On the design of IIR filters

Robert G. Jenssen

September 16, 2025

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

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

Copyright © 2017–2025 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 . . . 29

1.6 Controllability . . . 30

1.7 Observability . . . 31

1.8 Coordinate Transformations . . . 31

1.9 State variable descriptions and the transfer function . . . 31

1.9.1 Transformation of a transfer function to a state variable description . . . 32

1.9.2 Transformation of a transfer function to a state variable description by continued fraction expansion . . . 33

1.9.3 Transformation of a state variable description to a transfer function . . . 35

1.9.4 Sensitivity of the state variable description of a transfer function . . . 38

1.10 Time domain description . . . 38

1.11 Unit Pulse Response . . . 38

1.12 Factored state variable descriptions . . . 39

1.12.1 Factored state variable filters with fractional delays . . . 39

1.12.2 Construction of the factored state variable description . . . 39

1.13 Block processing and decimation filters . . . 43

2 Frequency transformations of Digital Filters **46**

2.1 Frequency Transformation of the Transfer Function . . . 47

2.2 Frequency Transformations of State Variable Filters . . . 48

2.3 An example: frequency transformations of a 5-th order elliptic filter . . . 50

3 Round-off noise in state variable filters **53**

3.1 Quantisation noise in digital filters . . . 53

3.2 Limit-cycle oscillations in digital filters . . . 54

3.3 State variable filters and wide sense stationary inputs . . . 54

3.3.1 The filter state covariance matrix . . . 54

3.3.2 The output response to white noise in a state variable . . . 55

3.3.3 Scaling State Variable Filters To Avoid Overflow . . . 56

3.4 Estimation of output round-off noise in state variable filters . . . 57

3.4.1 Rounding-to-minus-infinity quantisation noise . . . 59

3.5 Minimisation of round-off noise in the calculation of the state vector . . . 59

3.6 Coefficient sensitivity . . . 61

3.7 Factored state variable filters and wide sense stationary inputs . . . 61

3.8 Frequency transformations and round-off noise . . . 62

4 State variable filter realisation as a cascade of second order sections **63**

4.1 Second Order State Variable Filters Optimised for Overflow and Round-Off Noise . . . 63

4.2 Design equations for optimised second order state variable filters . . . 63

4.3 Block optimal second order cascade filter realisations . . . 66

4.4 An example of a second-order state-variable cascade filter . . . 66

4.4.1 Comparison of calculated noise gains . . . 66

4.4.2 Simulation results . . . 67

4.4.3 Comparison with an N=10 example . . . 69

4.5 Coefficient sensitivity and round-off noise of first-order and second-order all-pass filter sections . . . 70

4.5.1 Searching for realisations of all-pass filter transfer functions . . . 71

4.5.2 Maximum phase gradient and round-off noise of some first-order all-pass filter sections . . . 72

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

5 Filter synthesis by the Schur decomposition **92**

5.1 The Schur algorithm . . . 92

5.1.1 Computation of Schur polynomials . . . 92

5.1.2 Orthonormality of Schur Polynomials . . . 93

5.1.3 Polynomial Expansion Algorithm . . . 95

5.1.4 Power calculation using the Schur algorithm . . . 95

5.2 Derivation of Digital Lattice Filters . . . 95

5.2.1 Derivation of FIR, All-Pole and All-Pass Lattice Filters . . . 96
5.3 Derivation of the One-Multiplier IIR Lattice Filter . . . 98
5.4 Derivation of the Normalised Lattice Filter . . . 100
5.5 Derivation of the Scaled Normalised Lattice Filter . . . 101
5.5.1 Example: synthesis of a 3rd order Butterworth lattice filter . . . 102
5.6 State Variable Descriptions for Schur Lattice Filters . . . 104
5.6.1 State variable description of the Schur FIR lattice filter . . . 104
5.6.2 State variable description of the one-multiplier IIR lattice filter . . . 105
5.6.3 State variable description of a pipelined one-multiplier Schur lattice filter . . . 106
5.6.4 State variable description of a doubly-pipelined one-multiplier Schur lattice filter . . . 108
5.6.5 State variable description of an all-pass doubly-pipelined one-multiplier Schur lattice filter . . . 109
5.6.6 State variable description of the scaled-normalised IIR lattice filter . . . 110
5.7 Roundoff Noise Calculation in Schur Lattice Filters . . . 113
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 . . . 120
5.7.3 Round-off noise of the pipelined one multiplier lattice filter . . . 123
5.8 Examples of pipelining Schur lattice filters . . . 125
5.8.1 Pipelining a 4th order Schur normalised-scaled lattice filter . . . 125
5.8.2 Pipelining a 6th-order Schur one-multiplier lattice filter . . . 126
5.8.3 Frequency transformations of pipelined Schur lattice filters . . . 128
5.9 Examples of the coefficient sensitivity of the responses of direct-form and Schur lattice filters . . . 131
5.10 Summary . . . 135

6 Orthogonal state variable filters 136

6.1 Definition of orthogonal state variable filters . . . 136
6.2 The Lattice Orthogonal All-Pass Filter Section . . . 137
6.3 Noise gain of orthogonal filters . . . 137
6.4 Realisation of arbitrary filters from orthogonal sub-filters . . . 138
6.4.1 An example of structural variations . . . 140

7 Feedforward and feedback of state quantisation error in state variable filters 141

7.1 Problem formulation . . . 141
7.2 Minimisation of round-off noise with $\delta = I$ and $\eta = 0$ . . . 143

II Constrained optimisation of the IIR filter frequency response **145**

8 IIR filter design using Sequential Quadratic Programming with the transfer function defined by pole and zero locations **147**

8.1 Problem statement . . . 147

8.1.1 Solution of the constrained quasi-Newton optimisation problem . . . 148

8.1.2 Choice of active constraints . . . 149

8.1.3 Linearisation of peak constraints . . . 150

8.1.4 Ensuring the stability of the IIR filter . . . 151

8.1.5 Selecting an initial filter design . . . 151

8.2 Examples of IIR filter design with SQP and constrained pole and zero locations . . . 155

8.2.1 Introductory comments on the IIR filter design examples . . . 155

8.2.2 Tarczynski et al. Example 2 . . . 158

8.2.3 Deczky's Example 3 . . . 161

8.2.4 Deczky's Example 1 . . . 170

8.2.5 Low-pass R=2 decimation filter . . . 175

8.2.6 Band-pass R=2 decimation filter . . . 186

8.3 SQP PCLS design of a low-pass R=2 filter expressed in *gain-zero-pole* format with *SQP* . . . 193

8.3.1 Hilbert transform R=2 decimation filter . . . 196

8.3.2 R=2 differentiator filter . . . 199

8.3.3 Low-pass differentiator filter . . . 203

8.3.4 Pink noise filter . . . 209

8.3.5 Minimum phase R=2 low-pass filter . . . 212

8.3.6 Non-linear phase FIR low-pass filter . . . 215

8.3.7 Minimum phase FIR bandpass filter . . . 217

9 IIR filter design using Second Order Cone Programming **221**

9.1 Second Order Cone Programming . . . 221

9.2 Design of IIR filters with SOCP . . . 221

9.3 Using the *SeDuMi* SOCP solver . . . 223

9.4 An example of SOCP design of an IIR filter expressed in *gain-zero-pole* format with the *SeDuMi* SOCP solver . 224

9.5 An example of SOCP design of an IIR filter expressed in *gain-zero-pole* format with the *SCS* SOCP solver . . . 232

9.6 SOCP design of a non-linear phase FIR low-pass filter . . . 235

9.7 Comparison of FIR and IIR low-pass filters having approximately flat pass-band group delay with symmetric FIR filters . . . 239

9.8 SOCP design of a low-pass differentiator filter . . . 241

9.9 SOCP MMSE design of a low-pass R=2 filter expressed in *gain-zero-pole* format with *SeDuMi* . . . 243

9.10 SOCP MMSE design of a bandpass R=2 filter expressed in *gain-zero-pole* format with *SeDuMi* . . . 246

9.11 SOCP PCLS design of a multi-band-pass filter expressed in *gain-zero-pole* format with *SeDuMi* . . . 250

10 IIR filter design with a pre-defined structure **253**

10.1 Design of an IIR filter composed of second-order sections . . . 253

10.1.1 Linear constraints on the stability of second-order filter sections . . . 253

10.1.2 Linear constraints on limit-cycle oscillations in second-order filter sections . . . 254

10.1.3 Design of an IIR filter composed of second order sections with *SeDuMi* . . . 255

10.1.4 Some notes on the design of an IIR filter composed of second order sections with *SeDuMi* . . . 260

10.2 Design of an IIR filter as the sum of two all-pass filters . . . 261

10.2.1 Design of an IIR filter as the sum of two all-pass filters each composed of second-order sections . . . 261

10.2.2 Design of an IIR filter as the sum of an all-pass filter composed of second-order sections and a delay . . . 269

10.2.3 Design of an IIR filter as the sum of two all-pass filters each represented in pole-zero form . . . 272

10.2.4 Design of an IIR filter as the sum of a delay and an all-pass filter represented in pole-zero form . . . 297

10.2.5 Design of an IIR filter as the polyphase decomposition into two all-pass filters each represented in pole-zero form . . . 306

10.3 Design of IIR Schur lattice filters . . . 310

10.3.1 Design of one-multiplier IIR Schur lattice filters with SQP optimisation . . . 311

10.3.2 Design of one-multiplier IIR Schur lattice filters with SOCP optimisation . . . 319

10.3.3 Design of parallel one-multiplier IIR Schur all-pass lattice filters with SOCP optimisation . . . 352

10.3.4 Design of normalised-scaled IIR Schur lattice filters with SQP optimisation . . . 373

10.3.5 Design of parallel all-pass normalised-scaled IIR Schur lattice filters with SOCP optimisation . . . 379

10.4 Design of IIR filters with a sharp transition band by frequency response masking . . . 383

10.4.1 Review of Frequency Response Masking digital filters . . . 383

10.4.2 Design of an FRM digital filter with an IIR model filter consisting of a cascade of second-order sections using SOCP . . . 386

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 . . . 392

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 . . . 398

10.4.5 Design of an FRM digital filter with an IIR model filter consisting of parallel allpass filters . . . 403

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 . . . 409

10.4.7 Design of an FRM half-band digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation . . . 413

10.4.8 Design of an FRM Hilbert digital filter with an allpass Schur lattice model filter using SOCP and PCLS optimisation . . . 419

III Design of IIR filters with integer coefficients 425

11 Signed-digit representation of filter coefficients 428

11.1 Lim's method for allocating signed-digits to filter coefficients . . . 429

11.2 Ito's method for allocating signed-digits to filter coefficients . . . 430

11.3 Signed-digit allocation of the coefficients of a Schur one-multiplier lattice filter . . . 431

11.4 Signed-digit allocation of the coefficients of a symmetric FIR band-pass filter . . . 439

12 Exhaustive search for integer and signed-digit filter coefficients 445

13 Searching for integer and signed-digit filter coefficients with the *bit-flip* algorithm 446

13.1 Bit-flip search for the signed-digit coefficients of an R=2 direct-form bandpass IIR filter . . . 448

13.2 Bit-flip search for the signed-digit coefficients of a normalised-scaled lattice bandpass R=2 IIR filter . . . 450

13.3 Bit-flip search for the signed-digit coefficients of a one-multiplier lattice bandpass R=2 IIR filter . . . 453

13.4 Bit-flip search for the signed-digit coefficients of a one-multiplier parallel-allpass lattice bandpass IIR filter . . . 457

13.5 Bitflipping search for the signed-digit coefficients of a minimum-phase bandpass FIR filter . . . 460

13.6 Bit-flip search for the signed-digit coefficients of a symmetric bandpass FIR filter . . . 462

14 *Branch-and-bound* search for signed-digit coefficients 464

14.1 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a direct-form symmetric bandpass FIR filter . 466

14.2 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a lattice band-pass R=2 IIR filter . . . 468

14.3 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a lattice band-pass R=2 IIR filter . . . 471

14.4 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a lattice band-pass IIR filter . . . 474

14.5 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a one-multiplier pipelined lattice band-pass filter . . . 477

14.6 Branch-and-bound search for the 13-bit signed-digit coefficients of a Schur lattice band-pass Hilbert R=2 IIR filter 480

14.7 Branch-and-bound search for the 16-bit 4-signed-digit coefficients of a one-multiplier pipelined lattice low-pass filter . . . 484

14.8 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a one-multiplier lattice low-pass differentiator R=2 filter . . . 486

14.9 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass one-multiplier lattice low-pass IIR filter . . . 490

14.10 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass normalised-scaled lattice low-pass IIR filter . . . 493

14.11 Branch-and-bound search for the 8-bit 3-signed-digit coefficients of a parallel all-pass lattice IIR elliptic low-pass filter . . . 496

14.12 Branch-and-bound search for the 16-bit 4-signed-digit coefficients of a parallel all-pass lattice IIR elliptic low-pass filter . . . 499

14.13 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass IIR filter 502

14.14 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass Hilbert IIR filter . . . 505

14.15 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass lattice band-pass Hilbert IIR filter . . . 509

14.16 Branch-and-bound search for the coefficients of an FRM low-pass filter implemented with 12-bits and an average of 3-signed-digits . . . 513

14.17 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of a FRM Hilbert filter . . . 517

14.18 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of a FIR Hilbert filter . . . 521

14.19 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of a FIR Hilbert band-pass filter . . . 523

15 Successive coefficient relaxation search for signed-digit filter coefficients 525

15.1 SOCP-relaxation search for the signed-digit coefficients of a direct-form symmetric bandpass FIR filter . . . 525

15.2 SOCP-relaxation search for the signed-digit coefficients of a direct-form anti-symmetric low-pass differentiator FIR filter . . . 528

15.3 SQP-relaxation search for the signed-digit coefficients of a lattice bandpass R=2 IIR filter . . . 530

15.4 SQP-relaxation search for the signed-digit coefficients of a lattice lowpass IIR filter . . . 534

15.5 SOCP-relaxation search for the signed-digit coefficients of a lattice low-pass R=2 IIR filter . . . 538

15.6 SOCP-relaxation search for the signed-digit coefficients of a lattice bandpass R=2 IIR filter . . . 540

15.7 SOCP-relaxation search for the signed-digit coefficients of a lattice bandpass Hilbert R=2 IIR filter . . . 543

15.8 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice low-pass IIR filter . . . 547

15.9 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice elliptic low-pass IIR filter . . . 550

15.10 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice band-pass IIR filter . . . 552

15.11 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice band-pass Hilbert IIR filter . . . 556

15.12 SOCP-relaxation search for the signed-digit coefficients of a pipelined one-multiplier lattice low-pass IIR filter . . . 560

15.13 SOCP-relaxation search for the signed-digit coefficients of a lattice Hilbert R=2 IIR filter . . . 562

15.14 SOCP-relaxation search for the signed-digit coefficients of a one-multiplier lattice low-pass differentiator IIR filter 565

15.15 SOCP-relaxation search for the signed-digit coefficients of a pipelined one-multiplier lattice low-pass differentiator IIR filter . . . 567

15.16 SOCP-relaxation search for the signed-digit coefficients of a one-multiplier lattice R=2 low-pass differentiator IIR filter . . . 569

15.17 SOCP relaxation search for the 12-bit, 3-signed-digit coefficients of an FRM low-pass filter . . . 573

15.18 SOCP relaxation search for the 16-bit, 3-signed-digit coefficients of an FRM low-pass filter . . . 576

15.19 SOCP-relaxation search for the signed-digit coefficients of an FRM Hilbert IIR filter with an all-pass lattice model filter . . . 580

15.20 SOCP-relaxation search for the signed-digit coefficients of a direct-form FIR Hilbert filter . . . 584

15.21 SOCP-relaxation search for the signed-digit coefficients of a lattice FIR filter . . . 586

16 Semi-definite programming search for integer and signed-digit filter coefficients **591**

16.1 The *maximum-cut* problem of graph theory . . . 591

16.2 Semi-definite programming search for integer and signed-digit filter coefficients . . . 592

16.3 SDP-relaxation search for the signed-digit coefficients of a direct-form symmetric FIR filter . . . 594

16.4 SDP-relaxation search for the signed-digit coefficients of an FIR Hilbert filter . . . 596

16.5 SDP-relaxation search for the signed-digit coefficients of an FIR Hilbert band-pass filter . . . 599

16.6 SDP-relaxation search for the signed-digit coefficients of a lattice bandpass R=2 IIR filter . . . 601

16.7 SDP-relaxation search for the signed-digit coefficients of a parallel allpass lattice elliptic low-pass IIR filter . . . 605

16.8 SDP-relaxation search for the signed-digit coefficients of a parallel allpass lattice bandpass Hilbert IIR filter . . . 608

17 POP programming search for integer and signed-digit filter coefficients **612**

17.1 POP relaxation search for the signed-digit coefficients of a one-multiplier lattice bandpass R=2 filter . . . 612

17.2 POP relaxation search for the signed-digit coefficients of a one-multiplier lattice bandpass Hilbert R=2 filter . . . 616

17.3 POP-relaxation search for the signed-digit coefficients of a one-multiplier lattice low-pass differentiator R=2 IIR filter . . . 620

18 Comparison of filter coefficient search methods **625**

18.1 Comparison of filter coefficient search methods for a 5th order elliptic filter with 6-bit integer and 2-signed-digit coefficients . . . 625

18.1.1 Searching with the bit-flip algorithm . . . 625

18.1.2 Searching with the *Nelder-Mead* simplex algorithm . . . 632

18.1.3 Searching with the *simulated annealing* algorithm . . . 638

18.1.4 Searching with the *differential evolution* algorithm . . . 645

18.1.5 Summary of the search algorithm comparison . . . 651

18.2 Comparison of filter coefficient search methods for a low-pass differentiator filter with 12-bit integer and 3-signed-digit coefficients . . . 652

18.3 Comparison of filter coefficient search methods for a band-pass Hilbert filter with 13-bit integer and 3-signed-digit coefficients . . . 659

IV Appendixes **663**

A Review of Complex Variables **665**

A.1 Complex Functions . . . 665

A.2 Limit . . . 665

A.3 The Cauchy-Riemann Equations . . . 665

A.4 Line integrals in the complex plane . . . 666

A.5 Cauchy's Integral Theorem . . . 667

A.6 Cauchy's Integral Formula . . . 667

A.7 Derivatives of an analytic function . . . 668

A.8 Laurent's Theorem . . . 668

A.9 Residues . . . 668

A.10 Cauchy's Argument Principle . . . 669

A.11 Rouché's Theorem . . . 669

B Review of selected results from linear algebra **670**

B.1 Norm of a matrix . . . 670

B.2 Trace of a matrix . . . 670

B.3 Rank, range, span and null-space of a matrix . . . 671

B.4 Matrix determinants . . . 671

B.4.1 Definitions . . . 671

B.4.2 Matrix exponential . . . 672

B.5 Positive-definite matrixes . . . 672

B.6 Schur complement . . . 673

B.6.1 Schur complement of a matrix . . . 673

B.6.2 Schur complement of a positive definite matrix . . . 674

B.7 Convex vector spaces . . . 675

B.7.1 Definitions on convex sets . . . 675

B.7.2 The separating hyperplane theorem . . . 675

B.7.3 The S-procedure . . . 676

B.7.4 The *log det A* penalty function . . . 677

C Review of Chebyshev's polynomials **678**

C.1 Recurrence relations . . . 680

C.2 Differentiation and integration of the Chebyshev polynomials . . . 680

C.3 Chebyshev differential equations . . . 681

C.4 Orthogonality of the Chebyshev polynomials . . . 681

C.5 Approximation of functions by Clenshaw's recurrence . . . 681

D Review of Legendre's elliptic integrals and Jacobi's elliptic functions **684**

D.1 Doubly periodic functions . . . 684

D.2 Legendre's elliptic integrals . . . 684

D.3 Computation of Legendre's elliptic integrals . . . 685

D.4 Jacobi's theta functions . . . 687

D.5 Computation of Jacobi's theta functions . . . 687

D.6 Jacobi's elliptic functions . . . 688

D.7 Inverses of Jacobi's elliptic functions . . . 689

D.8 Elementary identities for the elliptic integrals and elliptic functions . . . 689

D.9 Related functions . . . 690

D.10 Octave implementations . . . 691

E Review of *Lanczos* tridiagonalisation of an unsymmetric matrix **693**

F Review of Lagrange Interpolation **695**

F.1 The barycentric Lagrange polynomial . . . 695

F.2 Node distributions . . . 696

G IIR filter amplitude, phase and group-delay frequency responses **698**

G.1 IIR filter responses . . . 699

G.1.1 IIR filter amplitude response . . . 699

G.1.2 IIR filter phase response . . . 699

G.1.3 IIR filter group-delay response . . . 700

G.2 Partial derivatives of the IIR filter responses . . . 700

G.2.1 Partial derivatives of the IIR filter amplitude response . . . 700

G.2.2 Partial derivatives of the IIR filter phase response . . . 701

G.2.3 Partial derivatives of the IIR filter group-delay response . . . 701

G.3 Second partial derivatives of the IIR filter response . . . 702

G.3.1 Second partial derivatives of the IIR filter amplitude response . . . 702

G.3.2 Second partial derivatives of the IIR filter phase response . . . 704

G.3.3 Second partial derivatives of the IIR filter group-delay response . . . 706

G.4 Octave implementations . . . 707

H Gradient of the IIR filter amplitude response with respect to frequency **708**

I Allpass filter frequency response **710**

I.1 Allpass filter phase response . . . 710

I.2 Allpass filter group delay response . . . 711

J Gradients of the digital state variable filter frequency response **713**

J.1 Gradients of the state variable filter complex frequency response . . . 713

J.2 State variable filter squared-magnitude response . . . 714

J.3 State variable filter phase response . . . 715

J.4 State variable filter group-delay response . . . 715

K Constrained non-linear optimisation **716**

K.1 Newton's method for a quadratic function . . . 716

K.2 Lagrange multipliers . . . 716

K.3 The dual problem . . . 717

K.4 Karush-Kuhn-Tucker conditions for constrained optimisation . . . 717

K.5 Constrained optimisation using Newton's method . . . 718

K.6 Local convergence . . . 719

K.7 Quasi-Newton methods . . . 719

K.7.1 Updating the Hessian approximation with the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) formula . . . 720

K.7.2 A modified Cholesky factorisation of the Hessian . . . 720

K.7.3 Wright's modification for degenerate constraints . . . 722

K.7.4 Bertsekas' modification to the Hessian . . . 723

K.8 Penalty and barrier methods . . . 724

K.8.1 Penalty functions . . . 724

K.8.2 Barrier functions . . . 726

K.9 Finding the step size . . . 726

K.9.1 Line search with the Golden-Section . . . 726

K.9.2 Inexact step-size selection . . . 727

K.9.3 Lanczos step-size selection . . . 728

K.10 Initial solution with the Goldfarb-Idnani algorithm . . . 729

K.11 Implementation examples . . . 735

L Fourier transform of the Gaussian function **740**

L.1 Preliminary results . . . 740

L.1.1 Integral of the Gaussian function . . . 740

L.1.2 Fourier transform of the derivative of a function . . . 740

L.2 Derivation of the Fourier transform of the Gaussian function in the frequency domain . . . 741

M Design of IIR digital filter transfer functions **742**

M.1 Design of discrete time filters with the bilinear transform . . . 742

M.1.1 Design of Butterworth IIR filters . . . 743

M.2 Low passband sensitivity IIR filters . . . 745

M.2.1 Structural Boundedness . . . 745

M.2.2 Filter realisation as the sum of all-pass functions . . . 746

M.2.3 A note on the numerical calculation of the spectral factor . . . 748

M.2.4 Examples of parallel all-pass filter synthesis . . . 748

M.3 Design of Elliptic IIR filters with a reduced number of multipliers . . . 756

M.3.1 Elliptic filter design with the *Landen transformation* . . . 756

M.3.2 Design of elliptic filters with minimal-Q . . . 757

M.4 *Saramäki*'s method for the design of IIR filters with zeros on the unit circle . . . 762

M.4.1 Optimisation of a low-pass filter with denominator order higher . . . 763

M.4.2 Optimisation of a low-pass filter with denominator order lower . . . 765

M.4.3 Surma-aho and Saramäki method of unconstrained optimisation of an initial IIR filter . . . 768

M.5 Johansson and Saramäki design of all-pass complementary IIR filters . . . 774

N Design of FIR digital filter transfer functions **781**

N.1 Low passband sensitivity FIR lattice filters . . . 781

N.1.1 Lattice decomposition of an FIR digital filter . . . 781

N.1.2 Finite-wordlength properties of the lattice FIR filter . . . 782

N.1.3 State variable description of the complementary FIR lattice filter . . . 783

N.1.4 Design of the complementary FIR digital filter . . . 784

N.1.5 Example: the minium-phase complementary filter of an FIR bandpass filter . . . 789

N.1.6 Estimating the MA coefficients of a filtered noise sequence . . . 793

N.1.7 Design of a complementary FIR lattice band-pass Hilbert filter . . . 798

N.2 Design of FIR digital filters with unconstrained optimisation of the piece-wise mean-squared error of the response 802

N.2.1 Zero-phase transfer functions of symmetric FIR filters . . . 802

N.2.2 Zero-phase transfer functions of anti-symmetric FIR filters . . . 803

N.2.3 Piece-wise mean-squared-error of the response of an FIR filter . . . 805

N.2.4 Examples of the design of FIR filters with unconstrained optimisation . . . 806

N.3 PCLS design of symmetric FIR digital filters with Lagrange multipliers . . . 813

N.3.1 Examples of the design of constrained least-squared error symmetric FIR filters with optimisation by the method of Lagrange multipliers . . . 813

N.4 PCLS design of non-symmetric FIR filters with SOCP . . . 822

N.4.1 Frequency response and gradients of a non-symmetric FIR filter . . . 822

N.4.2 Examples of the PCLS design of non-symmetric FIR filters with SOCP optimisation . . . 823

N.5 Constrained mini-max error optimisation of FIR digital filters . . . 827

N.5.1 The alternation theorem . . . 827

N.5.2 *Hofstetter*'s algorithm for mini-max FIR filter approximation . . . 828

N.5.3 *Parks-McClellan* algorithm for mini-max FIR filter approximation . . . 841

N.6 Design of FIR filters with maximally-linear pass-bands and equi-ripple stop-bands using the *Parks-McClellan* algorithm . . . 852

N.6.1 Design of FIR low-pass filters with maximally-flat pass-bands and equi-ripple stop-bands . . . 852

N.6.2 Design of FIR band-pass filters with maximally-flat pass-bands and equi-ripple stop-bands . . . 856

N.6.3 Design of maximally-linear FIR low-pass differentiators with equi-ripple stop-bands . . . 859

N.7 Closed-form design of maximally-linear FIR filters . . . 864

N.7.1 Closed-form design of maximally-flat low-pass FIR filters . . . 864

N.7.2 Closed-form design of maximally-flat FIR half-band filters . . . 876

N.7.3 Closed-form design of maximally-flat FIR Hilbert filters . . . 878

N.7.4 Closed-form design of maximally-linear FIR low-pass differentiators . . . 880

N.8 Linear Matrix Inequality(LMI) design of symmetric FIR filters . . . 885

N.8.1 The *Markov-Lukacs* theorem . . . 885

N.8.2 Trigonometric curves . . . 885

N.8.3 Moment matrix of trigonometric curves . . . 886

N.8.4 A *Markov-Lukacs* theorem for trigonometric curves . . . 886

N.8.5 The conic hull of $C_{a,b}$ . . . 887

N.8.6 Optimisation of the dual of a convex quadratic objective function . . . 888

N.8.7 Example of LMI design of a low pass symmetric FIR filter . . . 889

N.8.8 Example of LMI design of a band-pass symmetric FIR filter . . . 892

N.9 Design of half-band FIR filters . . . 897

N.9.1 Vaidyanathan's "TRICK" for the design of FIR half-band filters . . . 897

N.9.2 Design of equi-ripple FIR half-band filters . . . 898

N.10 Design of equi-ripple FIR filters with Zolotarev polynomials . . . 908

N.10.1 The Zolotarev polynomials . . . 908

N.10.2 Narrow-band FIR filter design with the Zolotarev polynomials . . . 916

N.10.3 Almost equi-ripple low-pass FIR filter design with the Zolotarev polynomials . . . 927

N.11 Design of FIR filters as a tapped cascade of sub-filters . . . 935

N.11.1 Transformations of linear-phase FIR filters . . . 935

N.11.2 Frequency-domain constraints on the prototype and sub-filter . . . 936

N.11.3 Filter design . . . 938

N.11.4 Filter design examples . . . 940

O Application of the *Kalman-Yakubovič-Popov* lemma to digital filter design **945**

O.1 The continuous-time KYP lemma . . . 945

O.1.1 A generalised S-procedure . . . 946

O.1.2 A finite-frequency continuous-time KYP lemma . . . 947

O.2 *Iwasaki* and *Hara*'s generalised KYP lemma for discrete-time systems . . . 950

O.2.1 Frequency transformations in the complex plane . . . 950

O.2.2 A generalised discrete-time KYP lemma . . . 951

O.2.3 Examples of FIR filter design with the generalised KYP lemma . . . 956

O.2.4 *Rantzer*'s transformation of the KYP lemma from discrete-time to continuous-time . . . 965

O.2.5 *Finsler*'s lemma transformation of the generalised KYP lemma . . . 966

O.2.6 The dual of the KYP lemma . . . 967

O.3 Generalisation of the KYP lemma to the union of disjoint frequency intervals . . . 969

O.3.1 Union of frequency intervals . . . 969

O.3.2 Generalised KYP lemma over a union of frequency intervals . . . 975

O.3.3 Examples of FIR filter design with the generalised KYP lemma extended to the union of disjoint frequency bands . . . 977

O.4 Design of one-multiplier Schur lattice filters with the KYP lemma . . . 982

O.4.1 Preliminaries . . . 982

O.4.2 Sequential approximation of BMI constraints . . . 983

O.4.3 Design of one-multiplier Schur lattice filters with BMI constraints derived from the KYP lemma . . . 986

O.4.4 Examples of the design of one-multiplier Schur lattice filters with BMI constraints derived from the KYP lemma . . . 989

Colophon **1003**

Bibliography **1021**

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. . . . 32
1.4 Continued fraction expansion of a rational transfer function. . . . 33
1.5 Transformation of state variable description to direct form. . . . 35
1.6 *Le Verrier*'s algorithm for finding the resolvent of the state transition matrix, A [195, Algorithm 8A.12]. . . . 36
1.7 *La Budde*'s algorithm for finding the characteristic polynomial of A [202, Algorithm 2]. . . . 37
3.1 Recursive calculation of the covariance matrix. . . . 55
3.2 Minimisation of the noise gain. . . . 60
3.3 Optimisation of the noise gain. . . . 60
4.1 Construction of optimised second order state variable filters[195, Figure 9.14.1]. . . . 64
4.2 Bomar second order optimised state variable filter sections [25, Equation 17]. (See also [195, Figure 9.12.1].) . . . 65
4.3 Bomar Type III second order optimised state variable filter sections [25, Equation 23]. . . . 65
5.1 Schur polynomial expansion (see *Parhi* [118, Section 12.2.3]). . . . 95
5.2 One-multiplier lattice sign assignment [4, Page 496]. . . . 99
5.3 Construction of a state variable description of the Schur FIR lattice filter. . . . 104
5.4 Construction of a state variable description of the Schur one-multiplier lattice filter. . . . 105
5.5 Construction of a state variable description of the pipelined Schur one-multiplier lattice filter. . . . 107
5.6 Construction of a state variable description of the doubly-pipelined Schur one-multiplier lattice filter. . . . 109
5.7 Construction of a state variable description of the all-pass doubly-pipelined Schur one-multiplier lattice filter. . . . 110
5.8 Construction of a state variable description of the Scaled-Normalised Lattice. . . . 111
5.9 Construction of a state variable description of the pipelined Schur one-multiplier lattice filter with denominator coefficients in z^{-2} only. . . . 127
5.10 Construction of a state variable description of the pipelined Schur one-multiplier all-pass lattice filter with denominator coefficients in z^{-2} only. . . . 128
7.1 Minimisation of round off noise in error feedback/feedforward state variable filters. (See *Williamson* [48, Theorem 5.2]). . . . 143
8.1 Exchange algorithm for multiband FIR filter of *Selesnick, Lang and Burrus* [92, p.498]. . . . 150
8.2 Compute the powers of a matrix. (See *Golub and van Loan* [59, Algorithm 11.2.2].) . . . 153
11.1 Conversion of 2's complement numbers to the canonical signed-digit representation (*Parhi* [118, Section 13.6.1]). 429
11.2 Modified signed-digit allocation heuristic of *Ito et al.* [210]. . . . 430
16.1 The maximum-cut algorithm of *Goemans* and *Williamson* [151] . . . 591
D.1 Carlson's algorithm for computing the R_F function [16, Algorithm 1] . . . 686
D.2 Carlson's algorithm for computing the R_C function [16, Algorithm 2] . . . 686
D.3 Carlson's algorithm for computing the R_J function [16, Algorithm 3] . . . 686
D.4 Carlson's algorithm for computing the R_D function [16, Algorithm 4] . . . 687
E.1 *Lanczos* tridiagonalisation of an unsymmetric matrix. . . . 694
K.1 The sequential quadratic programming method . . . 719
K.2 Cholesky factorisation of an $n \times n$ positive-definite symmetric matrix, W [44, Appendix D.1] . . . 721
K.3 Line search using the Armijo rule. . . . 727
K.4 Line search using Wolfe conditions. . . . 728
K.5 Line search using the Goldstein conditions. . . . 728
K.6 The *Goldfarb-Idnani* dual algorithm [41]. . . . 730
M.1 Filter realisation as the sum of all-pass functions. . . . 748
M.2 *Surma-aho* and *Saramäki* method for finding an initial IIR filter in *gain-pole-zero* form [116, pp. 958-959]. . . . 768
N.1 Construction of a state variable description of the complementary FIR lattice filter. . . . 783
N.2 Parks-McClellan FIR filter design [170, Section 7.4.3] [243, 96]. . . . 842
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 [150, Table 1]. . . . 868
N.4 *Zahradník* and *Vlček*'s algorithm [183, Table 1] for the evaluation of the coefficients, a_l, of $\mathcal{U}_n$. . . . 902
N.5 *Vlček* and *Unbehauen*'s backwards recursion for $Z_{p,q}(w) = \sum_{m=0}^{n} b_m w^m(w)$ [148, Table IV], [149]. . . . 911

N.6 Modified *Vlček* and *Unbehauen* backwards recursion for $Z_{p,q}(w) = \sum_{m=0}^{n} a_m T_m(w)$ [148, Table V], [149]. . . 912
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 [185, Table I]. 923
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 [157, Tables 4 and 5]. 928

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.* [220]). 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 [111][b]. Octave is an "almost" compatible open-source-software clone of the commercial *MATLAB* package [246]. 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* [229]. 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, 137] or by constraining the reflection coefficients of a tapped all-pass lattice filter implementation [4, 118]. A valid initial solution for the MMSE solver is found by "eye" or by using the WISE method of *Tarczynski et al.* [11]. A peak-constrained-least-squares (PCLS) solution is found by the exchange algorithm of *Selesnick et al.* [92]. 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.* [263] or that of *Ito et al.* [210] 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* [195] and chapter 12 of *"VLSI Digital Signal Processing Systems:Design and Implementation"* by *Parhi* [118].

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 9.7 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* [92]. Appendix N.4 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* [112], *Ruszczynski* [9] and *Bertsekas* [44]. I chose to write my own SQP solver in Octave. In Chapter 8, I follow *Deczky* [3] and *Richards* [137] 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* [250]. SOCP is a subclass of *convex programming* [215]. See the review articles by *Alizadeh* and *Goldfarb* [55] and *Lobo et al.* [146] 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* [229].

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.* [172] and *Vaidyanathan et al.* [180] 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* [144, 145], describe IIR digital filterbanks as a "polyphase" combination of all-pass digital filters. *Gray* and *Markel* [4] and *Parhi* [118, 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* [79], *Milić* and *Ćertić* [124] and *Lu and Hinamoto* [250] describe efficient, sharp transition-band IIR filter designs using the *frequency masking* approach described for FIR filters by *Lim* [261].

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.* [263] and *Ito et al.* [210] 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 floating-point 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.* [210]. 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 18.1, 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 [163].

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* [71] and *"The perfidious polynomial"* by *Wilkinson* [97]. 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 *qroots*, an implementation using 128-bit floating-point numbers based on *gsl_poly_complex_solve* from the GNU GSL library[70][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 [40] finds the expected roots. The package includes an Octave interface. The Matlab *multroot* solver [267] finds the expected roots.

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 floating-point 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 *qroots* 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* [195]. 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* [195, 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 $\boldsymbol{f}$

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

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

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

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

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

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

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

1.2 Filter difference equation

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

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

If $\star$ represents the convolution operation and $u(k)$ and $y(k)$ 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 $a(z)$ 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

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

and if λ is a root of $a(z)$ then

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

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

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

The case of repeated roots can be treated by differentiating Equation 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 a unit delay (z^{-1})
2. There are no delay free loops in the graph
3. There are a finite number of nodes and branches

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.

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:

- 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:

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

In the z-domain:

$$y(z) = b_0 z^2 + b_1 z^1 + b_2$$
$$u(z) = z^2 + a_1 z + a_0$$

The transfer function is:

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

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

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

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

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

$$y(z) = b_0 v + B_1(z)\, v$$
$$u(z) = v + a_0 A_1(z)\, v$$

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.

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.

Figure 1.4: Unit delay equivalent.

Figure 1.5: Signal flow graph of a state variable 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

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

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]:

$$
H(z) = D + C(zI - A)^{-1} B \tag{1.8}
$$

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

$$= D + CTT^{-1}\left(zI - A\right)^{-1}\left(T^{-1}\right)^{-1}T^{-1}B \tag{1.9}$$

$$= D + CT\left[T^{-1}\left(zI - A\right)T\right]^{-1}T^{-1}B \tag{1.10}$$

$$= D + CT\left(zI - T^{-1}AT\right)^{-1}T^{-1}B \tag{1.11}$$

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.

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* [195, Problems 8.16, 10.13 to 10.16] and *Mitra* and *Sherwood* [219]. Algorithm 1.4 calculates the continued fraction expansion of a rational transfer function.

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

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* [59, 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.

Le Verrier'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 *Le Verrier*'s algorithm [195, Algorithm 8A.12] shown in Algorithm 1.6. Note that *Le Verrier*'s algorithm calculates the characteristic polynomial

Algorithm 1.6 *Le Verrier*'s algorithm for finding the resolvent of the state transition matrix, A [195, 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

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:

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

When the characteristic polynomial is known, the resolvent and the transfer function can be calculated with the matrix recursion of *Le Verrier*'s algorithm [195, pp.332–333]. The m-file *Abcd2tf.m* implements *Le Verrier*'s algorithm. The oct-file *Abcd2tf.cc* implements *Le Verrier*'s algorithm with extended precision using the MPFR library [121].

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

La Budde [33] describes "a new algorithm for reducing an arbitrary real square matrix to tri-diagonal form using real similarity transformations". *Parlett* [187] commented that " ... this procedure is identical to the more familiar reduction by elimination methods. Therefore the same care is needed with the new technique as with elimination in treating the instabilities which can occur ...". *Wilkinson* [98, Section 57, Chapter 6], *Rehman* [213, Chapter 6], and *Rehman* and *Ipsen* [202] describe a *La Budde*'s style algorithm for the calculation of the characteristic polynomial of an *upper Hessenberg* matrix, reproduced here as Algorithm 1.7 [202, Algorithm 2]. *Rehman* [213, Appendix B] provides a *MATLAB* implementation, *labudde.m*, that I have edited for compatiblity with Octave. *Rehman* shows that the numerical performance of *La Budde*'s algorithm compares favourably with the eigenvalue method, $p(z) = \prod_{k=1}^{n} (z - \lambda_k)$, where the λ_k are the (possibly repeated) eigenvalues of A.

Algorithm 1.7 *La Budde*'s algorithm for finding the characteristic polynomial of A [202, 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 first stage of *La Budde*'s algorithm reduces the non-symmetric matrix, A, to *upper Hessenberg* form, H (see *Golub* and *van Loan* [59, 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* [202, 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 [98, 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 $2 \leq k \leq n$. The recursion for $p_k(z) = \sum_{m=0}^{k} z^{k-m} a_m^{(k)}$ is obtained by developing the determinant of $zI - H_k$ along the last row of H_k. Each term in the sum contains an element in the last column of H_k and a product of subdiagonal elements. Equating like powers of z in Equation 1.13 gives recursions for individual coefficients a_k.

1.9.4 Sensitivity of the state variable description of a transfer function

Thiele [128] 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 identities:

$$RR^{-1} = I \tag{1.14a}$$

$$\frac{\partial R}{\partial x} R^{-1} + R \frac{\partial R^{-1}}{\partial x} = 0 \tag{1.14b}$$

$$\frac{\partial R}{\partial x} = -R \frac{\partial R^{-1}}{\partial x} R \tag{1.14c}$$

where x 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 α, β, γ and δ 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 λ_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 [195, Equation 8.3.21]:

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

1.12 Factored state variable descriptions

See *Roberts* and *Mullis* [195, 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.

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

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

(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 [195, Figure 8.4.6].)

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

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:

$$S_0 = \{x_1, x_2, u\}$$
$$S_1 = \{w, x_1'\}$$
$$S_2 = \{y, x_2'\}$$

where:

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

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

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

The factored state variable equations are:

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

Factoring a feedback connection of state variable filters

Roberts and *Mullis* [195, 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:

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

$$
\begin{aligned}
v_1 &= c_1 x_2 + d_{12} w_2 \\
v_2 &= c_2 x_2 + d_{21} y_2 + d_{22} w_2
\end{aligned}
$$

The node variable sets are:

$$
\begin{aligned}
S_0 &= \{x_1, x_2, u_1, w_2\} \\
S_1 &= \{v_1\} \\
S_2 &= \{x_1', y_1, y_2\} \\
S_3 &= \{x_2', v_2\}
\end{aligned}
$$

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:

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

$$
= \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & 0 & b_1 & A_2 & b_2 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & d_{21} & c_2 & d_{22} \end{bmatrix} \begin{bmatrix} B_2 & A_1 & 0 & B_1 & 0 \\ D_{12} & C_1 & 0 & D_{11} & 0 \\ D_{22} & C_2 & 0 & D_{21} & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} 0 & c_1 & 0 & d_{12} \\ I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix}
$$

$$
= \begin{bmatrix} A_1 & B_2 c_1 & B_1 & B_2 d_{12} \\ b_1 C_2 & b_1 D_{22} c_1 + A_2 & b_1 D_{21} & b_1 D_{22} d_{12} + b_2 \\ C_1 & D_{12} c_1 & D_{11} & D_{12} d_{12} \\ d_{21} C_2 & d_{21} D_{22} c_1 + c_2 & d_{21} D_{21} & d_{21} D_{22} d_{12} + d_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u_1 \\ w_2 \end{bmatrix}
$$

1.13 Block processing and decimation filters

See *Roberts* and *Mullis* [195, 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 $P = 2$:

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

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 [195, Table 10.2.1].)

$$\begin{bmatrix} x(k+2) \\ y(k) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ C & 0 & D \end{bmatrix} \begin{bmatrix} x(k+1) \\ y(k) \\ u(k+1) \end{bmatrix}, \text{ second update} \tag{1.16}$$

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* [195, 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.

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[d] 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.

[d] See Chapter 3.

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.

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 Digital Filters

See *Roberts* and *Mullis* [195, 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 λ is a pole (or zero) $G(z)$, then $F(\lambda)$ is a pole (or zero) of $H(z)$. Thus if $|\lambda| < 1$ implies $|F(\lambda)| < 1$ then these properties are preserved.

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

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

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

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

This is a frequency transformation since

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

Order n frequency transformations are known as *Blaschke products*

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

where

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

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

Figure 2.1: Frequency warping function (after *Roberts* and *Mullis* [195, Figure 6.7.2]).

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* [195, Figure 6.7.3] show an algorithm for calculating the parameters of the frequency transformation $F(z)$. This algorithm is implemented by the Octave [111] function *phi2p*.

2.1 Frequency Transformation of the Transfer Function

See *Roberts* and *Mullis* [195, 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 $p(z)$:

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

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

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

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*.

2.2 Frequency Transformations of State Variable Filters

See *Mullis and Roberts* [35, Section III]. Given a (usually low-pass) prototype filter $H(z)$ parameterised by the state variable description $\{A, b, c, d\}$ construct the filter $G(z) = H(F(z))$ where

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

where $*$ 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

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

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 $S(k)$

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

Eliminating $x(k)$ gives

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

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 [35, Appendix A] give a direct proof. Start with the identity:

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

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

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

[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.

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*.

2.3 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* [195].

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 Δ is the maximum voltage represented and b_0 is the sign bit. The quantisation 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 quantisation levels are crossed between samples of $u(k)$. The signal-to-noise ratio obtained depends on the statistics of both the quantisation 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* [195, 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 realisation
- 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 realisations 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 calculating the covariance matrix are:

- find the auto-correlation sequence of the characteristic equation of the state transition matrix with the inverse-Levinson recursion. See *Roberts* and *Mullis* [195, Section 9.11, p. 393]. The Octave function *dlyap_levinson* implements this solution.
- use *Hammarling*'s solution of the discrete *Lyapunov* equation [216]. The Octave *Control* package contains the *dlyap* function that calls functions from an open-source version of the *SLICOT* library [54] 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 realisation. For a fixed point number representation the range of internal variables is:

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

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

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

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

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

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

This can be achieved by applying the coordinate transformation:

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

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 realisation is scaled. The round off noise at the output can be modeled as shown in Figure 3.2.

Where:

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

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

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

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

Recall that q is the quantisation 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 realisation. 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 quantisation and the round off in calculating the output. Neither term depends on the filter realisation. 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 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.

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.

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 Minimisation of round-off noise in the calculation of the state vector

Minimising the noise gain minimises the round off noise. *Mullis* and *Roberts* [195, Section 9.9] , [36] 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*.

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, $\mathcal{O}(n)$, but will have a larger than optimal noise gain. The noise gain for a non-optimal

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

$$K = DWD$$

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

Algorithm 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: $U_1^* U_1 = U_1 U_1^* = I$, where $*$ 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 is:

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

filter may be improved by distributing bits unevenly between the states. *Mullis* and *Roberts* [36, 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 quantisation 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 realisation 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

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

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

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

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

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

Since the corresponding state variable description is

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

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 2.2 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* [35, 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 mth 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* [195, 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 [25] 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* [195, 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 [25] 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 [26] 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* [195, 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[195, 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 [25, Equation 17]. (See also [195, 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 [25, 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 realisation can be block optimised by:

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

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 Nth 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 floating-point, 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 [167] *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 coefficients 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.	1.66	2.30	2.38

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.	1.70	2.35	2.37

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 coefficients 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 [180]. 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 ϕ_{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* [218] show a catalogue of realisations of minimum multiplier first and second order all-pass filters. They consider realisations 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 [218, 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 [218, 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 [103] 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 ϵ 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.* [72] 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* [118, 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 *Liu* [196]. This realisation has low noise gain for poles near $z = \pm\imath$.

The three port transfer matrix of the *Mitra* and *Hirano* [218] 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.* [72] 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* [115] 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-Liu 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 *Liu* 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-Liu 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-Liu 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 ϵ 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 *Liu* 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-Liu all-pass filter section with real poles.

Figure 4.25: Noise gain of the scaled and retimed second-order Ansari-and-Liu 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* [118, Chapter 12].

5.1 The Schur algorithm

From *Parhi* [118, 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:

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

where $k_N = \phi_0/\phi_N$ and $\tilde{\Phi}_N(z)$ is the reverse polynomial of $\Phi_N(z)$ defined by:

$$\tilde{\Phi}_N(z) = z^N \Phi_N\left(z^{-1}\right)$$

Figure 5.1: A filter structure implementing $H(z) = \frac{N_N(z)}{P_N(z)}$ (after *Parhi* [118, 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:

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

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* [118, Equation 12.6] points out that by inspection of $\Phi_{N-1}(z)$, the coefficients of increasing powers of z in $\Phi_{N-1}(z)$ are $\frac{1}{\sqrt{\phi_N^2 - \phi_0^2}}$ times the N determinants of the 2×2 submatrices formed by the first column and each succeeding column in the matrix:

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

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 [121, 68] 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:

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

where $*$ 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 α, β and γ:

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

3. Positive norm:

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

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

Some identities:

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

Proof: Recall that $\tilde{\Phi}_N(z) = z^N \Phi_N\left(z^{-1}\right)$. Then

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

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:

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

If

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

then:

$$\begin{aligned} \langle \Phi_N(z), P(z) \rangle &= \frac{p_N}{\phi_N} \\ \left\langle \tilde{\Phi}_N(z), P(z) \right\rangle &= \frac{p_0}{\phi_N} \end{aligned}$$

Also:

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

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

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

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

$$\left\{ \tilde{\Phi}_N(z), z\tilde{\Phi}_{N-1}(z), z^2 \tilde{\Phi}_{N-2}(z), \ldots, z^N \tilde{\Phi}_0(z) \right\}$$

also form an orthonormal basis. This basis can be used to synthesise an alternative lattice structure. *Parhi* [118, 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* [118, 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**
    $c_i = \frac{\tilde{Q}(0)}{\tilde{\Phi}_i(0)}$
    $Q(z) = Q(z) - c_i \Phi_i(z)$
  end for
$\tilde{Q}(z)$ and $\tilde{\Phi}_i(z)$ 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

$$\Phi_{i-1}(z) = \frac{z^{-1}\left\{\Phi_i(z) - k_i \tilde{\Phi}_i(z)\right\}}{s_i} \tag{5.1}$$

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

Note that since

$$\tilde{\Phi}_{i-1}(z) = z^{i-1} \Phi_{i-1}\left(z^{-1}\right)$$

$$= \frac{\tilde{\Phi}_i(z) - k_i \Phi_i(z)}{s_i}$$

by rearranging

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

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

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

The reverse Schur polynomial is

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

so

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

Substituting Equation 5.5 into Equation 5.2

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

so

$$\Psi_N(z) = z\Psi_{N-1}(z) + k_N\tilde{\Psi}_{N-1}(z) \tag{5.6}$$

and, applying the definition of the reverse polynomial

$$\tilde{\Psi}_N(z) = zk_N\Psi_{N-1}(z) + \tilde{\Psi}_{N-1}(z) \tag{5.7}$$

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 $\tilde{\Psi}_N(z)/\Psi_N(z)$. The relation between $\Phi_i(z)$ and $\Psi_i(z)$ is

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

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

$$k_i = \frac{\Psi_i(0)}{\tilde{\Psi}_i(0)} = \frac{\Phi_i(0)}{\tilde{\Phi}_i(0)}$$

Also the Ψ_i are orthogonal but not orthonormal since

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

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.

Figure 5.2: FIR filter structure (after *Parhi* [118, Fig. 12.8]).

Figure 5.3: All-pass and all-pole filter structure (after *Parhi* [118, Fig. 12.5]).

5.3 Derivation of the One-Multiplier IIR Lattice Filter

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

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

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

$$\Lambda_{N-1}(z) = \frac{z^{-1}\left\{\Lambda_N(z) - k_N\tilde{\Lambda}_N(z)\right\}}{1-\epsilon_N k_N}$$
$$\tilde{\Lambda}_{N-1}(z) = \frac{\tilde{\Lambda}_N(z) - k_N\Lambda_N(z)}{1-\epsilon_N k_N}$$

where $k_i = \Lambda_i(0)/\tilde{\Lambda}_i(0)$. Rearranging

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

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

$$k_i = \Lambda_i(0)/\tilde{\Lambda}_i(0) = \Phi_i(0)/\tilde{\Phi}_i(0)$$

Also the Λ_i are orthogonal but not orthonormal since

$$\langle\Lambda_i(z), \Lambda_i(z)\rangle = \langle\tilde{\Lambda}_i(z), \tilde{\Lambda}_i(z)\rangle \tag{5.10}$$
$$= \frac{(1+\epsilon_N k_N)(1+\epsilon_{N-1}k_{N-1})\cdots(1+\epsilon_{i+1}k_{i+1})}{(1-\epsilon_N k_N)(1-\epsilon_{N-1}k_{N-1})\cdots(1-\epsilon_{i+1}k_{i+1})} \tag{5.11}$$

The magnitude of $\langle\Lambda_i(z), \Lambda_i(z)\rangle$ can be adjusted by choosing the sign parameters so that the one-multiplier lattice filter can avoid the severe input scaling of the basic lattice filter and have 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* [93, Figure 3].

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. Note that if the ϵ_m sign parameters are changed then the Λ_i polynomials will also change.

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 ϵ_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

Figure 5.4: One-multiplier lattice structure (after *Parhi* [118, Fig. 12.11]).

Figure 5.5: Normalised lattice filter (after *Parhi* [118, Fig. 12.20]).

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

$$\Phi_{N-1}(z) = \frac{z^{-1}\left\{\Phi_N(z) - k_N \tilde{\Phi}_N(z)\right\}}{\sqrt{1-k_N^2}}$$

where $k_i = \Phi_i(0)/\tilde{\Phi}_i(0)$. The reverse polynomial is

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

So

$$\tilde{\Phi}_N(z) = \sqrt{1-k_N^2}\,\tilde{\Phi}_{N-1}(z) + k_N \Phi_N(z) \tag{5.13}$$

$$\Phi_{N-1}(z) = z^{-1}\left\{\sqrt{1-k_N^2}\,\Phi_N(z) - k_N \tilde{\Phi}_{N-1}(z)\right\} \tag{5.14}$$

$$\Phi_N(z) = \frac{1}{\sqrt{1-k_N^2}}\left\{z\Phi_{N-1}(z) + k_N \tilde{\Phi}_{N-1}(z)\right\} \tag{5.15}$$

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

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

(a) Original lattice section.

(b) After slow-down.

(c) After slow-down and re-timing.

Figure 5.6: Slowed and retimed lattice section.

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

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 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:

$$\sigma_{10}^{(i)} = \frac{c_i}{\sqrt{\sum_{j=0}^{i} c_j^2}} \tag{5.18}$$

$$\sigma_{11}^{(i)} = \frac{\sqrt{\sum_{j=0}^{i-1} c_j^2}}{\sqrt{\sum_{j=0}^{i} c_j^2}} \tag{5.19}$$

[a]The roundoff-noise performance of the transformed filter is the same as that of the original filter

Figure 5.7: Normalised-scaled lattice filter (after *Parhi* [118, Fig. 12.19]).

3. Any module N to 1:

$$\sigma_{20}^{(i)} = -\sigma_{02}^{(i)} = k_i \tag{5.20}$$

$$\sigma_{00}^{(i)} = \sigma_{22}^{(i)} = \sqrt{1-k_i^2} \tag{5.21}$$

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 σ_{10} and σ_{11} lattice coefficients from the c_i expansion coefficients. For convenience, *schurNSscale* combines $\sigma_{10}^{(0)}$ and $\sigma_{11}^{(1)}$. The Octave function *tf2schurNSlattice* calculates the coefficients of the normalised-scaled Schur lattice from the transfer function.

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

Parhi [118, Example 12.6.1] uses as an example a third order Butterworth low-pass filter with cutoff at angular frequency 0.1π (where the sampling frequency is normalised to 2π). The squared magnitude of the n-th order Butterworth response is defined to be

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

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 ω_c is the s-plane cutoff angular frequency and t_0 is the sampling interval) then *pre-warp* the s-plane frequency axis so that the corresponding cut-off in the s-plane is:

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

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

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

Figure 5.8: Butterworth 3rd order normalised-scaled lattice filter example (after *Parhi* [118, Fig. 12.20]).

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 $H(z)$.

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 $\hat{y}_n$ 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.

Given $\{k_1, k_2, \ldots, k_N\}$:

$y_0 = u$
$\hat{y}_0 = u$
for $n = 1, \ldots, N$ **do**
  $x'_n = \hat{y}_{n-1}$
  $y_n = k_n x_n + y_{n-1}$
  $\hat{y}_n = x_n + k_n y_{n-1}$
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:

$$\begin{bmatrix} y_0 \\ \hat{y}_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ y_1 \\ \hat{y}_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & k_1 & 0 & \cdots & 0 \\ k_1 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & & & & \cdots & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ \hat{y}_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

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

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

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

The Octave function *schurFIRlattice2Abcd* returns the state variable description of a Schur FIR lattice filter. The Octave script *schurFIRlattice2Abcd_symbolic_test.m* creates a symbolic state variable description of the 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 $\hat{y}_n$ the all-pass output of the n-th section.

Figure 5.9: State variable description of the Schur one-multiplier lattice filter.

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.

Given $\{k_1, k_2, \ldots, k_N\}$, $\{\epsilon_1, \epsilon_2, \ldots, \epsilon_N\}$ and $\{c_0, c_1, \ldots, c_N\}$:

$\hat{y}_0 = x_1$
for $n = 1, \ldots, N-1$ **do**
  $x'_n = -k_n \hat{y}_{n-1} + (1 + k_n \epsilon_n)\, x_{n+1}$
  $\hat{y}_n = (1 - k_n \epsilon_n)\, \hat{y}_{n-1} + k_n x_{n+1}$
end for
$x_N = -k_N \hat{y}_{N-1} + (1 + k_N \epsilon_N)\, u$
$\hat{y} = (1 - k_N \epsilon_N)\, \hat{y}_{N-1} + k_N u$
$y = c_0 x_1 + c_1 x_2 + \cdots + c_{N-1} x_N + c_N u$

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. The all-pass output of the Schur one-multiplier lattice filter is constructed as follows[b]:

$$\begin{bmatrix} \hat{y}_0 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & & \cdots & 0 \\ 0 & 1 & & & & \vdots \\ \vdots & & \ddots & & & \\ & & & \ddots & & \vdots \\ 0 & & & & 1 & 0 \\ 0 & \cdots & & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \hat{y}_1 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} -k_1 & (1 + k_1\epsilon_1) & 0 & \cdots & \cdots & 0 \\ (1 - k_1\epsilon_1) & k_1 & 0 & & & \vdots \\ 0 & 0 & 1 & & & \\ \vdots & & & \ddots & & \vdots \\ 0 & & & & 1 & 0 \\ 0 & & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_0 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

[b]Noting that $\det \begin{bmatrix} -k_l & 1 + k_l\epsilon_l \\ 1 - k_l\epsilon_l & k_l \end{bmatrix} = -1$ and $\det PQ = \det P \det Q$, then, by inspection, $\det A_{1\ldots l, 1\ldots l} = -1^l k_l$.

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

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

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

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). The Octave script *schurSchurOneMlattice2Abcd_symbolic_test.m* creates a symbolic state variable description of the Schur one multiplier lattice filter.

5.6.3 State variable description of a pipelined one-multiplier Schur lattice filter

Figure 5.10a shows a representation 3rd order lattice filter and Figure 5.10b shows the filter after pipelining by moving the second filter delay to the upper and lower branches of the signal flow graph. The total delay in each loop of the signal flow graph is unchanged so the transfer function of the filter is also unchanged. This pipelining 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[c].

(a) Original filter.

(b) Filter after retiming.

Figure 5.10: Example of pipelining a 3rd order lattice filter.

Figure 5.11 shows a second order segment of the pipelined Schur one-multiplier lattice filter in Figure 5.10b.

Construction of a state variable description of the pipelined Schur one-multiplier lattice filter is summarised in Algorithm 5.5.

[c]The tapped output could also 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. See *Kimura* and *Osada* [81].

Figure 5.11: Second order segment of the pipelined Schur one-multiplier lattice filter.

Algorithm 5.5 Construction of a state variable description of the pipelined Schur one-multiplier lattice filter.

Given $\{k_1, k_2, \ldots, k_N\}$, $\{\epsilon_1, \epsilon_2, \ldots, \epsilon_N\}$ and $\{c_0, c_1, \ldots, c_N\}$:

$y_0 = c_0 x_1$
$\hat{y}_0 = x_1$
for $n = 1, \ldots, \lceil \frac{N}{2} \rceil - 1$ **do**
$x'_{3n-2} = -k_{2n-1}\hat{y}_{n-1} - (1 + k_{2n-1}\epsilon_{2n-1})\, k_{2n} x_{3n} + (1 + k_{2n-1}\epsilon_{2n-1})(1 + k_{2n}\epsilon_{2n})\, x_{3n+1}$
$x'_{3n-1} = y_{n-1} - c_{2n-1} k_{2n} x_{3n} + c_{2n-1}(1 + k_{2n}\epsilon_{2n})\, x_{3n+1}$
$x'_{3n} = (1 - k_{2n-1}\epsilon_{2n-1})\, \hat{y}_{n-1} - k_{2n-1} k_{2n} x_{3n} + k_{2n-1}(1 + k_{2n}\epsilon_{2n})\, x_{3n+1}$
$y_n = x_{3n-1} + c_{2n} x_{3n+1}$
$\hat{y}_n = (1 - k_{2n}\epsilon_{2n})\, x_{3n} + k_{2n} x_{3n+1}$
end for
if N is odd **then**
$x'_{3\lceil \frac{N}{2} \rceil - 2} = -k_N \hat{y}_{\lceil \frac{N}{2} \rceil - 1} + (1 + k_N \epsilon_N)\, u$
$y = y_{\lceil \frac{N}{2} \rceil - 1} + c_N u$
$\hat{y} = (1 - k_N \epsilon_N)\, \hat{y}_{\lceil \frac{N}{2} \rceil - 1} + k_N u$
else
$x'_{3\frac{N}{2} - 2} = -k_{N-1}\hat{y}_{\frac{N}{2}-1} - (1 + k_{N-1}\epsilon_{N-1})\, k_N x_{3\frac{N}{2}} + (1 + k_{N-1}\epsilon_{N-1})(1 + k_N\epsilon_N)\, u$
$x'_{3\frac{N}{2} - 1} = y_{\frac{N}{2}-1} - c_{N-1} k_N x_{3\frac{N}{2}} + c_{N-1}(1 + k_N\epsilon_N)\, u$
$x'_{3\frac{N}{2}} = (1 - k_{N-1}\epsilon_{N-1})\, \hat{y}_{\frac{N}{2}-1} - k_{N-1} k_N x_{3\frac{N}{2}} + k_{N-1}(1 + k_N\epsilon_N)\, u$
$y = x_{3\frac{N}{2}-1} + c_N u$
$\hat{y} = (1 - k_N\epsilon_N)\, x_{3\frac{N}{2}} + k_N u$
end if

As in Section 5.6.2, the state variable description can be expressed as a series of matrix multiplications. For an even order filter:

$$\begin{bmatrix} y_0 \\ \hat{y}_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{3\frac{N}{2}} \\ u \end{bmatrix} = \begin{bmatrix} c_0 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & & & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{3\frac{N}{2}} \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ y_1 \\ \hat{y}_1 \\ x_4 \\ \vdots \\ x_{3\frac{N}{2}} \\ u \end{bmatrix} = \begin{bmatrix} 0 & -k_1 & 0 & 0 & -(1 + k_1\epsilon_1)\, k_2 & (1 + k_1\epsilon_1)(1 + k_2\epsilon_2) & \cdots & 0 \\ 1 & 0 & 0 & 0 & -c_1 k_2 & c_1(1 + k_2\epsilon_2) & \cdots & 0 \\ 0 & (1 - k_1\epsilon_1) & 0 & 0 & -k_1 k_2 & k_1(1 + k_2\epsilon_2) & \cdots & 0 \\ 0 & 0 & 0 & 1 & 0 & c_2 & \cdots & 0 \\ 0 & 0 & 0 & 0 & (1 - k_2\epsilon_2) & k_2 & \cdots & 0 \\ & & & & & & \ddots & \vdots \\ 0 & \cdots & & & & & \cdots\ 1 & 0 \\ 0 & \cdots & & & & & \cdots\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ \hat{y}_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{3\frac{N}{2}} \\ u \end{bmatrix}$$

$$
\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3\frac{N}{2}} \\ y \\ \hat{y} \end{bmatrix} = \begin{bmatrix}
1 & \cdots & & & & & \cdots & 0 \\
\vdots & \ddots & & & & & & \vdots \\
0 & \cdots & 0 & -k_{N-1} & 0 & 0 & -\left(1+k_{N-1}\epsilon_{N-1}\right)k_N & \left(1+k_{N-1}\epsilon_{N-1}\right)\left(1+k_N\epsilon_N\right) \\
0 & \cdots & 1 & 0 & 0 & 0 & -c_{N-1}k_N & c_{N-1}\left(1+k_N\epsilon_N\right) \\
0 & \cdots & 0 & 1-k_{N-1}\epsilon_{N-1} & 0 & 0 & -k_{N-1}k_N & k_{N-1}\left(1+k_N\epsilon_N\right) \\
0 & \cdots & 0 & 0 & 0 & 1 & 0 & c_N \\
0 & \cdots & 0 & 0 & 0 & 0 & \left(1-k_N\epsilon_N\right) & k_N
\end{bmatrix}
\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3\frac{N}{2}-3} \\ y_{\frac{N}{2}-1} \\ \hat{y}_{\frac{N}{2}-1} \\ x_{3\frac{N}{2}-2} \\ x_{3\frac{N}{2}-1} \\ x_{3\frac{N}{2}} \\ u \end{bmatrix}
$$

For an odd order filter, the final matrix multiplication is:

$$
\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3\lceil\frac{N}{2}\rceil-2} \\ y \\ \hat{y} \end{bmatrix} = \begin{bmatrix}
1 & \cdots & & & \cdots & 0 \\
\vdots & \ddots & & & & \vdots \\
0 & \cdots & 0 & -k_N & 0 & \left(1+k_N\epsilon_N\right) \\
0 & \cdots & 1 & 0 & 0 & c_N \\
0 & \cdots & 0 & \left(1-k_N\epsilon_N\right) & 0 & k_N
\end{bmatrix}
\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3\lceil\frac{N}{2}\rceil-3} \\ y_{\lceil\frac{N}{2}\rceil-1} \\ \hat{y}_{\lceil\frac{N}{2}\rceil-1} \\ x_{3\lceil\frac{N}{2}\rceil-2} \\ u \end{bmatrix}
$$

The Octave function *schurOneMlatticePipelined2Abcd* returns the state variable description of a pipelined Schur one-multiplier lattice filter (including the all-pass filter Aap, Bap, Cap and Dap output matrixes).

The Octave script *schurOneMlatticePipelined2Abcd_symbolic_test.m* creates a symbolic state variable description of a retimed Schur one-multiplier lattice filter.

5.6.4 State variable description of a doubly-pipelined one-multiplier Schur lattice filter

Figure 5.12 shows a first order segment of a doubly-pipelined Schur one-multiplier lattice filter. In this case, the state transition matrix is linear in the coefficients and there are two delays in each filter loop rather than the original single delay. In other words, the filter calculations are assumed to be performed at double the input sample rate.

Figure 5.12: First order segment of a doubly pipelined Schur one-multiplier lattice filter.

Construction of a state variable description of the doubly-pipelined Schur one-multiplier lattice filter is summarised in Algorithm 5.6.

As in Section 5.6.2, the state variable description can be expressed as a series of matrix multiplications:

$$
\begin{bmatrix} x'_1 \\ y_0 \\ \hat{y}_0 \\ x_2 \\ x_3 \\ \vdots \\ x_{3N+2} \\ u \end{bmatrix} = \begin{bmatrix}
0 & 1 & 0 & \cdots & 0 & 0 \\
c_0 & 0 & 0 & & 0 & 0 \\
1 & 0 & 0 & & 0 & 0 \\
0 & 1 & 0 & & 0 & 0 \\
0 & 0 & 1 & & 0 & 0 \\
\vdots & & & \ddots & \vdots & \vdots \\
0 & & & \cdots & 1 & 0 \\
0 & & & \cdots & 0 & 1
\end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{3N+2} \\ u \end{bmatrix}
$$

Algorithm 5.6 Construction of a state variable description of the doubly-pipelined Schur one-multiplier lattice filter.

Given $\{k_1, k_2, \ldots, k_N\}$, $\{\epsilon_1, \epsilon_2, \ldots, \epsilon_N\}$ and $\{c_0, c_1, \ldots, c_N\}$:

$y_0 = c_0 x_1$
$\hat{y}_0 = x_1$
$x'_1 = x_2$
for $n = 1, \ldots, N$ **do**
  $x'_{3n-1} = -k_n \hat{y}_{n-1} + (1 + k_n \epsilon_n)\, x_{3n+2}$
  $x'_{3n} = y_{n-1} + c_n x_{3n+2}$
  $x'_{3n+1} = (1 - k_n \epsilon_n)\, \hat{y}_{n-1} + k_n x_{3n+2}$
  $y_n = x_{3n}$
  $\hat{y}_n = x_{3n+1}$
end for
$x'_{3N+2} = u$
$y = y_N$
$\hat{y} = \hat{y}_N$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ x'_4 \\ y_1 \\ \hat{y}_1 \\ x_5 \\ x_6 \\ \vdots \\ x_{3N+2} \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & -k_1 & 0 & 0 & 0 & (1+k_1\epsilon_1) & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & c_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & (1-k_1\epsilon_1) & 0 & 0 & 0 & k_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & & & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ y_0 \\ \hat{y}_0 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_{3N+2} \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3N-1} \\ x'_{3N} \\ x'_{3N+1} \\ y_N \\ \hat{y}_N \\ x_{3N+2} \\ u \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & & & & & & & \vdots \\ 0 & \cdots & 0 & -k_N & 0 & 0 & 0 & (1+k_N\epsilon_N) & 0 \\ 0 & \cdots & 1 & 0 & 0 & 0 & 0 & c_N & 0 \\ 0 & \cdots & 0 & (1-k_N\epsilon_N) & 0 & 0 & 0 & k_N & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ \vdots \\ y_{N-1} \\ \hat{y}_{N-1} \\ x_{3N-1} \\ x_{3N} \\ x_{3N+1} \\ x_{3N+2} \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \vdots \\ x'_{3N+2} \\ y \\ \hat{y} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x'_1 \\ \vdots \\ y_N \\ \hat{y}_N \\ x_{3N+2} \\ u \end{bmatrix}$$

The Octave function *schurOneMlatticeDoublyPipelined2Abcd* returns the state variable description of a doubly-pipelined Schur one-multiplier lattice filter (including the all-pass filter Aap, Bap, Cap and Dap output matrixes).

