described in many textbooks (for example, *Roberts* and *Mullis* [168, Chapter 6]). I have included it here to justify the implementation in *butter2pq.m*.

#### 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 $\omega_c$ is found by the $s$-plane transformation $s \to \frac{s}{\omega_c}$. The high-pass response is found by the $s$-plane transformation $s \to \frac{1}{s}$.

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, $\lambda_k$:

$$\begin{aligned}\lambda_k &= \omega_c e^{\imath\theta_k} \\ \theta_k &= \frac{\pi}{2}\left(1+\frac{(2k-1)}{n}\right),\ 1 \le k \le n\end{aligned}$$

For each conjugate pole pair the corresponding low-pass filter second-order section with unity DC gain has transfer function[a]:

$$\begin{aligned}\hat{H}_k(s) &= \frac{\lambda_k\lambda_k^\dagger}{(s-\lambda_k)\left(s-\lambda_k^\dagger\right)} \\ &= \frac{\lambda_k\lambda_k^\dagger}{s^2-\left(\lambda_k+\lambda_k^\dagger\right)s+\lambda_k\lambda_k^\dagger} \\ &= \frac{\omega_c^2}{s^2-2\omega_c\cos\theta_k s+\omega_c^2}\end{aligned}$$

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 $\omega_c$ is found with the $s$-plane transformation $\frac{s}{\omega_c} \to \frac{\omega_c}{s}$. 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:

$$\hat{H}_k(s) = \frac{s}{\omega_c+s}$$

#### Design of second order Butterworth digital filters with the bilinear transform

For the Butterworth first order low-pass section:

$$\begin{aligned}H_{\frac{n+1}{2}}(z) &= \frac{\omega_c\left(1+z^{-1}\right)}{(\omega_c+1)+(\omega_2-1)z^{-1}} \\ &= d+\frac{qz^{-1}}{1+pz^{-1}}\end{aligned}$$

where

$$d = \frac{\omega_c}{\omega_c+1}$$

[a] † denotes complex conjugate transpose

$$
\begin{aligned}
q &= \frac{2\omega_c}{(\omega_c+1)^2} \\
p &= \frac{\omega_c-1}{\omega_c+1}
\end{aligned}
$$

For the Butterworth second order low-pass section:

$$
\begin{aligned}
H_k(z) &= \frac{g\left[1+z^{-1}\right]^2}{a_0+a_1z^{-1}+a_2z^{-2}} \\
&= d_0 + \frac{q_1z^{-1}+q_2z^{-2}}{1+p_1z^{-1}+p_2z^{-2}}
\end{aligned}
$$

where:

$$
\begin{aligned}
g &= \omega_c^2 \\
a_0 &= 1-2\omega_c\cos\theta_k+\omega_c^2 \\
a_1 &= 2\left(\omega_c^2-1\right) \\
a_2 &= 1+2\omega_c\cos\theta_k+\omega_c^2 \\
d_0 &= \frac{g}{a_0} \\
p_1 &= \frac{a_1}{a_0} \\
p_2 &= \frac{a_2}{a_0} \\
q_1 &= d_0\left(2-p_1\right) \\
q_2 &= d_0\left(1-p_2\right)
\end{aligned}
$$

Similarly, for the first order high-pass section:

$$
\begin{aligned}
H_{\frac{n+1}{2}}(z) &= \frac{\left(1-z^{-1}\right)}{(\omega_c+1)+(\omega_c-1)z^{-1}} \\
&= \frac{d}{w_c} - \frac{qz^{-1}}{1+pz^{-1}}
\end{aligned}
$$

and for the Butterworth second order high-pass section:

$$
\begin{aligned}
H_k(z) &= \frac{\left[1-z^{-1}\right]^2}{a_0+a_1z^{-1}+a_2z^{-2}} \\
&= t_0 + \frac{t_1z^{-1}+t_2z^{-2}}{1+p_1z^{-1}+p_2z^{-2}}
\end{aligned}
$$

where

$$
\begin{aligned}
t_0 &= \frac{d_0}{g} \\
t_1 &= -\frac{d_0}{g}\left(2+p_1\right) \\
t_2 &= \frac{q_2}{g}
\end{aligned}
$$

## M.2 Low passband sensitivity IIR filters

*Vaidyanathan et al.* [152, 149] 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).

### M.2.1 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 $m_0, m_1, \cdots$ such that the sensitivity of $|G(e^{\imath\omega})|$ with respect to each $m_i$ is very small in the pass-band. If $|G(e^{\imath\omega})| \leq 1$ for all $\omega$ 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 $|G(e^{\imath\omega})| = 1$ for certain frequencies $\omega_k$ in the passband, then at those frequencies, when an $m_i$ is perturbed the value of $|G(e^{\imath\omega})|$ can only decrease. In other words the first-order sensitivity is zero at $\omega_k$

$$\left.\frac{\partial |G(e^{\imath\omega_k})|}{\partial m_i}\right|_{m_i = m_{i_0}} = 0 \quad \forall i, \forall k$$

Now consider a stable all-pass function, $A_1(z)$

$$A_1(z) = \frac{a_m + a_{m-1} z^{-1} + \cdots + z^{-m}}{1 + a_1 z^{-1} + \cdots + a_m z^{-m}}$$
$$|A_1(e^{\imath\omega})| = 1$$

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

$$G(z) = \frac{1}{2}\left[A_1(z) + A_2(z)\right]$$

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

$$G(e^{\imath\omega}) = \frac{1}{2}\left[e^{\imath\theta_1(\omega)} + e^{\imath\theta_2(\omega)}\right]$$

where $\theta_1(\omega)$ and $\theta_2(\omega)$ are real-valued functions of $\omega$. Thus,

$$|G(e^{\imath\omega})| = \frac{1}{2}\left|1 + e^{\imath[\theta_2(\omega) - \theta_1(\omega)]}\right|$$

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 $|G(e^{\imath\omega})| \leq 1$ for all $\omega$. Accordingly, the structural lossless-ness of $A_1(z)$ and $A_2(z)$ induces structural bounded-ness in $G(z)$.

### M.2.2 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 $G(z)$

$$|H(e^{\imath\omega})|^2 = 1 - |G(e^{\imath\omega})|^2$$

In terms of the $z$-variable

$$\tilde{P}(z) P(z) + \tilde{Q}(z) Q(z) = \tilde{D}(z) D(z)$$

where $\tilde{P}(z) = P\left(z^{-1}\right)$ etc.

Computation of the spectral factor $Q(z)$ is simplified by the anti-symmetric nature of its coefficients.

$$Q^2(z) = P^2(z) - z^{-N} D\left(z^{-1}\right) D(z)$$

$D(z)$ 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

$$\begin{aligned} q_0 &= \sqrt{r_0} \\ q_1 &= \frac{r_1}{2q_0} \\ q_n &= \frac{r_n - \sum_{k=1}^{n-1} q_k q_{n-k}}{2q_0}, \quad 2 \le n \le N \end{aligned}$$

The specral 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

$$\begin{aligned} \tilde{P}(z) &\triangleq P\left(z^{-1}\right) = z^N P(z) \\ \tilde{Q}(z) &\triangleq Q\left(z^{-1}\right) = -z^N Q(z) \end{aligned}$$

and

$$\begin{aligned} [P(z) + Q(z)]\,[P(z) - Q(z)] &= z^{-N} D(z)\, D\left(z^{-1}\right) \\ [P(z) + Q(z)]\left[P\left(z^{-1}\right) + Q\left(z^{-1}\right)\right] &= D(z)\, D\left(z^{-1}\right) \end{aligned}$$

Moreover

$$P\left(z^{-1}\right) + Q\left(z^{-1}\right) = z^N [P(z) - Q(z)]$$

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 $z_1, z_2, \ldots, z_r$ be the zeros of $P(z) + Q(z)$ inside the unit circle and let $z_{r+1}, z_{r+2}, \ldots, z_N$ be those outside. Then

$$D(z) = \prod_{k=1}^{r} \left(1 - z^{-1} z_k\right) \prod_{k=r+1}^{N} \left(1 - z^{-1} z_k^{-1}\right)$$

and

$$[P(z) + Q(z)][P(z) - Q(z)] = \left[\prod_{k=1}^{r} \left(1 - z^{-1} z_k\right) \prod_{k=r+1}^{N} \left(1 - z^{-1} z_k^{-1}\right)\right] \left[\prod_{k=1}^{r} \left(z^{-1} - z_k\right) \prod_{k=r+1}^{N} \left(z^{-1} - z_k^{-1}\right)\right]$$

From which

$$P(z) + Q(z) = \alpha \left[\prod_{k=1}^{r} \left(1 - z^{-1} z_k\right) \prod_{k=r+1}^{N} \left(z^{-1} - z_k^{-1}\right)\right]$$
$$P(z) - Q(z) = \frac{1}{\alpha} \left[\prod_{k=1}^{r} \left(z^{-1} - z_k\right) \prod_{k=r+1}^{N} \left(1 - z^{-1} z_k^{-1}\right)\right]$$

where $\alpha$ is a real constant. This leads to the equations

$$G(z) + H(z) = \frac{P(z) + Q(z)}{D(z)} = \alpha \prod_{k=r+1}^{N} \left(\frac{z^{-1} - z_k^{-1}}{1 - z^{-1} z_k^{-1}}\right)$$
$$G(z) - H(z) = \frac{P(z) - Q(z)}{D(z)} = \frac{1}{\alpha} \prod_{k=1}^{r} \left(\frac{z^{-1} - z_k}{1 - z^{-1} z_k}\right)$$

Thus

$$G(z) + H(z) = \alpha A_1(z)$$
$$G(z) - H(z) = \frac{1}{\alpha} A_2(z)$$

where $A_1(z)$ and $A_2(z)$ are stable all-pass functions of order $N - r$ and $r$ respectively

$$A_1(z) = \prod_{k=r+1}^{N} \left(\frac{z^{-1} - z_k^{-1}}{1 - z^{-1} z_k^{-1}}\right)$$
$$A_2(z) = \prod_{k=1}^{r} \left(\frac{z^{-1} - z_k}{1 - z^{-1} z_k}\right)$$

The complementarity condition requires $\alpha^2 = 1$ so, finally

$$G(z) = \frac{1}{2}\left[A_1(z) + A_2(z)\right]$$
$$H(z) = \frac{1}{2}\left[A_1(z) - A_2(z)\right]$$

and $G(z)$ is implemented as the parallel combination of two all-pass functions.

This low-sensitivity realisation is summarised in Algorithm M.1.

The classical Butterworth, Chebyshev 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. $\left.\frac{\partial^k |G(e^{\imath\omega})|}{\partial \omega^k}\right|_{\omega=0} = 0$ for $k = 1, 2, \cdots, n_0$ where $n_0$ is some odd integer
   (In other words, $|G(e^{\imath\omega})|$ 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 $|G(e^{\imath\omega})| = 1$

Other filter pass-bands are obtained by frequency transformation.

The Octave script *vaidyanathan_allpass_example_test.m* implements the example transfer function shown in [152, Section V].

**Algorithm M.1** 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 $A_2(z)$

$$G(z) = \frac{1}{2}\left[A_1(z) + A_2(z)\right]$$
$$H(z) = \frac{1}{2}\left[A_1(z) - A_2(z)\right]$$

Furthermore, $H(z)$ is also bounded real and is doubly complementary to $G(z)$.

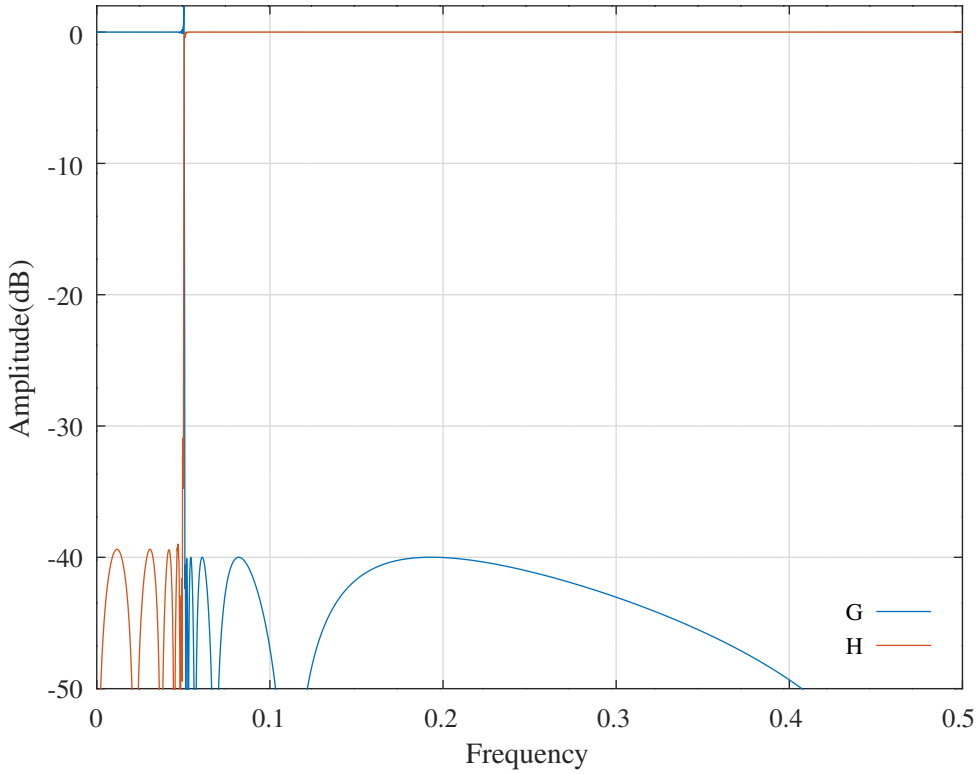



Figure M.1: 13th order elliptic filter, complementary filter and summed transfer functions

### M.2.3 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* abitrary precision floating point library [101, 164, 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 $0.05 f_S$, pass-band ripple 0.0005dB and stop-band ripple 40dB using *spectralfactor.oct*. Figures M.1 and M.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.

### M.2.4 Examples of parallel all-pass filter synthesis

#### 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.

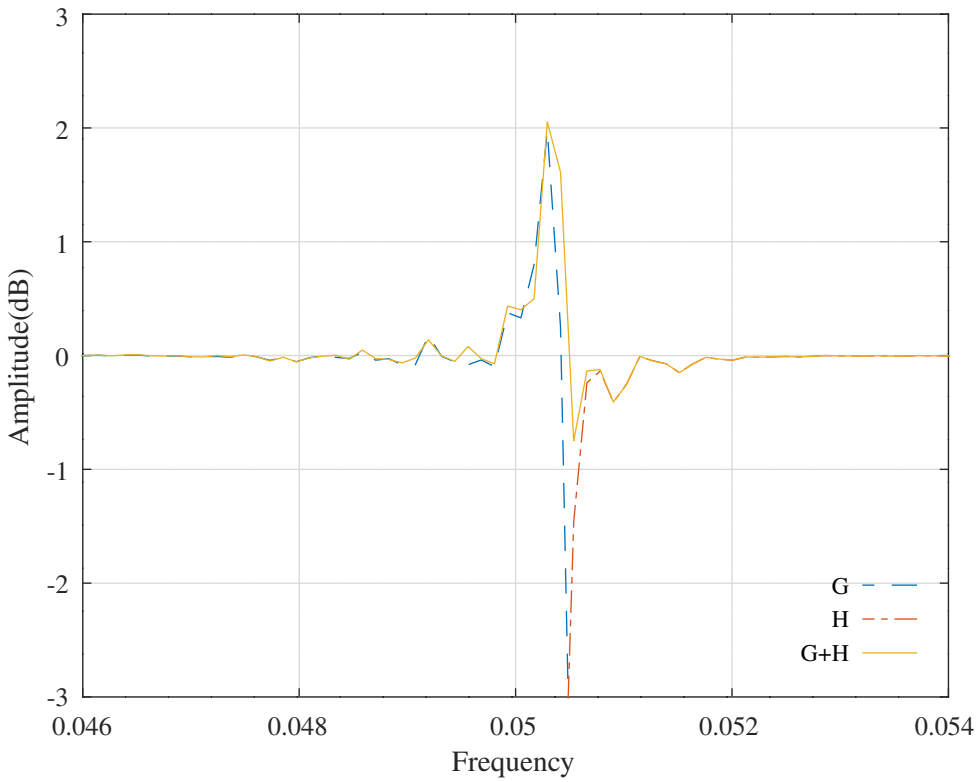



Figure M.2: 13th order elliptic filter, complementary filter and summed transfer functions transition band detail

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 = [     -488,      376 ]/512;
A1s11f = [      158,      352 ]/512;
A1s20f = [     -488,      376 ]/512;
A1s00f = [      158,      352 ]/512;
A1s02f = [      488,     -376 ]/512;
A1s22f = [      158,      352 ]/512;
```

and

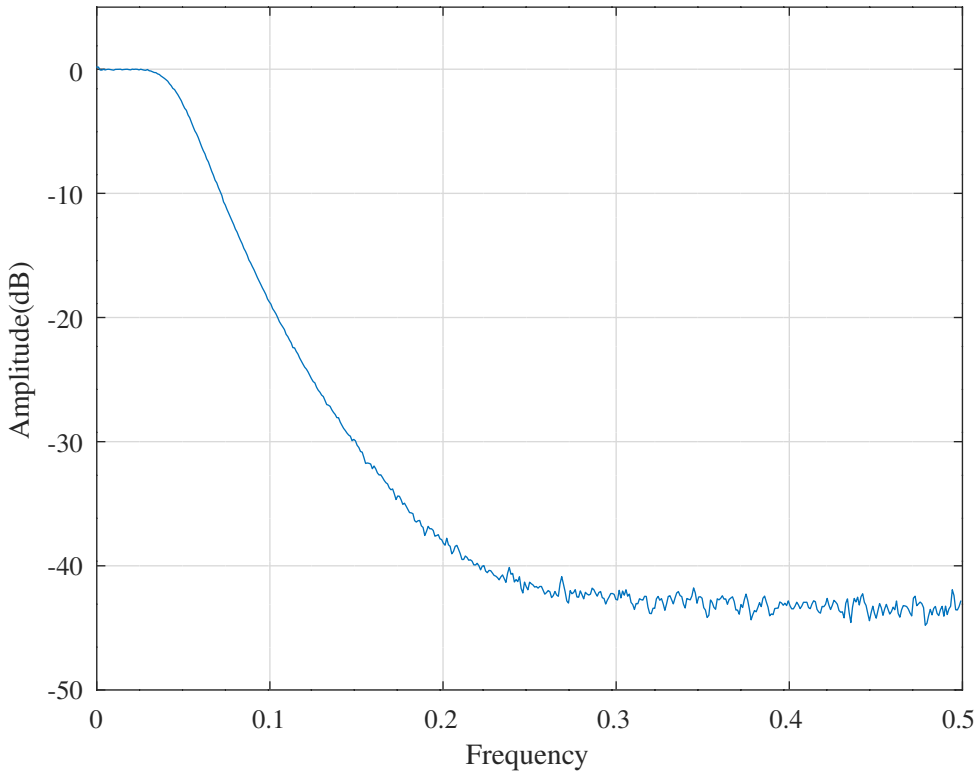



Figure M.3: Simulated amplitude response of the 3rd order Butterworth filter synthesised as the parallel combination of two all-pass filters implemented as normalised-scaled lattices with 10-bit 3-signed-digit coefficients

```
A2s10f = [      -368 ]/512;
A2s11f = [       352 ]/512;
A2s20f = [      -368 ]/512;
A2s00f = [       352 ]/512;
A2s02f = [       368 ]/512;
A2s22f = [       352 ]/512;
```

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 =  0.3492
varA1yapd =  0.3451
est_varA2yapd =  0.1629
varA2yapd =  0.1632
est_varyapd =  0.2530
varyapd =  0.2597
```

The standard deviations of the internal state delay storage elements are:

```
A1stdxf =  133.51   130.68
A2stdxf =  128.10
```

The amplitude response found from the cross-correlation of the input and output is shown in Figure M.3.

**6th order Butterworth band-pass**

The Octave script *butt6NSPABP_test.m* shows synthesis of a 6th order band-pass Butterworth filter as a parallel combination of two all-pass filters. 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 = [      -84,      488,      -76,      376 ]/512;
A1s11f = [      505,      158,      506,      352 ]/512;
A1s20f = [      -84,      488,      -76,      376 ]/512;
A1s00f = [      505,      158,      506,      352 ]/512;
A1s02f = [       84,     -488,       76,     -376 ]/512;
A1s22f = [      505,      158,      506,      352 ]/512;
```

and

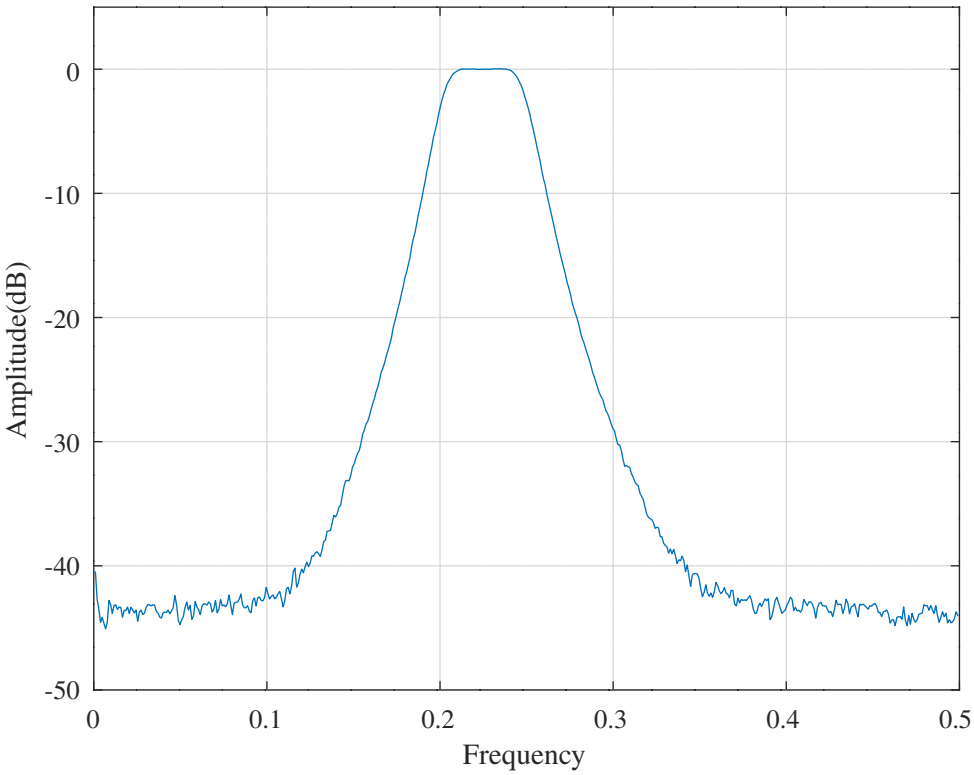



Figure M.4: Simulated amplitude response of the 6th order Butterworth band-pass filter synthesised as the parallel combination of two all-pass filters implemented as normalised-scaled lattices with 10-bit 3-signed-digit coefficients

```
A2s10f = [      -81,      368 ]/512;
A2s11f = [      506,      352 ]/512;
A2s20f = [      -81,      368 ]/512;
A2s00f = [      506,      352 ]/512;
A2s02f = [       81,     -368 ]/512;
A2s22f = [      506,      352 ]/512;
```

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 =  7.088e-01
varA1yapd =  6.891e-01
est_varA2yapd =  3.250e-01
varA2yapd =  3.305e-01
est_varyapd =  3.835e-01
varyapd =  3.889e-01
```

The standard deviations of the internal state delay storage elements are:

```
A1stdxf =  128.73  128.70  129.30  129.40
A2stdxf =  126.75  126.58
```

The amplitude response found from the cross-correlation of the input and output is shown in Figure M.4.

**20th order elliptic multi-band-pass**

The Octave script *ellip20OneMPAMB_test.m* shows synthesis of a 20th order multi-band-pass elliptic filter as the difference of two parallel all-pass filters. Annotated results of the script follow.

The prototype filter is a 5-th elliptic low-pass filter with cutoff frequency $f_c = 0.25$, pass-band ripple $0.5$dB and stop-band attenuation $40$dB.

The low-pass to multi-band-pass transformation defines the two pass-bands $0.05$ to $0.125$ and $0.175$ to $0.225$:

```
phi = [ 0.050, 0.125, 0.175, 0.225 ];
```

The multi-band-pass numerator and denominator polynomials are:

```
B = [     0.0222367857,    -0.2062046057,     0.9783850127,    -3.1191432836, ...
          7.4158462608,   -13.8280323127,    20.6788224219,   -24.7690078630, ...
         22.8635811740,   -13.8497964246,     0.0000000000,    13.8497964246, ...
        -22.8635811740,    24.7690078630,   -20.6788224219,    13.8280323127, ...
         -7.4158462608,     3.1191432836,    -0.9783850127,     0.2062046057, ...
         -0.0222367857 ]';
```

```
A = [     1.0000000000,   -11.2077820036,    64.3387049867,  -249.2399219984, ...
        726.5512969095, -1687.3317871425,  3229.9733333599, -5207.7948314400, ...
       7172.1667231069, -8511.5831269587,  8746.6271597328, -7794.7450109950, ...
       6014.3654584308, -3998.0802090981,  2269.4138776384, -1084.5000718524, ...
        426.9002692337,  -133.7555362622,    31.4942313827,    -4.9945132742, ...
          0.4044742430 ]';
```

The numerator polynomial is "sanitised" so that all zeros lie on the $z$-plane unit circle.

The multi-band-pass polynomials have even order. They are converted to odd order by adding a zero and a pole at $z = 1$. The resulting spectral factor is, after removing the zero at $z = 1$:

```
Qp = [     0.6363715249,    -7.4932195256,    45.1251257188,  -183.1987446896, ...
         559.2678154123, -1359.4624016594,  2722.6407116398, -4591.0679953962, ...
        6610.6987412724, -8200.3940342418,  8806.4979978881, -8200.3940342418, ...
        6610.6987412724, -4591.0679953962,  2722.6407116398, -1359.4624016594, ...
         559.2678154123,  -183.1987446896,    45.1251257188,    -7.4932195256, ...
           0.6363715249 ]';
```

The sum $B(z) + Qp(z)$ has roots

```
Z =
   0.25704 + 1.05045i
   0.25704 - 1.05045i
   0.78300 + 0.71880i
   0.78300 - 0.71880i
   0.98935 + 0.36462i
   0.98935 - 0.36462i
   0.45133 + 0.91101i
   0.45133 - 0.91101i
   0.45317 + 0.88790i
   0.45317 - 0.88790i
   0.94176 + 0.30369i
   0.94176 - 0.30369i
   0.69726 + 0.70078i
   0.69726 - 0.70078i
   0.14957 + 0.96734i
   0.14957 - 0.96734i
   0.37041 + 0.86443i
   0.37041 - 0.86443i
   0.75232 + 0.44289i
   0.75232 - 0.44289i
```

The numerator polynomials for the all-pass components of the multi-band-pass filter are:

```
A1 = [     0.6586083192,    -3.2676475618,     8.5811981987,   -14.8737326645, ...
          18.4195756513,   -16.5611469571,    10.6144228565,    -4.4787800562, ...
           1.0000000000 ];


A2 = [     0.6141347472,    -4.5364405578,    17.3103358916,   -44.2279424697, ...
          83.5847646214,  -122.4497241139,   142.2741939507,  -132.0273923796, ...
          97.2453884831,   -55.6152780216,    23.5865625104,    -6.7290019601, ...
           1.0000000000 ];
```

The 8-bit 4-signed-digit coefficients of the two one-multiplier lattice implementations are:

```
k1sd = [      -80,      110,     -102,      111, ...
              -92,       94,      -72,       84 ]/128;


epsilon1 = [ -1, -1,  1,  1, ...
              1,  1,  1,  1 ];
```

and

```
k2sd = [      -69,      118,     -109,      112, ...
             -106,      111,      -57,      114, ...
              -82,       95,      -83,       79 ]/128;


epsilon2 = [  1,  1,  1,  1, ...
              1,  1,  1,  1, ...
              1, -1,  1,  1 ];
```

The noise gains of each all-pass filter with signed-digit coefficients are (with exact state scaling):

```
A1ng =  15.000
A2ng =  23.0000
```

The estimated and simulated round-off noise variances at the combined all-pass output are (again with exact state scaling):

```
est_varysd =  0.9583
varyapsd =  0.9602
```

Figure M.5 shows the response with exact and signed-digit coefficients and the simulated amplitude response, found from the cross-correlation of the input and output, with signed-digit coefficients.

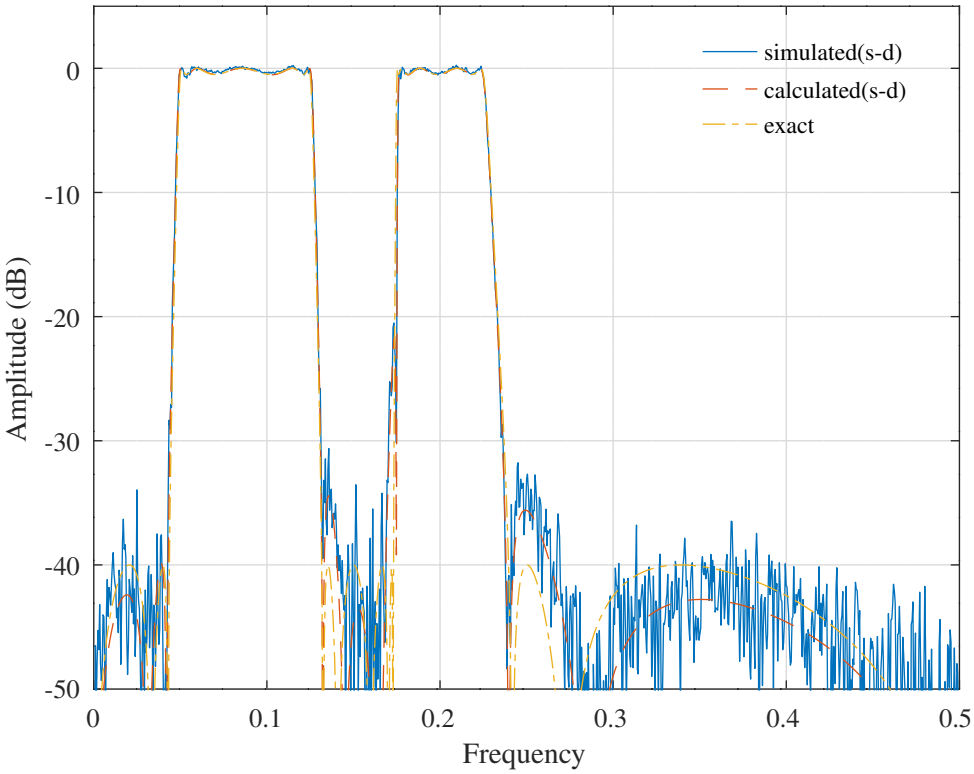



Figure M.5: Simulated amplitude response of the 20th order elliptic multi-band-pass filter synthesised as the parallel combination of two all-pass filters implemented as one-multiplier lattices with 8-bit 4-signed-digit coefficients

## M.3 Design of Elliptic IIR filters with a reduced number of multipliers

*Lutovac* and *Milić* [105, 121] describe the design of minimum Q-factor elliptic filters having a reduced number of multipliers. These filters are based on the continuous time minimal-Q elliptic filter design of *Rabrenović* and *Lutovac* [38, 37]. The minimal-Q factor elliptic filters have reduced component sensitivity at the expense of increased filter order or reduced selectivity.

### M.3.1 Elliptic filter design with the *Landen transformation*

In this section I follow the tutorial paper by *Orchard* and *Willson* [62]. Elliptic filters are an application of the *Jacobian elliptic functions* and *elliptic integrals* (reviewed in Appendix D). The original purpose of the elliptic functions was to invert the integral that finds the arc length of an ellipse. For example, the elliptic function $w = \operatorname{sn} u$ is defined by the *incomplete elliptic integral*:

$$u = \int_0^w \frac{dt}{\sqrt{(1-t^2)(1-\kappa^2 t^2)}} \tag{M.1}$$

For elliptic filter design, the *modulus*, $\kappa$, is real and $0 \leq k \leq 1$. The *modular angle*, $\theta$, is defined by $\kappa = \sin\theta$ and the *complementary modulus* is $\kappa' = \cos\theta$. The elliptic functions are *doubly-periodic* functions on the complex plane with *quarter-periods* $K$ and $K'$, where:

$$K = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-\kappa^2 t^2)}}$$

and similarly for $K'$. The elliptic functions are a generalisation of the circular and hyperbolic trigonometric functions. For $\kappa = 0$ and $K = \frac{\pi}{2}$, Equation M.1 is the $\arcsin$ function. For $\kappa = 1$ and $K = \infty$, Equation M.1 is the $\operatorname{arctanh}$ function.

The magnitude-squared response of an $n$-th order continuous time elliptic low-pass filter is:

$$|H(\omega)|^2 = \frac{1}{1+\epsilon^2 R_n^2(\omega, \omega_a)}$$

where $R_n(\omega, \omega_a)$ is an $n$-th order elliptic rational function, $\epsilon$ is the pass-band ripple factor and $\omega_a = \frac{1}{\kappa}$ is the stop-band edge frequency. Both $\omega_a$ and the frequency $\omega$ are normalised to $\omega_p$, the pass-band edge frequency. In the pass-band $R_n$ varies between 0 and 1 so that in the pass-band $|H(\omega)|^2$ varies between $\frac{1}{1+\epsilon^2}$ and 1. In the stop-band $|H(\omega)|^2$ varies between 0 and $\frac{1}{1+\epsilon^2 L_n^2}$ where $L_n = R_n(\omega_a)$ is called the discrimination factor.

*Orchard* and *Willson* show how to derive a continuous time elliptic filter with filter order $n$, pass-band ripple $\alpha_p$ and stop-band ripple $\alpha_a$ from the s-plane poles and zeros of an initial *Chebyshev Type 1* filter. They comment that:

> The fact that the degree $n$ must be an integer means that the smallest value of $n$ that meets the specification will normally provide some margin in performance that should be distributed amongst the quantities $\alpha_p$, $\alpha_a$ and $\kappa$. How best to do this requires some engineering judgement ...

The *Landen transformation* relates the elliptic functions, $\operatorname{ns}(uG, \gamma)$ and $\operatorname{ns}(uK, \kappa)$, for which the quarter-periods have $\frac{2G'}{G} = \frac{K'}{K}$ [62, Equation 8]:

$$\operatorname{ns}(uG, \gamma) = \frac{1}{1+\kappa}\left[\operatorname{ns}(uK, \kappa) + \frac{\kappa}{\operatorname{ns}(uK, \kappa)}\right]$$

*Orchard* and *Willson* [62, Section IV] derive a recurrence relation for the modulus, $\kappa$, that appears in successive Landen transformations:

$$\kappa_{n+1} = \left[\frac{\kappa_n}{1+\sqrt{1-\kappa_n^2}}\right]^2$$

This sequence, $\kappa_n$, tends rapidly to zero and the value of the elliptic function is considered to be equal to that of the the corresponding circular function value. In this case, $\operatorname{ns}(z, \kappa) \to \operatorname{cosec} z$ as $\kappa \to 0$. The required elliptic function value at the original modulus, $\kappa_0$, is found by reversing the recurrence:

$$\kappa_n = \frac{2\sqrt{\kappa_{n+1}}}{1+\kappa_{n+1}}$$

so that:

$$\mathrm{ns}\left(aK_n, \kappa_n\right) = \frac{1}{1+\kappa_{n+1}}\left[\mathrm{ns}\left(aK_{n+1}, \kappa_{n+1}\right) + \frac{\kappa_{n+1}}{\mathrm{ns}\left(aK_{n+1}, \kappa_{n+1}\right)}\right]$$

The normalized $n$-th order Chebyshev Type 1 filter with pass-band peak-to-peak ripple $\alpha_p = 10\log_{10}\left(1+\varepsilon^2\right)$ dB is defined by the parametric equations for the squared-magnitude of the filter attenuation[b]:

$$A\left(s\right)A\left(-s\right) = 1 + \varepsilon^2\cos^2\frac{nu\pi}{2}$$
$$\Omega = \cos\frac{u\pi}{2}$$

*Orchard* and *Willson* generalise the Chebyshev filter by changing the cosine function into the appropriate elliptic function equivalent. The resulting parametric equations are:

$$A\left(s\right)A\left(-s\right) = 1 + \varepsilon^2\,\mathrm{cd}^2\left(nuG, \gamma\right)$$
$$\Omega = \mathrm{cd}\left(uK, \kappa\right)$$
$$\frac{nK'}{K} = \frac{G'}{G}$$

The stop-band loss is $\alpha_a = 10\log_{10}\left(1+\frac{\varepsilon^2}{\gamma^2}\right)$dB. The following path in the complex $u$-plane maps the $u$ parameter from the complex plane to the $\Omega$ frequency axis with the desired amplitude response:

- Pass band: $u = 1$ to $u = 0$ maps to $\Omega = 0$ to $\Omega = 1$, $\mathrm{cd}\left(nuG, \gamma\right)$ varies between 1 and $-1$
- Transition band: $u = 0$ to $u = \imath\frac{K'}{K}$ maps to $\Omega = 1$ to $\Omega = \frac{1}{\kappa}$
- Stop band: $u = \imath\frac{K'}{K}$ to $u = 1 + \imath\frac{K'}{K}$ maps to $\Omega = \frac{1}{\kappa}$ to $\Omega = \infty$, $\mathrm{cd}\left(nuG, \gamma\right)$ has minimums at $\frac{1}{\gamma}$ and maximums at $-\frac{1}{\gamma}$

The $\mathrm{cd}$ elliptic function is used because it has a zero at the start of the path, at $u = 1$, and a pole at the end, at $u = 1 + \imath\frac{K'}{K}$.

*Orchard* and *Willson* show a *Matlab* file, *ellipap1.m* [62, Figure 7], that, given the filter order and pass-band and stop-band ripples, calculates the gain, poles and zeros of a continuous time elliptic filter by the *Landen transformation* of the poles and zeros of a *Type 1 Chebyshev* filter with the same pass-band and stop-band attenuation specifications.

### M.3.2 Design of elliptic filters with minimal-Q

*Rabrenović* and *Lutovac* [38, 37] derive the properties of a continuous time elliptic filter with poles having minimal-Q. The poles of the filter lie on a circle with radius $\sqrt{\omega_a}$ and the ripples in the pass-band and stop-band squared-magnitude response are equal:

$$1 - \frac{1}{1+\epsilon_{minQ}^2} = \frac{1}{1+\epsilon_{minQ}^2 L_n^2}$$

so that the for the minimal-Q elliptic filter:

$$\epsilon_{minQ} = \frac{1}{\sqrt{L_n}}$$

*Lutovac* and *Milić* [121] describe the design of an odd order, $n$, discrete-time elliptic filter implemented as the sum of an odd and an even order all-pass filter in which the all-pass filter components are expressed as products of second-order all-pass lattice sections:

$$\frac{b_m + a_m\left(1+b_m\right)z^{-1} + z^{-2}}{1 + a_m\left(1+b_m\right)z^{-1} + b_m z^{-2}}$$

with $m > 1$, and, for the odd-order filter, a first-order section:

$$\frac{a_1 + z^{-1}}{1 + a_1 z^{-1}}$$

[b] *Orchard* prefers the convention that $A\left(s\right)$ represents the filter attenuation, i.e.: the ratio of *input* to *output*.

For the second-order sections, with the poles at $z_m = r_m e^{\pm \imath \theta_m}$:

$$a_m = -2\frac{r_m \cos\theta_m}{1 + r_m^2}$$
$$b_m = r_m^2$$

where $m = 2, \ldots, \frac{n+1}{2}$. For the first-order section:

$$a_1 = -r_1$$

Following the notation of *Lutovac* and *Milić* [121] suppose that the digital filter specification has pass-band and stop-band frequencies $F_p$ and $F_a$ and corresponding pass-band ripple, $A_p$, and stop-band attenuation, $A_a$, given in $dB$. Assume that the digital filter is found by transformation of a prototype continuous time minimal-Q elliptic filter with a normalised pass band edge frequency of $\omega = 1$.

The pass-band and stop-band attenuations of the filter are $\alpha_p \leq A_p$ and $\alpha_a \geq A_a$ with:

$$\alpha_p = 10\log_{10}\left(1 + \frac{1}{L_n}\right)$$

and:

$$\alpha_a = 10\log_{10}(1 + L_n)$$

The 3dB frequency is given by:

$$\tan^2 \pi f_{3dB} = \tan \pi f_p \tan \pi f_a$$

The $s$-plane poles of the continuous time prototype are distributed around the circle $|s| = \sqrt{\omega_a}$. The bilinear transform

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

where

$$T = 2\frac{\sqrt{\omega_a}}{\tan \pi f_{3dB}} = 2\tan \pi f_p$$

maps that circle onto a circle in the $z$-plane orthogonal to the unit circle and centred on the real axis of the $z$-plane at $z = x_0$ with the $s$-plane point $\imath\sqrt{\omega_a}$ mapped to the $z$-plane point $z_{3dB} = e^{\imath 2\pi f_{3dB}}$ so that:

$$x_0 = \frac{1}{\cos 2\pi f_{3dB}} = \frac{1 - \tan^2 \pi f_{3dB}}{1 + \tan^2 \pi f_{3dB}}$$

For $m = 1$, the first order section has:

$$a_1 = -x_0\left(1 - \sqrt{1 - \frac{1}{x_0^2}}\right)$$

For $m > 1$, the second-order all-pass filter sections have, as shown in [121, Figure 4]:

$$r_m^2 + x_0^2 - 2r_m x_0 \cos\theta_m = x_0^2 - 1$$

so that:

$$2r_m \cos\theta_m = \frac{1 + r_m^2}{x_0}$$

and:

$$a_m = a = -\frac{1}{x_0}$$

The range of permitted values of $a$ is:

$$-\frac{1 - \tan^2 \pi F_p}{1 + \tan^2 \pi F_p} < a < -\frac{1 - \tan^2 \pi F_a}{1 + \tan^2 \pi F_a}$$

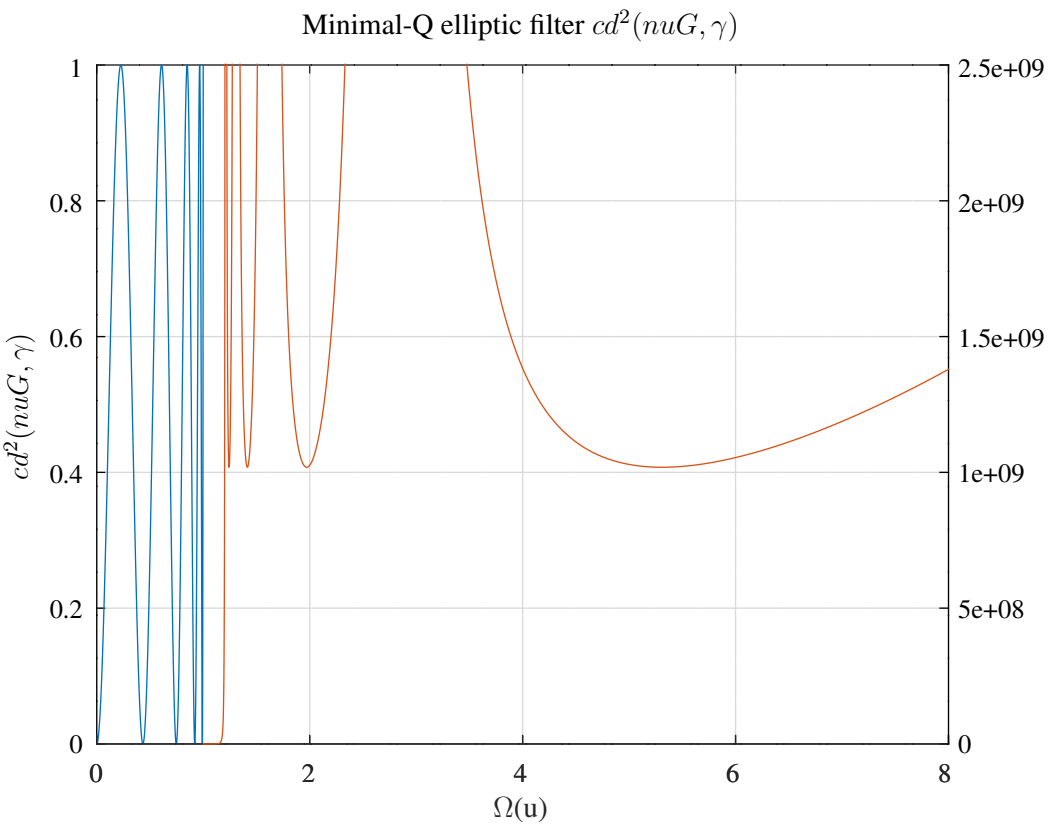



Figure M.6: Values of $\Omega$ and $\mathrm{cd}^2\,(nuG, \gamma)$ plotted as the complex parameter $u$ varies

The minimal-Q elliptic filter is implemented with a reduced number of multipliers by choosing $a$ to be a signed-digit number. *Lutovac* and *Milić* [121, Appendix A] show that the poles can be selected to alternate between the two all-pass branches with increasing angle or radius.

The Octave file *ellipMinQ_test.m* designs a discrete-time minimal-Q elliptic filter implemented as the sum of parallel all-pass filters. The filter order is $n = 9$, pass-band edge is $F_p = 0.1$, pass-band ripple specification is $0.1dB$, stop-band edge is $F_a = 0.125$, and the stop-band ripple specification is $40dB$. The second order lattice constant, $a$, was set to $-\frac{3}{4}$ with corresponding $f_{3dB} = 0.115027$. Setting $f_a = Fa$ gives a corresponding $f_p = 0.105715$. The continuous-time $s$-plane stop-band edge angular frequency is $\omega_a = 1.201010$ with elliptic filter moduluses $\kappa = 0.832632$ and $\gamma = 3.1329e - 05$. The calculation of the continuous-time $s$-plane pole and zero locations follows *Orchard* and *Willson* [62, Fig.6]. The resulting pass-band ripple is $\alpha_p = 0.0001361dB$ and the stop-band ripple is $\alpha_a = 45.04dB$. Figure M.6 shows the values of $\Omega$ and $\mathrm{cd}^2\,(nuG, \gamma)$ plotted as the complex parameter $u$ varies. Figure M.7 shows the amplitude response of the continuous-time $s$-plane filter when calculated with the transfer function derived from the $s$-plane gain, poles and zeros. Figure M.8 shows the pole-zero plot of the corresponding discrete-time $z$-plane filter. Figure M.9 shows the amplitude response of the discrete-time $z$-plane filter. The lattice coefficients are:

$$\{a_1, a, b_2, b_3, b_4, b_5\} = [-0.451416, -0.75, 0.350740, 0.619208, 0.822345, 0.949033]$$

The following 8-bit, 3-signed-digit lattice coefficients were found by brute-force search: $[-58, -96, 46, 79, 104, 121]$. The maximum stop-band response of the signed-digit coefficient filter is $-39.12dB$ at $0.1304$. Figure M.10 shows the amplitude response of the corresponding filter.

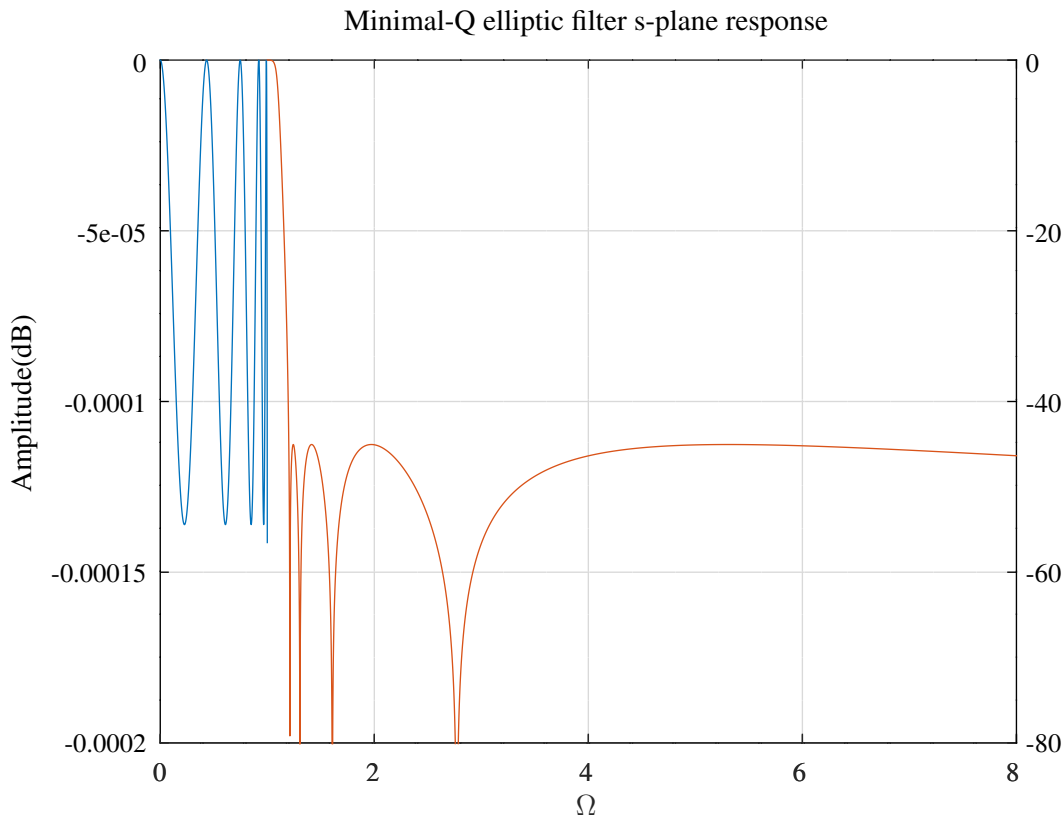



Figure M.7: Amplitude response of the continuous-time $s$-plane filter

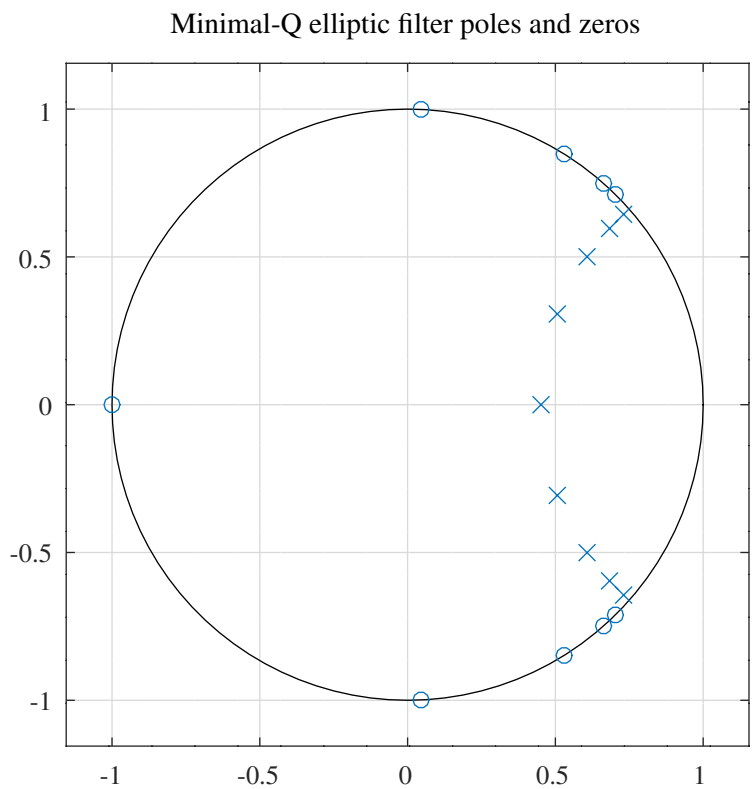



Figure M.8: Pole-zero plot of the discrete-time $z$-plane filter

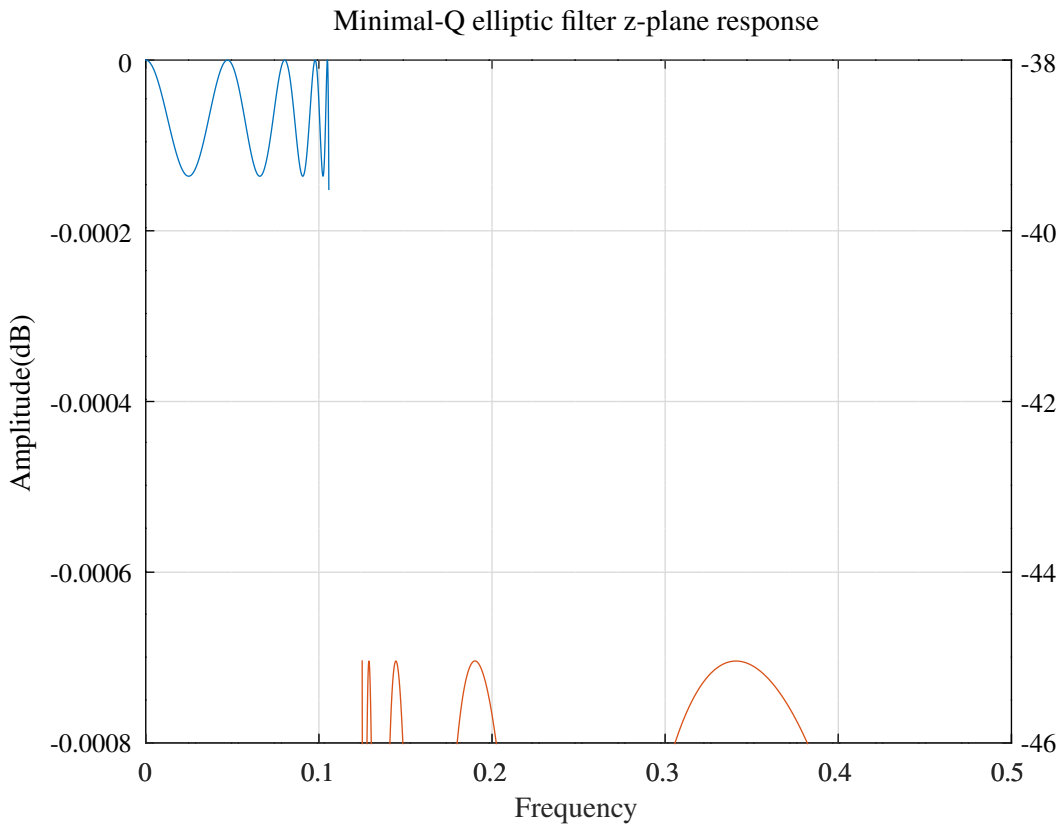



Figure M.9: Amplitude response of the discrete-time $z$-plane filter

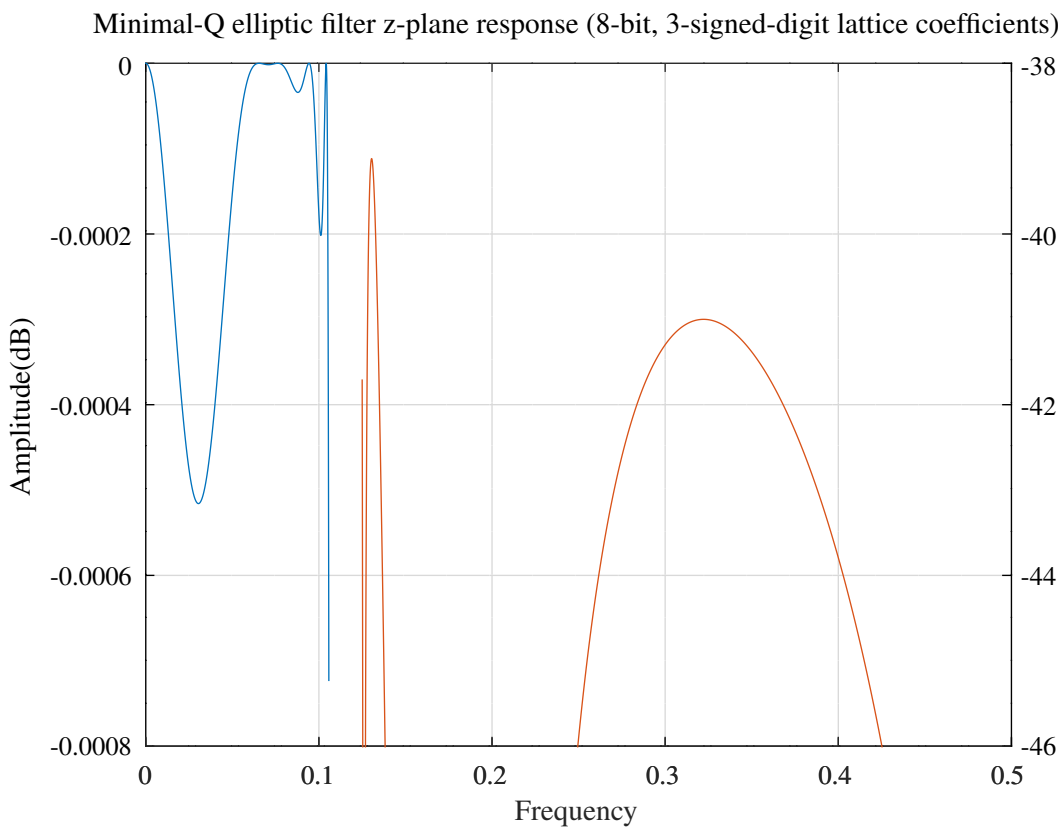



Figure M.10: Amplitude response of the discrete-time $z$-plane filter with 8-bit, 3-signed-digit coefficients

## M.4 *Saramäki*'s method for the design of IIR filters with zeros on the unit circle

Firstly, define the transfer function of an IIR filter:

$$H(z) = Kz^{n-m}\frac{\prod_{k=1}^{m}(z-a_k)}{\prod_{k=1}^{n}(z-b_k)}$$

The zeros, $a_k$, and poles, $b_k$, are real or conjugate pairs. When $n > m$ ($m > n$), $H(z)$ has $n-m$ zeros ($m-n$ poles) at the origin. A stable filter has $|b_k| < 1$. *Saramäki* [201] describes a procedure for designing an IIR filter with either equi-ripple pass-band and stop-band amplitude response. The algorithm transforms the frequency response so that the pass-band (stop-band) is equi-ripple and the amplitude response in the stop-band (pass-band) is optimised. He claims that the algorithm has improved numerical performance. Also, *Saramäki* finds that for $n \geq m$, his procedure produces better narrow-band filters than the corresponding elliptic filter. Conversely, when $n < m$, his algorithm produces better wide-band filters.

Write

$$H(z)\,H\left(\frac{1}{z}\right) = E\frac{\prod_{k=1}^{m}\left(z+\frac{1}{z}\right)\left(a_k-\frac{1}{a_k}\right)}{\prod_{k=1}^{n}\left(z+\frac{1}{z}\right)\left(b_k-\frac{1}{b_k}\right)}$$

*Saramäki*'s procedure maps $z+\frac{1}{z}$ to $v+\frac{1}{v}$ so that for $n \geq m$ ($n < m$) the pass-band (stop-band) on the unit circle in the $z$-plane is mapped to the entire upper unit circle on the $v$-plane. An equi-ripple pass-band (stop-band) response is obtained by construction. The remaining problem is, for $n \geq m$ ($n < m$), to find adjustable zeros (poles) so that the resulting $H(z)$ has $n-m$ zeros ($m-n$ poles) at the origin and the squared-magnitude function is equiripple in the stop-band (pass-band).

The all-pass function

$$F(v) = \prod_{k=1}^{r}\left(\frac{1-v_k v}{v-v_k}\right)$$

is, on the unit circle, $v = e^{\imath\Omega}$:

$$F\left(e^{\imath\Omega}\right) = e^{\imath f(\Omega)}$$

where:

$$f(\Omega) = \sum_{k=}^{r}\left\{\Omega - 2\arctan\left[\frac{\sin\Omega - \Im v_k}{\cos\Omega - \Re v_k}\right]\right\}$$

Now define:

$$G\left(v+\frac{1}{v}\right) = \frac{1}{2}\left[F(v) + \frac{1}{F(v)}\right]$$

so that, on the unit circle, $v = e^{\imath\Omega}$:

$$G\left(v+\frac{1}{v}\right) = \cos f(\Omega)$$

This function has $r+1$ alternating extrema, $\pm 1$, on $v = e^{\imath\Omega}$, $0 \leq \Omega \leq \pi$. Its value is $+1$ at $v = 1$ and $(-1)^r$ at $v = -1$.

*Saramäki* now introduces a mapping from $z = e^{\imath\omega}$, $\omega_1 \leq \omega \leq \omega_2$, to $v = e^{\imath\Omega}$, $0 \leq \Omega \leq \pi$:

$$v + \frac{1}{v} = C\left(z+\frac{1}{z}\right) + D$$

where:

$$C = \frac{2}{\cos\omega_1 - \cos\omega_2}$$
$$D = -\frac{2\left(\cos\omega_1 + \cos\omega_2\right)}{\cos\omega_1 - \cos\omega_2}$$

### M.4.1 Optimisation of a low-pass filter with denominator order higher

When $n \geq m$ set $\omega_1 = 0$ and $\omega_2 = \omega_p$. The resulting transformation maps the pass-band $z = e^{\imath\omega}$, $0 \leq \omega \leq \omega_p$, to the upper-unit circle $v = e^{\imath\Omega}$, $0 \leq \Omega \leq \pi$, and the stop-band, $\omega_s \leq \omega \leq \pi$, to the negative real axis part $[\zeta_1, \zeta_2]$. Solving the resulting quadratic equations for $\zeta_1$ and $\zeta_2$ gives:

$$\zeta_1 = -\frac{1}{2}\left[-2C\cos\omega_s - D - \sqrt{\left(-2C\cos\omega_s - D\right)^2 - 4}\right]$$
$$\zeta_2 = -\frac{1}{2}\left[2C - D - \sqrt{\left(2C - D\right)^2 - 4}\right]$$

The transformed magnitude-squared function is constructed as[c]:

$$\hat{H}\left(v\right)\hat{H}\left(\frac{1}{v}\right) = \frac{1}{1 + \frac{\Delta_p}{1-\Delta_p}\left[\frac{1}{2} + \left(-1\right)^n \frac{1}{4}\left[F\left(v\right) + \frac{1}{F(v)}\right]\right]}$$

where

$$F\left(v\right) = \prod_{k=1}^{n}\left(\frac{1 - \alpha_k v}{v - \alpha_k}\right)$$

and $1 - \Delta_p = \left(1 - \delta_p\right)^2$, $\delta_p = 1 - 10^{\frac{-dBap}{20}}$. By design, $\hat{H}\left(v\right)\hat{H}\left(\frac{1}{v}\right)$ has $n+1$ alternating extrema 1 and $1 - \Delta_p$ in $v = e^{\imath\Omega}$, $0 \leq \Omega \leq \pi$. We want to find $\alpha_k$ such that, in the $z$-plane, $H\left(z\right)H\left(\frac{1}{z}\right)$ has $m+1$ alternating extrema, $\Delta_s$ and 0, on $z = e^{\imath\omega}$, $\omega_s \leq \omega \leq \pi$, and $H\left(z\right)$ has $n - m$ zeros at the origin. The $z$-to-$v$-plane transformation maps $z = 0$ to $v = 0$ so set $\alpha_k = 0$ for $k = m+1, \ldots, n$. In addition, since $H\left(z\right)H\left(\frac{1}{z}\right)$ has $\lfloor\frac{m}{2}\rfloor$ minima, 0, on $z = e^{\imath\omega}$, $\omega_s < \omega < \pi$, $\hat{H}\left(v\right)\hat{H}\left(\frac{1}{v}\right)$ has $\lfloor\frac{m}{2}\rfloor$ double zeros in the transformed stop-band $[\zeta_1, \zeta_2]$. For $m$ odd, $H\left(z\right)$ has a zero at $z = -1$ which is transformed to $v = \zeta_2$. Thus, the $F\left(v\right)$ for the optimal $H\left(z\right)H\left(\frac{1}{z}\right)$ is:

$$F\left(\boldsymbol{\alpha}, v\right) = v^{-(n-m)}\left(\frac{1 - \zeta_2 v}{v - \zeta_2}\right)^q \prod_{k=1}^{\lfloor\frac{m}{2}\rfloor}\left(\frac{1 - \alpha_k v}{v - \alpha_k}\right)^2$$

where $q = 0$ for $m$ even and $q = 1$ for $m$ odd, $\zeta_1 < \alpha_k < \zeta_2$ and $\boldsymbol{\alpha} = \left[\alpha_1, \ldots, \alpha_{\lfloor\frac{m}{2}\rfloor}\right]$. The gradient of $F\left(\boldsymbol{\alpha}, v\right)$ with respect to $\alpha_k$ is:

$$\frac{\partial F\left(\boldsymbol{\alpha}, v\right)}{\partial \alpha_k} = 2\left[\frac{1 - v^2}{\left(1 - \alpha_k v\right)\left(v - \alpha_k\right)}\right]F\left(\boldsymbol{\alpha}, v\right)$$

By construction, $\left(-1\right)^n F\left(\boldsymbol{\alpha}, v\right) > 0$. The optimisation problem is to find $\boldsymbol{\alpha}$ and the scalar, $\Lambda$, such that at the $\lfloor\frac{m}{2}\rfloor + 1$ minima, $v_l$:

$$\log\left[\left(-1\right)^n F\left(\boldsymbol{\alpha}, v_l\right)\right] = \Lambda$$

*Saramäki* suggests a Remez-exchange type procedure for solving these non-linear equations. At each iteration, after linearising:

$$\log\left[\left(-1\right)^n F\left(\boldsymbol{\alpha}, v_l\right)\right] + \frac{\nabla_{\boldsymbol{\alpha}} F\left(\boldsymbol{\alpha}, v_l\right)}{F\left(\boldsymbol{\alpha}, v_l\right)}\Delta\boldsymbol{\alpha} = \Lambda + \Delta\Lambda$$

The initial $\boldsymbol{\alpha}$ are evenly spaced in $(\zeta_1, \zeta_2)$ and $\Lambda = 0$. The angles, $\omega_k$, of the $\lfloor\frac{m}{2}\rfloor$ complex conjugate zero pairs, $z = e^{\pm\imath\omega_k}$, of $H\left(z\right)$ are found from

$$\alpha_k + \frac{1}{\alpha_k} = 2C\cos\omega_k + D$$

The poles, $b_k$, of $H\left(z\right)$ are found by transforming the $n$ poles, $\beta_k$, of $\hat{H}\left(v\right)\hat{H}\left(\frac{1}{v}\right)$ that are the solutions of:

$$1 + \frac{\Delta_p}{1 - \Delta_p}\left[\frac{1}{2} + \left(-1\right)^n \frac{1}{4}\left[F\left(\boldsymbol{\alpha}, v\right) + \frac{1}{F\left(\boldsymbol{\alpha}, v\right)}\right]\right] = 0$$

Re-arranging gives:

$$F\left(\boldsymbol{\alpha}, v\right) = -\left(-1\right)^n\left[\frac{2 - \Delta_p}{\Delta_p} + \sqrt{\left(\frac{2 - \Delta_p}{\Delta_p}\right)^2 - 1}\right]$$

[c] Here I follow *Surma-aho* and *Saramäki* [96, Appendix] rather than *Saramäki* [201, Eqn.16]

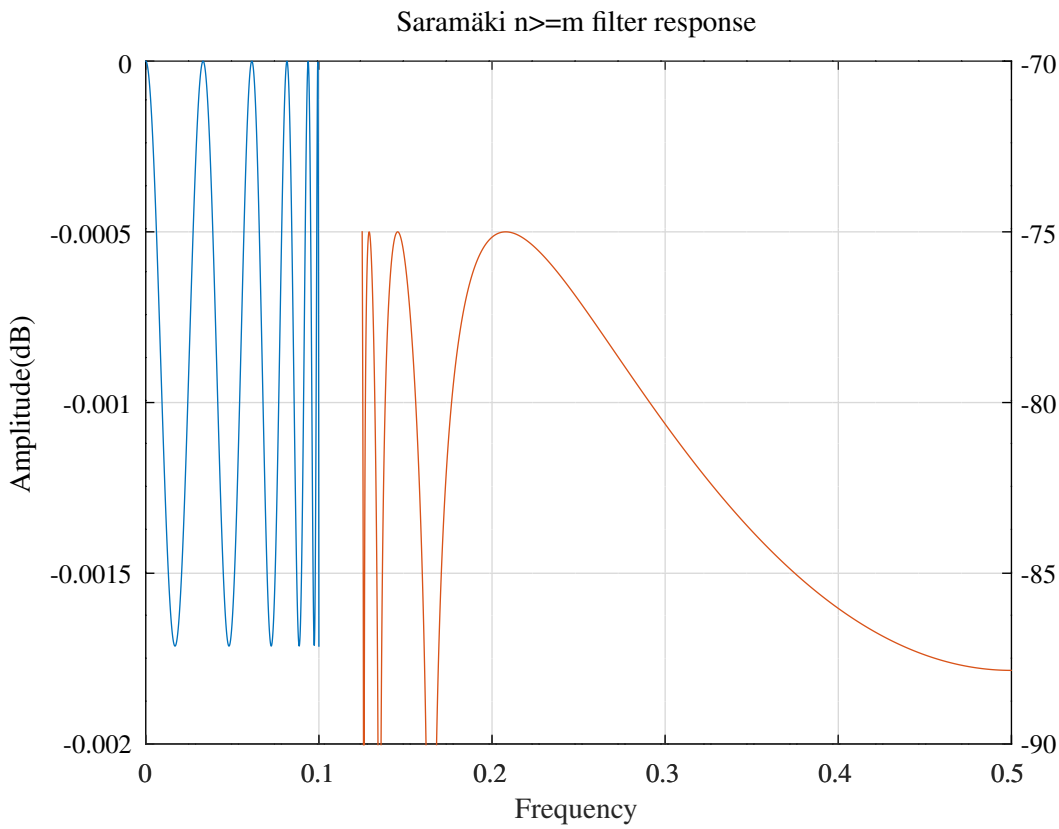



Figure M.11: Amplitude response of an IIR filter with $n = 11$ and $m = 6$, designed with the procedure of Saramäki

If the stop-band ripple, $\Delta_s$, is specified instead of the pass-band ripple, then the corresponding pass-band ripple, $\Delta_p$, is found from the stop-band equi-ripple maximum, $\lambda = (-1)^n F(\boldsymbol{\alpha}, v_l)$:

$$\Delta_p = \frac{1}{1 + \frac{\Delta_s}{1-\Delta_s}\left[\frac{1}{2} + \frac{1}{4}\left[\lambda + \frac{1}{\lambda}\right]\right]}$$

The scaling constant, $K$, is found from the condition $|H(e^{\imath\omega_p})|^2 = 1 - \Delta_p$.

The Octave function *saramakiFAvLogNewton* implements *Saramäki*'s procedure for $n \geq m$.

The Octave script *saramakiFAvLogNewton_test.m* designs an IIR filter with pass-band edge frequency, $fp = 0.1$, stop-band edge frequency, $fs = 0.125$, stop-band ripple, $dBas = 75$, denominator order, $n = 11$, and numerator order, $m = 6$. The resulting pass-band peak-to-peak ripple is $0.001713dB$. Figure M.11 shows the amplitude response of the filter. Figure M.12 shows the pole-zero plot of the filter. The numerator and denominator polynomials are, respectively:

```
n = [    0.0007007904,   -0.0026270912,    0.0053563235,   -0.0065837022, ...
         0.0053563235,   -0.0026270912,    0.0007007904 ];


d = [    1.0000000000,   -8.0831929697,   30.7252473784,  -72.2304919614, ...
       116.3931977738, -134.7455529109,  114.2001366939,  -70.7856326917, ...
        31.4234826049,   -9.5100073662,    1.7653436058,   -0.1522538136 ];
```

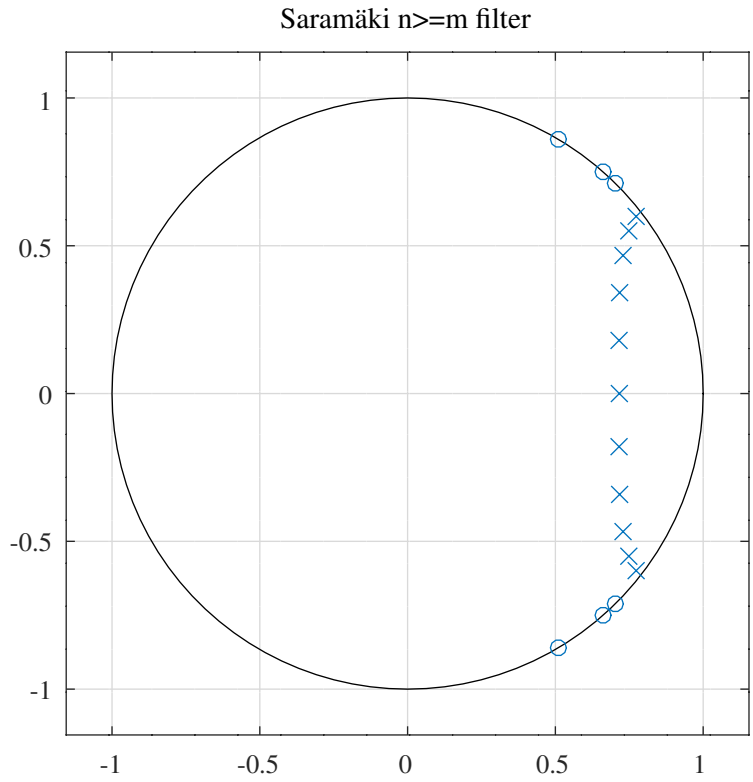



Figure M.12: Pole-zero plot of an IIR filter with $n = 11$ and $m = 6$, designed with the procedure of Saramäki

### M.4.2 Optimisation of a low-pass filter with denominator order lower

When $n < m$ set $\omega_1 = \omega_s$ and $\omega_2 = \pi$. The resulting transformation maps the stop-band $z = e^{\imath\omega}$, $\omega_s \leq \omega \leq \pi$, to the upper-unit circle $v = e^{\imath\Omega}$, $0 \leq \Omega \leq \pi$, and the pass-band, $0 \leq \omega \leq \omega_p$, to the positive real axis part $[\zeta_1, \zeta_2]$. Solving the resulting quadratic equations for $\zeta_1$ and $\zeta_2$ gives:

$$\zeta_1 = \frac{1}{2}\left[2C + D - \sqrt{(2C + D)^2 - 4}\right]$$
$$\zeta_2 = \frac{1}{2}\left[2C\cos\omega_p + D - \sqrt{(2C\cos\omega_p + D)^2 - 4}\right]$$

The transformed magnitude-squared function, $\hat{H}(v)\,\hat{H}\left(\frac{1}{v}\right)$, having $m+1$ alternating extrema $\Delta_s$ and 0 on $v = e^{\imath\Omega}$, $0 \leq \Omega \leq \pi$ is constructed as:

$$\hat{H}(v)\,\hat{H}\left(\frac{1}{v}\right) = \Delta_s\left[\frac{1}{2} + \frac{1}{4}\left(F(v) + \frac{1}{F(v)}\right)\right]$$

where:

$$F(v) = \prod_{k=1}^{m} \frac{1 - \beta_k v}{v - \beta_k}$$

is an all-pass function with $|\beta_k| < 1$ for $k = 1, \ldots, m$, and $\beta_k = 0$ for $k = m+1, \ldots, n$. The poles of $F(v)$ are poles of $\hat{H}(v)$ and the zeros of $F(v) + 1$ are the zeros of $\hat{H}(v)$.

Rewrite $F(v)$ as:

$$F(\boldsymbol{\beta}, v) = v^{-(m-n)}\left(\frac{1 - Rv}{v - R}\right)^q \prod_{k=1}^{\lfloor \frac{n}{2} \rfloor} \frac{1 + s_k v + r_k v^2}{v^2 + s_k v + r_k}$$

where $q = 0$ for $n$ even and $q = 1$ for $n$ odd. For $n$ odd, the parameter vector is $\boldsymbol{\beta} = \left[s_1, r_1, \ldots, s_{\lfloor \frac{n}{2} \rfloor}, r_{\lfloor \frac{n}{2} \rfloor}, R\right]$ where $R$ is a real pole lying on the real axis in $[-1, \zeta_1]$. By construction, $F(\boldsymbol{\beta}, v) > 0$ in the pass-band.

The gradients of $F(\boldsymbol{\beta}, v)$ with respect to the coefficients, $\boldsymbol{\beta}$, are:

$$\begin{aligned}
\frac{\partial F(\boldsymbol{\beta}, v)}{\partial s_k} &= \frac{(1-r_k)\, v\left(v^2-1\right)}{\left(1+s_k v+r_k v^2\right)\left(v^2+s_k v+r_k\right)} F(\boldsymbol{\beta}, v) \\
\frac{\partial F(\boldsymbol{\beta}, v)}{\partial r_k} &= \frac{\left(v^2-1\right)\left(v^2+s_k v+1\right)}{\left(1+s_k v+r_k v^2\right)\left(v^2+s_k v+r_k\right)} F(\boldsymbol{\beta}, v) \\
\frac{\partial F(\boldsymbol{\beta}, v)}{\partial R} &= \frac{1-v^2}{(v-R)(1-Rv)} F(\boldsymbol{\beta}, v)
\end{aligned}$$

If $\Delta_p$ is specified, then the optimisation problem is to find the stop band attenuation, $\Delta_s$, and $F(\boldsymbol{\beta}, v)$ such that $\hat{H}(v)\,\hat{H}\left(\frac{1}{v}\right)$ has $n+1$ alternating extrema 1 and $1-\Delta_p$ on $(\zeta_1, \zeta_2)$. *Saramäki* points out that, if $\Delta_s \ll 1$, then, in the pass-band:

$$\hat{H}(v)\,\hat{H}\left(\frac{1}{v}\right) \approx \frac{\Delta_s}{4} F(\boldsymbol{\beta}, v)$$

If $\gamma = \frac{4}{\Delta_s}$, then, in the pass-band, the Newton-Raphson update, $[\Delta\boldsymbol{\beta}, \Delta\gamma]$, is the solution of:

$$F(\boldsymbol{\beta}, v) + \nabla_{\boldsymbol{\beta}} F(\boldsymbol{\beta}, v)\,\Delta\boldsymbol{\beta} = (\gamma + \Delta\gamma)\left(1 - \frac{\Delta_p}{2} \pm \frac{\Delta_p}{2}\right)$$

*Saramäki* suggests that the initial values of $\boldsymbol{\beta}$ and $\Delta_s$ be found by transforming the $n+1$ frequencies of the pass-band extremal points of an order $n$ Chebyshev Type 1 filter from the $z$-plane to the $v$-plane and then solving the corresponding $n+1$ non-linear polynomial equations for $\hat{H}(v)\,\hat{H}\left(\frac{1}{v}\right)$ in the unknowns $e_0, \ldots, e_{n-1}$ and $\Delta_s$, with $e_n = 1$:

$$F(v) = v^{-(m-n)} \frac{\sum_{k=0}^{n} e_{n-k} v^k}{\sum_{k=0}^{n} e_k v^k}$$

The Octave function *saramakiFBvNewton* implements *Saramäki*'s procedure for $n < m$. The function initialises $F(\boldsymbol{\beta}, v)$ and $\Delta_s$ with a polynomial fitted to the $n+1$ frequencies of the pass-band extremal points. This method is simple but not robust. The Octave script *saramakiFBvNewton_test.m* designs an IIR filter with pass-band edge frequency, $fp = 0.2$, stop-band edge frequency, $fs = 0.35$, pass-band ripple, $dBap = 0.1$, denominator order, $n = 6$, and numerator order, $m = 9$. The resulting stop-band suppression is $127.29dB$. Figure M.13 shows the amplitude response of the filter. Figure M.14 shows the pole-zero plot of the filter. The numerator and denominator polynomials are, respectively:

```
n = [    0.0018065090,    0.0127354024,    0.0426992728,    0.0888119517, ...
         0.1258701526,    0.1258701526,    0.0888119517,    0.0426992728, ...
         0.0127354024,    0.0018065090 ];


d = [    1.0000000000,   -1.9501129639,    2.9114995274,   -2.5231419690, ...
         1.5889529275,   -0.5970515169,    0.1199588261 ];
```

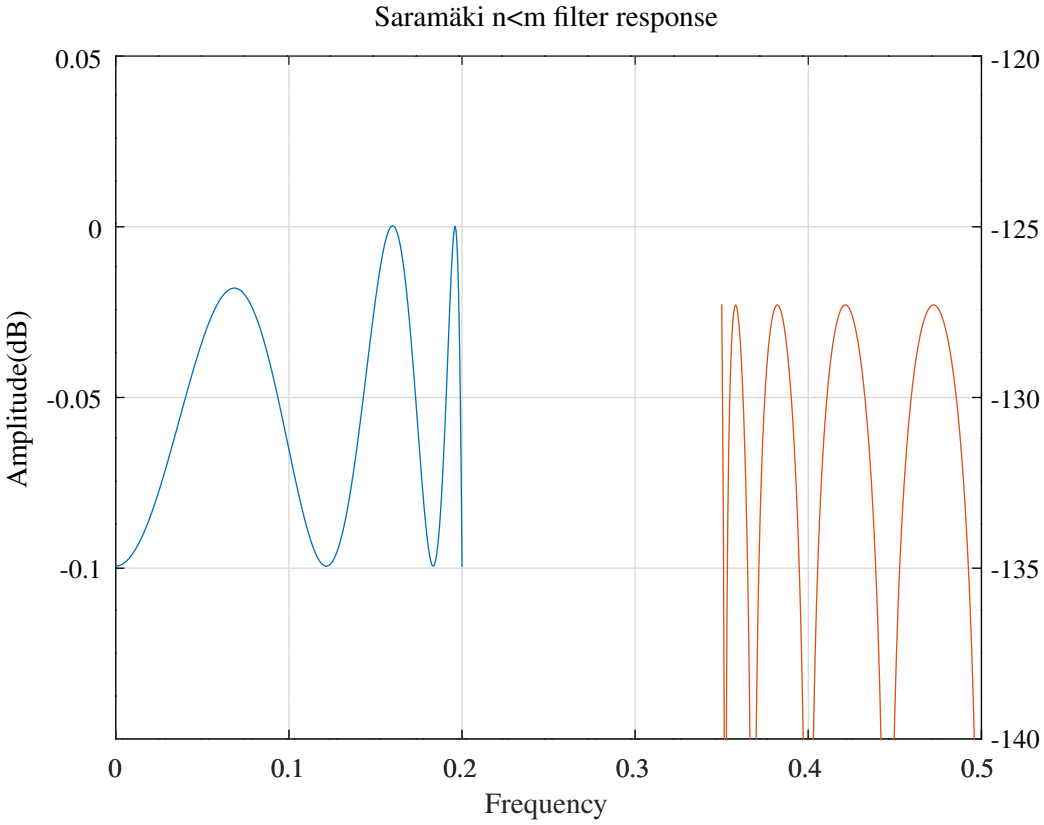



Figure M.13: Amplitude response of an IIR filter with $n = 6$ and $m = 9$, designed with the procedure of Saramäki

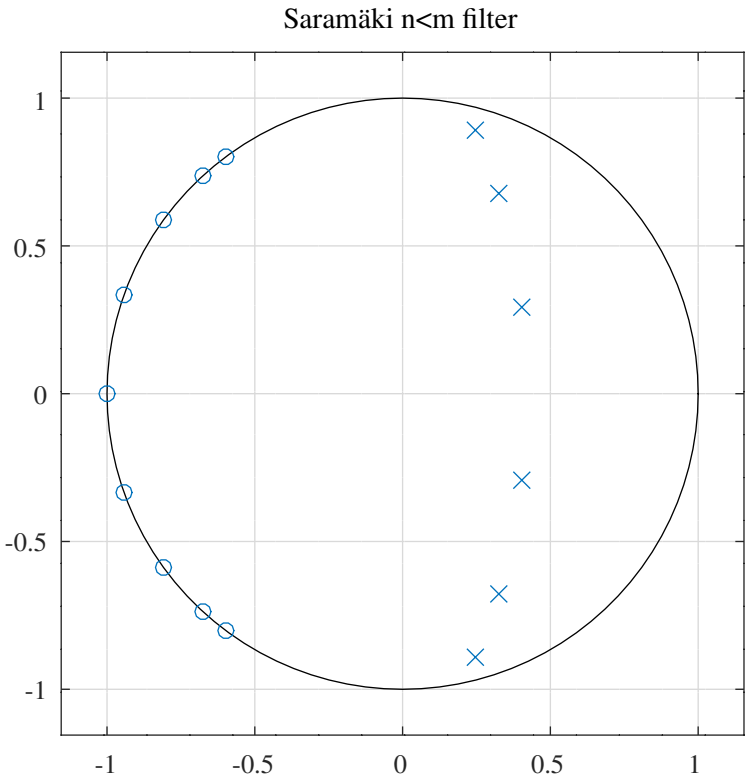



Figure M.14: Pole-zero of an IIR filter with $n = 6$ and $m = 9$, designed with the procedure of Saramäki

### M.4.3 Surma-aho and Saramäki method of unconstrained optimisation of an initial IIR filter

The method of *Tarczynski et al.* finds the polynomial transfer function of an initial filter. *Surma-aho* and *Saramäki* [96] describe an algorithm for finding an initial filter with approximately flat group delay in terms of the *gain-pole-zero* description[d] of the cascade-form transfer function, reproduced here as Algorithm M.2. The cascade form filter is decomposed into a minimum-phase filter and an all-pass equaliser.

**Algorithm M.2** *Surma-aho* and *Saramäki* method for finding an initial IIR filter in *gain-pole-zero* form [96, pp. 958-959]

**Step 1:** Determine the minimum order of an elliptic filter to meet the given amplitude criteria. Denote the minimum order by $n_{min}$ and set $k = 1$. Then, design an elliptic filter transfer function $H_{min}^k(z)$ such that it satisfies:
*Condition 1*: $\left|H_{min}^k(e^{\imath\omega})\right|$ oscillates in the stopband $[\omega_s, \pi]$ between $\delta_s$ and 0 achieving these values at $n_{min}+1$ points such that the value at $\omega = \omega_s$ is $\delta_s$. Here, $\delta_s$ is the specified stopband ripple.
*Condition 2:* $\left|H_{min}^k(e^{\imath\omega})\right|$ oscillates in the interval $\left[0, \Omega_p^k\right]$ $\left(\Omega_p^k \geq \omega_p\right)$ between 1 and $1-\delta_p^k$ achieving these values at $n_{min}+1$ points such that the value at $\omega = \Omega_p^k$ is $1-\delta_p^k$. For Candidate I, $\delta_p^k = \delta_p$, with $\delta_p$ being the specified passband ripple, whereas the passband region $\left[0, \Omega_p^k\right]$ is the widest possible to still meet the given stopband requirements. For Candidate II, $\Omega_p^k = \omega_p$ with $\omega_p$ being the specified passband edge, whereas $\delta_p^k$ is the smallest passband ripple to still meet the given stopband criteria.
**Step 2**: Cascade $H_{min}^k(e^{\imath\omega})$ with a stable all-pass equalizer with a transfer function $H_{all}^k(e^{\imath\omega})$ of order $n_{all}$. Determine the adjustable parameters of $H_{all}^k(z)$ and $\psi^k$ such that the maximum deviation of $\arg\left[H_{all}^k(z) H_{min}^k(z)\right]$ from the average slope $\phi_{ave}(\omega) = \psi^k\omega$ is minimized in the specified passband region, $[0, \omega_p]$. Let the poles of the all-pass filter be located at $z = z_1^k, z_2^k, \ldots, z_{n_{all}}^k$.
**Step 3:** Set $k = k+1$. Then, design a minimum-phase filter transfer function $H_{min}^k(z)$ of order $n_{min}+n_{all}$ such that it has $n_{all}$ fixed zeros at $z = z_1^{k-1}, z_2^{k-1}, \ldots, z_{n_{all}}^{k-1}$ and it satisfies Condition 1 of Step 1 with the same number of extremal points, that is, $n_{min}+1$, and Condition 2 of Step 1 with $n_{min}+n_{all}+1$ extremal points, instead of $n_{min}+1$ points.
**Step 4:** Like in Step 2, cascade $H_{min}^k(z)$ with a stable all-pass filter transfer function $H_{all}^k(z)$ of order $n_{all}$ and determine its adjustable parameters and $\psi^k$ such that the maximum of $\left|\arg\left[H_{all}^k(z) H_{min}^k(z)\right] - \psi^k\omega\right|$ is minimized in $[0, \omega_p]$. Let the poles of the all-pass filter be located at $z = z_1^k, z_2^k, \ldots, z_{n_{all}}^k$.
**Step 5:** If $\left|z_l^k - z_l^{k-1}\right| \leq \epsilon$ for $l = 1, 2, \ldots, n_{all}$ ($\epsilon$ is a small positive number), then stop. In this case, the zeros of the minimum-phase filter being located inside the unit circle and the poles of the all-pass equalizer coincide, reducing the overall order of $H(z) = H_{all}^k(z) H_{min}^k(z)$ from $n_{min}+2n_{all}$ to $n_{min}+n_{all}$. This filter is the desired initial filter with approximately linear-phase characteristics. Otherwise, go to Step 3.

*Saramäki* [202] and *Vaidyanathan* and *Mitra* [149] describe the conditions under which a filter transfer function can be implemented as the sum of two all-pass filters. *Surma-aho* and *Saramäki* modify their algorithm to design a filter composed of two parallel allpass filters, $H(z) = \frac{1}{2}\left[A(z) + B(z)\right]$, where the all-pass filter orders differ by 1 and the numerator polynomial of $H(z)$ is a symmetric, even-length FIR filter. Step 3 of Algorithm M.2 is modified so that:

> the minimum-phase filter is now of order $n_{min}+2n_{all}$ and it possesses double zeros at $z = z_1^k, z_2^k, \ldots, z_{n_{all}}^k$. Consequently, Condition 2 of Step 1 should be satisfied with $n_{min}+2n_{all}+1$ extremal points. In the case of Step 5, the algorithm is terminated when the double zeros of the minimum-phase filter being located inside the unit circle and the poles of the all-pass phase equaliser coincide. This reduces the overall order of $H(z) = H_{all}^k(z) H_{min}^k(z)$ from $n_{min}+3n_{all}$ to $n_{min}+2n_{all}$. The third modification in the low-pass case is that $n_{min}$ should be an odd number.

*Surma-aho* and *Saramäki* [96, Appendix] describe an implementation of Steps 1 and 3 of Algorithm M.2 that is similar to that described above in Section M.4.1, based on *Saramäki* [201].

Algorithm M.2 is intialised with a minimum-phase filter of order $n_{min}$ cascaded with an all-pass phase equaliser of order $n_{all}$. The squared-magnitude function is constructed with:

$$F(\boldsymbol{\alpha}, v) = \left(\frac{1-\xi_2 v}{v-\xi_2}\right)^q \times \prod_{k=1}^{\lfloor \frac{n_{min}}{2} \rfloor} \left(\frac{1-R_k v}{v-R_k}\right)^2 \times \prod_{k=1}^{n_{all}} \left(\frac{1-v\Gamma_k}{v-\Gamma_k}\right)^p$$

where $n = m = n_{min} + p \times n_{all}$, $q = 0$ if $n_{min}$ is even, $q = 1$ if $n_{min}$ is odd, $p = 2$ if the filter is to be realised as the parallel sum of two all-pass filters and $p = 1$ otherwise. The $v$-plane poles, $\Gamma_k$, are fixed and correspond to the $z$-plane poles of the all-pass filter. As above, optimising the stop-band peaks results in a new minimum-phase filter with order $n_{min} + p \times n_{all}$. This filter is then equalised with a new all-pass filter of order $n_{all}$. The procedure is repeated until the poles of the all-pass filter are cancelled by corresponding zeros of the minimum-phase filter. The resulting filter has order $n_{min} + p \times n_{all}$ and approximately linear phase in the pass-band.

[d] Although, unfortunately, the algorithm requires root-finding of intermediate polynomials.

**Low-pass filter example of the Surma-aho and Saramäki method**

The Octave script *surmaaho_lowpass_test.m* designs a low-pass filter with approximately flat pass-band delay. The filter specification is:

```
tol=1e-06 % Tolerance on coefficient update vector
nf=1000 % Frequency points across the band
nmin=7 % Minimum-phase filter order
nall=4 % All-pass phase equaliser filter order
fap=0.1 % Pass band amplitude response edge
dBap=0.100000 % Pass band amplitude response ripple
fas=0.125 % Stop band amplitude response edge
dBas=60.000000 % Stop band amplitude response ripple
fpp=0.08 % Initial pass band phase response edge
fpw=0.105 % Wider pass band phase response edge
tp=25 % Nominal pass band group delay
rho=0.968750 % Constraint on allpass pole radius
```

The gain-zero-pole coefficients are:

```
Ux=1,Vx=1,Mx=10,Qx=10,Rx=1
x = [   0.0011676867, ...
       -1.0000000000, ...
        0.8292402380, ...
        1.0000000000,   1.0000000000,   1.0000000000,   1.2132533887, ...
        1.2955253023, ...
        0.7950661045,   0.8940993950,   1.2939773298,   0.3403350177, ...
        0.1022103062, ...
        0.8307487201,   0.8522658190,   0.8566164877,   0.9130582446, ...
        0.9737984530, ...
        0.1658364238,   0.4787379723,   0.3290091013,   0.6074256121, ...
        0.6535665939 ]';
```

Figure M.15 shows the response of the filter. Figure M.16 shows the pole-zero plot of the filter.

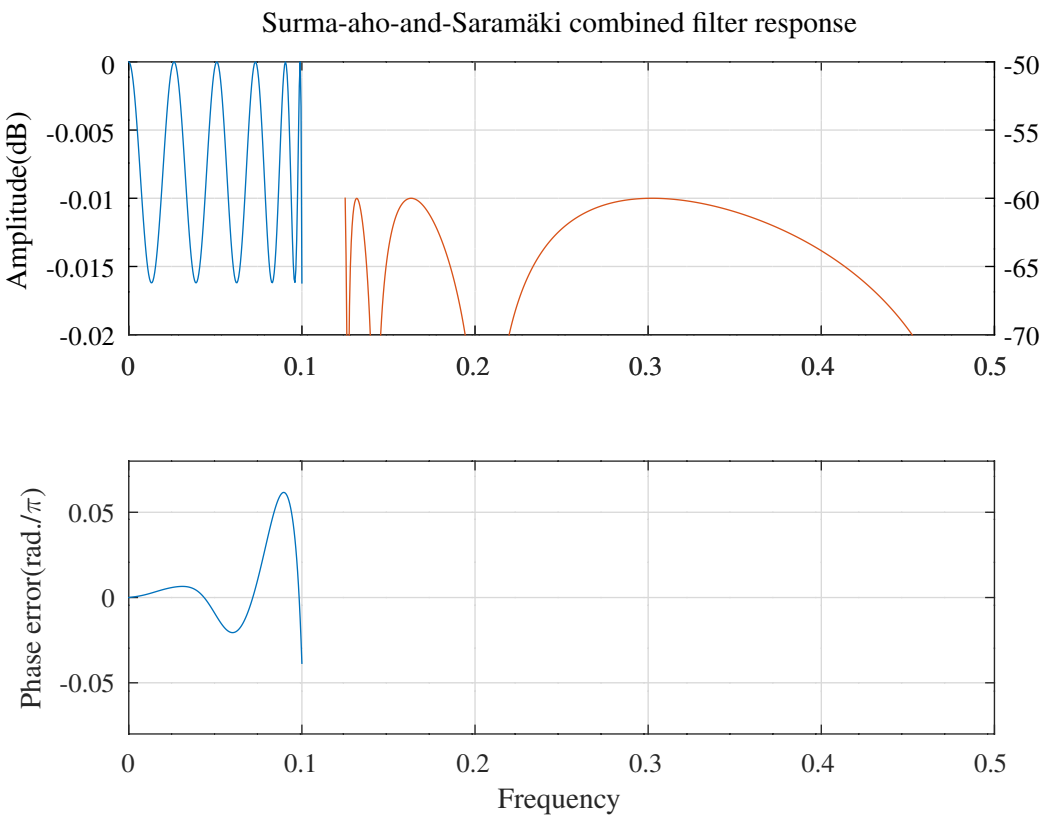



Figure M.15: Response of a low-pass IIR filter with $nmin = 7$ and $nall = 4$, designed with the procedure of Surma-aho and Saramäki

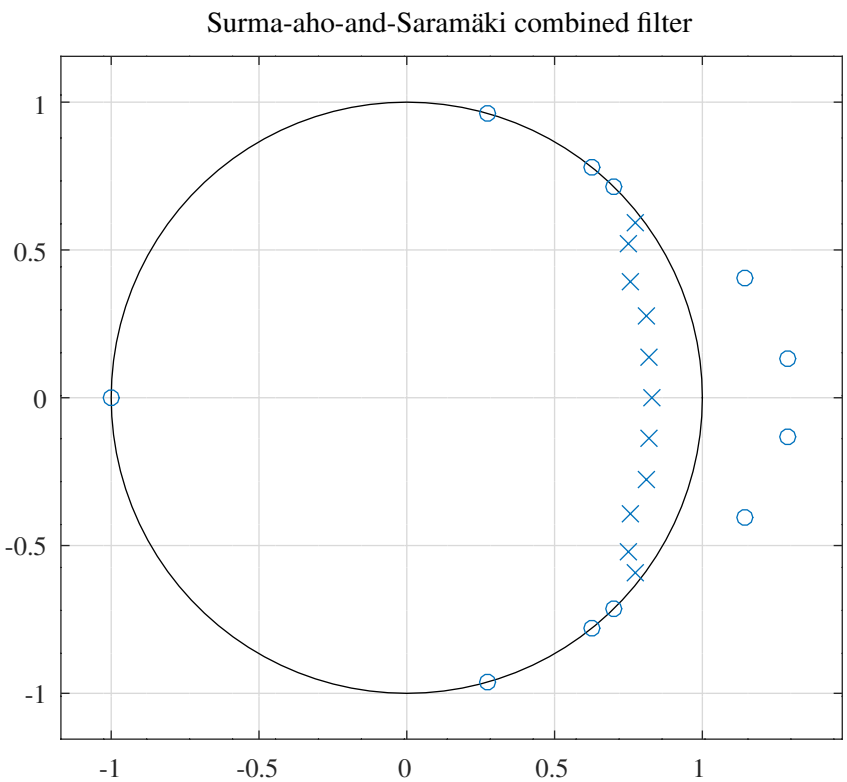



Figure M.16: Pole-zero plot of a low-pass IIR filter with $nmin = 7$ and $nall = 4$, designed with the procedure of Surma-aho and Saramäki

**Parallel all-pass low-pass filter example of the Surma-aho and Saramäki method**

The Octave script *surmaaho_parallel_allpass_lowpass_test.m* designs a low-pass filter with approximately flat pass-band delay, implemented as the sum of two all-pass filters. The filter specification is:

```
tol=1e-06 % Tolerance on coefficient update vector
nf=2000 % Frequency points across the band
nmin=7 % Minimum-phase filter order
nall=4 % All-pass phase equaliser filter order
fap=0.1 % Pass band amplitude response edge
dBap=0.100000 % Pass band amplitude response ripple
fas=0.125 % Stop band amplitude response edge
dBas=40.000000 % Stop band amplitude response ripple
fpp=0.08 % Initial pass band phase response edge
fpw=0.105 % Wider pass band phase response edge
tp=20 % Nominal pass band group delay
rho=0.968750 % Constraint on allpass pole radius
```

The combined gain-zero-pole coefficients are:

```
Ux=1,Vx=1,Mx=14,Qx=14,Rx=1
x = [   0.0058222631, ...
       -1.0000000000, ...
        0.7359976359, ...
        0.7374611692,   0.7830853230,   1.0000000000,   1.0000000000, ...
        1.0000000000,   1.2770000542,   1.3560035996, ...
        0.1197954841,   0.4101589301,   0.7934970298,   0.8762250189, ...
        1.2220444984,   0.4101589301,   0.1197954841, ...
        0.7287526515,   0.7669920494,   0.7748861071,   0.7909231674, ...
        0.7953147511,   0.8897749259,   0.9687752445, ...
        0.2165873655,   0.1168506815,   0.6178959041,   0.3839260889, ...
        0.4347709628,   0.7064801515,   0.7162074985 ]';
```

The all-pass filter pole coefficients are:

```
% All-pass single-vector representation
Va1=0,Qa1=8,Ra1=1
a1 = [   0.7669920485,   0.7748861074,   0.7909231683,   0.9687752451, ...
         0.1168506831,   0.6178959041,   0.3839260909,   0.7162074987 ]';
```

```
% All-pass single-vector representation
Va2=1,Qa2=6,Ra2=1
a2 = [   0.7359976401, ...
         0.7287526518,   0.7953147496,   0.8897749256, ...
         0.2165873617,   0.4347709611,   0.7064801511 ]';
```

Figure M.17 shows the response of the filter. There are $nmin + 2 \times nall + 1 = 16$ pass-band extrema. Figure M.18 shows the pole-zero plot of the filter. Figures M.19 and M.20 show the pole-zero plots of the all-pass filters.

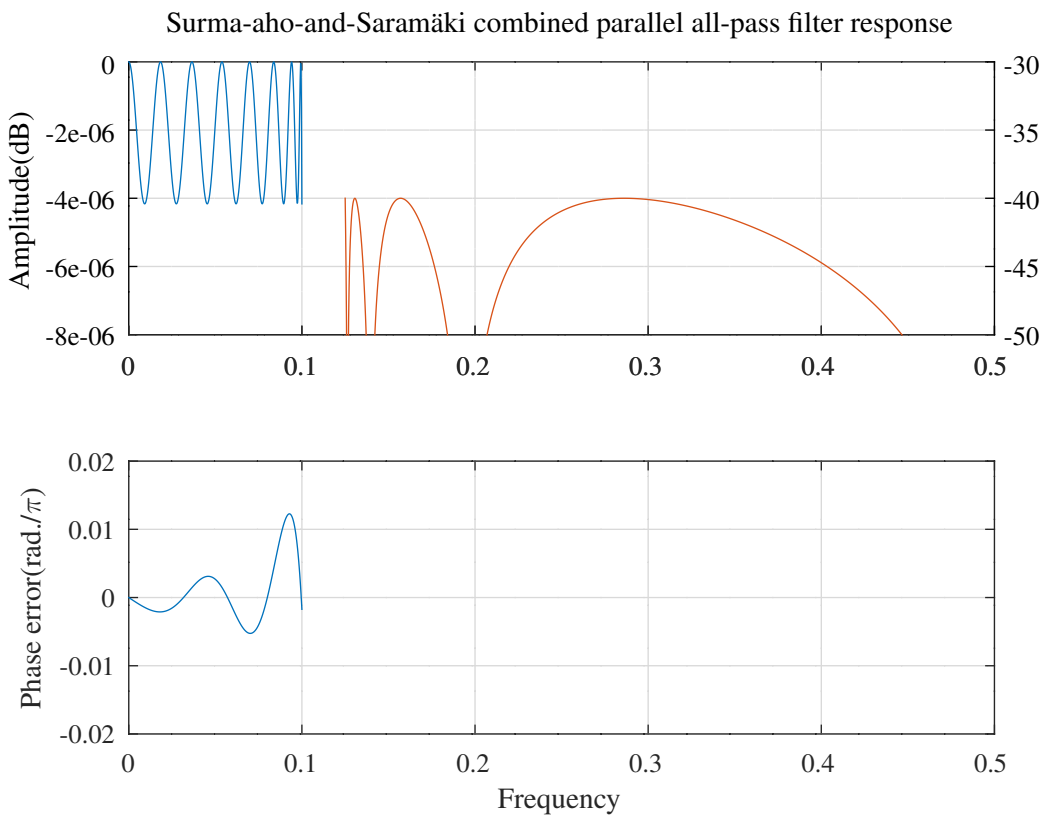



Figure M.17: Response of a parallel all-pass low-pass IIR filter with $nmin = 7$ and $nall = 4$, designed with the procedure of Surma-aho and Saramäki

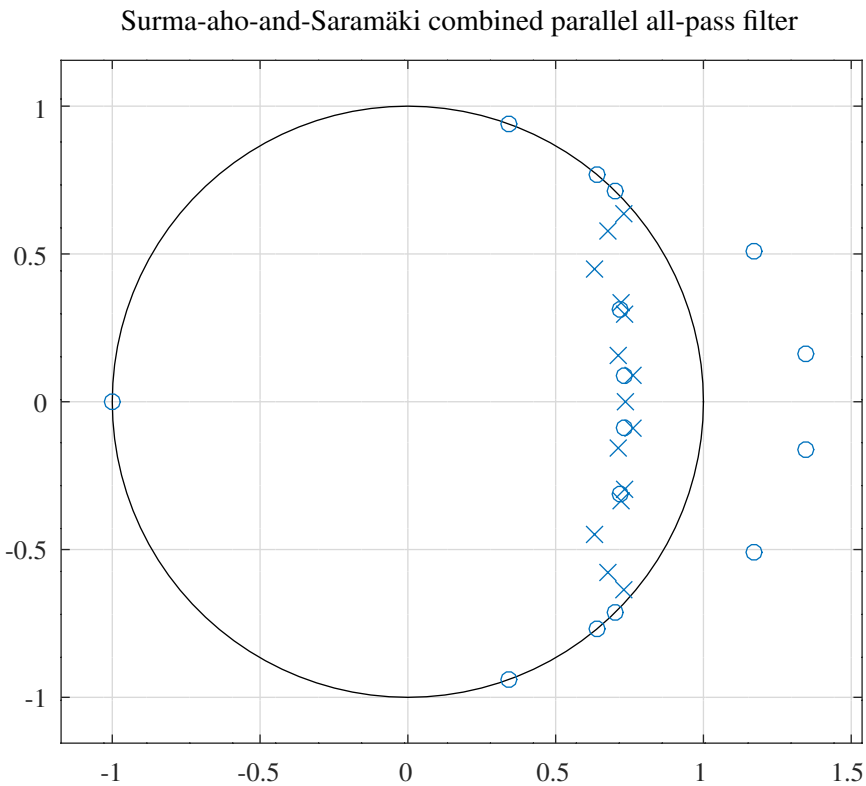



Figure M.18: Pole-zero plot of a parallel all-pass low-pass IIR filter with $nmin = 7$ and $nall = 4$, designed with the procedure of Surma-aho and Saramäki

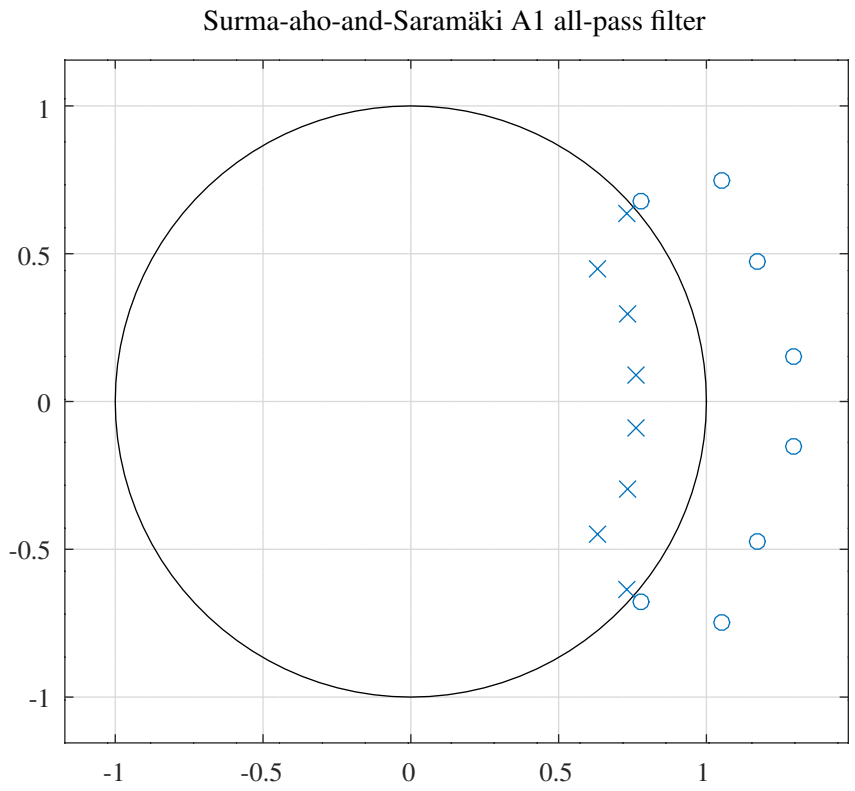



Figure M.19: Pole-zero plot of the A1 all-pass low-pass IIR filter designed with the procedure of Surma-aho and Saramäki

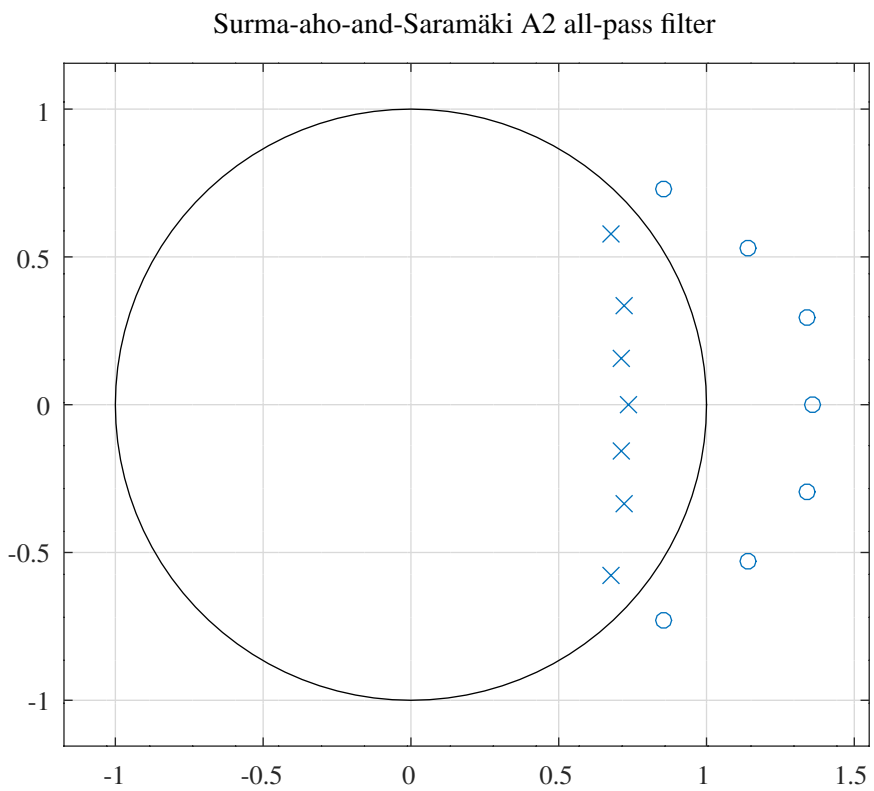



Figure M.20: Pole-zero plot of the A2 all-pass low-pass IIR filter designed with the procedure of Surma-aho and Saramäki

## M.5 Johansson and Saramäki design of all-pass complementary IIR filters

*Johansson* and *Saramäki* [65, Figure 1] describe a class of all-pass complementary IIR filters "realised as a tapped, cascaded interconnection of identical all-pass sub-filters". A pair of all-pass complementary filters has:

$$H(z) + H_C(z) = H_{ap}(z)$$

where $H_{ap}(z)$ is all-pass. A pair of magnitude complementary filters has:

$$|H(z)| + |H_C(z)| = 1$$

The transfer function of the filter is:

$$H(z) = \sum_{k=0}^{M} c_k \left[A_0(z)\right]^k \left[A_1(z)\right]^{M-k}$$

where $A_0(z)$ and $A_1(z)$ are stable all-pass filters, $M$ is even and $c_k = c_{M-k}$ for $k = 0, \ldots, M$. The all-pass complementary transfer function is:

$$H_C(z) = \left[A_0(z) A_1(z)\right]^{\frac{M}{2}} - H(z)$$

The filter design starts with a pair of all-pass complementary linear-phase FIR filters:

$$F(v) = \sum_{k=0}^{M} c_k v^{M-k}$$
$$F_C(v) = v^{\frac{M}{2}} - F(v)$$

where:

$$v^{-1} = \frac{A_0(z)}{A_1(z)}$$

is an all-pass frequency transformation. On the unit circle, $z = e^{\omega T}$ and $v = e^{\Omega T}$, where $T$ is the sampling interval. If $\phi_0(\omega T)$ is the phase response of $A_0(z)$ and $\phi_1(\omega T)$ is the phase response of $A_1(z)$, then:

$$\Omega T = \phi_1(\omega T) - \phi_0(\omega T)$$

and the zero-phase response of $F(v)$ is:

$$F_R(\Omega T) = c_{\frac{M}{2}} + 2 \sum_{k=0}^{\frac{M}{2}-1} c_k \cos\left[\left(\frac{M}{2} - k\right)\Omega T\right]$$

The gradients of $F_R$ with respect to the coefficients are:

$$\frac{\partial F_R(\Omega T)}{\partial c_k} = \begin{cases} 2\cos\left[\left(\frac{M}{2} - k\right)\Omega T\right] & k = 0, \ldots, \frac{M}{2} - 1 \\ 1 & k = \frac{M}{2} \end{cases}$$
$$\frac{\partial F_R(\Omega T)}{\partial \boldsymbol{a}_0} = 2\frac{\partial \phi_0(\omega T)}{\partial \boldsymbol{a}_0} \sum_{k=0}^{\frac{M}{2}-1} c_k \left(\frac{M}{2} - k\right) \sin\left[\left(\frac{M}{2} - k\right)\left(\phi_1(\omega T) - \phi_0(\omega T)\right)\right]$$
$$\frac{\partial F_R(\Omega T)}{\partial \boldsymbol{a}_1} = -2\frac{\partial \phi_1(\omega T)}{\partial \boldsymbol{a}_1} \sum_{k=0}^{\frac{M}{2}-1} c_k \left(\frac{M}{2} - k\right) \sin\left[\left(\frac{M}{2} - k\right)\left(\phi_1(\omega T) - \phi_0(\omega T)\right)\right]$$

where $\boldsymbol{a}_0$ and $\boldsymbol{a}_1$ represent the coefficients of the all-pass filters $A_0$ and $A_1$, respectively.