The Octave script *schurOneMlatticeDoublyPipelined2Abcd_symbolic_test.m* creates a symbolic state variable description of a doubly-pipelined Schur one-multiplier lattice filter.

5.6.5 State variable description of an all-pass doubly-pipelined one-multiplier Schur lattice filter

Figure 5.13 shows only the all-pass part of a first order segment of the doubly-pipelined Schur one-multiplier lattice filter of Figure 5.12.

Figure 5.13: First order segment of an all-pass doubly pipelined Schur one-multiplier lattice filter.

Construction of a state variable description of the all-pass doubly-pipelined Schur one-multiplier lattice filter is summarised in Algorithm 5.7.

Algorithm 5.7 Construction of a state variable description of the all-pass doubly-pipelined Schur one-multiplier lattice filter.

Given $\{k_1, k_2, \ldots, k_N\}$ and $\{\epsilon_1, \epsilon_2, \ldots, \epsilon_N\}$:

$\hat{y}_0 = x_1$
$x'_1 = x_2$
for $n = 1, \ldots, N$ **do**
$x'_{2n} = -k_n \hat{y}_{n-1} + (1 + k_n \epsilon_n)\, x_{2n+2}$
$x'_{2n+1} = (1 - k_n \epsilon_n)\, \hat{y}_{n-1} + k_n x_{2n+2}$
$\hat{y}_n = x_{2n+1}$
end for
$x'_{2N+2} = u$
$\hat{y} = \hat{y}_N$

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

In Figure 5.14, 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 $\hat{y}_n$ the all-pass output of the n-th section.

Figure 5.14: State variable description of the Schur Scaled-Normalised lattice filter.

Construction of the state variable description is summarised in Algorithm 5.8 [d].

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:

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

Algorithm 5.8 Construction of a state variable description of the Scaled-Normalised Lattice.

$\hat{y}_0 = x_1$
$y_0 = \sigma_{10}^{(0)} x_1$
for $n = 1, \ldots, N-1$ **do**
  $x'_n = \sigma_{02}^{(n)} \hat{y}_{n-1} + \sigma_{00}^{(n)} x_{n+1}$
  $\hat{y}_n = \sigma_{22}^{(n)} \hat{y}_{n-1} + \sigma_{20}^{(n)} x_{n+1}$
  $y_n = \sigma_{11}^{(n)} y_{n-1} + \sigma_{10}^{(n)} x_{n+1}$
end for
$x'_N = \sigma_{02}^{(N)} \hat{y}_{N-1} + \sigma_{00}^{(N)} u$
$\hat{y} = \sigma_{22}^{(N)} \hat{y}_{N-1} + \sigma_{20}^{(N)} u$
$y = \sigma_{11}^{(N)} y_{N-1} + \sigma_{10}^{(N)} u$

$$\begin{bmatrix} \hat{y}_0 \\ y_0 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ \sigma_{10}^{(0)} & 0 & \cdots & & \vdots \\ 0 & 1 & & & \\ \vdots & & \ddots & & \vdots \\ 0 & & & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \hat{y}_1 \\ y_1 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix} = \begin{bmatrix} \sigma_{02}^{(1)} & 0 & \sigma_{00}^{(1)} & 0 & 0 & \cdots & 0 \\ \sigma_{22}^{(1)} & 0 & \sigma_{20}^{(1)} & 0 & & & \vdots \\ 0 & \sigma_{11}^{(1)} & \sigma_{10}^{(1)} & 0 & & & \\ 0 & 0 & 0 & 1 & & & \\ \vdots & & & & \ddots & & \vdots \\ 0 & & & & & 1 & 0 \\ 0 & \cdots & & & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_0 \\ y_0 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

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

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

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

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).

Figure 5.15: Transposed normalised-scaled lattice filter module (after *Parhi* [118, Fig. 12.24]).

5.7 Roundoff Noise Calculation in Schur Lattice Filters

In this section I rely on the state-variable analysis of round-off noise presented by *Roberts* and *Mullis* [195], [35], [36] and the transposed signal flow graph analysis of *Parhi* [118, Section 12.7]. *Markel* and *Gray* [94] show the results of an experimental comparison of the roundoff noise performance of direct-form, parallel and lattice filter implementations of elliptic filters. They conclude that "the direct form is inferior to all the other structures studied" and that "the normalized structure (having four multiplications and two additions per section), appears to have remarkably robust roundoff noise characteristics". *Kimura* and *Osada* [81] show that the the pipelined or, so-called, *symmetric* lattice shown in Section 5.6.3 has improved roundoff noise performance compared to the lattice of Section 5.6.2.

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. In the following I try to distinguish between pipelining a filter to reduce the latency of internal calculations and retiming a filter to determine the noise gain without necessarily maintaining the desired filter transfer function.

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 5.15 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 floating-point, 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, $\tilde{\Phi}_i(z)$. 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 5.16 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 ($\sigma^2 = (1 + ng)/12$):

Figure 5.16: Amplitude response of the 3rd order Butterworth filter implemented with a Schur normalised-scaled lattice structure.

```
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
```

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 5.17 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 5.18, 5.19 and 5.20 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 5.21 shows part of the state trajectories for the Schur lattice implementation of the Butterworth filter in response to a sine wave input.

Figure 5.17: 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 5.18: Internal states in the 3rd order Butterworth filter implemented in a normalised-scaled Schur lattice structure with a random noise input.

Figure 5.19: Internal states in the 3rd order Butterworth filter implemented in the globally optimised state variable structure with a random noise input.

Figure 5.20: Internal states in the 3rd order Butterworth filter implemented in a direct-form structure with a random noise input.

Figure 5.21: Internal states in the 3rd order Butterworth filter implemented with a Schur lattice structure with a sine wave input.

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

Parhi [118, 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 5.22. 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.

Figure 5.22: Slowed and retimed normalised-scaled lattice for the 3rd order Butterworth example.

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 floating-point 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[e]. 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 5.23.

Figure 5.23: Amplitude response of the 3rd order Butterworth filter implemented with a normalised-scaled lattice structure and 10-bit 2 signed-digit coefficients.

The response demonstrates a drawback of the normalised-scaled lattice filters: they are not *structurally loss-less* in the sense of Vaidyanathan et al. [180, 177] (see Appendix M.2). The normalised-scaled lattice does not preserve the all-pass characteristic when its coefficients are truncated.

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

Section 2.2 shows the frequency transformation of a state variable filter. Section 3.8 shows some results, given by *Mullis* and *Roberts* [35, Section III], concerning the noise gain of a frequency transformed state variable filter. *Koshita et al.* [227] 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, $H(F(z))$

[e] *Parhi* [118, 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.

The Octave script *tfp2schurNSlattice2Abcd_test.m* exercises *tfp2schurNSlattice2Abcd* with the 5th order elliptic low-pass to multiple stop-band filter example of Section 2.3 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).

	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	61	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.

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 5.24 shows module m of $N, \ldots, 1$ of the transposed graph of a one multiplier lattice filter.

Figure 5.24: Transposed one multiplier lattice filter module.

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:

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

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

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

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

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

The Octave function *schurOneMlatticeFilter* implements this scaling.

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

The multiplier and sign coefficients are

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

The scaling factors for each section are

```
p =   3.03862   0.34657   1.80947
```

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

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

The scaled Butterworth filter noise gain with floating-point 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.

Figure 5.25: Slowed and retimed single-multipler lattice for the 3rd order Butterworth example.

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

The one-multiplier lattice round-off noise variance can also be determined from the state variable representation if the signal flow graph is slowed and retimed. Figure 5.25 shows an example of slowing and retiming a 3-order Butterworh filter. Each intermediate node in the filter is now associated with a state. The additional states alter the overall transfer function but the round-off noise gains are unchanged[f].

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 5.25) in a single wide accumulator and ignores the states on the top edge
- For "**ABCD**" *svf* calculates both the Butterworth output and the all-pass output in wide accumulators and ignores the states on both the top and bottom edges

In both these cases, states x_1 and x_2 in Figure 5.25 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.

[f]Retiming is equivalent to a change of variables from z^{-1} to z^{-2} in the filter transfer function. *Parhi* [118, Chapter 4] calls this "slowing" because it doubles the digital sample rate. Retiming is distinct from pipelining because the latter does not change the transfer function except for the addition of an integer number of samples of group-delay.

When filter type *"schur"* is selected the round-off noise gains for floating-point 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 floating-point 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 5.26 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*.

5.7.3 Round-off noise of the pipelined one multiplier lattice filter

The Octave script *schurOneMlatticePipelinedFilter_test.m* demonstrates calculation of the round-off noise of the pipelined one-multiplier lattice filter from the state variable description, as shown in Section 3.4.

Figure 5.26: Amplitude responses of the slowed and retimed one multiplier lattice 3rd order Butterworth filter implemented in Schur lattice form with 10-bit 3-signed-digit truncated coefficients.

(a) In original form.

(b) After retiming.

Figure 5.27: Signal flow graph of a 4th order normalised-scaled Schur lattice filter with a denominator polynomial having coefficients only in z^{-2}.

5.8 Examples of pipelining Schur lattice filters

5.8.1 Pipelining a 4th 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* [118, Chapter 4] discusses formal methods for pipelining a signal flow graph. Figure 5.27a 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 5.27b 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 pipelining scheme increases the group delay by 1 sample for each second-order section.

Following the notation of Figure 5.14, the state variable equations for the filter shown in Figure 5.27b are:

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

Figure 5.28: Simulated amplitude response of the pipelined 4th order normalised-scaled Schur lattice filter.

$$
\begin{aligned}
x_{10}(k+1) &= x_{11}(k) \\
x_{11}(k+1) &= u(k) \\
y(k) &= \sigma_{11}^{(4)} x_8(k) + \sigma_{10}^{(4)} x_{10}(k) \\
y_{ap}(k) &= \sigma_{22}^{(4)} x_7(k) + \sigma_{20}^{(4)} u(k)
\end{aligned}
$$

The Octave script *schur_pipelined_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.* [11] 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 5.28 shows the simulated amplitude response of the pipelined Schur lattice filter.

5.8.2 Pipelining a 6th-order Schur one-multiplier lattice filter

Figure 5.29 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 pipelined filter has an additional $\frac{N}{2} - 1$ states.

Algorithm 5.9 shows a state variable description of the pipelined Schur one-multiplier lattice filter with coefficients in z^{-2}.

Similarly, Algorithm 5.10 shows a state variable description of the pipelined Schur one-multiplier all-pass lattice filter with coefficients in z^{-2}.

The Octave function *schurOneMR2lattice2Abcd*, exercised by the Octave script *schurOneMR2lattice2Abcd_test.m*, implements pipelining of a one-multiplier Schur lattice filter with a denominator polynomial having coefficients in z^{-2} only. The Octave script *schurSchurOneMR2lattice2Abcd_symbolic_test.m* creates a symbolic state variable description of the Schur one multiplier 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 pipelining.

(c) Signal flow graph of a 6th order one-multiplier Schur all-pass lattice filter with a denominator polynomial having coefficients in z^{-2} after pipelining.

Figure 5.29: Original and pipelined signal flow graphs of a 6th order one-multiplier Schur lattice filter with a denominator polynomial having coefficients in z^{-2} only. The number of delays around each loop of the signal flow graph is unchanged.

Algorithm 5.9 Construction of a state variable description of the pipelined Schur one-multiplier lattice filter with denominator coefficients in z^{-2} only.

Given N even, $\{k_2, k_4 \ldots, k_N\}$, $\{\epsilon_2, \epsilon_4, \ldots, \epsilon_N\}$ and $\{c_0, c_1, \ldots, c_N\}$:

$x'_1 = x_2$
$x'_2 = -k_2 x_1 + (1 + k_2\epsilon_2)\, x_5$
$x'_3 = (1 - k_2\epsilon_2)\, x_1 + k_2 x_5$
$x'_4 = c_0 x_1 + c_1 x_2 + c_2 x_5$
for $n = 2, \ldots, \frac{N}{2} - 1$ **do**
  $x'_{3n-1} = -k_{2n} x_{3n-3} + (1 + k_{2n}\epsilon_{2n})\, x_{3n+2}$
  $x'_{3n} = (1 - k_{2n}\epsilon_{2n})\, x_{3n-3} + k_{2n} x_{3n+2}$
  $x'_{3n+1} = x_{3n-2} + c_{2n-1} x_{3n-1} + c_{2n} x_{3n+2}$
end for
$x'_{3\frac{N}{2}-1} = -k_N x_{3\frac{N}{2}-3} + (1 + k_N\epsilon_N)\, u$
$y = x_{3\frac{N}{2}-2} + c_{N-1} x_{3\frac{N}{2}-1} + c_N u$
$\hat{y} = (1 - k_N\epsilon_N)\, x_{3\frac{N}{2}-3} + k_N u$

Algorithm 5.10 Construction of a state variable description of the pipelined Schur one-multiplier all-pass lattice filter with denominator coefficients in z^{-2} only.

Given N even, $\{k_2, k_4 \ldots, k_N\}$ and $\{\epsilon_2, \epsilon_4, \ldots, \epsilon_N\}$:

$x'_1 = x_2$
for $n = 1, \ldots, \frac{N}{2} - 1$ **do**
$\quad x'_{2n} = -k_{2n} x_{2n-1} + (1 + k_{2n}\epsilon_{2n})\, x_{2n+2}$
$\quad x'_{2n+1} = (1 - k_{2n}\epsilon_{2n})\, x_{2n-1} + k_{2n} x_{2n+2}$
end for
$x'_N = -k_N x_{N-1} + (1 + k_N \epsilon_N)\, u$
$\hat{y} = (1 - k_N \epsilon_N)\, x_{N-1} + k_N u$

5.8.3 Frequency transformations of pipelined Schur lattice filters

An implementation of a rational transfer function having demominator coefficients only in powers of z^{-2} can be 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 and uses the WISE method of *Tarczynski et al.* [11] to ensure that the filter is stable. 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 5.30 shows the response of the low-pass prototype filter. Figure 5.31 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 5.32 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 band-pass filter have coefficients only in powers of z^{-2}.

Figure 5.30: Amplitude response of a low-pass filter prototype having denominator polynomial coefficients in z^{-2}.

Figure 5.31: Amplitude response of a band-pass filter with frequency transformation [0.1 0.25].

Figure 5.32: Amplitude response of a band-pass filter with frequency transformation [0.2 0.3].

5.9 Examples of the coefficient sensitivity of the responses of direct-form and Schur lattice filters

The Octave script *filter_coefficient_sensitivity_test.m* shows the coefficient sensitivities of a low-pass elliptic IIR filter implemented in direct-form, as a tapped Schur one-multiplier lattice and as the combination of parallel all-pass Schur one-multiplier lattices[g]. In addition it shows the coefficient sensitivities of a symmetric direct-form FIR filter with a similar amplitude response. The numerator and denominator transfer function polynomials of the direct-form IIR filter implementation are:

```
n0 = [   0.0209894754,  -0.0159055715,   0.0197510818,   0.0197510818, ...
        -0.0159055715,   0.0209894754 ];


d0 = [   1.0000000000,  -3.4893036833,   5.5630748041,  -4.8624991170, ...
         2.3227454901,  -0.4843475226 ];
```

The lattice and tap coefficients of the Schur one-multiplier lattice implementation of the elliptic filter are:

```
k0 = [  -0.7730675653,   0.9593779044,  -0.8814037135,   0.8266314536, ...
        -0.4843475226 ];


epsilon0 = [   1.0000000000,   1.0000000000,   1.0000000000,   1.0000000000, ...
               1.0000000000 ];


c0 = [   0.0560669836,   0.0952359202,   0.6315001836,   0.0850219299, ...
         0.1111855074,   0.0209894754 ];
```

The lattice coefficients of the parallel all-pass Schur one-multiplier lattice implementation are:

```
A1k0 = [  -0.8141343760,   0.7172567234 ];


A2k0 = [  -0.7526448254,   0.9374325533,  -0.6752777726 ];
```

The coefficients of the direct-form low-pass FIR filter are:

```
h0 = [  -0.0079112781,  -0.0052300593,  -0.0019510562,   0.0044878278, ...
         0.0110827325,   0.0138738078,   0.0105082134,   0.0022702960, ...
        -0.0060690041,  -0.0089572117,  -0.0038556818,   0.0064382725, ...
         0.0149008558,   0.0146536481,   0.0040226943,  -0.0112384944, ...
        -0.0207616164,  -0.0162583730,   0.0019781888,   0.0234109567, ...
         0.0325551366,   0.0189845973,  -0.0138994148,  -0.0474555116, ...
        -0.0558509090,  -0.0203011715,   0.0585188908,   0.1573068461, ...
         0.2391961447,   0.2709280488,   0.2391961447,   0.1573068461, ...
         0.0585188908,  -0.0203011715,  -0.0558509090,  -0.0474555116, ...
        -0.0138994148,   0.0189845973,   0.0325551366,   0.0234109567, ...
         0.0019781888,  -0.0162583730,  -0.0207616164,  -0.0112384944, ...
         0.0040226943,   0.0146536481,   0.0149008558,   0.0064382725, ...
        -0.0038556818,  -0.0089572117,  -0.0060690041,   0.0022702960, ...
         0.0105082134,   0.0138738078,   0.0110827325,   0.0044878278, ...
        -0.0019510562,  -0.0052300593,  -0.0079112781 ]';
```

Figure 5.33 shows the amplitude and group-delay responses of the IIR filter and Figure 5.34 shows the amplitude response of the FIR filter. Figure 5.35 shows the squared-amplitude and group-delay coefficient sensitivity responses of the direct-form low-pass elliptic IIR filter. Figure 5.36 shows the squared-amplitude and group-delay coefficient sensitivity responses of the tapped Schur one-multiplier lattice low-pass elliptic IIR filter. Figure 5.37 shows the squared-amplitude and group-delay coefficient sensitivity responses of the parallel all-pass Schur one-multiplier lattice low-pass elliptic IIR filter. Finally, for comparison, Figure 5.38 shows the squared-amplitude coefficient sensitivity responses of the direct-form low-pass FIR filter.

[g] See Appendix M.2

Figure 5.33: Amplitude and group-delay responses of a low-pass elliptic IIR filter.

Figure 5.34: Amplitude response of a direct-form low-pass FIR filter.

Figure 5.35: Squared amplitude and group-delay coefficient sensitivity responses of a direct-form low-pass elliptic IIR filter.

Figure 5.36: Squared amplitude and group-delay coefficient sensitivity responses of a tapped Schur one-multiplier lattice low-pass elliptic IIR filter.

Figure 5.37: Squared amplitude and group-delay coefficient sensitivity responses of a parallel all-pass Schur one-multiplier lattice low-pass elliptic IIR filter.

Figure 5.38: Squared amplitude coefficient sensitivity response of a direct-form low-pass FIR filter.

5.10 Summary

Given a transfer function $H(z) = N_N(z)/D_N(z)$ the design procedure for the normalised-scaled lattice filter is:

1. Compute the Schur polynomials from the denominator $D_N(z)$
2. Compute the parameters k_i for $i = N, N-1, \cdots 1$
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 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* [195, Section 10.4] for decomposing an arbitrary transfer function into 2×2 block diagonal coordinate rotations that can be "*manipulated to achieve either high-speed, parallel computation using a sparsely connected array of processor modules or low-speed single processor realisations*"[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 $UU^* = I$ then $U \in \mathcal{U}$, where $*$ 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 θ. 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 θ

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* [180, 177] 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 μ_i^2 are the *second order modes* or eigenvalues of KW. For comparison, a globally optimised minimum noise filter with equal word-lengths has

$$\begin{aligned} g_{min} &= \frac{1}{n}\sum_{i=1}^{n} K_{ii}W_{ii} \\ &= \left[\frac{1}{n}\sum_{i=1}^{n} \mu_i\right]^2 \end{aligned}$$

Thus the difference between g_{orth} and g_{min} is the difference between a second moment and the square of a first moment, which is the variance. The two are only equal if the μ_i are all equal. Recall that the second order modes of the globally optimised filter are invariant under a frequency transformation.

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 γ is a scalar. T_0 can be found as the product of a series of rotations (each of which are members of $\mathcal{O}(n)$) that zero out the leading elements of Γ

$$\begin{bmatrix} \gamma_i & \gamma_{i+1} \end{bmatrix} \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix} = \begin{bmatrix} 0 & \gamma'_{i+1} \end{bmatrix}$$

where θ_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 [195, 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* [195, pp. 460-461] describe the following procedure for factoring a two-input, two-output all-pass filter, F, into the product of 2×2 block diagonal coordinate rotations of the form

$$\begin{bmatrix} T_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathcal{O}(n+1)$$

that preserve G_0'[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 Γ. 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 ± 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* [59, 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* [59, 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* [118, 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 2×2 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 2×2 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 2×2 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 2×2 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 2×2 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* [195, 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 [74, 251, 48] in order to improve the roundoff noise performance of the filter with a small increase in complexity. *Mullis* and *Roberts* [37] 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* [48] . *Lu* and *Hinamoto* [251] 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* [48] .

7.1 Problem formulation

Figure 7.1 [251, Figure 2] shows a block diagram of a state variable digital filter with feedback and feed-forward of the state quantisation error e. The state variable quantiser is Q, the feedback error transfer function is δ and the feed-forward error transfer function is η.

As shown in Section 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 [48] 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* [251, 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 [48, 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, $\{\mu_i^2\}$ of $K_0 W_0$, and $\{\rho_i^2\}$ of $K_0 P_0$, are invariant under a similarity transformation, T, and are positive. In Section 6.3, the eigenvalues $\{\mu_i^2\}$ are referred to as the *second-order modes* of $H(z)$. Williamson calls $\{\rho_i^2\}$ the *residue modes*. Further, he calls a realisation $\{A_0, B_0, C_0, D\}$ *input balanced* if $K_0 = I$ and $W_0 = M^2 = \operatorname{diag}\{\mu_1^2, \ldots, \mu_N^2\}$, 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 [48, Section V] considers the "problem of finding the optimal transformation, T, and the optimal integer residue feedback matrixes δ_T and η_T, which together minimise the output round-off noise". He proves the theorem shown as Algorithm 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* [48, Theorem 5.2]).

Let M define the second-order modes of a stable Nth order filter, $H(z)$, with an input-balanced realisation, $\{A_0, B_0, C_0, D\}$. Consider the finite word length implementation $\{A_T = T^{-1} A_0 T, B_T = T^{-1} B_0, C_T = T^\top C_0, D\}$. Then subject to the l_2-norm scaling constraint, the identity state residue correction, $\delta_T = I$, and the zero output residue correction, $\eta_T = 0$, the minimum noise filter is defined by

$$\begin{aligned} T &= R_1 \Pi R_0^\top \\ \Pi &= \operatorname{diag}\{\pi_1, \ldots, \pi_N\} \end{aligned}$$

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

$$\begin{aligned} \operatorname{diag}\{R_0 \Pi^{-1} R_0^\top\} &= I \\ R_1^\top P_0 R_1 \quad &\text{is} \quad \text{diagonal} \end{aligned}$$

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* [48, 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* [21]

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 λ_i are the Lagrange multipliers (or *dual variables*) for those constraints. The *Karush-Kuhn-Tucker* conditions for the minimum are:

$$\nabla_x \left\{ \mathcal{E}(x) - \sum_{i \in \mathcal{A}(x)} \lambda_i g_i(x) \right\} = 0 \tag{8.3}$$

$$g_i(x) \geq 0 \tag{8.4}$$

$$\lambda_i \geq 0 \tag{8.5}$$

$$\langle \lambda_i, g_i(x) \rangle = 0 \tag{8.6}$$

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

The second-order *Taylor expansion* of $\mathcal{E}(x)$ at x^k is:

$$\mathcal{E}(x) \approx \mathcal{E}(x^k) + \nabla_x \mathcal{E}(x^k)(x - x^k) + \frac{1}{2}(x - x^k)^\top \nabla_x^2 \mathcal{E}(x^k)(x - x^k) \tag{8.7}$$

Express the *Karush-Kuhn-Tucker* conditions (Equations 8.3 and 8.4) for a minimum at x^k in terms of the gradient of the second-order expansion of the error and linearised constraints as (see Appendix K.1):

$$\nabla_x \mathcal{E}(x^k) + \nabla_x^2 \mathcal{E}(x^k)(x - x^k) - \sum_{i \in \mathcal{A}(x)} \lambda_i \nabla_x g_i(x^k) = 0 \tag{8.8}$$

$$g_i(x^k) + \nabla_x g_i(x^k)(x - x^k) \geq 0$$

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

$$\begin{aligned} \mathcal{W}_k d^k - \mathcal{B}_k \lambda^k &= -\nabla_x \mathcal{E}(x^k) \\ -\mathcal{B}_k^\top d^k &= g(x^k) \end{aligned} \tag{8.9}$$

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 λ^k and the direction vector d^k:

$$\begin{aligned} \lambda^k &= -\left(\mathcal{B}_k^\top \mathcal{W}_k^{-1} \mathcal{B}_k\right)^{-1} \left[g(x^k) - \mathcal{B}_k^\top \mathcal{W}_k^{-1} \nabla_x \mathcal{E}(x^k)\right] \\ d^k &= -\mathcal{W}_k^{-1} \left[\nabla_x \mathcal{E}(x^k) - \mathcal{B}_k \lambda^k\right] \end{aligned}$$

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. τ^k is found by a line search for the minimum of the Lagrangian function

$$\mathcal{L}(\tau^k) = \mathcal{E}(x^k + \tau^k d^k) - \sum_{i \in \mathcal{A}(x^k)} \lambda_i^k g_i(x^k + \tau^k d^k)$$

subject to the constraints. The iteration finishes when the *Karush-Kuhn-Tucker* conditions are satisfied.

8.1.2 Choice of active constraints

Selesnick, Lang and Burrus [91] 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

$$\nabla_x \mathcal{E}_A(x, f) - \sum_{i=1}^{q} \lambda_i \nabla_x A(x, f) + \sum_{i=q+1}^{r} \lambda_i \nabla_x A(x, f) = 0$$

$$\begin{aligned} A(x, f_i) &= L(f_i) \quad \text{for} \quad 1 \le i \le q \\ A(x, f_i) &= U(f_i) \quad \text{for} \quad q+1 \le i \le r \end{aligned}$$

Algorithm 8.1 Exchange algorithm for multiband FIR filter of *Selesnick, Lang and Burrus* [92, p.498].

1. *Initialisation*: Initialise the constraint sets R and S to the empty set.
2. *Minimisation with Equality Constraints*: Calculate the Lagrange multipliers associated with the filter that minimizes $\mathcal{E}_A(\omega)$ subject to the equality constraints $A(\omega_i) = L(\omega_i)$ for $\omega \in S_L$, and $A(\omega_i) = U(\omega_i)$ for $\omega \in S_U$.
3. *Karush-Kuhn-Tucker Conditions*: If there is a constraint set frequency ω_i, for which the Lagrange multiplier λ_i is negative, then remove from the active constraint set, S, the frequency corresponding to the most negative multiplier, and go back to step 2. Otherwise go on to step 4.
4. *Check for Violation over R*: Calculate the new filter response, $A(\omega_i)$ for frequencies in the alternate constraint set, R. If $A(\omega_i) < L(\omega_i)$ or $A(\omega_i) > H(\omega_i)$ for some $\omega_i \in R$, then remove the frequency corresponding to the greatest violation from R, append that frequency to S, and go back to step 2.
5. *Multiple Exchange of Constraint Set*: Overwrite the previous constraint set, R, with the current constraint set, S. Set the current constraint set S equal to $S_L \cup S_U$, where S_L is the set of frequency points ω, in $[0, \pi]$ satisfying both $A'(\omega_i) = 0$ and $A(\omega_i) \leq L(\omega_i)$ (ie: troughs failing the constraint) and where S_U, is the set of frequency points ω, in $[0, \pi]$ satisfying both $A'(\omega_i) = 0$ and $A(\omega_i) \geq U(\omega_i)$ (ie: peaks failing the constraint).
6. *Check for Convergence*: If $A(\omega) \geq L(\omega) - \epsilon$ for all frequency points in S_L and if $A(\omega) \leq U(\omega) + \epsilon$ for all frequency points in S_U, then convergence has been achieved. Otherwise, go back to step 2.

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

$$\begin{aligned} A(x, f_i) &\geq L(f_i) \quad \text{for} \quad 1 \leq i \leq q \\ A(x, f_i) &\leq U(f_i) \quad \text{for} \quad q+1 \leq i \leq r \end{aligned}$$

Selesnick et al. [92, p.498] modify the algorithm of *Selesnick et al.* [91] 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. [91, Section IV.A] justify the removal of the most negative Lagrange multiplier as follows:

> The constraints are on the values of $A(\omega_i)$ for the frequency points ω_i in a constraint set. On each iteration, the constraint set is updated so that at convergence, the only frequency points at which equality constraints are imposed are those where $A(\omega)$ touches the constraint. The equality constrained problem is solved with Lagrange multipliers. The algorithm below associates an inequality-constrained problem with each equality constrained one. According to the Kuhn-Tucker conditions, the solution to the *equality* constrained problem solves the corresponding *inequality* constrained problem if all the Lagrange multipliers are non-negative (where the signs of the multipliers are defined appropriately). If on some iteration a multiplier is negative, then the solution to the equality constrained problem does not solve the corresponding inequality constrained one. For this reason, before the constraint set is updated in the algorithm described below, constraints corresponding to negative multipliers (when they appear) are sequentially dropped from the constraint set. In this way, an inequality constrained problem is solved on each iteration, albeit over a smaller constraint set. It turns out that in the special case of a lowpass filter design considered here, this simple iterative technique converges in practice.

The Octave function files *cl2lp.m* and *cl2bp.m*, written by *Selesnick* [223], are, respectively, low-pass and band-pass implementations by *Selesnick* of the PCLS filter design method.

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* [100, 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 Δ_{A_p} is the pass band ripple for the desired gain A_D and Δ_{A_S} is the stop band ripple. After linearising about x^k these constraints become

$$A_D - A\left(x^k\right) - \nabla_x A\left(x^k\right)^\top \left(x - x^k\right) \geq 0$$
$$A\left(x^k\right) + \nabla_x A\left(x^k\right)^\top \left(x - x^k\right) - [A_D - \Delta_{A_P}] \geq 0$$
$$\Delta_{A_S} - A\left(x^k\right) - \nabla_x A\left(x^k\right)^\top \left(x - x^k\right) \geq 0$$

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

$$[T_D + \Delta_D] - \hat{T}(x) \geq 0$$
$$\hat{T}(x) - [T_D - \Delta_T] \geq 0$$

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

$$[T_D + \Delta_T] - T\left(x^k\right) - \nabla_x T\left(x^k\right)^\top \left(x - x^k\right) \geq 0$$
$$T\left(x^k\right) + \nabla_x T\left(x^k\right)^\top \left(x - x^k\right) - [T_D - \Delta_T] \geq 0$$

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* [137] 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.* [11] 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* [139] describes a method for finding successive coefficient vectors based on Rouché's theorem. *Dumitrescu and Niemistø* [21] 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* [250] express the denominator polynomial as a product of second-order sections and derive a linear inequality stability constraint on the denominator coefficients. *Lu* [258] 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 ω_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.* [11], propose minimising the error with the following, so-called, *WISE* barrier function:

$$(1-\lambda)\,\mathcal{E}_H + \lambda \sum_{t=T+1}^{T+M} f^2(t) \tag{8.10}$$

where $\mathcal{E}_H$ is the filter response error defined in Equation 1, λ, T and M are suitable constants and $f(t)$ is the impulse response of the filter $F(z) = \frac{1}{D(z)}$. *Tarczynski et al.* provide heuristics for selecting λ, T and M. Typically, $\lambda \in \left[10^{-10}, 10^{-3}\right]$, $T \in [100, 500]$ and $M = RN_d$. The barrier function (the second part of Equation 8.10) is intended to be small when the filter is stable and increase rapidly otherwise. *Roberts and Mullis* [195, 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:

$$A = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -d_{N_d} & -d_{N_d-1} & \cdots & -d_1 \end{bmatrix}$$

$$
\begin{aligned}
B &= \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix}^{\top} \\
C &= \begin{bmatrix} -d_{N_d} & \cdots & -d_1 \end{bmatrix} \\
D &= 1
\end{aligned}
$$

Alternatively, for the filter transfer function, $H(z)$, given by Equation 8.1, *Tarczynski et al.* [11, 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 ith 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 [59, 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* [59, 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 [11] with $nN = 24$, $nD = 2$ and $R = 2$ by unconstrained minimisation of Equation 8.10 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.

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.

Surma-aho and Saramäki method of designing an IIR initial filter

Appendix M.4.3 describes the method of *Surma-aho* and *Saramäki* [116] 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.

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 8.3.2 and Section 8.3.3 assume a zero at $z = 1$ and design a correction filter that provides the desired combined response.

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. Instead I have written my own Octave SQP solver, *iir_sqp_mmse*, based on the BFGS update algorithm and, in the PCLS solver, *iir_slb*, the constraints are calculated at the same time as the error. Other open-source Quadratic Programming (QP) solvers exist. For example, the *Proximal Interior-Point Quadratic Programming Solver*(PIQP) of *Schwan et al.* [222].

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.* [11]. 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 *Goldfarb-Idnani* algorithm [41] 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 ω_a etc. of linear constraints at the corresponding index into A_{du} or A_{dl} etc. Function *iir_sqp_mmse ()* calls *sqp_bfgs* to minimise the objective function *iir_sqp_mmse_fx ()* with linear constraints calculated by *iir_sqp_mmse_gx ()*. These functions calculate the squared-error in the response amplitude and delay and linear constraints by calls to *iirE ()*. That function calculates the error with trapezoidal integration. If the frequency bands are not contiguous then zeros in the weight vector can make frequency transition bands. 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
  ```

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* [92, 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 [105, 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.* [11]. The coefficients of the numerator and denominator polynomials of the filter are:

```
N0 = [   0.0055318056,   0.0168959019,   0.0074747069,  -0.0015217632, ...
        -0.0019752367,   0.0069417252,   0.0033977968,  -0.0102845651, ...
        -0.0055115952,   0.0171242060,   0.0104429077,  -0.0353411015, ...
        -0.0284871651,   0.1348459465,   0.4155092437,   0.6323652597, ...
         0.6374870441,   0.4464420276,   0.1788987531,  -0.0679345776, ...
         0.2506266798,  -0.3305090495,   0.2959985521,  -0.1721580560, ...
         0.0604525821 ]';


D0 = [   1.0000000000,   1.1781972853,   0.2453690259 ]';
```

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 [100, 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
ftol=0.001 % Tolerance on relative coefficient update size
ctol=0.0001 % 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=0.008 % 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=4 % Pass band group delay weight(PCLS pass)
fas=0.3 % Stop band amplitude response edge
dBas=33 % Stop band minimum attenuation
Was=0.5 % 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" [100, 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.

Figure 8.7: Deczky Example 3, response for initial coefficients.

Figure 8.8: Deczky Example 3, pole-zero plot for initial coefficients.

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.

Figure 8.9: Deczky Example 3, MMSE optimised response after pass 1.

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 8.10: 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.

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.

Figure 8.13: Deczky Example 3, PCLS optimised response.

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 = [   0.0029049245, ...
         0.9990328236,   1.0096808743,   1.3609184214,   1.8056550955, ...
         2.3289105439, ...
         2.0373440160,   2.7480604712,   1.7464023490,   0.7076274551, ...
         0.0064789670, ...
         0.5044781156,   0.5955675051,   0.6436015203, ...
         0.3467515178,   1.0704877022,   1.4195190836 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0029049245,  -0.0120923832,   0.0126342080,  -0.0048234121, ...
         0.0152963684,  -0.0119358474,  -0.0339340201,   0.0044722937, ...
         0.0974146375,   0.1245108056,   0.0968059419 ]';
```

and

```
D1 = [   1.0000000000,  -1.7142706221,   1.8605194627,  -1.3350803612, ...
         0.6606998730,  -0.2171635413,   0.0373921789 ]';
```

Figure 8.14: Deczky Example 3, PCLS optimised passband response.

Figure 8.15: Deczky Example 3, PCLS optimised pole-zero plot.

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
ftol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-05 % Tolerance on constraints
fap=0.15 % Pass band amplitude response edge
dBap=0.3 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
ftp=0.2 % Pass band group delay response edge
tp=9 % Nominal filter group delay
tpr=0.04 % 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=40 % Stop band minimum attenuation
Was=0.9 % 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.

Figure 8.16: Deczky Example 3a PCLS optimised response.

Figure 8.17: Deczky Example 3a PCLS optimised passband response.

Figure 8.18: Deczky Example 3a PCLS optimised pole-zero plot.

The PCLS optimised filter vector is, in gain-zero-pole form:

```
Ud1=0,Vd1=0,Md1=10,Qd1=6,Rd1=1
d1 = [   0.0021159040, ...
         0.9689010818,   0.9756769415,   0.9918356657,   1.9827298306, ...
         2.6361494320, ...
         2.8280832061,   2.2957098753,   1.9558871116,   0.7698710804, ...
         0.0581312824, ...
         0.4731004194,   0.5818286016,   0.7114940800, ...
         0.3770342627,   1.1110410306,   1.5000418771 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0021159040,  -0.0089464826,   0.0041794009,   0.0160163331, ...
        -0.0020314230,  -0.0383520887,  -0.0118420299,   0.0891536923, ...
         0.1635990629,   0.1320542317,   0.0508175225 ]';
```

and

```
D1 = [   1.0000000000,  -1.4966872659,   1.6632694658,  -1.2223868355, ...
         0.6319834923,  -0.2168875866,   0.0383565362 ]';
```

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=800 % Frequency points across the band
ftol=0.001 % Tolerance on relative coefficient update size
ctol=0.0001 % 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=35 % Stop band minimum attenuation
Was=0.5 % Stop band amplitude weight
Ux0=0 % Number of real zeros
Vx0=0 % Number of real poles
Mx0=12 % 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 8.19: Deczky Example 1, response for initial coefficients.

Figure 8.20: Deczky Example 1, MMSE 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:

Figure 8.21: Deczky Example 1, MMSE optimised pole-zero plot.

```
Ud1=0,Vd1=0,Md1=12,Qd1=6,Rd1=1
d1 = [   0.0179121417, ...
         0.7891185928,   0.8129889075,   0.8848534238,   0.9777075513, ...
         1.6932356904,   1.7803497262, ...
         2.4167367454,   2.9158576326,   2.0606436887,   1.9003781715, ...
         1.0753633271,   0.3603977268, ...
         0.2774417366,   0.5965852491,   0.9323461688, ...
         0.7714032623,   1.6315434364,   1.7227553771 ]';
```

and the corresponding transfer function numerator and denominator polynomials are, respectively:

```
N1 = [   0.0179121417,  -0.0127218471,  -0.0006447757,   0.0127863896, ...
         0.0111942075,  -0.0380807532,  -0.0165110512,   0.2071653315, ...
         0.5131071510,   0.6378566092,   0.4839405381,   0.2210127103, ...
         0.0501426624 ]';
```

and

```
D1 = [   1.0000000000,  -0.0431096315,   1.1814972869,  -0.3047961210, ...
         0.3402511998,  -0.1104979564,   0.0238145835 ]';
```

Figure 8.22: Deczky Example 1, PCLS optimised response.

Figure 8.23: Deczky Example 1, PCLS optimised passband response.

d1(pcls):fap=0.25,dBap=1,Wap=1,fas=0.3,dBas=35,Was=0.5,ftp=0.25,tp=8,tpr=2,Wtp=0.01

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
ftol_wise=1e-07 % Tolerance on WISE relative coef. update
ftol_mmse=1e-05 % Tolerance on MMSE relative coef. update
ftol_pcls=0.0001 % Tolerance on PCLS relative coef. update
ctol=1e-05 % Tolerance on constraints
fap=0.1 % Pass band amplitude response edge
dBap=0.2 % Pass band amplitude peak-to-peak ripple
Wap=2 % 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=1 % Pass band group delay weight
fas=0.25 % Stop band amplitude response edge
dBas=40 % Stop band minimum attenuation
Was=1 % 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 = [   0.0153457152, ...
         0.9989828454,   1.0037651543,   1.0093064431,   1.0124454174, ...
         1.6405345470, ...
         1.6398817972,   2.0856919352,   2.4667860496,   2.9043249511, ...
         0.3255881802, ...
         0.2796615191,   0.4416637044,   0.5621645760, ...
         0.0100662016,   1.2833811358,   1.8829096155 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0153457152,   0.0239727125,  -0.0084149620,  -0.0669407798, ...
        -0.0748171359,   0.0211353443,   0.1769581600,   0.2812314379, ...
         0.2607513077,   0.1488097529,   0.0433638182 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,  -0.4644470781,   0.0000000000, ...
         0.4498082075,   0.0000000000,  -0.2418712348,   0.0000000000, ...
         0.1014511597,   0.0000000000,  -0.0354006140,   0.0000000000, ...
         0.0048214292 ]';
```

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
ftol_mmse=1e-05 % Tolerance on MMSE relative coef. update
ftol_pcls=0.001 % Tolerance on PCLS relative coef. update
ctol=1e-05 % Tolerance on constraints
dmax=0.05 % Maximum coefficient step size
rho=0.999 % Maximum pole radius
fap=0.1 % Pass band amplitude response edge
dBap=0.3 % 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=0.04 % Pass band group delay peak-to-peak ripple
Wtp=0.5 % Pass band group delay weight
fas=0.25 % Stop band amplitude response edge
dBas=43 % Stop band minimum attenuation
Was=2 % 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 = [   0.0003251757, ...
         1.0009625006,   1.0221507504,   1.0371618052,   1.0507308990, ...
         1.6702230075,   4.4329820684, ...
         1.6715286239,   2.1322099948,   2.5438615538,   2.9810923211, ...
         0.3264020203,   3.1239694194, ...
         0.3249618565,   0.4418906613,   0.5698935795, ...
         0.2526404775,   1.2070071406,   1.7841784903 ]';
```

The numerator and denominator polynomials of the PCLS optimised filter are:

```
N1 = [   0.0003251757,   0.0035051655,   0.0117823128,   0.0094162077, ...
        -0.0195513288,  -0.0509747745,  -0.0314876361,   0.0586495930, ...
         0.1682300940,   0.2180108778,   0.1770027621,   0.0893656483, ...
         0.0221615664 ]';


D1 = [   1.0000000000,   0.0000000000,  -0.7023888718,   0.0000000000, ...
         0.5957432039,   0.0000000000,  -0.3422155797,   0.0000000000, ...
         0.1449315610,   0.0000000000,  -0.0457169207,   0.0000000000, ...
         0.0066970255 ]';
```

Figure 8.32: Alternative PCLS optimised, low-pass, decimator, R=2, filter response.

Figure 8.33: Alternative PCLS optimised, low-pass, decimator, R=2, filter pole-zero plot.

An interpolated low-pass R=2 decimator

Lyons [209] 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
ftol_wise=1e-07 % Tolerance on WISE coefficient update
ftol_mmse=1e-05 % Tolerance on MMSE coefficient update
ftol_pcls=0.0001 % Tolerance on PCLS coefficient update
ctol=1e-05 % Tolerance on constraints
fap=0.09 % Pass band amplitude response edge
dBap=0.4 % 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=2 % 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 = [   0.0132507544, ...
         0.9248609270,   0.9646116133,   0.9655144426,   0.9714435877, ...
         0.9872582516,   1.5246726191, ...
         1.5856149100,   2.2100979062,   2.9319355309,   2.5433224842, ...
         1.1724583669,   0.2731441873, ...
         0.4413542474,   0.5623722793,   0.6698489524, ...
         0.3523574521,   1.1369876110,   1.6695416537 ]';
```

The numerator and denominator polynomials of the PCLS optimised un-interpolated filter are:

```
N1 = [   0.0132507544,   0.0128590240,  -0.0120247481,  -0.0323077894, ...
        -0.0275491776,  -0.0065725288,   0.0317963396,   0.0835413322, ...
         0.1195822159,   0.1228291860,   0.1009964114,   0.0619602504, ...
         0.0210216068 ];

D1 = [   1.0000000000,   0.0000000000,  -1.1691660038,   0.0000000000, ...
         1.1795672634,   0.0000000000,  -0.8187435647,   0.0000000000, ...
         0.4198901744,   0.0000000000,  -0.1507506069,   0.0000000000, ...
         0.0276424286 ];
```

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, 0.0935 , stop-band edge, 0.179 , and interpolation factor, P. The coefficients of the FIR anti-aliasing filter are:

```
b = [  -0.0122665130,  -0.0224862537,   0.0104672251,   0.1148819738, ...
        0.2463346485,   0.3074634834,   0.2463346485,   0.1148819738, ...
        0.0104672251,  -0.0224862537,  -0.0122665130 ];
```

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.

8.2.6 Band-pass R=2 decimation filter

The Octave script *iir_sqp_slb_bandpass_R2_test.m* implements an $R = 2$ decimation band-pass filter. The *sqp* loop is implemented by the Octave function *iir_slb*. The filter specification is:

```
n=500 % Frequency points across the band
ftol=0.001 % 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=0.7 % Pass band amplitude peak-to-peak ripple
dBas=34 % Stop band amplitude peak-to-peak ripple
Wasl=2 % Lower stop band weight
Wap=1 % Pass band weight
Wasu=2 % Upper stop band weight
ftpl=0.1 % Pass band group delay response lower edge
ftpu=0.2 % Pass band group delay response upper edge
tp=16 % Nominal filter group delay
tpr=0.08 % Pass band group delay peak-to-peak ripple
Wtp=0.5 % Pass band group delay weight
U=2 % Number of real zeros
V=0 % Number of real poles
M=18 % Number of complex zeros
Q=10 % Number of complex poles
R=2 % Denominator polynomial decimation factor
```

The 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 of $x0$ 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

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 8.39: IIR band-pass R=2 decimation filter, response of the initial filter.

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

The response of the PCLS optimised 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 = [   0.0134647981, ...
        -0.9929301330,   0.9965885338, ...
         0.9953548381,   0.9959209634,   0.9964131437,   0.9972019717, ...
         0.9972143000,   0.9981543971,   0.9982991693,   1.3182284262, ...
         1.3393860390, ...
         1.7678651956,   1.9835223382,   2.2361536721,   2.4917081035, ...
         1.5985769610,   0.2832877639,   2.6830085236,   1.0684895812, ...
         0.7433712831, ...
         0.6173155564,   0.6372388191,   0.6621990030,   0.7420642570, ...
         0.8574690708, ...
         1.8224245999,   2.3666117616,   1.3413177624,   2.7202016364, ...
         0.9869729725 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0134647981,   0.0093004749,   0.0152707492,   0.0131542601, ...
         0.0343778467,   0.0309994584,   0.0230693329,  -0.0039462668, ...
        -0.0020928898,  -0.0077387408,  -0.0384148526,  -0.0917402027, ...
        -0.0993293254,  -0.0079764270,   0.1126065389,   0.1704330531, ...
         0.0912920938,  -0.0265169717,  -0.1061551592,  -0.0850334100, ...
        -0.0397898161 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,   1.3256635986,   0.0000000000, ...
         1.5192420671,   0.0000000000,   1.6305448996,   0.0000000000, ...
         1.6074037683,   0.0000000000,   1.2729263064,   0.0000000000, ...
         0.9223931613,   0.0000000000,   0.5341167488,   0.0000000000, ...
         0.2710986450,   0.0000000000,   0.0971371622,   0.0000000000, ...
         0.0274736202 ]';
```

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.

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 R=2 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.

8.3 SQP PCLS design of a low-pass R=2 filter expressed in *gain-zero-pole* format with *SQP*

The Octave script *iir_sqp_slb_lowpass_R2_test.m* designs an $R = 2$ lowpass filter. After some experimentation, the 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
ftol_wise=1e-07 % Tolerance on WISE relative coef. update
ftol_mmse=1e-05 % Tolerance on MMSE relative coef. update
ftol_pcls=0.0001 % Tolerance on PCLS relative coef. update
ctol=0.0001 % Tolerance on constraints
fap=0.1 % Pass band amplitude response edge
dBap=0.06 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
Wat=0 % Transition band weight
fas=0.15 % Stop band amplitude response edge
dBas=53 % Stop band minimum attenuation
Was=1 % Stop band amplitude weight
```

The initial filter was calculated by optimising a "guess" filter with the WISE method of *Tarczynski et al.* described in Section 8.1.5. The response of the initial filter is shown in Figure 8.47. The response of the filter after MMSE and PCLS SQP optimimisation is shown in Figure 8.48. The corresponding pole-zero plot is shown in Figure 9.23. The estimate of the optimised filter vector, in the gain, zeros and poles form of Equation 8.2, is:

```
Ud1=0,Vd1=0,Md1=12,Qd1=6,Rd1=2
d1 = [   0.0015347528, ...
         0.9831259171,   0.9865862455,   1.0000549581,   1.0588399235, ...
         1.2178165199,   1.3676396170, ...
         2.4732268884,   1.2469818524,   0.9665196706,   2.0777243688, ...
         2.7566056975,   2.9958632972, ...
         0.4293178287,   0.6391970557,   0.8827491826, ...
         0.4992229735,   1.1330235684,   1.3538904996 ]';
```

and the corresponding transfer function numerator and denominator polynomials are:

```
N1 = [   0.0015347528,   0.0088566179,   0.0234715838,   0.0404362597, ...
         0.0558416556,   0.0697397780,   0.0788142211,   0.0767020842, ...
         0.0639378432,   0.0481184395,   0.0325993050,   0.0166224841, ...
         0.0044909916 ]';


D1 = [   1.0000000000,   0.0000000000,  -1.6757354959,   0.0000000000, ...
         2.2730061632,   0.0000000000,  -1.7981147661,   0.0000000000, ...
         1.0106395919,   0.0000000000,  -0.3464569308,   0.0000000000, ...
         0.0586816110 ]';
```

Figure 8.47: Initial response of an IIR low-pass R=2 filter after WISE optimisation

Figure 8.48: Response of an IIR low-pass R=2 filter after PCLS SQP optimisation

Low-pass IIR filter : U=0,V=0,M=12,Q=6,R=2,fap=0.1,dBap=0.06,fas=0.15,dBas=53,ctol=0.0001

Figure 8.49: Pole-zero plot of an IIR low-pass R=2 filter after PCLS SQP optimisation

Figure 8.50: Response of the initial R=2 Hilbert filter designed by *tarczynski_hilbert_R2_test.m* with and $t_d = 5.5$. The phase response shown is adjusted for the nominal delay.

8.3.1 Hilbert transform R=2 decimation filter

A Hilbert transform filter[13, p.118] has the transfer function

$$H_d(\omega) = -\imath sign(\omega) e^{-\imath\omega t_d}$$

The Octave script *iir_sqp_slb_hilbert_R2_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_R2_test.m*. The initial filter coefficients are, in gain-pole-zero form:

```
Ux0=7,Vx0=4,Mx0=4,Qx0=2,Rx0=2
x0 = [   0.0583724709, ...
        -1.8365373257,  -0.9608560046,  -0.7010433710,  -0.1500760172, ...
         0.4457772524,   0.8810567061,   1.0626244324, ...
        -0.2059885485,   0.3100927767,   0.4539231406,   0.9235051879, ...
         1.6386081452,   1.7935951733, ...
         1.0002239162,   2.0743421390, ...
         0.2291758025, ...
         1.6565178057 ]';
```

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 8.50. 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 π, is shown adjusted for the group-delay by adding ωt_d where ω is the angular frequency vector for which the response is calculated.

MMSE optimisation of the Hilbert transform R=2 decimator

Figure 8.51 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 8.51: Response of the R=2 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 8.52 shows the response after PCLS optimisation of the MMSE optimised filter of Figure 8.51. Figure 8.53 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=4,Md1=4,Qd1=2,Rd1=2
d1 = [   0.0097214827, ...
        -2.5499210935,  -0.8187128758,  -0.6533887915,  -0.1699507204, ...
         0.5446741523,   0.6040251951,   1.2207969563, ...
         0.0318104932,   0.0778485845,   0.4468547469,   0.6660207676, ...
         2.2930747306,   2.5067221381, ...
         1.0288556457,   2.0718916077, ...
         0.1144217003, ...
         1.9808036063 ]';
```

and in transfer function form the numerator and denominator polynomials are:

```
N1 = [   0.0097214827,   0.0181338123,   0.0261849096,   0.0532768313, ...
         0.1189149089,   0.5049005964,  -0.8556310943,  -0.8967761219, ...
         0.6578259852,   0.3500997900,  -0.1390929809,  -0.0299067782 ]';

D1 = [   1.0000000000,   0.0000000000,  -1.1313139503,   0.0000000000, ...
         0.3236997811,   0.0000000000,  -0.0128911109,   0.0000000000, ...
         0.0030351616,   0.0000000000,  -0.0003961336,   0.0000000000, ...
         0.0000096492 ]';
```

Figure 8.52: Response of the R=2 Hilbert filter after PCLS optimisation. The phase response shown is adjusted for the nominal delay.

Figure 8.53: Pole-zero plot of the R=2 Hilbert filter after PCLS optimisation.

8.3.2 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_R2_test.m* designs a differentiator filter with the specification:

```
ftol=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
tp=5.5 % Pass band group delay
tpr=0.0089 % Pass band group delay peak-to-peak ripple
Wtp=0.0135 % Pass band group delay weight
ppr=1.5 % Nominal pass band phase(rad./$\pi$))
ppr=0.00067 % Phase pass band peak-to-peak ripple(rad./$\pi$))
Wpp=0.0275 % 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_R2_test.m*. The initial filter coefficients, in gain-pole-zero form, are:

```
Ux0=4,Vx0=2,Mx0=4,Qx0=2,Rx0=2
x0 = [   0.0032220979, ...
        -0.2657598934,   0.3525812318,   1.0067973925,   2.4623083161, ...
        -0.1903203457,   0.2967914127, ...
         2.5879786687,   2.6254493051, ...
         2.3918211428,   1.2321633423, ...
         0.2195422820, ...
         1.5304416128 ]';
```

Figure 8.54 shows the initial filter amplitude, phase and group delay responses. The phase response is adjusted for the nominal group delay. Figure 8.55 shows the pole-zero plot of the initial filter.

MMSE optimisation of the R=2 differentiator filter

Figure 8.56 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 8.57 shows the amplitude response error and the phase and group delay responses of the 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 8.58 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 8.54: 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 8.55: Pole-zero plot of the initial R=2 differentiator filter.

Figure 8.56: 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 = [   0.0007087471, ...
        -0.2762483771,   0.3634138917,   1.0000000000,   3.7824345784, ...
        -0.0497002899,   0.1281250750, ...
         3.2208181336,   3.6327180316, ...
         2.3068350892,   1.1208563860, ...
         0.0250132888, ...
         0.3027409589 ]';
```

The numerator and denominator polynomials are:

```
N1 = [   0.0007087471,  -0.0026257530,   0.0059052267,  -0.0134787944, ...
         0.0418177725,  -0.4011644048,   0.3976234067,   0.0080572030, ...
        -0.0368434038 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,  -0.1261762987,   0.0000000000, ...
        -0.0019972866,   0.0000000000,   0.0002550070,   0.0000000000, ...
        -0.0000039841 ]';
```

Figure 8.57: 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 8.58: Pole-zero plot of the R=2 differentiator filter after PCLS optimisation.

8.3.3 Low-pass differentiator filter

The Octave script *iir_sqp_slb_lowpass_differentiator_test.m* designs an IIR low-pass differentiator filter with the specification:

```
maxiter=20000 % Maximum iterations
dmax=0.05 % SQP step-size constraint
ftol=0.001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
nN=11 % Correction filter order
fap=0.3 % Amplitude pass band upper edge
Arp=0.02 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.02 % Amplitude transition band peak-to-peak ripple
Wat=0.001 % Amplitude transition band weight
fas=0.4 % Amplitude stop band lower edge
Ars=0.02 % Amplitude stop band peak-to-peak ripple
Was=1 % Amplitude stop band weight
fpp=0.3 % Phase pass band upper edge
pp=0.5 % Phase pass band nominal phase(rad./pi)
ppr=0.0002 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=0.5 % Phase pass band weight
ftp=0.3 % Group-delay pass band upper edge
tp=10 % Pass band group delay
tpr=0.04 % Pass band group delay peak-to-peak ripple
Wtp=2 % Pass band group delay 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 = [  -0.0038995749,   0.0128814540,  -0.0192557591,   0.0140687918, ...
         0.0051997662,  -0.0236949150,   0.0151976655,   0.0292205204, ...
        -0.0560062376,  -0.1508573959,  -0.1800922176,  -0.0938936083 ]';


D0 = [   1.0000000000,  -0.9264034393,   1.8089065044,  -2.0146510639, ...
         1.9544707562,  -1.6086081725,   1.1187131246,  -0.6479553283, ...
         0.3032794320,  -0.1081616811,   0.0258714248,  -0.0027949401 ]';
```

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 8.59.

Figure 8.60 shows the response errors of the low-pass differentiator after PCLS optimisation. Figure 8.61 shows the pole-zero plot of the low-pass differentiator filter after PCLS optimisation. The PCLS optimised overall filter coefficients are, in gain-zero-pole form:

```
Ud1z=2,Vd1z=1,Md1z=10,Qd1z=10,Rd1z=1
d1z = [   0.0030716155, ...
         -1.4868704541,   1.0000000000, ...
          0.5584509883, ...
          0.7554465311,   0.9855010404,   1.6683899807,   1.7685524580, ...
          1.8047801735, ...
          3.0485208792,   2.5838847837,   1.5590342970,   0.9274277865, ...
          0.3092685973, ...
          0.4244833527,   0.5601276496,   0.5665700978,   0.6850617791, ...
          0.8936707180, ...
          1.5994121161,   0.6076532488,   1.2349106011,   1.8778872092, ...
          2.1671025361 ]';
```

and, in transfer function form, the numerator and denominator polynomials of the correction filter are, respectively:

Figure 8.59: Response of the initial low-pass differentiator filter. The phase response shown is adjusted for the nominal delay.

```
N1 = [   0.0030716155,  -0.0028746723,  -0.0019961347,   0.0104432230, ...
        -0.0146774768,   0.0048463910,   0.0218201282,  -0.0468840286, ...
         0.0166276888,   0.2286082380,   0.2356275693,   0.0717862524 ]';
```

and

```
D1 = [   1.0000000000,  -0.4094447778,   0.9282645875,  -0.9231005527, ...
         0.8404285437,  -0.6695436325,   0.4638648696,  -0.2759776059, ...
         0.1367719275,  -0.0531459536,   0.0137179113,  -0.0037984135 ]';
```

Figure 8.60: Response errors of the low-pass differentiator filter after PCLS optimisation. The phase response shown is adjusted for the nominal delay.

Figure 8.61: Pole-zero plot of the low-pass differentiator filter after PCLS optimisation.

Alternate low-pass differentiator filter

The Octave script *iir_sqp_slb_lowpass_differentiator_alternate_test.m* designs an IIR low-pass differentiator filter with the specification:

```
maxiter=20000 % Maximum iterations
dmax=0.05 % SQP step-size constraint
ftol=0.001 % Tolerance on coef. update
ctol=0.0001 % Tolerance on constraints
n=1000 % Frequency points across the band
nN=11 % Correction filter order
fap=0.18 % Amplitude pass band upper edge
Arp=0.004 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.005 % Amplitude transition band peak-to-peak ripple
Wat=0.001 % Amplitude transition band weight
fas=0.3 % Amplitude stop band lower edge
Ars=0.005 % Amplitude stop band peak-to-peak ripple
Was=1 % Amplitude stop band weight
fpp=0.18 % Phase pass band upper edge
pp=0.5 % Nominal pass band phase(rad./pi)
ppr=0.0001 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=0.2 % Phase pass band weight
ftp=0.18 % Group-delay pass band upper edge
tp=10 % Pass band group delay
tpr=0.02 % Pass band group delay peak-to-peak ripple
Wtp=1 % Pass band group delay weight
```

The filter is designed as $1 - z^{-1}$ in series with a correction filter. An initial correction filter was designed by the method of *Tarczynski et al* with the Octave script *tarczynski_lowpass_differentiator_alternate_test.m*. The inital filter numerator and denominator polynomials are:

```
N0 = [  -0.0039011053,   0.0111815147,  -0.0031240402,  -0.0185589788, ...
         0.0148762179,   0.0153469250,  -0.0062196048,  -0.0264177227, ...
        -0.0359736120,  -0.0347609878,  -0.0454036634,  -0.0250135165 ]';

D0 = [   1.0000000000,  -1.9198804102,   1.8651506498,   0.1400528890, ...
        -2.4550971365,   2.9675047372,  -1.3973579936,  -0.5558263471, ...
         1.3479998776,  -0.9949752700,   0.3870666764,  -0.0684190344 ]';
```

Figure 8.62 shows the amplitude, group delay and phase responses of the initial low-pass differentiator filter. The phase response is adjusted for linear-phase delay.

The PCLS optimised filter coefficients of the alternate low-pass differentiator filter are, in gain-zero-pole form:

```
Ud1z=2,Vd1z=1,Md1z=10,Qd1z=10,Rd1z=1
d1z = [   0.0025732860, ...
         -0.8494672355,   1.0000000000, ...
          0.5494930791, ...
          0.7803007145,   0.8854904051,   0.9862671716,   1.7476315773, ...
          1.8828041046, ...
          1.2374000063,   2.3148868199,   1.9297205874,   0.8759301126, ...
          0.2848761991, ...
          0.5242516809,   0.5314903168,   0.5517636610,   0.7710371586, ...
          0.8939130715, ...
          1.2235507523,   0.9695692544,   0.5184539970,   1.1811361018, ...
          1.3624978769 ]';
```

and, in transfer function form, the numerator and denominator polynomials of the correction filter are, respectively:

```
N1 = [   0.0025732860,  -0.0093170096,   0.0129556487,  -0.0069148463, ...
         0.0030011375,  -0.0110755455,   0.0060371255,   0.0176506884, ...
         0.0172434776,   0.0180443832,   0.0091040316,   0.0109907065 ]';
```

Figure 8.62: Amplitude, phase and group delay responses of the initial low-pass differentiator filter. The phase response shown is adjusted for the nominal delay.

and

```
D1 = [   1.0000000000,  -3.4216055390,   7.0143460758,  -9.9645280999, ...
        10.6466895807,  -8.8131933343,   5.7120342363,  -2.8876817337, ...
         1.1178162982,  -0.3179206340,   0.0607313232,  -0.0061699264 ]';
```

Figure 8.63 shows the response errors of the low-pass differentiator after PCLS optimisation of the correction filter.

Figure 8.64 shows the pole-zero plot of the low-pass differentiator after PCLS optimisation.

Figure 8.63: Response errors of the alternate low-pass differentiator filter after PCLS optimisation. The phase response shown is adjusted for the nominal delay.

Figure 8.64: Pole-zero plot of the alternate low-pass differentiator filter after PCLS optimisation.

8.3.4 Pink noise filter

Pink noise has equal noise power per octave. A ideal pink noise filter has the transfer function $H_d(\omega) = \frac{K}{\sqrt{\omega}} e^{-\imath\omega t_p}$ where K is a constant and t_p 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
ftol=2e-05 % Tolerance on coefficient update
ctol=2e-05 % Tolerance on constraints
fat=0.005 % Amplitude transition band width
AdBr=0.2 % Relative amplitude peak-to-peak ripple (dB)
Wap=20 % 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 = [   0.0255851293, ...
        -1.5620845290,  -0.4017319825,   0.7639165304, ...
         0.8665502283, ...
         0.6228596425,   0.7051456642,   1.5127224396,   1.5917319581, ...
         2.4347551354,   1.3244577532,   0.8495364259,   1.9999966038, ...
         0.5119823592,   0.5165412276,   0.5887006329,   0.6529466678, ...
         0.7106518242, ...
         0.4803411390,   1.6426184146,   2.7845842649,   2.3667939247, ...
         1.2980134977 ]';
```

Figure 8.65 compares the amplitude response of the initial filter to the desired response. Figure 8.66 shows the amplitude and group delay responses of the filter after MMSE and PCLS optimisation. Figure 8.67 shows the errors of the amplitude and group delay responses of the filter after MMSE and PCLS optimisation. Figure 8.68 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 = [   0.0008945050, ...
        -2.7364529921,   0.7543819702,   0.8769577214, ...
         0.9593599766, ...
         0.4831570984,   0.9706013507,   3.0858894894,   3.1574436876, ...
         0.0000006876,   0.0282890145,   0.6902900838,   1.8857948884, ...
         0.3400311701,   0.3536714987,   0.5049778740,   0.7543819583, ...
         0.9687500000, ...
         2.4149336740,   1.2491966214,   0.3722946165,   0.0000000153, ...
         0.0222914123 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively:

```
N1 = [   0.0008945050,  -0.0041183008,   0.0100778349,  -0.0189415534, ...
         0.0417363986,   0.1399029363,  -0.9156112258,   1.8389793511, ...
        -1.8544969765,   1.0186342744,  -0.2908628591,   0.0338089140 ]';
```

and

```
D1 = [   1.0000000000,  -5.0612142427,  10.7401401667, -12.3559901570, ...
         8.3261552066,  -3.3463077079,   0.8447125372,  -0.2097525304, ...
         0.0937711131,  -0.0425445730,   0.0129214316,  -0.0018896001 ]';
```

Figure 8.65: Response of the initial pink noise filter designed by *tarczynski_pink_test.m*.

Figure 8.66: Amplitude and group-delay responses of the pink noise filter after PCLS optimisation.

Figure 8.67: Errors of the amplitude and group-delay responses of the pink noise filter after PCLS optimisation.

Figure 8.68: Pole-zero plot of the pink noise filter after PCLS optimisation.

8.3.5 Minimum phase R=2 low-pass 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 low-pass filter specified by:

```
n=1000 % Frequency points across the band
ftol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-06 % Tolerance on constraints
fap=0.125 % Pass band amplitude response edge
dBap=0.035 % 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 8.69 shows the response of the PCLS optimised filter. Figure 8.70 shows the pass-band detail of the response of the PCLS optimised filter. Figure 8.71 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 = [   0.0057850642, ...
        -0.9960937500,  -0.9960937500, ...
         0.2014475831, ...
         0.9960937500,   0.9960937500,   0.9960937500,   0.9960937500, ...
         2.5241208511,   1.6488814710,   1.9560345955,   2.2379621908, ...
         0.3683986482,   0.7501205808, ...
         1.3927934451,   1.8465852883 ]';
```

and in transfer function form the numerator and denominator polynomials are, respectively

```
N1 = [   0.0057850642,   0.0332827681,   0.0992435611,   0.1986711813, ...
         0.2929728882,   0.3313972927,   0.2906885079,   0.1955850857, ...
         0.0969401372,   0.0322567909,   0.0055630163 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,   0.0766167941,   0.0000000000, ...
         0.5890866685,   0.0000000000,  -0.1479179000,   0.0000000000, ...
         0.0799844126,   0.0000000000,  -0.0153836815 ]';
```

[b] If the inverse filter is required then the zeros must be within the unit circle, $|z| \leq \rho < 1$.

Figure 8.69: Response of the R=2 minimum phase filter after PCLS optimisation.

Figure 8.70: Passband response of the R=2 minimum phase filter after PCLS optimisation.

Figure 8.71: Pole-zero plot of the R=2 minimum phase filter after PCLS optimisation.

8.3.6 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=2 % Number of real zeros
V=0 % Number of real poles
M=38 % Number of complex zeros
Q=0 % Number of complex poles
R=1 % Denominator polynomial decimation factor
n=1000 % Frequency points across the band
ftol=0.001 % Tolerance on relative coefficient update size
ctol=2e-05 % Tolerance on constraints
fap=0.1 % Pass band amplitude response edge
dBap=0.3 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band amplitude weight
Wat=0.001 % Transition band amplitude weight
ftp=0.1 % Pass band group-delay response edge
td=10 % Pass band group-delay
tdr=0.1 % Pass band amplitude peak-to-peak ripple
Wtp=0.1 % Pass band group-delay weight
fas=0.25 % Stop band amplitude response edge
dBas=65 % Stop band minimum attenuation
Was=10 % Stop band amplitude weight
```

Figure 8.72 shows the response of the PCLS optimised filter. Figure 8.73 shows the pass-band detail of the response of the PCLS optimised filter. The coefficients of the PCLS optimised FIR low-pass filter are, in gain-pole-zero form:

```
Ud1=2,Vd1=0,Md1=38,Qd1=0,Rd1=1
d1 = [   0.0011026096, ...
        -1.2135348915,   0.7449884640, ...
         0.0442379467,   0.0472489605,   0.7511752422,   0.7634210827, ...
         0.8004285985,   0.9150429822,   0.9486019909,   0.9950248578, ...
         0.9965275521,   0.9966107580,   0.9972112762,   0.9995772138, ...
         1.0002795267,   1.0044092031,   1.0302082491,   1.0401873539, ...
         1.0486372684,   1.0619938225,   1.6713448758, ...
         1.2439521134,   1.8530117983,   0.2218036152,   0.4372557499, ...
         0.6698237220,   1.2972212671,   1.0254598980,   2.2731960320, ...
         1.5825878108,   2.1137055557,   1.6664590394,   1.8028966565, ...
         1.9572865066,   2.4316940566,   2.7366841690,   2.9459691356, ...
         2.9504494595,   2.6206860086,   0.3107037385 ]';
```

and in transfer function form the FIR polynomial is:

```
N1 = [   0.0011026096,   0.0052973350,   0.0100247544,   0.0053005990, ...
        -0.0159597916,  -0.0407757061,  -0.0346476860,   0.0308923628, ...
         0.1435531986,   0.2485136499,   0.2879714368,   0.2448431046, ...
         0.1492590137,   0.0487165135,  -0.0225815954,  -0.0521133568, ...
        -0.0433700953,  -0.0115667803,   0.0200768668,   0.0316030483, ...
         0.0191452696,  -0.0036657898,  -0.0185974459,  -0.0169309403, ...
        -0.0034324313,   0.0099105744,   0.0133933848,   0.0061364842, ...
        -0.0042709971,  -0.0087648122,  -0.0050756131,   0.0014447962, ...
         0.0043505696,   0.0025421512,  -0.0003696932,  -0.0013346468, ...
        -0.0006151025,  -0.0000033225,  -0.0000021079,   0.0000000015, ...
        -0.0000000027 ]';
```

Figure 8.72: Response of the non-linear phase FIR low-pass filter after SQP PCLS optimisation.

Figure 8.73: Pass-band response of the non-linear phase FIR low-pass filter after SQP PCLS optimisation.

8.3.7 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 8.74 shows the response of the PCLS optimised filter. Figure 8.75 shows the pass-band detail of the response of the PCLS optimised filter. Figure 8.76 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 = [   0.0169905534, ...
         0.9687500000,   0.9687500000, ...
         0.4431419764,   0.8515604493,   0.8560142280,   0.9687500000, ...
         0.9687500000,   0.9687500000,   0.9687500000,   0.9687500000, ...
         0.9687500000,   0.9687500000,   0.9687500000,   0.9687500000, ...
         0.9687500000,   0.9687500000, ...
         3.1384307300,   0.7974242670,   1.0529559629,   1.5875621320, ...
         1.6692980888,   2.2901842195,   0.2385568786,   0.2985326714, ...
         1.8792277899,   2.0794964407,   2.5156376865,   2.7695420185, ...
         3.1256324361,   3.1352445417 ]';
```

and in transfer function form the FIR polynomial is:

```
Nd1 = [   0.0169905534,   0.0587613058,   0.0680725441,  -0.0187608648, ...
         -0.1591094619,  -0.2037159250,  -0.0574157751,   0.1719520151, ...
          0.2626271293,   0.1295763218,  -0.0745493036,  -0.1482304225, ...
         -0.0737930905,   0.0038807647,  -0.0072484662,  -0.0459543394, ...
         -0.0224728333,   0.0448068754,   0.0709917640,   0.0324125481, ...
         -0.0123701564,  -0.0159243216,   0.0025946622,   0.0025711773, ...
         -0.0164175341,  -0.0247285149,  -0.0104845830,   0.0086373347, ...
          0.0125257205,   0.0054843112,   0.0008274989 ]';
```

The script also calculates the FIR filter with complementary amplitude response. The combined response is shown in Figure 8.77. Figure 8.78 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 = [   0.0381938748, ...
        -0.6718451274,  -0.6718451273, ...
         0.4989323888,   0.7177956823,   0.7207810504,   0.7381619916, ...
         0.7440295149,   0.7641210698,   1.0000000001,   1.0068171485, ...
         1.0068617088,   1.2534068471,   1.3185837084,   1.3431774663, ...
         1.3482525841,   1.3655004861, ...
         0.0000003348,   1.6000494635,   2.8834392325,   2.4471688099, ...
         2.0219809357,   0.1225934805,   0.9135345728,   1.1659585379, ...
         0.7046903964,   0.3515077712,   1.5085489667,   1.8095361806, ...
         2.2338823971,   2.6629941044 ]';
```

and in transfer function form the FIR polynomial is:

Figure 8.74: Response of the minimum phase FIR bandpass filter after PCLS optimisation.

Figure 8.75: Passband response of the minimum phase FIR bandpass filter after PCLS optimisation.

Figure 8.76: Pole-zero plot of the minimum phase FIR bandpass filter after PCLS optimisation.

Figure 8.77: Combined response of the PCLS optimised minimum phase FIR bandpass filter and the complementary FIR filter.

Figure 8.78: Pole-zero plot of the complementary FIR filter.

```
Nc1 = [   0.0381938748,   0.0267622529,   0.0600073170,  -0.0191314051, ...
          0.0094246623,  -0.0716494626,   0.1177266441,   0.0028209845, ...
          0.2346370938,  -0.0762561439,   0.3412664895,  -0.0815426216, ...
          0.4892113189,   0.1030869417,  -0.2531851099,   0.3731330915, ...
          0.0914367014,  -0.2251082481,  -0.1119135193,   0.0202782233, ...
         -0.0080408922,  -0.1005485900,  -0.0815941573,   0.0080173696, ...
          0.0722821599,   0.0545687664,   0.0168868351,  -0.0135581369, ...
         -0.0196333666,  -0.0005406963,   0.0034726845 ]';
```

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* [66] or directly by *Orchard* and *Willson*'s Newton-Raphson solution [78]. 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* [55], 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, ϵ, 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, ϵ is an auxiliary 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:

$$H(\boldsymbol{x}, \omega) \approx H(\boldsymbol{x}_k, \omega) + \nabla_x H(\boldsymbol{x}_k, \omega)^\top (\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}_k)^\top \nabla_x^2 H(\boldsymbol{x}_k, \omega) (\boldsymbol{x} - \boldsymbol{x}_k)$$

Unfortunately the IIR filter Hessian matrix, $\nabla_x^2 H(\boldsymbol{x}_k, \omega)$, is unlikely to be positive definite.

Lu and Hinamoto [250] add linear constraints on the maximum filter response in the transition band, Ω_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, ϵ minimises the weighted sum of, for example, the amplitude and delay errors, $\mathcal{E}_A$ and $\mathcal{E}_T$. Separately minimising the amplitude and delay errors implies the minimisation of of two auxiliary variables ϵ_A and ϵ_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 ω_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

$$\begin{aligned}
|H(\boldsymbol{x}_k, \omega_i)|^2 &= H(\boldsymbol{x}_k, \omega_i) H(\boldsymbol{x}_k, \omega_i)^* \\
\nabla_x |H(\boldsymbol{x}_k, \omega_i)|^2 &= (\nabla_x H(\boldsymbol{x}_k, \omega_i)) H(\boldsymbol{x}_k, \omega_i)^* + H(\boldsymbol{x}_k, \omega_i) (\nabla_x H(\boldsymbol{x}_k, \omega_i))^* \\
&= 2\Re\nabla_x H(\boldsymbol{x}_k, \omega_i) \Re H(\boldsymbol{x}_k, \omega_i) + 2\Im\nabla_x H(\boldsymbol{x}_k, \omega_i) \Im H(\boldsymbol{x}_k, \omega_i)
\end{aligned}$$

where $*$ represents the complex conjugate. The squared-magnitude response can be used in linear constraints on the upper and lower peaks of the pass-band and stop-band magnitude responses

$$\begin{aligned}
|H(\boldsymbol{x}_k, \omega_i)|^2 + (\boldsymbol{x} - \boldsymbol{x}_k)^\top \nabla_x |H(\boldsymbol{x}_k, \omega_i)|^2 &\geq H_{d,l}^2(\omega_i) \\
|H(\boldsymbol{x}_k, \omega_i)|^2 + (\boldsymbol{x} - \boldsymbol{x}_k)^\top \nabla_x |H(\boldsymbol{x}_k, \omega_i)|^2 &\leq H_{d,u}^2(\omega_i)
\end{aligned}$$

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* [229]. *Lu* [255, 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 [255, Section III], an SOCP problem with linear and quadratic constraints.

[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 $\boldsymbol{c}_t^{(i)}$

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

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 ϵ is the MMSE error, $\boldsymbol{\delta}$ is the step direction from coefficient vector $\boldsymbol{x}_k$, and β is the constraint on the coefficient step-size, $\|\boldsymbol{\delta}\| \leq \beta$.

In a similar fashion to the MMSE error used in Chapter 8, the MMSE frequency response constraint can be expressed as a weighted sum of pass-band amplitude response, A, stop-band amplitude response, S, pass-band phase response, P, and the pass-band group-delay response, T. In this example I do not attempt to control the transition band amplitude response. In *SeDuMi* format:

$$\begin{aligned}
\boldsymbol{A}_i^\top &= \begin{bmatrix} \mathbf{0} & W_a(\omega_a)\nabla_x A(\boldsymbol{x}_k,\omega_a) \\ \mathbf{0} & W_s(\omega_s)\nabla_x S(\boldsymbol{x}_k,\omega_s) \\ \mathbf{0} & W_p(\omega_p)\nabla_x P(\boldsymbol{x}_k,\omega_p) \\ \mathbf{0} & W_t(\omega_t)\nabla_x T(\boldsymbol{x}_k,\omega_t) \end{bmatrix} \\
\boldsymbol{b}_i^\top &= \begin{bmatrix} 1 & 0 & \mathbf{0} \end{bmatrix} \\
\boldsymbol{c}_i &= \begin{bmatrix} W_a(\omega_a) & [A(\boldsymbol{x}_k,\omega_a) - A_d(\omega_a)] \\ W_s(\omega_s) & [S(\boldsymbol{x}_k,\omega_s) - S_d(\omega_s)] \\ W_p(\omega_p) & [P(\boldsymbol{x}_k,\omega_p) - P_d(\omega_p)] \\ W_t(\omega_t) & [T(\boldsymbol{x}_k,\omega_t) - T_d(\omega_t)] \end{bmatrix} \\
d_i &= 0
\end{aligned}$$

where $\omega_a \in \Omega_a$, $\omega_s \in \Omega_s$, $\omega_p \in \Omega_p$ and $\omega_t \in \Omega_t$ are the pass-band amplitude, stop-band amplitude, pass band-phase and pass-band group delay response grid frequencies and $A_d(\omega_a)$, $S_d(\omega_s)$, $P_d(\omega_p)$ and $T_d(\omega_t)$ are the desired pass-band amplitude, stop-band amplitude, pass-band phase and pass-band delay 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

Figure 9.1: Deczky Example 3, response after PCLS SOCP SeDuMi optimisation.

```
n=500 % Frequency points across the band
ftol=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 SeDuMi 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 PCLS SOCP SeDuMi optimised filter is, in the gain, zeros and poles form of Equation 8.2:

```
Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [   0.0035740640, ...
         1.0016917275,   1.0426858189,   1.3738380396,   1.7885438566, ...
         2.1775834773, ...
         2.0163068001,   2.7080423732,   1.7568391279,   0.7280652659, ...
         0.1769271653, ...
         0.4985455481,   0.5938635994,   0.6371735459, ...
         0.3457401088,   1.0938156169,   1.4357571761 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

Figure 9.2: Deczky Example 3, passband response after PCLS SOCP SeDuMi optimisation.

Figure 9.3: Deczky Example 3, pole-zero plot after PCLS SOCP SeDuMi optimisation.

```
N2 = [   0.0035740640,  -0.0132003541,   0.0131953996,  -0.0069649820, ...
         0.0196601768,  -0.0125427135,  -0.0330401539,  -0.0046799391, ...
         0.1039733667,   0.1263013022,   0.1116239316 ]';
```

and

```
D2 = [   1.0000000000,  -1.6549366776,   1.7732301581,  -1.2595104519, ...
         0.6194324765,  -0.2043796556,   0.0355875893 ]';
```

The *deczky3a_socp_test.m* script also calls the SeDuMi solver through the function *iir_socp_mmse.m*. This 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
ftol=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 SeDuMi 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 PCLS SOCP SeDuMi 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 = [   0.0024139083, ...
         1.0756728053,   1.0771015829,   1.1394847731,   1.6128588008, ...
         2.1414832371, ...
         1.9094998568,   2.7456002614,   2.0904146609,   0.3771576758, ...
         1.0594793992, ...
         0.6697608716,   0.7373185510,   0.7782272718, ...
         0.3559556660,   1.0084511975,   1.4622689450 ]';
```

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* [55, 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 $\nabla_x^2 \mathcal{E}(\boldsymbol{x})$ and updated with the BFGS Hessian matrix update algorithm shown in Appendix K.7.1.

The filter specification is

Figure 9.4: Deczky Example 3, response after PCLS SOCP SeDuMi optimisation, alternative specification.

Figure 9.5: Deczky Example 3, passband response after PCLS SOCP SeDuMi optimisation, alternative specification.

Figure 9.6: Deczky Example 3, pole-zero plot after PCLS SOCP SeDuMi optimisation, alternative specification.

```
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
ftol=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 SeDuMi 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 PCLS SOCP SeDuMi optimised filter is, in the gain, zeros and poles form of Equation 8.2:

```
Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [   0.0028603138, ...
         1.1153501977,   1.1326296806,   1.1339461180,   1.8249222537, ...
         2.1304788384, ...
         2.7293068379,   2.0494180703,   1.8375924429,   0.7412322936, ...
         0.2300983416, ...
         0.5590030319,   0.6426477668,   0.7101804871, ...
         0.3514324633,   1.0470003089,   1.4478422721 ]';
```

The corresponding transfer function numerator and denominator polynomials are:

Figure 9.7: Deczky Example 3, response after PCLS SOCP SeDuMi optimisation with BFGS update of the Hessian.

Figure 9.8: Deczky Example 3, passband response after PCLS SOCP SeDuMi 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, pole-zero plot after PCLS SOCP SeDuMi optimisation with BFGS update of the Hessian.

```
N2 = [   0.0028603138,  -0.0090269812,   0.0046355815,   0.0042886002, ...
         0.0125876623,  -0.0167147809,  -0.0287900040,   0.0082896024, ...
         0.0946955139,   0.1063565685,   0.0887237875 ]';


D2 = [   1.0000000000,  -1.8667409142,   2.1994778556,  -1.7319696866, ...
         0.9458074774,  -0.3424436179,   0.0650896301 ]';
```

Figure 9.10: Deczky Example 3, response after PCLS SOCP SCS optimisation.

9.5 An example of SOCP design of an IIR filter expressed in *gain-zero-pole* format with the *SCS* SOCP solver

The *deczky3_scs_test.m* script calls the SCS [30, 31, 32] SOCP solver through the Octave function *iir_scs_mmse.m*. The filter specification is:

```
n=1000 % Frequency points across the band
ftol=0.001 % Tolerance on relative coefficient update size
ctol=0.0001 % 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=0.02 % Pass band group delay peak-to-peak ripple
Wtp=0.5 % Pass band group delay weight
fas=0.3 % Stop band amplitude response edge
dBas=30 % Stop band minimum attenuation
Was=2.5 % 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 SCS pass in *deczky3_scs_test.m* is shown in Figure 9.10 with pass-band details shown in Figure 9.11 and pole-zero plot shown in Figure 9.12.

Figure 9.11: Deczky Example 3, passband response after PCLS SOCP SCS optimisation.

Figure 9.12: Deczky Example 3, pole-zero plot after PCLS SOCP SCS optimisation.

The PCLS SOCP SCS optimised filter is, in the gain, zeros and poles form of Equation 8.2:

```
Ud2=0,Vd2=0,Md2=10,Qd2=6,Rd2=1
d2 = [   0.0065513205, ...
         0.9587357413,   0.9894383793,   1.3634947554,   1.7930368228, ...
         1.8699439224, ...
         2.7208392719,   2.0198972707,   1.6817353665,   0.7885443710, ...
         0.2764068608, ...
         0.3780606913,   0.4803995956,   0.6248532968, ...
         0.2569673776,   1.0515302558,   1.3173337162 ]';
```

The corresponding transfer function numerator and denominator polynomials are:

```
N2 = [   0.0065513205,  -0.0210585240,   0.0263462745,  -0.0200523491, ...
         0.0267397772,  -0.0143009359,  -0.0353363085,  -0.0066563428, ...
         0.1025054554,   0.1390940993,   0.1232102149 ]';


D2 = [   1.0000000000,  -1.5214574361,   1.4914102997,  -0.9349816378, ...
         0.3892799035,  -0.1028396991,   0.0128790683 ]';
```

9.6 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
ftol=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 9.13 shows the response of the PCLS optimised filter. Figure 9.14 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 = [  0.00183850, ...
       -1.16373624,  0.86741439, ...
        0.86794030,  0.87007422,  0.87081645,  0.87271561, ...
        0.87279405,  0.88806783,  0.93790891,  0.94529516, ...
        0.95353423,  0.95607372,  0.96403101,  0.96949825, ...
        0.97335201,  0.97615592,  0.97827153,  0.97992807, ...
        0.98129188,  0.98250161,  0.98369124,  0.98449833, ...
        0.98507460,  0.98696215,  0.98986566,  0.99331110, ...
        0.99866283,  1.33714311,  1.35104590, ...
        0.47289031,  0.31206724,  0.15947847,  0.78063685, ...
        0.62941581,  0.94160151,  1.49199847,  1.59212881, ...
        1.40443036,  1.70032366,  1.80330121,  1.90661167, ...
        2.01087552,  2.11603991,  2.22192878,  2.32836579, ...
        2.43519213,  2.54227288,  2.64948515,  1.32286734, ...
        2.75673341,  2.86397147,  2.97139551,  3.07844038, ...
        1.26478669,  0.66461304,  0.22194839 ]';
```

and in transfer function form the FIR polynomial is:

```
N1 = [  0.00183850,  0.00669618,  0.00984558,  0.00489649, ...
       -0.00706684, -0.01450198, -0.00581601,  0.01532408, ...
        0.02709522,  0.00816303, -0.03460740, -0.05781437, ...
       -0.01074535,  0.11514994,  0.26190158,  0.33750112, ...
        0.28616911,  0.13701573, -0.01506483, -0.08306156, ...
       -0.05276935,  0.01654790,  0.05191155,  0.02912849, ...
       -0.01716353, -0.03740229, -0.01656445,  0.01784173, ...
        0.03003792,  0.01244801, -0.01178606, -0.01724318, ...
       -0.00219780,  0.01372981,  0.01331001, -0.00085224, ...
       -0.01215536, -0.00939889,  0.00244786,  0.00990551, ...
        0.00616403, -0.00308755, -0.00761775, -0.00380997, ...
        0.00294171,  0.00541357,  0.00212080, -0.00244456, ...
       -0.00359293, -0.00106224,  0.00182174,  0.00225987, ...
        0.00044621, -0.00150146, -0.00205694, -0.00138083, ...
       -0.00045085 ]';
```

Figure 9.13: Response of the non-linear phase FIR low-pass filter after SOCP PCLS optimisation.

Figure 9.14: 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 9.15 shows the response of the PCLS optimised filter. Figure 9.16 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 = [  0.00680980, ...
       -0.96180950,  2.23804734, ...
        0.79806377,  0.80484222,  0.80648473,  0.94366224, ...
        0.94520949,  0.94677469,  0.94771816,  0.94980865, ...
        0.95007443,  0.95599489,  0.96348362,  0.97995414, ...
        0.99656804,  1.67870264, ...
        0.16044623,  0.47499265,  0.81349725,  2.54423189, ...
        2.34407390,  2.74375403,  2.14458744,  2.94180934, ...
        1.93894152,  1.74561785,  1.55778974,  1.38974327, ...
        1.27394340,  0.40169462 ]';
```

and in transfer function form the FIR polynomial is:

```
N1 = [  0.00680980, -0.00618917, -0.01820481, -0.01586487, ...
        0.00818593,  0.04389894,  0.05260043, -0.00252628, ...
       -0.12056356, -0.24808837, -0.31084705, -0.26689404, ...
       -0.14161792, -0.00984964,  0.05960469,  0.05184468, ...
        0.00505133, -0.02856448, -0.02612038, -0.00249740, ...
        0.01366102,  0.01014706, -0.00315731, -0.00952879, ...
       -0.00389843,  0.00539214,  0.00765763,  0.00190095, ...
       -0.00539991, -0.00765377, -0.00467825 ]';
```

Figure 9.15: Response of the alternative non-linear phase FIR low-pass filter after SOCP PCLS optimisation.

Figure 9.16: Pass-band response of the alternative non-linear phase FIR low-pass filter after SOCP PCLS optimisation.

9.7 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
tp=10 % Pass band group-delay
tpr=0.2 % Pass band amplitude peak-to-peak ripple
Wtp=0.1 % 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
```

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
ftol=0.0005 % Tolerance on relative coefficient update size
ctol=0.0001 % 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=0.6 % Pass band amplitude peak-to-peak ripple
Wtp=0.005 % 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 9.17 compares the amplitude responses of the four filters and Figure 9.18 compares the delay responses (the flat delay of 15 samples of the order 30 symmetric FIR filter is not shown).

Figure 9.17: Comparison of FIR and IIR low-pass filters, amplitude responses.

Figure 9.18: Comparison of FIR and IIR low-pass filters, group delay responses.

9.8 SOCP design of a low-pass differentiator filter

The Octave script *iir_socp_slb_lowpass_differentiator_test.m* designs an IIR low-pass differentiator filter with the specification:

```
maxiter=2000 % Maximum iterations
dmax=0.05 % SQP step-size constraint
ftol=0.001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
nN=11 % Correction filter order
fap=0.3 % Amplitude pass band upper edge
Arp=0.02 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.02 % Amplitude transition band peak-to-peak ripple
Wat=0.001 % Amplitude transition band weight
Ars=0.02 % Amplitude stop band peak-to-peak ripple
Was=1 % Amplitude stop band weight
tp=10 % Pass band group delay
tpr=0.08 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
pp=0.5 % Phase pass band nominal phase(rad./pi)
ppr=0.0004 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=0.2 % 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.

As shown in Section 8.3.3, the Octave script *tarczynski_lowpass_differentiator_test.m* designs an initial filter by the method of *Tarczynski et al*.

Figure 9.19 shows the response errors of the low-pass differentiator after PCLS optimisation. Figure 9.20 shows the pole-zero plot of the low-pass differentiator filter after PCLS optimisation. The PCLS optimised overall filter coefficients are, in gain-zero-pole form:

```
Ud1z=2,Vd1z=1,Md1z=10,Qd1z=10,Rd1z=1
d1z = [   0.0028834725, ...
         -1.0214159134,   1.0000000000, ...
          0.4787107456, ...
          0.7569056374,   1.0915932462,   1.6525068722,   1.8325296967, ...
          1.9462224379, ...
          2.4218671238,   2.6376026574,   1.5192464141,   0.8329330514, ...
          0.3638365205, ...
          0.4302598878,   0.5022822816,   0.5220527618,   0.7195893332, ...
          0.9215850659, ...
          0.3933290961,   1.1077015496,   1.3494605915,   1.8343327444, ...
          2.0904052114 ]';
```

and, in transfer function form, the numerator and denominator polynomials of the correction filter are, respectively:

```
N1 = [   0.0028834725,  -0.0063492243,   0.0040323209,   0.0063612724, ...
        -0.0101788021,   0.0153224192,   0.0175826721,  -0.0423901376, ...
         0.0396388852,   0.2093512868,   0.1964917481,   0.0698388599 ]';
```

and

```
D1 = [   1.0000000000,  -0.6613834954,   1.2493084854,  -1.3434227693, ...
         1.2516091316,  -1.0361343441,   0.7129311676,  -0.4277208413, ...
         0.2054596794,  -0.0747403266,   0.0192958919,  -0.0026797834 ]';
```

Figure 9.19: Response errors of the low-pass differentiator filter after PCLS optimisation. The phase response shown is adjusted for the nominal delay.

Figure 9.20: Pole-zero plot of the low-pass differentiator filter after PCLS optimisation.

9.9 SOCP MMSE design of a low-pass R=2 filter expressed in *gain-zero-pole* format with *SeDuMi*

The Octave script *iir_socp_slb_lowpass_R2_test.m* uses the *SeDuMi* SOCP solver to design an $R = 2$ lowpass filter, similar to that shown in Section 8.3. After some experimentation, the 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
ftol_wise=1e-07 % Tolerance on WISE relative coef. update
ftol_mmse=1e-05 % Tolerance on MMSE relative coef. update
ftol_pcls=0.0001 % Tolerance on PCLS relative coef. update
ctol=1e-05 % Tolerance on constraints
fap=0.1 % Pass band amplitude response edge
dBap=0.05 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
Wat=1 % Transition band weight
fas=0.15 % Stop band amplitude response edge
dBas=60 % Stop band minimum attenuation
Was=10 % Stop band amplitude weight
```

The *SeDuMi* solver seems to require the initial filter to be a closer approximation than that required by the SQP solver. The initial filter was calculated by optimising a "guess" filter with the WISE method of *Tarczynski et al.* described in Section 8.1.5. The response of the initial filter is shown in Figure 9.21. The response of the filter after PCLS SOCP optimimisation is shown in Figure 9.22. The corresponding pole-zero plot is shown in Figure 9.23. The estimate of the optimised filter vector, in the gain, zeros and poles form of Equation 8.2, is:

```
Ud1=0,Vd1=0,Md1=12,Qd1=6,Rd1=2
d1 = [   0.0034966427, ...
         0.9977358567,   0.9996098640,   0.9999984551,   0.9999993691, ...
         1.0000234922,   1.0000530201, ...
         0.9623337038,   1.7893116877,   1.1609939282,   2.4246470024, ...
         2.9628683136,   2.5898388944, ...
         0.4846775314,   0.6841759164,   0.8991355655, ...
         0.4891849365,   1.1124996121,   1.3338403119 ]';
```

and the corresponding transfer function numerator and denominator polynomials are:

```
N1 = [   0.0034966427,   0.0128502867,   0.0250425776,   0.0377285766, ...
         0.0512800977,   0.0623332995,   0.0663712169,   0.0621988585, ...
         0.0510614717,   0.0374944334,   0.0248646801,   0.0127662804, ...
         0.0034786285 ]';


D1 = [   1.0000000000,   0.0000000000,  -1.8831880576,   0.0000000000, ...
         2.6462246302,   0.0000000000,  -2.2393624281,   0.0000000000, ...
         1.3262001235,   0.0000000000,  -0.4851997754,   0.0000000000, ...
         0.0888979391 ]';
```

Figure 9.21: Initial response of an IIR low-pass R=2 filter after WISE optimisation

Figure 9.22: Response of an IIR low-pass R=2 filter after PCLS SOCP optimisation

Low-pass IIR filter : U=0,V=0,M=12,Q=6,R=2,fap=0.1,dBap=0.05,fas=0.15,dBas=60

Figure 9.23: Pole-zero plot of an IIR low-pass R=2 filter after PCLS SOCP optimisation

Figure 9.24: IIR band-pass R=2 decimation filter, response of the initial filter calculated with the WISE method of *Tarcynzki et al.* .

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

The Octave script *iir_socp_slb_bandpass_R2_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
ftol=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=0.3 % 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=0 % Number of real poles
M=18 % Number of complex zeros
Q=10 % Number of complex poles
R=2 % Denominator polynomial decimation factor
```

The *SeDuMi* solver seems to require the initial filter to be a closer approximation than that required by the SQP solver. In this case, the initial filter is calculated by the Octave function *tarczynski_bandpass_R2_test.m*. The response of the initial filter is shown in Figure 9.24. After PCLS SOCP optimimisation, the response of the resulting filter is shown in Figure 9.25 with pass-

band detail shown in Figure 9.26. The corresponding pole-zero plot is shown in Figure 9.27. 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 = [   0.0189068439, ...
        -0.2550677149,   0.8026098317, ...
         0.9896659144,   0.9928948767,   0.9932682145,   0.9959532021, ...
         0.9991992291,   1.0444000165,   1.0643550757,   1.2902588866, ...
         1.3017427646, ...
         0.2707539494,   1.9965366659,   1.7728857750,   1.6002474463, ...
         2.5504312119,   2.2763848464,   2.9949146789,   0.7867781878, ...
         1.1057353974, ...
         0.6133623871,   0.6167106744,   0.6570179043,   0.7203289359, ...
         0.7589708989, ...
         2.4178101455,   1.8725788107,   1.3265411207,   2.7422379028, ...
         1.0292509442 ]';
```

and the corresponding transfer function numerator and denominator polynomials (found with Octave function *x2tf*) are, respectively:

```
N1 = [   0.0189068439,   0.0180109547,   0.0290193716,   0.0362383670, ...
         0.0549790686,   0.0605938912,   0.0383480080,   0.0214119958, ...
         0.0016246892,  -0.0030383978,  -0.0494406630,  -0.0942418434, ...
        -0.0817501122,   0.0159008175,   0.1421378605,   0.1709076687, ...
         0.1033767849,  -0.0232265551,  -0.0643291842,  -0.0636640611, ...
        -0.0127288382 ]';
```

and

```
D1 = [   1.0000000000,   0.0000000000,   1.5128709762,   0.0000000000, ...
         1.7004084218,   0.0000000000,   1.7166098547,   0.0000000000, ...
         1.5478766430,   0.0000000000,   1.1446382750,   0.0000000000, ...
         0.7605817975,   0.0000000000,   0.4157257731,   0.0000000000, ...
         0.2001925731,   0.0000000000,   0.0714589474,   0.0000000000, ...
         0.0184613558 ]';
```

Figure 9.25: IIR band-pass R=2 decimation filter with delay tp=16 samples, response of the filter after PCLS SOCP optimisation.

Figure 9.26: 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=0.3,Wap=1,fasl=0.05,fasu=0.25,dBas=30,Wasl=0.5,Wasu=1,tp=16,Wtp=1

Figure 9.27: IIR band-pass R=2 decimation filter with delay tp=16 samples, pole-zero plot of the filter after PCLS SOCP optimisation.

9.11 SOCP PCLS 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:

```
ftol=0.001 % Tolerance on coefficient update
ctol=0.001 % Tolerance on constraints
maxiter=1000 % SOCP iteration limit
npoints=700 % Frequency points across the band
nplot=1000 % 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=20 % Nominal pass band 1 filter group delay
tp2=15 % 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=0.001 % Delay pass band 1 weight
Wtp2=0.001 % Delay pass band 2 weight
```

The initial filter is designed by frequency transformation of a prototype elliptic filter. The gain-zero-pole description of the SOCP PCLS optimised filter is:

```
Ux2=2,Vx2=0,Mx2=18,Qx2=20,Rx2=1
x2 = [   0.0431912271, ...
        -0.9998575856,   0.9963938666, ...
         0.9648834973,   0.9686915224,   0.9817932250,   0.9838463404, ...
         0.9947940502,   0.9973416586,   0.9988047204,   1.0003251084, ...
         1.0070541557, ...
         0.4364897217,   0.6552203283,   1.4748858861,   0.4575408332, ...
         0.6379274214,   0.8680952078,   1.0531935881,   1.0861673621, ...
         1.5310066473, ...
         0.8924733587,   0.8925154580,   0.9434458331,   0.9585655794, ...
         0.9759390748,   0.9790381576,   0.9834902521,   0.9914989917, ...
         0.9927006442,   0.9961339179, ...
         1.2312628148,   1.3795416802,   0.5550460298,   0.4741824803, ...
         1.1065172225,   0.6328802515,   1.4142059128,   0.4711107033, ...
         0.6306916494,   1.0941906499 ]';
```

The numerator and denominator polynomials of the SOCP PCLS optimised filter are:

```
N2 = [     0.0431912271,    -0.4372208705,     2.2338127281,    -7.5774403512, ...
          18.9691975118,   -36.8676771746,    56.8671003003,   -69.4443402121, ...
```

Figure 9.28: IIR multi-band filter, response after SOCP PCLS optimisation.

```
          64.3719109891,   -37.9102926504,    -2.7481372883,    42.2888789335, ...
         -66.0808303155,    68.4468932874,   -54.3645666755,    34.3055079362, ...
         -17.1996118165,     6.6941506967,    -1.9208317884,     0.3652631887, ...
          -0.0349529264 ]';


D2 = [     1.0000000000,   -11.2870463436,    65.0254563167,  -252.1501624008, ...
         734.5471064397, -1703.0655371872,  3253.0020999071, -5232.7568899411, ...
        7190.8975794849, -8518.6719721315,  8743.8320615346, -7789.8345787897, ...
        6015.1178890896, -4006.8350320187,  2282.6778955478, -1096.8888753591, ...
         435.1566586550,  -137.7846526724,    32.8953708153,    -5.3113903151, ...
           0.4404950207 ]';
```

Figure 9.28 shows the amplitude and delay responses of the PCLS SOCP optimised filter, calculated with the *iirA* and *iirT* functions respectively. Figure 9.29 shows the pass-band response of the filter. Figure 9.30 shows the pole-zero plot of the filter.

Figure 9.29: IIR multi-band filter, pass-band response after SOCP PCLS optimisation.

Figure 9.30: Pole-zero plot of the IIR multi-band filter 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* [249]).

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,   0.0004069328,   0.0076883394,   0.0051931734, ...
       -0.0115231075,  -0.0210543663,   0.0019486080,   0.0417050317, ...
        0.0578424347,   0.0390044904,   0.0123776723 ]';
```

and

```
d = [   1.0000000000,  -2.4277939765,   3.0446749922,  -2.3351332894, ...
        1.1383562680,  -0.3371599419,   0.0468622434 ]';
```

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. [180] 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* [249], 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

$$
\left\| \nabla H^2(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i)^\top \boldsymbol{\delta} + H^2(\boldsymbol{a}_k, \boldsymbol{b}_k, \omega_i) \right\| \leq \left|H_d(\omega_i)\right|^2
$$

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 = [   0.6633914912,   0.0814907517,   0.3631050430,  -1.1902766262, ...
         0.4909956847,  -0.6364899703,   0.5204352646,   1.1134122429, ...
         0.2456655839,  -0.1910909588,  -0.2858567007 ]';
```

and

```
b1 = [  -0.2621457649,   0.0752261038,  -0.0146249222,   0.3528100144, ...
         1.1954081752,   0.3589309964,   0.3222312861,  -0.4008746221, ...
        -0.9697179449,   0.2653072759,  -0.8822769049,   0.8651325258 ]';
```

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,  -0.1595630695,  -0.4253296980,   0.5343547601, ...
         -0.1495588393,  -0.0933684176,   0.1191926192,  -0.0438668272, ...
          0.0133239686,   0.0114994145,  -0.0142010213,  -0.0043225424 ]';
```

and

```
Db1 = [   1.0000000000,  -0.6111260756,   0.0430031207,   0.8370204526, ...
         -0.5985320838,   0.0193691464,   0.1327051221,  -0.1425225781, ...
          0.0659398513,   0.0100223744,  -0.0160424753,   0.0049737734, ...
         -0.0008765179 ]';
```

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=2000 % 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 = [  -0.6949656665,  -1.2758556915,   0.6784984923,  -1.1578595447, ...
         0.9076672356 ]';
```

and

```
b1 = [  -1.2017642733,   0.8111469603,  -1.1485263029,   0.9728748591, ...
        -1.3547829769,   0.5434986238 ]';
```

The corresponding allpass filter denominator polynomials are

```
Da1 = [   1.0000000000,  -3.1286809028,   4.7547759494,  -4.0726352414, ...
          1.9666266813,  -0.4279951971 ]';
```

and

```
Db1 = [   1.0000000000,  -3.7050735532,   6.8919119845,  -7.6650818292, ...
          5.3550404960,  -2.2108959160,   0.4288989414 ]';
```

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 π 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 ωD, 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, ...
          0.0198375902,  -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,  -0.5292155426,   0.3637414508,   0.1968593387, ...
             0.0368342352,  -0.0661347500,  -0.0989859293,  -0.0823773849, ...
            -0.0478345361,  -0.0194755173,  -0.0051849899,  -0.0007373686 ]';
```

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, θ, 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
ftol=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=0.02 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas=0.171 % Stop band amplitude response edge
dBas=84.02 % 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,  -2.9455518420,   4.3142523445,  -3.5602028436, ...
          1.6652151227,  -0.3491170589 ]';
```

and

```
Db1 = [   1.0000000000,  -3.5167501497,   6.3330836055,  -6.8239321146, ...
          4.6386858287,  -1.8606678900,   0.3509796844 ]';
```

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 = [   0.0009313128,  -0.0007620770,   0.0031292939,  -0.0007387238, ...
           0.0035407146,   0.0014623754,   0.0014623754,   0.0035407146, ...
          -0.0007387238,   0.0031292939,  -0.0007620770,   0.0009313128 ]';
```

and

```
Dab1 = [   1.0000000000,  -6.4623019917,  21.0061058313, -44.2107086162, ...
          66.2470114372, -73.7166474530,  61.9124212081, -39.1501846163, ...
          18.2453309409,  -5.9674155331,   1.2340475797,  -0.1225329952 ]';
```

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

Design of an IIR lowpass filter with a flat passband delay as the sum of two allpass filters 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 filter A denominator order
Va=1 % Allpass filter A no. of real poles
Qa=10 % Allpass filter A no. of complex poles
Ra=1 % Allpass filter A decimation
mb=12 % Allpass filter B denominator order
Vb=2 % Allpass filter B no. of real poles
Qb=10 % Allpass filter B no. of complex poles
Rb=1 % Allpass 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. Figure 10.27 shows the PCLS optimised filter response. Figure 10.28. shows the PCLS optimised filter response in the pass-band. Figure 10.29 shows the phase responses of the two all-pass fiters. Figures 10.30 and 10.31 show the pole-zero plots of the branch allpass filters. Figure 10.32 shows the pole-zero plot of the overall filter.

The denominator polynomials of the two all-pass filters are

```
Da1 = [   1.0000000000,   0.2156943181,  -0.2362045094,   0.0806331078, ...
         -0.3308786705,  -0.0902643123,   0.1864167905,  -0.0412753326, ...
          0.1103226648,   0.0331097421,  -0.0825071082,   0.0235217259 ]';
```

and

```
Db1 = [   1.0000000000,  -0.3121341593,   0.0300812543,   0.5277515452, ...
         -0.2905884764,   0.1369495613,   0.1357028852,  -0.2297949550, ...
          0.1512967466,  -0.0392311512,  -0.0814416096,   0.0758579330, ...
         -0.0464753837 ]';
```

The corresponding filter numerator and denominator polynomials of the overall filter are

```
Nab1 = [  -0.0114768289,  -0.0120077928,   0.0027344013,   0.0157283045, ...
           0.0292364206,   0.0159616866,  -0.0252626610,  -0.0469578932, ...
          -0.0239498023,   0.0620467349,   0.1854920592,   0.2679168545, ...
           0.2679168545,   0.1854920592,   0.0620467349,  -0.0239498023, ...
          -0.0469578932,  -0.0252626610,   0.0159616866,   0.0292364206, ...
           0.0157283045,   0.0027344013,  -0.0120077928,  -0.0114768289 ]';
```

Figure 10.26: Parallel all-pass filters with flat pass-band delay, initial response found with the WISE barrier function.

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 with flat pass-band delay, phase responses after PCLS SOCP optimisation.

Figure 10.30: Parallel all-pass filters with flat pass-band delay, pole-zero plot of the A allpass filter branch after PCLS SOCP optimisation.

Figure 10.31: Parallel all-pass filters with flat pass-band delay, pole-zero plot of the B allpass filter branch after PCLS SOCP optimisation.

Figure 10.32: Parallel all-pass filters with flat pass-band delay, pole-zero plot of the overall filter after PCLS SOCP optimisation.

and

```
Dab1 = [   1.0000000000,  -0.0964398413,  -0.2734488198,   0.6886005045, ...
          -0.5399078124,  -0.0349462484,   0.4810728035,  -0.5331031436, ...
           0.2580464711,   0.1353539329,  -0.3669092866,   0.2895604496, ...
          -0.0481222668,  -0.1209724772,   0.1411666482,  -0.0740470836, ...
          -0.0039245097,   0.0445307849,  -0.0399668730,   0.0143862350, ...
           0.0031810757,  -0.0097132539,   0.0056188590,  -0.0010931812 ]';
```

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.

Design of an IIR low-pass differentiator filter as the sum of two all-pass filters The Octave script *parallel_allpass_socp_slb_lowpass_differentiator_test.m* uses the SeDuMi SOCP solver to design a low-pass differentiator filter consisting of $1 - z^{-1}$ followed by a correction filter, implemented as the sum of two all-pass filters[a]. 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 parallel all-pass low-pass differentiator filter specification is:

```
polyphase=0 % Use polyphase combination
difference=0 % Use difference combination
tol=1e-05 % Tolerance on coefficient update vector
ctol=1e-06 % Tolerance on constraints
n=1000 % Frequency points across the band
ma=11 % Allpass correction filter A denominator order
Va=1 % Allpass correction filter A no. of real poles
Qa=10 % Allpass correction filter A no. of complex poles
Ra=1 % Allpass correction filter A decimation
```

[a]Section 8.3.3 shows the design of a low-pass differentiator filter with coefficients given in gain-pole-zero form. The numerator polynomial of the filter designed in Section 8.3.3 is not symmetric so that filter cannot be decomposed into allpass filters with the spectral factorisation method described in Appendix M.2.

Figure 10.33: Initial response of the low-pass differentiator filter implemented as the sum of two all-pass filters. The phase response shown is adjusted for the nominal delay.

```
mb=12 % Allpass correction filter B denominator order
Vb=2 % Allpass correction filter B no. of real poles
Qb=10 % Allpass correction filter B no. of complex poles
Rb=1 % Allpass correction filter B decimation
fap=0.3 % Pass band amplitude response edge
Arp=0.100000 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
Art=0.100000 % Transition band amplitude response ripple
Wat=0.02 % Transition band amplitude response weight
fas=0.4 % Stop band amplitude response edge
Ars=0.100000 % Stop band amplitude response ripple
Was=2 % Stop band amplitude response weight
tp=12 % Pass band nominal group delay
tpr=0.2 % Pass band nominal group delay ripple
Wtp=0.5 % Pass band group delay response weight
pp=0.5 % Pass band nominal phase
ppr=0.002 % Pass band phase peak-to-peak-ripple
Wpp=0.05 % Pass band phase response weight
rho=0.990000 % Constraint on allpass pole radius
```

The Octave script *tarczynski_parallel_allpass_lowpass_differentiator_test.m* designs an initial low-pass differentiator correction filter with the *WISE* method described in Section 8.1.5. Figure 10.33. shows the initial low-pass differentiator filter response. The phase response shown is adjusted for the nominal delay.

After PCLS optimisation, the denominator polynomials of the all-pass filters in the low-pass differentiator correction filter are:

```
Da1 = [   1.0000000000,   0.1366934979,   0.2547195620,   0.1076017366, ...
          0.0932700968,   0.1099032307,   0.0060429138,   0.1025284303, ...
         -0.0120272770,   0.0613044948,   0.0079527679,   0.0238300440 ]';

Db1 = [   1.0000000000,  -0.2559292552,  -0.0310035280,  -0.2191146023, ...
          0.0753755558,  -0.0281169132,  -0.0775757809,   0.0994858265, ...
         -0.1043543702,   0.0628936942,  -0.0312994079,   0.0016091951, ...
          0.0206422663 ]';
```

Figure 10.34: Response error of the low-pass differentiator filter implemented as the sum of two all-pass filters after PCLS SOCP optimisation. The phase response shown is adjusted for the nominal delay.

The corresponding numerator and denominator polynomials of the correction filter are:

```
Nab1 = [   0.0222361551,   0.0031424106,   0.0163544405,   0.0140306704, ...
           0.0010641089,   0.0334824359,   0.0119769262,   0.0107784828, ...
           0.0699991425,  -0.0113058358,   0.0027128456,   0.3104135490, ...
           0.3104135490,   0.0027128456,  -0.0113058358,   0.0699991425, ...
           0.0107784828,   0.0119769262,   0.0334824359,   0.0010641089, ...
           0.0140306704,   0.0163544405,   0.0031424106,   0.0222361551 ]';


Dab1 = [   1.0000000000,  -0.1192357573,   0.1887321689,  -0.1809410342, ...
           0.1032584738,   0.0090703105,  -0.1107729035,   0.1669679784, ...
          -0.1690465729,   0.1311641707,  -0.0782780868,   0.0329631599, ...
          -0.0031784051,  -0.0106139866,   0.0135808497,  -0.0157675933, ...
           0.0144284718,  -0.0091419816,   0.0060626860,  -0.0018083197, ...
           0.0011002232,   0.0005323950,   0.0002025103,   0.0004919061 ]';
```

Figure 10.34. shows the response error of the low-pass differentiator filter after PCLS optimisation. The phase response shown is adjusted for the nominal delay.

Figure 10.35 shows the phase responses of the parallel all-pass correction filters (including a series $1 - z^{-1}$) after PCLS SOCP optimisation.

Figure 10.36 shows the pole-zero plot of the low-pass differentiator after PCLS optimisation.

Figure 10.37 shows the response of the correction filter of the low-pass differentiator after PCLS optimisation. The amplitude response plot demonstrates that the parallel allpass filter is not an appropriate structure for the low-pass differentiator.

Figure 10.35: Phase responses of the low-pass differentiator filter parallel allpass filters (including a series $1 - z^{-1}$) after PCLS SOCP optimisation.

Figure 10.36: Pole-zero plot of the parallel-allpass low-pass differentiator filter after PCLS SOCP optimisation.

Figure 10.37: Response of the correction filter of the parallel-allpass low-pass differentiator filter after PCLS SOCP optimisation.

Design of an IIR band-pass differentiator filter as the difference of two all-pass filters The Octave script *parallel_allpass_socp_slb_bandpass_differentiator_test.m* uses the SeDuMi SOCP solver to design a band-pass differentiator filter consisting of $1 - z^{-2}$ followed by a correction filter, implemented as the sum 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 parallel all-pass band-pass differentiator filter specification is:

```
polyphase=0 % Use polyphase combination
difference=1 % Use difference combination
ftol=0.0001 % Tolerance on coefficient update vector
ctol=1e-06 % Tolerance on constraints
rho=0.990000 % Constraint on allpass pole radius
n=1000 % Frequency points across the band
ma=11 % Allpass correction filter A denominator order
Va=3 % Allpass correction filter A no. of real poles
Qa=8 % Allpass correction filter A no. of complex poles
Ra=1 % Allpass correction filter A decimation
mb=11 % Allpass correction filter B denominator order
Vb=3 % Allpass correction filter B no. of real poles
Qb=8 % Allpass correction filter B no. of complex poles
Rb=1 % Allpass correction filter B decimation
fasl=0.05 % Lower stop band amplitude response edge
Arsl=0.040000 % Lower stop band amplitude response ripple
Wasl=10 % Lower stop band amplitude response weight
Watl=0.01 % Lower transition band amplitude response weight
fapl=0.1 % Lower pass band amplitude response edge
fapu=0.2 % Upper pass band amplitude response edge
Arp=0.040000 % Pass band amplitude response ripple
Wap=10 % Pass band amplitude response weight
Watu=0.05 % Upper transition band amplitude response weight
fasu=0.25 % Upper stop band amplitude response edge
Arsu=0.040000 % Upper stop band amplitude response ripple
Wasu=20 % Upper stop band amplitude response weight
tp=10 % Pass band nominal group delay
tpr=0.2 % Pass band nominal group delay ripple
Wtp=0.5 % Pass band group delay response weight
pp=0.5 % Pass band nominal phase
ppr=0.002 % Pass band phase peak-to-peak-ripple
Wpp=1 % Pass band phase response weight
```

The Octave script *tarczynski_parallel_allpass_bandpass_differentiator_test.m* designs an initial band-pass differentiator correction filter with the *WISE* method described in Section 8.1.5. Figure 10.38. shows the initial band-pass differentiator filter response. The phase response shown is adjusted for the nominal delay.

After PCLS optimisation, the denominator polynomials of the all-pass filters in the band-pass differentiator correction filter are:

```
Da1 = [   1.0000000000,  -0.0428781431,   0.1012499778,   0.1667827057, ...
         -0.4555616763,  -0.2095208882,   0.2736510543,  -0.1287811368, ...
         -0.1069509105,   0.1367996596,  -0.0743304114,   0.0227667826 ]';
```

```
Db1 = [   1.0000000000,  -0.2151155618,  -0.0277416600,   0.1766284499, ...
         -0.4164298900,  -0.1475855558,   0.3579548466,  -0.1237608807, ...
         -0.1156536926,   0.1571682181,  -0.0613506914,   0.0069762701 ]';
```

The corresponding numerator and denominator polynomials of the correction filter are:

```
Nab1 = [   0.0078952563,  -0.0087890399,  -0.0041737348,  -0.0014271634, ...
          -0.0079400366,  -0.0240874472,  -0.0349815162,  -0.0267377041, ...
           0.0369283680,   0.0933101169,   0.0766527610,   0.0000000000, ...
          -0.0766527610,  -0.0933101169,  -0.0369283680,   0.0267377041, ...
           0.0349815162,   0.0240874472,   0.0079400366,   0.0014271634, ...
           0.0041737348,   0.0087890399,  -0.0078952563 ]';
```

Figure 10.38: Initial response of the band-pass differentiator filter implemented as the sum of two all-pass filters. The phase response shown is adjusted for the nominal delay.

```
Dab1 = [   1.0000000000,  -0.2579937049,   0.0827320736,   0.3228202206, ...
          -0.9182514641,  -0.2279955001,   0.6829383898,  -0.4858060904, ...
          -0.0328563232,   0.5754958375,  -0.4700831765,   0.0169249729, ...
           0.2811457190,  -0.1918024785,  -0.0430267516,   0.1312731590, ...
          -0.0730188154,  -0.0054721170,   0.0329426085,  -0.0234543129, ...
           0.0090927882,  -0.0019153069,   0.0001588272 ]';
```

Figure 10.39. shows the response error of the band-pass differentiator filter after PCLS optimisation. The phase response shown is adjusted for the nominal delay.

Figure 10.40 shows the phase responses of the parallel all-pass correction filters after PCLS SOCP optimisation.

Figure 10.41 shows the pole-zero plot of the band-pass differentiator after PCLS optimisation.

Figure 10.39: Response error of the band-pass differentiator filter implemented as the sum of two all-pass filters after PCLS SOCP optimisation. The phase response shown is adjusted for the nominal delay.

Figure 10.40: Phase responses of the band-pass differentiator filter parallel allpass filters after PCLS SOCP optimisation.

Figure 10.41: Pole-zero plot of the parallel-allpass band-pass differentiator filter 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

$$|H(\omega)|^2 = \frac{1 - \cos(\phi_1(\omega) - \phi_2(\omega))}{2}$$
$$\phi_H(\omega) = \frac{\phi_1(\omega) + \phi_2(\omega)}{2} + \frac{\pi}{2}$$

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 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=5000 % Lower stop band amplitude response weight
Wasu=2000 % 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
tp=16.000000 % Pass band nominal group-delay response (samples)
tpr=0.040000 % Pass band group-delay response ripple(samples)
Wtp=2 % Pass band group-delay response weight
```

The 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 10.42.

The denominator polynomials of the all-pass filters in the final band-pass filter are:

```
Da1 = [   1.0000000000,  -2.5325801662,   4.0959553736,  -4.3236122475, ...
          3.2470049707,  -1.6189738648,   0.4262856819,   0.0694557779, ...
         -0.0849921406,   0.0291379534,   0.0093083205 ]';
```

Figure 10.42: Band-pass filter implemented as the difference of two all-pass filters, initial response found with the WISE barrier function.

```
Db1 = [   1.0000000000,  -3.1463944023,   5.1684444621,  -5.4357765482, ...
          3.9977082748,  -1.9274524477,   0.4481937097,   0.1075092992, ...
         -0.0869000740,   0.0037468727,   0.0111599370 ]';
```

Figure 10.43 and Figure 10.44 show the PCLS optimised parallel allpass bandpass filter response.

Figure 10.45 shows the pole-zero plot of the PCLS optimised parallel allpass bandpass filter.

Figure 10.46 compares the phase responses of the parallel all-pass filters.

Figure 10.43: Band-pass filter implemented as the difference of two all-pass filters, response after PCLS SOCP optimisation.

Figure 10.44: Band-pass filter implemented as the difference of two all-pass filters, pass-band response after PCLS SOCP optimisation.

Figure 10.45: Band-pass filter implemented as the difference of two all-pass filters, pole-zero plot after PCLS SOCP optimisation.

Figure 10.46: 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
ftol=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
tp=16 % Pass band nominal group-delay response (samples)
tpr=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,  -1.5470113180,   1.5218988117,   0.3354486343, ...
         -1.7645745432,   2.1407836925,  -0.6286401530,  -0.6632373420, ...
          1.1917892848,  -0.6863193032,   0.2359509061 ]';
```

```
Db1 = [   1.0000000000,  -2.2713668379,   2.1463236694,   0.2850693270, ...
         -2.6789188603,   2.8972467153,  -0.8052815626,  -1.1034796926, ...
          1.5650684505,  -0.8587189829,   0.2407905505 ]';
```

The final filter response is shown in Figure 10.47 and Figure 10.48. The phase response shown is adjusted for the nominal delay.

Figure 10.47: 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 10.48: 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 [85] 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= 0.50 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas= 0.20 % Stop band amplitude response edge
dBas=66.00 % Stop band amplitude response ripple
Was=100000 % Stop band amplitude response weight
rho=0.992188 % Constraint on allpass pole radius
```

There are no group delay constraints.

The initial allpass filter was designed by the Octave script *tarczynski_parallel_allpass_delay_test.m* using the *WISE* method described in Section 8.1.5. The response of the initial filter is shown in Figure 10.49.

The final filter response is shown in Figure 10.50 with pass-band and stop-band details shown in Figure 10.51. The pole-zero plot of the filter is shown in Figure 10.52. The denominator polynomial of the final all-pass filter is

```
Da1 = [   1.0000000000,  -0.5306119261,   0.3600779810,   0.1924164868, ...
          0.0375479235,  -0.0530508762,  -0.0701563968,  -0.0406597376, ...
         -0.0023928238,   0.0197144079,   0.0216873176,   0.0130269420, ...
          0.0045043166 ]';
```

Figure 10.49: Parallel delay and all-pass filter, initial response found with the WISE barrier function.

Figure 10.50: Parallel delay and all-pass filter, response after PCLS SOCP optimisation.

Figure 10.51: Parallel delay and all-pass filter, detail of the pass-band and stop-band responses after PCLS SOCP optimisation.

Figure 10.52: Parallel delay and all-pass filter, pole-zero plot of the filter 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

```
ftol=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= 0.04 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas= 0.20 % Stop band amplitude response edge
dBas=40.00 % 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 coefficient step-size
```

There are no group delay constraints.

The final filter response is shown in Figure 10.53 with pass-band and stop-band details shown in Figure 10.54. Figure 10.55 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,  -0.4846639502,   0.3786076859,   0.1798998290, ...
          0.0155034908,  -0.0563195121,  -0.0455797014,  -0.0042299378, ...
          0.0251832874,   0.0302992689,   0.0078471979,  -0.0003823938, ...
          0.0000123701 ]';
```

Figure 10.53: Parallel delay and all-pass filter, response after PCLS SQP optimisation.

Figure 10.54: Parallel delay and all-pass filter, detail of the pass-band and stop-band responses after PCLS SQP optimisation.

Figure 10.55: 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.

Figure 10.56: Parallel delay and all-pass filter, with delay constraint, initial response found with the WISE barrier function.

Design of an IIR filter as the sum of a delay and an all-pass filter represented in pole-zero form using SOCP with a group delay constraint

The Octave script *parallel_allpass_flat_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 with a constraint on the pass-band 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=0.0001 % Tolerance on coefficient update vector
ctol=1e-06 % Tolerance on constraints
n=1000 % Frequency points across the band
m=12 % Allpass filter denominator order
V=2 % Allpass filter no. of real poles
Q=10 % Allpass filter no. of complex poles
R=1 % Allpass filter decimation
DD=11 % Parallel delay
fap= 0.17 % Pass band amplitude response edge
dBap= 0.55 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
Wat=1 % Transition band amplitude response weight
fas= 0.20 % Stop band amplitude response edge
dBas=40.00 % Stop band amplitude response ripple
Was=100 % Stop band amplitude response weight
ftp= 0.15 % Pass band group delay response edge
td=10.5 % Pass band nominal group delay
tdr=0.7 % Pass band nominal group delay ripple
Wtp=1 % Pass band group delay response weight
rho=0.992188 % Constraint on allpass pole radius
```

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 10.56.

The final filter response is shown in Figure 10.57 with pass-band and stop-band details shown in Figure 10.58. The pole-zero plot of the filter is shown in Figure 10.59. The denominator polynomial of the final all-pass filter is

Figure 10.57: Parallel delay and all-pass filter, with delay constraint, response after PCLS SOCP optimisation.

```
Da1 = [   1.0000000000,  -0.3546496991,   0.4763258615,   0.2299780453, ...
          0.0269068157,  -0.0645171213,  -0.0526779520,  -0.0054587571, ...
          0.0285812992,   0.0174527983,   0.0066967347,  -0.0179407959, ...
         -0.0119021797 ]';
```

Figure 10.58: Parallel delay and all-pass filter, with delay constraint, detail of the pass-band and stop-band responses after PCLS SOCP optimisation.

Figure 10.59: Parallel delay and all-pass filter, with delay constraint, pole-zero plot of the filter after PCLS SOCP optimisation.

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 [144, 145] 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-08 % Tolerance on constraints
n=500 % Frequency points across the band
ma=11 % Allpass model filter A denominator order
K=100 % Scale factor
Va=5 % Allpass model filter A no. of real poles
Qa=6 % Allpass model filter A no. of complex poles
Ra=2 % Allpass model filter A decimation
mb=11 % Allpass model filter B denominator order
Vb=5 % Allpass model filter B no. of real poles
Qb=6 % Allpass model filter B no. of complex poles
Rb=2 % Allpass model filter B decimation
fap=0.24 % Pass band amplitude response edge
dBap=0.000010 % Pass band amplitude response ripple
Wap=1 % Pass band amplitude response weight
fas=0.26 % Stop band amplitude response edge
dBas=100.000000 % 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.

At first, 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 $flat_delay = false$. These initial allpass filters have 3 real poles and 4 complex pole pairs. I find that better stop-band attenuation is obtained by modifying these initial allpass filters to use 5 real poles and 3 complex pole pairs. The response after PCLS optimisation with the *SeDuMi* SOCP solver is shown in Figure 10.60 and Figure 10.61. The denominator polynomials of the two all-pass filters are

```
Da1 = [   1.0000000000,   0.0000000000,   1.8208190117,   0.0000000000, ...
          0.2658910834,   0.0000000000,  -0.9415621239,  -0.0000000000, ...
         -0.2627220629,  -0.0000000000,   0.1534532368,   0.0000000000, ...
          0.0205443329,   0.0000000000,  -0.0118105147,  -0.0000000000, ...
          0.0000150963,   0.0000000000,   0.0003162827,   0.0000000000, ...
          0.0000864028,   0.0000000000,  -0.0000668071 ]';
```

and

```
Db1 = [   1.0000000000,   0.0000000000,   2.3207486699,   0.0000000000, ...
          1.0512779731,   0.0000000000,  -0.9736773292,  -0.0000000000, ...
         -0.6919929275,  -0.0000000000,   0.1126148345,   0.0000000000, ...
          0.0901594545,   0.0000000000,  -0.0143176157,  -0.0000000000, ...
         -0.0035577637,  -0.0000000000,   0.0005611501,   0.0000000000, ...
          0.0002257587,   0.0000000000,  -0.0000483526 ]';
```

Figure 10.60: Polyphase combination of two allpass filters, response after PCLS SOCP optimisation.

Figure 10.61: 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
rho=0.968750 % Constraint on allpass pole radius
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
ftp=0.22 % Pass band group delay response edge
td=22 % Pass band nominal group delay
tdr=0.08 % Pass band nominal group delay ripple
Wtp=1 % Pass band group delay response weight
fas=0.28 % Stop band amplitude response edge
dBas=60.000000 % Stop band amplitude response ripple
Was=1 % Stop band amplitude response weight
```

In this case the script ignores the pass band amplitude, optimises the stop band amplitude and optimises for a flat filter group delay across the pass band.

The initial parallel allpass filters are designed by the Octave script *tarczynski_polyphase_allpass_test.m* with the *WISE* method described in Section 8.1.5. In this case that script has $flat_delay = true$. Figure 10.62 shows the response after PCLS optimisation with the *SeDuMi* SOCP solver. Figure 10.63 shows the pass band response. The denominator polynomials of the all-pass filters are

```
Da1 = [   1.0000000000,   0.0000000000,  -0.0074821067,  -0.0000000000, ...
          0.0029564945,   0.0000000000,  -0.0014377948,  -0.0000000000, ...
          0.0008328235,   0.0000000000,  -0.0006389406,  -0.0000000000, ...
          0.0006164901,   0.0000000000,  -0.0004976485,  -0.0000000000, ...
          0.0002808878,   0.0000000000,  -0.0000654225,  -0.0000000000, ...
         -0.0001320817,  -0.0000000000,  -0.0000493925 ]';
```

and

```
Db1 = [   1.0000000000,   0.0000000000,   0.4892918730,   0.0000000000, ...
         -0.1209918478,  -0.0000000000,   0.0576049681,   0.0000000000, ...
         -0.0331406802,  -0.0000000000,   0.0206058879,   0.0000000000, ...
         -0.0132386320,  -0.0000000000,   0.0087073041,   0.0000000000, ...
         -0.0057778117,  -0.0000000000,   0.0037875051,   0.0000000000, ...
         -0.0023331808,  -0.0000000000,   0.0020234556 ]';
```

Figure 10.62: Polyphase combination of two all-pass filters with flat pass-band delay, response after PCLS SOCP optimisation.

Figure 10.63: Polyphase combination of two all-pass filters with flat pass-band delay, pass-band response after PCLS SOCP optimisation.

10.3 Design of IIR Schur lattice filters

There is a large literature describing adaptive IIR lattice filters. For example, *Regalia* [171] 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* [178]). 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 5.8 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 5.8 is lower Hessenberg. The matrix resolvent, $(zI - A)^{-1}$ is calculated by direct matrix inversion rather than *Le Verrier's* algorithm (see Algorithm 1.6). *Xu Zhong* [260] 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 lattice coefficients derived in Chapter 5 for the normalised-scaled lattice. The results of this section are intended to be used in the following Part III to optimise the choice of fixed-point lattice coefficients.

10.3.1 Design of one-multiplier IIR Schur lattice filters with SQP optimisation

Design of a one-multiplier IIR 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:

```
ftol=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 10.64 and 10.65 show the overall and passband response of the filter after SQP PCLS optimisation. Figure 10.66 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:

```
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 ];
```

The PCLS SQP optimised Schur one-multiplier low-pass filter numerator and denominator polynomials are:

```
N2 = [  -0.0052625492,   0.0027205868,   0.0029400838,   0.0075076108, ...
        -0.0035447126,  -0.0182712642,  -0.0123464150,   0.0278035927, ...
         0.0723506870,   0.0702344183,   0.0382923739 ];


D2 = [   1.0000000000,  -2.2390960503,   2.8579123328,  -2.3665133117, ...
         1.3229965619,  -0.4769209831,   0.0878212134,   0.0000000000, ...
        -0.0000000000,  -0.0000000000,   0.0000000000 ];
```

Figure 10.64: Schur one-multiplier lattice lowpass filter, response after SQP PCLS optimisation.

Figure 10.65: Schur one-multiplier lattice lowpass filter, passband response after SQP PCLS optimisation.

Figure 10.66: Schur one-multiplier lattice lowpass filter, pole-zero plot after SQP PCLS optimisation.

Design of a one-multiplier IIR Schur lattice band-pass R=2 filter using SQP

The Octave script *schurOneMlattice_sqp_slb_bandpass_R2_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:

```
ftol_mmse=2e-05 % Tolerance on coef. update for MMSE
ftol_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_R2_test.m*, as shown in Section 8.2.6. The denominator polynomial of the filter has coefficients in z^{-2} only.

Figure 10.67: Response of the R=2 Schur one-multiplier lattice band-pass filter after SQP MMSE optimisation.

Figure 10.67 shows the overall response of the band-pass filter after SQP MMSE optimisation. Figures 10.68 and 10.69 show the overall and passband response of the band-pass filter after SQP PCLS optimisation. Figure 10.70 shows the pole-zero plot of the band-pass filter after SQP PCLS optimisation.

The SQP PCLS optimised $R = 2$ 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 ];
```

The corresponding overall transfer function numerator and denominator polynomial coefficients are:

Figure 10.68: Response of the R=2 Schur one-multiplier lattice band-pass filter after SQP PCLS optimisation.

Figure 10.69: Passband response of the R=2 Schur one-multiplier lattice band-pass filter after SQP PCLS optimisation.

Schur 1-multiplier SQP PCLS:fapl=0.1,fapu=0.2,dBap=2,fasl=0.05,fasu=0.25,dBas=36,Wtp=1,Was=100000

Figure 10.70: Pole-zero plot of the R=2 Schur one-multiplier lattice band-pass filter after SQP PCLS optimisation.

```
N2 = [   0.0058051915,   0.0012706269,   0.0118334335,   0.0176878978, ...
         0.0381312659,   0.0319388656,   0.0251518542,   0.0069289127, ...
         0.0056626625,  -0.0173425626,  -0.0608005653,  -0.1007087591, ...
        -0.0769181188,   0.0235594241,   0.1211872739,   0.1432972866, ...
         0.0605790600,  -0.0337599144,  -0.0851805010,  -0.0611581082, ...
        -0.0269399170 ];


D2 = [   1.0000000000,   0.0000000000,   1.5714300377,   0.0000000000, ...
         1.8072457649,   0.0000000000,   1.7877578723,   0.0000000000, ...
         1.5881253280,   0.0000000000,   1.1504604633,   0.0000000000, ...
         0.7458129561,   0.0000000000,   0.4015652852,   0.0000000000, ...
         0.1861402747,   0.0000000000,   0.0599184246,   0.0000000000, ...
         0.0150432836 ];
```

Design of a one-multiplier IIR Schur lattice Hilbert R=2 filter using SQP

The Octave script *schurOneMlattice_sqp_slb_hilbert_R2_test.m* implements the design of an $R = 2$ IIR one-multiplier Schur lattice Hilbert filter with SQP and PCLS. The specification of the filter is:

```
n=400 % Frequency points across the band
dmax=0.05 % Constraint on coefficient update size
ftol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-05 % Tolerance on constraints
dBar=0.067 % Amplitude response peak-to-peak ripple
dBat=0.067 % Amplitude transition peak-to-peak ripple
Wap=1 % Amplitude response weight
Wat=0.1 % Amplitude transition weight
fpt=0.06 % Phase response transition edge
pp=5 % Phase response nominal phase(rad./pi)
ppr=0.032 % Phase response peak-to-peak ripple(rad./pi)
Wpp=1 % Phase response weight
Wpt=0 % Phase response transition weight
ftt=0.08 % Group delay response transition edge
```

Figure 10.71: R=2 Schur one-multiplier lattice Hilbert filter, response after SQP PCLS optimisation.

```
td=5.5 % Nominal filter group delay(samples)
tdr=0.2 % Group delay peak-to-peak ripple(samples)
Wtp=0.005 % Group delay weight
Wtt=0 % Group delay transition weight
```

The initial filter is that of the Octave script *iir_sqp_slb_hilbert_R2_test.m* designed by the method of *Tarczynski et al.* with the Octave script *tarczynski_hilbert_R2_test.m*, as shown in Section 8.3.1. The denominator polynomial of the filter has coefficients in z^{-2} only. Figure 10.71 shows the response of the $R = 2$ Hilbert filter after SQP PCLS optimisation. The phase response shown has been adjusted by the nominal pass-band group delay, $\omega\tau_p$. The values shown on the phase axis of the phase response plot are multiples of π radians. Figure 10.72 shows the pole-zero plot of the $R = 2$ Hilbert filter after SQP PCLS optimisation.

The SQP PCLS optimised $R = 2$ Schur one-multiplier all-pass lattice and numerator tap coefficients of the Hilbert filter are:

```
k2 = [   0.0000000000,  -0.8105864394,   0.0000000000,   0.2655181645, ...
         0.0000000000,  -0.0336215102,   0.0000000000,   0.0012379481, ...
         0.0000000000,  -0.0003065055,   0.0000000000,   0.0001110418 ];


epsilon2 = [  0, -1,  0, -1, ...
              0,  1,  0, -1, ...
              0,  1,  0, -1 ];


p2 = [   2.2737060699,   2.2737060699,   0.7354111726,   0.7354111726, ...
         0.9653257368,   0.9653257368,   0.9983458736,   0.9983458736, ...
         0.9995825399,   0.9995825399,   0.9998889644,   0.9998889644 ];


c2 = [  -0.0437647324,  -0.0526095103,  -0.1506187275,  -0.1846670626, ...
        -0.1806322200,  -0.2725453506,  -0.6912602247,   0.5823145464, ...
         0.1563236592,   0.0717407970,   0.0366475456,   0.0185442080, ...
         0.0085434874 ];
```

The corresponding denominator and numerator transfer function polynomial coefficients are:

Figure 10.72: R=2 Schur one-multiplier lattice Hilbert filter, pole-zero plot after SQP PCLS optimisation.

```
N2 = [   0.0085434874,   0.0185421488,   0.0278028376,   0.0525237812, ...
         0.1209067958,   0.5127157259,  -0.8411010704,  -0.8436009173, ...
         0.5854879321,   0.3108113374,  -0.1404023796,  -0.0507067446, ...
         0.0074778235 ];

D2 = [   1.0000000000,   0.0000000000,  -1.0347810198,   0.0000000000, ...
         0.3003897727,   0.0000000000,  -0.0349984170,   0.0000000000, ...
         0.0015884698,   0.0000000000,  -0.0004214094,   0.0000000000, ...
         0.0001110418 ];
```

Figure 10.73: Response of the Schur one-multiplier lattice lowpass filter after PCLS SOCP optimisation.