The diagonal of the Hessian of $F_R$ with respect to the coefficients is:

$$\frac{\partial^2 F_R(\Omega T)}{\partial c_k^2} = 0$$

$$\frac{\partial^2 F_R(\Omega T)}{\partial \boldsymbol{a}_0^2} = 2\frac{\partial^2 \phi_0(\omega T)}{\partial \boldsymbol{a}_0^2}\sum_{k=0}^{\frac{M}{2}-1} c_k\left(\frac{M}{2}-k\right)\sin\left[\left(\frac{M}{2}-k\right)(\phi_1(\omega T)-\phi_0(\omega T))\right]\cdots$$
$$-2\left[\frac{\partial \phi_0(\omega T)}{\partial \boldsymbol{a}_0}\right]^2\sum_{k=0}^{\frac{M}{2}-1} c_k\left(\frac{M}{2}-k\right)^2\cos\left[\left(\frac{M}{2}-k\right)(\phi_1(\omega T)-\phi_0(\omega T))\right]$$
$$\frac{\partial^2 F_R(\Omega T)}{\partial \boldsymbol{a}_1^2} = -2\frac{\partial^2 \phi_1(\omega T)}{\partial \boldsymbol{a}_1^2}\sum_{k=0}^{\frac{M}{2}-1} c_k\left(\frac{M}{2}-k\right)\sin\left[\left(\frac{M}{2}-k\right)(\phi_1(\omega T)-\phi_0(\omega T))\right]\cdots$$
$$-2\left[\frac{\partial \phi_1(\omega T)}{\partial \boldsymbol{a}_1}\right]^2\sum_{k=0}^{\frac{M}{2}-1} c_k\left(\frac{M}{2}-k\right)^2\cos\left[\left(\frac{M}{2}-k\right)(\phi_1(\omega T)-\phi_0(\omega T))\right]$$

As an example, *Johansson* and *Saramäki* [65, Figure 2] design a band-stop filter with $M = 6$, lower pass-band edge, $\omega_{apl}T = 0.3\pi$, lower stop-band edge, $\omega_{asl}T = 0.4\pi$, upper stop-band edge $\omega_{asu}T = 0.5\pi$ and upper pass-band edge, $\omega_{apu}T = 0.6\pi$. The all-pass transformation phase varies in the range $[0, \Omega_p T]$ across the lower pass-band, in the range $[\pi - \Omega_p T, \pi + \Omega_p T]$ across the stop band and in the range $[2\pi - \Omega_p T, 2\pi]$ across the upper pass-band, where $\Omega_p T = 0.038489$. The specified amplitude peak ripple in each band of the filter is $\delta_p = \delta_s = 0.00001$. The amplitude response of the prototype FIR filter is designed to have pass-band and stop-band widths that correspond to the pass-band peak-to-peak phase and stop-band peak-to-peak phase, respectively, of the all-pass transformation. In other words, the amplitude response of the prototype FIR filter is:

$$\begin{aligned} 1-\delta_p \le |F_R(\Omega T)| \le 1+\delta_p, &\ \ \Omega T \in [0, \Omega_p T] \\ |F_R(\Omega T)| \le \delta_s, &\ \ \Omega T \in [\pi - \Omega_p T, \pi] \end{aligned}$$

and the phase response of the all-pass transformation is such that:

$$\begin{aligned} -\Omega_p T \le \Omega T \le \Omega_p T, &\ \ \omega T \in [0, \omega_{apl} T] \\ \pi - \Omega_p T \le \Omega T \le \pi + \Omega_p T, &\ \ \omega T \in [\omega_{asl} T, \omega_{asu} T] \\ 2\pi - \Omega_p T \le \Omega T \le 2\pi + \Omega_p T, &\ \ \omega T \in [\omega_{apu} T, \pi] \end{aligned}$$

*Johansson* and *Saramäki* recommend designing the all-pass transformation with the prototype IIR filter:

$$G(z) = \frac{A_0(z) + A_1(z)}{2}$$

having magnitude response:

$$|G(\omega T)| = \left|\cos\frac{\phi_1(\omega T) - \phi_0(\omega T)}{2}\right|$$

and

$$\begin{aligned} 1-\Delta_p \le |G(\omega T)| \le 1+\Delta_p, &\ \ \omega T \in [0, \omega_{apl} T] \cup [\omega_{apu} T, \pi] \\ |G(\omega T)| \le \Delta_s, &\ \ \omega T \in [\omega_{asu} T, \omega_{asl} T] \end{aligned}$$

where $\Delta_p = 1 - \cos\frac{\Omega_p T}{2}$ and $\Delta_s = \cos\frac{\pi - \Omega_p T}{2}$.

To design a magnitude complementary pair of filters, $F_R(\Omega T)$ must vary between $1 - \delta_p$ and 1 in the pass-band and between 0 and $\delta_s$ in the stop-band. This is acheived by first designing an equi-ripple zero-phase FIR filter, $E_R(\Omega T)$, with pass-band ripple, $d_p$, and stop-band ripple, $d_s$, given by:

$$d_p = \frac{\frac{\delta_p}{2}}{1 - \frac{\delta_p}{2} - \frac{\delta_s}{2}}$$
$$d_s = \frac{\frac{\delta_s}{2}}{1 - \frac{\delta_p}{2} - \frac{\delta_s}{2}}$$

so that:

$$F_R(\Omega T) = \frac{E_R(\Omega T) + d_s}{1 + d_p + d_s}$$

The Octave script *johansson_cascade_allpass_bandstop_test.m* designs a pair of band-stop and band-pass filters. The band-pass filter is both all-pass and magnitude complementary to the band-stop filter. The band-stop filter specification is:

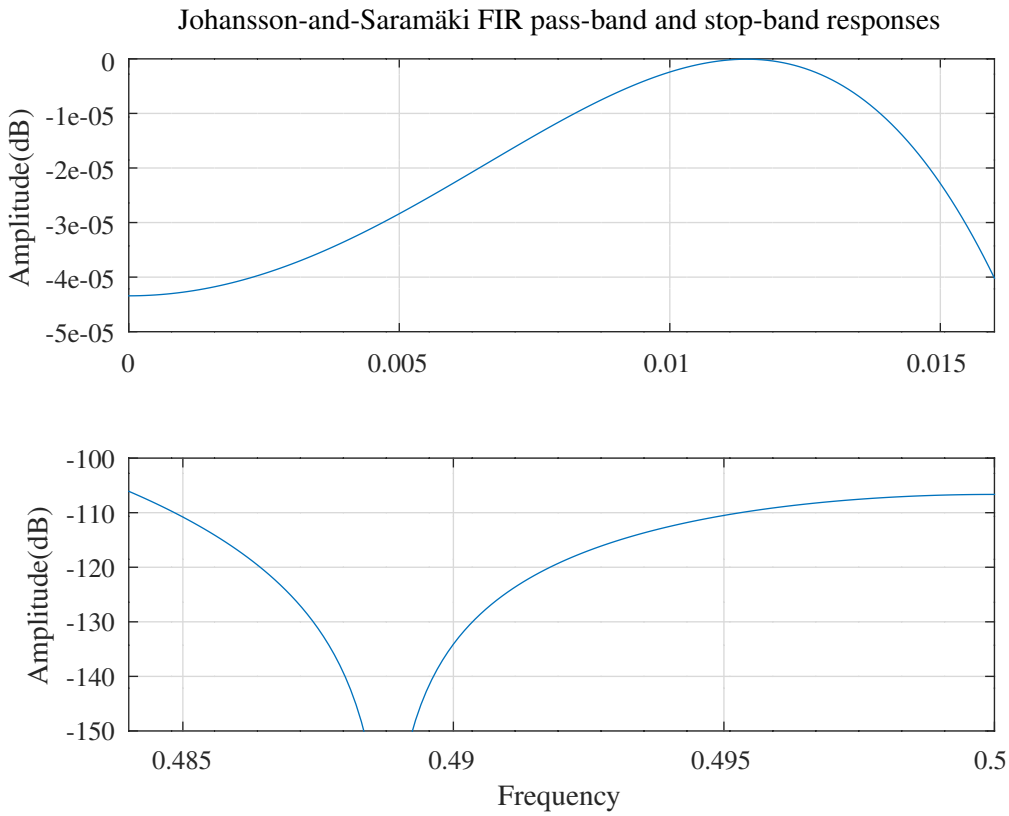



Figure M.21: Pass-band and stop-band responses of the Johansson and Saramäki prototype FIR filter

```
tol=1e-08 % Tolerance on coefficient update vector
ctol=1e-08 % Tolerance on constraints
nf=5000 % Frequency points across the band
M=6 % Prototype FIR filter order
Fap=0.016 % Prototype FIR pass-band amplitude response edge
delta_p=0.000005 % FIR pass-band amplitude response ripple
delta_s=0.000005 % FIR stop-band amplitude response ripple
fapl=0.15 % Pass-band amplitude response lower edge
fasl=0.2 % Stop-band amplitude response lower edge
fasu=0.25 % Stop-band amplitude response upper edge
fapu=0.3 % Pass-band amplitude response upper edge
```

The script designs the prototype FIR filter with the *directFIRsymmetric_slb* and *directFIRsymmetric_socp_mmse* functions. Figure M.21 shows the response of the prototype FIR filter. The prototype FIR filter coefficients are:

```
f1 = [  -0.0314881200,  -0.0000085599,   0.2814857078,   0.5000169443, ...
         0.2814857078,  -0.0000085599,  -0.0314881200 ]';
```

The script designs the prototype IIR filter as the low-pass to band-stop transformation of a parallel all-pass elliptic filter found with the *ellip* function from the Octave-Forge *signal* package. Figure M.22 shows the response of the prototype IIR filter. The prototype IIR filter parallel all-pass polynomial coefficients are:

```
bsA0 = [   1.0000000000,  -0.5650807120,   1.6504676367,  -0.4790677580, ...
           0.7284677906 ];

bsA1 = [   1.0000000000,  -0.2594846657,   0.6383217013 ];
```

Figure M.23 shows the pass-band and stop-band zero-phase response of the band-stop filter. Figure M.24 shows the responses of the complementary filters. Figure M.25 shows the pass-band and stop-band responses of the complementary filters.

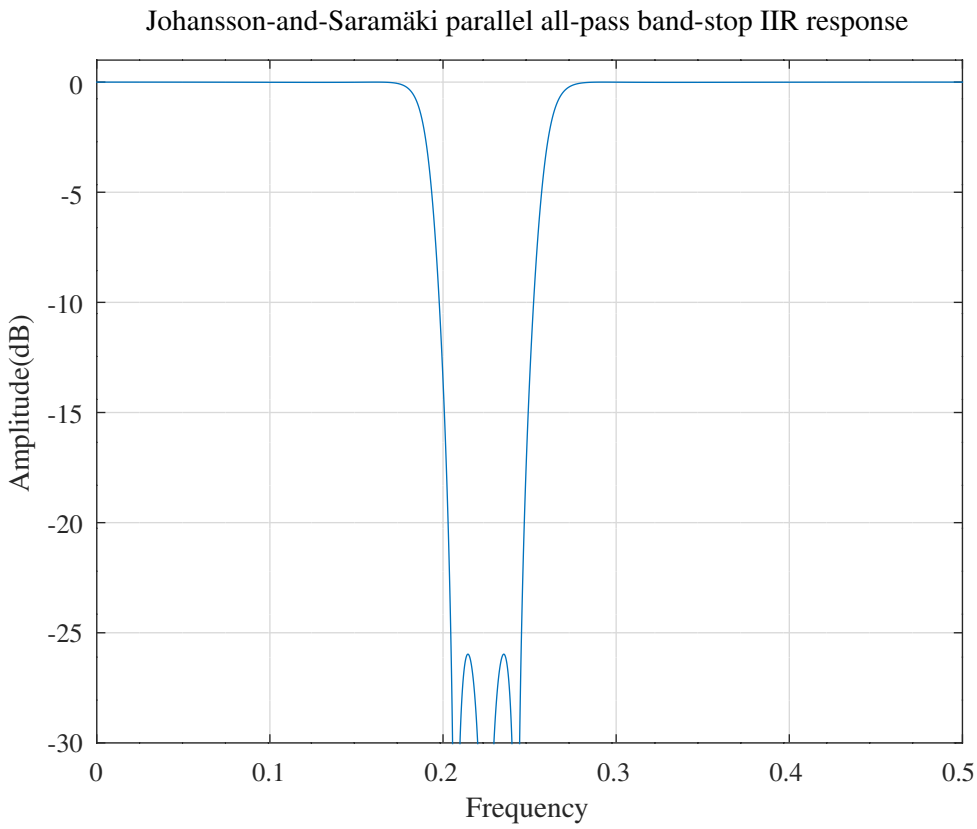



Figure M.22: Response of the Johansson and Saramäki prototype IIR filter

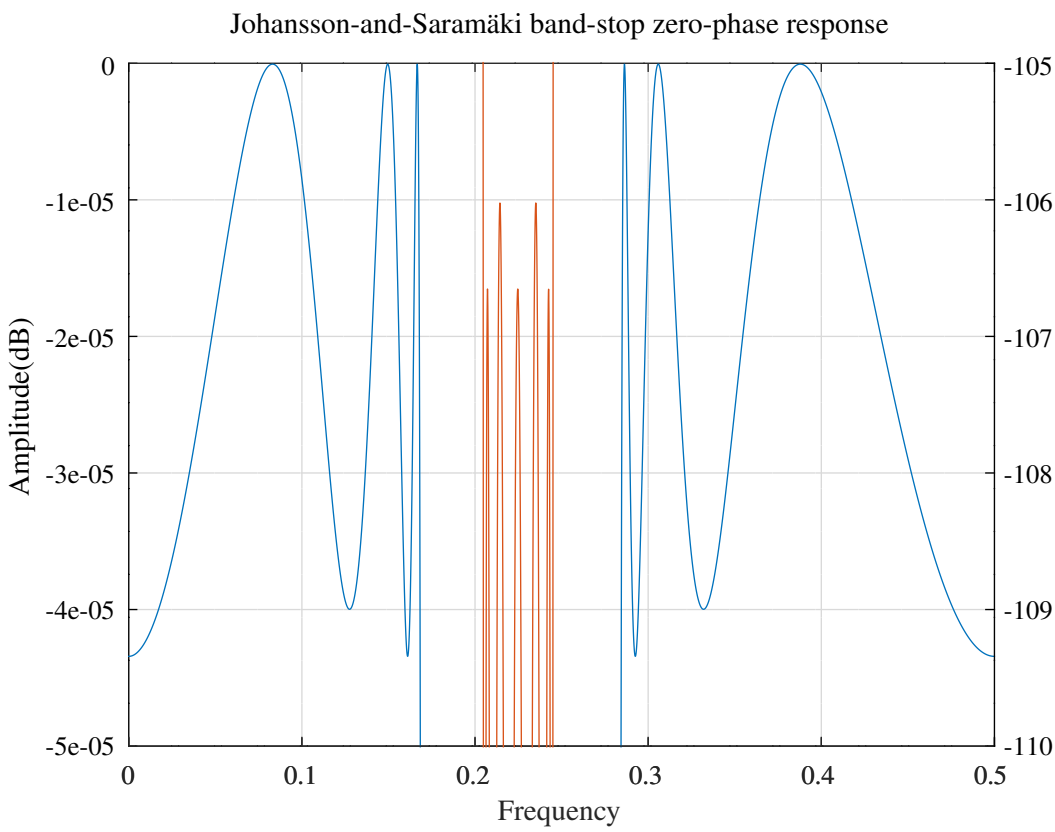



Figure M.23: Pass-band and stop-band zero-phase response of the Johansson and Saramäki band-stop IIR filter

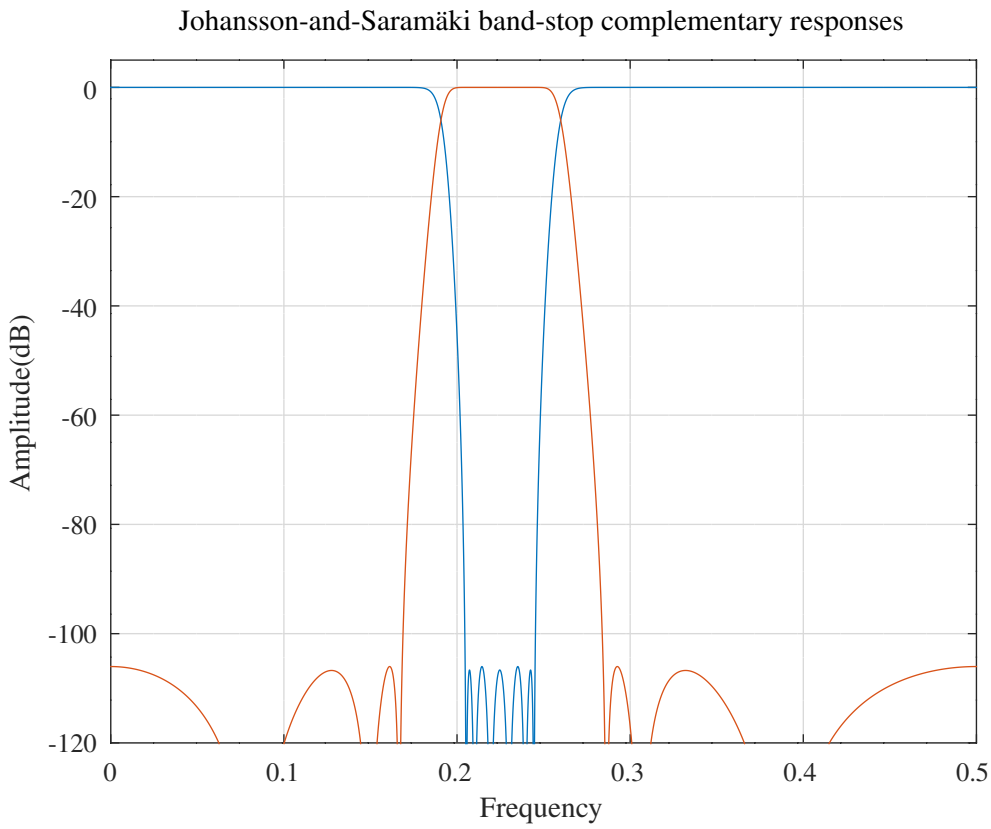



Figure M.24: Responses of the Johansson and Saramäki band-stop complementary IIR filters

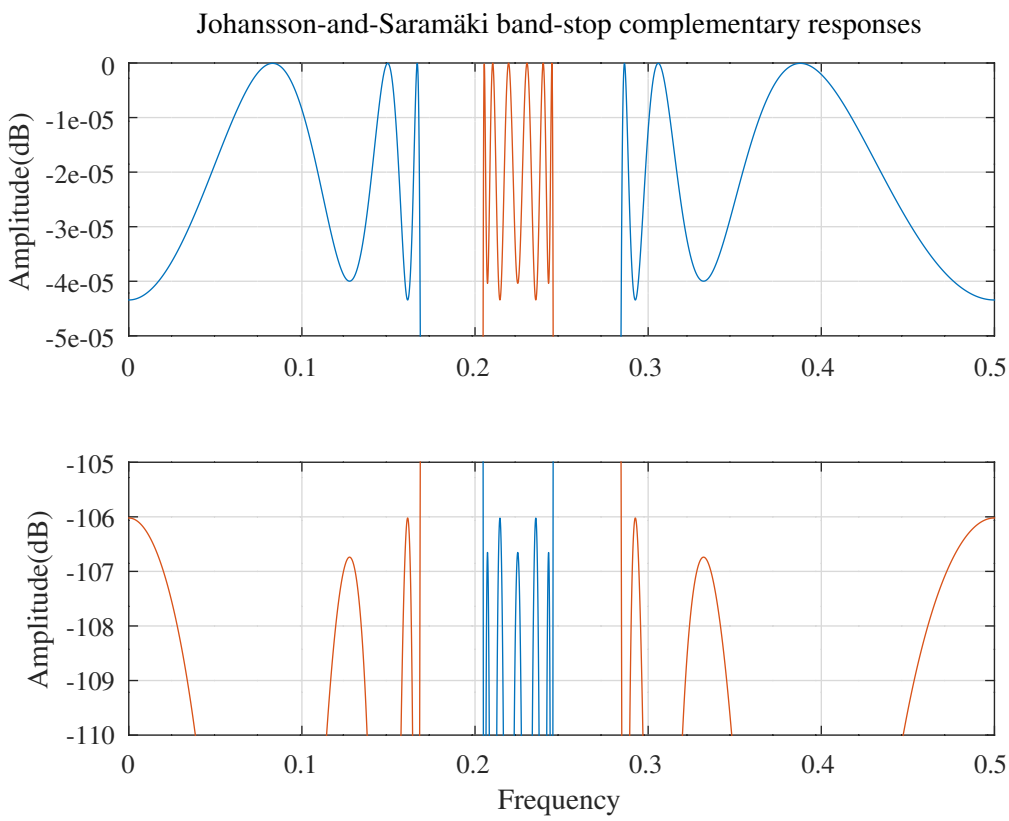



Figure M.25: Pass-band and stop-band responses of the Johansson and Saramäki band-stop complementary IIR filters

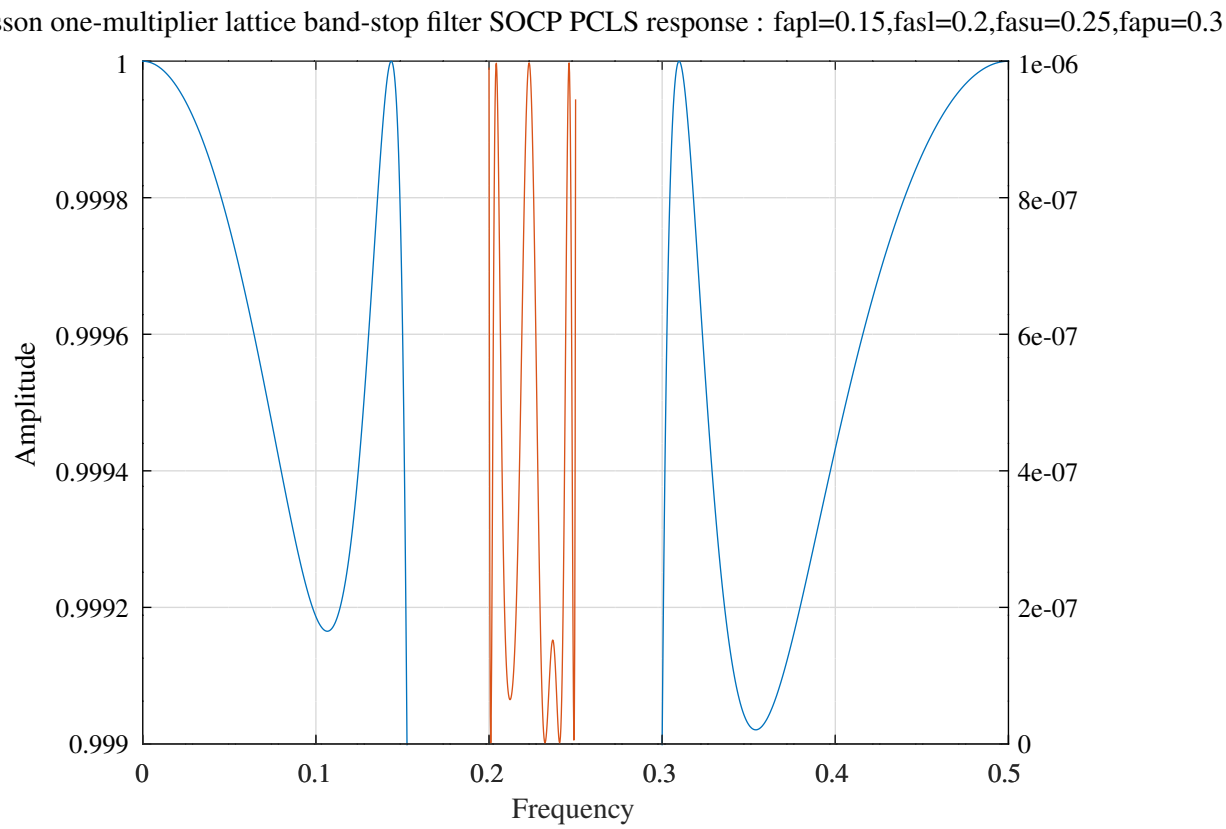



Figure M.26: Responses of the Johansson and Saramäki band-stop filter with PCLS constraints

The Octave script *johanssonOneMlattice_socp_slb_bandstop_test.m* designs a band-stop filter with PCLS constraints. The all-pass filters are implemented as Schur one-multiplier lattice filters. The band-stop filter specification is:

```
tol=1e-07 % Tolerance on coef. update
ctol=2e-08 % Tolerance on constraints
nf=2000 % Frequency points across the band
rho=0.992188 % Constraint on allpass coefficients
fapl=0.15 % Amplitude pass band lower edge
fasl=0.2 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
fapu=0.3 % Amplitude pass band upper edge
delta_p=0.001 % Amplitude pass band peak ripple
delta_s=1e-06 % Amplitude stop band peak ripple
Wap=1 % Amplitude pass band weight
Was=10 % Amplitude stop band weight
```

The initial band-stop filter is that designed by the Octave script *johansson_cascade_allpass_bandstop_test.m*. The resulting band-stop filter coefficients are:

```
f = [  -0.0290844866,   0.0045420742,   0.2790841989,   0.4909162740, ...
        0.2790841989,   0.0045420742,  -0.0290844866 ]';


k0 = [  -0.1871473030,   0.8123348300,  -0.1351134955,   0.6385521164 ]';


k1 = [  -0.1587800398,   0.4753385732 ]';
```

Figure M.26 shows the pass-band and stop-band responses of the filter.

The Octave script *branch_bound_johanssonOneMlattice_bandstop_test.m* designs a band-stop filter having 3-signed-digit 16-bit coefficients with the *branch-and-bound* algorithm (see Chapter 14). The all-pass filters are implemented as Schur one-multiplier lattice filters. The band-stop filter specification is:

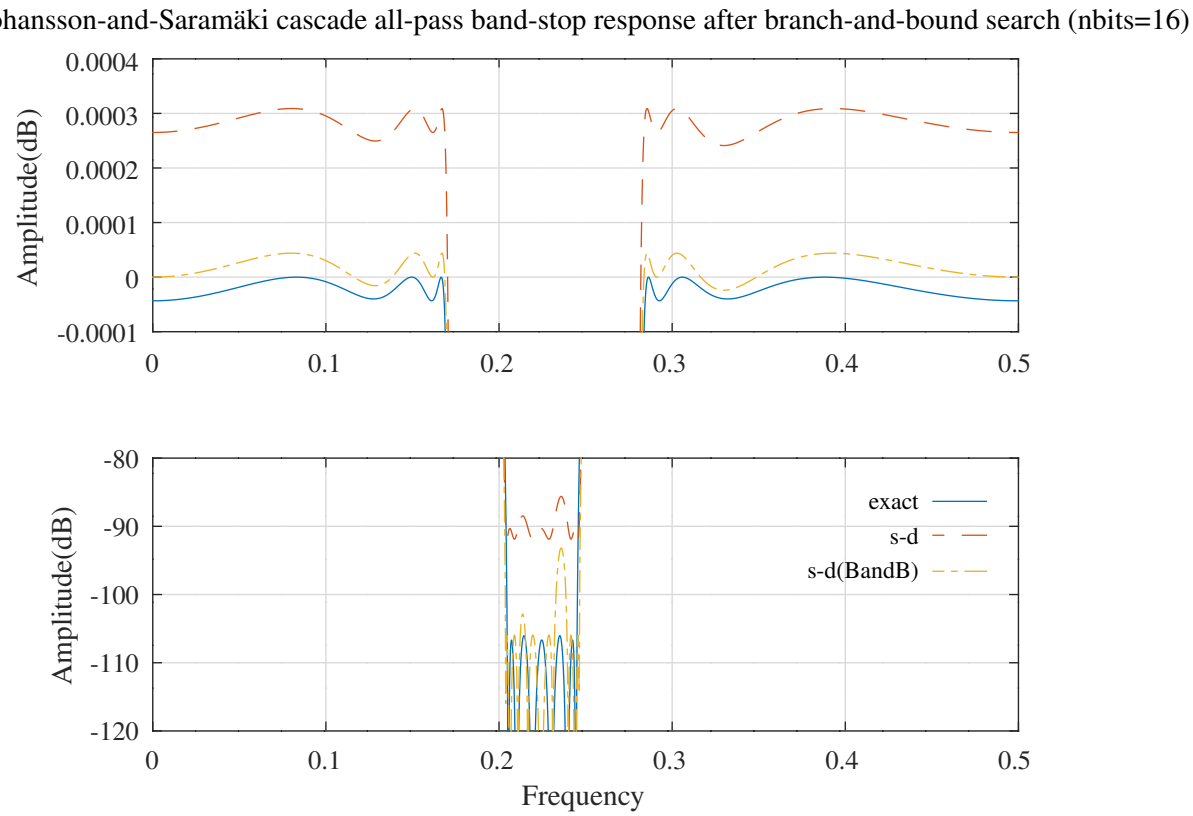



Figure M.27: Responses of the Johansson and Saramäki band-stop filter with 3-signed-digit, 16-bit coefficients designed with the *branch-and-bound* algorithm

```
nbits=16 % Coefficient bits
ndigits=3 % Coefficient signed-digits
nf=5000 % Frequency points across the band
fapl=0.15 % Amplitude pass band lower edge
fasl=0.2 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
fapu=0.3 % Amplitude pass band upper edge
Wap=1 % Amplitude pass band weight
Was=1 % Amplitude stop band weight
```

The initial band-stop filter is that designed by the Octave script *johansson_cascade_allpass_bandstop_test.m*. The resulting band-stop filter 3-signed-digit, 16-bit coefficients are:

```
f_min = [    -1032,        0,     9224,    16384, ...
              9224,        0,    -1032 ]'/32768;


k0_min = [    -5632,    29696,    -4736,    24064 ]'/32768;


k1_min = [    -5184,    20992 ]'/32768;
```

Figure M.27 shows the pass-band and stop-band responses of the filter.

# Appendix N

# Design of FIR digital filter transfer functions

This chapter reviews methods of designing FIR digital filter transfer functions.

## N.1 Low passband sensitivity FIR lattice filters

### N.1.1 Lattice decomposition of an FIR digital filter

*Vaidyanathan* [148] 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 $|H(e^{\imath\omega})| \leq 1$, and the bounded-real complementary filter, $G(z)$, with $|H(e^{\imath\omega})|^2 + |G(e^{\imath\omega})|^2 = 1$. Assume that $H_{m+1}(z)$ and $G_{m+1}(z)$ are complementary FIR filter transfer function polynomials of order $m+1$:

$$H_{m+1}(z) = \sum_{n=0}^{m+1} h_{m+1,n} z^{-n} \tag{N.1}$$

$$G_{m+1}(z) = \sum_{n=0}^{m+1} g_{m+1,n} z^{-n} \tag{N.2}$$

and write $\tilde{H}_{m+1}(z) = H_{m+1}^{\top}\left(z^{-1}\right)$, where $^{\top}$ denotes transpose.

$A_{m+1}(z) = [H_{m+1}(z)\ \ G_{m+1}(z)]^{\top}$ represents the corresponding all-pass filter:

$$\tilde{A}_{m+1}(z) A_{m+1}(z) = 1$$

or

$$\tilde{H}_{m+1}(z) H_{m+1}(z) + \tilde{G}_{m+1}(z) G_{m+1}(z) = 1 \tag{N.3}$$

After expanding Equation N.3 with Equations N.1 and N.2, the term in $z^{m+1}$ is, by inspection:

$$h_{m+1,m+1} h_{m+1,0} + g_{m+1,m+1} g_{m+1,0} = 0 \tag{N.4}$$

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:

$$H_m(z) = H_{m+1}(z)$$
$$G_m(z) = zG_{m+1}(z)$$

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 [148, Equations 13 and 14]:

$$k_{m+1} = \frac{-g_{m+1,m+1}}{\sqrt{h_{m+1,m+1}^2 + g_{m+1,m+1}^2}} \ , \ \hat{k}_{m+1} = \frac{h_{m+1,m+1}}{\sqrt{h_{m+1,m+1}^2 + g_{m+1,m+1}^2}} \tag{N.5}$$

or [148, Equation 23]:

$$k_{m+1} = \frac{h_{m+1,0}}{\sqrt{h_{m+1,0}^2 + g_{m+1,0}^2}} \ , \ \hat{k}_{m+1} = \frac{g_{m+1,0}}{\sqrt{h_{m+1,0}^2 + g_{m+1,0}^2}} \tag{N.6}$$

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 N.6. (I found that Equation N.6 has better numerical performance than Equation N.5).

### N.1.2 Finite-wordlength properties of the lattice FIR filter

Section N.1.3 shows that transfer function of the complementary FIR lattice is the result of $N$ orthogonal transformations:

$$\begin{bmatrix} H_N & G_N \end{bmatrix}^\top = \tau_N \cdots \tau_1 \begin{bmatrix} H_0 & G_0 \end{bmatrix}^\top$$

where the $\tau_m$ 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* [148][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 3.3.3 describes state scaling).
3. The coefficient sensitivities to $k_m$ are:

$$\frac{\partial}{\partial k_m} \begin{bmatrix} H_N & G_N \end{bmatrix}^\top = \tau_N \cdots \tau_{m+1} \begin{bmatrix} 1 & 0 \\ 0 & z^{-1} \end{bmatrix} \tau_{m-1} \cdots \tau_1 \begin{bmatrix} H_0 & G_0 \end{bmatrix}^\top$$

   so the coefficient sensitivities of the FIR lattice are bounded:

$$\left| \frac{\partial H_N}{\partial k_m} \right|^2 + \left| \frac{\partial G_N}{\partial k_m} \right|^2 = 1$$

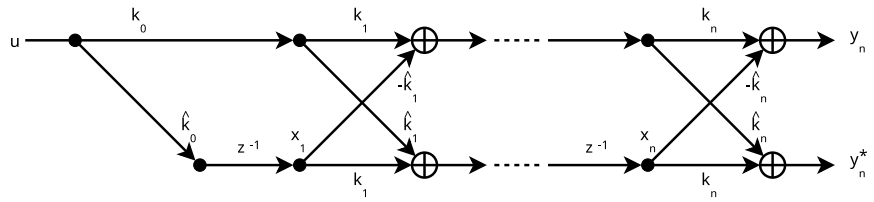



Figure N.1: Structure of the complementary FIR lattice filter (see *Vaidyanathan* [148, Figure 3])

**Algorithm N.1** Construction of a state variable description of the complementary FIR lattice filter

$y_0^* = \hat{k}_0 u$
$y_0 = k_0 u$
**for** $n = 1, \ldots, N$ **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**
$y = y_N$

### N.1.3 State variable description of the complementary FIR lattice filter

Figure N.1 (see Figure 3 of *Vaidyanathan* [148]) 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 N.1.

As shown in Section 1.12.2, 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.

### N.1.4 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 [63], is simpler and more accurate than the cepstral method of *Mian* and *Nainer* [54].

#### *Orchard* and *Willson*'s Newton-Raphson solution

If an order $N$ FIR filter $H(z)$ is bounded-real then $|H(e^{\imath\omega})| \leq 1$. The magnitude-squared complementary filter $1-H^\dagger(z)H(z)$, where † means *complex conjugate transpose*, is linear-phase and has double zeros on the unit-circle when $|H(e^{\imath\omega})| = 1$. *Orchard* and *Willson* [63] 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:

$$H(z) = z^{-3}\left[h_0 + h_1\left(z + z^{-1}\right) + h_2\left(z^2 + z^{-2}\right) + h_3\left(z^3 + z^{-3}\right)\right]$$

$H(z)$ is assumed to be the product of a minimum phase factor:

$$F_1(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}$$

and the corresponding maximum phase factor:

$$\begin{aligned} F_2(z) &= z^{-3} F_1\left(z^{-1}\right) \\ &= a_3 + a_2 z^{-1} + a_1 z^{-2} + a_0 z^{-3} \end{aligned}$$

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:

$$\begin{aligned} h_0 &= a_0^2 + a_1^2 + a_2^2 + a_3^2 \\ h_1 &= a_0 a_1 + a_1 a_2 + a_2 a_3 \\ h_2 &= a_1 a_3 + a_0 a_2 \\ h_3 &= a_0 a_3 \end{aligned}$$

The Newton-Raphson method solves a system of equations $f(\boldsymbol{a}) = 0$ by successive approximations:

$$f(\boldsymbol{a}_k) + J(\boldsymbol{a}_k)\left[\boldsymbol{a}_{k+1} - \boldsymbol{a}_k\right] = 0$$

or:

$$\boldsymbol{a}_{k+1} = \boldsymbol{a}_k - J^{-1}(\boldsymbol{a}_k) f(\boldsymbol{a}_k)$$

where $J(\boldsymbol{a})$ is the Jacobian matrix of $f(\boldsymbol{a})$.

In this case:

$$f(\boldsymbol{a}) = \begin{bmatrix} h_0 - a_0^2 - a_1^2 - a_2^2 - a_3^2 \\ h_1 - a_0 a_1 - a_1 a_2 - a_2 a_3 \\ h_2 - a_1 a_3 - a_0 a_2 \\ h_3 - a_0 a_3 \end{bmatrix}$$

$$
\begin{aligned}
J(\boldsymbol{a}) &= -\begin{bmatrix} 2a_0 & 2a_1 & 2a_2 & 2a_3 \\ a_1 & a_2+a_0 & a_3+a_1 & a_2 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & 0 & 0 & a_0 \end{bmatrix} \\
&= -\begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 & 0 \\ a_2 & a_3 & 0 & 0 \\ a_3 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ 0 & a_0 & a_1 & a_2 \\ 0 & 0 & a_0 & a_1 \\ 0 & 0 & 0 & a_0 \end{bmatrix}
\end{aligned}
$$

The function *minphase.m* solves for the coefficient update, $\boldsymbol{d}_{k+1} = \boldsymbol{a}_{k+1} - \boldsymbol{a}_k$, by matrix left division. The C++ file *minphase.cc* implements the *minphase* algorithm as an *oct-file* using the *Eigen* [58] 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*.

### *Mian* and *Nainer*'s cepstral method

*Mian* and *Nainer* [54] 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, $\boldsymbol{x}$, is

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n}$$

and that the inverse $z$-transform is

$$x(n) = \frac{1}{2\pi\imath} \oint_C X(z) z^{n-1} dz$$

where the contour, $C$, is centred on the origin and lies within the region of convergence. Cauchy's integral lemma is

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

The *complex cepstrum* is the inverse z-transform of $\hat{X}(z) = \log X(z) = \log |X(z)| + \imath \arg X(z)$ on the unit circle:

$$\hat{x}(n) = \frac{1}{2\pi\imath} \oint_C \log X(z) z^{n-1} dz$$

The *real cepstrum* is the inverse z-transform of $\log |X(z)|$ on the unit circle.

For example, suppose $X(z) = K\left(1 - az^{-1}\right)(1 - bz)$, 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:

$$\hat{x}(n) = \frac{1}{2\pi\imath} \oint_C \left[\log K + \log\left(1 - az^{-1}\right) + \log(1 - bz)\right] z^{n-1} dz$$

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 $\boldsymbol{x}$ is minimum-phase then the cepstrum, $\hat{\boldsymbol{x}}$, is causal. Similarly, if $\boldsymbol{x}$ is maximum-phase then the cepstrum is anti-causal.

The complex cepstrum is related to the original sequence, $\boldsymbol{x}$, by a recursion. In the z-domain:

$$\frac{d}{dz}\hat{X}(z) = \frac{1}{X(z)}\frac{d}{dz}X(z)$$
$$\left[-z\frac{d}{dz}\hat{X}(z)\right]X(z) = -z\frac{d}{dz}X(z)$$

So, in the time domain:

$$n\hat{x}(n) \star x(n) = nx(n)$$

In other words:

$$x(n) = \begin{cases} e^{\hat{x}(0)} & n = 0 \\ \sum_{k=-\infty}^{\infty} \frac{k}{n}\hat{x}(n)\, x(n-k) & n \neq 0 \end{cases} \tag{N.7}$$

**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 † means *complex conjugate transpose*. By construction, $F(z)$ is real, even, $0 \leq F(e^{\imath\omega}) \leq 1$ 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 N.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 circle[a]:

```
% 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=ifft(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]);
```

[a]Note 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 $z^{-N}$.

### *Sturm*'s example of minimum phase spectral factorisation by semi-definite programming

*Sturm* [195, Equation 13, Section 3.2] claims that the solution to the following semi-definite programming problem is a minimum phase spectral factorisation, $x_k$, of an autocovariance sequence, $r_k$:

$$\begin{aligned} \textbf{minimise} \quad & \sum_{l=1}^{m} (m-l)\, x_{l,l} \\ \textbf{subject to} \quad & \sum_{l=1}^{m-k} x_{l,l+k} = r_k \quad k = 0, \ldots, m-1 \\ & X \succ 0 \end{aligned}$$

where $X$ is a symmetric matrix with elements $x_{k,l}$. Unfortunately, *Sturm*'s reference justifying this statement is not accessible but the associated website states that similar work is described by *Dumitrescu et al.* [21] and *Dumitrescu* [19].

*Sturm* contributed a spectral factorisation example to the 7'th DIMACS challenge [31]. It is reproduced as the Octave script *sedumi_minphase_test.m* which solves the following problem:

$$\begin{aligned} \textbf{minimise} \quad & \sum_{l=1}^{n} r_l y_l \\ \textbf{subject to} \quad & \text{Toeplitz}\left[y_1, \frac{1}{2}y_2, \ldots, \frac{1}{2}y_n\right] - \text{diag}\left[n-1, \ldots, 1, 0\right] \succeq 0 \end{aligned}$$

I have reversed the sign of the term in $[n-1, \ldots, 1, 0]$ so that the zeros of the complementary filter are inside the unit-circle.

### *Tuqan* and *Vaidyanathan*'s semi-definite programming solution

*Tuqan* and *Vaidyanathan* [85] propose a state-space approach to solving the optimisation problem:

$$\begin{aligned} \textbf{maximise} \quad & \int_{-\pi}^{\pi} |H\left(e^{\imath\omega}\right)|^2 W\left(e^{\imath\omega}\right) \\ \textbf{subject to} \quad & \frac{1}{M}\sum_{k=0}^{M-1} |H\left(e^{\imath(\omega-2\pi k)/M}\right)|^2 = |H\left(e^{\imath\omega}\right)|^2|_{\downarrow M} \end{aligned}$$

$H(\omega)$ is an order $N$ FIR filter, known as the *compaction filter*, and $W(\omega)$ is a weighting function. $M$ is the decimation factor applied to the output of the filter $H(z)$. $H(z)$ is to be used in an orthonormal filter-bank with $H_k\left(e^{\imath\omega}\right) H_l\left(e^{\imath\omega}\right) = \delta_{k-l}$. Assume that the input to the filter is a zero-mean Gaussian noise sequence, $x_n$, with power spectrum $S_{xx}\left(e^{\imath\omega}\right)$, autocorrelation sequence, $r_n$, that the output sequence is $y_n$ and that $F(z) = H(z)\, H\left(z^{-1}\right)$. Setting $W(\omega) = S_{xx}(\omega)$, the objective function is the variance of the output sequence:

$$\sigma_y^2 = r_0 + 2\sum_{n=1}^{N} f_n r_n$$

and the constraints are:

$$\begin{aligned} f_{Mn} &= \delta_n \\ F(\omega) &= 1 + 2\sum_{n=1}^{N} f_n \cos\omega n \;\geq 0\; \forall\omega \end{aligned}$$

Suppose $D(z)$ is implemented as a direct form FIR filter with the state space representation:

$$\begin{aligned} A_d &= \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \\ B_d &= \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix}^\top \\ C_d &= \begin{bmatrix} f_N & \cdots & f_1 \end{bmatrix} \\ D_d &= \frac{1}{2} \end{aligned}$$

The *discrete time positive real lemma*, a corollary of the *Kalman-Yakubovitch-Popov lemma*, states that $D(z)$ is positive real if-and-only-if there exists a real, symmetric, positive definite matrix, $P_d$, and real matrixes $W_d$ and $L_d$ such that:

$$\begin{aligned} P_d - A_d^\top P_d A_d &= L_d^\top L_d \\ C_d - A_d^\top P_d B_d &= L_d^\top W_d \\ D_d + D_d^\top - B_d^\top P_d B_d &= W_d^\top W_d \end{aligned}$$

By substitution [85, Theorem 1] $H(z) = W_d + L_d \left(zI - A_d^{-1}\right) B_d$ is seen to be a spectral factor of $F(z)$. The minimum phase solution, $H_{min}(z)$, is that for which $P_{d_{min}}$ is the minimum element of the convex set of symmetric positive definite matrixes satisfying the LMI. *Tuqan* and *Vaidyanathan* show [85, Theorem 3] that, if $D_d + D_d^\top - B_d^\top P_d B_d \neq 0$, then $P_{d_{min}}$ is the unique solution of the following *algebraic Riccati equation*:

$$P_d = A_d^\top P_d A_d + \left(C_d^\top - A_d^\top P_d B_d\right)\left(D_d + D_d^\top - B_d^\top P_d B_d\right)^{-1}\left(C_d^\top - A_d^\top P_d B_d\right)^\top$$

or:

$$\begin{aligned} P_d &= A_1^\top P_d A_1 + A_1^\top P_d B_d \left(R - B_d^\top P_d B_d\right)^{-1} B_d^\top P_d A_1 + C_d^\top R^{-1} C_d \\ A_1 &= A_d - B_d R^{-1} C_d \\ R &= D_d + D_d^\top \succ 0 \end{aligned}$$

In this case, the autocorrelation function, $\{D_d, C_d\}$, of the complementary filter is known in advance and the initial semi-definite programming solution for $P_d$ and $C_d$ is not required. In addition, the filter is known to be SISO so that, for the minimum phase solution, $H_{min}(z)$:

$$\begin{aligned} W_d &= \left(D_d + D_d^\top - B_d^\top P_{d_{min}} B_d\right)^{\frac{1}{2}} \\ L_d &= W_d^{-1}\left(C_d - B_d^\top P_{d_{min}} A_d\right) \end{aligned}$$

### N.1.5 Example: the minium-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 filter specification is:

```
tol=1e-06 % tolerance on results
N=31 % filter order
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
Wasl=2 % Stop band amplitude response lower weight
Wap=1 % Pass band amplitude response weight
Wasu=2 % Stop band amplitude response upper weight
```

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 ($eps = 2.2204e-16$). The bandpass and complementary filters are:

```
brz = [ -0.005108333779,  0.003660671781, -0.012383396144, -0.006947750356, ...
         0.021448945190,  0.042973631538,  0.025415530577, -0.008724449666, ...
        -0.004783022289,  0.027072600200,  0.000746818465, -0.110359730794, ...
        -0.181875048113, -0.073706903519,  0.156538506093,  0.277521749504, ...
         0.156538506093, -0.073706903519, -0.181875048113, -0.110359730794, ...
         0.000746818465,  0.027072600200, -0.004783022289, -0.008724449666, ...
         0.025415530577,  0.042973631538,  0.021448945190, -0.006947750356, ...
        -0.012383396144,  0.003660671781, -0.005108333779 ]';
```

```
brzc = [  0.702278455158, -0.284747150190,  0.195991006199,  0.310333248307, ...
          0.135915839465, -0.007622203880,  0.013539408392,  0.068315146105, ...
          0.019358425439, -0.088556180208, -0.123568391645, -0.052846068256, ...
          0.035521731841,  0.060100581877,  0.029846895503,  0.000251446156, ...
         -0.003826320303,  0.001544785942, -0.000527786189, -0.006517762705, ...
         -0.006982385693, -0.002252609052,  0.001415579659,  0.001682492398, ...
          0.000733967870,  0.000312597850,  0.000190007105, -0.000036513338, ...
         -0.000173379619,  0.000038188995, -0.000037157731 ]';
```

The reflection coefficients of the bandpass and complementary filters are:

```
k = [  0.999973545923,  0.999997439113,  0.999892198494,  0.999912068377, ...
       0.999560459113,  0.996590306134,  0.997582105159,  0.999848632328, ...
       0.997255806727,  0.999503325334,  0.999993493602,  0.993567986529, ...
       0.962834847250,  0.987076261596,  0.964636965455,  0.863342901944, ...
       0.964636965455,  0.987076261596,  0.962834847250,  0.993567986529, ...
       0.999993493602,  0.999503325334,  0.997255806727,  0.999848632328, ...
       0.997582105159,  0.996590306134,  0.999560459113,  0.999912068377, ...
       0.999892198494,  0.999997439113, -0.007273751039 ]';
```

```
khat = [  0.007273751039, -0.002263132143,  0.014683030694,  0.013261052526, ...
         -0.029646055025, -0.082509161431, -0.069497794685,  0.017398633019, ...
          0.074032803197,  0.031513531157, -0.003607319586,  0.113237167684, ...
          0.270090830870,  0.160251220882, -0.263582102727, -0.504617710413, ...
         -0.263582102727,  0.160251220882,  0.270090830870,  0.113237167684, ...
         -0.003607319586,  0.031513531157,  0.074032803197,  0.017398633019, ...
         -0.069497794685, -0.082509161431, -0.029646055025,  0.013261052526, ...
          0.014683030694, -0.002263132143,  0.999973545923 ]';
```

Figure N.2 shows the simulated FIR lattice bandpass and complementary filter amplitude responses. Figure N.3 shows the zeros of the complementary filter.

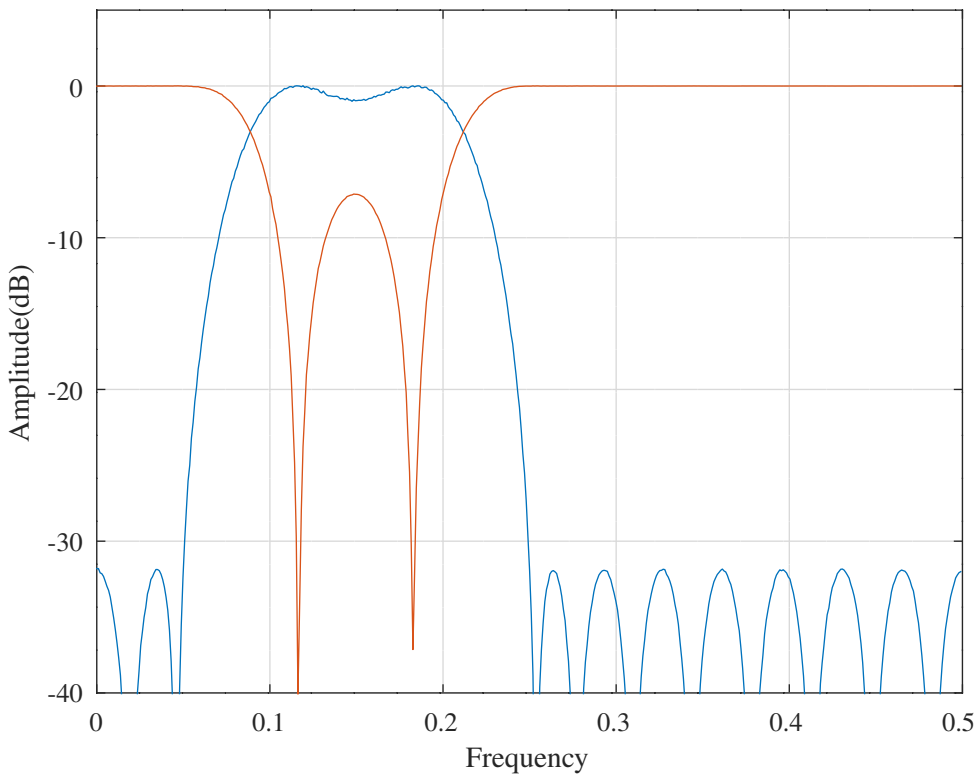



Figure N.2: Simulated amplitude responses of the FIR lattice band-pass filter and the minimum-phase complementary filter found with the Newton-Raphson method of *Orchard* and *Willson* [63].

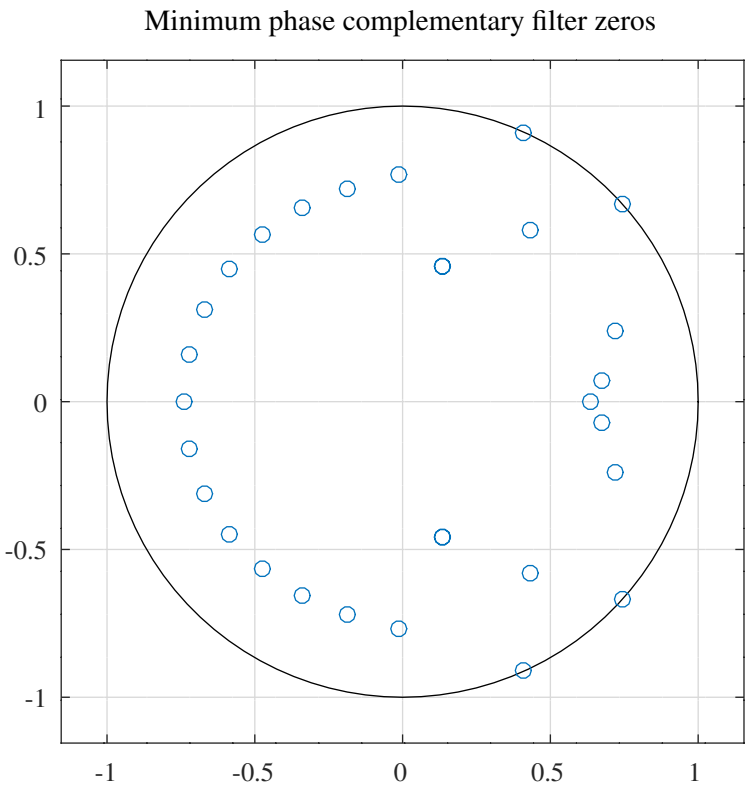



Figure N.3: Zeros of the complementary filter found with the Newton-Raphson method of *Orchard* and *Willson* [63].

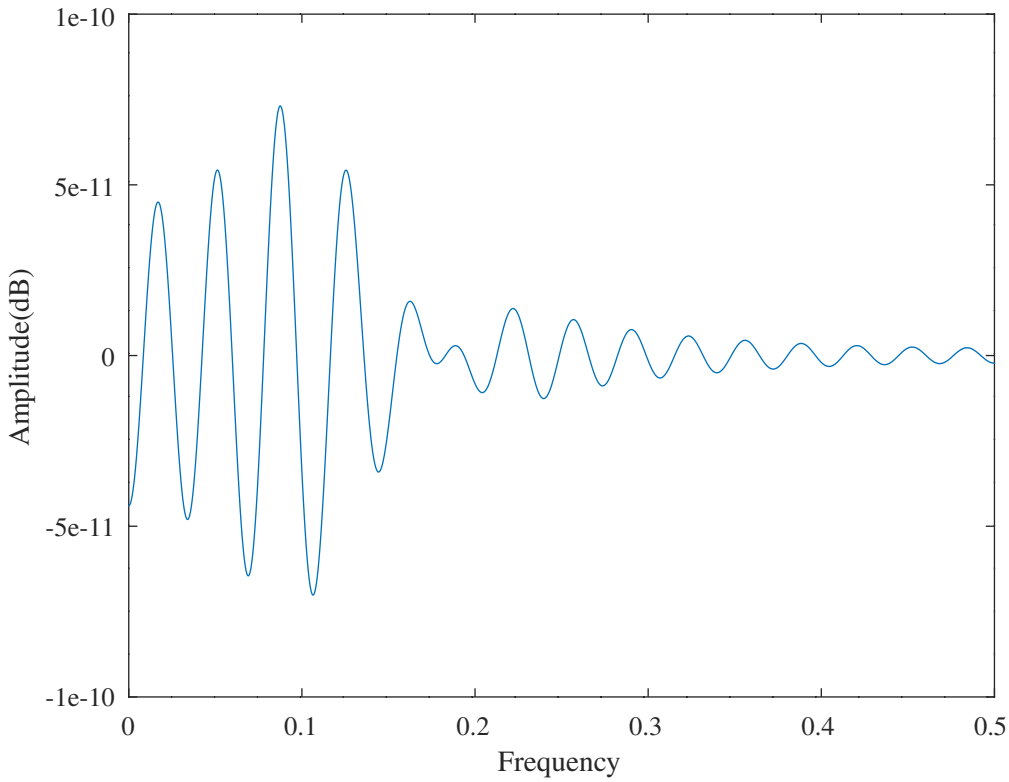



Figure N.4: Combined amplitude responses of the FIR band-pass filter and the minimum-phase complementary filter found with the cepstral method of *Mian* and *Nainer* [54]

Unfortunately, in the case of the cepstral method, $F(\alpha z)$ with $\alpha \approx 1.15$ is real and even but not positive. Fortuitously, 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 N.4.

The Octave script *directFIRnonsymmetric_sdp_minimum_phase_test.m* finds the complementary filter by following the semi-definite programming method of *Sturm* [195, Section 3.2]. Figure N.5 shows the resulting combined response.

The Octave script *tuqanFIRnonsymmetric_dare_minimum_phase_test.m* attempts to find the complementary filter by following the Riccati equation method of *Tuqan* and *Vaidyanathan* [85, Equation 38]. The Riccati equation is solved with the *dare* function from the Octave-Forge *control* toolbox [136]. Figure N.6 shows the resulting complementary filter response.

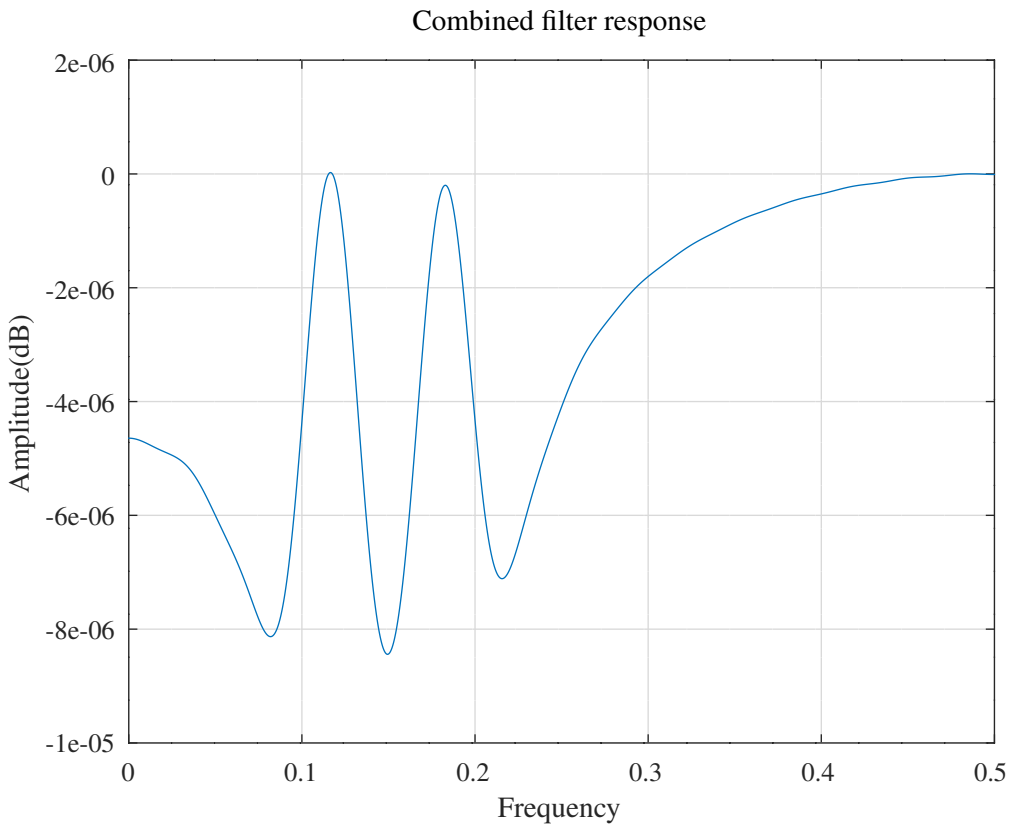



Figure N.5: Combined amplitude responses of the FIR band-pass filter and the complementary filter found with the semi-definite programming method of *Sturm* [195, Section 3.2].

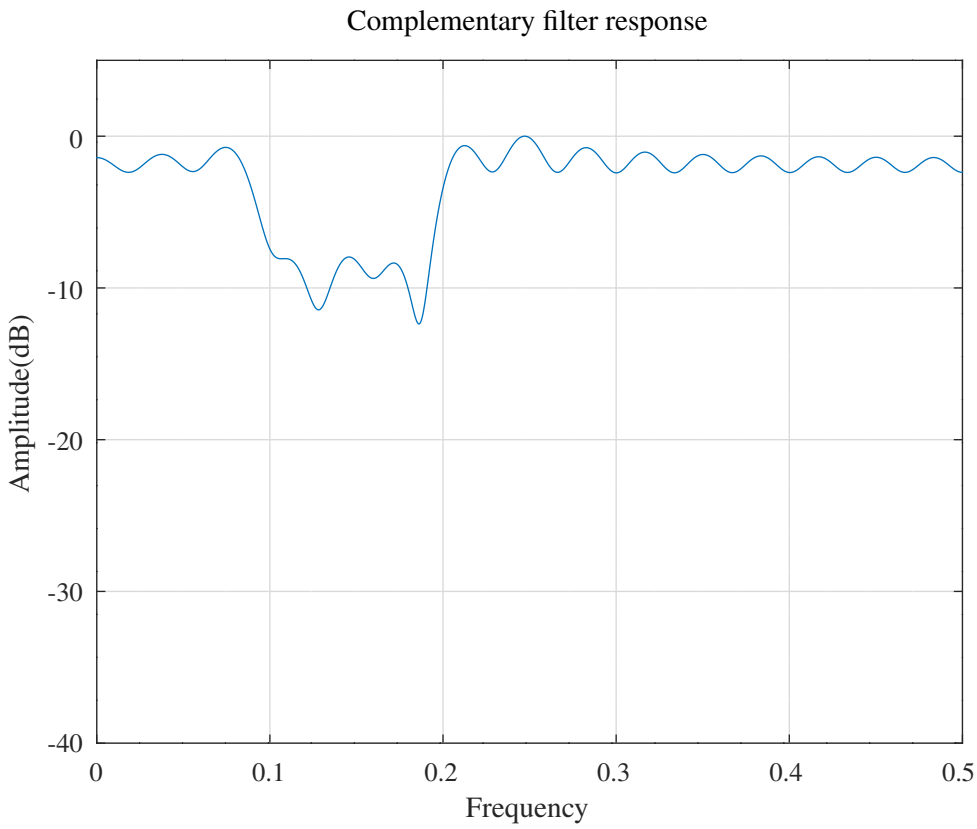



Figure N.6: Response of the complementary FIR filter found with the Riccati equation method of *Tuqan* and *Vaidyanathan* [85, Equation 38]

### N.1.6 Estimating the MA coefficients of a filtered noise sequence

This Section follows the description by *Dumitrescu et al* [21] of the estimation of the MA (or FIR) coefficients of a filtered noise sequence. It is intended to provide some justification for *Sturm*'s spectral factorisation example shown in Section N.1.4.

#### Autocovariance of a noise sequence

Suppose we have $N$ samples, $\hat{x}_l$ of a real valued stationary random time series, $x_l$:

$$\hat{x}_l = \begin{cases} x_l & 1 \leq l \leq N \\ 0 & \text{otherwise} \end{cases}$$

An estimate of the autocovariance[b] of $\hat{x}_l$ is[c]:

$$\begin{aligned} \hat{r}_k &= \frac{1}{N} \sum_{l=1}^{N} \hat{x}_l \hat{x}_{l+|k|} \\ &= \begin{cases} \frac{1}{N} \sum_{l=1}^{N-|k|} x_l x_{l+|k|} & |k| < N \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The expectation, or mean, of this estimate is:

$$\begin{aligned} \mathcal{E}\left\{\hat{r}_k\right\} &= \frac{1}{N} \sum_{l=1}^{N-|k|} \mathcal{E}\left\{x_l x_{l+|k|}\right\} \\ &= \begin{cases} (N-|k|)\, r_k & |k| < N \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

where $r_k$ is the autocovariance function of $x_l$. This definition of the estimator of the autocovariance, $\hat{r}_k$, is biased because $\mathcal{E}\left\{\hat{r}_k\right\} \neq r_k$. An unbiased estimator is:

$$\check{r}_k = \frac{1}{N-|k|} \sum_{l=1}^{N-|k|} x_l x_{l+|k|} \quad |k| < N$$

#### Autocovariance of the response of an FIR filter to white noise

Given an FIR filter transfer function, $H(z) = \sum_{k=0}^{n} h_k z^{-k}$, with $y_k = \sum_{l=0}^{n} h_l e_{k-l}$ where $e_k$ is zero mean, unit variance, white Gaussian noise, then the autocovariance sequence of $y_k$ is [21, Equation 2 II]:

$$\begin{aligned} r_k &= \mathcal{E}\left\{y_m y_{m-k}\right\} \\ &= \mathcal{E}\left\{\sum_{l=0}^{n} h_l e_{m-l} \sum_{p=0}^{n} h_p e_{m-k-p}\right\} \\ &= \mathcal{E}\left\{\sum_{l=0}^{n} \sum_{p=0}^{n} h_l h_p e_{m-l} e_{m-k-p}\right\} \\ &= \sum_{l=0}^{n} \sum_{p=-k}^{n-k} h_l h_{p+k} \mathcal{E}\left\{e_{m-l} e_{m-p}\right\} \\ &= \sum_{l=0}^{n-k} h_l h_{l+k} \end{aligned}$$

[b] *covariance* and *correlation* are similar concepts. *correlation* is unit-less and varies between $-1$ and 1.

[c] Note that for $0 \leq k < N$:

$$\hat{r}_{-k} = \frac{1}{N} \sum_{l=1}^{N} \hat{x}_l \hat{x}_{l-k} = \frac{1}{N} \sum_{l=1-k}^{N-k} \hat{x}_{l+k} \hat{x}_l = \frac{1}{N} \sum_{l=1}^{N-k} x_{l+k} x_l$$

$$
\begin{aligned}
&= \sum_{l=k}^{n} h_l h_{l-k} \\
&= h^\top A_k h \\
&= \operatorname{trace} A_k h h^\top
\end{aligned}
$$

for $k = 0, \ldots, n$, $r_{-k} = r_k$, $r_k = 0$ for $|k| > n$, $h = [h_0, h_1, \ldots, h_n]$ and $A_k$ denotes the matrix with unit entries on the $k$th diagonal.

### Spectral factorisation of the autocovariance sequence

The following constraints on the polynomial $R(z) = \sum_{k=-n}^{n} r_k z^{-k}$ are equivalent [21, Section II]:

- $R(z)$ has a spectral factorisation $R(z) = H(z) H(-z)$
- the positiveness of $R(e^{\imath\omega})$ is equivalent to the real positiveness of

$$
R_+(e^{\imath\omega}) = \Re\left\{\frac{r_0}{2} + \sum_{k=1}^{n} r_k e^{-\imath k\omega}\right\} \geq 0
$$

- the *positive real* condition on a polynomial

$$
R_+(z) = \frac{r_0}{2} + \sum_{k=1}^{n} r_k z^{-k}
$$

  is defined, for the trace parameterisation, as $r_k = \operatorname{trace} A_k Q$ where $Q$ is an $(n+1) \times (n+1)$ positive semi-definite matrix.

*Dumitrescu et al.* remark that, since $Q$ is positive semidefinite, it may be expressed as $Q = GG^\top$ where $G \in \mathbb{R}^{(n+1)\times\nu}$, $\nu = \operatorname{rank} Q$ and $G(z)$ represents an infinite family of SIMO spectral factors of $R(z)$.

### Primal formulation of the covariance estimation problem

*Dumitrescu et al.* [21] show algorithms for estimating the MA (or FIR) parameters of a noisy sequence. In this Section I follow *Dumitrescu et al.*'s treatment of the, so-called, *trace parameterisation*[d]. *Dumitrescu et al.* begin by defining a *positive real* sequence $r_k$:

$$
r_0 + \sum_{k=1}^{n} r_k \cos k\omega \geq 0
$$

for all $\omega \in [0, \pi]$ and $r_k \in \mathbb{R}$. Given $N$ samples of the filter output $y_1, \ldots, y_N$, the covariance estimates

$$
\hat{r}_k = \frac{1}{N} \sum_{l=k+1}^{N} y_l y_{l-k}
$$

have variance-covariances [153, Section II] $\mathcal{E}\{[\hat{r}_k - r_k][\hat{r}_l - r_l]\}$, which can be approximated by [205, Equation B.48]:

$$
\hat{W}_{kl} \approx \frac{1}{N^2} \sum_{m=-n}^{n} (N - |m|)\left[\hat{r}_m \hat{r}_{m+k-l} + \hat{r}_{m-l}\hat{r}_{m+k}\right] \tag{N.8}
$$

where $k, l = 0, 1, \ldots$ and $N \gg n$.

*Dumitrescu et al.* [21, Section III] define the $(n + n_\gamma + 1) \times (n + n_\gamma + 1)$ matrix $\hat{W}$ with entries $\hat{W}_{kl}$ and the vectors $\hat{r} = [\hat{r}_0, \ldots, \hat{r}_n]$ and $\hat{\gamma} = \left[\hat{r}_{n+1}, \ldots, \hat{r}_{n+n_\gamma}\right]$. Following *Stoica et al.* [153, Equation 7], they propose the weighted least squares fitting criterion:

$$
f = \frac{1}{2}\begin{bmatrix} \hat{r} - r \\ \hat{\gamma} \end{bmatrix}^\top \hat{W}^{-1} \begin{bmatrix} \hat{r} - r \\ \hat{\gamma} \end{bmatrix}
$$

[d] The main purpose of their paper is to show that the trace parameterisation is equivalent to a *Kalman-Yakubovich-Popov* lemma parameterisation. In addition they describe the *primal* and *dual* solutions for each parameterisation.

and suggest that $n_\gamma = \sqrt{N}$. If $\hat{W}$ is partitioned as

$$\hat{W} = \begin{bmatrix} \hat{W}_{11} & \hat{W}_{12} \\ \hat{W}_{12}^\top & \hat{W}_{22} \end{bmatrix}$$

where $\hat{W}_{11}$ is $(n+1) \times (n+1)$, then the fitting criterion becomes:

$$f = \frac{1}{2} (r - \tilde{r})^\top \Gamma^{-1} (r - \tilde{r}) + \text{ const.}$$

where

$$\begin{aligned} \tilde{r} &= \hat{r} - \hat{W}_{12} \hat{W}_{22}^{-1} \hat{\gamma} \\ \Gamma &= \hat{W}_{11} - \hat{W}_{12} \hat{W}_{22}^{-1} \hat{W}_{12}^\top \end{aligned}$$

In the trace parameterisation, the problem of estimating the covariances becomes [21, Equation 17]:

$$\begin{aligned} \textbf{minimise} \quad & \frac{1}{2} (r - \tilde{r})^\top \Gamma^{-1} (r - \tilde{r}) \\ \textbf{subject to} \quad & r_k = \operatorname{trace} A_k Q, \quad k = 0, \ldots, n \\ & Q \succ 0 \end{aligned} \tag{N.9}$$

**Dual formulation of the covariance estimation problem**

The Lagrangian of the primal problem is

$$\begin{aligned} L(r, Q, \mu, X) &= f(r) - \operatorname{trace} XQ + \sum_{k=0}^{n} \mu_k (\operatorname{trace} A_k Q - r_k) \\ &= f(r) + \operatorname{trace} \left[ -X + \left( \sum_{k=0}^{n} \mu_k A_k \right) Q \right] - \mu^\top r \\ &= f(r) + \frac{1}{2} \operatorname{trace} \left[ -2X + \left( \sum_{k=0}^{n} \mu_k \left( A_k + A_k^\top \right) \right) Q \right] - \mu^\top r \end{aligned}$$

where the positive semidefinite symmetric matrix, $X$, is the Lagrange multiplier for the linear matrix inequality $Q \succeq 0$. Differentiating the Lagrangian with respect to $r$:

$$\frac{\partial L(r, Q, \mu, X)}{\partial r} = \Gamma^{-1} (r - \tilde{r}) - \mu$$

so that the solution to the primal problem is $r^\star = \tilde{r} + \Gamma \mu$. The terms of the Lagrangian in $r$ are bounded from below by

$$\begin{aligned} f(r^\star) &= \frac{1}{2} \mu^\top \Gamma \mu - \mu^\top (\Gamma \mu + \tilde{r}) \\ &= -\frac{1}{2} \mu^\top \Gamma \mu - \mu^\top \tilde{r} \end{aligned}$$

The Lagrangian is affine in the elements of $Q$ so that the dual function is defined to be

$$\begin{aligned} g(X, \mu) &= \inf_{r, Q} L(r, Q, \mu, X) \\ &= \begin{cases} -\frac{1}{2} \mu^\top \Gamma \mu - \mu^\top \tilde{r} & \text{if } X = \frac{1}{2} \sum_{k=0}^{n} \mu_k \left( A_k + A_k^\top \right) \\ -\infty & \text{otherwise} \end{cases} \end{aligned}$$

If $\Gamma$ has the Cholesky decomposition $\Gamma = G^\top G$, then

$$\frac{1}{2} \mu^\top \Gamma \mu + \mu^\top \tilde{r} = \frac{1}{2} \left( G\mu + G^{-\top} \tilde{r} \right)^\top \left( G\mu + G^{-\top} \tilde{r} \right) - \frac{1}{2} \tilde{r}^\top \Gamma^{-1} \tilde{r}$$

The dual problem is [21, Equations 24 and 47]:

$$\begin{aligned} \textbf{minimise} \quad & \eta \\ \textbf{subject to} \quad & \| G\mu + G^{-\top} \tilde{r} \| \leq \eta \\ & X(\mu) = \operatorname{Toeplitz} \left[ \mu_0, \frac{1}{2} \mu_1, \ldots, \frac{1}{2} \mu_n \right] \succeq 0 \end{aligned} \tag{N.10}$$

The sign of the objective is reversed to get a minimisation problem.

*Dumitrescu et al.* refer to an approach for fast solutions of SDP problems with Toeplitz LMIs [225]. *Dumitrescu et al.* suggest spectral factorisation of $r^\star$ by solving a Ricatti equation. This appears to refer to the method proposed by *Tuqan* and *Vaidyanathan* [85, Theorem 3]. Also see *Sayed* and *Kailath* [6, Section 5].

*Dumitrescu et al.* [21, Theorem 2 and Section IV] show that the primal solution pair $\{r^\star, Q^\star\}$ and the dual solution $\mu^\star$ have the following strong duality properties

$$\begin{aligned} \frac{1}{2}\left(\tilde{r}-r^\star\right)^\top \Gamma^{-1}\left(\tilde{r}-r^\star\right) &= -\frac{1}{2}\mu^{\star\top}\Gamma\mu^\star - \mu^{\star\top}\tilde{r} \\ r^\star &= \Gamma\mu^\star + \tilde{r} \\ X\left(\mu^\star\right)Q^\star &= 0 \\ \mu^{\star\top} r^\star &= 0 \end{aligned} \tag{N.11}$$

*Dumitrescu et al.* note that the FIR parameter vector, $h^\star$, is an eigenvector of $X\left(\mu^\star\right)$, corresponding to the eigenvalue $0$.

### An example

The Octave script *dumitrescu_MA_estimation_test.m* attempts to implement the, so-called, CFD solution of *Dumitrescu et al.* [21, Equation 47] described above. The specifications of the original filter and estimation procedure are:

```
n=20 % Filter order
ne=20 % Filter order for estimation
N=2000 % Number of filtered noise samples
Rz=0.980000 % Radius of filter zeros
```

Figures N.7 and N.8 show the amplitude responses and zeros, respectively, of the original FIR filter and the estimated filter found by the CFD semi-definite programming method. The minimum-phase filter was calculated with the method of *Orchard* and *Willson* shown previously in Section N.1.4. I found that:

- for zero radiuses $Rz = 0.9$, $\tilde{r}$ is positive and $r^\star = \tilde{r}$
- for zero radiuses $Rz = 0.98$:
  - $\tilde{r}$ is not positive
  - the limits on $l$ in Equation N.8 are widened to $l = \pm\left(n + n_\gamma\right)$ so that $\hat{W}_{kl}$ is positive-definite and SOCP estimation is possible
- for zero radiuses $Rz = 0.99$ $r^\star$ has zeros very close to the unit-circle and *minphase* fails

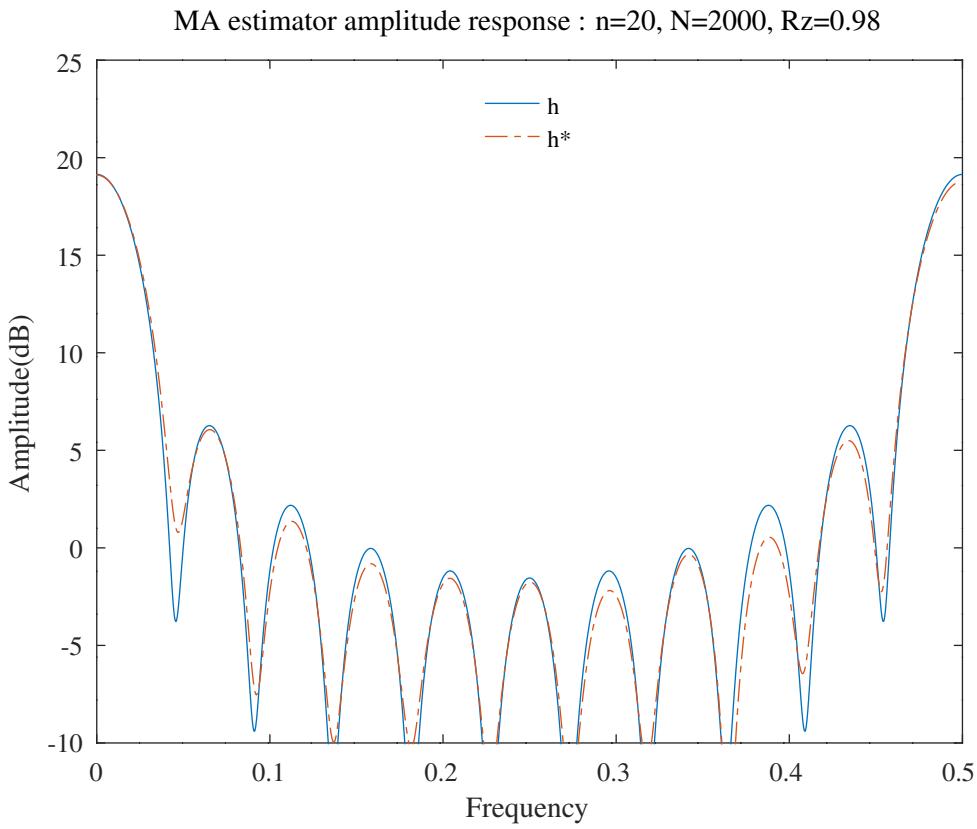



Figure N.7: Amplitude responses of the example FIR filter and the estimated filter found with *Dumitrescu et al.*'s CFD semi-definite programming method.

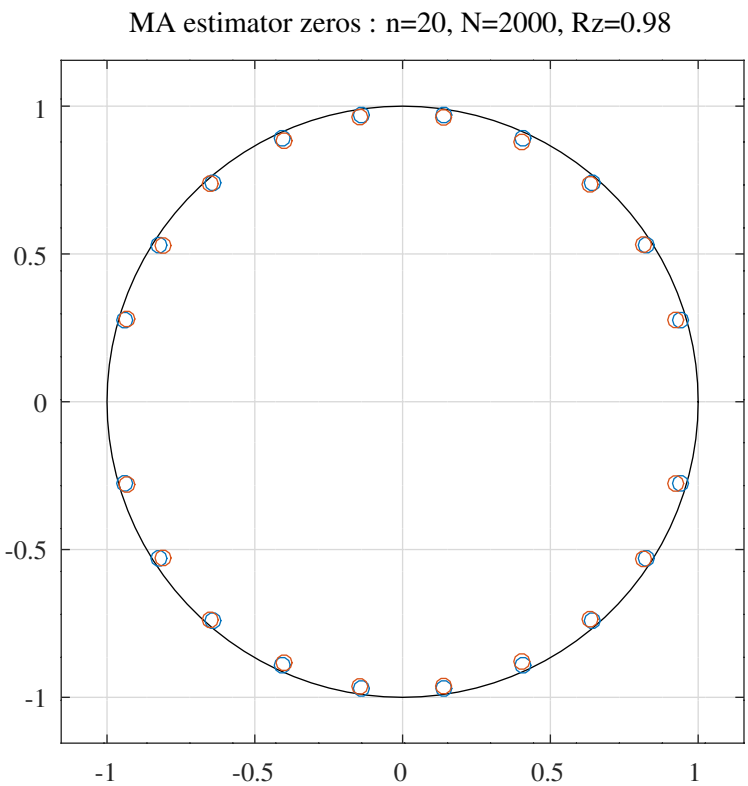



Figure N.8: Zeros of the example FIR filter and the estimated filter found with *Dumitrescu et al.*'s CFD semi-definite programming method.

## N.2 Design of an FIR lattice band-pass Hilbert filter

The Octave script *complementaryFIRlattice_socp_slb_bandpass_test.m* designs a PCLS optimised FIR lattice Hilbert band-pass filter with order 16. The filter specification is:

```
tol=1e-06 % Tolerance on coef. update
ctol=1e-06 % Tolerance on constraints
nplot=1000 % Frequency points across the band
length(k0)=17 % Num. FIR lattice coefficients
sum(k0~=0)=17 % Num. non-zero FIR lattice coef.s
fsl=0.05 % Lower stop band upper edge
fpl=0.1 % Pass band lower edge
fpu=0.2 % Pass band upper edge
fsu=0.25 % Upper stop band lower edge
dBap=3 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
dBas=20 % Amplitude stop band peak-to-peak ripple
Wasl=100 % Ampl. lower stop band weight
Wasu=100 % Ampl. upper stop band weight
tp=5 % Pass band group delay
tpr=1 % Delay pass band peak-to-peak ripple
Wtp=0.1 % Delay pass band weight
pp/pi=1.50 % Pass band phase(adjusted for tp,units of pi)
ppr/pi=0.02 % Phase pass band peak-to-peak ripple(units of pi)
Wpp=1 % Phase pass band weight
```

The initial filter is the band-pass filter designed by the Octave script *iir_sqp_slb_fir_17_bandpass_test.m* in gain-pole-zero form. The corresponding initial minimum-phase FIR lattice filters were calculated by the Octave function *complementaryFIRlattice*.

Figure N.9 shows the zeros of the initial FIR lattice band-pass and complementary filters.

The PCLS optimised filter lattice coefficients are:

```
k2 = [   0.9843749874,   0.9843631757,   0.9843749817,   0.9843749784, ...
         0.9843459672,   0.9843565254,   0.9843634536,   0.9843597582, ...
         0.9843749877,   0.9616092364,   0.9783249979,   0.9843523723, ...
         0.8651000807,   0.9391120075,   0.9843712627,   0.9843749891, ...
         0.1431375883 ]';
```

and:

```
khat2 = [  -0.0468090736,  -0.0110913025,   0.0396599420,   0.0865990217, ...
            0.0779802487,   0.0419015507,   0.0369863740,   0.0160169197, ...
           -0.1702565164,  -0.4765225365,  -0.4461363669,   0.1095851846, ...
            0.6318802083,   0.5215848311,   0.0125879245,  -0.2158720333, ...
            0.9843107113 ]';
```

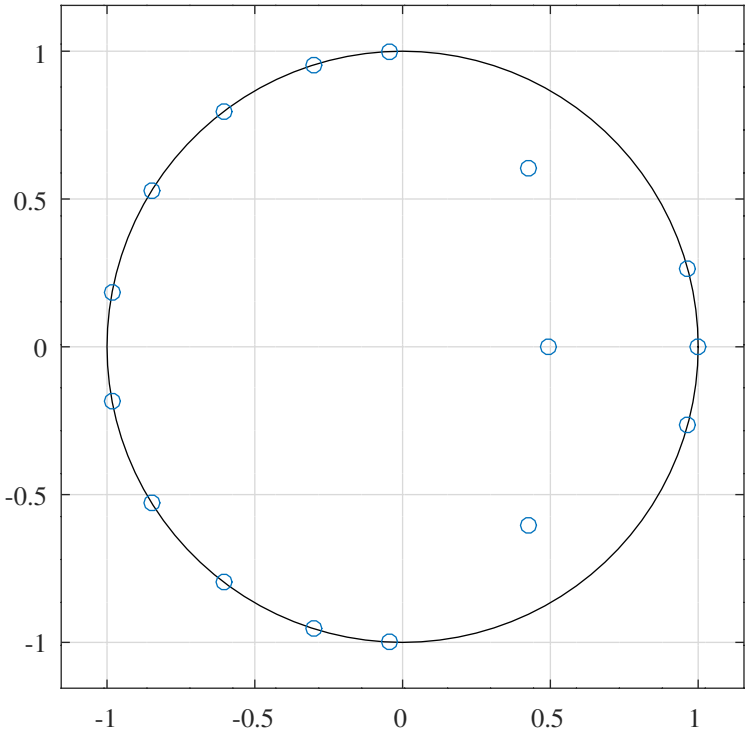
The corresponding direct-form FIR band-pass Hilbert filter coefficients are:

```
Nh2 = [   0.0905516697,   0.0672322315,  -0.0672208463,  -0.2162026902, ...
         -0.2047023001,  -0.0057471936,   0.1918350844,   0.2011736026, ...
          0.0531978852,  -0.0805104681,  -0.0901114111,  -0.0324363609, ...
          0.0009615963,  -0.0060123457,  -0.0150377974,  -0.0165810987, ...
          0.0296104123 ];
```

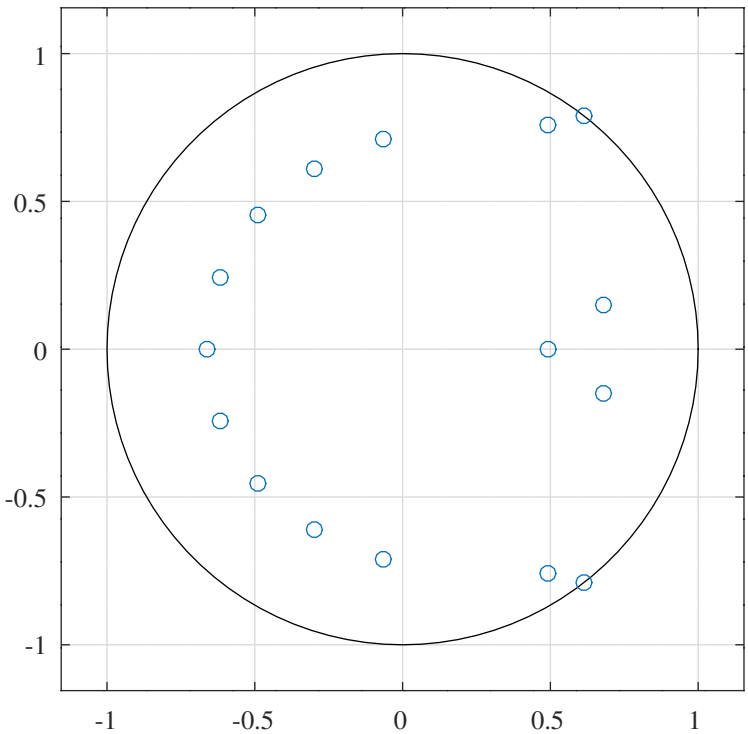
Figure N.10 shows the amplitude, phase and delay responses of the PCLS optimised FIR lattice band-pass Hilbert filter. The phase response shown is adjusted for the nominal delay. The FIR lattice band-pass Hilbert filter was simulated with a uniformly distributed random noise input. The standard-deviations of the 16 states are:

```
std(xxk2) = [   0.0468090736,   0.0474151814,   0.0605190127,   0.1024072148, ...
                0.1248541553,   0.1287002124,   0.1307390023,   0.1292894867, ...
                0.1974856527,   0.4528922757,   0.5518254881,   0.5742691443, ...
                0.7041656705,   0.6000123686,   0.5877751105,   0.5358053818 ];
```

Figure N.11 shows the zeros of the PCLS optimised FIR lattice band-pass Hilbert filter and the complementary filter. The FIR lattice filters need not be minimum-phase.




(a) Zeros of the initial FIR lattice band-pass filter




(b) Zeros of the initial FIR lattice band-pass complementary filter

Figure N.9: Zeros of the initial band-pass FIR lattice filter and the complementary filter

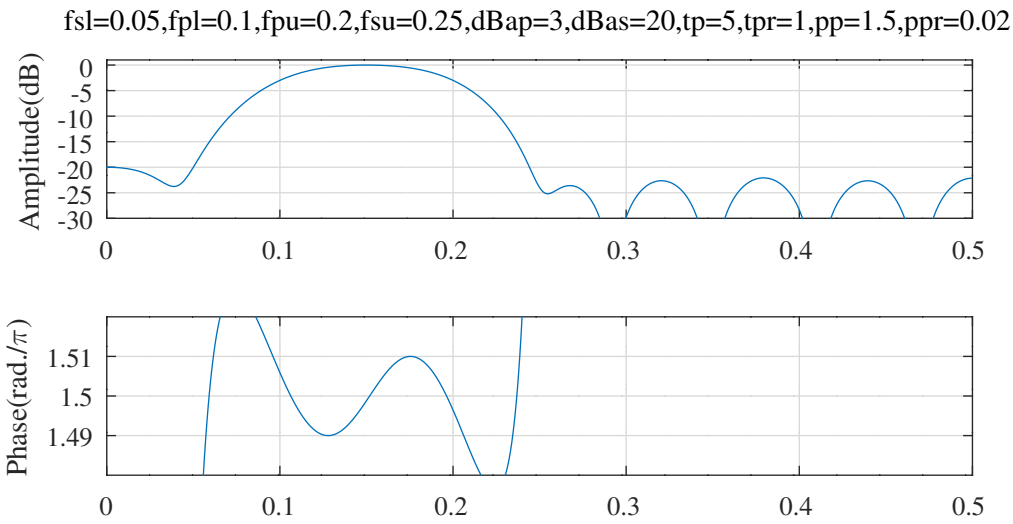
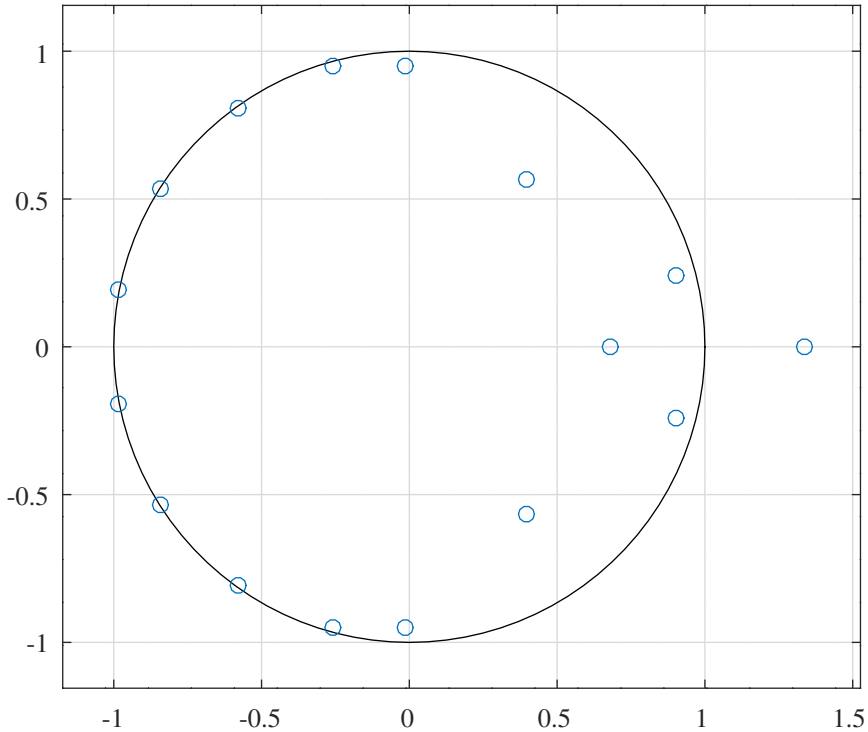
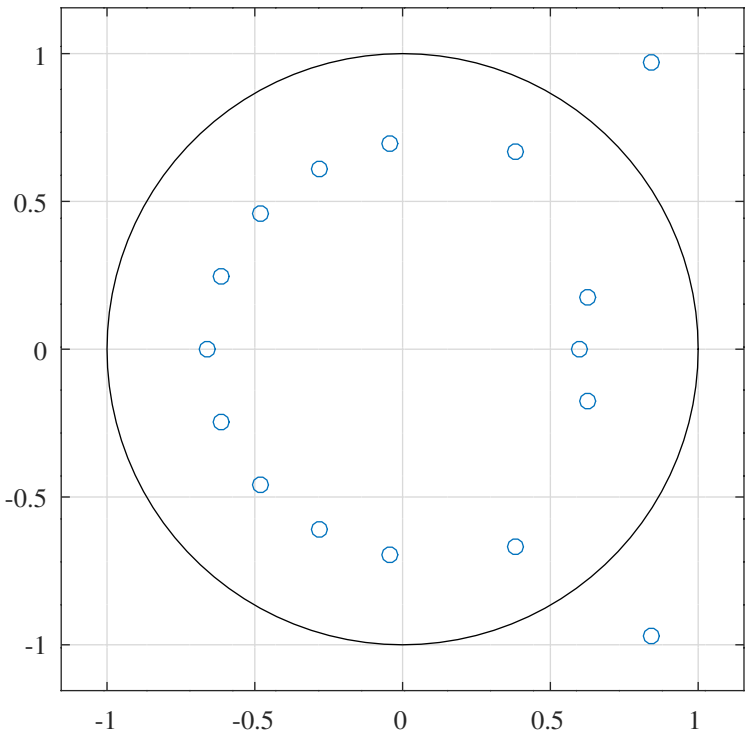



Figure N.10: Response of the PCLS optimised FIR lattice band-pass Hilbert filter. The phase response shown is adjusted for the nominal delay.




(a) Zeros of the PCLS optimised FIR lattice band-pass Hilbert filter




(b) Zeros of the PCLS optimised FIR lattice complementary filter

Figure N.11: Zeros of the PCLS optimised FIR lattice band-pass Hilbert filter and the complementary filter

## N.3 Constrained least-squared error optimisation of FIR digital filters

This section describes the design of constrained, symmetric, even-order, FIR filters by the method of Lagrange multipliers. It is based on the article by *Selesnick*, *Lang* and *Burrus* [75].

FIR filters are a subset of IIR filters. FIR filters are described by a transfer function polynomial in $z^{-1}$:

$$H(z) = \sum_{n=0}^{N} h_n z^{-n}$$

This is called the *direct-form* implementation of the FIR filter.

The transfer function of an even-order, odd-length, $N = 2M$, symmetric FIR filter is:

$$\begin{aligned} H(z) &= \sum_{n=0}^{2M} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n z^{-n} + h_M z^{-M} + \sum_{n=M+1}^{2M} h_n z^{-n} \\ &= z^{-M} \left\{ h_M + \sum_{n=0}^{M-1} h_n \left[ z^{M-n} + z^{-M+n} \right] \right\} \end{aligned}$$

where $h_n = h_{N-n}$. The zero-phase amplitude response is:

$$A(\omega) = h_M + \sum_{n=0}^{M-1} 2h_n \cos\left[(M-n)\omega\right]$$

The transfer function of an odd-order, even-length, $N = 2M - 1$, symmetric FIR filter is:

$$\begin{aligned} H(z) &= \sum_{n=0}^{2M-1} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n z^{-n} + \sum_{n=M}^{2M-1} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n \left[ z^{-n} + z^{n-(2M-1)} \right] \\ &= z^{-M+\frac{1}{2}} \sum_{n=0}^{M-1} h_n \left[ z^{-n+M-\frac{1}{2}} + z^{n-M+\frac{1}{2}} \right] \end{aligned}$$

where $h_n = h_{N-1-n}$ and $N = 2M - 1$. The zero-phase amplitude response is:

$$A(\omega) = 2 \sum_{n=0}^{M-1} h_n \cos\left[ \left( M - \frac{1}{2} - n \right) \omega \right]$$

In the following, I will almost always consider odd-length, even-order, symmetric FIR filters.

The *symmetric-direct-form* implementation requires approximately half the number of multipliers used by the direct-form implementation. A symmetric FIR filter has a linear phase response or, equivalently, a constant group delay response. The direct-form and symmetric-direct-form FIR filter implementations combine all multiplications and additions into a single calculation. These arithmetic operations could be pipelined in a tree structure at the expense of increased delay through the filter. The *transposed-direct-form* FIR filter is an alternative FIR filter implementation that uses *Horner's Rule* to evaluate the filter polynomial:

$$h(z) = \left( \ldots \left( \left( h_N z^{-1} + h_{N-1} \right) z^{-1} + h_{N-2} \right) z^{-1} + \ldots + h_1 \right) z^{-1} + h_0$$

The transposed-direct-form pipelines the arithmetic operations but cannot make use of a symmetry in the filter polynomial.

Given a desired amplitude response, $A_d(\omega)$, and a weight function, $W(\omega)$, the mean-squared-error of the amplitude response of the FIR filter is:

$$\mathcal{E} = \frac{1}{\pi} \int_0^{\pi} W(\omega) \left[ A(\omega) - A_d(\omega) \right]^2 d\omega$$

The design problem is to minimise the mean-squared-error subject to constraints on the amplitude response:

$$\begin{aligned} A(\omega_p) &\geq L(\omega_p) \\ A(\omega_q) &\leq U(\omega_q) \end{aligned}$$

The method of *Lagrange multipliers* defines a *Lagrangian* function:

$$\mathcal{L}(\omega) = \mathcal{E} - \sum_p \lambda_p \left[A(\omega_p) - L(\omega_p)\right] - \sum_q \lambda_q \left[U(\omega_q) - A(\omega_q)\right]$$

where $\lambda_p$ and $\lambda_q$ are the *Lagrange multipliers* for the amplitude constraints at the corresponding frequencies $\omega_p$ and $\omega_q$. The method of Lagrange multipliers minimises $\mathcal{E}$ subject to the constraints by solving the following system of equations for $h_n$, $\lambda_p$ and $\lambda_q$:

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial h_n} - \sum_p \lambda_p \frac{\partial A(\omega_p)}{\partial h_n} + \sum_q \lambda_q \frac{\partial A(\omega_q)}{\partial h_n} &= 0 \\ A(\omega_p) &= L(\omega_p) \\ A(\omega_q) &= U(\omega_q) \end{aligned}$$

where:

$$\frac{\partial \mathcal{E}}{\partial h_n} = \frac{2}{\pi} \int_0^\pi W(\omega) \left[A(\omega) - A_d(\omega)\right] \frac{\partial A(\omega)}{\partial h_n} d\omega$$

and, for an even-order, symmetric, FIR filter:

$$\frac{\partial A(\omega)}{\partial h_n} = \begin{cases} 2\cos(M-n)\omega & n = 0, \ldots, M-1 \\ 1 & n = M \end{cases}$$

The minimisation problem can be written in matrix form as:

$$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{G}^\top \\ \boldsymbol{G} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{h} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{D} \end{bmatrix} \tag{N.12}$$

where $\boldsymbol{R}$ is a symmetric matrix. The solution is:

$$\begin{aligned} \boldsymbol{\lambda} &= \left(\boldsymbol{G}\boldsymbol{R}^{-1}\boldsymbol{G}^\top\right)^{-1} \left(\boldsymbol{G}\boldsymbol{R}^{-1}\boldsymbol{C} - \boldsymbol{D}\right) \\ \boldsymbol{h} &= \boldsymbol{R}^{-1}\left(\boldsymbol{C} - \boldsymbol{G}^\top\boldsymbol{\lambda}\right) \end{aligned}$$

When the Lagrange multipliers are non-negative, $\lambda_p, \lambda_q \geq 0$, then the *Karush-Kuhn-Tucker* conditions guarantee that the equality-constrained problem also solves the inequality-constrained problem. The exchange algorithm of *Selesnick et al.* solves successive minimisation problems with constraints on the peaks of the response. If any Lagrange multipliers are negative at some iteration then the corresponding constraint frequencies are sequentially dropped so that an inequality-constrained problem is solved.

### N.3.1 Examples of the design of constrained least-squared error symmetric FIR filters

This section shows an example of the design of an even-order, symmetric, FIR filter by the exchange algorithm described by *Selesnick et al.* [76, p.498] and summarised in Section 8.1.2. The impulse response is $\boldsymbol{h} = [h_0, \ldots, h_M]^\top$ with piece-wise constant desired amplitude, $A_d(k)$, and weight, $W(k)$, over contiguous regions $\Omega_k \subset [0, \pi]$.

In each region, $\Omega_k$, the mean-squared error is:

$$\mathcal{E} = \frac{1}{\pi} \int_{\Omega_k} W(k) \left[A(\omega) - A_d(k)\right]^2 d\omega$$

where:

$$\int A(\omega)\, d\omega = h_M \omega + 2 \sum_{n=0}^{M-1} h_n \frac{\sin(M-n)\omega}{M-n}$$

and:

$$\int A(\omega)^2 d\omega = h_M^2\omega - 4h_M \sum_{n=0}^{M-1} h_n \frac{\sin(M-n)\omega}{M-n} \quad \ldots$$
$$+ 2\sum_{n=0}^{M-1}\sum_{m=0}^{M-1} h_n h_m \frac{\sin(2M-m-n)\omega}{2M-m-n} + 2\sum_{n=0}^{M-1} h_n^2\omega + 2\sum_{n=0}^{M-1}\sum_{m=0,m\neq n}^{M-1} h_n h_m \frac{\sin(n-m)\omega}{n-m}$$

Similarly, the gradient of the mean-squared error with respect to the coefficients is:

$$\frac{\partial \mathcal{E}}{\partial h_n} = \frac{2}{\pi}\int_{\Omega_k} W(k)\left[A(\omega) - A_d(k)\right]\frac{\partial A(\omega)}{\partial h_n} d\omega$$

which can be expressed in the form $2\boldsymbol{q} + 2\boldsymbol{Q}\boldsymbol{h}$. The components of $\boldsymbol{q}$ and $\boldsymbol{Q}$ are found with:

$$\int \frac{\partial A(\omega)}{\partial h_n} d\omega = \begin{cases} 2\dfrac{\sin(M-n)\omega}{M-n} & n = 0,\ldots,M-1 \\ \omega & n = M \end{cases}$$

and:

$$\int A(\omega)\frac{\partial A(\omega)}{\partial h_n} d\omega = \begin{cases} 2\displaystyle\sum_{m=0}^{M} h_m \frac{\sin(2M-m-n)\omega}{2M-m-n} + 2h_n\omega + 2\sum_{m=0,m\neq n}^{M-1} h_m \frac{\sin(n-m)\omega}{n-m} & n = 0,\ldots,M-1 \\ h_M\omega + 2\displaystyle\sum_{m=0}^{M-1} h_m \frac{\sin(M-m)\omega}{M-m} & n = M \end{cases}$$

since, for $n = 0,\ldots,M-1$:

$$\int A(\omega)\cos(M-n)\omega\, d\omega = \sum_{m=0}^{M} h_m \int \cos(2M-m-n)\omega\, d\omega + \sum_{m=0}^{M-1} h_m \int \cos(n-m)\omega\, d\omega$$

The octave function *directFIRsymmetricEsqPW* implements the *piece-wise* calculation of $\mathcal{E}$ and its gradients with respect to $h_n$.

### Design of a constrained least-squared error symmetric FIR low-pass filter

The Octave script *directFIRsymmetric_slb_lowpass_test.m* implements the design of an even-order symmetric FIR low-pass filter with constraints on the amplitude response by the method of Lagrange multipliers. The matrix equation shown above in Equation N.12 is solved by left division in the Octave function *directFIRsymmetric_mmsePW* and the exchange algorithm of *Selesnick et al.* is implemented by the Octave function *directFIRsymmetric_slb*. The low-pass filter specification is:

```
tol=1e-05 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
nplot=1000 % Frequency points across the band
M=15 % M+1 distinct coefficients
fap=0.1 % Amplitude pass band edge
dBap=0.02 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
fas=0.2 % Amplitude stop band edge
dBas=52 % Amplitude stop band peak-to-peak ripple
Was=10 % Amplitude stop band weight
```

The bandwidth of the constrained low-pass filter is slightly narrower than that of the initial unconstrained filter. The amplitude responses of the initial and optimised low-pass filters are shown in Figure N.12. The distinct optimised filter coefficients are:

```
hM1 = [   0.0015415562,   0.0021622752,  -0.0004724047,  -0.0057312301, ...
         -0.0081256352,  -0.0016428074,   0.0123722369,   0.0218047186, ...
          0.0115474186,  -0.0202014306,  -0.0510078994,  -0.0433674249, ...
          0.0270095515,   0.1450930022,   0.2573031209,   0.3034299040 ]';
```

Figure N.13 shows the initial response and the response obtained when one coefficient of the exact solution is rounded and the remaining coefficients are optimised with the pass-band ripple constraint relaxed.