10.3.2 Design of one-multiplier IIR Schur lattice filters with SOCP optimisation

Design of a one-multiplier IIR 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:

```
ftol=1e-05 % Tolerance on coefficient update vector
ctol=1e-06 % Tolerance on constraints
n=500 % Frequency points across the band
% length(c0)=10 % Tap coefficients
% length(k0~=0)=9 % 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=9 % Nominal pass band filter group delay
tpr=0.02 % Delay pass band peak-to-peak ripple
Wtp=1 % Delay pass band weight
fas=0.35 % Amplitude stop band edge
dBas=47 % Amplitude stop band peak-to-peak ripple
Was=100 % Amplitude stop band weight
```

The initial filter is a heavily modified version of the "IPZS-1" filter of Section 8.2.3. Figures 10.73 and 10.74 show the overall and passband response after SOCP PCLS optimisation. Figure 10.75 shows the pole-zero plot of the resulting filter after SOCP PCLS optimisation.

The PCLS SOCP optimised Schur one-multiplier all-pass lattice and numerator tap coefficients of the low-pass filter are:

```
k2 = [  -0.6805896882,   0.6554707502,  -0.5345555025,   0.3522006886, ...
        -0.1701042028,   0.0464958011,   0.0000000000,  -0.0000000000, ...
        -0.0000000000 ];
```

Figure 10.74: Passband response of the Schur one-multiplier lattice lowpass filter after PCLS SOCP optimisation.

Figure 10.75: Pole-zero plot of the Schur one-multiplier lattice lowpass filter after PCLS SOCP optimisation.

```
epsilon2 = [ -1, -1, -1, -1, ...
              1, -1, -1,  1, ...
              1 ];


p2 = [   1.0572014786,   0.4608943466,   1.0102970565,   0.5564059140, ...
         0.8038815715,   0.9545365441,   1.0000000000,   1.0000000000, ...
         1.0000000000 ];


c2 = [   0.4221917396,   0.9215162682,   0.0845637922,  -0.1284245257, ...
        -0.0468041909,   0.0148832977,   0.0204119363,   0.0022524223, ...
        -0.0078339795,  -0.0041454449 ];
```

The PCLS SOCP optimised Schur one-multiplier low-pass filter numerator and denominator polynomials are:

```
N2 = [  -0.0041454449,  -0.0006492081,   0.0079134864,   0.0073784188, ...
        -0.0088622901,  -0.0303332588,  -0.0027853027,   0.0955230298, ...
         0.1350770445,   0.0866034669 ];


D2 = [   1.0000000000,  -1.7331725827,   1.9097044166,  -1.4065810831, ...
         0.7229035801,  -0.2503217091,   0.0464958011 ];
```

Design of a one-multiplier IIR Schur lattice low-pass R=2 filter using SOCP

The Octave script *schurOneMlattice_socp_slb_lowpass_R2_test.m* implements the design of a lowpass $R = 2$ IIR one-multiplier Schur lattice filter with denominator polynomial coefficients only in z^{-2} using the *SeDuMi* SOCP solver with PCLS constraints. The filter specification is[b]:

```
tol=1e-09 % Tolerance on WISEJ convergence
ftol=1e-05 % Tolerance on coefficient update vector
ctol=1e-07 % Tolerance on constraints
norder=16 % Filter order
n=1000 % Frequency points across the band
% length(c0)=17 % Tap coefficients
% length(k0~=0)=16 % Num. non-zero all-pass coef.s
rho=0.992188 % Constraint on allpass coefficients
fap=0.15 % Amplitude pass band edge
dBap=0.06 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
fas=0.175 % Amplitude stop band edge
dBas=60 % Amplitude stop band peak-to-peak ripple
Wasi=1000 % Amplitude stop band weight (initial filter)
Was=1e+06 % Amplitude stop band weight
```

The initial filter is designed by the *WISE* method described in Section 8.1.5. Figure 10.76 shows the pass-band and stop-band responses after SOCP PCLS optimisation. Figure 10.77 shows the pole-zero plot of the filter after SOCP PCLS optimisation.

The PCLS SOCP optimised $R = 2$ Schur one-multiplier all-pass lattice and numerator tap coefficients of the low-pass filter are:

```
k2 = [   0.0000000000,   0.2971455815,   0.0000000000,   0.9649373809, ...
         0.0000000000,  -0.0176330813,   0.0000000000,   0.6266880627, ...
         0.0000000000,  -0.2811954488,   0.0000000000,   0.2342811916, ...
         0.0000000000,  -0.1085693370,   0.0000000000,   0.0320248710 ];


epsilon2 = [  0, -1,  0,  1, ...
              0,  1,  0, -1, ...
              0,  1,  0, -1, ...
              0,  1,  0, -1 ];
```

[b]For comparison, Figure 10.14 shows the magnitude response of an elliptic low-pass filter with a similar frequency specification.

Figure 10.76: Pass-band and stop-band responses of the R=2 Schur one-multiplier lattice lowpass filter after PCLS SOCP optimisation.

Figure 10.77: Pole-zero plot of the R=2 Schur one-multiplier lattice lowpass filter after PCLS SOCP optimisation.

```
p2 = [   1.3288937338,   1.3288937338,   1.8053105669,   1.8053105669, ...
         0.2411569284,   0.2411569284,   0.2454474290,   0.2454474290, ...
         0.5123593764,   0.5123593764,   0.6840328722,   0.6840328722, ...
         0.8684591016,   0.8684591016,   0.9684718851,   0.9684718851 ];

c2 = [   0.0486057108,   0.0946217386,   0.0620057754,   0.0060095167, ...
        -0.2968631558,  -0.5836966363,  -0.5210290678,  -0.1121307958, ...
         0.2236145424,   0.4257078680,   0.3841466326,   0.3520139463, ...
         0.2012879361,   0.1208302368,   0.0510185781,   0.0174815598, ...
         0.0032567921 ];
```

The corresponding transfer function numerator and denominator polynomials are:

```
N2 = [   0.0032567921,   0.0169217151,   0.0503122292,   0.1091403588, ...
         0.1931201391,   0.2921818199,   0.3860541649,   0.4504091243, ...
         0.4673015967,   0.4328120339,   0.3550213269,   0.2565211448, ...
         0.1596529730,   0.0847274617,   0.0342632287,   0.0095165503, ...
         0.0003521044 ];

D2 = [   1.0000000000,   0.0000000000,   0.2847938913,   0.0000000000, ...
         1.5954618186,   0.0000000000,  -0.2220327677,   0.0000000000, ...
         0.8516755753,   0.0000000000,  -0.3692602888,   0.0000000000, ...
         0.2511113903,   0.0000000000,  -0.0993375015,   0.0000000000, ...
         0.0320248710 ];
```

Design of an alternative one-multiplier IIR Schur lattice low-pass R=2 filter using SOCP

The Octave script *schurOneMlattice_socp_slb_lowpass_R2_alternate_test.m* uses the *SeDuMi* SOCP solver to design an $R = 2$ lowpass filter, similar to that shown in Section 9.9. The filter specification is:

```
tol=1e-09 % Tolerance on WISEJ convergence
ftol=1e-05 % Tolerance on coefficient update vector
ctol=1e-07 % Tolerance on constraints
norder=12 % Filter order
n=1000 % Frequency points across the band
% length(c0)=13 % Tap coefficients
% length(k0~=0)=12 % Num. non-zero all-pass coef.s
rho=0.992188 % Constraint on allpass coefficients
fap=0.1 % Amplitude pass band edge
dBap=0.06 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
fas=0.2 % Amplitude stop band edge
dBas=60.5 % Amplitude stop band peak-to-peak ripple
Wasi=1000 % Amplitude stop band weight (initial filter)
Was=1e+06 % Amplitude stop band weight
```

The initial filter was calculated with the WISE method of *Tarczynski et al.* described in Section 8.1.5. The response of the initial filter is shown in Figure 10.78. The response of the filter after PCLS SOCP optimimisation is shown in Figure 10.79. The corresponding pole-zero plot is shown in Figure 10.80. The optimised Schur one-multiplier lattice filter coefficients are:

```
k2 = [   0.0000000000,  -0.3658734354,   0.0000000000,   0.7240146906, ...
         0.0000000000,  -0.2884187687,   0.0000000000,   0.0191242697, ...
         0.0000000000,   0.1035960600,   0.0000000000,  -0.0482725647 ];

epsilon2 = [  0,  1,  0,  1, ...
              0,  1,  0, -1, ...
              0, -1,  0,  1 ];

c2 = [  -0.1070605550,  -0.1134239795,  -0.0195151824,   0.0510430340, ...
         0.2781924693,   0.3550008164,   0.2609056733,   0.2229217556, ...
         0.1587879443,   0.0974266577,   0.0442116664,   0.0174223324, ...
         0.0047077162 ];
```

Figure 10.78: Initial response of a Schur one-multiplier lattice low-pass R=2 filter after WISE optimisation

The corresponding transfer function numerator and denominator polynomials are:

```
N2 = [   0.0047077162,   0.0165813118,   0.0380847191,   0.0691376992, ...
         0.1042040816,   0.1310871963,   0.1404319257,   0.1280459772, ...
         0.0981058162,   0.0616180032,   0.0298246457,   0.0095444728, ...
         0.0013224156 ];


D2 = [   1.0000000000,   0.0000000000,  -0.8481260499,   0.0000000000, ...
         0.8950270223,   0.0000000000,  -0.1987478745,   0.0000000000, ...
        -0.1114712259,   0.0000000000,   0.1442958759,   0.0000000000, ...
        -0.0482725647 ];
```

Figure 10.79: Response of a Schur one-multiplier lattice low-pass R=2 filter after PCLS SOCP optimisation

Figure 10.80: Pole-zero plot of a Schur one-multiplier lattice low-pass R=2 filter after PCLS SOCP optimisation

Figure 10.81: Response of the Schur one-multiplier lattice band-pass filter after SOCP PCLS optimisation.

Design of a one-multiplier IIR Schur lattice band-pass filter using SOCP

The Octave script *schurOneMlattice_socp_slb_bandpass_test.m* implements the design of a band-pass IIR one-multiplier Schur lattice filter with SOCP and PCLS. The specification of the filter is:

```
ftol=0.0001 % Tolerance on coef. update
ctol=1e-06 % Tolerance on constraints
n=500 % Frequency points across the band
% length(c0)=21 % Tap coefficients
% sum(k0~=0)=20 % Num. non-zero all-pass coef.s
rho=0.992188 % Constraint on allpass coefficients
fapl=0.08 % Amplitude pass band lower edge
fapu=0.22 % Amplitude pass band upper edge
dBap=0.08 % 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=0.032 % Delay pass band peak-to-peak ripple
Wtp=1 % Delay pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
dBas=40 % Amplitude stop band peak-to-peak ripple
Wasl=20000 % Ampl. lower stop band weight
Wasu=10000 % Ampl. upper stop band weight
```

The initial filter is that found by the Octave script *tarczynski_bandpass_R1_test.m*. Figure 10.81 shows the response of the band-pass filter after SOCP PCLS optimisation and Figure 10.82 shows the pole-zero plot of the band-pass filter.

The SOCP PCLS optimised Schur one-multiplier allpass lattice and numerator tap coefficients of the band-pass filter are:

```
k3 = [  -0.8819779925,   0.9720595279,  -0.6809223793,   0.7957904491, ...
        -0.5918788800,   0.8242030694,  -0.6591282107,   0.7616138100, ...
```

Figure 10.82: Pole-zero plot of the Schur one-multiplier lattice band-pass filter after SOCP PCLS optimisation.

```
        -0.5894634045,   0.5193032064,  -0.1853436630,   0.0465082133, ...
         0.0242030328,   0.0198009633,  -0.0031593646,   0.0117697720, ...
        -0.0256753340,   0.0297303315,  -0.0182479692,   0.0040375593 ];


epsilon3 = [  1,  1,  1, -1, ...
             -1,  1,  1, -1, ...
             -1, -1,  1, -1, ...
             -1, -1,  1, -1, ...
              1, -1,  1, -1 ];


p3 = [   0.2504974297,   1.0002971829,   0.1190654906,   0.2732823369, ...
         0.8104036090,   0.4103372003,   0.1273825516,   0.2810311028, ...
         0.7639574923,   0.3882575110,   0.6902501039,   0.8326095765, ...
         0.8722766442,   0.8936501670,   0.9115240130,   0.9144084133, ...
         0.9252348794,   0.9493035468,   0.9779589568,   0.9959705588 ];


c3 = [  -0.1927560779,  -0.1251201990,  -0.8796562133,  -0.6257463310, ...
         0.0417666152,   0.9949254913,   1.4659623808,  -0.6654235793, ...
        -0.1801057425,  -0.0155582699,   0.0143028369,  -0.0234858573, ...
        -0.0179463428,   0.0183529854,   0.0305094312,   0.0115043387, ...
        -0.0025893373,   0.0027140534,   0.0088123238,   0.0044021652, ...
        -0.0090423196 ];
```

The corresponding overall transfer function numerator and denominator polynomial coefficients are:

```
N3 = [  -0.0090423196,   0.0568956561,  -0.1734940185,   0.3405233949, ...
        -0.4778908697,   0.5131658365,  -0.4400040439,   0.3033929616, ...
        -0.1592038938,   0.0621737425,  -0.0368458427,   0.0317948591, ...
        -0.0179106511,   0.0237339118,  -0.0157652862,  -0.0219152853, ...
         0.0388803090,  -0.0408081857,   0.0464330150,  -0.0336517719, ...
         0.0098689938 ];
```

```
D3 = [   1.0000000000,  -5.8072781591,  17.3240446725, -34.5076431881, ...
        50.7060450244, -57.4077294257,  51.0412306057, -35.8205720374, ...
        19.9768934954,  -9.4202109524,   5.0142496643,  -4.3182588742, ...
         4.6694830446,  -4.6288642895,   3.8850394693,  -2.6974724988, ...
         1.5097529031,  -0.6536575045,   0.2056347656,  -0.0416949016, ...
         0.0040375593 ];
```

Design of an R=2 one-multiplier IIR Schur lattice band-pass filter using SOCP

The Octave script *schurOneMR2lattice_socp_slb_bandpass_test.m* implements the design of an $R = 2$ band-pass IIR one-multiplier Schur lattice filter with SOCP and PCLS. The filter denominator polynomial has coefficients only in z^{-2}. The specification of the filter is:

```
ftol=0.0001 % Tolerance on coef. update
ctol=1e-07 % 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
rho=0.992188 % Constraint on allpass coefficients
fapl=0.095 % Amplitude pass band lower edge
fapu=0.205 % Amplitude pass band upper edge
dBap=0.1 % 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=35 % Amplitude stop band peak-to-peak ripple
Wasl=1000 % Ampl. lower stop band weight
Wasu=1000 % Ampl. upper stop band weight
```

The initial filter is that found by the Octave script *tarczynski_bandpass_R2_test.m*. Figure 10.83 shows the response of the $R = 2$ band-pass filter after SOCP PCLS optimisation and Figure 10.84 shows the pole-zero plot of the $R = 2$ band-pass filter.

The SOCP PCLS optimised Schur one-multiplier allpass lattice and numerator tap coefficients of the $R = 2$ band-pass filter are:

```
k3 = [   0.0000000000,   0.7397152222,   0.0000000000,   0.4544367025, ...
         0.0000000000,   0.2562888117,   0.0000000000,   0.4790518007, ...
         0.0000000000,   0.3267156653,   0.0000000000,   0.4115581409, ...
         0.0000000000,   0.2520363779,   0.0000000000,   0.2346150129, ...
         0.0000000000,   0.0932190281,   0.0000000000,   0.0559853590 ];

epsilon3 = [  0,  1,  0, -1, ...
              0, -1,  0,  1, ...
              0, -1,  0,  1, ...
              0, -1,  0, -1, ...
              0, -1,  0, -1 ];

p3 = [   1.1868985633,   1.1868985633,   0.4590911358,   0.4590911358, ...
         0.7495899364,   0.7495899364,   0.9742408504,   0.9742408504, ...
         0.5781917813,   0.5781917813,   0.8116365343,   0.8116365343, ...
         0.5240394494,   0.5240394494,   0.6780039396,   0.6780039396, ...
         0.8611087942,   0.8611087942,   0.9454975690,   0.9454975690 ];

c3 = [  -0.0862776853,  -0.0416408600,   0.2463030260,   0.6337092508, ...
         0.3188307030,  -0.0451247265,  -0.2416205385,  -0.2299924650, ...
        -0.1000897798,   0.0790169743,   0.0405281471,  -0.0023029170, ...
         0.0113820725,   0.0694472719,   0.0460850969,   0.0066951811, ...
        -0.0104628535,   0.0011884331,   0.0026604617,   0.0000420009, ...
        -0.0139862926 ];
```

Figure 10.83: Response of the R=2 Schur one-multiplier lattice band-pass filter after SOCP PCLS optimisation.

Figure 10.84: Pole-zero plot of the R=2 Schur one-multiplier lattice band-pass filter after SOCP PCLS optimisation.

The corresponding overall transfer function numerator and denominator polynomial coefficients are:

```
N3 = [  -0.0139862926,   0.0000396495,  -0.0226083768,   0.0010883210, ...
        -0.0357051475,   0.0062738244,  -0.0153768544,   0.0438191130, ...
         0.0080416396,   0.0641568277,   0.0508825337,   0.1045152188, ...
         0.0271751857,  -0.0452799886,  -0.1503392199,  -0.1169878739, ...
        -0.0195280813,   0.1015628105,   0.1322403887,   0.0921235370, ...
         0.0346242600 ];


D3 = [   1.0000000000,   0.0000000000,   1.7960364728,   0.0000000000, ...
         2.2340772342,   0.0000000000,   2.5226374348,   0.0000000000, ...
         2.5821373732,   0.0000000000,   2.1767206087,   0.0000000000, ...
         1.6355183268,   0.0000000000,   0.9922372165,   0.0000000000, ...
         0.5233379253,   0.0000000000,   0.1934785928,   0.0000000000, ...
         0.0559853590 ];
```

Design of a pipelined IIR Schur one-multiplier lattice band-pass filter using SOCP

The Octave script *schurOneMlatticePipelined_socp_slb_bandpass_test.m* designs a band-pass Schur one-multiplier lattice filter implemented in the pipelined form shown in Section 5.6.3. The initial filter is designed by the Octave script *tarczynski_bandpass_R1_test.m*. The filter specification is:

```
ftol=0.0001 % Tolerance on coef. update
ctol=1e-07 % Tolerance on constraints
n=500 % Frequency points across the band
% length(c0)=21 % Tap coefficients
rho=0.992188 % Constraint on allpass coefficients
fapl=0.095 % Amplitude pass band lower edge
fapu=0.205 % Amplitude pass band upper edge
dBap=0.0872961 % 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=40 % Amplitude stop band peak-to-peak ripple
Wasl=1000 % Ampl. lower stop band weight
Wasu=1000 % Ampl. upper stop 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=0.04 % Delay pass band peak-to-peak ripple
Wtp=0.1 % Delay pass band weight
```

The PCLS SOCP optimised pipelined Schur one-multiplier lattice band-pass filter coefficients are:

```
k2 = [  -0.7732552141,   0.8924652873,  -0.6768976385,   0.8078651061, ...
        -0.6160082572,   0.8106403286,  -0.6716700024,   0.7670437994, ...
        -0.6224364997,   0.5020684105,  -0.2420046076,   0.0825660294, ...
         0.0083760162,   0.0050786411,   0.0105093992,   0.0119594503, ...
         0.0063017769,   0.0056047300,  -0.0047177095,  -0.0060632591 ]';


epsilon0 = [   1.0000000000,   1.0000000000,   1.0000000000,  -1.0000000000, ...
              -1.0000000000,   1.0000000000,   1.0000000000,  -1.0000000000, ...
              -1.0000000000,  -1.0000000000,   1.0000000000,  -1.0000000000, ...
              -1.0000000000,  -1.0000000000,  -1.0000000000,  -1.0000000000, ...
              -1.0000000000,   1.0000000000,  -1.0000000000,  -1.0000000000 ];


c2 = [  -0.0847547765,  -0.0629943999,  -0.7086003038,  -0.5713757353, ...
         0.0066337172,   0.7532718541,   1.2479313248,  -0.5675889276, ...
        -0.1952132846,  -0.0138995769,  -0.0010654616,   0.0209749603, ...
        -0.0155210901,   0.0184747433,   0.0283358552,   0.0239095902, ...
        -0.0026288105,  -0.0131355060,   0.0075055110,   0.0035298704, ...
        -0.0089283144 ]';
```

```
kk2 = [  -0.7776703164,  -0.6715422983,  -0.5461494415,  -0.4978856637, ...
         -0.5139685801,  -0.5317592147,  -0.5301385544,  -0.4895238451, ...
         -0.3256998659,  -0.1380772341,  -0.0416771647,  -0.0169295799, ...
          0.0062884007,  -0.0223263149,   0.0108369126,  -0.0294629737, ...
          0.0063041742,  -0.0076993247,   0.0055016976 ]';


ck2 = [  -0.0975674484,   0.0000000000,  -0.4638758589,   0.0000000000, ...
          0.7315975997,   0.0000000000,  -0.4837420662,   0.0000000000, ...
         -0.0514568268,   0.0000000000,   0.0355679288,   0.0000000000, ...
          0.0034167758,   0.0000000000,   0.0128971272,   0.0000000000, ...
          0.0144812443,   0.0000000000,  -0.0008743842 ]';
```

The corresponding overall transfer function numerator and denominator polynomials are:

```
N2 = [  -0.0089283144,   0.0569459060,  -0.1772126716,   0.3566485072, ...
        -0.5159117603,   0.5744931302,  -0.5149361324,   0.3766415378, ...
        -0.2173430128,   0.0970681457,  -0.0480094412,   0.0250217752, ...
         0.0005459925,  -0.0015218840,   0.0082897787,  -0.0370226215, ...
         0.0468549081,  -0.0447444867,   0.0459137944,  -0.0337590519, ...
         0.0120991200,  -0.0002769635,  -0.0006140638,  -0.0000472241, ...
        -0.0000005831,   0.0000000018,   0.0000000001,   0.0000000000, ...
         0.0000000000,   0.0000000000,   0.0000000000 ];


D2 = [   1.0000000000,  -5.8848344013,  17.7987850211, -36.0280476907, ...
        53.9458100445, -62.5222649718,  57.2993510842, -41.8480994615, ...
        24.4378134860, -11.6269528692,   4.9784917764,  -2.4463478599, ...
         1.4819991704,  -0.7489561804,   0.0464622316,   0.4170400564, ...
        -0.5298484378,   0.3909327484,  -0.1924838671,   0.0603219763, ...
        -0.0097309828 ];
```

Figure 10.85 shows the response of the band-pass filter after PCLS optimisation and Figure 10.86 shows the pole-zero plot of the band-pass filter after PCLS optimisation.

Figure 10.85: PCLS optimised response of a pipelined Schur one-multiplier lattice band-pass filter.

Figure 10.86: Pole-zero plot of the PCLS optimised pipelined Schur one-multiplier lattice band-pass filter.

Figure 10.87: R=2 Schur one-multiplier lattice Hilbert filter, response after SOCP PCLS optimisation.

Design of an R=2 IIR Schur one-multiplier lattice Hilbert filter using SOCP

The Octave script *schurOneMlattice_socp_slb_hilbert_R2_test.m* implements the design of an $R = 2$ IIR one-multiplier Schur lattice Hilbert filter with SOCP and PCLS. The specification of the filter is:

```
n=400 % Frequency points across the band
dmax=Inf % Constraint on coefficient update size
ftol=0.0001 % Tolerance on relative coefficient update size
ctol=1e-05 % Tolerance on constraints
dBar=0.067 % Amplitude response peak-to-peak ripple
dBat=0.067 % Amplitude transition peak-to-peak ripple
Wap=1 % Amplitude response weight
Wat=0.1 % Amplitude transition weight
fpt=0.06 % Phase response transition edge
pp=5 % Phase response nominal phase(rad./pi)
ppr=0.02 % Phase response peak-to-peak ripple(rad./pi)
Wpp=1 % Phase response weight
Wpt=0 % Phase response transition weight
ftt=0.08 % Group delay response transition edge
td=5.5 % Nominal filter group delay(samples)
tdr=0.16 % Group delay peak-to-peak ripple(samples)
Wtp=0.005 % Group delay weight
Wtt=0 % Group delay transition weight
```

The initial filter is that of the Octave script *iir_sqp_slb_hilbert_R2_test.m* designed by the method of *Tarczynski et al.* with the Octave script *tarczynski_hilbert_R2_test.m*, as shown in Section 8.3.1. The denominator polynomial of the filter has coefficients in z^{-2} only. Figure 10.87 shows the response of the $R = 2$ Hilbert filter after SOCP PCLS optimisation. The phase response shown has been adjusted by the nominal pass-band group delay, $\omega\tau_p$. The values shown on the phase axis of the phase response plot are multiples of π radians. Figure 10.88 shows the pole-zero plot of the $R = 2$ Hilbert filter after SOCP PCLS optimisation.

The SOCP PCLS optimised $R = 2$ Schur one-multiplier all-pass lattice and numerator tap coefficients of the Hilbert filter are:

```
k2 = [   0.0000000000,  -0.9167025619,   0.0000000000,   0.4392844215, ...
```

Figure 10.88: R=2 Schur one-multiplier lattice Hilbert filter, pole-zero plot after SOCP PCLS optimisation.

```
         0.0000000000,  -0.0007270227,   0.0000000000,   0.0007326354, ...
         0.0000000000,  -0.0002678109,   0.0000000000,  -0.0002076647 ];


epsilon2 = [  0, -1,  0, -1, ...
              0,  1,  0, -1, ...
              0,  1,  0,  1 ];


p2 = [   2.9882650834,   2.9882650834,   0.6229562981,   0.6229562981, ...
         0.9980667373,   0.9980667373,   0.9987926184,   0.9987926184, ...
         0.9995246374,   0.9995246374,   0.9997923569,   0.9997923569 ];


c2 = [  -0.0299129634,  -0.0359370025,  -0.1941475738,  -0.2289340723, ...
        -0.1776967550,  -0.2636823925,  -0.6863246009,   0.5868552741, ...
         0.1562560346,   0.0693242207,   0.0424955665,   0.0186961405, ...
         0.0111132630 ];
```

The corresponding denominator and numerator transfer function polynomial coefficients are:

```
N2 = [   0.0111132630,   0.0186922579,   0.0278203934,   0.0446227941, ...
         0.1050074529,   0.5029369488,  -0.8729131038,  -1.0062253621, ...
         0.7960483993,   0.4770601872,  -0.1766555879,  -0.0370889510, ...
        -0.0097946981 ];


D2 = [   1.0000000000,   0.0000000000,  -1.3197157594,   0.0000000000, ...
         0.4405664182,   0.0000000000,  -0.0018115048,   0.0000000000, ...
         0.0009945794,   0.0000000000,   0.0000062475,   0.0000000000, ...
        -0.0002076647 ];
```

Design of an IIR Schur one-multiplier lattice band-pass Hilbert filter using SOCP

The Octave script *schurOneMlattice_socp_slb_bandpass_hilbert_test.m* implements the design of an IIR Schur one-multiplier lattice band-pass Hilbert filter with SOCP and PCLS. The specification of the filter is:

```
maxiter=10000 % Maximum iterations
dmax=0.1 % SQP step-size constraint
ftol=0.0001 % Tolerance on coef. update
ctol=1e-06 % Tolerance on constraints
n=1000 % Frequency points across the band
fasl=0.05 % Amplitude stop band lower edge
fapl=0.09 % Amplitude pass band lower edge
fapl=0.21 % Amplitude pass band upper edge
fasu=0.25 % Amplitude stop band upper edge
dBap=0.15 % Amplitude pass band peak-to-peak ripple(dB)
Wap=1 % Amplitude pass band weight
Watl=0.001 % Amplitude lower transition band weight
Watu=0.001 % Amplitude upper transition band weight
dBas=37 % Amplitude stop band peak-to-peak ripple(dB)
Wasl=100 % Amplitude lower stop band weight
Wasu=100 % Amplitude upper stop band weight
fppl=0.1 % Phase pass band lower edge
fppu=0.2 % Phase pass band upper edge
pp=1.5 % Nominal pass band phase(rad./pi)
ppr=0.002 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=1 % Phase pass band weight
ftpl=0.1 % Delay pass band lower edge
ftpu=0.2 % Delay pass band upper edge
tp=12 % Pass band group delay
tpr=0.2 % Pass band group delay peak-to-peak ripple
Wtp=1 % Pass band group delay weight
fdpl=0.09 % dAsqdw pass band lower edge
fdpu=0.21 % dAsqdw pass band upper edge
dpr=0.6 % dAsqdw pass band peak-to-peak ripple
Wdp=0.001 % dAsqdw pass band weight
```

The initial filter is that found by the Octave script *tarczynski_bandpass_hilbert_test.m*. Figure 10.89 shows the response of the band-pass Hilbert filter after SOCP PCLS optimisation and Figure 10.90 shows the pole-zero plot of the band-pass Hilbert filter.

The SOCP PCLS optimised Schur one-multiplier allpass lattice and numerator tap coefficients of the band-pass Hilbert filter are:

```
k2 = [  -0.9484309977,   0.9684730243,  -0.9446922552,   0.5096309461, ...
        -0.6730330483,   0.7238103425,  -0.8104619340,   0.6537271493, ...
        -0.7373073318,   0.6406345117,  -0.6686255005,   0.5524776953, ...
        -0.4306240446,   0.2059231034,  -0.0580646312,   0.0041170564 ]';


epsilon2 = [  1,  1,  1,  1, ...
              1,  1, -1, -1, ...
              1,  1,  1,  1, ...
              1, -1,  1, -1 ]';


p2 = [   0.1970901127,   1.2114709046,   0.1533166816,   0.9091213937, ...
         0.5181410613,   1.1720559424,   0.4691452309,   0.1517961754, ...
         0.3317290739,   0.8530959628,   0.3992639959,   0.8959423068, ...
         0.4810328137,   0.7624965821,   0.9396506722,   0.9958913839 ]';


c2 = [  -0.0561964305,  -0.1335282282,  -0.1728811460,  -0.0025596527, ...
         0.6340018837,   0.2218376578,  -0.3936959290,  -2.1300731266, ...
         0.0554895353,   0.0792517305,   0.0569091299,  -0.0077065507, ...
         0.0651304524,   0.0255800220,   0.0020475437,  -0.0067423554, ...
         0.0019107579 ]';
```

The corresponding overall transfer function numerator and denominator polynomial coefficients are:

Figure 10.89: Response of the Schur one-multiplier lattice band-pass Hilbert filter after SOCP PCLS optimisation.

Figure 10.90: Pole-zero plot of the Schur one-multiplier lattice band-pass Hilbert filter after SOCP PCLS optimisation.

```
N2 = [   0.0019107579,  -0.0206646039,   0.1024786211,  -0.3042007162, ...
         0.6318972926,  -1.0242395598,   1.4217082335,  -1.8074667533, ...
         2.1457831816,  -2.3897013431,   2.4880509728,  -2.3142547528, ...
         1.8315781052,  -1.2122251628,   0.6361690167,  -0.2238216248, ...
         0.0370009263 ];

D2 = [   1.0000000000,  -7.3007717706,  26.9724254824, -67.0750481741, ...
       125.3896145845, -185.9472234065, 225.1216511699, -225.9362827170, ...
       189.1743899451, -132.0276394495,  76.1992817019, -35.7879847330, ...
        13.3105251350,  -3.7488857895,   0.7401679048,  -0.0881213358, ...
         0.0041170564 ];
```

Design of an R=2 IIR Schur one-multiplier lattice band-pass Hilbert filter using SOCP

The Octave script *schurOneMlattice_socp_slb_bandpass_hilbert_R2_test.m* implements the design of an $R = 2$ IIR Schur one-multiplier lattice band-pass Hilbert filter with SOCP and PCLS. The specification of the filter is:

```
maxiter=2000 % Maximum iterations
dmax=0.1 % SQP step-size constraint
ftol=0.0001 % Tolerance on coef. update
ctol=1e-06 % Tolerance on constraints
n=1000 % Frequency points across the band
fasl=0.05 % Amplitude stop band lower edge
fapl=0.1 % Amplitude pass band lower edge
fapl=0.2 % Amplitude pass band upper edge
fasu=0.25 % Amplitude stop band upper edge
dBap=0.16 % Amplitude pass band peak-to-peak ripple(dB)
Wap=1 % Amplitude pass band weight
Watl=0.001 % Amplitude lower transition band weight
Watu=0.001 % Amplitude upper transition band weight
dBas=34 % Amplitude stop band peak-to-peak ripple(dB)
Wasl=10 % Amplitude lower stop band weight
Wasu=10 % Amplitude upper stop band weight
fppl=0.1 % Phase pass band lower edge
fppu=0.2 % Phase pass band upper edge
pp=3.5 % Nominal pass band phase(rad./pi)
ppr=0.002 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=1 % Phase pass band weight
ftpl=0.1 % Delay pass band lower edge
ftpu=0.2 % Delay pass band upper edge
tp=16 % Pass band group delay
tpr=0.2 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
fdpl=0.1 % dAsqdw pass band lower edge
fdpu=0.2 % dAsqdw pass band upper edge
dpr=0.8 % dAsqdw pass band peak-to-peak ripple
Wdp=0.001 % dAsqdw pass band weight
```

The initial filter is that found by the Octave script *tarczynski_bandpass_hilbert_R2_test.m*. Figure 10.91 shows the response of the $R = 2$ band-pass Hilbert filter after SOCP PCLS optimisation and Figure 10.92 shows the pole-zero plot of the $R = 2$ band-pass Hilbert filter.

The SOCP PCLS optimised Schur one-multiplier allpass lattice and numerator tap coefficients of the $R = 2$ band-pass Hilbert filter are:

```
k2 = [   0.0000000000,   0.7001011163,   0.0000000000,   0.4261124087, ...
         0.0000000000,   0.2886396233,   0.0000000000,   0.5012523996, ...
         0.0000000000,   0.3358172035,   0.0000000000,   0.4272308707, ...
         0.0000000000,   0.2615024930,   0.0000000000,   0.2478782477, ...
         0.0000000000,   0.0973834149,   0.0000000000,   0.0606765500 ]';

epsilon2 = [  0,  1,  0, -1, ...
              0, -1,  0,  1, ...
              0, -1,  0,  1, ...
              0, -1,  0, -1, ...
              0, -1,  0, -1 ]';
```

Figure 10.91: Response of the R=2 Schur one-multiplier lattice band-pass Hilbert filter after SOCP PCLS optimisation.

Figure 10.92: Pole-zero plot of the R=2 Schur one-multiplier lattice band-pass Hilbert filter after SOCP PCLS optimisation.

```
p2 = [   1.0986430837,   1.0986430837,   0.4614309005,   0.4614309005, ...
         0.7273947071,   0.7273947071,   0.9790188225,   0.9790188225, ...
         0.5642929102,   0.5642929102,   0.8002658763,   0.8002658763, ...
         0.5069638411,   0.5069638411,   0.6625925133,   0.6625925133, ...
         0.8534705837,   0.8534705837,   0.9410573663,   0.9410573663 ]';


c2 = [  -0.0420598224,   0.0981910014,   0.4723663448,   0.2649154812, ...
        -0.1745141683,  -0.4392432771,  -0.2488290179,  -0.0237240948, ...
         0.1979372060,   0.1601911857,   0.0323032308,   0.0040201827, ...
         0.0668033783,   0.0513460808,   0.0013306865,  -0.0357162951, ...
        -0.0119214202,  -0.0032260123,  -0.0052285174,  -0.0139349362, ...
        -0.0067181691 ]';
```

The corresponding overall transfer function numerator and denominator polynomial coefficients are:

```
N2 = [  -0.0067181691,  -0.0130894124,  -0.0168997653,  -0.0260157344, ...
        -0.0343062014,  -0.0573704696,  -0.0459040705,  -0.0543940340, ...
        -0.0205469861,  -0.0459527328,   0.0196744779,   0.0343556596, ...
         0.1294488948,   0.0495649489,  -0.0037316870,  -0.1582756345, ...
        -0.1222130480,  -0.0881412313,   0.0425873383,   0.0417710745, ...
         0.0491839828 ];


D2 = [   1.0000000000,   0.0000000000,   1.7844886109,   0.0000000000, ...
         2.3017513266,   0.0000000000,   2.6554516800,   0.0000000000, ...
         2.7279729293,   0.0000000000,   2.3030763886,   0.0000000000, ...
         1.7364385842,   0.0000000000,   1.0533753286,   0.0000000000, ...
         0.5568523627,   0.0000000000,   0.2053014963,   0.0000000000, ...
         0.0606765500 ];
```

Design of a low-pass differentiator filter with an IIR Schur one-multiplier lattice correction filter using SOCP

The Octave script *schurOneMlattice_socp_slb_lowpass_differentiator_test.m* calls *schurOneMlattice_slb* to improve the filter designed with *tarczynski_lowpass_differentiator_test.m*. The low-pass differentiator filter consists of $1 - z^{-1}$ and a Schur one-multiplier lattice correction filter. The filter specification is similar to that of Section 8.3.3:

```
maxiter=20000 % Maximum iterations
dmax=0.1 % SQP step-size constraint
ftol=1e-05 % Tolerance on coef. update
ctol=1e-07 % Tolerance on constraints
n=1000 % Frequency points across the band
nN=11 % Correction filter order
fap=0.3 % Amplitude pass band upper edge
Arp=0.004 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.02 % Amplitude transition band peak-to-peak ripple
Wat=0.0001 % Amplitude transition band weight
Ars=0.02 % Amplitude stop band peak-to-peak ripple
Was=0.1 % Amplitude stop band weight
ftp=0.3 % Delay pass band upper edge
tp=10 % Pass band group delay
tpr=0.04 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
fpp=0.3 % Phase pass band upper edge
pp=1.5 % Nominal pass band phase(rad./pi)
ppr=0.0008 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=1 % Phase pass band weight
fdp=0.3 % dAsqdw pass band upper edge
cpr=0.04 % Correction filter dCsqdw pass band peak-to-peak ripple
Wdp=0.1 % Correction filter dCsqdw pass band weight
```

The *schurOneMlattice_socp_mmse* function accepts a constraint on the gradient of the filter squared-amplitude response, $\frac{dA^2}{d\omega}$. For the correction filter, the required squared-amplitude response is $C^2 = \frac{A^2}{A_z^2}$, where

$$\begin{aligned} A &= \frac{\omega}{2} \\ A^2 &= \frac{\omega^2}{4} \\ \frac{dA^2}{d\omega} &= \frac{\omega}{2} = A \end{aligned}$$

and the response of the zero at $z = 1$ is:

$$\begin{aligned} H_z &= 1 - e^{\imath\omega} \\ &= 2\imath e^{\frac{\imath\omega}{2}} \sin\frac{\omega}{2} \\ A_z &= 2\sin\frac{\omega}{2} \\ \frac{dA_z^2}{d\omega} &= 2A_z\frac{dA_z}{d\omega} \\ &= 2\sin\omega \end{aligned}$$

By the chain rule for differentiation, the gradient of A^2 is:

$$\frac{dA^2}{d\omega} = A_z^2\frac{dC^2}{d\omega} + \frac{A^2}{A_z^2}\frac{dA_z^2}{d\omega}$$

Substituting and rearranging to obtain the desired gradient of C^2 in terms of the desired overall squared amplitude response, A^2 and the squared amplitude response of the zero at $z = 1$, A_z^2:

$$\frac{dC^2}{d\omega} = A\left[\frac{1 - A\cot\frac{\omega}{2}}{A_z^2}\right]$$

The PCLS SOCP optimised Schur one-multiplier lattice filter coefficients of the correction filter are:

```
k2 = [   0.3245174840,   0.5781526596,  -0.3974727312,   0.3520518111, ...
        -0.2991718343,   0.2343584869,  -0.1633806872,   0.0985543464, ...
        -0.0489077827,   0.0186202839,  -0.0061187713 ]';


c2 = [   0.1613046180,  -0.2321622076,  -0.6602583304,   0.0329292928, ...
         0.0689297307,  -0.0476499369,   0.0160058942,   0.0023352353, ...
        -0.0095424050,   0.0070946652,  -0.0026431334,  -0.0007082368 ]';
```

The corresponding transfer function numerator and denominator polynomials of the correction filter are:

```
N2 = [  -0.0007082368,  -0.0025609060,   0.0066336221,  -0.0109725605, ...
         0.0092626279,   0.0028846517,  -0.0233805409,   0.0344281761, ...
         0.0065864853,  -0.2382746702,  -0.3143327597,  -0.1236520776 ];


D2 = [   1.0000000000,  -0.0932664376,   0.7498114721,  -0.6874499446, ...
         0.6232044218,  -0.5018712149,   0.3574571547,  -0.2230854059, ...
         0.1209522104,  -0.0552113812,   0.0191902628,  -0.0061187713 ];
```

Figure 10.93 shows the amplitude error, phase and group delay responses of the low-pass differentiator filter after PCLS optimisation and Figure 10.94 shows the pole-zero plot of the low-pass differentiator filter after PCLS optimisation.

Figure 10.93: PCLS optimised amplitude error, phase and group delay responses of a low-pass differentiator filter composed of a Schur one-multiplier lattice correction filter and $1 - z^{-1}$.

Figure 10.94: Pole-zero plot of the PCLS optimised low-pass differentiator filter composed of a Schur one-multiplier lattice correction filter in series with $1 - z^{-1}$.

Design of an alternative low-pass differentiator filter with an IIR Schur one-multiplier lattice correction filter using SOCP

The Octave script *schurOneMlattice_socp_slb_lowpass_differentiator_alternate_test.m* calls *schurOneMlattice_slb* to improve the filter designed with *tarczynski_lowpass_differentiator_alternate_test.m*. The low-pass differentiator filter consists of $1 - z^{-1}$ and a Schur one-multiplier lattice correction filter. The filter specification is similar to that of Section 8.3.3:

```
maxiter=20000 % Maximum iterations
dmax=0.1 % SQP step-size constraint
ftol=1e-05 % Tolerance on coef. update
ctol=1e-07 % Tolerance on constraints
n=1000 % Frequency points across the band
nN=11 % Correction filter order
fap=0.18 % Amplitude pass band upper edge
Arp=0.005 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.005 % Amplitude transition band peak-to-peak ripple
Wat=0.0001 % Amplitude transition band weight
Ars=0.005 % Amplitude stop band peak-to-peak ripple
Was=1 % Amplitude stop band weight
tp=10 % Pass band group delay
tpr=0.01 % Pass band group delay peak-to-peak ripple
Wtp=1 % Pass band group delay weight
pp=1.5 % Nominal pass band phase(rad./pi)
ppr=0.0002 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=1 % Phase pass band weight
fdp=0.18 % dAsqdw pass band upper edge
cpr=0.1 % Correction filter dCsqdw pass band peak-to-peak ripple
Wdp=0.1 % Correction filter dCsqdw pass band weight
```

The PCLS SOCP optimised alternative Schur one-multiplier lattice filter coefficients of the correction filter are:

```
k2 = [  -0.5471624990,   0.6820335176,  -0.4175565712,  -0.1781235862, ...
         0.5405933547,  -0.5077775684,   0.2096515704,   0.1257004580, ...
        -0.2046664244,   0.1091919652,  -0.0244564293 ]';
```

```
c2 = [   0.0668018222,  -0.1296378723,  -0.4899974900,  -0.1417542738, ...
         0.0190937537,   0.0710348472,  -0.0069908265,  -0.0120241991, ...
         0.0018993889,   0.0087070260,   0.0004217389,  -0.0031520246 ]';
```

The corresponding transfer function numerator and denominator polynomials of the correction filter are:

```
N2 = [  -0.0031520246,   0.0055567843,   0.0025937496,  -0.0100965890, ...
        -0.0021695991,   0.0138082343,   0.0140244884,  -0.0222018037, ...
        -0.0612878117,  -0.0598504235,  -0.0305728373,  -0.0067277728 ];
```

```
D2 = [   1.0000000000,  -1.6323982763,   1.3649441992,   0.1524017167, ...
        -1.6467688846,   1.8607598726,  -0.8757139552,  -0.1984611572, ...
         0.5879978426,  -0.4133336643,   0.1490492888,  -0.0244564293 ];
```

Figure 10.95 shows the amplitude error, phase and group delay responses of the alternative low-pass differentiator filter after PCLS optimisation and Figure 10.94 shows the pole-zero plot of the alternative low-pass differentiator filter after PCLS optimisation.

Figure 10.95: PCLS optimised amplitude error, phase and group delay responses of a low-pass differentiator filter composed of an alternative Schur one-multiplier lattice correction filter in series with $1 - z^{-1}$.

Figure 10.96: Pole-zero plot of the PCLS optimised low-pass differentiator filter composed of an alternative Schur one-multiplier lattice correction filter in series with $1 - z^{-1}$.

Design of a low-pass differentiator filter with a pipelined IIR Schur one-multiplier lattice correction filter using SOCP

The Octave script *schurOneMlatticePipelined_socp_slb_lowpass_differentiator_test.m* calls *schurOneMlatticePipelined_slb* to improve the filter designed with *tarczynski_lowpass_differentiator_test.m*. The low-pass differentiator filter consists of $1 - z^{-1}$ and a Schur one-multiplier lattice correction filter implemented in the pipelined form shown in Section 5.6.3. The filter specification is similar to that of Section 8.3.3:

```
maxiter=20000 % Maximum iterations
dmax=0 % SQP step-size constraint
ftol=0.001 % Tolerance on coef. update
ctol=5e-08 % Tolerance on constraints
n=400 % Frequency points across the band
nN=11 % Correction filter order
fap=0.3 % Amplitude pass band upper edge
Arp=0.004 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.008 % Amplitude transition band peak-to-peak ripple
Wat=0.0001 % Amplitude transition band weight
Ars=0.008 % Amplitude stop band peak ripple
Was=1 % Amplitude stop band weight
tp=10 % Pass band group delay
tpr=0.04 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
pp=1.5 % Nominal pass band phase(rad./pi)
ppr=0.0008 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=1 % Phase pass band weight
```

The PCLS SOCP optimised pipelined Schur one-multiplier lattice filter coefficients of the correction filter are:

```
k2 = [   0.1646587548,   0.6704988214,  -0.6038317162,   0.3307199588, ...
        -0.4527972871,   0.4652019780,  -0.2654650462,   0.2396211209, ...
        -0.0451891423,  -0.1350142233,  -0.0057905277 ]';

c2 = [   0.1817087418,  -0.2409469442,  -1.1756723893,  -0.0510043956, ...
         0.0442543901,  -0.0826630456,   0.0081490138,  -0.0095617331, ...
        -0.0137427972,   0.0292087017,  -0.0020733031,  -0.0011253488 ]';

kk2 = [   0.4219838632,  -0.3615100542,  -0.0779436129,  -0.0748109228, ...
          0.0760473973,   0.0865686357,   0.1667858154,   0.0168524326, ...
          0.1455152105,   0.0012684854 ]';

ck2 = [   0.0914302957,   0.0000000000,  -0.0579589347,   0.0000000000, ...
         -0.0620298178,   0.0000000000,  -0.0178332871,   0.0000000000, ...
          0.0230776399,   0.0000000000 ]';
```

The corresponding transfer function numerator and denominator polynomials of the correction filter are:

```
N2 = [  -0.0011253488,  -0.0026975878,   0.0039308506,  -0.0067361265, ...
         0.0030308984,   0.0070330763,  -0.0195985511,   0.0196149304, ...
         0.0249939303,  -0.2279720838,  -0.4695423787,  -0.3769157862, ...
        -0.1568253039,  -0.0509390569,  -0.0161348008,  -0.0024488165, ...
        -0.0001352073 ];

D2 = [   1.0000000000,   0.5654160050,   0.8902423422,  -0.1632122768, ...
         0.3604776873,  -0.2112486427,   0.1572692662,  -0.0967215813, ...
         0.0543103553,  -0.0280626729,   0.0112171921,  -0.0054277222 ];
```

Figure 10.97 shows the amplitude error, phase and group delay responses of the low-pass differentiator filter after PCLS optimisation and Figure 10.98 shows the pole-zero plot of the low-pass differentiator filter after PCLS optimisation.

Figure 10.97: PCLS optimised amplitude error, phase and group delay responses of a low-pass differentiator filter composed of a pipelined Schur one-multiplier lattice correction filter and $1 - z^{-1}$.

Figure 10.98: Pole-zero plot of the PCLS optimised low-pass differentiator filter composed of a pipelined Schur one-multiplier lattice correction filter in series with $1 - z^{-1}$.

Design of a low-pass differentiator filter with an R=2 IIR Schur one-multiplier lattice correction filter using SOCP

The Octave script *schurOneMlattice_socp_slb_lowpass_differentiator_R2_test.m* designs a low-pass differentiator filter consisting of a Schur one-multiplier lattice correction filter in series with $1 - z^{-1}$ with constraints on the squared-amplitude, phase, group delay and on the derivative of the squared-amplitude with respect to angular frequency. The denominator of the correction filter has terms only in z^{-2} (ie: $R = 2$). The filter specification is:

```
maxiter=20000 % Maximum iterations
dmax=0 % SQP step-size constraint
ftol=1e-06 % Tolerance on coef. update
ctol=1e-08 % Tolerance on constraints
n=1000 % Frequency points across the band
nN=10 % Correction filter order
fap=0.2 % Amplitude pass band upper edge
Arp=0.0009 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.004 % Amplitude transition band peak-to-peak ripple
Wat=0.0001 % Amplitude transition band weight
Ars=0.007 % Amplitude stop band peak-to-peak ripple
Was=0.1 % Amplitude stop band weight(PCLS)
fpp=0.2 % Phase pass band upper edge
pp=1.5 % Nominal pass band phase(rad./pi)
ppr=0.0002 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=1 % Phase pass band weight
ftp=0.2 % Delay pass band upper edge
tp=9 % Pass band group delay
tpr=0.006 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
fdp=0.2 % dAsqdw pass band upper edge
cpr=0.013 % Correction filter pass band dCsqdw peak-to-peak ripple
Wdp=0.1 % Correction filter pass band dCsqdw weight
```

The Schur one-multiplier lattice coefficients of the PCLS SOCP optimised correction filter are:

```
k2 = [   0.0000000000,   0.2065205470,   0.0000000000,  -0.0281020445, ...
         0.0000000000,   0.0093023973,   0.0000000000,  -0.0038787444, ...
         0.0000000000,   0.0012820857 ]';


epsilon2 = [  -0.0000000000,   1.0000000000,  -0.0000000000,   1.0000000000, ...
              -0.0000000000,  -1.0000000000,  -0.0000000000,   1.0000000000, ...
              -0.0000000000,  -1.0000000000 ]';


c2 = [  -0.0212083759,  -0.2188914462,  -0.2870345994,  -0.0306418043, ...
         0.0731356951,  -0.0170264555,  -0.0231060228,   0.0170587049, ...
         0.0012358212,  -0.0062679753,   0.0022181847 ]';
```

The corresponding correction filter numerator and denominator polynomial coefficients are:

```
N2 = [   0.0022181847,  -0.0062599392,   0.0016787930,   0.0157161394, ...
        -0.0227974703,  -0.0132156271,   0.0674605209,  -0.0332183997, ...
        -0.2598747453,  -0.2583845612,  -0.0835347840 ];


D2 = [   1.0000000000,   0.0000000000,   0.2004144265,   0.0000000000, ...
        -0.0261222080,   0.0000000000,   0.0084913768,   0.0000000000, ...
        -0.0036217895,   0.0000000000,   0.0012820857 ];
```

Figure 10.99 shows the amplitude error, phase and group delay response errors of the $R = 2$ low-pass differentiator filter after PCLS SOCP optimisation (with sample rate $f_S = 1$). Figure 10.100 shows the error in the gradient of the squared-amplitude response of the $R = 2$ correction filter of the low-pass differentiator filter after PCLS SOCP optimisation. Figure 10.101 shows the error in the gradient of the squared-amplitude response of the $R = 2$ low-pass differentiator filter after PCLS SOCP optimisation. Figure 10.102 shows the pole-zero plot of the $R = 2$ low-pass differentiator filter.

Figure 10.99: PCLS optimised amplitude error, phase and group delay responses of a low-pass differentiator filter composed of a Schur one-multiplier lattice correction filter with a denominator polynomial having terms only in z^{-2} in series with $1 - z^{-1}$.

Figure 10.100: PCLS optimised error of the gradient of the squared amplitude response of the correction filter.

Figure 10.101: PCLS optimised error of the gradient of the squared amplitude response of the low-pass differentiator filter.

Figure 10.102: Pole-zero plot of the PCLS optimised low-pass differentiator filter composed of a Schur one-multiplier lattice correction filter with a denominator polynomial having terms only in z^{-2} in series with $1 - z^{-1}$.

Design of a band-pass differentiator filter with an IIR Schur one-multiplier lattice correction filter using SOCP

The Octave script *schurOneMlattice_socp_slb_bandpass_differentiator_test.m* calls *schurOneMlattice_slb* to improve the filter designed with *tarczynski_bandpass_differentiator_test.m*. The band-pass differentiator filter consists of $\left(1-z^{-1}\right)\left(1+z^{-1}\right)$ in series with a Schur one-multiplier lattice correction filter. The filter specification is:

```
maxiter=20000 % Maximum iterations
dmax=0.1 % SQP step-size constraint
ftol=1e-05 % Tolerance on coef. update
ctol=1e-07 % Tolerance on constraints
n=1000 % Frequency points across the band
nN=11 % Correction filter order
fasl=0.05 % Amplitude stop band lower edge
fapl=0.1 % Amplitude pass band lower edge
fapl=0.2 % Amplitude pass band upper edge
fasu=0.25 % Amplitude stop band upper edge
Arp=0.02 % Amplitude pass band peak-to-peak ripple
Wap=0.1 % Amplitude pass band weight
Watl=0.001 % Amplitude lower transition band weight
Watu=0.001 % Amplitude upper transition band weight
Ars=0.02 % Amplitude stop band peak-to-peak ripple
Wasl=0.1 % Amplitude lower stop band weight
Wasu=0.1 % Amplitude upper stop band weight
fppl=0.1 % Phase pass band lower edge
fppu=0.2 % Phase pass band upper edge
pp=3.5 % Nominal pass band phase(rad./pi)
ppr=0.002 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=0.5 % Phase pass band weight
ftpl=0.1 % Delay pass band lower edge
ftpu=0.2 % Delay pass band upper edge
tp=12 % Pass band group delay
tpr=0.2 % Pass band group delay peak-to-peak ripple
Wtp=0.02 % Pass band group delay weight
fdpl=0.1 % dAsqdw pass band lower edge
fdpu=0.2 % dAsqdw pass band upper edge
dpr=0.62 % dAsqdw pass band peak-to-peak ripple
Wdp=0.01 % dAsqdw pass band weight
```

The PCLS SOCP optimised Schur one-multiplier lattice filter coefficients of the correction filter are:

```
k2 = [  -0.6782593333,   0.7716171328,  -0.5974716635,   0.8321049234, ...
        -0.4567902568,   0.2349663203,   0.5501940897,  -0.5371231614, ...
         0.5845106772,  -0.2985411988,   0.1432214527 ]';
```

```
c2 = [   0.0232461021,  -0.3433853242,  -0.0655639943,   0.0272704323, ...
         0.0835782758,   0.0069440654,  -0.0197108995,   0.0121269408, ...
        -0.0018494197,  -0.0060186836,  -0.0149804305,  -0.0006529892 ]';
```

The corresponding transfer function numerator and denominator polynomials of the correction filter are:

```
N1 = [  -0.0006529892,  -0.0106508721,   0.0347673563,  -0.0617586699, ...
         0.0688733914,  -0.0577406211,   0.0366999427,   0.0020951913, ...
        -0.0256721220,   0.0224383259,  -0.0259852204,   0.0143706353 ];
```

```
D1 = [   1.0000000000,  -3.3446792409,   6.1185071192,  -6.7837798866, ...
         4.0053495596,   1.0878774642,  -5.1733085434,   6.2556181653, ...
        -4.6435804244,   2.3633348507,  -0.7714472267,   0.1432214527 ];
```

Figure 10.103 shows the amplitude error, phase and group delay responses of the band-pass differentiator filter after PCLS optimisation and Figure 10.104 shows the pass-band *dAsqdw* response of the band-pass differentiator filter after PCLS optimisation. Figure 10.105 shows the pole-zero plot of the band-pass differentiator filter after PCLS optimisation.

Figure 10.103: PCLS optimised amplitude error, phase and group delay responses of a band-pass differentiator filter composed of a Schur one-multiplier lattice correction filter and $1 - z^{-2}$.

Figure 10.104: PCLS optimised pass-band dAsqdw response of a band-pass differentiator filter composed of a Schur one-multiplier lattice correction filter and $1 - z^{-2}$.

Figure 10.105: Pole-zero plot of the PCLS optimised band-pass differentiator filter composed of a Schur one-multiplier lattice correction filter in series with $1 - z^{-2}$.

10.3.3 Design of parallel one-multiplier IIR Schur all-pass lattice filters with SOCP optimisation

Design of a low-pass filter with parallel IIR 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. For such a filter the all-pass reflection coefficient sign assignments can be recalculated without affecting the response. 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 filter 1 denominator order
m2=12 % Allpass 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=10 % 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.

The low-pass filter SOCP PCLS optimised Schur one-multiplier all-pass lattice coefficients are:

```
A1k = [   0.7710295204,  -0.0878871127,  -0.2677056631,  -0.0635149923, ...
         -0.0592195263,   0.2448549901,  -0.1439983042,  -0.0043115430, ...
          0.1649856955,  -0.1593160701,   0.0532524579 ];

A1epsilon = [  1,  1,  1,  1, ...
               1, -1,  1,  1, ...
               1,  1, -1 ];

A2k = [   0.3876308734,  -0.2737154211,   0.1867714032,   0.1640923406, ...
         -0.0465239641,   0.0416437221,  -0.2014866175,   0.1799905345, ...
          0.0056640858,  -0.1788158072,   0.1502470192,  -0.0542820187 ];

A2epsilon = [  1,  1,  1, -1, ...
               1, -1, -1, -1, ...
              -1, -1, -1,  1 ];
```

The corresponding transfer function numerator and denominator polynomial coefficients are:

```
N1 = [  -0.0005147804,  -0.0049596978,  -0.0105502183,  -0.0031533307, ...
         0.0217728460,   0.0373542989,   0.0033488872,  -0.0707822375, ...
        -0.0931351261,   0.0310831178,   0.2765114952,   0.4805891169, ...
         0.4805891169,   0.2765114952,   0.0310831178,  -0.0931351261, ...
        -0.0707822375,   0.0033488872,   0.0373542989,   0.0217728460, ...
        -0.0031533307,  -0.0105502183,  -0.0049596978,  -0.0005147804 ]';

D1 = [   1.0000000000,   0.8347525334,  -0.4958730504,  -0.2435821932, ...
         0.2011267259,   0.1914683204,   0.0172585043,  -0.4564079820, ...
         0.2641443935,   0.2964690719,  -0.4045713072,   0.0766336794, ...
         0.1089298032,  -0.0523985809,   0.0326015068,  -0.0941713935, ...
         0.0643295491,   0.0427130571,  -0.0904795729,   0.0484935338, ...
         0.0087478143,  -0.0263529629,   0.0141879433,  -0.0028906509 ]';
```

Figures 10.106 and 10.107 show the overall and passband responses of the filter after SOCP PCLS optimisation and Figure 10.108 shows the pole-zero plot of the filter. Figure 10.109 shows the coefficient sensitivity responses of the filter.

Figure 10.106: Parallel Schur one-multiplier all-pass lattice low-pass filter, response after SOCP PCLS optimisation.

Figure 10.107: Parallel Schur one-multiplier all-pass lattice low-pass filter, passband response after SOCP PCLS optimisation.

Figure 10.108: Parallel Schur one-multiplier all-pass lattice low-pass filter, pole-zero plot after SOCP PCLS optimisation.

Figure 10.109: Parallel Schur one-multiplier all-pass lattice low-pass filter, coefficient sensitivity responses after SOCP PCLS optimisation.

Design of a low-pass filter with a delay in parallel with a one-multiplier IIR Schur all-pass lattice filter using SOCP

The Octave script *schurOneMPAlatticeDelay_socp_slb_lowpass_test.m* implements the design of a low-pass IIR filter composed of the sum of a delay and a one-multiplier Schur lattice all-pass filter with SOCP and PCLS optimisation. For such a filter the all-pass reflection coefficient sign assignments can be recalculated without affecting the response. The filter specification is similar to that described in Section 10.2.4:

```
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 order
DD=11 % Delay order
rho=0.992188 % Constraint on allpass coefficients
fap=0.15 % Amplitude pass band edge
dBap=0.2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=0 % Transition pass band weight
fas=0.2 % Amplitude stop band edge
dBas=66 % amplitude stop band peak-to-peak ripple
Was=1000 % Amplitude stop band weight
```

The initial all-pass filter was designed with the Octave function *WISE_DA.m* using the *WISE* method described in Section 8.1.5 and is the same as that used in Section 10.2.4. The low-pass filter SOCP PCLS optimised Schur one-multiplier all-pass lattice coefficients are:

```
A1k = [  -0.4530542361,   0.5781728907,   0.2628131000,   0.0076641040, ...
         -0.1283105496,  -0.1389062973,  -0.0795356965,  -0.0125620789, ...
          0.0241394981,   0.0278898111,   0.0160165189,   0.0049058482 ];
```

The corresponding all-pass denominator and overall transfer function denominator polynomial coefficients are:

```
Da1 = [   1.0000000000,  -0.5312501000,   0.3519954023,   0.1733797903, ...
          0.0095693876,  -0.0839024378,  -0.0971865434,  -0.0596378612, ...
         -0.0127507348,   0.0157736583,   0.0210989906,   0.0134099011, ...
          0.0049058482 ];
```

The overall transfer function numerator polynomial coefficients are:

```
Na1 = [   0.0024529241,   0.0067049505,   0.0105494953,   0.0078868291, ...
         -0.0063753674,  -0.0298189306,  -0.0485932717,  -0.0419512189, ...
          0.0047846938,   0.0866898952,   0.1759977012,   0.2343749500, ...
          0.2343749500,   0.1759977012,   0.0866898952,   0.0047846938, ...
         -0.0419512189,  -0.0485932717,  -0.0298189306,  -0.0063753674, ...
          0.0078868291,   0.0105494953,   0.0067049505,   0.0024529241 ];
```

Figure 10.110 shows the passband and stopband responses of the filter after SOCP PCLS optimisation and Figure 10.111 shows the pole-zero plot of the filter.

Figure 10.110: Parallel delay and Schur one-multiplier all-pass lattice low-pass filter, response after SOCP PCLS optimisation.

Figure 10.111: Parallel delay and Schur one-multiplier all-pass lattice low-pass filter, pole-zero plot after SOCP PCLS optimisation.

Figure 10.112: Parallel delay and Schur one-multiplier all-pass lattice low-pass filter with flat pass-band delay, amplitude and delay responses after SOCP PCLS optimisation.

Similarly, the Octave script *schurOneMPAlatticeDelay_socp_slb_lowpass_flat_delay_test.m* implements the design of a low-pass IIR filter with flat pass-band delay composed of the sum of a delay and a one-multiplier Schur lattice all-pass filter with SOCP and PCLS optimisation. 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 order
DD=11 % Delay order
rho=0.992188 % Constraint on allpass coefficients
fap=0.1 % Amplitude pass band edge
dBap=1e-05 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=0 % Transition pass band weight
fas=0.2 % Amplitude stop band edge
dBas=50 % Amplitude stop band peak-to-peak ripple
Was=1000 % Amplitude stop band weight
ftp=0.1 % Amplitude stop band edge
tp=11 % Nominal pass band delay
tpr=0.04 % Delay pass band peak-to-peak ripple
Wtp=10 % Delay pass band weight
```

Figure 10.112 shows the passband and stopband amplitude and passband delay responses of the filter after SOCP PCLS optimisation and Figure 10.113 shows the pole-zero plot of the filter.

The low-pass filter SOCP PCLS optimised Schur one-multiplier all-pass lattice filter coefficients are:

```
A1k = [  -0.4390056051,   0.5189533399,   0.2330782093,   0.0196753654, ...
         -0.0733534421,  -0.0607822860,  -0.0114720768,   0.0212902857, ...
          0.0197818397,   0.0054837546,  -0.0064443752,  -0.0060322673 ];
```

The overall transfer function and all-pass filter denominator polynomial coefficients are:

Figure 10.113: Parallel delay and Schur one-multiplier all-pass lattice low-pass filter with flat pass-band delay, pole-zero plot after SOCP PCLS optimisation.

```
Da1 = [   1.0000000000,  -0.5372848520,   0.3495071434,   0.1807175071, ...
          0.0354872552,  -0.0396210835,  -0.0428176885,  -0.0149887923, ...
          0.0111882539,   0.0134908181,   0.0068375965,  -0.0032030949, ...
         -0.0060322673 ];
```

The overall transfer function numerator polynomial coefficients are:

```
Na1 = [  -0.0030161336,  -0.0016015475,   0.0034187983,   0.0067454091, ...
          0.0055941270,  -0.0074943962,  -0.0214088442,  -0.0198105417, ...
          0.0177436276,   0.0903587535,   0.1747535717,   0.2313575740, ...
          0.2313575740,   0.1747535717,   0.0903587535,   0.0177436276, ...
         -0.0198105417,  -0.0214088442,  -0.0074943962,   0.0055941270, ...
          0.0067454091,   0.0034187983,  -0.0016015475,  -0.0030161336 ];
```

Figure 10.114: Parallel Schur one-multiplier all-pass lattice band-pass filter, response after SOCP PCLS optimisation.

Design of a parallel one-multiplier IIR Schur 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 10.114 and 10.115 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:

Figure 10.115: Parallel Schur one-multiplier all-pass lattice band-pass filter, passband response after SOCP PCLS optimisation.

```
A1k = [  -0.3887314381,   0.6647113553,   0.5102176699,  -0.5390050446, ...
          0.6391764898,  -0.3370461355,  -0.0288058888,   0.3551179240, ...
         -0.2476137875,   0.1535667938 ];


A1epsilon = [  1,  1, -1, -1, ...
              -1, -1,  1, -1, ...
              -1, -1 ];


A2k = [  -0.7496639351,   0.7373151502,   0.5069121326,  -0.5829397310, ...
          0.6569245139,  -0.2553125712,   0.0136383101,   0.3440396743, ...
         -0.2756364380,   0.1404553791 ];


A2epsilon = [ -1, -1,  1, -1, ...
              -1,  1, -1,  1, ...
               1, -1 ];
```

The corresponding coefficients of the all-pass filter denominator polynomial coefficients are:

```
A1d = [   1.0000000000,  -1.2694173000,   0.9630851728,   0.7785340619, ...
         -1.8332672241,   1.7268712967,  -0.4895218755,  -0.4848448075, ...
          0.7711004756,  -0.4367147158,   0.1535667938 ]';


A2d = [   1.0000000000,  -1.9071506281,   1.2751872028,   0.9984025388, ...
         -2.4997957590,   2.1971251576,  -0.5676377617,  -0.7145302630, ...
          0.9955856416,  -0.5380683258,   0.1404553791 ]';
```

The corresponding coefficients of the overall filter transfer function numerator and denominator polynomial coefficients are:

```
N2 = [   0.0065557074,  -0.0066124546,  -0.0070408729,   0.0140895630, ...
        -0.0116456808,   0.0037787576,   0.0028426196,  -0.0188351006, ...
         0.0254770785,  -0.0007095911,   0.0000000000,   0.0007095911, ...
        -0.0254770785,   0.0188351006,  -0.0028426196,  -0.0037787576, ...
         0.0116456808,  -0.0140895630,   0.0070408729,   0.0066124546, ...
        -0.0065557074 ]';


D2 = [   1.0000000000,  -3.1765679281,   4.6592423767,  -1.6785565875, ...
        -5.8571202759,  12.5479205297, -11.1076167509,   0.9963741201, ...
        10.4449228230, -14.2910311290,   8.7701997343,   0.4648555871, ...
        -6.2978883453,   6.3792383316,  -3.1086999098,   0.0575666541, ...
         1.1846957845,  -1.0275188252,   0.4961764605,  -0.1439683586, ...
         0.0215692822 ]';
```

Design of a parallel one-multiplier IIR Schur 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:

```
tol=0.0001 % Tolerance on coefficient update vector
ctol=4e-06 % Tolerance on constraints
difference=1 % Use difference of all-pass filters
rho=0.999000 % Constraint on all-pass pole radius
n=1000 % Frequency points across the band
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=0.100000 % Pass band amplitude response ripple(dB)
dBas=40.000000 % Stop band amplitude response ripple(dB)
Wasl=100 % Lower stop band amplitude response weight
Watl=0.001 % Lower transition band amplitude response weight
Wap=1 % Pass band amplitude response weight
Watu=0.001 % Upper transition 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
tp=16.000000 % Pass band nominal group-delay response(samples)
tpr=0.020000 % 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
pp=3.500000 % Pass band nominal phase response(rad./pi)
ppr=0.000400 % Pass band phase response ripple(rad./pi)
Wpp=200 % Pass band phase response weight
fdpl=0.1 % Pass band dAsqdw response lower edge
fdpu=0.2 % Pass band dAsqdw response upper edge
dp=0.000000 % Pass band nominal dAsqdw response
dpr=0.400000 % Pass band dAsqdw response ripple
Wdp=200 % Pass band dAsqdw response weight
```

The initial parallel all-pass filters were designed by the Octave script *tarczynski_parallel_allpass_bandpass_hilbert_test.m*. Figure 10.116 shows 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 PCLS SOCP optimised Schur one-multiplier all-pass lattice coefficients are:

```
A1k = [  -0.4547519802,   0.8294538823,  -0.2290886121,   0.0509697202, ...
          0.6797324792,  -0.3289202955,   0.1033177030,   0.5237933034, ...
         -0.3767482866,   0.2666719576 ];


A1epsilon = [  1,  1,  1, -1, ...
              -1, -1, -1,  1, ...
               1, -1 ];


A2k = [  -0.8096068256,   0.8781058530,  -0.2981458648,   0.0182027132, ...
          0.6950393583,  -0.2987878409,   0.1121206651,   0.5251767594, ...
         -0.3625084023,   0.2703654885 ];


A2epsilon = [  1,  1,  1, -1, ...
              -1, -1, -1,  1, ...
               1, -1 ];
```

The corresponding overall transfer function denominator and numerator polynomial coefficients are:

Figure 10.116: Response of a parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter after SOCP PCLS optimisation. The phase response shown is adjusted for the nominal delay.

```
N2 = [   0.0018467655,   0.0000551038,  -0.0097238630,   0.0251947584, ...
        -0.0168215416,  -0.0176605440,   0.0425822411,  -0.0212706685, ...
        -0.0164408713,   0.0153192361,   0.0000000000,  -0.0153192361, ...
         0.0164408713,   0.0212706685,  -0.0425822411,   0.0176605440, ...
         0.0168215416,  -0.0251947584,   0.0097238630,  -0.0000551038, ...
        -0.0018467655 ]';


D2 = [   1.0000000000,  -3.7460310222,   6.8322417508,  -5.3058431985, ...
        -3.6346952605,  15.4046532057, -18.5952239555,   7.1808940874, ...
        11.9139994214, -23.0059183734,  17.6811468951,  -1.7291074777, ...
       -11.1153708748,  13.1270359783,  -6.8172159291,  -0.1982372377, ...
         3.3897148977,  -3.0029867707,   1.5076206608,  -0.4543052120, ...
         0.0720988941 ]';
```

Design of a parallel one-multiplier IIR Schur 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:

```
ftol=0.0001 % Tolerance on coefficient 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:

```
A1k = [  -0.8284814286,   0.9619285274,  -0.5895615255,   0.8587700995, ...
         -0.7055500672,   0.7605379424,  -0.4239830987,   0.7855591572 ];


A1epsilon = [  1,  1,  1, -1, ...
              -1,  1,  1, -1 ];


A2k = [  -0.8236548196,   0.9115734805,  -0.8137477287,   0.8741940439, ...
         -0.7512126727,   0.9917707622,  -0.2544288989,   0.9046419603, ...
         -0.7502018217,   0.8162944796,  -0.4682769792,   0.7041672063 ];


A2epsilon = [ -1, -1, -1, -1, ...
               1,  1,  1, -1, ...
               1,  1,  1,  1 ];
```

The corresponding all-pass lattice denominator polynomial coefficients are:

```
A1d = [   1.0000000000,  -4.4968571460,  10.6566184928, -16.6695396051, ...
         18.8009550606, -15.5311900846,   9.2862214452,  -3.6948890857, ...
          0.7855591572 ];
```

Figure 10.117: Schur one-multiplier parallel all-pass lattice low-pass to multi-band filter, response after SOCP PCLS optimisation. The initial filter is designed by a low-pass to multi-band frequency transformation.

```
A2d = [     1.0000000000,    -6.9149264445,    24.6254315642,   -58.6403760039, ...
          103.2746321466,  -141.0696537496,   152.9193179927,  -132.4336048357, ...
           91.0371712706,   -48.5791213810,    19.2163497401,    -5.1053455940, ...
            0.7041672063 ];
```

The overall transfer function numerator and denominator polynomial coefficients are:

```
N2 = [   0.0406959754,  -0.4275439740,   2.2512200643,  -7.8311432807, ...
        20.0292980336, -39.6699264870,  62.2727656006, -77.4233218347, ...
        73.3788233512, -45.1553771151,   0.0000000000,  45.1553771151, ...
       -73.3788233512,  77.4233218347, -62.2727656006,  39.6699264870, ...
       -20.0292980336,   7.8311432807,  -2.2512200643,   0.4275439740, ...
        -0.0406959754 ];
```

```
D2 = [     1.0000000000,   -11.4117835906,    66.3774864532,  -259.7366965370, ...
         763.4654507177, -1786.4220548323,  3445.0207142243, -5597.8217442489, ...
        7775.1703861558, -9315.5393082376,  9676.4796979797, -8729.3105083499, ...
        6829.0800361519, -4610.7957110410,  2663.2310818665, -1297.6994570369, ...
         521.9879368416,  -167.5099790416,    40.4983178313,    -6.6123707067, ...
           0.5531649970 ];
```

Figure 10.117 shows the amplitude and delay response of the PCLS SOCP optimised parallel all-pass filter. Figure 10.118 shows the pass-band response of the filter.

Figure 10.118: Schur one-multiplier parallel all-pass lattice low-pass to multi-band filter, pass-band response after SOCP PCLS optimisation. The initial filter is designed by a low-pass to multi-band frequency transformation.

Similarly, 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:

```
ftol=0.0001 % Tolerance on coefficient 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=20 % Order of the first all-pass filter
mb=20 % Order of the second all-pass filter
fas1u=0.06 % Amplitude stop band 1 upper edge
fap1l=0.09 % Amplitude pass band 1 lower edge
fap1u=0.1 % Amplitude pass band 1 upper edge
fas2l=0.13 % Amplitude stop band 2 lower edge
fas2u=0.16 % Amplitude stop band 2 upper edge
fap2l=0.19 % Amplitude pass band 2 lower edge
fap2u=0.22 % 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=30 % 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=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=0.09 % Delay pass band 1 lower edge
ftp1u=0.1 % Delay pass band 1 upper edge
ftp2l=0.19 % Delay pass band 2 lower edge
ftp2u=0.22 % 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=0.1 % Delay pass band 1 peak-to-peak ripple
tpr2=0.1 % 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:

```
A1k = [   0.8414714253,   0.4061229567,  -0.0361621392,   0.2879275053, ...
         -0.3365366737,  -0.1368834034,   0.4527641501,   0.4549372198, ...
         -0.0127499924,  -0.1564100626,  -0.0908844613,   0.0069962272, ...
          0.0435941336,  -0.0148848316,   0.0730116512,   0.0147821388, ...
         -0.1711078476,   0.0353361546,  -0.0489257318,  -0.0294527709 ];

A1epsilon = [  1, -1,  1, -1, ...
               1, -1,  1, -1, ...
               1,  1, -1, -1, ...
              -1,  1, -1, -1, ...
              -1, -1,  1,  1 ];

A2k = [   0.8243216801,   0.1406684024,  -0.3021626968,   0.5081936633, ...
          0.2197032157,   0.0239193127,   0.2677219107,   0.2679750817, ...
         -0.1090420890,  -0.1669341575,  -0.1131400213,  -0.0185637078, ...
          0.0566852527,   0.0498976945,   0.1462569676,   0.0665930847, ...
         -0.1639157341,   0.0142955290,  -0.1129035688,  -0.0663234423 ];

A2epsilon = [  1, -1,  1, -1, ...
               1, -1, -1,  1, ...
               1,  1, -1,  1, ...
              -1, -1,  1, -1, ...
              -1, -1,  1,  1 ];
```

The corresponding all-pass lattice denominator polynomial coefficients are:

```
A1d = [   1.0000000000,   1.2518437255,   0.2476039704,  -0.2779476259, ...
         -0.2965656722,  -0.2834291305,   0.5104578672,   1.0235632389, ...
          0.4876973335,  -0.2509661922,  -0.4116434523,  -0.1566751518, ...
          0.0331804379,   0.0252305458,   0.1164130779,   0.0591848822, ...
         -0.1690331208,  -0.1301159603,  -0.0331954327,  -0.0857535568, ...
         -0.0294527709 ];
```

```
A2d = [   1.0000000000,   0.9606677419,  -0.0719892750,   0.1967478525, ...
          0.6937392675,   0.0537312473,   0.0094939713,   0.4075195921, ...
          0.2398750884,  -0.1734065812,  -0.2199845826,  -0.0225321833, ...
          0.0280327825,  -0.0424546744,   0.1504180003,   0.1377972091, ...
         -0.1589177955,  -0.1519252923,  -0.0883181943,  -0.1761217203, ...
         -0.0663234423 ];
```

The overall transfer function numerator and denominator polynomial coefficients are:

```
N2 = [   0.0184353357,   0.0725502109,   0.1058805938,   0.0630163444, ...
        -0.0132976993,  -0.0791271236,  -0.1180648201,  -0.0708767938, ...
         0.0438903725,   0.0854110712,   0.0022138335,  -0.0627680644, ...
         0.0334623407,   0.2043328578,   0.1996602079,  -0.0531745243, ...
        -0.2983873529,  -0.2449241281,   0.0421128907,   0.1803887288, ...
        -0.0000000000,  -0.1803887288,  -0.0421128907,   0.2449241281, ...
         0.2983873529,   0.0531745243,  -0.1996602079,  -0.2043328578, ...
        -0.0334623407,   0.0627680644,  -0.0022138335,  -0.0854110712, ...
        -0.0438903725,   0.0708767938,   0.1180648201,   0.0791271236, ...
         0.0132976993,  -0.0630163444,  -0.1058805938,  -0.0725502109, ...
        -0.0184353357 ];
```

```
D2 = [   1.0000000000,   2.2125114674,   1.3782205804,   0.0665460515, ...
         0.3586310116,   0.4225789889,   0.4533702868,   1.7158842924, ...
         1.9101929074,   0.2568782421,  -0.6413212076,  -0.2316684727, ...
         0.0680161820,   0.0888756075,   0.5774410742,   0.7751210811, ...
        -0.3192733604,  -1.1591610374,  -0.6757909569,  -0.3590446906, ...
        -0.3177723803,   0.1486135562,   0.2317027116,  -0.2514545889, ...
        -0.4527912707,  -0.3101733987,  -0.1528347986,  -0.0334780170, ...
         0.0985160968,   0.1471233636,   0.0253665323,  -0.0679675477, ...
        -0.0191628010,   0.0027320890,   0.0055804633,   0.0519491479, ...
         0.0547675695,   0.0265243952,   0.0199059148,   0.0108747438, ...
         0.0019534092 ];
```

Figure 10.119 shows the amplitude and delay response of the PCLS SOCP optimised parallel all-pass filter. Figure 10.120 shows the pass-band response of the filter.

Figure 10.119: Schur one-multiplier parallel all-pass lattice multi-band filter, response after SOCP PCLS optimisation.

Figure 10.120: Schur one-multiplier parallel all-pass lattice multi-band filter, pass-band response after SOCP PCLS optimisation.

Design of a parallel all-pass doubly-pipelined one-multiplier IIR Schur lattice low-pass filter using SOCP

The state variable description of an all-pass doubly-pipelined one-multiplier Schur lattice filter is described in Section 5.6.5. The Octave script *schurOneMPAlatticeDoublyPipelinedAntiAliased_socp_slb_lowpass_test.m* implements the design of a low-pass IIR filter composed of the sum of parallel all-pass doubly-pipelined one-multiplier Schur lattice filters in series with a parallel anti-aliasing filter using SOCP and PCLS optimisation. The specification of the filter is:

```
maxiter=20000 % Maximum iterations
ftol=0.001 % Tolerance on coef. update
ctol=1e-07 % Tolerance on constraints
rho=0.992188 % Constraint on reflection coefficients
ma=6 % Order of all-pass filter 1
mb=7 % Order of all-pass filter 2
maa=7 % Order of anti-aliasing filter
n=1000 % Frequency points across the band
fap=0.1 % Amplitude pass band upper edge
dBap=0.1 % Amplitude pass band peak-to-peak ripple(dB)
Wap=1 % Amplitude pass band weight
Wat=0.01 % Amplitude transition band weight
fas=0.175 % Amplitude stop band lower edge
dBas=60 % Amplitude stop band peak ripple(dB)
Was_wise=0.1 % Initial amplitude stop band weight
Was=100 % Amplitude stop band weight
fpp=0.1 % Pass band phase upper edge
pp=0 % Nominal pass band phase(rad./pi)
ppr=0.002 % Pass band phase peak-to-peak ripple(rad./pi)
Wpp=1 % Pass band phase weight
ftp=0.1 % Pass band group delay upper edge
tp=15 % Pass band group delay(samples)
tpr=0.2 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
fdp=0.1 % Pass band dAsqdw upper edge
dpr=2 % Pass band dAsqdw peak-to-peak ripple
Wdp=1 % Pass band dAsqdw weight
```

The initial parallel doubly-pipelined all-pass filters are designed by the *WISE_PA* function with the *WISE* method described in Section 8.1.5. The initial anti-aliasing filter is a Butterworth half-band low-pass filter. The SOCP PCLS optimised doubly-pipelined all-pass lattice coefficients are:

```
A1k2 = [  -0.1909752399,   0.1005910405,   0.0962671474,  -0.2031363145, ...
           0.2000142439,  -0.0664665304 ]';

A2k2 = [   0.0546169441,   0.5524948846,  -0.2449923964,  -0.1564182816, ...
           0.2446354074,  -0.1889996893,   0.0587559528 ]';
```

and the anti-aliasing filter all-pass lattice coefficients are:

```
Aaa1k2 = [   0.0000000000,   0.6710147378,   0.0000000000,   0.0261214655 ]';

Aaa2k2 = [   0.0000000000,   0.2257703302,   0.0000000000 ]';
```

The corresponding doubly-pipelined all-pass lattice denominator polynomial coefficients are:

```
DA1k2 = [   1.0000000000,   0.0000000000,  -0.2739817882,   0.0000000000, ...
            0.1078621238,   0.0000000000,   0.1398470740,   0.0000000000, ...
           -0.2532282826,   0.0000000000,   0.2173412399,   0.0000000000, ...
           -0.0664665304,   0.0000000000,  -0.0000000000 ];

DA2k2 = [   1.0000000000,   0.0000000000,  -0.1078495528,   0.0000000000, ...
            0.4356152412,   0.0000000000,  -0.1169859084,   0.0000000000, ...
           -0.2392959732,   0.0000000000,   0.2788990196,   0.0000000000, ...
           -0.1946840161,   0.0000000000,   0.0587559528,   0.0000000000, ...
           -0.0000000000 ];
```

and the anti-aliasing filter all-pass lattice denominator polynomial coefficients are:

```
DAaa1k2 = [   1.0000000000,   0.0000000000,   0.6885426261,   0.0000000000, ...
              0.0261214655 ];


DAaa2k2 = [   1.0000000000,   0.0000000000,   0.2257703302,   0.0000000000 ];
```

The overall transfer function numerator and denominator polynomial coefficients are:

```
N2 = [  -0.0000000000,  -0.0000000000,  -0.0000503529,  -0.0004352049, ...
        -0.0012489876,  -0.0014524832,   0.0003711412,   0.0044867671, ...
         0.0088668364,   0.0085287357,  -0.0005192398,  -0.0163055677, ...
        -0.0290486406,  -0.0204959770,   0.0239313861,   0.1094007439, ...
         0.2198663900,   0.3252483590,   0.3893845596,   0.3893845596, ...
         0.3252483590,   0.2198663900,   0.1094007439,   0.0239313861, ...
        -0.0204959770,  -0.0290486406,  -0.0163055677,  -0.0005192398, ...
         0.0085287357,   0.0088668364,   0.0044867671,   0.0003711412, ...
        -0.0014524832,  -0.0012489876,  -0.0004352049,  -0.0000503529, ...
        -0.0000000000,  -0.0000000000 ];


D2 = [   1.0000000000,   0.0000000000,   0.5324816153,   0.0000000000, ...
         0.4054867977,   0.0000000000,   0.3523697228,   0.0000000000, ...
        -0.4256311791,   0.0000000000,   0.2293163572,   0.0000000000, ...
        -0.0091431138,   0.0000000000,  -0.1160923718,   0.0000000000, ...
         0.1382492309,   0.0000000000,  -0.0877371214,   0.0000000000, ...
         0.0128140997,   0.0000000000,   0.0222233129,   0.0000000000, ...
        -0.0199927188,   0.0000000000,   0.0066418087,   0.0000000000, ...
         0.0006509996,   0.0000000000,  -0.0005574777,   0.0000000000, ...
        -0.0000230313,   0.0000000000,   0.0000000000,   0.0000000000, ...
         0.0000000000,   0.0000000000 ];
```

Figure 10.121 shows the amplitude and pass-band phase and delay response of the initial parallel all-pass, doubly-pipelined, anti-aliased filter. Figure 10.122 shows the amplitude and pass-band phase and delay response of the PCLS SOCP optimised parallel all-pass, doubly-pipelined, anti-aliased filter. In both figures the phase is adjusted for the nominal delay. The *schurOneMAPlatticeDoublyPipelined2Abcd.m* function adds an output delay of z^{-2} in order to place all coefficients in the transition matrix, A. The effect of this extra delay is not shown in Figure 10.122.

Figure 10.121: Amplitude and pass-band phase and group-delay responses of the initial doubly-pipelined and anti-aliased Schur one-multiplier parallel all-pass lattice low-pass filter. The phase is adjusted for the nominal delay. The effect of the extra output delay of the implementation is not shown.

Figure 10.122: Doubly-pipelined and anti-aliased Schur one-multiplier parallel all-pass lattice low-pass filter amplitude and pass-band phase and group-delay responses after SOCP PCLS optimisation. The phase is adjusted for the nominal delay. The effect of the extra output delay of the implementation is not shown.

10.3.4 Design of normalised-scaled IIR Schur lattice filters with SQP optimisation

Design of an approximately normalised-scaled IIR Schur lattice low-pass filter using SQP

The Octave script *schurNSlattice_sqp_slb_lowpass_test.m* implements the design of a lowpass approximately normalised-scaled IIR Schur lattice filter with PCLS and SQP. The filter coefficients are allowed to vary independently and the resulting filter is not, in fact, normalised-scaled. 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 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.

The specification of the filter is:

```
ftol=0.0001 % Tolerance on coefficient update
ctol=1e-05 % Tolerance on constraints
n=500 % 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=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=0.1 % Delay pass band peak-to-peak ripple
Wtp=0.01 % Delay pass band weight
fas=0.3 % Amplitude stop band edge
dBas=36 % amplitude stop band peak-to-peak ripple
Was=1000 % Amplitude stop band weight
```

Figures 10.123 and 10.124 show the overall and passband response of the filter after PCLS SQP optimisation. Figure 10.125 shows the pole-zero plot of the filter after PCLS SQP optimisation.

The PCLS SQP optimised Schur normalised-scaled all-pass lattice and numerator tap coefficients of the low-pass filter are:

```
s10_2 = [   0.9264513567,   0.1152940006,  -0.1192003216,  -0.0441970678, ...
            0.0232422300,   0.0259930923,  -0.0034407910,  -0.0128985630, ...
           -0.0024065496,   0.0054752723 ];


s11_2 = [   1.0052477837,   0.9082044912,   0.9761300906,   1.0094266892, ...
            1.0854082474,   1.0316983639,   1.0592144362,   1.0687664900, ...
            1.0739802683,   0.7412544053 ];


s20_2 = [  -0.5970533029,   0.7171705548,  -0.5056592608,  -0.1136577915, ...
            0.9839704466,   0.0441000000,   0.0000000000,   0.0000000000, ...
            0.0000000000,   0.0000000000 ];


s00_2 = [   0.8035005457,   0.5774865137,   0.7368565622,   0.8659868445, ...
            0.8044284234,   0.9999000000,   1.0000000000,   1.0000000000, ...
            1.0000000000,   1.0000000000 ];


s02_2 = [   0.7712564164,  -0.7446390770,   0.6815569172,  -0.5657532054, ...
            0.4200140379,  -0.1741629822,  -0.0000000000,  -0.0000000000, ...
           -0.0000000000,  -0.0000000000 ];


s22_2 = [   0.6615901421,   0.8950771164,   0.9158924916,   0.8838964529, ...
            0.9946619491,   0.9990271218,   1.0000000000,   1.0000000000, ...
            1.0000000000,   1.0000000000 ];
```

Figure 10.123: Schur approximately normalised-scaled lattice lowpass filter response after SQP PCLS optimisation.

Figure 10.124: Schur approximately normalised-scaled lattice lowpass filter passband response after SQP PCLS optimisation.

Figure 10.125: Schur approximately normalised-scaled lattice lowpass filter, pole-zero plot after SQP PCLS optimisation.

The corresponding transfer function numerator and denominator polynomial coefficients are:

```
N2 = [   0.0054752723,  -0.0114838987,   0.0039715166,   0.0032146168, ...
         0.0150119586,  -0.0129596118,  -0.0334792898,   0.0023991894, ...
         0.0979808775,   0.1190279517,   0.1013555284 ];


D2 = [   1.0000000000,  -1.7716074415,   2.0235811292,  -1.5414205768, ...
         0.8089777355,  -0.2790317564,   0.0502130091,  -0.0000000000, ...
         0.0000000000,   0.0000000000,  -0.0000000000 ];
```

The diagonal of the state covariance matrix, K, of the optimised filter is:

```
diag(K) = [      0.3205,      0.2841,      0.4807,      0.6414, ...
                 0.7546,      1.0449,      1.0000,      1.0000, ...
                 1.0000,      1.0000 ]';
```

Figure 10.126: R=2 Schur normalised-scaled lattice band-pass filter, response after SQP PCLS optimisation.

Design of a normalised-scaled R=2 IIR Schur lattice band-pass filter with SQP optimisation

The Octave script *schurNSlattice_sqp_slb_bandpass_R2_test.m* implements the design of a band-pass $R = 2$ IIR normalised-scaled Schur lattice filter with PCLS and SQP. The specification of the filter is:

```
ftol=0.001 % Tolerance on coef. update
ctol=0.0001 % Tolerance on constraints
n=500 % Frequency points across the band
dmax=0.050000 % Constraint on norm of coefficient SQP step size
rho=0.999000 % 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=0.1 % Delay pass band lower edge
ftpu=0.2 % Delay pass band upper edge
tp=16 % Nominal passband filter group delay
tpr=0.4 % Delay pass band peak-to-peak ripple
Wtp=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_R2_test.m*, as shown in Section 8.2.6. The denominator polynomial of the filter has coefficients in z^{-2} only.

Figures 10.126 and 10.127 show the overall and passband response of the band-pass $R = 2$ filter after PCLS SQP optimisation. Figure 10.128 shows the pole-zero plot of the band-pass $R = 2$ filter after PCLS SQP optimisation.

The SQP PCLS optimised $R = 2$ Schur normalised-scaled allpass lattice and numerator tap coefficients of the band-pass filter are:

Figure 10.127: R=2 Schur normalised-scaled lattice band-pass filter, passband response after SQP PCLS optimisation.

Figure 10.128: Pole-zero plot of an R=2 Schur normalised-scaled lattice band-pass filter after PCLS SQP optimisation.

```
s00_2 = [   1.0000000000,   0.4304707389,   1.0000000000,   0.8136550353, ...
            1.0000000000,   0.8570125954,   1.0000000000,   0.9192785414, ...
            1.0000000000,   0.9990000000,   1.0000000000,   0.7186802702, ...
            1.0000000000,   0.9032048197,   1.0000000000,   0.9990000000, ...
            1.0000000000,   0.5590235947,   1.0000000000,   0.7757650757 ];


s02_2 = [  -0.0000000000,  -0.7435495124,  -0.0000000000,  -0.9990000000, ...
           -0.0000000000,  -0.3589102359,  -0.0000000000,  -0.5249601199, ...
           -0.0000000000,  -0.3017501791,  -0.0000000000,  -0.5507362580, ...
           -0.0000000000,  -0.3199724996,  -0.0000000000,  -0.5403694270, ...
           -0.0000000000,   0.0116215565,  -0.0000000000,  -0.6715374450 ];


s22_2 = [   1.0000000000,   0.2013078460,   1.0000000000,   0.3317443588, ...
            1.0000000000,   0.6024974813,   1.0000000000,   0.9635435061, ...
            1.0000000000,   0.8673252673,   1.0000000000,   0.9900208878, ...
            1.0000000000,   0.9976768984,   1.0000000000,   0.0257515147, ...
            1.0000000000,   0.9990000000,   1.0000000000,   0.9925820455 ];


s20_2 = [   0.0000000000,   0.2296789714,   0.0000000000,   0.5310880322, ...
            0.0000000000,   0.4654455866,   0.0000000000,   0.5323578456, ...
            0.0000000000,  -0.1539912632,   0.0000000000,  -0.3284837727, ...
            0.0000000000,   0.5727035520,   0.0000000000,   0.9990000000, ...
            0.0000000000,  -0.4549281392,   0.0000000000,   0.1215766546 ];


s10_2 = [  -0.5455641117,  -1.1796608296,  -1.3183659915,  -0.5084196176, ...
            1.2799086240,   1.1512930556,   0.5867343115,  -0.2816699622, ...
           -0.5804802866,  -0.1720577660,   0.2529169056,  -0.5956111812, ...
           -0.4479295193,  -0.3962827696,   1.4947394695,   1.3216001293, ...
           -0.8396431811,   0.0394447506,   0.4937516994,   0.0075536940 ];


s11_2 = [   0.9963187340,   0.7647178626,   0.8944538664,   1.1350207335, ...
            1.2318043678,   0.7191083680,   1.0720298252,   1.1512028559, ...
            1.4296188213,   1.3570146852,   3.1238499960,   0.7589154197, ...
            0.5872098698,   2.3587807001,   2.7512852828,   1.9451040265, ...
            0.9817414197,   0.3753096910,   1.1618664685,   0.0441168299 ];
```

The corresponding transfer function denominator and numerator polynomial coefficients are:

```
N2 = [   0.0075536940,   0.0168983042,   0.0119248783,   0.0213256996, ...
         0.0231466579,   0.0359709987,   0.0131287796,  -0.0002010402, ...
        -0.0299961213,  -0.0172958221,  -0.0639881564,  -0.0869206673, ...
        -0.1219284163,  -0.0074210191,   0.1045729603,   0.2104757560, ...
         0.1124267004,  -0.0141119546,  -0.1517544834,  -0.1221201701, ...
        -0.0803650772 ];


D2 = [   1.0000000000,   0.0000000000,   1.3710371111,   0.0000000000, ...
         1.2831516744,   0.0000000000,   1.3211280533,   0.0000000000, ...
         1.2192743772,   0.0000000000,   0.9241287361,   0.0000000000, ...
         0.6286746990,   0.0000000000,   0.3549582599,   0.0000000000, ...
         0.1735203264,   0.0000000000,   0.0638793417,   0.0000000000, ...
         0.0165763294 ];
```

10.3.5 Design of parallel all-pass normalised-scaled IIR Schur lattice filters with SOCP optimisation

Design of a parallel all-pass approximately normalised scaled IIR Schur lattice low-pass filter with SOCP optimisation

The Octave script *schurNSPAlattice_socp_slb_lowpass_test.m* implements the design of a low-pass IIR filter composed of the difference of two parallel approximately normalised scaled Schur lattice all-pass filters with PCLS and SOCP optimisation. This design is only approximately normalised-scaled since the s20 and s00 coefficients are assumed to be independent rather than related by $\sigma_{00} = \sqrt{1 - \sigma_{20}^2}$. The design enforces $\sigma_{02} = -\sigma_{20}$ and $\sigma_{s22} = \sigma_{00}$. The initial parallel all-pass filters were designed by the Octave script *tarczynski_parallel_allpass_test.m*. The specification of the filter is:

```
ftol=1e-05 % Tolerance on coefficient update vector
ctol=1e-07 % Tolerance on constraints
n=800 % Frequency points across the band
rho=0.999000 % Constraint on allpass coefficients
fap=0.125 % Amplitude pass band edge
dBap=0.5 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=0.1 % 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
tp=11.5 % Nominal pass band filter group delay
tpr=0.04 % Delay pass band peak-to-peak ripple
Wtp=1 % Delay pass band weight
```

The low-pass filter SOCP PCLS optimised Schur approximately normalised scaled all-pass lattice coefficients are:

```
A1s20 = [   0.7826737409,  -0.0714157913,  -0.2755651077,  -0.1030535490, ...
           -0.1064124795,   0.2250673343,  -0.1249822938,   0.0241082769, ...
            0.1772275346,  -0.1635584607,   0.0456102900 ]';


A1s00 = [   0.6216764269,   0.9969039294,   0.9611721088,   0.9949938562, ...
            0.9952219812,   0.9744090381,   0.9927597047,   0.9989781124, ...
            0.9841483964,   0.9869941035,   0.9989616988 ]';


A2s20 = [   0.3671728947,  -0.2966842868,   0.2226927352,   0.2139762162, ...
           -0.0179779931,   0.0463578909,  -0.1985750068,   0.1844890161, ...
            0.0090345416,  -0.1826197059,   0.1437902801,  -0.0578143019 ]';


A2s00 = [   0.9295828394,   0.9539685973,   0.9750115069,   0.9773007370, ...
            0.9989562089,   0.9987933449,   0.9816924285,   0.9832933047, ...
            0.9989553810,   0.9835198952,   0.9897642702,   0.9987365353 ]';
```

Figure 10.129 shows the overall and passband amplitude and group delay responses of the filter after PCLS SOCP optimisation.

Figure 10.129: Response of a parallel all-pass approximately normalised-scaled Schur lattice low-pass filter after SOCP PCLS optimisation.

Design of a parallel all-pass approximately normalised scaled IIR Schur lattice band-pass Hilbert filter using SOCP

The Octave script *schurNSPAlattice_socp_slb_bandpass_hilbert_test.m* implements the design of a band-pass IIR Hilbert filter composed of the difference of two parallel normalised scaled Schur lattice all-pass filters with SOCP and PCLS optimisation. In fact, this design is only approximately normalised-scaled since the s20 and s00 coefficients are assumed to be independent rather than related by $\sigma_{00} = \sqrt{1 - \sigma_{20}^2}$. The design enforces $\sigma_{02} = -\sigma_{20}$ and $\sigma_{s22} = \sigma_{00}$. The specification of the filter is:

```
tol=0.0001 % Tolerance on coefficient update vector
ctol=5e-05 % Tolerance on constraints
difference=1 % difference of all-pass filters
rho=0.999000 % Constraint on allpass coefficients
sxx_symmetric=1 % enforce s02=-s20 and s22=s00
n=1000 % Frequency points across the band
fasl=0.05 % Amplitude stop band lower edge
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
fasu=0.25 % Amplitude stop band upper edge
dBap=0.08 % Amplitude pass band peak-to-peak ripple
dBas=37 % amplitude stop band peak-to-peak ripple
Wasl=400 % Amplitude lower stop band weight
Watl=0.001 % Amplitude transition band lower weight
Wap=1 % Amplitude pass band weight
Watu=0.001 % Amplitude upper transition band weight
Wasu=400 % Amplitude upper stop band weight
ftpl=0.11 % Delay pass band lower edge
ftpu=0.19 % Delay pass band upper edge
tp=16 % Nominal pass band filter group delay
tpr=0.008 % Delay pass band peak-to-peak ripple
Wtp=2 % Delay pass band weight
fppl=0.11 % Phase pass band lower edge
fppu=0.19 % Phase pass band upper edge
pp=3.5 % Nominal pass band filter phase
ppr=0.002 % Phase pass band peak-to-peak ripple
Wpp=100 % Phase pass band weight
```

The initial parallel all-pass filters were designed by the Octave script *tarczynski_parallel_allpass_bandpass_hilbert_test.m*. Figure 10.130 shows 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 Hilbert filter SOCP PCLS optimised approximately normalised scaled Schur all-pass lattice filter coefficients are:

```
A1s20 = [  -0.4812329751,   0.8503241630,  -0.2124283561,   0.0342135476, ...
            0.6695469917,  -0.2901542417,   0.0826051541,   0.4364261339, ...
           -0.3360523706,   0.2219576788 ]';


A1s00 = [   0.8710282238,   0.5596318356,   0.9823955308,   0.9881091846, ...
            0.7659994854,   0.9661658013,   0.9989918022,   0.8561304698, ...
            0.9520200400,   0.9789750210 ]';


A2s20 = [  -0.8198304545,   0.8705764133,  -0.2957706531,   0.0031230435, ...
            0.6891270586,  -0.2548445960,   0.0863898652,   0.4469016853, ...
           -0.3139675930,   0.2306942975 ]';


A2s00 = [   0.5798080571,   0.4601699282,   0.9548302130,   0.9975777367, ...
            0.7464463606,   0.9691397328,   0.9947343494,   0.8789664995, ...
            0.9494088496,   0.9765544442 ]';
```

The script simulates this filter with a uniform random noise input to the *schurNSlatticeFilter* function. Figure 10.131 shows the amplitude, phase and group delay responses of the simulated filter. The phase response shown is adjusted for the nominal delay. The responses are smoothed by a moving window filter.

Figure 10.130: Response of a parallel Schur approximately normalised scaled all-pass lattice band-pass Hilbert filter after SOCP PCLS optimisation. The phase response shown is adjusted for the nominal delay.

Figure 10.131: Simulated response of a parallel all-pass approximately normalised-scaled Schur lattice band-pass Hilbert filter. The phase response shown is adjusted for the nominal delay.

Figure 10.132: Simple frequency response masking filter.

Figure 10.133: 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 10.132. This simple frequency response masking filter technique is only suitable for designing narrow-band filters. Lim [261] describes the design of digital filters with sharp transition bands with wider bandwidths using FIR model and masking filters. Lu and Hinamoto [250] describe the design of digital filters with an IIR model filter and FIR masking filters using a structure, shown in Figure 10.133, 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 [79] 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 10.134. 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 10.135 [261, Section III, Figure 4]. Figure 10.135a shows the response of a prototype filter a with frequency response, $F_a(\omega)$, having passband and

Figure 10.134: Johansson and Wanhammar's frequency response masking filter structure.

stopband edge frequencies of θ and ϕ respectively. Figure 10.135b shows the response of the *complementary* filter $F_c(z) = z^{-D} - F_a(z)$. Figure 10.135c 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 10.135d, 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 10.135e, has passband edge $\omega_p = \frac{2m\pi+\theta}{M}$ and stopband edge $\omega_s = \frac{2m\pi+\phi}{M}$. Alternatively, Figures 10.135f and 10.135g 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 10.135a, for Figure 10.135e

$$\begin{aligned} m &= \lfloor \frac{\omega_p M}{2\pi} \rfloor \\ \theta &= \omega_p M - 2m\pi \\ \phi &= \omega_s M - 2m\pi \end{aligned}$$

and for Figure 10.135g

$$\begin{aligned} m &= \lceil \frac{\omega_s M}{2\pi} \rceil \\ \theta &= 2m\pi - \omega_s M \\ \phi &= 2m\pi - \omega_p M \end{aligned}$$

where $\lfloor x \rfloor$ and $\lceil x \rceil$ represent the largest integer less than x and the smallest integer larger than x respectively.

Figure 10.135: 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 [250, Section V] The FRM filter structure is shown in Figure 10.133. 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 [250, 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 = [  -0.0667264485,   0.2072379724,  -0.2564288787,   0.0445084083, ...
        0.2504619393,  -0.0228221368,  -0.8467977459,  -0.4400852109, ...
       -0.6461068568,   1.1962269573,  -0.1176617192 ]';
```

The model filter denominator polynomial is

```
d = [   1.0000000000,  -1.1392092158,   1.0801770789,  -1.0241484897, ...
        0.3983763818,   0.0747241150,  -0.2413146528,   0.1863163732, ...
       -0.0782003558,   0.0174506101,  -0.0016902169 ]';
```

The masking filter polynomial is

```
aa = [   0.0037168368,  -0.0091745988,  -0.0029947953,   0.0149019252, ...
        -0.0084125073,  -0.0131311784,   0.0227050116,   0.0020457162, ...
        -0.0214679889,   0.0345970647,  -0.0133656744,  -0.0454675155, ...
         0.0734989422,  -0.0105590899,  -0.1387286863,   0.2788564015, ...
         0.6784834603,   0.2788564015,  -0.1387286863,  -0.0105590899, ...
         0.0734989422,  -0.0454675155,  -0.0133656744,   0.0345970647, ...
        -0.0214679889,   0.0020457162,   0.0227050116,  -0.0131311784, ...
        -0.0084125073,   0.0149019252,  -0.0029947953,  -0.0091745988, ...
         0.0037168368 ]';
```

The complementary masking filter polynomial is

```
ac = [   0.0017264062,  -0.0048730648,  -0.0004511635,   0.0074951130, ...
        -0.0065898215,  -0.0051544916,   0.0136077603,  -0.0029613565, ...
        -0.0089449370,   0.0278671481,  -0.0228490112,  -0.0300028725, ...
         0.0718619282,  -0.0265100829,  -0.1223758020,   0.2864957398, ...
         0.6526963763,   0.2864957398,  -0.1223758020,  -0.0265100829, ...
         0.0718619282,  -0.0300028725,  -0.0228490112,   0.0278671481, ...
        -0.0089449370,  -0.0029613565,   0.0136077603,  -0.0051544916, ...
        -0.0065898215,   0.0074951130,  -0.0004511635,  -0.0048730648, ...
         0.0017264062 ]';
```

Figure 10.136: Second order cascade FRM filter response.

Figure 10.137: Second order cascade FRM filter passband response.

Figure 10.138: Second order cascade FRM model filter response.

Figure 10.139: Second order cascade FRM masking filter responses.

Figure 10.140: Comparison of the second order cascade FRM filter with a linear-phase FIR filter designed with *remez*.

Figure 10.136 shows the overall response of the resulting FRM filter. Figure 10.137 shows the passband response of the FRM filter. Figure 10.138 shows the decimated response of the FRM model filter. Figure 10.139 shows the responses of the FRM masking filters. Figure 10.140 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 10.133 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 10.141: 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 [250, 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 10.141 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 = [   0.0023072067,   0.0001004998,  -0.0027009187,  -0.0009118762, ...
        0.0070683537,   0.0022809590,  -0.0152588768,   0.0325623629, ...
       -0.0258497547,   0.0075253412,   0.0039808664 ]';
```

and the IIR model filter denominator polynomial is

```
d = [   1.0000000000,   0.5103321859,   0.8307417974,   0.3705503351, ...
        0.0900090972,  -0.0126066756,  -0.0129428074,  -0.0028778601, ...
       -0.0002557166,  -0.0000085521,  -0.0000008915 ]';
```

The FIR masking filter polynomial is

```
aa = [   0.1269145974,   0.1734400528,  -0.4578826228,   0.2829389922, ...
         0.1394182675,  -0.1805542094,   0.0675961726,   0.2274211424, ...
        -0.2391590255,  -0.0631222160,   0.3888974987,  -0.2334031908, ...
        -0.2629044691,   0.4008656834,   0.0826279326,  -0.4882750388, ...
```

Figure 10.142: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation, overall response.

```
 0.2621935021,   0.4023232824,  -0.3411513354,   0.5627021602, ...
 0.2816829763,   0.5627021602,  -0.3411513354,   0.4023232824, ...
 0.2621935021,  -0.4882750388,   0.0826279326,   0.4008656834, ...
-0.2629044691,  -0.2334031908,   0.3888974987,  -0.0631222160, ...
-0.2391590255,   0.2274211424,   0.0675961726,  -0.1805542094, ...
 0.1394182675,   0.2829389922,  -0.4578826228,   0.1734400528, ...
 0.1269145974 ]';
```

The complementary FIR masking filter polynomial is

```
ac = [  -0.0026969743,  -0.0087527898,   0.0166331172,  -0.0068670676, ...
        -0.0084816844,   0.0049259385,   0.0040698641,  -0.0113915352, ...
         0.0024609268,   0.0213832121,   0.0015428433,  -0.0417158570, ...
         0.0391893229,   0.0136891487,  -0.0441833606,   0.0173797890, ...
         0.0628603921,  -0.0759849998,  -0.0826548163,   0.2904856694, ...
         0.6122604039,   0.2904856694,  -0.0826548163,  -0.0759849998, ...
         0.0628603921,   0.0173797890,  -0.0441833606,   0.0136891487, ...
         0.0391893229,  -0.0417158570,   0.0015428433,   0.0213832121, ...
         0.0024609268,  -0.0113915352,   0.0040698641,   0.0049259385, ...
        -0.0084816844,  -0.0068670676,   0.0166331172,  -0.0087527898, ...
        -0.0026969743 ]';
```

Figure 10.142 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 10.143 shows the passband response of the resulting FRM filter. Figure 10.144 shows the responses of the resulting FRM masking filters. Figure 10.145 shows the decimated response of the resulting FRM model filter.

Figure 10.143: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation, passband response.

Figure 10.144: FRM gain-pole-zero format IIR model filter with SOCP and PCLS optimisation, masking filter responses.

Figure 10.145: 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 10.134, 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:

$$\begin{aligned} \frac{\partial\phi_Z(M\omega)}{\partial\omega} &= DM + \frac{\partial\phi_R(M\omega)}{\partial\omega} \\ \frac{\partial A(\omega)}{\partial\omega} &= \frac{\boldsymbol{a}_a^\top \boldsymbol{s}_a(\omega) + \boldsymbol{a}_c^\top \boldsymbol{s}_c(\omega)}{2} \\ \frac{\partial B(\omega)}{\partial\omega} &= \frac{\boldsymbol{a}_a^\top \boldsymbol{s}_a(\omega) - \boldsymbol{a}_c^\top \boldsymbol{s}_c(\omega)}{2} \\ \boldsymbol{s}_a(\omega) &= -2\left[\begin{array}{cccc} 0 & \sin\omega & \ldots & \frac{(n_a-1)}{2}\sin\frac{(n_a-1)}{2}\omega \end{array}\right]^\top \\ \boldsymbol{s}_c(\omega) &= -2\left[\begin{array}{cccc} 0 & \sin\omega & \ldots & \frac{(n_c-1)}{2}\sin\frac{(n_c-1)}{2}\omega \end{array}\right]^\top \end{aligned}$$

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:

```
ftol=0.0002 % Tolerance on coefficient update vector
ctol=2e-05 % Tolerance on constraints
n=500 % 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=0.311 % Stop band edge
dBas=40 % Stop band attenuation ripple
Was=10 % Stop band weight
rho=0.96875 % Constraint on allpass pole radius
```

The passband edge frequency is are the same as for the design example given by Lu and Hinamoto [250, 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 10.146 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,  -0.0370756433,   0.4913918693,   0.0164261157, ...
       -0.1026968244,  -0.0054910649,   0.0429327831,   0.0144934693, ...
       -0.0192442155,  -0.0006637845,   0.0027694354 ]';
```

Figure 10.146: FRM gain-pole-zero format allpass model filter with WISE optimisation, initial response.

The FIR masking filter polynomial is

```
aa = [  -0.0009226458,  -0.0031693694,   0.0101266721,  -0.0108964138, ...
         0.0013603971,   0.0074104163,  -0.0034798522,  -0.0090769471, ...
         0.0096483711,   0.0079157431,  -0.0079557068,  -0.0221451038, ...
         0.0425455947,  -0.0097745859,  -0.0420257754,   0.0317036517, ...
         0.0524409300,  -0.0849276757,  -0.0572188949,   0.2962867814, ...
         0.5836061881,   0.2962867814,  -0.0572188949,  -0.0849276757, ...
         0.0524409300,   0.0317036517,  -0.0420257754,  -0.0097745859, ...
         0.0425455947,  -0.0221451038,  -0.0079557068,   0.0079157431, ...
         0.0096483711,  -0.0090769471,  -0.0034798522,   0.0074104163, ...
         0.0013603971,  -0.0108964138,   0.0101266721,  -0.0031693694, ...
        -0.0009226458 ]';
```

The complementary FIR masking filter polynomial is

```
ac = [   0.0011481133,   0.0003668308,  -0.0044883786,   0.0087168929, ...
        -0.0080745365,   0.0013626580,   0.0079542019,  -0.0101715725, ...
         0.0017512779,   0.0123036363,  -0.0120933482,  -0.0079383402, ...
         0.0340947219,  -0.0369649293,   0.0030862766,   0.0490715254, ...
        -0.0651332060,   0.0033316095,   0.1306576061,  -0.2661213108, ...
        -0.6758347650,  -0.2661213108,   0.1306576061,   0.0033316095, ...
        -0.0651332060,   0.0490715254,   0.0030862766,  -0.0369649293, ...
         0.0340947219,  -0.0079383402,  -0.0120933482,   0.0123036363, ...
         0.0017512779,  -0.0101715725,   0.0079542019,   0.0013626580, ...
        -0.0080745365,   0.0087168929,  -0.0044883786,   0.0003668308, ...
         0.0011481133 ]';
```

Figure 10.147 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 10.148 shows the passband response of the resulting FRM filter. Figure 10.149 shows the responses of the resulting FRM masking filters. Figure 10.150 shows the response of the resulting FRM model filter.

Figure 10.147: FRM gain-pole-zero format allpass model filter with SOCP and PCLS optimisation, overall response.

Figure 10.148: FRM gain-pole-zero format allpass model filter, passband response with SOCP and PCLS optimisation.

Figure 10.149: FRM gain-pole-zero format allpass model filter with SOCP and PCLS optimisation, masking filter responses.

Figure 10.150: 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 10.134 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

$$\frac{\partial\boldsymbol{c}_R}{\partial\omega} = -\left[\boldsymbol{v} - \frac{\partial\phi_R(M\omega)}{\partial\omega}\right] \odot \boldsymbol{s}_R$$

$$
\begin{aligned}
\frac{\partial \boldsymbol{c}_S}{\partial \omega} &= -\left[\boldsymbol{v} - \frac{\partial \phi_S(M\omega)}{\partial \omega}\right] \odot \boldsymbol{s}_S \\
\frac{\partial \boldsymbol{s}_R}{\partial \omega} &= \left[\boldsymbol{v} - \frac{\partial \phi_R(M\omega)}{\partial \omega}\right] \odot \boldsymbol{c}_R \\
\frac{\partial \boldsymbol{s}_S}{\partial \omega} &= \left[\boldsymbol{v} - \frac{\partial \phi_S(M\omega)}{\partial \omega}\right] \odot \boldsymbol{c}_S
\end{aligned}
$$

The $\odot$ 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 10.151: 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 10.151. 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,  -0.4789292094,   0.7133959943,  -0.3331502818, ...
        0.0486974865,   0.0489595164,  -0.0181775201,  -0.0029581823, ...
        0.0050655685 ]';
```

and

```
s = [   1.0000000000,  -0.6634151167,   0.2705169341,   0.0214942878, ...
       -0.0643160251,   0.0160254478,   0.0075138995,   0.0015661902 ]';
```

Figure 10.152: FRM filter with gain-pole-zero format parallel allpass model filter, SOCP PCLS optimised overall response.

The FIR masking filter polynomial is

```
aa = [  -0.0033081614,  -0.0076254625,  -0.0221045656,  -0.0027477944, ...
         0.0240489687,  -0.0242908733,  -0.0466630146,   0.0636342753, ...
         0.0959178593,  -0.1189950023,  -0.1868789009,   0.1984905609, ...
         0.5416360430,   0.3801775548,   0.0603957369,  -0.0018029484, ...
         0.0456642320,  -0.0132802351,  -0.0395326160,   0.0145026696, ...
         0.0204129708,  -0.0132322155,  -0.0058610685,   0.0178827827, ...
         0.0222062470 ]';
```

and the complementary FIR masking filter polynomial is

```
ac = [  -0.0184879765,  -0.0470724068,   0.0106915006,   0.0462131283, ...
        -0.0355029980,  -0.0582434471,   0.0377568032,   0.0594476351, ...
         0.0091837562,  -0.0576066453,  -0.1473245910,   0.1046713944, ...
         0.5688366440,   0.4614500900,  -0.0211916804,  -0.0282162312, ...
         0.1503099440,  -0.0582223400,  -0.1411625578,   0.0633333804, ...
         0.0768208545,  -0.0590353950,  -0.0231963753,   0.0488868328, ...
         0.0144948323 ]';
```

Figure 10.152 shows the overall response of the resulting SOCP and PCLS optimised FRM filter. Figure 10.153 shows the passband response of the resulting FRM filter. Figure 10.154 shows the impulse response of the resulting FRM masking filters. Figure 10.155 shows the responses of the resulting FRM masking filters. Figure 10.156 shows the response of the resulting FRM model filter[c].

[c]The responses shown are calculated with the allpass filter polynomial derived from the gain-pole-zero form.

Figure 10.153: FRM filter with gain-pole-zero format parallel allpass model filter, SOCP PCLS optimised passband response.

Figure 10.154: FRM filter with gain-pole-zero format parallel allpass model filter, SOCP PCLS optimised FIR masking filters.

Figure 10.155: FRM filter with gain-pole-zero format parallel allpass model filter, SOCP PCLS optimised FIR masking filter responses.

Figure 10.156: 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 10.134 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 10.146. After SOCP and PCLS optimisation of the initial filter the resulting model filter all-pass Schur lattice filter has coefficients:

```
k1 = [  -0.0170488063,   0.5835334427,   0.0141768856,  -0.1421897203, ...
        -0.0090637979,   0.0586602510,   0.0068247121,  -0.0258140879, ...
        -0.0070798506,   0.0128622066 ]';
```

```
epsilon1 = [  1,  1, -1,  1, ...
              1, -1, -1,  1, ...
              1, -1 ];
```

The distinct coefficients of the FIR masking filter polynomial are

```
u1 = [   0.5770019542,   0.3015063993,  -0.0568543794,  -0.0848107814, ...
         0.0510353924,   0.0332810505,  -0.0421637355,  -0.0097686869, ...
         0.0391533517,  -0.0177578295,  -0.0139921117,   0.0079442439, ...
         0.0098973410,  -0.0086087034,  -0.0041465617,   0.0074889216, ...
         0.0011683628,  -0.0098549867,   0.0080752648,   0.0006680358, ...
        -0.0007619109 ]';
```

The distinct coefficients of the complementary FIR masking filter polynomial are

Figure 10.157: FRM filter with Schur one-multiplier lattice filter allpass model filter, amplitude, delay and phase responses after SOCP and PCLS optimisation.

```
v1 = [  -0.6677637407,  -0.2736258155,   0.1317115957,   0.0046718648, ...
        -0.0660797170,   0.0480285040,   0.0049245587,  -0.0361702491, ...
         0.0284806536,  -0.0025303063,  -0.0181838193,   0.0132298483, ...
         0.0027570119,  -0.0113319875,   0.0079502718,   0.0021081062, ...
        -0.0074323566,   0.0059063638,  -0.0015252669,  -0.0033094399, ...
         0.0018222005 ]';
```

Figure 10.157 shows the response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 10.158 shows the responses of the resulting FRM masking filters. Figure 10.159 shows the response of the resulting FRM model filter. Figure 10.160 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 10.158: FRM filter with Schur one-multiplier lattice allpass model filter, masking filter responses after SOCP and PCLS optimisation.

Figure 10.159: FRM filter with Schur one-multiplier lattice allpass model filter, model filter response after SOCP and PCLS optimisation.

Figure 10.160: 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 10.134. 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. [241, 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:

```
ftol=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 10.161 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 = [   0.5531474970,  -0.1244390352,   0.0416133287,  -0.0135756789, ...
         0.0021334103 ]';

epsilon2 = [  1,  1, -1,  1, ...
             -1 ];
```

The FIR masking filter polynomials are:

```
u2 = [  -0.0004599640,   0.0023567217,  -0.0070284251,   0.0129804885, ...
        -0.0309283574,   0.0346723005,  -0.0508947335,   0.0580268416, ...
         0.4385505465 ]';

v2 = [   0.0065942610,  -0.0044815273,   0.0066990076,  -0.0031543556, ...
        -0.0069802168,   0.0307163651,  -0.0814993830,   0.3141056047 ]';
```

Figure 10.162 shows the overall response of the resulting SOCP optimised, PCLS constrained FRM filter. Figure 10.163 shows the passband response of the resulting FRM filter. Figure 10.164 shows the responses of the resulting FRM masking filters. Figure 10.165 shows the response of the resulting FRM model filter.

Figure 10.161: FRM half-band filter, initial response.

Figure 10.162: FRM half-band filter, overall response after SOCP and PCLS optimisation .

Figure 10.163: FRM half-band filter, passband response after SOCP and PCLS optimisation.

Figure 10.164: 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 10.165: 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. [126] 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
ftol=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=0.02 % Pass band delay weight
fpp=0.01 % Phase pass band edge
fps=0.49 % Phase stop band edge
pp=-0.5 % Nominal passband phase(rad./pi)(adjusted for delay)
ppr=0.002 % Peak-to-peak pass band phase ripple(rad./pi)
Wpp=0.2 % 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 10.166 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 = [  -0.5737973427,  -0.1357905114,  -0.0532786375,  -0.0211277419, ...
        -0.0087726879 ]';


epsilon2 = [ -1,  1,  1,  1, ...
              1 ];
```

and FIR masking filter polynomials:

```
u2 = [  -0.0009286412,  -0.0025491082,  -0.0071011297,  -0.0128201595, ...
        -0.0309917540,  -0.0342912227,  -0.0517524688,  -0.0569899755, ...
         0.4399047223 ]';
```

Figure 10.166: FRM Hilbert filter, initial response.

```
v2 = [   0.0065502342,   0.0043736075,   0.0072012541,   0.0020581423, ...
        -0.0078892530,  -0.0311821987,  -0.0808713504,  -0.3144298329 ]';
```

Figure 10.167 shows the response of the FRM Hilbert filter after SOCP and PCLS optimisation.

Figure 10.167: 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 [117, 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, α, must be within the following region:
>
> $$\lceil \frac{r-1}{2} \rceil \leq \alpha \leq r - 1$$
>
> Because integer $\alpha \geq 1$,$r \geq 2$ must be assumed. In order to yield *maximum redundancy* in the balanced digit set Σ_r, one can choose the following value for the maximum magnitude
>
> $$\alpha = \lfloor \frac{r}{2} \rfloor$$

Further:

> The original motivation of using signed-digit number system is to eliminate carry propagation chains in addition (or subtraction). To break the carry chain, the lower bound on α should be made tighter as
>
> $$\lceil \frac{r+1}{2} \rceil \leq \alpha \leq r - 1$$

Parhi [118, Section 13.6] lists the following properties of the *canonical* binary signed-digit (CSD) representation of a binary signed-digit number $A = a_{W-1}a_{W-2}\cdots a_1a_0$ where each $a_i \in \{-1, 0, 1\}$:

- no 2 consecutive bits in a CSD number are non-zero
- the CSD representation of a number contains the 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 [118, 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* [118, 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. [263] 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 quantisation step-size to coefficient sensitivity ratio will lead to a good design"*.

Lim et al. first prove properties of the signed-digit representation [263, 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 Δx_k in coefficient x_k is:

$$\Delta H(\omega, k) \approx \frac{\partial H(\omega)}{\partial x_k}\Delta x_k$$

Lim et al. use the average of the coefficient sensitivity to define a cost, c_k, for the k'th coefficient:

$$c_k = 0.36\log_2 |x_k| + 0.36\log_2 \int_0^{\pi} \left| \frac{\partial H(\omega)}{\partial x_k} \right| d\omega$$

Given a total of R signed-digits, *Lim et al.* assign a single signed-digit at a time to the coefficient with the largest cost. After a coefficient is given a signed-digit, its cost is decreased by one. The process is repeated until all R digits have been allocated.

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

Ito et al. [210] describe a heuristic for allocating signed-digits to the coefficients of an FIR filter. Suppose $\boldsymbol{x} = \{x_1, \ldots, x_K\}$ are the floating-point 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.* [210].

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 10.3.1 with coefficients that are floating-point 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 filter is 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 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 Figures 11.6 and 11.7 compare the passband responses for 10-bit 2-signed-digit coefficients.

Figures 11.8, 11.9, 11.10, 11.11, 11.12, 11.13, and 11.14 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 5.29). 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 floating-point 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 11.1: Comparison of the cost function for an R=2 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 11.2: Comparison of the maximum stopband response in the region [0.26,0.5) for an R=2 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 an R=2 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 an R=2 Schur one-multiplier lattice bandpass filter with 6-bit to 16-bit integer coefficients found by rounding, pproximation 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 amplitude responses for an R=2 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 amplitude responses for an R=2 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 passband delay responses for an R=2 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 11.8: Comparison of the cost function for an R=2 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 maximum stopband response in the region [0.26,0.5) for an R=2 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 total number of signed-digits required to implement the coefficients of an R=2 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 11.11: Comparison of the estimated noise gain of an R=2 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 11.12: Comparison of the amplitude responses for an R=2 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 passband amplitude responses for an R=2 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 11.14: Comparison of the passband amplitude responses for an R=2 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.* .

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

This section compares the performance of a direct-form even-order symmetric FIR bandpass with coefficients that are floating-point, 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 11.15 compares the cost for each allocation method, Figure 11.16 compares the maximum stopband response in the frequency range $[0.26, 0.5)$, Figure 11.17 compares the total number of signed-digits required to implement the coefficient multipliers for 2 signed-digits allocated to each non-zero coefficient and Figures 11.18 and 11.19 compare the responses for 10-bit 2-signed-digit coefficients. Figures 11.20, 11.21, 11.22, 11.23 and 11.24 show the corresponding results for an allocation of 3 signed-digits to each non-zero 10-bit coefficient.

Figure 11.15: 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.* .

Figure 11.16: 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 11.17: 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 11.18: Comparison of the amplitude 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 11.19: Comparison of the pass-band amplitude 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 11.20: 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 11.21: 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 11.22: 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 11.23: Comparison of the amplitude 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.* .

Figure 11.24: Comparison of the pass-band amplitude 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 floating-point coefficients of the Schur one-multiplier lattice band-pass filter designed in Section 10.3.2. 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^{32} 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-flip* algorithm

Krukowski and Kale [7], describe a "bit-flip" algorithm for fixed-point filter design, shown in flow-graph form in Figure 13.1 (from [7, Figure 2]).

The bit-flip 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-flip algorithm is implemented in the Octave function *bitflip*.

In the following examples I apply the bit-flip 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-flip 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 floating-point filter coefficients and the filter coefficients approximated by 8-bit rounded, 8-bit rounding optimised with the bit-flip algorithm, 8-bit 2-signed-digits and 8-bit 2-signed-digits optimised with the bit-flip algorithm. The cost function weights the stop-band amplitude error by $W_{asl} = W_{asu} = 30$. Bit-flip starts at bit 6 of 8 bits and uses a mask size of 3 bits.

Figure 13.1: The bit-flip algorithm of *Krukowski and Kale* [7, Figure 2].

Figure 13.2: Amplitude responses of an direct-form R=2 IIR band-pass filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

13.1 Bit-flip search for the signed-digit coefficients of an R=2 direct-form bandpass IIR filter

The Octave script *bitflip_directIIR_bandpass_R2_test.m* applies the bit-flip algorithm to a direct-form $R = 2$ IIR bandpass filter with the filter transfer function polynomials found in Section 8.2.6. I do not attempt to ensure that the bit-flipped filter transfer function is stable. The *bitflip* optimised 8-bit 2 signed-digit direct-form IIR transfer function coefficients are:

```
n_bfsd = [        2,        1,        2,        1, ...
                  3,        1,        1,       -1, ...
                  1,       -1,       -5,      -12, ...
                -12,       -1,       14,       20, ...
                 12,       -3,      -14,      -12, ...
                 -5 ]/128;


d_bfsd = [      144,        0,      160,        0, ...
                192,        0,      192,        0, ...
                192,        0,      160,        0, ...
                120,        0,       72,        0, ...
                 36,        0,       14,        0, ...
                  3 ]/128;
```

A total of 22 adders is required to implement these signed-digit coefficients.

Figures 13.2 and 13.3 show the amplitude and group delay responses. Table 13.1 compares the cost result for each test.

Figure 13.3: Group-delay responses of a direct-form R=2 IIR band-pass filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Band-pass direct form R=2 IIR	Cost
Exact	1.4838
8-bit rounded	2.3565
8-bit rounded with bitflipping	1.5612
8-bit 2-signed-digit	5.6876
8-bit 2-signed-digit with bitflipping	3.2620

Table 13.1: Summary of the cost results for the direct form R=2 IIR bandpass filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

13.2 Bit-flip search for the signed-digit coefficients of a normalised-scaled lattice bandpass R=2 IIR filter

The Octave script *bitflip_schurNSlattice_bandpass_R2_test.m* applies the bit-flip algorithm to the example $R = 2$ bandpass filter implemented as a Schur normalised-scaled lattice filter. The *bitflip* optimised 8-bit 2 signed-digit lattice filter coefficients are:

```
s10_bfsd = [     -112,     -120,      -80,       -2, ...
                   72,       72,       28,      -15, ...
                  -24,       -7,        1,       -6, ...
                  -10,       -4,        4,        4, ...
                    0,       -1,        2,        2 ]'/128;


s11_bfsd = [       56,       48,       96,      128, ...
                   96,      120,      127,      127, ...
                  127,      130,      132,      128, ...
                  128,      128,      128,      128, ...
                  128,      128,      128,       68 ]'/128;


s20_bfsd = [        0,       64,        0,       31, ...
                    0,       24,        0,       34, ...
                    0,       60,        0,       32, ...
                    0,       32,        0,       63, ...
                    0,       63,        0,        4 ]'/128;


s00_bfsd = [      128,      112,      128,      120, ...
                  128,      120,      128,      112, ...
                  128,      120,      128,      112, ...
                  128,      127,      128,      127, ...
                  128,      128,      128,      128 ]'/128;


s02_bfsd = [        0,      -72,        0,      -48, ...
                    0,      -36,        0,      -56, ...
                    0,      -40,        0,      -40, ...
                    0,      -28,        0,      -20, ...
                    0,       -8,        0,       -4 ]'/128;


s22_bfsd = [      128,       96,      128,      126, ...
                  128,      120,      128,      112, ...
                  128,      127,      128,      120, ...
                  128,      127,      128,      127, ...
                  128,      128,      128,      128 ]'/128;
```

A total of 52 adders is required to implement these signed-digit coefficients.

Figures 13.4 and 13.5 show the amplitude and group delay responses of the filter. Table 13.2 compares the cost result for each test.

Figure 13.4: Amplitude responses of an R=2 band-pass filter synthesised as a normalised-scaled lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Figure 13.5: Group delay responses of an R=2 band-pass filter synthesised as a normalised-scaled lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Band-pass R=2 Schur normalised-scaled	Cost
Exact	1.4838
8-bit rounded	1.6404
8-bit rounded with bitflipping	1.0149
8-bit 2-signed-digit	2.8931
8-bit 2-signed-digit with bitflipping	1.4418

Table 13.2: Summary of the cost results for the R=2 bandpass filter synthesised as a normalised-scaled lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

13.3 Bit-flip search for the signed-digit coefficients of a one-multiplier lattice bandpass R=2 IIR filter

The Octave script *bitflip_schurOneMlattice_bandpass_R2_test.m* applies the bit-flip algorithm to the example bandpass filter implemented as a Schur one-multiplier lattice $R = 2$ filter. Note that, in this example, the one-multiplier state scaling coefficients are not truncated. The *bitflip* optimised 8-bit, rounded, one-multiplier lattice bandpass filter coefficients are:

```
k_bf = [        0,        70,        0,        45, ...
                0,        35,        0,        52, ...
                0,        39,        0,        40, ...
                0,        27,        0,        21, ...
                0,         9,        0,         4 ]/128;


c_bf = [        6,       -10,      -51,       -48, ...
               -1,        38,       58,        23, ...
               -8,       -12,       -5,         1, ...
               -3,        -6,       -3,         2, ...
                2,         0,        0,         1, ...
                1 ]/128;
```

A total of 34 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,        64,        0,        40, ...
                  0,        31,        0,        56, ...
                  0,        40,        0,        40, ...
                  0,        28,        0,        20, ...
                  0,         8,        0,         3 ]/128;


c_bfsd = [        6,       -10,      -48,       -48, ...
                 -1,        40,       60,        24, ...
                 -8,       -12,       -5,         1, ...
                 -3,        -6,       -3,         2, ...
                  2,         0,        0,         1, ...
                  1 ]/128;
```

A total of 20 adders is required to implement these signed-digit coefficients.

Figures 13.6 and 13.7 show the filter amplitude and group delay responses. Figures 13.8 and 13.9 show the filter amplitude and group delay 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. Table 13.3 compares the cost result for each test.

Figure 13.6: Amplitude responses of an R=2 band-pass filter synthesised as a one-multiplier lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Figure 13.7: Group-delay responses of an R=2 band-pass filter synthesised as a one-multiplier lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Figure 13.8: Amplitude responses of an R=2 band-pass filter synthesised as a one-multiplier lattice filter with floating-point 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-flip 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-flip algorithm.

Figure 13.9: Group-delay responses of an R=2 band-pass filter synthesised as a one-multiplier lattice filter with floating-point 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-flip 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-flip algorithm.

Band-pass R=2 Schur one-multiplier	Cost
Exact	1.4838
8-bit rounded	1.7069
8-bit rounded with bit-flipping	1.0890
8-bit 2-signed-digit	2.6675
8-bit 2-signed-digit with bit-flipping	1.8226
8-bit 2-signed-digit(Lim alloc.)	3.4210
8-bit 2-signed-digit(Lim alloc.) with bit-flipping	2.2064
8-bit 2-signed-digit(Ito alloc.)	4.5380
8-bit 2-signed-digit(Ito alloc.) with bit-flipping	3.4567

Table 13.3: Summary of the cost results for the bandpass R=2 filter synthesised as a one-multiplier lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip 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.

13.4 Bit-flip 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-flip 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 initial IIR filter in this example is that designed by the Octave script *schurOneMPAlattice_socp_slb_bandpass_test.m*. The *bitflip* optimised 8-bit, rounded, parallel-allpass one-multiplier lattice bandpass filter coefficients are:

```
A1k_bf = [      -50,       84,       67,      -69, ...
                 82,      -43,       -4,       45, ...
                -32,       20 ]/128;


A2k_bf = [      -96,       93,       67,      -75, ...
                 84,      -33,        2,       44, ...
                -35,       18 ]/128;
```

A total of 34 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:

```
A1k_bfsd = [      -48,       80,       72,      -68, ...
                   80,      -40,       -4,       48, ...
                  -32,       24 ]/128;


A2k_bfsd = [      -96,       96,       65,      -72, ...
                   80,      -33,        2,       48, ...
                  -36,       17 ]/128;
```

A total of 17 adders is required to implement these signed-digit coefficients.

Figures 13.10 and 13.11 show the filter amplitude and group delay responses with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Figures 13.12 and 13.13. show the filter amplitude and group delay 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. Table 13.4 compares the cost result for each test.

Band-pass Schur parallel-allpass one-multiplier	Cost
Exact	1.0878
8-bit rounded	1.9779
8-bit rounded with bit-flipping	1.3743
8-bit 2-signed-digit	8.2848
8-bit 2-signed-digit with bit-flipping	7.3066
8-bit 2-signed-digit(Lim alloc.)	15.3864
8-bit 2-signed-digit(Lim alloc.) with bit-flipping	5.7757
8-bit 2-signed-digit(Ito alloc.)	7.9759
8-bit 2-signed-digit(Ito alloc.) with bit-flipping	4.6859

Table 13.4: Summary of the cost results for the bandpass filter synthesised as a parallel-allpass one-multiplier lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip 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.

Figure 13.10: Amplitude responses of a band-pass filter synthesised as a parallel-allpass one-multiplier lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Figure 13.11: Group delay responses of a band-pass filter synthesised as a parallel-allpass one-multiplier lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Figure 13.12: Amplitude responses of a band-pass filter synthesised as a parallel-allpass one-multiplier lattice filter with floating-point 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-flip 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-flip algorithm.

Figure 13.13: Group-delay responses of a band-pass filter synthesised as a parallel-allpass one-multiplier lattice filter with floating-point 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-flip 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-flip algorithm.

Band-pass Schur FIR	Cost
Exact	2.1910
8-bit rounded	2.2011
8-bit rounded with bitflipping	2.0522
8-bit 2-signed-digit	2.2894
8-bit 2-signed-digit with bitflipping	2.0952

Table 13.5: Summary of the cost results for the FIR minimum-phase bandpass filter synthesised as a Schur FIR lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

13.5 Bitflipping search for the signed-digit coefficients of a minimum-phase band-pass FIR filter

Section 8.3.7 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 [266].