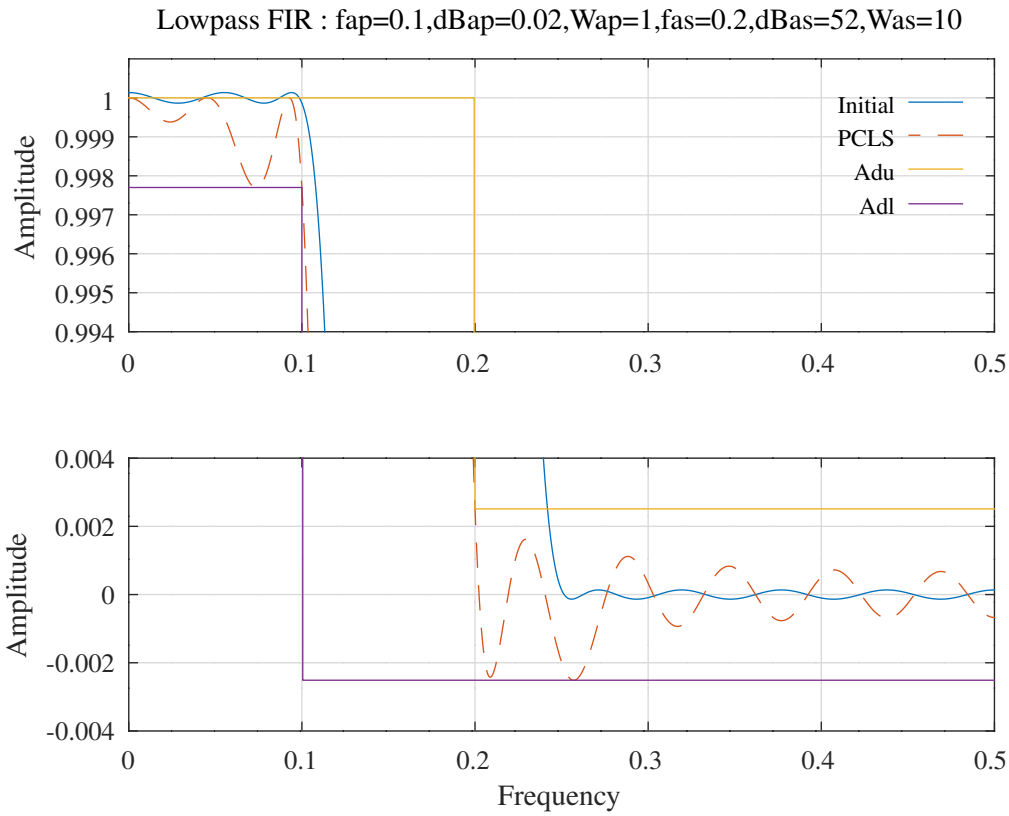



Figure N.12: Amplitude response of the initial FIR low-pass filter, upper and lower response constraint mask and the optimised low-pass FIR filter designed with the method of Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

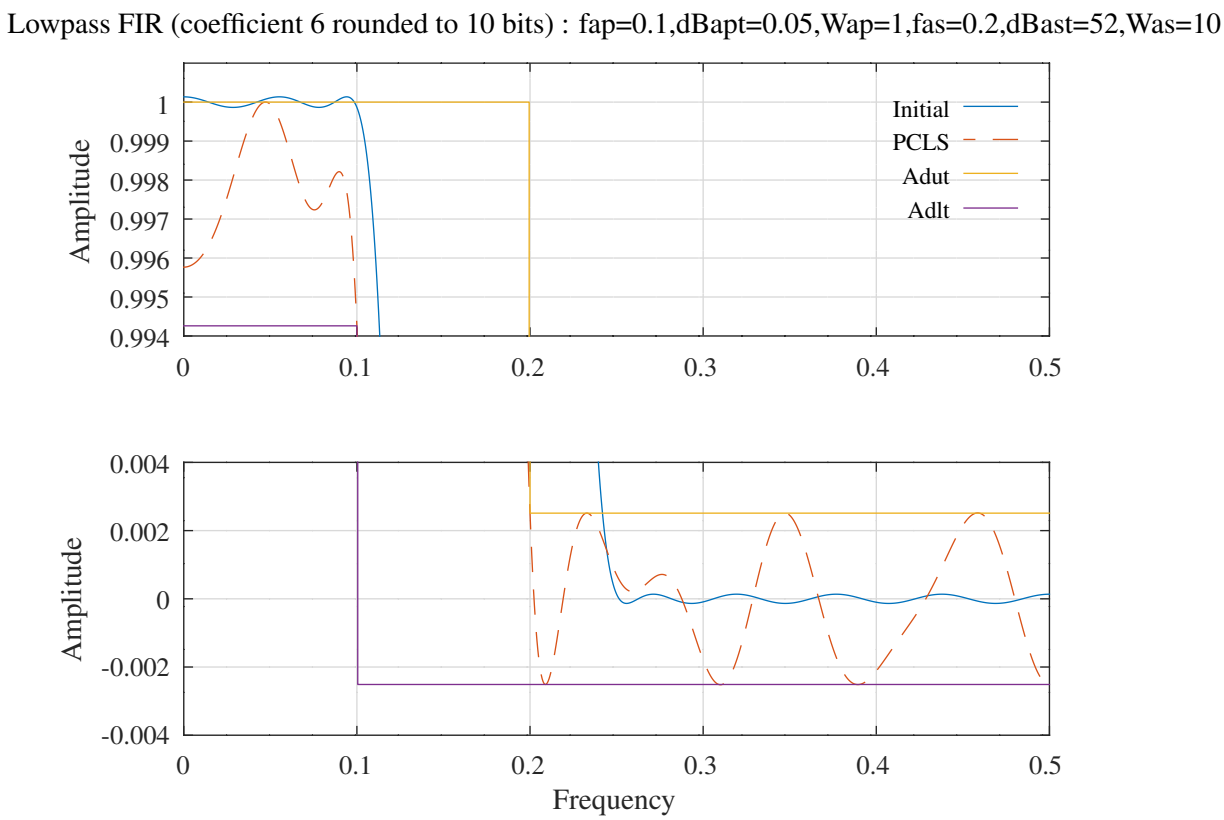



Figure N.13: Amplitude response of the initial FIR low-pass filter, upper and lower response constraint mask and the optimised band pass FIR filter designed with one coefficient rounded and the remaining coefficients optimised by the method of Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

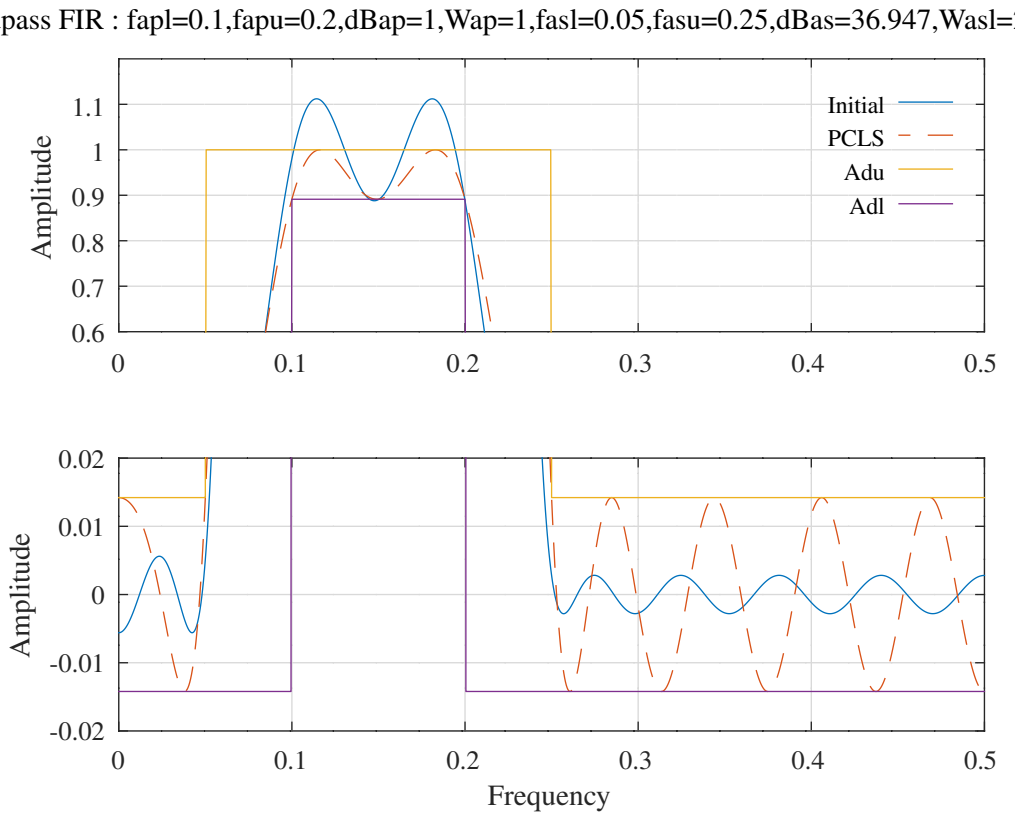



Figure N.14: Amplitude response of the initial FIR band-pass filter, upper and lower response constraint mask and the optimised band-pass FIR filter designed with the method of Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

**Design of a constrained least-squared error symmetric FIR band-pass filter**

Similarly to the previous example, the Octave script *directFIRsymmetric_slb_bandpass_test.m* implements the design of an even-order symmetric FIR band-pass filter with constraints on the amplitude response by the method of Lagrange multipliers. The band-pass filter specification is:

```
tol=1e-05 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
nplot=1000 % Frequency points across the band
M=16 % M+1 distinct coefficients
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
Wap=1 % Amplitude pass band weight
dBap=1 % Amplitude pass band peak-to-peak ripple
fasl=0.05 % Amplitude lower stop band edge
Wasl=20 % Amplitude lower stop band weight
fasu=0.25 % Amplitude upper stop band edge
Wasu=40 % Amplitude upper stop band weight
dBas=36.947 % Amplitude stop band peak-to-peak ripple
```

The amplitude responses of the initial and optimised band-pass filters are shown in Figure N.14. The distinct optimised filter coefficients are:

```
hM1 = [  -0.0058181010,   0.0017787857,  -0.0047084625,  -0.0143846688, ...
         -0.0077550125,   0.0219788564,   0.0432578789,   0.0247317110, ...
         -0.0077853817,  -0.0010276677,   0.0304650309,   0.0009925325, ...
         -0.1110651112,  -0.1806101683,  -0.0725659905,   0.1536437055, ...
          0.2719559562 ]';
```

Figure N.15 shows the initial response and the response obtained when three coefficients of the exact solution are rounded and the remaining coefficients are optimised with the stop-band ripple constraint relaxed.

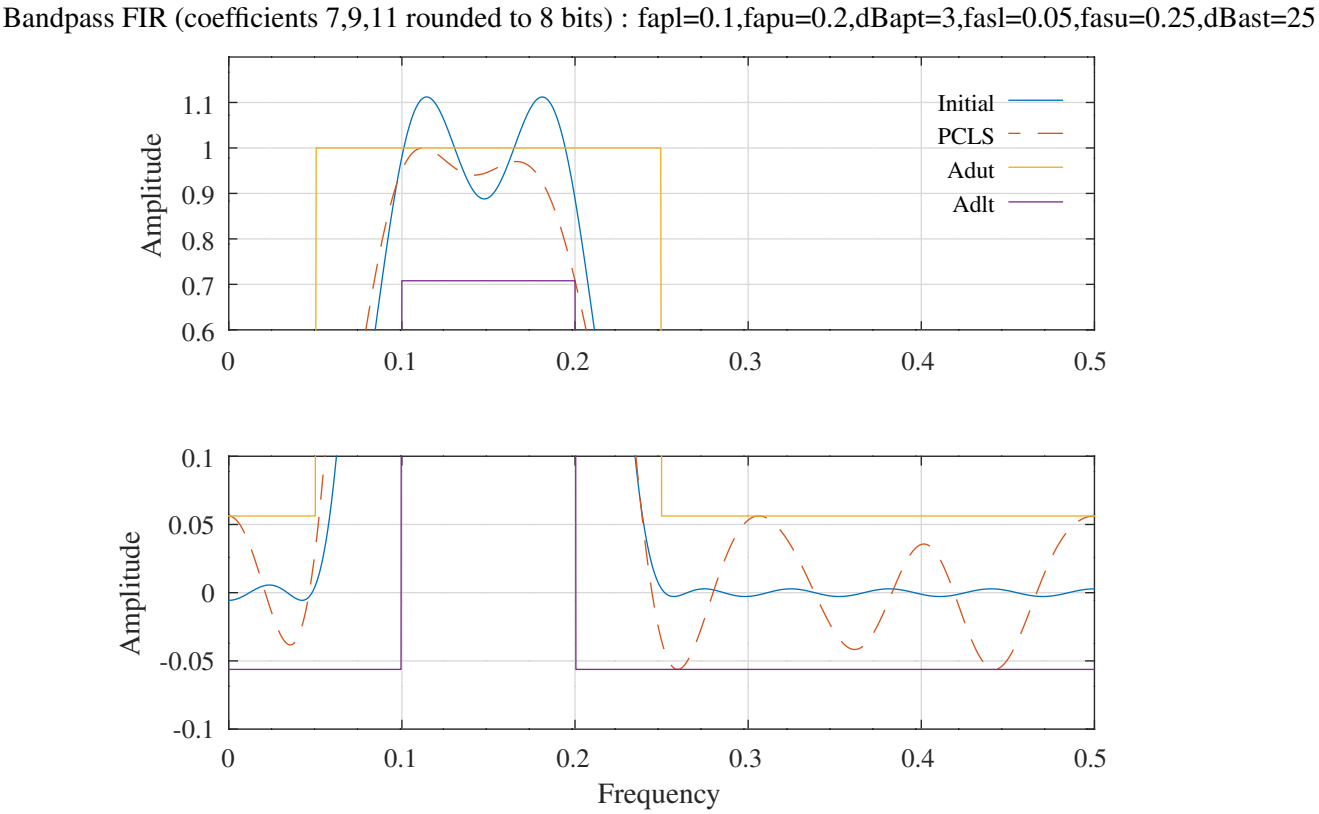



Figure N.15: Amplitude response of the initial FIR band-pass filter, upper and lower response constraint mask and the optimised band-pass FIR filter designed with three coefficients rounded and the remaining coefficients optimised by the method of Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

## N.4 Constrained least-mean-squared error optimisation of FIR Hilbert digital filters

### N.4.1 Frequency response of a Hilbert filter

The frequency response of an ideal Hilbert transform filter is [143, pp.790-792]:

$$H(\omega) = \begin{cases} -\imath & 0 < \omega < \pi \\ \imath & -\pi < \omega < 0 \end{cases}$$

The corresponding impulse response is:

$$\begin{aligned} \hat{h}_k &= \frac{1}{2\pi}\int_{-\pi}^{0} \imath e^{\imath\omega k} d\omega - \frac{1}{2\pi}\int_{0}^{\pi} \imath e^{\imath\omega k} d\omega \\ &= \begin{cases} \frac{2}{\pi k}\sin^2\frac{\pi k}{2} & k \neq 0 \\ 0 & k = 0 \end{cases} \end{aligned}$$

Clearly $\hat{h}_k = -\hat{h}_{-k}$ and $\hat{h}_{2k} = 0$.

If the FIR Hilbert filter is truncated to even order $4M-2$ and odd length $4M-1$, then there are $M$ distinct coefficients, the group-delay of the filter is $2M-1$ samples and the z-transform of the filter is:

$$\begin{aligned} H(z) &= z^{-2M+1}\sum_{k=-2M+1}^{2M-1} \hat{h}_k z^{-k} \\ &= z^{-2M+1}\sum_{k=1}^{2M-1} \hat{h}_k \left[z^{-k} - z^{k}\right] \\ &= z^{-2M+1}\sum_{k=1}^{M} \hat{h}_{2k-1} \left[z^{-2k+1} - z^{2k-1}\right] \\ &= z^{-2M+1}\sum_{k=0}^{M-1} \hat{h}_{2k+1} \left[z^{2k+1} - z^{-2k-1}\right] \\ &= z^{-2M+1}\sum_{k=0}^{M-1} h_{2M-2k-2} \left[z^{2k+1} - z^{-2k-1}\right] \end{aligned}$$

where $h_k = \hat{h}_{-2M+1+k}$, $h_{2M-1+k} = \hat{h}_k$ and $\hat{h}_{2k+1} = -\hat{h}_{-2k-1} = -h_{2M-2k-2}$. The zero-phase amplitude response of the even order FIR Hilbert filter is:

$$\begin{aligned} A(\omega) &= 2\sum_{k=0}^{M-1} h_{2M-2k-2}\sin(2k+1)\omega \\ &= 2\sum_{k=0}^{M-1} h_{2k}\sin(2M-2k-1)\omega \end{aligned} \tag{N.13}$$

Similarly, the distinct coefficients of an odd order $2M-1$, even length $2M$, FIR Hilbert filter are $\hat{h}_k$ with $k = \frac{1}{2}, \frac{3}{2}, \ldots, M-\frac{1}{2}$:

$$\begin{aligned} H(z) &= z^{-M+\frac{1}{2}}\sum_{k=-M+\frac{1}{2}}^{M-\frac{1}{2}} \hat{h}_k z^{-k} \\ &= z^{-M+\frac{1}{2}}\sum_{k=\frac{1}{2}}^{M-\frac{1}{2}} \hat{h}_k \left[z^{-k} - z^{k}\right] \\ &= z^{-M+\frac{1}{2}}\sum_{k=0}^{M-1} \hat{h}_{k+\frac{1}{2}} \left[z^{-k-\frac{1}{2}} - z^{k+\frac{1}{2}}\right] \end{aligned}$$

$$= z^{-M+\frac{1}{2}} \sum_{k=0}^{M-1} h_{M-1-k} \left[ z^{k+\frac{1}{2}} - z^{-k-\frac{1}{2}} \right]$$

where $h_k = \hat{h}_{-M+\frac{1}{2}+k}$ and $h_{M-1-k} = -\hat{h}_{k+\frac{1}{2}}$. The zero-phase amplitude response is:

$$\begin{aligned} A(\omega) &= 2 \sum_{k=0}^{M-1} h_{M-1-k} \sin(2k+1) \frac{\omega}{2} \\ &= 2 \sum_{k=0}^{M-1} h_k \sin(2M-2k-1) \frac{\omega}{2} \end{aligned} \tag{N.14}$$

Note that, in practice, the odd order $h_k$ is interpolated with zeros and Equation N.14 and Equation N.13 are effectively identical.

### N.4.2 Design of a least-mean-squared error even-order FIR Hilbert filter

The design problem is to minimise the mean-squared-error subject to constraints on the zero-phase amplitude response of the Hilbert filter:

$$\begin{aligned} \mathcal{E} &= \frac{1}{\pi} \int_0^{\pi} W(\omega) \left[ A(\omega) + 1 \right]^2 d\omega \\ A(\omega_p) &\geq L(\omega_p) \\ A(\omega_q) &\leq U(\omega_q) \end{aligned}$$

where $Ad(\omega) = -1$.

In a similar fashion to that shown in Appendix N.3, the method of *Lagrange multipliers* optimises the mean-squared-error by solving the following system of equations for $\lambda_p$, $\lambda_q$ and the coefficients of an even order $4M-2$ Hilbert filter, $h_{2k}$, with $k = 0, 1, \ldots, M-1$:

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial h_{2k}} - \sum_p \lambda_p \frac{\partial A(\omega_p)}{\partial h_{2k}} + \sum_q \lambda_q \frac{\partial A(\omega_q)}{\partial h_{2k}} &= 0 \\ A(\omega_p) &= L(\omega_p) \\ A(\omega_q) &= U(\omega_q) \end{aligned}$$

where:

$$\frac{\partial \mathcal{E}}{\partial h_{2k}} = \frac{2}{\pi} \int_0^{\pi} W(\omega) \left[ A(\omega) + 1 \right] \frac{\partial A(\omega)}{\partial h_{2k}} d\omega$$

and:

$$\frac{\partial A(\omega)}{\partial h_{2k}} = 2 \sin(2M-2k-1)\omega$$

This system of equations can be expressed in matrix form with components given by: [e]

$$\begin{aligned} \int \frac{\partial A(\omega)}{\partial h_{2k}} d\omega &= -2 \frac{\cos(2M-2k-1)\omega}{(2M-2k-1)} \\ \int A(\omega) \frac{\partial A(\omega)}{\partial h_{2k}} d\omega &= 2h_{2k}\omega + 2 \sum_{l=0, l \neq k}^{M-1} h_{2l} \frac{\sin(2k-2l)\omega}{2k-2l} - 2 \sum_{l=0}^{M-1} h_{2l} \frac{\sin(4M-2k-2l-2)\omega}{4M-2k-2l-2} \end{aligned}$$

The Octave script *directFIRhilbert_slb_test.m* implements the design of an FIR Hilbert filter with constraints on the amplitude response by the method of Lagrange multipliers. The matrix equation shown above in Equation N.12 is solved by left division in the Octave function *directFIRhilbert_mmsePW* and the PCLS exchange algorithm of *Selesnick et al.* is implemented by the Octave function *directFIRhilbert_slb*. The FIR Hilbert filter specification is:

[e] $2 \sin x \sin y = \cos(x-y) - \cos(x+y)$

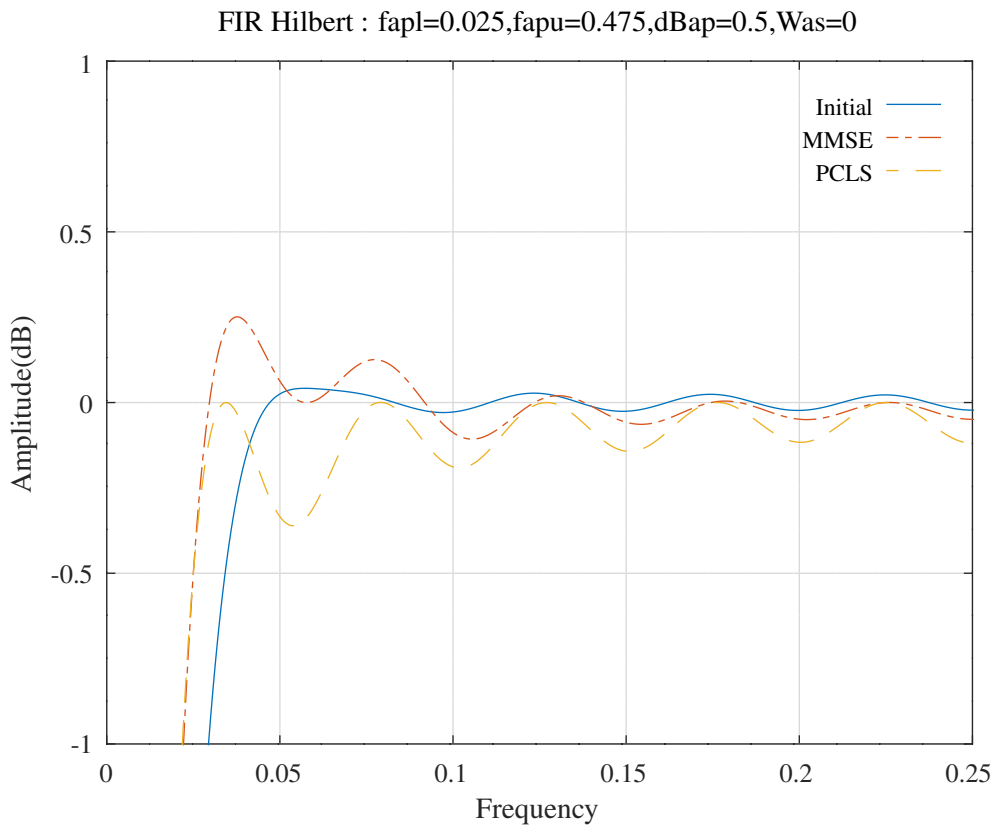



Figure N.16: Amplitude response of the initial FIR Hilbert filter and the optimised FIR Hilbert filter designed with the method of Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

```
tol=0.0001 % Tolerance on coefficient update vector
ctol=0.0001 % Tolerance on constraints
npoints=1000 % Frequency points across the band
M=10 % M distinct coefficients
fapl=0.025 % Amplitude pass band lower edge
fapu=0.475 % Amplitude pass band upper edge
Wap=1 % Amplitude pass band weight
dBap=0.5 % Amplitude pass band peak-to-peak ripple
Was=0 % Amplitude stop band weight
```

The group delay of the filter is $2M - 1$ samples. The bandwidth of the constrained Hilbert filter is slightly wider than that of the initial unconstrained filter. The absolute value of the PCLS pass-band response is constrained to be less than 1. The amplitude responses of the initial and optimised Hilbert filters are shown in Figure N.16. The FIR Hilbert filter pass-band response is symmetrical about the frequency 0.25, where the sampling rate is normalised to 1. The distinct optimised filter coefficients are:

```
hM2 = [  -0.0086302951,  -0.0133649396,  -0.0194607169,  -0.0273205074, ...
         -0.0379833788,  -0.0525480847,  -0.0750429861,  -0.1138083316, ...
         -0.2013619122,  -0.6286182035 ]';
```

The corresponding FIR filter is:

```
h=kron([hM2(:);-flipud(hM2(:))],[1;0])(1:(end-1));
```

### N.4.3 Design of a least-mean-squared error even-order FIR Hilbert band-pass filter

The Octave script *directFIRhilbert_bandpass_slb_test.m* designs an FIR Hilbert filter with a band-pass response. The corresponding filter specification is:

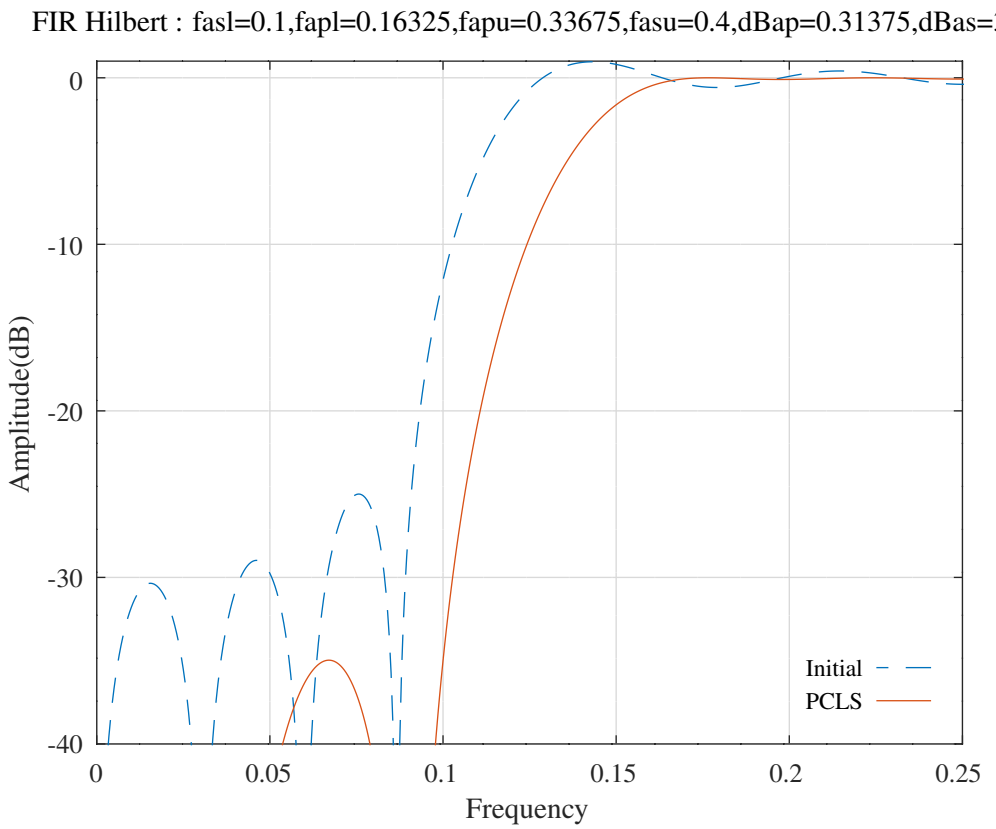



Figure N.17: Amplitude response of the initial FIR Hilbert filter and the optimised FIR Hilbert bandpass filter designed with the method of Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

```
tol=0.0001 % Tolerance on coefficient update vector
ctol=0.0001 % Tolerance on constraints
npoints=1000 % Frequency points across the band
M=8 % M distinct coefficients
fasl=0.1 % Amplitude stop band lower edge
fapl=0.16325 % Amplitude pass band lower edge
fapu=0.33675 % Amplitude pass band upper edge
fasu=0.4 % Amplitude stop band upper edge
Wap=1 % Amplitude pass band weight
dBap=0.31375 % Amplitude pass band peak-to-peak ripple
Wat=0.001 % Amplitude transition band weight
Was=10 % Amplitude stop band weight
dBas=35 % Amplitude stop band peak ripple
```

The amplitude responses of the initial and optimised Hilbert filters are shown in Figure N.17. The pass-band responses are shown in Figure N.18. The distinct optimised filter coefficients are:

```
hM2 = [  -0.0104589390,   0.0031738998,   0.0291468051,  -0.0144604946, ...
         -0.0629162600,   0.0550052923,   0.1596092306,  -0.4239235327 ]';
```

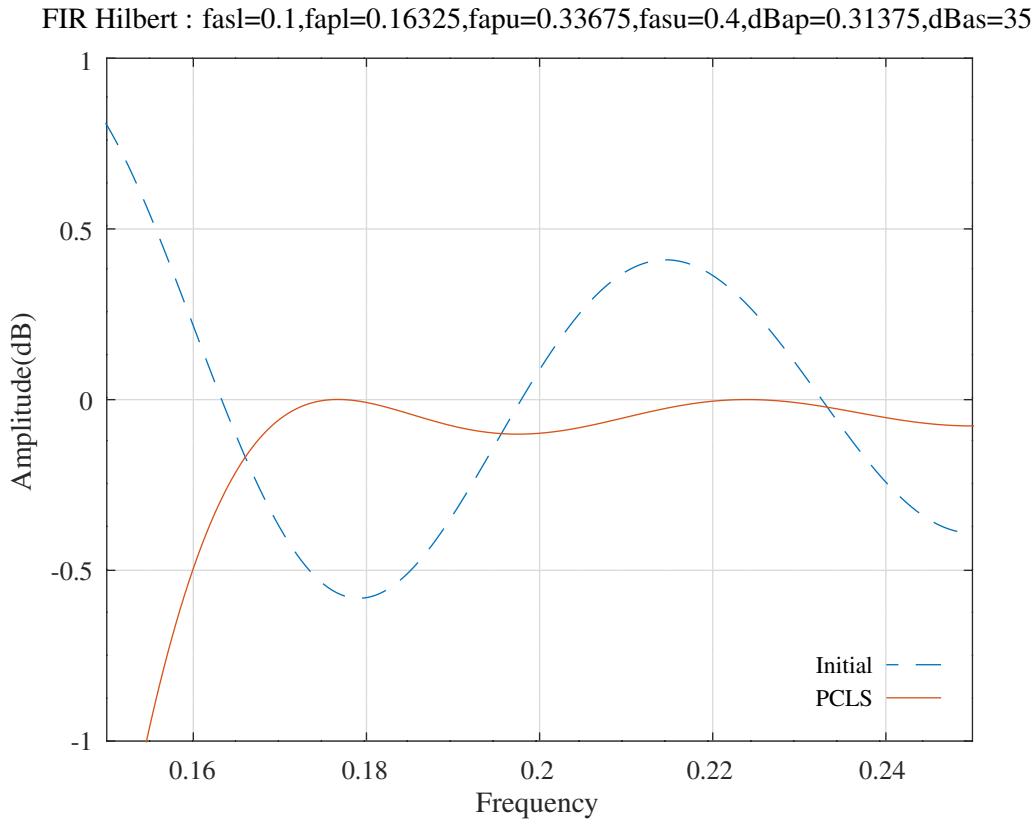



Figure N.18: Pass-band amplitude response of the initial FIR Hilbert bandpass filter and the optimised FIR Hilbert bandpass filter designed with the method of Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

## N.5 Peak-constrained-least-squares design of non-symmetric FIR filters

This section describes peak-constrained-least-squares (PCLS) design of non-symmetric FIR filters with SOCP optimisation and the exchange algorithm of *Selesnick*, *Lang* and *Burrus* [75].

### N.5.1 Frequency response and gradients of a non-symmetric FIR filter

The complex frequency response of an order $N$ non-symmetric FIR filter with coefficients $h_k$ is:

$$H(\omega) = \sum_{k=0}^{N} h_k e^{-\imath k\omega}$$

#### Squared magnitude response of a non-symmetric FIR filter

The squared magnitude response is differentiable. The squared magnitude response of a non-symmetric FIR filter is:

$$\begin{aligned}
|H(\omega)|^2 &= H(\omega)^\dagger H(\omega) \\
&= \left(\sum_{l=0}^{N} h_l e^{\imath l\omega}\right)\left(\sum_{k=0}^{N} h_k e^{-\imath k\omega}\right) \\
&= \sum_{l=0}^{N}\sum_{k=0}^{N} h_l h_k \left(\cos l\omega \cos k\omega + \sin l\omega \sin k\omega\right) \\
&= \sum_{l=0}^{N}\sum_{k=0}^{N} h_l h_k \cos(l-k)\omega
\end{aligned}$$

The gradient with respect to $h_k$ of the squared magnitude of a non-symmetric FIR filter is:

$$\begin{aligned}
\frac{\partial |H(\omega)|^2}{\partial h_k} &= e^{\imath k\omega}\sum_{l=0}^{N} h_l e^{-\imath l\omega} + e^{-\imath k\omega}\sum_{l=0}^{N} h_l e^{\imath l\omega} \\
&= 2\sum_{l=0}^{N} h_l \cos(l-k)\omega
\end{aligned}$$

The calculation of the squared magnitude response of a non-symmetric FIR filter is implemented in the Octave function *directFIRnonsymmetricAsq.m* exercised by the Octave script *directFIRnonsymmetricAsq_test.m*.

#### Phase response of a non-symmetric FIR filter

The phase response of a non-symmetric FIR filter is:

$$\begin{aligned}
\arg H(\omega) &= \arctan \frac{\Im H(\omega)}{\Re H(\omega)} \\
&= -\arctan \frac{\sum_{k=0}^{N} h_k \sin k\omega}{\sum_{k=0}^{N} h_k \cos k\omega}
\end{aligned}$$

The gradient with respect to $h_k$ of the phase response of a non-symmetric FIR filter is:

$$\begin{aligned}
|H(\omega)|^2 \frac{\partial \arg H(\omega)}{\partial h_k} &= \Re H(w) \frac{\partial \Im H(w)}{\partial h_k} - \Im H(w) \frac{\partial \Re H(w)}{\partial h_k} \\
&= -\sin k\omega \sum_{l=0}^{N} h_l \cos l\omega + \cos k\omega \sum_{l=0}^{N} h_l \sin l\omega \\
&= \sum_{l=0}^{N} h_l \sin\,(l-k)\omega
\end{aligned}$$

The calculation of the phase response of a non-symmetric FIR filter is implemented in the Octave function *directFIRnonsymmetricP.m* exercised by the Octave script *directFIRnonsymmetricP_test.m*.

**Group delay response of a non-symmetric FIR filter**

The group delay response, $T(\omega)$, of a non-symmetric FIR filter is:

$$\begin{aligned}
T(\omega) &= -\frac{\partial \arg H(\omega)}{\partial \omega} \\
|H(\omega)|^2 T(\omega) &= \sum_{l=0}^{N} h_l \cos l\omega \sum_{k=0}^{N} k h_k \cos k\omega + \sum_{l=0}^{N} h_l \sin l\omega \sum_{k=0}^{N} k h_k \sin k\omega \\
&= \sum_{k=0}^{N} \sum_{l=0}^{N} k h_k h_l \cos(l-k)\omega
\end{aligned}$$

The gradient with respect to $h_k$ of the group delay response of a non-symmetric FIR filter is:

$$\begin{aligned}
\frac{\partial |H(\omega)|^2}{\partial h_k} T(\omega) + |H(\omega)|^2 \frac{\partial T(\omega)}{\partial h_k} &= \sum_{k=0}^{N} \sum_{l=0}^{N} h_l \cos[(l-k)\omega] \frac{\partial}{\partial h_k} k h_k + \sum_{k=0}^{N} \sum_{l=0}^{N} l h_l \frac{\partial}{\partial h_k} h_k \cos(k-l)\omega \\
&= \sum_{l=0}^{N} (l+k) h_l \cos(l-k)\omega
\end{aligned}$$

The calculation of the group delay response of a non-symmetric FIR filter is implemented in the Octave function *directFIRnonsymmetricT.m* exercised by the Octave script *directFIRnonsymmetricT_test.m*.

## N.5.2 Examples of the PCLS design of non-symmetric FIR filters

**Design of a non-symmetric FIR low pass filter**

The Octave script *directFIRnonsymmetric_socp_slb_lowpass_test.m* designs a low pass non-symmetric FIR filter with a nominal pass band delay using SOCP optimisation and the PCLS exchange algorithm of *Selesnick*, *Lang* and *Burrus*. The FIR filter specification is:

```
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
n=500 % Frequency points across the band
N=30 % FIR filter order
fap=0.15 % Amplitude pass band edge
dBap=1 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=1e-08 % Amplitude transition band weight
fas=0.2 % Amplitude stop band edge
dBas=40 % amplitude stop band peak-to-peak ripple
Was=1000 % Amplitude stop band weight
ftp=0.125 % Delay pass band edge
td=10 % Nominal pass band filter group delay
tdr=1 % Delay pass band peak-to-peak ripple
Wtp=0.5 % Delay pass band weight
```

The resulting FIR impulse response is:

```
h = [  -0.00514359,  -0.00010522,   0.00903057,   0.01387585, ...
        0.00115297,  -0.02706471,  -0.04272784,  -0.00549323, ...
        0.09937991,   0.23445372,   0.32328674,   0.30323800, ...
        0.17772249,   0.02034029,  -0.07856735,  -0.07822720, ...
       -0.01246172,   0.04662598,   0.05113023,   0.01028393, ...
       -0.03030618,  -0.03523108,  -0.00825181,   0.01957184, ...
        0.02380736,   0.00681772,  -0.01145923,  -0.01507939, ...
       -0.00584348,   0.00414479,   0.01058012 ]';
```

Figure N.19 shows the amplitude response. Figure N.20 shows the pass band amplitude, phase error and delay responses. The pass band phase error is adjusted for the nominal delay.

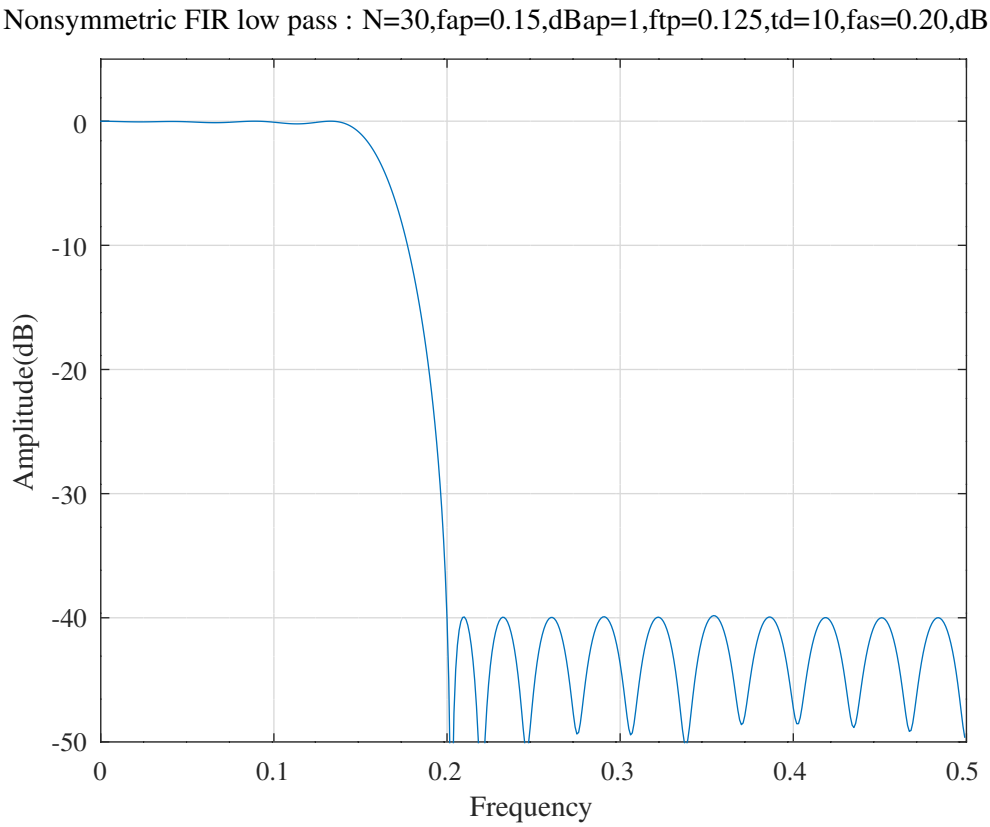



Figure N.19: Amplitude response of a non-symmetric low pass FIR filter designed with SOCP and PCLS.

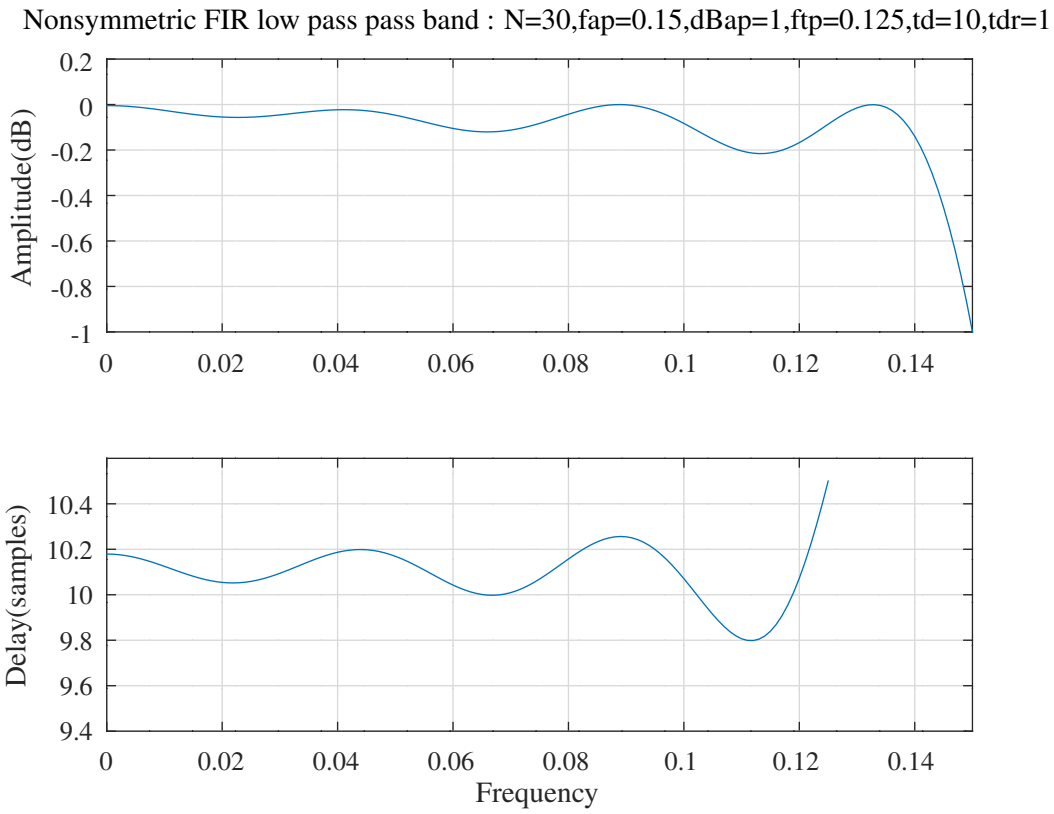



Figure N.20: Pass band amplitude, phase error and delay responses of a non-symmetric low pass FIR filter designed with SOCP and PCLS. The pass band phase error is adjusted for the nominal delay.

**Design of a non-symmetric FIR band pass Hilbert filter**

The Octave script *directFIRnonsymmetric_socp_slb_bandpass_hilbert_test.m* designs a band pass non-symmetric FIR Hilbert filter with a nominal pass band delay using SOCP optimisation and the PCLS exchange algorithm of *Selesnick*, *Lang* and *Burrus*. The FIR filter specification is:

```
tol=0.001 % Tolerance on coefficient update vector
ctol=1e-05 % Tolerance on constraints
n=500 % Frequency points across the band
N=50 % FIR filter order
fapl=0.1 % Pass band squared amplitude lower edge
fapu=0.2 % Pass band squared amplitude upper edge
dBap=0.25 % Pass band squared amplitude ripple
Wap=1 % Pass band squared amplitude weight
fasl=0.05 % Lower stop band squared amplitude lower edge
fasu=0.25 % Upper stop band squared amplitude upper edge
dBas=40 % Stop band squared amplitude response ripple
Wasl=10 % Lower stop band squared amplitude weight
Wasu=20 % Upper stop band squared amplitude weight
ftpl=0.11 % Pass band group delay response lower edge
ftpu=0.19 % Pass band group delay response upper edge
td=10 % Pass band nominal group delay
tdr=0.5 % Pass band group delay response ripple
Wtp=0.5 % Pass band group delay response weight
fppl=0.11 % Pass band phase response lower edge
fppu=0.19 % Pass band phase response upper edge
pd=2.5 % Pass band initial phase (multiples of pi)
ppr=0.02 % Pass band phase response ripple
Wpp=2 % Pass band phase response weight
```

The resulting FIR impulse response is:

```
h = [   0.0035324444,  -0.0139319228,  -0.0094095914,   0.0005149648, ...
       -0.0130492773,  -0.0559859751,  -0.0581356581,   0.0485629272, ...
        0.1956228757,   0.2030320169,  -0.0010246770,  -0.2422353720, ...
       -0.2755442515,  -0.0813542493,   0.1152702456,   0.1330345501, ...
        0.0381148192,   0.0029135455,   0.0541067766,   0.0733941728, ...
        0.0039369788,  -0.0684442066,  -0.0606532570,  -0.0121474951, ...
       -0.0046318830,  -0.0335045767,  -0.0317864266,   0.0128213468, ...
        0.0437733231,   0.0287316552,   0.0031810564,   0.0050190016, ...
        0.0184321224,   0.0105749516,  -0.0133427761,  -0.0228629870, ...
       -0.0114642320,  -0.0003250787,  -0.0023759376,  -0.0066059635, ...
       -0.0017847168,   0.0064154097,   0.0079254383,   0.0032096424, ...
        0.0000385525,   0.0007584860,   0.0014249791,  -0.0001012961, ...
       -0.0013577718,  -0.0015152917,  -0.0007674169 ]';
```

Figure N.21 shows the amplitude response. Figure N.22 shows the pass band amplitude, phase and delay responses. The pass band phase response shown is adjusted for the nominal delay. (In fact the response requires an additional $\pi$ radians phase shift or multiplication of the amplitude by $-1$).

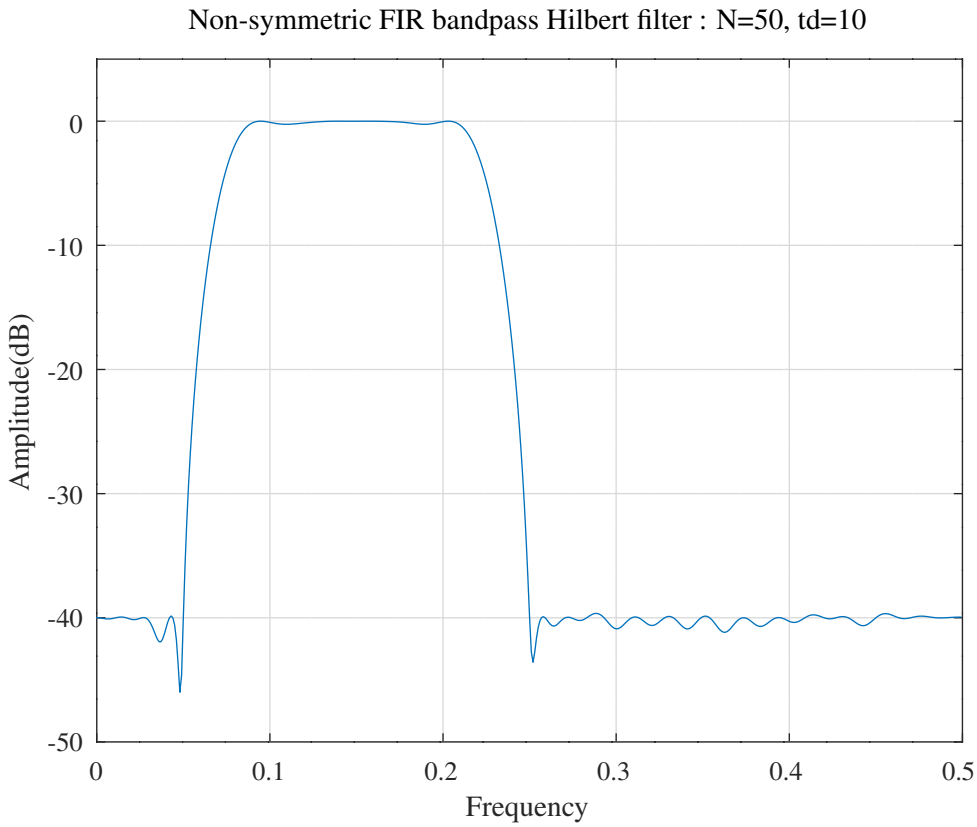



Figure N.21: Amplitude response of a non-symmetric band pass FIR Hilbert filter designed with SOCP and PCLS.

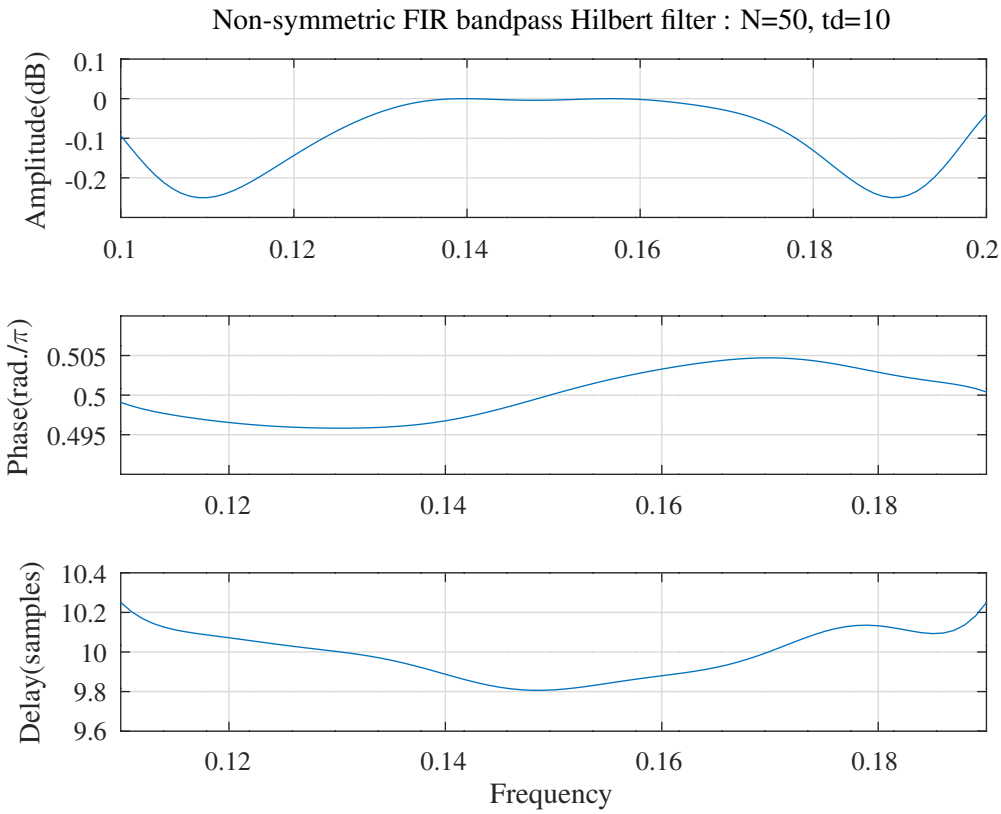



Figure N.22: Pass band amplitude, phase and delay responses of a non-symmetric band pass FIR Hilbert filter designed with SOCP and PCLS. The pass band phase response shown is adjusted for the nominal delay.

## N.6 Constrained mini-max error optimisation of FIR digital filters

### N.6.1 The alternation theorem

A symmetric or linear phase FIR filter with even order $2M$ and a symmetric impulse response with $2M+1$ coefficients has zero-phase frequency response:

$$\begin{aligned} A(\omega) &= h_M + 2\sum_{k=0}^{M-1} h_k \cos(M-k)\omega \\ &= \sum_{k=0}^{M} a_k \cos k\omega \\ &= \sum_{k=0}^{M} a_k T_k(\cos\omega) \end{aligned}$$

where $T_k(x)$ is the $k$-th order *Chebyshev polynomial of the first kind*. This section considers methods of minimising the maximum error:

$$\max_{\omega} E(\omega) = W(\omega)\left|A(\omega) - A_d(\omega)\right|$$

where $W(\omega)$ is a weighting function and $A_d(\omega)$ is the desired response. This is also called the *Chebyshev distance*. For a given filter order (assumed to be even) the number of frequency constraints is limited by the *approximation* theorem, shown by *Roberts* and *Mullis* [168, p. 192] as:

*Theorem (Chebyshev)*: Let $d(x)$ be a real-valued function which is continuous on $-1 \le x \le 1$.

1. There is a unique polynomial $p(x)$ of degree at most $M$ which minimizes $\|d(x) - p(x)\|_\infty$.
2. $p(x)$ is that polynomial if and only if there exist points $-1 \le x_0 \le x_1 \le \ldots \le x_{n+1} \le 1$ for which

$$d(x_i) - p(x_i) = (-1)^i \varepsilon\,, \quad i = 0, 1, \ldots, n+1$$

where $|\varepsilon| = \|d(x) - p(x)\|_\infty$.

*Oppenheim* and *Schafer* show a similar theorem [143, Page 489]:

*Alternation Theorem*: Let $F_P$ denote the closed subset consisting of the disjoint union of closed subsets of the real axis $x$. Then

$$P(x) = \sum_{k=0}^{r} a_k x^r$$

is an $r$-th order polynomial. Also, $D_P(x)$ denotes a given desired function of $x$ that is continuous on $F_P$; $W_P(x)$ is a positive function, continuous on $F_P$, and

$$E_P(x) = W_P(x)\left[D_P(x) - P(x)\right]$$

is the weighted error. The maximum error is defined as

$$\|E\| = \max_{x \in F_P} |E_P(x)|$$

A necessary and sufficient condition that $P(x)$ be the unique $r$th order polynomial that minimises $\|E\|$ is that $E_P(x)$ exhibit *at least* $(r+2)$ alternations; i.e., there must exist at least $(r+2)$ values $x_i$ in $F_P$ such that $x_1 < x_2 < \ldots < x_{r+2}$ and such that $E_P(x_i) = -E_P(x_{i+1}) = \|E\|$ for $i = 1, 2, \ldots, (r+1)$.

*Oppenheim* and *Schafer* illustrate the conditions under which an extremum of a polynomial function is also considered to be an *alternation* [143, Figure 7.34]. They point out that, for a polynomial of order $M$, the Alternation Theorem requires that the

optimal approximation have at least $M+2$ alternations (the *equi-ripple* approximation) and that a low-pass filter may have at most $M+3$ alternations (the *extra-ripple* case). They include alternations at the pass-band edge (for which the polynomial amplitude is $1-\delta_p$) and at the stop-band edge (for which the amplitude is $+\delta_s$). Further, the polynomial amplitude will be equi-ripple except possibly at $\omega=0$ or $\omega=\pi$.

### N.6.2 *Hofstetter*'s algorithm for mini-max FIR filter approximation

*Hofstetter* [41] describes an iterative technique for the design of FIR transfer functions that have the specified pass-band and stop-band ripple. The technique:

> begins by making an initial estimate of the frequencies at which these extrema will occur and the uses Lagrange interpolation to obtain a polynomial that goes through the maximum allowable ripple values at these frequencies. ... The next stage of the algorithm is to locate the frequencies at which the extrema of the first Lagrange interpolation polynomial occur. These frequencies are taken to be a second, hopefully improved, guess as to the frequencies at which the extrema of the filter response will achieve the desired ripple values. ... The algorithm now "closes the loop" by using these new frequencies to construct a Lagrange interpolation polynomial that achieves the desired ripple values at these frequencies. The extrema of this new polynomial are then located and used to start the next cycle of the algorithm. The algorithm is reminiscent of, but different from, the Remes exchange algorithm used in the theory of Tchebycheff approximation.

The pass-band constraint frequencies are $\omega_1,\ldots,\omega_{N_p}$ and the stop-band constraint frequencies are $\omega_{N_p+1},\ldots,\omega_{N_p+N_s}$, where $M+1=N_p+N_s$, $\omega_1=0$ and $\omega_{N_p+N_s}=\pi$. At each iteration, the algorithm solves the system of equations:

$$A(\omega_l)=\begin{cases}1+(-1)^{l+c}\,\delta_p & \text{for } 1\le l\le N_p\\ (-1)^{l+c}\,\delta_s & \text{for } N_p+1\le l\le N_p+N_s\end{cases}$$

where $c$ is chosen to be 0 or 1. These equations can be solved efficiently by *barycentric Lagrange interpolation*[f] [41, Page 5] [206, Page 192] [83].

The Octave script *hofstetterFIRsymmetric_lowpass_test.m* uses an implementation of *Hofstetter*'s algorithm in the Octave function *hofstetterFIRsymmetric.m* to design a low-pass filter. The filter specification is:

```
M=41 % Filter order is 2*M
fap=0.1 % Amplitude pass band edge
nMp+1=10 % Amplitude pass band alternations
deltap=0.0001 % Amplitude pass band peak-to-peak ripple
fas=0.2 % Amplitude stop band edge
nMs+1=32 % Amplitude stop band alternations
deltas=1e-06 % Amplitude stop band peak-to-peak ripple
nplot=2000 % Number of frequencies
tol=1e-05 % Tolerance on convergence
```

Figure N.23 shows the pass-band and stop-band amplitude responses of the filter. Figure N.24 shows the zeros of the filter. The distinct filter coefficients are:

```
hM = [  -0.0000062744,  -0.0000244944,  -0.0000551827,  -0.0000819533, ...
        -0.0000704428,   0.0000126058,   0.0001616955,   0.0003015039, ...
         0.0003011099,   0.0000545772,  -0.0004030099,  -0.0008306872, ...
        -0.0008611077,  -0.0002397438,   0.0008969019,   0.0019246655, ...
         0.0019925489,   0.0005973054,  -0.0018561923,  -0.0039713247, ...
        -0.0040237875,  -0.0011533299,   0.0036425566,   0.0075576337, ...
         0.0074125952,   0.0018804923,  -0.0068946723,  -0.0136868237, ...
        -0.0129477931,  -0.0026876364,   0.0129511570,   0.0246280536, ...
         0.0226298007,   0.0034332146,  -0.0258331372,  -0.0483977600, ...
        -0.0449658253,  -0.0039620478,   0.0704678795,   0.1575442544, ...
         0.2274359299,   0.2541534897 ]';
```

The Octave script *hofstetterFIRsymmetric_bandpass_test.m* calls the Octave function *hofstetterFIRsymmetric.m* to design a band-pass filter. The filter specification is:

[f] The Octave *polyfit* built-in function inverts the corresponding *Vandermonde* matrix

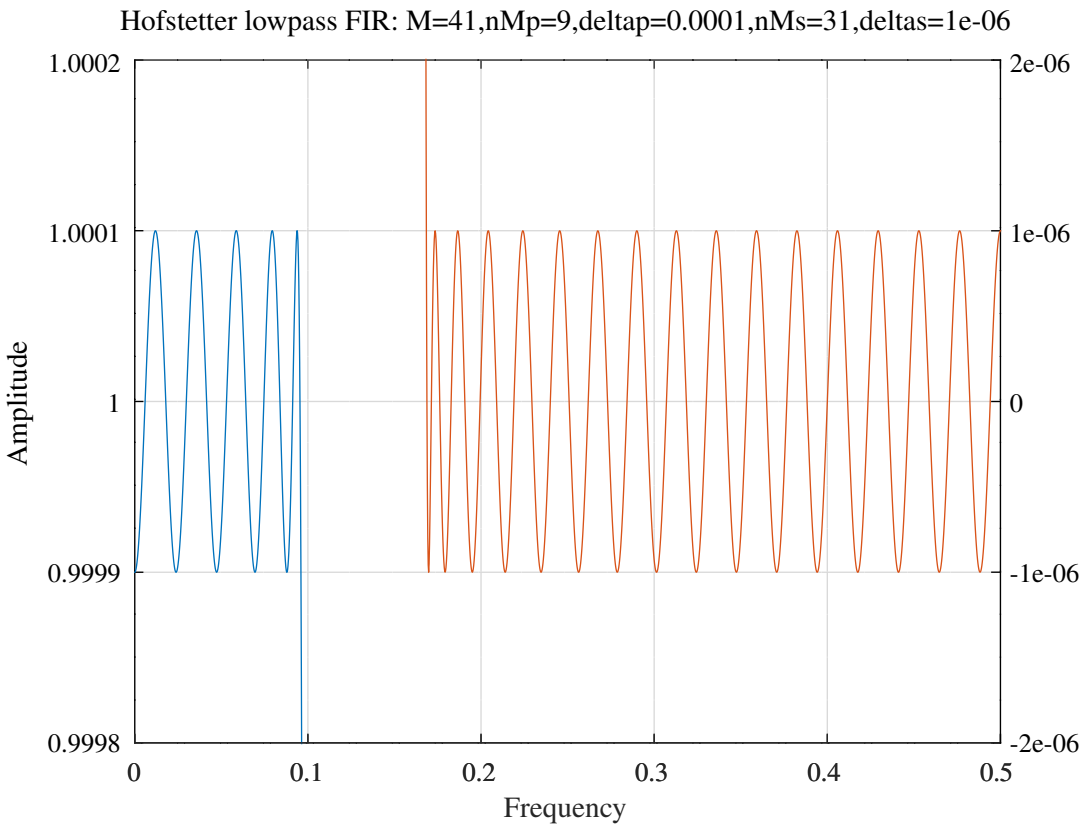



Figure N.23: Response of a mini-max FIR low-pass filter designed with *Hofstetter*'s algorithm

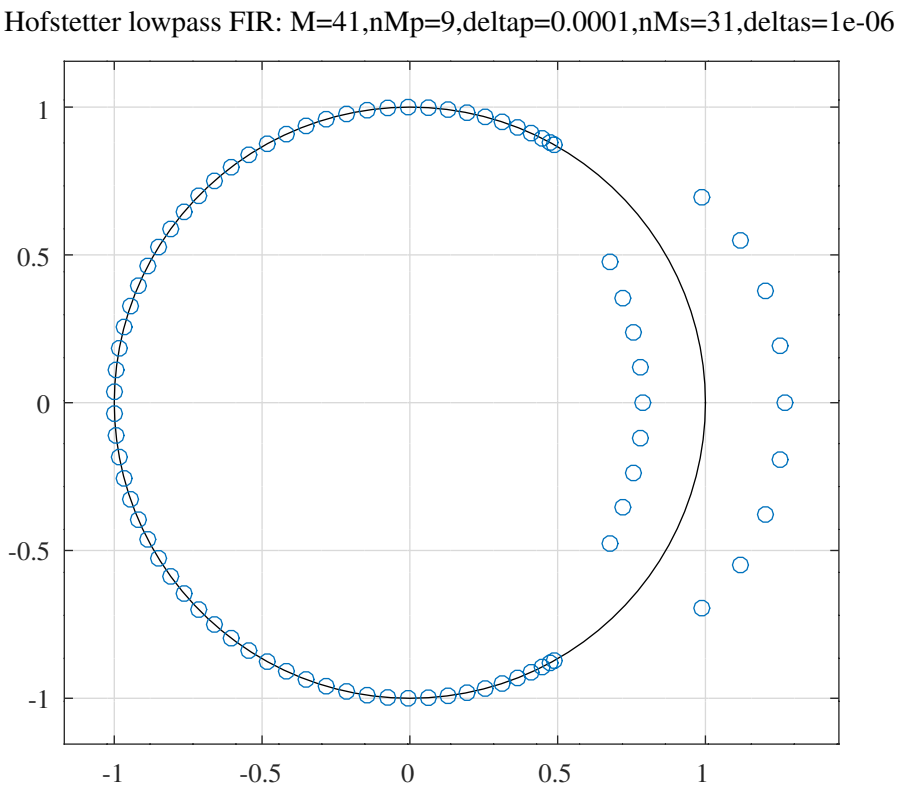



Figure N.24: Zeros of a mini-max FIR low-pass filter designed with *Hofstetter*'s algorithm

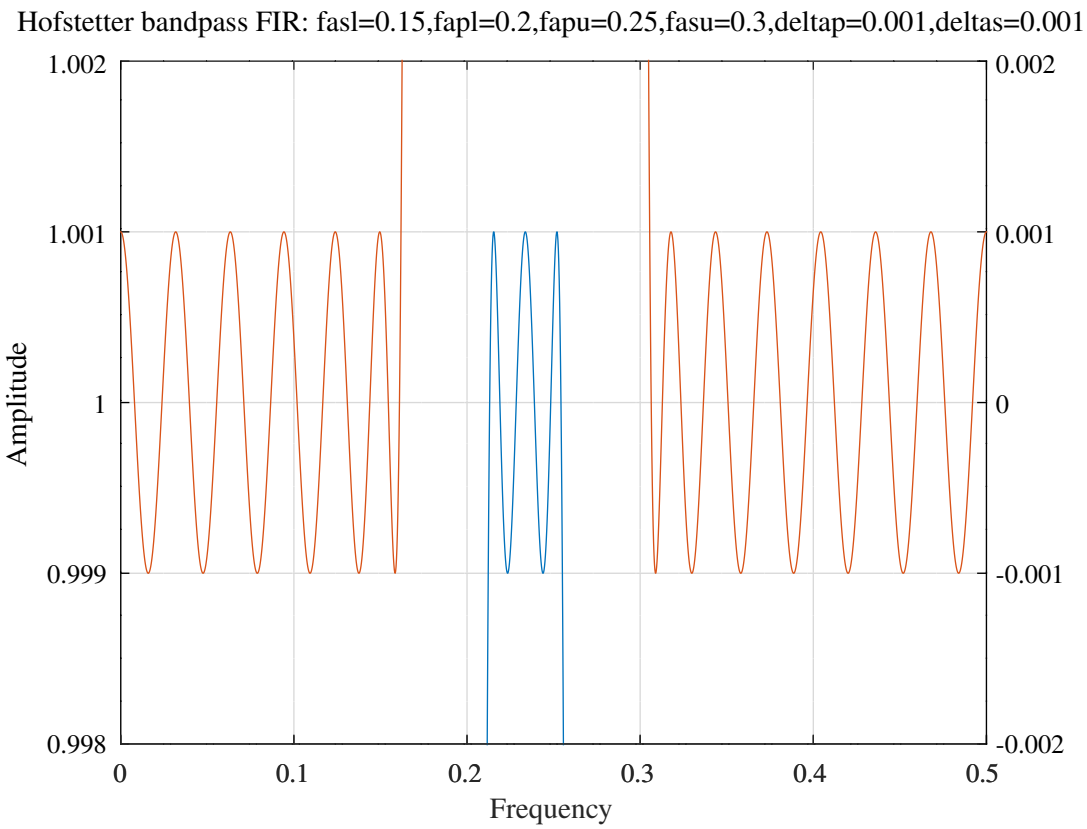



Figure N.25: Response of a mini-max FIR band-pass filter designed with *Hofstetter*'s algorithm

```
M=30 % Filter order is 2*M (M+1 distinct coefficients)
fasl=0.15 % Amplitude stop band lower edge
fapl=0.2 % Amplitude pass band lower edge
fapu=0.25 % Amplitude pass band upper edge
fasu=0.3 % Amplitude stop band upper edge
deltap=0.001 % Amplitude pass band peak-to-peak ripple
deltas=0.001 % Amplitude stop band peak-to-peak ripple
nplot=2000 % Number of frequencies
tol=1e-05 % Tolerance on convergence
```

Figure N.25 shows the pass-band and stop-band amplitude responses of the filter. Figure N.26 shows the zeros of the filter. The distinct filter coefficients are:

```
hM = [   0.0016403235,   0.0003811934,  -0.0031154274,  -0.0014736974, ...
         0.0041494943,   0.0026338117,  -0.0033919821,  -0.0021854812, ...
         0.0005763605,  -0.0018192528,   0.0028371388,   0.0098604875, ...
        -0.0041768995,  -0.0193489621,   0.0016309399,   0.0246939527, ...
         0.0032233259,  -0.0198235341,  -0.0044329061,   0.0022230559, ...
        -0.0062571008,   0.0239834184,   0.0346597112,  -0.0484236390, ...
        -0.0792843518,   0.0589851401,   0.1296762482,  -0.0483396210, ...
        -0.1697773982,   0.0186531279,   0.1850850479 ]';
```

The Octave script *hofstetterFIRsymmetric_multiband_test.m* calls the Octave function *hofstetterFIRsymmetric.m* to design a multi-band filter. The filter specification is:

```
M=30 % Filter order is 2*M
fasu1=0.1 % Amplitude first stop-band upper edge
fapl1=0.15 % Amplitude first pass band lower edge
fapu1=0.2 % Amplitude first pass band upper edge
fasl2=0.25 % Amplitude second stop band lower edge
fasu2=0.3 % Amplitude second stop band upper edge
fapl2=0.35 % Amplitude second pass band lower edge
fapu2=0.4 % Amplitude second pass band upper edge
fasl3=0.45 % Amplitude third stop band lower edge
```

Hofstetter bandpass FIR: fasl=0.15,fapl=0.2,fapu=0.25,fasu=0.3,deltap=0.001,deltas=0.001

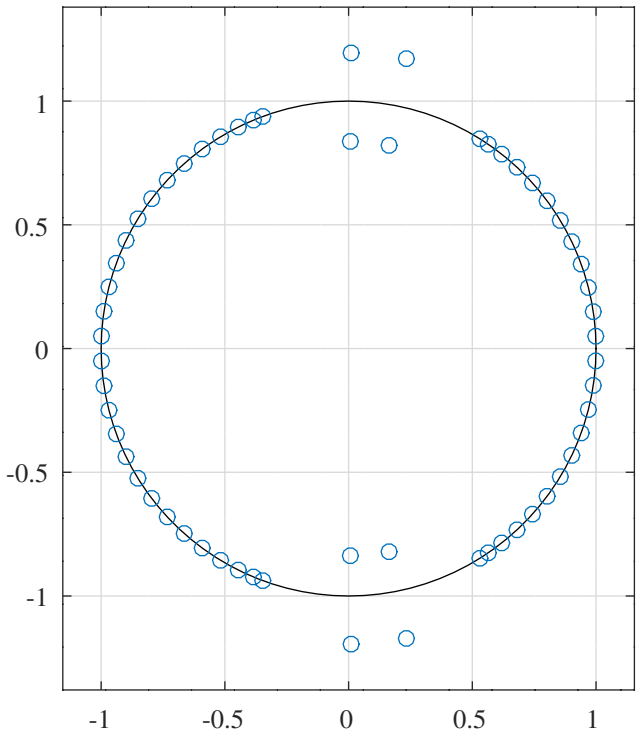



Figure N.26: Zeros of a mini-max FIR band-pass filter designed with *Hofstetter*'s algorithm

```
nplot=2000 % Number of frequency grid points in [0,0.5]
maxiter=100 % Maximum iterations
tol=1e-12 % Tolerance on convergence
```

Some experimentation was required to find the number of extrema in each band that approximately satisfied the specification. Figure N.27 shows the pass-band and stop-band amplitude responses of the filter. Figure N.28 shows the zeros of the filter. The distinct filter coefficients are:

```
hM = [  -0.0069124566,   0.0077531943,   0.0019595959,  -0.0005177889, ...
         0.0001182649,  -0.0063168091,   0.0014745510,  -0.0006395502, ...
         0.0018024181,  -0.0104721027,   0.0209069674,   0.0112324220, ...
        -0.0214733804,   0.0017516866,  -0.0314988245,   0.0168413105, ...
         0.0289331993,  -0.0063863456,  -0.0032179597,   0.0109003138, ...
         0.0150475050,  -0.0727626671,   0.0078394926,  -0.0351408173, ...
         0.0672651896,   0.1895697499,  -0.1682471918,  -0.0573225385, ...
        -0.0867759493,  -0.0462400577,   0.3400571569 ]';
```

Hofstetter multi-band FIR: M=30,fasu1=0.1,fapl1=0.15,fapu1=0.2,fasl2=0.25,fasu2=0.3,fapl2=0.35,fapu2=0.4,fasl3=0.45

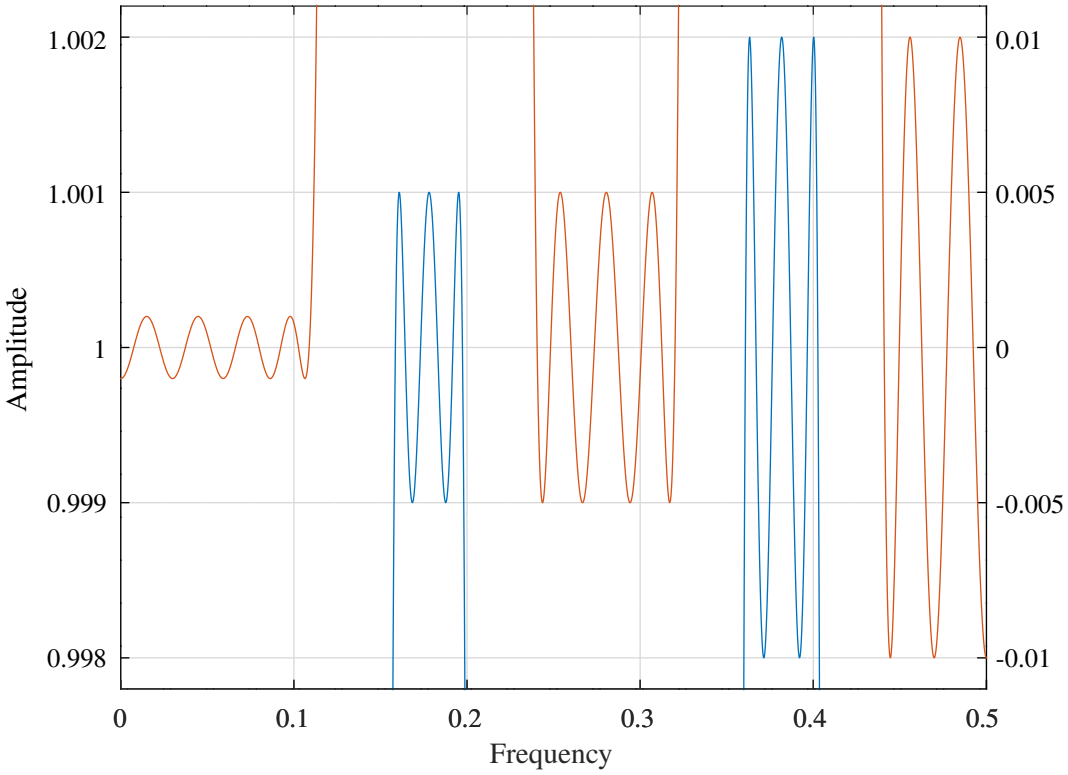



Figure N.27: Response of a mini-max FIR multi-band filter designed with *Hofstetter*'s algorithm

Hofstetter multi-band FIR: M=30,fasu1=0.1,fapl1=0.15,fapu1=0.2,fasl2=0.25,fasu2=0.3,fapl2=0.35,fapu2=0.4,fasl3=0.45

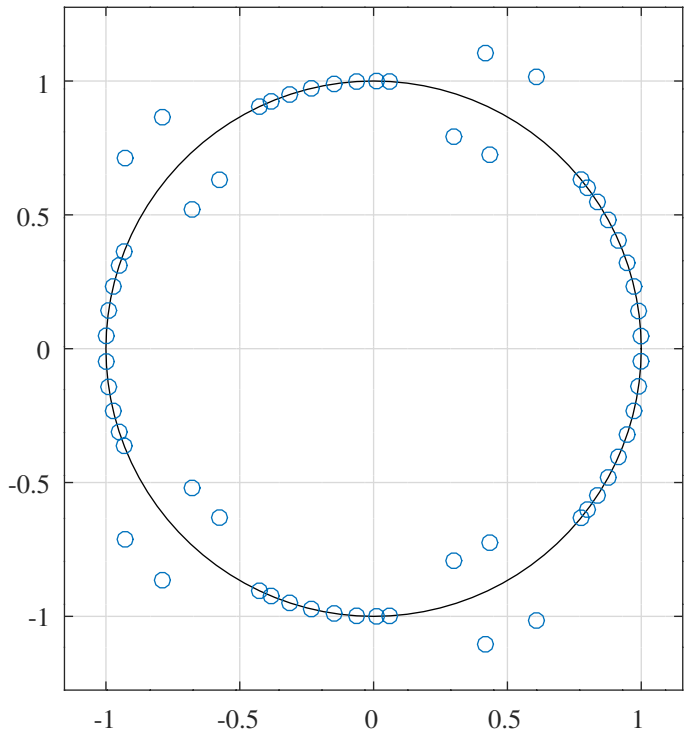



Figure N.28: Zeros of a mini-max FIR multi-band filter designed with *Hofstetter*'s algorithm

***Selesnick-Burrus* modification to *Hofstetter*'s algorithm for low-pass filters**

*Selesnick* and *Burrus* [74] point out that whilst *Hofstetter*'s algorithm "produces equi-ripple filters with the specified $\delta_p$ and $\delta_s$, it is not widely used because if allows limited control over the location of the band edges and only produces extra-ripple filters". They add a non-extremal frequency constraint in the transition band at $\omega_t$.

Suppose $\omega_1, \ldots, \omega_{M+1}$ is the ordered reference set of frequencies in $[0, \pi]$ including $\omega_t$. Let $\omega_1, \ldots, \omega_{q-1}$ be the frequencies in the pass-band, less than $\omega_t$, and $\omega_{q+1}, \ldots, \omega_{M+1}$ be the frequencies in the stop-band, greater than $\omega_t$. The system of equations to be solved at each iteration is:

$$\begin{aligned} A(\omega_k) &= 1 + (-1)^{k+c}\,\delta_p \quad \text{for } 1 \le k \le q-1 \\ A(\omega_t) &= A_t \\ A(\omega_k) &= (-1)^{k+c+1}\,\delta_s \quad \text{for } q+1 \le k \le M+1 \end{aligned}$$

where $c$ is chosen to equal 0 or 1, whichever yields $A(\omega_{q-1}) = 1 + \delta_p$.

The exchange algorithm is described as follows:

> Let $S$ be the set obtained by appending $\omega_t$ to the set of extrema of $A(\omega)$ in $[0, \pi]$:$S$ will have either $M+1$ or $M+2$ frequencies and will include both 0 and $\pi$. If $S$ has $M+1$ frequencies, then take the new reference set to be $S$. If $S$ has $M+2$ frequencies, then remove either 0 or $\pi$ from $S$ according to the following rules:
>
> 1. If $A(\omega)$ has no extrema in the open interval $(0, \omega_t)$, then remove 0 from $S$.
> 2. If $A(\omega)$ has no extrema in the open interval $(\omega_t, \pi)$ then remove $\pi$ from $S$.
> 3. Otherwise, let $\omega_a$ be the extrema of $A(\omega)$ in $(0, \omega_t)$ closest to 0, and let $\omega_b$ be the extrema of $A(\omega)$ in $(\omega_t, \pi)$ closest to $\pi$. If $|A(0) - A(\omega_a)|\,\delta_s < |A(\pi) - A(\omega_b)|\,\delta_p$ then remove 0 from $S$, otherwise remove $\pi$ from $S$.

The Octave script *selesnickFIRsymmetric_lowpass_test.m* designs a low-pass filter with the implementation of *Hofstetter*'s algorithm with the modifications of *Selesnick* and *Burrus* in the Octave function *selesnickFIRsymmetric_lowpass.m*. The extremal frequencies are found by quadratic interpolation on a coarse grid. The filter specification is:

```
M=85 % Filter order is 2*M
deltap=1e-06 % Amplitude pass band peak-to-peak ripple
deltas=1e-08 % Amplitude stop band peak-to-peak ripple
ft=0.15 % Amplitude transition band frequency
At=deltas % Amplitude at transition band frequency
nf=1000 % Number of frequencies
tol=1e-10 % Tolerance on convergence
```

Figure N.29 shows the pass-band and stop-band amplitude responses of the filter. Figure N.30 shows the zeros of the filter. The distinct filter coefficients are:

```
hMb = [ -0.000000028464, -0.000000153958, -0.000000403711, -0.000000727907, ...
        -0.000000892974, -0.000000546300,  0.000000570241,  0.000002265612, ...
         0.000003664746,  0.000003400698,  0.000000401587, -0.000004991701, ...
        -0.000010311780, -0.000011528427, -0.000005231465,  0.000008360603, ...
         0.000023488812,  0.000030146071,  0.000019275280, -0.000010100860, ...
        -0.000046245182, -0.000067204352, -0.000051859592,  0.000004763823, ...
         0.000081191523,  0.000133669184,  0.000117644666,  0.000018117502, ...
        -0.000129093667, -0.000243323166, -0.000237847085, -0.000076164707, ...
         0.000186850059,  0.000411913210,  0.000441094361,  0.000196406775, ...
        -0.000244907152, -0.000655562899, -0.000763744250, -0.000417472963, ...
         0.000284238786,  0.000988506442,  0.001249667324,  0.000791673374, ...
        -0.000273011969, -0.001420365493, -0.001949797405, -0.001387242585, ...
         0.000162895090,  0.001953346945,  0.002922263494,  0.002291701772, ...
         0.000115518653, -0.002579849867, -0.004234794301, -0.003618670001, ...
        -0.000659833998,  0.003280997826,  0.005972817456,  0.005523459974, ...
         0.001609851838, -0.004026537376, -0.008260761443, -0.008240290194, ...
        -0.003179707219,  0.004776364405,  0.011315604898,  0.012175913404, ...
         0.005740487402, -0.005483681310, -0.015590866477, -0.018172416563, ...
        -0.010064339136,  0.006099495087,  0.022236869275,  0.028410286058, ...
         0.018233496964, -0.006577897529, -0.035134157916, -0.050931701456, ...
        -0.038934677581,  0.006881382067,  0.079164410066,  0.158436702296, ...
         0.219889962745,  0.243014607974 ]';
```

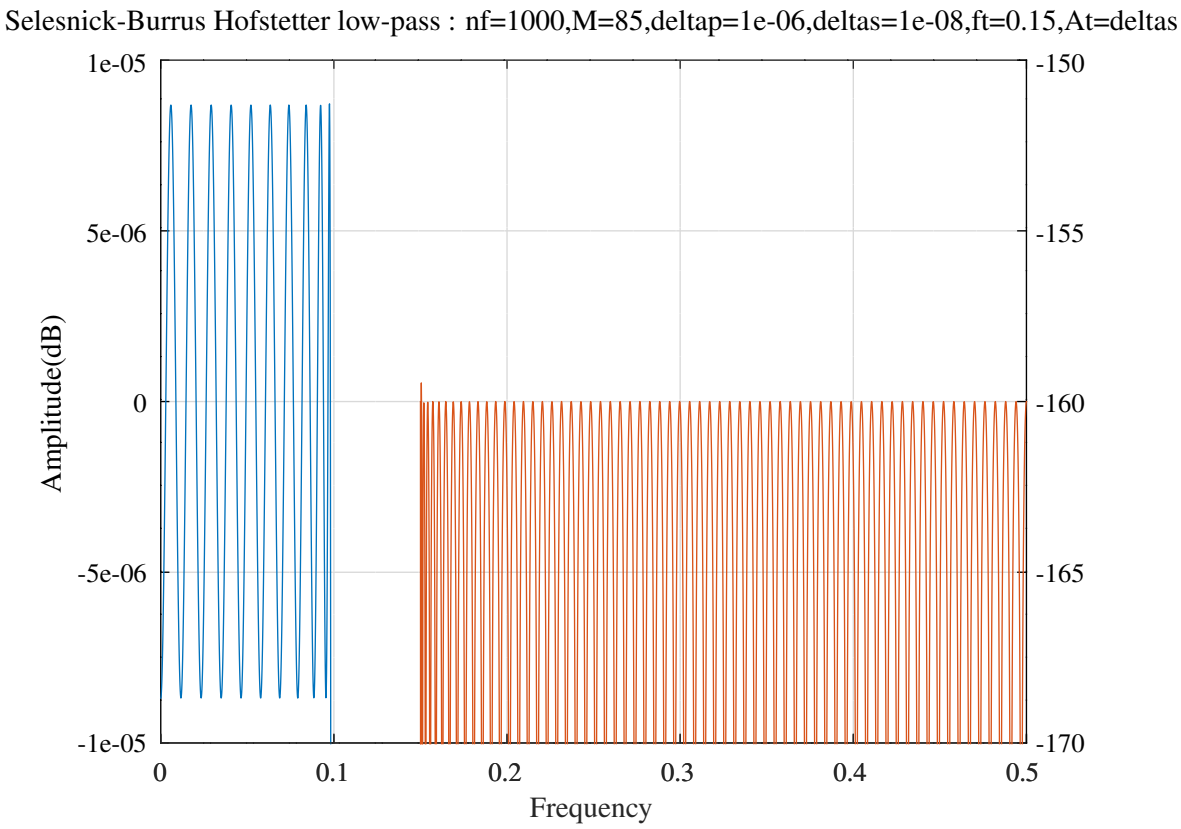



Figure N.29: Response of a mini-max FIR low-pass filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*

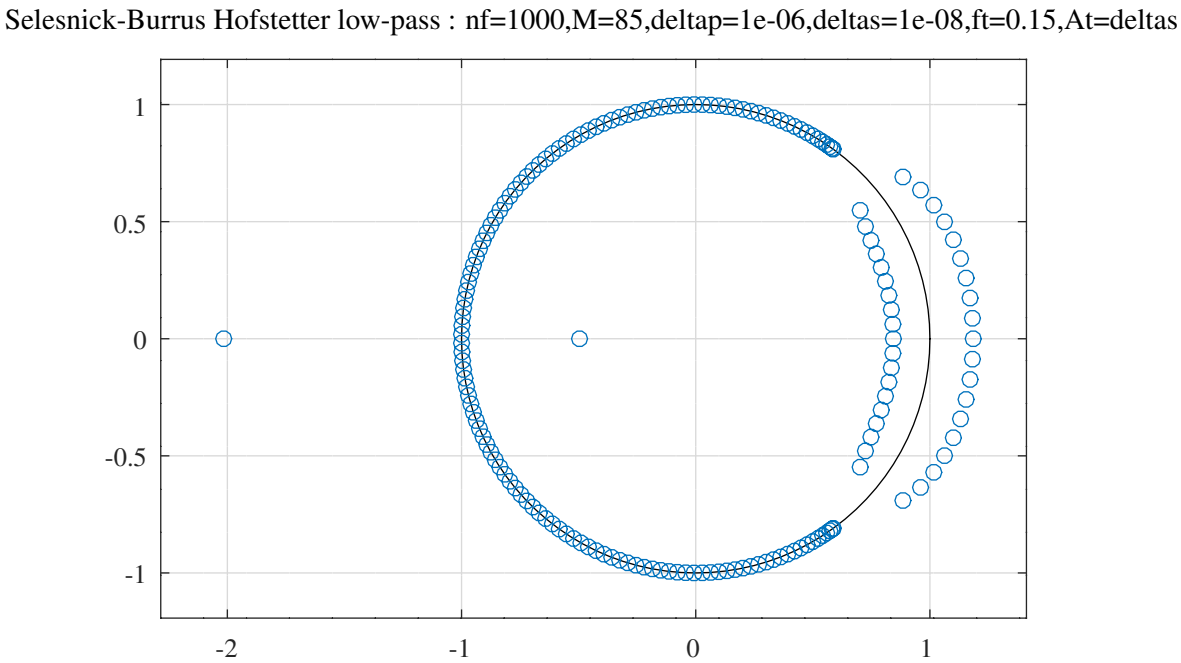



Figure N.30: Zeros of a mini-max FIR low-pass filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*

The Octave script *selesnickFIRsymmetric_halfband_test.m* calls the Octave function *selesnickFIRsymmetric_lowpass* to design a half-band filter with the specification:

```
M=199 % Filter order is 2*M
delta=1e-06 % Amplitude peak-to-peak ripple
ft=0.25 % Amplitude transition band frequency
At=0.5 % Amplitude at transition band frequency
nf=2000 % Number of frequencies
tol=1e-10 % Tolerance on convergence
```

For the half-band filter, $\delta_p = \delta_s$, $f_t = 0.25$ and $A_t = 0.5$. Figure N.31 shows the amplitude response of the half-band filter. The distinct filter coefficients are:

```
hM = [ -0.000000748558,  0.000000000000,  0.000000603215,  0.000000000000, ...
       -0.000000843471,  0.000000000000,  0.000001144926,  0.000000000000, ...
       -0.000001518383,  0.000000000000,  0.000001975976,  0.000000000000, ...
       -0.000002531273,  0.000000000000,  0.000003199387,  0.000000000000, ...
       -0.000003997086,  0.000000000000,  0.000004942909,  0.000000000000, ...
       -0.000006057285,  0.000000000000,  0.000007362648,  0.000000000000, ...
       -0.000008883565,  0.000000000000,  0.000010646856,  0.000000000000, ...
       -0.000012681718,  0.000000000000,  0.000015019857,  0.000000000000, ...
       -0.000017695606,  0.000000000000,  0.000020746059,  0.000000000000, ...
       -0.000024211195,  0.000000000000,  0.000028134007,  0.000000000000, ...
       -0.000032560623,  0.000000000000,  0.000037540439,  0.000000000000, ...
       -0.000043126237,  0.000000000000,  0.000049374311,  0.000000000000, ...
       -0.000056344588,  0.000000000000,  0.000064100749,  0.000000000000, ...
       -0.000072710348,  0.000000000000,  0.000082244931,  0.000000000000, ...
       -0.000092780151,  0.000000000000,  0.000104395888,  0.000000000000, ...
       -0.000117176362,  0.000000000000,  0.000131210254,  0.000000000000, ...
       -0.000146590824,  0.000000000000,  0.000163416029,  0.000000000000, ...
       -0.000181788657,  0.000000000000,  0.000201816450,  0.000000000000, ...
       -0.000223612244,  0.000000000000,  0.000247294121,  0.000000000000, ...
       -0.000272985565,  0.000000000000,  0.000300815635,  0.000000000000, ...
       -0.000330919162,  0.000000000000,  0.000363436959,  0.000000000000, ...
       -0.000398516066,  0.000000000000,  0.000436310020,  0.000000000000, ...
       -0.000476979166,  0.000000000000,  0.000520691015,  0.000000000000, ...
       -0.000567620643,  0.000000000000,  0.000617951169,  0.000000000000, ...
       -0.000671874287,  0.000000000000,  0.000729590893,  0.000000000000, ...
       -0.000791311804,  0.000000000000,  0.000857258593,  0.000000000000, ...
       -0.000927664555,  0.000000000000,  0.001002775826,  0.000000000000, ...
       -0.001082852685,  0.000000000000,  0.001168171059,  0.000000000000, ...
       -0.001259024280,  0.000000000000,  0.001355725119,  0.000000000000, ...
       -0.001458608168,  0.000000000000,  0.001568032601,  0.000000000000, ...
       -0.001684385417,  0.000000000000,  0.001808085218,  0.000000000000, ...
       -0.001939586653,  0.000000000000,  0.002079385630,  0.000000000000, ...
       -0.002228025473,  0.000000000000,  0.002386104185,  0.000000000000, ...
       -0.002554283079,  0.000000000000,  0.002733297041,  0.000000000000, ...
       -0.002923966807,  0.000000000000,  0.003127213709,  0.000000000000, ...
       -0.003344077459,  0.000000000000,  0.003575737742,  0.000000000000, ...
       -0.003823540551,  0.000000000000,  0.004089030545,  0.000000000000, ...
       -0.004373991051,  0.000000000000,  0.004680493905,  0.000000000000, ...
       -0.005010962059,  0.000000000000,  0.005368248902,  0.000000000000, ...
       -0.005755739754,  0.000000000000,  0.006177483074,  0.000000000000, ...
       -0.006638362059,  0.000000000000,  0.007144321892,  0.000000000000, ...
       -0.007702674910,  0.000000000000,  0.008322516718,  0.000000000000, ...
       -0.009015303348,  0.000000000000,  0.009795667219,  0.000000000000, ...
       -0.010682595676,  0.000000000000,  0.011701175158,  0.000000000000, ...
       -0.012885245283,  0.000000000000,  0.014281569434,  0.000000000000, ...
       -0.015956639253,  0.000000000000,  0.018008282147,  0.000000000000, ...
       -0.020586554565,  0.000000000000,  0.023933921137,  0.000000000000, ...
       -0.028469265238,  0.000000000000,  0.034983913239,  0.000000000000, ...
       -0.045173699724,  0.000000000000,  0.063447988407,  0.000000000000, ...
       -0.105974775276,  0.000000000000,  0.318267024988,  0.500000000000 ]';
```

Assuming $M$ is odd, the following Octave code converts an even-order half-band filter with $M+1$ distinct coefficients, $h_M$, into a Hilbert filter (see Section N.4) and plots the amplitude response:

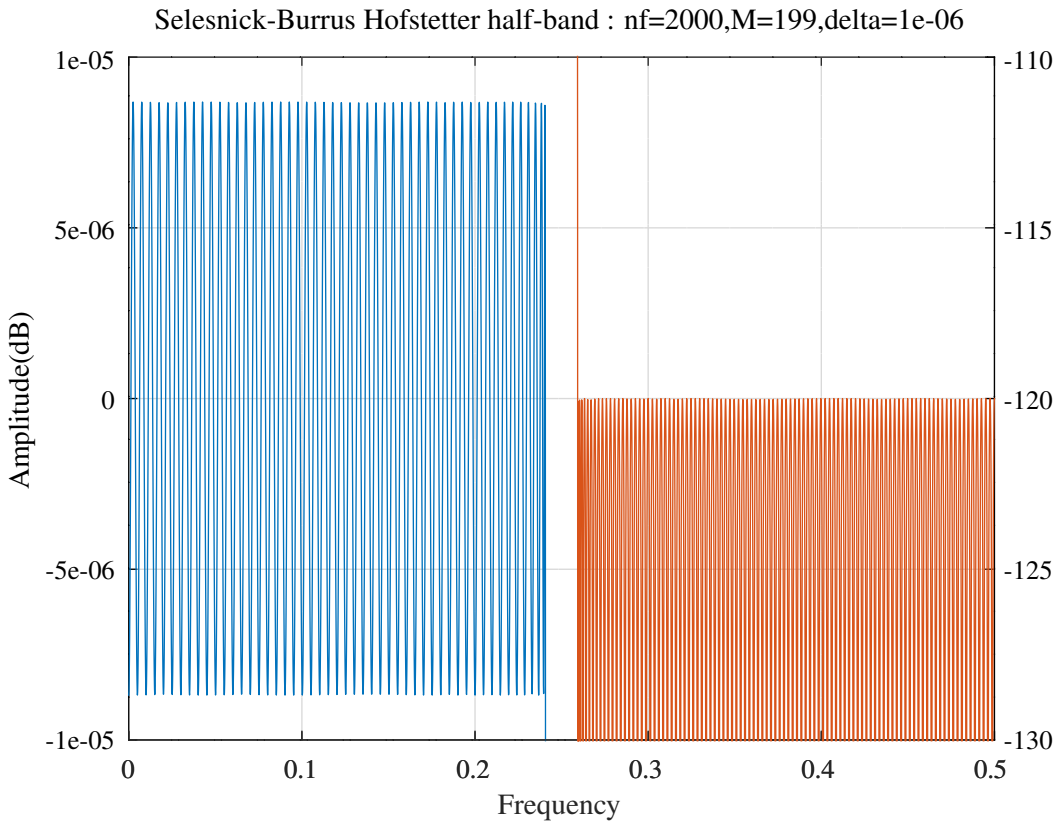



Figure N.31: Pass-band and stop-band response of a mini-max FIR half-band filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*

```
altm1=zeros((2*M)+1,1);
altm1(1:2:end)=((-1).^(0:M))';
hhilbert=2*[hM(1:M);0;hM(M:-1:1)].*altm1;
[Hhilbert,w]=freqz(hhilbert,1,4096);
subplot(211)
plot(w*0.5/pi,imag(Hhilbert.*exp(j*w*M)));
subplot(212)
plot(w*0.5/pi,mod((unwrap(angle(Hhilbert))+(w*(M)))/pi,2));
```

Figure N.32 shows the amplitude and phase responses of the Hilbert filter derived from the half-band filter shown in Figure N.31. The phase response shown is adjusted for the nominal delay. Figure N.33 shows the zeros of the Hilbert filter.

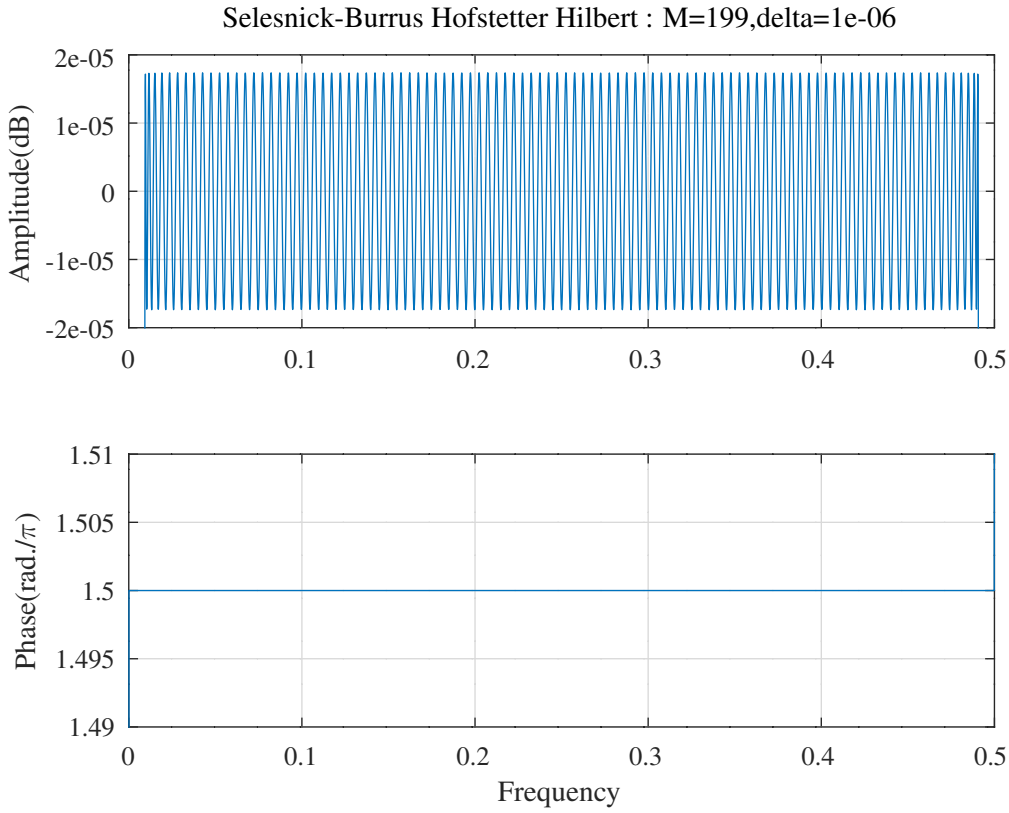



Figure N.32: Amplitude response of a mini-max FIR Hilbert filter derived from the half-band filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*. The phase response shown is adjusted for the nominal delay.

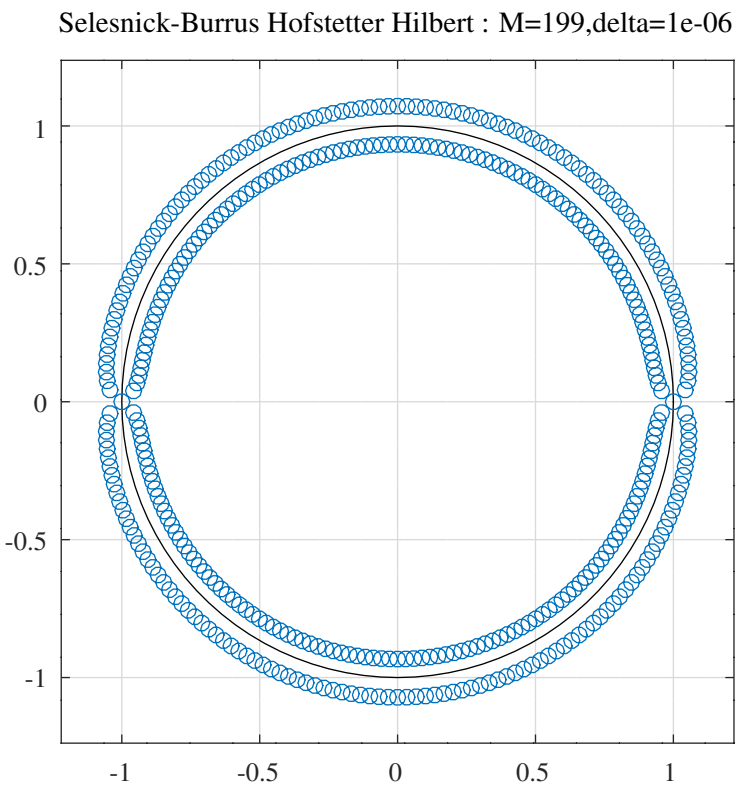



Figure N.33: Zeros of a mini-max FIR Hilbert filter derived from the half-band filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*

***Selesnick-Burrus* modification to *Hofstetter*'s algorithm for band-pass filters**

*Selesnick* and *Burrus* [74, Section II.C] also describe an exchange algorithm for band-pass filters with lower stop-band, pass-band and upper stop-band errors $\delta_{sl}$, $\delta_p$ and $\delta_{su}$ and lower and upper transition-band frequencies $\omega_{tl}$ and $\omega_{tu}$. An additional constraint makes the slope of $A(\omega)$ is equal in magnitude and opposite in sign at the transition frequencies. *Selesnick* and *Burrus* state that: "because the reference set must contain $M-2$ extremal frequencies, it will therefore be necessary to exclude 0, 1, 2 or 3 local minima and maxima when updating the reference set." Suppose $\omega_1, \ldots, \omega_L$ are the local extrema of $A(\omega)$ listed in order. The procedure for excluding maxima and minima is:

1. To exclude 1 local extremum $(L = M-1)$, use the same update rule used for the low-pass case.
2. To exclude 2 local extrema $(L = M)$, find the index $k$ that minimises

$$[E(\omega_k) - E(\omega_{k+1})](-1)^{k+c}$$

   where $c = 1$ if $\omega_1$ is a local maximum and $c = 0$ if $\omega_1$ is a local minimum. $E(\omega)$ denotes the error function. ... If $1 < k < M-2$, then exclude $\omega_k$ and $\omega_{k+1}$ from the reference set. If $k = 1$ or $K = L$, then exclude $\omega_k$ and use the procedure above for excluding 1 local extremum.
3. To exclude 3 local extrema $(L = M+1)$, use the procedure for excluding 1 extremum, followed by the procedure for excluding 2 extrema.

The Octave script *selesnickFIRsymmetric_bandpass_test.m* designs a band-pass filter with the implementation of *Hofstetter*'s algorithm with the modifications of *Selesnick* and *Burrus* in the Octave function *selesnickFIRsymmetric_bandpass.m*. Unlike the MATLAB implementation made available by *Selesnick*, *firebp.m* [189], this function uses Lagrange interpolation to find the response in the $x = \cos\omega$ domain rather than using left division to find the coefficients directly and does not attempt to implement the constraint on the slope of the response at the transition frequencies. Consequently, the exchange algorithm uses a reference set of $M-1$, rather than $M-2$, extremal frequencies plus the two transition band frequencies and only 1 or, at most, 2 extremal frequencies need be removed. The case of removing 3 frequencies, listed above, should not occur. The extremal frequencies are found by quadratic interpolation. The filter specification is:

```
M=255 % Filter order is 2*M
deltasl=1e-05 % Amplitude lower stop-band peak ripple
deltap=0.0001 % Amplitude pass-band peak ripple
deltasu=1e-05 % Amplitude upper stop-band peak ripple
ftl=0.2 % Amplitude lower transition band frequency
ftu=0.35 % Amplitude upper transition band frequency
At=1-deltap % Amplitude at transition band frequencies
nf=1800 % Number of frequencies
tol=1e-10 % Tolerance on convergence
```

Figure N.34 shows the pass-band and stop-band amplitude responses of the filter. Figure N.35 shows the zeros of the filter. The distinct filter coefficients are:

```
hMc = [  0.000006348104, -0.000004299772,  0.000002791447,  0.000009838670, ...
        -0.000016340858, -0.000006019899,  0.000025662821, -0.000004340389, ...
        -0.000017844362,  0.000005442330, -0.000000992472,  0.000010620744, ...
         0.000009883327, -0.000026480225, -0.000001831907,  0.000017373474, ...
        -0.000003349623,  0.000012615043, -0.000011740680, -0.000029515055, ...
         0.000028860412,  0.000014647020, -0.000013259780,  0.000006121987, ...
        -0.000031307438,  0.000000332339,  0.000054063278, -0.000018940638, ...
        -0.000022538967,  0.000002793994, -0.000025938125,  0.000048995871, ...
         0.000031811923, -0.000069723386, -0.000001577563,  0.000012750729, ...
         0.000000678249,  0.000068211551, -0.000046261348, -0.000078387345, ...
         0.000061313076,  0.000014458111,  0.000016790759,  0.000026596959, ...
        -0.000119252699,  0.000005987478,  0.000114357621, -0.000029943763, ...
         0.000006871030, -0.000043830112, -0.000100158101,  0.000144616338, ...
         0.000065528880, -0.000109425488,  0.000005994184, -0.000073179582, ...
         0.000025111674,  0.000203530237, -0.000109736461, -0.000127839115, ...
         0.000058072923, -0.000038930719,  0.000154180099,  0.000074480824, ...
        -0.000281837312,  0.000015800669,  0.000125310194, -0.000004544061, ...
         0.000158816666, -0.000179119834, -0.000240950249,  0.000271665592, ...
         0.000077747785, -0.000035352918,  0.000032436155, -0.000331854462, ...
         0.000079587394,  0.000394072083, -0.000158990504, -0.000076345160, ...
```

```
-0.000086038884, -0.000207397887,  0.000452335822,  0.000143609536, ...
-0.000426326002,  0.000022739249, -0.000077028971,  0.000109437010, ...
 0.000503163159, -0.000400473835, -0.000382749070,  0.000291151099, ...
-0.000012161800,  0.000323835672,  0.000088649247, -0.000774652256, ...
 0.000145648260,  0.000467739505, -0.000080985809,  0.000244152034, ...
-0.000479622050, -0.000509464746,  0.000828988281,  0.000180599990, ...
-0.000292953384,  0.000014956067, -0.000677404552,  0.000335717600, ...
 0.000970443325, -0.000576311941, -0.000325873837, -0.000052388245, ...
-0.000303245387,  0.001069921512,  0.000167657619, -0.001182030229, ...
 0.000158324623,  0.000088528999,  0.000269223481,  0.000955378153, ...
-0.001097703840, -0.000822697082,  0.000956012790,  0.000073060567, ...
 0.000477728662, -0.000014682912, -0.001681103433,  0.000598828660, ...
 0.001220846125, -0.000415350313,  0.000226819785, -0.000995399707, ...
-0.000821376128,  0.002011891804,  0.000221179920, -0.001024157901, ...
 0.000019981883, -0.001092279214,  0.000947864329,  0.001927345428, ...
-0.001626655970, -0.000818722074,  0.000286464723, -0.000311781241, ...
 0.002088481356, -0.000057364181, -0.002667194802,  0.000679818328, ...
 0.000642939327,  0.000433504005,  0.001489401469, -0.002489129321, ...
-0.001393945169,  0.002490125792,  0.000147918304,  0.000400273138, ...
-0.000341599756, -0.003110668101,  0.001765115869,  0.002592858165, ...
-0.001421579178, -0.000020733734, -0.001692847084, -0.001027079861, ...
 0.004218162369, -0.000087089567, -0.002675389246,  0.000255453314, ...
-0.001417978704,  0.002143317472,  0.003312427574, -0.003932530802, ...
-0.001568142489,  0.001420158804, -0.000186277188,  0.003577490038, ...
-0.000893678505, -0.005331272168,  0.002189310348,  0.001928673924, ...
 0.000335233479,  0.001990922832, -0.005051516167, -0.001919733998, ...
 0.005741697967, -0.000075708622, -0.000351537519, -0.000977590569, ...
-0.005264439891,  0.004440765253,  0.004937878811, -0.004060464754, ...
-0.000627352283, -0.002432662289, -0.000898037335,  0.008325472917, ...
-0.001444328528, -0.006231467990,  0.001358817120, -0.001405661833, ...
 0.004381690294,  0.005342739239, -0.009073023135, -0.002529165721, ...
 0.004614325789, -0.000015944569,  0.005858413528, -0.003204388672, ...
-0.010557126060,  0.006446445303,  0.004827569264, -0.000855351079, ...
 0.002440692451, -0.010399991588, -0.002063264984,  0.013596114575, ...
-0.001623396236, -0.003074055783, -0.002074581181, -0.009388956865, ...
 0.011599299771,  0.009934250972, -0.012126719485, -0.001901334082, ...
-0.003005749719,  0.000066580740,  0.018918858005, -0.006808261837, ...
-0.016649742979,  0.006401117119, -0.000447227938,  0.010471622183, ...
 0.010345118848, -0.026778344016, -0.004019801453,  0.017767910595, ...
-0.000568212557,  0.012696374655, -0.013150834662, -0.031170881014, ...
 0.028120772493,  0.017739615313, -0.010212181979,  0.004206187305, ...
-0.039328008739,  0.000125788500,  0.068164616995, -0.018850625782, ...
-0.029191308541, -0.008185528095, -0.053154143451,  0.116687783855, ...
 0.095961593836, -0.255252526630, -0.047822580790,  0.319358331526 ]';
```

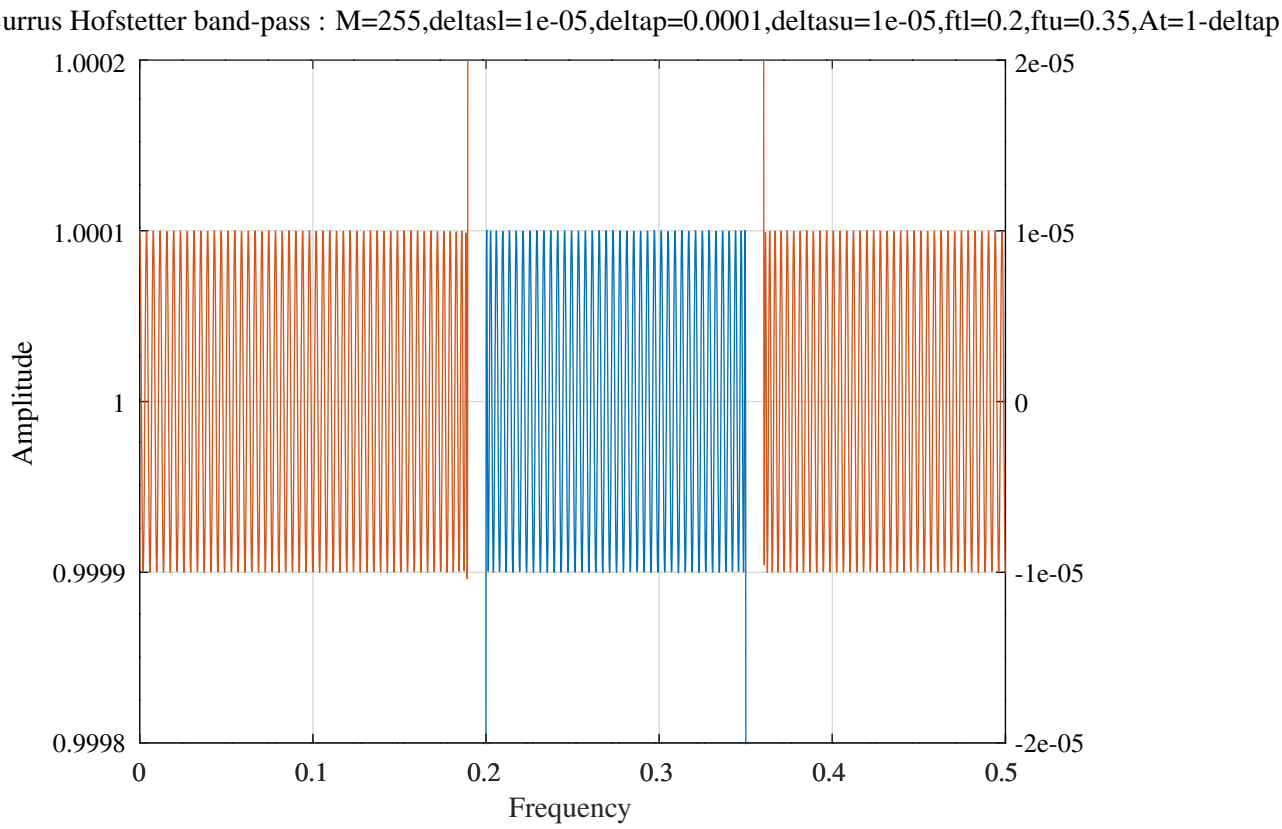



Figure N.34: Response of a mini-max FIR band-pass filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*

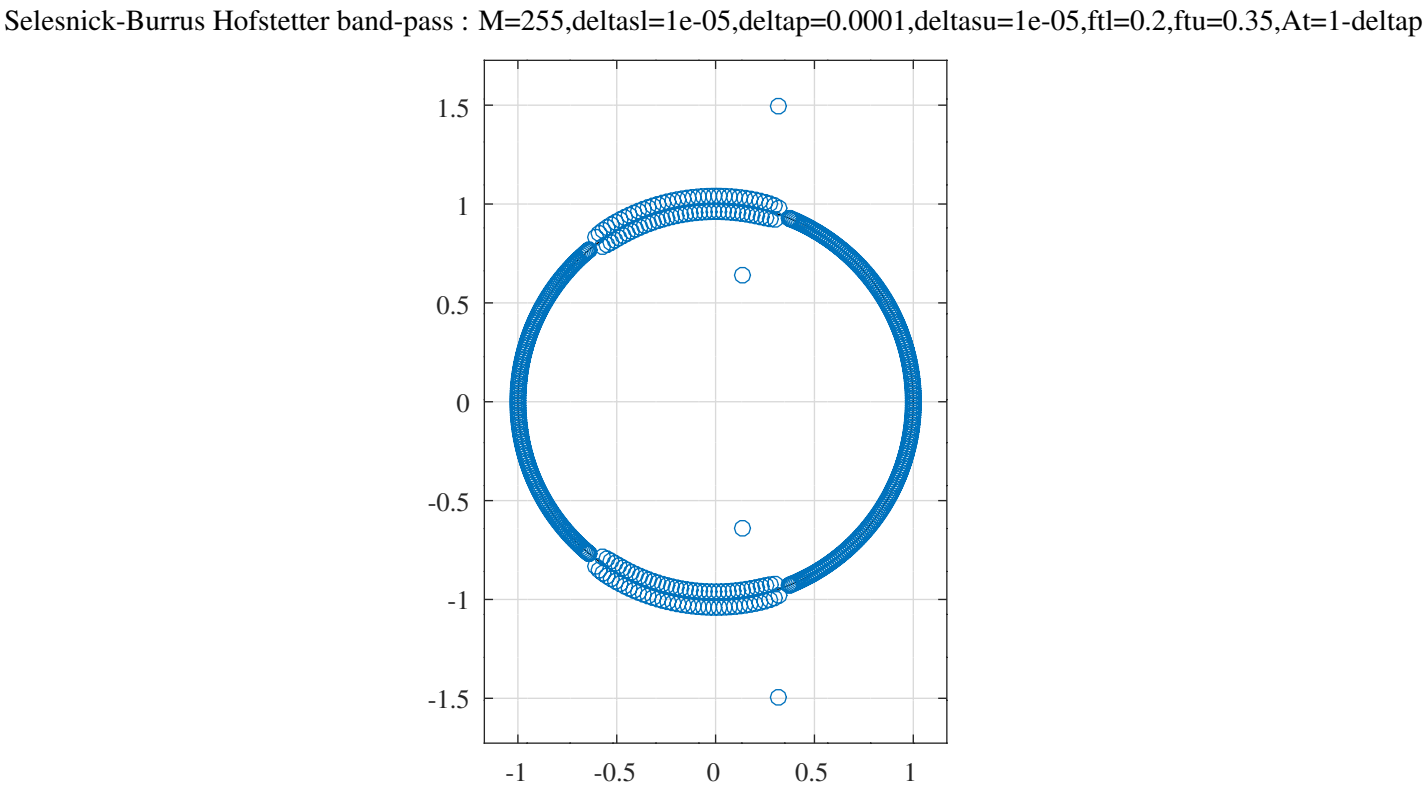



Figure N.35: Zeros of a mini-max FIR band-pass filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*

### N.6.3 *Parks-McClellan* algorithm for mini-max FIR filter approximation

*Parks* and *McClellan* [206, 78][143, Section 7.4.3] modify *Hofstetter*'s algorithm by substituting non-extremal amplitude constraints at the filter band edges. This allows the designer to specify $\omega_p$, $\omega_s$ and the ratio $\delta_p/\delta_s$ but does not allow direct specification of $\delta_p$ and $\delta_s$. *Parks* and *McClellan* [206] and *Oppenheim* and *Schafer* [143, Section 7.4.3] describe the algorithm shown in Algorithm N.2[g].

**Design of a low-pass FIR filter with the *Parks-McClellan* algorithm** The Octave *signal* package includes *remez.cc*, an *oct-file* implementation of the *Parks-McClellan* algorithm. Alternatively, the Octave script *mcclellanFIRsymmetric_lowpass_test.m* calls the Octave function *mcclellanFIRsymmetric* to design the *Parks-McClellan* low-pass filter example [206, Table I]. The filter specification is:

```
M=14 % Filter order is 2*M
fap=0.17265 % Amplitude pass band edge
fas=0.26265 % Amplitude stop band edge
K=5 % Stop band weight
nplot=10000 % Number of frequency grid points in [0,0.5]
tol=1e-12 % Tolerance on convergence
```

Figure N.36 shows the pass-band and stop-band amplitude responses of the filter. Figure N.37 shows the zeros of the filter. The stop-band ripple found is $\rho$ =0.00167218 and the distinct filter coefficients are:

```
hM = [  -0.0020632913,  -0.0059266754,  -0.0016831407,   0.0086376134, ...
         0.0092367643,  -0.0095748911,  -0.0221622199,   0.0023744709, ...
         0.0391202673,   0.0202379100,  -0.0564060230,  -0.0755221777, ...
         0.0694057740,   0.3072654865,   0.4257593871 ]';
```

[g]None of the references justify the calculation of $\rho$ (eg: [143, Equation 7.101]). The following is my unsatisfactory attempt. Suppose that the alternations of $A(x_k)$ mean that $\sum_{k=1}^{M+2} |A(x_k) - A_d(x_k)|$ may be approximated by $\sum_{k=1}^{M+2} (-1)^{k+1} A(x_k)/2$ and that, since $|x - x_k| \leq 2$, $(x - x_k)$ is replaced with 2. For the minimum possible approximation error:

$$\sum_{k=1}^{M+2} b_k \left[ A_d(x_k) - \frac{(-1)^{k+1}\rho}{W(x_k)} \right] = 0$$

The result follows. In his MATLAB scripts [189], *Selesnick* avoids this difficulty by directly calculating $\rho$ and the coefficients of $A(x)$ by left-division.

**Algorithm N.2** Parks-McClellan FIR filter design [143, Section 7.4.3] [206, 78]

1. The optimum filter satisfies the equations:

$$W(\omega_k)\left[A_d(\omega_k) - A(\omega_k)\right] = (-1)^{k+1}\rho\,, \quad k = 1, 2, \ldots, (M+2) \tag{N.15}$$

where

$$A(e^{\imath\omega}) = \sum_{k=0}^{M} a_k (\cos\omega)^k$$

For a lowpass filter, the approximation polynomial has values $A(\omega_k) = 1 \pm K\rho$ if $W(\omega) = \frac{1}{K}$ in $0 \le \omega_k \le \omega_p$ and $A(\omega_k) = \pm\rho$ if $W(\omega) = 1$ in $\omega_s \le \omega_k \le \pi$ [143, Equation 7.85].

2. Begin by guessing a set of alternation frequencies, $\omega_1, \omega_2, \ldots, \omega_{M+2}$. This set includes the pass-band and stop-band frequencies, $\omega_p$ and $\omega_s$. If $\omega_k = \omega_p$ then $\omega_{k+1} = \omega_s$.

3. The required values at the extremal frequencies are given by [143, Equation 7.101]:

$$\rho = \frac{\sum_{k=1}^{M+2} b_k A_d(x_k)}{\sum_{k=1}^{M+2} \frac{b_k(-1)^{k+1}}{W(x_k)}}$$

where [143, Equation 7.102]

$$b_k = \prod_{l=1, l\neq k}^{M+2} \frac{1}{(x_k - x_l)}$$

and $x_k = \cos\omega_k$.

4. $A(\omega)$ is an $M$-th order polynomial in $x = \cos\omega$ so we can interpolate through $M+1$ of the $M+2$ known values of $A(x)$. The Lagrange interpolation formula gives [143, Equation 7.103a]

$$A(x) = \frac{\sum_{k=1}^{M+1} \frac{d_k}{(x-x_k)} C_k}{\sum_{k=1}^{M+1} \frac{d_k}{(x-x_k)}}$$

where [143, Equation 7.103b]

$$C_k = A_d(x) - \frac{(-1)^{k+1}\rho}{W(x_k)}$$

and [143, Equation 7.103c]

$$d_k = \prod_{l=1, l\neq k}^{M+1} \frac{1}{(x_k - x_l)}$$

5. Now $A(x)$ and the error, $E(x) = A(x) - A_d(x)$, can be calculated at a dense grid of frequencies. If $|E(x)| \le \rho$ for all $x$ in the pass-band and stop-band, then the optimal filter is found. Otherwise, this procedure is repeated with a new set of extremal frequencies chosen at the $M+2$ greatest values of $|E(x)|$.

6. The impulse response can be approximated with equally spaced samples of the frequency response (calculated by Lagrange interpolation) and the *discrete Fourier transform pair* [168, Section 4.5]:

$$F(l) = \sum_{k=0}^{K-1} f(k) W_K^{-kl}$$
$$f(k) = \frac{1}{K}\sum_{l=0}^{K-1} F(l) W_K^{kl}$$

where, $f(k)$ is an even order, $N = 2M$, odd length, $K = 2M+1$, FIR filter and $W_K = e^{\frac{2\pi\imath}{K}}$.

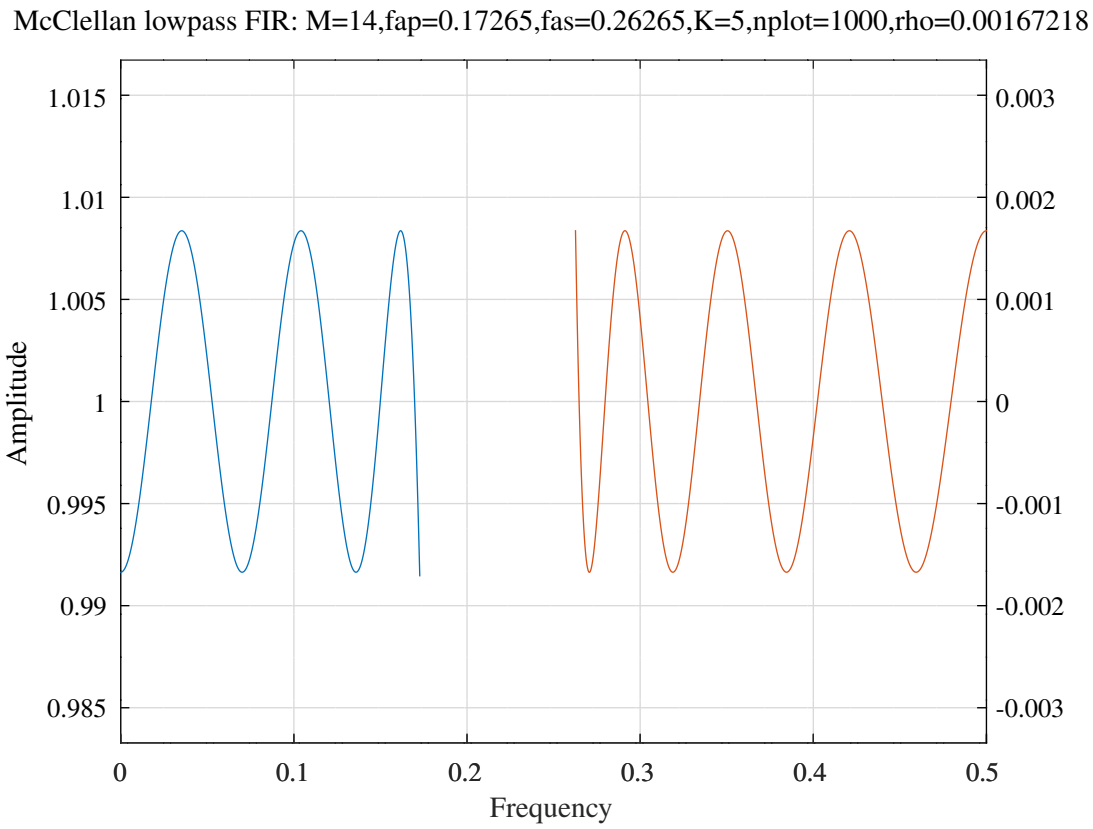



Figure N.36: Response of a mini-max FIR low-pass filter designed with the *Parks-McClellan* algorithm [206, Table I]

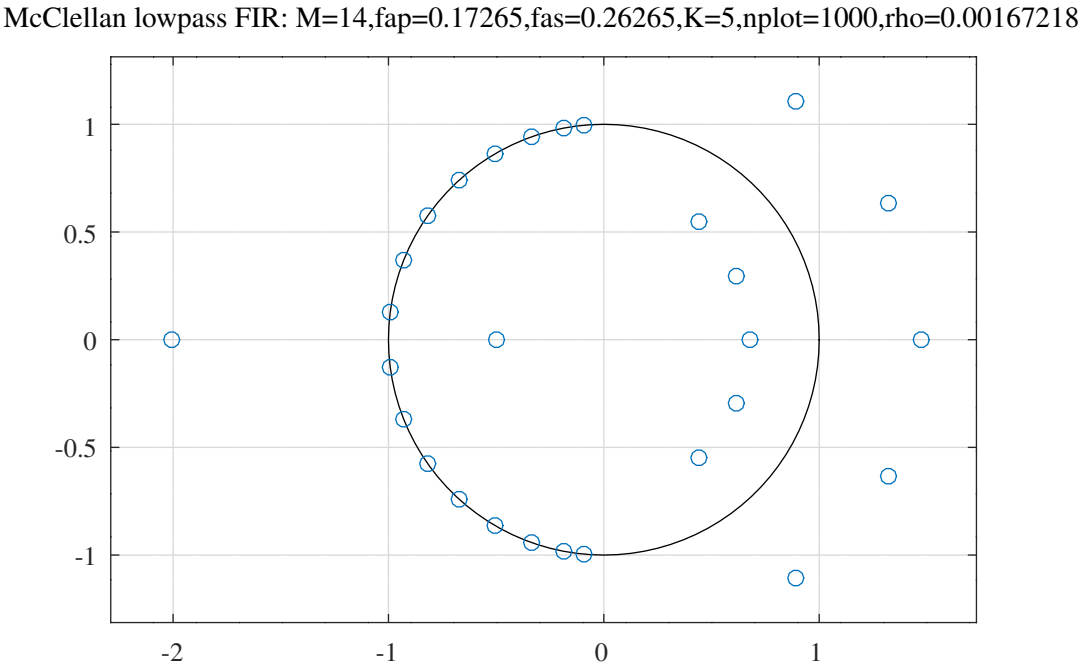



Figure N.37: Zeros of a mini-max FIR low-pass filter designed with the *Parks-McClellan* algorithm [206, Table I]

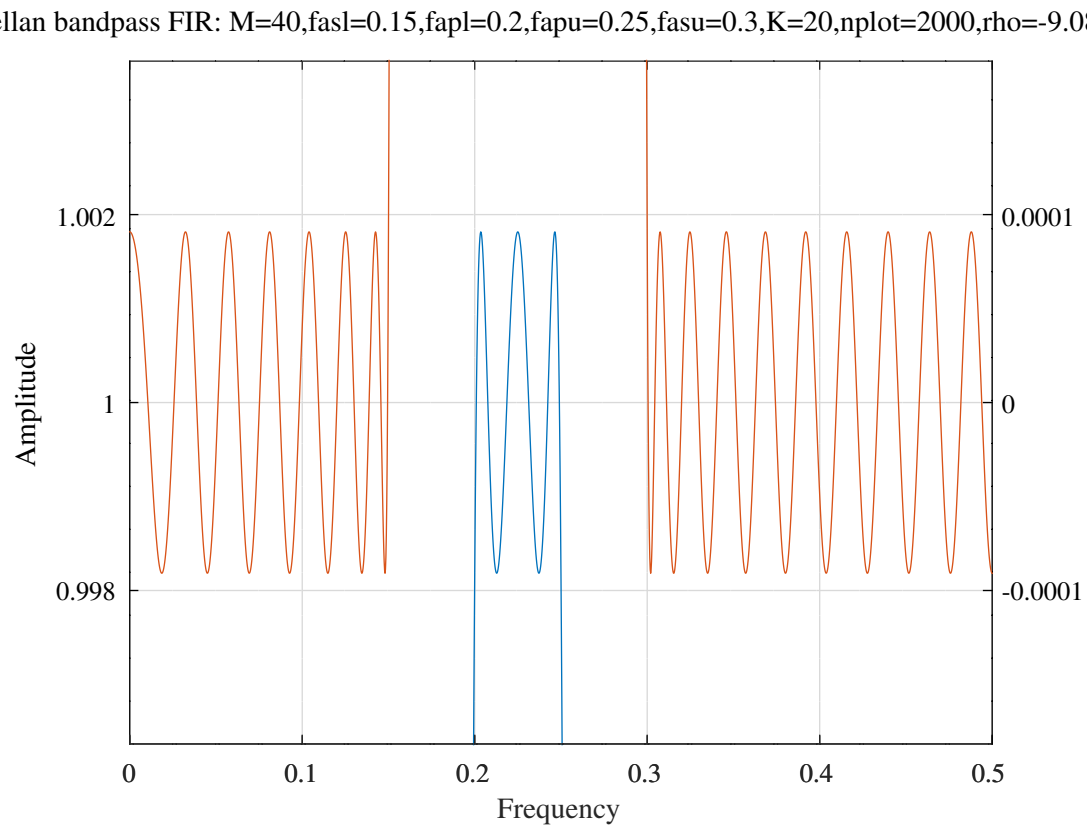



Figure N.38: Response of a mini-max FIR band-pass filter designed with the *Parks-McClellan* algorithm [206, Table I]

**Design of a band-pass FIR filter with the *Parks-McClellan* algorithm** The Octave script *mcclellanFIRsymmetric_bandpass_test.m* designs the band-pass filter with the specification:

```
M=40 % Filter order is 2*M
fasl=0.15 % Amplitude stop band lower edge
fapl=0.2 % Amplitude pass band lower edge
fapu=0.25 % Amplitude pass band upper edge
fasu=0.3 % Amplitude stop band upper edge
K=20 % Stop band weight
nplot=2000 % Number of frequency grid points in [0,0.5]
tol=1e-12 % Tolerance on convergence
```

In this case, the Octave function *mcclellanFIRsymmetric* does a preliminary search for error alternation failures when finding the error extremal frequencies. In my experience, failure of this error extremal amplitude alternation search indicates that the filter specification is unrealistic. Figure N.38 shows the pass-band and stop-band amplitude responses of the filter. Figure N.39 shows the zeros of the filter. The stop-band ripple found is $\rho$ =-0.00009082 and the distinct filter coefficients are:

```
hM = [  -0.0001825792,  -0.0000454355,   0.0004370679,   0.0002807949, ...
        -0.0005520292,  -0.0004924576,   0.0003279243,   0.0002034241, ...
         0.0001098090,   0.0011408993,   0.0000084476,  -0.0033146927, ...
        -0.0014230452,   0.0050412028,   0.0037898840,  -0.0046801365, ...
        -0.0049343476,   0.0020859086,   0.0019942325,   0.0002287784, ...
         0.0059232581,   0.0017378064,  -0.0154830996,  -0.0096436533, ...
         0.0202482490,   0.0192036966,  -0.0156768643,  -0.0206396147, ...
         0.0049234838,   0.0048498311,  -0.0000052363,   0.0281745803, ...
         0.0154584790,  -0.0653441300,  -0.0573965919,   0.0857782989, ...
         0.1158919480,  -0.0734661088,  -0.1675416638,   0.0289464179, ...
         0.1881653480 ]';
```

McClellan bandpass FIR: M=40,fasl=0.15,fapl=0.2,fapu=0.25,fasu=0.3,K=20,nplot=2000,rho=-9.08201e-05

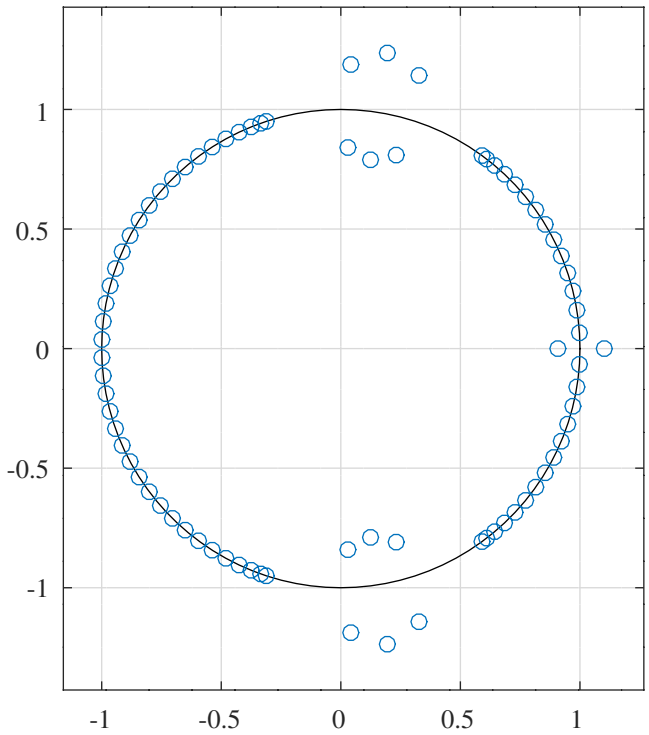



Figure N.39: Zeros of a mini-max FIR band-pass filter designed with the *Parks-McClellan* algorithm [206, Table I]

**Design of a multi-band FIR filter with the *Parks-McClellan* algorithm** The Octave script *mcclellanFIRsymmetric_multiband_test.m* calls the Octave function *mcclellanFIRsymmetric* to design a multi-band filter with the specification:

```
M=30 % Filter order is 2*M
fasu1=0.1 % Amplitude first stop-band upper edge
fapl1=0.15 % Amplitude first pass band lower edge
fapu1=0.2 % Amplitude first pass band upper edge
fasl2=0.25 % Amplitude second stop band lower edge
fasu2=0.3 % Amplitude second stop band upper edge
fapl2=0.35 % Amplitude second pass band lower edge
fapu2=0.4 % Amplitude second pass band upper edge
fasl3=0.45 % Amplitude third stop band lower edge
K1=2 % First stop band weight
K2=4 % Second stop band weight
K3=8 % Third stop band weight
nplot=2000 % Number of frequency grid points in [0,0.5]
tol=1e-12 % Tolerance on convergence
```

Figure N.40 shows the amplitude response of the filter. Figure N.41 shows the zeros of the filter. The pass-band ripple found is $\rho$ =0.00113842 and the distinct filter coefficients are:

```
hM = [   0.0020852534,  -0.0017262027,  -0.0000084962,   0.0003285677, ...
        -0.0014901131,  -0.0085441600,   0.0063716710,   0.0041746799, ...
         0.0017192812,   0.0043711270,  -0.0061979483,  -0.0006439915, ...
        -0.0043622061,  -0.0027483676,  -0.0258056098,   0.0296490648, ...
         0.0335773014,  -0.0180841232,   0.0066018472,  -0.0280836542, ...
        -0.0023059288,  -0.0198877239,   0.0048779015,  -0.0319820481, ...
         0.0791193238,   0.1774151448,  -0.1910626448,  -0.0521172035, ...
        -0.1086275176,  -0.0549671680,   0.4144310402 ]';
```

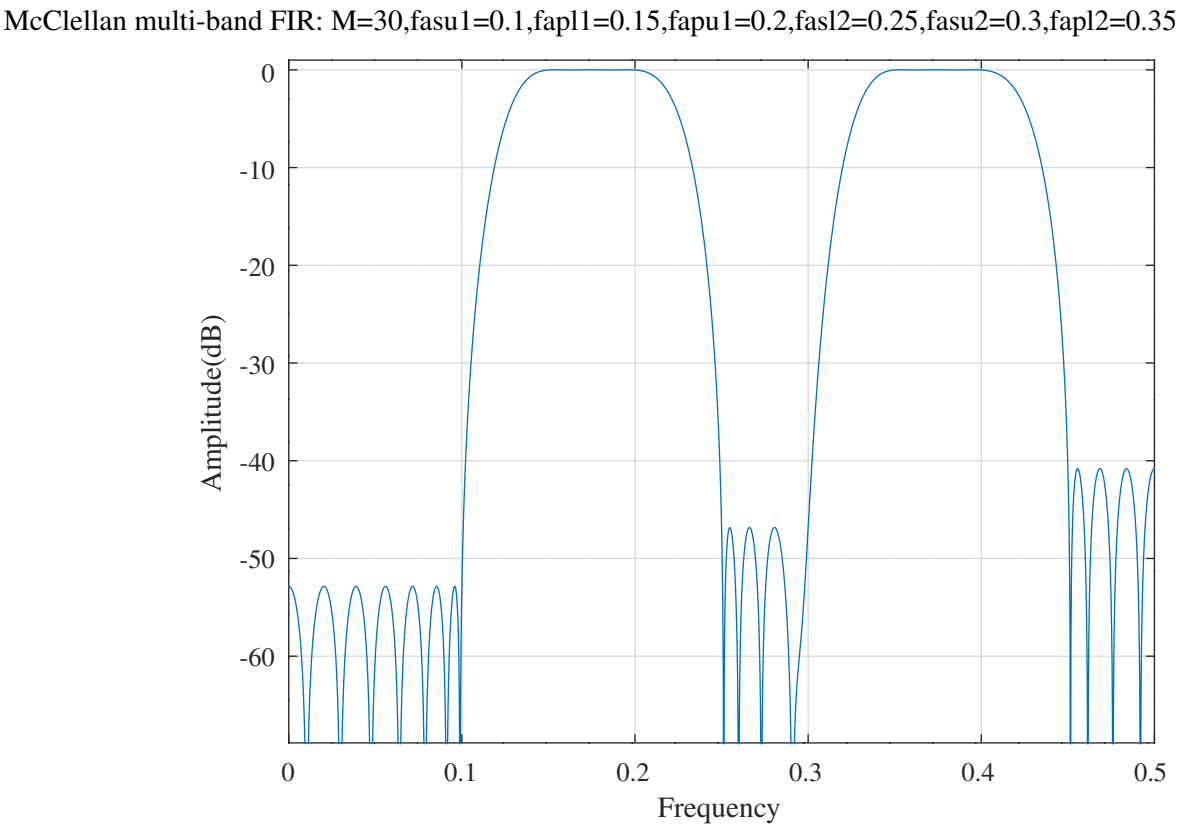



Figure N.40: Response of a mini-max FIR multi-band filter designed with the *Parks-McClellan* algorithm [206, Table I]

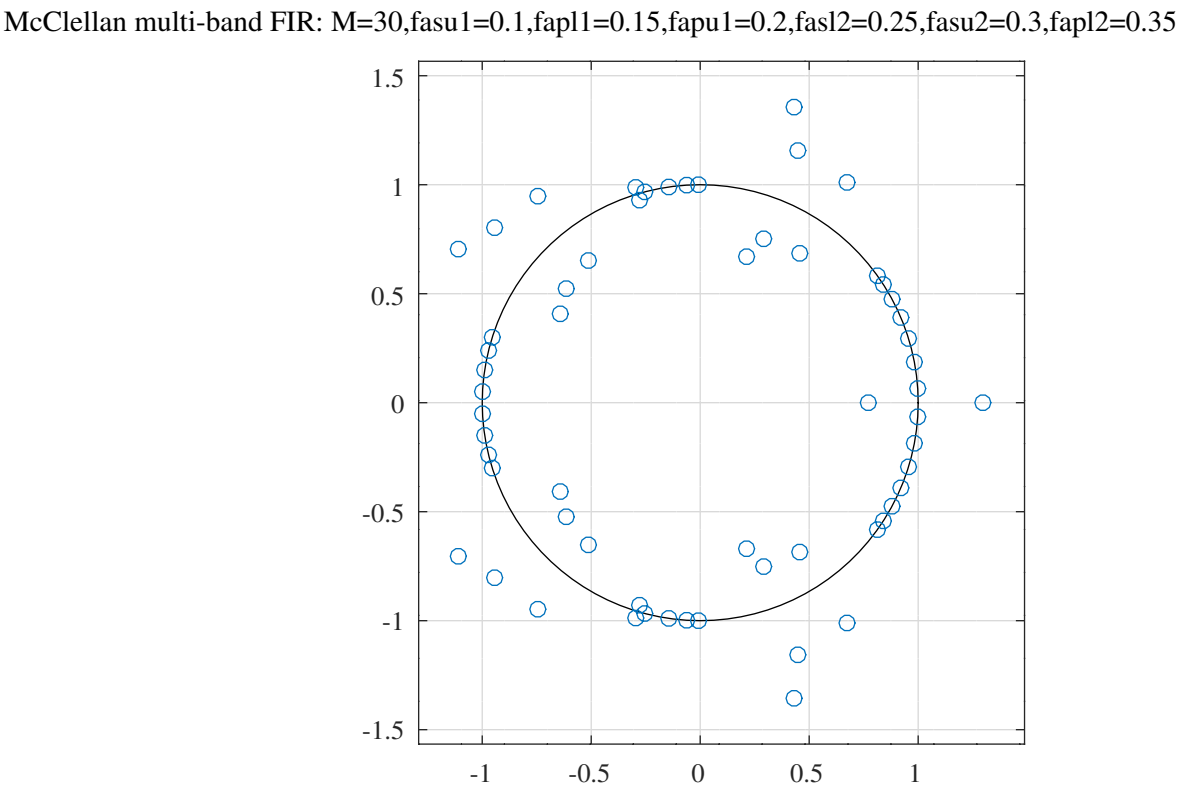



Figure N.41: Zeros of a mini-max FIR multi-band filter designed with the *Parks-McClellan* algorithm [206, Table I]

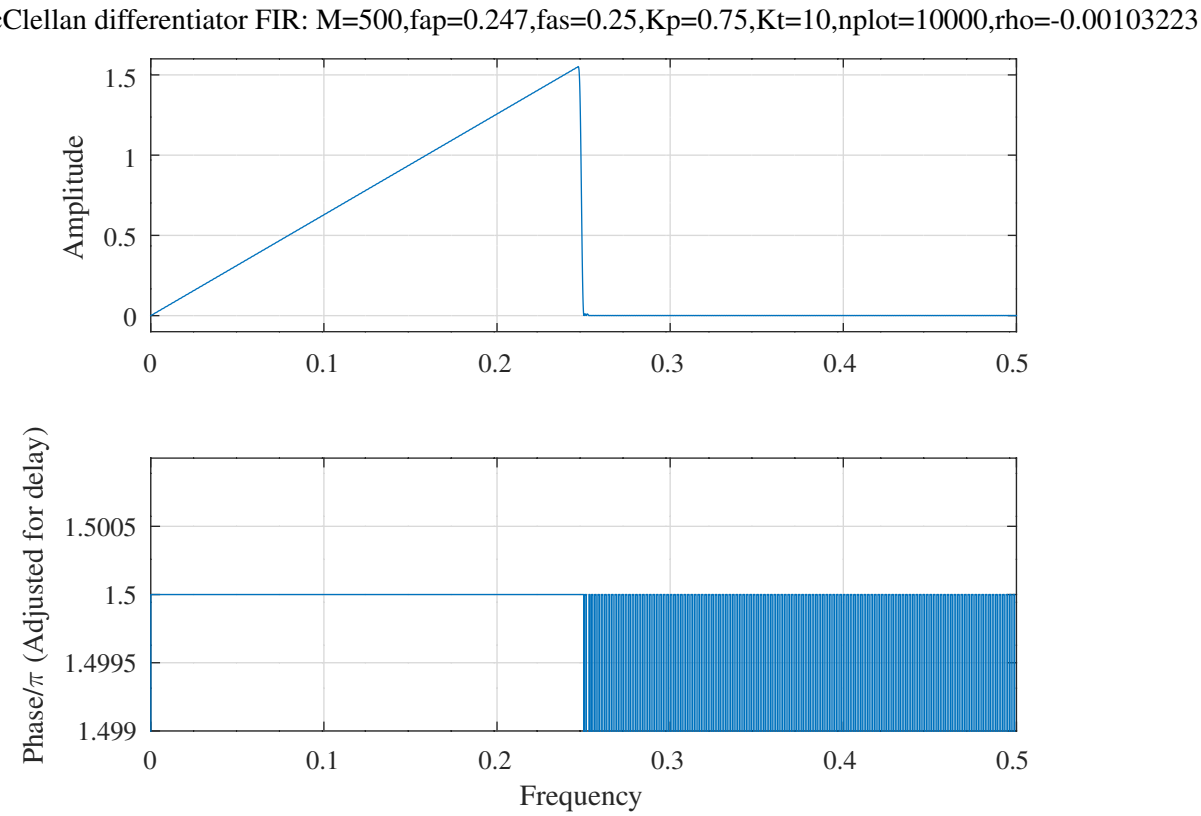



Figure N.42: Amplitude and phase responses of a mini-max FIR differentiator filter designed with the *Parks-McClellan* algorithm.

**Design of a low-pass differentiator FIR filter with the *Parks-McClellan* algorithm** The Octave script *mcclellanFIRdifferentiator_test.m* calls the Octave function *mcclellanFIRdifferentiator* to design a low-pass FIR differentiator filter with the specification:

```
M=500 % Filter order is 2*M
fap=0.247 % Amplitude pass band edge
fas=0.25 % Amplitude stop band edge
Kp=0.75 % Pass band weight
Kt=10 % Transition band weight
nplot=10000 % Number of frequency grid points in [0,0.5]
maxiter=100 % Maximum iterations
tol=1e-10 % Tolerance on convergence
```

The ripple found is $\rho$ =-0.00103223 . Figure N.42 shows the amplitude and phase responses of the filter. Figure N.43 shows the amplitude error responses of the filter.

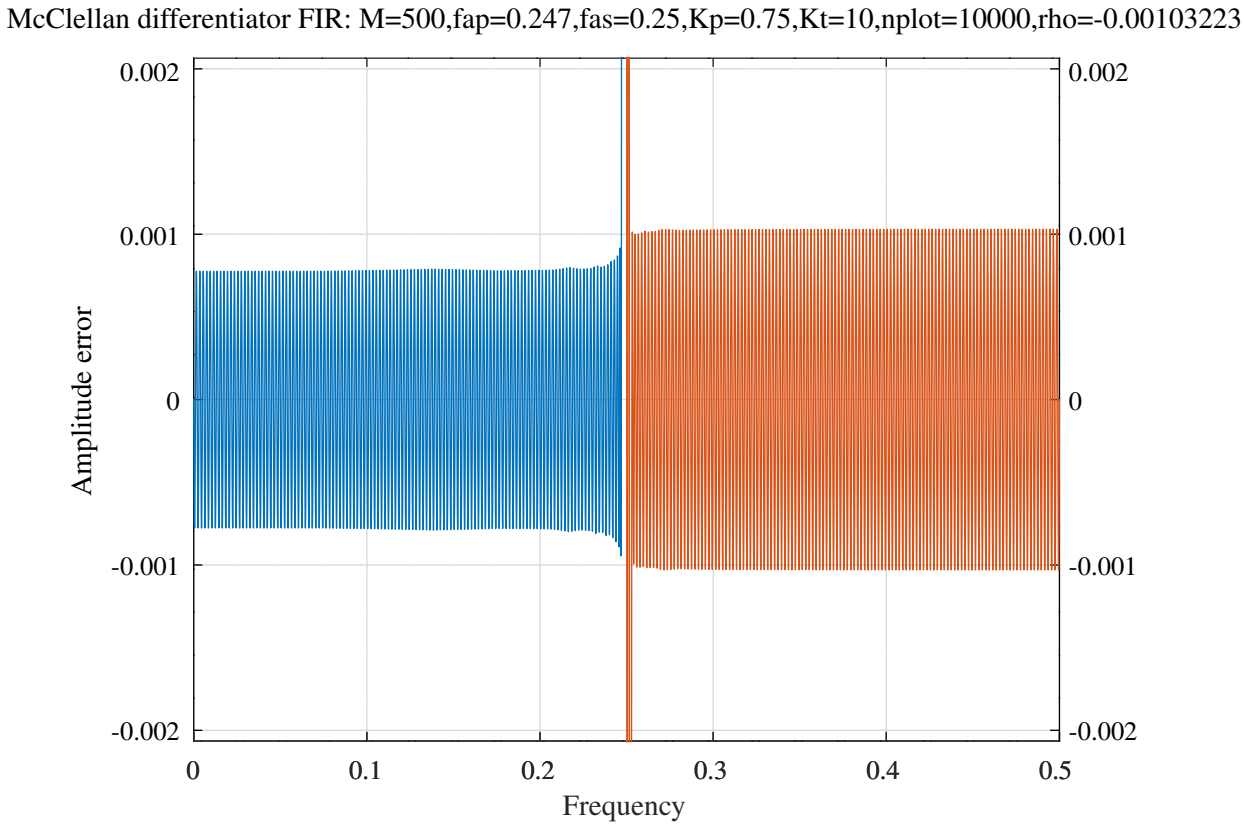



Figure N.43: Amplitude error of a mini-max FIR differentiator filter designed with the *Parks-McClellan* algorithm.

***Selesnick-Burrus* modification to the *Parks-McClellan* algorithm**

*Selesnick* and *Burrus* describe modifications to the *Parks-McClellan* algorithm [74, Section II.A] that allow either the pass-band or stop-band ripple, $\delta_p$ or $\delta_s$, to be fixed and the other minimised. The user specifies $M$, $\omega_p$, $\omega_s$, $K_p$, $K_s$, $\eta_p$ and $\eta_s$ satisfying:

$$\delta_p = K_p\delta + \eta_p$$
$$\delta_s = K_s\delta + \eta_s$$

The parameters $K_p, K_s\eta_p$ and $\eta_s$ must be non-negative and satisfy the inequalities $K_p + \eta_p > 0$ and $K_s + \eta_s > 0$. When $\eta_p = \eta_s = 0$ the linear relationship reduces to the usual *Parks-McClellan* algorithm. Alternatively, if $K_s = \eta_p = 0$, then the stop-band ripple, $\delta_s = \eta_s$ and the pass-band ripple, $\delta_p$, is minimised.

The linear system of equations to be solved at each iteration is, in addition to the affine equations for $\delta_p$ and $\delta_s$:

$$A(\omega_k) = \begin{cases} 1 + (-1)^{k+c}\,\delta_p & \text{for } 1 \le k \le q \\ (-1)^{k+c}\,\delta_s & \text{for } q+1 \le k \le M+2 \end{cases}$$

where $\omega_p = \omega_q$ and $\omega_s = \omega_{q+1}$ and $c$ is chosen to be 0 or 1 so that $A(\omega_q) = 1 - \delta_p$.

If $\delta_p$ or $\delta_s$ are found to be negative then the interpolation step should be repeated. If $\delta_p < 0$:

$$A(\omega_k) = \begin{cases} 1 + (-1)^{k+c}\,\delta_p & \text{for } 1 \le k \le q \\ 0 & \text{for } q+1 \le k \le M+2 \end{cases}$$
$$\delta_s = 0$$

If $\delta_s < 0$:

$$A(\omega_k) = \begin{cases} 1 & \text{for } 1 \le k \le q \\ (-1)^{k+c}\,\delta_s & \text{for } q+1 \le k \le M+2 \end{cases}$$
$$\delta_p = 0$$

The exchange algorithm is described as follows:

> The procedure to update the reference set from one iteration to the next is the multiple exchange of the PM algorithm: Let *S* be the set obtained by appending $\omega_p$ and $\omega_s$ to the set of extrema of $A(\omega)$ in $[0, \pi]$. *S* will have either $M+2$ or $M+3$ frequencies and will include both 0 and $\pi$. If *S* has $M+2$ frequencies, then take the new reference set to be *S*. If *S* has $M+3$ frequencies, then remove either 0 or $\pi$ from *S* according to the following rule: If $\omega = 0$ is a local maximum of $A(\omega)$ then let $\alpha = 1$, otherwise set $\alpha = -1$. If $\omega = \pi$ is a local maximum of $A(\omega)$ then let $\beta = 1$, otherwise set $\beta = -1$. If
>
> $$[A(0) - 1]\,\alpha - \delta_p < A(\pi)\,\beta - \delta_s$$
>
> then remove 0 from *S*, otherwise remove $\pi$ from *S*, and take the new reference set to be the resulting set S. The expressions on each side of the inequality indicate the amount by which the error exceeds its intended value. $\alpha$ and $\beta$ must be chosen appropriately because both the magnitude and the sign of this value is important: negative values appear in the design of filters possessing a scaled extra-ripple. The rule states that the frequency to be retained in S is the one at which the error exceeds its intended value the most.

A MATLAB example, *faffine.m*, of low-pass filter design with this algorithm is available [189]. The Octave script *affineFIRsymmetric_lowpass_test.m* calls *affineFIRsymmetric_lowpass.m* to design a low-pass filter with the following specification:

```
M=27 % Filter order is 2*M
fp=0.2 % Amplitude pass band edge
fs=0.25 % Amplitude stop band edge
kappap=1 % Pass-band kappa
kappas=0 % Stop-band kappa
etap=0 % Pass-band eta
etas=0.001 % Stop-band eta
nplot=2000 % Number of frequency points
maxiter=100 % Maximum iterations
tol=1e-12 % Tolerance on convergence
```

Affine lowpass FIR : M=27,fp=0.2,fs=0.25,kappap=1,kappas=0, etap=0,etas=0.001,nplot=2000,deltap=0.00644777,deltas=0.001

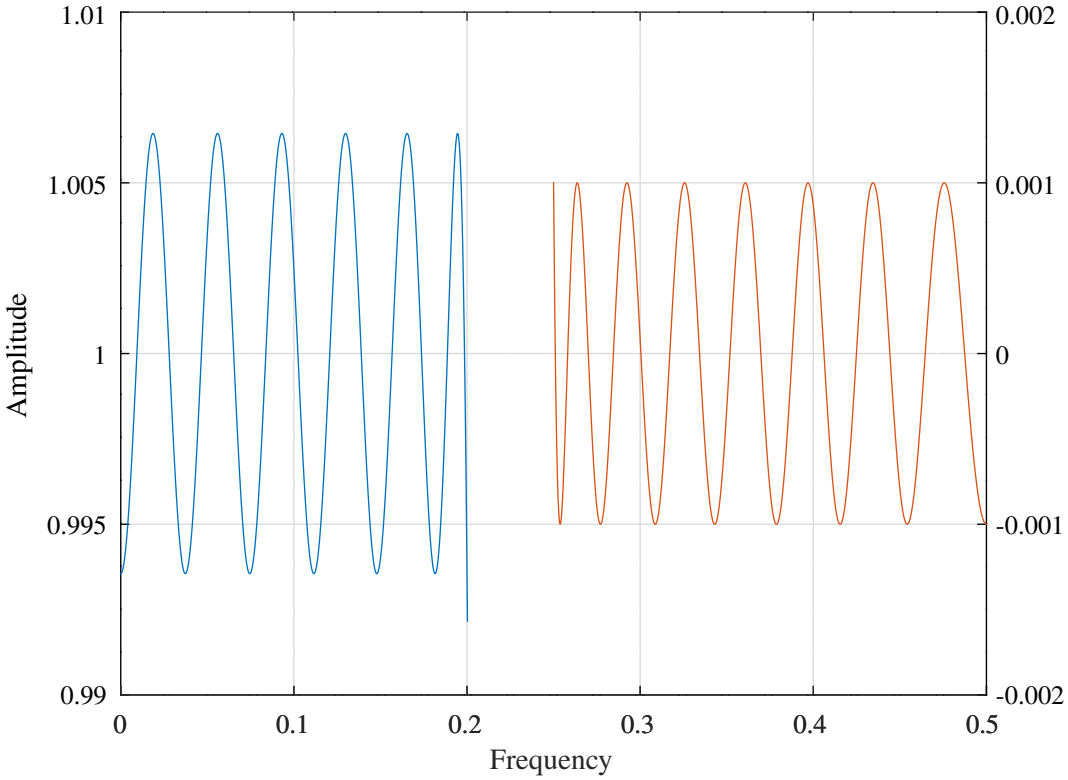



Figure N.44: Response of a mini-max FIR low-pass filter designed with the *Parks-McClellan* algorithm modified by *Selesnick* and *Burrus* [74, Section II.A].

The resulting filter has $\delta_p$ =0.00644777 . Figure N.44 shows the pass-band and stop-band amplitude responses of the filter. The distinct filter coefficients are:

```
hM = [  -0.0015354915,  -0.0027935586,  -0.0005350974,   0.0025612838, ...
         0.0019847095,  -0.0030146978,  -0.0039715920,   0.0023433449, ...
         0.0066374062,  -0.0004379921,  -0.0091369881,  -0.0033095998, ...
         0.0108888765,   0.0088870186,  -0.0107796012,  -0.0162298272, ...
         0.0077051550,   0.0247940105,  -0.0002036318,  -0.0338385150, ...
        -0.0137467925,   0.0423716626,   0.0385605735,  -0.0493950809, ...
        -0.0899799941,   0.0540125856,   0.3127505254,   0.4443748462 ]';
```

## N.7 Design of FIR filters with maximally-linear pass-bands and equi-ripple stop-bands using the *Parks-McClellan* algorithm

*Vaidyanathan* [147] and *Selesnick* and *Burrus* [73] describe the design of FIR filters with flat pass-bands and equi-ripple stop-bands. *Selesnick* and *Burrus* interpret the flat pass-band FIR filter structure proposed by *Vaidyanathan* [147, Figure 4] as [73, Figure 2]:

$$H(z) = z^{-\frac{N-1}{2}} + H_L(z) H_M(z)$$

where:

$$H_L(z) = \left(\frac{1 - z^{-1}}{2}\right)^L$$

The filter length, $N$, is odd and $H_M(z)$ is a high-pass filter with a symmetric impulse response of length $N - L$. The frequency reponse of $H_M(\omega)$ is:

$$H_M(\omega) = e^{-\imath\omega\left(\frac{N-L-1}{2}\right)} A_M(\omega)$$

where $A_M(\omega)$ is a real valued function. When $L$ is chosen to be even:

$$\left(\frac{1 - e^{-\imath\omega}}{2}\right)^L = (-1)^{\frac{L}{2}} \left(e^{-\frac{\imath\omega}{2}}\right)^L \left(\sin\frac{\omega}{2}\right)^L$$

Therefore the frequency response is:

$$H(\omega) = e^{-\imath\omega\left(\frac{N-1}{2}\right)} A(\omega)$$

where the amplitude response is:

$$A(\omega) = 1 + (-1)^{\frac{L}{2}} \left(\sin\frac{\omega}{2}\right)^L A_M(\omega)$$

Let $M = (N - L - 1)/2$ then:

$$A_M(\omega) = h_M(M) + 2\sum_{k=0}^{M-1} h_M(k) \cos[\omega(M - k)]$$

where $h_M$ are the coefficients of $H_M$.

### N.7.1 Design of FIR low-pass filters with maximally-flat pass-bands and equi-ripple stop-bands

*Selesnick* and *Burrus* describe exchange algorithms for the cases in which the maximally-flat pass-band, low-pass, FIR filter design:

1. specifies $N$, $L$, $\omega_s$ and minimises $\delta_s$
2. specifies $N$, $L$, $\delta_s$ and minimises $\omega_s$

#### Maximally-flat pass-band and fixed $\omega_s$

In the first case, $A_M(\omega)$ is found by minimising [73, Section II.A]:

$$\|[A_M(\omega) - D(\omega)] W(\omega)\|_\infty$$

where:

$$D(\omega) = -\frac{(-1)^{\frac{L}{2}}}{\left(\sin\frac{\omega}{2}\right)^L}$$

$$W(\omega) = \begin{cases} 0 & \text{for } \omega < \omega_s \\ (-1)^{\frac{L}{2}} \left(\sin\frac{\omega}{2}\right)^L & \text{for } \omega \geq \omega_s \end{cases}$$

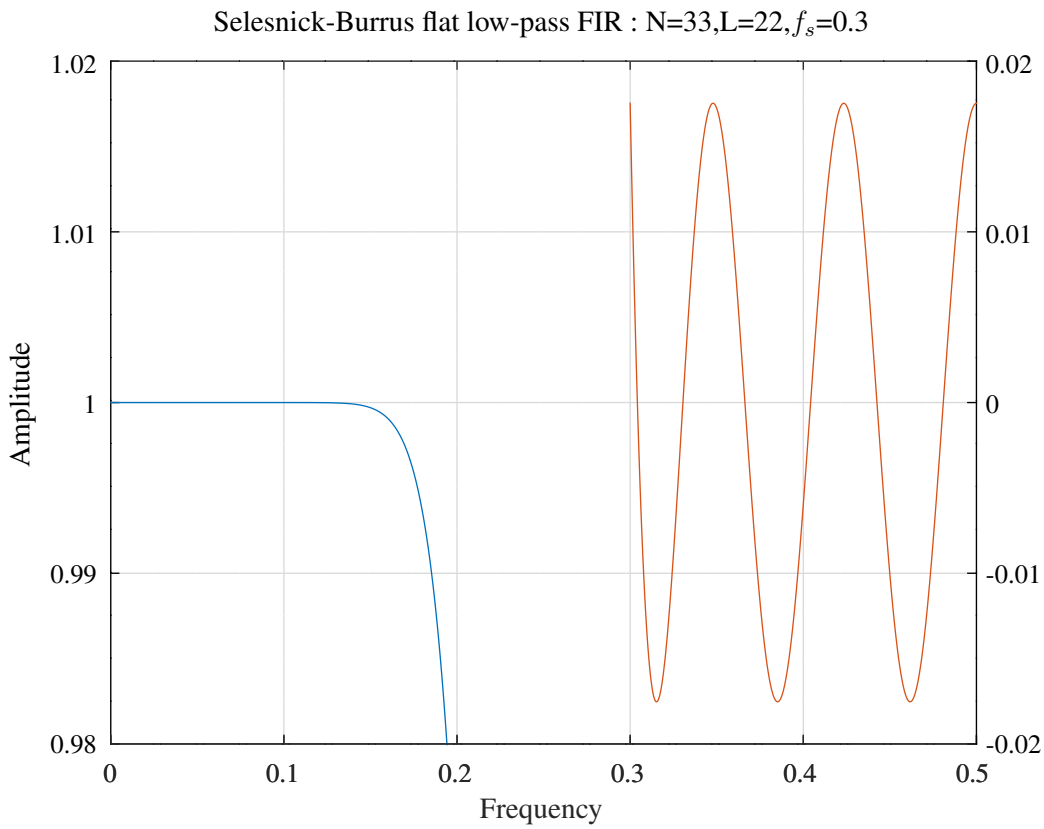



Figure N.45: Response of a mini-max FIR maximally-flat pass-band low-pass filter with fixed $\omega_s$ designed with the *Parks-McClellan* algorithm [73, Section II.A].

On each iteration, a reference set of stopband frequencies is updated and the filter $H_M$ is found such that $A(\omega)$ alternately interpolates $\delta_s$ and $-\delta_s$, over the reference set frequencies. The size of the reference set is $q = (N - L + 3)/2$.

The Octave script *mcclellanFIRsymmetric_flat_lowpass_test.m* calls the Octave function *mcclellanFIRsymmetric* to design a maximally-flat FIR low-pass filter having fixed $\omega_s$ with the *Parks-McClellan* algorithm (Algorithm N.2). The specification of the filter is:

```
N=33 % Filter length
L=22 % Filter flat-ness
fs=0.3 % Amplitude stop band frequency
nplot=4000 % Number of frequency points
tol=1e-10 % Tolerance on convergence
```

Figure N.45 shows the pass-band and stop-band amplitude responses of the fixed $\omega_s$ filter. The distinct filter coefficients of $H_M(z)$ are:

```
hM = [    56.17896774,   466.93894436,  1839.26934699,  4492.82391996, ...
        7488.61684242,  8849.58705438 ]';
```

The distinct filter coefficients of the overall filter, $H(z)$, are:

```
hA = [  0.000013394110, -0.000183343493,  0.001083363085, -0.003486589406, ...
        0.006051274414, -0.003407242506, -0.006369164439,  0.009523748743, ...
        0.008065137563, -0.021430848277, -0.009486047978,  0.043915909948, ...
        0.010310734167, -0.093052907047, -0.010646959048,  0.313738455002, ...
        0.510722170324 ]';
```

**Maximally-flat pass-band and fixed $\delta_s$**

In the second case [73, Section II.B]:

> Like the Remez algorithm, this approach employs a set of stopband reference frequencies. On each iteration: 1) an interpolation problem is solved and 2) the reference set is updated. The reference set here, however, does not contain the stopband edge (indeed, it is not specified). Therefore the reference set contains $(N - L + 1)/2$ stopband frequencies. ... At each iteration, the local extremal frequencies of $A(\omega)$ in $(0, \pi]$ are found and are taken to be the reference set frequencies for the next iteration.

The Octave script *selesnickFIRsymmetric_flat_lowpass_test.m* calls the Octave function *selesnickFIRsymmetric_flat_lowpass* to design a maximally-flat FIR low-pass filter having fixed $\delta_s$. The specification of the filter is:

```
N=33 % Filter length
L=22 % Filter flat-ness
deltas=0.01 % Amplitude stop band peak ripple
```

Figure N.46 shows the pass-band and stop-band amplitude responses of the fixed $\delta_s$ filter. The Octave function *selesnickFIRsymmetric_flat_lowpass* uses Lagrange interpolation to find the filter amplitude response and removes spurious peaks in the pass-band amplitude response that are due to numerical errors. The distinct filter coefficients of $H_M(z)$ are:

```
hM = [    43.91080519,   372.91677570,  1494.48125206,  3697.49748358, ...
        6212.16961201,  7361.28481996 ]';
```

The distinct filter coefficients of the overall filter, $H(z)$, are:

```
hA = [  0.000010469152, -0.000141411051,  0.000818659827, -0.002541531294, ...
        0.004055046944, -0.001227041686, -0.006392990415,  0.006672187289, ...
        0.009840325210, -0.018250303773, -0.013299129454,  0.040971335274, ...
        0.016130924455, -0.090969568900, -0.017947413469,  0.312986334142, ...
        0.518568215500 ]';
```

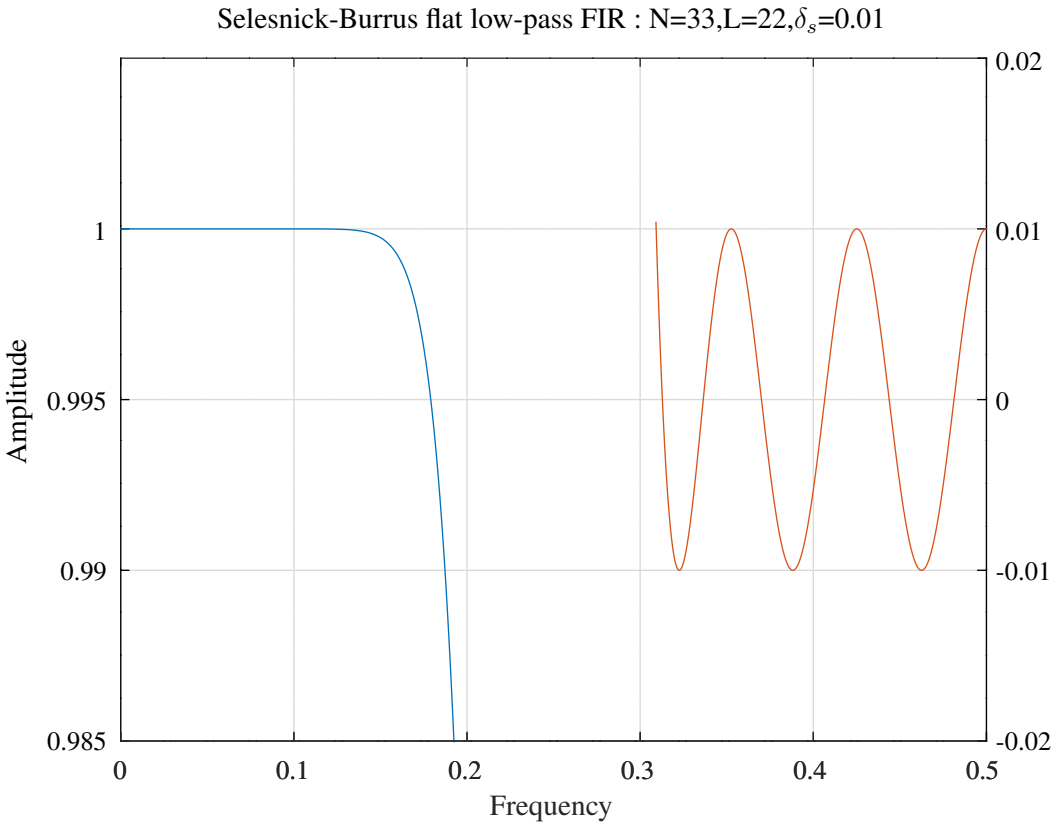



Figure N.46: Response of a mini-max FIR maximally-flat pass-band low-pass filter with fixed $\delta_s$ [73, Section II.B].

### N.7.2 Design of FIR band-pass filters with maximally-flat pass-bands and equi-ripple stop-bands

*Selesnick* and *Burrus* describe the design of maximally-flat pass-band, band-pass, FIR filters with centre frequency, $\omega_p$, pass-band width $2\omega_t$ and upper and lower stop-band ripple $\delta_{su}$ and $\delta_{sl}$, respectively. For maximally-flat band-pass FIR filters:

$$H_L(z) = \left[\frac{1 - 2z^{-1}\cos\omega_p + z^{-2}}{4}\right]^{\frac{L}{2}}$$

where $L$ is a multiple of 4, $N$ is odd. The amplitude response is:

$$A(\omega) = 1 + (-1)^{\frac{L}{2}}\left[\frac{\cos\omega_p - \cos\omega}{2}\right]^{\frac{L}{2}} A_M(\omega)$$

where $A_M$ is the amplitude response of a symmetric FIR filter, $H_M$, of length $N - L$. *Selesnick* and *Burrus* describe exchange algorithms for the two cases:

1. specifies $N$, $L$, $\omega_p$, $\omega_t$, $K = \delta_{su}/\delta_{sl}$ and minimises $\delta_{sl}$
2. specifies $N$, $L$, $\delta_{su}$ and $\delta_{sl}$ and maximises $\omega_t$

The exchange algorithms described in this section place all extremal frequencies in the stop-bands. The *Remez* exchange algorithm requires that the error alternate in sign over the set of reference frequencies. Consequently, at each iteration, only one of the stop-band edges can be included in the set of reference frequencies.

#### Maximally-flat pass-band and fixed $\omega_p$ and $\omega_t$

For convenience, set $\omega_a = \omega_p - \omega_t$ and $\omega_b = \omega_p + \omega_t$. In this case [73, Section III]:

> The reference set is updated by the following procedure: First compute the set of all local extremal frequencies of $A(\omega)$ in $[0, \pi]$. Calling this set $R$, remove $\omega_p$ from $R$. $R$ will then contain either $(N - L + 1)/2$ or $(N - L + 3)/2$ frequencies. If $R$ contains $(N - L + 3)/2$ frequencies, then remove either 0 or $\pi$ as follows: if $|A(\pi)| < K|A(0)|$ then remove $\pi$, otherwise remove 0. Next append either $\omega_a$ or $\omega_b$ to $R$: if $|A(\omega_b)| < K|A(\omega_a)|$ then append $\omega_a$, otherwise append $\omega_b$. $R$ is the new reference set and has size $(N - L + 3)/2$. On each iteration, the filter $H_M$ is found such that $A(\omega)$ interpolates $\delta_{sl}(-1)^k$ over the reference set frequencies in the first stop-band and $K\delta_{sl}(-1)^k$ in the second stop-band.

The Octave script *mcclellanFIRsymmetric_flat_bandpass_test.m* calls the Octave function *mcclellanFIRsymmetric* to design a maximally-flat FIR band-pass filter having fixed $\omega_p$ and $\omega_t$ and $K = \delta_{su}/\delta_{sl}$. The specification of the filter is:

```
N=55 % Filter length
L=8 % Filter flat-ness
fp=0.2 % Amplitude pass-band centre frequency
ft=0.05 % Amplitude pass-band half-width frequency
K=2 % Amplitude stop-band ripple ratio
nplot=4000 % Number of frequency points
tol=1e-10 % Tolerance on convergence
```

Figure N.47 shows the detailed pass-band and stop-band amplitude responses of the filter. The distinct filter coefficients of $H_M(z)$ are:

```
hM = [     1.70555584,     3.17821682,    -5.09701343,   -19.07316638, ...
          -4.76049523,    47.75504982,    57.98434573,   -53.19114156, ...
        -167.68691503,   -41.82256908,   272.22952127,   296.56112747, ...
        -214.69078694,  -642.37351794,  -167.60342750,   821.69750188, ...
         853.07954884,  -502.92454392, -1496.85361123,  -422.07303994, ...
        1551.08341809,  1569.37603234,  -695.67269651, -2152.30662247 ]';
```

The distinct filter coefficients of the overall filter, $H(z)$, are:

```
hA = [  0.006662327500, -0.004055269899, -0.008683546807, -0.002873321599, ...
        0.008005029674,  0.007713691662, -0.004255694702, -0.009847048614, ...
       -0.001812617050,  0.004798782809,  0.001027331693,  0.000266976513, ...
        0.008495033147,  0.004817900540, -0.018885705210, -0.025569566382, ...
        0.013381986383,  0.051133914935,  0.017876953299, -0.058493690708, ...
       -0.064941412550,  0.029188820612,  0.098830498685,  0.031751715332, ...
       -0.091124032213, -0.093779841405,  0.037083123198,  0.119938222140 ]';
```

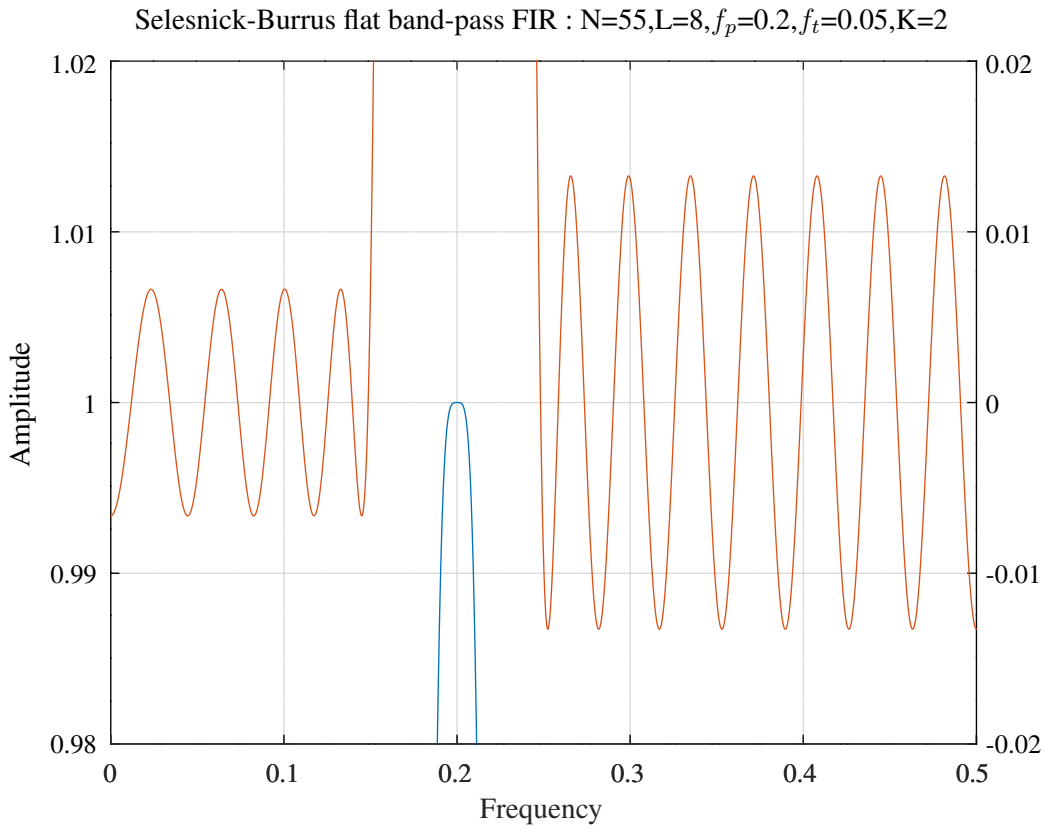



Figure N.47: Pass-band and stop-band responses of a mini-max FIR maximally-flat pass-band band-pass filter with fixed $\omega_p$ and $\omega_t$ [73, Section III].

**Maximally-flat pass-band and fixed $\delta_{su}$ and $\delta_{sl}$**

In the second case [73, Section III]:

> A similar algorithm is used for approach (2) in which $\delta_{sl}$ and $\delta_{su}$ are specified and the stop-band edges are left variable. The reference set is updated in the same manner, except no stopband edge is appended to $R$. Let $\omega_1, \ldots \omega_q$ denote the reference set frequencies ordered in increasing order. On each iteration, the filter $H_M$ is found such that:
>
> $$A(\omega) = \delta_{sl}(-1)^{k+c} \qquad \text{for } \omega_k < \omega_p$$
> $$A(\omega) = \delta_{su}(-1)^{k+c+1} \qquad \text{for } \omega_k > \omega_p$$
>
> where $c$ equals 0 or 1, whichever gives $A(\omega) = -\delta_{sl}$ at the highest reference frequency less than $\omega_p$.

The Octave script *selesnickFIRsymmetric_flat_bandpass_test.m* calls the Octave function *selesnickFIRsymmetric_flat_bandpass* to design a maximally-flat FIR low-pass filter having fixed $\delta_{su}$ and $\delta_{sl}$. The specification of the filter is:

```
N=55 % Filter length
L=16 % Filter maximal flat-ness
deltasl=0.01 % Amplitude lower stop-band peak ripple
deltasu=0.02 % Amplitude upper stop-band peak ripple
fp=0.2 % Amplitude pass-band centre frequency
ft=0.04 % Initial amplitude pass-band half-width
nplot=4000 % Number of frequency
tol=1e-09 % Tolerance
```

Figure N.48 shows the pass-band and stop-band amplitude responses of the fixed $\delta_{su}$ and $\delta_{sl}$ filter. The distinct filter coefficients of $H_M(z)$ are:

```
hM = [     346.6174393,    1384.1759764,     475.0397672,   -6894.9867351, ...
        -12695.2181907,    7309.6402723,   47779.7574138,   34867.3658180, ...
        -77730.9674673, -154784.1927737,    6678.4653510,  292558.9876402, ...
        247910.0879215, -256448.0019794, -582855.1939345, -107334.9576096, ...
        681059.0642255,  644221.8804970, -312203.9143812, -912262.6344293 ]';
```

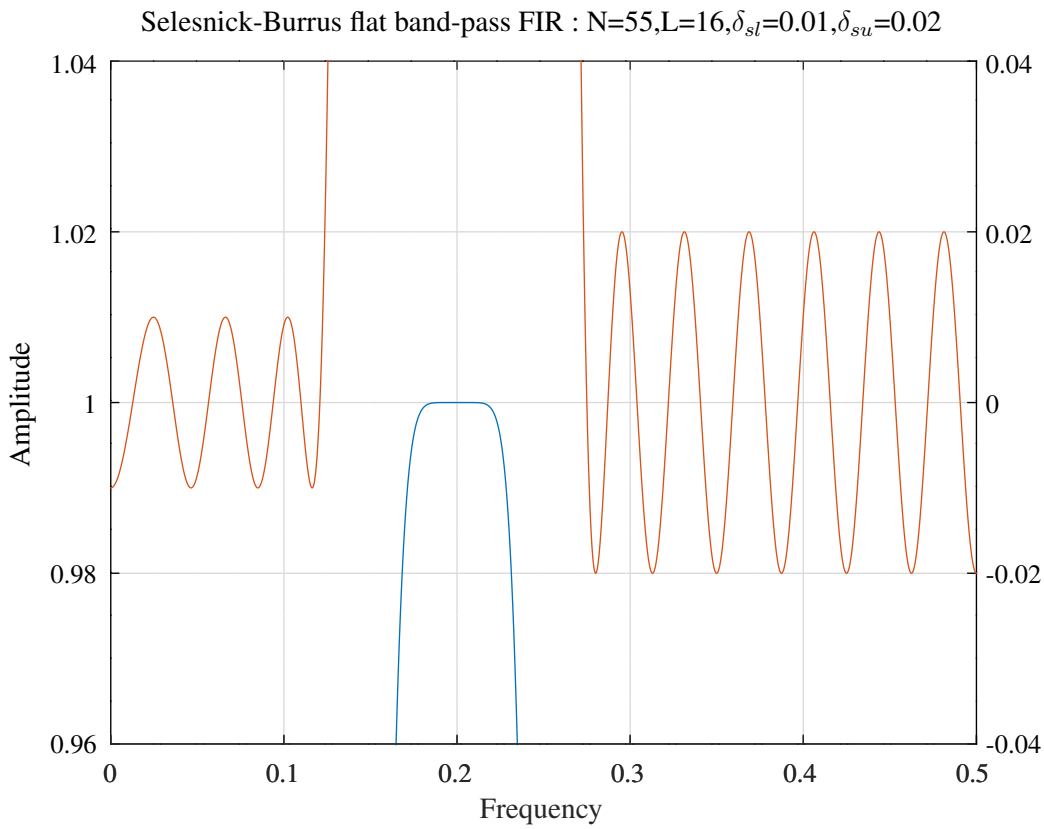



Figure N.48: Response of a mini-max FIR maximally-flat pass-band band-pass filter with fixed $\delta_{su}$ and $\delta_{sl}$ [73, Section III].

The distinct filter coefficients of the overall filter, $H(z)$, are:

```
hA = [  0.005288962392, -0.005029218937,  0.001698717395,  0.000837899810, ...
       -0.006722826962, -0.007310340028,  0.003652712225,  0.008669566372, ...
        0.000373708830, -0.002473206124,  0.003854128365, -0.003966841418, ...
       -0.020504739337, -0.007257285427,  0.027139879204,  0.025442553022, ...
       -0.010768618013, -0.017976446123,  0.000514558620, -0.018567051350, ...
       -0.041227875018,  0.027796612785,  0.118120851465,  0.044099737518, ...
       -0.146972969140, -0.162797555955,  0.068053509975,  0.222063151703 ]';
```

### N.7.3 Design of maximally-linear FIR low-pass differentiators with equi-ripple stop-bands

The ideal full-band differentiator response is $H(\omega) = \imath\omega$ where $|\omega| \leq \pi$. If the input signal contains broadband random noise then a differentiator filter with a low-pass amplitude response is preferred. *Selesnick* and *Burrus* [73, Section IV, Figure 5][h] describe the design of low-pass FIR differentiator filters with equi-ripple stop-bands and "a specified degree of tangency, $L$, at $\omega = 0$". They present the design equations shown below.

#### Even-length FIR differentiators

If $N$ and $L$ are even and $S(z) = -\left(1 - z^{-1}\right)/2$ and $C(z) = -\left(1 - z^{-1}\right)^2/2$, so that $S(\omega) = \sin\frac{\omega}{2}$ and $C(\omega) = 1 - \cos\omega$, then the amplitude response is:

$$A(\omega) = \sin\left(\frac{\omega}{2}\right)\left[2 + d_1 C(\omega) + d_2 C^2(\omega) + \ldots + d_{\frac{L}{2}-1} C^{\frac{L}{2}-1}(\omega) + A_M C^{\frac{L}{2}}(\omega)\right]$$

where $A_M(\omega)$ is an arbitrary cosine polynomial of degree $M = (N - L - 1)/2$ and:

$$d_k = \frac{1 \cdot 3 \cdot 5 \cdots (2k-1)}{k! \cdot (2k+1) \cdot 2^{2k-1}}$$

For the amplitude response, $A(0) = 0$, $A'(0) = 1$ and $A^{(k)}(0) = 0$ for $k = 2, \ldots, L$.

#### Odd-length FIR differentiators

If $N$ is odd and $L$ is even and $S(z) = -\left(1 - z^{-2}\right)/2$ so that $S(\omega) = \sin\omega$, then the amplitude response is:

$$A(\omega) = \sin(\omega)\left[1 + d_1 C(\omega) + d_2 C^2(\omega) + \ldots + d_{\frac{L}{2}-1} C^{\frac{L}{2}-1}(\omega) + A_M C^{\frac{L}{2}}(\omega)\right]$$

where $A_M(\omega)$ is an arbitrary cosine polynomial and:

$$d_k = \frac{k!}{1 \cdot 3 \cdot 5 \cdots (2k+1)}$$

#### Failed design of an odd length differentiator with *mcclellanFIRsymmetric*

The Octave script *mcclellanFIRsymmetric_flat_differentiator_fail_test.m* calls the Octave function *mcclellanFIRsymmetric* to design the stop-band filter, $A_M(\omega)$, of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator filter. The filter specification is similar to that of the example of *Selesnick* and *Burrus* [73, Figure 6]:

```
N=59 % Filter length
L=34 % Filter flat-ness
fs=0.2 % Amplitude stop band frequency
```

Figure N.49 shows the response of the length $M + 1$ FIR stop-band filter, $A_M(\omega)$. The FIR stop-band filter has small errors. Figure N.50 shows the amplitude response of the differentiator filter. The small errors in the response of $A_M$ result in very large errors in the differentiator stop-band amplitude response.

[h] The caption to Figure 5 should say "N even"

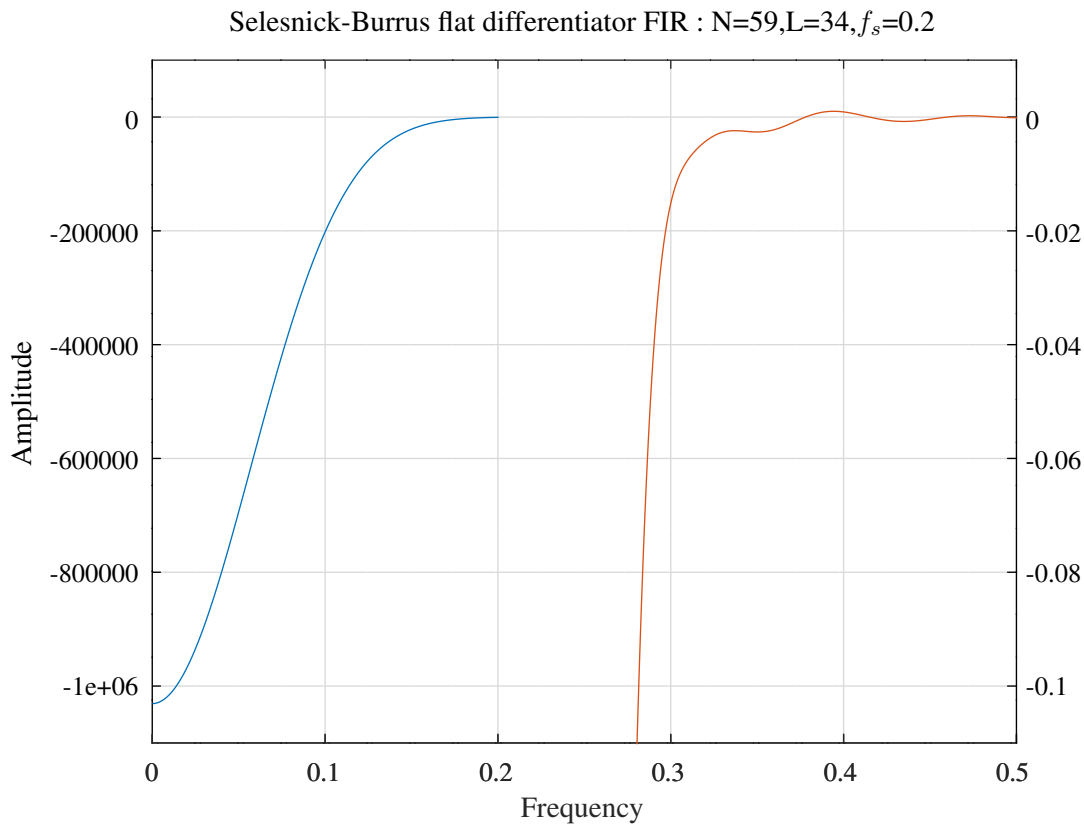



Figure N.49: Response of the $A_M$ filter of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator [73, Section IV].

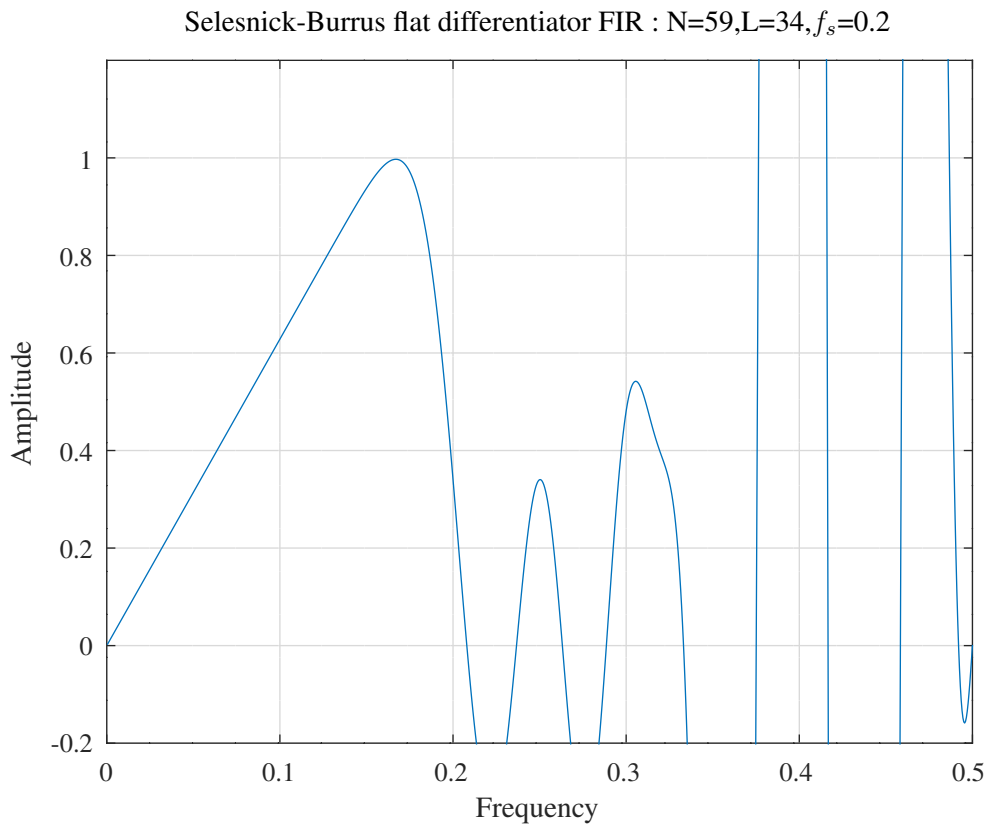



Figure N.50: Response of the failed design of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator [73, Section IV].

**Design of an odd length differentiator with *mcclellanFIRantisymmetric_flat_differentiator***

The Octave function *mcclellanFIRantisymmetric_flat_differentiator* designs maximally-linear pass-band, equi-ripple stop-band, FIR differentiator filters. It follows the MATLAB function *chebdiff.m* of *Selesnick* [190] for even-length filters and adds the odd-length case. The Octave script *mcclellanFIRantisymmetric_flat_differentiator_test.m* exercises this function to design an even-length, odd-order and an odd-length, even-order filter. The filter specification for the even-length filter is similar to that of the example of *Selesnick* and *Burrus* [73, Figure 6]. The specification of the odd-length filter is:

```
N=57 % Filter length
L=34 % Filter flat-ness
deltas=0.0001 % Amplitude stop band ripple
```

The coefficients of the odd-length, even-order, anti-symmetric, FIR differentiator filter are[i]:

```
hA57 = [ -0.000000026687,  0.000000490000, -0.000003958470,  0.000017980629, ...
         -0.000047304433,  0.000057983906,  0.000034848673, -0.000205142638, ...
          0.000104960387,  0.000475677202, -0.000593304133, -0.000865109279, ...
          0.001806247068,  0.001319859178, -0.004353873501, -0.001749075510, ...
          0.009179985235,  0.002096147899, -0.017729632353, -0.002508748862, ...
          0.032351684085,  0.003724726890, -0.057627851331, -0.008372615640, ...
          0.106138256967,  0.029574595700, -0.239992860846, -0.311662304083, ...
         -0.000000000023,  0.311662304123,  0.239992860823, -0.029574595697, ...
         -0.106138256955,  0.008372615619,  0.057627851351, -0.003724726905, ...
         -0.032351684076,  0.002508748857,  0.017729632355, -0.002096147900, ...
         -0.009179985234,  0.001749075509,  0.004353873501, -0.001319859178, ...
         -0.001806247068,  0.000865109279,  0.000593304133, -0.000475677202, ...
         -0.000104960387,  0.000205142638, -0.000034848673, -0.000057983906, ...
          0.000047304433, -0.000017980629,  0.000003958470, -0.000000490000, ...
          0.000000026687 ]';
```

Figure N.51 shows the pass-band and stop-band amplitude responses of the odd-length even-order FIR maximally-linear pass-band differentiator filter. The width of the stop-band is determined by the stop-band ripple specification. There are $M = (N-1-L)/2$ stop-band extremal frequencies. Figure N.52 shows the zeros of the differentiator filter. There are single zeros at $z = \pm 1$. Figure N.53 shows the values of odd or even $N$ and even $L$ with *deltas=1e-4* and *tol=1e-8* for which the Octave function *mcclellanFIRantisymmetric_flat_differentiator* converges to a feasible filter.

[i]This truncation introduces ripple of about $\pm 1e-7$ into the maximally-linear low-pass differentiator response.

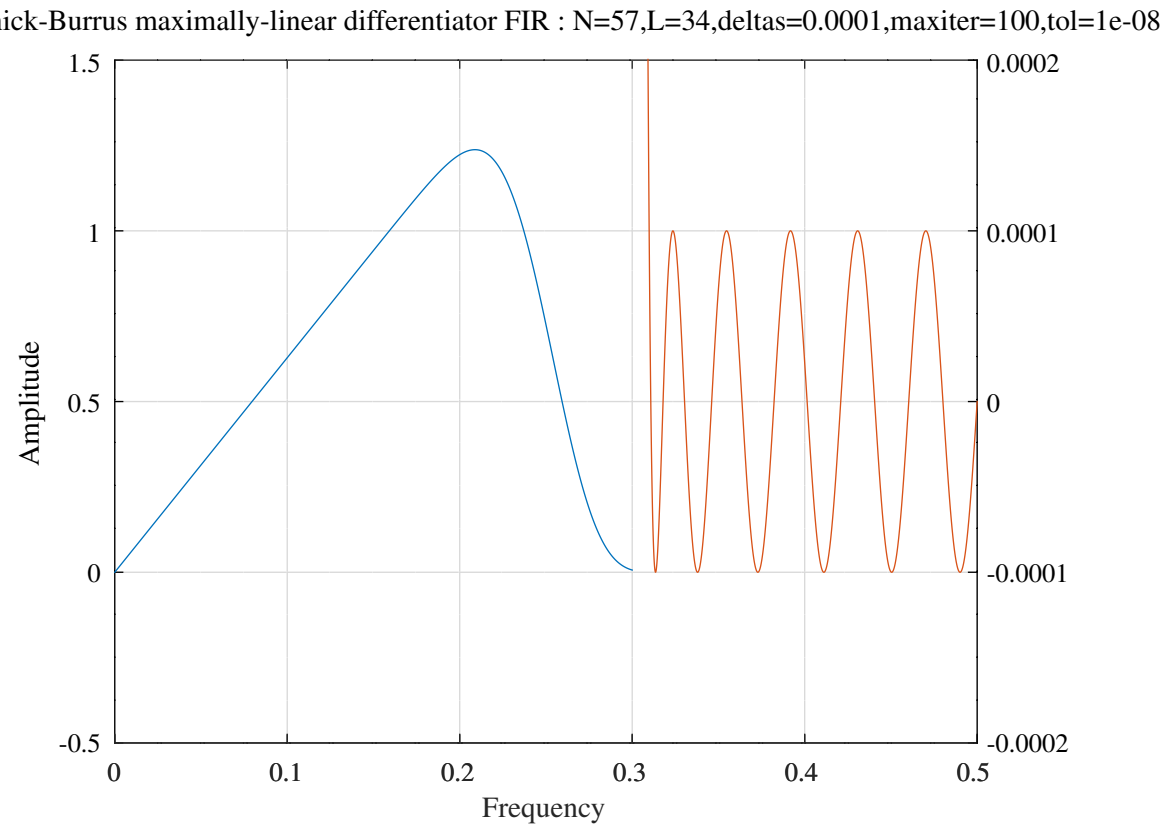



Figure N.51: Pass-band and stop-band amplitude responses of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator [73, Section IV].

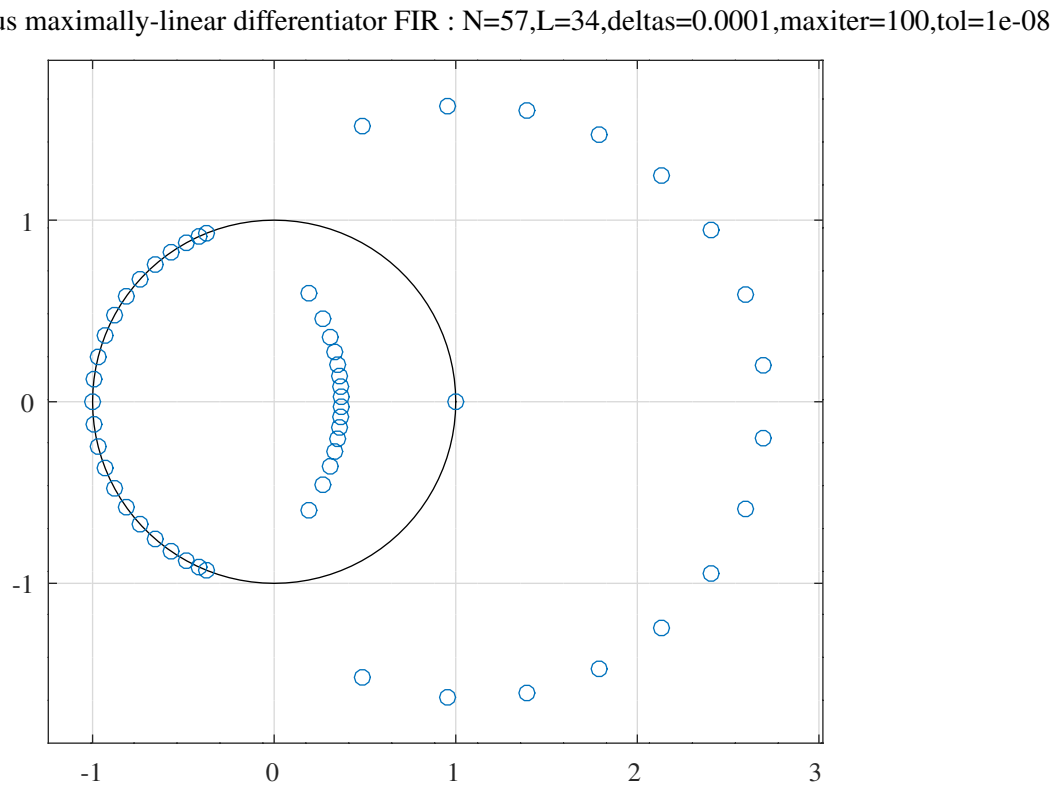



Figure N.52: Zeros of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator [73, Section IV].

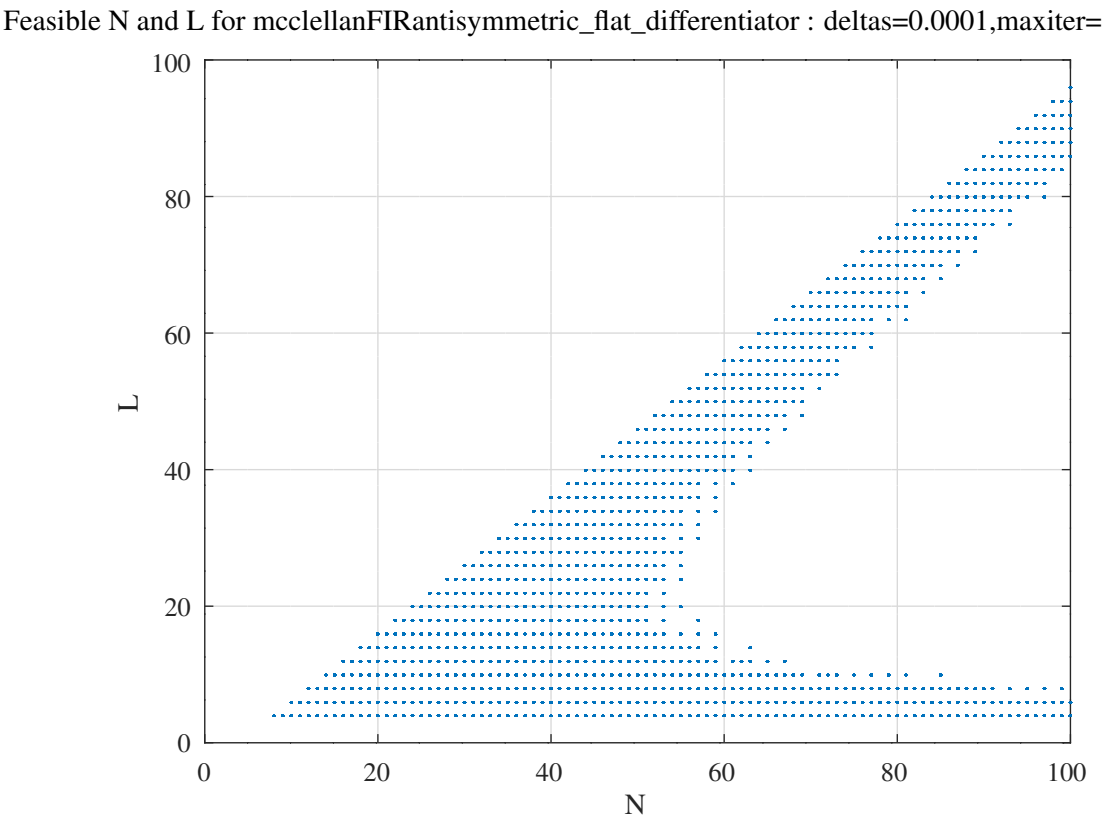



Figure N.53: Feasible values of odd or even $N$ and even $L$ with *deltas=1e-4* and *tol=1e-8* for the maximally-linear pass-band, equi-ripple stop-band, FIR differentiator designed by the Octave function *mcclellanFIRantisymmetric_flat_differentiator*

.

## N.8 Closed-form design of maximally-linear FIR filters

Maximally-linear filters are preferred when a signal is repeated or re-transmitted many times. The linearity constraints allow a closed-form solution for the filter coefficients. *Samadi* and *Nishihara* give a historical survey of maximally-flat FIR filters [187]. They emphasise their use since the late 19th century by actuaries for *graduation* or data-smoothing. Maximally-linear FIR filters are typically longer that equi-ripple FIR filters but may have fewer multipliers.

### N.8.1 Closed-form design of maximally-flat low-pass FIR filters

***Herrmann*'s closed form design of FIR low-pass filters that are maximally-flat at $\omega = 0$ and $\omega = \pi$**

*Herrman* [135] describes a method of designing maximally-flat low-pass FIR filters with a closed form expression for the coefficients. The zero-phase frequency response of the filter is:

$$A(\omega) = \sum_{k=0}^{M} d_k \cos k\omega$$

where the length of the filter is $N = 2M + 1$. The maximally-flat constraints on the low-pass filter coefficients are:

$$\begin{aligned} A(\omega)\big|_{\omega=0} &= 1 \\ A(\omega)\big|_{\omega=\pi} &= 0 \\ \frac{d^p}{d\omega^p} A(\omega)\bigg|_{\omega=0} &= 0 \quad p = 1, 2, \ldots, 2L-1 \\ \frac{d^q}{d\omega^q} A(\omega)\bigg|_{\omega=\pi} &= 0 \quad q = 1, 2, \ldots, 2K-1 \end{aligned}$$

where $M = L + K - 1$ and $L$ and $K$ represent the required degrees of "flatness" at $\omega = 0$ and $\omega = \pi$. *Herrmann* defines a transformation $\cos\omega = 1 - 2x$ on the interval $[0, 1]$ by which $A(\omega)$ is transformed into:

$$P_{M,K}(x) = \sum_{k=0}^{M} a_k x^k$$

$P_{M,K}(x)$ has $K$ zeros at $x = 1$ and $L = M - K + 1$ zeros at $x = 0$. *Herrmann* states that:

$$P_{M,K}(x) = (1-x)^K \sum_{k=0}^{M-K} \binom{M+k-1}{k} x^k$$

*Herrmann* gives an empirical relation for the filter cut-off frequency, $x_c$:

$$K = M - \lfloor M x_c + 0.5 \rfloor$$

*Rajagopal* and *Roy* [107] derive $P_{M,K}(x)$ from the properties of the *Bernstein polynomial*[j] representation of a function $f(x)$ on the interval $[0, 1]$:

$$B_M(f; x) = \sum_{k=0}^{M} f\left(\frac{k}{M}\right) \binom{M}{k} x^k (1-x)^{M-k}$$

where the values of $f(x)$ at $x = \frac{0}{M}, \frac{1}{M}, \ldots, \frac{M}{M}$ are known. They point out that for a low-pass function, $f(x)$:

$$f\left(\frac{k}{M}\right) = \begin{cases} 1, & 0 \le k \le M - K \\ 0, & M - K + 1 \le k \le M \end{cases}$$

we get:

$$B_{M,K}(f; x) = \sum_{k=0}^{M-K} \binom{M}{k} x^k (1-x)^{M-k}$$

[j] In computer graphics a two- or three-dimensional *Bernstein polynomial* [52] is called a *Bezier curve*. The *DeCasteljau recursion* is a numerically stable method for calculation of the value of the Bernstein polynomial.

Expanding and simplifying, *Rajagopal* and *Roy* [107, Appendix A] derive *Herrmann*'s $P_{M,K}(x)$:

$$B_{M,K}(x) = (1-x)^K \sum_{k=0}^{M-K} \binom{M+k-1}{k} x^k \tag{N.16}$$

Identifying $x$ with $-\frac{1}{4}\left(1-z^{-1}\right)^2$ and $1-x$ with $\frac{1}{4}\left(1+z^{-1}\right)^2$, the filter transfer function corresponding to the *Bernstein polynomial* representation of the low-pass function, $f(x)$, is:

$$H(z) = \frac{1}{4^M} \sum_{k=0}^{M-K} (-1)^k \binom{M}{k} \left(1-z^{-1}\right)^{2k} \left(1+z^{-1}\right)^{2(M-k)}$$

Alternatively, applying *Bernstein interpolation* to $f(x)$:

$$B_M(f;x) = \sum_{k=0}^{M} \Delta^k f(0) \binom{M}{k} x^k$$

where $\Delta^k f(0)$ is the $k$th forward difference[k] at $x=0$ calculated from the values of $f(x)$ at $x = \frac{0}{M}, \frac{1}{M}, \ldots, \frac{M}{M}$, *Rajagopal* and *Roy* show that [107, Equation 23]:

$$\begin{aligned} a_0 &= 1 \\ a_k &= \begin{cases} 0, & k = 1, \ldots, L-1 \\ (-1)^{k-L+1} \cdot \frac{L}{k} \cdot \binom{M-L}{k-L} \cdot \binom{M}{L}, & k = L, \ldots, M \end{cases} \end{aligned}$$

In this case the filter transfer function is:

$$H(z) = \sum_{k=0}^{M} a_k \left[-\frac{1}{4}\left(1-z^{-1}\right)^2\right]^k$$

Figure N.54 shows the $x$-domain responses of maximally-flat lowpass filters for $M=10$ and $K=1,\ldots,M$ designed with the Octave function *herrmannFIRsymmetric_flat_lowpass*. It reproduces *Herrmann*'s Figure 1 [135].

The distinct coefficients of the $z$-domain impulse response of an $M=19$ and $K=11$ maximally-flat low-pass filter are:

```
hM19K11 = [  0.000000159191,  0.000000672138, -0.000001957109, -0.000012098491, ...
             0.000007057453,  0.000102837177,  0.000027686932, -0.000548464945, ...
            -0.000430046377,  0.002056743542,  0.002490944957, -0.005758881918, ...
            -0.009666694648,  0.012477577489,  0.029411432653, -0.021390132839, ...
            -0.080658625913,  0.029411432653,  0.308820042861,  0.467320630385 ];
```

The numerical range of the maximally-flat filter $a_k$ coefficients increases rapidly with the length of the filter.

[k] The forward differences are:

$$\begin{aligned} \Delta^0 f_n &= f_n \\ \Delta^1 f_n &= f_{n+1} - f_n \\ \Delta^k f_n &= \Delta^{k-1} f_{n+1} - \Delta^{k-1} f_n \\ &= \sum_{p=0}^{k} (-1)^p \binom{k}{p} f_{n+p} \end{aligned}$$

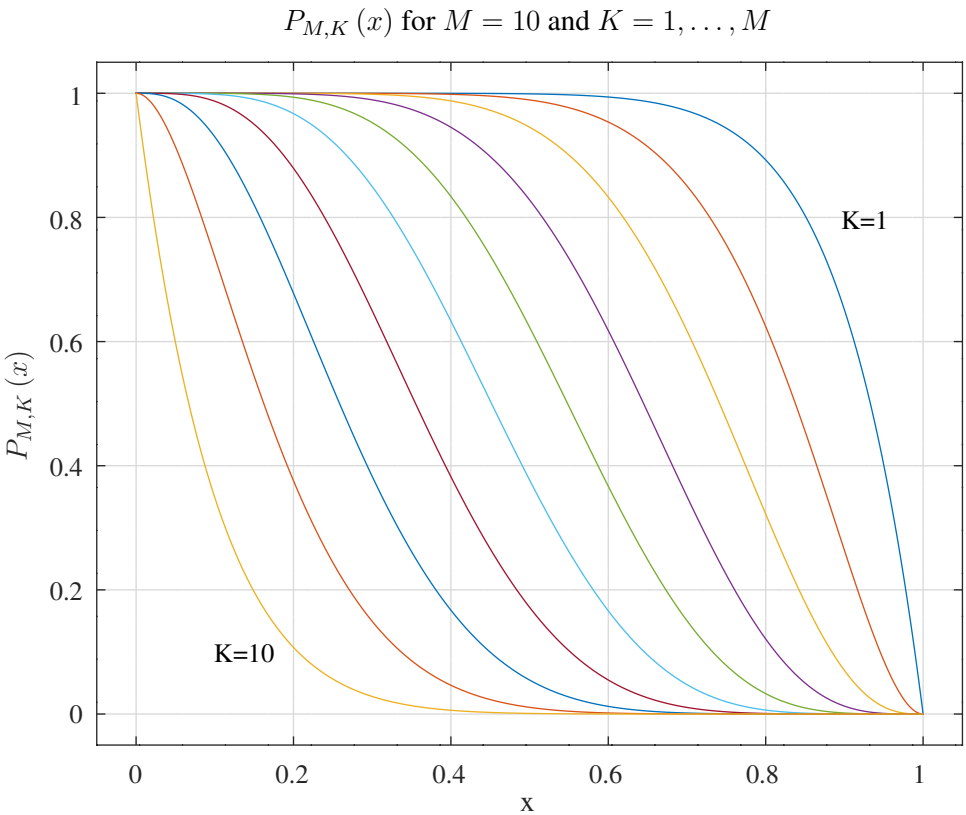



Figure N.54: $x$-domain amplitude responses of maximally-flat low-pass filters designed by the method of *Herrmann* with $M = 10$ and $K = 1, \ldots, M$ [135, Figure 1].

**Closed-form design of FIR filters by *Nuevo et al.*'s interpolation with a maximally-flat model filter**

*Nuevo et al.* [226] describe a general method for the design of *interpolated FIR filters* of the form $H(z) = H_M\left(z^P\right)G(z)$. $H_M(z)$ is called the model filter, designed to meet the required frequency specification in $z^P$. $H_M(z)$ is transformed by interpolating $P-1$ zeros between each impulse response coefficient creating $P$ images of the desired frequency response. The $G(z)$ filter places zeros at each of the $P-1$ unwanted images. This method is claimed to "implement most practical FIR filters with significant savings in the number of arithmetic operations".

The Octave script *nuevoFIRsymmetric_flat_bandpass_test.m* applies this method to the design of a bandpass filter. The maximally flat model filter is designed by the Octave function *herrmannFIRsymmetric_flat_lowpass* with $M = 19$ and $fc = 0.2$ giving $K = 12$. The interpolation factor is $P = 8$. The script constructs an interpolator filter, $G(z)$, with double zeros at $z = 1$ and $z = -1$, a pair of zeros at $z = \pm\imath$ and pairs of zeros at on the unit circle at $\omega = \pm 2\pi\,[3, 29, 35, 47, 48, 50, 62]\,/\,(16P)$. The latter were adjusted "by-eye". The resulting filter has a model filter with length 39 and an overall filter length of 325. The model filter has 20 distinct coefficients and the interpolator filter requires 8 distinct non-power-of-two multipliers. Figure N.55 shows the responses of the interpolated maximally-flat model filter and the interpolator filter. Figure N.56 shows the overall response of the interpolated filter. The distinct coefficients of the model filter are:

```
hM = [ -0.000000115775, -0.000000977656, -0.000001682143,  0.000008798903, ...
        0.000035928853, -0.000008798903, -0.000247553748, -0.000258101150, ...
        0.000888689188,  0.001935758628, -0.001537852688, -0.007637447678, ...
       -0.001205449691,  0.020284404047,  0.017043474829, -0.039348693565, ...
       -0.068841149448,  0.057879182976,  0.303865710623,  0.434291748796 ]';
```

The coefficients of the two sections of the interpolator filter are:

```
hza = [ -1.000000000000,  0.000000000000,  1.000000000000,  0.000000000000, ...
         1.000000000000,  0.000000000000, -1.000000000000 ]';
```

and

```
hzb = [ -0.004136886270, -0.017852189988, -0.038019167415, -0.047930433674, ...
        -0.031132075251,  0.011819680270,  0.059109120400,  0.079676236949, ...
         0.059109120400,  0.011819680270, -0.031132075251, -0.047930433674, ...
        -0.038019167415, -0.017852189988, -0.004136886270 ]';
```

Figure N.57 shows the overall response of the filter with 16-bit rounded coefficients. The model filter now has 16 distinct coefficients, a total of 24 multipliers are required and the overall filter length is 261. The distinct 16-bit coefficients of the model filter are:

```
hMf = [        1,        0,       -8,       -8, ...
              29,       63,      -50,     -250, ...
             -40,      665,      558,    -1289, ...
           -2256,     1897,     9957,    14231 ]'/32768;
```

The 16-bit coefficients of the interpolator filter are:

```
hzbf = [     -136,     -585,    -1246,    -1571, ...
            -1020,      387,     1937,     2611, ...
             1937,      387,    -1020,    -1571, ...
            -1246,     -585,     -136 ]'/32768;
```

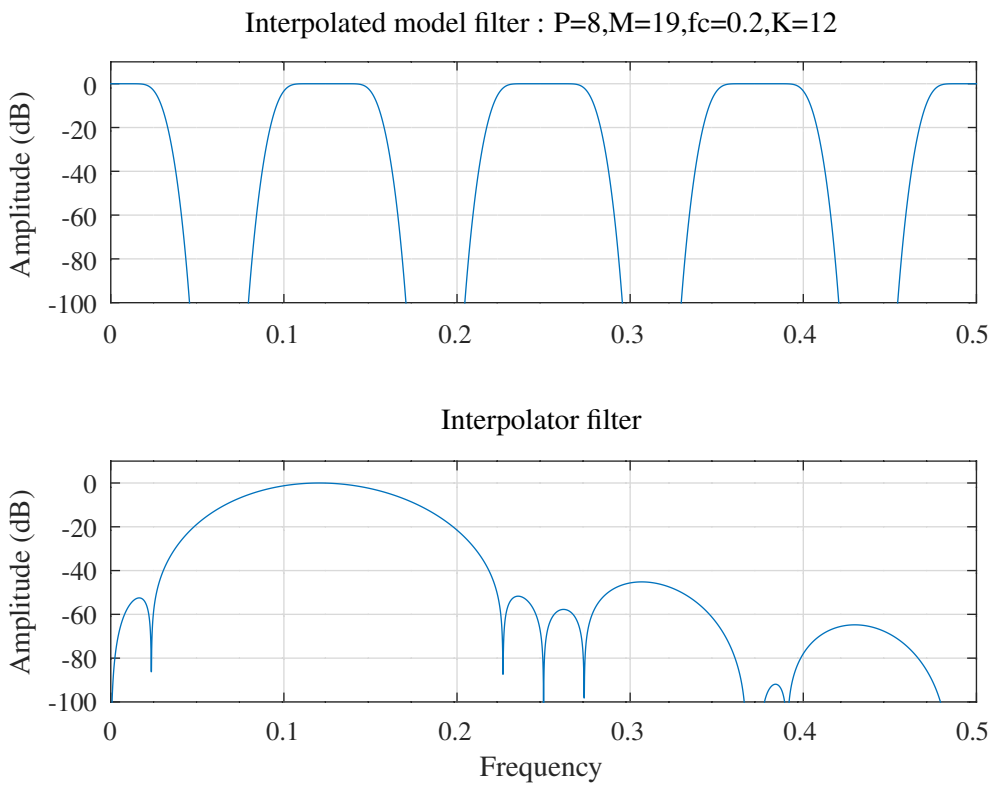



Figure N.55: Amplitude response of the interpolated maximally-flat model filter and interpolator filter with $P = 8$, $M = 19$, $fc = 0.2$, $K = 12$.

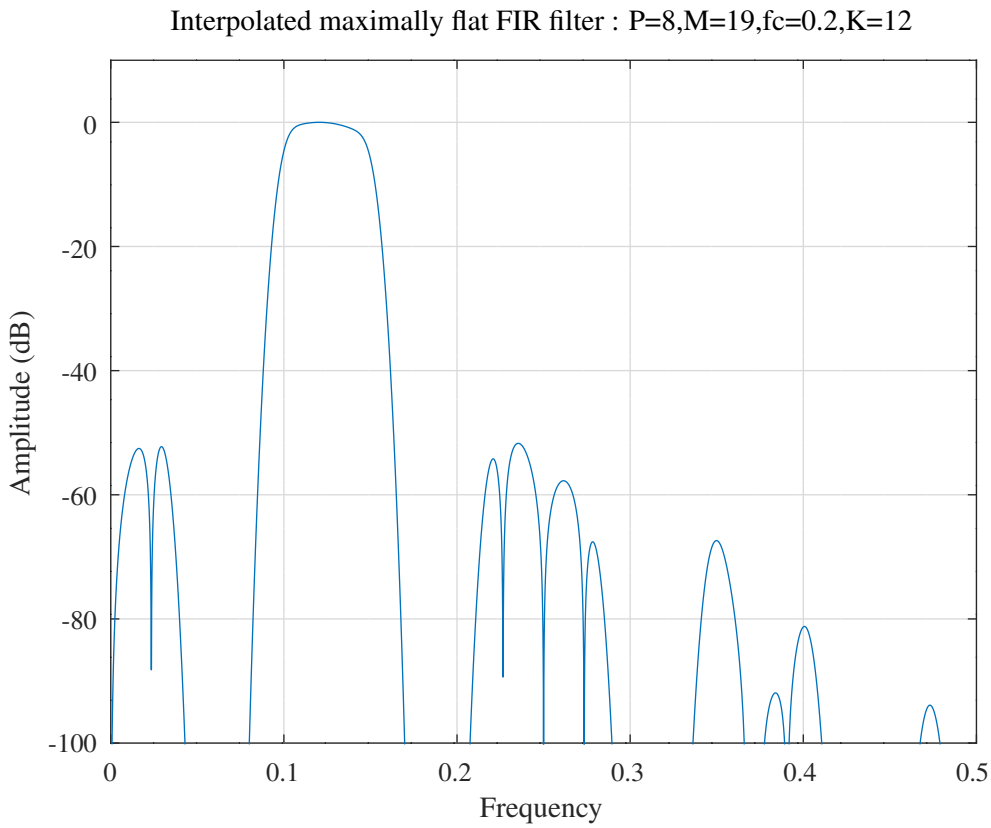



Figure N.56: Amplitude response of an interpolated maximally-flat band-pass filter with $P = 8$, $M = 19$, $fc = 0.2$, $K = 12$.

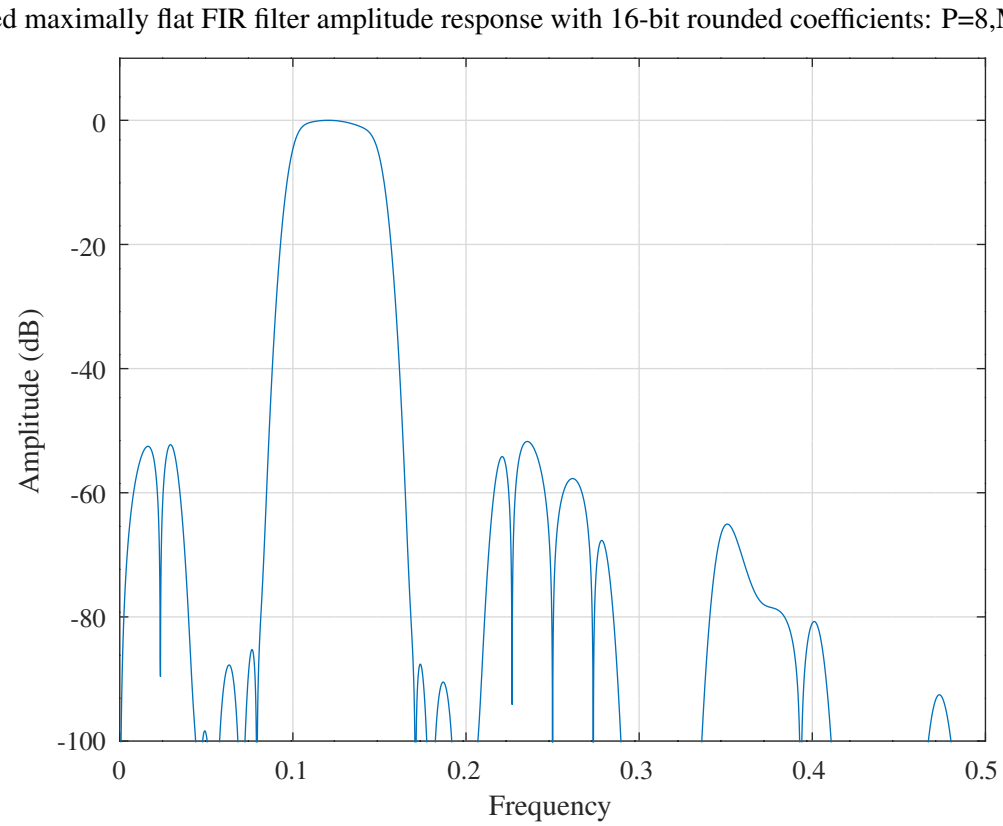



Figure N.57: Amplitude response of an interpolated maximally-flat band-pass filter with $P = 8$, $M = 19$, $fc = 0.2$, $K = 12$ and 16-bit rounded coefficients.

***Vlček et al.*'s recursive algorithm for the calculation of the coefficients of odd-length, symmetric, maximally-flat, FIR low-pass filters**

*Vlček et al.* [129] give a recursive algorithm for the coefficients of an odd-length, even-order, symmetric, maximally flat, low-pass FIR filter. They begin with a length $N = 2M + 1$ $z$-domain impulse response [129, Equations 1 to 4]:

$$\begin{aligned} H(z) &= \sum_{k=0}^{N-1} h_k z^{-k} \\ &= z^{-M}\left[h_M + 2\sum_{k=1}^{M} h_{M-n} \cdot \frac{1}{2}\left(z^k + z^{-k}\right)\right] \end{aligned}$$

The zero-phase frequency response is:

$$A(v) = \left[a_0 + \sum_{k=1}^{M} a_k T_k(v)\right]$$

where $a_0 = h_M$, $a_k = 2h_{M-m}$, $v = \cos\omega$ and $T_k(\omega) = \cos k\omega$ is a *Chebyshev polynomial of the first kind*. If the impulse response has all zeros at $z = \pm 1$ then the zero-phase amplitude response is a *Bernstein* polynomal. Re-writing Equation N.16 in terms of $v$ [129, Equation 9][1], $A(v) = C_{M,K}(v)$, where:

$$C_{M,K}(v) = \left(\frac{1+v}{2}\right)^K \sum_{k=0}^{M-K} \binom{M+k-1}{k} \left(\frac{1-v}{2}\right)^k$$

*Vlček et al.* state that:

$$\frac{d}{dv} C_{M,K}(v) = 2^{-M} M \binom{M-1}{M-K} (1-v)^{M-K} (1+v)^{K-1} \tag{N.17}$$

Differentiating again [129, Equation 11]:

$$\left(1-v^2\right)\frac{d^2}{dv} C_{M,K}(v) + \left[(M-2K+1) + (M-1)v\right]\frac{d}{dv} C_{M,K} = 0$$

The Chebyshev polynomials of the first kind have the property:

$$\frac{d}{dv} T_k(v) = kU_{k-1}(v)$$

where $U_k(v)$ is a *Chebyshev polynomial of the second kind*. An alternative expression of Equation N.17 is:

$$\frac{d}{dv} C_{M,K}(v) = \sum_{k=1}^{M} k a_k U_{k-1}(v)$$

Substituting $\alpha_k = k a_k$:

$$\begin{aligned} \left(1-v^2\right)\frac{d^2}{dv^2} C_{M,K}(v) &= \sum_{k=1}^{M} \alpha_k \left(1-v^2\right)\frac{d}{dv} U_{k-1}(v) \\ &= -\sum_{k=1}^{M}\left(\frac{k-1}{2}\right)\alpha_k U_k(v) + \sum_{k=1}^{M}\left(\frac{k+1}{2}\right)\alpha_k U_{k-2}(v) \end{aligned}$$

The second-order differential equation becomes:

$$\sum_{k=1}^{M}\frac{M-k}{2}\alpha_k U_k(v) + \sum_{k=1}^{M}(M-2K+1)\alpha_k U_{k-1}(v) + \sum_{k=1}^{M}\frac{M+k}{2}\alpha_k U_{k-2}(v) = 0$$

Setting $\alpha_0 = 0$ and $U_{-1}(v) = 0$:

$$\sum_{k=1}^{M+1}\frac{M-k+1}{2}\alpha_{k-1} U_{k-1}(v) + \sum_{k=1}^{M}(M-2K+1)\alpha_k U_{k-1}(v) + \sum_{k=1}^{M-1}\frac{M+k+1}{2}\alpha_{k+1} U_{k-1}(v) = 0$$

[1] In the notation of *Vlček et al.*, $p = M - K$ and $q = K - 1$.