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 = [   0.0713258700,   0.1209281155,   0.0456067102,  -0.1338082647, ...
        -0.2413596270,  -0.1406146033,   0.0853228019,   0.2158600693, ...
         0.1473817378,  -0.0064564851,  -0.0787037564,  -0.0439798845, ...
         0.0004903782,  -0.0069780611,  -0.0290429531,  -0.0175756495, ...
         0.0121525097 ]';
```

The Schur FIR lattice multiplier coefficients of the initial filter are

```
k_ex = [   0.7029397495,   0.2279826972,  -0.3955567423,  -0.5882455490, ...
          -0.4532559631,   0.0080174573,   0.4231213590,   0.4912837385, ...
           0.2373642214,  -0.2331586460,  -0.5588886861,  -0.7324700649, ...
          -0.8699862154,   0.6534907148,  -0.5512846248,   0.1703801117 ];
```

The initial FIR filter has order 17, 16 lattice coefficients and 32 multiplies per sample which is similar to the one-multiplier Schur lattice bandpass filter of the previous example. The *bitflip* optimised 8-bit 2-signed-digit Schur FIR lattice coefficients are:

```
k_bfsd = [       80,       36,      -48,      -72, ...
                -56,       12,       63,       63, ...
                 56,      -30,      -72,      -96, ...
               -112,       80,      -72,       18 ]/128;
```

A total of 32 adders is required to implement these signed-digit coefficients.

Figures 13.14 and 13.15 show the initial filter amplitude and group delay responses and the filter responses after coefficient truncation. Table 13.5 compares the cost results for each truncation method.

Figure 13.14: Amplitude responses of a minimum-phase band-pass filter synthesised as a Schur FIR lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Figure 13.15: Group-delay responses of a minimum-phase band-pass filter synthesised as a Schur FIR lattice filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip 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 floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip 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.

13.6 Bit-flip search for the signed-digit coefficients of a symmetric bandpass FIR filter

Appendix N.4 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-flip 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 *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.

Figure 13.16 shows the initial filter response and the filter responses after coefficient truncation. The bit-flip 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 13.17.

Figure 13.16: Amplitude and group-delay responses of a direct-form symmetric FIR band-pass filter with floating-point coefficients, 8-bit rounded coefficients, 8-bit rounded coefficients optimised with the bit-flip algorithm, 8-bit 2-signed-digit coefficients and 8-bit 2-signed-digit coefficients optimised with the bit-flip algorithm.

Figure 13.17: Amplitude and group-delay responses of a direct-form symmetric FIR band-pass filter with floating-point 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-flip 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-flip algorithm.

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], [203, 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 floating-point 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.* [210] 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.

	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 floating-point coefficients, 8-bit 3-signed-digit coefficients and 8-bit 3-signed-digit coefficients optimised with the branch-and-bound 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
```

Figures 14.2 and 14.3 show the initial filter pass-band and stop-band amplitude responses and the filter responses after coefficient truncation to 8-bit 3-signed-digits and branch-and-bound search.

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.

Figure 14.2: Pass-band amplitude responses of a direct-form symmetric FIR band-pass filter with floating-point coefficients, 8-bit 3-signed-digit coefficients and 8-bit 3-signed-digit coefficients optimised with the branch-and-bound algorithm.

Figure 14.3: Stop-band amplitude responses of a direct-form symmetric FIR band-pass filter with floating-point 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 R=2 IIR filter

The Octave script *branch_bound_schurOneMlattice_bandpass_R2_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 10.3.1 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 ϵ one-multiplier lattice cofficients are not recalculated since that would require rescaling the c_{min} tap coefficients. Figure 14.4 compares the pass-band responses of the filter with floating-point coefficients, the initial 8-bit 3-signed-digit coefficients and 8-bit 3-signed-digit coefficients found by branch-and-bound search. Figure 14.5 shows the filter stop-band response and Figure 14.6 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 14.4: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass R=2 filter with 8-bit 3-signed-digit coefficients found by branch-and-bound search.

Figure 14.5: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass R=2 filter with 8-bit 3-signed-digit coefficients found by branch-and-bound search

Figure 14.6: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass R=2 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 R=2 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 R=2 IIR filter

The Octave script *branch_bound_schurOneMlattice_bandpass_R2_10_nbits_test.m* uses the branch-and-bound heuristic to optimise the response of the band-pass $R = 2$ Schur one-multiplier lattice filter of Section 10.3.1. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
cscale=2 % Scaling for c tap coefficients
npoints=250 % Frequency points across the band
% length(c0)=21 % Num. tap coefficients
% sum(k0~=0)=10 % Num. non-zero all-pass coef.s
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=1 % Delay pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
Wasl=100000 % Amplitude lower stop band weight
Wasu=10000 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 10 bits and 3 signed-digits. 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 branch with the lowest error is selected.

The filter coefficients found by the branch-and-bound search are:

```
k_min = [        0,      336,        0,      255, ...
                 0,      176,        0,      216, ...
                 0,      152,        0,      129, ...
                 0,       78,        0,       54, ...
                 0,       19,        0,        7 ]'/512;


c_min = [       73,      -14,     -312,     -492, ...
              -164,      126,      416,      312, ...
                18,      -88,      -81,      -13, ...
               -11,      -37,      -27,        5, ...
                26,       17,        3,        1, ...
                 5 ]'/1024;
```

Figure 14.7 compares the pass-band responses of the filter with floating-point coefficients, 10-bit 3-signed-digit coefficients and 10-bit 3-signed-digits found with branch-and-bound search. Figure 14.8 shows the filter stop-band response and Figure 14.9 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 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 14.7: Comparison of the pass-band amplitude responses for an R=2 Schur one-multiplier lattice bandpass filter with 10-bit 3-signed-digit coefficients and performing branch-and-bound search.

Figure 14.8: Comparison of the stop-band amplitude responses for an R=2 Schur one-multiplier lattice bandpass filter with 10-bit 3-signed-digit coefficients performing branch-and-bound search.

Figure 14.9: Comparison of the pass-band group delay responses for an R=2 Schur one-multiplier lattice bandpass filter with 10-bit 3-signed-digit coefficients and performing branch-and-bound search.

	Cost	Signed-digits	Shift-and-adds
Exact	7.9778e-03		
10-bit 3-signed-digit	1.3944e-02	79	48
10-bit 3-signed-digit(branch-and-bound)	9.8387e-03	81	50

Table 14.3: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for an R=2 Schur one-multiplier lattice bandpass filter with 10-bit 3-signed-digit coefficients and performing branch-and-bound search.

14.4 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 $R = 2$ Schur one-multiplier lattice filter of Section 10.3.2. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
use_schurOneMlattice_allocsd_Lim=1
use_schurOneMlattice_allocsd_Ito=0
ftol=0.0001 % Tolerance on coefficient update
ctol=1e-07 % Tolerance on constraints
maxiter=1000 % SQP iteration limit
npoints=250 % Frequency points across the band
% length(c0)=21 % Num. tap coefficients
% sum(k0~=0)=20 % 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=0.4 % 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=0.2 % Delay pass band peak-to-peak ripple
Wtp=1 % Delay pass band weight
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=1000000 % Amplitude lower stop band weight
Wasu=100000 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 10 bits and an average of 3 signed-digits allocated with the heuristic of *Lim et al.*. 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 PCLS SQP optimised with the filter specification given above. For this example the branch-and-bound search makes thousands of branches and requires many hours CPU time on my PC.

The filter coefficients found by the branch-and-bound search are:

```
k_min = [     -447,      504,     -336,      418, ...
              -304,      423,     -340,      391, ...
              -305,      258,      -94,       23, ...
                11,        8,        0,        7, ...
                -9,       14,       -8,        4 ]'/512;


c_min = [      -98,      -63,     -449,     -319, ...
                34,      509,      511,     -336, ...
               -80,        2,        9,       -8, ...
                -3,       10,       12,        4, ...
                 0,        1,        2,        0, ...
                -3 ]'/512;
```

Figure 14.10 compares the pass-band responses of the filter with floating-point coefficients, 10-bit 3-signed-digit coefficients allocated with the heuristic of *Lim et al.* and 10-bit 3-signed-digits found with branch-and-bound search. Figure 14.11 shows the filter stop-band response and Figure 14.12 shows the filter pass-band group delay response. Table 14.4 compares the cost and the number of 10 bit shift-and-add operations required to implement the coefficient multiplications found with the branch-and-bound search. 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 14.10: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit 3-signed-digit coefficients allocated with the heuristic of *Lim et al.* and performing branch-and-bound search.

Figure 14.11: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice bandpass filter with 10-bit 3-signed-digit coefficients allocated with the heuristic of *Lim et al.* and performing branch-and-bound search.

Figure 14.12: Comparison of the pass-band group delay responses for a Schur one-multiplier lattice bandpass filter with 10-bit 3-signed-digit coefficients allocated with the heuristic of *Lim et al.* and performing branch-and-bound search.

	Cost	Signed-digits	Shift-and-adds
Exact	2.6459e-03		
10-bit 3-signed-digit(Lim)	2.1510e-02	97	57
10-bit 3-signed-digit(branch-and-bound)	1.7949e-03	90	52

Table 14.4: 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 3-signed-digit coefficients allocated with the heuristic of *Lim et al.* and performing branch-and-bound search.

14.5 Branch-and-bound search for the 10-bit 3-signed-digit coefficients of a one-multiplier pipelined lattice band-pass filter

The Octave script *branch_bound_schurOneMlatticePipelined_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 10.3.2 designed with SOCP as described in Section 5.6.3 and implemented in the pipelined form. The $k_{n-1}k_n$ and $c_{2n-1}k_{2n}$ coefficient combinations are treated as additional coefficients. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
npoints=250 % Frequency points across the band
% length(c0)=21 % Num. tap coefficients
% sum(k0~=0)=20 % Num. non-zero all-pass coef.s
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
Wap=10 % 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=1 % Delay pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
Wasl=100000 % Amplitude lower stop band weight
Wasu=10000 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 10 bits with 3 signed-digits. 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.

The inputs to the branch-and-bound search are the upper and lower bounds of the truncated exact coefficients. At each branch the script selects the coefficient with the largest difference between upper and lower signed-digit approximations and selects the branch with the lowest response error value. The filter coefficients found by the branch-and-bound search are:

```
k_min = [     -450,      497,     -352,      416, ...
              -296,      416,     -336,      392, ...
              -296,      265,      -94,       23, ...
                13,       11,       -1,        7, ...
               -14,       15,       -9,        2 ]'/512;


c_min = [      -98,      -65,     -452,     -320, ...
                22,      510,      736,     -352, ...
               -94,       -7,        8,      -13, ...
               -10,       10,       15,        5, ...
                -2,        2,        4,        3, ...
                -4 ]'/512;


kk_min = [     -432,     -336,     -280,     -242, ...
               -249,     -276,     -257,     -232, ...
               -158,      -50,       -5,        0, ...
                  1,       -1,       -1,       -1, ...
                 -1,       -1,       -1 ]'/512;


ck_min = [      -62,        0,     -255,        0, ...
                416,        0,     -260,        0, ...
                 -5,        0,       -1,        0, ...
                  0,        0,        0,        0, ...
                  1,        0,        0 ]'/512;
```

Figure 14.13 compares the pass-band responses of the filter with floating-point coefficients, 10-bit signed-digit coefficients and 10-bit signed-digits found with branch-and-bound search. Figure 14.14 shows the filter stop-band response and Figure 14.15 shows the filter pass-band group delay response. Table 14.5 compares the cost and the number of 10 bit shift-and-add operations required to implement the coefficient multiplications found with the branch-and-bound search.

Figure 14.13: Comparison of the pass-band amplitude responses for a Schur one-multiplier pipelined lattice bandpass filter with 10-bit 3 signed-digit coefficients found by branch-and-bound search.

Figure 14.14: Comparison of the stop-band amplitude responses for a Schur one-multiplier pipelined lattice bandpass filter with 10-bit 3 signed-digit coefficients found by branch-and-bound search.

Figure 14.15: Comparison of the pass-band group delay responses for a Schur one-multiplier pipelined lattice bandpass filter with 10-bit 3 signed-digit coefficients found by branch-and-bound search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.0226		
10-bit 3-signed-digit	0.4562	136	77
10-bit 3-signed-digit(branch-and-bound)	0.0496	147	81

Table 14.5: Comparison of the cost and number of 10 bit shift-and-add operations required to implement the coefficient multiplications for a pipelined Schur one-multiplier pipelined lattice bandpass filter with 10-bit 3 signed-digit coefficients found by branch-and-bound search.

14.6 Branch-and-bound search for the 13-bit signed-digit coefficients of a Schur lattice band-pass Hilbert R=2 IIR filter

The Octave script *branch_bound_schurOneMlattice_bandpass_hilbert_R2_13_nbits_test.m* uses the branch-and-bound heuristic to optimise the response of the band-pass $R = 2$ Schur one-multiplier lattice filter of Section 10.3.2. The filter specification is:

```
nbits=13 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
n=1000 % Frequency points across the band
% length(c0)=21 % Num. tap coefficients
% sum(k0~=0)=10 % Num. non-zero all-pass coef.s
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
Wap=0.1 % Amplitude pass band weight
Watl=0.001 % Amplitude lower transition band weight
Watu=0.001 % Amplitude upper transition band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
Wasl=1 % Amplitude lower stop band weight
Wasu=10000 % Amplitude upper stop band weight
fppl=0.1 % Phase pass band lower edge
fppu=0.2 % Phase pass band upper edge
pp=3.5 % Nominal passband filter phase(rad./pi)
Wpp=0.1 % Phase 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
Wtp=0.1 % Delay pass band weight
fdpl=0.1 % dAsqdw pass band lower edge
fdpu=0.2 % dAsqdw pass band upper edge
dp=0 % Nominal passband filter dAsqdw
Wdp=0.01 % dAsqdw pass band weight
```

The filter coefficients are truncated to 13 bits and an average of 3 signed-digits allocated with the heuristic of *Ito et al.*. 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 and selects the branch with the lowest response error value.

The filter coefficients found by the branch-and-bound search are:

```
k_min = [      -447,       504,      -336,       418, ...
               -304,       423,      -340,       391, ...
               -305,       258,       -94,        23, ...
                 11,         8,         0,         7, ...
                 -9,        14,        -8,         4 ]'/512;


c_min = [       -98,       -63,      -449,      -319, ...
                 34,       509,       511,      -336, ...
                -80,         2,         9,        -8, ...
                 -3,        10,        12,         4, ...
                  0,         1,         2,         0, ...
                 -3 ]'/512;
```

Figure 14.16 compares the pass-band responses of the filter with floating-point coefficients, 13-bit 3-signed-digit coefficients allocated with the heuristic of *Lim et al.* and 10-bit 3-signed-digits found with branch-and-bound search. Figure 14.17 shows the filter stop-band response and Figure 14.18 shows the filter pass-band phase response adjusted for the nominal filer group delay. Figure 14.19 shows the filter pass-band group delay response. Table 14.6 compares the cost and the number of 13 bit shift-and-add operations required to implement the coefficient multiplications found with the branch-and-bound search.

Figure 14.16: Comparison of the pass-band amplitude responses for a Schur one-multiplier lattice R=2 bandpass Hilbert filter with 13-bit coefficients having an average of 3-signed-digits allocated with the heuristic of *Ito et al.* and performing branch-and-bound search.

Figure 14.17: Comparison of the stop-band amplitude responses for a Schur one-multiplier lattice R=2 bandpass Hilbert filter with 13-bit coefficients having an average of 3-signed-digits allocated with the heuristic of *Ito et al.* and performing branch-and-bound search.

Figure 14.18: Comparison of the pass-band phase responses for a Schur one-multiplier lattice R=2 bandpass Hilbert filter with 13-bit coefficients having an average of 3-signed-digits allocated with the heuristic of *Ito et al.* and performing branch-and-bound search. The phase responses are adjusted for the nominal filter group-delay.

Figure 14.19: Comparison of the pass-band group-delay responses for a Schur one-multiplier lattice R=2 bandpass Hilbert filter with 13-bit coefficients having an average of 3-signed-digits allocated with the heuristic of *Ito et al.* and performing branch-and-bound search.

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Exact	2.5803e-03	-33.99		
13-bit 3-signed-digit	3.4972e-03	-31.39	88	57
13-bit 3-signed-digit(Ito)	2.4835e-03	-33.23	84	53
13-bit 3-signed-digit(b-and-b)	1.9652e-03	-33.60	89	58

Table 14.6: Comparison of the cost and number of 13-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier lattice R=2 bandpass Hilbert filter with 13-bit coefficients having an average of 3-signed-digits allocated with the heuristic of *Ito et al.* and performing branch-and-bound search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.00027853		
16-bit 4-signed-digit	0.00161778	123	92
16-bit 4-signed-digit(branch-and-bound)	0.00063430	122	91

Table 14.7: Comparison of the cost and number of 16-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier pipelined lattice low-pass filter with 16 bit 4 signed-digit coefficients found by branch-and-bound search.

14.7 Branch-and-bound search for the 16-bit 4-signed-digit coefficients of a one-multiplier pipelined lattice low-pass filter

The Octave script *branch_bound_schurOneMlatticePipelined_lowpass_16_nbits_test.m* performs branch-and-bound search to optimise the response of a Schur one-multiplier pipelined lattice low-pass filter implemented in the pipelined form shown in Section 5.6.3 with 16 bit 4 signed-digit coefficients. The $k_{n-1}k_n$ and $c_{2n-1}k_{2n}$ coefficient combinations are treated as additional coefficients. The inputs to the branch-and-bound search are the upper and lower bounds of the truncated exact coefficients. At each branch the script selects the coefficient with the largest difference between upper and lower signed-digit approximations and selects the branch with the lowest response error value. The filter specification is:

```
N=9 % Filter order
nbits=16 % Coefficient bits
ndigits=4 % Nominal average coefficient signed-digits
npoints=1000 % Frequency points across the band
% length(c0)=10 % Num. tap coefficients
% sum(k0~=0)=9 % Num. non-zero all-pass coef.s
fap=0.15 % Amplitude pass band edge
Wap=1 % Amplitude pass band weight
Wat=0.01 % Amplitude transition band weight
fas=0.2 % Amplitude stop band edge
Was=100000000 % Amplitude stop band weight
```

The signed-digit lattice coefficients found by the branch-and-bound search are:

```
k_min = [   -20288,    32016,   -26112,    28992, ...
            -27632,    26688,   -23424,    14912, ...
             -4165 ]'/32768;


c_min = [    -8138,    -3296,     8440,    20996, ...
             84480,    15552,     2132,     1720, ...
               281,       42 ]'/32768;


kk_min = [   -19840,   -25472,   -23040,   -24446, ...
             -22496,   -19072,   -10624,    -1900 ]'/32768;


ck_min = [    -3232,        0,    18576,        0, ...
              12672,        0,      782,        0 ]'/32768;
```

Figure 14.20 shows the amplitude pass-band response of the filter with 16-bit 4-signed-digit coefficients found by branch-and-bound search. Figure 14.21 shows the corresponding filter stop-band amplitude response. Table 14.7 compares the cost and the number of 16 bit shift-and-add operations required to implement the coefficient multiplications found by branch-and-bound search.

Figure 14.20: Pass band amplitude response for a Schur one-multiplier pipelined lattice low-pass filter with 16 bit 4 signed-digit coefficients found by branch-and-bound search.

Figure 14.21: Stop band amplitude response for a Schur one-multiplier pipelined lattice low-pass filter with 16 bit 4 signed-digit coefficients found by branch-and-bound search.

14.8 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a one-multiplier lattice low-pass differentiator R=2 filter

The Octave script *branch_bound_schurOneMlattice_lowpass_differentiator_R2_12_nbits_test.m* performs branch-and- bound search to optimise the response of a Schur one-multiplier lattice low-pass differentiator $R = 2$ filter implemented as a $1 - z^{-1}$ filter followed by a correction filter. The correction filter has 12 bit coefficients with an average of 3 signed-digits, allocated with the heuristic of *Lim et al.*, and a denominator polynomial having only terms in z^{-2}. The inputs to the branch-and-bound search are the upper and lower bounds of the truncated exact coefficients. At each branch the script selects the coefficient with the largest difference between upper and lower signed-digit approximations and selects the branch with the lowest response error value. The filter specification is:

```
nbits=12 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
ftol=0.0001 % Tolerance on coefficient update
ctol=1e-08 % Tolerance on constraints
maxiter=1000 % SQP iteration limit
n=1000 % Frequency points across the band
% length(c0)=11 % Num. tap coefficients
% sum(k0~=0)=5 % Num. non-zero all-pass coef.s
dmax=0.250000 % Constraint on norm of coefficient SQP step size
rho=0.999512 % Constraint on allpass coefficients
fap=0.2 % Amplitude pass band edge
Arp=0.001 % Amplitude pass band peak-to-peak ripple
Wap=2 % Amplitude pass band weight
Art=0.02 % Amplitude transition band peak-to-peak ripple
Wat=0.0001 % Amplitude transition band weight
fas=0.4 % Amplitude stop band edge
Ars=0 % Amplitude stop band peak-to-peak ripple
Was=1 % Amplitude lower stop band weight
fpp=0.2 % Phase pass band edge
pp=1.5 % Nominal passband filter phase(rad./pi)
ppr=0.0006 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=1 % Phase pass band weight
ftp=0.2 % Group delay pass band edge
tp=9 % Nominal passband filter group delay(samples)
tpr=0.006 % Group delay pass band peak-to-peak ripple
Wtp=1 % Group delay pass band weight
fdp=0.2 % Correction filter dCsqdw pass band edge
cpr=0.04 % Correction filter dCsqdw pass band peak-to-peak ripple
Wdp=0.005 % Correction filter dCsqdw pass band weight
```

The numbers of signed digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
k_allocsd_digits = [  0,  4,  0,  3, ...
                      0,  2,  0,  2, ...
                      0,  3 ]';


c_allocsd_digits = [  3,  4,  4,  3, ...
                      4,  3,  3,  3, ...
                      1,  3,  3 ]';
```

The correction filter signed-digit lattice coefficients found by the branch-and-bound search are:

```
k_min = [        0,      420,        0,      -57, ...
                 0,       18,        0,       -8, ...
                 0,        2 ]'/2048;


c_min = [      -44,     -448,     -588,      -62, ...
               149,      -35,      -47,       35, ...
                 2,      -13,        5 ]'/2048;
```

The corresponding correction filter transfer function polynomials are:

Figure 14.22: Pass-band amplitude error response of a Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial coefficients only in z^{-2}, with 12 bit coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing branch-and-bound search.

```
N_min = [   0.0024414062,  -0.0063414574,   0.0014616626,   0.0157439326, ...
           -0.0227064177,  -0.0133058291,   0.0672092889,  -0.0328849775, ...
           -0.2604138131,  -0.2580922194,  -0.0834874571 ];


D_min = [   1.0000000000,   0.0000000000,   0.1990876198,   0.0000000000, ...
           -0.0259700550,   0.0000000000,   0.0079858585,   0.0000000000, ...
           -0.0037118248,   0.0000000000,   0.0009765625 ];
```

Figure 14.22, Figure 14.23, Figure 14.24 and Figure 14.25 show the pass-band amplitude error (compared to a desired response of $\frac{\omega}{2}$), stop-band amplitude, pass-band phase and pass-band group-delay responses of the low-pass differentiator filter. Figure 14.26 shows the pole-zero plot of the low-pass differentiator filter with signed-digit coefficients found by branch-and-bound search. Table 14.8 compares the cost of the correction filter 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	2.7112e-05		
12-bit 3-signed-digit	9.1055e-05	41	25
12-bit 3-signed-digit(Lim)	5.7238e-05	42	26
12-bit 3-signed-digit(branch-and-bound)	1.2285e-05	41	25

Table 14.8: Comparison of the cost of the correction filter and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a Schur one-multiplier lattice low-pass differentiator filter, having denominator polynomial coefficients only in z^{-2}, with 12-bit integer coefficients found by allocating an average of 3 signed-digits to each coefficient using the heuristic of *Lim et al.* found by performing branch-and-bound search.

Figure 14.23: Stop-band amplitude response of a Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial coefficients only in z^{-2}, with 12 bit coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing branch-and-bound search.

Figure 14.24: Pass-band phase response of a Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial coefficients only in z^{-2}, with 12 bit coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim 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 of a Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial coefficients only in z^{-2}, with 12 bit coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing branch-and-bound search.

Figure 14.26: Pole-zero plot of the Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial coefficients only in z^{-2} with 12 bit, 3 signed-digit coefficients found by performing branch-and-bound search.

14.9 Branch-and-bound search for the 12-bit 3-signed-digit coefficients of a parallel all-pass one-multiplier 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 10.3.3 with 12-bit integer coefficients. The initial filter is found by the Octave script *schurOneMPAlattice_socp_slb_lowpass_test.m*. 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
NA1k=11 % Allpass model filter 1 denominator order
NA2k=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
tp=11.5 % Nominal pass band filter group delay
tpr=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 ϵ coefficients can be recalculated.

The numbers of signed digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
A1k_allocsd_digits = [  4,  3,  3,  1, ...
                        1,  2,  5,  1, ...
                        3,  6,  3 ]';


A2k_allocsd_digits = [  3,  3,  2,  3, ...
                        2,  3,  3,  3, ...
                        2,  3,  4,  6 ]';
```

The signed-digit lattice coefficients found by the heuristic of *Ito et al.* are:

```
A1k0_sd = [     1576,     -176,     -548,     -128, ...
                -128,      504,     -295,       -8, ...
                 336,     -326,      108 ]'/2048;


A2k0_sd = [      800,     -560,      384,      336, ...
                 -96,       84,     -416,      368, ...
                  12,     -368,      308,     -111 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are:

```
A1k_min = [     1592,     -168,     -536,     -128, ...
                -128,      480,     -296,       -4, ...
                 352,     -337,      120 ]'/2048;
```

Figure 14.27: 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.

	Cost	Signed-digits	Shift-and-adds
Exact	0.000194		
12-bit 3-signed-digit(Ito)	0.000456	63	40
12-bit 3-signed-digit(SOCP b-and-b)	0.000151	60	37

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 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.

```
A2k_min = [      800,     -580,      384,      336, ...
                 -96,       84,     -400,      368, ...
                  10,     -376,      321,     -123 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 60 signed-digits and 37 shift-and-add operations.

Figure 14.27 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 14.28 shows the corresponding filter pass-band group-delay response. Figure 14.29 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.

Figure 14.28: 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 14.29: 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.

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

The Octave script *branch_bound_schurNSPAlattice_lowpass_12_nbits_test.m* performs branch-and-bound search to optimise the response of the parallel approximately normalised-scaled all-pass Schur lattice low-pass filter of Section 10.3.5 with 12-bit integer coefficients. The filter specification is:

```
sxx_symmetric=1 % Enforce s02=-s20 and s22=s00
nbits=12 % Coefficient word length
ndigits=3 % Average number of signed digits per coef.
tol=0.001 % Tolerance on coefficient update vector
ctol=1e-06 % Tolerance on constraints
n=400 % Frequency points across the band
difference=0 % Use difference of all-pass filters
rho=0.999000 % Constraint on allpass coefficients
fap=0.125 % Amplitude pass band edge
dBap=0.6 % 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=45 % amplitude stop band peak-to-peak ripple
Was=1000 % Amplitude stop band weight
ftp=0.175 % Delay pass band edge
tp=11.5 % Nominal pass band filter group delay
tpr=0.1 % Delay pass band peak-to-peak ripple
Wtp=1 % Delay pass band weight
```

The filter coefficients are truncated to 12 bits and 3 signed-digits. The script enforces the symmetric relations $s_{02} = -s_{20}$ and $s_{22} = s_{00}$. The script does not enforce normalised-scaling with the relation $s_{02} = \sqrt{1 - s_{00}^2}$. 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 are:

```
A1s20_sd = [     1600,     -146,     -560,     -208, ...
                 -216,      464,     -256,       49, ...
                  368,     -336,       92 ]'/2048;


A1s00_sd = [     1272,     2042,     1968,     2038, ...
                 2038,     2000,     2033,     2046, ...
                 2016,     2020,     2046 ]'/2048;


A2s20_sd = [      752,     -608,      456,      440, ...
                  -37,       95,     -400,      376, ...
                   19,     -376,      296,     -118 ]'/2048;


A2s00_sd = [     1904,     1952,     2000,     2000, ...
                 2046,     2046,     2012,     2014, ...
                 2046,     2014,     2028,     2045 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are:

```
A1s20_min = [     1600,     -124,     -552,     -208, ...
                  -228,      456,     -265,       52, ...
                   368,     -336,       94 ]'/2048;


A1s00_min = [     1272,     2045,     1968,     2042, ...
                  2046,     1985,     2038,     2043, ...
                  2018,     2024,     2040 ]'/2048;
```

Figure 14.30: Pass-band amplitude responses for a parallel all-pass approximately normalised-scaled Schur lattice low-pass filter with 12 bit 3 signed-digit integer coefficients found by direct truncation and by performing branch-and-bound search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.002675		
12-bit 3-signed-digit	0.003258	260	168
12-bit 3-signed-digit(SOCP b-and-b)	0.002346	260	168

Table 14.10: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a parallel all-pass approximately normalised-scaled Schur lattice low-pass filter with 12 bit, 3 signed-digit integer coefficients found by direct truncation and by performing branch-and-bound search.

```
A2s20_min = [      752,     -608,      440,      440, ...
                   -34,      104,     -416,      376, ...
                    18,     -376,      296,     -120 ]'/2048;


A2s00_min = [     1904,     1968,     1976,     2015, ...
                  2044,     2038,     2032,     2008, ...
                  2030,     2024,     2024,     2043 ]'/2048;
```

Figures 14.30 and 14.31 show the pass-band amplitude and group delay responses of the filter with floating-point coefficients and with 12-bit, 3-signed-digit coefficients found by simple truncation and by branch-and-bound search. Figure 14.32 shows the corresponding stop-band amplitude response.

Table 14.10 compares the cost and the number of 12 bit shift-and-add operations required to implement the signed-digit coefficient multiplications.

Figure 14.31: Pass-band group delay responses for a parallel all-pass approximately normalised-scaled Schur lattice low-pass filter with 12 bit 3 signed-digit integer coefficients found by direct truncation and by performing branch-and-bound search.

Figure 14.32: Stop-band amplitude responses for a parallel all-pass approximately normalised-scaled Schur lattice low-pass filter with 12 bit 3 signed-digit integer coefficients found by direct truncation and by performing branch-and-bound search.

14.11 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=0.0001 % 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 14.33 shows the amplitude response of the filter with 8-bit, 2-signed-digit coefficients and branch-and-bound search. Figure 14.34 shows the filter pass-band amplitude response. Table 14.11 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.

Figure 14.33: 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 14.34: 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.

	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 14.11: 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.

14.12 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 numbers of signed digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
A1k_allocsd_digits = [  4,  3,  5,  5, ...
                        4,  4 ]';

A2k_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 14.35 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 14.36 shows the amplitude stop-band response. Table 14.12 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 14.35: 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 14.36: 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 14.12: 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.

14.13 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 10.3.3 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=46.000000 % Stop band amplitude response ripple(dB)
Wasl=20000 % Lower stop band amplitude response weight
Wasu=20000 % 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 numbers of signed digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
A1k_allocsd_digits = [  3,  4,  2,  3, ...
                        3,  4,  1,  4, ...
                        2,  3 ]';


A2k_allocsd_digits = [  6,  4,  2,  4, ...
                        3,  2,  2,  3, ...
                        3,  2 ]';
```

The signed-digit lattice coefficients found by the heuristic of *Ito et al.* are:

```
A1k0_sd = [     -800,     1360,     1040,    -1104, ...
                1312,     -688,      -64,      728, ...
                -508,      316 ]'/2048;


A2k0_sd = [    -1535,     1512,     1040,    -1192, ...
                1344,     -520,       28,      704, ...
                -568,      288 ]'/2048;
```

Figure 14.37: 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.

	Cost	Signed-digits	Shift-and-adds
Exact	0.011057		
12-bit 3-signed-digit(Ito)	0.014413	57	37
12-bit 3-signed-digit(SOCP b-and-b)	0.009863	58	38

Table 14.13: 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.

The signed-digit lattice coefficients found by the branch-and-bound search are:

```
A1k_min = [     -800,     1360,     1040,    -1092, ...
                1312,     -688,      -32,      688, ...
                -480,      292 ]'/2048;


A2k_min = [    -1531,     1503,     1032,    -1186, ...
                1344,     -528,       48,      672, ...
                -529,      272 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 58 signed-digits and 38 shift-and-add operations.

Figure 14.37 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 14.38 shows the corresponding filter pass-band group-delay response. Figure 14.39 shows the corresponding filter stop-band amplitude response. 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 signed-digit allocation heuristic of *Ito et al.* with the branch-and-bound search.

Figure 14.38: 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 14.39: 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.

14.14 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 10.3.3 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
NA1k=10 % Allpass model filter 1 denominator order
NA2k=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=20 % 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=37.000000 % 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
tp=16.000000 % Pass band nominal group-delay response(samples)
tpr=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
pp=3.500000 % Pass band nominal phase response(rad./pi)
ppr=0.040000 % Pass band phase response ripple(rad./pi)
Wpp=10 % Pass band phase response weight
fdpl=0.1 % Pass band dAsqdw response lower edge
fdpu=0.2 % Pass band dAsqdw response upper edge
dp=0.000000 % Pass band nominal dAsqdw response
dpr=0.400000 % Pass band dAsqdw response ripple
Wdp=10 % Pass band dAsqdw 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 = [     -232,      432,     -134,       33, ...
                 352,     -168,       38,      271, ...
                -191,      134 ]'/512;


A2k_min = [     -416,      446,     -158,        8, ...
                 352,     -152,       52,      268, ...
                -184,      142 ]'/512;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 57 signed-digits and 37 shift-and-add operations.

Figure 14.40: 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.

	Cost	Signed-digits	Shift-and-adds
Exact	0.001521		
10-bit 3-signed-digit	0.130427	59	39
10-bit 3-signed-digit(SOCP b-and-b)	0.003424	57	37

Table 14.14: 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.

Figure 14.40 shows the amplitude pass-band response of the filter with 10-bit, 3-signed-digit coefficients. Figure 14.41 shows the corresponding filter pass-band phase response (in multiples of π and adjusted for the nominal delay). Figure 14.42 shows the corresponding filter pass-band group-delay response. Figure 14.43 shows the corresponding filter stop-band amplitude response. Table 14.14 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 14.41: 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 14.42: 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 14.43: 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.

14.15 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 10.3.3 with 12-bit integer coefficients. The filter specification is:

```
use_best_branch_and_bound_found=1
enforce_pcls_constraints_on_final_filter=0
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
NA1k=10 % Allpass model filter 1 denominator order
NA2k=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=20 % 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=37.000000 % 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
tp=16.000000 % Pass band nominal group-delay response(samples)
tpr=0.160000 % 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
pp=3.500000 % Pass band nominal phase response(rad./pi)
ppr=0.002000 % Pass band phase response ripple(rad./pi)
Wpp=10 % Pass band phase response weight
fdpl=0.1 % Pass band dAsqdw response lower edge
fdpu=0.2 % Pass band dAsqdw response upper edge
dp=0.000000 % Pass band nominal dAsqdw response
dpr=0.400000 % Pass band dAsqdw response ripple
Wdp=10 % Pass band dAsqdw 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 numbers of signed-digits allocated to each lattice coefficient by the heuristic of *Ito et al.* are:

```
A1k_allocsd_digits = [  3,  4,  3,  3, ...
                        4,  3,  3,  3, ...
                        2,  2 ]';


A2k_allocsd_digits = [  4,  3,  3,  1, ...
                        4,  3,  2,  3, ...
                        4,  3 ]';
```

The 12-bit, average of 3-signed-digit, lattice coefficients allocated with the algorithm of *Ito et al.* are:

Figure 14.44: 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.

```
A1k0_sd = [     -928,     1696,     -472,      104, ...
                1392,     -672,      208,     1072, ...
                -768,      544 ]'/2048;


A2k0_sd = [    -1656,     1800,     -608,       32, ...
                1424,     -608,      224,     1072, ...
                -744,      552 ]'/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 = [     -944,     1680,     -509,      136, ...
                1376,     -672,      208,     1064, ...
                -768,      576 ]'/2048;


A2k_min = [    -1632,     1800,     -624,       64, ...
                1424,     -592,      192,     1080, ...
                -744,      584 ]'/2048;
```

The signed-digit lattice coefficients found by the branch-and-bound search are implemented with 59 signed-digits and 39 shift-and-add operations.

Figure 14.44 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 14.45 shows the corresponding filter pass-band phase response (in multiples of π and adjusted for the nominal delay). Figure 14.46 shows the corresponding filter pass-band group-delay response. Figure 14.47 shows the corresponding filter stop-band amplitude response. Table 14.15 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 14.45: 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 14.46: 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 14.47: 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.

	Cost	Signed-digits	Shift-and-adds
Exact	0.001521		
12-bit 3-signed-digit(Ito)	0.004182	60	40
12-bit 3-signed-digit(SOCP b-and-b)	0.001027	59	39

Table 14.15: 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.

14.16 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 heuristic of *Lim et al.*. The numbers of signed-digits allocated to the coefficients of the all-pass lattice filter are:

```
k_allocsd_digits = [  2,  4,  1,  4, ...
                      1,  4,  3,  3, ...
                      3,  4 ]';
```

The numbers of signed-digits allocated to the distinct coefficients of the FIR masking filter are:

```
u_allocsd_digits = [  5,  5,  4,  2, ...
                      4,  3,  4,  3, ...
                      4,  2,  4,  3, ...
                      3,  3,  2,  2, ...
                      1,  3,  3,  2, ...
                      1 ]';
```

The numbers 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,  3,  3,  2, ...
                      3,  3,  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 = [      -34,     1200,       32,     -291, ...
               -16,      121,       14,      -52, ...
               -13,       22 ]'/2048;
```

[b]The search was stopped after about 36 hours.

Figure 14.48: Comparison of the amplitude responses of an FRM low-pass filter with floating-point coefficients, with 12 bit coefficients having 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,      617,     -117,     -160, ...
               104,       68,      -95,      -21, ...
                79,      -36,      -28,       17, ...
                14,      -18,       -8,       20, ...
                 4,      -20,       16,        0, ...
                -2 ]'/2048;


v_min = [    -1364,     -559,      270,        9, ...
              -136,      100,       11,      -74, ...
                58,       -4,      -36,       28, ...
                 5,      -23,       17,        5, ...
               -15,       11,       -4,       -6, ...
                 6 ]'/2048;
```

Figures 14.48, 14.49, and 14.50 compare the amplitude, phase and group delay responses of the FRM low-pass filter with floating-point coefficients 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 14.16 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.

Figure 14.49: Comparison of the pass-band phase responses of an FRM low-pass filter with floating-point coefficients, with 12 bit coefficients having an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by branch-and-bound search.

Figure 14.50: Comparison of the pass-band delay responses of an FRM low-pass filter with floating-point coefficients, with 12 bit coefficients having an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by branch-and-bound search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.001692		
12-bit 3-signed-digit(Lim)	0.003160	122	70
12-bit 3-signed-digit(branch-and-bound)	0.001086	126	75

Table 14.16: 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 having an average of 3-signed-digits allocated by the method of *Lim et al.* and with the coefficients found by branch-and-bound search.

14.17 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of a FRM Hilbert filter

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 14.51, 14.52 and 14.53 compare the amplitude, phase and delay responses of the FRM Hilbert filter with floating-point coefficients, with 12-bit 2-signed-digit coefficients and with the 12-bit 2-signed-digit coefficients found by branch-and-bound search. Figure 14.54 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 14.17 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 14.51: Comparison of the amplitude response of an FRM Hilbert filter with floating-point coefficients and with 12 bit 2-signed digit integer coefficients found by branch-and-bound search

Figure 14.52: Comparison of the phase response of an FRM Hilbert filter with floating-point coefficients and with 12 bit 2-signed digit integer coefficients found by branch-and-bound search. The phase response shown is adjusted for the nominal delay.

Figure 14.53: Comparison of the delay response of an FRM Hilbert filter with floating-point coefficients and with 12 bit 2-signed digit integer coefficients found by branch-and-bound search.

Figure 14.54: 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 14.17: 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.

14.18 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of a 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.3.1. 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 numbers of signed-digits allocated to each coefficient with the heuristic of *Ito et al.* are:

```
hM_allocsd_digits = [  4,  4,  1,  1, ...
                       1,  1,  1,  4, ...
                       1,  1,  1,  1, ...
                       1,  1,  1,  1, ...
                       2,  2,  1,  1, ...
                       2,  2,  2,  2, ...
                       2,  1,  2,  2, ...
                       2,  3,  4,  2, ...
                       4,  2,  2,  3, ...
                       3,  2,  3,  4 ]';
```

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 14.55, compares the amplitude responses of the FIR Hilbert filter with floating-point coefficients, 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 14.18 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.

Figure 14.55: Comparison of the amplitude response of a direct-form FIR Hilbert filter with floating-point coefficients and with 12-bit 2-signed digit integer coefficients found by branch-and-bound search.

	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 14.18: 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.

	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 14.19: 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.

14.19 Branch-and-bound search for the 12-bit 2-signed-digit coefficients of a 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.3.1. 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
hM_allocsd_digits = [  2,  1,  1,  1, ...
                       1,  2,  4,  4 ]';
```

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 14.56, compares the amplitude responses of the FIR Hilbert band-pass filter with floating-point coefficients, 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 14.19 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.

Figure 14.56: Comparison of the amplitude response of a direct-form FIR Hilbert bandpass filter with floating-point coefficients and with 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

Figure 15.1: Comparison of the pass-band 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.

	Cost	Signed-digits	Shift-and-adds
Exact	0.000931		
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.

```
hM_allocsd_digits = [  1,  1,  2,  6, ...
                       6,  2,  1,  1, ...
                       2,  3,  1,  3, ...
                       4,  6,  6,  3 ]';
```

The distinct direct-form symmetric FIR bandpass filter coefficients found by the SOCP-relaxation search are:

```
hM_min = [       -1,      -16,      -33,      -15, ...
                 39,       68,       32,       -8, ...
                 28,       82,       -2,     -236, ...
               -366,     -142,      302,      530 ]'/2048;
```

Figures 15.1 and 15.2 compares the responses of the filter with floating-point 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 15.1 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.

Figure 15.2: Comparison of the stop-band 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.

15.2 SOCP-relaxation search for the signed-digit coefficients of a direct-form anti-symmetric low-pass differentiator FIR filter

The Octave script *socp_relaxation_directFIRantisymmetric_lowpass_differentiator_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of a direct-form anti-symmetric lowpass differentiator 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
fap=0.2 % Amplitude pass band edge
Arp=0.002 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.01 % Amplitude transition band peak-to-peak ripple
Wat=0.01 % Amplitude transition band weight
fas=0.4 % Amplitude stop band edge
Ars=0.004 % Amplitude stop band peak-to-peak ripple
Was=10 % Amplitude stop band weight
```

The initial filter is a truncated version of a maximally-flat low-pass differentiator FIR filter designed with the method of *Selesnick* [88], described in Appendix N.7.4. 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 *directFIRantisymmetric_socp_mmse*. The results of this MMSE optimisation are then passed to *directFIRantisymmetric_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 initial floating point distinct direct-form anti-symmetric lowpass differentiator filter FIR coefficients are:

```
hM0 = [  -0.0002859946,   0.0003307233,   0.0004481107,  -0.0015612582, ...
          0.0007039229,   0.0032247172,  -0.0054052856,  -0.0016067748, ...
          0.0137599771,  -0.0106624797,  -0.0180520188,   0.0395606279, ...
         -0.0007558041,  -0.0853239242,   0.0845169514,   0.2521949203 ]';
```

The numbers of signed-digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
hM_allocsd_digits = [  1,  2,  1,  2, ...
                       2,  2,  3,  2, ...
                       3,  4,  3,  5, ...
                       2,  5,  5,  6 ]';
```

The distinct direct-form anti-symmetric FIR lowpass differentiator filter coefficients found by the SOCP-relaxation search are:

```
hM_min = [       -1,        1,        1,       -3, ...
                  1,        7,      -11,       -3, ...
                 28,      -22,      -37,       81, ...
                 -2,     -175,      174,      516 ]'/2048;
```

Figure 15.3 compares the responses of the filter with floating-point coefficients, 12-bit 3 signed-digit coefficients, 12-bit coefficients having an average of 3 signed-digits allocated with the algorithm of *Lim et al.* and 12-bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and SOCP-relaxation search. Table 15.2 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 *Lim et al.* with SOCP-relaxation search.

Figure 15.3: Comparison of the amplitude responses for a direct-form anti-symmetric FIR lowpass differentiatorfilter 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.

	Cost	Signed-digits	Shift-and-adds
Exact	1.0770e-03		
12-bit 3-signed-digit	1.0941e-03	33	17
12-bit 3-signed-digit(Lim)	1.0786e-03	36	20
12-bit 3-signed-digit(SOCP-relax)	1.0789e-03	35	19

Table 15.2: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a direct-form anti-symmetric lowpass differentiator FIR 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.

15.3 SQP-relaxation search for the signed-digit coefficients of a lattice bandpass R=2 IIR filter

The Octave script *sqp_relaxation_schurOneMlattice_bandpass_R2_10_nbits_test.m* performs successive SQP relaxations to optimise the response of the SQP optimised band-pass $R = 2$ Schur one-multiplier lattice filter of Section 10.3.1. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
ftol=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=2 % Delay pass band weight
fasl=0.05 % Amplitude stop band(1) lower edge
fasu=0.25 % Amplitude stop band(1) upper edge
dBas=33 % Amplitude stop band(1) peak-to-peak ripple
fasll=0.04 % Amplitude stop band(2) lower edge
fasuu=0.26 % Amplitude stop band(2) upper edge
dBass=36 % Amplitude stop band(2) peak-to-peak ripple
Wasl=500000 % Amplitude lower stop band weight
Wasu=1e+06 % 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
k_allocsd_digits = [  0,  4,  0,  1, ...
                      0,  3,  0,  4, ...
                      0,  3,  0,  1, ...
                      0,  3,  0,  3, ...
                      0,  2,  0,  1 ]';


c_allocsd_digits = [  6,  1,  3,  2, ...
                      3,  1,  3,  3, ...
                      2,  6,  2,  1, ...
                      6,  6,  6,  6, ...
                      2,  1,  6,  1, ...
                      1 ]';
```

The filter coefficients found by the SQP-relaxation search are:

Figure 15.4: Comparison of the pass-band amplitude responses for an R=2 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.

```
k_min = [        0,      329,        0,      256, ...
                 0,      176,        0,      215, ...
                 0,      148,        0,      128, ...
                 0,       76,        0,       52, ...
                 0,       18,        0,        8 ]'/512;


c_min = [       38,       -8,     -156,     -248, ...
               -81,       64,      200,      152, ...
                 8,      -43,      -40,       -8, ...
                -4,      -17,      -12,        2, ...
                12,        8,        1,        0, ...
                 2 ]'/512;
```

Figure 15.4 compares the pass-band responses of the filter with floating-point coefficients, 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 15.5 shows the filter stop-band response and Figure 15.6 shows the filter pass-band group delay response. Table 15.3 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 SQP-relaxation search. A further 51 additions are required by the lattice filter structure.

Figure 15.5: Comparison of the stop-band amplitude responses for an R=2 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 15.6: Comparison of the pass-band group delay responses for an R=2 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	0.0155		
10-bit 3-signed-digit	0.0361	73	42
10-bit 3-signed-digit(Ito)	0.0654	65	34
10-bit 3-signed-digit(SQP-relax)	0.0192	63	33

Table 15.3: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for an R=2 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.

15.4 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 10.3.1 with 10-bit signed-digit coefficients. The filter specification is:

```
ftol=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 numbers of signed-digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
k_allocsd_digits = [  5,  5,  5,  5, ...
                      4,  1,  0,  0, ...
                      0,  0 ]';


c_allocsd_digits = [  3,  5,  4,  3, ...
                      3,  2,  3,  1, ...
                      1,  1,  0 ]';
```

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 15.7 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. Figures 15.8 and 15.9 show the corresponding filter pass-band response. Figure 15.10 shows the filter pole-zero plot. Table 15.4 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 15.7: Amplitude 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 15.8: Pass-band amplitude 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 15.9: Pass-band 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 15.10: 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 15.4: 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 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.

15.5 SOCP-relaxation search for the signed-digit coefficients of a lattice low-pass R=2 IIR filter

The Octave script *socp_relaxation_schurOneMlattice_lowpass_R2_13_nbits_test.m* performs successive SOCP relaxations to optimise the response of the SOCP optimised low-pass R=2 Schur one-multiplier lattice filter of Section 10.3.2 with 13-bit 4-signed-digit coefficients and transfer function denominator polynomial coefficients only in z^{-2}. The filter coefficients are truncated to 13 bits allocated with an average of 4 signed-digits by the heuristic of *Ito et al.* as shown in Section 11.2. The filter specification is:

```
socp_relaxation_schurOneMlattice_lowpass_R2_allocsd_Lim=0
socp_relaxation_schurOneMlattice_lowpass_R2_allocsd_Ito=1
nbits=13 % Bits-per-coefficient
ndigits=4 % Average signed-digits-per-coefficient
ftol=0.0001 % Tolerance on coef. update
ctol=5e-07 % Tolerance on constraints
n=1000 % Frequency points across the band
fap=0.15 % Amplitude pass band upper edge
dBap=0.1 % Amplitude pass band peak-to-peak ripple(dB)
Wap=1 % Amplitude pass band weight
fas=0.18 % Amplitude stop band lower edge
dBas=55 % Amplitude stop band peak-to-peak ripple(dB)
Was=5e+06 % Amplitude stop band weight
```

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. This script includes an option to first successively optimise the approximations to the reflection coefficients, recalculate the Schur orthogonal basis coefficients and the tap coefficients and then successively optimise the approximations to the tap coefficients.

The $R = 2$ lattice filter coefficients found by the SOCP-relaxation search are:

```
k_min = [        0,     1216,        0,     3951, ...
                 0,      -72,        0,     2560, ...
                 0,    -1152,        0,      960, ...
                 0,     -446,        0,      133 ]'/4096;

c_min = [      200,      388,      254,       24, ...
             -1216,    -2392,    -2128,     -448, ...
               917,     1744,     1572,     1441, ...
               824,      496,      208,       72, ...
                14 ]'/4096;
```

The corresponding filter transfer function polynomials are:

```
N_min = [   0.0034179688,   0.0170073509,   0.0501049067,   0.1093038492, ...
            0.1930466523,   0.2919617793,   0.3849455276,   0.4500587765, ...
            0.4665175189,   0.4320357091,   0.3532962633,   0.2554173212, ...
            0.1587486509,   0.0840554147,   0.0334571586,   0.0089101239, ...
            0.0002288558 ];

D_min = [   1.0000000000,   0.0000000000,   0.2845431566,   0.0000000000, ...
            1.5935368208,   0.0000000000,  -0.2232042314,   0.0000000000, ...
            0.8502799491,   0.0000000000,  -0.3696928219,   0.0000000000, ...
            0.2517603730,   0.0000000000,  -0.0995325980,   0.0000000000, ...
            0.0324707031 ];
```

Figure 15.11 compares the pass-band and stop-band responses of the filter with transfer function denominator polynomial with coefficients in only z^{-2} andwith floating-point coefficients, 13-bit 4-signed-digit integer coefficients, 13-bit signed-digit integer coefficients allocated with the algorithm of *Ito et al.* and 13-bit signed-digit integer coefficients allocated with the algorithm of *Ito et al.* and SOCP-relaxation search. Table 15.5 compares the cost and the number of 13 bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the SOCP-relaxation search.

Figure 15.11: Comparison of the pass-band and stop-band amplitude responses for a R=2 Schur one-multiplier lattice lowpass filter with transfer function denominator polynomial with coefficients in only z^{-2} and with 13-bit 4-signed-digit integer coefficients 13-bit integer signed-digit coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SOCP-relaxation search.

	Cost	Signed-digits	Shift-and-adds
Exact	7.3087e-05		
13-bit 4-signed-digit	1.1583e-04	83	58
13-bit 4-signed-digit(Ito)	1.0239e-04	74	49
13-bit 4-signed-digit(SOCP-relax)	1.1498e-04	75	50

Table 15.5: Comparison of the cost and number of 13-bit shift-and-add operations required to implement the coefficient multiplications for a R=2 Schur one-multiplier lattice low-pass filter with transfer function denominator polynomial with coefficients in only z^{-2} and with 13-bit integer 4-signed-digit integer coefficients 13-bit integer signed-digit coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Ito et al.* and performing SOCP-relaxation search.

15.6 SOCP-relaxation search for the signed-digit coefficients of a lattice bandpass R=2 IIR filter

The Octave script *socp_relaxation_schurOneMlattice_bandpass_R2_10_nbits_test.m* performs successive SOCP relaxations to optimise the response of the SQP optimised band-pass $R = 2$ Schur one-multiplier lattice filter of Section 10.3.1 with 10-bit signed-digit coefficients. The filter specification is:

```
use_schurOneMlattice_allocsd_Ito=0
use_schurOneMlattice_allocsd_Lim=1
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
ftol=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=36 % Amplitude stop band(2) peak-to-peak ripple
Wasl=500000 % Amplitude lower stop band weight
Wasu=1e+06 % 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_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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
k_allocsd_digits = [  0,  5,  0,  5, ...
                      0,  5,  0,  3, ...
                      0,  3,  0,  4, ...
                      0,  3,  0,  4, ...
                      0,  3,  0,  2 ]';


c_allocsd_digits = [  3,  3,  4,  3, ...
                      3,  4,  4,  4, ...
                      3,  4,  4,  2, ...
                      2,  3,  3,  1, ...
                      2,  1,  1,  1, ...
                      1 ]';
```

The filter coefficients found by the SOCP-relaxation search are:

Figure 15.12: Comparison of the pass-band amplitude responses for an R=2 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 SOCP-relaxation search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.0164		
10-bit 3-signed-digit(Lim)	0.0520	77	46
10-bit 3-signed-digit(SOCP-relax)	0.0193	74	44

Table 15.6: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for an R=2 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 SOCP-relaxation search.

```
k_min = [        0,      339,        0,      260, ...
                 0,      180,        0,      216, ...
                 0,      152,        0,      128, ...
                 0,       76,        0,       51, ...
                 0,       17,        0,        7 ]'/512;


c_min = [       37,       -6,     -151,     -246, ...
               -82,       62,      201,      152, ...
                 8,      -42,      -41,       -6, ...
                -4,      -17,      -13,        4, ...
                12,        8,        1,        0, ...
                 2 ]'/512;
```

Figure 15.12 compares the pass-band responses of the filter with floating-point coefficients, 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 SOCP-relaxation search. Figure 15.13 shows the filter stop-band response and Figure 15.14 shows the filter pass-band group delay response. Table 15.6 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 *Ito et al.* with the SOCP-relaxation search.

Figure 15.13: Comparison of the stop-band amplitude responses for an R=2 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 SOCP-relaxation search.

Figure 15.14: Comparison of the pass-band group delay responses for an R=2 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 SOCP-relaxation search.

15.7 SOCP-relaxation search for the signed-digit coefficients of a lattice bandpass Hilbert R=2 IIR filter

The Octave script *socp_relaxation_schurOneMlattice_bandpass_hilbert_R2_13_nbits_test.m* performs successive SOCP relaxations to optimise the response of the SOCP optimised band-pass Hilbert $R = 2$ Schur one-multiplier lattice filter of Section 10.3.2 with 13-bit signed-digit coefficients. The filter specification is:

```
nbits=13 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
ftol=0.0001 % Tolerance on coef. update
ctol=1e-06 % Tolerance on constraints
n=1000% Frequency points across the band
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=0.26 % 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=32 % Amplitude stop band peak-to-peak ripple
Wasl=10 % Amplitude lower stop band weight
Wasu=10 % 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=0.2 % Delay pass band weight
fppl=0.1 % Pass band phase response lower edge
fppu=0.2 % Pass band phase response upper edge
pp=3.5 % Pass band initial phase response (rad./pi)
ppr=0.0048 % Pass band phase response ripple(rad./pi)
Wpp=2 % Pass band phase response weight
fdpl=0.1 % Pass band dAsqdw response lower edge
fdpu=0.2 % Pass band dAsqdw response upper edge
dp=0 % Pass band initial dAsqdw response (rad./pi)
dpr=1.2 % Pass band dAsqdw response ripple(rad./pi)
Wdp=0.001 % Pass band dAsqdw response weight
```

The filter coefficients are truncated to 13 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 Octave function *schurOneMlattice_slb* PCLS optimises the remaining active coefficient values. The script selects the closer of the upper or lower signed-digit values as the final choice for that coefficient.

The numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
k_allocsd_digits = [  0,  4,  0,  4, ...
                      0,  3,  0,  2, ...
                      0,  6,  0,  6, ...
                      0,  6,  0,  6, ...
                      0,  3,  0,  6 ]';

c_allocsd_digits = [  3,  3,  3,  2, ...
                      4,  3,  2,  1, ...
                      4,  2,  1,  1, ...
                      2,  3,  1,  2, ...
                      1,  1,  2,  5, ...
                      1 ]';
```

The 13-bit 3-signed-digit filter coefficients found by the SOCP-relaxation search are:

```
k_min = [        0,     2880,        0,     1744, ...
                 0,     1184,        0,     2044, ...
                 0,     1371,        0,     1745, ...
                 0,     1073,        0,     1016, ...
                 0,      400,        0,      249 ]'/4096;
```

Figure 15.15: Comparison of the pass-band amplitude responses for an R=2 Schur one-multiplier lattice band-pass Hilbert filter with 13-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.

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Exact	0.001115	-34.0		
13-bit 3-signed-digit	0.003104	-31.4	88	57
13-bit 3-signed-digit(Ito)	0.001434	-29.3	80	49
13-bit 3-signed-digit(min)	0.001660	-32.0	81	50

Table 15.7: Comparison of the cost and number of 13-bit shift-and-add operations required to implement the coefficient multiplications for an R=2 Schur one-multiplier lattice band-pass Hilbert filter with 13-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.

```
c_min = [     -168,      400,     1936,     1088, ...
              -720,    -1796,    -1022,     -128, ...
               808,      640,      128,       16, ...
               272,      208,        4,     -144, ...
               -32,      -16,      -14,      -52, ...
               -32 ]'/4096;
```

Figure 15.15 compares the pass-band responses of the band-pass Hilbert $R = 2$ filter with floating-point coefficients, 13-bit, 3-signed-digit coefficients allocated with the algorithm of *Ito et al.* and 13-bit 3-signed-digit coefficients allocated with the algorithm of *Ito et al.* found with SOCP-relaxation search. Figure 15.16 shows the band-pass Hilbert filter stop-band response. Figure 15.17 shows the band-pass Hilbert filter pass-band phase response. Figure 15.18 shows the band-pass Hilbert filter pass-band group delay response. Table 15.7 compares the cost and the number of 13-bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Ito et al.* with the SOCP-relaxation search.

Figure 15.16: Comparison of the stop-band amplitude responses for an R=2 Schur one-multiplier lattice band-pass Hilbert filter with 13-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.

Figure 15.17: Comparison of the pass-band phase responses for an R=2 Schur one-multiplier lattice band-pass Hilbert filter with 13-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.

Figure 15.18: Comparison of the pass-band group delay responses for an R=2 Schur one-multiplier lattice band-pass Hilbert filter with 13-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 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 10.3.3 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
tp=11.5 % Nominal pass band filter group delay
tpr=0.2 % Delay pass band peak-to-peak ripple
Wtp=2 % Delay pass band weight
```

The filter weights differ from those used in Section 14.9.

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 numbers of signed-digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
A1k_allocsd_digits = [  4,  3,  4,  3, ...
                        3,  4,  3,  2, ...
                        3,  3,  1 ]';


A2k_allocsd_digits = [  4,  4,  3,  4, ...
                        3,  2,  3,  3, ...
                        1,  3,  3,  3 ]';
```

The signed-digit lattice coefficients found by the heuristic of *Lim et al.* are:

```
A1k_sd = [     1576,     -176,     -548,     -130, ...
               -121,      501,     -296,       -9, ...
                336,     -328,      128 ]'/2048;


A2k_sd = [      792,     -561,      383,      336, ...
                -95,       80,     -416,      368, ...
                 16,     -368,      304,     -111 ]'/2048;
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are:

Figure 15.19: Pass-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	0.001845		
12-bit 3-signed-digit(Lim)	0.005873	66	43
12-bit 3-signed-digit(SOCP-relax)	0.002311	65	42

Table 15.8: 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.

```
A1k_min = [     1576,     -176,     -547,     -133, ...
               -114,      508,     -292,       -2, ...
                336,     -328,      128 ]'/2048;


A2k_min = [      792,     -558,      376,      332, ...
                 -95,       80,     -416,      368, ...
                   4,     -368,      312,     -129 ]'/2048;
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are implemented with 65 signed-digits and 42 shift-and-add operations.

Figures 15.19 and 15.20 shows the pass-band amplitude and group-delay responses of the filter with 12-bit 3-signed-digit coefficients allocated with the algorithm of *Lim et al.* and SOCP-relaxation search. Figure 15.21 shows the corresponding filter stop-band amplitude response. Table 15.8 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 15.20: Pass-band group delay 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 15.21: 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.

15.9 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice elliptic low-pass IIR filter

The Octave script *socp_relaxation_schurOneMPAlattice_elliptic_lowpass_16_nbits_test.m* performs successive SOCP relaxations to optimise the response of a parallel Schur one-multiplier all-pass lattice elliptic low-pass filter. with 16-bit 5-signed-digit integer coefficients. The initial filter design is;

```
[N0,D0]=ellip(11,0.02,84,2*0.15);
```

The initial transfer function polynomials are:

```
N0 = [   0.0009293269,  -0.0007898645,   0.0031439811,  -0.0008194769, ...
         0.0035413538,   0.0013979326,   0.0013979326,   0.0035413538, ...
        -0.0008194769,   0.0031439811,  -0.0007898645,   0.0009293269 ];

D0 = [   1.0000000000,  -6.4891487546,  21.1744373739, -44.7352568821, ...
        67.2938395531, -75.1847257433,  63.4160244544, -40.2860479200, ...
        18.8696899055,  -6.2066204107,   1.2918999342,  -0.1292850044 ];
```

The filter specification for SOCP relaxation design is:

```
nbits=16 % Coefficient word length
ndigits=5 % Signed digits per coef.
ftol=1e-08 % Tolerance on coefficient update vector
ctol=1e-10 % Tolerance on constraints
n=1000 % Frequency points across the band
rho=0.992188 % Constraint on allpass coefficients
fap=0.15 % Amplitude pass band edge
dBap=0.03 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Wat=1e-08 % Amplitude transition band weight
fas=0.171 % Amplitude stop band edge
dBas=81 % amplitude stop band peak-to-peak ripple
Was=10000000 % Amplitude stop band weight
```

The filter coefficients are truncated to 16 bits and 5 signed-digits. 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 16-bit 5-signed-digit lattice coefficients are:

```
A1k0_sd = [   -19680,    32377,   -25792,    28320, ...
              -23524,    11812 ]'/32768;

A2k0_sd = [   -22590,    30739,   -26768,    23872, ...
              -11752 ]'/32768;
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are:

```
A1k_min = [   -19680,    32376,   -25792,    28304, ...
              -23520,    11810 ]'/32768;

A2k_min = [   -22590,    30740,   -26768,    23872, ...
              -11752 ]'/32768;
```

Figure 15.22: Pass-band and stop-band amplitude responses for a parallel Schur one-multiplier all-pass lattice elliptic low-pass filter with 16-bit 5-signed-digit coefficients found by SOCP-relaxation search.

	Cost	Signed-digits	Shift-and-adds
Initial	7.52e-06		
16-bit 5-signed-digit	3.14e-05	55	44
16-bit 5-signed-digit(SOCP-relax)	7.54e-06	52	41

Table 15.9: 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 low-pass filter with 16-bit 5-signed-digit coefficients found by SOCP-relaxation search.

The corresponding transfer function polynomials are:

```
N_min = [   0.0008850098,  -0.0006059066,   0.0027168839,  -0.0001958075, ...
            0.0029517664,   0.0016390144,   0.0016390144,   0.0029517664, ...
           -0.0001958075,   0.0027168839,  -0.0006059066,   0.0008850098 ];


D_min = [   1.0000000000,  -6.4890992269,  21.1736211297, -44.7317244038, ...
           67.2854181823, -75.1717409466,  63.4021924570, -40.2757065929, ...
           18.8643470435,  -6.2048220583,   1.2915556402,  -0.1292593032 ];
```

Figure 15.22 shows the pass-band and stop-band amplitude responses of the low-pass elliptic filter with 16-bit 5-signed-digit coefficients. Table 15.9 compares the cost and the number of 16 bit shift-and-add operations required to implement the coefficient multiplications found by SOCP-relaxation search.

15.10 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 10.3.3 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=0.001 % 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=0.400000 % 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
A1k_allocsd_digits = [  4,  4,  3,  4, ...
                        4,  2,  1,  3, ...
                        3,  3 ]';


A2k_allocsd_digits = [  5,  4,  3,  4, ...
                        4,  2,  0,  2, ...
                        3,  2 ]';
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are:

```
A1k_min = [     -816,     1328,     1024,    -1068, ...
                1360,     -640,      -64,      776, ...
                -542,      316 ]'/2048;


A2k_min = [    -1530,     1464,     1029,    -1158, ...
                1376,     -508,        0,      768, ...
                -568,      320 ]'/2048;
```

Figure 15.23: Pass-band mplitude 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 57 signed-digits and 38 shift-and-add operations. The corresponding transfer function coefficient are:

```
N_min = [  -0.0009765625,   0.0092279105,  -0.0203790524,   0.0055942630, ...
            0.0249733618,  -0.0410551000,   0.0202685628,   0.0115383743, ...
           -0.0352765563,   0.0475626681,   0.0000000000,  -0.0475626681, ...
            0.0352765563,  -0.0115383743,  -0.0202685628,   0.0410551000, ...
           -0.0249733618,  -0.0055942630,   0.0203790524,  -0.0092279105, ...
            0.0009765625 ];


D_min = [   1.0000000000,  -3.1902575493,   4.7147238205,  -1.7133781624, ...
           -5.9801970885,  12.9090789584, -11.4678988258,   0.9748102969, ...
           11.0156003891, -15.0587024940,   9.2028122886,   0.6330733468, ...
           -6.8360134873,   6.8774025450,  -3.3144580892,   0.0097130576, ...
            1.3251342124,  -1.1367936110,   0.5479336481,  -0.1590289199, ...
            0.0241088867 ];
```

Figures 15.23 and 15.24 show the pass-band amplitude and group-delay responses of the filter with 12-bit 3-signed-digit coefficients allocated with the algorithm of *Lim et al.* and SOCP-relaxation search. Figure 15.25 shows the corresponding filter stop-band amplitude response. Figure 15.26 shows the filter pole-zero plot. Table 15.10 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 15.24: Pass-band group-delay 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.

Figure 15.25: 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.

Figure 15.26: Pole-zero plot of 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.

	Cost	Signed-digits	Shift-and-adds
Exact	0.029927		
12-bit 3-signed-digit(Lim)	0.151154	57	38
12-bit 3-signed-digit(SOCP-relax)	0.012220	57	38

Table 15.10: 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.

15.11 SOCP-relaxation search for the signed-digit coefficients of a parallel all-pass lattice band-pass Hilbert IIR filter

The Octave script *socp_relaxation_schurOneMPAlattice_bandpass_hilbert_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of the parallel Schur one-multiplier all-pass lattice band-pass Hilbert filter of Section 10.3.3 with 12-bit integer coefficients. The filter specification is:

```
nbits=12 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
ftol=0.0001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=1000% Frequency points across the band
NA1k=10 % All-pass filter a order
NA2k=10 % All-pass filter b order
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=0.2 % Amplitude pass band peak-to-peak ripple
Wap=20 % Amplitude pass band weight
fasl=0.05 % Amplitude stop band lower edge
fasu=0.25 % Amplitude stop band upper edge
dBas=37 % Amplitude stop band peak-to-peak ripple
Wasl=50000 % Amplitude lower stop band weight
Wasu=5000 % Amplitude upper stop band weight
fppl=0.11 % Pass band phase response lower edge
fppu=0.19 % Pass band phase response upper edge
pp=3.5 % Pass band initial phase response (rad./pi)
ppr=0.012 % Pass band phase response ripple(rad./pi)
Wpp=20 % Pass band phase response weight
ftpl=0.11 % Pass band delay lower edge
ftpu=0.19 % 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
fdpl=0.1 % Pass band dAsqdw response lower edge
fdpu=0.2 % Pass band dAsqdw response upper edge
dp=0 % Pass band initial dAsqdw response (rad./pi)
dpr=0.6 % Pass band dAsqdw response ripple(rad./pi)
Wdp=0.001 % Pass band dAsqdw 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 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
A1k_allocsd_digits = [  3,  4,  3,  3, ...
                        4,  3,  3,  3, ...
                        2,  2 ]';

A2k_allocsd_digits = [  4,  3,  3,  1, ...
                        4,  3,  2,  3, ...
                        4,  3 ]';
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are:

```
A1k_min = [     -928,     1696,     -472,      104, ...
                1392,     -672,      216,     1072, ...
                -768,      544 ]'/2048;
```

Figure 15.27: Pass-band and stop-band amplitude responses 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 SOCP-relaxation search.