For $k = M + 1$, the first sum gives:

$$\frac{M - M}{2} \alpha_M U_M(v) = 0$$

Comparing the coefficient of the highest power of $v$ in the expansion of $\frac{d}{dv} A(v)$ in Chebyshev polynomials of the second kind, $U_{M-1}(v)$, with that in $\frac{d}{dv} C_{M,K}(v)$:

$$2^{M-1} M a_M = (-1)^{M-K} 2^{-M} M \begin{pmatrix} M-1 \\ M-K \end{pmatrix}$$
$$\alpha_M = (-1)^{M-K} 2^{-2M+1} M \begin{pmatrix} M-1 \\ M-K \end{pmatrix}$$

Setting $k = M$ gives:

$$\left( \frac{1}{2} \alpha_{M-1} + (M - 2K + 1) \alpha_M \right) U_{k-1}(v) = 0$$

or:

$$\alpha_{M-1} = -2(M - 2K + 1) \alpha_M$$

Finally, $a_0$ is given by the value of $A(v)$ at $v = 1$ or $\omega = 0$:

$$a_0 = 1 - \sum_{k=1}^{M} a_M$$

Algorithm N.3 summarises the calculation. It reproduces Table I of *Vlček et al.*.

---

**Algorithm N.3** *Vlček et al.*'s backwards recursion for the calculation of the impulse response of an odd-length, symmetric, maximally-flat, low-pass FIR filter [129, Table 1].

---

**Require:** $M$, $K$

*Initialisation:*
$\alpha_M = (-1)^{M-K} 2^{-2M+1} M \begin{pmatrix} M-1 \\ M-K \end{pmatrix}$
$\alpha_{M-1} = -2(M - 2K + 1) \alpha_M$

*Body:*
**for** $k = M - 1$ **down to** $2$ **do**
  $\frac{M-k+1}{2} \alpha_{k-1} = (2K - 1 - M) \alpha_k - \frac{M+k+1}{2} \alpha_{k+1}$
**end for**

*Integration:*
**for** $k = M$ **down to** $1$ **do**
  $a_k = \frac{\alpha_k}{k}$
**end for**
$a_0 = 1 - \sum_{k=1}^{M} a_M$

*Impulse response:*
**for** $k = M$ **down to** $1$ **do**
  $h_{M \pm k} = \frac{a_k}{2}$
**end for**
$h_M = a_0$

---

The backwards recursion of *Vlček et al.* can successfully calculate the coefficients of much longer filters than is possible with the "direct" calculation of *Herrman* [135] or *Rajagopal* and *Roy* [107]. The Octave function *vlcekFIRsymmetric_flat_lowpass* calculates the distinct coefficients of a symmetric, odd-length, maximally-flat FIR low-pass filter by the backwards recursion. Figure N.58 shows the amplitude responses of maximally-flat low-pass filters with $M = 300$ and $K = 10, 30, \ldots, 290$.

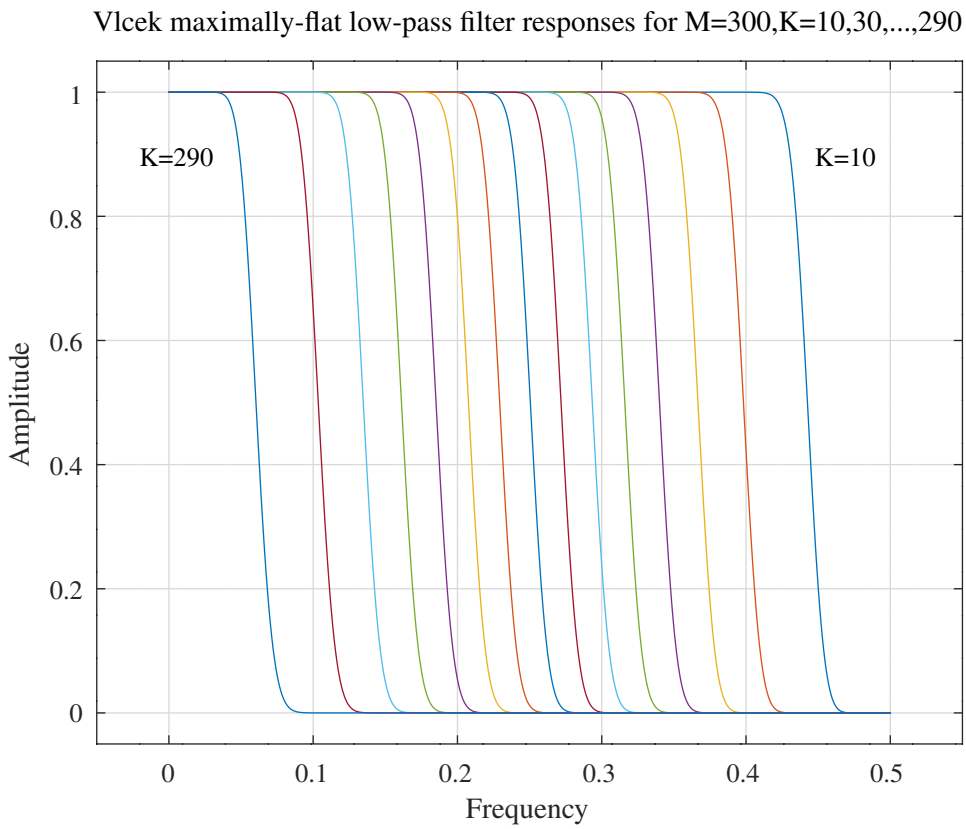



Figure N.58: Amplitude responses of maximally-flat low-pass filters designed by the backwards recursion of *Vlček et al.* [129, Table 1] with $M = 300$ and $K = 10, 30, \ldots, 290$.

**Working for *Vlček et al.* Equation 14 [129]** The recursion for the Chebyshev polynomials of the first kind is:

$$\begin{aligned} T_0(v) &= 1 \\ T_1(v) &= v \\ T_k(v) &= 2vT_{k-1}(v) - T_{k-2}(v) \end{aligned}$$

The recursion for the Chebyshev polynomials of the second kind is:

$$\begin{aligned} U_0(v) &= 1 \\ U_1(v) &= 2v \\ U_k(v) &= 2vU_{k-1}(v) - U_{k-2}(v) \end{aligned}$$

Also:

$$T_k(v) = U_k(v) - vU_{k-1}(v)$$

The derivatives of the Chebyshev polynomials are:

$$\begin{aligned} \frac{d}{dv}T_k(v) &= kU_{k-1}(v) \\ \left(v^2 - 1\right)\frac{d}{dv}U_k(v) &= (k+1)\,T_{k+1} - vU_k(v) \end{aligned}$$

Accordingly:

$$\begin{aligned} \left(1 - v^2\right)\frac{d}{dv}U_{k-1}(v) &= vU_{k-1}(v) - kT_k \\ &= vU_{k-1}(v) - kU_k(v) + kvU_{k-1}(v) \\ &= (k+1)\,vU_{k-1}(v) - kU_k(v) \\ &= (k+1)\left(\frac{U_k(v) + U_{k-2}(v)}{2}\right) - kU_k(v) \\ &= -\left(\frac{k-1}{2}\right)U_k(v) + \left(\frac{k+1}{2}\right)U_{k-2}(v) \end{aligned}$$

### N.8.2 Closed-form design of maximally-flat FIR half-band filters

***Gumacos*'s closed form design of FIR half-band filters that are maximally-flat at $\omega = 0$**

*Gumacos* [27, Equation 1] represents the zero-phase amplitude response of a half-band filter, $A(\omega)$, by a power series:

$$A(\omega) = \sum_{k=0}^{M} a_k \cos(2k+1)\omega$$

where $a_0 = 1$ and $N = 4M + 3$ is the length of the filter. Scaling and mean-value are to be determined later. The condition for maximal flatness at $\omega = 0$ is:

$$A^{(l)}(\omega)\Big|_{\omega=0} = 0 \quad l = 1, \ldots, 2M$$

resulting in a set of $M$ constraint equations:

$$1 + \sum_{k=1}^{M} (2k+1)^{2l} a_k = 0 \quad l = 1, \ldots, M$$

*Gumacos* solves this system of equations algebraically [27, Equation 4]:

$$a_k = \frac{(-1)^k}{2k+1} \cdot \frac{M!\,(M+1)!}{(M-k)!\,(M+k+1)!} \quad k = 0, 1, \ldots, M$$

Recursively:

$$\begin{aligned} a_0 &= 1 \\ a_k &= -a_{k-1} \cdot \frac{2k-1}{2k+1} \cdot \frac{M-k+1}{M+k+1} \quad k = 1, \ldots, M \end{aligned}$$

The $z$-domain transfer function is:

$$H(z) = z^{-2M-1}\left[h_0 + \sum_{k=0}^{M} h_{2k+1}\left(z^{2k+1} + z^{-2k-1}\right)\right]$$

where:

$$\begin{aligned} h_0 &= \frac{1}{2} \\ h_1 &= \left[4\sum_{k=0}^{M} a_k\right]^{-1} \\ h_{2k+1} &= h_1 a_k \quad k = 2, \ldots, M \end{aligned}$$

The Octave script *gumacosFIRsymmetric_flat_halfband_test.m* calls the Octave function *gumacosFIRsymmetric_flat_halfband.m* to design half-band filters with $M = 5, 10, 15, 20$ and 25. Figure N.59 shows the amplitude responses of the resulting half-band filters. Figure N.60 shows the amplitude responses of the corresponding Hilbert filters.

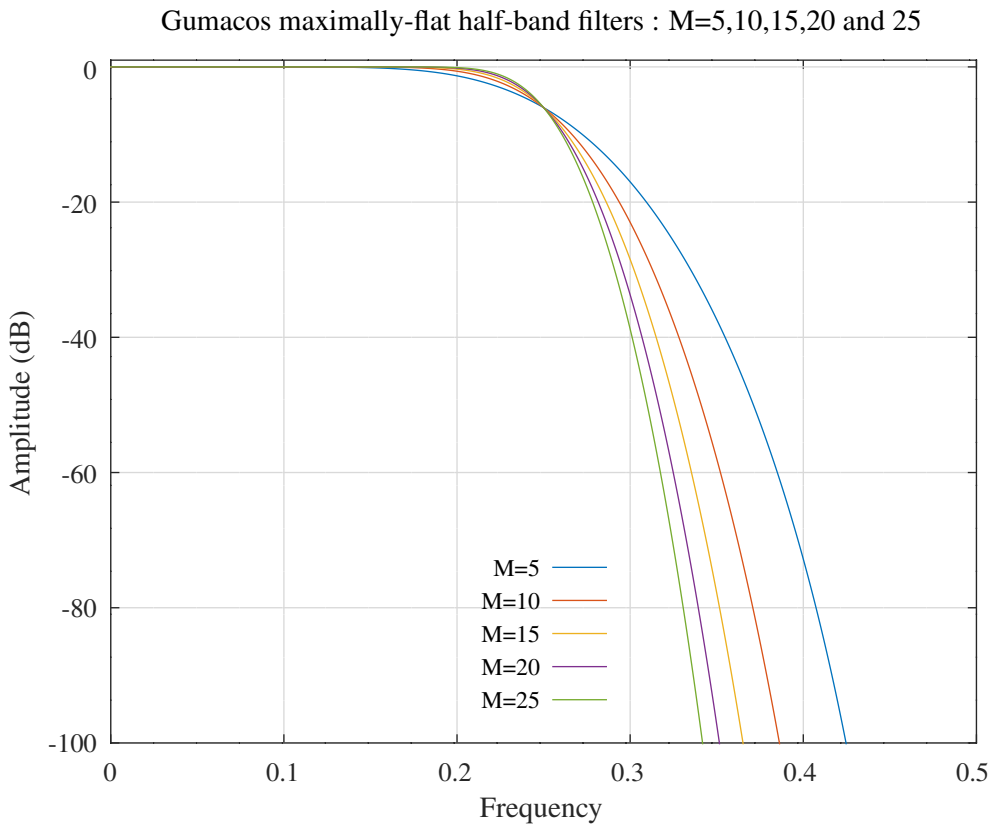



Figure N.59: Amplitude responses of maximally-flat half-band FIR filters designed by the method of *Gumacos* [27] with $M = 5, 10, 15, 20$ and 25.

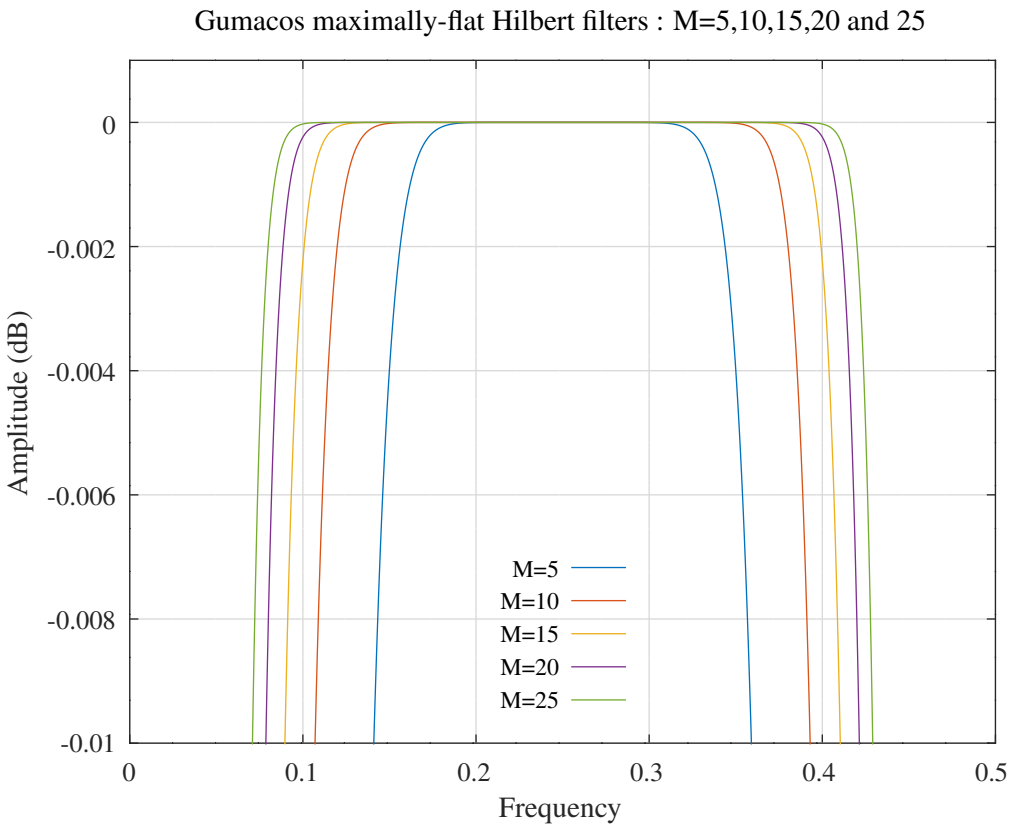



Figure N.60: Amplitude responses of maximally-flat Hilbert FIR filters derived from half-band FIR filters designed by the method of *Gumacos* [27] with $M = 5, 10, 15, 20$ and 25.

### N.8.3 Closed-form design of maximally-flat FIR Hilbert filters

#### *Pei* and *Wang*'s design of maximally-flat Hilbert FIR filters by truncated power series

*Pei* and *Wang* [192] describe the closed form design of maximally-flat Hilbert, differentiator and fractional-delay FIR filters by truncation of a power series. Here I follow their derivation of a closed form expression for a maximally-flat FIR Hilbert filter [192, Section II.A]. The Hilbert filter frequency response is $H(\omega) = -\imath \operatorname{sign} \omega$ where $-\pi < \omega < \pi$ and sign is the *sign* function:

$$\operatorname{sign} \omega = \begin{cases} -1 & \omega < 0 \\ 0 & \omega = 0 \\ 1 & \omega > 0 \end{cases}$$

*Pei* and *Wang* choose to represent $\operatorname{sign} x$ by:

$$\operatorname{sign} x = \frac{x}{\sqrt{x^2}} \quad x \neq 0$$

If $f(u) = u^{-\frac{1}{2}}$ then:

$$\operatorname{sign} x = x f\left(x^2\right) \quad x \neq 0$$

The Taylor series expansion of this representation of the $\operatorname{sign} x$ function at $x = c$ is:

$$\operatorname{sign} x = \frac{x}{\sqrt{c}} \left[ 1 + \sum_{k=1}^{\infty} \frac{(2k-1)!!}{(2k)!!} \left(1 - \frac{x^2}{c}\right)^k \right] \tag{N.18}$$

where $2k!!$ and $(2k-1)!!$ are the double factorial:

$$(2k-1)!! = 1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2k-1)$$
$$(2k)!! = 2 \cdot 4 \cdot 6 \cdot \ldots \cdot (2k)$$

This series converges for $-1 < \left(1 - \frac{x^2}{c}\right) < 1$ or $c > \frac{1}{2}x^2$.

*Pei* and *Wang* observe that:

$$\operatorname{sign} \omega = \operatorname{sign} \sin \omega = \operatorname{sign} \sin \frac{\omega}{2}$$

where $-\pi < \omega < \pi$. Consequently, if $\operatorname{sign} x$ is approximated by a polynomial in $x$, then $\sin \omega$ or $\sin \frac{\omega}{2}$ can be substituted for $x$. Subsituting $x = \sin \omega$ and using the first $M$ terms of the power series, the frequency response of the Hilbert filter is approximated by:

$$H(\omega, c) = -\imath \frac{\sin \omega}{\sqrt{c}} \left[ 1 + \sum_{k=1}^{M} \frac{(2k-1)!!}{(2k)!!} \left(1 - \frac{\sin^2 \omega}{c}\right)^k \right]$$

In the $z$-plane, substituting $\frac{\imath}{2}\left(z^{-1} - z\right)$ for $\sin \omega$ and multiplying by $z^{-2M-1}$, the causal transfer function that approximates a Hilbert filter is:

$$H(z, c) = -\frac{1 - z^{-2}}{2\sqrt{c}} \left[ z^{-2M} + \sum_{k=1}^{M} \frac{(2k-1)!!}{(2k)!!} z^{-2(M-k)} \left( z^{-2} + \frac{1}{c} \left( \frac{1 - z^{-2}}{2} \right)^2 \right)^k \right]$$

In particular, if $c = 1$, then $H(\omega)$ is maximally flat at $\omega = \frac{\pi}{2}$ and:

$$H(z) = -\frac{1 - z^{-2}}{2} \left[ z^{-2M} + \sum_{k=1}^{M} \frac{(2k-1)!!}{(2k)!!} z^{-2(M-k)} \left( \frac{1 + z^{-2}}{2} \right)^{2k} \right]$$

*Pei* and *Wang* [192, Figure 2] show a realisation of this order $4M + 2$ transfer function with $M$ coefficients of the form $\frac{2k-1}{2k}$.

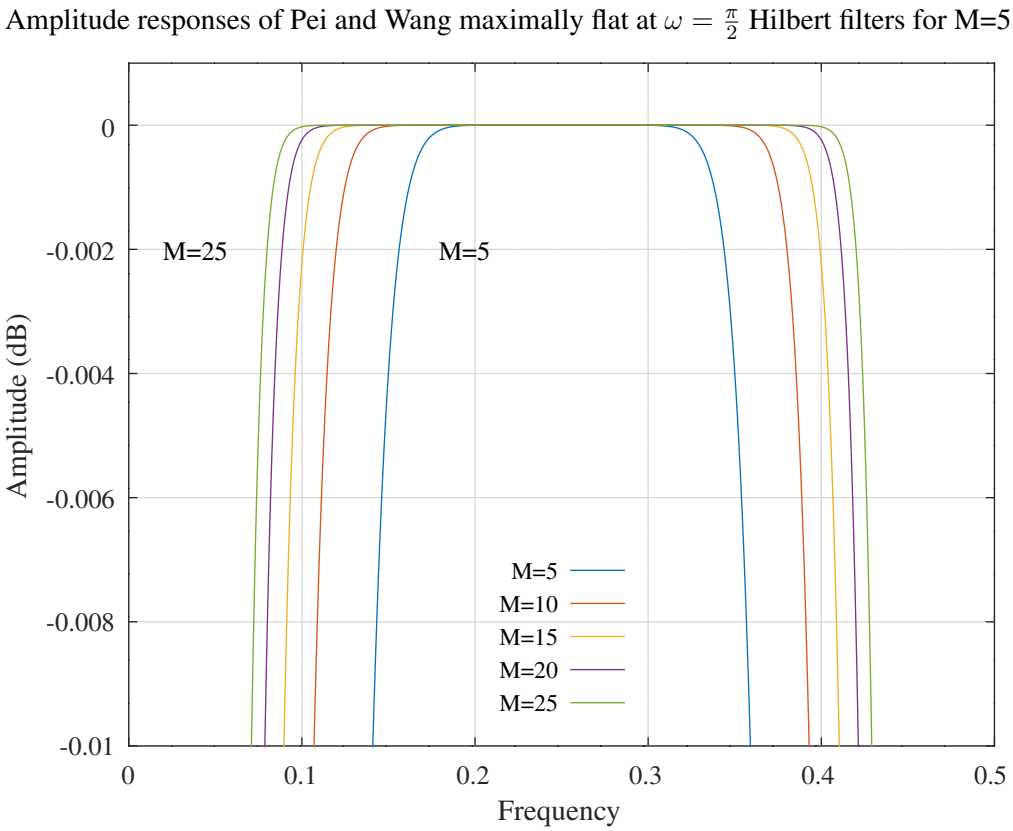



Figure N.61: Amplitude responses of *Pei* and *Wang* [192, Section II.A] maximally-flat at $\omega = \frac{\pi}{2}$ Hilbert FIR filters.

The Octave code to calculate the amplitude response for $M = 1, \ldots, 25$ is:

```
nplot=1024;
w=(0:(nplot-1))'*pi/nplot;
kM=1:25;
AM=sin(w).*(1+cumsum(cumprod(((2*kM)-1)./(2*kM)).*(cos(w).^(2*kM)),2));
```

Figure N.61 shows the amplitude responses of the maximally-flat at $\omega = \frac{\pi}{2}$ Hilbert filters for $M = 5, 10, \ldots, 25$. These Hilbert filters are identical to the *Gumacos* Hilbert filters.

**Addendum** The Taylor series expansion of $f(x) = x^{-\frac{1}{2}}$ at $x = c$ is:

$$\begin{aligned} f(x) &= f(c) + \sum_{k=1}^{\infty} \frac{f^{(k)}(c)}{k!} (x-c)^k \\ &= c^{-\frac{1}{2}} + \sum_{k=1}^{\infty} \frac{f^{(k)}(c)}{k!} (-1)^k c^k \left(1 - \frac{x}{c}\right)^k \end{aligned}$$

The derivatives of $f(x)$ are:

$$\begin{aligned} f^{(1)}(x) &= -\frac{1}{2} \cdot x^{-1-\frac{1}{2}} \\ f^{(2)}(x) &= (-1)^2 \frac{1}{2} \cdot \frac{3}{2} \cdot x^{-2-\frac{1}{2}} \\ &\vdots \\ f^{(k)}(x) &= (-1)^k \frac{1}{2} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{2k-1}{2} \cdot x^{-k-\frac{1}{2}} \end{aligned}$$

Since $k! \cdot 2^k = (2k)!!$ , Equation N.18 follows.

### N.8.4 Closed-form design of maximally-linear FIR low-pass differentiators

*Kumar* and *Roy* [22] and *Selesnick* [72] describe the closed-form design of maximally-linear low-pass FIR differentiator filters. *Yoshida et al.* [207] show formulas for an FIR differentiator that is maximally-linear at $\omega = 0$ and maximally flat at $\omega = \pi$ with a given group-delay at $\omega = 0$. *Khan et al.* [71] show formulas for the coefficients of an FIR differentiator that is maximally-linear at the middle of the frequency band, $\omega = \frac{\pi}{2}$. *Purczyński* and *Pawelczak* [87] extend the work of *Kumar et al.* [23] to show formulas for the weighting coefficients of FIR differentiators that are maximally-linear at frequencies $\pi/p$, where $p$ is a positive integer.

The derivation of the coefficients of the maximally-linear FIR differentiator begins with the constraints on zero-phase amplitude response, $A(\omega)$. When the constraint is at $\omega = 0$ [72, Equations 6 to 8]:

$$\begin{aligned} A(\omega)\big|_{\omega=0} &= 0 \\ \frac{d}{d\omega} A(\omega)\bigg|_{\omega=0} &= 1 \\ \frac{d^k}{d\omega^k} A(\omega)\bigg|_{\omega=0} &= 0 \quad k = 2, \ldots, 2L \end{aligned}$$

*Kumar etal.* [23] define similar constraints at $\omega = \frac{\pi}{p}$. *Selesnick* [72, Equation 9] adds a constraint on the amplitude response at $\omega = \pi$:

$$\frac{d^k}{d\omega^k} A(\omega)\bigg|_{\omega=\pi} = 0 \quad k = 0, \ldots, 2M$$

*Yoshida et al.* [207, Equation 3d] add a constraint on the group-delay of the frequency response, $H(\omega)$, at $\omega = 0$:

$$-\frac{d}{d\omega} \arg H(\omega)\bigg|_{\omega=0} = \tau$$

#### *Kumar* and *Roy* closed form design of FIR low-pass differentiators that are maximally-linear at $\omega = 0$

*Kumar* and *Roy* [22, Equation 2] approximate the zero-phase amplitude response, $A(\omega)$, of an FIR differentiator filter, by a power series:

$$A(\omega) = \sum_{k=1}^{n} d_k \sin k\omega$$

where $n = \frac{N-1}{2}$ and $N$ is the order of the filter, assumed to be odd. Applying the maximally-linear constraints at $\omega = 0$, the system of constraint equations is [22, Equation 4]:

$$\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2^2 & 3^2 & \ldots & n^2 \\ 1 & 2^4 & 3^4 & \ldots & n^4 \\ \vdots & & & & \vdots \\ 1 & 2^{2n-2} & 3^{2n-2} & \ldots & n^{2n-2} \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_n \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $D_k = kd_k$. As $n$ increases, this system of equations rapidly becomes numerically unstable. *Kumar* and *Roy* solve this system of constraint equations algebraically as [22, Equation 14]:

$$d_k = \left[ k \begin{pmatrix} 2n-1 \\ n-1 \end{pmatrix} \right]^{-1} \sum_{l=0}^{n-1} (-1)^l \begin{pmatrix} n \\ l \end{pmatrix} \sum_{m=0}^{2l} (-1)^m \begin{pmatrix} 2l \\ m \end{pmatrix} \begin{pmatrix} 2n-2l \\ n-k-m \end{pmatrix}$$

This appears to be a misprint. $\begin{pmatrix} 2n-2l \\ n-k-m \end{pmatrix}$ is not defined for $n-k-m < 0$. For example, when $k = n$, $l = n-1$ and $m = 2n-2$.

***Selesnick* closed form design of FIR low-pass differentiators that are maximally-linear at $\omega = 0$ and maximally-flat at $\omega = \pi$**

*Selesnick* [72, Equations 10 and 11] derives the $z$-domain transfer function, $H(z)$, of a maximally-linear differentiator from a transformation of the interval $[0, 1]$ to polynomials on the upper half-circle of $|z| = 1$:

$$P(x) = \sum_{k=0}^{n} p_k x^k$$
$$H(z) = P\left(\frac{-z + 2 - z^{-1}}{4}\right)$$

*Selesnick* [72, Section IV] gives the following equation for the $z$-domain transfer function of *Type III*[m] and *Type IV* FIR differentiators that are maximally-linear at $\omega = 0$ and maximally-flat at $\omega = \pi$:

$$H(z) = \left(\frac{1 - z^{-1}}{2}\right)\left(\frac{1 + z^{-1}}{2}\right)^K z^{-L} \sum_{n=0}^{L} c_n \left[\frac{-z + 2 - z^{-1}}{4}\right]^n$$

where the $c_n$ are defined recursively:

$$c_0 = 2$$
$$c_1 = K + \frac{1}{3}$$
$$c_n = \frac{\left(8n^2 + 4Kn - 10n - K + 3\right) c_{n-1} - (2n + K - 3)^2 c_{n-2}}{2n(2n+1)}$$

The length of the impulse response is $N = K + 2L + 2$. When $K$ is even $H(z)$ is a *Type IV* transfer function and when $K$ is odd $H(z)$ is *Type III* transfer function. When $K = 0$, the differentiator is "full-band". The Octave script *selesnickFIRantisymmetric_flat_differentiator_test.m* calls the Octave function *selesnickFIRantisymmetric_flat_differentiator* to design a series of differentiator filters. Figure N.62 shows the differentiator filters for length $N = 30$ and $K = 0, 4, 8, 12, 16, 20$ and $24$ [72, Figure 1]. Figure N.63 shows the differentiator filters for length $N = 31$ and $K = 1, 5, 9, 13, 17, 21$ and $25$ [72, Figure 2].

[m] *Selesnick* uses the terminology of *Oppenheim* and *Schafer* [143, Section 5.7.3]: *Type I* filters are symmetric and have even order, *Type II* filters are symmetric and have odd order, *Type III* filters are anti-symmetric and have even order and *Type IV* filters are anti-symmetric and have odd order.

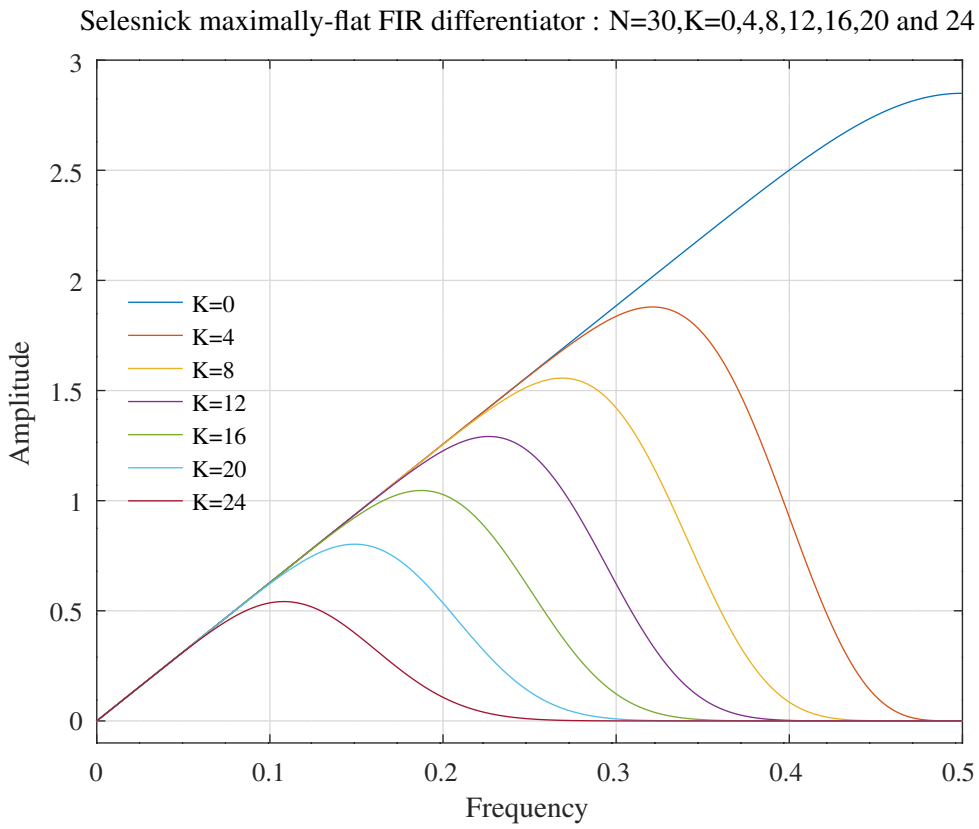



Figure N.62: Amplitude response of length $N = 30$, maximally-linear pass-band, maximally-flat stop-band, FIR differentiators with $K = 0, 4, 8, 12, 16, 20$ and 24 [72, Figure 1].

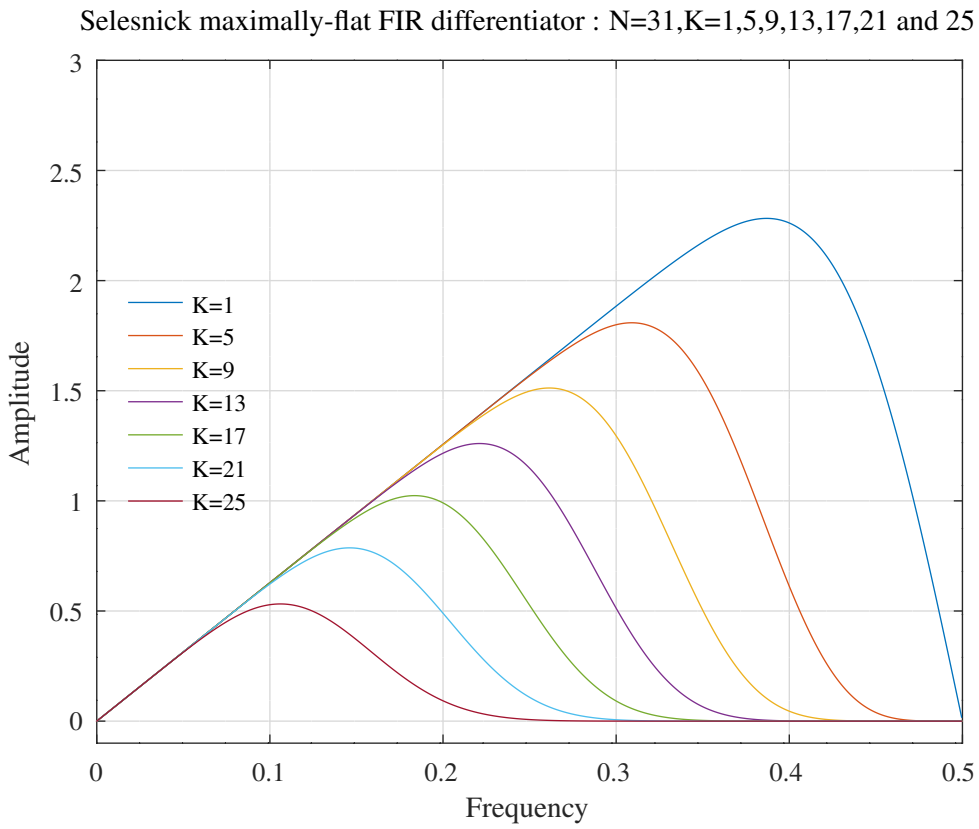



Figure N.63: Amplitude response of length $N = 31$, maximally-linear pass-band, maximally-flat stop-band, FIR differentiators with $K = 1, 5, 9, 13, 17, 21$ and 25 [72, Figure 2].

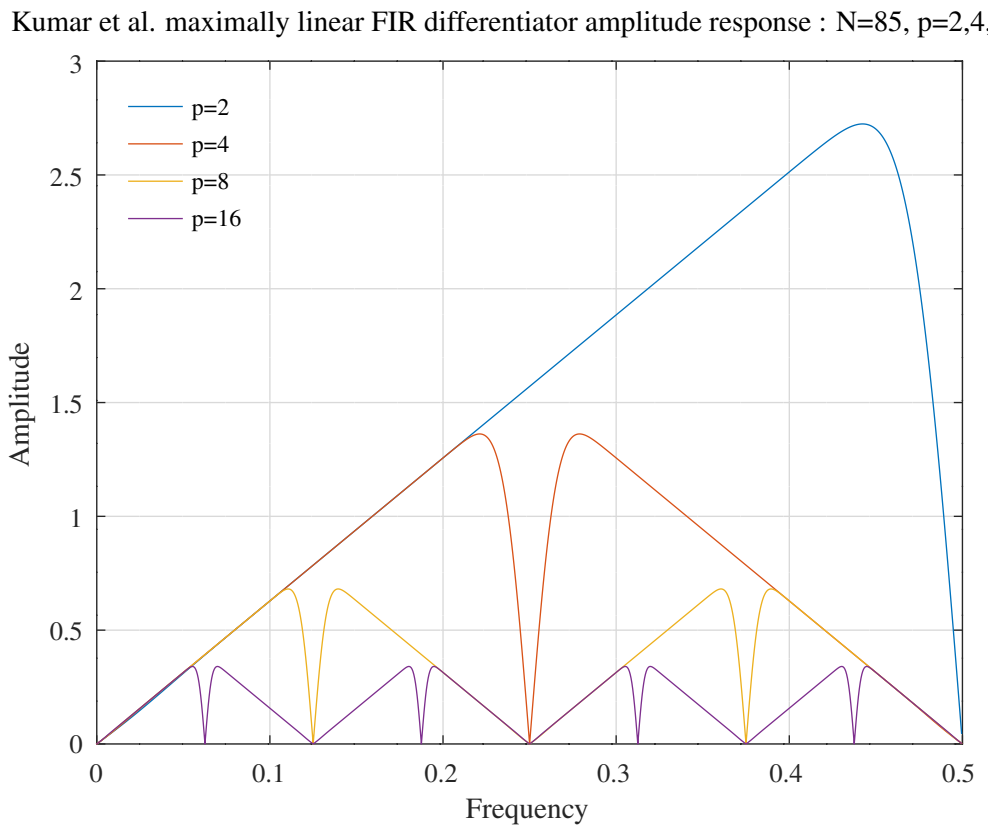



Figure N.64: Amplitude responses of FIR differentiators that are maximally-linear at $\frac{\pi}{p}$ with length $N = 101$ and $p = 2, 4, 8$ and 16 designed by the method of *Kumar et al.* [23] and with the recursive calculation of the $d_k$ performed by the method of *Purczyński* and *Pawelczak* [87, Equation 7]

### ***Kumar et al.*** **closed form design of FIR low-pass differentiators that are maximally-linear at** $\omega = \frac{\pi}{p}$

*Kumar et al.* [23, Equation 9][n] approximate the amplitude response of an FIR differentiator that is maximally-linear at $\omega = \frac{\pi}{p}$ by:

$$|H(\omega)| = \frac{\pi}{2p} \sum_{k=1}^{m} d_{2k-1} \sin\left((2k-1) \cdot \frac{p}{2} \cdot \omega\right) + \frac{1}{2p} \sum_{k=1}^{m} d_{2k} \sin\left(2k \cdot \frac{p}{2} \cdot \omega\right)$$

where $p$ is a positive integer, $m = \frac{n}{2}$, $n = \frac{N-1}{2}$, $n$ is even, and $N$ is the length of the filter, assumed to be odd. They show a realisation of the filter [23, Figure 1]. *Purczyński* and *Pawelczak* solve the resulting system of constraint equations algebraically to find the following recursive closed-form solutions for the coefficients [87, Equation 7]:

$$d_1 = \frac{m}{4^{2m-1}} \begin{pmatrix} 2m \\ m \end{pmatrix}^2$$
$$d_{2k-1} = d_{2k-3} \cdot \frac{2k-3}{2k-1} \cdot \frac{m-k+1}{m+k-1} \quad k = 2, \ldots, m$$

and

$$d_2 = -\frac{2m}{m+1}$$
$$d_{2k} = d_{2k-2} \cdot \frac{k-1}{k} \cdot \frac{m-k+1}{m+k} \quad k = 2, \ldots, m$$

The Octave script *purczynskiFIRantisymmetric_flat_differentiator_test.m* calls the Octave function *purczynskiFIRantisymmetric_flat_differentiator* to design FIR differentiators that are maximally flat at $\omega = \frac{\pi}{p}$ where $p$ is a multiple of 2. Figure N.64 shows the amplitude responses of the differentiators for $p = 2, 4, 8$ and 16. Figure N.65 shows the corresponding amplitude errors, $|H_p(\omega)| - \omega$, of the differentiators.

[n] See also *Khan et al.* [71, Equation 1] for design of differentiators that are maximally-linear at $\omega = \frac{\pi}{2}$

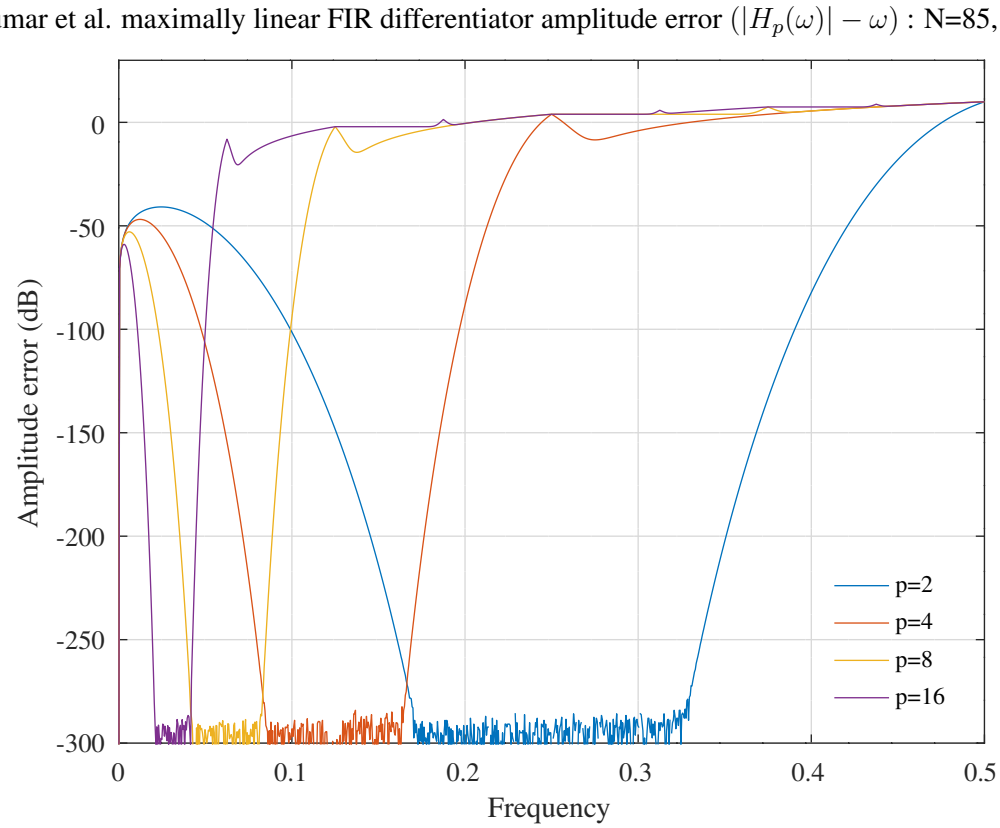



Figure N.65: Amplitude response errors, $|H_p(\omega)| - \omega$, of FIR differentiators that are maximally-linear at $\frac{\pi}{p}$ with length $N = 101$ and $p = 2, 4, 8$ and 16 designed by the method of *Kumar et al.* [23] and with the recursive calculation of the $d_k$ performed by the method of *Purczyński* and *Pawelczak* [87, Equation 7]

## N.9 Linear Matrix Inequality(LMI) design of symmetric FIR filters

This section describes peak-constrained LMI design of even-order symmetric FIR filters described by *Tuan et al.* [61]. They use arguments from convex optimisation to derive results similar to those of *Davidson et al.* [200]. In particular, they derive a version of the *Markov-Lukacs* theorem for trigonometric curves. The primal form of the SDP solution of the LMI FIR frequency constraints requires $\mathcal{O}\left(n^2\right)$ variables. *Tuan et al.* derive a dual form of the SDP with $\mathcal{O}\left(n\right)$ variables.

### N.9.1 The *Markov-Lukacs* theorem

*Després* [18] shows the following version of the *Markov-Lukacs* theorem:

$P_n$ is the set of $n$th order polynomials. The convex set $P_n^+ = \{p\left(x\right) \in P_n \ : \ p\left(x\right) \geq 0 \ \forall \ x \in [0,1]\}$.

$$p\left(x\right) \in P_n^+ \Leftrightarrow \begin{cases} \exists \ a\left(x\right) \in P_k, b\left(x\right) \in P_{k-1} \text{ such that } p\left(x\right) = a\left(x\right)^2 + x\left(1-x\right)b\left(x\right)^2, & n = 2k \\ \exists \ a\left(x\right) \in P_k, b\left(x\right) \in P_k \quad \text{ such that } p\left(x\right) = xa\left(x\right)^2 + \left(1-x\right)b\left(x\right)^2, & n = 2k+1 \end{cases}$$

*Proof*: By induction:

$n = 2$:

$$p\left(x\right) = \left(\left(1-x\right)\sqrt{p\left(0\right)} - x\sqrt{p\left(1\right)}\right)^2 + x\left(1-x\right)b^2$$

$n \in 2\mathbb{N}$:

$$\begin{aligned} p\left(x\right) &= \prod_{k=1}^{\frac{n}{2}} p_k\left(x\right), \ p_k \in P_2^+ \\ &= \prod_{k=1}^{\frac{n}{2}} \left|a_k\left(x\right) + \imath \, b_k\sqrt{x\left(1-x\right)}\right|^2 \\ &= \left|\prod_{k=1}^{\frac{n}{2}} \left(a_k\left(x\right) + \imath \, b_k\sqrt{x\left(1-x\right)}\right)\right|^2 \\ &= \left|a\left(x\right) + \imath b\left(x\right)\sqrt{x\left(1-x\right)}\right|^2 \\ &= a\left(x\right)^2 + x\left(1-x\right)b\left(x\right) \end{aligned}$$

$n \in 2\mathbb{N}+1$:

$$\begin{aligned} xp\left(x\right) = \hat{p}\left(x\right) &= \hat{a}\left(x\right)^2 + x\left(1-x\right)\hat{b}\left(x\right)^2 \\ &= \left(xa\left(x\right)\right)^2 + x\left(1-x\right)b\left(x\right)^2 \\ p\left(x\right) &= xa\left(x\right)^2 + \left(1-x\right)b\left(x\right)^2 \end{aligned}$$

Note that $p\left(x\right)$ is not unique:

$$1 = 1^2 + x\left(1-x\right)0^2 = \left(1-2x\right)^2 + x\left(1-x\right)2^2$$

### N.9.2 Trigonometric curves

*Tuan et al.* [61, Section I] define the trigonometric curve, $C_{a,b}$:

$$\varphi\left(\omega\right) = \left(1, \cos\omega, \ldots, \cos n\omega\right)^\top$$

$$C_{a,b} := \{\varphi(\omega) \ : \ \cos\omega \in [\cos a, \cos b]\} \subset \mathbb{R}^{n+1}$$

and its polar, $C^{\star}_{a,b}$:

$$C^{\star}_{a,b} = \left\{u \in \mathbb{R}^{n+1} \ : \ \langle u, v\rangle \geq 0 \ \forall \ v \in C_{a,b}\right\}$$

A linear constraint on the variable $x \in \mathbb{R}^m$ is:

$$Ax + d \in C^{\star}_{a,b}, \ A \in \mathbb{R}^{(n+1)\times m}, \ d \in \mathbb{R}^{n+1}$$

or:

$$\langle Ax + d, \varphi_n(\omega)\rangle \geq 0, \ \cos\omega \in [\cos a, \cos b]$$

### N.9.3 Moment matrix of trigonometric curves

Let $\phi_k(t) = \left(1, t, \ldots, t^k\right)^{\top}$. The $k$-th order moment matrix of $\phi_k(t)$ is:

$$\begin{aligned}\mathcal{M}_k(t) &= \phi_k(t)\,\phi_k^{\top}(t) \\ &= \begin{bmatrix} 1 & t & \cdots & t^k \\ t & t^2 & \cdots & t^{k+1} \\ \cdots & \cdots & \cdots & \cdots \\ t^k & t^{k+1} & \cdots & t^{2k} \end{bmatrix}\end{aligned}$$

Similarly, define the matrix $\mathcal{T}_k(\omega)$:

$$\begin{aligned}\mathcal{T}_k(\omega) &= \varphi_k(\omega)\,\varphi_k^{\top}(\omega) \\ &= \begin{bmatrix} 1 & \cos\omega & \cdots & \cos k\omega \\ \cos\omega & \frac{1}{2}(1+\cos 2\omega) & \cdots & \frac{1}{2}(\cos(k-1)\omega + \cos(k+1)\omega) \\ \cdots & \cdots & \cdots & \cdots \\ \cos k\omega & \frac{1}{2}(\cos(k-1)\omega + \cos(k+1)\omega) & \cdots & \frac{1}{2}(1+\cos 2k\omega) \end{bmatrix}\end{aligned}$$

The matrix $T_k(y)$ is created from $\mathcal{T}_k(\omega)$ with a change of variable, $\cos l\omega \to y_l$:

$$T_k(y) = \begin{bmatrix} y_0 & y_1 & \cdots & y_k \\ y_1 & \frac{1}{2}(y_0 + y_2) & \cdots & \frac{1}{2}(y_{k-1} + y_{k+1}) \\ \cdots & \cdots & \cdots & \cdots \\ y_k & \frac{1}{2}(y_{k-1} + y_{k+1}) & \cdots & \frac{1}{2}(y_0 + y_{2k}) \end{bmatrix}$$

For convenience, also define:

$$\begin{aligned}\mathcal{T}_{l,k}(\omega) &= \cos l\omega \ \mathcal{T}_k(\omega) \\ T_{l,k}(y) &= T_k(y_l, y_{l+1}, \ldots, y_{l+2k}) \\ &= \begin{bmatrix} y_l & \cdots & & \\ \frac{1}{2}\left(y_{|l-1|} + y_{l+1}\right) & \frac{1}{4}\left(y_{|l-2|} + 2y_l + y_{l+2}\right) & \cdots & \\ \cdots & \cdots & \cdots & \cdots \\ \frac{1}{2}\left(y_{|l-k|} + y_{l+k}\right) & \frac{1}{4}\left(y_{|l-k-1|} + y_{|l-k+1|} + y_{|l+k-1|} + y_{l+k+1}\right) & \cdots & \frac{1}{4}\left(y_{|l-2k|} + 2y_l + y_{l+2k}\right) \end{bmatrix}\end{aligned}$$

*Tuan et al* [61, Appendix I] give a basis for $T_{l,k}(y)$ comprised of symmetric matrixes:

$$\left[\mathcal{E}^m_{l,k}\right]_{i,j} = \frac{1}{4}\left(\delta_{m-|i+j-l-2|} + \delta_{m-|i+j+l-2|} + \delta_{m-|i-j-l|} + \delta_{m-|i-j+l|}\right)$$

where $\delta_m$ is the *Kronecker delta function*. Thus:

$$T_{l,k}(y) = \sum_{m=0}^{l+2k} y_m \mathcal{E}^m_{l,k}$$

### N.9.4 A *Markov-Lukacs* theorem for trigonometric curves

*Tuan et al* [60, Appendix 1] prove the following version of the *Markov-Lukacs* theorem:

Any algebraic polynomial, $P(t) = \phi_n^\top(t)\, x$, that is non-negative on $[a, b] \subset (-\infty, \infty)$ can be written as:

$$P(t) = \begin{cases} \langle X, \mathcal{M}_k(t)\rangle + (b-t)(t-a)\langle Z, \mathcal{M}_{k-1}(t)\rangle, & n = 2k \\ (t-a)\langle X, \mathcal{M}_k(t)\rangle + (b-t)\langle Z, \mathcal{M}_{k-1}(t)\rangle, & n = 2k+1 \end{cases} \tag{N.19}$$

where there exist $X, Z \succeq 0$.

*Tuan et al* [61, Theorem 2] prove the following version of the *Markov-Lukacs* theorem for trigonometric polynomials:

Any algebraic polynomial, $P(\omega) = \varphi_n^\top(\omega)\, x$, that is non-negative on $[\cos a, \cos b]$ can be written as:

$$P(\omega) = \begin{cases} \langle X, \mathcal{T}_k(\omega)\rangle + \langle Z, \mathcal{F}_k^{a,b}(\omega)\rangle, & n = 2k \\ \langle \cos b\, Z - \cos a\, X, \mathcal{T}_k(\omega)\rangle + \langle X - Z, \mathcal{T}_{1,k1}(\omega)\rangle, & n = 2k+1 \end{cases} \tag{N.20}$$

where there exist $X, Z \succeq 0$ and

$$\mathcal{F}_k^{a,b}(\omega) = (\cos b + \cos a)\,\mathcal{T}_{1,k-1}(\omega) - \frac{1}{2}\mathcal{T}_{2,k-1}(\omega) - \left(\frac{1}{2} + \cos a \cos b\right)\mathcal{T}_{k-1}(\omega)$$

The Chebyshev polynomials of the first kind are:

$$\cos l\omega = \sum_{m=0}^{l} b_{lm} \cos^m \omega$$

Hence there is a triangular, non-singular transformation, $B_k$, such that:

$$\varphi_k(\omega) = B_k \phi_k(\cos\omega)$$

and:

$$\mathcal{M}_k(\cos\omega) = B_k^{-1}\mathcal{T}_k(\omega)\, B_k^{-\top}$$

Equation N.20 is found by substitution of $\mathcal{M}_k(\cos\omega)$ into Equation N.19.

*Tuan et al* [61, Lemma 1] prove the following lemma:

If $P(\omega) = \varphi_n^\top(\omega)\, x$ can be represented as shown in Equation N.20, then for every $y = (y_0, y_1, \ldots, y_n)^\top \in \mathbb{R}^{n+1}$:

$$y^\top x = \begin{cases} \langle X, T_k(y)\rangle + \langle Z, F_k^{a,b}(y)\rangle, & n = 2k \\ \langle X, T_{1,k}(y) - \cos a\, T_k(y)\rangle + \langle Z, \cos b\, T_k(y) - T_{1,k}(y)\rangle, & n = 2k+1 \end{cases}$$

where $X, Z \succeq 0$ and:

$$F_k^{a,b}(y) = (\cos b + \cos a)\, T_{1,k-1}(y) - \frac{1}{2}T_{2,k-1}(y) - \left(\frac{1}{2} + \cos a \cos b\right) T_{k-1}(y)$$

### N.9.5 The conic hull of $C_{a,b}$

The *convex hull* of a set $C \subset \mathbb{R}^n$ is the smallest convex set in $\mathbb{R}^n$ that contains $C$. Likewise, The *conic hull* of a set $C \subset \mathbb{R}^n$ is the smallest cone in $\mathbb{R}^n$ that contains $C$. The polar set of $C$ is the cone $C^\star = \{x : \langle x, y\rangle \geq 0\ \forall\, y \in C\}$. *Tuan et al* [61, Theorem 3] prove the following version of the *Markov-Lukacs* theorem for the conic hull of trigonometric polynomials:

For the variables $y = (y_0, y_1, \ldots, y_{2k}) \in \mathbb{R}^{n+1}$ define the following LMI constraints:

$$\begin{cases} T_k(y) \succeq 0, F_k^{a,b}(y) \succeq 0, & n = 2k \\ \cos b\, T_k(y) \succeq T_{1,k}(y) \succeq \cos a\, T_k(y), & n = 2k+1 \end{cases} \tag{N.21}$$

The convex hull of the set $C_{a,b}$ is characterised by the LMI constraints:

$$\text{cohull}\, C_{a,b} = \{(y_0, y_1, \cdots, y_n) \ : \ \text{Equation } N.21,\ y_0 = 1\}$$

The conic hull of $C_{a,b}$ is defined by:

$$\text{cone}\, C_{a,b} = \{(y_0, y_1, \cdots, y_n) \ : \ \text{Equation } N.21\}$$

Substituting the $\mathcal{E}_{l,k}^m$ basis matrixes [61, Theorem 4]:

The trigonometric polynomial, $P(\omega) = \sum_{m=0}^{n} x_m \cos m\omega$ is non-negative on $[\cos a, \cos b]$ if-and-only-if there are positive semi-definite matrixes $X$ and $Z$ such that:

$$x_m = \begin{cases} \langle X, \mathcal{E}_{0,k}^m \rangle + \langle Z, \mathcal{F}_{m,k}^{a,b} \rangle, & n = 2k \\ \langle X, \mathcal{E}_{1,k}^m \rangle - \cos a\, \mathcal{E}_{0,k}^m + \langle Z, \cos b\, \mathcal{E}_{0,k}^m - \mathcal{E}_{1,k}^m \rangle, & n = 2k+1 \end{cases} \tag{N.22}$$

When $n = 2k$:

$$\mathcal{F}_{m,k}^{a,b} = (\cos b + \cos a)\, \mathcal{E}_{1,k-1}^m - \frac{1}{2}\mathcal{E}_{2,k-1}^m - \left(\frac{1}{2} + \cos a \cos b\right) \mathcal{E}_{0,k-1}^m$$

$$F_k^{a,b}(y) = \sum_{m=0}^{2k} y_m \mathcal{F}_{m,k}^{a,b}$$

If $e_m$ is the unit vector in $\mathbb{R}^{n+1}$ with 1 at the $m$-th component, then the LMI constraint over $[\cos a_i, \cos b_i]$ is:

$$e_{m+1}^\top (A_i x + d_i) = \begin{cases} \langle X_i, \mathcal{E}_{0,k}^m \rangle + \langle Z_i, \mathcal{F}_{m,k}^{a_i,b_i} \rangle, & n = 2k \\ \langle X_i, \mathcal{E}_{1,k}^m - \cos a_i\, \mathcal{E}_{0,k}^m \rangle + \langle Z_i, \cos b_i\, \mathcal{E}_{0,k}^m - \mathcal{E}_{1,k}^m \rangle, & n = 2k = 1 \end{cases} \tag{N.23}$$

where $X_i, Z_i \succeq 0$.

### N.9.6 Optimisation of the dual of a convex quadratic objective function

The convex quadratic optimisation problem over a set of trigonometric polynomials is:

$$\begin{aligned} \textbf{minimise} \quad & x^\top Q x + q^\top x \\ \textbf{subject to} \quad & \text{Equation } N.23 \end{aligned}$$

where $Q \succeq 0$. The dual problem is :

$$\begin{aligned} \max_{y^i \in C_i} \min_x \left[ x^\top Q x + q^\top x - \sum_i (A_i x + d_i)^\top y^i \right] &= \max_{y^i \in C_i} \min_x \left[ x^\top Q x + x^\top \left( q - \sum_i A_i^\top y^i \right) - \sum_i d_i^\top y^i \right] \\ &= \max_{y^i \in C_i} \left[ -\sum_i d_i^\top y^i - \frac{1}{4} \left( q - \sum_i A_i^\top y^i \right)^\top Q^{-1} \left( q - \sum_i A_i^\top y^i \right) \right] \end{aligned}$$

Using Equation N.21 the dual problem becomes:

$$\begin{aligned} \textbf{maximise} \quad & \left[ -\sum_i d_i^\top y^i - \frac{1}{4} \left( q - \sum_i A_i^\top y^i \right)^\top Q^{-1} \left( q - \sum_i A_i^\top y^i \right) \right] \\ \textbf{subject to} \quad & \begin{cases} T_k(y^i) \succeq 0, F_k^{a_i,b_i}(y^i) \succeq 0, & n = 2k \\ \cos b_i\, T_k(y^i) \succeq T_{1,k}(y^i) \succeq \cos a_i\, T_k(y^i), & n = 2k+1 \end{cases} \end{aligned} \tag{N.24}$$

In SDP form:

$$\begin{array}{ll} \textbf{maximise} & -\nu - \sum_i d_i^\top y^i \\ \textbf{subject to} & \begin{bmatrix} \nu & \left(q - \sum_i A_i^\top y^i\right)^\top \\ \left(q - \sum_i A_i^\top y^i\right) & 4Q \end{bmatrix} \succeq 0 \\ & \text{Equation } N.24 \end{array}$$

The optimal solution of the dual problem, $x_\star$ and $y_\star^i$, requires finding the $n+1$ scalar variables for each constraint, $y_\star^i$, that satisfy:

$$\sum_i \left(A_i x_\star + d_i\right)^\top y_\star^i = 0$$

$$x_\star = -\frac{1}{2} Q^{-1} \left(q - \sum_i A_i^\top y_\star^i\right)$$

### N.9.7 Example of LMI design of a low pass symmetric FIR filter

The low pass filter design primal problem is:

$$\begin{array}{rl} \textbf{minimise} & x^\top Q x + q^\top x \\ \textbf{subject to} & x + (\delta_p - 1)\, e_1 \in C^\star_{\omega_p,0} \\ & -x + (\delta_p + 1)\, e_1 \in C^\star_{\omega_p,0} \\ & x + \delta_s e_1 \in C^\star_{\pi,\omega_s} \\ & -x + \delta_s e_1 \in C^\star_{\pi,\omega_s} \end{array}$$

The Octave script *directFIRsymmetric_sdp_lowpass_test.m* uses YALMIP and SeDuMi to solve the dual problem of designing a low pass filter with a specification similar to that of the example of *Tuan et al.* [61, Figure 3]:

```
M=200 % M+1 distinct coefficients
fap=0.03 % Amplitude pass band edge
dBap=1 % Amplitude pass band peak-to-peak ripple(dB)
Wap=1 % Amplitude pass band weight
Wat=0 % Amplitude transition band weight
fas=0.0358 % Amplitude stop band edge
dBas=60 % Amplitude stop band peak-to-peak ripple(dB)
Was=100 % Amplitude stop band weight
```

Figure N.66 shows the zero-phase amplitude responses in the pass band and the stop band. I suspect that the response shown by *Tuan et al.* was obtained by using a more "aggressive" specification and by ignoring the consequent numerical problems reported by SeDuMi. SDPT3 fails with both of the example filter specifications [61, Figures 3 and 4]. In addition, I found that calculating $x_\star$ with the Octave left division operator failed.

The Octave script *mcclellanFIRsymmetric_lowpass_alternate_test.m* uses the *Parks-McClellan* algorithm [206] implemented in the Octave function *mcclellanFIRsymmetric* to design a filter similar to that of *Tuan et al.*'s Figure 3:

```
M=200 % Filter order is 2*M
fap=0.03 % Amplitude pass band edge
fas=0.0358 % Amplitude stop band edge
K=50 % Stop band weight
ngrid=4000 % Number of frequency grid points in [0,0.5]
```

Figure N.67 shows the zero-phase pass band and stop band amplitude responses.

The Octave script *selesnickFIRsymmetric_lowpass_alternate_test.m* uses *Hofstetter*'s algorithm [41] with the *Selesnick-Burrus* modification [74] implemented in the Octave function *selesnickFIRsymmetric_lowpass* to design an FIR filter similar to that of *Tuan et al.*'s Figure 4:

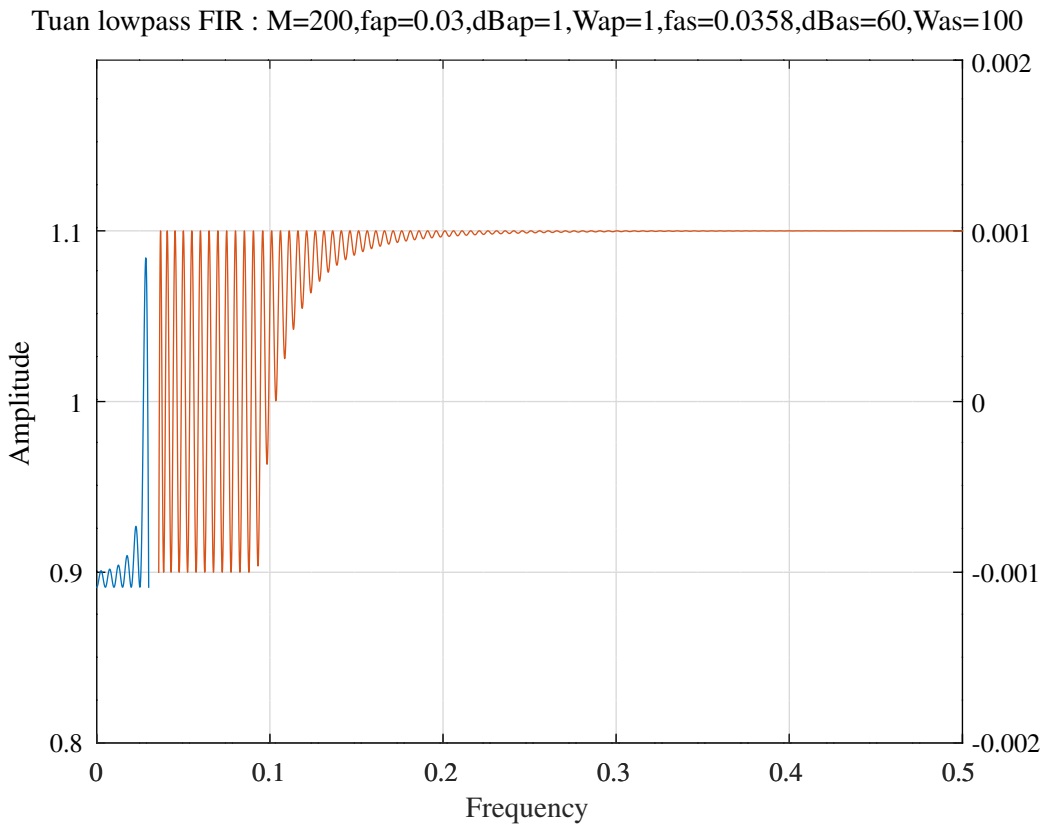



Figure N.66: Pass band and stop band zero-phase amplitude responses of an $M = 200$ symmetric low pass FIR filter designed with the LMI procedure of *Tuan et al.* [61, Figure 3].

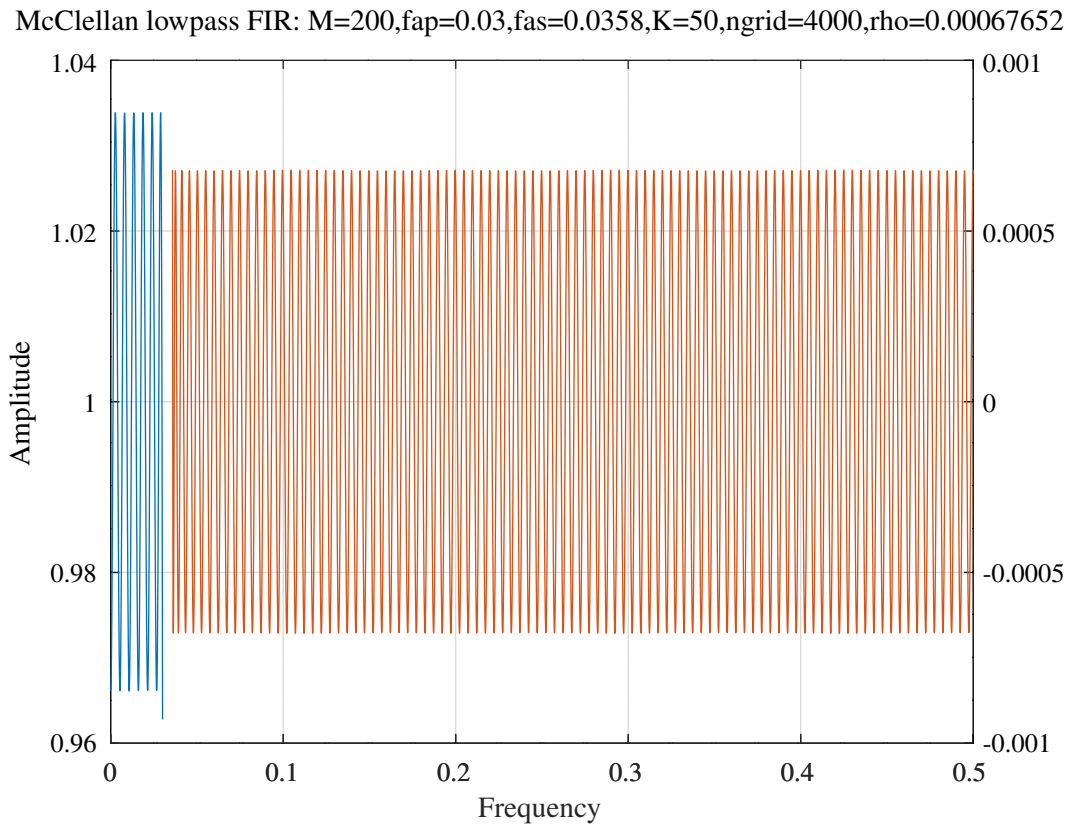



Figure N.67: Pass band and stop band responses of a mini-max FIR low-pass filter designed with the *Parks-McClellan* algorithm. The filter specification is similar to that of the example of *Tuan et al.* [61, Figure 3]

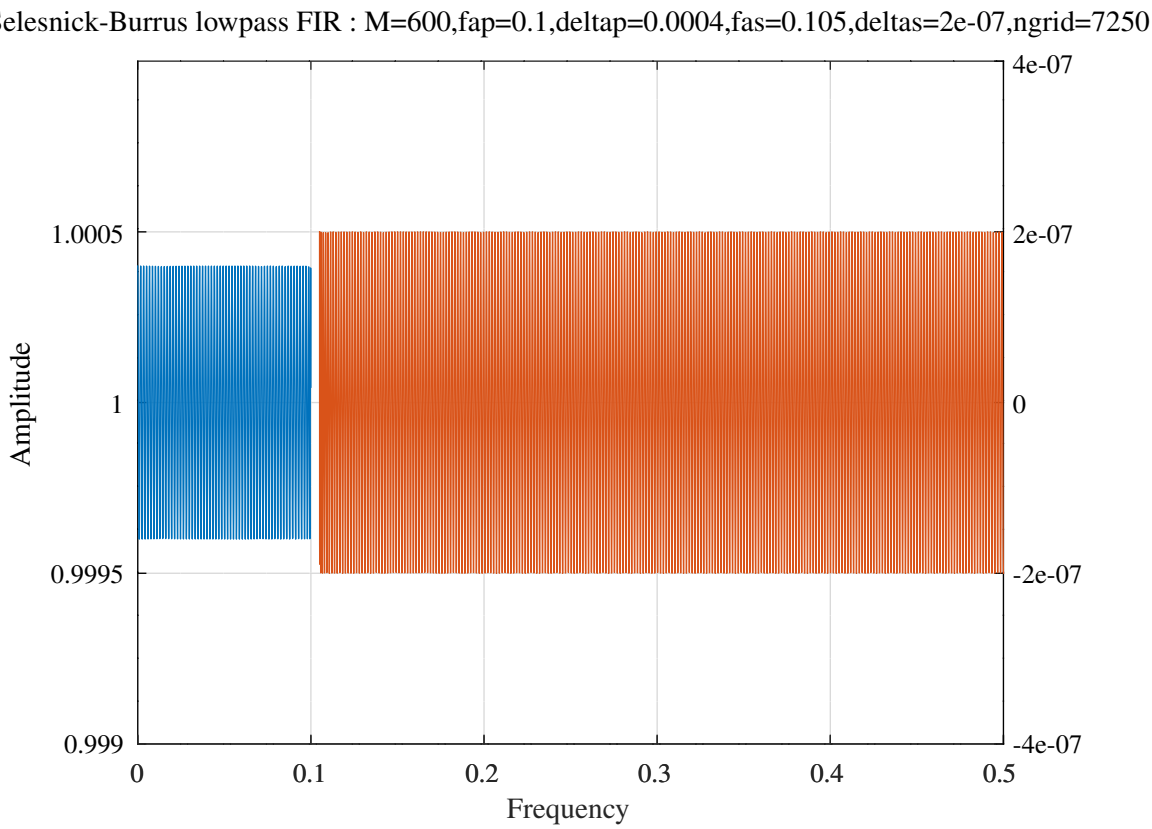



Figure N.68: Pass band and stop band responses of a mini-max FIR low-pass filter designed with the *Selesnick-Burrus* modification to *Hofstetter*'s algorithm. The filter specification is similar to the example of *Tuan et al.* [61, Figure 4]

```
M=600 % Filter order is 2*M
fap=0.1 % Amplitude pass band edge
deltap=0.0004 % Amplitude pass band ripple
fas=0.105 % Amplitude stop band edge
deltas=2e-07 % Amplitude stop band ripple
ngrid=7250 % Number of frequency grid points in [0,0.5]
maxiter=200 % Maximum number of iterations
tol=1e-12 % Tolerance on convergence
```

Figure N.68 shows the zero-phase pass band and stop band amplitude responses. The transition frequency is set to $f_{as}$. The actual pass band edge frequency is $f_{ap}$ =0.100053 .

The *Parks-McClellan* algorithm and *Hofstetter*'s algorithm with the *Selesnick-Burrus* modification are much faster and more stable than my implementation of *Tuan et al.*'s LMI design technique.

### N.9.8 Example of LMI design of a band pass symmetric FIR filter

The Octave script *directFIRsymmetric_sdp_bandpass_test.m* uses the LMI procedure of *Tuan et al.* to design a band pass filter with a specification similar to that of the example of *Pipeleers et al.* [51, Figure 4]:

```
M=15 % M+1 distinct coefficients, order 2*M
fasl=0.05 % Amplitude lower stop band edge
fapl=0.15 % Amplitude lower pass band edge
fapu=0.25 % Amplitude upper pass band edge
fasu=0.35 % Amplitude upper stop band edge
deltap=0.000154 % Amplitude pass band peak-to-peak ripple
deltas=0.005 % Amplitude stop band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=0 % Amplitude transition band weight
Was=1 % Amplitude stop band weight
```

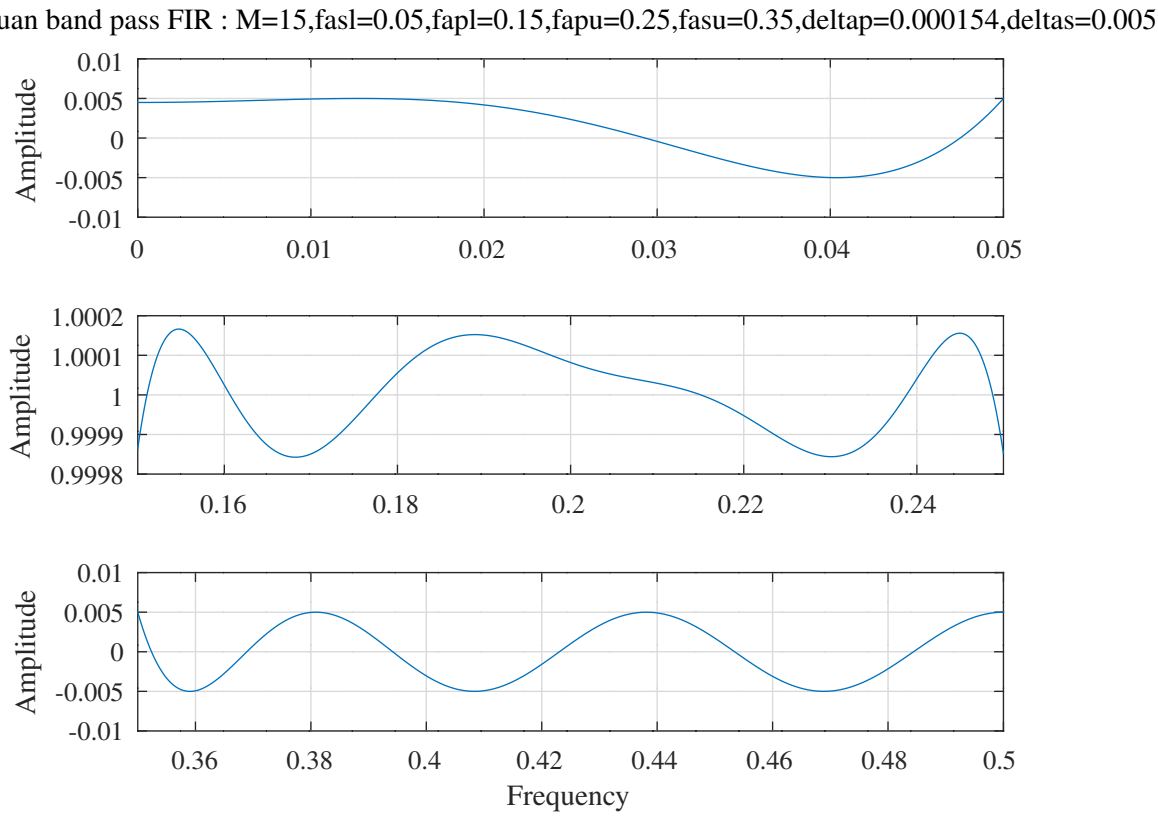



Figure N.69: Pass band and stop band zero-phase amplitude responses of an $M = 15$ symmetric band pass FIR filter designed with the LMI procedure of *Tuan et al.* [61].

The distinct filter coefficients are:

```
hM15 = [  -0.0050212398,  -0.0004914857,  -0.0030950469,  -0.0113105471, ...
           0.0059868031,   0.0160489627,  -0.0010735428,   0.0273232937, ...
           0.0504467150,  -0.0193569744,  -0.0283613440,   0.0179456384, ...
          -0.1414980697,  -0.2449774887,   0.1224896438,   0.4343808766 ]';
```

Figure N.69 shows the zero-phase amplitude responses in the pass band and the stop bands.

For comparison, the Octave script *directFIRsymmetric_socp_slb_bandpass_test.m* designs a symmetric FIR filter using the PCLS algorithm of *Selesnick et al.* [75] with the specification:

```
tol=1e-06 % Tolerance on coef. update
ctol=1e-07 % Tolerance on constraints
maxiter=5000 % SOCP iteration limit
npoints=500 % Frequency points across the band
fapl=0.15 % Amplitude pass band lower edge
fapu=0.25 % Amplitude pass band upper edge
deltap=0.000307 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.35 % Amplitude stop band upper edge
deltas=0.005 % Amplitude stop band peak-to-peak ripple
Wasl=1000 % Amplitude lower stop band weight
Wasu=1000 % Amplitude upper stop band weight
```

The distinct coefficients of this filter are:

```
hM1 = [  -0.00502644,  -0.00049450,  -0.00309351,  -0.01131063, ...
          0.00598537,   0.01605137,  -0.00107514,   0.02730841, ...
          0.05043304,  -0.01936237,  -0.02837561,   0.01793398, ...
         -0.14147613,  -0.24494782,   0.12246831,   0.43432293 ]';
```

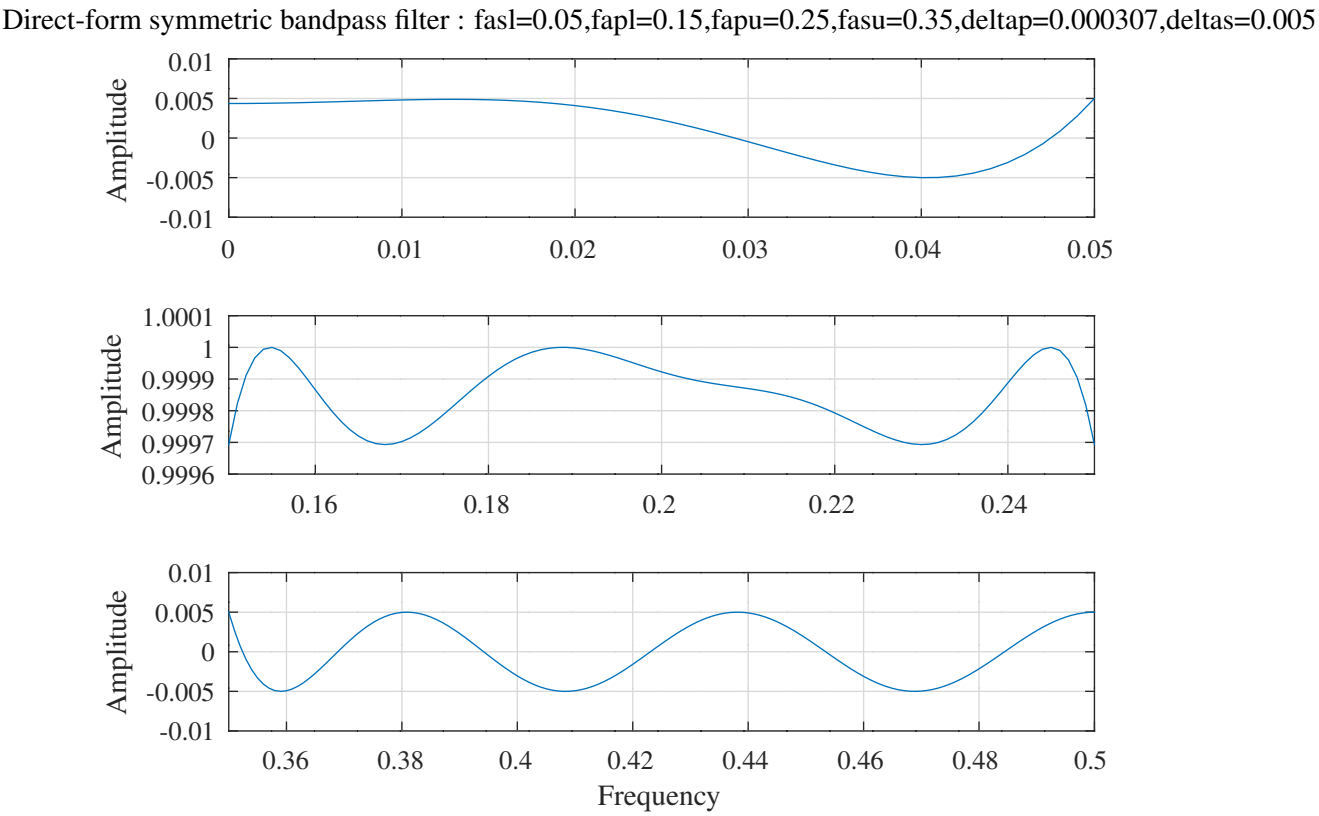



Figure N.70: Zero-phase amplitude response of an $M = 15$ symmetric band pass FIR filter designed with the PCLS optimisation procedure of *Selesnick et al.* [75]

Figure N.70 shows the zero-phase amplitude response.

Similarly, the Octave script *directFIRnonsymmetric_socp_slb_bandpass_test.m* designs a non-symmetric FIR filter using the PCLS algorithm of *Selesnick et al.* with the specification:

```
tol=1e-06 % Tolerance on coefficient update vector
ctol=1e-07 % Tolerance on constraints
n=500 % Frequency points across the band
N=30 % FIR filter order
fapl=0.15 % Pass band squared amplitude lower edge
fapu=0.25 % Pass band squared amplitude upper edge
deltap=0.0005 % Pass band squared amplitude ripple
Wap=1 % Pass band squared amplitude weight
fasl=0.05 % Lower stop band squared amplitude lower edge
fasu=0.35 % Upper stop band squared amplitude upper edge
deltas=0.005 % Stop band squared amplitude response ripple
Watl=0 % Lower transition band squared amplitude weight
Watu=0 % Upper transition band squared amplitude weight
Wasl=1000 % Lower stop band squared amplitude weight
Wasu=1000 % Upper stop band squared amplitude weight
ftpl=0.15 % Pass band group delay response lower edge
ftpu=0.25 % Pass band group delay response upper edge
td=14 % Pass band nominal group delay
tdr=0.02 % Pass band group delay response ripple
Wtp=0.1 % Pass band group delay response weight
```

The coefficients of this filter are:

```
h = [  -0.0019642625,   0.0007578958,  -0.0075787731,  -0.0106375452, ...
        0.0020404034,  -0.0050800933,  -0.0081235923,   0.0399859924, ...
        0.0431671090,  -0.0048672004,   0.0549337448,   0.0496905196, ...
       -0.2311337481,  -0.3094786638,   0.1335790745,   0.4027702014, ...
        0.0945777818,  -0.1576495799,  -0.0532050324,  -0.0068589551, ...
```

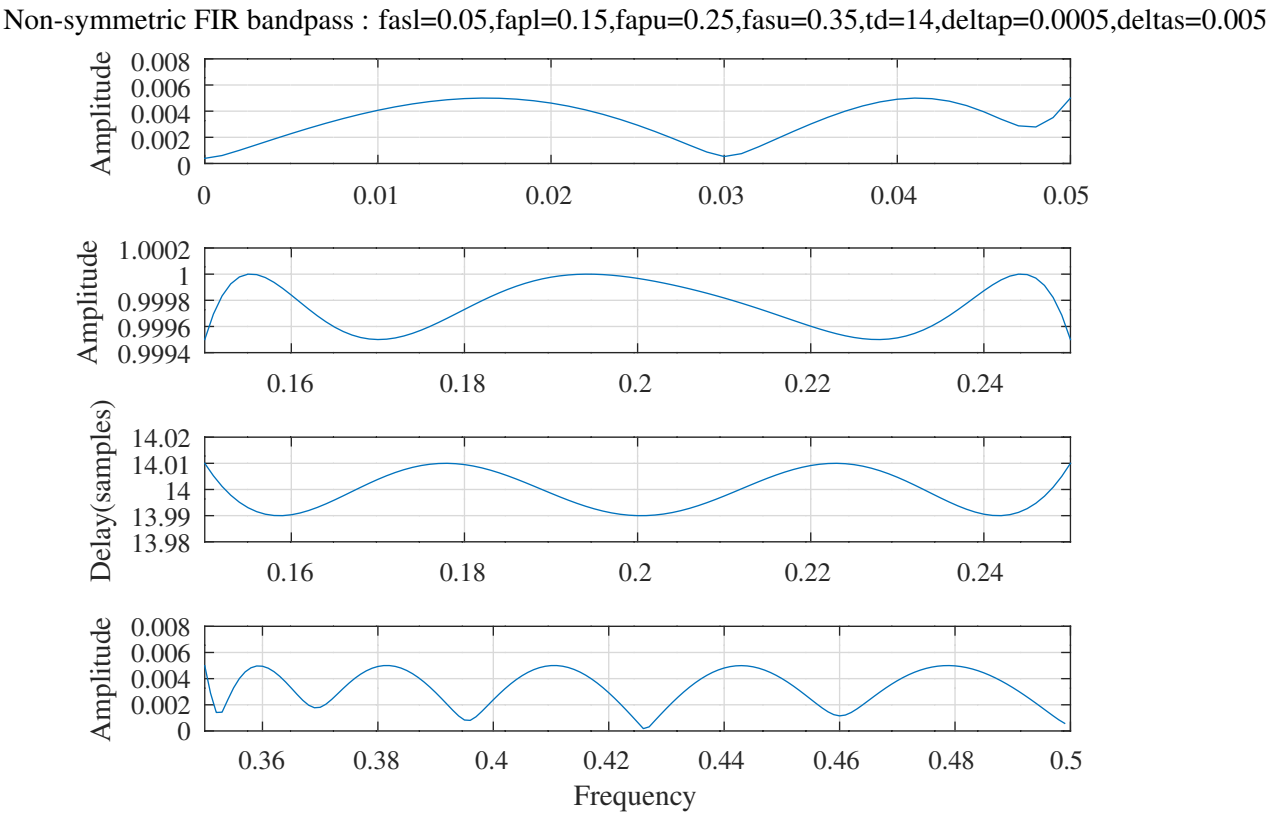



Figure N.71: Pass band amplitude and delay responses and stop band amplitude response of an $M = 15$ non-symmetric band pass FIR filter designed with the PCLS optimisation procedure of *Selesnick et al.* [75]

```
-0.0719163693,  -0.0210123115,   0.0377263407,   0.0066611764, ...
 0.0047149564,   0.0252440369,   0.0055235877,  -0.0077889125, ...
 0.0022898825,  -0.0021626030,  -0.0045963640 ]';
```

Figure N.71 shows the amplitude and delay responses in the pass band and the stop band amplitude response.

For this example the LMI optimisation procedure of *Tuan et al.* is much faster than the PCLS design procedure of *Selesnick et al.* [75].

## N.10 Design of half-band FIR filters

### N.10.1 Vaidyanathan's "TRICK" for the design of FIR half-band filters

*Vaidyanathan* and *Nguyen* [151] describe a method for the design of half-band FIR filters with pass-band edge $f_p$ and length $4M+3$:

1. Design a low-pass FIR filter, $G(z)$ with odd order $2M+1$, pass band edge $2f_p$ and transition band $2f_p$ to $0.5$
2. Construct the half-band FIR filter:

$$h(n) = \begin{cases} \frac{1}{2}g\left(\frac{n}{2}\right), & n = 0, 2, \ldots, 4M+2 \\ 0, & n = 1, 3, \ldots, 4M+1, \quad n \neq 2M+1 \\ \frac{1}{2} & n = 2M+1 \end{cases}$$

The $z$-domain transfer function of $H(z)$ is:

$$H(z) = \frac{G\left(z^2\right) + z^{-(2M+1)}}{2}$$

The Octave script *vaidyanathan_trick_test.m* designs a half-band filter with pass-band edge $fp = 0.24$ and $M = 80$ using *Vaidyanathan*'s "TRICK". The script calls the Octave *remez* function to design a low-pass filter with pass-band edge $2fp = 0.48$ and stop-band edge $0.499$. The *remez* function fails to converge for $M > 81$. Figure N.72 shows the frequency response of the filter.

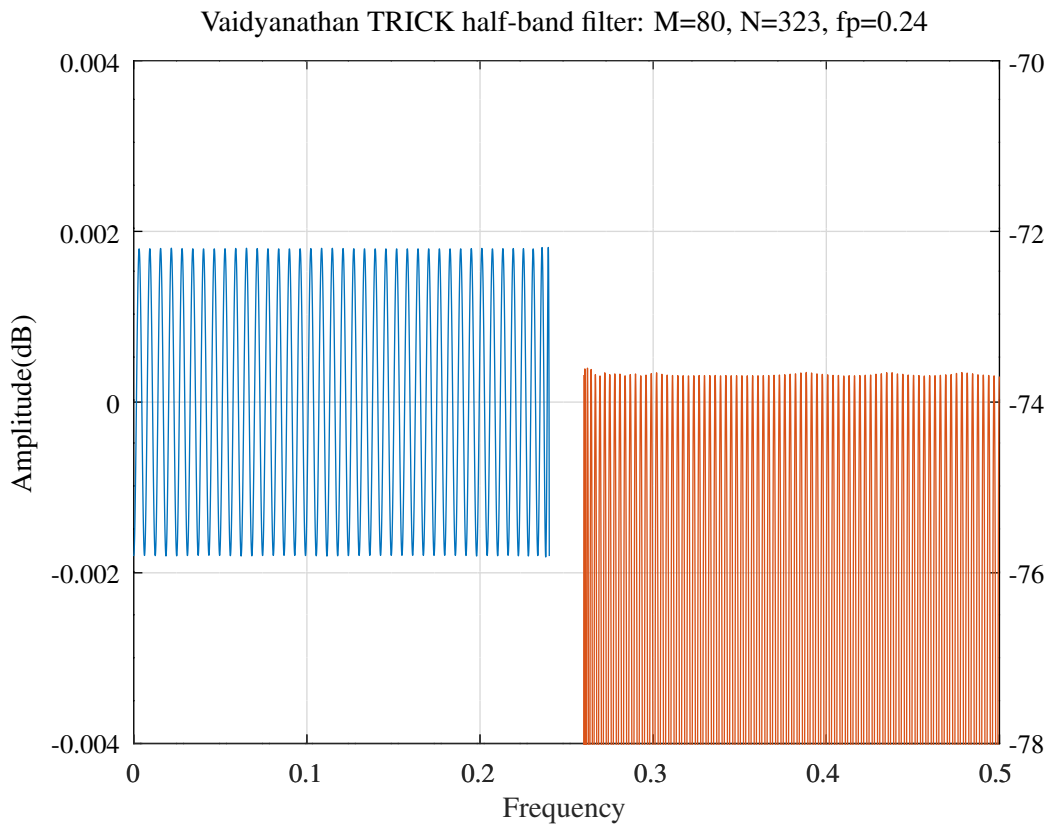



Figure N.72: FIR half-band filter with $fp = 0.24$ and $M = 80$ designed with Vaidyanathan's "TRICK" [151]

### N.10.2 Design of equi-ripple FIR half-band filters

*Zahradník*, *Vlček* and *Unbehauen* [158] describe the closed-form design of almost-equiripple half-band FIR filters. The impulse response, $h$, of a length $4n+3$ half-band FIR filter is defined by the transfer function:

$$H(z) = \sum_{k=0}^{4n+2} h_k z^{-k} \tag{N.25}$$

where:

$$\begin{aligned} h_{2n+1} &= \frac{1}{2} \\ h_{2n+1\pm 2k} &= 0\,, \quad k = 1, \ldots, n \end{aligned}$$

Substituting $v = \frac{1}{2}\left(z + \frac{1}{z}\right)$, the frequency response is:

$$H\left(e^{\imath\omega T}\right) = e^{-\imath(2n+1)\omega T} Q\left(\cos\omega T\right)$$

In particular, the pass-band edge is at $v_p = \cos\omega_p T$. Here $Q(v)$, the *zero-phase frequency response*, is:

$$Q(v) = \frac{1}{2} + \sum_{k=0}^{n} c_{2k+1} T_{2k+1}(v) \tag{N.26}$$

Where the $T_k(v)$ are the Chebyshev polynomials of the first kind and the coefficients, $c_{2k+1}$, of the expansion of $Q$ are the distinct symmetric FIR coefficients of the impulse response of $H$. Appendix C reviews the properties of the Chebyshev polynomials of the first and second kinds. The latter are written $U_k(v)$.

*Vlček* and *Unbehauen* [126, Section IV] show that the equivalent zero-phase response of an IIR filter is:

$$Q(v) = \frac{1}{1 + \varepsilon^2 F^2(v)}$$

The equiripple properties of $F(v)$ correspond to a differential equation that can be solved by elliptic integrals, giving a set of parametric equations for $F$ and $v$ in terms of Jacobian elliptic functions of a complex variable, $u$. With the choice of half-band FIR filter order, the solution of interest here is [126, Equations 33 and 34]:

$$\begin{aligned} F(u) &= \mathrm{cd}\left(\frac{Mu}{\kappa_1}, \kappa_1\right) \\ v &= \mathrm{dn}(u, \kappa) \end{aligned}$$

where $F(v) = \pm\frac{1}{\kappa_1}$ are the stop-band extremal values and $\kappa_1$ is the elliptic modulus [126, Equation 16]. The complex parameter $u$ follows the path shown in [126, Figure 8].

*Zahradník et al.* use the elliptic function identity

$$\mathrm{dn}^2(u, \kappa) = \kappa'^2 + \left(1 - \kappa'^2\right)\mathrm{cn}^2(u, \kappa)$$

to give:

$$x = \frac{2v^2 - 1 - \kappa'^2}{1 - \kappa'^2} = 2\,\mathrm{cn}^2(u, \kappa) - 1$$

Returning to the case of an almost-equiripple half-band FIR filter, *Zahradník et al.* [158, Section II] use the relation between the Chebyshev polynomials of the first and second kinds:

$$\frac{dT_{2n+1}(v)}{dv} = (2n+1)\,U_{2n}(v)$$

to define $G(v) = U_{2n}(v)$ as the generating function for $Q(v)$:

$$Q(v) = \frac{1}{2} + \frac{1}{\mathcal{N}} \int G(v)\,dv$$

where $\mathcal{N}$ is a normalising factor. Apparently by analogy with the equiripple IIR filter, *Zahradník et al.* [158, Equation 13] substitute $x$ into the double angle identity for $U_{2n}(\cos\omega)$[o] and redefine the generating function as:

$$G(v) = U_n\left(\frac{2v^2 - 1 - \kappa'^2}{1 - \kappa'^2}\right) + U_{n-1}\left(\frac{2v^2 - 1 - \kappa'^2}{1 - \kappa'^2}\right)$$

*Zahradník* and *Vlček* [155, Equation 10] modify this to give a better equiripple approximation:

$$G(v) = AU_n\left(\frac{2v^2 - 1 - \kappa'^2}{1 - \kappa'^2}\right) + BU_{n-1}\left(\frac{2v^2 - 1 - \kappa'^2}{1 - \kappa'^2}\right)$$

*Zahradník* and *Vlček* [155, Equations 11 to 14] provide numerical approximations for $n$, $\kappa'$, $A$ and $B$:

$$\begin{aligned}
n &= \frac{a_s - 18.18840664\omega_p T + 33.64775300}{18.54155181\omega_p T - 29.13196871} \\
\kappa' &= \frac{n\omega_p T - 1.57111377n + 0.00665857}{-1.01927560n + 0.37221484} \\
A &= \left(0.01525753n + 0.03682344 + \frac{9.24760314}{n}\right)\kappa' + 1.01701407 + \frac{0.73512298}{n} \\
B &= \left(0.00233667n - 1.35418408 + \frac{5.75145813}{n}\right)\kappa' + 1.02999650 - \frac{0.72759508}{n}
\end{aligned}$$

where $a_s < 0$ is the stop-band attenuation in $dB$. Alternatively, $A$ and $B$ can be obtained numerically by solving:

$$Q(v_p) = \begin{cases} Q(1), & \text{if n is odd} \\ Q(v_{01}), & \text{if n is even} \end{cases}$$

and:

$$\begin{aligned}
Q(v_{01}) &= 1, \quad \text{if n is odd} \\
Q(1) &= 1, \quad \text{if n is even}
\end{aligned}$$

where $v_{01}$ is the first zero of the generating function $U_{2n}(\mathrm{cn}(u,\kappa))$ (ie: the first extremal value of $Q(v)$ as $1 \to v$) [158, Equation 15]:

$$v_{01} = \sqrt{\kappa'^2 + (1 - \kappa'^2)\cos^2\frac{\pi}{2n+1}}$$

Writing:

$$\mathcal{U}_n(v) = \int U_n\left(\frac{2v^2 - 1 - \kappa'^2}{1 - \kappa'^2}\right) dv$$

the zero-phase transfer function is:

$$Q(v) = \frac{1}{2} + \frac{A}{\mathcal{N}}\mathcal{U}_n(v) + \frac{B}{\mathcal{N}}\mathcal{U}_{n-1}(v)$$

The normalisation factor $\mathcal{N}$ is [155, Equation 17]:

$$\mathcal{N} = \begin{cases} 2\left[A\mathcal{U}_n(v_{01}) + B\mathcal{U}_{n-1}(v_{01})\right], & \text{if } n \text{ is odd} \\ 2\left[A\mathcal{U}_n(1) + B\mathcal{U}_{n-1}(1)\right], & \text{if } n \text{ is even} \end{cases}$$

*Zahradník* and *Vlček* [155, Figures 1 and 2] show plots of the generating function, $G(v)$ and zero-phase frequency response, $Q(v)$ for $n = 20$, $\kappa' = 0.03922835$, $A = 1.08532371$, $B = 0.95360863$ and $\mathcal{N} = 0.55091994$. The Octave script *zahradnik_halfband_test.m* an reproduces these as Figure N.73 and Figure N.74. Figure N.75 shows the corresponding frequency response.

---

o

$$\begin{aligned}
U_{2n}(\cos\omega) &= \frac{\sin[(2n+1)\omega]\cos\omega + \sin\omega\cos[(2n+1)\omega]}{2\sin\omega\cos\omega} + \frac{\sin[(2n+1)\omega]\cos\omega - \sin\omega\cos[(2n+1)\omega]}{2\sin\omega\cos\omega} \\
&= U_n(\cos 2\omega) + U_{n-1}(\cos 2\omega) \\
&= U_n\left(2\cos^2\omega - 1\right) + U_{n-1}\left(2\cos^2\omega - 1\right)
\end{aligned}$$

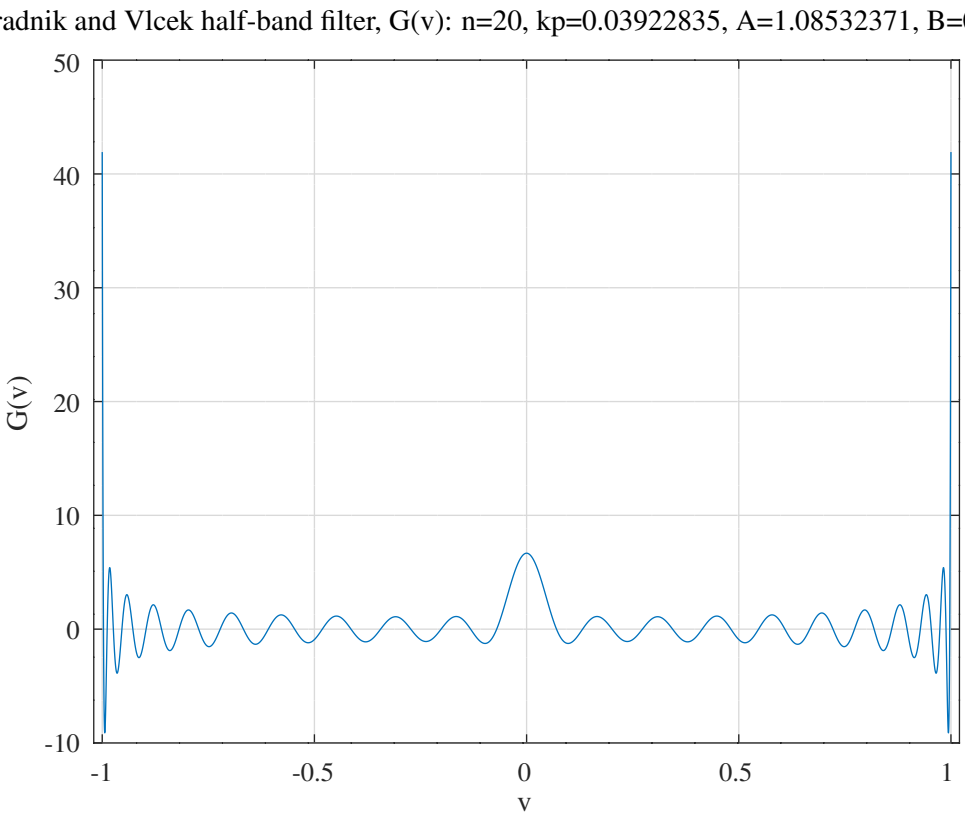



Figure N.73: Equiripple FIR half-band filter generating function, $G(v)$, for $n = 20$, $\kappa' = 0.03922835$, $A = 1.08532371$, $B = 0.95360863$ and $N = 0.55091994$ [155, Figure 1]

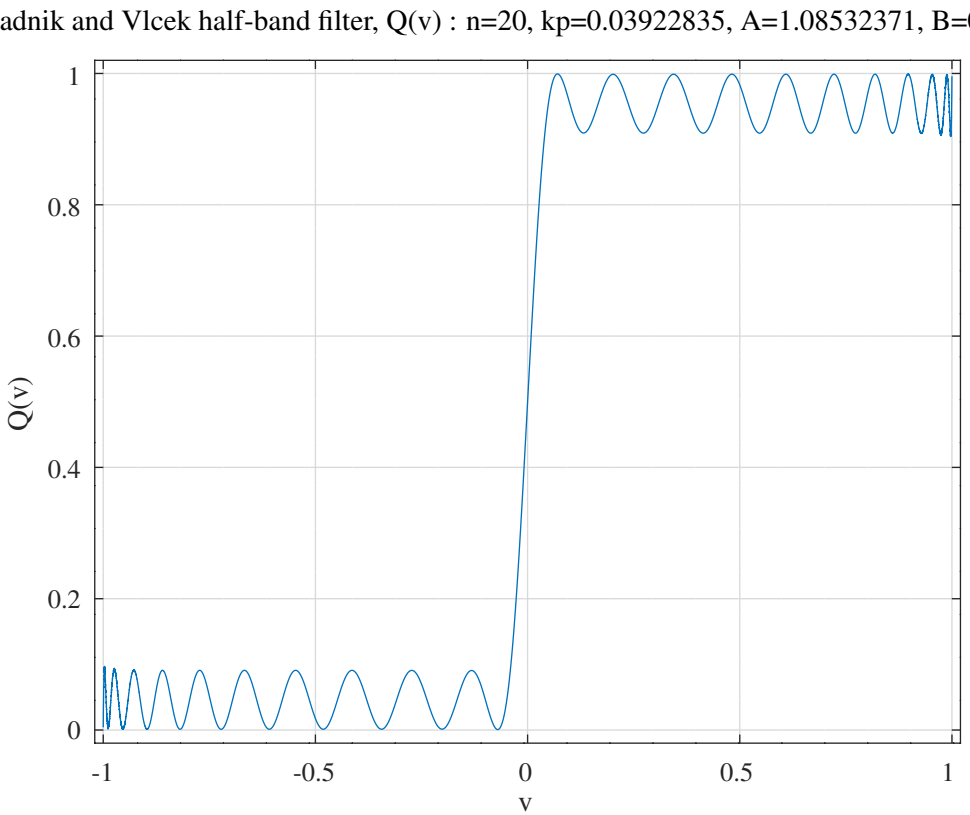



Figure N.74: Equiripple FIR half-band filter zero-phase frequency response, $Q(v)$, for $n = 20$, $\kappa' = 0.03922835$, $A = 1.08532371$, $B = 0.95360863$ and $N = 0.55091994$ [155, Figure 2]

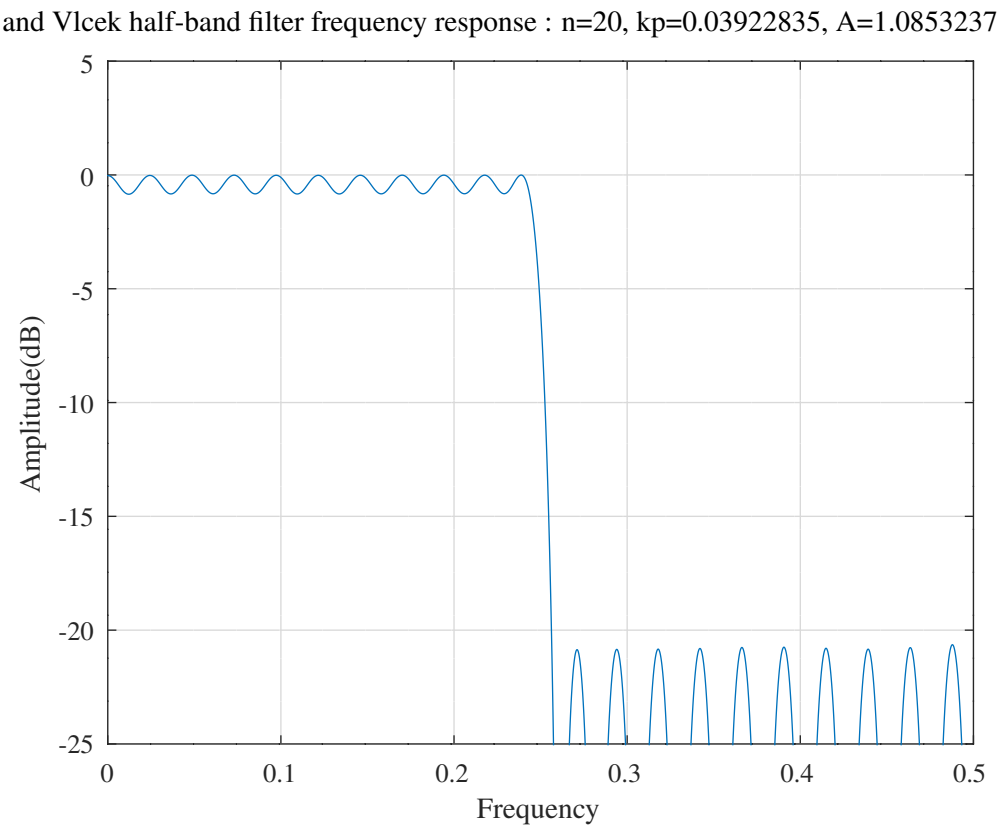



Figure N.75: Equiripple FIR half-band filter frequency response for $n = 20$, $\kappa' = 0.03922835$, $A = 1.08532371$, $B = 0.95360863$ and $N = 0.55091994$

The Chebyshev polynomials of the second kind satisfy the differential equation:

$$\left(1-v^2\right)\frac{d^2U_n\left(v\right)}{dv^2} - 3v\frac{dU_n\left(v\right)}{dv} + n\left(n+2\right)U_n\left(v\right) = 0 \tag{N.27}$$

Substituting $x$ [155, Equation 8]:

$$v\left(v^2-\kappa'^2\right)\left[\left(1-v^2\right)\frac{d^2U_n\left(v\right)}{dv^2} - 3v\frac{dU_n\left(v\right)}{dv}\right] + \left[\left(\kappa'^2+2v^2\right)\left(1-v^2\right)\right]\frac{dU_n\left(v\right)}{dv} + 4v^3n\left(n+2\right)U_n\left(v\right) = 0 \tag{N.28}$$

Equation N.28 is arranged so that its solution is simplified by using Equation N.27. Equation N.28 is solved by the series expansion:

$$U_n\left(\frac{2v^2-1-\kappa'^2}{1-\kappa'^2}\right) = \sum_{l=0}^{n}\alpha_{2l}U_{2l}\left(v\right) \tag{N.29}$$

Integrating this expansion results in an expression of the form shown in Equation N.26. *Zahradník* and *Vlček* [155, Table 1] show an algorithm for the evaluation of the coefficients, $a_l$, of $\mathcal{U}_n$, reproduced here, with a correction to $\alpha_{2n-4}$, as Algorithm N.4.

The Octave script *zahradnik_halfband_test.m* implements the design of an equiripple FIR half-band filter using Algorithm N.4. The pass-band edge is $f_p = 0.240$, the stop-band attenuation is $a_s = -140dB$ and $n = 118$ for a filter length of 475. The value of $n$ given by Equation 11 of *Zahradník* and *Vlček*[155] is increased slightly. The values of $A$ and $B$ are found by a "brute-force" search of all the extremal values in the pass-band. Figure N.76 shows the amplitude response of the half-band FIR filter. The distinct coefficients of the half-band filter are (see Equation N.25):

```
h_distinct = [   0.0000000341,  -0.0000000357,   0.0000000541,  -0.0000000788, ...
                 0.0000001110,  -0.0000001527,   0.0000002058,  -0.0000002725, ...
                 0.0000003557,  -0.0000004584,   0.0000005841,  -0.0000007368, ...
                 0.0000009212,  -0.0000011424,   0.0000014061,  -0.0000017189, ...
                 0.0000020880,  -0.0000025215,   0.0000030285,  -0.0000036188, ...
                 0.0000043034,  -0.0000050945,   0.0000060053,  -0.0000070505, ...
                 0.0000082459,  -0.0000096088,   0.0000111583,  -0.0000129148, ...
                 0.0000149006,  -0.0000171396,   0.0000196578,  -0.0000224831, ...
                 0.0000256455,  -0.0000291772,   0.0000331128,  -0.0000374889, ...
```

**Algorithm N.4** *Zahradník* and *Vlček*'s algorithm [155, Table 1] for the evaluation of the coefficients, $a_l$, of $\mathcal{U}_n$.