```
A2k_min = [    -1656,     1800,     -608,       32, ...
                1424,     -608,      224,     1072, ...
                -740,      552 ]'/2048;
```

The signed-digit lattice coefficients found by the SOCP-relaxation search are implemented with 60 signed-digits and 40 shift-and-add operations. The corresponding transfer function coefficient are:

```
N_min = [  -0.0019531250,   0.0005672872,   0.0088529673,  -0.0241459676, ...
            0.0153506339,   0.0200372221,  -0.0438538002,   0.0183324544, ...
            0.0236509505,  -0.0216184911,   0.0000000000,   0.0216184911, ...
           -0.0236509505,  -0.0183324544,   0.0438538002,  -0.0200372221, ...
           -0.0153506339,   0.0241459676,  -0.0088529673,  -0.0005672872, ...
            0.0019531250 ];


D_min = [   1.0000000000,  -3.7368698120,   6.8021101742,  -5.2620062118, ...
           -3.6430193502,  15.3312740119, -18.4619809906,   7.0949470054, ...
           11.8616989500, -22.8375277384,  17.5256949600,  -1.6953652784, ...
          -11.0292242829,  13.0121256464,  -6.7535890033,  -0.1968616982, ...
            3.3582386885,  -2.9751065241,   1.4941999753,  -0.4504863853, ...
            0.0715942383 ];
```

Figure 15.27 shows the pass-band and stop-band amplitude responses and Figures 15.28 and 15.29 show the pass-band phase and group-delay responses of the band-pass Hilbert filter with 12-bit 3-signed-digit coefficients allocated with the algorithm of *Ito et al.* and SOCP-relaxation search. Figure 15.30 shows the filter pole-zero plot. Table 15.11 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 SOCP-relaxation search.

Figure 15.28: Pass-band phase responses 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 SOCP-relaxation search.

Figure 15.29: Pass-band group-delay responses 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 SOCP-relaxation search.

Figure 15.30: Pole-zero plot of 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 SOCP-relaxation search.

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Exact	0.001521	-39.9		
12-bit 3-signed-digit	0.104527	-16.7	60	40
12-bit 3-signed-digit(Ito)	0.004232	-37.8	60	40
12-bit 3-signed-digit(min)	0.003223	-37.6	60	40

Table 15.11: 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 SOCP-relaxation search.

15.12 SOCP-relaxation search for the signed-digit coefficients of a pipelined one-multiplier lattice low-pass IIR filter

The Octave script *socp_relaxation_schurOneMlatticePipelined_lowpass_15_nbits_test.m* performs successive SOCP relaxations to optimise the response of an implementation of the low-pass filter shown in Section 10.3.2 as a pipelined Schur one-multiplier lattice filter with 15-bit signed-digit coefficients. The filter specification is:

```
socp_relaxation_schurOneMlatticePipelined_lowpass_allocsd_Lim=1
socp_relaxation_schurOneMlatticePipelined_lowpass_allocsd_Ito=0
nbits=15 % Bits-per-coefficient
ndigits=4 % Average signed-digits-per-coefficient
ftol=0.001 % Tolerance on coef. update
ctol=1e-06 % Tolerance on constraints
n=1000 % Frequency points across the band
fap=0.15 % Amplitude pass band upper edge
dBap=0.11 % Amplitude pass band peak-to-peak ripple(dB)
Wap=1 % Amplitude pass band weight
ftp=0.25 % Group-delay pass band upper edge
tp=9 % Nominal pass band group-delay(samples)
tpr=0.04 % Group-delay pass band peak-to-peak ripple(samples)
Wtp=1 % Group-delay pass band weight
fas=0.35 % Amplitude stop band lower edge
dBas=46 % Amplitude stop band peak-to-peak ripple(dB)
Was=100 % Amplitude stop band weight
```

The filter coefficients are truncated to 15 bits allocated with an average of 4 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 *schurOneMlatticePipelined_socp_mmse*. The results of this MMSE optimisation are then passed to *schurOneMlatticePipelined_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 of the pipelined Schur one-multiplier lattice filter with an average of 4 signed-digits allocated by the algorithm of *Lim et al.* are:

```
k0_sd = [   -11151,    10740,    -8758,     5768, ...
             -2788,      762,        0,        0, ...
                 0 ]'/16384;


c0_sd = [     6917,    15098,     1392,    -2104, ...
              -767,      244,      336,       32, ...
              -128,      -68 ]'/16384;


kk0_sd = [    -7308,    -5744,    -3085,     -982, ...
               -130,        0,        0,        0 ]'/16384;


ck0_sd = [     9896,        0,     -736,        0, ...
                  0,        0,        0,        0 ]'/16384;
```

The signed-digit filter coefficients of the pipelined Schur one-multiplier lattice filter found by the SOCP-relaxation search are:

```
k_min = [   -11150,    10740,    -8758,     5776, ...
             -2788,      762,        0,        0, ...
                 0 ]'/16384;


c_min = [     6917,    15098,     1392,    -2105, ...
              -767,      244,      336,       32, ...
              -128,      -66 ]'/16384;
```

Figure 15.31: Comparison of the amplitude and group delay responses of a pipelined one-multiplier Schur lattice lowpass filter with 15-bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search.

	Cost	Signed-digits	Shift-and-adds
Exact	2.5484e-04		
15-bit 4-signed-digit	2.6228e-04	84	60
15-bit 4-signed-digit(Lim)	1.6223e-04	90	67
15-bit 4-signed-digit(SOCP-relax)	1.9840e-04	91	68

Table 15.12: Comparison of the cost and number of 15-bit shift-and-add operations required to implement the coefficient multiplications for a low-pass pipelined one-multiplier Schur lattice filter with 15-bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search.

```
kk_min = [    -7310,    -5744,    -3085,     -980, ...
               -129,        0,        0,        0 ]'/16384;


ck_min = [     9900,        0,     -736,        0, ...
                  0,        0,        0,        0 ]'/16384;
```

Figure 15.31 compares the amplitude and group delay responses of a low-pass pipelined one-multiplier Schur lattice filter with floating-point coefficients, 15-bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and 15-bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and found by SOCP-relaxation search. Table 15.12 compares the cost, the number of signed-digits and the number of 15-bit shift-and-add operations required to implement the coefficient multiplications found by the signed-digit allocation heuristic of *Lim et al.* and by SOCP-relaxation search.

15.13 SOCP-relaxation search for the signed-digit coefficients of a lattice Hilbert R=2 IIR filter

The Octave script *socp_relaxation_schurOneMlattice_hilbert_R2_10_nbits_test.m* performs successive SOCP relaxations to optimise the response of the Hilbert R=2 Schur one-multiplier lattice filter of Section 10.3.1 with 10-bit integer coefficients. The filter specification is:

```
ftol=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
ftt=0.08 % Transition band width [0,ftt]
dBar=0.2 % Amplitude pass band peak-to-peak ripple
Wat=0.1 % Amplitude transition band weight
Wap=1 % Amplitude pass band weight
tp=5.5 % Nominal pass band filter group delay (samples)
ppr=0.01 % Phase pass band peak-to-peak ripple (rad./pi)
Wpp=2 % 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 15.3, 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
k_allocsd_digits = [  0,  5,  0,  3, ...
                      0,  0,  0,  0, ...
                      0,  0,  0,  0 ]';


c_allocsd_digits = [  3,  3,  3,  3, ...
                      3,  3,  4,  4, ...
                      3,  3,  2,  2, ...
                      1 ]';
```

The signed-digit filter coefficients found by the heuristic of *Lim et al.* are:

```
k0_sd = [        0,     -469,        0,      225, ...
                 0,        0,        0,        0, ...
                 0,        0,        0,        0 ]'/512;


c0_sd = [      -15,      -18,     -100,     -116, ...
               -92,     -135,     -351,      300, ...
                80,       35,       24,       10, ...
                 8 ]'/512;
```

The signed-digit filter coefficients found by the SOCP-relaxation search are:

```
k_min = [        0,     -469,        0,      225, ...
                 0,        0,        0,        0, ...
                 0,        0,        0,        0 ]'/512;
```

Figure 15.32: Comparison of the amplitude responses for a R=2 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.

	Cost	Signed-digits	Shift-and-adds
Exact	0.001716		
10-bit 3-signed-digit(Lim)	0.003722	42	27
10-bit 3-signed-digit(SOCP-relax)	0.001594	41	26

Table 15.13: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for a R=2 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.

```
c_min = [       -15,       -18,      -100,      -118, ...
                -92,      -135,      -353,       300, ...
                 80,        36,        20,         9, ...
                  4 ]'/512;
```

Figures 15.32, 15.33 and 15.34 compare the amplitude, phase and delay responses of the filter with floating-point coefficients, 10-bit signed-digit coefficients allocated by the heuristic of *Lim et al.* and 10-bit signed-digit coefficients allocated by the heuristic of *Lim et al.* with SOCP-relaxation search. The phase responses shown are adjusted for the nominal filter delay.

Table 15.13 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 15.33: Comparison of the phase responses for a R=2 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.

Figure 15.34: Comparison of the delay responses for a R=2 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.

15.14 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 8.3.3 as the series combination of $1 - z^{-1}$ and a Schur one-multiplier lattice correction filter with 12-bit integer signed-digit coefficients. The filter specification is:

```
socp_relaxation_schurOneMlattice_lowpass_differentiator_allocsd_Lim=1
socp_relaxation_schurOneMlattice_lowpass_differentiator_allocsd_Ito=0
nbits=12 % Bits-per-coefficient
ndigits=3 % Average signed-digits-per-coefficient
ftol=0.0001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=400 % Frequency points across the band
fap=0.3 % Amplitude pass band upper edge
Arp=0.006 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.02 % Amplitude transition band peak-to-peak ripple
Wat=0.0001 % Amplitude transition band weight
fas=0.4 % Amplitude stop band lower edge
Ars=0.02 % Amplitude stop band peak-to-peak ripple
Was=1 % Amplitude stop band weight
tp=10 % Pass band group delay
tpr=0.08 % Pass band group delay peak-to-peak ripple
Wtp=1 % Pass band group delay weight
pp=1.5 % Phase pass band nominal phase(rad./pi))
ppr=0.004 % Phase pass band peak-to-peak ripple(rad./pi))
Wpp=1 % Phase pass band weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits allocated by the heuristic of *Lim 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
k_allocsd_digits = [ 4, 4, 4, 4, ...
                     4, 4, 3, 3, ...
                     3, 2, 2 ]';
```

```
c_allocsd_digits = [ 4, 4, 4, 3, ...
                     3, 3, 3, 1, ...
                     2, 2, 2, 1 ]';
```

The corresponding signed-digit coefficients of the Schur one-multiplier lattice correction filter are:

```
k0_sd = [      664,     1184,     -816,      720, ...
              -612,      480,     -336,      200, ...
              -100,       40,      -12 ]'/2048;
```

```
c0_sd = [      330,     -475,    -1352,       67, ...
               140,      -98,       33,        4, ...
               -20,       15,       -5,       -1 ]'/2048;
```

The signed-digit filter coefficients of the Schur one-multiplier lattice correction filter found by the SOCP-relaxation search are:

Figure 15.35: 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 *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	0.0001921		
12-bit 3-signed-digit(Lim)	0.000422	65	42
12-bit 3-signed-digit(SOCP-relax)	0.000539	64	41

Table 15.14: 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 correction 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.

```
k_min = [      664,     1184,     -816,      720, ...
              -612,      480,     -336,      200, ...
              -100,       40,      -12 ]'/2048;


c_min = [      330,     -475,    -1352,       67, ...
               142,      -98,       32,        4, ...
               -20,       14,       -6,       -1 ]'/2048;
```

Figure 15.35 compares the amplitude error, phase and group delay responses of the low-pass differentiator filter with floating-point coefficients, 12-bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and 12-bit signed-digit coefficients allocated with the algorithm of *Lim et al.* and found by SOCP-relaxation search. The phase responses shown are adjusted for the nominal filter delay. Table 15.14 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.* and by SOCP-relaxation search.

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

The Octave script *socp_relaxation_schurOneMlatticePipelined_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 10.3.2 as the series combination of $1 - z^{-1}$ and a pipelined Schur one-multiplier lattice correction filter with 12-bit integer signed-digit coefficients. The filter specification is:

```
nbits=12 % Bits-per-coefficient
ndigits=3 % Average signed-digits-per-coefficient
ftol=0.001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=400 % Frequency points across the band
fap=0.3 % Amplitude pass band upper edge
Arp=0.01 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
fas=0.4 % Amplitude stop band lower edge
Ars=0.006 % Amplitude stop band peak ripple
Was=1 % Amplitude stop band weight
tp=10 % Pass band group delay
tpr=0.08 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
ppr=0.008 % Phase pass band peak-to-peak ripple(rad./pi))
Wpp=1 % Phase pass band weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits allocated 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 *schurOneMlatticePipelined_socp_mmse*. The results of this MMSE optimisation are then passed to *schurOneMlatticePipelined_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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
k_allocsd_digits = [ 3, 4, 4, 4, ...
                     3, 2, 2, 3, ...
                     3, 6, 6 ]';


c_allocsd_digits = [ 2, 2, 4, 3, ...
                     3, 3, 1, 2, ...
                     2, 2, 6, 6 ]';


kk_allocsd_digits = [ 3, 3, 2, 3, ...
                      3, 3, 4, 1, ...
                      6, 1 ]';


ck_allocsd_digits = [ 2, 0, 2, 0, ...
                      1, 0, 2, 0, ...
                      2, 0 ]';
```

The signed-digit filter coefficients of the Schur one-multiplier lattice correction filter found by the SOCP-relaxation search are:

```
k_min = [      336,     1376,    -1240,      676, ...
              -928,      960,     -544,      492, ...
               -92,     -276,      -11 ]'/2048;


c_min = [      384,     -496,    -2400,     -104, ...
                92,     -168,       16,      -20, ...
               -28,       60,       -4,       -1 ]'/2048;
```

Figure 15.36: Comparison of the amplitude error, phase and group delay responses for a pipelined 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	6.2589e-05		
12-bit 3-signed-digit(Ito)	2.5781e-04	96	58
12-bit 3-signed-digit(SOCP-relax)	1.7823e-04	97	59

Table 15.15: Comparison of the cost and number of 12-bit shift-and-add operations required to implement the coefficient multiplications for a pipelined Schur one-multiplier lattice correction 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.

```
kk_min = [      864,     -736,     -160,     -156, ...
                156,      176,      340,       32, ...
                299,        2 ]'/2048;


ck_min = [      192,        0,     -120,        0, ...
               -128,        0,      -36,        0, ...
                 48,        0 ]'/2048;
```

Figure 15.36 compares the amplitude error, phase and group delay responses of the pipelined low-pass differentiator filter with floating-point 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 found by SOCP-relaxation search. The phase responses shown are adjusted for the nominal filter delay. Table 15.15 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.* and by SOCP-relaxation search.

15.16 SOCP-relaxation search for the signed-digit coefficients of a one-multiplier lattice R=2 low-pass differentiator IIR filter

The Octave script *socp_relaxation_schurOneMlattice_lowpass_differentiator_R2_12_nbits_test.m* performs successive SOCP relaxations to optimise the response of an implementation of the low-pass differentiator $R = 2$ filter shown in Section 10.3.2 as the series combination of $1 - z^{-1}$ and a Schur one-multiplier lattice correction filter having denominator polynomial coefficients only in z^{-2} with 12-bit coefficients each having an average of 3 signed-digits allocated by the heuristic of *Lim et al.* and found by SOCP-relaxation search. The filter specification is:

```
socp_relaxation_schurOneMlattice_lowpass_differentiator_R2_allocsd_Lim=1
socp_relaxation_schurOneMlattice_lowpass_differentiator_R2_allocsd_Ito=0
nbits=12 % Bits-per-coefficient
ndigits=3 % Average signed-digits-per-coefficient
ftol=0.0001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=1000 % Frequency points across the band
fap=0.2 % Amplitude pass band upper edge
Arp=0.001 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.01 % Amplitude transition band peak-to-peak ripple
Wat=0.0001 % Amplitude transition band weight
fas=0.4 % Amplitude stop band lower edge
Ars=0.003 % Amplitude stop band peak-to-peak ripple
Was=0.1 % Amplitude stop band weight
fpp=0.2 % Phase pass band upper edge
pp=1.5 % Phase pass band nominal phase(rad./pi))
ppr=0.0004 % Phase pass band peak-to-peak ripple(rad./pi))
Wpp=1 % Phase pass band weight
ftp=0.2 % Amplitude pass band upper edge
tp=9 % Pass band group delay(samples)
tpr=0.006 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
fdp=0.2 % Correction filter dAsqdw pass band upper edge
cpr=0.03 % Correction filter dAsqdw pass band peak-to-peak ripple)
Wdp=0.005 % Correction filter dAsqdw pass band weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits allocated 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_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 numbers of signed-digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
k_allocsd_digits = [ 0, 3, 0, 3, ...
                     0, 2, 0, 3, ...
                     0, 3 ]';


c_allocsd_digits = [ 3, 4, 4, 3, ...
                     4, 2, 3, 3, ...
                     2, 3, 3 ]';
```

The signed-digit filter coefficients of the Schur one-multiplier lattice correction filter found by the SOCP-relaxation search are:

```
k_min = [        0,      416,        0,      -57, ...
                 0,       18,        0,       -7, ...
                 0,        2 ]'/2048;


c_min = [      -44,     -449,     -588,      -63, ...
               148,      -34,      -48,       35, ...
                 2,      -12,        4 ]'/2048;
```

Figure 15.37: Pass-band amplitude error response of a Schur one-multiplier lattice lowpass differentiator filter, having denominator polynomial coefficients only in z^{-2}, 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 corresponding coefficients of the filter transfer function polynomials are:

```
N_min = [   0.0019531250,  -0.0058536530,   0.0013607526,   0.0158604764, ...
           -0.0231929742,  -0.0128753752,   0.0667041521,  -0.0332393313, ...
           -0.2607570166,  -0.2584290613,  -0.0829409967 ];


D_min = [   1.0000000000,   0.0000000000,   0.1971936226,   0.0000000000, ...
           -0.0259993193,   0.0000000000,   0.0080895491,   0.0000000000, ...
           -0.0032253936,   0.0000000000,   0.0009765625 ];
```

Figure 15.37, Figure 15.38, Figure 15.39 and Figure 15.40 show the pass-band amplitude error (compared to a desired response of $\frac{\omega}{2}$), stop-band amplitude, pass-band phase (adjusted for the nominal filter delay) and pass-band group delay responses. Figure 15.41 shows the pole-zero plot of the filter found by SOCP-relaxation search. Table 15.16 compares the cost, the number of signed-digits and the number of 12-bit shift-and-add operations required to implement the coefficient multiplications.

	Cost	Signed-digits	Shift-and-adds
Exact	4.0719e-06		
12-bit 3-signed-digit	1.6830e-05	41	25
12-bit 3-signed-digit(Lim)	1.0954e-05	41	25
12-bit 3-signed-digit(SOCP-relax)	5.6397e-06	37	21

Table 15.16: 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 low-pass differentiator correction filter having denominator polynomial coefficients only in z^{-2}, 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 15.38: Stop-band amplitude response of a Schur one-multiplier lattice lowpass differentiator filter, having denominator polynomial coefficients only in z^{-2}, 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 15.39: Pass-band phase response of a Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial coefficients only in z^{-2}, with 12 bit coefficients found by allocating an average of 3-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SOCP-relaxation search. The phase response shown is adjusted for the nominal delay.

Figure 15.40: Pass-band group delay response of a Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial coefficients only in z^{-2}, with 12 bit 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 15.41: Pole-zero plot of the Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial terms only in z^{-2} with 12 bit, 3 signed-digit coefficients, found by performing SOCP-relaxation search.

15.17 SOCP relaxation search for the 12-bit, 3-signed-digit coefficients of an 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, 3 signed-digit coefficient. The filter specification is:

```
socp_relaxation_schurOneMAPlattice_frm_12_nbits_test_allocsd_Lim=0
socp_relaxation_schurOneMAPlattice_frm_12_nbits_test_allocsd_Ito=0
n=1000 % Frequency points across the band
nbits=12 % Bits-per-coefficient
ndigits=3 % Average signed-digits-per-coefficient
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 = [      -35,     1184,       29,     -292, ...
               -19,      120,       13,      -52, ...
               -13,       21 ]'/2048;


epsilon_min = [  1,  1, -1,  1, ...
                 1, -1, -1,  1, ...
                 1, -1 ];


u_min = [     1184,      608,     -118,     -176, ...
               104,       68,      -88,      -20, ...
                81,      -40,      -25,       16, ...
                21,      -19,       -8,       15, ...
                 2,      -20,       18,        0, ...
                -2 ]'/2048;


v_min = [    -1344,     -560,      270,       10, ...
              -134,       97,       10,      -73, ...
                56,        1,      -37,       28, ...
                 5,      -22,       16,        4, ...
               -16,       12,        1,       -6, ...
                 3 ]'/2048;
```

Figures 15.42, 15.43 and 15.44 compare the amplitude, phase and delay responses of the FRM low-pass filter with floating-point coefficients and with 12-bit, 3-signed-digit coefficients found by SOCP-relaxation search. The phase response shown is adjusted for the nominal delay.

Table 15.17 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 15.42: Comparison of the amplitude responses of an FRM low-pass filter with floating-point coefficients and with 12 bit, 3-signed-digit coefficients found by SOCP-relaxation search.

Figure 15.43: Comparison of the pass-band phase responses of an FRM low-pass filter with floating-point coefficients and with 12 bit, 3-signed-digit coefficients found by SOCP-relaxation search. The phase response shown is adjusted for the nominal delay.

Figure 15.44: Comparison of the pass-band delay responses of an FRM low-pass filter with floating-point coefficients and with 12 bit, 3-signed-digit coefficients found by SOCP-relaxation search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.001693		
12-bit 3-signed-digit	0.004768	120	68
12-bit 3-signed-digit(SOCP-relax)	0.001608	120	69

Table 15.17: 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, 3-signed-digit coefficients found by SOCP-relaxation search.

15.18 SOCP relaxation search for the 16-bit, 3-signed-digit coefficients of an FRM low-pass filter

The Octave script *socp_relaxation_schurOneMAPlattice_frm_16_nbits_test.m* uses socp-relaxation search to optimise the response of the FRM low-pass filter of Section 10.4.6 with 16-bit coefficients each having an average of 3 signed-digits allocated by the method of *Ito et al.*. The filter specification is:

```
n=1000 % Frequency points across the band
tol=0.0001 % Tolerance on coefficient update vector
ctol=1e-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=0.2 % 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/126.25 % Pass band delay peak-to-peak ripple
Wtp=1 % Pass band magnitude-squared weight
fpp=0.3 % Phase pass band edge
ppr=0.02*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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
k_allocsd_digits = [  3,  5,  2,  3, ...
                      2,  2,  1,  3, ...
                      2,  6 ]';


u_allocsd_digits = [  5,  6,  4,  4, ...
                      3,  2,  5,  1, ...
                      3,  2,  6,  1, ...
                      2,  3,  2,  2, ...
                      1,  2,  2,  2, ...
                      1 ]';


v_allocsd_digits = [  6,  4,  3,  2, ...
                      3,  4,  2,  3, ...
                      6,  2,  3,  6, ...
                      3,  4,  2,  1, ...
                      2,  2,  6,  2, ...
                      2 ]';
```

Section 14.16. The filter coefficients found by the socp-relaxation search are:

```
k_min = [     -572,    19136,      480,    -4672, ...
              -288,     1920,      256,     -832, ...
              -224,      351 ]'/32768;


epsilon_min = [  1,  1, -1,  1, ...
                 1, -1, -1,  1, ...
                 1, -1 ];
```

Figure 15.45: Comparison of the amplitude responses of an FRM low-pass filter with floating-point coefficients, with 16 bit coefficients having an average of 3-signed-digits allocated by the method of *Ito et al.* and with the coefficients found by SOCP-relaxation search.

```
u_min = [    18896,     9861,    -1880,    -2752, ...
              1664,     1088,    -1402,     -256, ...
              1264,     -576,     -447,      256, ...
               320,     -304,     -136,      240, ...
                16,     -320,      264,       10, ...
               -32 ]'/32768;


v_min = [   -21872,    -8956,     4320,      144, ...
             -2160,     1576,      160,    -1184, ...
               920,      -80,     -592,      445, ...
                96,     -374,      260,       64, ...
              -240,      192,      -61,     -112, ...
                72 ]'/32768;
```

Figures 15.45, 15.46, Figures 15.47, compare the amplitude, phase and delay responses of the FRM low-pass filter with floating-point coefficients, with 16-bit coefficients having an average of 3-signed-digits allocated by the method of *Ito et al.* and with the coefficients found by SOCP-relaxation search. The phase response shown is adjusted for the nominal delay.

Table 15.18 compares the cost and the number of 16 bit shift-and-add operations required to implement the coefficient multiplications found by SOCP-relaxation search.

Figure 15.46: Comparison of the pass-band phase responses of an FRM low-pass filter with floating-point coefficients, with 16 bit coefficients having an average of 3-signed-digits allocated by the method of *Ito et al.* and with the coefficients found by SOCP-relaxation search. The phase response shown is adjusted for the nominal delay.

Figure 15.47: Comparison of the pass-band delay responses of an FRM low-pass filter with floating-point coefficients, with 16 bit coefficients having an average of 3-signed-digits allocated by the method of *Ito et al.* and with the coefficients found by SOCP-relaxation search.

	Cost	Signed-digits	Shift-and-adds
Exact	0.001693		
16-bit 3-signed-digit(Ito)	0.002048	147	95
16-bit 3-signed-digit(SOCP-relax)	0.001230	143	91

Table 15.18: Comparison of the cost and number of 16-bit shift-and-add operations required to implement the coefficient multiplications for an FRM low-pass filter with 16-bit coefficients with an average of 3-signed-digits allocated by the method of *Ito et al.* and with the coefficients found by SOCP-relaxation search.

15.19 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 15.3, 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 numbers of signed-digits allocated to the coefficients of the all-pass lattice filter, FIR masking filter and FIR complementary masking filter by the heuristic of *Lim et al.* are:

```
k_allocsd_digits = [  2,  2,  2,  1, ...
                      1 ]';


u_allocsd_digits = [  1,  1,  2,  2, ...
                      1,  2,  3,  3, ...
                      4 ]';


v_allocsd_digits = [  1,  2,  2,  1, ...
                      2,  3,  3,  3 ]';
```

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;
```

	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 15.19: 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.

```
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;


v_min = [       16,       10,       14,        4, ...
               -16,      -64,     -164,     -641 ]'/2048;
```

Table 15.19 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 15.48, 15.50 and 15.49 compare the amplitude, delay and phase responses of the filter with floating-point coefficients, 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 π radians.

Figure 15.48: 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.

Figure 15.49: 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 π radians. The FRM model filter is implemented as a Schur one-multiplier all-pass lattice filter.

Figure 15.50: 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.

15.20 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 15.19. The filter specification is:

```
M=40 % Number of distinct coefficients
nbits=12 % Coefficient bits
ndigits=2 % Nominal average coefficient signed-digits
ftol=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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
hM_allocsd_digits = [  4,  4,  1,  1, ...
                       1,  1,  1,  4, ...
                       1,  1,  1,  1, ...
                       1,  1,  1,  1, ...
                       2,  2,  1,  1, ...
                       2,  2,  2,  2, ...
                       2,  1,  2,  2, ...
                       2,  3,  4,  2, ...
                       4,  2,  2,  3, ...
                       3,  2,  3,  4 ]';
```

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, ...
```

Figure 15.51: 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 15.20: 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.

```
 -18,      -20,      -24,      -28, ...
 -28,      -32,      -36,      -40, ...
 -48,      -52,      -60,      -68, ...
 -79,      -96,     -112,     -142, ...
-184,     -258,     -432,    -1304 ]'/2048;
```

Figure 15.51 compares the amplitude responses of the filter with floating-point coefficients, 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 15.20 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.

15.21 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 15.52 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 floating-point response and below *dBasu* in the "stop-band".

The 16-bit 3-signed-digit direct form coefficients are:

Figure 15.52: 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 floating-point 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, -0.123094764, -0.155489023, -0.179478987, ...
           -0.188391851, -0.179478987, -0.155489023, -0.123094764, ...
           -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:

	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 15.21: 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.

```
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 15.21 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).

Figure 15.53 compares the overall amplitude response of the filter for floating-point direct-form 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 15.54 compares the amplitude responses in the "pass-band" (up to *dBas*). Figure 15.55 compares the group delay responses in the "pass-band" (up to *dBas*).

Figure 15.53: Comparison of the amplitude response of the FIR Gaussian filter for floating-point 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 15.54: 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 15.55: 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

16.1 The *maximum-cut* problem of graph theory

Ito et al. [211] point out that optimisation of a filter response with integer coefficients is equivalent to the *weighted-maximum-cut* problem of graph theory. *Goemans* and *Williamson* [151] describe a randomised approximation algorithm for the weighted-maximum-cut problem, shown as Algorithm 16.1.

Algorithm 16.1 The maximum-cut algorithm of *Goemans* and *Williamson* [151]

Given a uni-directed graph, G, with a set of N vertexes, V, a set of edges, E, and non-negative weights $w_{ij} = w_{ji}$ on the edges $(ij) \in E$, the *maximum cut* problem is that of finding the set of vertexes, S, that maximises the weight of the edges in the *cut* $\left(S, \bar{S}\right)$; that is, the weight of the edges with one endpoint in S and the other in $\bar{S}$. The weight of the maximum cut, W_{MC}, is given by the integer quadratic program:

$$\begin{aligned} &\textbf{maximise} && \frac{1}{2}\sum_{i<j} w_{ij}\left(1 - y_i y_j\right) \\ &\textbf{subject to} && y_i \in \{1, -1\} \quad \forall i \in V \end{aligned}$$

where $S = \{i | y_i = 1\}$ and $\bar{S} = \{j | y_j = -1\}$. This optimisation restricts y_i to be a 1-dimensional vector with unit Euclidean norm. The optimisation can be relaxed to a quadratic program by allowing the y_i to be replaced by N-dimensional vectors $\boldsymbol{v_i}$ lying on the N-dimensional unit sphere, $\mathcal{S}_N$:

$$\begin{aligned} &\textbf{maximise} && \frac{1}{2}\sum_{i<j} w_{ij}\left(1 - \boldsymbol{v}_i^\top \boldsymbol{v}_j\right) \\ &\textbf{subject to} && \boldsymbol{v}_i \in \mathcal{S}_N \quad \forall i \in V \end{aligned}$$

Given a set of solution vectors, $\boldsymbol{v}_i$, and a vector, $\boldsymbol{r}$, uniformly distributed on $\mathcal{S}_N$, set $S = \left\{i | \boldsymbol{v}_i^\top \boldsymbol{r} \geq 0\right\}$. The cut, W, defined by the random hyperplane with normal $\boldsymbol{r}$, has the expected weight $E[W] \geq \alpha W_{MC}$ where:

$$\alpha = \min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta} > 0.87856$$

16.2 Semi-definite programming search for integer and signed-digit filter coefficients

This section follows the method introduced by *Lu* [256] and *Ito et al.* [211] 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 floating-point 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

$$\hat{\boldsymbol{x}} = \frac{(\boldsymbol{u}+\boldsymbol{l})}{2}$$
$$\boldsymbol{\delta} = \frac{(\boldsymbol{u}-\boldsymbol{l})}{2}$$

If $y_i = 1$, $\hat{x}_i + y_i\delta_i = u_i$, and if $y_i = -1$, $\hat{x}_i + y_i\delta_i = l_i$. Define the filter coefficient vector as $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{y} \odot \boldsymbol{\delta}$, where $\odot$ 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, $\mathcal{E}^2$, of the filter amplitude response is approximately:

$$\mathcal{E}^2 \approx \frac{1}{\pi}\int_0^\pi W(\omega)\left[\hat{A}(\omega) + A_\delta(\omega) - A_d(\omega)\right]^2 d\omega$$

The weighted mean-squared-error of the amplitude response of an IIR filter with signed-digit coefficients, $\hat{\boldsymbol{x}} + (\boldsymbol{y} \odot \boldsymbol{\delta})$, is approximately:

$$\mathcal{E}^2 \approx \mathcal{E}^2(\hat{\boldsymbol{x}}) + \nabla_x\mathcal{E}^2(\hat{\boldsymbol{x}})^\top(\boldsymbol{y}\odot\boldsymbol{\delta}) + \frac{1}{2}(\boldsymbol{y}\odot\boldsymbol{\delta})^\top\nabla_x^2\mathcal{E}^2(\hat{\boldsymbol{x}})(\boldsymbol{y}\odot\boldsymbol{\delta}) \tag{16.1}$$

If $\boldsymbol{Y} = \boldsymbol{y}\boldsymbol{y}^\top$ and $\boldsymbol{\Delta} = \boldsymbol{\delta}\boldsymbol{\delta}^\top$, then the mean-squared-error can be expressed in the form[b]:

$$\mathcal{E}^2 \approx \frac{1}{2}\operatorname{trace}\left((\boldsymbol{Q}\odot\boldsymbol{\Delta})\boldsymbol{Y}\right) + (\boldsymbol{q}\odot\boldsymbol{\delta})^\top\boldsymbol{y} + \text{const}$$

The approximate minimum mean-squared-error is found by the integer programming optimisation:

$$\begin{aligned} \textbf{minimise} \quad & \operatorname{trace}\left(\hat{\boldsymbol{Q}}\boldsymbol{Y}\right) + 2\hat{\boldsymbol{q}}^\top\boldsymbol{y} \\ \textbf{subject to} \quad & y_i \in \{-1, 1\} \end{aligned}$$

The integer programming constraints are equivalent to:

$$Y_{ii} = 1, \quad \operatorname{rank}\begin{bmatrix} \boldsymbol{Y} & \boldsymbol{y} \\ \boldsymbol{y}^\top & 1 \end{bmatrix} = 1, \quad \begin{bmatrix} \boldsymbol{Y} & \boldsymbol{y} \\ \boldsymbol{y}^\top & 1 \end{bmatrix} \succeq \boldsymbol{0}$$

where $\succeq 0$ means that the matrix is positive semi-definite. *Lu* [256, 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:

$$\begin{aligned} \textbf{minimise} \quad & \operatorname{trace}\left(\hat{\boldsymbol{Q}}\hat{\boldsymbol{Y}}\right) + 2\hat{\boldsymbol{q}}^\top\boldsymbol{y} \\ \textbf{subject to} \quad & \bar{\boldsymbol{Y}} = \begin{bmatrix} \hat{\boldsymbol{Y}} & \boldsymbol{y} \\ \boldsymbol{y}^\top & 1 \end{bmatrix} \succeq \boldsymbol{0} \end{aligned}$$

Define a length $P = \frac{M(M-1)}{2} + M$ augmented variable, $\bar{\boldsymbol{y}}$, consisting of the M elements of $\boldsymbol{y}$ and the $\frac{M(M-1)}{2}$ distinct off-diagonal elements of $\hat{\boldsymbol{Y}}$. The relaxed optimisation problem becomes:

$$\begin{aligned} \textbf{minimise} \quad & \boldsymbol{c}^\top\bar{\boldsymbol{y}} \\ \textbf{subject to} \quad & \bar{\boldsymbol{Y}} \succeq \boldsymbol{0} \end{aligned}$$

[a] This is an equality for an FIR filter transfer function.
[b] The *trace* of a matrix is defined in Appendix B.2.

Murumatsu and *Suzuki* [143, Section 3.3] and *Ito et al.* [211], [212, Equation 19] 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 $i < j$ and assuming $-1 \leq y \leq 1$.

Also [212, Equation 20]:

$$\begin{aligned} \hat{Y}_{ij} + \hat{Y}_{ik} + \hat{Y}_{jk} &\geq -1 \\ \hat{Y}_{ij} - \hat{Y}_{ik} - \hat{Y}_{jk} &\geq -1 \\ -\hat{Y}_{ij} - \hat{Y}_{ik} + \hat{Y}_{jk} &\geq -1 \\ -\hat{Y}_{ij} + \hat{Y}_{ik} - \hat{Y}_{jk} &\geq -1 \end{aligned} \tag{16.3}$$

for $i < j < k$. In practice, Equation 16.3 introduces a large number of constraints and I have found that Equation 16.2 is sufficient.

Lu [255, Section V] shows how to solve this *semidefinite programming* optimisation problem with the *SeDuMi* [229] 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 is the number of distinct components of $\bar{\boldsymbol{Y}}$, 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* [256] 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, σ_l. "*The motivation ... is that a perfect solution* $\hat{\boldsymbol{Y}}^*$ *... 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.*"

Ito et al. [212, Equation 23] suggest the following relaxation of the SDP problem to a linear programming (LP) problem:

$$\begin{aligned} \textbf{minimise} \quad & 2 \sum_{i<j} \hat{Q}_{ij} \hat{Y}_{ij} + \sum_i \hat{Q}_{ii} + 2\hat{\boldsymbol{q}}^\top \boldsymbol{y} \\ \textbf{subject to} \quad & \text{Equation 16.2} \\ & \text{Equation 16.3} \\ & -1 \leq y_i \leq 1 \end{aligned} \tag{16.4}$$

In the following, the SDP coefficient optimisation proceeds in successive relaxation optimisation steps. At each step, firstly, the response is SDP optimised for the remaining coefficients that are to be fixed to signed-digit values, one of the fixed coefficients is retained and the response of the remaining active floating-point coefficients is SOCP optimised. It is not possible to simultaneously optimise for both fixed and floating point coefficients in Equation 16.4 as this introduces terms in $y_m y_n$, where y_m are fixed to signed-digit values and y_n are not[c].

[c] SeDuMi supports hyperbolic or *rotated Lorentz cone* constraints with disjoint sets of decision variables. See the discussion of SOCP problems with hyperbolic constraints in *Lobo et al.* [146, Section 2.3].

16.3 SDP-relaxation search for the signed-digit coefficients of a direct-form symmetric FIR filter

Following Appendix N.2, the components of the amplitude responses of a direct-form symmetric 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 piece-wise constant amplitude response of a direct-form symmetric band-pass FIR filter. The filter specification is:

```
M=15 % Number of distinct coefficients
nbits=10 % Coefficient bits
ndigits=2 % Nominal average coefficient signed-digits
ftol=0.0001 % Tolerance on coef. update
ctol=0.0001 % Tolerance on constraints
n=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 lower edge
fasu=0.25 % Amplitude stop band upper edge
dBas=36 % Amplitude stop band peak-to-peak ripple
Wasl=40 % Amplitude lower stop band weight
Wasu=40 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 10 bits and 2 signed-digits. The Octave function *directFIRsymmetric_sdp_mmsePW* implements SDP optimisation with the triangle inequalities of Equation 16.2. Successive relaxation optimisation is performed by calling *directFIRsymmetric_sdp_mmsePW* to find the SDP optimised active coefficients, fixing one of these coefficients to a signed-digit value and then calling *directFIRsymmetric_slb* to SOCP PCLS optimise the remaining active coefficients.

The distinct direct-form symmetric FIR bandpass filter 10-bit 2-signed-digit coefficients are:

```
hM1_sd = [        0,       -5,      -12,       -5, ...
                 12,       24,       12,       -3, ...
                  3,       20,        0,      -60, ...
                -96,      -40,       80,      144 ]'/512;
```

The distinct direct-form symmetric FIR bandpass filter 10-bit 2-signed-digit coefficients found by "global" SDP optimisation are:

```
hM1_sdp = [        0,       -6,      -12,       -5, ...
                  12,       24,       12,       -4, ...
                   3,       20,        0,      -62, ...
                 -96,      -40,       80,      144 ]'/512;
```

The distinct direct-form symmetric FIR bandpass filter 10-bit 2-signed-digit coefficients found by successive relaxation SDP optimisation are:

```
hM1_min = [        0,       -6,      -12,       -6, ...
                  12,       24,       12,       -4, ...
                   2,       20,        2,      -60, ...
                 -96,      -40,       80,      144 ]'/512;
```

Figure 16.1 compares the pass-band and stop-band responses of the filter with floating-point coefficients, 10-bit 2-signed-digit coefficients and 10-bit 2-signed-digit coefficients found by "global" and successive relaxation 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 "global" and successive relaxationSDP optimisation.

Figure 16.1: Comparison of the amplitude responses for a direct-form symmetric FIR bandpass filter with 10-bit 2-signed-digit coefficients and 10-bit 2-signed-digit coefficients found by "global" and successive relaxation SDP optimisation.

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Initial	1.389e-03	0.07924		
10-bit 2-signed-digit	3.146e-03	0.08532	28	14
10-bit 2-signed-digit(SDP)	3.149e-03	0.08725	27	13
10-bit 2-signed-digit(min)	2.032e-03	0.09449	27	12

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 2-signed-digit coefficients and 10-bit 2-signed-digit coefficients found by "global" and successive relaxation SDP optimisation.

16.4 SDP-relaxation search for 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
ftol=1e-05 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=500 % Frequency points across the band
fapl=0.01 % Amplitude pass band lower edge
fapu=0.49 % Amplitude pass band upper edge
dBap=0.2 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Was=0 % Amplitude stop band weight
```

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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
hM1_allocsd_digits = [  4,  1,  1,  4, ...
                        1,  1,  1,  3, ...
                        1,  1,  1,  1, ...
                        1,  1,  2,  2, ...
                        2,  1,  4,  1, ...
                        2,  2,  2,  2, ...
                        1,  1,  2,  3, ...
                        2,  3,  2,  2, ...
                        2,  2,  2,  4, ...
                        3,  2,  3,  4 ]';
```

The distinct direct-form Hilbert FIR filter coefficients found with signed-digit allocation by the heuristic of *Ito et al.* are:

```
hM1_Ito = [       -2,       -2,       -2,       -2, ...
                  -2,       -4,       -4,       -4, ...
                  -4,       -4,       -8,       -8, ...
                  -8,       -8,      -10,      -12, ...
                 -12,      -16,      -16,      -16, ...
                 -20,      -24,      -24,      -28, ...
                 -32,      -32,      -40,      -42, ...
                 -48,      -54,      -60,      -72, ...
                 -80,      -96,     -112,     -141, ...
                -184,     -260,     -432,    -1304 ]'/2048;
```

The distinct direct-form Hilbert FIR filter coefficients found with signed-digit allocation by the heuristic of *Ito et al.* by "global" SDP optimisation are:

```
hM1_sdp = [       -2,       -1,       -2,       -2, ...
                  -2,       -4,       -4,       -4, ...
                  -4,       -4,       -8,       -8, ...
                  -8,       -8,      -10,      -12, ...
                 -14,      -16,      -16,      -16, ...
                 -20,      -24,      -24,      -28, ...
                 -32,      -32,      -40,      -42, ...
                 -48,      -54,      -62,      -72, ...
                 -80,      -96,     -112,     -141, ...
                -184,     -258,     -432,    -1304 ]'/2048;
```

The distinct direct-form Hilbert FIR filter coefficients found with signed-digit allocation by the heuristic of *Ito et al.* by successive relaxation SDP optimisation are:

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 "global" and successive relaxation SDP optimisation.

```
hM1_min = [        -1,        -1,        -1,        -2, ...
                   -2,        -2,        -4,        -4, ...
                   -4,        -4,        -4,        -8, ...
                   -8,        -8,       -10,       -10, ...
                  -12,       -16,       -15,       -16, ...
                  -20,       -20,       -24,       -28, ...
                  -32,       -32,       -36,       -42, ...
                  -48,       -54,       -60,       -68, ...
                  -80,       -96,      -112,      -141, ...
                 -184,      -258,      -432,     -1304 ]'/2048;
```

The corresponding FIR filter is:

```
h=[kron(hM1_min,[1;0]);-flipud(kron(hM1_min,[0;1]))](1:end-1);
```

Figure 16.2 compares the amplitude responses of the filter with floating-point coefficients, 12-bit signed-digit coefficients allocated by the algorithm of *Ito et al.*, 12-bit signed-digit coefficients allocated by the algorithm of *Ito et al.* with "global" SDP optimisation and 12-bit signed-digit coefficients allocated by the algorithm of *Ito et al.* with successive relaxation SDP optimisation. Table 16.2 compares, for each filter design, 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.

	Cost	Pass-band response	Signed-digits	Shift-and-adds
Initial	8.101e-07	0.01078		
Exact(SOCP)	8.101e-07	0.00231		
12-bit 2-signed-digit	2.078e-04	0.04514	72	32
12-bit 2-signed-digit(Ito)	1.192e-05	0.01241	69	29
12-bit 2-signed-digit(SDP)	1.147e-05	0.01421	69	29
12-bit 2-signed-digit(min)	9.737e-06	0.01205	70	30

Table 16.2: Comparison of the cost, maximum pass-band response error 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 by the heuristic of *Ito et al.* and performing "global" and successive relaxation SDP optimisation.

16.5 SDP-relaxation search for 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.3.1. The filter specification is:

```
M=8 % Number of distinct coefficients
nbits=12 % Coefficient bits
ndigits=2 % Nominal average coefficient signed-digits
ftol=1e-05 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=5000 % Frequency points across the band
fapl=0.1632 % Amplitude pass band lower edge
fapu=0.3368 % Amplitude pass band upper edge
dBap=0.24 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Was=10 % Amplitude stop band weight
```

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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
hM1_allocsd_digits = [  2,  1,  1,  2, ...
                        1,  1,  4,  4 ]';
```

The distinct direct-form Hilbert band-pass FIR filter coefficients found with signed-digit allocation with the heuristic of *Ito et al.* are:

```
hM1_Ito = [      -20,        8,       64,      -30, ...
                -128,      128,      327,     -868 ]'/2048;
```

The distinct direct-form Hilbert band-pass FIR filter coefficients found with signed-digit allocation with the heuristic of *Ito et al.* by SDP "global" optimisation are:

```
hM1_sdp = [      -20,        4,       64,      -30, ...
                -128,       64,      326,     -868 ]'/2048;
```

The distinct direct-form Hilbert band-pass FIR filter coefficients found with signed-digit allocation with the heuristic of *Ito et al.* by sucessive relaxation SDP optimisation are:

```
hM1_min = [      -24,        8,       64,      -34, ...
                -128,      128,      332,     -876 ]'/2048;
```

The corresponding FIR filter is:

```
h=[kron(hM1_min,[1;0]);-flipud(kron(hM1_min,[0;1]))](1:end-1);
```

Figure 16.3 compares the amplitude responses of the filter with floating-point coefficients, 12-bit signed-digit coefficients allocated by the algorithm of *Ito et al.*, 12-bit signed-digit coefficients allocated by the algorithm of *Ito et al.* with "global" SDP optimisation and 12-bit signed-digit coefficients allocated by the algorithm of *Ito et al.* with successive relaxation SDP optimisation[d]. Table 16.3 compares, for each filter design, 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.

[d]The amplitude response is symmetric about $f = 0.25$.

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 "global" and successive relaxation SDP.

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Initial	4.006e-04	0.03324		
12-bit 2-signed-digit	2.159e-03	0.02341	16	8
12-bit 2-signed-digit	2.159e-03	0.02341	16	8
12-bit 2-signed-digit(Ito)	5.067e-04	0.01855	16	8
12-bit 2-signed-digit(SDP)	7.150e-03	0.07213	16	8
12-bit 2-signed-digit(min)	3.618e-04	0.01413	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 "global" and successive relaxation SDP optimisation.

16.6 SDP-relaxation search for the signed-digit coefficients of a lattice bandpass R=2 IIR filter

The Octave script *sdp_relaxation_schurOneMlattice_bandpass_R2_10_nbits_test.m* performs successive SDP relaxations to optimise the response of the SQP optimised band-pass R=2 Schur one-multiplier lattice filter of Section 10.3.1. The filter specification is:

```
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
ftol=0.0001 % Tolerance on coefficient 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=1.5 % 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=5000 % Amplitude lower stop band weight
Wasu=10000 % 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=1 % 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 15.3. 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Lim et al.* are:

```
k_allocsd_digits = [  0,  4,  0,  5, ...
                      0,  4,  0,  4, ...
                      0,  3,  0,  4, ...
                      0,  3,  0,  4, ...
                      0,  2,  0,  2 ]';


c_allocsd_digits = [  2,  3,  4,  3, ...
                      3,  4,  4,  4, ...
                      3,  4,  4,  2, ...
                      2,  3,  3,  1, ...
                      3,  2,  1,  1, ...
                      2 ]';
```

The filter coefficients found with signed-digit allocation with the heuristic of *Lim et al.* are:

```
k0_Lim = [        0,      344,        0,      254, ...
                  0,      177,        0,      214, ...
                  0,      152,        0,      129, ...
                  0,       76,        0,       52, ...
                  0,       20,        0,        8 ]'/512;
```

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Exact	0.006778	-36.0		
10-bit 3-signed-digit	0.011697	-35.8	73	42
10-bit 3-signed-digit(Lim)	0.010182	-35.2	78	47
10-bit 3-signed-digit(SDP)	0.005147	-34.8	79	48
10-bit 3-signed-digit(min)	0.006438	-34.8	78	48

Table 16.4: Comparison of the cost and number of 10-bit shift-and-add operations required to implement the coefficient multiplications for an R=2 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.

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Exact	0.006705	-36.0		
12-bit 4-signed-digit	0.007088	-35.4	96	65
12-bit 4-signed-digit(Lim)	0.006778	-35.8	102	71
12-bit 4-signed-digit(SDP)	0.006106	-35.8	102	71
12-bit 4-signed-digit(min)	0.006535	-35.8	102	71

Table 16.5: 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 bandpass filter with 12-bit integer coefficients found by allocating an average of 4-signed-digits to each coefficient using the heuristic of *Lim et al.* and performing SDP-relaxation search.

```
c0_Lim = [        36,        -7,      -153,      -247, ...
                 -84,        63,       204,       154, ...
                   9,       -42,       -41,        -6, ...
                  -5,       -18,       -13,         2, ...
                  13,         8,         1,         1, ...
                   3 ]'/512;
```

The filter coefficients found by the SDP-relaxation search are:

```
k_min = [          0,       344,         0,       253, ...
                   0,       180,         0,       214, ...
                   0,       156,         0,       130, ...
                   0,        79,         0,        50, ...
                   0,        18,         0,         6 ]'/512;


c_min = [         36,        -6,      -153,      -247, ...
                 -84,        64,       204,       155, ...
                   9,       -42,       -40,        -6, ...
                  -5,       -19,       -13,         2, ...
                  13,         8,         1,         0, ...
                   6 ]'/512;
```

Figure 16.4 compares the pass-band responses of the $R = 2$ filter with floating-point 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 performing SDP-relaxation search. Figure 16.5 shows the filter stop-band response and Figure 16.6 shows the filter pass-band group delay response. Figure 16.7 shows the history of the difference of the successive relaxed signed-digit coefficients from the initial floating-point values. Table 16.4 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.* and performing SDP-relaxation search. For comparison, Table 16.5 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.* and performing SDP-relaxation search with the Octave script *sdp_relaxation_schurOneMlattice_bandpass_R2_12_nbits_test.m*.

Figure 16.4: Comparison of the pass-band amplitude responses for an R=2 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 16.5: Comparison of the stop-band amplitude responses for an R=2 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 pass-band group delay responses for an R=2 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: History of the difference of the successive SDP relaxed signed-digit coefficients from the initial floating-point values.

16.7 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_14_nbits_test.m* performs successive SDP relaxations to optimise the response of an elliptic low-pass filter implemented with 14-bit, 4-signed digit coefficients. The filter specification is:

```
nbits=14 % Coefficient bits
ndigits=4 % Nominal average coefficient signed-digits
ftol=1e-05 % Tolerance on coefficient 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=0.08 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
fas=0.175 % Amplitude stop band edge
dBas=72 % Amplitude stop band peak-to-peak ripple
Was=1e+07 % Amplitude stop band weight
```

The filter specification is similar to that for the filter designed with branch-and-bound search shown in Section 14.12. For comparison, 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, 5-signed-digit coefficients .

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 14-bit, 4-signed-digit coefficients.

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 15.3. 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 14-bit, 4-signed-digit filter coefficients found by the SDP-relaxation search are:

```
A1k_min = [    -4928,     8092,    -6432,     7088, ...
               -5884,     2960 ]'/8192;


A2k_min = [    -5648,     7682,    -6688,     5984, ...
               -2940 ]'/8192;
```

Figures 16.8 and 16.9 compare the pass-band and stop-band responses of the filter with floating-point coefficients, 14-bit coefficients having 4 signed-digits each, signed-digit coefficients found with "global" SDP approximation and those found by SDP-relaxation search. Figure 16.10 shows the history of the difference of the successive relaxed signed-digit coefficients from the initial floating-point values. Table 16.6 compares the cost and the number of 14 bit shift-and-add operations required to implement the 11 coefficient multiplications.

Figure 16.8: Comparison of the pass-band amplitude responses for a parallel all-pass Schur one-multiplier lattice elliptic low-pass filter with 14-bit, 4-signed-digit integer coefficients and performing SDP relaxation search.

Figure 16.9: Comparison of the stop-band amplitude responses for a parallel all-pass Schur one-multiplier lattice elliptic low-pass filter with 14-bit, 4-signed-digit integer coefficients and performing SDP relaxation search.

Figure 16.10: History of the difference of the successive SDP relaxed signed-digit coefficients from the initial floating-point values.

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Initial	7.52e-06	-84.0		
14-bit 4-signed-digit	3.89e-03	-40.4	44	33
14-bit 4-signed-digit(SDP)	2.14e-05	-51.3	44	33
14-bit 4-signed-digit(min)	9.55e-06	-71.3	43	32

Table 16.6: Comparison of the cost, stop-band attenuation and number of 14-bit shift-and-add operations required to implement the coefficient multiplications for a parallel all-pass Schur one-multiplier lattice elliptic-lowpass filter with 14-bit, 4-signed-digit coefficients and performing SDP relaxation search.

16.8 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 *Ito et al.* as shown in Section 11.1. The filter specification is:

```
nbits=13 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
ftol=0.0001 % Tolerance on coef. update
ctol=1e-05 % Tolerance on constraints
n=1000% Frequency points across the band
NA1k=10 % All-pass filter a order
NA2k=10 % All-pass filter b order
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=0.15 % Amplitude pass band peak-to-peak ripple
Wap=20 % Amplitude pass band weight
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=50000 % Amplitude lower stop band weight
Wasu=5000 % Amplitude upper stop band weight
ftpl=0.11 % Pass band delay lower edge
ftpu=0.19 % Pass band delay upper edge
tp=16 % Nominal pass band filter group delay
tpr=0.16 % Delay pass band peak-to-peak ripple
Wtp=2 % Delay pass band weight
fppl=0.11 % Pass band phase response lower edge
fppu=0.19 % Pass band phase response upper edge
pp=3.5 % Pass band initial phase response (rad./pi)
ppr=0.004 % Pass band phase response ripple(rad./pi)
Wpp=20 % Pass band phase response weight
fdpl=0.1 % Pass band dAsqdw response lower edge
fdpu=0.2 % Pass band dAsqdw response upper edge
dp=0 % Pass band initial dAsqdw response (rad./pi)
dpr=1 % Pass band dAsqdw response ripple(rad./pi)
Wdp=0.001 % Pass band dAsqdw 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 15.3. 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 numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
A1k_allocsd_digits = [  3,  4,  3,  3, ...
                        4,  3,  3,  3, ...
                        2,  2 ]';


A2k_allocsd_digits = [  4,  3,  3,  1, ...
                        4,  3,  2,  3, ...
                        4,  3 ]';
```

The filter coefficients found when the number of signed-digits is allocated by the heuristic of *Ito et al.* are:

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Exact	0.001521	-39.9		
13-bit 3-signed-digit	0.121881	-16.2	60	40
13-bit 3-signed-digit(Ito)	0.004232	-37.8	60	40
13-bit 3-signed-digit(SDP)	0.008065	-22.5	60	40
13-bit 3-signed-digit(min)	0.001264	-36.1	60	40

Table 16.7: 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 *Ito et al.* and performing SDP-relaxation search.

```
A1k0_Ito = [    -1856,     3392,     -944,      208, ...
                 2784,    -1344,      416,     2144, ...
                -1536,     1088 ]'/4096;


A2k0_Ito = [    -3312,     3600,    -1216,       64, ...
                 2848,    -1216,      448,     2144, ...
                -1488,     1104 ]'/4096;
```

The filter coefficients found by a single "global" SDP optimisation are:

```
A1k0_sdp = [    -1888,     3392,     -944,      208, ...
                 2800,    -1344,      432,     2160, ...
                -1536,     1088 ]'/4096;


A2k0_sdp = [    -3312,     3592,    -1216,      128, ...
                 2848,    -1216,      480,     2160, ...
                -1480,     1104 ]'/4096;
```

The filter coefficients found by the SDP-relaxation search are:

```
A1k_min = [    -1856,     3392,     -952,      200, ...
                2784,    -1344,      416,     2128, ...
               -1536,     1088 ]'/4096;


A2k_min = [    -3312,     3592,    -1248,       64, ...
                2848,    -1216,      448,     2144, ...
               -1480,     1120 ]'/4096;
```

Figures 16.11, Figure 16.12 and Figure 16.13 compare the pass band responses of the filter with floating-point coefficients, 13 bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and 13 bit signed-digit coefficients allocated with the algorithm of *Ito et al.* and SDP-relaxation search. The pass band phase response shown is adjusted for the nominal delay. Likewise, Figure 16.14 shows the filter stop band response. Table 16.7 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 *Ito et al.* with the SDP-relaxation search.

Figure 16.11: 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 *Ito et al.* and performing SDP-relaxation search.

Figure 16.12: 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 *Ito et al.* and performing SDP-relaxation search. The phase responses shown are adjusted for the nominal delay.

Figure 16.13: 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 *Ito et al.* and performing SDP-relaxation search.

Figure 16.14: 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 *Ito et al.* and performing SDP-relaxation search.

Chapter 17

POP programming search for integer and signed-digit filter coefficients

Lu [136] and *Lu* and *Hinamoto* [252] 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:

$$\begin{aligned} \textbf{minimise} \quad & \mathcal{E}\left(\boldsymbol{x_k}\right) + \nabla_x \mathcal{E}\left(\boldsymbol{x_k}\right)^\top \Delta \boldsymbol{x} + \frac{1}{2} \Delta \boldsymbol{x}^\top \nabla_x^2 \mathcal{E}\left(\boldsymbol{x_k}\right) \Delta \boldsymbol{x} \\ \textbf{subject to} \quad & \left(\boldsymbol{x_k} + \Delta \boldsymbol{x} - \boldsymbol{x}_u\right)^\top \left(\boldsymbol{x_k} + \Delta \boldsymbol{x} - \boldsymbol{x}_l\right) = \mathbf{0} \end{aligned}$$

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 $\boldsymbol{x}_l$ are the upper and lower bounds on the truncated coefficients, $\boldsymbol{x_k} + \Delta \boldsymbol{x}$. *Lasserre* [131] and *Waki et al.* [83] 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* [84], 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. In contrast to the SDP optimisation method shown in Section 16.2, POP optimisation allows successive relaxation optimisation of the response with both signed-digit and floating-point active coefficients.

17.1 POP relaxation search for the signed-digit coefficients of a one-multiplier lattice bandpass R=2 filter

The Octave script *pop_relaxation_schurOneMlattice_bandpass_R2_10_nbits_test.m* performs SparsePOP optimisation of the truncated coefficients of the SQP optimised band-pass Schur one-multiplier lattice $R = 2$ filter described in Section 10.3.1. The filter specification is:

```
use_kc0_coefficient_bounds=1
use_schurOneMlattice_allocsd_Lim=0
use_schurOneMlattice_allocsd_Ito=1
use_fix_coefficient_difference_greater_than_alpha=1
nbits=10 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
ftol=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
alpha_min=0.800000 % Threshold on coefficients to fix
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
```

	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)	0.0313	65	35

Table 17.1: Comparison of the cost and number of 10 bit shift-and-add operations required to implement the coefficient multiplications for an R=2 Schur one-multiplier lattice bandpass filter with 10 bit 3 signed-digit integer coefficients.

```
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=1e+06 % Amplitude upper stop band weight
```

The filter coefficients are truncated to 10 bits and an average of 3 signed-digits allocated by the heuristic of *Ito et al.*. At each coefficient relaxation step the script finds the upper and lower signed-digit approximations to the current set of active coefficients, selects those coefficients for which the floating-point values are are closer than a threshold value to those bounds [257, Section II.C]. The Octave function *schurOneMlattice_pop_socp_mmse.m* calls *SparsePOP* with a second order polynomial equality constraint requiring that the previously selected coefficients be equal to the corresponding signed-digit coefficient upper or lower bound. The remaining active coefficients are allowed to vary within the corresponding initial upper and lower bounds.

The $R = 2$ Schur lattice filter coefficients found by the SparsePOP relaxation search are:

```
k_min = [        0,      340,        0,      256, ...
                 0,      176,        0,      212, ...
                 0,      152,        0,      128, ...
                 0,       79,        0,       52, ...
                 0,       19,        0,        7 ]'/512;


c_min = [       36,       -8,     -156,     -248, ...
               -81,       64,      200,      152, ...
                 8,      -42,      -40,       -8, ...
                -4,      -18,      -13,        2, ...
                12,        8,        1,        0, ...
                 3 ]'/512;
```

Table 17.1 compares the cost, the number of signed-digits and the number of 10 bit shift-and-add operations required to implement the coefficient multiplications. Figure 17.1 compares the pass-band responses of the filter with floating-point coefficients and with 10 bit 3 signed-digit coefficients optimised with POP-relaxation search. Similarly, Figure 17.2 shows the filter stop-band response and Figure 17.3 shows the filter pass-band group delay response.

Figure 17.1: Comparison of the pass-band amplitude responses for an R=2 Schur one-multiplier lattice bandpass filter with 10 bit 3 signed-digit coefficients optimised with POP-relaxation search.

Figure 17.2: Comparison of the stop-band amplitude responses for an R=2 Schur one-multiplier lattice bandpass filter with 10 bit 3 signed-digit coefficients optimised with POP-relaxation search.

Figure 17.3: Comparison of the pass-band group delay responses for an R=2 Schur one-multiplier lattice bandpass filter with 10 bit 3 signed-digit coefficients optimised with POP-relaxation search.

17.2 POP relaxation search for the signed-digit coefficients of a one-multiplier lattice bandpass Hilbert R=2 filter

The Octave script *pop_relaxation_schurOneMlattice_bandpass_hilbert_R2_14_nbits_test.m* performs SparsePOP optimisation of the truncated coefficients of the SOCP optimised band-pass Hilbert $R = 2$ Schur one-multiplier lattice filter described in Section 10.3.2. The filter specification is:

```
ftol=0.001 % Tolerance on coef. update
ctol=0.0002 % Tolerance on constraints
nbits=14 % Coefficient bits
ndigits=4 % Nominal average coefficient signed-digits
use_kc0_coefficient_bounds=1
use_schurOneMlattice_allocsd_Lim=0
use_schurOneMlattice_allocsd_Ito=1
use_fix_coefficient_difference_greater_than_alpha=1
use_maximum_number_of_fixed_coefficients_is_alpha_num=1
alpha_num=3 % Fix at most alpha_num coefficients with POP
alpha_min=0.5 % Minimum threshold for alpha
rho=0.999 % Upper limit on abs(k)
n=1000% Frequency points across the band
fapl=0.1 % Amplitude pass band lower edge
fapu=0.2 % Amplitude pass band upper edge
dBap=0.2 % Amplitude pass band peak-to-peak ripple
Wap=2 % Amplitude 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=1 % Amplitude lower stop band weight
Wasu=1 % Amplitude upper stop band weight
fppl=0.1 % Pass band phase response lower edge
fppu=0.2 % Pass band phase response upper edge
pp=3.5 % Pass band initial phase response (rad./pi)
ppr=0.003 % Pass band phase response ripple(rad./pi)
Wpp=2 % Pass band phase response 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=0.3 % Delay pass band peak-to-peak ripple
Wtp=0.2 % Delay pass band weight
fdpl=0.1 % Pass band dAsqdw response lower edge
fdpu=0.2 % Pass band dAsqdw response upper edge
dp=0 % Pass band initial dAsqdw response (rad./pi)
dpr=1 % Pass band dAsqdw response ripple
Wdp=0.001 % Pass band dAsqdw response weight
```

The filter coefficients are truncated to 14 bits and an average of 4 signed-digits allocated by the heuristic of *Ito et al.*. At each coefficient relaxation step the script finds the upper and lower signed-digit approximations to the current set of active coefficients, selects those coefficients for which the floating-point values are are closer than a threshold value to those bounds [257, Section II.C]. The Octave function *schurOneMlattice_pop_socp_mmse.m* calls *SparsePOP* with a second order polynomial equality constraint requiring that the previously selected coefficients be equal to the corresponding signed-digit coefficient upper or lower bound. The remaining active coefficients are allowed to vary within the corresponding initial upper and lower bounds.

The signed-digits allocated by the heuristic of *Ito et al.* are:

```
k_allocsd_digits = [  0,  4,  0,  4, ...
                      0,  4,  0,  8, ...
                      0,  8,  0,  8, ...
                      0,  8,  0,  8, ...
                      0,  3,  0,  8 ]';

c_allocsd_digits = [  4,  3,  4,  3, ...
                      5,  3,  3,  2, ...
                      5,  2,  1,  1, ...
                      2,  3,  1,  2, ...
                      2,  2,  2,  3, ...
                      8 ]';
```

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Exact	1.4062e-03	-33.99		
14-bit 4-signed-digit	1.4114e-03	-33.27	110	79
14-bit 4-signed-digit(Ito)	1.6814e-03	-33.25	94	63
14-bit 4-signed-digit(POP min.)	1.2420e-03	-32.12	96	65

Table 17.2: Comparison of the cost and number of 14 bit shift-and-add operations required to implement the coefficient multiplications for an R=2 Schur one-multiplier lattice bandpass Hilbert filter with 14 bit 4 signed-digit integer coefficients.

The Schur lattice filter coefficients found by the SparsePOP relaxation search are:

```
k_min = [        0,     5824,        0,     3424, ...
                 0,     2324,        0,     4121, ...
                 0,     2751,        0,     3500, ...
                 0,     2135,        0,     2027, ...
                 0,      800,        0,      497 ]'/8192;


c_min = [     -344,      800,     3868,     2208, ...
             -1416,    -3568,    -2012,     -192, ...
              1640,     1280,      256,       32, ...
               544,      392,      -16,     -288, ...
               -96,      -48,      -48,     -114, ...
               -55 ]'/8192;
```

The corresponding transfer function polynomial coefficients are:

```
N_min = [  -0.0067138672,  -0.0130717456,  -0.0175063097,  -0.0282573877, ...
           -0.0351002738,  -0.0607288904,  -0.0489107579,  -0.0601657053, ...
           -0.0251110693,  -0.0527690139,   0.0138106915,   0.0251576646, ...
            0.1239257075,   0.0414096520,  -0.0065452251,  -0.1661664091, ...
           -0.1256302003,  -0.0912992506,   0.0461724559,   0.0457162551, ...
            0.0540620410 ];


D_min = [   1.0000000000,   0.0000000000,   1.7877054214,   0.0000000000, ...
            2.2879119313,   0.0000000000,   2.6444309191,   0.0000000000, ...
            2.7184140867,   0.0000000000,   2.2958734461,   0.0000000000, ...
            1.7308819795,   0.0000000000,   1.0511716735,   0.0000000000, ...
            0.5563415000,   0.0000000000,   0.2057550070,   0.0000000000, ...
            0.0606689453 ];
```

Table 17.2 compares the cost, the number of signed-digits and the number of 13 bit shift-and-add operations required to implement the coefficient multiplications. Figure 17.4 compares the pass-band responses of the filter with floating-point coefficients and with 13 bit 4 signed-digit coefficients optimised with POP-relaxation search. Similarly, Figure 17.5 shows the filter stop-band amplitude response, Figure 17.6 shows the filter pass-band phase response and Figure 17.7 shows the filter pass-band group delay response.

Figure 17.4: Comparison of the pass-band amplitude responses for an R=2 Schur one-multiplier lattice bandpass filter with 14 bit 4 signed-digit coefficients optimised with POP-relaxation search.

Figure 17.5: Comparison of the stop-band amplitude responses for an R=2 Schur one-multiplier lattice bandpass Hilbert filter with 14 bit 4 signed-digit coefficients optimised with POP-relaxation search.

Figure 17.6: Comparison of the pass-band phase responses for an R=2 Schur one-multiplier lattice bandpass Hilbert filter with 14 bit 4 signed-digit coefficients optimised with POP-relaxation search.

Figure 17.7: Comparison of the pass-band group delay responses for an R=2 Schur one-multiplier lattice bandpass Hilbert filter with 14 bit 4 signed-digit coefficients optimised with POP-relaxation search.

17.3 POP-relaxation search for the signed-digit coefficients of a one-multiplier lattice low-pass differentiator R=2 IIR filter

The Octave script *pop_relaxation_schurOneMlattice_lowpass_differentiator_R2_12_nbits_test.m* performs successive POP relaxations to optimise the response of an implementation of the low-pass differentiator $R = 2$ filter shown in Section 10.3.2 as the series combination of $1 - z^{-1}$ and a Schur one-multiplier lattice correction filter having denominator polynomial coefficients only in z^{-2} with 12-bit coefficients each having an average of 3 signed-digits allocated by the heuristic of *Ito et al.* and found by POP-relaxation search. The filter specification is:

```
ftol=0.0001 % Tolerance on coef. update
ctol=1e-08 % Tolerance on constraints
nbits=12 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
use_kc0_coefficient_bounds=1
use_schurOneMlattice_allocsd_Lim=1
use_schurOneMlattice_allocsd_Ito=0
use_fix_coefficient_difference_greater_than_alpha=1
alpha_num=3 % Fix at most alpha_num coefficients with POP
alpha_min=0.5 % Minimum threshold for alpha
rho=0.9999 % Upper limit on abs(k)
n=1000% Frequency points across the band
nN=10 % Correction filter order
fap=0.2 % Amplitude pass band upper edge
Arp=0.002 % Amplitude pass band peak-to-peak ripple
Wap=1 % Amplitude pass band weight
Art=0.002 % Amplitude transition band peak-to-peak ripple
Wat=0.0001 % Amplitude transition band weight
fas=0.4 % Amplitude stop band lower edge
Ars=0.0104 % Amplitude stop band peak-to-peak ripple
Was=0.1 % Amplitude stop band weight(PCLS)
fpp=0.2 % Phase pass band upper edge
pp=1.5 % Nominal pass band phase(rad./pi)
ppr=0.002 % Phase pass band peak-to-peak ripple(rad./pi)
Wpp=1 % Phase pass band weight
ftp=0.2 % Delay pass band upper edge
tp=9 % Pass band group delay
tpr=0.03 % Pass band group delay peak-to-peak ripple
Wtp=0.1 % Pass band group delay weight
fdp=0.2 % Correction filter pass band dCsqdw upper edge
cpr=0.04 % Correction filter pass band dCsqdw peak-to-peak ripple
Wdp=0.005 % Correction filter pass band dCsqdw weight
```

The filter coefficients are truncated to 12 bits allocated with an average of 3 signed-digits allocated 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 those coefficients for which the floating-point values are are closer than a threshold value to those bounds [257, Section II.C]. The Octave function *schurOneMlattice_slb* performs PCLS optimisation. The Octave function *schurOneMlattice_pop_socp_mmse.m* calls *SparsePOP* with a second order polynomial equality constraint requiring that the previously selected coefficients be equal to the corresponding signed-digit coefficient upper or lower bound. The remaining active coefficients are allowed to vary within the corresponding initial upper and lower bounds.

The numbers of signed-digits allocated to each coefficient by the heuristic of *Ito et al.* are:

```
k_allocsd_digits = [ 0, 3, 0, 3, ...
                     0, 2, 0, 3, ...
                     0, 3 ]';


c_allocsd_digits = [ 3, 4, 4, 3, ...
                     4, 2, 3, 3, ...
                     2, 3, 3 ]';
```

The signed-digit filter coefficients of the Schur one-multiplier lattice correction $R = 2$ filter found by the POP-relaxation search are:

Figure 17.8: Plot of the pass-band amplitude error response of a Schur one-multiplier lattice lowpass R=2 differentiator filter, having denominator polynomial coefficients only in z^{-2}, 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 POP-relaxation search.