**Require:** $n$, $\kappa'$

*Initialisation:*

$\alpha_{2n} = \left(1-\kappa'^2\right)^{-n}$

$\alpha_{2n-2} = -\left[1+2n\kappa'^2\right]\alpha_{2n}$

$\alpha_{2n-4} = -\frac{4n+1+(n-1)(2n-1)\kappa'^2}{2n}\alpha_{2n-2} - \frac{(2n+1)\left[(n+1)\kappa'^2+1\right]}{2n}\alpha_{2n}$

*Body:*

**for** $l = n$ **down to** $3$ **do**

$$\begin{aligned}\alpha_{2l-6} = \{&-\left[3\left(n\left(n+2\right)-\left(l-2\right)l\right)+2l-3+2\left(l-2\right)\left(2l-3\right)\kappa'^2\right]\alpha_{2l-4} \quad \ldots\\ &-\left[3\left(n\left(n+2\right)-\left(l-1\right)\left(l+1\right)\right)+2\left(2l-1\right)+2l\left(2l-1\right)\kappa'^2\right]\alpha_{2l-2} \quad \ldots\\ &-\left[n\left(n+2\right)-\left(l-1\right)\left(l+1\right)\right]\alpha_{2l}\} \;/\; \left[n\left(n+2\right)-\left(l-3\right)\left(l-1\right)\right]\end{aligned}$$

**end for**

*Integration:*

**for** $l = 0$ **to** $n$ **do**

$a_{2l+1} = \frac{\alpha_{2l}}{2l+1}$

**end for**

*Impulse response:*

$h_{2n+1} = 0$

**for** $l = 0$ **to** $n$ **do**

$h_{2n+1\pm(2l+1)} = \frac{a_{2l+1}}{2}$

**end for**

```
0.0000423450,  -0.0000477230,   0.0000536675,  -0.0000602259, ...
0.0000674483,  -0.0000753879,   0.0000841011,  -0.0000936473, ...
0.0001040891,  -0.0001154927,   0.0001279274,  -0.0001414664, ...
0.0001561864,  -0.0001721678,   0.0001894950,  -0.0002082563, ...
0.0002285441,  -0.0002504550,   0.0002740901,  -0.0002995548, ...
0.0003269591,  -0.0003564181,   0.0003880516,  -0.0004219848, ...
0.0004583483,  -0.0004972785,   0.0005389176,  -0.0005834144, ...
0.0006309244,  -0.0006816100,   0.0007356418,  -0.0007931984, ...
0.0008544672,  -0.0009196457,   0.0009889418,  -0.0010625751, ...
0.0011407779,  -0.0012237969,   0.0013118941,  -0.0014053497, ...
0.0015044629,  -0.0016095554,   0.0017209738,  -0.0018390926, ...
0.0019643187,  -0.0020970954,   0.0022379079,  -0.0023872894, ...
0.0025458286,  -0.0027141780,   0.0028930647,  -0.0030833024, ...
0.0032858064,  -0.0035016112,   0.0037318922,  -0.0039779922, ...
0.0042414538,  -0.0045240591,   0.0048278800,  -0.0051553407, ...
0.0055092966,  -0.0058931364,   0.0063109132,  -0.0067675156, ...
0.0072688957,  -0.0078223735,   0.0084370535,  -0.0091244019, ...
0.0098990619,  -0.0107800330,   0.0117924146,  -0.0129700600, ...
0.0143597475,  -0.0160279843,   0.0180726140,  -0.0206437101, ...
0.0239837549,  -0.0285116499,   0.0350187401,  -0.0452008792, ...
0.0634674499,  -0.1059864668,   0.3182709027,   0.5000000000 ];
```

Reconstruct $h$ with:

```
n=118;
h=zeros(1,(4*n)+3);
h(1:2:((2*n)+1))=h_distinct(1:(n+1));
h(end:-2:((2*n)+3))=h_distinct(1:(n+1));
h((2*n)+2)=0.5;
```

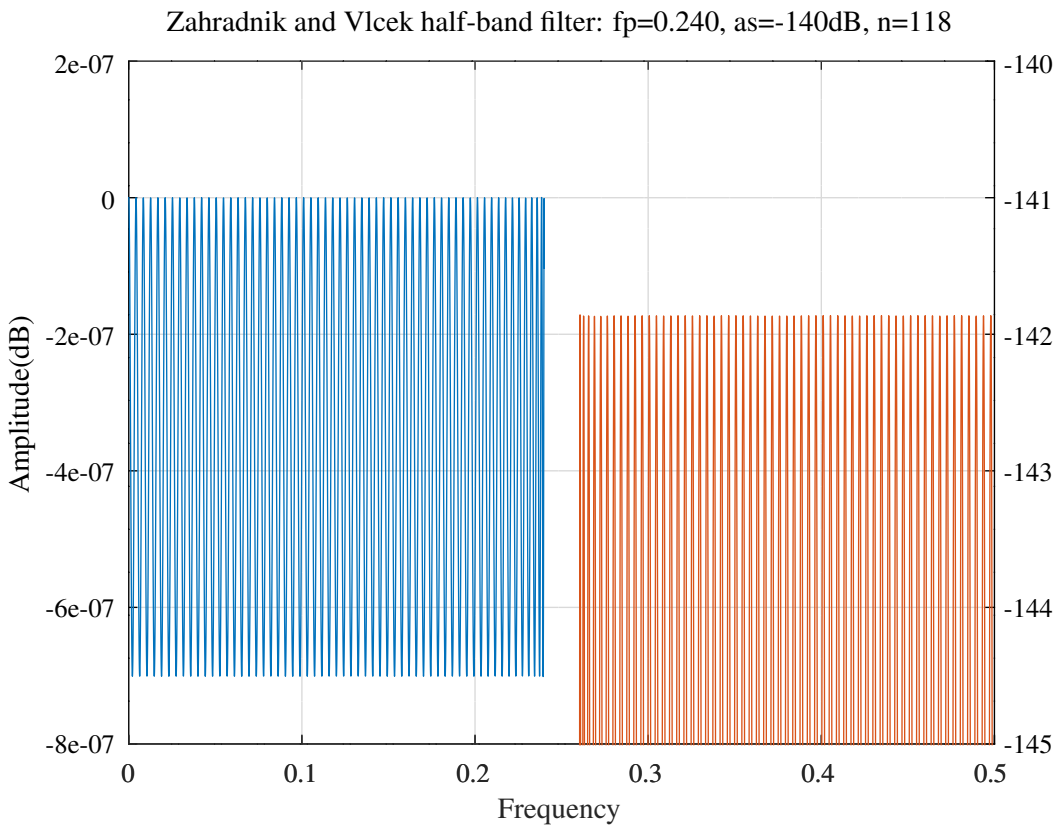



Figure N.76: Equiripple FIR half-band filter with $f_p = 0.240$ and $a_s = -140dB$ and $n = 118$ calculated by Algorithm N.4 [155, Table 1]

**Addendum**

**Working for *Zahradník* and *Vlček* Equation 8 [155]**

The chain rule gives:

$$\frac{dU}{dv} = \frac{dU}{dx}\frac{dx}{dv}$$
$$\frac{d^2U}{dv^2} = \frac{d^2U}{dx^2}\left(\frac{dx}{dv}\right)^2 + \frac{dU}{dx}\frac{d^2x}{dv^2}$$

where:

$$\frac{dx}{dv} = \frac{4v}{1-\kappa'^2}$$
$$\frac{d^2x}{dv^2} = \frac{4}{1-\kappa'^2}$$

Rearranging:

$$x\frac{dU}{dx} = x\left(\frac{dx}{dv}\right)^{-1}\frac{dU}{dv} = \frac{2v^2-1-\kappa'^2}{4v}\frac{dU}{dv}$$
$$\left(1-x^2\right)\frac{d^2U}{dx^2} = \left(1-x^2\right)\left(\frac{dx}{dv}\right)^{-2}\left[\frac{d^2U}{dv^2} - \left(\frac{dx}{dv}\right)^{-1}\frac{d^2x}{dv^2}\frac{dU}{dv}\right]$$
$$= \frac{\left(1-v^2\right)\left(v^2-\kappa'^2\right)}{4v^2}\left[\frac{d^2U}{dv^2} - \frac{1}{v}\frac{dU}{dv}\right]$$

Substituting into Equation N.27 gives Equation N.28.

**Working for *Zahradník* and *Vlček* Table 1 [155, Appendix and Table 1]**

Equations 18 to 25 are experimentally confirmed in the Octave script *chebyshevU_test.m* under the assumption that $U_l(v) = 0$ for $l \leq 0$.

**Working for Equation 25**

$$\begin{aligned}
U_{l+3}(v) &= 2vU_{l+2}(v) - U_{l+1}(v) \\
&= 2v\left(2vU_{l+1}(v) - U_l(v)\right) - U_{l+1}(v) \\
&= 4v^2\left(2vU_l(v) - U_{l-1}(v)\right) - \left(2vU_l(v) - U_{l-1}(v)\right) - U_{l+1}(v) - U_{l-1}(v) \\
&= 8v^3U_l(v) - 4v^2U_{l-1}(v) - 2U_{l+1}(v) - U_{l-1}(v) \\
4v^2U_{l-1}(v) &= 2v\left(2vU_{l-1}(v) - U_{l-2}(v)\right) + 2vU_{l-2}(v) \\
&= 2vU_l(v) - U_{l-1} + U_{l-1} + 2vU_{l-2}(v) - U_{l-3}(v) + U_{l-3}(v) \\
&= U_{l+1}(v) + 2U_{l-1}(v) + U_{l-3}(v)
\end{aligned}$$

Finally,

$$8v^3U_l(v) = U_{l+3}(v) + 3U_{l+1}(v) + 3U_{l-1}(v) + U_{l-3}(v)$$

Substituting Equation N.29 gives the result.

**Working for Equation 23** Equation 23 is simplified by applying Equation 6.

$$\begin{aligned}
-v\left[\left(1-v^2\right)\frac{d^2U_{2l}(v)}{dv^2} - 3v\frac{dU_{2l}(v)}{dv}\right] &= 4l(l+1)vU_{2l}(v) \\
&= 2l(l+1)\left[U_{2l+1}(v) + U_{2l-1}(v)\right] \\
v^3\left[\left(1-v^2\right)\frac{d^2U_{2l}(v)}{dv^2} - 3v\frac{dU_{2l}(v)}{dv}\right] &= -4l(l+1)v^3U_{2l}(v) \\
&= -\frac{1}{2}l(l+1)\left[U_{2l+3}(v) + 3U_{2l+1}(v) + 3U_{2l-1}(v) + U_{2l-3}(v)\right]
\end{aligned}$$

**Working for Equation 24** For the $\kappa'^2$ part of Equation 24:

$$\begin{aligned}
4v^2U_l(v) &= 2v\left(2vU_l(v) - U_{l-1}(v)\right) + 2vU_{l-1}(v) - U_{l-2}(v) + U_{l-2}(v) \\
&= 2vU_{l+1}(v) - U_l(v) + 2U_l(v) + U_{l-2}(v) \\
&= U_{l+2}(v) + 2U_l(v) + U_{l-2}(v) \\
4\left(1-v^2\right)U_l(v) &= -\left[U_{l+2}(v) - 2U_l(v) + U_{l-2}(v)\right]
\end{aligned}$$

Using Equation 20:

$$\begin{aligned}
\left(1-v^2\right)\frac{dU_{2l}(v)}{dv} &= \sum_{p=0}^{l} 4p\left(1-v^2\right)U_{2p-1}(v) \\
&= -\sum_{p=0}^{l} p\left[U_{2p+1}(v) - 2U_{2p-1}(v) + U_{2p-3}(v)\right] \\
&= -\left[\sum_{p=1}^{l+1}(p-1)U_{2p-1}(v) - \sum_{p=0}^{l}2pU_{2p-1}(v) + \sum_{p=-1}^{l-1}(p+1)U_{2p-1}(v)\right] \\
&= -\left[lU_{2l+1}(v) + (l-1)U_{2l-1}(v) - 2lU_{2l-1}(v)\right] \\
&= (l+1)U_{2l-1}(v) - lU_{2l+1}(v)
\end{aligned}$$

Similarly, for the $2v^2\left(1-v^2\right)$ part of Equation 24:

$$\begin{aligned}
2v^2\left(1-v^2\right)\frac{dU_{2l}(v)}{dv} &= 2v^2\sum_{p=0}^{l} 4p\left(1-v^2\right)U_{2p-1}(v) \\
&= 2v^2\left[(l+1)U_{2l-1}(v) - lU_{2l+1}(v)\right] \\
&= \frac{l+1}{2}\left[U_{2l+1}(v) + 2U_{2l-1}(v) + U_{2l-3}(v)\right] - \frac{l}{2}\left[U_{2l+3}(v) + 2U_{2l+1}(v) + U_{2l-1}(v)\right]
\end{aligned}$$

**Working for Equation 26** Collecting terms in $U_{2l+3}(v)$ from Equation 23:

$$-\frac{1}{2}l(l+1)\alpha_{2l}$$

Collecting terms in $U_{2l+3}(v)$ from Equation 24:

$$-\frac{1}{2}l\alpha_{2l}$$

Collecting terms in $U_{2l+3}(v)$ from Equation 25:

$$\frac{1}{2}n(n+2)\alpha_{2l}$$

Giving:

$$U_{2l+3}(v):\frac{1}{2}\left[n(n+2)-l(l+2)\right]\alpha_{2l}$$

**Working for Equation 27** Collecting terms in $U_{2l+1}(v)$ from Equation 23:

$$-\frac{3}{2}l(l+1)\alpha_{2l}+2l(l+1)\kappa'^2\alpha_{2l}$$

Collecting terms in $U_{2l+1}(v)$ from Equation 24:

$$-l\kappa'^2\alpha_{2l}+\frac{1}{2}\left[(l+1)-2l\right]\alpha_{2l}$$

Collecting terms in $U_{2l+1}(v)$ from Equation 25:

$$\frac{3}{2}n(n+2)\alpha_{2l}$$

Giving:

$$U_{2l+1}(v):\frac{1}{2}\left[3(n(n+2)-l(l+2))+2l+1\right]\alpha_{2l}+l(2l+1)\kappa'^2\alpha_{2l}$$

**Working for Equation 28** Collecting terms in $U_{2l-1}(v)$ from Equation 23:

$$-\frac{3}{2}l(l+1)\alpha_{2l}+2l(l+1)\kappa'^2\alpha_{2l}$$

Collecting terms in $U_{2l-1}(v)$ from Equation 24:

$$(l+1)\kappa'^2\alpha_{2l}+\frac{1}{2}\left[2(l+1)-l\right]\alpha_{2l}$$

Collecting terms in $U_{2l-1}(v)$ from Equation 25:

$$\frac{3}{2}n(n+2)\alpha_{2l}$$

Giving:

$$U_{2l-1}(v):\frac{1}{2}\left[3(n(n+2)-l(l+2))+2(2l+1)\right]\alpha_{2l}+(l+1)(2l+1)\kappa'^2\alpha_{2l}$$

**Working for Equation 29** Collecting terms in $U_{2l-3}(v)$ from Equation 23:

$$-\frac{1}{2}l(l+1)\alpha_{2l}$$

Collecting terms in $U_{2l-3}(v)$ from Equation 24:

$$\frac{1}{2}(l+1)\alpha_{2l}$$

Collecting terms in $U_{2l-3}(v)$ from Equation 25:

$$\frac{1}{2}n(n+2)\alpha_{2l}$$

Giving:

$$U_{2l-3}(v):\frac{1}{2}\left[n(n+2)-(l-1)(l+1)\right]\alpha_{2l}$$

**Working for Equations 30 to 33** Changing the summation variables:

$$\begin{aligned}
&\sum_{l=3}^{n+3}\frac{1}{2}\left[n\left(n+2\right)-\left(l-3\right)\left(l-1\right)\right]\alpha_{2l-6}U_{2l-3}\left(v\right) \quad \ldots \\
&+\sum_{l=2}^{n+2}\left\{\frac{1}{2}\left[3\left(n\left(n+2\right)-\left(l-2\right)l\right)+2l-3\right]+\left(l-2\right)\left(2l-3\right)\kappa'^2\right\}\alpha_{2l-4}U_{2l-3}\left(v\right) \quad \ldots \\
&+\sum_{l=1}^{n+1}\left\{\frac{1}{2}\left[3\left(n\left(n+2\right)-\left(l-1\right)\left(l+1\right)\right)+2\left(2l-1\right)\right]+l\left(2l-1\right)\kappa'^2\right\}\alpha_{2l-2}U_{2l-3}\left(v\right) \quad \ldots \\
&+\sum_{l=0}^{n}\frac{1}{2}\left[n\left(n+2\right)-\left(l-1\right)\left(l+1\right)\right]\alpha_{2l}U_{2l-3}\left(v\right) \\
&=0
\end{aligned}$$

Assuming that $U_l\left(v\right)=0$ for $l\leq 0$:

$$\begin{aligned}
&\sum_{l=3}^{n+3}\left[n\left(n+2\right)-\left(l-3\right)\left(l-1\right)\right]\alpha_{2l-6} \quad \ldots \\
&+\sum_{l=2}^{n+2}\left[3\left(n\left(n+2\right)-\left(l-2\right)l\right)+2l-3+2\left(l-2\right)\left(2l-3\right)\kappa'^2\right]\alpha_{2l-4} \quad \ldots \\
&+\sum_{l=2}^{n+1}\left[3\left(n\left(n+2\right)-\left(l-1\right)\left(l+1\right)\right)+2\left(2l-1\right)+2l\left(2l-1\right)\kappa'^2\right]\alpha_{2l-2} \quad \ldots \\
&+\sum_{l=2}^{n}\left[n\left(n+2\right)-\left(l-1\right)\left(l+1\right)\right]\alpha_{2l} \\
&=0
\end{aligned} \tag{N.30}$$

The recurrence is initialised by calculating $\alpha_{2n}$ from the $n+3$ term, $\alpha_{2n-2}$ from the $n+2$ term and $\alpha_{2n-4}$ from the $n+1$ term in Equation N.30. The remaining terms $\alpha_{2n-6}\ldots\alpha_0$ are calculated by backwards recurrence for $l=n,n-1,\ldots,3$. Equation N.30 is implemented in the *body* of Table 1.

**Working for Equations 34 and 35** The $n+3$ term in N.30 gives:

$$\left[n\left(n+2\right)-l\left(l+2\right)\right]\alpha_{2n}=0$$

$\alpha_{2n}$ is a free variable. *Zahradník* and *Vlček* choose $\alpha_{2n}=\left(1-\kappa'^2\right)^{-n}$.

**Working for Equation 36** The $n+2$ term in N.30 gives:

$$\begin{aligned}
\frac{1}{2}\left[n\left(n+2\right)-\left(n-1\right)\left(n+1\right)\right]\alpha_{2n-2}+\left\{\frac{1}{2}\left[3\left(n\left(n+2\right)-n\left(n+2\right)\right)+2n+1\right]+n\left(2n+1\right)\kappa'^2\right\}\alpha_{2n}&=0 \\
\left[n\left(n+2\right)-\left(n-1\right)\left(n+1\right)\right]\alpha_{2n-2}+\left[2n+1+2n\left(2n+1\right)\kappa'^2\right]\alpha_{2n}&=0 \\
\left(2n+1\right)\alpha_{2n-2}+\left[2n+1+2n\left(2n+1\right)\kappa'^2\right]\alpha_{2n}&=0 \\
\alpha_{2n-2}+\left[1+2n\kappa'^2\right]\alpha_{2n}&=0
\end{aligned}$$

**Working for Equation 37** The $n+1$ term in N.30 gives:

$$\begin{aligned}
\frac{1}{2}\left[n\left(n+2\right)-\left(n-2\right)n\right]\alpha_{2n-4}+ \quad \ldots \\
\left\{\frac{1}{2}\left[3\left(n\left(n+2\right)-\left(n-1\right)\left(n+1\right)\right)+2n-1\right]+\left(n-1\right)\left(2n-1\right)\kappa'^2\right\}\alpha_{2n-2}+ \quad \ldots \\
\left\{\frac{1}{2}\left[3\left(n\left(n+2\right)-n\left(n+2\right)\right)+2\left(2n+1\right)\right]+\left(n+1\right)\left(2n+1\right)\kappa'^2\right\}\alpha_{2n}=0 \\
2n\alpha_{2n-4}+\left[4n+1+\left(n-1\right)\left(2n-1\right)\kappa'^2\right]\alpha_{2n-2}+\left(2n+1\right)\left[\left(n+1\right)\kappa'^2+1\right]\alpha_{2n}=0
\end{aligned}$$

## N.11 Design of equi-ripple FIR filters with Zolotarev polynomials

The *Zolotarev* polynomials are an extension of the *Chebyshev* polynomials that have found applications in the design of narrow-band FIR filters [127, 220, 154]. *Zahradník* [154] shows that the Zolotarev polynomials can provide a closed-form design of an FIR low-pass filter with an "almost" equi-ripple approximation in two separate frequency bands. Appendix D reviews the notation for *Legendre*'s elliptic integrals and *Jacobi*'s elliptic functions.

### N.11.1 The Zolotarev polynomials

*Chen* and *Parks* [220, Section II], define "a Zolotarev polynomial of degree $N$, with $L$ zeros in the interval $(\alpha, \beta)$ and $N - L$ zeros in the interval $(-1, 1)$. The function value oscillates between $-1$ and $+1$ in both intervals $[\alpha, \beta]$ and $[-1, 1]$, and is larger than unity in the interval $(1, \alpha)$. It has $N - L + 1$ extremal points of value $+1$ or $-1$ in $[-1, 1]$ and $L + 1$ extremal points of value $+1$ or $-1$ in $[\alpha, \beta]$. The closed-form expression for a Zolotarev polynomial uses elliptic functions. It has three independent parameters: degree $N$, the number of zeros $L$ in $(\alpha, \beta)$, and the elliptic function modulus $\kappa$". Denoting such a polynomial by $f_{N,L}(u, \kappa)$, in parametric form [127, Equations 3 and 4]:

$$x = \frac{\operatorname{sn}^2(u, \kappa) + \operatorname{sn}^2\left(\frac{LK}{N}, \kappa\right)}{\operatorname{sn}^2(u, \kappa) - \operatorname{sn}^2\left(\frac{LK}{N}, \kappa\right)} \tag{N.31}$$

$$f_{N,L}(u, \kappa) = \frac{(-1)^L}{2}\left[\left\{\frac{H\left(u - \frac{LK}{N}, \kappa\right)}{H\left(u + \frac{LK}{N}, \kappa\right)}\right\}^N + \left\{\frac{H\left(u + \frac{LK}{N}, \kappa\right)}{H\left(u - \frac{LK}{N}, \kappa\right)}\right\}^N\right] \tag{N.32}$$

where $\kappa$ is the *elliptic modulus*, $K = F\left(\frac{\pi}{2}, \kappa\right)$ and $H(u, \kappa)$ is the *Jacobi Eta* function. The complex variable $u$ follows the path [220, Figure 2]:

- $u \in [0, \imath K']$ maps to $x \in [-1, 1]$
- $u \in [\imath K', K + \imath K']$ maps to $x \in [1, \alpha]$
- $u \in [K + \imath K', K]$ maps to $x \in [\alpha, \beta]$

*Vlček* and *Unbehauen* [127] prefer the notation $Z_{p,q}(w, \kappa)$ where $p$ counts the number of zeros to the right of the maximum, $q$ counts the number of zeros to the left and *Vlček* and *Unbehauen* introduce the independent variable $w$, related to the discrete-time $z$-domain by:

$$w = \frac{1}{2}\left(z + z^{-1}\right)\big|_{z=e^{\imath\omega T}} = \cos\omega T$$

The intervals $x \in (-1, 1) \cup (\alpha, \beta)$ are transformed from the $x$-domain to the $w$-domain intervals $(-1, w_s) \cup (w_p, 1)$ by [127, Equation 6]:

$$w = x\operatorname{cn}^2(u_0, \kappa) - \operatorname{sn}^2(u_0, \kappa) \tag{N.33}$$

$$= \frac{\operatorname{sn}^2(u, \kappa)\operatorname{cn}^2(u_0, \kappa) + \operatorname{cn}^2(u, \kappa)\operatorname{sn}^2(u_0, \kappa)}{\operatorname{sn}^2(u, \kappa) - \operatorname{sn}^2(u_0, \kappa)} \tag{N.34}$$

where $u_0 = \frac{p}{p+q}K(\kappa)$. The inverse transformation is [220, Equation 2] [46, Section 22.6(iv)]:

$$\begin{aligned} u &= \operatorname{arcsc}\left(\imath\operatorname{sc}(u_0, \kappa)\sqrt{\frac{1+w}{1-w}}, \kappa\right) \\ &= \imath\operatorname{arcsn}\left(\operatorname{sc}(u_0, \kappa)\sqrt{\frac{1+w}{1-w}}, \sqrt{1-\kappa^2}\right) \end{aligned} \tag{N.35}$$

The extremal values of the Zolotarev polynomial $Z_{p,q}(w, \kappa)$ of degree $n = p + q$ alternate between $-1$ and $1$ $p + 1$ times in the interval $(-1, w_s)$ and $q + 1$ times in the interval $(w_p, 1)$.

For convenience, *Zahradnik et al.* [156, Equation 7] rewrite Equation N.32 as:

$$Z_{p,q}(u, \kappa) = (-1)^p \cosh\left\{n\log\left[\frac{H(u + u_0, \kappa)}{H(u - u_0, \kappa)}\right]\right\}$$

Alternatively, the Zolotarev polynomials can be expressed in terms of the Chebyshev polynomials of the first kind, $T_n(\cos\Phi) = \cos n\Phi$ [127, Equations 26 and 27]:

$$\begin{aligned} Z_{p,q}(u,\kappa) &= (-1)^p T_{p+q}(\cos\Phi(u,\kappa)) \\ \cos\Phi(u,\kappa) &= \frac{1}{2}\left[\frac{H(u+u_0,\kappa)}{H(u-u_0,\kappa)} + \frac{H(u-u_0,\kappa)}{H(u+u_0,\kappa)}\right] \end{aligned} \tag{N.36}$$

*Whittaker* and *Watson* [42, Page 480] define the *Jacobi Eta* function as:

$$H(u,\kappa) = -\imath q^{-\frac{1}{4}} e^{\frac{\imath\pi u}{2K}} \Theta(u+\imath K',\kappa) \tag{N.37}$$

where $\Theta(u,\kappa)$ is the *Jacobi Theta* function, an even function with period $2K+\imath 2K'$. *Chen* and *Parks* [220, Appendix, Equation A3] show (with confusing typesetting) that in the central lobe of the Zolotarev polynomial, for which $u = \sigma + \imath K'$ with $\sigma \in [0, K]$:

$$\left\{\frac{H(u+u_0,\kappa)}{H(u-u_0,\kappa)}\right\}^n = (-1)^p \left\{\frac{\Theta(\sigma+u_0,\kappa)}{\Theta(\sigma-u_0,\kappa)}\right\}^n$$

The maximiser of the Zolotarev polynomial in the central lobe is [220, Appendix]:

$$u_m = \sigma_m + \imath K'$$

where:

$$\begin{aligned} \kappa' &= \sqrt{1-\kappa^2} \\ K' &= F\left(\frac{\pi}{2},\kappa'\right) \\ \sigma_m &= \operatorname{arcsn}\left(\left[\frac{Z(u_0,\kappa)}{\kappa^2 \operatorname{sn}(u_0,\kappa)\left[\operatorname{cn}(u_0,\kappa)\operatorname{dn}(u_0,\kappa) + \operatorname{sn}(u_0,\kappa) Z(u_0,\kappa)\right]}\right]^{\frac{1}{2}}, \kappa\right) \end{aligned} \tag{N.38}$$

and $Z(u,\kappa)$ is the *Jacobi Zeta* function. The corresponding maximum value is:

$$f_m = (-1)^p \cosh\left[n \log \frac{\Theta(\sigma_m+u_0,\kappa)}{\Theta(\sigma_m-u_0,\kappa)}\right] \tag{N.39}$$

The Octave script *zolotarev_vlcek_unbehauen_test.m* calculates $Z_{5,9}(w, 0.78)$, shown as Figure 1 by *Vlček* and *Unbehauen* [127] and shown here as Figure N.77.

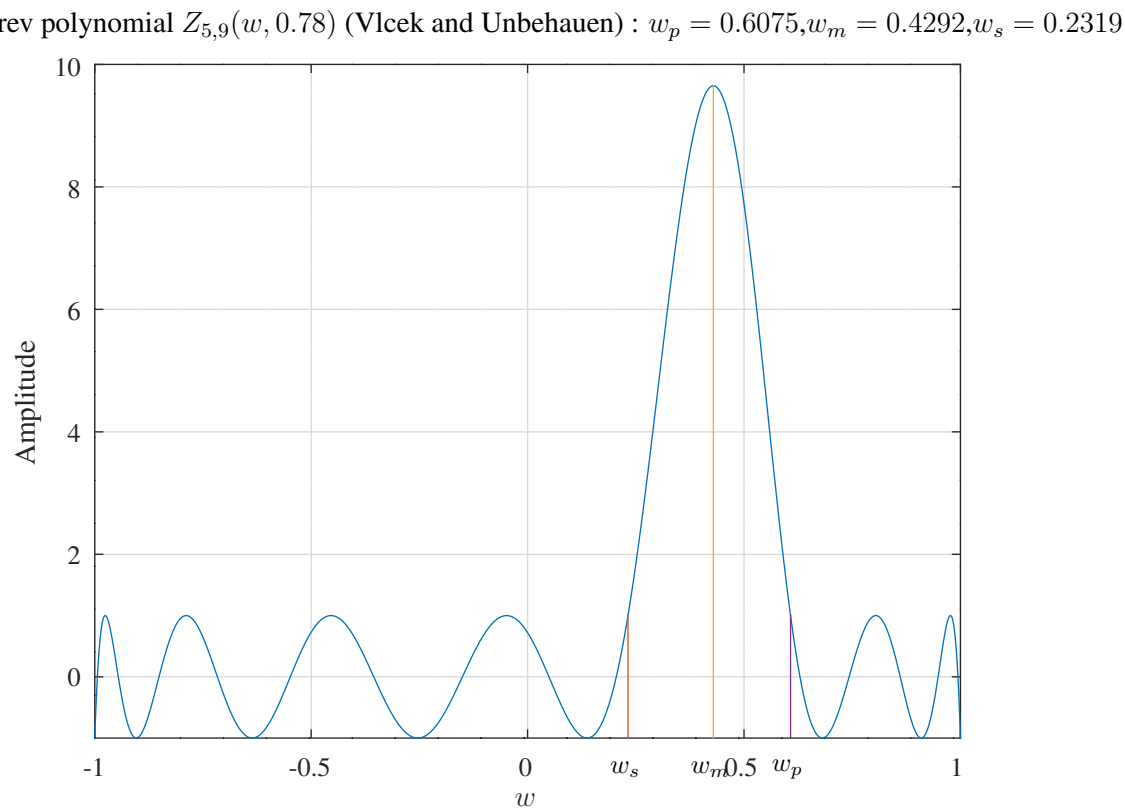



Figure N.77: Zolotarev polynomial $Z_{5,9}(u, 0.78)$

Following *Levy* [172, Equation 28] or *Vlček* and *Unbehauen* [126, Equations 22 and 23], *Vlček* and *Unbehauen* assert that, by inspection, the Zolotarev polynomials, $Z_{p,q}(w,\kappa)$, satisfy the following differential equation [127, Equation 7]:

$$\left(1-w^2\right)\left(w-w_p\right)\left(w-w_s\right)\left(\frac{df}{dw}\right)^2 = n^2\left(1-f^2\right)\left(w-w_m\right)^2 \tag{N.40}$$

Quoting *Vlček* and *Zahradník* [133, Section III]: "This nonlinear differential equation expresses the fact that the first derivative $Z'_{p,q}(w,\kappa)$ does not vanish at the points $w=\pm 1$, $w_p$, $w_s$, where $Z_{p,q}(w,\kappa)=\pm 1$ for which the right hand side of eq. (3) vanishes, and that $w=w_m$ is a turning point corresponding to the local extrema at which $Z_{p,q}(w,\kappa)\neq\pm 1$." *Levy* [172] and *Vlček* and *Unbehauen* [127] show the solution of this non-linear differential equation with elliptic functions. Alternatively, differentiating Equation N.40 gives the following linear second-order differential equation [127, Equation 63]:

$$g_2(w)\left[\left(1-w^2\right)\frac{d^2f}{dw^2}-w\frac{df}{dw}\right]-\left(1-w^2\right)g_1(w)\frac{df}{dw}+n^2g_0(w)f=0 \tag{N.41}$$

where:

$$\begin{aligned} g_2(w) &= \left(w-w_p\right)\left(w-w_s\right)\left(w-w_m\right) \\ g_1(w) &= \left(w-w_p\right)\left(w-w_s\right)-\left(w-w_m\right)\left(w-\frac{w_p+w_s}{2}\right) \\ g_0(w) &= \left(w-w_m\right)^3 \end{aligned} \tag{N.42}$$

This differential equation can be solved by substituting the power series:

$$f(w)=\sum_{m=0}^{n} b_m w^m$$

*Vlček* and *Unbehauen* summarise the resulting recurrence relations in [127, Table IV], reproduced here (with corrections [128]) as Algorithm N.5.

The power series $b_m$ coefficients of $Z_{5,15}(u,0.77029)$ found with Algorithm N.5 are:

```
b_5_15 = [       -0.935306,        7.261946,      197.181637,     -497.153495, ...
             -6876.186281,     9666.711688,    92878.149607,   -82519.667919, ...
           -633477.646079,   368487.201207,  2459186.249287,  -937494.520569, ...
          -5751547.258502,  1410434.491871,  8248725.033331, -1243195.392071, ...
          -7097006.615827,   593999.697155,  3362259.347747,  -118888.629814, ...
           -674338.319614 ];
```

The corresponding $z$-domain discrete-time impulse response coefficients are:

```
h_5_15 = [  -0.0531541684,  -0.0187425987,  -0.0029739578,   0.0184629608, ...
             0.0320251989,   0.0269417021,   0.0032905111,  -0.0262222295, ...
            -0.0430236735,  -0.0344479981,  -0.0029979528,   0.0335763755, ...
             0.0524256541,   0.0401078216,   0.0020985347,  -0.0394119519, ...
            -0.0587615930,  -0.0430273457,  -0.0007539335,   0.0427632649, ...
             0.0609976102,   0.0427632649,  -0.0007539335,  -0.0430273457, ...
            -0.0587615930,  -0.0394119519,   0.0020985347,   0.0401078216, ...
             0.0524256541,   0.0335763755,  -0.0029979528,  -0.0344479981, ...
            -0.0430236735,  -0.0262222295,   0.0032905111,   0.0269417021, ...
             0.0320251989,   0.0184629608,  -0.0029739578,  -0.0187425987, ...
            -0.0531541684 ];
```

*Vlček* and *Unbehauen* also show the expansion of $Z_{p,q}(w,\kappa)$ in Chebyshev polynomials of the first kind, $T_m(w)$:

$$f(w)=\sum_{m=0}^{n} a_m T_m(w)$$

The resulting recurrence relations are shown in Algorithm N.6 [127, Table V with corrections][P].

The Chebyshev polynomial expansion coefficients of the example shown by *Vlček* and *Unbehauen* [128, Table VI] are:

```
a_3_6 = [   0.0985975441,   0.0979370695,  -0.0986423865,  -0.1934009697, ...
           -0.0935059158,   0.0955182760,   0.1823184146,   0.0857438837, ...
           -0.0887676563,  -1.0857982595 ];
```

[P]The Maxima script *zolotarev_vlcek_unbehauen_table_v.max* and my working by hand did not reproduce Table V of *Vlček* and *Unbehauen* [127, 128]. However, the Octave script *zolotarev_vlcek_unbehauen_test.m* shows that, for $\kappa=0.78$, $p=5$ and $q=9$, my Algorithm N.5 and Algorithm N.6 produce, to within round-off error, the same Zolotarev polynomial as the calculation using elliptic functions in the Octave function *zolotarev_chen_parks.m*.

**Algorithm N.5** *Vlček* and *Unbehauen*'s backwards recursion for $Z_{p,q}(w) = \sum_{m=0}^{n} b_m w^m(w)$ [127, Table IV], [128]

**Require:** $p$, $q$ and $\kappa$

*Initialisation*:

$n = p + q$

$u_0 = \frac{p}{p+q} K(\kappa)$

$w_p = 2\,\mathrm{cd}^2(u_0, \kappa) - 1$

$w_s = 2\,\mathrm{cn}^2(u_0, \kappa) - 1$

$w_q = \frac{w_p + w_s}{2}$

$w_m = w_s + 2\frac{\mathrm{sn}(u_0,\kappa)\,\mathrm{cn}(u_0,\kappa)}{\mathrm{dn}(u_0,\kappa)} Z(u_0, \kappa)$

$\beta_n = 1$, $\beta_{n+1} = \beta_{n+2} = \beta_{n+3} = \beta_{n+4} = 0$

*Body*:

**for** $m = n + 2$ **down to** $3$ **do**

$d_1 = (m+2)(m+1) w_p w_s w_m$

$d_2 = -(m+1)(m-1) w_p w_s - (m+1)(2m+1) w_m w_q$

$d_3 = w_m \left(n^2 w_m^2 - m^2 w_p w_s\right) + m^2 (w_m - w_q) + 3m(m-1) w_q$

$d_4 = (m-1)(m-2)(w_p w_s - w_m w_q - 1) - 3w_m \left[n^2 w_m - (m-1)^2 w_q\right]$

$d_5 = (2m-5)(m-2)(w_m - w_q) + 3w_m \left[n^2 - (m-2)^2\right]$

$d_6 = n^2 - (m-3)^2$

$\beta_{m-3} = \frac{1}{d_6} \sum_{\mu=1}^{5} d_\mu \beta_{m+3-\mu}$

**end for**

*Normalisation*:

$s_n = \sum_{m=0}^{n} \beta_m$

**for** $m = 0$ **to** $n$ **do**

$b_m = (-1)^p \frac{\beta_m}{s_n}$

**end for**

**Algorithm N.6** Modified *Vlček* and *Unbehauen* backwards recursion for $Z_{p,q}(w) = \sum_{m=0}^{n} a_m T_m(w)$ [127, Table V], [128]

**Require:** $p$, $q$ and $\kappa$

*Initialisation*:

$n = p + q$

$u_0 = \frac{p}{p+q} K(\kappa)$

$w_p = 2\,\mathrm{cd}^2(u_0, \kappa) - 1$

$w_s = 2\,\mathrm{cn}^2(u_0, \kappa) - 1$

$w_q = \frac{w_p + w_s}{2}$

$w_m = w_s + 2\frac{\mathrm{sn}(u_0,\kappa)\,\mathrm{cn}(u_0,\kappa)}{\mathrm{dn}(u_0,\kappa)} Z(u_0, \kappa)$

$\alpha_n = 1,\ \alpha_{n+1} = \alpha_{n+2} = \alpha_{n+3} = \alpha_{n+4} = \alpha_{n+5} = 0$

*Body*:

**for** $m = n + 2$ **down to** $3$ **do**

$c_7 = n^2 - (m-3)^2$

$c_6 = 2\left[(m-2)(m-3)w_p + (m-2)(m-3)w_s + \left((m-2)(m-1) - 3n^2\right)w_m + (m-2)w_q\right]$

$c_5 = 3\left[n^2 - (m-1)^2\right] + \quad \ldots$

$\quad 4\left[3n^2 w_m^2 - (m-1)^2 w_m w_p - (m-1)^2 w_m w_s - (m-1)(m-2) w_p w_s - (m-1) w_m w_q\right]$

$c_4 = 4\left[m^2 w_p + m^2 w_s + \left(m^2 - 3n^2\right) w_m\right] + 8\left[-n^2 w_m^3 + m^2 w_m w_p w_s\right]$

$c_3 = 3\left[n^2 - (m+1)^2\right] + \quad \ldots$

$\quad 4\left[3n^2 w_m^2 - (m+1)^2 w_m w_p - (m+1)^2 w_m w_s - (m+1)(m+2) w_p w_s + (m+1) w_m w_q\right]$

$c_2 = 2\left[(m+2)(m+3)w_p + (m+2)(m+3)w_s + \left((m+2)(m+1) - 3n^2\right)w_m - (m+2)w_q\right]$

$c_1 = n^2 - (m+3)^2$

$\alpha_{m-3} = -\frac{\sum_{\mu=1}^{6} c_\mu \alpha_{m+4-\mu}}{c_7}$

**end for**

*Normalisation*:

$s_n = -\frac{\alpha_0}{2} + \sum_{m=1}^{n} \alpha_m$

$a_0 = (-1)^p \frac{\alpha_0}{2 s_n}$

**for** $m = 1$ **to** $n$ **do**

$a_m = (-1)^p \frac{\alpha_m}{s_n}$

**end for**

## Addendum

**Working for Equation N.35** Re-writing Equation N.34:

$$\begin{aligned} w &= \frac{\mathrm{sn}^2(u,\kappa)\,\mathrm{cn}^2(u_0,\kappa) + \mathrm{cn}^2(u,\kappa)\,\mathrm{sn}^2(u_0,\kappa)}{\left(1 - \mathrm{sn}^2(u_0,\kappa)\right)\mathrm{sn}^2(u,\kappa) - \left(1 - \mathrm{sn}^2(u,\kappa)\right)\mathrm{sn}^2(u_0,\kappa)} \\ &= \frac{\mathrm{sn}^2(u,\kappa)\,\mathrm{cn}^2(u_0,\kappa) + \mathrm{cn}^2(u,\kappa)\,\mathrm{sn}^2(u_0,\kappa)}{\mathrm{cn}^2(u_0,\kappa)\,\mathrm{sn}^2(u,\kappa) - \mathrm{cn}^2(u,\kappa)\,\mathrm{sn}^2(u_0,\kappa)} \\ &= \frac{\mathrm{sc}^2(u,\kappa) + \mathrm{sc}^2(u_0,\kappa)}{\mathrm{sc}^2(u,\kappa) - \mathrm{sc}^2(u_0,\kappa)} \end{aligned} \tag{N.43}$$

Rearranging:

$$\begin{aligned} -(1-w)\,\mathrm{sc}^2(u,\kappa) &= (1+w)\,\mathrm{sc}^2(u_0,\kappa) \\ \mathrm{sc}(u,\kappa) &= \pm\imath\,\mathrm{sc}(u_0,\kappa)\sqrt{\frac{1+w}{1-w}} \end{aligned}$$

Using the *Jacobi imaginary transformation* [46, Table 22.6.1], $\operatorname{sn}(\imath u, \kappa') = \imath \operatorname{sc}(u, \kappa)$, and selecting the "positive" value:

$$u = \imath \operatorname{arcsn}\left(\operatorname{sc}(u_0, \kappa)\sqrt{\frac{1+w}{1-w}}, \sqrt{1-\kappa^2}\right)$$

The $\operatorname{sc}(u, \kappa)$ function has a zero at $u = 0$, a pole at $u = K$, period $2K + \imath 4K'$, $\operatorname{sc}(\imath K', \kappa) = \imath$ and $\operatorname{sc}(K + \imath K', \kappa) = \imath \kappa'^{-1}$ [46, Tables 22.4.1, 22.4.2 and 22.5.1].

Substituting $\varphi_s = \frac{\omega_s T}{2}$, $\varphi_p = \frac{\pi - \omega_p T}{2}$ [127, Page 726] and $\sin \varphi_s = \operatorname{sn}(u_0, \kappa)$ [127, Equation 82] into Equation N.43 and recalling that $w_p = \cos \omega_p T$, at $u = K + \imath K'$:

$$w_p = \frac{-\frac{1}{\kappa'^2} + \tan^2 \varphi_s}{-\frac{1}{\kappa'^2} - \tan^2 \varphi_s}$$

Rearranging:

$$\begin{aligned}
\kappa'^2 &= \cot^2 \varphi_s \frac{1 - w_p}{1 + w_p} \\
&= \cot^2 \varphi_s \tan^2 \frac{\omega_p T}{2} \\
&= \cot^2 \varphi_s \cot^2 \varphi_p
\end{aligned}$$

That is, the complementary elliptic modulus, $\kappa' = \cot \varphi_s \cot \varphi_p$.

**Working for linearising Equation 7 [127]** Rearrange Equation 7 as:

$$\frac{df}{dw} = g\sqrt{1 - f^2}$$

where:

$$g(w) = \frac{n(w - w_m)}{\sqrt{(1 - w^2)(w - w_p)(w - w_s)}}$$

Differentiating both sides:

$$\frac{d^2 f}{dw^2} = \frac{dg}{dw}\sqrt{1 - f^2} - g\frac{1}{2}\frac{1}{\sqrt{1 - f^2}} 2f \frac{df}{dw}$$

Substituting $\frac{df}{dw}$:

$$\frac{d^2 f}{dw^2} = \frac{1}{g}\frac{dg}{dw}\frac{df}{dw} - g^2 f$$

If $w_q = \frac{w_p + w_s}{2}$:

$$\frac{1}{g}\frac{dg}{dw} = \frac{1}{w - w_m} + \frac{w}{1 - w^2} - \frac{w - w_q}{(w - w_p)(w - w_s)}$$

Rearranging:

$$\begin{aligned}
&\left[\left(1 - w^2\right)(w - w_m)(w - w_p)(w - w_s)\right]\frac{d^2 f}{dw^2} \quad \ldots \\
&- \left[\left(1 - w^2\right)(w - w_p)(w - w_s) + w(w - w_m)(w - w_p)(w - w_s) - \left(1 - w^2\right)(w - w_m)(w - w_q)\right]\frac{df}{dw} \quad \ldots \\
&+ n^2 (w - w_m)^3 f = 0
\end{aligned}$$

**Working for Equation 73 [127]** The Chebyshev polynomial of the first kind satisfies the differential equation:

$$\left(1 - w^2\right)\frac{d^2 T_n(w)}{dw^2} - w\frac{dT_n(w)}{dw} + n^2 T_n(w) = 0$$

Equation N.41 is deliberately written in a form that can be solved with an expansion of $Z_{p,q}(w)$ in Chebyshev polynomials of the first kind, $T_m(w)$:

$$f(w) = \sum_{m=0}^{n} a_m T_m(w)$$

so that:

$$\left(1 - w^2\right)\frac{d^2 f}{dw^2} - w\frac{df}{dw} = -\sum_{m=0}^{n} m^2 a_m T_m(w)$$

**Working for Equation 74 [127]** Equation 74 uses the following properties of the Chebyshev polynomials:

$$\begin{aligned}\frac{dT_m(w)}{dw} &= mU_{m-1}(w)\\ \left(1-w^2\right)U_{m-1}(w) &= wT_m(w) - T_{m+1}(w)\\ T_{m-1}(w) - wT_m(w) &= wT_m(w) - T_{m+1}(w)\end{aligned}$$

so that:

$$\left(1-w^2\right)\frac{df}{dw} = \sum_{m=0}^{n} ma_m\left[T_{m-1}(w) - wT_m(w)\right]$$

**Working for Equation 75 [127]** Equation 75 results directly from substituting Equations 72, 73 and 74 into Equation 63 (shown above as Equation N.41):

$$-\sum_{m=0}^{n} m^2 a_m g_2(w) T_m(w) - \sum_{m=0}^{n} ma_m g_1(w)\left[T_{m-1}(w) - wT_m(w)\right] + \sum_{m=0}^{n} a_m n^2 g_0(w) T_m(w) = 0$$

**Working for Table V [127]** Expanding $g_0(w)$, $g_1(w)$ and $g_2(w)$ in Equation 75 and setting $w_q = \frac{w_p+w_s}{2}$:

$$\begin{aligned}&-\sum_{m=0}^{n} m^2\alpha_m\left[(w-w_p)(w-w_s)(w-w_m)\right]T_m(w) \quad \ldots\\ &-\sum_{m=0}^{n} m\alpha_m\left[(w-w_p)(w-w_s)-(w-w_m)(w-w_q)\right]\left[T_{m-1}(w)-wT_m(w)\right] \quad \ldots\\ &+\sum_{m=0}^{n}\alpha_m\left[n^2(w-w_m)^3\right]T_m(w) = 0\end{aligned}$$

$$\begin{aligned}&-\sum_{m=0}^{n} m^2\alpha_m\left[w^3 - w_pw^2 - w_sw^2 - w_mw^2 + w_mw_pw + w_mw_sw + w_pw_sw - w_pw_mw_s\right]T_m(w) \quad \ldots\\ &-\sum_{m=0}^{n} m\alpha_m\left[w^2 - w_pw - w_sw + w_pw_s - w^2 + w_mw + w_qw - w_qw_m\right]\left[T_{m-1}(w) - wT_m(w)\right] \quad \ldots\\ &+\sum_{m=0}^{n} n^2\alpha_m\left[w^3 - 3w_mw^2 + 3w_m^2w - w_m^3\right]T_m(w) = 0\end{aligned}$$

After much algebra (shown in the Octave script *zolotarev_vlcek_unbehauen_table_v_test.m*) I arrived at:

$$\begin{aligned}&\sum_{m=3}^{n+3}\left[n^2-(m-3)^2\right]\alpha_{m-3}T_m(w) \quad \ldots\\ &+\sum_{m=2}^{n+2} 2\left[(m-2)(m-3)w_p + (m-2)(m-3)w_s \quad \ldots\right.\\ &\qquad\left.+\left((m-2)(m-1)-3n^2\right)w_m + (m-2)w_q\right]\alpha_{m-2}T_m(w) \quad \ldots\\ &+\sum_{m=1}^{n+1} 3\left[n^2-(m-1)^2\right] + \quad \ldots\\ &\qquad 4\left[3n^2w_m^2 - (m-1)^2w_mw_p - (m-1)^2w_mw_s - (m-1)(m-2)w_pw_s - (m-1)w_mw_q\right]\alpha_{m-1}T_m(w) \quad \ldots\\ &+\sum_{m=0}^{n} 4\left[m^2w_p + m^2w_s + \left(m^2-3n^2\right)w_m\right] + 8\left[-n^2w_m^3 + m^2w_mw_pw_s\right]\alpha_mT_m(w) \quad \ldots\\ &+\sum_{m=-1}^{n-1} 3\left[n^2-(m+1)^2\right] + \quad \ldots\\ &\qquad 4\left[3n^2w_m^2 - (m+1)^2w_mw_p - (m+1)^2w_mw_s - (m+1)(m+2)w_pw_s + (m+1)w_mw_q\right]\alpha_{m+1}T_m(w) \quad \ldots\end{aligned}$$

$$+ \sum_{m=-2}^{n-2} 2\left[(m+2)(m+3)w_p + (m+2)(m+3)w_s + \left((m+2)(m+1) - 3n^2\right)w_m - (m+2)w_q\right]\alpha_{m+2}T_m(w) \quad \ldots$$

$$+ \sum_{m=-3}^{n-3} \left[n^2 - (m+3)^2\right]\alpha_{m+3}T_m(w) = 0$$

Algorithm N.6 follows by collecting terms in the Chebyshev polynomials of equal order. The recursion is initialised by setting $\alpha_n = 1$. At each backwards recursion, $m = n+2, \ldots, 3$, the next coefficient calculated is $\alpha_{m-3}$.

## N.11.2 Narrow-band FIR filter design with the Zolotarev polynomials

### Degree equations for the Zolotarev polynomials

If the Zolotarev polynomial is scaled to lie in the range $[0, 1]$ and the stop-band specification is $a_{s_{dB}} = 20 \log_{10} a_s < 0$:

$$\frac{2}{1 + f_m} \leq 10^{a_{s_{dB}}/20} \tag{N.44}$$

Rearranging:

$$f_m \geq y_m = 2 \times 10^{-a_{s_{dB}}/20} - 1 \tag{N.45}$$

Recalling Equation N.39, the scaled maximum value is:

$$\cosh \left[ n \log \frac{\Theta(\sigma_m + u_0, \kappa)}{\Theta(\sigma_m - u_0, \kappa)} \right] \geq y_m \tag{N.46}$$

and the stop-band degree equation is:

$$n \geq \frac{\operatorname{arccosh} y_m}{\log \frac{\Theta(\sigma_m + u_0, \kappa)}{\Theta(\sigma_m - u_0, \kappa)}}$$

Alternatively, solving for $y_m$ with $\lambda = e^x$:

$$\frac{1}{2} \left[ \lambda + \lambda^{-1} \right] \geq y_m \tag{N.47}$$

$$\lambda^2 - 2 y_m \lambda + 1 \geq 0 \tag{N.48}$$

$$\tag{N.49}$$

The stop-band degree equation becomes [127, Equation 85]:

$$n \geq \frac{\log \left( y_m + \sqrt{y_m^2 - 1} \right)}{\log \frac{\Theta(\sigma_m + u_0, \kappa)}{\Theta(\sigma_m - u_0, \kappa)}} \tag{N.50}$$

*Zahradník et al.* [156] derive a degree equation for narrow band-pass Zolotarev polynomial FIR filters in terms of the pass-band attenuation specification, $a_{p_{dB}} = 20 \log_{10} a_p < 0$ and the pass-band bandwidth, $\Delta_p T = \omega_{p2} T - \omega_{p1} T$. Their procedure selects the elliptic function modulus, $\kappa$, by setting [156, Equation 13]:

$$\frac{\Theta(\sigma_{p1} + u_0, \kappa)}{\Theta(\sigma_{p1} - u_0, \kappa)} = \frac{\Theta(\sigma_{p2} + u_0, \kappa)}{\Theta(\sigma_{p2} - u_0, \kappa)}$$

The response is scaled so that the pass-band degree equation is [156, Equation 14](with corrections[q]):

$$n \geq \frac{\operatorname{arccosh} \left( 2 \times 10^{\left(a_{p_{dB}} - a_{s_{dB}}\right)/20} - 1 \right)}{\log \frac{\Theta(\sigma_{p1} + u_0, \kappa)}{\Theta(\sigma_{p1} - u_0, \kappa)}}$$

Unfortunately, the pass-band bandwidth relationship usually does not hold after the elliptic function quarter-period, $K(\kappa)$, is translated to integral values of $n$, $p$ and $q$,

### The Zolotarev polynomial FIR filter design procedure of *Vlčk* and *Unbehauen*

*Vlček* and *Unbehauen* [127, Section VII] give the following procedure for the design of narrow-band FIR filters based on the Zolotarev polynomials with specified stop-band bandwidth and stop-band attenuation:

1. Specify the desired pass-band and stop-band edges at angular frequencies $\omega_p T < \omega_s T$, and the stop-band attenuation, $a_{s_{dB}} = 20 \log_{10} a_s < 0$
2. Evaluate the elliptic function modulus, $\kappa$, for $\varphi_p = \frac{\pi - \omega_p T}{2}$ and $\varphi_s = \frac{\omega_s T}{2}$, $\kappa' = \cot \varphi_p \cot \varphi_s$

[q] *Zahradnik et al.* use the Jacobi *Eta* function, the ratio of Jacobi *Eta* functions is claimed to be real at $w_{p1}$ and log is missing.

3. Use the degree equation, Equation N.50, to find the degree, $n$, that satisfies the attenuation requirement

$$f_m > y_m = 2 \times 10^{-a_{s_{dB}}/20} - 1$$

corresponding to the maximum of the scaled Zolotarev polynomial.

4. Determine integer values of $p$ and $q$ corresponding to:

$$\frac{q}{n} K(\kappa) = F(\varphi_p, \kappa) \ , \quad \frac{p}{n} K(\kappa) = F(\varphi_s, \kappa)$$

5. Calculate the resulting values of $w_p$, $w_s$ and $w_m$:

$$\begin{aligned}
u_0 &= \frac{p}{n} K(\kappa) \ , \quad u_q = \frac{q}{n} K(\kappa) \\
w_p &= \cos \omega_p T = 2 \operatorname{sn}^2(u_q, \kappa) - 1 \\
w_s &= \cos \omega_s T = 1 - 2 \operatorname{sn}^2(u_0, \kappa) \\
w_m &= \cos \omega_m T = w_s + 2 \frac{\operatorname{sn}(u_0, \kappa) \operatorname{cn}(u_0, \kappa)}{\operatorname{dn}(u_0, \kappa)} Z(u_0, \kappa) \\
u_m &= \sigma_m + \imath K' \\
f_m &= Z_{p,q}(u_m, \kappa)
\end{aligned}$$

6. Perform Algorithm N.6 to find the Chebyshev polynomial coefficients, $a_m$, and convert these to the impulse response [127, Equation 87]: $a_0 = h_M$ and $a_m = 2h_{M-m}$.

The Octave script *zolotarev_vlcek_unbehauen_test.m* designs an FIR filter that approximates that shown in Figure 4 of *Vlček* and *Unbehauen* [127]. The initial specification is $fp = 0.1$, $fs = 0.15$ and $a_{s_{dB}} = -20dB$. The final filter has:

```
p=6 % Zeros in lower stop-band
q=17 % Zeros in upper stop-band
N=46 % Degree of FIR polynomial
k=0.770292 % Elliptic modulus
fp=0.105432 % Lower stop-band edge
fmax=0.131418 % Pass-band centre frequency
fs=0.157429 % Upper stop-band edge
delta=-21.224034 % Stop-band attenuation(dB)
```

The $z$-domain impulse reponse coefficients of the FIR filter are:

```
h_6_17 = [   0.0295797662,   0.0123578798,   0.0011255068,  -0.0142207618, ...
            -0.0229948939,  -0.0169353965,   0.0027813101,   0.0244417779, ...
             0.0326289454,   0.0191172985,  -0.0098456126,  -0.0359475478, ...
            -0.0404719137,  -0.0177401179,   0.0193576127,   0.0466303771, ...
             0.0446158675,   0.0125842442,  -0.0296518673,  -0.0542174707, ...
            -0.0439450649,  -0.0045524365,   0.0385342685,   0.1003921567, ...
             0.0385342685,  -0.0045524365,  -0.0439450649,  -0.0542174707, ...
            -0.0296518673,   0.0125842442,   0.0446158675,   0.0466303771, ...
             0.0193576127,  -0.0177401179,  -0.0404719137,  -0.0359475478, ...
            -0.0098456126,   0.0191172985,   0.0326289454,   0.0244417779, ...
             0.0027813101,  -0.0169353965,  -0.0229948939,  -0.0142207618, ...
             0.0011255068,   0.0123578798,   0.0295797662 ];
```

Figure N.78 shows the normalised Zolotarev polynomial of the FIR filter. Figure N.79 shows the frequency response of the FIR filter. Figure N.80 shows the zeros of the FIR filter. Note that these are all *double* zeros.

The Octave script *zolotarev_zahradnik_degree_test.m* attempts to reproduce the examples given by *Zahradník et al.* [156]. Figure N.81 shows the normalised Zolotarev polynomial, $Z_{100,37}(w, 0.4)$, calculated with Equation 7 of *Zahradník et al.* The Octave script *zolotarev_zahradnik_degree_test.m* checks the inverse transformation of $w$ to $u$ shown in Equation N.35. Both the $\operatorname{arcsc}$ and $\operatorname{arccn}$ Octave functions accurately invert $w$ to $u$ over the range of the pass-band in Example 1 of *Zahradník et al.*. A 3rd-order polynomial fit over that range is accurate to 4 decimal places. I failed to reproduce the results of *Zahradník et al.* for their Example 1 with a pass-band constraint. Figure N.82 shows a filter designed with the stop-band bandwidth and attenuation of *Zahradník et al.*'s Example 2. Figure N.83 shows the main lobe of the response.

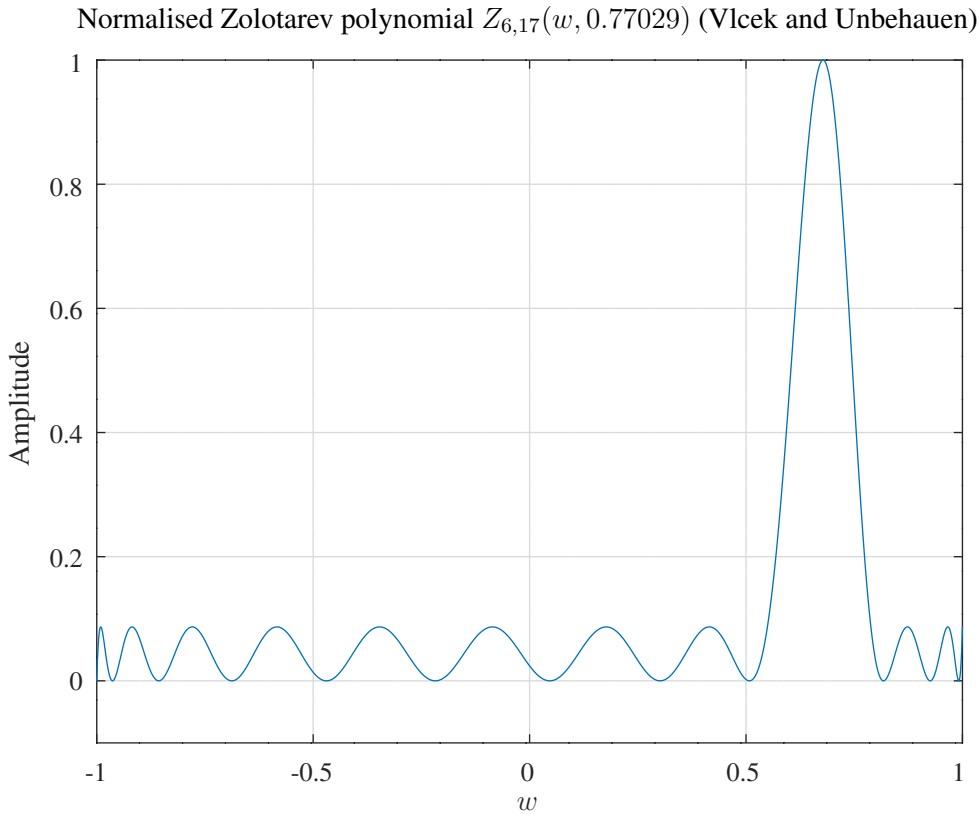



Figure N.78: Normalised Zolotarev polynomial approximating that of Figure 4 in *Vlček* and *Unbehauen* [127]

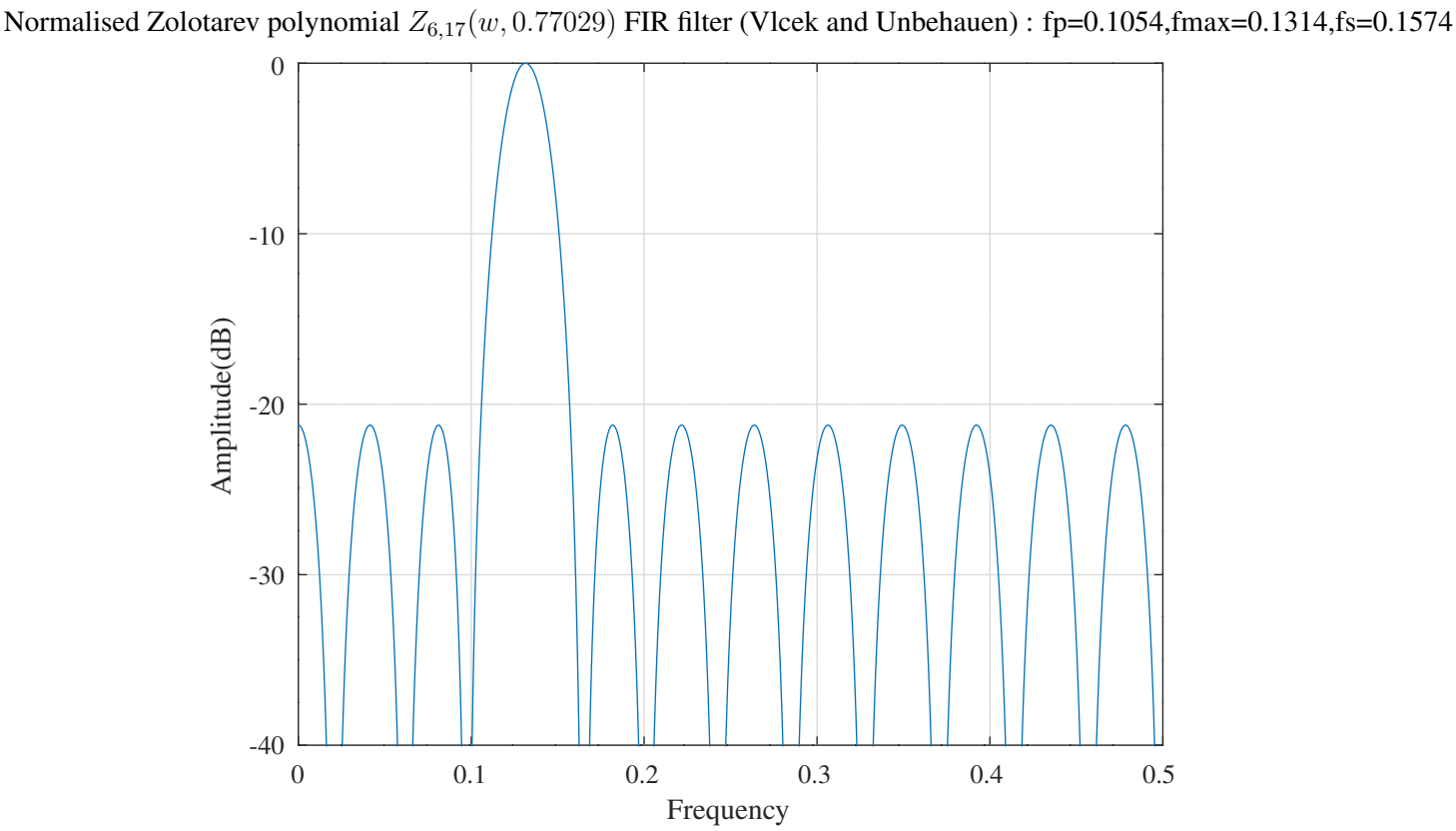



Figure N.79: FIR filter response approximating that of Figure 4 in *Vlček* and *Unbehauen* [127]

Normalised Zolotarev polynomial $Z_{6,17}(w, 0.77029)$ (Vlcek and Unbehauen)

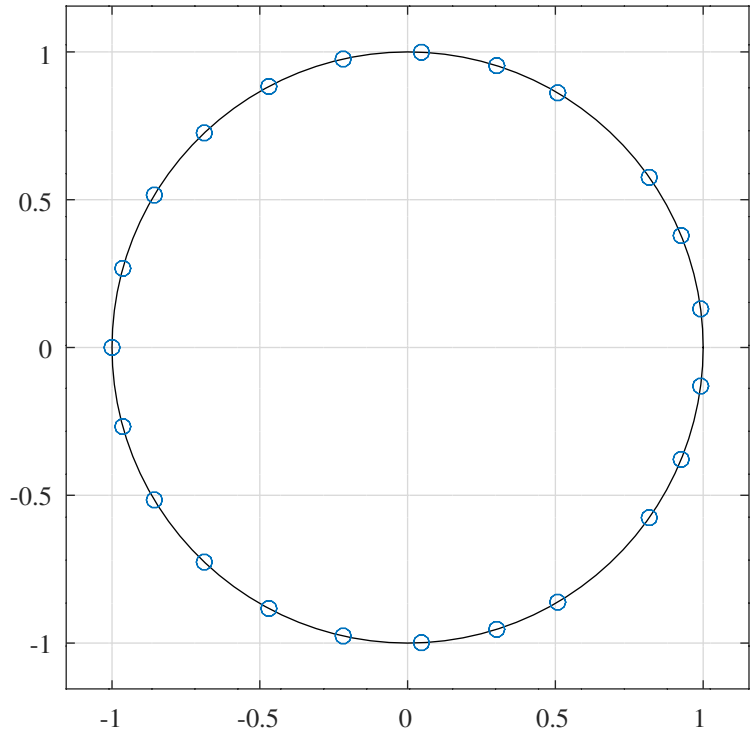



Figure N.80: Zeros of an FIR filter approximating that of Figure 4 in *Vlček* and *Unbehauen* [127]

Normalised Zolotarev polynomial $Z_{100,37}(w, 0.4)$ : $ws_1$=-0.6361, $wm$=-0.6610, $ws_2$=-0.6851, fmax=44.1418

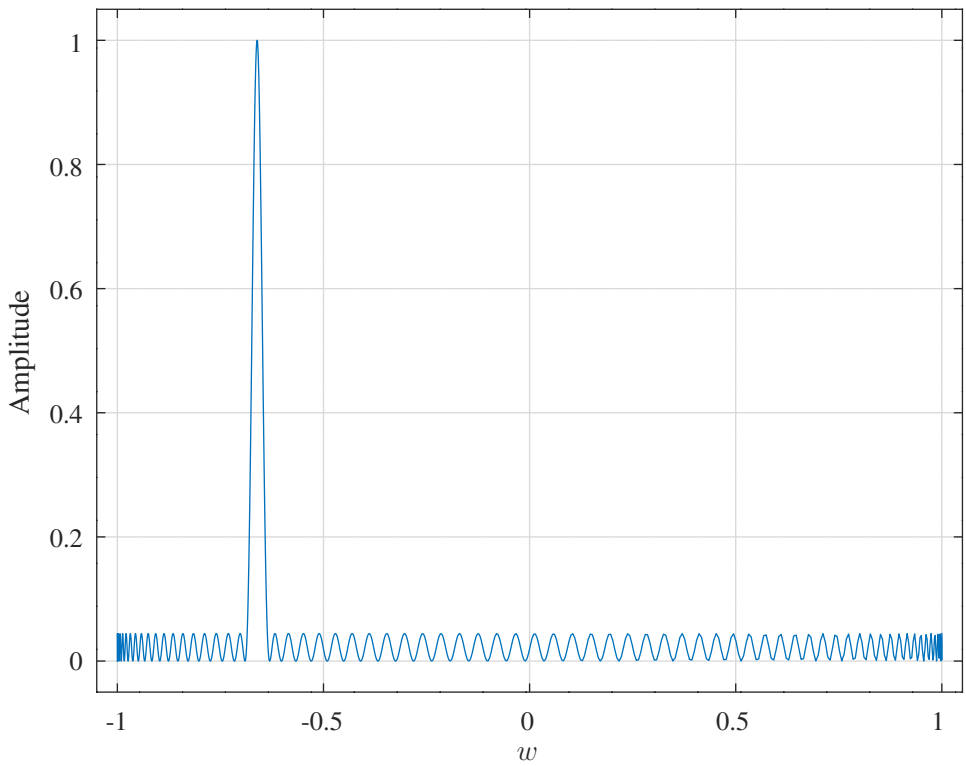



Figure N.81: Normalised Zolotarev polynomial $Z_{100,37}(w, 0.4)$ calculated with Equation 7 of *Zahradník et al.* [132]

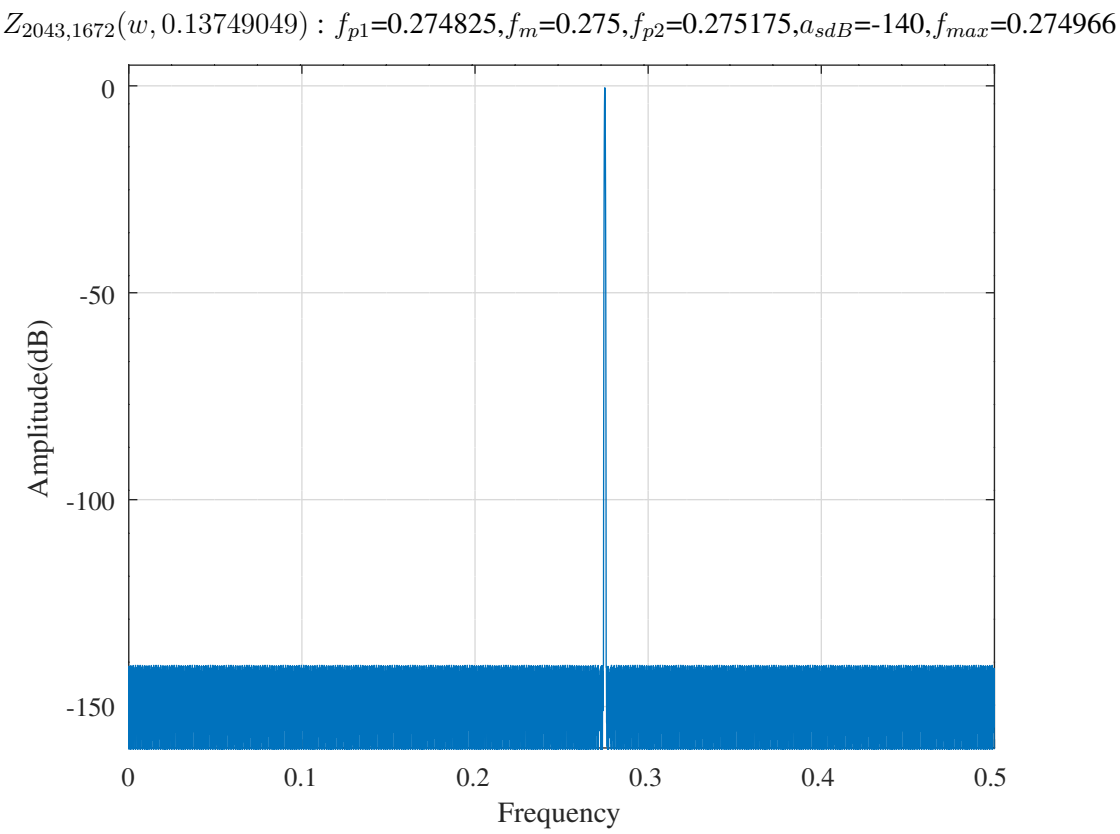



Figure N.82: Normalized Zolotarev polynomial FIR filter approximating Example 2 of *Zahradník et al.* [132] with stop-band constraints.

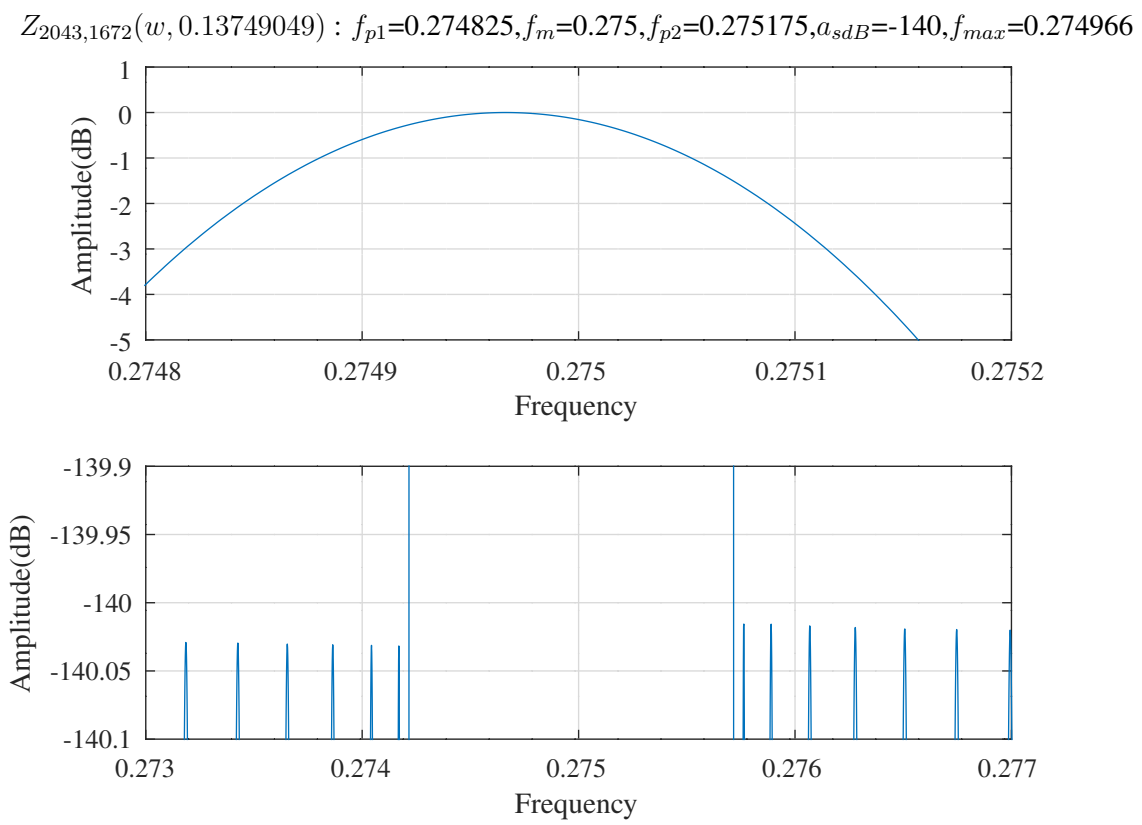



Figure N.83: Main lobe of the response of a normalized Zolotarev polynomial FIR filter approximating Example 2 of *Zahradník et al.* [132] with stop-band constraints.

**The Zolotarev polynomial FIR filter cascade structure design procedure of *Zahradník* et al.**

It is often convenient to express an FIR filter polynomial as the cascade product of sub-filters [203, 39, 102, 103, 223]. As I show in the Introduction, it may be difficult to accurately determine the roots of the FIR filter polynomial and so determine the sub-filters. For example, see *Smith* [102]. *Zahradník, Šusta et al.* [157] derive an expression for the double-roots of the normalised Zolotarev polynomial:

$$Q_{p,q}(w,\kappa) = \frac{1 + Z_{p,q}(w,\kappa)}{1 + Z_{p,q}(w_{max},\kappa)}$$

where $w_{max}$ is the centre-frequency of the main lobe. Their Figure 2 shows that $Q_{p,q}(w,\kappa)$ has $\lfloor \frac{p}{2} \rfloor$ double zeros to the right of the maximum ($u \in [K + \imath K', K]$ and $w \in [w_p, 1]$), $\lfloor \frac{q}{2} \rfloor$ double zeros to the left of the maximum, ($u \in [0, \imath K']$ or $w \in [-1, w_s]$), for a total of $\frac{p+q-2}{2}$ double zeros as well as:

- zeros at $w = \pm 1$ if $p$ is odd and $q$ is odd
- one zero at $w = 1$ if $p$ is odd and $q$ is even
- one zero at $w = -1$ if $p$ is even and $q$ is odd
- no other zeros if $p$ is even and $q$ is even

Recall Equation N.36:

$$\begin{aligned} Z_{p,q}(u,\kappa) &= (-1)^p T_{p+q}(\cos\Phi(u,\kappa)) \\ \cos\Phi(u,\kappa) &= \frac{1}{2}\left[\frac{H(u+u_0,\kappa)}{H(u-u_0,\kappa)} + \frac{H(u-u_0,\kappa)}{H(u+u_0,\kappa)}\right] \end{aligned}$$

On the "vertical" sections of the path of $u$ corresponding to the locations of the zeros of $Q(w,\kappa)$, substituting $u - u_0$ into Equation N.37:

$$H(u-u_0,\kappa) = \begin{cases} -H^\dagger(u+u_0,\kappa) & u \in [0, \imath K'] \\ H^\dagger(u+u_0,\kappa) & u \in [K + \imath K', K] \end{cases} \tag{N.51}$$

where $H^\dagger$ is the complex conjugate transpose of $H$. The $\Phi(u,\kappa)$ function of Equation N.36 becomes:

$$\Phi(u,\kappa) = \begin{cases} -2\arg H(u+u_0,\kappa) + \pi & u \in [0, \imath K'] \\ 2\arg H(u+u_0,\kappa) & u \in [K + \imath K', K] \end{cases}$$

The double-zero locations can be found by interpolation of the multiples of $\frac{\pi}{n}$ into $2\arg H(u+u_0,\kappa)$. Figure N.84 shows the locations of the double zeros of $Q_{p,q}(w, 0.75)$ for pairs of odd and even $p$ and $q$ [157, Figure 2]. Figure N.85 shows the locations of the double zeros of $Q_{p,q}(w, 0.75)$ for pairs of odd and even $p$ and $q$ superimposed on $2\arg H(u+u_0, 0.75)$, normalised to $\frac{n}{\pi}$, in the regions on either side of the central lobe.

*Zahradník, Šusta et al.* [157, Table I] show an algorithm, reproduced below as Algorithm N.7, for the calculation of the coefficients of the expansion in Chebyshev polynomials of the first kind of the product of the factors, $(w - w_l)$, of a polynomial. They recommend ordering the roots in ascending order of $z$-domain angular frequency and the $m$ sub-filters select the roots with the ordering $\rho(l)$, $l = 1, \ldots, m$ [157, Equation 8]:

$$\rho(l) = \begin{cases} \frac{m+l}{2} & m \text{ and } l \text{ both odd or both even} \\ \frac{m-l+1}{2} & \text{otherwise} \end{cases}$$

In addition, they recommend normalising each sub-filter to the value at the frequency of the central lobe maximum of the overall filter [157, Equation 14].

The Octave script *zolotarevFIRcascade_test.m* attempts to reproduce *Zahradník, Šusta et al.*'s Examples 1 and 2. As mentioned previously, I do not understand their method of calculating the degree of the filter from the required pass-band bandwidth. Figure N.86 reproduces Figure 4 of *Zahradník, Šusta et al.*, illustrating their Example 1. Sub-filters 1 and 2 are identical, as expected after examining *Zahradník, Šusta et al.*'s Figure 3. However, the sub-filter 1 and sub-filter 2 shown in their Figure 4 appear to have a similar shape but a different gain, the filter impulse responses shown in their Table II are all different and I do not understand how their Equation 15 corresponds to the numbering shown in their Figure 3. I tried randomly rearranging the roots with the Octave *randperm* function and found that the range of the responses of the resulting sub-filters was far greater.

Figure N.87 shows the double-zero locations near $\pm 1$ and the central lobe of $Q_{2159,540}(w, 0.16238959)$ for *Zahradník, Šusta et al.*'s Example 2 [157]. Figure N.88 attempts to reproduce *Zahradník, Šusta et al.*s Figure 5 showing $Q_{2159,540}(w, 0.16238959)$ with 3 sub-filters. Algorithm N.7, the calculation of the coefficients of the Chebyshev polynomials of the first kind, is implemented with the *MPFR* [1] library in the Octave *octfile roots2T.cc*.

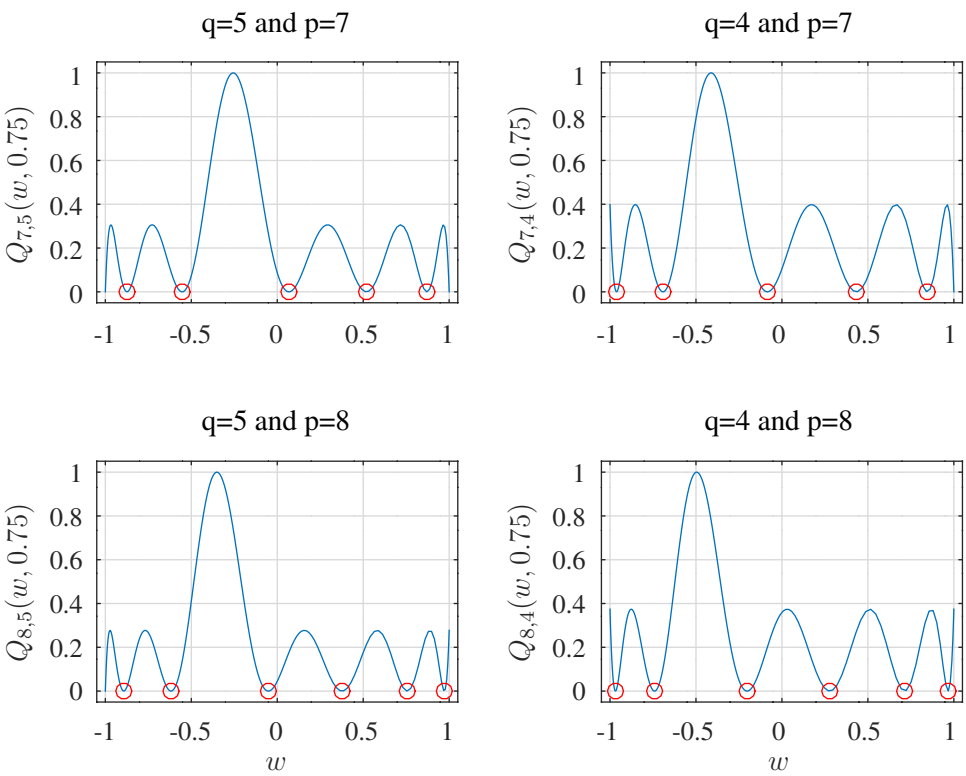



Figure N.84: Double-zero locations of $Q_{p,q}(w, 0.75)$ [157, Figure 2]

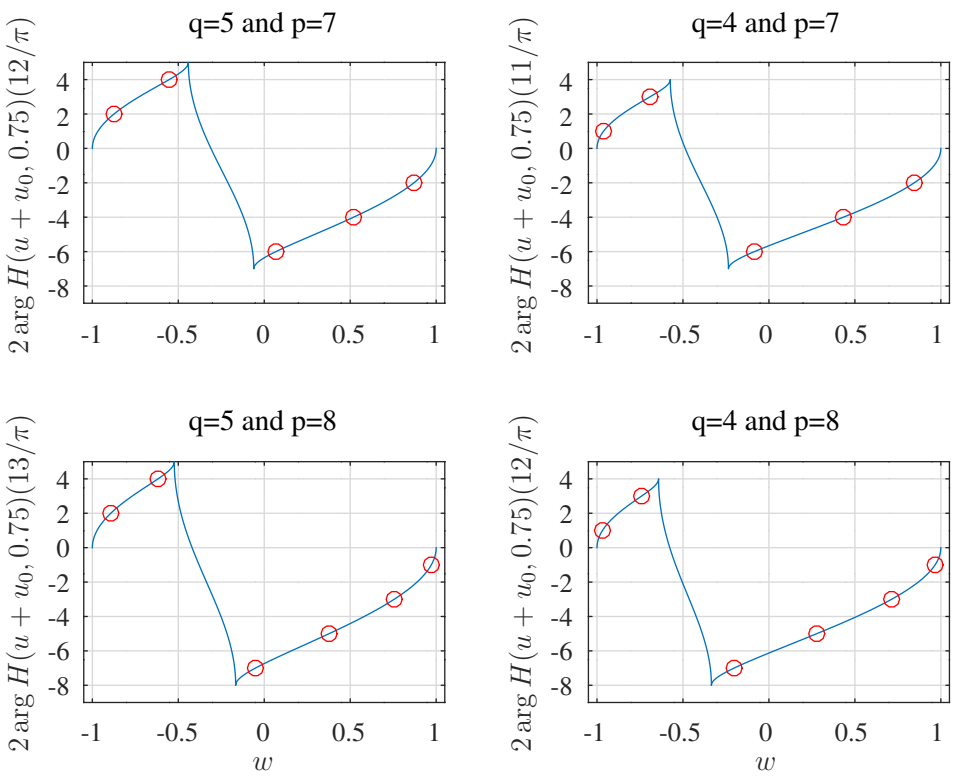



Figure N.85: Double-zero locations of $Q_{p,q}(w, 0.75)$ superimposed on $2 \arg H(u + u_0, 0.75)$ normalised to $\frac{n}{\pi}$

**Algorithm N.7** Recursion of *Zahradník, Šusta et al.* for the calculation of the scaled coefficients of the expansion, in Chebyshev polynomials of the first kind, of the product of the $n$ factors, $(w - w_m)$, of a polynomial [157, Table I]

**Require:** $w_0, \ldots, w_{n-1}$

*Initialisation*:

$\alpha_0^{(0)} = -w_0, \alpha_1^{(0)} = 1$

*Body*:

**for** $m = 1$ **to** $n - 1$ **do**

$\alpha_{m+1}^{(m-1)} = 0$

$\alpha_0^{(m)} = \alpha_1^{(m-1)} - 2w_m\alpha_0^{(m-1)}$

$\alpha_1^{(m)} = \alpha_2^{(m-1)} + 2\alpha_0^{(m-1)} - 2w_m\alpha_1^{(m-1)}$

**for** $l = 2$ **to** $m$ **do**

$\alpha_l^{(m)} = \alpha_{l-1}^{(m-1)} + \alpha_{l+1}^{(m-1)} - 2w_m\alpha_l^{(m-1)}$

**end for**

$\alpha_{m+1}^{(m)} = 1$

**end for**

*Output*:

$a_k = \alpha_k^{(n)} \quad k = 0, \ldots, n$

.

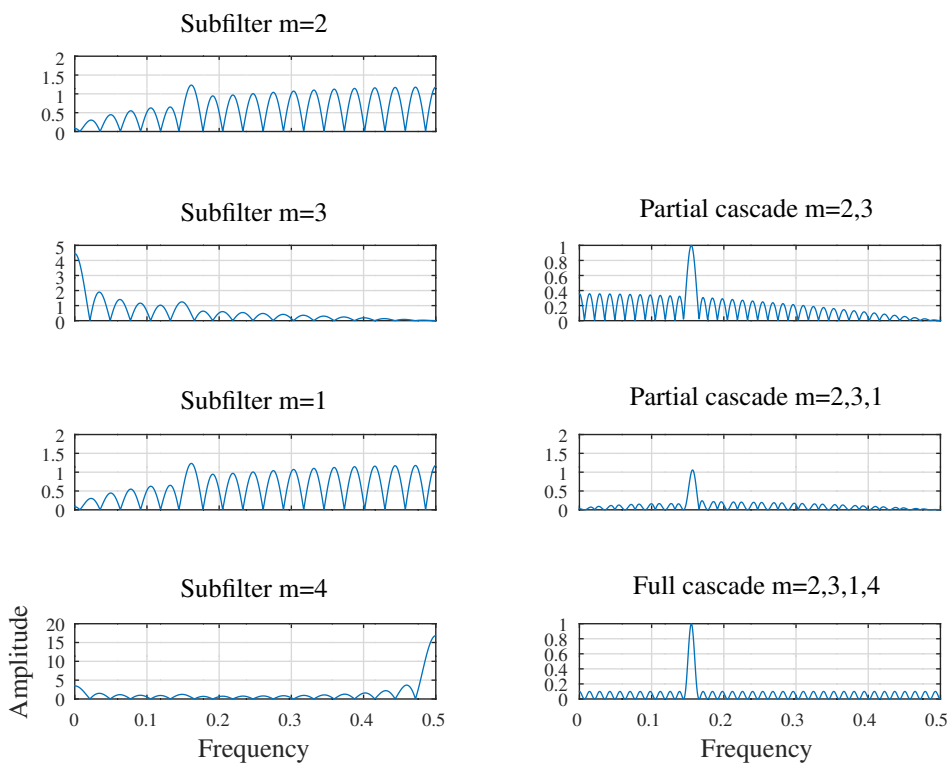



Figure N.86: Amplitude responses of the 4 sub-filters and the partial cascades of the sub-filters of a normalised Zolotarev band-pass filter with zero-phase response $Q_{22,49}(w, 0.46850107)$ [157, Figure 4]

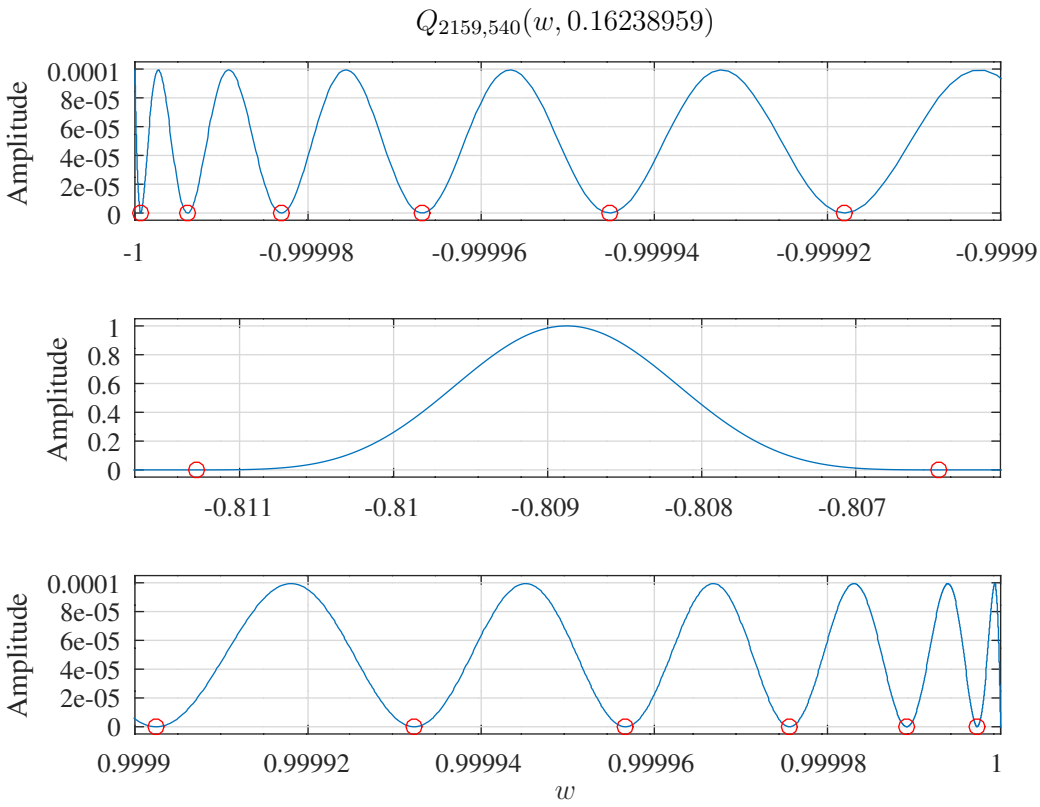



Figure N.87: Detailed view of the double-zero locations of $Q_{2159,540}(w, 0.16238959)$ near $\pm 1$ and the central lobe for *Zahradník, Šusta et al.*'s Example 2 [157]

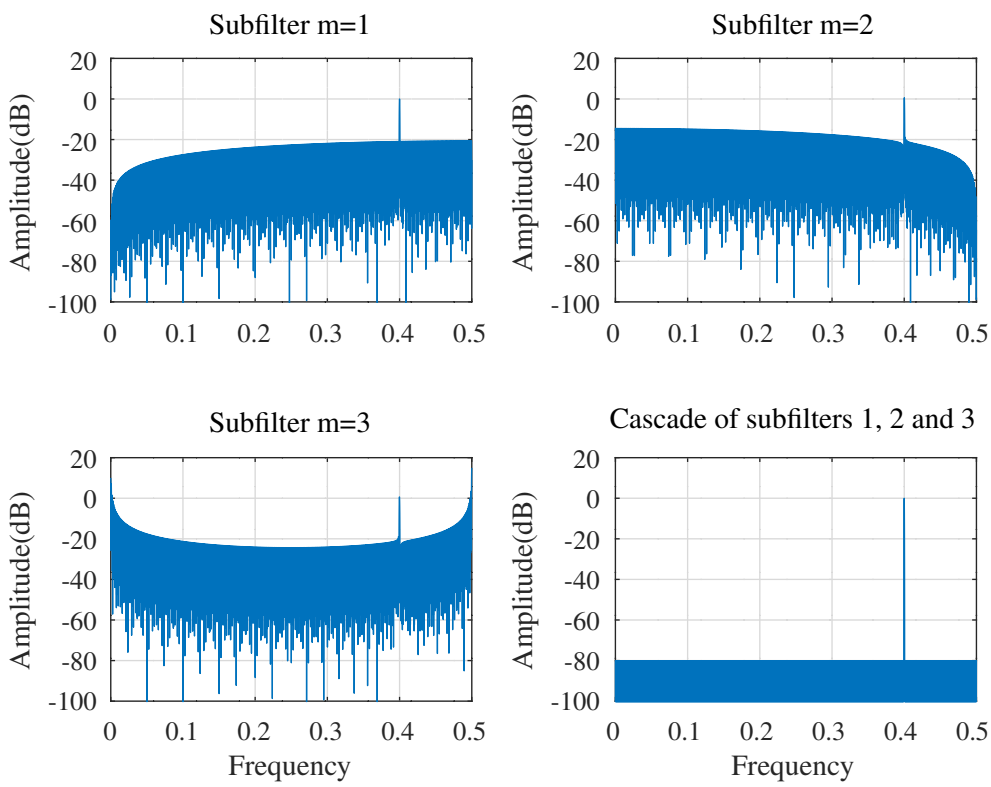



Figure N.88: Amplitude responses of the 3 sub-filters cascade of the sub-filters of a normalised Zolotarev band-pass filter with zero-phase response $Q_{2159,540}(w, 0.16238959)$ [157, Figure 5]

**Addendum**

**Working for Equation N.51** Substituting $u = -u_0 + \imath y K'$, where $y \in [0,1]$, into Equation N.37:

$$\begin{aligned} H^\dagger\left(-u_0 + \imath y K', \kappa\right) &= \imath q^{-\frac{1}{4}} e^{\frac{\imath\pi\left(u_0 + \imath y K'\right)}{2K}} \Theta\left(-u_0 - \imath y K' - \imath K', \kappa\right) \\ &= -H\left(u_0 + \imath y K', \kappa\right) \end{aligned}$$

Substituting $u = K - u_0 + \imath y K'$, where $y \in [0,1]$, into Equation N.37:

$$\begin{aligned} H^\dagger\left(K - u_0 + \imath y K', \kappa\right) &= \imath q^{-\frac{1}{4}} e^{\frac{\imath\pi\left(-K + u_0 + \imath y K'\right)}{2K}} \Theta\left(K - u_0 - \imath y K' - \imath K', \kappa\right) \\ &= -\imath q^{-\frac{1}{4}} e^{\frac{\imath\pi\left(K + u_0 + \imath y K'\right)}{2K}} \Theta\left(-K - u_0 - \imath y K' - \imath K', \kappa\right) \\ &= H\left(K + u_0 + \imath y K', \kappa\right) \end{aligned}$$

**Working for Algorithm N.7** This is an application of the recurrence relations of the Chebyshev polynomials of the first kind:

$$\begin{aligned} T_0\left(w\right) &= 1 \\ T_1\left(w\right) &= w \\ T_{n+1}\left(w\right) &= 2wT_n\left(w\right) - T_{n-1}\left(w\right) \end{aligned}$$

For the first zero (dropping the argument, $w$, of $T_0$ etc.):

$$\left(w - w_0\right) = \alpha_1^{(0)} T_1 + \alpha_0^{(0)} T_0$$

so that $\alpha_1^{(0)} = 1$ and $\alpha_0^{(0)} = -w_0$. Multiplying by the second zero:

$$\begin{aligned} \left(w - w_1\right)\left[\alpha_1^{(0)} T_1 + \alpha_0^{(0)} T_0\right] &= \frac{\alpha_1^{(0)}}{2}\left(2wT_1 - T_0\right) + \frac{\alpha_1^{(0)}}{2} T_0 + \alpha_0^{(0)} w T_0 - w_1 \alpha_1^{(0)} T_1 - w_1 \alpha_0^{(0)} T_0 \\ &= \alpha_1^{(0)} \frac{T_2}{2} + \left(2\alpha_0^{(0)} - 2w_1\alpha_1^{(0)}\right)\frac{T_1}{2} + \left(\alpha_1^{(0)} - 2w_1\alpha_0^{(0)}\right)\frac{T_0}{2} \end{aligned}$$

so that $\alpha_2^{(1)} = 1$, $\alpha_1^{(1)} = 2\alpha_0^{(0)} - 2w_1\alpha_1^{(0)}$ and $\alpha_0^{(1)} = \alpha_1^{(0)} - 2w_1\alpha_0^{(0)}$ . For the third zero, ignoring the scaling by 2:

$$\begin{aligned} \left(w - w_2\right)\left[\alpha_2^{(1)} T_2 + \alpha_1^{(1)} T_1 + \alpha_0^{(1)} T_0\right] &= \frac{\alpha_2^{(1)}}{2}\left(2wT_2 - T_1\right) + \frac{\alpha_2^{(1)}}{2} T_1 + \frac{\alpha_1^{(1)}}{2}\left(2wT_1 - T_0\right) + \frac{\alpha_1^{(1)}}{2} T_0 + \alpha_0^{(1)} w T_0 \quad \ldots \\ &\quad - w_2\alpha_2^{(1)} T_2 - w_2\alpha_1^{(1)} T_1 - w_2\alpha_0^{(1)} T_0 \\ &= \alpha_2^{(1)}\frac{T_3}{2} + \left(\alpha_1^{(1)} - 2w_2\alpha_2^{(1)}\right)\frac{T_2}{2} \quad \ldots \\ &\quad + \left(\alpha_2^{(1)} + 2\alpha_0^{(1)} - 2w_2\alpha_1^{(1)}\right)\frac{T_1}{2} + \left(\alpha_1^{(1)} - 2w_2\alpha_0^{(1)}\right)\frac{T_0}{2} \end{aligned}$$

In general, for $m = 1, 2, \ldots$ :

$$\begin{aligned} \left(w - w_m\right)\sum_{l=0}^{m} \alpha_l^{(m-1)} T_l &= \sum_{l=1}^{m} \frac{\alpha_l^{(m-1)}}{2}\left(2wT_l - T_{l-1}\right) + \sum_{l=1}^{m} \frac{\alpha_l^{(m-1)}}{2} T_{l-1} + \alpha_0^{(m-1)} w T_0 - \sum_{l=0}^{m} w_m \alpha_l^{(m-1)} T_l \\ &= \sum_{l=1}^{m} \alpha_l^{(m-1)} \frac{T_{l+1}}{2} + \sum_{l=1}^{m} \alpha_l^{(m-1)} \frac{T_{l-1}}{2} + 2\alpha_0^{(m-1)} \frac{T_1}{2} - \sum_{l=0}^{m} 2w_m \alpha_l^{(m-1)} \frac{T_l}{2} \\ &= \sum_{l=2}^{m+1} \alpha_{l-1}^{(m-1)} \frac{T_l}{2} + \sum_{l=0}^{m-1} \alpha_{l+1}^{(m-1)} \frac{T_l}{2} + 2\alpha_0^{(m-1)} \frac{T_1}{2} - \sum_{l=0}^{m} 2w_m \alpha_l^{(m-1)} \frac{T_l}{2} \\ &= \alpha_m^{(m-1)} \frac{T_{m+1}}{2} + \alpha_{m-1}^{(m-1)} \frac{T_m}{2} + \sum_{l=2}^{m-1} \alpha_{l-1}^{(m-1)} \frac{T_l}{2} \quad \ldots \\ &\quad + \sum_{l=2}^{m-1} \alpha_{l+1}^{(m-1)} \frac{T_l}{2} + \alpha_2^{(m-1)} \frac{T_1}{2} + \alpha_1^{(m-1)} \frac{T_0}{2} + 2\alpha_0^{(m-1)} \frac{T_1}{2} \quad \ldots \end{aligned}$$

$$
\begin{aligned}
&- 2w_m\alpha_m^{(m-1)}\frac{T_m}{2} - \sum_{l=2}^{m-1} 2w_m\alpha_l^{(m-1)}\frac{T_l}{2} - 2w_m\alpha_1^{(m-1)}\frac{T_1}{2} - 2w_m\alpha_0^{(m-1)}\frac{T_0}{2} \\
&= \alpha_m^{(m-1)}\frac{T_{m+1}}{2} + \left[\alpha_{m-1}^{(m-1)} - 2w_m\alpha_m^{(m-1)}\right]\frac{T_m}{2} \quad \dots \\
&\quad + \sum_{l=2}^{m-1}\left[\alpha_{l-1}^{(m-1)} + \alpha_{l+1}^{(m-1)} - 2w_m\alpha_l^{(m-1)}\right]\frac{T_l}{2} \quad \dots \\
&\quad + \left[\alpha_2^{(m-1)} + 2\alpha_0^{(m-1)} - 2w_m\alpha_1^{(m-1)}\right]\frac{T_1}{2} + \left[\alpha_1^{(m-1)} - 2w_m\alpha_0^{(m-1)}\right]\frac{T_0}{2}
\end{aligned}
$$

### N.11.3 Almost equi-ripple low-pass FIR filter design with the Zolotarev polynomials

*Vlček* and *Zahradník* [132, 133] describe the use of modified Zolotarev polynomials to design "almost equi-ripple" low-pass FIR filters. They assume a solution of the form $\sqrt{1-w^2}S_{p,q}(w,\kappa)$[r] and use $S_{p,q}(w,\kappa)$ as the generating polynomial of a low-pass FIR filter[s]:

$$S_{p,q}(w,\kappa) = \sum_{m=0}^{n} a_m^+ U_m(w)$$
$$\mathcal{S}(w) = \int S_{p,q}(w,\kappa)\,dw$$
$$Q(w) = \frac{\mathcal{S}(w) - \mathcal{N}_1}{\mathcal{N}_2 - \mathcal{N}_1}$$

where $U_m(w)$ is the $m$'th Chebyshev polynomial of the second kind, $Q(w)$ is the zero-phase transfer function of the FIR filter and the normalising constants $\mathcal{N}_1$ and $\mathcal{N}_2$ are:

$$\mathcal{N}_1 = \begin{cases} \mathcal{S}(w=-1) & \text{for q even} \\ \mathcal{S}\left(w = \cos\frac{n\pi}{n+1}\right) & \text{for q odd} \end{cases}$$
$$\mathcal{N}_2 = \begin{cases} \mathcal{S}(w=1) & \text{for p even} \\ \mathcal{S}\left(w = \cos\frac{\pi}{n+1}\right) & \text{for p odd} \end{cases}$$

The frequency response of a symmetric FIR filter of length $2n+3$ is:

$$H(\omega) = \sum_{m=0}^{2n+2} h_m e^{-\imath m\omega T}$$

where the impulse response, $h_m = h_{2n+2-m}$ , $m = 0,\ldots,n$. The zero-phase frequency response, $Q(\omega)$, is related to the frequency response, $H(\omega)$, by:

$$\begin{aligned}
H(\omega) &= e^{-\imath(n+1)\omega T}\left[h_{n+1} + \sum_{m=0}^{n} h_m e^{-\imath[m-(n+1)]\omega T} + \sum_{m=n+2}^{2n+2} h_m e^{-\imath[m-(n+1)]\omega T}\right] \\
&= e^{-\imath(n+1)\omega T}\left[h_{n+1} + \sum_{m=0}^{n} h_m e^{-\imath[m-(n+1)]\omega T} + \sum_{m=0}^{n} h_{2n+2-m} e^{-\imath[2n+2-m-(n+1)]\omega T}\right] \\
&= e^{-\imath(n+1)\omega T}\left[h_{n+1} + \sum_{m=0}^{n} h_m e^{-\imath[m-(n+1)]\omega T} + \sum_{m=0}^{n} h_{2n+2-m} e^{\imath[m-(n+1)]\omega T}\right] \\
&= e^{-\imath(n+1)\omega T}\left[h_{n+1} + 2\sum_{m=0}^{n} h_m \cos\left[(n+1)-m\right]\omega T\right] \\
&= e^{-\imath(n+1)\omega T} Q(\omega)
\end{aligned}$$

*Vlček* and *Zahradník* [132, Tables 4 and 5] provide an algorithm for calculating the $2(p+q)+3$ coefficients of the impulse response of the FIR filter, reproduced here (with altered normalisation) as Algorithm N.8. I have combined *Vlček* and *Zahradník*'s Tables 4 and 5 because the normalisation step in Table 4 normalises the $a^+$ coefficients as if they are coefficients of $\frac{d}{dw}T_m(w)$ rather than $U_m(w)$.

[r] See *Riblet* [66], "If one uses $x = -\cos\theta$ for a frequency variable instead of $\omega$, ... the problem of designing for equal-ripple performance reduces to finding even and odd polynomials $P_n(x)$ so that $P_n(x)/\sqrt{1-x^2}$ oscillates between $\pm 1$ exactly $n+1$ times in a prescribed interval $-1 < -x_c \le x \le x_c < 1$. ... $P_n(x)$ is given then by $2P_n(x) = \left(1+\sqrt{1-x_c^2}\right)T_n\left(\frac{x}{x_c}\right) - \left(1-\sqrt{1-x_c^2}\right)T_{n-2}\left(\frac{x}{x_c}\right)$"

[s] The normalisation is shown in *Vlček* and *Zahradník* [132, 133] as:

$$Q(w) = -\mathcal{N}_1 + \frac{1}{\mathcal{N}_2 - \mathcal{N}_1}\mathcal{S}(w)$$

**Algorithm N.8** *Vlček* and *Zahradník*'s algorithm for the evaluation of the impulse response, $h(m)$, of an "almost equi-ripple" low-pass FIR filter [132, Tables 4 and 5]

**Require:** $p$, $q$ and $\kappa$

*Initialisation*:

$n = p + q$

$u_0 = \frac{2p+1}{2n+2} K(\kappa)$

$w_p = 2\,\mathrm{cd}^2(u_0, \kappa) - 1$

$w_s = 2\,\mathrm{cn}^2(u_0, \kappa) - 1$

$w_q = \frac{w_p + w_s}{2}$

$w_m = w_s + 2\frac{\mathrm{sn}(u_0,\kappa)\,\mathrm{cn}(u_0,\kappa)}{\mathrm{dn}(u_0,\kappa)} Z(u_0, \kappa)$

$\alpha_n = 1, \alpha_{n+1} = \alpha_{n+2} = \alpha_{n+3} = \alpha_{n+4} = \alpha_{n+5} = 0$

*Body*:

**for** $m = n + 2$ **down to** $3$ **do**

$8c_1 = n(n+2) - (m+3)(m+5)$

$4c_2 = 3w_m[n(n+2) - (m+2)(m+4)] + (m+3)(2m+7)(w_m - w_q)$

$8c_3 = 3[n(n+2) - (m+1)(m+3)] + 12w_m\left[(n+1)^2 w_m - (m+2)^2 w_q\right] \ldots$
$\quad -4(m+2)(m+3)(w_p w_s - w_m w_q)$

$2c_4 = 3\left[(n+1)^2 w_m - (m+1)^2 w_q\right] - (m+1)^2(w_m - w_q) \quad \ldots$
$\quad +2w_m\left[(n+1)^2 w_m^2 - (m+1)^2 w_p w_s\right]$

$8c_5 = 3[n(n+2) - (m-1)(m+1)] + 12w_m\left[(n+1)^2 w_m - m^2 w_q\right] \quad \ldots$
$\quad -4m(m-1)(w_p w_s - w_m w_q)$

$4c_6 = 3w_m[n(n+2) - (m-2)m] + (m-1)(2m-3)(w_m - w_q)$

$8c_7 = n(n+2) - (m-3)(m-1)$

$\alpha_{m-3} = \frac{1}{c_7}\sum_{l=1}^{6}(-1)^l c_l \alpha_{m+4-l}$

**end for**

*Normalisation*:

$s_n = \sum_{m=0}^{n}(m+1)\alpha_m$

**for** $m = 0$ **to** $n$ **do**

$a_m^+ = (-1)^p (n+1)\frac{\alpha_m}{s_n}$

**end for**

*Integration*:

**for** $m = 0$ **to** $n$ **do**

$a_{m+1} = \frac{a_m^+}{m+1}$

**end for**

*Impulse response*:

$h_{n+1} = -\frac{\mathcal{N}_1}{\mathcal{N}_2 - \mathcal{N}_1}$

**for** $m = 1$ **to** $n+1$ **do**

$h_{n+1\pm m} = \frac{1}{2}\frac{a_m}{\mathcal{N}_2 - \mathcal{N}_1}$

**end for**

.

*Vlček* and *Zahradník* [132, Section 7] provide the following design procedure:

1. Specify the minimum attenuation in the stop-band, $a_{s_{dB}} = 20\log_{10} a_s < 0$. Specify the pass-band frequency $\omega_p T$ and the stop-band frequency $\omega_s T$. The maximum width of the transition band is $\Delta\omega T = (\omega_s - \omega_p)\, T$.

2. Calculate the degree, $n = p + q$, of the generating polynomial, $S_{p,q}(w)$, using the approximation[t]:

$$\frac{(\xi_1 n + \xi_2)\,\Delta\omega T}{\pi} + \xi_3 + \frac{\xi_4}{(n+\xi_5)^{\xi_6}} = a_{s_{dB}}$$

   where $\xi_1 = -14.02925485$, $\xi_2 = -32.86119410$, $\xi_3 = -5.80117336$, $\xi_4 = 2.99564719$, $\xi_5 = -21.24188066$ and $\xi_6 = 0.28632078$.

3. Determine integer values $p = \lfloor n\frac{(\omega_s+\omega_p)T}{2\pi}\rceil$, $q = n - p$

4. Calculate the elliptic function modulus $\kappa = \sqrt{1 - \left(\frac{1-\hat{\kappa}}{1+\hat{\kappa}}\right)^2}$, where $\hat{\kappa}$ is given by the approximation:

$$\hat{\kappa} = \left\{\left[\chi_1 + \frac{\chi_2}{(p+\chi_3)^{\chi_4}}\right] n + \chi_5 p + \chi_6\right\} w_p + \chi_7 + \frac{\chi_8}{(p+\chi_9)^{\chi_{10}}} + \frac{1}{(n+\chi_{11}p+\chi_{12})^{\chi_{13}p+\chi_{14}}}$$

   for $w_p = \cos\frac{\pi-\Delta\omega T}{2}$ and $\chi_1 = -0.00452871$, $\chi_2 = 0.51350112$, $\chi_3 = 2.56407699$, $\chi_4 = 1.12297611$, $\chi_5 = 0.01473844$, $\chi_6 = 0.14824220$, $\chi_7 = 0.00245539$, $\chi_8 = 0.52499043$, $\chi_9 = 0.75104615$, $\chi_{10} = 1.29448910$, $\chi_{11} = -1.06038228$, $\chi_{12} = 0.64247743$, $\chi_{13} = -0.00932499$, $\chi_{14} = 1.88486768$.

5. Perform Algorithm N.8 to find the impulse response, $h_m$.

The Octave script *zolotarev_vlcek_zahradnik_test.m* reproduces *Vlček* and *Zahradník*'s Figures 1 to 4 [132], illustrating the design of of an almost equi-ripple FIR filter with $p = 4$, $q = 11$ and $\kappa = 0.75$. Figure N.89 shows the iso-extremal function, $\sqrt{1-w^2}S_{4,11}(w, 0.75)$, Figure N.90 shows the generating function, Figure N.91 shows zero-phase transfer function and Figure N.92 shows the amplitude frequency response. The coefficients of the expansion of the zero-phase response in Chebyshev polynomials of the first kind found with Algorithm N.8 [132, Table 4] are:

```
a_4_11 = [   0.3018151112,   0.4512145672,   0.2822759289,   0.0868799820, ...
            -0.0561809419,  -0.1053948923,  -0.0720513180,  -0.0045415003, ...
             0.0453863810,   0.0518476231,   0.0229242778,  -0.0132284961, ...
            -0.0312378770,  -0.0240752981,  -0.0030110098,   0.0144231969, ...
             0.0529542655 ];
```

The corresponding $z$-domain discrete-time impulse response coefficients are:

```
h_4_11 = [   0.0264771327,   0.0072115984,  -0.0015055049,  -0.0120376491, ...
            -0.0156189385,  -0.0066142480,   0.0114621389,   0.0259238116, ...
             0.0226931905,  -0.0022707502,  -0.0360256590,  -0.0526974461, ...
            -0.0280904709,   0.0434399910,   0.1411379644,   0.2256072836, ...
             0.3018151112,   0.2256072836,   0.1411379644,   0.0434399910, ...
            -0.0280904709,  -0.0526974461,  -0.0360256590,  -0.0022707502, ...
             0.0226931905,   0.0259238116,   0.0114621389,  -0.0066142480, ...
            -0.0156189385,  -0.0120376491,  -0.0015055049,   0.0072115984, ...
             0.0264771327 ];
```

The Octave script *zolotarev_vlcek_zahradnik_test.m* also reproduces *Vlček* and *Zahradník*'s Figure 5, shown here as Figure N.93.

Finally, the Octave script *zolotarev_vlcek_zahradnik_test.m* follows the procedure of *Vlček* and *Zahradník* to design a low-pass FIR filter with $a_{s_{dB}} = -120$, $fp = 0.15$ and $fs = 0.175$ [132, Section 8]. For this filter $n = 162$, $p = 53$, $q = 109$ and the filter length is 327. The value of the elliptic modulus calculated in the script, $\kappa = 0.5605516759$, differs slightly from that reported by *Vlček* and *Zahradník*, $\kappa = 0.55830966$. This may be due to round-off error in the given values of $\chi$. The frequency response of the filter is shown in Figure N.94.

[t] I found that, for a filter with $a_{s_{dB}} = -100$, $f_p = 0.24$ and $f_s = 0.25$, the degree equation of *Vlček* and *Zahradník* [132, Equation 23] grossly over-estimates the required $n$ as $n = 336$ when $n = 198$ is sufficient.

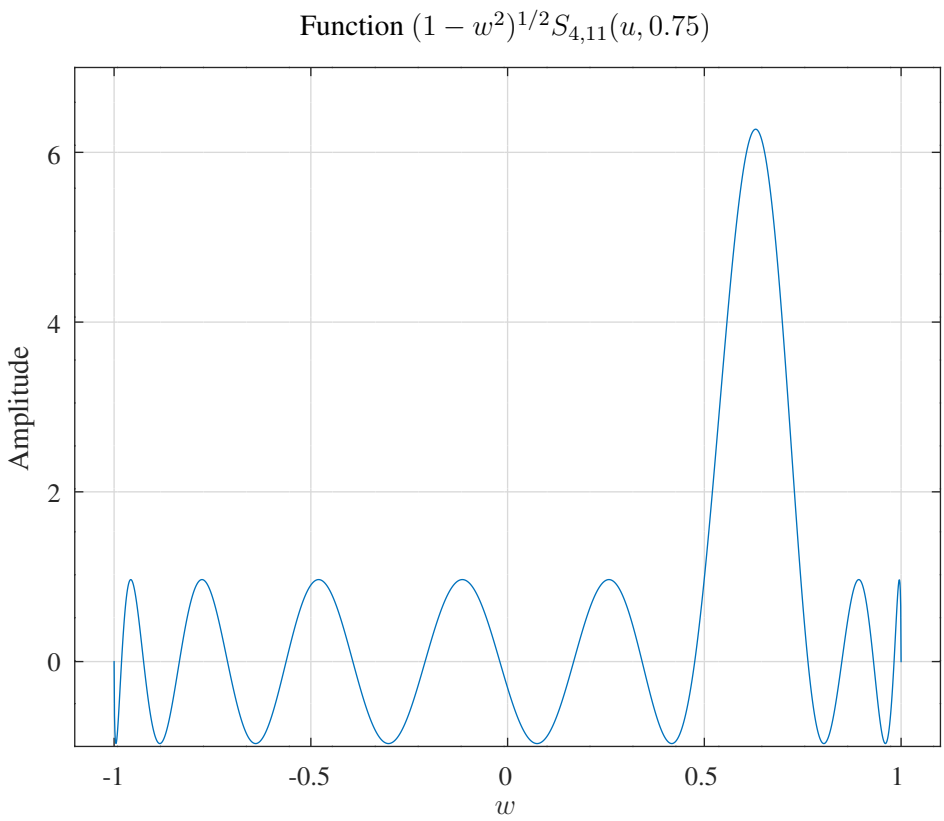



Figure N.89: Iso-extremal function $\sqrt{1-w^2}S_{4,11}\left(w,0.75\right)$, corresponding to Figure 1 of *Vlček* and *Zahradník* [132], $p=4$, $q=11$, $\kappa=0.75$

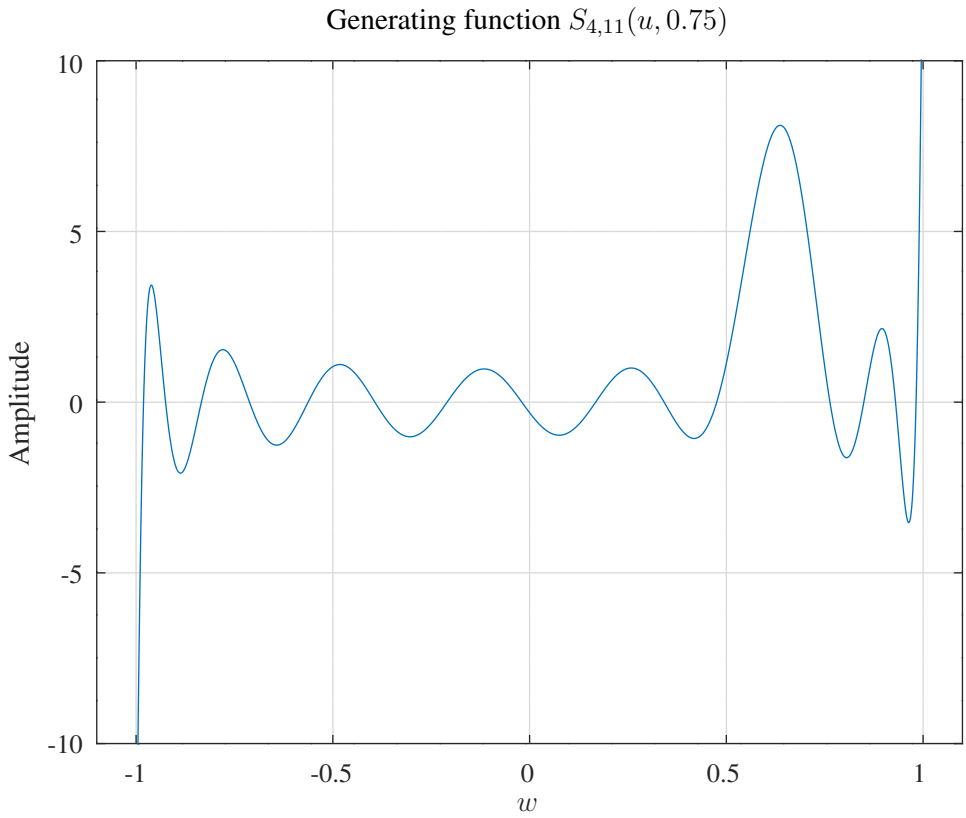



Figure N.90: FIR filter generating function corresponding to Figure 2 of *Vlček* and *Zahradník* [132], $p=4$, $q=11$, $\kappa=0.75$

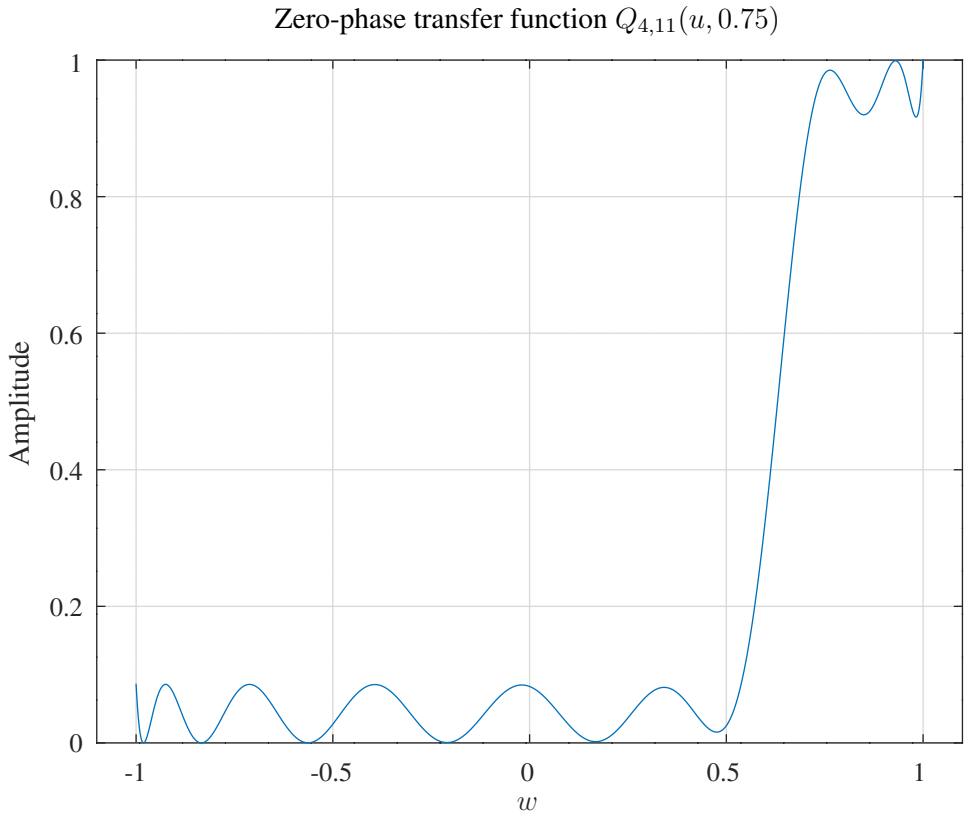



Figure N.91: FIR filter zero-phase response corresponding to Figure 3 of *Vlček* and *Zahradník* [132], $p = 4$, $q = 11$, $\kappa = 0.75$

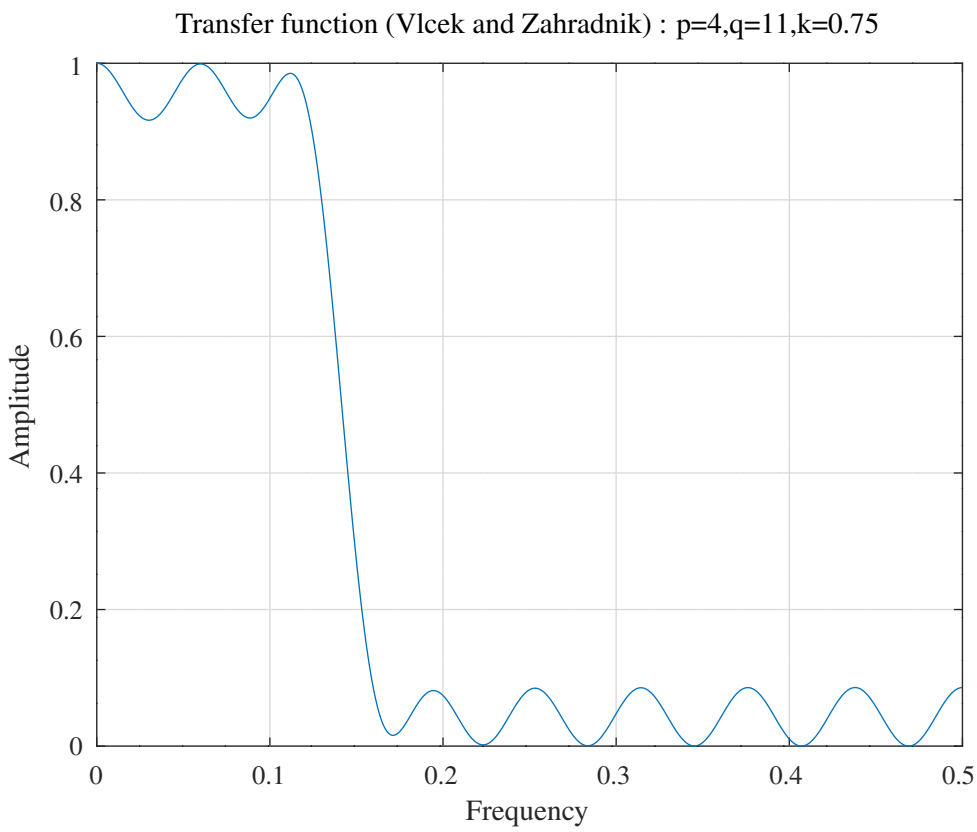



Figure N.92: FIR filter amplitude response corresponding to Figure 4 of *Vlček* and *Zahradník* [132], $p = 4$, $q = 11$, $\kappa = 0.75$

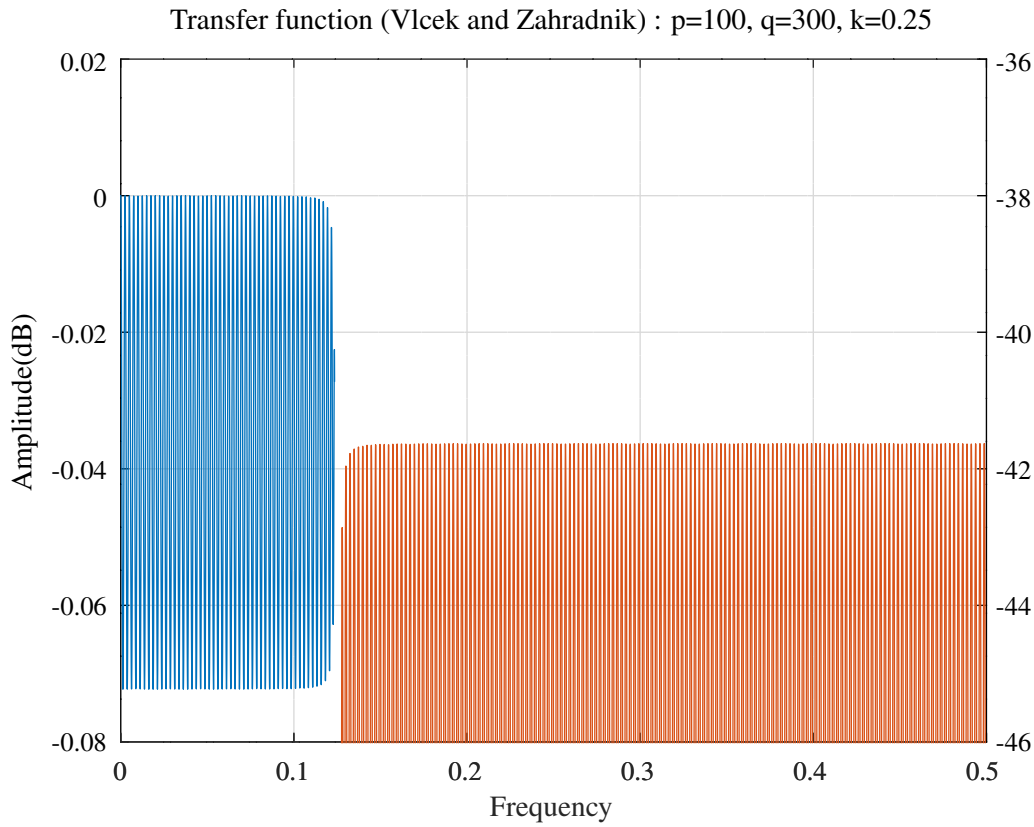



Figure N.93: FIR filter amplitude response corresponding to Figure 5 of *Vlček* and *Zahradník* [132], $p = 100$, $q = 300$, $\kappa = 0.25$

Figure N.94: FIR filter amplitude response of a low-pass FIR filter with $a_{s_{dB}} = -120$, $fp = 0.15$ and $fs = 0.175$ designed with the procedure of *Vlček* and *Zahradník* [132, Section 8]

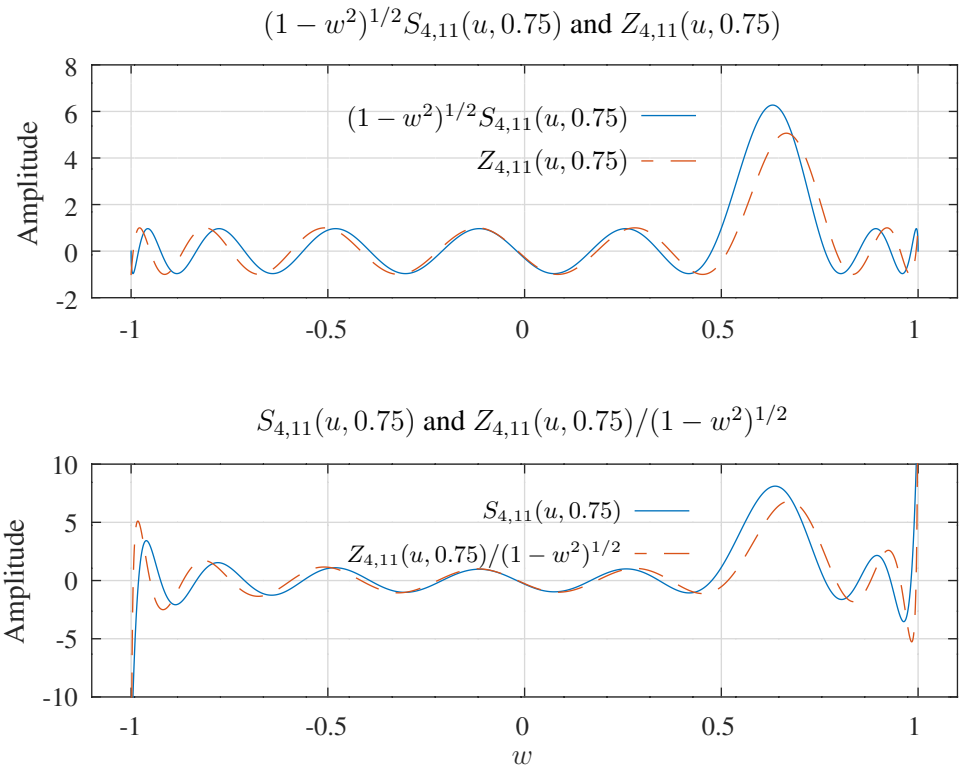



Figure N.95: Comparison of the iso-extremal function, $\sqrt{1-w^2}S_{4,11}(w,0.75)$, the Zolotarev polynomial, $Z_{4,11}(w,0.75)$, calculated by the Chebyshev expansion by *Vlček* and *Unbehauen* [127], the generating function, $S_{4,11}(w,0.75)$, of *Vlček* and *Zahradník* [132] and the modified Zolotarev polynomial, $Z_{4,11}(w,0.75)/\sqrt{1-w^2}$.

**Addendum**

I can successfully design filters with Table 4 of *Vlček* and *Zahradník* [132]. On the other hand, I find that the generating function, $S_{p,q}(w,\kappa)$, calculated by Table 4, does not satisfy their Equation 17. Figure N.95, created by the Octave script *zolotarev_vlcek_zahradnik_test.m*, compares the Zolotarev polynomial, $Z_{4,11}(w,0.75)$, found by direct calculation [220], the modified Zolotarev polynomial, $Z_{4,11}(w,0.75)/\sqrt{1-w^2}$, the function, $\sqrt{1-w^2}S_{4,11}(w,0.75)$, and the generating polynomial $S_{4,11}(w,0.75)$.

I am not sure what *Vlček* and *Zahradník* [132, p.747] mean by "the second solution of (14) in the form $\sqrt{1-w^2}S_{p,q}(w,\kappa)$". Initially, I assumed that they are referring to the possibility of a second, linearly independent solution of Equation N.41[u]. However, after examining Figure N.89 [132, Figure 1], it is clear that $\sqrt{1-w^2}S_{p,q}(w,\kappa)$ is not a solution of Equation N.40, as it is zero rather than $\pm1$ at $w=\pm1$, as shown for $Z_{5,9}(w,0.78)$ in Figure N.77. *Vlček* and *Zahradník*'s Table 4 shows that the main lobe edge is modified to $u_0=\frac{p+\frac{1}{2}}{n+1}K(\kappa)$. This suggests that the $n^2$ term in Equation N.41 should be replaced by $(n+1)^2$. The Maxima script *zolotarev_vlcek_zahradnik_test_table_4.max* substitutes $\sqrt{1-w^2}f(w)$ into Equation N.41 with this modification. The resulting differential equation is[v]:

$$g_2(w)\left[\left(1-w^2\right)\frac{d^2f}{dw^2}-3w\frac{df}{dw}\right]-g_1(w)\left(1-w^2\right)\frac{df}{dw}+\left[(n+1)^2g_0(w)+wg_1(w)-g_2(w)\right]f=0 \tag{N.52}$$

The Octave script *zolotarev_vlcek_zahradnik_test.m* shows that the $S_{p,q}(w,\kappa)$ function calculated by Table 4 of *Vlček* and *Zahradník* satisfies this differential equation.

The Chebyshev polynomials of the second kind, $U_m(w)$, satisfy the differential equation:

$$\left(1-w^2\right)\frac{d^2U_m(w)}{dw^2}-3w\frac{dU_m(w)}{dw}+m(m+2)U_m(w)=0$$

Differentiating $U_m(w)$ gives:

$$\left(1-w^2\right)\frac{dU_m(w)}{dw}=(m+2)wU_m(w)-(m+1)U_{m+1}(w)$$

[u]For example, see *Morse* and *Feshbach* [146, Section 5.2].

[v]Consistent with Equation N.42 rather than following the changes in notation from *Vlček* and *Unbehauen* [127, Equation 64] to *Vlček* and *Unbehauen* [132, Equation 15].

The Chebyshev polynomials of the second kind obey the following relations:

$$\begin{aligned}
2wU_m(w) &= U_{m+1}(w) + U_{m-1}(w) \\
4w^2U_m(w) &= U_{m+2}(w) + 2U_m(w) + U_{m-2}(w) \\
8w^3U_m(w) &= U_{m+3}(w) + 3U_{m+1}(w) + 3U_{m-1}(w) + U_{m-3}(w) \\
&\vdots
\end{aligned}$$

The linear differential equation for $S_{p,q}(w,\kappa)$ can be solved by substituting an expansion in Chebyshev polynomials of the second kind:

$$S_{p,q}(w) = \sum_{m=0}^{n} a_m^+ U_m(w)$$

so that:

$$\begin{aligned}
&- g_2(w) \sum_{m=0}^{n} m(m+2)\, a_m^+ U_m(x) \quad \ldots \\
&- g_1(w) \sum_{m=0}^{n} a_m^+ \left[(m+2)\, wU_m(w) - (m+1)\, U_{m+1}(w)\right] \quad \ldots \\
&+ \left[(n+1)^2 g_0(w) + wg_1(w) - g_2(w)\right] \sum_{m=0}^{n} a_m^+ U_m(w) = 0
\end{aligned} \tag{N.53}$$

The Maxima script *zolotarev_vlcek_zahradnik_table_4.max* calculates the recurrence relations from Equation N.53. The results agree with Table 4 of *Vlček* and *Zahradník* [132] when inserted into the Octave function *zolotarev_vlcek_zahradnik.m*.

## N.12 Design of FIR filters as a tapped cascade of sub-filters

This chapter describes the design of FIR filters as a series cascade of sub-filters. See, for example, *Saramäki* [203], *Shiung et al.* [39], *Smith* [102] or *Lim* and *Liu* [223]. The cascade connection permits an improved response and/or reduced complexity by partition of the FIR filter or the use of fewer multipliers.

*Saramäki* [203] describes a the design of a tapped cascade of identical sub-filters by "mapping a prototype filter into a composite filter by means of a frequency travsformation. The transformation determines the sub-filter, whereas the prototype filter determines the tap coefficients." He claims that his proposed approach results in an overall filter order that is approximately 30% higher than the order, $L$, of the correponding mini-max direct-form FIR filter but has approximately $\sqrt{2.6L}$ multipliers. *Saramäki* [203, Section V] gives experimental results for the effect of the number of sub-filters on performance measures such as overall filter order, number of distinct multipliers, number of delay elements, round-off error and coefficient sensitivity.

### N.12.1 Transformations of linear-phase FIR filters

The transfer function, $H(Z)$, of a linear-phase FIR filter with impulse response $h_n$ of length $2N+1$, and symmetry $h_{N-n} = h_n$ can be expressed in terms of the *zero-phase* transfer function, $\tilde{H}_0(Z)$, as:

$$\begin{aligned}\tilde{H}_0(Z) &= z^{-N} H(Z) \\ &= h_N + \sum_{n=1}^{N} h_{N-n}\left[Z^n + Z^{-n}\right]\end{aligned}$$

This can be re-written as:

$$\begin{aligned}\tilde{H}_0(v) &= h_N + \sum_{n=1}^{N} 2h_{N-n} T_n(v) \\ &= \sum_{n=0}^{N} a_n v^n\end{aligned}$$

where $v = \frac{1}{2}\left(Z + Z^{-1}\right)$ and $T_n(v)$ are the Chebychev polynomials of the first kind, defined by $T(\cos\theta) = \cos n\theta$. In the following, $\tilde{H}_0(v)$ is also called the *prototype* filter.

$\tilde{H}_0(v)$ can be converted to another transfer function by substitution of a zero-phase sub-filter, $v = \bar{H}_M(z)$, where:

$$\begin{aligned}H_M(z) &= \sum_{r=0}^{2M} \bar{h}_r z^{-r} \\ \bar{H}_M(z) &= z^M H_M(z) \\ &= \left[\bar{h}_M + \sum_{r=0}^{M} \bar{h}_{M-r}\left(z^r + z^{-r}\right)\right]\end{aligned}$$

The overall zero-phase transfer function is then:

$$H_0(z) = \sum_{n=0}^{N} a_n \left[\bar{H}_M(z)\right]^n \tag{N.54}$$

This is an order $NM$ polynomial in $\frac{1}{2}\left(z + z^{-1}\right)$ with a corresponding impulse response of length $2NM+1$.

#### Implementations of transformed FIR filters

*Saramäki* [203, Figure 1] shows several implementations of Equation N.54. The first is [203, Figure 1a and Equation 6]:

$$\begin{aligned}H(z) &= z^{-NM} H_0(z) \\ &= \sum_{n=0}^{N} a_n z^{-(N-n)M}\left[H_M(z)\right]^n\end{aligned} \tag{N.55}$$

An alternative implementation is derived from the recursion relation for Chebychev polynomials of the first kind:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ &\vdots \\ T_n(x) &= 2xT_{n-1}(x) - T_{n-2}(x) \end{aligned}$$

Substitution gives [203, Figure 1c and Equation 9]:

$$\begin{aligned} H(z) &= h_N z^{-NM} G_0(z) + \sum_{n=1}^{N} 2h_{N-n} z^{-(N-n)M} G_n(z) \\ G_n(z) &= z^{-nM} T_n\left(z^M H_M(z)\right) \end{aligned}$$

The transfer functions $G_n(z)$ can be implemented recursively as:

$$\begin{aligned} G_0(z) &= 1 \\ G_1(z) &= H_M(z) \\ G_n(z) &= 2H_M(z)\, G_{n-1}(z) - z^{-2M} G_{n-2}(x) \end{aligned}$$

**Frequency domain relations**

Setting $Z = e^{\imath\Omega}$ and $z = e^{\imath\omega}$, the zero-phase frequency responses of the prototype filter, $\tilde{H}_0(Z)$, and transformed filter, $H_0(z)$, are:

$$\begin{aligned} \tilde{H}_0\left(e^{\imath\Omega}\right) &= \sum_{n=0}^{N} a_n \left[\cos\Omega\right]^n \\ H_0\left(e^{\imath\omega}\right) &= \sum_{n=0}^{N} a_n \left[P_M(\omega)\right]^n \end{aligned}$$

where the zero-phase frequency response of the sub-filter, $H_M(z)$, is:

$$P_M(\omega) = \bar{h}_M + \sum_{r=1}^{M} 2\bar{h}_{M-r} \cos r\omega$$

$\tilde{H}_0\left(e^{\imath\Omega}\right)$ and $H_0\left(e^{\imath\omega}\right)$ are related by:

$$\Omega = g_M(\omega) = \arccos P_M(\omega)$$

If $0 \leq g_M(\omega) \leq \pi$ and $0 \leq \omega \leq \pi$ this transformation preserves the amplitude characteristics of $\tilde{H}_0\left(e^{\imath\Omega}\right)$ and distorts the frequency axis [203, Figure 2].

## N.12.2 Frequency-domain constraints on the prototype and sub-filter

**Specifications Based on a Normalized Prototype Filter**

Let the specifications of $H_0\left(e^{\imath\omega}\right)$ in the pass-bands, $I_p$, and stop-bands, $I_s$, be:

$$\begin{aligned} 1 - \delta_p \leq H_0\left(e^{\imath\omega}\right) \leq 1 + \delta_p \quad \omega \in I_p \\ -\delta_s \leq H_0\left(e^{\imath\omega}\right) \leq \delta_s \quad \omega \in I_s \end{aligned} \tag{N.56}$$

The specifications for the prototype filter, $\tilde{H}_0\left(e^{\imath\Omega}\right)$, are:

$$\begin{aligned} 1 - \delta_p \leq \tilde{H}_0\left(e^{\imath\Omega}\right) \leq 1 + \delta_p \quad 0 \leq \Omega \leq \Omega_p \\ -\delta_s \leq \tilde{H}_0\left(e^{\imath\Omega}\right) \leq \delta_s \quad \Omega_s \leq \Omega \leq \pi \end{aligned} \tag{N.57}$$

The requirements for $g_M(\omega)$ are:

$$\begin{aligned} 0 \leq g_M(\omega) \leq \Omega_p \quad & \text{for } \omega \in I_p \\ \Omega_s \leq g_M(\omega) \leq \pi \quad & \text{for } \omega \in I_s \end{aligned}$$

The requirements for the zero-phase response of the sub-filter are:

$$\begin{aligned} \cos\Omega_p \leq P_M(\omega) \leq 1 \quad & \text{for } \omega \in I_p \\ -1 \leq P_M(\omega) \leq \cos\Omega_s \quad & \text{for } \omega \in I_s \end{aligned} \tag{N.58}$$

For fixed values of $\Omega_p$, and $\Omega_s$, both the prototype filter and the sub-filter can be designed using minimax FIR design software. *Saramäki* calls this type of specification "normalised-prototype-filter-based"(NPFB).

### General specifications

The same overall frequency response can be obtained by replacing the sub-filter, $H_M(z)$, with:

$$\hat{H}_M(z) = AH_M(z) + Bz^{-M}$$

which has the zero-phase response:

$$\hat{P}_M(\omega) = AP_M(\omega) + B \tag{N.59}$$

The overall zero-phase frequency response becomes:

$$\hat{H}_0(e^{\imath\omega}) = \sum_{n=0}^{N} \hat{a}_n \left[\hat{P}_M(\omega)\right]^n$$

which corresponds to a frequency transformation:

$$x = A\cos\Omega + B \tag{N.60}$$

The prototype frequency response becomes:

$$\hat{H}_0(x) = \tilde{H}_0(e^{\imath\omega})|_{x=A\cos\Omega+B} = \sum_{n=0}^{N} \hat{a}_n x^n$$

The zero-phase frequency response is obtained from $\hat{H}_0(x)$ by the substitution $x = \frac{1}{2}\left(z + z^{-1}\right)$. Likewise, the prototype filter frequency response is obtained by the substitution $x = \cos\Omega$. $H_0(e^{\imath\omega})$ and $\hat{H}_0(x)$ are related by $x = \hat{P}_M(\omega)$. If $A > 0$, then Equation N.60 maps the pass-band, $[0, \Omega_p]$, and the stop-band, $[\Omega_s, \pi]$, of the $\Omega$-plane onto the $x$-plane regions $[x_{p1}, x_{p2}]$ and $[x_{s1}, x_{s2}]$, respectively, where:

$$\begin{aligned} x_{p1} &= \quad A\cos\Omega_p + B \\ x_{p2} &= \quad A + B \\ x_{s1} &= -A + B \\ x_{s2} &= \quad A\cos\Omega_s + B \end{aligned} \tag{N.61}$$

If $\tilde{H}_0\left(e^{\imath\Omega}\right)$ satisfies Equation N.57, then $1 - \delta_p \leq \tilde{H}_0(x) \leq 1 + \delta_p$ on $[x_{p1}, x_{p2}]$ and $-\delta_s \leq \tilde{H}_0(x) \leq \delta_s$ on $[x_{s1}, x_{s2}]$ [203, Figure 3]. Similarly, if $P_M$ satisfies Equation N.58, then the new sub-filter frequency response $\hat{P}_M(\omega)$ is within the limits $x_{p1}$ and $x_{p2}$ on $I_p$ and within the limits $x_{s1}$ and $x_{s2}$ on $I_s$. The simultaneous specifications for the prototype filter and sub-filter are [203, Figure 4]:

$$\begin{aligned} 1 - \delta_p \leq \hat{H}_0(x) \leq 1 + \delta_p \quad & \text{for } x \in [x_{p1}, x_{p2}] \\ \delta_s \leq \hat{H}_0(x) \leq \delta_s \quad & \text{for } x \in [x_{s1}, x_{s2}] \\ x_{p1} \leq \hat{P}_M(\omega) \leq x_{p2} \quad & \text{for } \omega \in I_p \\ x_{s1} \leq \hat{P}_M(\omega) \leq x_{s2} \quad & \text{for } \omega \in I_s \end{aligned} \tag{N.62}$$

The design of the sub-filter $\hat{P}_M(\omega)$ can be performed by standard FIR filter design algorithms. The design of the prototype filter, $\hat{H}_0(x)$ can be performed by using the transformation of Equation N.60 to map the problem to the $\Omega$-plane, designing the filter to meet the specifications of Equation N.57, then mapping the result back to the $x$-plane.

#### Specifications based on the normalised sub-filter

For the special case of $x_{p1} = 1-\hat{\delta}_p$, $x_{p2} = 1+\hat{\delta}_p$, $x_{s1} = -\hat{\delta}_s$ and $x_{s2} = \hat{\delta}_s$, *Saramäki* refers to the "normalised-sub-filter-based" (NFSB) specifications:

$$\begin{aligned} 1-\delta_p \leq \hat{H}_0(x) \leq 1+\delta_p \quad & \text{for } x \in \left[1-\hat{\delta}_p, 1+\hat{\delta}_p\right] \\ \delta_s \leq \hat{H}_0(x) \leq \delta_s \quad & \text{for } x \in \left[-\hat{\delta}_s, \hat{\delta}_s\right] \\ 1-\hat{\delta}_p \leq \hat{P}_M(\omega) \leq 1+\hat{\delta}_p \quad & \text{for } \omega \in I_p \\ -\hat{\delta}_s \leq \hat{P}_M(\omega) \leq \hat{\delta}_s \quad & \text{for } \omega \in I_s \end{aligned} \tag{N.63}$$

The NPFB and NSFB specifications are related by the transformation of Equation N.60 and the subsitution of Equation N.61 with $x_{p2} - x_{s1}$ and $x_{p2} + x_{s1}$ giving:

$$\begin{aligned} A &= \frac{1+\hat{\delta}_p+\hat{\delta}_s}{2} \\ B &= \frac{1+\hat{\delta}_p-\hat{\delta}_s}{2} \end{aligned}$$

$\cos\Omega_p$ and $\cos\Omega_s$ are related to $\hat{\delta}_p$ and $\hat{\delta}_p$ by the relative proportions:

$$\begin{aligned} \frac{1-\cos\Omega_p}{2} &= \frac{2\hat{\delta}_p}{1+\hat{\delta}_p+\hat{\delta}_s} \\ \frac{1+\cos\Omega_s}{2} &= \frac{2\hat{\delta}_s}{1+\hat{\delta}_p+\hat{\delta}_s} \end{aligned}$$

so that:

$$\begin{aligned} \cos\Omega_p &= \frac{1+\hat{\delta}_s-3\hat{\delta}_p}{1+\hat{\delta}_p+\hat{\delta}_s} \\ \cos\Omega_s &= \frac{3\hat{\delta}_s-\hat{\delta}_p-1}{1+\hat{\delta}_p+\hat{\delta}_s} \end{aligned}$$

Alternatively, given $\cos\Omega_p$ and $\cos\Omega_s$:

$$\begin{bmatrix} 3+\cos\Omega_p & -1+\cos\Omega_p \\ 1+\cos\Omega_s & -3+\cos\Omega_s \end{bmatrix} \begin{bmatrix} \hat{\delta}_p \\ \hat{\delta}_s \end{bmatrix} = \begin{bmatrix} 1-\cos\Omega_p \\ -1-\cos\Omega_s \end{bmatrix}$$

### N.12.3 Filter design

*Saramäki* [203, Section IV] considers four separate design problems.

#### Approximation Problem I

Given the composite filter specifications of Equation N.56 and $N$, the number of sub-filters, find the tap coefficients and the sub-filter such that the sub-filter order $2M$ is minimised.

*Saramäki* suggests that this approximation problem may be solved by:

1. Given the prototype filter order, $2N$, find the minimum sub-filter order, $2M$, which is required by the overall filter to meet the specification of Equation N.56. This can be done by choosing the prototype filter pass-band edge, $\Omega_p$ as the unknown and, for each $\Omega_p$ finding the prototype filter with the minimum $\Omega_s$ that satisfies Equation N.57.
2. Find the coefficients of the sub-filter of the resulting order $2M$ and the coefficients of the prototype filter of the given order $2N$ to minimise the absolute value of the error function on $I_p \cup I_s$.

Minimising $\Omega_s$ maximises the allowable variation of $P_M(\omega)$ on $I_s$ and, consequently, the minimum required sub-filter order to meet Equation N.57. The solution is obtained by reducing $\Omega_s$ until the given ripple ratio $\frac{\delta_p}{\delta_s}$ attains the specified maximum value. Alternatively, the *Selesnik-Burrus* modification to the *Hofstetter* low-pass filter design method, shown in Section N.6.2, finds the filter that minimises $\Omega_s$ for the given $\Omega_p$, $\delta_p$ and $\delta_s$.

**Approximation Problem II**

Given the composite filter specifications of Equation N.56 and the sub-filter order $2M$, find the tap coefficients and the sub-filter such that $N$, the number of sub-filters, is minimised.

*Saramäki* suggests this approximation problem may be solved by:

1. Given the NPFB specifications of Equations N.58 and N.57, increase $N$ until, at least at one extraripple solution of the prototype sub-filter, the given sub-filter order, $2M$, is large enough to meet the specification of Equation N.58. Denote by $\Omega_p^{(i)}$ and $\Omega_s^{(i)}$, $i = 1, 2, \ldots, r$, the pass-band and stop-band edge angles of those extra-ripple solutions at which the sub-filter meets the criteria. At this point, the minimum number of sub-filters is known and both the prototype filter and sub-filter orders $2N$ and $2M$ are known.
2. Given the NSFB specifications of Equation N.63, consider the ripple ratio $k = \frac{\hat{\delta}_p}{\hat{\delta}_s}$ as a primary unknown. Find the value of $k$ near each of the points

$$k_0^{(i)} = \frac{1 - \cos \Omega_p^{(i)}}{1 + \cos \Omega_s^{(i)}} \ , \quad i = 1, \ldots, r$$

that minimises the passband ripple, $\delta_p$, of $\hat{H}_0(x)$ for the given ripple ratio $\frac{\delta_p}{\delta_s}$. Select the resulting value of $k_0^{(i)}$ giving the smallest pass-band ripple.

**Prescribed Subfilter**

The sub-filter may be determined in advance, perhaps to reduce hardware requirements. In this case, *Saramäki* suggests that the tap coefficients be optimised to reduce the number of sub-filters, $N$, by:

1. Determine the maximum and minimum values of the sub-filter frequency response $\hat{P}_M(\omega)$ on $I_p$, denoted by $x_{p1}$ and $x_{p2}$. Similarly, determine the maximum and minimum values $x_{s1}$ and $x_{s2}$ on $I_s$.
2. Determine $\hat{H}_0(x)$ to satisfy the specifications of Equation N.62 with the minimum value of $N$:
   (a) Determine $A$ and $B$ and then $\Omega_p$ and $\Omega_s$ from Equation N.61.
   (b) Design the prototype filter of minimum even-order $2N$ to meet the specification of Equation N.57.
   (c) Transform the resulting frequency response to the $x$-plane with Equation N.60.

**Prescribed tap coefficients**

*Saramäki* suggests that if the prototype frequency response, $\hat{H}_0(x)$ is given:

1. Determine the regions $X_p$ and $X_s$ where $1 - \delta_p \leq \hat{H}_0(x) \leq 1 + \delta_p$ and $\delta_s \leq \hat{H}_0(x) \leq \delta_s$, respectively. Let $x_{p1}$ and $x_{p2}$ ($x_{s1}$ and $x_{s2}$) denote the maximum (minimum) values of $X_p$ ($X_s$).
2. Find the minimum even-order sub-filter to satisfy the specification of Equation N.62.

### N.12.4 Filter design examples

#### Approximation Problem I low-pass filter example

*Saramäki*'s Approximation Problem I assumes that the prototype filter order, $2N$, or number of sub-filters, $N$, is given. First, given the prototype filter pass-band edge, $\Omega_p$, search for the minimum prototype filter transition width, $\Delta\Omega$, that satisfies the amplitude specifications, $\delta_p$ and $\delta_s$. Next, search for the minimum sub-filter order, $2M$, that, given $\Omega_p$ and $\Delta\Omega$, satisfies the overall filter frequency specifications $\omega \in I_p$ and $\omega \in I_p$. The Octave script *saramakiFIRcascade_ApproxI_lowpass_test.m* attempts to reproduce the design shown in *Saramäki*'s Figure 5 [203, Figure 5][w]. In this script, both searches are made with the Octave function *selesnickFIRsymmetric_lowpass*, an implementation of *Hofstetter*'s low-pass filter design method with the *Selesnik-Burrus* modification shown in Section N.6.2. Table N.1 shows the specifications and derived parameters for this example of Approximation Problem I. The minimum sub-filter order found is $M = 23$ and the overall filter impulse response length is $2NM + 1 = 277$. Figure N.96 compares the minimum sub-filter order, $2M$, and the minimum prototype filter transition width, $\Delta\Omega$, found with *selesnickFIRsymmetric_lowpass*. Figure N.97 is a plot, similar to that of *Saramäki*'s Figure 6, showing the mapping of the prototype filter to the sub-filter and the overall response. Figure N.98 shows the pass-band and stop-band of the filter.

| | |
|---|---|
| N | 6 |
| $f_p$ | 0.1375 |
| $f_s$ | 0.15 |
| $\delta_p$ | 0.01 |
| $\delta_s$ | 0.001 |
| $M$ | 23 |
| $2NM+1$ | 277 |
| $\Omega_p/2\pi$ | 0.146000 |
| $\Omega_s/2\pi$ | 0.338541 |
| A | 0.637756 |
| B | 0.487266 |
| $\hat{\delta}_p$ | 0.125022 |
| $\hat{\delta}_s$ | 0.150490 |

Table N.1: Parameters of *Saramäki* FIR cascade Approximation Problem I example filter.

The $N + 1$ distinct coefficients of the prototype filter are:

```
hN = [   0.0107774477,   0.0177655796,  -0.0238617114,  -0.0700664677, ...
         0.0371535282,   0.3045732560,   0.4573167353 ]';
```

The $N + 1$ tap coefficients, $a_n$ in Equation N.55, are:

```
aN = [   0.3137313609,   1.2072011137,   0.9183896108,  -1.2711549242, ...
        -1.4164223584,   0.5684985464,   0.6897566509 ];
```

The $M + 1$ distinct coefficients of the sub-filter are:

```
hM = [   0.0349913571,   0.1139326345,  -0.0329200706,  -0.0100592129, ...
        -0.0267514167,  -0.0120584706,   0.0100933358,   0.0284182665, ...
         0.0262381556,   0.0024703229,  -0.0273071240,  -0.0398277551, ...
        -0.0215731935,   0.0186827541,   0.0524233219,   0.0497981852, ...
         0.0028464710,  -0.0626863843,  -0.0973752026,  -0.0565442512, ...
         0.0694019272,   0.2424103979,   0.3919484265,  -0.3131049496 ]';
```

[w] The plot shown in *Saramäki*'s Figure 6 has the same filter parameters except for $N = 7$.

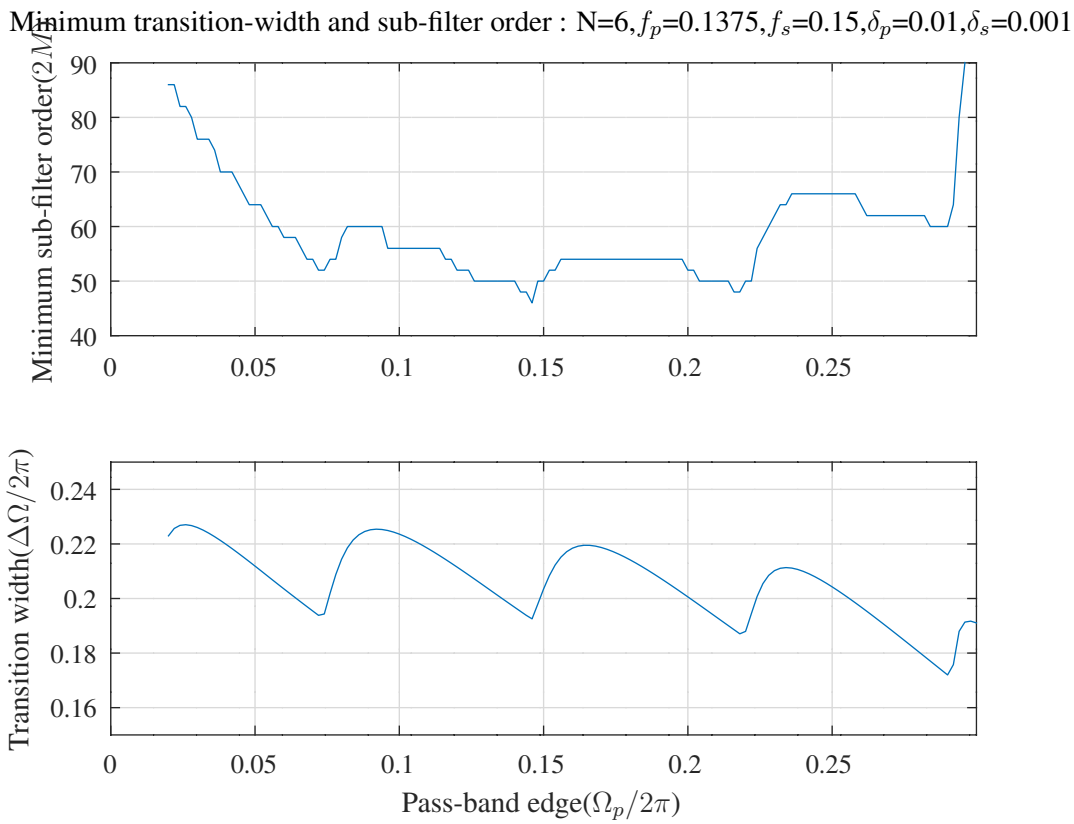



Figure N.96: Comparison of the minimum sub-filter order, $2M$, and the minimum prototype filter transition width, $\Delta\Omega$, found with *selesnickFIRsymmetric_lowpass* [203, Figure 5].

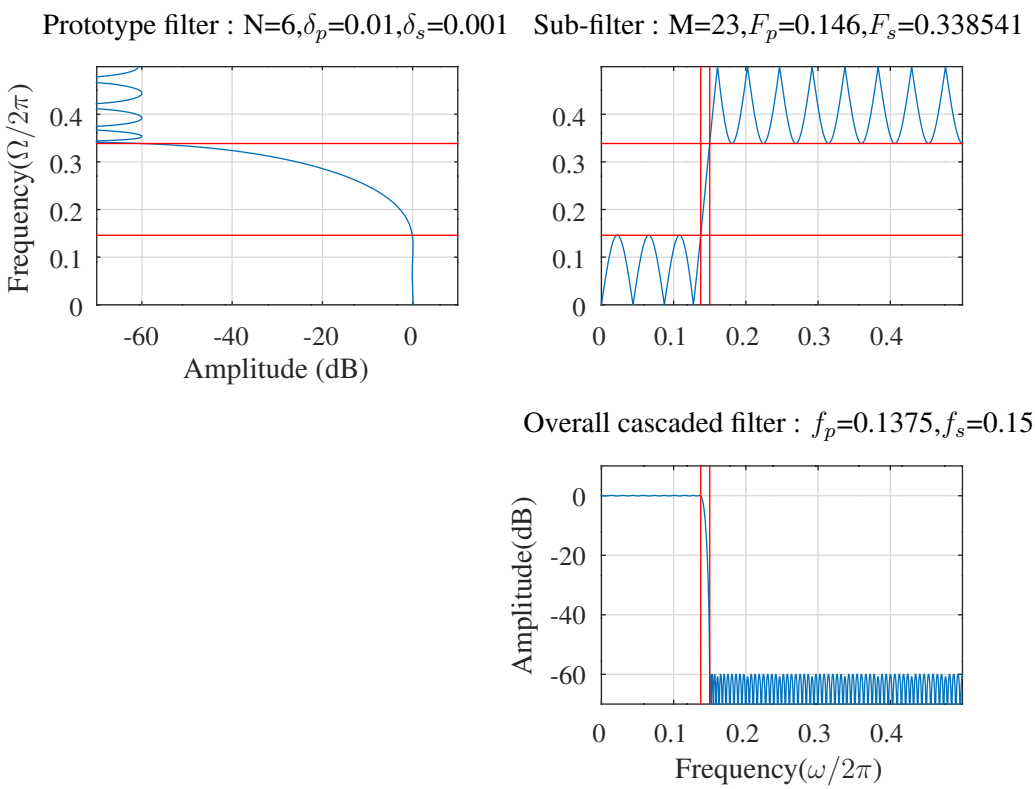



Figure N.97: Approximation problem I solution mapping the prototype filter to the sub-filter and the overall response.

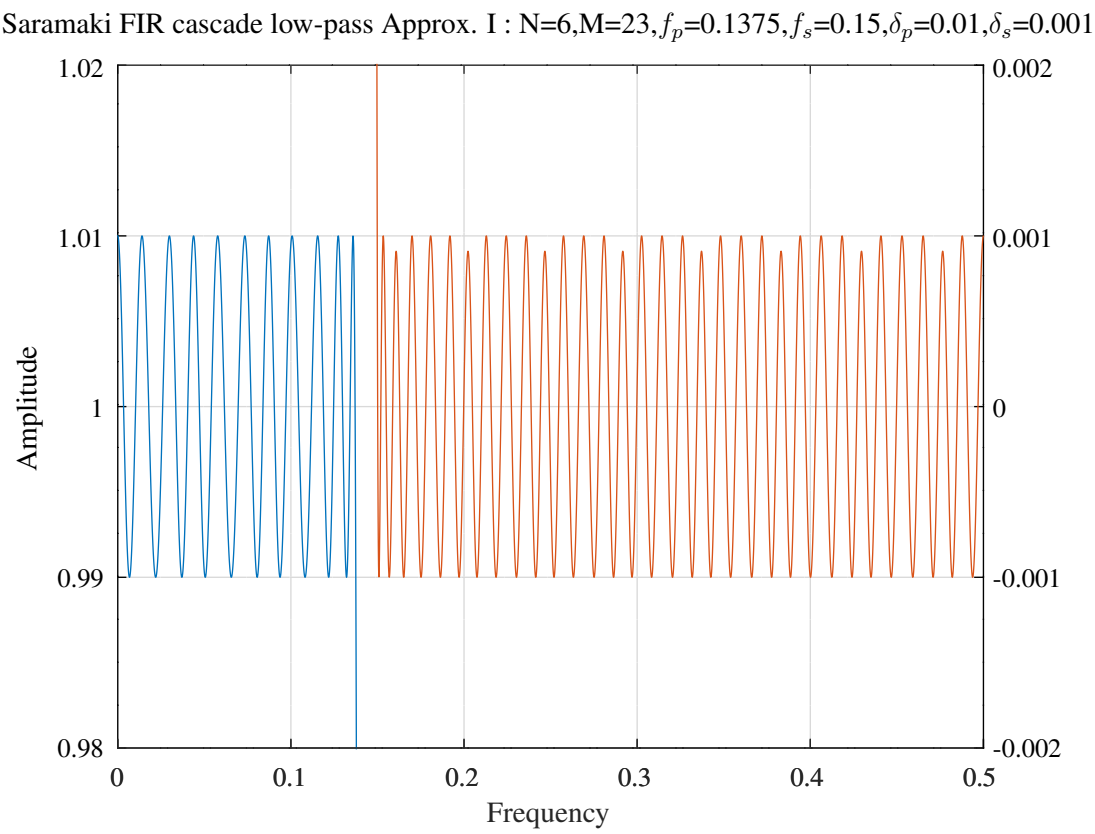



Figure N.98: Detailed view of the pass-band and stop-band response of the Approximation Problem I low-pass filter example.

**Approximation Problem II multi-band filter example**

*Saramäki*'s Approximation Problem II assumes that the sub-filter order, $2M$, is given. The Octave script *saramakiFIRcascade_ApproxII_multiband_test.m* attempts to reproduce *Saramäki*'s multi-band example [203, Figure 17 and Section VI]. Table N.2 shows the specifications and derived parameters for this example of Approximation Problem II.

| | |
|---|---|
| M | 25 |
| $f_{su1}$ | 0.0625 |
| $f_{pl1}$ | 0.075 |
| $f_{pu1}$ | 0.125 |
| $f_{sl2}$ | 0.1375 |
| $f_{su2}$ | 0.2375 |
| $f_{pl2}$ | 0.25 |
| $f_{pu2}$ | 0.4 |
| $f_{sl3}$ | 0.4125 |
| $\delta_p$ | 0.01 |
| $\delta_s$ | 0.001 |
| $N$ | 6 |
| $2NM+1$ | 301 |
| $K$ $(W_s/W_p)$ | 0.838000 |
| $E$ (RMS error) | 0.003856 |
| $\Omega_p/2\pi$ | 0.145947 |
| $\Omega_s/2\pi$ | 0.339365 |
| $\Omega_{sN}/2\pi$ | 0.338367 |
| A | 0.636811 |
| B | 0.487942 |
| $\hat{\delta}_p$ | 0.135959 |
| $\hat{\delta}_s$ | 0.141037 |

Table N.2: Parameters of *Saramäki* FIR cascade Approximation Problem II multi-band example filter.

The script sets the ratio of stop-band to pass-band weights $K = \frac{W_s}{W_p}$ and calls the Octave function *mcclellanFIRsymmetric* to design an order $2M = 50$ multi-band sub-filter that satisfies the frequency specifications, $\omega_{su1}$ etc., of the filter pass-bands and stop-bands. If the sub-filter is feasible then the script next searches for the minimum order $2N$ prototype filter that satisfies the overall filter $\delta_p$ and $\delta_s$ amplitude response ripple specifications. The script does this by calling the Octave function *selesnickFIRsymmetric_lowpass* with fixed prototype filter pass-band edge frequency, $\Omega_p$ at $1-\delta_p$, and increasing $N$ until the resulting prototype filter stop-band edge frequency, $\Omega_{sN}$, satisfies the value of $\Omega_s$ corresponding to the stop-band ripple of the sub-filter. A final design is selected from the set of feasible designs by choosing the design with the lowest RMS error in the amplitude response. The sub-filter order is given as $2M = 50$. The minimum prototype filter order found is $2N = 12$ and the overall filter impulse response length is $2NM+1 = 301$. Figure N.99 is a plot, similar to that of *Saramäki*'s Figure 17, showing the mapping of the prototype filter to the sub-filter and the overall response. Figure N.100 shows the pass-band and stop-band of the filter.

The $N+1$ distinct coefficients of the prototype filter are:

```
hN = [   0.0108151088,   0.0177369180,  -0.0239519611,  -0.0700472296, ...
         0.0373113899,   0.3045551067,   0.4571405151 ]';
```

The $N+1$ tap coefficients, $a_n$ in Equation N.55, are:

```
aN = [   0.3129835955,   1.2067627712,   0.9218208532,  -1.2698545571, ...
        -1.4214818200,   0.5675813761,   0.6921669615 ];
```

The $M+1$ distinct coefficients of the sub-filter are:

```
hM = [   0.0387992748,   0.0340758453,   0.0666153212,  -0.0563330856, ...
        -0.0901452151,   0.0304951584,  -0.0471249545,   0.0082373858, ...
        -0.0481509242,   0.0254690581,   0.0214454822,  -0.0595864252, ...
        -0.0073808648,   0.0235659319,   0.1153460758,   0.1064478219, ...
        -0.0027298992,   0.1310680361,  -0.0288847206,  -0.1033224269, ...
        -0.2393709560,  -0.2025373728,   0.2770127253,  -0.1806058042, ...
        -0.0554313449,  -0.0464020739 ]';
```

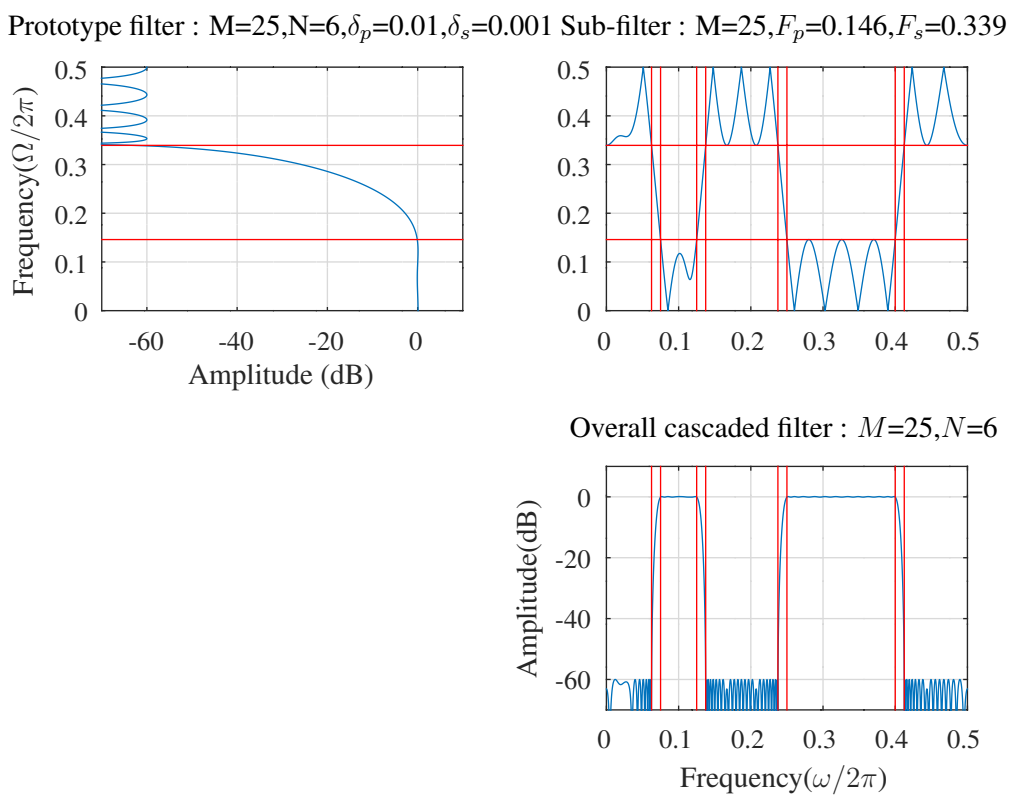



Figure N.99: Approximation problem II solution mapping the prototype filter to the sub-filter and the overall response.

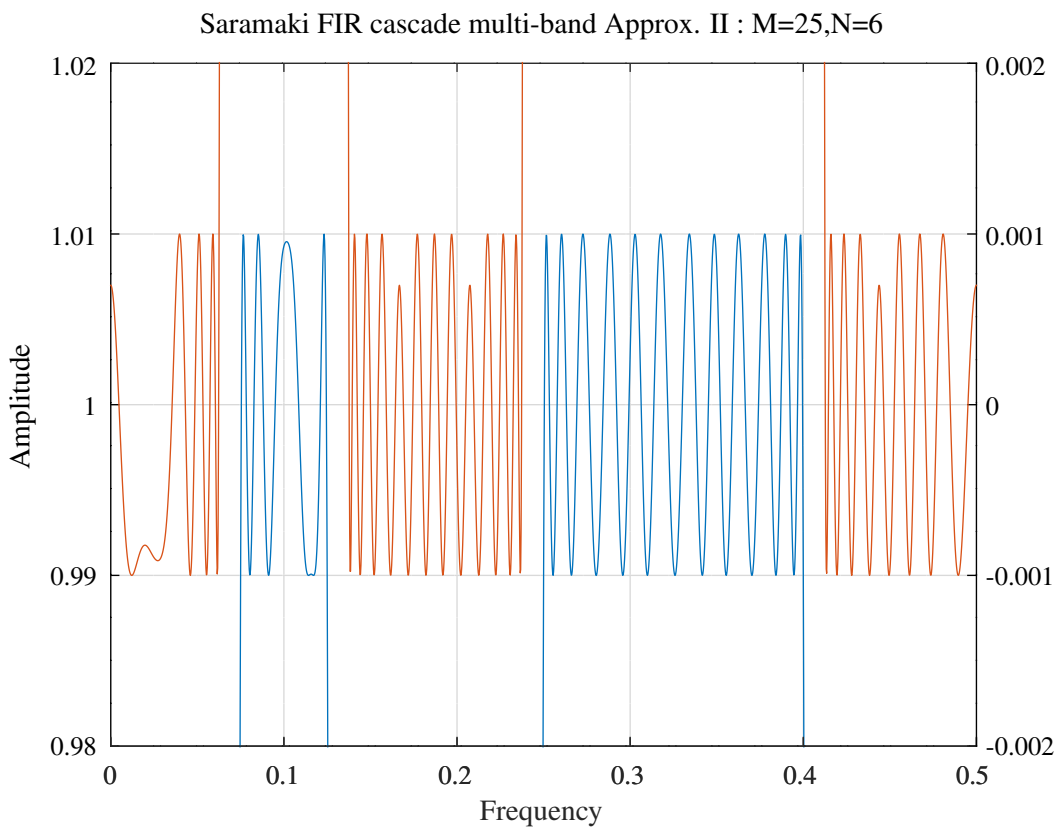



Figure N.100: Detailed view of the pass-band and stop-band response of the Approximation Problem II multi-band filter example.

# Appendix O

# Application of the *Kalman-Yakubovič-Popov* lemma to digital filter design

The *Kalman-Yakubovič-Popov* (KYP) lemma [170] is a fundamental result from the theory of control systems. The continuous time KYP states that, for a state-variable system with $A \in \mathbb{C}^{n\times n}$ and $(A, B)$ controllable, the frequency domain inequality(FDI):

$$\sigma\left(H\left(\omega\right), \Pi\right) = \begin{bmatrix} \left(\imath\omega I - A\right)^{-1} B \\ I \end{bmatrix}^{\dagger} \Theta \begin{bmatrix} \left(\imath\omega I - A\right)^{-1} B \\ I \end{bmatrix} \leq 0 \quad \omega \in \mathbb{R} \cup \{\infty\} \tag{O.1}$$

(where $M^{\dagger}$ means complex conjugate transpose of $M$) holds if and only if the following linear matrix inequality(LMI) is feasible for a solution $P \in \mathbb{H}^n$, the set of Hermitian $n \times n$ matrixes:

$$\begin{bmatrix} A^{\dagger}P + PA & PB \\ B^{\dagger}P & 0 \end{bmatrix} + \Theta \preceq 0 \tag{O.2}$$

For the purposes of filter design, the lemma states that an infinite dimension frequency response constraint is equivalent to a constraint on the coefficients of the state variable description. If $H\left(\omega\right) = C\left(\imath\omega I - A\right)^{-1} B + D$ is the transfer function of the filter with $A \in \mathbb{R}^{n\times n}$, then we can write:

$$\Theta = \begin{bmatrix} C & D \\ 0 & I \end{bmatrix}^{\top} \Pi \begin{bmatrix} C & D \\ 0 & I \end{bmatrix}$$

For example, if $\Pi = \begin{bmatrix} I & 0 \\ 0 & -\varepsilon^2 I \end{bmatrix}$ then the corresponding frequency domain constraint is $\left|H\left(\omega\right)\right|^2 \leq \varepsilon^2$. If $H\left(\omega\right)$ is single-input-single-output, the desired frequency response is $H_D\left(\omega\right)$ and if

$$\Pi = \begin{bmatrix} 1 & -H_D\left(\omega\right) \\ -H_D^{\dagger}\left(\omega\right) & \left|H_D\left(\omega\right)\right|^2 - \varepsilon^2 \end{bmatrix}$$

then the corresponding frequency domain constraint is $\left|H\left(\omega\right) - H_D\left(\omega\right)\right|^2 \leq \varepsilon^2$. Similarly, if

$$\Pi = \begin{bmatrix} 1 & 0 \\ 0 & -\left|H_D\left(\omega\right)\right|^2 - \varepsilon^2 \end{bmatrix}$$

then $\left|H\left(\omega\right)\right|^2 - \left|H_D\left(\omega\right)\right|^2 \leq \varepsilon^2$.

The KYP lemma has been generalised to discrete time state variable systems and to unions of frequency domain inequalities [50, 199, 198, 200, 51]. If the filter parameter vector is $\rho \in \mathbb{R}^p$, then the LMI of Equation O.2 corresponds to an SDP optimisation problem with $p + \frac{n(n+1)}{2}$ variables. *Vandenburghe et al.* [109] show that it is more efficient to solve the *dual* or *primal-dual* SDP problems. I follow *Iwasaki* and *Hara* [199].

## O.1 *Iwasaki* and *Hara*'s generalised KYP lemma

### O.1.1 The generalised S-procedure

The so-called *S-procedure*[a] forms the basis of proofs of the KYP lemma. It "converts a constrained inequality to an unconstrained inequality with multipliers". The classical version of the S-procedure asserts that, given matrixes $\Theta, M \in \mathbb{H}^n$, the following

[a] Also see Section B.7.3

statements are equivalent:

$$\xi^* \Theta \xi \preceq 0 \quad \forall \xi \in \mathbb{C}^n \text{ such that } \xi^* M \xi \succeq 0 \tag{O.3a}$$

$$\exists \tau \in \mathbb{R} \quad \text{such that } \tau \geq 0 \text{ and } \Theta + \tau M \preceq 0 \tag{O.3b}$$

where $\xi^*$ is the *Moore-Penrose* inverse. The S-procedure replaces the non-linear condition of Equation O.3a with the linear condition of Equation O.3b. The latter condition is easier to satisfy as it is a search for the scalar multiplier, $\tau$, satisfying convex constraints.

*Iwasaki* and *Hara* [199, Section II.A] write an alternative definition of the S-procedure:

$$\begin{aligned}
\boldsymbol{M} &:= \{\tau M \ : \ \tau \in \mathbb{R}, \ \tau \geq 0\} \\
\boldsymbol{N} &:= \{N \ : \ N = N^* \preceq 0\} \\
\mathcal{S} &:= \xi \xi^* \\
\boldsymbol{S}(\boldsymbol{M}) &:= \{\mathcal{S} \in \mathbb{H}^n \ : \ \mathcal{S} \neq 0, \ \mathcal{S} \succeq 0, \ \text{trace}(\boldsymbol{M}\mathcal{S}) \geq 0\} \\
\boldsymbol{S}_1 &:= \{\mathcal{S} \in \boldsymbol{S}(\boldsymbol{M}) \ : \ \text{rank}(\mathcal{S}) = 1\} \\
\text{trace}(\Theta \boldsymbol{S}_1) \leq 0, \ \forall \mathcal{S} \in \boldsymbol{S}_1 &\Leftrightarrow (\Theta + \boldsymbol{M}) \cap \boldsymbol{N} \neq \emptyset \\
\text{trace}(\Theta \boldsymbol{S}_1) < 0, \ \forall \mathcal{S} \in \boldsymbol{S}_1 &\Leftrightarrow (\Theta + \boldsymbol{M}) \cap \text{interior}(\boldsymbol{N}) \neq \emptyset
\end{aligned}$$

The S-procedure is said to be *loss-less* if the equivalence holds in both directions. *Iwasaki* and *Hara* [199, Section II.B] make the following definitions:

*Definition 1*: A set $\boldsymbol{M} \subset \mathbb{H}^n$ is:

- *admissible* if it is a non-empty closed convex cone and interior $(\boldsymbol{N}) \cap \boldsymbol{M} = \emptyset$
- *regular* if $\boldsymbol{N} \cap \boldsymbol{M} = \{0\}$
- *rank-one-separable* if $\boldsymbol{S} = \text{cohull}\, \boldsymbol{S}_1$

and prove the following theorem:

*Theorem 1 (S-procedure)*: Let an admissible and regular set $\boldsymbol{M} \subset \mathbb{H}^n$ be given. The S-procedure is lossless for arbitrary $\Theta \in \mathbb{H}^n$ if-and-only-if $\boldsymbol{M}$ is rank-one-separable.

and lemma:

*Lemma 1*: Let $\boldsymbol{M} \subset \mathbb{H}^n$ be a rank-one-separable set. Then the set $F^* \boldsymbol{M} F + \boldsymbol{P}$ is rank-one-separable for any matrix $F \in \mathbb{C}^{n \times m}$ and subset $\boldsymbol{P} \subset \mathbb{H}^n$ of positive-semidefinite matrixes containing the origin.

### O.1.2 Frequency ranges as curves in the complex plane

The frequency response constraint examples shown above apply to the whole of the positive $\imath\omega$ axis in the continuous-time *s-plane*. For a discrete-time filter, the frequency response constraints are defined over a union of frequencies on the unit-circle in the *z-plane*. *Iwasaki* and *Hara* [199, Section IV] define a curve in the complex plane as:

*Definition 2*: A curve on the complex plane is a collection of infinitely many points $\lambda(t) \in \mathbb{C}$ continuously parameterized by $t$ for $t_o \leq t \leq t_f$ where $t_o, t_f \in \mathbb{R} \cup \{\pm\infty\}$ and $t_o < t_f$.

*Iwasaki* and *Hara* consider the class of curves:

$$\mathbf{\Lambda}\left(\Phi,\Psi\right) := \left\{\lambda \in \mathbb{C} : \sigma\left(\lambda,\Phi\right) = 0, \sigma\left(\lambda,\Psi\right) \geq 0\right\}$$

where $\Phi, \Psi \in \mathbb{H}^2$. If $G \in \mathbb{C}^{n\times m}$ and $\Pi \in \mathbb{H}^{n+m}$ then the function $\sigma : \mathbb{C}^{n\times m} \times \mathbb{H}^{n+m} \mapsto \mathbb{H}^m$ is defined by:

$$\sigma\left(G,\Pi\right) := \begin{bmatrix} G \\ I_m \end{bmatrix}^{\dagger} \Pi \begin{bmatrix} G \\ I_m \end{bmatrix}$$

The set $\mathbf{\Lambda}\left(\Phi,0\right)$ represents a curve if-and-only-if $\det\left(\Phi\right) < 0$. For example,

$$\sigma\left(\lambda, \begin{bmatrix} 1 & -\lambda_c \\ -\lambda_c^{\dagger} & \left|\lambda_c\right|^2 - r^2 \end{bmatrix}\right) = \left|\lambda - \lambda_c\right|^2 - r^2 = 0$$

defines a circle with radius $r$ centred at $\lambda_c$ and

$$\sigma\left(\lambda, \begin{bmatrix} 0 & a+\imath b \\ a-\imath b & 2c \end{bmatrix}\right) = 2\left(a\Re\lambda + b\Im\lambda + c\right) = 0$$

defines a straight line with the normal vector $a+\imath b \neq 0$. The set $\mathbf{\Lambda}\left(0,\Psi\right)$ represents a region in the complex plane with boundary $\mathbf{\Lambda}\left(\Psi,0\right)$ whenever $\det\left(\Psi\right) < 0$.

*Iwasaki* and *Hara* prove the following lemma [199, Lemma 2, Appendix 1]:

*Lemma 2*: Let $\Phi$,$\Psi \in \mathbb{H}^2$ be given. Suppose $\det\left(\Psi\right) < 0$. Then there exists a transformation, $T$, such that:

$$\Phi = T^{\dagger}\Phi_o T$$
$$\Phi_o = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$\Psi = T^{\dagger}\Psi_o T$$
$$\Psi_o = \begin{bmatrix} \alpha & \beta \\ \beta & \gamma \end{bmatrix}$$

where $\alpha, \beta, \gamma \in \mathbb{R}$ and $T \in \mathbb{C}^{2\times 2}$. In particular, $\alpha$ and $\gamma$ can be ordered so that $0 \leq \alpha \leq \gamma$ or $\alpha < 0 < \gamma$.

*Iwasaki* and *Hara* show that $\sigma\left(s,\Phi_o\right) = 0$ if-and-only-if $s = \imath\omega$ for some $\omega \in \mathbb{R}$. The set $\Lambda\left(\Phi_o,\Psi_o\right)$ is the imaginary axis if $0 \leq \alpha \leq \gamma$ and is a partial segment of the imaginary axis if $\alpha \leq 0 \leq \gamma$. Otherwise it is empty or a single point.

*Iwasaki* and *Hara* state the following proposition [199, Proposition 1]:

*Proposition 1*: Let $\Phi$,$\Psi \in \mathbb{H}^2$ be given and define the set $\mathbf{\Lambda}\left(\Phi,\Psi\right)$ as above. Then the set $\mathbf{\Lambda}\left(\Phi,0\right)$ represents curves on the complex plane if and only if the following two conditions hold:

- $\det\left(\Phi\right) < 0$
- either $0 \leq \alpha \leq \gamma$ or $\alpha < 0 < \gamma$

If $0 \leq \alpha \leq \gamma$, then $\mathbf{\Lambda}\left(\Phi,\Psi\right)$ is a circle or a line specified by $\mathbf{\Lambda}\left(\Phi,0\right)$ since $\mathbf{\Lambda}\left(\Phi,0\right) \subseteq \mathbf{\Lambda}\left(0,\Psi\right)$. Alternatively, if $\alpha < 0 < \gamma$, then $\mathbf{\Lambda}\left(\Phi,\Psi\right)$ is a partial segment (or segments) of a circle or line specified by $\mathbf{\Lambda}\left(\Phi,0\right) \cap \mathbf{\Lambda}\left(0,\Psi\right)$.

By an appropriate choice of $\Phi$ and $\Psi$ the set $\mathbf{\Lambda}$ can be specialised to define a range of the frquency variable $\lambda$. For the discrete-time frequency on the unit circle in the complex plane:

$$\mathbf{\Lambda} = \left\{\lambda = e^{\imath\omega} : \omega \in \mathbf{\Theta}\right\} \tag{O.4}$$
$$\sigma\left(\lambda,\Phi\right) = 0 \tag{O.5}$$
$$\Phi = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \tag{O.6}$$

where $\mathbf{\Theta}$ is a subset of the real numbers specified by $\Psi$ as shown in Table O.1.

*Iwasaki* and *Hara* [198, Section 3.2] justify the low frequency case by noting that $z + z^{\dagger} \geq 2\cos\omega_l$ and choose $\Psi$ accordingly. Similarly, for the band pass case, the frequency interval condition can be written as $\left|\omega - \omega_c\right| \leq \omega_w$ or $\cos\left(\omega - \omega_c\right) \geq \cos\omega_w$ where $0 \leq \omega_w \leq \pi$. This condition can be written as $\sigma\left(e^{\imath\omega},\Psi\right) \geq 0$, again with $\Psi$ as shown in Table O.1.

| | LF | MF | HF |
|---|---|---|---|
| $\boldsymbol{\Theta}$ | $\lvert\omega\rvert \leq \omega_l$ | $\omega_1 \leq \omega \leq \omega_2$ | $\omega_h \leq \omega \leq \pi$ |
| $\Psi$ | $\begin{bmatrix} 0 & 1 \\ 1 & -2\cos\omega_l \end{bmatrix}$ | $\begin{bmatrix} 0 & e^{\imath\omega_c} \\ e^{-\imath\omega_c} & -2\cos\omega_w \end{bmatrix}$ | $\begin{bmatrix} 0 & -1 \\ -1 & 2\cos\omega_h \end{bmatrix}$ |

Table O.1: $\Psi$ matrixes for distrete-time frequency ranges $\boldsymbol{\Theta}$ given by *Iwasaki* and *Hara* [199, p.47]. $\omega_c = (\omega_1 + \omega_2)/2$ and $\omega_w = (\omega_1 - \omega_2)/2$.

### O.1.3 The generalised KYP lemma

*Iwasaki* and *Hara* [199, Section V.B, p.49] prove the following *generalised* KYP lemma for state variable systems:

*Theorem 4(State variable, nonstrict inequality version):* Let matrixes $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$, $\Theta \in \mathbb{H}^{n+m}$ and $\Phi,\Psi \in \mathbb{H}^2$ be given and the pair $(A, B)$ is controllable. $\boldsymbol{\Lambda}$ is the set of curves in the complex plane. $\boldsymbol{\Omega}$ is the set of eigenvalues of $A$ in $\boldsymbol{\Lambda}$. The following statements are equivalent:

1. For each $\lambda \in \boldsymbol{\Lambda} \setminus \boldsymbol{\Omega}$:

$$\begin{bmatrix} (\lambda I - A)^{-1} B \\ I \end{bmatrix}^{\dagger} \Theta \begin{bmatrix} (\lambda I - A)^{-1} B \\ I \end{bmatrix} \leq 0 \tag{O.7}$$

2. There exist $P, Q \in \mathbb{H}^n$ such that $Q \succeq 0$ and:

$$\begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^{\dagger} (\Phi \otimes P + \Psi \otimes Q) \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \Theta \preceq 0 \tag{O.8}$$

*Iwasaki* and *Hara* [199, Section VII, p.52] add a lemma:

*Lemma 8:* For a transfer function $H(\lambda) = C(\lambda I - A)^{-1} B + D$, with $m$ inputs and $p$ outputs, the condition $\sigma(H(\lambda), \Pi) \leq 0 \; \forall \lambda \in \boldsymbol{\Lambda}$ holds if and only if there exist Hermitian matrices $P$ and $Q \succeq 0$ such that

$$\begin{bmatrix} \Gamma(P, Q, A, B, C, D) & [C \; D]^{\dagger} \Pi_{11} \\ \Pi_{11} [C \; D] & -\Pi_{11} \end{bmatrix} \preceq 0$$

holds, where the matrix $\Pi \in \mathbb{H}^{m+p}$ is such that

$$\Pi = \begin{bmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{12}^{\dagger} & \Pi_{22} \end{bmatrix}, \Pi_{11} \in \mathbb{H}^p \text{ and } \Pi_{11} \succeq 0$$

and

$$\Gamma(P, Q, A, B, C, D) = \begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^{\dagger} (\Phi \otimes P + \Psi \otimes Q) \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \begin{bmatrix} 0 & C^{\dagger}\Pi_{12} \\ \Pi_{12}^{\dagger} C & D^{\dagger}\Pi_{12} + \Pi_{12}^{\dagger} D + \Pi_{22} \end{bmatrix}$$

This lemma follows from the theorem, the expansion of $\Theta$ and from the properties of the *Schur complement*[b]. The expansion of $\Theta$ is bi-linear in the FIR filter coefficients. Application of the Schur complement linearises this expansion:

$$\begin{aligned} \Theta = \begin{bmatrix} C & D \\ 0 & I \end{bmatrix}^{\dagger} \Pi \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} &= \begin{bmatrix} C^{\dagger} & 0 \\ D^{\dagger} & I \end{bmatrix} \begin{bmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{12}^{\dagger} & \Pi_{22} \end{bmatrix} \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} \\ &= \begin{bmatrix} C^{\dagger}\Pi_{11} & C^{\dagger}\Pi_{12} \\ D^{\dagger}\Pi_{11} + \Pi_{12}^{\dagger} & D^{\dagger}\Pi_{12} + \Pi_{22} \end{bmatrix} \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} \\ &= \begin{bmatrix} C^{\dagger}\Pi_{11}C & C^{\dagger}\Pi_{12} + C^{\dagger}\Pi_{11}D \\ \Pi_{12}^{\dagger}C + D^{\dagger}P_{11}C & D^{\dagger}\Pi_{11}D + D^{\dagger}\Pi_{12} + \Pi_{12}^{\dagger}D + \Pi_{22} \end{bmatrix} \end{aligned}$$

[b] See Appendix B.6.

$$
\begin{aligned}
&= \begin{bmatrix} 0 & C^\dagger \Pi_{12} \\ \Pi_{12}^\dagger C & D^\dagger \Pi_{12} + \Pi_{12}^\dagger D + \Pi_{22} \end{bmatrix} + \begin{bmatrix} C^\dagger \Pi_{11} C & C^\dagger \Pi_{11} D \\ D^\dagger \Pi_{11} C & D^\dagger \Pi_{11} D \end{bmatrix} \\
&= \begin{bmatrix} 0 & C^\dagger \Pi_{12} \\ \Pi_{12}^\dagger C & D^\dagger \Pi_{12} + \Pi_{12}^\dagger D + \Pi_{22} \end{bmatrix} + \begin{bmatrix} C & D \end{bmatrix}^\dagger \Pi_{11} \begin{bmatrix} C & D \end{bmatrix}
\end{aligned}
$$

The proposition that Equation O.8 (LMI) implies Equation O.7 (FDI) can be justified in the discrete time case as follows [162, A.14]. Firstly note that:

$$
\begin{aligned}
\begin{bmatrix} A & B \\ I & 0 \end{bmatrix} \begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix} &= \begin{bmatrix} A (e^{\imath\omega} I - A)^{-1} B + B \\ (e^{\imath\omega} I - A)^{-1} B \end{bmatrix} \\
&= \begin{bmatrix} A (e^{\imath\omega} I - A)^{-1} B + (e^{\imath\omega} I - A)(e^{\imath\omega} I - A)^{-1} B \\ (e^{\imath\omega} I - A)^{-1} B \end{bmatrix} \\
&= \begin{bmatrix} e^{\imath\omega} I \\ I \end{bmatrix} (e^{\imath\omega} I - A)^{-1} B
\end{aligned}
$$

For the low pass case, $|\omega| \leq \omega_l \leq \pi$:

$$
\begin{aligned}
&\begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix}^\dagger \begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^\dagger (\Phi \otimes P + \Psi \otimes Q) \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} \begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix} \\
&= B^\dagger (e^{\imath\omega} I - A)^{-\dagger} \begin{bmatrix} e^{\imath\omega} I \\ I \end{bmatrix}^\dagger \begin{bmatrix} -P & Q \\ Q & P - 2Q\cos\omega_l \end{bmatrix} \begin{bmatrix} e^{\imath\omega} I \\ I \end{bmatrix} (e^{\imath\omega} I - A) B \\
&= 2\left[\cos\omega - \cos\omega_l\right] B^\dagger (e^{\imath\omega} I - A)^{-\dagger} Q (e^{\imath\omega} I - A)^{-1} B \\
&\geq 0
\end{aligned}
$$

For the band pass case, $\omega_1 \leq |\omega| \leq \omega_2 \leq \pi$:

$$
\begin{aligned}
&B^\dagger (e^{\imath\omega} I - A)^{-\dagger} \begin{bmatrix} e^{\imath\omega} I \\ I \end{bmatrix}^\dagger \begin{bmatrix} -P & e^{\imath\frac{\omega_2+\omega_1}{2}} Q \\ e^{-\imath\frac{\omega_2+\omega_1}{2}} Q & P - 2Q\cos\frac{\omega_2-\omega_1}{2} \end{bmatrix} \begin{bmatrix} e^{\imath\omega} I \\ I \end{bmatrix} (e^{\imath\omega} I - A) B \\
&= 2\left[\cos\left(\omega - \frac{\omega_2+\omega_1}{2}\right) - \cos\left(\frac{\omega_2-\omega_1}{2}\right)\right] B^\dagger (e^{\imath\omega} I - A)^{-\dagger} Q (e^{\imath\omega} I - A)^{-1} B \\
&\geq 0
\end{aligned}
$$

For the high pass case, $\omega_h \leq |\omega| \leq \pi$:

$$
\begin{aligned}
&B^\dagger (e^{\imath\omega} I - A)^{-\dagger} \begin{bmatrix} e^{\imath\omega} I \\ I \end{bmatrix}^\dagger \begin{bmatrix} -P & -Q \\ -Q & P + 2Q\cos\omega_h \end{bmatrix} \begin{bmatrix} e^{\imath\omega} I \\ I \end{bmatrix} (e^{\imath\omega} I - A) B \\
&= 2\left(\cos\omega_h - \cos\omega\right) B^\dagger (e^{\imath\omega} I - A)^{-\dagger} Q (e^{\imath\omega} I - A)^{-1} B \\
&\geq 0
\end{aligned}
$$

If $Q \succeq 0$ and the LMI, Equation O.8, holds, then in each of the three cases:

$$
\begin{aligned}
&\begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix}^\dagger \Theta \begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix} \\
&= \begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix}^\dagger \left( \begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^\dagger (\Phi \otimes P + \Psi \otimes Q) \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \Theta \right) \begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix} \cdots \\
&\quad - \begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix}^\dagger \begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^\dagger (\Phi \otimes P + \Psi \otimes Q) \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} \begin{bmatrix} (e^{\imath\omega} I - A)^{-1} B \\ I \end{bmatrix} \\
&\leq 0
\end{aligned}
$$

### O.1.4 Examples of FIR filter design with the generalised KYP lemma

**Amplitude constraints** The pass-band and stop-band errors are:

$$
|H(e^{\imath\omega}) - e^{-\imath d\omega}| < \varepsilon_p \quad |\omega| \leq \omega_p
$$

$$|H(e^{\imath\omega})| < \varepsilon_s \quad |\omega| \geq \omega_s$$

where $d$ is the desired pass band delay in samples and $\omega_p$ and $\omega_s$ are the pass band and stop band angular frequencies respectively. An additional specification prevents over-shoot in the transition band:

$$|H(e^{\imath\omega})| < 1 + \varepsilon_t \quad \omega_p \leq \omega \leq \omega_s$$

Alternatively apply an overall maximum amplitude constraint:

$$|H(e^{\imath\omega})| < 1 + \varepsilon_{max} \quad -\pi < \omega < \pi$$

**Frequency response** The frequency response of an order $n$ single-input-single-output FIR filter is:

$$H(\omega) = \sum_{k=0}^{n} h_k e^{-\imath k\omega}$$

**State variable description** For direct-form FIR filter implementations, the $A$ and $B$ matrixes are constant representations of the successive delays and the $C$ and $D$ matrixes represent the impulse response response coefficients. The state variable description is:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \mathbf{0}_{(n-1)\times 1} & \boldsymbol{I}_{n-1} \\ 0 & \mathbf{0}_{1\times(n-1)} \end{bmatrix} & \begin{bmatrix} \mathbf{0}_{(n-1)\times 1} \\ 1 \end{bmatrix} \\ \begin{bmatrix} h_n & \dots & h_d - \Delta & \dots & h_1 \end{bmatrix} & [h_0] \end{bmatrix}$$

where, if the desired pass band delay is $d$ samples, $\Delta = 1$ in the pass-band and $\Delta = 0$ otherwise.

**Squared error calculation** The Octave function *directFIRnonsymmetricEsqPW.m* calculates the piece-wise weighted mean-squared-error of the response. Over each piece, $p$, for frequency range $[\omega_{pl}, \omega_{pu}]$:

$$\begin{aligned}
\mathcal{E}_p^2 &= \frac{1}{\pi}\int_{\omega_{pl}}^{\omega_{pu}} |H(\omega) - R_p e^{-\imath d_p \omega}|^2 d\omega \\
&= \frac{1}{\pi}\int_{\omega_{pl}}^{\omega_{pu}} \left[\sum_{k=0}^{n} h_k e^{-\imath k\omega} - R_p e^{-\imath d_p\omega}\right]\left[\sum_{k=0}^{n} h_k e^{\imath k\omega} - R_p e^{\imath d_p\omega}\right] d\omega \\
&= \frac{1}{\pi}\int_{\omega_{pl}}^{\omega_{pu}} \left[\sum_{k=0}^{n}\sum_{l=0}^{n} h_k h_l \cos(k-l)\omega - 2R_p\sum_{k=0}^{n} h_k \cos(k-d_p)\omega + R_p^2\right] d\omega \\
&= \frac{1}{\pi}\left[\omega\sum_{k=0}^{n} h_k^2 + \sum_{k=0}^{n}\sum_{l=0,l\neq k}^{n} h_k h_l \frac{\sin(k-l)\omega}{k-l} - 2R_p h_{d_p}\omega - 2R_p\sum_{k=0,k\neq d_p}^{n} h_k \frac{\sin(k-d_p)\omega}{k-d_p} + R_p^2\omega\right]_{\omega_{pl}}^{\omega_{pu}}
\end{aligned}$$

where, in each piece, $R_p$ is the desired amplitude and $d_p$ is the nominal delay in samples. The weighted sum of the piece-wise mean-squared-errors, $\mathcal{E}^2$, is a quadratic form:

$$\mathcal{E}^2 = \boldsymbol{h}^\top \boldsymbol{S}\boldsymbol{h} + 2\boldsymbol{h}^\top \boldsymbol{s} + c$$

where $\boldsymbol{h}$ is the filter impulse response and $\boldsymbol{S}$ is real, symmetric and positive definite. $\boldsymbol{S}$ has a *Cholesky decomposition* into lower and upper triangular matrixes, $\boldsymbol{S} = \boldsymbol{L}\boldsymbol{U}$, where $\boldsymbol{L}^\top = \boldsymbol{U}$. The SOCP minimisation problem is [2, Section 14.7.3(i)]:

$$\begin{aligned}
&\textbf{minimise} && \delta \\
&\textbf{subject to} && \|\boldsymbol{U}\boldsymbol{h} + \boldsymbol{L}^{-1}\boldsymbol{s}\| \leq \delta
\end{aligned}$$

**SDP constraints** Assuming $\Psi_{11} = 0$ and $\Psi_{21}^\dagger = \Psi_{12}$, for each constraint:

$$\begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^\dagger \begin{bmatrix} -P & \Psi_{12}Q \\ \Psi_{12}^\dagger Q & P + \Psi_{22}Q \end{bmatrix} \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \begin{bmatrix} C & D \end{bmatrix}^\dagger \Pi \begin{bmatrix} C & D \end{bmatrix} \preceq 0$$

where $P$ and $Q$ are Hermitian and $Q \succeq 0$. *Pipeleers* and *Vandeberghe* [50] show that if the $A$, $B$ and $\Theta$ matrixes are real valued then the $P$ and $Q$ matrixes can be constrained to be real and symmetric.

### Design of a nonsymmetric low pass FIR filter with given pass band delay

As an example, *Iwasaki* and *Hara* [199, Section VII.B.2, pp. 53-55] describe the design of a low-pass FIR filter with order $n = 30$ and a nominal pass-band group delay of $d = 10$ samples, $\varepsilon_s = 0.01$, $\omega_p = 0.3\pi$ and $\omega_s = 0.4\pi$. They optimised the pass band sqared-error, $|H(w) - e^{-j\omega d}|^2 < \varepsilon_p^2$ and found $\varepsilon_p = 0.0569$. *Iwasaki* and *Hara* compare the performance of symmetric FIR filters with order 20 and 10 samples delay and order 30 and 15 samples delay. The Octave script *directFIRnonsymmetric_kyp_lowpass_test.m* unsuccessfully attempts to reproduce this filter design example with *YALMIP* [89, 82] and SeDuMi[c]. This script optimises the maximum squared-error of the pass band minus nominal delay response and applies a constraint, $\varepsilon_{max}^2$, to the maximum overall squared-amplitude response. I found that I could not obtain the combination of band edges and amplitude responses given in the example. In addition, I found that when I use the $\varepsilon_p^2$ constraint as the YALMIP objective function, SeDuMi gives the message "numerical problems", meaning that the result is only accurate to `pars.bigeps`. The filter specification is:

```
N=30 % FIR filter order
d=10 % Nominal FIR filter delay
fap=0.15 % Amplitude pass band edge
fas=0.2 % Amplitude stop band edge
Esq_s=0.0001 % Squared amplitude stop band error
Esq_max=1.1 % Maximum overall squared amplitude
```

The resulting FIR impulse response is:

```
h = [ -0.0023662932,  0.0146336075,  0.0318179853,  0.0347157199, ...
       0.0074935791, -0.0382567401, -0.0587372915, -0.0073013038, ...
       0.1200768355,  0.2648504641,  0.3384933976,  0.2884460008, ...
       0.1427249947, -0.0083692582, -0.0808540027, -0.0579331536, ...
       0.0076237937,  0.0468265244,  0.0322427339, -0.0095033788, ...
      -0.0337559833, -0.0219250275,  0.0073005502,  0.0232055585, ...
       0.0140244705, -0.0063448394, -0.0169278240, -0.0105233595, ...
       0.0031907526,  0.0107458268,  0.0111728102 ];
```

Figure O.1 shows the amplitude response. Figure O.2 shows the pass band amplitude, phase error and delay responses. The pass band phase error shown is adjusted for the nominal delay. The overall maximum amplitude response is $\varepsilon_{max} = 1.0468$ . The maximum error in the pass band compared with a pure delay of $d$ samples is $\varepsilon_p = 0.0733$ . The maximum stop band amplitude is $\varepsilon_s = 0.0105$ .

### Design of a non-symmetric FIR band pass filter with given pass band delay

The Octave script *directFIRnonsymmetric_kyp_bandpass_test.m* uses the *Iwasaki* and *Hara*'s generalised KYP lemma to design a band pass FIR filter with the following specification:

```
N=30 % FIR filter order
d=10 % Nominal FIR filter delay
fasl=0.1 % Amplitude stop band lower edge
fapl=0.175 % Amplitude pass band lower edge
fapu=0.225 % Amplitude pass band upper edge
fasu=0.3 % Amplitude stop band upper edge
Esq_z=5e-05 % Squared amplitude pass band - delay error
Esq_s=0.0001 % Squared amplitude stop band error
```

This example does not minimise the pass band squared-error, $\varepsilon_p^2$. Instead it finds a solution that satisfies the SDP constraints. This example does not constrain the maximum amplitude response in the transition band. Instead it applies a constraint, $\varepsilon_{max}^2$, to the maximum overall squared-amplitude response. The resulting FIR impulse response is:

```
h = [ -0.0103310242, -0.0024158430,  0.0030906630, -0.0070639234, ...
       0.0099294830,  0.0631107470,  0.0323281562, -0.1184998735, ...
      -0.1552816116,  0.0706216712,  0.2575416442,  0.0832838643, ...
      -0.2116742994, -0.1907367590,  0.0609807654,  0.1436885397, ...
       0.0264321213, -0.0291918402,  0.0072521867, -0.0124064259, ...
      -0.0578311706, -0.0187250333,  0.0442574756,  0.0320729627, ...
      -0.0080018109, -0.0077761515,  0.0013592394, -0.0101907755, ...
      -0.0122061971,  0.0037558161,  0.0106945580 ];
```

[c]The Octave script *directFIRnonsymmetric_socp_slb_lowpass_test.m*, described in Section N.5.2, designs a similar filter using SOCP PCLS optimisation.

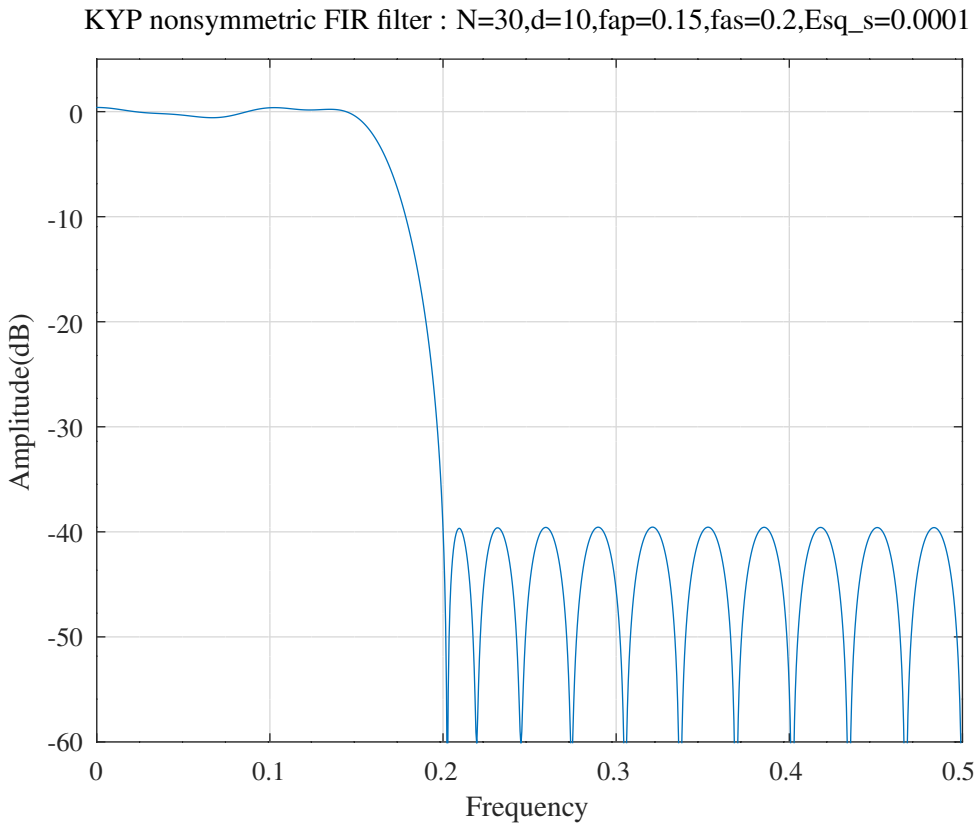



Figure O.1: Amplitude response of a non-symmetric FIR low pass filter designed with the KYP lemma.

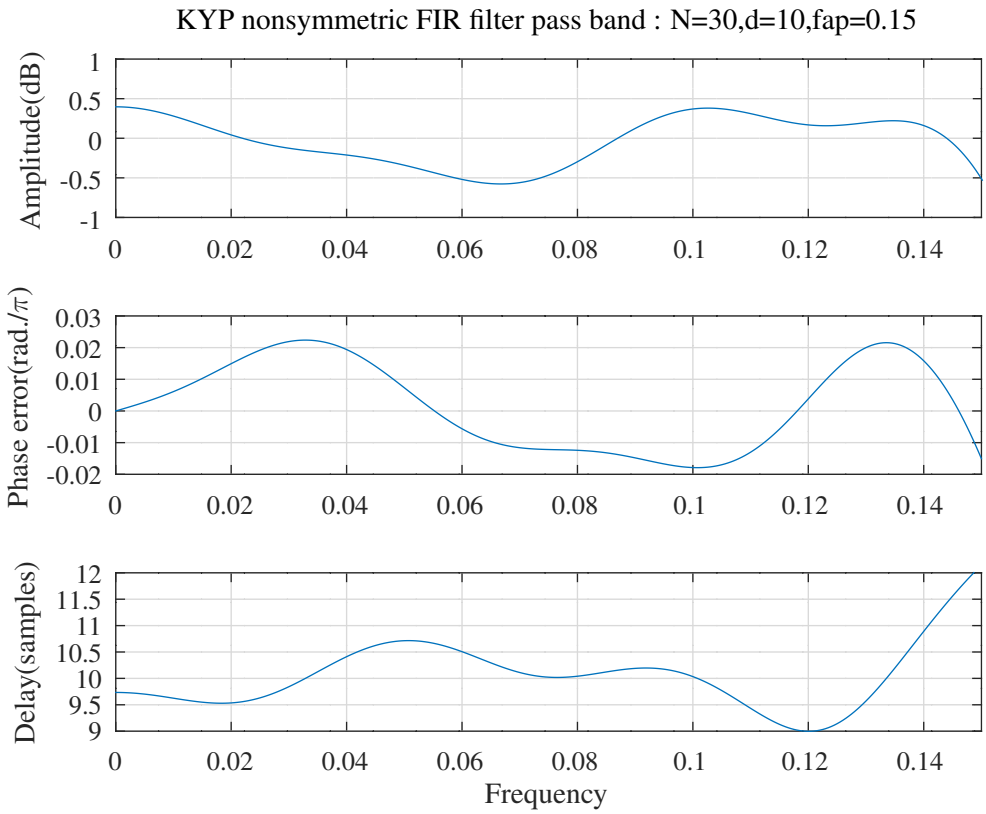



Figure O.2: Pass band amplitude, phase error and delay responses of a non-symmetric FIR low pass filter designed with the KYP lemma. The pass band phase error shown is adjusted for the nominal delay.

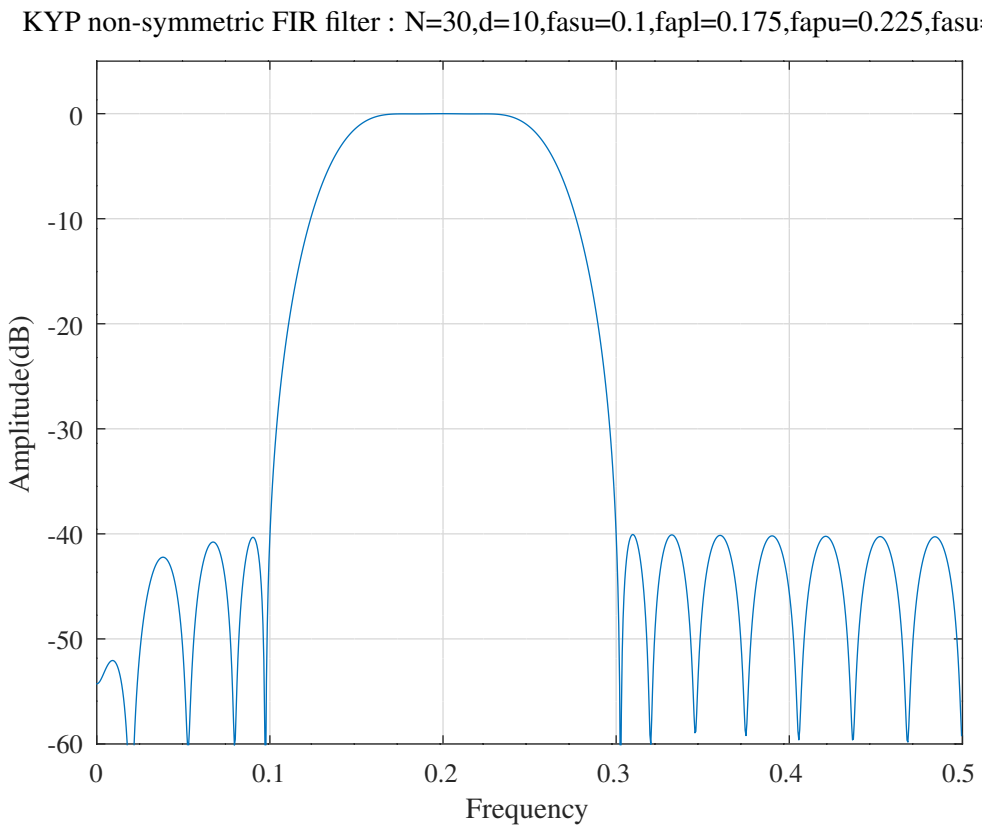



Figure O.3: Amplitude response of a non-symmetric FIR band pass filter designed with the KYP lemma.

As shown in Section N.1, the complementary FIR lattice reflection coefficients are:

```
k = [  0.99984301,  0.99993221,  0.99951441,  0.99901998, ...
       0.99993825,  0.99897433,  0.99995307,  0.99998552, ...
       0.99901529,  0.99976727,  0.98966443,  0.99765228, ...
       0.99070625,  0.99162048,  0.99978176,  0.99979392, ...
       0.99801406,  0.96857970,  0.91799740,  0.98572567, ...
       0.80945640,  0.98758466,  0.92684266,  0.97094863, ...
       0.99754032,  0.99612265,  0.99996901,  0.99995984, ...
       0.99986687,  0.99995731, -0.01711678 ]';


kc = [ -0.01771890, -0.01164384,  0.03116005,  0.04426146, ...
       -0.01111259, -0.04528017, -0.00968760, -0.00538150, ...
       -0.04436732,  0.02157315,  0.14340267,  0.06848312, ...
       -0.13601882, -0.12918525,  0.02089078, -0.02030085, ...
       -0.06299149,  0.24870338,  0.39658640, -0.16835946, ...
       -0.58717998, -0.15708765,  0.37544998,  0.23928802, ...
       -0.07009496, -0.08797532, -0.00787297, -0.00896246, ...
       -0.01631714,  0.00924035,  0.99985350 ]';
```

Figure O.3 shows the amplitude response. Figure O.4 shows the pass band amplitude, phase error and delay responses. The pass band phase error shown is adjusted for the nominal delay.

**Design of a non-symmetric FIR band pass Hilbert filter with given pass band delay**

The Octave script *directFIRnonsymmetric_kyp_bandpass_hilbert_test.m* uses the *Iwasaki* and *Hara*'s generalised KYP lemma to design a band pass FIR Hilbert filter with the following specification:

```
N=40 % FIR filter order
d=10 % Nominal FIR filter delay
fasl=0.1 % Amplitude stop band lower edge
```

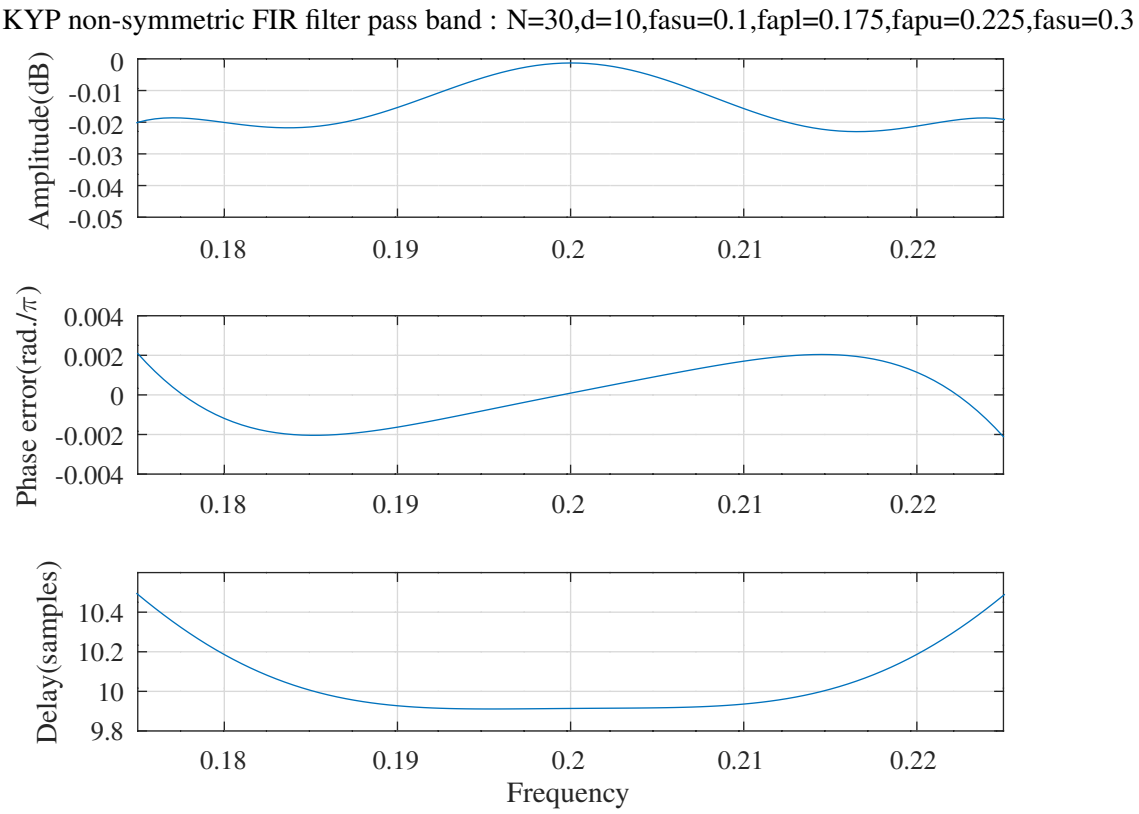



Figure O.4: Pass band amplitude, phase error and delay responses of a non-symmetric FIR band pass filter designed with the KYP lemma. The pass band phase error shown is adjusted for the nominal delay.

```
fapl=0.175 % Amplitude pass band lower edge
fapu=0.225 % Amplitude pass band upper edge
fasu=0.3 % Amplitude stop band upper edge
Esq_z=2.3e-05 % Squared amplitude pass band - delay error
Esq_s=0.0001 % Squared amplitude stop band error
```

The pass band desired response is set to $-\imath e^{-\imath\omega d}$. The resulting FIR impulse response is:

```
h = [  0.0016451782, -0.0122146548, -0.0036268747, -0.0030544970, ...
      -0.0242204356, -0.0002386386,  0.0910798809,  0.0838214135, ...
      -0.1117444209, -0.2221385485,  0.0001179247,  0.2674641644, ...
       0.1622607188, -0.1479548630, -0.1974239705,  0.0002650843, ...
       0.0849382368,  0.0172966894,  0.0126233584,  0.0543993724, ...
      -0.0003795999, -0.0750673624, -0.0394775891,  0.0280436221, ...
       0.0225545127,  0.0000805980,  0.0149377517,  0.0148584072, ...
      -0.0168726134, -0.0237724984,  0.0001632242,  0.0092719524, ...
       0.0012836263,  0.0019676242,  0.0056971651, -0.0001212541, ...
      -0.0049776164, -0.0018481084,  0.0008297249, -0.0001270555, ...
       0.0001768761 ];
```

Figure O.5 shows the amplitude response. Figure O.6 shows the pass band amplitude, phase and delay responses. The pass band phase shown is adjusted for the nominal delay.

**Design of a nonsymmetric high pass FIR filter with given pass band delay**

This section describes the design of a high-pass FIR filter with order $n = 30$ and a nominal pass-band group delay of $d = 10$ samples, $\varepsilon_s = 0.01$, $\omega_p = 0.3\pi$ and$\omega_s = 0.4\pi$. They optimised the pass band sqared-error, $|H(w) - e^{-j\omega d}|^2 < \varepsilon_p^2$ and found $\varepsilon_p = 0.0569$. *Iwasaki* and *Hara* compare the performance of symmetric FIR filters with order 20 and 10 samples delay and order 30 and 15 samples delay. The Octave script *directFIRnonsymmetric_kyp_highpass_test.m* unsuccessfully attempts to reproduce this filter design example with *YALMIP* [89, 82] and SeDuMi. This script optimises the maximum squared-error of the pass band minus nominal delay response and applies a constraint, $\varepsilon_{max}^2$, to the maximum overall squared-amplitude response. I found that I could not obtain the combination of band edges and amplitude responses given in the example. The filter specification is:

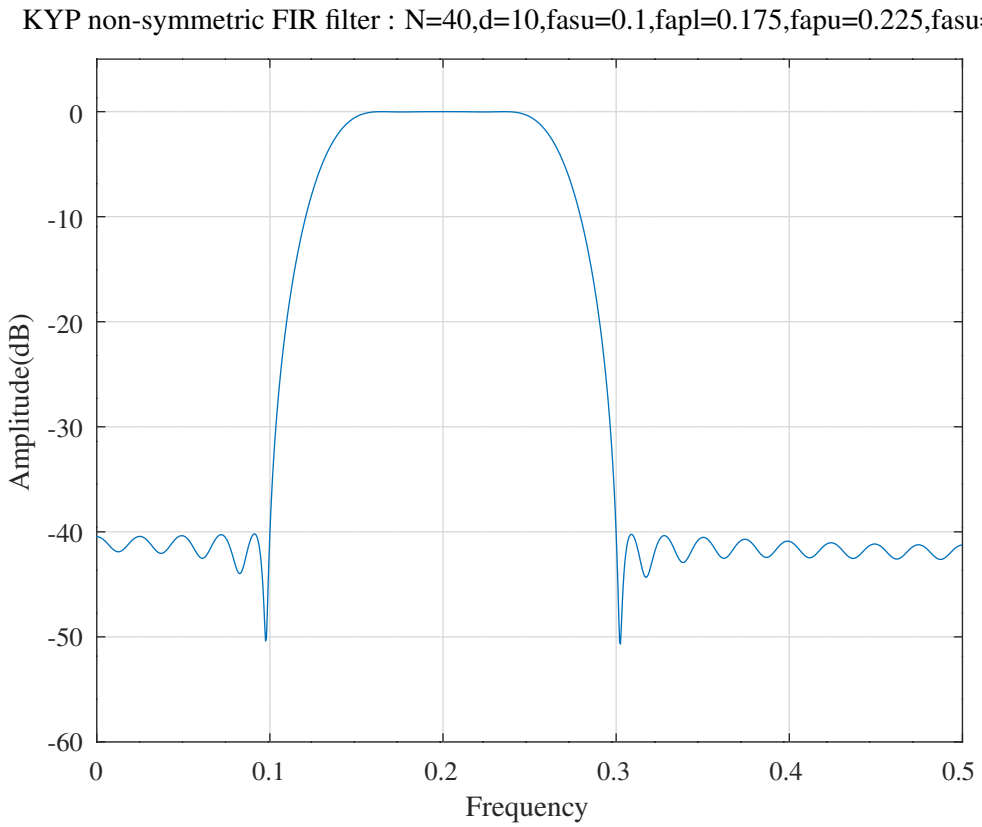



Figure O.5: Amplitude response of a non-symmetric FIR band pass Hilbert filter designed with the KYP lemma.

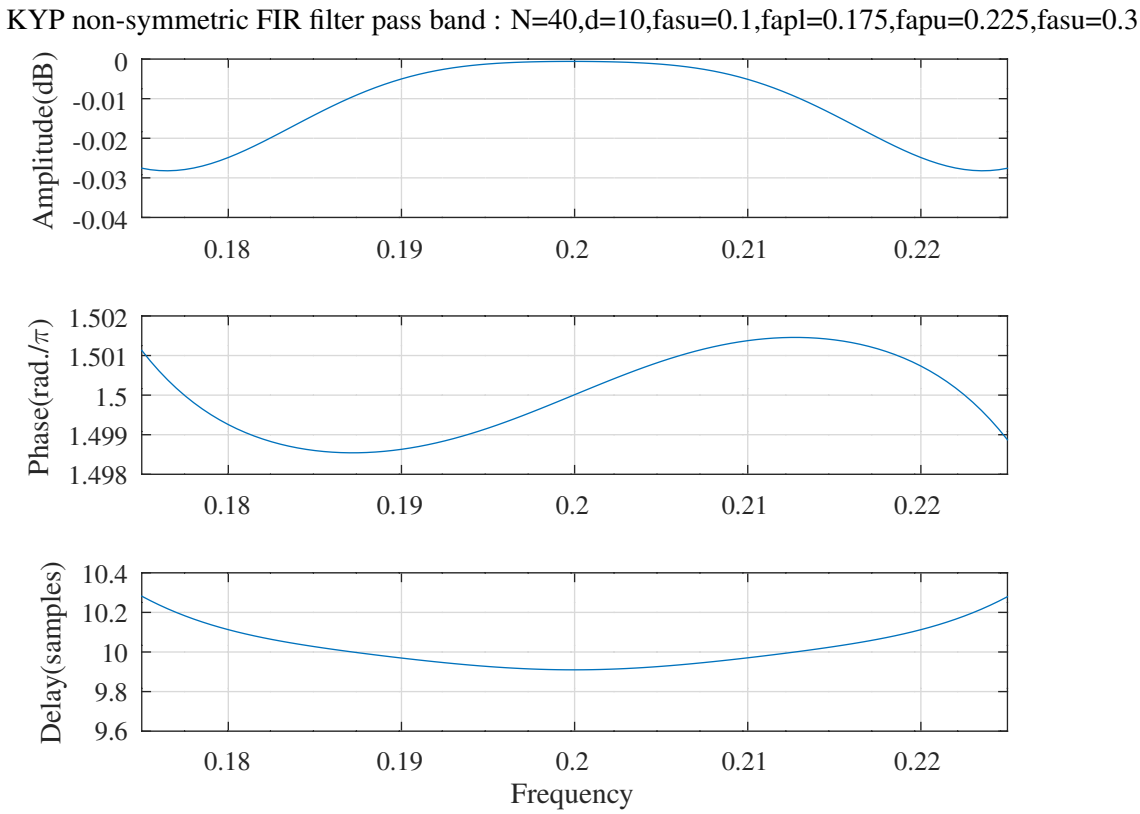



Figure O.6: Pass band amplitude, phase and delay responses of a non-symmetric FIR band pass Hilbert filter designed with the KYP lemma. The pass band phase shown is adjusted for the nominal delay.

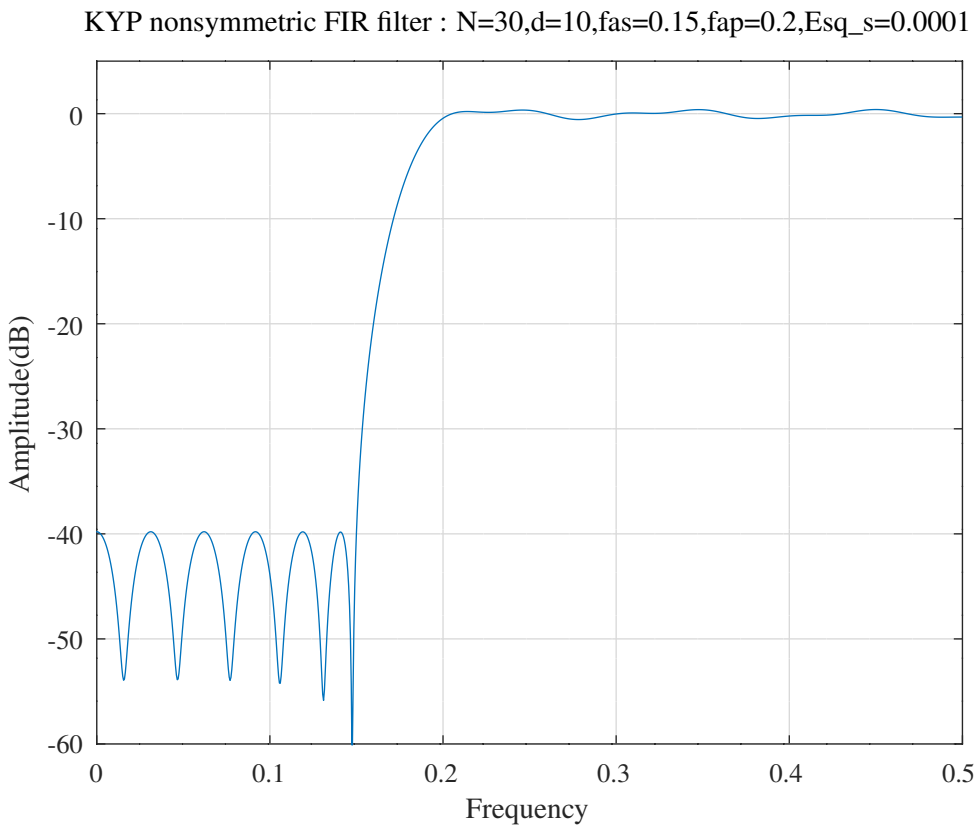



Figure O.7: Amplitude response of a non-symmetric FIR high pass filter designed with the KYP lemma.

```
N=30 % FIR filter order
d=10 % Nominal FIR filter delay
fas=0.15 % Amplitude stop band edge
fap=0.2 % Amplitude pass band edge
Esq_s=0.0001 % Squared amplitude stop band error
Esq_max=1.1 % Maximum overall squared amplitude
```

The resulting FIR impulse response is:

```
h = [ -0.0051682232,  0.0377046325, -0.0150541435, -0.0306548982, ...
      -0.0172516896,  0.0289067848,  0.0624230117,  0.0227810927, ...
      -0.1137943745, -0.2719471525,  0.6378398067, -0.2975000294, ...
      -0.1311057003,  0.0266958884,  0.0872777369,  0.0429369807, ...
      -0.0272122085, -0.0521248189, -0.0154962722,  0.0230177684, ...
       0.0337810155,  0.0059548228, -0.0232351672, -0.0194928103, ...
       0.0013316629,  0.0171827730,  0.0113154511, -0.0032890486, ...
      -0.0149404208, -0.0068107164,  0.0161614545 ];
```

Figure O.7 shows the amplitude response. Figure O.8 shows the pass band amplitude, phase error and delay responses. The pass band phase error shown is adjusted for the nominal delay. The overall maximum amplitude response is $\varepsilon_{max} = 1.0463$ . The maximum error in the pass band compared with a pure delay of $d$ samples is $\varepsilon_p = 0.0655$ . The maximum stop band amplitude is $\varepsilon_s = 0.0102$ .

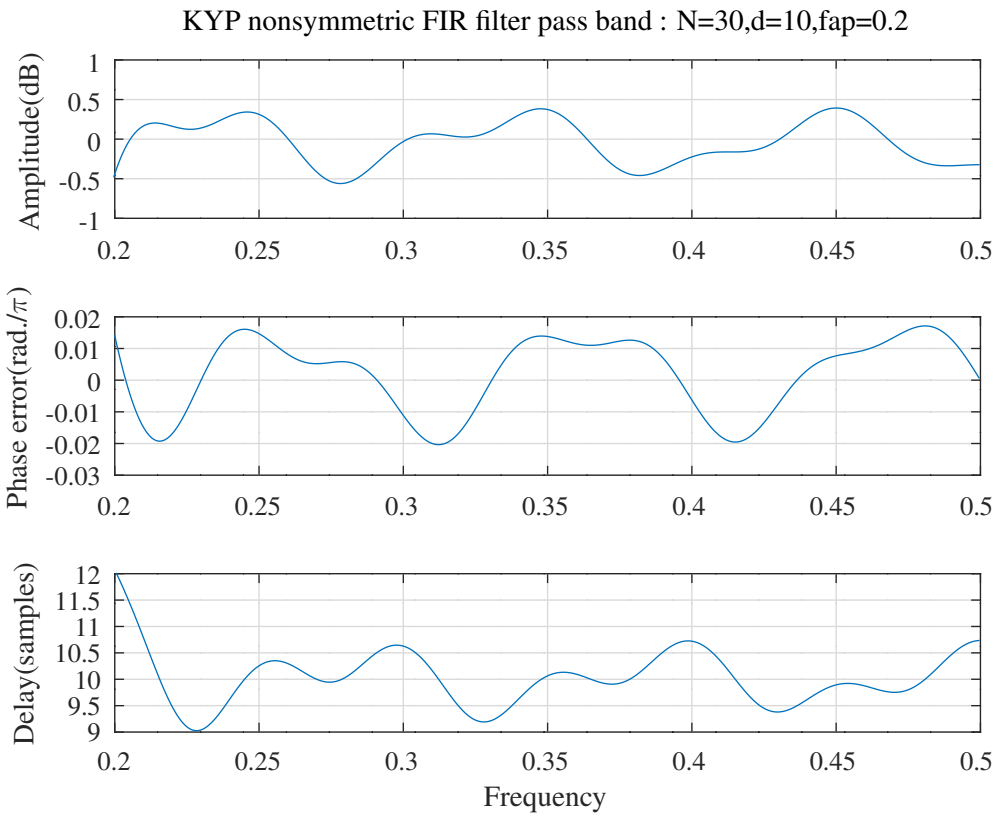



Figure O.8: Pass band amplitude, phase error and delay responses of a non-symmetric FIR high pass filter designed with the KYP lemma. The pass band phase error shown is adjusted for the nominal delay.

## O.2 Generalisation of the KYP lemma to a union of frequency intervals

*Pipeleers*, *Iwasaki* and *Hara* [51] describe the generalisation of the KYP lemma to a union of frequency intervals.

### O.2.1 Union of frequency intervals

*Pipeleers*, *Iwasaki* and *Hara* generalise $\mathbf{\Lambda}(\Phi, \Psi)$, to a union of $l \in \mathbb{I}$ curves with Hermitian matrixes $\Phi, \Psi \in \mathbb{H}^{l+1}$ and a mapping $L_l(\lambda) \ : \ \mathbb{C} \to \mathbb{C}^{l+1}$ defined as:

$$\begin{aligned} L_l(\lambda) &= \left[\lambda^l,\ \lambda^{l-1},\ \cdots,\ \lambda,\ 1\right]^\top,\ l \in \mathbb{I} \\ L_0(\lambda) &= 1 \end{aligned}$$

If $\mathbf{\Lambda}(\Phi, \Psi)$ is unbounded then it is extended with

$$L_l(\infty) = [1\ 0_{1,l}]^\top$$

*Assumption 1*: Similarly to *Lemma 2* above, a similarity transformation $T \in \mathbb{C}^{2\times(l+1)}$ with full row rank of $\Phi_o$ and $\Psi_o$ is assumed to exist. For each $s \in \mathbf{\Lambda}(\Phi_o, \Psi_o)$, the $l$-th degree polynomial in $\lambda$ is:

$$\begin{aligned} [1\ \ -s]\ T\ L_l(\lambda) &= 0, \quad \text{for } s \neq \infty \\ [0\ \ \ 1]\ T\ L_l(\lambda) &= 0, \quad \text{for } s = \infty \end{aligned}$$

has $l$ distinct roots grouped in the set $\in \mathcal{R}_T(s)$.

*Assumption 2*: When $l \geq 2$ we assume that there exists a Hermitian matrix $R \in \mathbb{H}^l$ such that:

$$\begin{aligned} L_{l-1}(\lambda)^\dagger\ R\ L_{l-1}(\lambda) &> 0\ \forall \lambda \in \mathbf{\Lambda}(\Phi_o, \Psi_o) \\ L_{l-1}(\lambda_i)^\dagger\ R\ L_{l-1}(\lambda_j) &= 0\ \forall \lambda_i, \lambda_j \in \mathcal{R}_T(s),\ i \neq j,\ \forall s \in \mathbf{\Lambda}(\Phi_o, \Psi_o) \end{aligned}$$

Assumption 1 allows a large variety of curves in $\mathbb{C}$ but only the union of segments of a circle or straight line has been found to also comply with Assumption 2. The similarity transformation defines a mapping between the curves $\mathbf{\Lambda}(\Phi, \Psi)$ and $\mathbf{\Lambda}(\Phi_o, \Psi_o)$, which corresponds to a segment of the imaginary axis. Each $s \in \mathbf{\Lambda}(\Phi_o, \Psi_o)$ is mapped onto the $l$ roots grouped in $\mathcal{R}_T(s) \subset \mathbf{\Lambda}(\Phi, \Psi)$, while all $\lambda \in \mathcal{R}_T(s)$ are mapped into the same $s$:

$$s = \begin{cases} \frac{[1\ 0]\ TL_l(\lambda)}{[0\ 1]\ TL_l(\lambda)} = \frac{t_1(\lambda)}{t_2(\lambda)} & \text{if } t_2(\lambda) \neq 0 \\ \infty & \text{otherwise} \end{cases}$$

The similarity transformation can be reformulated as:

$$1 - s\frac{t_2(\lambda)}{t_1(\lambda)} = 0$$

The curve $\mathbf{\Lambda}(\Phi, \Psi)$ corresponds to the set of complex numbers $\lambda$ that solve this root locus equation for some $s \in \mathbf{\Lambda}(\Phi_o, \Psi_o)$. As for every such $s$ the roots in $\mathcal{R}_T(s)$ must be distinct, only root locuses without branching points are allowed.

*Pipeleers*, *Iwasaki* and *Hara* [51, Section 3] describe the construction of the matrixes $\Phi, \Psi$ and $R$ over a union of segments of the real axis. The extension of this mapping to the imaginary axis or unit circle in the complex plane is obtained by a Möbius transformation. Firstly, on the real axis $\Phi_r$ and $\Psi_r$ are defined by:

$$\begin{aligned} \Phi_r &= \begin{bmatrix} 0 & \imath \\ -\imath & 0 \end{bmatrix} \\ \Psi_r &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{aligned}$$

such that $\mathbf{\Lambda}(\Phi_r, \Psi_r) = \mathbb{R}_+ \cup \{\infty\}$. In addition, for $l \in \mathbb{I}$, the matrix $J_l \in \mathbb{R}^{2l\times(l+1)}$ is defined as:

$$J_l = \begin{bmatrix} I_l & 0_{l,1} \\ 0_{l,1} & I_l \end{bmatrix}$$

*Lemma* [51, Lemma 3.1]: Let $2l$ scalars $\alpha_k, \beta_k \in \mathbb{R} \cup \{\infty\}$, $k \in \mathbb{I}_{\leqslant}$, be given that satisfy

$$\alpha_1 < \beta_1 < \alpha_2 < \cdots < \beta_{l-1} < \alpha_l < \beta_l$$

Let the vectors $a, b \in \mathbb{R}^{l+1}$ be defined by:

$$\prod_{k=1}^{l} (\lambda - \alpha_k) = a^\top L_l(\lambda)$$
$$\prod_{k=1}^{l} (\lambda - \beta_k) = b^\top L_l(\lambda)$$

where $(\lambda - \alpha_1) = 1$ for $\alpha_1 = -\infty$ and $(\lambda - \beta_l) = 1$ for $\beta_l = \infty$ and set $\tilde{T} = [-a\ b]^\top$. Then the matrixes $\Phi, \Psi \in \mathbb{H}^{l+1}$ given by:

$$\Phi = \tilde{T}^\dagger \Phi_r \tilde{T}$$
$$\Psi = \tilde{T}^\dagger \Psi_r \tilde{T}$$

satisfy Assumptions 1 and 2, and

$$\mathbf{\Lambda}(\Phi, \Psi) = \bigcup_{k=1}^{l} [\alpha_k, \beta_k]$$

In particular, a matrix $R \in \mathbb{S}^l$ satisfying Assumption 2 is given by the unique solution of:

$$\Phi = J_l^\dagger (\Phi_r \otimes R) J_l$$

*Pipeleers*, *Iwasaki* and *Hara* [51, Example 2] show an example of the union of two discrete time frequency intervals on the unit circle:

$$\mathbf{\Lambda}(\Phi, \Psi) = \left\{\lambda = e^{\imath\theta} \ : \ \theta \in [\eta_1, \zeta_1] \cup [\eta_2, \zeta_2]\right\}$$

with $-\pi < \eta_1 < \zeta_1 < \eta_2 < \zeta_2 < \pi$. A Möbius transform that maps the unit circle to the real axis is:

$$\mu(\lambda) = -\imath \frac{\lambda - 1}{\lambda + 1} \quad \rightarrow \quad M = \begin{bmatrix} 0 & c \\ \bar{c} & 0 \end{bmatrix}$$

The two frequency intervals are mapped into the real intervals $\begin{bmatrix} \alpha_k & \beta_k \end{bmatrix}$ with $\alpha_k = \mu(e^{\imath\eta_k})$ and $\beta_k = \mu\left(e^{\imath\zeta_k}\right)$. The corresponding matrixes $\Phi$ and $\Psi$ are:

$$\Phi = \tilde{T}^\dagger \begin{bmatrix} 0 & \imath c \\ -\imath\bar{c} & 0 \end{bmatrix} \tilde{T}$$
$$\Psi = \tilde{T}^\dagger \begin{bmatrix} 0 & c \\ \bar{c} & 0 \end{bmatrix} \tilde{T}$$

with:

$$\tilde{T} = \begin{bmatrix} -1 & e^{\imath\eta_1} + e^{\imath\eta_2} & -e^{\imath\eta_1} e^{\imath\eta_2} \\ 1 & -e^{\imath\zeta_1} - e^{\imath\zeta_2} & e^{\imath\zeta_1} e^{\imath\zeta_2} \end{bmatrix}$$
$$c = (1 + \imath\alpha_1)(1 + \imath\alpha_2)(1 - \imath\beta_1)(1 - \imath\beta_2)$$
$$R = M^\dagger \begin{bmatrix} \beta_1 + \beta_2 - \alpha_1 - \alpha_2 & \alpha_1\alpha_2 - \beta_1\beta_2 \\ \alpha_1\alpha_2 - \beta_1\beta_2 & \beta_1\beta_2(\alpha_1 + \alpha_2) - \alpha_1\alpha_2(\beta_1 + \beta_2) \end{bmatrix} M$$

### O.2.2 Generalised KYP lemma over a union of frequency intervals

For $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$ and $l \in \mathbb{I}$, define $F_l(A, B) \in \mathbb{C}^{(l+1)n\times(n+ml)}$:

$$F_l(A, B) = \begin{bmatrix} A^l & A^{l-1}B & A^{l-2}B & \cdots & B \\ A^{l-1} & A^{l-2}B & \cdots & B & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ A & B & 0 & \cdots & 0 \\ I & 0 & \cdots & \cdots & 0 \end{bmatrix}$$

Similarly, define $G_l(A,B) \in \mathbb{C}^{(n+m)\times(n+ml)}$[d]:

$$G_l(A,B) = \left(I_l \otimes \begin{bmatrix} I_n \\ 0_{m,n} \end{bmatrix}\right) \begin{bmatrix} F_{l-1}(A,B) & 0_{nl,m} \end{bmatrix} + \left(I_l \otimes \begin{bmatrix} 0_{n,m} \\ I_m \end{bmatrix}\right) \begin{bmatrix} 0_{ml,n} & F_{l-1}(0_{m,m}, I_m) \end{bmatrix}$$

For example:

$$G_2(A,B) = \begin{bmatrix} A & B & 0_{n,m} \\ 0_{m,n} & 0_{m,m} & I_m \\ I_n & 0_{n,m} & 0_{n,m} \\ 0_{m,n} & I_m & 0_{m,m} \end{bmatrix}$$

In addition, for $\lambda \in \mathbb{C}$ define the set $\mathcal{N}_{A,B}(\lambda)$ as:

$$\mathcal{N}_{A,B}(\lambda) = \begin{cases} (x,u) \in \mathbb{C}^n \times \mathbb{C}^m \ : \ (\lambda I - A)\, x = Bu & \text{for } \lambda \neq \infty \\ \{0\} \times \mathbb{C}^m & \text{for } \lambda = \infty \end{cases}$$

Note the if $\det(\lambda I - A) \neq 0$, every element of $\mathcal{N}_{A,B}(\lambda)$ is of the form:

$$\begin{bmatrix} (\lambda I - A)^{-1} B \\ I \end{bmatrix} u$$

for some $u \in \mathbb{C}^m$.

*Theorem* [51, Theorem 2.2]: Let Hermitian matrixes $\Phi,\ \Psi \in \mathbb{H}^{l+1}$ and $\Theta \in \mathbb{H}^{m+n}$, and matrixes $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{n\times m}$, be given, with $B$ of full column rank and $(A,B)$ controllable. Suppose $\Phi$ and $\Psi$ satisfy Assumptions 1 and 2, and let $R \in \mathbb{H}^l$ be a matrix satisfying Assumption 2. Then the following statements are equivalent:

1. The inequality:

$$\begin{bmatrix} x \\ u \end{bmatrix}^{\dagger} \Theta \begin{bmatrix} x \\ u \end{bmatrix} \leq 0$$

   holds for all $(x,u) \in \mathcal{N}_{A,B}(\lambda)$

2. There exist $P, Q \in \mathbb{H}^n$ that satisfy $Q \succeq 0$ and

$$F_l(A,B)^{\dagger} (\Phi \otimes P + \Psi \otimes Q)\, F_l(A,B) + G_l(A,B)^{\dagger} (R \otimes \Theta)\, G_l(A,B) \preceq 0$$

### O.2.3 Example of design of a non-symmetric band pass FIR filter with the union of the upper and lower stop bands

Section N.9 shows examples of the design of the band pass filter example shown by *Pipeleers et al.* [51, Figure 4]. The Octave script *directFIRnonsymmetric_kyp_union_bandpass_test.m* designs a band pass filter with the stop band LMI expressed as the union of the upper and lower stop band frequency intervals, specified as $[0,\ f_{asl}] \cup [f_{asu},\ \pi]$, so that $0 = \eta_1 < \zeta_1 < \eta_2 < \zeta_2 = \pi$. In this case, the example of the union of two frequency intervals provided by *Pipeleers et al.* does not apply and the Möbius transformation must be constructed with $\alpha_1 = 0$ and $\beta_2 = \infty$ since:

$$\begin{aligned} \mu\left(e^{\imath\theta}\right) &= -\imath \frac{e^{\imath\theta} - 1}{e^{\imath\theta} + 1} \\ &= \tan\frac{\theta}{2} \end{aligned}$$

and:

$$\lim_{\theta \to \pi-} \tan\frac{\theta}{2} = +\infty$$

[d] *Pipeleers et al.* [51, Equation 2.8] show the last term as $F_l(0_{m,m}, I_m)$.

In this case:

$$
\begin{aligned}
a &= \begin{bmatrix} 1 & -\alpha_2 & 0 \end{bmatrix}^\top \\
b &= \begin{bmatrix} 0 & -1 & \beta_1 \end{bmatrix}^\top \\
\tilde{T} &= \begin{bmatrix} -1 & \alpha_2 & 0 \\ 0 & -1 & \beta_1 \end{bmatrix} \\
\Phi &= \begin{bmatrix} -1 & 0 \\ \alpha_2 & -1 \\ 0 & \beta_1 \end{bmatrix} \begin{bmatrix} 0 & \imath \\ -\imath & 0 \end{bmatrix} \begin{bmatrix} -1 & \alpha_2 & 0 \\ 0 & -1 & \beta_1 \end{bmatrix} \\
&= \begin{bmatrix} -1 & 0 \\ \alpha_2 & -1 \\ 0 & \beta_1 \end{bmatrix} \begin{bmatrix} 0 & -\imath & \imath\beta_1 \\ \imath & -\imath\alpha_2 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & \imath & -\imath\beta_1 \\ -\imath & 0 & \imath\alpha_2\beta_1 \\ \imath\beta_1 & -\imath\alpha_2\beta_1 & 0 \end{bmatrix} \\
\Psi &= \begin{bmatrix} -1 & 0 \\ \alpha_2 & -1 \\ 0 & \beta_1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & \alpha_2 & 0 \\ 0 & -1 & \beta_1 \end{bmatrix} \\
&= \begin{bmatrix} -1 & 0 \\ \alpha_2 & -1 \\ 0 & \beta_1 \end{bmatrix} \begin{bmatrix} 0 & -1 & \beta_1 \\ -1 & \alpha_2 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 1 & -\beta_1 \\ 1 & -2\alpha_2 & \alpha_2\beta_1 \\ -\beta_1 & \alpha_2\beta_1 & 0 \end{bmatrix}
\end{aligned}
$$

$R$ is the solution of:

$$
\begin{aligned}
\Phi &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \imath r_{11} & 0 & \imath r_{12} \\ -\imath r_{11} & 0 & -\imath r_{12} & 0 \\ 0 & \imath r_{12} & 0 & \imath r_{22} \\ -\imath r_{12} & 0 & -\imath r_{22} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \imath r_{11} & \imath r_{12} \\ -\imath r_{11} & -\imath r_{12} & 0 \\ 0 & \imath r_{12} & \imath r_{22} \\ -\imath r_{12} & -\imath r_{22} & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & \imath r_{11} & \imath r_{12} \\ -\imath r_{11} & 0 & \imath r_{22} \\ -\imath r_{12} & -\imath r_{22} & 0 \end{bmatrix}
\end{aligned}
$$

Thus:

$$
R = \begin{bmatrix} 1 & -\beta_1 \\ -\beta_1 & \alpha_2\beta_1 \end{bmatrix}
$$

For the stop band, $|H|^2 \leq \varepsilon_s^2$, and:

$$
\Pi = \begin{bmatrix} 1 & 0 \\ 0 & -\varepsilon_s^2 \end{bmatrix}
$$

The terms in $\begin{bmatrix} C & D \end{bmatrix}$ are linearised by applying the Schur complement to $R \otimes \Theta$:

$$
\begin{aligned}
R \otimes \Theta &= R \otimes \left( \begin{bmatrix} C & D \\ 0 & I \end{bmatrix}^\dagger \begin{bmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{12}^\dagger & \Pi_{22} \end{bmatrix} \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} \right) \\
&= R \otimes \begin{bmatrix} 0 & C^\dagger \Pi_{12} \\ \Pi_{12}^\dagger C & D^\dagger \Pi_{12} + \Pi_{12}^\dagger D + \Pi_{22} \end{bmatrix} + R \otimes \left( \begin{bmatrix} C & D \end{bmatrix}^\dagger \Pi_{11} \begin{bmatrix} C & D \end{bmatrix} \right) \\
&= R \otimes \begin{bmatrix} 0 & C^\dagger \Pi_{12} \\ \Pi_{12}^\dagger C & D^\dagger \Pi_{12} + \Pi_{12}^\dagger D + \Pi_{22} \end{bmatrix} + \left( I_2 \otimes \begin{bmatrix} C & D \end{bmatrix} \right)^\dagger (R \otimes \Pi_{11}) \left( I_2 \otimes \begin{bmatrix} C & D \end{bmatrix} \right)
\end{aligned}
$$

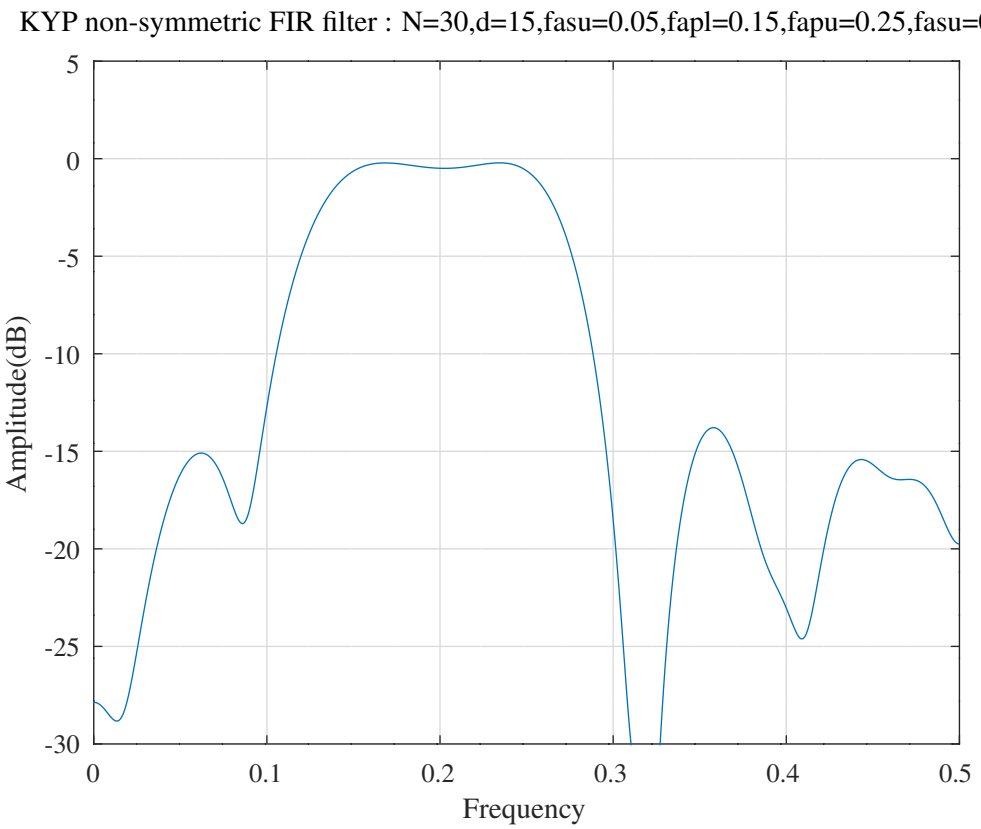



Figure O.9: Amplitude response of a non-symmetric FIR band pass filter designed with the generalised KYP lemma over a union of stop band regions.

where I have made repeated use of the mixed product rule, $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$.

The Octave script *directFIRnonsymmetric_kyp_union_bandpass_test.m* uses *Pipeleer et al.*'s generalised KYP lemma to design a band pass FIR filter with the following specification, similar to that of their example [51, Figure 4]:

```
N=30 % FIR filter order
d=15 % Nominal FIR filter delay
fasl=0.05 % Amplitude stop band lower edge
fapl=0.15 % Amplitude pass band lower edge
fapu=0.25 % Amplitude pass band upper edge
fasu=0.35 % Amplitude stop band upper edge
Esq_z=0.01 % Squared amplitude pass band - delay error
Esq_s=1e-06 % Squared amplitude stop band error
```

The resulting FIR impulse response is:

```
h = [ -0.0000863536,  0.0045206514, -0.0246970128, -0.0013927864, ...
      -0.0211035218, -0.0523641214, -0.0170485311,  0.0333698035, ...
       0.0123636532,  0.0125932183,  0.0685008238,  0.0355249100, ...
      -0.1461948601, -0.1759365802,  0.0727690680,  0.3676607794, ...
       0.0714861380, -0.1845297421, -0.1622513002,  0.0144234157, ...
       0.0457802681, -0.0125225388,  0.0003169245,  0.0380461241, ...
       0.0012412903, -0.0417205380,  0.0266507435, -0.0064239148, ...
       0.0005978355, -0.0000131737,  0.0000001173 ];
```

Figure O.9 shows the amplitude response. Figure O.10 shows the pass band amplitude, phase error and delay responses. The pass band phase error is adjusted for the nominal delay.

**Unfortunately, this implementation does not succeed in finding a correct solution for the stop band attenuation.**

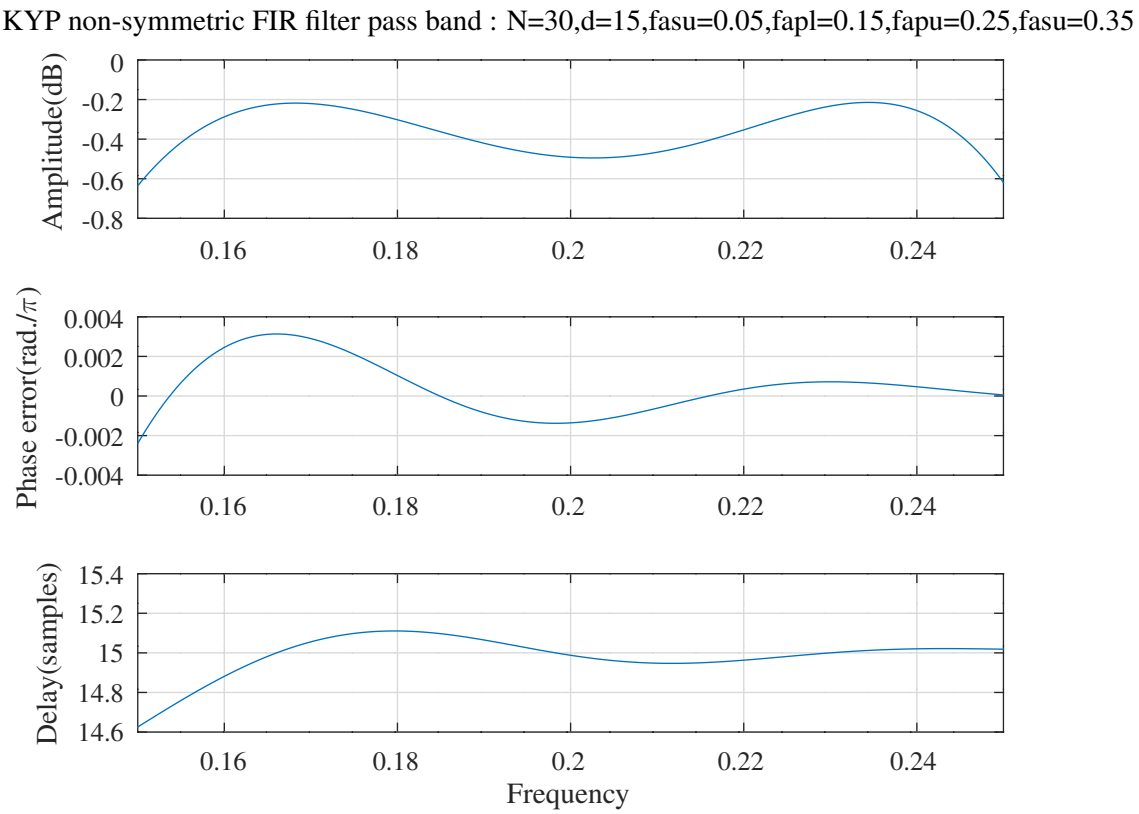



Figure O.10: Pass band amplitude, phase error and delay responses of a non-symmetric FIR band pass filter designed with the generalised KYP lemma over a union of stop band regions. The pass band phase error is adjusted for the nominal delay.

# Colophon

## Software requirements

I currently use the Fedora 36 distribution of Linux [47] with:

- `uname -r` is *5.19.8-200.fc36.x86_64*
- *gcc* version *12.1.1 20220819 (Red Hat 12.2.1-1)*
- a local build of *octave-7.2.0* with the *struct*, *io*, *statistics*, *optim*, *control*, *signal*, *symbolic*[e] and *parallel* Octave-Forge packages
- the Octave scripts use the *qt* graphics toolkit with *qt-4.8.7-67* and a local build of *GraphicsMagick*
- *texlive-2021-55* and various packages are installed
- *dia-0.97.3-20* is used for line drawings

The local build of *octave-7.2.0* and Octave-Forge packages assumes that the following packages are installed in addition to the base Fedora distribution:

```
dnf install \
readline-devel make lzip sharutils gcc gcc-c++ gcc-gfortran gmp-devel \
mpfr-devel cmake gnuplot-latex m4 gperf bison flex openblas-devel patch \
texinfo texinfo-tex librsvg2 librsvg2-devel librsvg2-tools icoutils autoconf \
automake libtool pcre pcre-devel freetype freetype-devel gnupg2 python3-sympy \
rapidjson-devel
```

The oct-file *src/minphase.cc* assumes the *eigen3-devel* package is installed.

The following packages are required to build this document:

```
dnf install dia epstool \
texlive-algorithmicx texlive-appendix texlive-boondox \
texlive-calculator texlive-chngcntr texlive-dvipng texlive-dvisvgm \
texlive-environ texlive-epstopdf texlive-esint texlive-esint-type1 \
texlive-fontaxes texlive-fouriernc texlive-fourier texlive-framed \
texlive-gsftopk texlive-kpfonts texlive-latex-base-dev texlive-latex-bin-dev \
texlive-latex-graphics-dev texlive-ly1 texlive-mathdesign \
texlive-multirow texlive-nag texlive-needspace texlive-newpx \
texlive-newtx texlive-obsolete texlive-pdfcrop texlive-powerdot \
texlive-pst-blur texlive-pst-pdf texlive-pst-slpe texlive-rotfloat \
texlive-scheme-basic texlive-threeparttable texlive-tocbibind \
texlive-trimspaces texlive-type1cm texlive-upquote texlive-wrapfig \
texlive-dvisvgm
```

[e] The *symbolic* Octave-Forge package assumes that the *python3-sympy* Fedora package is installed.

## *Makefile*

This document is built with *GNU make-4.3-7*. The *Makefile* in the project archive builds the PDF version of this document from the LaTeXsource in *DesignOfIIRFilters.tex*. The project archive contains the files required to generate the figures shown in this document. The line diagrams are in *dia* format [55] 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 5.7.1, has the corresponding *Makefile* fragment, *butt3NS_test.mk*[f]:

```
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 \
tf2schurNSlattice.m schurNSlatticeNoiseGain.m schurNSlatticeFilter.m \
svf.m KW.m optKW.m tf2Abcd.m crossWelch.m p2n60.m qroots.m \
schurexpand.oct schurdecomp.oct schurNSscale.oct \
schurNSlattice2Abcd.oct qzsolve.oct
```

.

## Octave

The *build-octave.sh* script builds a patched, local, command-line only, version of *octave-7.2.0* avoiding the vagaries of the Fedora packagers.

The *build-octave.sh* script builds Octave and the associated numerical libraries with the *g++* option `-march=nehalem`. The patch file, *octave.patch*, is a shell *here* document within *build-octave.sh*. The *octave-7.2.0/configure* script is patched so that the Octave `OCTAVE_VERSION` command returns `7.2.0-robj`.

The local build of Octave links to locally built versions of the following libraries: *arpack-ng-3.8.0* [11], *fftw-3.3.10* [48], *glpk-5.0* [165], *lapack-3.10.1* [110], *qrupdate-1.1.2* [167], *SuiteSparse-5.13.0* [197], *sundials-6.3.0* [114] and *GraphicsMagick-1.3.38* [90]. The local build of the *LAPACK* library includes the deprecated routines.

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.

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.

The *fftw-3.3.8* release notes (May 28th, 2018-2019-2020) state:

> Fixed AVX, AVX2 for gcc-8. By default, FFTW 3.3.7 was broken with gcc-8. AVX and AVX2 code assumed that the compiler honors the distinction between +0 and -0, but gcc-8 -ffast-math does not. The default CFLAGS included -ffast-math . This release ensures that FFTW works with gcc-8 -ffast-math , and removes -ffast-math from the default CFLAGS for good measure.

[f] *github.com* enforces a limit of 1000 files per directory in a repository. To conform to this limit the **_test.m* Octave scripts were moved to *src/test* and the **_test.mk* Makefile fragments were moved to *src/mk*.

### Octave packages and solvers

The *build-octave.sh* script installs the following Octave Forge packages:

```
computer=x86_64-pc-linux-gnu
octave version=7.2.0-robj (HG-ID=6c1e310b2230)
Package Name  | Version | Installation directory
--------------+---------+-----------------------
     control *|   3.4.0 | .../octave-7.2.0/share/octave/packages/control-3.4.0
          io  |   2.6.4 | /usr/local/octave-7.2.0/share/octave/packages/io-2.6.4
       optim  |   1.6.2 | .../octave-7.2.0/share/octave/packages/optim-1.6.2
    parallel  |   4.0.1 | .../octave-7.2.0/share/octave/packages/parallel-4.0.1
      signal *|   1.4.2 | .../octave-7.2.0/share/octave/packages/signal-1.4.2
  statistics  |   1.4.3 | .../share/octave/packages/statistics-1.4.3
      struct  |  1.0.18 | .../octave-7.2.0/share/octave/packages/struct-1.0.18
    symbolic  |   3.0.1 | .../octave-7.2.0/share/octave/packages/symbolic-3.0.1
```

The *build-octave.sh* script installs the *SeDuMi*, *SDPT3*, *YALMIP* and *SparsePOP* SDP solvers from the forked repositories at `https://github.com/robertgj` and an Octave version of *gloptipoly3*.

### Problems with Octave

The *octave.patch* here document within *build-octave.sh* patches files to add an Octave warning id and to prevent a *valgrind* warning.

### Problems with Octave-Forge packages

The following packages are patched with shell *here* documents within *build-octave.sh*:

- the *signal-1.4.2* package `zplane` function is patched to avoid a LaTeX error and the `grpdelay` function is patched to correctly handle the `grpdelay(a,b,w` API.
- the *parallel-4.0.1* package is patched to make it compatible with *octave-7.2.0*.
- *SparsePOP* uses the `collect` function from *SymPy*. This is not included in the *symbolic-3.0.1* package. I add a version of *collect.m* based on *factor.m*.

## Building Octave

The Octave builtin function `__config_octave_info__` shows the build configuration.

### Building Octave with LTO and PGO

The following commands build Octave linked with the system *lapack* and *blas* shared libraries with link-time-optimisation (LTO) and profile-guided-optimisation (PGO):

```
#!/bin/sh

BLDOPTS="-m64 -mtune=generic -O2"
export CFLAGS="-I"$LOCAL_PREFIX"/include -std=c11 "$BLDOPTS
export CXXFLAGS="-I"$LOCAL_PREFIX"/include -std=c++11 "$BLDOPTS
export FFLAGS=$BLDOPTS
export LDFLAGS="-L"$LAPACK_DIR" -L"$LOCAL_PREFIX"/lib"

$OCTAVE_DIR/configure $OCTAVE_CONFIG_OPTIONS \
  --prefix=$OCTAVE_INSTALL_DIR --with-blas="-lblas" --with-lapack="-llapack"
```

```
export PGO_GEN_FLAGS="-pthread -fopenmp -fprofile-generate"
make XTRA_CFLAGS=$PGO_GEN_FLAGS XTRA_CXXFLAGS=$PGO_GEN_FLAGS V=1 -j6
find . -name \*.gcda -exec rm {} ';'
make V=1 -j6 check
find . -name \*.o -exec rm -f {} ';'
find . -name \*.lo -exec rm -f {} ';'
find . -name \*.la -exec rm -f {} ';'
export PGO_LTO_FLAGS="-pthread -fopenmp -fprofile-use -flto=6 -ffat-lto-objects"
make XTRA_CFLAGS="$PGO_LTO_FLAGS" XTRA_CXXFLAGS="$PGO_LTO_FLAGS" V=1 -j6
```

The PGO profile information is generated by running the Octave test suite.

#### Building Octave for debugging

To build a debugging version of Octave:

```
#!/bin/sh

BLDOPTS="-m64 -O0"
export CFLAGS="-I"$LOCAL_PREFIX"/include -std=c11 -ggdb3 "$BLDOPTS
export CXXFLAGS="-I"$LOCAL_PREFIX"/include -std=c++11 -ggdb3 "$BLDOPTS
export FFLAGS="-ggdb3 "$BLDOPTS
export LDFLAGS="-L"$LAPACK_DIR" -L"$LOCAL_PREFIX"/lib"

$OCTAVE_DIR/configure $OCTAVE_CONFIG_OPTIONS \
  --prefix=$OCTAVE_INSTALL_DIR --with-blas="-lblas" --with-lapack="-llapack"

make V=1 -j 6
```

#### Building Octave oct-files for debugging

To debug an oct-file with *valgrind* and *gdb*:

```
valgrind --vgdb=yes --vgdb-error=0 octave-cli -p src src/test/scriptname
```

In a separate shell run:

```
   gdb octave-cli
```

and issue the following gdb commands:

```
   target remote | vgdb
   continue
```

To test an oct-file with *address-sanitizer* add these flags:

```
-g -fsanitize=address -fsanitize=undefined -fno-sanitize=vptr -fno-omit-frame-pointer
```

and “pre-load” the *address-sanitizer* library:

```
LD_PRELOAD=/usr/lib64/libasan.so.8 octave-cli --eval oct_file_test_script
```

### Installing Octave-Forge packages

To install Octave packages from a remote Octave-Forge file-server:

```
octave-cli --eval 'pkg install -forge struct io statistics 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 of Octave and benchmarking *blas* and *lapack* libraries with a shared library *LTO* and *PGO* build of Octave. The Linux kernel used was *5.19.8-200.fc36.x86_64* from the RedHat Fedora 36 Linux distribution. The system compiler is *gcc version 12.1.1 20220819 (Red Hat 12.2.1-1)*. 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 $4.5GHz$. To do this, as the *root* user:

```
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 O.2. It is assumed to run in the *benchmark* sub-directory. The Octave build benchmark test loops 100 times finding the amplitude, group delay and phase response of a filter. Numerical differences between the *blas* and *lapack* implementations make it impossible to usefully benchmark with a filter design script like *iir_sqp_slb_bandpass_test.m* or *decimator_R2_test.m*. 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*. The Octave build is *configure*d for command-line only and local libraries with the folowing options:

```
export OCTAVE_CONFIG_OPTIONS=" \
       --disable-docs \
       --disable-java \
       --disable-atomic-refcount \
       --without-fltk \
       --without-qt \
       --without-sndfile \
       --without-portaudio \
       --without-qhull \
       --without-magick \
       --without-glpk \
       --without-hdf5 \
       --with-arpack-includedir=$LOCAL_PREFIX/include \
       --with-arpack-libdir=$LOCAL_PREFIX/lib \
       --with-qrupdate-includedir=$LOCAL_PREFIX/include \
       --with-qrupdate-libdir=$LOCAL_PREFIX/lib \
       --with-amd-includedir=$LOCAL_PREFIX/include \
       --with-amd-libdir=$LOCAL_PREFIX/lib \
       --with-camd-includedir=$LOCAL_PREFIX/include \
       --with-camd-libdir=$LOCAL_PREFIX/lib \
       --with-colamd-includedir=$LOCAL_PREFIX/include \
       --with-colamd-libdir=$LOCAL_PREFIX/lib \
       --with-ccolamd-includedir=$LOCAL_PREFIX/include \
       --with-ccolamd-libdir=$LOCAL_PREFIX/lib \
       --with-cholmod-includedir=$LOCAL_PREFIX/include \
       --with-cholmod-libdir=$LOCAL_PREFIX/lib \
```

| Octave build | IIR benchmark execution time (seconds) |
|---|---|
| Debug, shared reference lapack and blas | 272.9 |
| Generic, shared reference lapack and blas | 47.5 |
| Generic, shared reference lapack and blas, LTO | 47.8 |
| Generic, shared reference lapack and blas, PGO | 48.7 |
| Generic, shared reference lapack and blas, LTO, PGO | 47.0 |

Table O.2: Average execution time for 10 runs of IIR benchmark scripts with various *Octave* builds.

```
--with-cxsparse-includedir=$LOCAL_PREFIX/include \
--with-cxsparse-libdir=$LOCAL_PREFIX/lib \
--with-umfpack-includedir=$LOCAL_PREFIX/include \
--with-umfpack-libdir=$LOCAL_PREFIX/lib \
--with-fftw3-includedir=$LOCAL_PREFIX/include \
--with-fftw3-libdir=$LOCAL_PREFIX/lib \
--with-fftw3f-includedir=$LOCAL_PREFIX/include \
--with-fftw3f-libdir=$LOCAL_PREFIX/lib"
```

### Benchmarking *blas* and *lapack* libraries

The shell script *library-benchmark.sh* benchmarks the shared library *LTO* and *PGO* version of Octave with the *linpack.m* [120] script and the previous IIR benchmark script. The *linpack.m* script was modified to produce repeatable residuals. The alternative *blas* and *lapack* library versions used are local builds of *blas-3.10.1* and *lapack-3.10.1* and the Fedora 36 *lapack*, *blas*, *atlas* [13], *gsl* [163] and *openblas* [142] packages[g]

```
atlas.x86_64                3.10.3-19.fc36
blas.x86_64                 3.10.1-1.fc36
lapack.x86_64               3.10.1-1.fc36
gsl.x86_64                  2.6-6.fc36
gsl-devel.x86_64            2.6-6.fc36
openblas.x86_64             0.3.19-3.fc36
openblas-threads.x86_64     0.3.19-3.fc36
```

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 *libtatlas* was set at compile-time by the package builder. For *libopenblasp*, the number of threads is set to, for example 2, by:

```
export OPENBLAS_NUM_THREADS=2
```

Table O.3 shows the average MFLOPS or execution time for 10 runs of each test using the shared library *PGO* and *LTO* build of Octave. The normalised residuals for the two ATLAS libraries varied between two values. The largest is shown.

### Benchmarking *freqz*, *iirA* and *schurOneMPAlatticeAsq*

The Octave script *benchmark_iirA_freqz.sh* compares the average execution time of the Octave *signal* package function *freqz* (calculated at equal frequency intervals with an FFT) and the *iirA* and *schurOneMPAlatticeAsq* functions (calculated at arbitrary frequencies with matrix multiplication). Figure O.11 shows the mean execution times for frequency vectors with a length that is a multiple of 200. Figure O.12 shows the mean execution times for frequency vectors with a length that is a power of 2.

[g]The test script *test/02/t0294a.sh* compares the output of *DGESVD* for a problematic matrix and the *libtatlas*, system *libblas* and "local" *libblas* libraries:
> ./dgesvd_test_atlas
0.14709002182058989
> ./dgesvd_test_sysblas
0.14709002182060862
Note: The following floating-point exceptions are signalling: IEEE_DENORMAL
> LD_LIBRARY_PATH=/usr/local/octave-7.1.0/lib ./dgesvd_test_blas
0.14709002182060862

| *blas* library | linpack.m | | IIR benchmark script |
|---|---|---|---|
| | MFLOPS | Normalised residual | Execution time (seconds) |
| Generic libblas/liblapack | 3945 | 21.3 | 46.9 |
| Intel libblas/liblapack | 3871 | 21.3 | 46.8 |
| Haswell libblas/liblapack | 4737 | 18.5 | 46.4 |
| Nehalem libblas/liblapack | 3881 | 21.3 | 46.8 |
| Skylake libblas/liblapack | 4506 | 18.5 | 46.7 |
| Fedora 36 libblas/liblapack | 3908 | 21.3 | 46.9 |
| libgslcblas | 3912 | 21.3 | 46.9 |
| libsatlas | 13217 | 14.3 | 46.5 |
| libtatlas | 20725 | 14.3 | 47.2 |
| libopenblas | 34054 | 9.9 | 45.3 |
| libopenblasp (1 thread) | 34065 | 9.9 | 45.3 |
| libopenblasp (2 threads) | 48710 | 12.1 | 45.8 |
| libopenblasp (4 threads) | 62354 | 10.8 | 45.6 |

Table O.3: Benchmark results for *linpack.m* and IIR benchmark with various *blas* and *lapack* implementations.

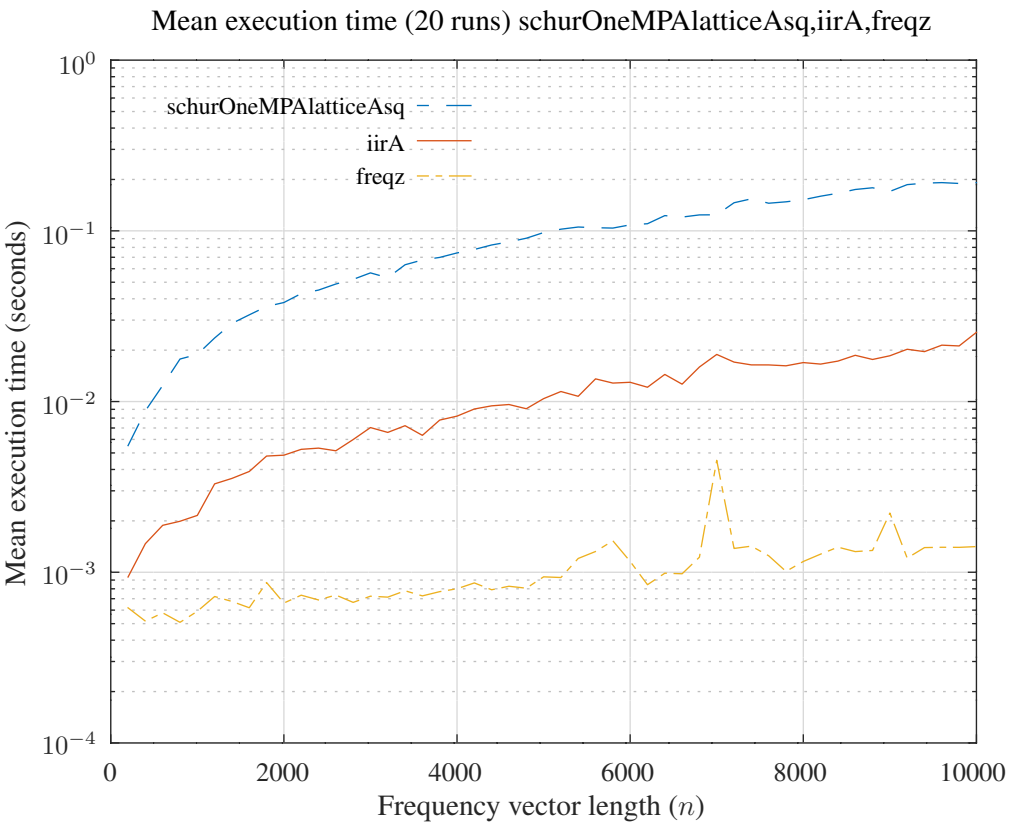



Figure O.11: Mean execution times of *freqz*, *iirA* and *schurOneMPAlatticeAsq* for frequency vectors with a length that is a multiple of 200

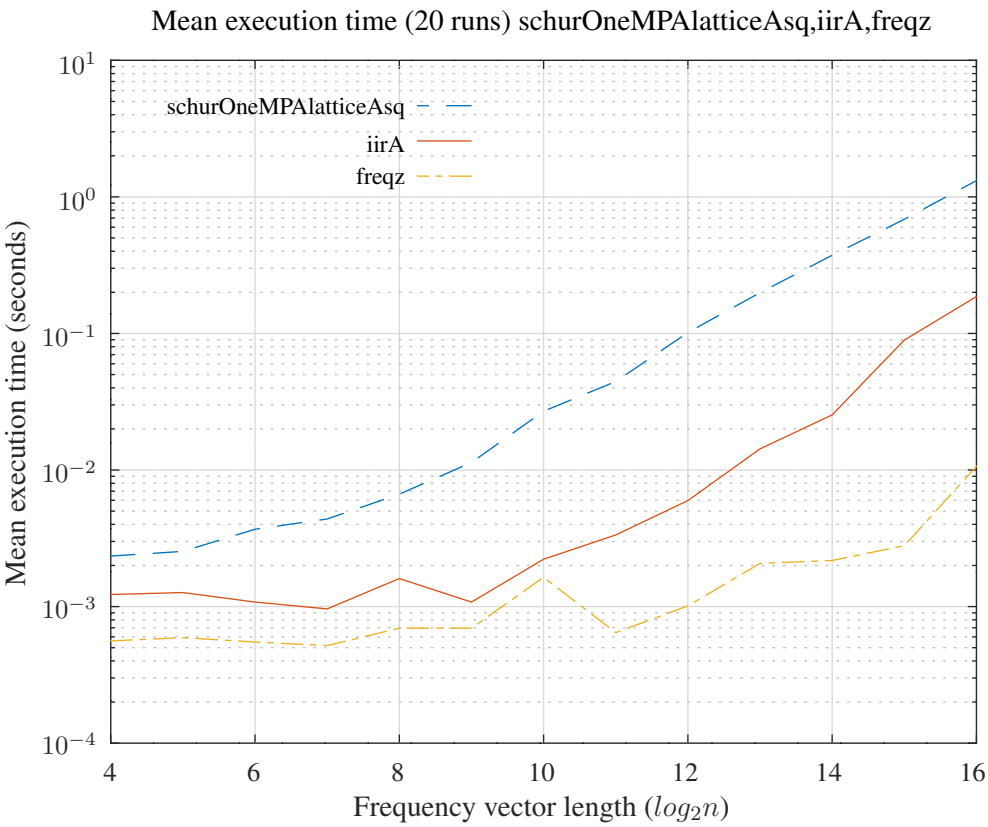



Figure O.12: Mean execution times of *freqz*, *iirA* and *schurOneMPAlatticeAsq* for frequency vectors with a length that is a power of 2

# Solvers

## SeDuMi

The *SeDuMi* source is available from *LeHigh University* as *SeDuMi_1_3.zip* [194]. Unfortunately, this version does not seem to be actively maintained. An "unofficial" *GitHub* [193] fork is compatible with Octave. A fork of this version is used in this project. It runs without modification under Octave[h].

The *SeDuMi* source depends on the *OpenBLAS f77blas.h* and *openblas_config.h* headers. *install_sedumi.m* assumes they are present in */usr/include/openblas*. A 64-bit version of *SeDuMi* requires *-DOPENBLAS_USE64BITINT*. The *install_sedumi.m* script can be called with a *mex* template. For example, to enable the *mxAssert* checks:

```
octave:1> mexcmd = ['mex -O2 -DMEX_DEBUG -DOCTAVE -I/usr/include/openblas %s '];
octave:2> install_sedumi ('-rebuild', mexcmd);
```

### Problems with SeDuMi

The *SeDuMi* User Guide [195, Section 4] shows an example of optimising a complex valued Toeplitz matrix. The Octave script *sedumi_toepest_test.m* runs this example. Unfortunately, the $Z$ matrix found by this script is different to that shown in the *SeDuMi* User Guide. The Octave script *sedumi_real_toepest_test.m* expresses the $Z$ and $P$ matrixes in terms of the real and imaginary parts of the matrix elements (see Appendix B.5). The $Z$ matrix found by that script agrees with that shown in the *SeDuMi* User Guide. *Löfberg* [88] shows five different *YALMIP* solutions for this problem, reproduced in the Octave script *yalmip_complex_test.m*. The $Z$ matrixes found by that script agree with the $Z$ matrix shown in the *SeDuMi* User Guide. *YALMIP* supports complex-valued constraints for all solvers by automatically converting complex-valued problems to real-valued problems.

The Octave script *sedumi_profiler_test.m* profiles the *SeDuMi/examples/test_sedumi.m* script:

```
   #   Function Attr     Time (s)   Time (%)        Calls
-------------------------------------------------------
  89   binary *            59.960      31.37        49115
 153   psdscale            42.273      22.12         3971
 148    getada3            26.719      13.98          153
 173        eig            21.743      11.38         3926
 171   binary \             7.827       4.09         2370
  81   binary /             7.271       3.80        11906
 179 psdframeit             4.739       2.48          348
 168 psdinvjmul             4.693       2.46          173
```

The *SeDuMi* m-file *SeDuMi/psdscale.m* is complex and has no comments.

## SparsePOP

*SparsePOP* [68] is "a semidefinite relaxations package for polynomial programming". *SparsePOP303* runs under Octave with the modifications shown in *SparsePOP303.patch*.

The *readGMS.m* and *convert2.m* functions are altered to support the Octave-Forge *symbolic* package [141]. The latter is based on *SymPy* [196], a Python library for symbolic mathematics. The Octave *sparsePOP_test.m* script shows an example of *sparsePOP* reading the *Bex5_2_5.gms* GAMS format example with the Octave-Forge *symbolic* package.

[h] The *SeDuMi-1.3* source file *vec.m* shadows a built-in Octave function.

#### Problems with SparsePOP

The *sparsePOP_solveExample_test.m* script runs the SparsePOP *solveExample.m* function, with the *'sedumi'* solver, excluding problems $k_exclude = [36, 45, 46, 54, 70, 71, 88, 92]$. For those $k$ I catch exceptions like:

```
Caught exception at k=36!
## '2' is not defined in the line of 'Variables'.
## Should check the line of the objective function in 'Babel.gms'.
warning: Called readGMS>getObjPoly at line 1306
warning: Called readGMS at line 341
warning: Called sparsePOP at line 324
warning: Called solveExample at line 263
warning: Called solveExample_test at line 27
```

I run *sparsePOP* with *param.mex=0*. The *mexconv1* mex-file occasionally causes a *valgrind* warning when *make_mexdata* incorrectly calculates the size of *objPoly.supports*. For example, when running the SparsePOP *solveExample* function with *valgrind*:

```
85: randomwithEQ(20,2,4,4,3201)

SparsePOP 3.03
by H.Waki, S.Kim, M.Kojima, M.Muramatsu,
   H.Sugimoto and M.Yamashita, September 2018

==66559== Invalid read of size 8
==66559==    at 0xC346D21: mexFunction (mexconv1.cpp:180)
```

## SDPT3

*SDPT3* [94, 93, 171] is a MATLAB software for semidefinite-quadratic-linear programming. The version used in this project is a fork of the "unofficial" GitHub repository [130].

#### Problems with SDPT3

The *sdpt3.patch* file shows a modification that suppresses a divide-by-zero warning when running the *maxcut* example from *sqlpdemo* with a feasible initial solution.

## YALMIP

*YALMIP* [89, 82] is a toolbox for modelling and optimisation in MATLAB and Octave. *YALMIP* calls semi-definite programming (SDP) solvers such as *SeDuMi* and *SDPT3*.

## gloptipoly3

*gloptipoly3* [34, 35, 112, 113] is "intended to solve, or at least approximate, the Generalized Problem of Moments (GPM), an infinite-dimensional optimization problem which can be viewed as an extension of the classical problem of moments".

## MPSolve

The *MPSolve* [32] is a high precsion root-finder with an oct-file implementation. Examples of using the built-in Octave *roots*, the quad-float implementation in *qroots* and *mps_roots*:

```
octave-cli --eval 'norm(roots(bincoeff(20,0:20))+1)'
ans =  1.4835

octave-cli -p src --eval 'norm(qroots(bincoeff(20,0:20))+1)'
ans =  0.19206

octave-cli --eval 'pkg load mpsolve;norm(mps_roots(bincoeff(20,0:20))+1)'
ans =    1.0819e-13

octave-cli --eval 'pkg load mpsolve;norm(mps_roots(bincoeff(20,0:20))+1)'
ans =    1.1108e-13

octave-cli --eval 'pkg load mpsolve;norm(mps_roots(bincoeff(20,0:20))+1)'
ans =    1.0910e-13
```

Unfortunately, there are bugs:

```
octave:1> pkg load mpsolve
octave:2> pkg list mpsolve
Package Name  | Version | Installation directory
--------------+---------+-----------------------
     mpsolve *|   3.1.7 | .../octave-5.2.0/share/octave/packages/mpsolve-3.1.7
octave:3> mps_roots([1 0 0 0 0 0 0],'u')
ans =

   1.1840e-316 +  0.0000e+00i
    2.1322e-81 + 2.8664e+161i
   1.9253e+161 + 7.3681e+228i
    3.4546e-86 +  3.5641e-57i
   4.5663e+257 + 2.1335e+233i
   3.6247e+233 +  6.2591e-85i
```

# QEMU emulation

I endeavour to ensure that the Octave scripts required to build this document run successfully under the QEMU emulation of an Intel Nehalem CPU. The following is a general indication of how to set up QEMU emulation to achieve this.

## QEMU user-mode emulation

To run an Octave script with the QEMU user-mode emulation, run, for example:

```
qemu-x86_64 -cpu Nehalem `which octave` --no-gui -p src src/test/butt3NS_test.m
```

This runs the `octave-cli` binary and associated dynamic linker in the QEMU x86_64 user-mode emulation of an Intel Nehalem CPU.

To build this document with the QEMU user-mode emulation of Octave, run:

```
make OCTAVE="qemu-x86_64 -cpu Nehalem `which octave` --no-gui"
```

## QEMU virtual machine emulation

Unfortunately, user-mode QEMU emulation only uses one thread and is is not completely portable. I build `octave` and the associated linear algebra libraries with `-march=nehalem`. The system compiler links `octave` to the system *libm*, *libquadmath* and *libstdc++* shared libraries from the Fedora *glibc*, *libquadmath* and *libstdc++* packages, respectively. Running `gcc -v` shows that the system compiler is itself compiled with `-mtune=generic`. I believe that, currently, this is equivalent to `-march=nehalem` but it may change in future. QEMU emulation on a virtual machine allows use of known libraries, multiple CPUs, and builds this document much more quickly than user-mode emulation.

### Create the VM

*Buettner* [99] describes configuration of a QEMU [166] based virtual machine with the RedHat *virt-manager* application. In order to be able to shut down the virtual machines I found it necessary to also install the Fedora *acpid* and *qemu-guest-agent* packages and enable the associated services on the host.

I created a QEMU VM from the Fedora 36 Server net install ISO image with 4GB RAM, 10GB disk, 8 Nehalem CPUs, bridged networking on device `virbr0`, root password `password` and SSH root access with password. Following *Buettner*'s advice I removed the tablet, sound, serial port 1, channel qemu-ga, channel spice and USB redirector 1 and 2. In the following the IPv4 address will be determined by your network settings and the VM DHCP lease. The host firewall is set to allow trusted access by the *ssh* service. Recover the pointer from the *virt-manager* terminal by depressing the `left-Ctrl` and `left-Alt` keys.

If necessary, edit the *user* and *group* in `/etc/libvirt/qemu.conf` and add the user to the *qemu*, *kvm* and *libvirt* groups:

```
sudo usermod -aG qemu ${USER}
sudo usermod -aG kvm ${USER}
sudo usermod -aG libvirt ${USER}
```

`virsh` requires that *LIBVIRT_DEFAULT_URI* is set in the shell environment:

```
export LIBVIRT_DEFAULT_URI=qemu:///system
```

Log out and log back in to effect these changes.

### Set up the VM

I did not attempt to set up a shared directory or file server. I communicate with the VM through `ssh` and transfer files with `scp`. A vertical ellipsis indicates discarded output.

```
$ virsh start Fedora36Server
Domain 'Fedora36Server' started

$ virsh net-list
Name      State    Autostart   Persistent
--------------------------------------------
 default   active   yes         yes

$ virsh net-dhcp-leases default
.
.
$ ssh root@192.168.122.100
.
.
[root@localhost ~]# uname -rs
Linux 5.19.10-200.fc36.x86_64
[root@localhost ~] dnf install wget readline-devel lzip sharutils gcc gcc-c++ \
gcc-gfortran gmp-devel mpfr-devel make cmake gnuplot-latex m4 gperf bison flex \
openblas-devel patch texinfo texinfo-tex icoutils librsvg2-tools librsvg2 \
dia epstool autoconf automake libtool pcre pcre-devel freetype freetype-devel \
dia epstool texlive-algorithmicx texlive-appendix texlive-boondox \
```

```
texlive-calculator texlive-chngcntr texlive-dvipng texlive-environ \
texlive-epstopdf texlive-esint texlive-esint-type1 texlive-fontaxes \
texlive-fouriernc texlive-fourier texlive-framed texlive-gsftopk \
texlive-kpfonts texlive-latex-base-dev texlive-latex-bin-dev \
texlive-latex-graphics-dev texlive-ly1 texlive-mathdesign texlive-multirow \
texlive-nag texlive-needspace texlive-newpx texlive-newtx \
texlive-pdfcrop texlive-powerdot texlive-pst-blur texlive-pst-pdf \
texlive-pst-slpe texlive-rotfloat texlive-scheme-basic \
texlive-threeparttable texlive-tocbibind texlive-trimspaces \
texlive-type1cm texlive-upquote texlive-wrapfig texlive-dvisvgm \
hdf5 hdf5-devel qt qscintilla-qt5 qscintilla-qt5-devel \
qhull qhull-devel portaudio portaudio-devel libsndfile libsndfile-devel \
GraphicsMagick-c++ GraphicsMagick-c++-devel libcurl libcurl-devel \
gl2ps gl2ps-devel fontconfig-devel mesa-libGLU mesa-libGLU-devel \
qt5-qttools qt5-qttools-common qt5-qttools-devel rapidjson-devel python3-sympy
.
.
[root@localhost ~]# wget https://github.com/robertgj/DesignOfIIRFilters/archive/master.zip
[root@localhost ~]# unzip master.zip
[root@localhost ~]# mkdir -p /usr/local/src/octave
[root@localhost ~]# cp DesignOfIIRFilters-master/build-octave.sh /usr/local/src/octave
[root@localhost ~]# cd /usr/local/src/octave
[root@localhost octave]# wget https://ftp.gnu.org/gnu/gnu-keyring.gpg
.
.
[root@localhost octave]# gpg2 --import gnu-keyring.gpg
.
.
[root@localhost octave]# sh ./build-octave.sh
.
.
[root@localhost ~]# cd
[root@localhost ~]# echo "export PATH=$PATH:/usr/local/octave-7.2.0/bin" >> .bashrc
[root@localhost ~]# shutdown -h now
Connection to 192.168.122.100 closed by remote host.
Connection to 192.168.122.100 closed.
$
```

This edited example session shows:

- starting the VM with *virsh*
- logging into the VM with *ssh*
- system update with *dnf*
- installation of the Fedora packages required to build this document
- download of the source for this document from *GitHub*
- building a local version of Octave with the *build-octave.sh* script

**Build this document on the VM**

Octave will only use the *qt* graphics toolkit if it finds an X display. Otherwise it uses the *gnuplot* graphics toolkit. Use *virt-manager* to set the display to *Type* 'VNC server', *Listen type* 'Address' and *Address* 'All interfaces'. The *-Y* option to `ssh` enables trusted X11 forwarding.

```
$ virsh start Fedora36Server
$ ssh -Y root@192.168.122.100
[root@localhost ~]# dnf install xorg-x11-xauth xorg-x11-xinit \
xorg-x11-drv-qxl xorg-x11-xinit-session xrdb xmodmap
[root@localhost ~]# echo "X11Forwarding yes" >>/etc/ssh/sshd_config
[root@localhost ~]# echo "X11UseLocalhost no" >>/etc/ssh/sshd_config
[root@localhost ~]# echo "AddressFamily inet" >>/etc/ssh/sshd_config
```

```
[root@localhost ~]# systemctl restart sshd.service
[root@localhost ~]# octave --no-gui --eval "available_graphics_toolkits"
ans =
{
  [1,1] = gnuplot
  [1,2] = qt
}
[root@localhost ~]# cd DesignOfIIRFilters-master
[root@localhost DesignOfIIRFilters-master]# make -k -O -j 8
```

### Copy a file to the VM

```
$ virsh start Fedora36Server
$ scp Makefile scp://root@192.168.122.100//root/Makefile
```

### Copy this document from the VM

```
$ virsh start Fedora36Server
$ scp root@192.168.122.100:/root/DesignOfIIRFilters-master/DesignOfIIRFilters.pdf \
  DesignOfIIRFilters.pdf
```

### Run the test scripts on the VM

```
$ virsh start Fedora36Server
$ ssh -Y root@192.168.122.100
[root@localhost ~]# cd DesignOfIIRFilters-master
[root@localhost DesignOfIIRFilters-master]# make batchtest
.
.
.
```

### Export the VM to libvirt

```
$ virsh start Fedora36Server
$ virsh dumpxml Fedora36Server
```

### Remove an existing VM

```
$ sudo virsh shutdown Fedora36Server
$ sudo virsh undefine Fedora36Server
```

### Zero the disk image before compressing it

Shut down the VM and, on the host, run[i]:

```
$ sudo virsh shutdown Fedora36Server
$ sudo virt-sparsify -v -x --in-place your-image-file
```

Alternatively, fill unused VM disc sectors with 0:

[i] First install the Fedora *guestfs-tools* package.

```
$ virsh start Fedora36Server
$ ssh -Y root@192.168.122.100
[root@localhost ~]# cd /boot
[root@localhost ~]# dd if=/dev/zero of=zerofile
[root@localhost ~]# sync
[root@localhost ~]# rm -f zerofile
[root@localhost ~]# cd /
[root@localhost ~]# dd if=/dev/zero of=zerofile
[root@localhost ~]# sync
[root@localhost ~]# rm -f zerofile
[root@localhost ~]# shutdown -h now
```

## Testing

The *test* directory contains regression test shell scripts. The test scripts must be run from the project root directory. The shell script *batchtest.sh* runs multiple tests in parallel. Alternatively, run `make batchtest`.

## Aegis

I use the *aegis* software change management system written by the late Peter Miller [159]. To build *aegis* with the patch provided in this project:

```
tar -xf aegis-4.24.tar.gz
cd aegis-4.24
patch -p1 < ../aegis-4.24.patch
CXXFLAGS=-O2 ./configure
make && make install
```

Replacing *Makefile.in* with *Makefile.in.solib* builds binaries linked to a shared library. The *configure* script may prompt you to install the *groff*, *file-devel*, *uuid-devel* etc. packages. The */usr/local/com/aegis/state* file lists existing Aegis projects. The */usr/local/com/aegis/user* directory contains files listing the per-user state of existing Aegis projects. I use this */.aegisrc*:

```
default_development_directory = "CHANGES";
default_project_directory = "AEGIS";
```

and add to */.bashrc*:

```
if [ -f /usr/local/etc/profile.d/aegis.sh ]; then
  . /usr/local/etc/profile.d/aegis.sh
fi
```

*aegis.conf* configures *aegis* to use the file comparison functions from Miller's *fhist* [160] project. *fhist* depends on the *libexplain*[161] library. *fhist* and *libexplain* are compiled similarly to *aegis* with the patches provided in this project.

## Monochrome printing

Monochrome printing of this document is supported by:

- hiding coloured hyperlinks in the *printed* PDF document by compiling this document with:

  ```
  pdflatex '\newcommand\DesignOfIIRFiltersMono{}\input{DesignOfIIRFilters}'
  ```

  The *DesignOfIIRFiltersMono* flag enables the following LaTeXcode:

```
\usepackage[hidelinks]{hyperref}
```

- setting the Octave default line palette to all black in *src/test_common.m*:

```
if getenv("OCTAVE_ENABLE_MONOCHROME")
  set(0,"defaultaxescolororder",zeros(size(get(0,"defaultaxescolororder"))));
endif
```

The following command enables both these changes:

```
make cleanall && OCTAVE_ENABLE_MONOCHROME=1 make -j 6 monochrome
```

# 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. 96, 98, 599, 758

[2] A. Antoniou and W.-S. Lu. *Practical Optimization: Algorithms and Engineering Applications*. Springer Science+Business Media, LLC, 2007. ISBN 0-387-71106-6. 221, 519, 788

[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. 18, 19, 149, 159, 168

[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. 17, 18, 19, 74, 101

[5] A. H. Land and A. G. Doig. An Automatic Method Of Solving Discrete Programming Problems. *Econometrica*, 28(3):497–520, July 1960. 392

[6] A. H. Sayed and T. Kailath. A Survey of Spectral Factorization Methods. *Numerical Linear Algebra With Applications*, 08:467–496, 2001. 648

[7] A. Krukowski and I. Kale. Two Approaches for fixed-point filter design: "bit-flipping" algorithm and constrained downhill Simplex method. In *Proceedings of the Fifth International Symposium on Signal Processing and its Applications*, volume 2, pages 965–968, 1999. 19, 378, 379

[8] A. Nishihara and K. Sugahara. A Synthesis of Digital Filters with Minimum Pole Sensitivity. *The Transactions of the IECE of Japan*, E65(5):234–240, May 1982. 73

[9] A. P. Ruszczynzki. *Nonlinear Optimization*. Princeton University Press, 2006. ISBN 0-691-11915-5. 19, 569, 571, 577, 579

[10] 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. 18, 127, 129, 149, 150, 151, 153, 156

[11] A collection of Fortran77 subroutines designed to solve large scale eigenvalue problems. https://github.com/opencollab/arpack-ng. 804

[12] Athanasios Papoulis. *Signal Analysis*. McGraw-Hill International, 1981. ISBN 0-07-066468-4. 190

[13] Automatically Tuned Linear Algebra Software. http://math-atlas.sourceforge.net/. 808

[14] Automatic Differentiation. http://www.autodiff.org. 549

[15] B. C. Carlson. Computing Elliptic Integrals by Duplication. *Numerische Mathematik*, 33:1–16, 1979. 17, 534, 535, 536

[16] B. C. Carlson. POWER SERIES FOR INVERSE JACOBIAN ELLIPTIC FUNCTIONS. *MATHEMATICS OF COMPUTATION*, 77(263):1615–1621, July 2008. 538

[17] B. Christianson. Global Convergence using De-linked Goldstein or Wolfe Linesearch Conditions. *AMO - Advanced Modeling and Optimization*, 1(1):25–31, 2009. 581

[18] B. Després. Positive polynomials and numerical approximation. https://www.math.univ-paris13.fr/~marchal/exposedespres.pdf, 2020. 725

[19] B. Dumitrescu. Bounded Real Lemma for FIR MIMO Systems. *IEEE Signal Processing Letters*, 12(7):496–499, July 2005. 639

[20] B. Dumitrescu and R. Niemistö. Multistage IIR Filter Design Using Convex Stability Domains Defined by Positive Realness. *IEEE Transactions on Signal Processing*, 52(4):962–974, April 2004. 145, 149

[21] B. Dumitrescu, I. Tăbuş and P. Stoica. On the Parameterization of Positive Real Sequences and MA Parameter Estimation. *IEEE Transactions on Signal Processing*, 49(11):2630–2639, November 2001. 639, 645, 646, 647, 648

[22] B. Kumar and S. C. D. Roy. COEFFICIENTS OF MAXIMALLY LINEAR, FIR DIGITAL DIFFERENTIATORS FOR LOW FREQUENCIES. *Electronics Letters*, 24(9):563–565, 28th April 1988. 720

[23] B. Kumar, S. C. D. Roy and H. Shah. On the Design of FIR Digital Differentiators which Are Maximally Linear at the Frequency $\pi/p$, $p \in \{PositiveIntegers\}$. *IEEE Transactions on Signal Processing*, 40(9):2334–2338, September 1992. 720, 723, 724

[24] 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. 17, 65, 67

[25] 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. 65

[26] Boost Library Documentation. Jacobi Zeta Function. https://www.boost.org/doc/libs/1_79_0/libs/math/doc/html/math_toolkit/ellint/jacobi_zeta.html. 540

[27] C. Gumacos. Weighting Coefficients for Certain Maximally Flat Nonrecursive Digital Filters. *IEEE Transactions on Circuits and Systems*, 25(4):234–235, April 1978. 716, 717

[28] 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. 49, 50, 64, 112, 124

[29] 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. 61, 63, 112

[30] 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. 139

[31] Center for Discrete Mathematics and Theoretical Computer Science. The DIMACS library of mixed semi-definite quadratic linear programs. http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/FILTER/minphase.mat.gz. 639

[32] D. A. Bini, G. Fiorentino and L. Robol. Multiprecision Polynomial Solver. https://github.com/robol/MPSolve. 19, 813

[33] D. Goldfarb and A. Idnani. A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs. *Mathematical Programming*, 27:1–33, 1983. 17, 153, 582, 583, 586

[34] D. Henrion, J. B. Lasserre and Johan Löfberg. GloptiPoly 3 - moments, optimization and semidefinite programming. https://homepages.laas.fr/henrion/software/gloptipoly3/. 812

[35] D. Henrion, J. B. Lasserre and Johan Löfberg. GloptiPoly 3 - moments, optimization and semidefinite programming. https://arxiv.org/pdf/0709.2559.pdf. 812

[36] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999. ISBN 1-886529-00-0. 19, 570, 571, 573, 576, 577, 578, 579, 580

[37] D. Rabrenović and M. Lutovac. Minimum Stopband Attenuation of Cauer Filters without Elliptic Functions and Integrals. *IEEE Transactions on Circuits and Systems-I:Fundamental Theory and Applications*, 40(9):618–621, September 1993. 607, 608

[38] D. Rabrenović and M. Lutovac. Elliptic filters with minimal Q-factors. *Electronics Letters*, 30(3):206–207, 3rd February 1994. 607, 608

[39] C.-S. Yang D. Shiung, Y.-Y. Yang. Improving FIR Filters by Using Cascade Techniques. *IEEE Signal Processing Magazine*, pages 108–114, May 2016. 758, 772

[40] 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. 17, 139, 140, 141, 142

[41] E. M. Hofstetter. A New Technique for the Design of Non-Recursive Digital Filters, December 1970. Massachusetts Institute of Technology, Lincoln Laboratory, Technical Note 1970-42. 671, 729

[42] E. T. Whittaker and G. N. Watson, editor. *A Course of Modern Analysis*. Cambridge University Press, 4th edition, 1927. ISBN 978-0-521-58807-2. 533, 746

[43] Erwin Kreyszig. *Advanced Engineering Mathematics*. John Wiley and Sons, Fifth edition, 1983. ISBN 0-471-88941-5. 513, 516

[44] NICONET e.V. The Control and Systems Library SLICOT. http://www.slicot.org. 56

[45] F. Alizadeh and D. Goldfarb. Second-Order Cone Programming. *Mathematical Programming*, 95:3–51, 2001. 19, 213, 221

[46] 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, editors. NIST Digital Library of Mathematical Functions. http://dlmf.nist.gov/. 533, 534, 536, 537, 538, 539, 540, 591, 745, 750

[47] Redhat Fedora Linux distribution. https://getfedora.org/. 803

[48] A fast, free C FFT library. http://www.fftw.org. 804

[49] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996. ISBN 0-8018-5413-8. 36, 38, 136, 137, 151, 519, 520, 542, 543, 574, 575, 581

[50] G. Pipeleers and L. Vandenberghe. Generalized KYP Lemma with Real Data. *IEEE Transactions on Automatic Control*, 56(12):2942–2946, December 2011. 783, 788

[51] G. Pipeleers, T. Iwasaki and S. Hara. GENERALIZING THE KYP LEMMA TO MULTIPLE FREQUENCY INTERVALS. *SIAM Journal on Control and Optimization*, 52(6):3618–3638, 2014. 731, 783, 796, 797, 798, 800

[52] G. T. Cargo and O. Shisha. The Bernstein Form of a Polynomial. *JOURNAL OF RESEARCH of the Notional Bureau of Standards-B. Mathematics and Mathematical Physics*, 70B(1):79–81, January-March 1966. 707

[53] Jean Gallier. The Schur Complement and Symmetric Positive Semidefinite (and Definite) Matrices. https://www.cis.upenn.edu/~jean/schur-comp.pdf, August 2019. 519, 521

[54] 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. 212, 636, 637, 643

[55] gnome.org. Dia diagram editor. https://wiki.gnome.org/Apps/Dia/. 804

[56] David Goldberg. What Every Computer Scientist Should Know About Floating-point Arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991. 19

[57] G.Stoyanov, Zl. Nikolova, K. Ivanova, V. Anzova. Design and Realization of Efficient IIR Digital Filter Structures Based on Sensitivity Minimizations. In *TELSIKS 2007*, pages 299–308, September 2007. 74, 76

[58] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. 637

[59] 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. 139

[60] H. D. Tuan, N. T. Hoang, H. Q. Ngo, H. Tuy and B. Vo. A dual frequency-selective bounded real lemma and its applications to IIR filter design. In *Proceedings of the IEEE Conference on Decision and Control*, January 2007. 726

[61] H. D. Tuan, T. T. Son, B. Vo and T. Q. Nguyen. Efficient Large-Scale Filter/Filterbank Design via LMI Characterization of Trigonometric Curves. *IEEE Transactions on Signal Processing*, 55(9):4393–4404, September 2007. 725, 726, 727, 728, 729, 730, 731, 732

[62] H. J. Orchard and Alan N. Willson. Elliptic Functions for Filter Design. *IEEE Transactions on Circuits and Systems-I:Fundamental Theory and Applications*, 44(4):273–287, April 1997. 607, 608, 610

[63] 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. 212, 636, 642

[64] 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. 19, 318

[65] H. Johansson and T. Saramäki. A Class of Complementary IIR Filters. In *Proceedings of the IEEE International Symposium on Circuits and Systems VLSI*, May 1999. 625, 626

[66] H. Riblet. The Application of a New Class of Equal-Ripple Functions to Some Familiar Transmission-Line Problems. *IEEE Transactions on Microwave Theory and Techniques*, 12:415–421, July 1964. 764

[67] 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. 457

[68] 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. 457, 811

[69] 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. 272

[70] I. Pólik and T. Terlaky. A Survey of the S-Lemma. *SIAM Review*, 49(3):371–418, 2007. 525

[71] I. R. Khan, M. Okuda and R. Ohba. New design of FIR digital differentiators having maximally linearity at middle of the frequency band. In *International Symposium on Communications and Information Technologies 2004*, pages 178–183, October 2004. 720, 723

[72] I. W. Selesnick. Maximally Flat Low-Pass Digital Differentiators. *IEEE Transactions on Circuits and Systems-II:Analog and Digital Signal Processing*, 49(3):219–223, March 2002. 720, 721, 722

[73] I. W. Selesnick and C. S. Burrus. Exchange Algorithms for the Design of Linear Phase FIR Filters and Differentiators Having Flat Monotonic Passbands and Equiripple Stopbands. *IEEE Transactions on Circuits and Systems-II:Analog and Digital Signal Processing*, 43(9):671–675, September 1996. 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705

[74] I. W. Selesnick and C. S. Burrus. Exchange Algorithms that Complement the Parks-McClellan Algorithm for Linear-Phase FIR Filter Design. *IEEE Transactions on Circuits and Systems-II:Analog and Digital Signal Processing*, 44(2):137–143, February 1997. 676, 681, 693, 694, 729

[75] 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. 147, 148, 654, 665, 732, 733, 734

[76] 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. 17, 18, 19, 148, 154, 655

[77] J. G. VanAntwerp and R. D. Braatz. A tutorial on linear and bilinear matrix inequalities. *Journal of Process Control*, 10:363–385, 2000. 526

[78] J. H. McClellan, T. W. Parks and L. R. Rabiner. A Computer Program for Designing Optimum FIR Linear Phase Digital Filters. *IEEE Transactions on Audio and Electroacoustics*, 21(6):506–526, December 1973. 17, 684, 685

[79] 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. 19

[80] J. H. Wilkinson, editor. *The Algebraic Eigenvalue Problem*. Oxford University Press, Inc., 1988. ISBN 0-198-53418-3. 38

[81] 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. 148, 159

[82] J. Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004. 789, 792, 812

[83] J.-P. Berrut and L. N. Trefethen. Barycentric Lagrange Interpolation. *SIAM Review*, 46(3):501–517, 2004. 545, 671

[84] J, Szczupak, S. K. Mitra and J. Fadavi-Ardekani. A Computer-Based Synthesis Method of Structurally LBR Digital AllPass Networks. *IEEE Transactions on Circuits and Systems*, 35(6):755–760, June 1988. 73

[85] J. Tuqan and P. P. Vaidyanathan. A State Space Approach to the Design of Globally Optimal FIR Energy Compaction Filters. *IEEE Transactions on Signal Processing*, 48(10):2822–2838, October 2000. 639, 640, 643, 644, 648

[86] J. W. Adams and J. L. Sullivan. Peak Constrained Least-Squares Optimization. *IEEE Transactions on Signal Processing*, 46(2):306–321, February 1998. 154

[87] Jan Purczyński and Cezary Pawelczak. Maximally Linear FIR Digital Differentiators in Frequencies of $\pi/p$-Simplified Formulas of Weighting Coefficients. *IEEE Transactions on Signal Processing*, 50(4):978–981, April 2002. 720, 723, 724

[88] Johan Löfberg. Complex-valued problems. https://yalmip.github.io/inside/complexproblems/. 811

[89] Johan Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. https://yalmip.github.io/. 789, 792, 812

[90] John Cristy and Bob Friesenhahn. GraphicsMagick Image Processing System. http://www.graphicsmagick.org. 804

[91] 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. 18, 48

[92] Jorge Nocedal and Stephen J.Wright. *Numerical Optimization*. Springer, second edition, 2006. ISBN 0-387-30303-0. 19, 569, 570, 573

[93] K. C. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3 — a Matlab software package for semidefinite programming. *Optimization Methods and Software*, 11:545–581, 1999. 812

[94] K. C. Toh, R. H. Tütüncü, and M. J. Todd. SDPT3 - a MATLAB software package for semidefinite-quadratic-linear programming. http://www.math.cmu.edu/~reha/sdpt3.html. 812

[95] K. Ivanova and G.Stoyanov. A New Low-sensitivity Second-order Allpass Section Suitable for Fractional Delay Filter Realizations. In *TELSIKS 2007*, pages 317–320, September 2007. 76

[96] K. Surma-aho and T. Saramäki. A Systematic Technique fo Designing Approximately Linear Phase Recursive Digital Filters. *IEEE Transactions on Circuits and Systems-II:Analog and Digital Signal Processing*, 46(7):956–963, July 1999. 17, 151, 614, 619

[97] Kai Hwang. *Computer Arithmetic : Principles, Architecture and Design*. John Wiley and Sons, 1979. ISBN 0-471-03496-7. 362

[98] Keshab K. Parhi. *VLSI Digital Signal Processing Systems : Design and Implementation*. Wiley Inter-Science, 1999. ISBN 0-471-24186-5. 17, 18, 19, 74, 95, 96, 97, 98, 99, 100, 102, 103, 104, 105, 106, 113, 119, 125, 137, 362, 363

[99] Kevin Buettner. Configure and run a QEMU-based VM outside of libvirt with virt-manager. https:developers.redhat.com/author/kevin-buettner. 814

[100] 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. 581

[101] L. Fousse et al. MPFR: A multiple-precision binary floating-point library with correct rounding. http://www.mpfr.org. 96, 599

[102] L. M. Smith. Decomposition of FIR Digital Filters for Realization Via the Cascade Connection of Subfilters. *IEEE Transactions on Signal Processing*, 46(6):1681–1684, June 1998. 758, 772

[103] L. M. Smith and M. E. Henderson Jr. Roundoff Noise Reduction in Cascade Realizations of FIR Digital Filters. *IEEE Transactions on Signal Processing*, 48(4):1196–1200, April 2000. 758

[104] L. Milić and J. Ćertić. Two-Channel IIR Filter Banks Utilizing the Frequency-Response Masking Technique. *Telfor Journal*, 1(2):45–48, 2009. 19

[105] L. Milić and M. Lutovac. Design of Multiplierless Elliptic IIR Filters with a Small Quantization Error. *IEEE Transactions on Signal Processing*, 47(2):469–479, February 1999. 607

[106] 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. 354

[107] L. R. Rajagopal and S. C. D. Roy. Design of Maximally-Flat FIR Filters Using the Bernstein Polynomial. *IEEE Transactions on Circuits and Systems*, 34(12):1587–1590, December 1987. 707, 708, 714

[108] L. Theile. On the Sensitivity of Linear State-Space Systems. *IEEE Transactions on Circuits and Systems*, 33(5):502–510, May 1986. 39

[109] R. Wallin L. Vandenberghe, V. Balakrishnan and A. Hansson. On the implementation of primal-dual interior-point methods for semidefinite programming problems derived from the KYP lemma. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 4658–4663, December 2003. 783

[110] LAPACK-Linear Algebra PACKage. http://www.netlib.org/lapack/. 804

[111] J. B. Lasserre. Global Optimization with Polynomials and the Problem of Moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001. 457

[112] J. B. Lasserre. A semidefinite programming approach to the generalized problem of moments. *Mathematical Programming*, 112:65–92, 2008. 812

[113] J. B. Lasserre. *An Introduction to Polynomial and Semi-Algebraic Optimization*. Cambridge University Press, 2015. ISBN 978-1-107-63069-7. 812

[114] Lawrence Livermore National Laboratory. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers. https://computing.llnl.gov/projects/sundials. 804

[115] 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. 457

[116] 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. 18, 19, 149, 549

[117] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions With Formulas, Graphs and Mathematical Tables*. Number 55 in Applied Mathematics Series. National Bureau of Standards, 1972. 533

[118] 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. 149

[119] M. J. D. Powell. A Fast Algorithm for Nonlinearly Constrained Optimization Calculations. *Lecture Notes in Mathematics*, 630:144–157, 1978. 569, 573, 577

[120] M. J. Rutter. Linpack benchmark in Octave/Matlab. http://www.tcm.phy.cam.ac.uk/~mjr/linpack/linpack.m, November 2015. 808

[121] M. Lutovac and L. Milić. Design Of Computationally Efficient Elliptic IIR Filters with a Reduced Number of Shift-and-Add Operations in Multipliers. *IEEE Transactions on Signal Processing*, 45(10):2422–2430, October 1997. 607, 608, 609, 610

[122] M. Murumatsu and T. Suzuki. A NEW SECOND-ORDE CONE PROGRAMMING RELAXATION FOR MAX-CUT PROBLEMS. *Journal of the Operations Research Society of Japan*, 46(2):164–177, 2003. 466

[123] 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. 19, 279

[124] 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. 19, 279

[125] M. S. Lobo, L. Vandenberghe, S. Boyd and H. Lebret. Applications of second-order cone programming. Linear Algebra and its Applications, 1998. 19

[126] M. Vlček and R. Unbehauen. Analytical Solutions for Design of IIR Equiripple Filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(10):1518–1531, October 1989. 533, 736, 747

[127] M. Vlček and R. Unbehauen. Zolotarev Polynomials and Optimal FIR Filters. *IEEE Transactions on Signal Processing*, 47(3):717–730, March 1999. 17, 745, 746, 747, 748, 749, 750, 751, 753, 754, 755, 756, 770

[128] M. Vlček and R. Unbehauen. Corrections to "Zolotarev Polynomials and Optimal FIR Filters". *IEEE Transactions on Signal Processing*, 48(7):2171, July 2000. 17, 747, 748, 749

[129] M. Vlček, P. Zahradník and R. Unbehauen. Analytical Design of FIR Filters. *IEEE Transactions on Signal Processing*, 48(9):2705–2709, September 2000. 17, 713, 714, 715

[130] Michael Grant. SDPT3 version 4.0: MATLAB/Octave software for semidefinite-quadratic-linear programming. https://github.com/sqlp/sdpt3. 812

[131] 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. 580

[132] M.Vlček and P. Zahradník. Almost Equiripple Low-pass FIR Filters. *Circuits Syst Signal Processing*, 32:743–757, 2013. DOI 10.1007/s00034-012-9484-0. 17, 756, 757, 764, 765, 766, 767, 768, 769, 770, 771

[133] M.Vlček and P. Zahradník. Approximation of Almost Equiripple Low-pass FIR Filters. In *2013 European Conference on Circuit Theory and Design*, September 2013. DOI: 10.1109/ECCTD.2013.6662301. 747, 764

[134] N. J. Higham. The numerical stability of barycentric Lagrange interpolation. *IMA Journal of Numerical Analysis*, 24:547–556, 2004. 546

[135] O. Herrmann. On the Approximation Problem in Nonrecursive Digital Filter Design. *IEEE Transactions on Circuits and Systems*, pages 411–413, May 1971. 707, 708, 709, 714

[136] Octave Forge. Computer-Aided Control System Design. https://octave.sourceforge.io/control/. 643

[137] Octave Forge. Non-linear optimization toolbox. https://octave.sourceforge.io/optim/. 19, 491, 497, 504

[138] Octave Forge. Non-linear optimization toolbox nonlin_min function. https://octave.sourceforge.io/optim/function/nonlin_min.html. 497

[139] Octave Forge. Parallel execution package. https://octave.sourceforge.io/parallel/. 558

[140] Octave Forge. Signal processing toolbox. https://octave.sourceforge.io/signal/. 68

[141] Octave Forge. Symbolic calculation toolbox. https://octave.sourceforge.io/symbolic/. 549, 811

[142] OpenBlas: An optimised BLAS library. http://www.openblas.net/. 808

[143] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, 2nd edition, 1998. ISBN 0-13-754920-2. 17, 660, 670, 684, 685, 721

[144] P. A. Regalia. Stable and Efficient Lattice Algorithms for Adaptive IIR Filtering. *IEEE Transactions on Signal Processing*, 40(2):375–388, February 1992. 285

[145] 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. 19

[146] P. Morse and H. Feshbach. *Methods Of Theoretical Physics Part I*. McGraw-Hill Book Company, 1953. Library of Congress Catalog Card Number: 52-11515. 770

[147] P. P. Vaidyanathan. Optimal Design of Linear-Phase FIR Digital Filters with Very Flat Passbands and Equiripple Stopbands. *IEEE Transactions on Circuits and Systems*, 32(9):904–917, September 1985. 695

[148] 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. 633, 634, 635

[149] 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. 119, 133, 596, 619

[150] P. P. Vaidyanathan and S. K. Mitra. A Unified Structural Interpretation of Some Well-Known Stability-Test Procedures for Linear Systems. *Proceedings of the IEEE*, 75(4):478–497, April 1987. 285

[151] P. P. Vaidyanathan and Truong Q. Nguyen. A "TRICK" for the Design of FIR Half-Band Filters. *IEEE Transactions on Circuits and Systems*, 34(3):297–300, March 1987. 735

[152] 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. 19, 72, 119, 133, 245, 596, 598

[153] P. Stoica, T. McKelvey and J. Mari. MA Estimation in Polynomial Time. *IEEE Transactions on Signal Processing*, 48(7):1999–2012, July 2000. 646

[154] P. Zahradník. Equiripple Approximation of Low-Pass FIR Filters. *IEEE Transactions on Circuits and Systems-II:Express Briefs*, 65(4):526–530, April 2018. 745

[155] P. Zahradník and M. Vlček. Equiripple Approximation of Half-Band FIR Filters. *IEEE Transactions on Circuits and Systems-II:Express Briefs*, 56(12):941–945, December 2009. 17, 737, 738, 739, 740, 741

[156] P. Zahradník, M. Šusta, and B. Šimak. Degree of Equiripple Narrow Bandpass FIR Filter. *IEEE Transactions on Circuits and Systems-II:Express Briefs*, 62(8):771–775, August 2015. 745, 753, 754

[157] P. Zahradník, M. Šusta, B. Šimak and M. Vlček. Cascade Structure of Narrow Equiripple Bandpass FIR Filters. *IEEE Transactions on Circuits and Systems-II:Express Briefs*, 64(4):407–411, April 2017. 17, 758, 759, 760, 761

[158] P. Zahradník, M.Vlček and R. Unbehauen. Almost Equiripple FIR Half-Band Filters. *IEEE Transactions on Circuits and Systems-I:Fundamental Theory and Applications*, 46(6):744–748, June 1999. 736, 737

[159] Peter Miller. aegis-4.24. https://sourceforge.net/projects/aegis/files/aegis/4.24/aegis-4.24.tar.gz. 817

[160] Peter Miller. fhist-1.21.D001. http://fhist.sourceforge.net. 817

[161] Peter Miller. libexplain-1.4. http://libexplain.sourceforge.net. 817

[162] Pierre-Alexandre Bliman. Advanced tools for system analysis and design by LMI techniques. Lecture notes., 2004 November. 787

[163] GNU Project. GMP - GNU Scientific Library. https://www.gnu.org/software/gsl/. 19, 808

[164] GNU Project. GMP: The GNU Multiple Precision Arithmetic Library. http://gmplib.org. 96, 599

[165] GNU Project. GNU Linear Programming Kit. https://www.gnu.org/software/glpk/. 804

[166] QEMU : the FAST! processor emulator. https://www.qemu.org/. 814

[167] qrupdate is a Fortran library for fast updates of QR and Cholesky decompositions. https://sourceforge.net/projects/qrupdate/. 804

[168] R. A. Roberts and C. T. Mullis. *Digital Signal Processing*. Addison Wesley, 1987. ISBN 0-201-16350-0. 17, 18, 25, 34, 37, 40, 42, 44, 45, 47, 48, 55, 56, 65, 66, 67, 112, 133, 135, 136, 137, 150, 594, 670, 685

[169] R. Ansari and B. Liu. A Class of Low-Noise Computationally Efficient Recursive Digital Filters with Applications to Sampling Rate Alterations. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33(1):90–97, February 1985. 76

[170] R. E. Kalman. Lyapunov Functions for the Poblem of Lur'e in Automatic Control. *Proceedings of the National Academy of Sciences of the United States of America*, 49(2):201–205, February 1963. 783

[171] R. H. Tütüncü, K. C. Toh and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming, Ser. B*, 95(2):189–217, 2003. 812

[172] R. Levy. Generalized Rational Function Approximation in Finite Intervals Using Zolotarev Functions. *IEEE Transactions on Microwave Theory and Techniques*, 18(12):1052–1064, December 1970. 747

[173] R. Rehman and I. C. F. Ipsen. La Budde's Method for Computing Characteristic Polynomials. http://arxiv.org/pdf/1104.3769v1, April 2011. 17, 38

[174] R. Sedgwick. *Algorithms in C++*. Addison-Wesley, 1990. ISBN 0-201-51059-6. 392

[175] R. Storn and K. Price. Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. http://www1.icsi.berkeley.edu/ftp/pub/techreports/1995/tr-95-012.pdf. 504

[176] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970. ISBN 0-691-08069-0. 524

[177] Richard Lyons. Interpolated Narrowband Lowpass FIR Filters. *IEEE Signal Processing Magazine*, pages 51–57, January 2003. 179

[178] 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. 17, 18, 19, 364, 392

[179] 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. 465, 466

[180] Rizwana Rehman. Numerical Computation of the Characteristic Polynomial of a Complex Matrix. http://www.lib.ncsu.edu/resolver/1840.16/6262, 2010. 38

[181] S. Boyd and L. Vandenberghe. *Convex Opimization*. Cambridge University Press, 2008. ISBN 0-521-83378. 19, 519

[182] S. Hammarling. Numerical solution of the discrete-time, convergent, non-negative definite Lyapunov equation. *Systems and Control Letters*, 17:137–139, 1991. 56

[183] S. J. Wright. Superlinear Convergence of a Stabilized SQP Method to a Degenerate Solution. *Computational Optimization and Applications*, 11:253–275, 1998. 575

[184] S. K. Mitra and K. Hirano. Digital All-Pass Networks. *IEEE Transactions on Circuits and Systems*, 21(5):688–700, September 1974. 73, 76

[185] 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. 34

[186] 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. 18

[187] Saed Samadi and Akinori Nishihara. The World of Flatness. *IEEE Circuits and Systems Magazine*, 7(3, Third Quarter):38–44, 2007. 707

[188] I. W. Selesnick. A collection of scripts for constrained-least-squares FIR filter design. http://dsp.rice.edu/software/, Under "FIR and IIR Filter Design Algorithms", Constrained Least Square FIR Filter Design (Redirects to ConstrainedLeastSquaresAllprogs.tar.zip on web.archive.org). 148

[189] I. W. Selesnick. Exchange Algorithms Complementing the Parks-McClellan Algorithm. http://dsp.rice.edu/software, Under "FIR and IIR Filter Design Algorithms", Exchange Algorithms Complementing the Parks-McClellan Algorithm (ParkMcClellansAllPrograms.tar.zip on web.archive.org). 681, 684, 693

[190] I. W. Selesnick. Symmetric FIR Filters - Flat Passbands, Chebyshev Stopbands. http://dsp.rice.edu/software/, Under "FIR and IIR Filter Design Algorithms", Symmetric FIR Filters - Flat Passbands, Chebyshev Stopbands (fircheb.tar.zip on web.archive.org). 704

[191] 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. 124

[192] Soo-Chang Pei and Peng-Hua Wang. Closed-Form Design of Maximally Flat FIR Hilbert Transformers, Differentiators, and Fractional Delayers by Power Series Expansion. *IEEE Transactions on Circuits and Systems-I:Fundamental Theory and Applications*, 48(4):389–398, April 2001. 718, 719

[193] J. Sturm. SeDuMi_1_3 at GitHub. https://github.com/sqlp/sedumi. 18, 19, 215, 466, 811

[194] J. Sturm. SeDuMi_1_3 at LeHigh University. http://sedumi.ie.lehigh.edu/sedumi. 811

[195] J. Sturm. Using SeDuMi 1.02, A MATLAB Toolbox for Optimization over Symmetric Cones (Updated for Version 1.05). Octave SeDuMi installation at site/m/SeDuMi/doc/SeDuMi_Guide_105R5.pdf. 639, 643, 644, 811

[196] SymPy Development Team. SymPy. https://www.sympy.org/en/index.html. 811

[197] T. A. Davis et al. SuiteSparse: a suite of sparse matrix packages . http://people.engr.tamu.edu/davis/suitesparse.html. 804

[198] T. Iwasaki and S. Hara. Generalised KYP Lemma: Unified Characterization of Frequency Domain Inequalities with Applications to System Design. Technical report, DEPARTMENT OF MATHEMATICAL INFORMATICS, GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY THE UNIVERSITY OF TOKYO, August 2003. 783, 785

[199] T. Iwasaki and S. Hara. Generalised KYP Lemma: Unified Frequency Domain Inequalities With Design Applications. *IEEE Transactions on Automatic Control*, 50(1):41–59, January 2005. 783, 784, 785, 786, 789

[200] T. N. Davidson, Z.-Q. Luo and J. F. Sturm. Linear Matrix Inequality Formulation of Spectral Mask Constraints With Applications to FIR Filter Design. *IEEE Transactions on Signal Processing*, 50(11):2702–2715, November 2002. 725, 783

[201] T. Saramäki. Design of Optimum Recursive Digital Filters with Zeros on the Unit Circle. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 31(2):450–458, April 1983. 613, 614, 619

[202] T. Saramäki. On the Design of Digital Filters as a Sum of Two All-Pass Filters. *IEEE Transactions on Circuits and Systems*, 32(11):1191–1193, November 1985. 619

[203] T. Saramäki. Design of FIR Filters as a Tapped Cascaded Interconnection of Identical Subfilters. *IEEE Transactions on Circuits and Systems*, 34(9):1011–1029, September 1987. 758, 772, 773, 774, 775, 777, 778, 780

[204] 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. 348

[205] T. Söderström and P. Stoica. *System Identification*. Prentice Hall International, http://user.it.uu.se/~ts/sysidbook.pdf, 1989. ISBN 0-13-881236-5. 646

[206] T. W. Parks and J. H. McClellan. Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase. *IEEE Transactions on Circuit Theory*, 19(2):189–194, March 1972. 17, 671, 684, 685, 686, 687, 688, 690, 729

[207] T. Yoshida, Y. Sugiara and N. Aikawa. A General Expression of the Low-Pass Maximally Flat FIR Digital Differentiators. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2015. 720

[208] The MathWorks. MATLAB: The Language of Technical Computing. http://mathworks.com. 18

[209] Ulf T. Jönsson. A Lecture on the S-Procedure. https://people.kth.se/~uj/5B5746/Lecture.ps. 521, 524, 525

[210] William H. Press Saul A. Teukolsky William T. Vetterling and Brian P. Flannery. *Numerical Recipes In C : The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992. ISBN: 0-521-43108-5. 530

[211] 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. 237, 245

[212] 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. 19, 149, 214, 318, 321, 322, 329, 334

[213] 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. 139, 140

[214] 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. 457

[215] William Schelter. Maxima, a Computer Algebra System. http://maxima.sourceforge.net/. 549, 558

[216] Wolfram. Mathematica. http://functions.wolfram.com/EllipticIntegrals/JacobiZeta/02. 540

[217] 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. 215, 466

[218] Wu-Sheng Lu. Design of FIR Filters with Discrete Coefficients: A Semidefinite Programming Relaxation Approach. *Proceedings of the IEEE International Symposium on Circuits and Systems*, II:297–300, 2001. 465, 466

[219] 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. 149

[220] Xiangkun Chen and T. W. Parks. Equiripple Approximation of Low-Pass FIR Filters. *IEEE Transactions on Circuits and Systems*, 33(11):1065–1071, November 1986. 745, 746, 770

[221] Xu Zhong. On Inverses and Generalized Inverses of Hessenberg Matrices. *Linear Algebra and its Applications*, 101:167–180, 1988. 285

[222] 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. 19, 318

[223] Y. C. Lim and B. Liu. Design of Cascade Form FIR Filters with Discrete Valued Coefficients. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36(11):1735–1739, November 1988. 758, 772

[224] 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. 18, 19, 363

[225] Y. Genin, Yu. Nesterov and P. Van Dooren. Optimization over positive polynomial matrices. In *Math. Theory Network Syst. Conf., Perpignan, France*, 2000. Paper SI27-7. 648

[226] Y. Nuevo, Dong Chengu-Yu and S. K. Mitra. Interpolated Finite Impulse Response Filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 32(3):563–570, June 1984. 710

[227] 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. 388