```
k_min = [        0,      416,        0,      -56, ...
                 0,       20,        0,       -8, ...
                 0,        2 ]'/2048;


c_min = [      -44,     -450,     -588,      -63, ...
               150,      -36,      -47,       35, ...
                 3,      -13,        5 ]'/2048;
```

The corresponding coefficients of the correction filter transfer function polynomials are:

```
N_min = [   0.0024414062,  -0.0063414574,   0.0019450095,   0.0157555109, ...
           -0.0226103514,  -0.0138055639,   0.0676520510,  -0.0333953070, ...
           -0.2603386125,  -0.2587010639,  -0.0828708642 ];


D_min = [   1.0000000000,   0.0000000000,   0.1972618103,   0.0000000000, ...
           -0.0253062888,   0.0000000000,   0.0089701854,   0.0000000000, ...
           -0.0037136078,   0.0000000000,   0.0009765625 ];
```

Figure 17.8 shows the pass-band amplitude error compared to a desired response of $\frac{\omega}{2}$ for exact coefficients, 12-bit 3-signed-digit coefficients, coefficient with an average of 12-bit 3-signed-digits allocated with the heuristic of *Ito et al.* and coefficients found by POP-relaxation search. Similarly, Figure 17.9 shows the stop-band amplitude response, Figure 17.10 shows the pass-band phase response, adjusted for the nominal delay, and Figure 17.11 shows the pass-band group delay response of the low-pass differentiator filter. Figure 17.12 shows the pass-band amplitude error of the correction filter. Figure 17.13 shows the pole-zero plot of the correction filter with coefficients found by POP-relaxation search. Table 17.3 compares the cost, the maximum stop-band response, the number of signed-digits and the number of 12-bit shift-and-add operations required to implement the coefficient multiplications.

Figure 17.9: Plot of the stop-band amplitude response of a Schur one-multiplier lattice lowpass differentiator filter, having denominator polynomial coefficients only in z^{-2}, 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 POP-relaxation search.

Figure 17.10: Plot of the pass-band phase response of a Schur one-multiplier lattice lowpass differentiator R=2 filter, having denominator polynomial coefficients only in z^{-2}, with 12-bit signed-digit 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. The phase response shown is adjusted for the nominal delay.

Figure 17.11: Plot of the pass-band delay response of a Schur one-multiplier lattice lowpass differentiator R=2 filter, having denominator polynomial coefficients only in z^{-2}, 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 POP-relaxation search.

Figure 17.12: Error in the pass-band squared-amplitude response of the correction filter of a Schur one-multiplier lattice low-pass differentiator R=2 filter with 12 bit, 3 signed-digit coefficients and denominator polynomialterms only in z^{-2} found by performing POP-relaxation search.

Figure 17.13: Pole-zero plot of the R=2 Schur one-multiplier lattice low-pass differentiator filter having denominator polynomial terms only in z^{-2} with 12 bit, 3 signed-digit coefficients found by performing POP-relaxation search.

	Cost	Stop-band response(dB)	Signed-digits	Shift-and-adds
Exact	4.0719e-06	-49.11		
12-bit 3-signed-digit	1.6830e-05	-41.52	41	25
12-bit 3-signed-digit(Lim)	1.0954e-05	-45.09	41	25
12-bit 3-signed-digit(POP-relax)	1.0937e-05	-45.90	40	24

Table 17.3: 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 low-pass differentiator R=2 correction filter having denominator polynomial coefficients only in z^{-2}, 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 POP-relaxation search.

Chapter 18

Comparison of filter coefficient search methods

18.1 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 floating-point 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.

18.1.1 Searching with the bit-flip 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-flip algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 18.1 and 18.2 show the overall and passband responses of the prototype elliptic filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm. The cost function used with the signed-digit coefficients does not enforce the normalised-scaled orthogonal symmetry of $s_{00} = s_{22}$ and $s_{02} = -s_{20}$.

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-flip	1.1289	1.5299	0.9861	1.1097	0.9929
Signed-digit	2.3373	2.7157	3.6059	3.9013	∞
Signed-digit with bit-flip	0.8744	2.1571	1.3824	3.1746	1.1334

Table 18.1: 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 floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm

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-flip algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 18.3 and 18.4 show the overall and passband responses of the filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the bit-flip 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-flip algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 18.5 and 18.6 show the overall and passband responses of the parallel all-pass filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients and 2 signed-digit coefficients optimised with the bit-flip 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-flip algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 18.7 and 18.8 show the overall and passband responses of the parallel all-pass filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm and 6 bit 2 signed-digit coefficients and 6 bit 2 signed-digit coefficients optimised with the bit-flip 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-flip algorithm using $nbits = 6$, $bitstart = 4$ and $msize = 3$. Figures 18.9 and 18.10 show the overall and passband responses of the filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm. Note that the 2nd order state variable cascade filter obtained by simply converting the floating-point coefficients to 6 bit 2 signed-digit coefficients is unstable.

Table 18.1 compares the cost result for each test of the bit-flip algorithm.

The normalised-scaled 6 bit 2 signed-digit coefficients optimised with the bit-flip 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 18.1: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.2: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.3: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.4: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.5: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.6: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.7: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.8: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm, 6 bit 2 signed-digit coefficients, and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.9: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Figure 18.10: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip algorithm and 6 bit 2 signed-digit coefficients optimised with the bit-flip algorithm.

Cost	Normalised-scaled lattice	One-multiplier lattice	Normalised-scaled PA lattice	One-multiplier PA lattice	2nd Order cascade
Exact	1.0008	1.0008	1.0008	1.0008	1.0008
Rounded	1.5432	1.5299	1.2459	1.1097	1.2362
Rounded with simplex	1.0397	1.4617	0.9861	1.1097	0.8395
Signed-digit	2.3373	2.7157	3.6059	3.9013	∞
Signed-digit with simplex	1.6581	2.3918	3.2559	3.1746	3.2134

Table 18.2: 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 floating-point 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.

18.1.2 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 [163]. Figures 18.11 and 18.12 show the overall and passband responses of the prototype elliptic filter with floating-point 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 18.13 and 18.14 show the overall and passband responses of the filter with floating-point 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 18.15 and 18.16 show the overall and passband responses of the parallel all-pass filter with floating-point 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 18.17 and 18.18 show the overall and passband responses of the parallel all-pass filter with floating-point 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 18.19 and 18.20 show the overall and passband responses of the filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip 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 floating-point coefficients to 6 bit 2 signed-digit coefficients is unstable.

Table 18.2 compares the cost result for each test of the simplex algorithm.

Figure 18.11: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with floating-point 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 18.12: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with floating-point 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 18.13: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with floating-point 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 18.14: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with floating-point 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 18.15: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with floating-point 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 18.16: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with floating-point 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 18.17: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with floating-point 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 18.18: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with floating-point 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 18.19: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with floating-point 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 18.20: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with floating-point 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.

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	∞
Signed-digit with samin	0.8309	1.7673	1.0841	1.6706	∞

Table 18.3: 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 floating-point 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.

18.1.3 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 [163] implementation of the simulated annealing algorithm, *nonlin_min* [164, Demonstration 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. You may need to do more runs to find the best filter.

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 [163]. Figures 18.21 and 18.22 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with floating-point 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 18.23 and 18.24 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with floating-point 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 18.25 and 18.26 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with floating-point 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 18.27 and 18.28 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with floating-point 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 18.29 and 18.30 show the overall and passband responses of the filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip 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 floating-point coefficients to 6 bit 2 signed-digit coefficients is unstable.

Table 18.3 shows the minimum cost result for 10 runs of each test.

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 18.21: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with floating-point 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 18.22: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with floating-point 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 18.23: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with floating-point 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 18.24: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with floating-point 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 18.25: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with floating-point 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 18.26: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with floating-point 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 18.27: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with floating-point 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 18.28: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with floating-point 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 18.29: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with floating-point 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 18.30: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with floating-point 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.

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	∞
Signed-digit with de_min	0.7768	1.9147	1.1879	1.3505	0.6995

Table 18.4: 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 floating-point 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.

18.1.4 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 [163] implementation of the *differential evolution* algorithm of *Storn* and *Price* [204], *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. You may need to do more runs to find the best filter. 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 [163]. Figures 18.31 and 18.32 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with floating-point 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 18.33 and 18.34 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with floating-point 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 18.35 and 18.36 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with floating-point 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 18.37 and 18.38 show, for the best results from 10 runs, the overall and passband responses of the prototype elliptic filter with floating-point 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 18.39 and 18.40 show the overall and passband responses of the filter with floating-point coefficients, 6 bit rounded coefficients, 6 bit rounded coefficients optimised with the bit-flip 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 floating-point coefficients to 6 bit 2 signed-digit coefficients is unstable.

Table 18.4 shows the minimum cost result for 10 runs of each test.

Figure 18.31: Amplitude response of the 5th order elliptic low-pass filter synthesised as a normalised-scaled lattice filter with floating-point 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 18.32: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a scaled=normalised lattice filter with floating-point 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 18.33: Amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with floating-point 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 18.34: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a one multiplier lattice filter with floating-point 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 18.35: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with floating-point 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 18.36: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass normalised-scaled lattice filters with floating-point 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 18.37: Amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with floating-point 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 18.38: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as parallel all-pass one multiplier lattice filters with floating-point 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 18.39: Amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with floating-point 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 18.40: Pass-band amplitude response of the 5th order elliptic low-pass filter synthesised as a cascade of 2nd order state variable sections with floating-point 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.

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-flip	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	∞
Signed-digit with bit-flip	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	∞
Signed-digit with de_min	0.7768	1.9147	1.1879	1.3505	0.6995

Table 18.5: 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 floating-point 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.

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 18.6: 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.

18.1.5 Summary of the search algorithm comparison

Table 18.5 compares the cost result for each of the search algorithms. The relative time consumed by the algorithms is, in increasing order: simplex, bit-flip, 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.

Table 18.6 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.

18.2 Comparison of filter coefficient search methods for a low-pass differentiator filter with 12-bit integer and 3-signed-digit coefficients

This section compares the performance of an IIR tapped Schur one-multiplier lattice low-pass differentiator filter, with nominal pass-band response $-\imath\frac{\omega}{2}$, for which the transfer function denominator polynomial has terms only in z^{-2}. Section 5.3 describes the derivation of the Schur tapped one-multiplier lattice. Section 5.6.2 describes the state-variable description of the Schur tapped one-multiplier lattice filter. Section 5.8.2 describes pipelining a 6th order tapped Schur one-multiplier lattice with coefficients only in z^{-2}. The example scripts referenced in this section design a low-pass differentiator filter implemented as a zero at $z = 1$ in series with an order-10 correction filter.

Tne Octave script *schurOneMlattice_socp_slb_lowpass_differentiator_R2_test*, described in Section 10.3.2, designs the filter with floating point coefficients.

The Octave script *branch_bound_schurOneMlattice_lowpass_differentiator_R2_12_nbits_test*, described in Section 14.8, performs branch-and-bound search for 12-bit coefficients with an average of 3-signed-digits. At each branch, a coefficient is fixed to an upper or lower signed-digit approximation and the remaining un-fixed coefficients are SOCP-optimised.

The Octave script *socp_relaxation_schurOneMlattice_lowpass_differentiator_R2_12_nbits_test*, described in Section 15.16, performs SOCP-relaxation search for 12-bit coefficients with an average of 3-signed-digits.

The Octave script *pop_relaxation_schurOneMlattice_lowpass_differentiator_R2_12_nbits_test*, described in Section 17.3, performs POP-relaxation search for 12-bit coefficients with an average of 3-signed-digits.

In each script the number signed-digits of each coefficient is allocated by the method of *Lim et al.*, described in Section 11.1. The constraints and cost function band weights differ between each script in order to find the "best" response. In this section the band weights of the cost function are the same in each case. The common filter specification is:

```
nbits=12 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
nN=10 % Correction filter order
n=1000% Frequency points across the band
fap=0.2 % Amplitude pass band upper edge
Wap=1 % Amplitude pass band weight
Wat=0.0001 % Amplitude transition band weight
fas=0.4 % Amplitude stop band lower edge
Was=1 % Amplitude stop band weight(PCLS)
fpp=0.2 % Phase pass band upper edge
pp=1.5 % Nominal pass band phase(rad./pi)
Wpp=1 % Phase pass band weight
ftp=0.2 % Delay pass band upper edge
tp=9 % Pass band group delay
Wtp=1 % Pass band group delay weight
fdp=0.2 % Correction filter pass band dCsqdw upper edge
Wdp=1 % Correction filter pass band dCsqdw weight
```

Figure 18.41 compares the pass-band error amplitude response of each filter for 12-bit, 3-signed-digit coefficients and 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.*[c]. Figure 18.42 compares the stop-band error amplitude response of each filter for 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* found with *Branch-and-Bound* search, *SOCP-relaxation* search and *POP-relaxation* search. Figure 18.43 compares the stop-band error amplitude response of each filter for signed-digit coefficients. Figure 18.44 compares the stop-band error amplitude response of each filter found by searching for 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* . Figure 18.45 compares the pass-band phase response of each filter for signed-digit coefficients. The phase shown is adjusted for the nominal filter group-delay. Figure 18.46 compares the pass-band phase response of each filter found by searching for 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* . Figure 18.47 compares the pass-band group-delay response of each filter for signed-digit coefficients. Figure 18.48 compares the pass-band group-delay response of each filter found by searching for 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* . Figure 18.49 compares the gradient of the pass-band squared-amplitude response error with-respect-to angular frequency, $\frac{d|A|^2}{d\omega}$, of each filter for signed-digit coefficients. Figure 18.50 compares the gradient of the pass-band squared-amplitude response error with-respect-to angular frequency of each filter found by searching for 12-bit coefficients with an average of 3-signed-digits allocated by the method of *Lim et al.* . Table 18.7 compares, for each filter, a cost function, the maximum pass-band and stop-band response errors, the total number of signed-digits required by the 12-bit coefficients and the number of shift-and-add operations required to implement the filter coefficient multiplications.

[c]The plot shows the numerical difference between the desired and actual responses

Figure 18.41: Comparison of the pass-band amplitude response errors of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having 3-signed-digits or an average of 3-signed-digits, allocated by the method of *Lim et al.*.

Figure 18.42: Comparison of the pass-band amplitude response errors of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Lim et al.*, found by branch-and-bound, SOCP-relaxation and POP-relaxation search.

Figure 18.43: Comparison of the stop-band amplitude response errors of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having 3-signed-digits or an average of 3-signed-digits, allocated by the method of *Lim et al.*.

Figure 18.44: Comparison of the stop-band amplitude response errors of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Lim et al.*, found by branch-and-bound, SOCP-relaxation and POP-relaxation search.

Figure 18.45: Comparison of the pass-band phase response of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having 3-signed-digits or an average of 3-signed-digits, allocated by the method of *Lim et al.*. The phase response shown is adjusted for the nominal delay.

Figure 18.46: Comparison of the pass-band phase response of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Lim et al.*, found by branch-and-bound, SOCP-relaxation and POP-relaxation search. The phase response shown is adjusted for the nominal delay.

Figure 18.47: Comparison of the pass-band group-delay response of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having 3-signed-digits or an average of 3-signed-digits, allocated by the method of *Lim et al.*.

Figure 18.48: Comparison of the pass-band group-delay response of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Lim et al.*, found by branch-and-bound, SOCP-relaxation and POP-relaxation search.

Figure 18.49: Comparison of the gradient of the pass-band squared-amplitude response error with-respect-to angular frequency of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit 3-signed-digit coefficients and 12-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Lim et al.* .

Figure 18.50: Comparison of the gradient of the pass-band squared-amplitude response error with-respect-to angular frequency of a tapped, Schur one-multiplier, R=2, lattice low-pass differentiator filter with 12-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Lim et al.*, found by branch-and-bound, SOCP-relaxation and POP-relaxation search.

	Correction filter cost	Max. pass amp. error	Max. stop amp. error	Max. pass phase error (rad./π)	Max. pass delay error (samples)	12-bit signed digits	Shift-and-adds
Exact	2.69e-05	4.50e-04	3.51e-03	9.64e-05	3.00e-03		
Signed-Digit	1.05e-04	1.21e-03	8.39e-03	9.61e-04	1.61e-02	41	25
Signed-Digit(Lim)	1.83e-04	7.08e-03	8.23e-03	1.50e-03	2.35e-02	38	22
Branch-and-bound	3.65e-05	6.17e-04	4.84e-03	2.78e-04	8.83e-03	41	25
SOCP-relaxation	3.22e-05	6.22e-04	6.24e-03	1.96e-04	4.64e-03	37	21
POP-relaxation	6.60e-05	1.05e-03	5.07e-03	8.99e-04	1.13e-02	40	24

Table 18.7: Comparison of the response errors and the total number of signed digits in the 12-bit coefficients, each having an average of 3-signed digits, allocated by the method of *Lim et al.*, required for a tapped, Schur one-multiplier lattice, R=2, low-pass differentiator filter found by branch-and-bound, SOCP-relaxation and POP-relaxation search.

18.3 Comparison of filter coefficient search methods for a band-pass Hilbert filter with 13-bit integer and 3-signed-digit coefficients

This section compares the performance of an IIR tapped Schur one-multiplier lattice band-pass Hilbert filter for which the transfer function denominator polynomial has terms only in z^{-2}. Section 5.3 describes the derivation of the Schur tapped one-multiplier lattice. Section 5.6.2 describes the state-variable description of the Schur tapped one-multiplier lattice filter. Section 5.8.2 describes pipelining a 6th order tapped Schur one-multiplier lattice with coefficients only in z^{-2}.

Tne Octave script *schurOneMlattice_socp_slb_bandpass_hilbert_R2_test*, described in Section 10.3.2, designs the filter with floating point coefficients.

The Octave script *branch_bound_schurOneMlattice_bandpass_hilbert_R2_13_nbits_test*, described in Section 14.6, performs branch-and-bound search for 13-bit coefficients with an average of 3-signed-digits. At each branch, a coefficient is fixed to an upper or lower signed-digit approximation and the remaining un-fixed coefficients are SOCP-optimised.

The Octave script *socp_relaxation_schurOneMlattice_bandpass_hilbert_R2_13_nbits_test*, described in Section 15.7, performs SOCP-relaxation search for 13-bit coefficients with an average of 3-signed-digits.

The Octave script *pop_relaxation_schurOneMlattice_bandpass_hilbert_R2_13_nbits_test* performs POP-relaxation search for 13-bit coefficients with an average of 3-signed-digits. Unfortunately, I could not find a good filter specification for this example that completed successfully when run under QEMU, so the results of this script are not included here.

In each script the number signed-digits of each coefficient is allocated by the method of *Ito et al.*, described in Section 11.2. The constraints and cost function band weights differ between each script in order to find the "best" response. In this section the band weights of the cost function are the same in each case. The common filter specification is:

```
nbits=13 % Coefficient bits
ndigits=3 % Nominal average coefficient signed-digits
n=1000% Frequency points across the band
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=20 % Amplitude lower stop band weight
Wasu=10 % 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
Wtp=1 % Delay pass band weight
fppl=0.1 % Pass band phase response lower edge
fppu=0.2 % Pass band phase response upper edge
pp=3.5 % Pass band initial phase response (rad./pi)
Wpp=2 % Pass band phase response weight
fdpl=0.1 % Pass band dAsqdw response lower edge
fdpu=0.2 % Pass band dAsqdw response upper edge
Wdp=0.001 % Pass band dAsqdw response weight
```

Figure 18.51 compares the pass-band amplitude response of each filter with 13-bit coefficients each having an average of 3-signed-digits allocated by the method of *Ito et al.* found with branch-and-bound search and SOCP-relaxation search. Figure 18.52 compares the stop-band amplitude response of each filter with 13-bit coefficients each having an average of 3-signed-digits allocated by the method of *Ito et al.* found with branch-and-bound search and SOCP-relaxation search. Figure 18.53 compares the pass-band phase response of each filter with 13-bit coefficients each having an average of 3-signed-digits allocated by the method of *Ito et al.* found with branch-and-bound search and SOCP-relaxation search. The phase response is adjusted for the nominal delay. Figure 18.54 compares the pass-band group-delay response of each filter with 13-bit coefficients each having an average of 3-signed-digits allocated by the method of *Ito et al.* found with branch-and-bound search and SOCP-relaxation search. Figure 18.55 compares the gradient of the pass-band squared-amplitude response with-respect-to angular frequency, $\frac{d|A|^2}{d\omega}$, of each filter with 13-bit coefficients each having an average of 3-signed-digits allocated by the method of *Ito et al.* found with branch-and-bound search and SOCP-relaxation search. Table 18.8 compares, for each filter, a cost function, the maximum pass-band and stop-band responses, the total number of signed-digits required by the 13-bit coefficients and the number of shift-and-add operations required to implement the filter coefficient multiplications.

Figure 18.51: Comparison of the pass-band amplitude response errors of a tapped, Schur one-multiplier, R=2, lattice band-pass Hilbert filter with 13-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Ito et al.*, found with branch-and-bound search and SOCP-relaxation search.

Figure 18.52: Comparison of the stop-band amplitude response errors of a tapped, Schur one-multiplier, R=2, lattice band-pass Hilbert filter with 13-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Ito et al.*, found with branch-and-bound search and SOCP-relaxation search.

Figure 18.53: Comparison of the pass-band phase response of a tapped, Schur one-multiplier, R=2, lattice band-pass Hilbert filter with 13-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Ito et al.*, found with branch-and-bound search and SOCP-relaxation search. The phase response shown is adjusted for the nominal delay.

Figure 18.54: Comparison of the pass-band group-delay response of a tapped, Schur one-multiplier, R=2, lattice band-pass Hilbert filter with 13-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Ito et al.*, found with branch-and-bound search and SOCP-relaxation search.

Figure 18.55: Comparison of the gradient of the pass-band squared-amplitude response error with-respect-to angular frequency of a tapped, Schur one-multiplier, R=2, lattice band-pass Hilbert filter with 13-bit coefficients, each having an average of 3-signed-digits allocated by the method of *Ito et al.*, found with branch-and-bound search and SOCP-relaxation search.

	Hilbert filter cost	Min. pass amplitude (dB)	Max. stop amplitude (dB)	Max. pass phase error (rad./π)	Max. pass delay error (samples)	13-bit signed digits	Shift-and-adds
Exact	3.39e-03	-0.16	-33.99	1.00e-03	1.00e-01		
Signed-Digit	9.41e-03	-0.29	-31.39	7.30e-03	2.18e-01	88	57
Signed-Digit(Ito)	4.28e-03	-0.26	-30.77	4.05e-03	1.90e-01	81	50
Branch-and-bound	4.03e-03	-0.24	-33.60	2.04e-03	1.32e-01	89	58
SOCP-relaxation	4.44e-03	-0.26	-32.04	2.30e-03	1.44e-01	81	50

Table 18.8: Comparison of the responses and the total number of signed digits in the 13-bit coefficients, each having an average of 3-signed digits, allocated by the method of *Ito et al.*, required for a tapped, Schur one-multiplier lattice, R=2, band-pass Hilbert filter found by branch-and-bound search and SOCP-relaxation search.

Part IV

Appendixes

Appendix A

Review of Complex Variables

This chapter summarises *Kreyszig* [53, 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 ℓ 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 ϵ we can find a real positive δ such that, for all $z \neq z_0$ in the disk $|z - z_0| < \delta$, $|f(z) - \ell| < \epsilon$. We write:

$$\lim_{z \to z_0} f(z) = \ell$$

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 Δz approaches z with $\Delta y = 0$ then the derivative is -1 but if Δz approaches z with $\Delta x = 0$ then the derivative is 1. Hence the limit approaches different values along different paths to $z = x + \imath y$.

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 Δz 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 Δz 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, ζ_i, between z_{i-1} and z_i. Then form the sums

$$S_n = \sum_{i=1}^{n} f(\zeta_i) \Delta z_i$$

where $\Delta z_i = z_i - z_{i-1}$. Define $\zeta_i = \varepsilon_i + \imath \eta_i$ 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 ρ of K smaller than δ then

$$\left| \frac{f(z) - f(z_0)}{z - z_0} \right| < \frac{\epsilon}{\rho}$$

at each point of K. Since the length of K is $2\pi\rho$

$$\left| \int_K \frac{f(z) - f(z_0)}{z - z_0} dz \right| < 2\pi\epsilon$$

at each point of K and the second term is shown to be zero.

A.7 Derivatives of an analytic function

If $f(z)$ is analytic in D then it has derivatives of all orders in D which are also analytic functions in D

$$f^{(n)}(z_0) = \frac{n!}{2\pi\imath} \oint_c \frac{f(z)}{(z-z_0)^{n+1}} dz, \; n = 1, 2, \cdots$$

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 Δz approaches zero the right hand side approaches zero. The general formula follows by induction.

A.8 Laurent's Theorem

If $f(z)$ is *analytic* on two concentric circles with centre a and in the annulus between them, then $f(z)$ can be represented by the *Laurent* series

$$f(z) = \sum_{n=0}^{\infty} b_n (z-a)^n + \sum_{n=1}^{\infty} \frac{c_n}{(z-a)^n}$$

where

$$\begin{aligned} b_n &= \frac{1}{2\pi\imath} \oint_C \frac{f(z^*)}{(z^* - a)^{n+1}} dz^* \\ c_n &= \frac{1}{2\pi\imath} \oint_C (z^* - a)^{n-1} f(z^*) \, dz^* \end{aligned}$$

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* [53, 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 optimisation. See, for example, *Golub and Van Loan* [59], *Boyd* and *Vandenberghe* [215, Appendixes A, B and C], *Antoniou* and *Lu* [2, Appendix A], *Gallier* [65] and many, many other on-line resources.

B.1 Norm of a matrix

The vector norms are:

$$\begin{aligned} \|x\|_p &= (|x_1|^p + \cdots + |x_n|^p)^{\frac{1}{p}} \quad p \geq 1 \\ \|x\|_\infty &= \max_{1 \leq i \leq n} |x_i| \end{aligned}$$

$\|x\|$ is assumed to mean $\|x\|_2$.

The matrix norm, $\|A\|_{\alpha,\beta}$, is:

$$\|A\|_{\alpha,\beta} = \sup_{x \neq 0} \frac{\|Ax\|_\beta}{\|x\|_\alpha}$$

Golub and Van Loan [59, 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 elements on the diagonal:

$$\text{trace}\,(\boldsymbol{T}) = \sum_n T_{nn}$$

For a column vector, $\boldsymbol{y}$, and a matrix, $\boldsymbol{Q}$:

$$\begin{aligned}
\boldsymbol{y}^\top \boldsymbol{Q}\boldsymbol{y} &= \sum_k \sum_l Q_{kl} y_k y_l \\
&= \sum_k \sum_l Q_{kl} Y_{kl} \\
&= \sum_k \sum_l Q_{kl} Y_{lk} \\
&= \sum_k [QY]_{kk} \\
&= \operatorname{trace}(\boldsymbol{QY})
\end{aligned}$$

where $\boldsymbol{Y} = \boldsymbol{y}\boldsymbol{y}^\top$ is symmetric. If $\boldsymbol{Q}$ is also symmetric, then:

$$\operatorname{trace}(\boldsymbol{QY}) = 2\sum_{k<l} Q_{kl} Y_{kl} + \sum_k Q_{kk} Y_{kk}$$

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

Golub and Van Loan [59, 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 [59, 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 σ_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:

$$\begin{aligned}
\operatorname{rank}(A) &= r \\
\operatorname{null}(A) &= \operatorname{span}\{v_{r+1}, \ldots, v_n\} \\
\operatorname{range}(A) &= \operatorname{span}\{u_1, \ldots, u_r\} \\
A &= \sum_{k=1}^r \sigma_k u_k v_k^\top = U\Sigma V^\top
\end{aligned}$$

The columns of U, U^*, V and V^* each form an *orthonormal basis*. The column vectors, u_k, are called *left singular vectors*. Similarly, the column vectors, v_k, are called *right singular vectors*.

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, $z^* Z z$ is real and greater than 0 ($*$ represents the complex conjugate transpose operation). If $z = x + \imath y$, then:

$$\begin{aligned} z^* Z z &= \left(x^\top - \imath y^\top\right)(X + \imath Y)(x + \imath y) \\ &= \left(x^\top X - \imath y^\top X + \imath x^\top Y + y^\top Y\right)(x + \imath y) \\ &= x^\top X x - \imath y^\top X x + \imath x^\top Y x + y^\top Y x + \imath x^\top y + y^\top X y - x^\top Y y + \imath y^\top Y y \\ &= \left(x^\top X x + y^\top X y\right) + \imath\left(x^\top Y x + y^\top Y y\right) \end{aligned}$$

$Z \succ 0$ 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* [65]. See also *Jönsson* [247, 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 $x = A^{-1}(c - By)$:

$$\begin{aligned} \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} A^{-1} + A^{-1}B\left(D - CA^{-1}B\right)^{-1} CA^{-1} & -A^{-1}B\left(D - CA^{-1}B\right)^{-1} \\ -\left(D - CA^{-1}B\right)^{-1} CA^{-1} & \left(D - CA^{-1}B\right)^{-1} \end{bmatrix} \\ &= \begin{bmatrix} A^{-1} & -A^{-1}B\left(D - CA^{-1}B\right)^{-1} \\ 0 & \left(D - CA^{-1}B\right)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \\ &= \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} \begin{bmatrix} A^{-1} & 0 \\ 0 & \left(D - CA^{-1}B\right)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \end{aligned}$$

[a] A corollary is: $\det \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \det\left[A - BD^{-1}C\right] \times \det D$.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ 0 & I \end{bmatrix}$$

$D - CA^{-1}B$ 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:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} \left(A - BD^{-1}C\right)^{-1} & -A^{-1}B\left(D - CA^{-1}B\right)^{-1} \\ -\left(D - CA^{-1}B\right)^{-1}CA^{-1} & \left(D - CA^{-1}B\right)^{-1} \end{bmatrix}$$

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

B.7.1 Definitions on convex sets

See, for example, *Jönsson* [247, Section 2]. 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* [247, Section 3]. Also see *Rockafellar* [205, 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 $\hat{V}$
- in applications considering a finite dimensional Hilbert space over $\mathbb{R}$, the dual space is $\hat{V} = V$ itself, and the linear functional $\hat{v}(v) := \langle v, \hat{v} \rangle$ is the inner product
- the *affine hull* of S, denoted by $\operatorname{aff} S$, is 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 $z \in \hat{V}$ 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 [247, 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:

$S_1 : \sigma_0(y) \geq 0 \; \forall \, y \in V$ 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, σ_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 [247] gives the following example of the application of the S-procedure to a quadratic programming problem with one quadratic constraint:

$$
\begin{aligned}
\textbf{minimise} \quad & y^\top Q_0 y + 2 s_0^\top y + r_0 \\
\textbf{subject to} \quad & y^\top Q_1 y + 2 s_1^\top y + r_1 \geq 0
\end{aligned}
$$

A lower bound is obtained by the semi-definite relaxation:

$$
\begin{aligned}
\textbf{minimise} \quad & \operatorname{trace} Q_0 Y + 2 s_0^\top y + r_0 \\
\textbf{subject to} \quad & \operatorname{trace} Q_1 Y + 2 s_1^\top y + r_1 \geq 0 \\
& \begin{bmatrix} Y & y \\ y^\top & 1 \end{bmatrix} \succeq 0
\end{aligned}
$$

since $\operatorname{trace} Q_k y y^\top = y^\top Q_k y$ and the second constraint is equivalent to $Y \succeq y y^\top$.

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* [86].

B.7.4 The *log det A* penalty function

The $\log\det A$ function is widely used as a penalty function in convex optimisation. *VanAntwerp* and *Braatz* [95, 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 $S = \left\{A \,:\, A = A^\top \succ 0\right\}$. 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 λ_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, λ_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 seven 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 seven Chebyshev polynomials of the second kind.

[a] The convolution of the k'th Dirichlet kernel, $D_k(x)$, with a function, f, of period 2π, 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 ω 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 [248, Sections 5.4 and 5.8] calculates the weighted sum:

$$S(x)=\sum_{k=0}^{n}a_k\phi_k(x)$$

where ϕ_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 auxiliary 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* [52, Chapters 20, 21 and 22] and the *NIST Digital Library of MathematicalFunctions* [56, Chapters 19, 20 and 22][a]. *Vlček* and *Unbehauen* [147, Appendix] provide a brief review of the elliptic function theory required for IIR filter design.

D.1 Doubly periodic functions

Whittaker and *Watson* [52, Section 20.1] give the following definition of elliptic functions as doubly-periodic functions:

> Let ω_1 and ω_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 [52, 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* [52, 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* [56, 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* [138]. 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 κ 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* [56, Equation 19.2.8]. $K'(\kappa) = K(\kappa')$.

$E(\phi, \kappa)$ is the *incomplete elliptic integral of the second kind* [56, 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* [56, 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 [16] and [56, 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 [16, 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 η is the reverse of that shown by *Carlson* [16, 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* [56, 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 [16, Algorithm 1] computes Carlson's R_F function, Algorithm D.2 [16, Algorithm 2] computes Carlson's R_C function, Algorithm D.3 [16, Algorithm 3] computes Carlson's R_J function, and Algorithm D.4 [16, Algorithm 4] computes Carlson's R_D function.

Algorithm D.1 Carlson's algorithm for computing the R_F function [16, 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 [16, 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 [16, 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 [16, 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 [56, 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 [56, 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 $\tau' = -\frac{1}{\tau}$ [56, Section

[b] See [56, Section 23.15(i)].Suppose $\mathcal{A}$ denotes a bilinear transformation on τ:

$$\mathcal{A}\tau = \frac{a\tau + b}{c\tau + d}$$

where a, b, c and d are integers and $ad - bc = 1$. A *modular function* $f(\tau)$ is a function of τ 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 [56, Section 22.2]. The nome, q, is, in terms of the modulus, κ:

$$q = e^{-\pi\frac{K'(\kappa)}{K(\kappa)}}$$

The elliptic modulus, κ, 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 [56, Section 22.2]:

> As a function of z with fixed κ, 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 κ, with simple poles and zeros, and each is meromorphic in κ 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 [56, 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 [56, 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 [56, 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 sn, cn and cd elliptic functions are, respectively [56, 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 [17, 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 κ 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 κ 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 sn and cn functions are [56, 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 [56, Section 22.6].

D.9 Related functions

Jacobi's *amplitude* function [56, 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 [56, 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 [56, 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 [56, 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 [56].

Alternatively [56, 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 κ.

The *Zeta* function is periodic in $2K$ [56, 22.16.34]:

$$Z(x+2K,\kappa) = Z(x,\kappa)$$

Also [56, 22.16.33]:

$$Z(x+K,\kappa) = Z(x,\kappa) - \kappa^2 \operatorname{sn}(x,\kappa)\operatorname{cd}(x,\kappa)$$

The *Boost* C++ library [29] and *Mathematica* [254] 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 [56], shows the values of Jacobi's *Zeta* function for several elliptic moduluses, κ. Confusingly, this figure expresses the x argument as a multiple of π. 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* [59, 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 γ_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* [59, 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* [102].

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 [159] 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 coefficient vector
- symbolic differentiation by a symbolic algebra package. For example, *Maxima* [253] or the Octave-Forge *symbolic* calculation toolbox [168]
- 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 [15, *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.

Richards [137] 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

$$\begin{aligned}
\left|\left(e^{\imath\omega} - re^{\imath\theta}\right)\right| &= \left|(\cos\omega - r\cos\theta) + \imath\,(\sin\omega - r\sin\theta)\right| \\
&= \left\{\cos^2\omega - 2r\cos\omega\cos\theta + r^2\cos^2\theta + \sin^2\omega - 2r\sin\omega\sin\theta + r^2\sin^2\theta\right\}^{\frac{1}{2}} \\
&= \left\{1 - 2r\cos(\omega - \theta) + r^2\right\}^{\frac{1}{2}}
\end{aligned}$$

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:

$$\begin{aligned}
H\left(e^{\imath\omega}\right) &= A(\omega)\, e^{\imath\Theta(\omega)} \\
\ln H\left(e^{\imath\omega}\right) &= \ln A(\omega) + \imath\Theta(\omega) \\
T(\omega) = -\frac{d}{d\omega}\Theta(\omega) &= -\Im\left\{\frac{H'\left(e^{\imath\omega}\right)}{H\left(e^{\imath\omega}\right)}\right\}
\end{aligned}$$

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:

$$\begin{aligned}
P(\omega) = & \left[(V + Q)R - (U + M)\right]\omega + \cdots \\
& \cdots \sum_{j=1}^{U}\arctan\left(\frac{\sin\omega}{\cos\omega - R_{0j}}\right) - \cdots \\
& \cdots \sum_{j=1}^{V}\left\{\sum_{i=0}^{R-1}\arctan\left(\frac{\sin\omega - R_{pj}^s\sin(s2\pi i)}{\cos\omega - R_{pj}^s\cos(s2\pi i)}\right)\right\} + \cdots \\
& \cdots \sum_{j=1}^{\frac{M}{2}}\arctan\left(\frac{\sin 2\omega - 2r_{0j}\cos\theta_{0j}\sin\omega}{\cos 2\omega - 2r_{0j}\cos\theta_{0j}\cos\omega + r_{0j}^2}\right) - \cdots \\
& \cdots \sum_{j=1}^{\frac{Q}{2}}\left\{\sum_{i=0}^{R-1}\arctan\left(\frac{\sin 2\omega - 2r_{pj}^s\cos\left[s(\theta_{pj} + 2\pi i)\right]\sin\omega}{\cos 2\omega - 2r_{pj}^s\cos\left[s(\theta_{pj} + 2\pi i)\right]\cos\omega + r_{pj}^{2s}}\right)\right\}
\end{aligned}$$

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:

$$
\begin{aligned}
T(\omega) = & -\left[(V+Q)R-(U+M)\right]-\cdots \\
& \cdots \sum_{j=1}^{U} \frac{1-R_{0j}\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^{2}}+\cdots \\
& \cdots \sum_{j=1}^{V}\left\{\sum_{i=0}^{R-1}\frac{1-R_{pj}^{s}\cos\left(\omega-s2\pi i\right)}{1-2R_{pj}^{s}\cos\left(\omega-s2\pi i\right)+R_{pj}^{2s}}\right\}-\cdots \\
& \cdots \sum_{j=1}^{\frac{M}{2}}\left\{\frac{1-r_{0j}\cos\left(\omega-\theta_{0j}\right)}{1-2r_{0j}\cos\left(\omega-\theta_{0j}\right)+r_{0j}^{2}}+\frac{1-r_{0j}\cos\left(\omega+\theta_{0j}\right)}{1-2r_{0j}\cos\left(\omega+\theta_{0j}\right)+r_{0j}^{2}}\right\}+\cdots \\
& \cdots \sum_{j=1}^{\frac{Q}{2}}\left\{\sum_{i=0}^{R-1}\frac{1-r_{pj}^{s}\cos\left(\omega-s\left(\theta_{pj}+2\pi i\right)\right)}{1-2r_{pj}^{s}\cos\left(\omega-s\left(\theta_{pj}+2\pi i\right)\right)+r_{pj}^{2s}}\right\}+\cdots \\
& \cdots \sum_{j=1}^{\frac{Q}{2}}\left\{\sum_{i=0}^{R-1}\frac{1-r_{pj}^{s}\cos\left(\omega+s\left(\theta_{pj}+2\pi i\right)\right)}{1-2r_{pj}^{s}\cos\left(\omega+s\left(\theta_{pj}+2\pi i\right)\right)+r_{pj}^{2s}}\right\}
\end{aligned}
$$

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:

$$
\begin{aligned}
\frac{\partial A(\omega)}{\partial K} &= \frac{A(\omega)}{K} \\
\frac{\partial A(\omega)}{\partial R_{0j}} &= A(\omega)\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^{2}}\right\} \\
\frac{\partial A(\omega)}{\partial R_{pj}} &= -sR_{pj}^{s-1}A(\omega)\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^{s}-\cos\left(\omega-s2\pi i\right)}{1-2R_{pj}^{s}\cos\left(\omega-s2\pi i\right)+R_{pj}^{2s}}\right\} \\
\frac{\partial A(\omega)}{\partial r_{0j}} &= A(\omega)\left\{\frac{r_{0j}-\cos\left(\omega-\theta_{0j}\right)}{1-2r_{0j}\cos\left(\omega-\theta_{0j}\right)+r_{0j}^{2}}+\frac{r_{0j}-\cos\left(\omega+\theta_{0j}\right)}{1-2r_{0j}\cos\left(\omega+\theta_{0j}\right)+r_{0j}^{2}}\right\} \\
\frac{\partial A(\omega)}{\partial \theta_{0j}} &= A(\omega)\left\{\frac{r_{0j}\sin\left(\omega+\theta_{0j}\right)}{1-2r_{0j}\cos\left(\omega+\theta_{0j}\right)+r_{0j}^{2}}-\frac{r_{0j}\sin\left(\omega-\theta_{0j}\right)}{1-2r_{0j}\cos\left(\omega-\theta_{0j}\right)+r_{0j}^{2}}\right\} \\
\frac{\partial A(\omega)}{\partial r_{pj}} &= -sr_{pj}^{s-1}A(\omega)\sum_{i=0}^{R-1}\left\{\frac{r_{pj}^{s}-\cos\left(\omega-s\left(\theta_{pj}+2\pi i\right)\right)}{1-2r_{pj}^{s}\cos\left(\omega-s\left(\theta_{pj}+2\pi i\right)\right)+r_{pj}^{2s}}+\cdots\right. \\
&\quad \left.\cdots\frac{r_{pj}^{s}-\cos\left(\omega+s\left(\theta_{pj}+2\pi i\right)\right)}{1-2r_{pj}^{s}\cos\left(\omega+s\left(\theta_{pj}+2\pi i\right)\right)+r_{pj}^{2s}}\right\} \\
\frac{\partial A(\omega)}{\partial \theta_{pj}} &= -sr_{pj}^{s}A(\omega)\sum_{i=0}^{R-1}\left\{\frac{\sin\left(\omega+s\left(\theta_{pj}+2\pi i\right)\right)}{1-2r_{pj}^{s}\cos\left(\omega+s\left(\theta_{pj}+2\pi i\right)\right)+r_{pj}^{2s}}-\cdots\right. \\
&\quad \left.\cdots\frac{\sin\left(\omega-s\left(\theta_{pj}+2\pi i\right)\right)}{1-2r_{pj}^{s}\cos\left(\omega-s\left(\theta_{pj}+2\pi i\right)\right)+r_{pj}^{2s}}\right\}
\end{aligned}
$$

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

$$\begin{aligned}
\frac{\partial P(\omega)}{\partial r_{pj}} &= -\sum_{i=0}^{R-1}\left\{\frac{P_{pD}\frac{\partial P_{pN}}{\partial r_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial r_{pj}}}{P_{pN}^2 + P_{pD}^2}\right\} \\
\frac{\partial P(\omega)}{\partial \theta_{pj}} &= -\sum_{i=0}^{R-1}\left\{\frac{P_{pD}\frac{\partial P_{pN}}{\partial \theta_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial \theta_{pj}}}{P_{pN}^2 + P_{pD}^2}\right\}
\end{aligned}$$

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

The partial derivatives of the group delay are

$$\frac{\partial T(\omega)}{\partial K} = 0$$

$$
\begin{aligned}
\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\left(\omega - s2\pi i\right) - 2R_{pj}^s}{\left[1 - 2R_{pj}^s\cos\left(\omega - s2\pi i\right) + R_{pj}^{2s}\right]^2} \\
\frac{\partial T(\omega)}{\partial r_{0j}} &= \frac{2r_{0j} - \left(r_{0j}^2 + 1\right)\cos\left(\omega - \theta_{0j}\right)}{\left[1 - 2r_{0j}\cos\left(\omega - \theta_{0j}\right) + r_{0j}^2\right]^2} + \frac{2r_{0j} - \left(r_{0j}^2 + 1\right)\cos\left(\omega + \theta_{0j}\right)}{\left[1 - 2r_{0j}\cos\left(\omega + \theta_{0j}\right) + r_{0j}^2\right]^2} \\
\frac{\partial T(\omega)}{\partial \theta_{0j}} &= \frac{r_{0j}(r_{0j}^2 - 1)\sin\left(\omega - \theta_{0j}\right)}{\left[1 - 2r_{0j}\cos\left(\omega - \theta_{0j}\right) + r_{0j}^2\right]^2} - \frac{r_{0j}(r_{0j}^2 - 1)\sin\left(\omega + \theta_{0j}\right)}{\left[1 - 2r_{0j}\cos\left(\omega + \theta_{0j}\right) + 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\left(\omega - s\left(\theta_{pj} + 2\pi i\right)\right) - 2r_{pj}^s}{\left[1 - 2r_{pj}^s\cos\left(\omega - s\left(\theta_{pj} + 2\pi i\right)\right) + r_{pj}^{2s}\right]^2} + \cdots\right. \\
&\quad \left.\cdots\frac{\left(r_{pj}^{2s} + 1\right)\cos\left(\omega + s\left(\theta_{pj} + 2\pi i\right)\right) - 2r_{pj}^s}{\left[1 - 2r_{pj}^s\cos\left(\omega + s\left(\theta_{pj} + 2\pi i\right)\right) + 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\left(\omega - s\left(\theta_{pj} + 2\pi i\right)\right)}{\left[1 - 2r_{pj}^s\cos\left(\omega - s\left(\theta_{pj} + 2\pi i\right)\right) + r_{pj}^{2s}\right]^2} - \cdots\right. \\
&\quad \left.\cdots\frac{\sin\left(\omega + s\left(\theta_{pj} + 2\pi i\right)\right)}{\left[1 - 2r_{pj}^s\cos\left(\omega + s\left(\theta_{pj} + 2\pi i\right)\right) + r_{pj}^{2s}\right]^2}\right\}
\end{aligned}
$$

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):

$$
\begin{aligned}
\frac{\partial^2 A(\omega)}{\partial K^2} &= 0 \\
\frac{\partial^2 A(\omega)}{\partial K \partial R_{0j}} &= \frac{1}{K}\frac{\partial A(\omega)}{\partial R_{0j}} \\
\frac{\partial^2 A(\omega)}{\partial K \partial R_{pj}} &= \frac{1}{K}\frac{\partial A(\omega)}{\partial R_{pj}} \\
\frac{\partial^2 A(\omega)}{\partial K \partial r_{0j}} &= \frac{1}{K}\frac{\partial A(\omega)}{\partial r_{0j}} \\
\frac{\partial^2 A(\omega)}{\partial K \partial \theta_{0j}} &= \frac{1}{K}\frac{\partial A(\omega)}{\partial \theta_{0j}} \\
\frac{\partial^2 A(\omega)}{\partial K \partial r_{pj}} &= \frac{1}{K}\frac{\partial A(\omega)}{\partial r_{pj}} \\
\frac{\partial^2 A(\omega)}{\partial K \partial \theta_{pj}} &= \frac{1}{K}\frac{\partial A(\omega)}{\partial \theta_{pj}} \\
\frac{\partial^2 A(\omega)}{\partial R_{0j}^2} &= A(\omega)\frac{\sin^2\omega}{\left[1 - 2R_{0j}\cos\omega + R_{0j}^2\right]^2} \\
\frac{\partial^2 A(\omega)}{\partial R_{0k}\partial R_{0j}} &= \frac{\partial A(\omega)}{\partial R_{0k}}\left\{\frac{R_{0j} - \cos\omega}{1 - 2R_{0j}\cos\omega + R_{0j}^2}\right\} \\
\frac{\partial^2 A(\omega)}{\partial R_{pj}\partial R_{0j}} &= \frac{\partial A(\omega)}{\partial R_{pj}}\left\{\frac{R_{0j} - \cos\omega}{1 - 2R_{0j}\cos\omega + R_{0j}^2}\right\} \\
\frac{\partial^2 A(\omega)}{\partial r_{0j}\partial R_{0j}} &= \frac{\partial A(\omega)}{\partial r_{0j}}\left\{\frac{R_{0j} - \cos\omega}{1 - 2R_{0j}\cos\omega + R_{0j}^2}\right\} \\
\frac{\partial^2 A(\omega)}{\partial \theta_{0j}\partial R_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{0j}}\left\{\frac{R_{0j} - \cos\omega}{1 - 2R_{0j}\cos\omega + R_{0j}^2}\right\}
\end{aligned}
$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}\partial R_{0j}} = \frac{\partial A(\omega)}{\partial r_{pj}}\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{pj}\partial R_{0j}} = \frac{\partial A(\omega)}{\partial \theta_{pj}}\left\{\frac{R_{0j}-\cos\omega}{1-2R_{0j}\cos\omega+R_{0j}^2}\right\}$$

$$\begin{aligned}\frac{\partial^2 A(\omega)}{\partial R_{pj}^2} &= -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial R_{pj}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}+\cdots\\ &\cdots\ sR_{pj}^{s-2}A(\omega)\sum_{i=0}^{R-1}\left\{\frac{R_{pj}^{3s}+(s-3)R_{pj}^{2s}\cos(\omega-s2\pi i)}{\left[1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}\right]^2}+\cdots\right.\\ &\cdots\ \left.\frac{\left(2\cos^2(\omega-s2\pi i)-2s+1\right)R_{pj}^s+(s-1)\cos(\omega-s2\pi i)}{\left[1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}\right]^2}\right\}\end{aligned}$$

$$\frac{\partial^2 A(\omega)}{\partial R_{pk}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial R_{pk}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{0j}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial r_{0j}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{0j}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial \theta_{0j}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial r_{pj}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{pj}\partial R_{pj}} = -sR_{pj}^{s-1}\frac{\partial A(\omega)}{\partial \theta_{pj}}\left\{\sum_{i=0}^{R-1}\frac{R_{pj}^s-\cos(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\right\}$$

$$\begin{aligned}\frac{\partial^2 A(\omega)}{\partial r_{0j}^2} &= \frac{\partial A(\omega)}{\partial r_{0j}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}+\cdots\\ &\cdots\ A(\omega)\left\{\frac{2\sin^2(\omega-\theta_{0j})}{\left[1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2\right]^2}-\frac{1}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\cdots\right.\\ &\cdots\ \left.\frac{2\sin^2(\omega+\theta_{0j})}{\left[1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2\right]^2}-\frac{1}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}\end{aligned}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{0k}\partial r_{0j}} = \frac{\partial A(\omega)}{\partial r_{0k}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}$$

$$\begin{aligned}\frac{\partial^2 A(\omega)}{\partial \theta_{0j}\partial r_{0j}} &= \frac{\partial A(\omega)}{\partial \theta_{0j}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}+\cdots\\ &\cdots\ A(\omega)\left\{\frac{\left(1-r_{0j}^2\right)\sin(\omega+\theta_{0j})}{\left[1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2\right]^2}-\frac{\left(1-r_{0j}^2\right)\sin(\omega-\theta_{0j})}{\left[1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2\right]^2}\right\}\end{aligned}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{0k}\partial r_{0j}} = \frac{\partial A(\omega)}{\partial \theta_{0k}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}\partial r_{0j}} = \frac{\partial A(\omega)}{\partial r_{pj}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{pj}\partial r_{0j}} = \frac{\partial A(\omega)}{\partial \theta_{pj}}\left\{\frac{r_{0j}-\cos(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}+\frac{r_{0j}-\cos(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}\right\}$$

$$\begin{aligned}\frac{\partial^2 A(\omega)}{\partial \theta_{0j}^2} &= \frac{\partial A(\omega)}{\partial \theta_{0j}}\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}-\frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\}+\cdots\\ &\cdots\ A(\omega)\left\{\frac{\left(r_{0j}+r_{0j}^3\right)\cos(\omega+\theta_{0j})-2r_{0j}^2}{\left[1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2\right]^2}+\frac{\left(r_{0j}+r_{0j}^3\right)\cos(\omega-\theta_{0j})-2r_{0j}^2}{\left[1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2\right]^2}\right\}\end{aligned}$$

$$\frac{\partial^2 A(\omega)}{\partial \theta_{0k}\partial \theta_{0j}} = \frac{\partial A(\omega)}{\partial \theta_{0k}}\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2}-\frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial r_{pj}\partial\theta_{0j}} = \frac{\partial A(\omega)}{\partial r_{pj}}\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2} - \frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\}$$

$$\frac{\partial^2 A(\omega)}{\partial\theta_{pj}\partial\theta_{0j}} = \frac{\partial A(\omega)}{\partial\theta_{pj}}\left\{\frac{r_{0j}\sin(\omega+\theta_{0j})}{1-2r_{0j}\cos(\omega+\theta_{0j})+r_{0j}^2} - \frac{r_{0j}\sin(\omega-\theta_{0j})}{1-2r_{0j}\cos(\omega-\theta_{0j})+r_{0j}^2}\right\}$$

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

$$\begin{aligned}
\frac{\partial^2 A(\omega)}{\partial r_{pj}^2} &= -s\frac{\partial A(\omega)}{\partial r_{pj}}\sum_{i=0}^{R-1}\left\{\frac{N_n}{D_n}+\frac{N_p}{D_p}\right\} - \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_p r_{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_n r_{pj}}{D_n^2}\right]\right\} \\
\frac{\partial^2 A(\omega)}{\partial\theta_{pk}\partial r_{pj}} &= -s\frac{\partial A(\omega)}{\partial\theta_{pk}}\sum_{i=0}^{R-1}\left\{\frac{N_n}{D_n}+\frac{N_p}{D_p}\right\} \\
\frac{\partial^2 A(\omega)}{\partial\theta_{pj}^2} &= -s\frac{\partial A(\omega)}{\partial\theta_{pj}}\sum_{i=0}^{R-1}\left\{\frac{r_{pj}^s\sin(\omega+s(\theta_{pj}+2\pi i))}{D_p} - \frac{r_{pj}^s\sin(\omega-s(\theta_{pj}+2\pi i))}{D_n}\right\} - \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\}
\end{aligned}$$

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

The non-zero second partial derivatives of the phase response are:

$$\begin{aligned}
\frac{\partial^2 P(\omega)}{\partial R_{0j}^2} &= \frac{\partial P(\omega)}{\partial R_{0j}}\frac{2\cos\omega - 2R_{0j}}{1-2R_{0j}\cos\omega + R_{0j}^2} \\
\frac{\partial^2 P(\omega)}{\partial R_{pj}^2} &= -s(s-1)R_{pj}^{s-2}\sum_{i=0}^{R-1}\frac{\sin(\omega-s2\pi i)}{1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}}\cdots \\
&\quad - 2s^2R_{pj}^{2s-2}\sum_{i=0}^{R-1}\frac{\sin(\omega-s2\pi i)\left[\cos(\omega-s2\pi i)-R_{pj}^s\right]}{\left[1-2R_{pj}^s\cos(\omega-s2\pi i)+R_{pj}^{2s}\right]^2}
\end{aligned}$$

For convenience, write:

$$\frac{\partial^2 P_{0N}(\omega)}{\partial r_{0j}^2} = 0$$
$$\frac{\partial^2 P_{0N}(\omega)}{\partial r_{0j}\theta_{0j}} = 2\sin\theta_{0j}\sin\omega$$
$$\frac{\partial^2 P_{0N}(\omega)}{\partial \theta_{0j}^2} = 2r_{0j}\cos\theta_{0j}\sin\omega$$
$$\frac{\partial^2 P_{0D}(\omega)}{\partial r_{0j}^2} = 2$$
$$\frac{\partial^2 P_{0D}(\omega)}{\partial r_{0j}\theta_{0j}} = 2\sin\theta_{0j}\cos\omega$$
$$\frac{\partial^2 P_{0D}(\omega)}{\partial \theta_{0j}^2} = 2r_{0j}\cos\theta_{0j}\cos\omega$$

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:

$$\frac{\partial^2 P_{pN}(\omega)}{\partial r_{pj}^2} = -2s(s-1)r_{pj}^{s-2}\cos\left[s(\theta_{pj} + 2\pi i)\right]\sin\omega$$
$$\frac{\partial^2 P_{pN}(\omega)}{\partial r_{pj}\theta_{pj}} = 2s^2 r_{pj}^{s-1}\sin\left[s(\theta_{pj} + 2\pi i)\right]\sin\omega$$
$$\frac{\partial^2 P_{pN}(\omega)}{\partial \theta_{pj}^2} = 2s^2 r_{pj}^{s}\cos\left[s(\theta_{pj} + 2\pi i)\right]\sin\omega$$
$$\frac{\partial^2 P_{pD}(\omega)}{\partial r_{pj}^2} = -2s(s-1)r_{pj}^{s-2}\cos\left[s(\theta_{pj} + 2\pi i)\right]\cos\omega + 2s(2s-1)r_{pj}^{2s-2}$$
$$\frac{\partial^2 P_{pD}(\omega)}{\partial r_{pj}\theta_{pj}} = 2s^2 r_{pj}^{s-1}\sin\left[s(\theta_{pj} + 2\pi i)\right]\cos\omega$$
$$\frac{\partial^2 P_{pD}(\omega)}{\partial \theta_{pj}^2} = 2s^2 r_{pj}^{s}\cos\left[s(\theta_{pj} + 2\pi i)\right]\cos\omega$$

then:

$$\begin{aligned}
\frac{\partial^2 P(\omega)}{\partial r_{pj}^2} = -\sum_{i=0}^{R-1}\Bigg\{& \frac{\frac{\partial P_{pD}}{\partial r_{pj}}\frac{\partial P_{pN}}{\partial r_{pj}} + P_{pD}\frac{\partial^2 P_{pN}}{\partial r_{pj}^2} - \frac{\partial P_{pN}}{\partial r_{pj}}\frac{\partial P_{pD}}{\partial r_{pj}} - P_{pN}\frac{\partial^2 P_{pD}}{\partial r_{pj}^2}}{P_{pN}^2 + P_{pD}^2}\cdots \\
&- \frac{\left[P_{pD}\frac{\partial P_{pN}}{\partial r_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial r_{pj}}\right]\left[2P_{pN}\frac{\partial P_{pN}}{\partial r_{pj}} + 2P_{pD}\frac{\partial P_{pD}}{\partial r_{pj}}\right]}{\left[P_{pN}^2 + P_{pD}^2\right]^2}\Bigg\}
\end{aligned}$$

$$\frac{\partial^2 P(\omega)}{\partial r_{pj}\partial\theta_{pj}} = -\sum_{i=0}^{R-1}\left\{\frac{\frac{\partial P_{pD}}{\partial\theta_{pj}}\frac{\partial P_{pN}}{\partial r_{pj}} + P_{pD}\frac{\partial^2 P_{pN}}{\partial r_{pj}\partial\theta_{pj}} - \frac{\partial P_{pN}}{\partial\theta_{pj}}\frac{\partial P_{pD}}{\partial r_{pj}} - P_{pN}\frac{\partial^2 P_{pD}}{\partial r_{pj}\partial\theta_{pj}}}{P_{pN}^2 + P_{pD}^2}\cdots\right.$$
$$\left.-\frac{\left[P_{pD}\frac{\partial P_{pN}}{\partial r_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial r_{pj}}\right]\left[2P_{pN}\frac{\partial P_{pN}}{\partial\theta_{pj}} + 2P_{pD}\frac{\partial P_{pD}}{\partial\theta_{pj}}\right]}{\left[P_{pN}^2 + P_{pD}^2\right]^2}\right\}$$

$$\frac{\partial^2 P(\omega)}{\partial\theta_{pj}^2} = -\sum_{i=0}^{R-1}\left\{\frac{\frac{\partial P_{pD}}{\partial\theta_{pj}}\frac{\partial P_{pN}}{\partial\theta_{pj}} + P_{pD}\frac{\partial^2 P_{pN}}{\partial\theta_{pj}^2} - \frac{\partial P_{pN}}{\partial\theta_{pj}}\frac{\partial P_{pD}}{\partial\theta_{pj}} - P_{pN}\frac{\partial^2 P_{pD}}{\partial\theta_{pj}^2}}{P_{pN}^2 + P_{pD}^2}\cdots\right.$$
$$\left.-\frac{\left[P_{pD}\frac{\partial P_{pN}}{\partial\theta_{pj}} - P_{pN}\frac{\partial P_{pD}}{\partial\theta_{pj}}\right]\left[2P_{pN}\frac{\partial P_{pN}}{\partial\theta_{pj}} + 2P_{pD}\frac{\partial P_{pD}}{\partial\theta_{pj}}\right]}{\left[P_{pN}^2 + P_{pD}^2\right]^2}\right\}$$

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:

$$\frac{\partial^2 T(\omega)}{\partial R_{0j}^2} = \frac{2\left(R_{0j}^3\cos\omega - 3R_{0j}^2 + 3R_{0j}\cos\omega - \cos 2\omega\right)}{\left[1 - 2R_{0j}\cos\omega + R_{0j}^2\right]^3}$$

$$\frac{\partial^2 T(\omega)}{\partial R_{pj}^2} = -sR_{pj}^{s-2}\sum_{i=0}^{R-1}\left\{\frac{(s+1)\cos(\omega - s2\pi i)R_{pj}^{4s} + 2\left[(s-1)\cos^2(\omega - s2\pi i) - (2s+1)\right]R_{pj}^{3s}}{\left[1 - 2R_{pj}^s\cos(\omega - s2\pi i) + R_{pj}^{2s}\right]^3} + \cdots\right.$$
$$\left.\cdots\frac{6\cos(\omega - s2\pi i)R_{pj}^{2s} + 2\left[(2s-1) - (s+1)\cos^2(\omega - s2\pi i)\right]R_{pj}^s - (s-1)\cos(\omega - s2\pi i)}{\left[1 - 2R_{pj}^s\cos(\omega - s2\pi i) + R_{pj}^{2s}\right]^3}\right\}$$

$$\frac{\partial^2 T(\omega)}{\partial r_{0j}^2} = \frac{2\left[r_{0j}^3\cos(\omega - \theta_{0j}) - 3r_{0j}^2 + 3r_{0j}\cos(\omega - \theta_{0j}) - \cos 2(\omega - \theta_{0j})\right]}{\left[1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right]^3} + \cdots$$
$$\frac{2\left[r_{0j}^3\cos(\omega + \theta_{0j}) - 3r_{0j}^2 + 3r_{0j}\cos(\omega + \theta_{0j}) - \cos 2(\omega + \theta_{0j})\right]}{\left[1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right]^3}$$

$$\frac{\partial^2 T(\omega)}{\partial\theta_{0j}\partial r_{0j}} = \frac{\sin(\omega + \theta_{0j})\left[r_{0j}^4 + 2r_{0j}^3\cos(\omega + \theta_{0j}) - 6r_{0j}^2 + 2r_{0j}\cos(\omega + \theta_{0j}) + 1\right]}{\left[1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right]^3} - \cdots$$
$$\frac{\sin(\omega - \theta_{0j})\left[r_{0j}^4 + 2r_{0j}^3\cos(\omega - \theta_{0j}) - 6r_{0j}^2 + 2r_{0j}\cos(\omega - \theta_{0j}) + 1\right]}{\left[1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right]^3}$$

$$\frac{\partial^2 T(\omega)}{\partial\theta_{0j}^2} = -\frac{r_{0j}\left(r_{0j}^2 - 1\right)\left[r_{0j}^2\cos(\omega - \theta_{0j}) - 2r_{0j}\left(1 + \sin^2(\omega - \theta_{0j})\right) + \cos(\omega - \theta_{0j})\right]}{\left[1 - 2r_{0j}\cos(\omega - \theta_{0j}) + r_{0j}^2\right]^3} - \cdots$$
$$\frac{r_{0j}\left(r_{0j}^2 - 1\right)\left[r_{0j}^2\cos(\omega + \theta_{0j}) - 2r_{0j}\left(1 + \sin^2(\omega + \theta_{0j})\right) + \cos(\omega + \theta_{0j})\right]}{\left[1 - 2r_{0j}\cos(\omega + \theta_{0j}) + r_{0j}^2\right]^3}$$

$$\frac{\partial^2 T(\omega)}{\partial r_{pj}^2} = -s\sum_{i=1}^{R-1} r_{pj}^{s-2}\left\{\frac{2r_{pj}^s\left(s\left(r_{pj}^{2s} - 1\right) - \left(r_{pj}^{2s} + 1\right)\right)\cos^2(\omega + s(\theta_{pj} + 2\pi i))}{D_p^3} + \cdots\right.$$
$$\cdots\frac{\left(s\left(r_{pj}^{4s} - 1\right) + \left(r_{pj}^{4s} + 6r_{pj}^{2s} + 1\right)\right)\cos(\omega + s(\theta_{pj} + 2\pi i)) - 4sr_{pj}^s\left(r_{pj}^{2s} - 1\right) - 2r_{pj}^s\left(r_{pj}^{2s} + 1\right)}{D_p^3} + \cdots$$
$$\cdots\frac{2r_{pj}^s\left(s\left(r_{pj}^{2s} - 1\right) - \left(r_{pj}^{2s} + 1\right)\right)\cos^2(\omega - s(\theta_{pj} + 2\pi i))}{D_n^3} + \cdots$$
$$\left.\cdots\frac{\left(s\left(r_{pj}^{4s} - 1\right) + \left(r_{pj}^{4s} + 6r_{pj}^{2s} + 1\right)\right)\cos(\omega - s(\theta_{pj} + 2\pi i)) - 4sr_{pj}^s\left(r_{pj}^{2s} - 1\right) - 2r_{pj}^s\left(r_{pj}^{2s} + 1\right)}{D_n^3}\right\}$$

$$\frac{\partial^2 T(\omega)}{\partial\theta_{pj}\partial r_{pj}} = s^2\sum_{i=1}^{R-1} r_{pj}^{s-1}\left\{\frac{\left[2r_{pj}^s\left(r_{pj}^{2s} + 1\right)\cos(\omega - s(\theta_{pj} + 2\pi i)) + \left(r_{pj}^{4s} - 6r_{pj}^{2s} + 1\right)\right]\sin(\omega - s(\theta_{pj} + 2\pi i))}{D_n^3} - \cdots\right.$$
$$\left.\cdots\frac{\left[2r_{pj}^s\left(r_{pj}^{2s} + 1\right)\cos(\omega + s(\theta_{pj} + 2\pi i)) + \left(r_{pj}^{4s} - 6r_{pj}^{2s} + 1\right)\right]\sin(\omega + s(\theta_{pj} + 2\pi i))}{D_p^3}\right\}$$

$$\frac{\partial^2 T(\omega)}{\partial \theta_{pj}^2} = s^2 \sum_{i=1}^{R-1} r_{pj}^s \left(r_{pj}^{2s} - 1\right) \left\{ \frac{2r_{pj}^s \cos^2\left(\omega + s\left(\theta_{pj} + 2\pi i\right)\right) + \left(r_{pj}^{2s} + 1\right) \cos\left(\omega + s\left(\theta_{pj} + 2\pi i\right)\right) - 4r_{pj}^s}{D_p^3} + \cdots \right.$$
$$\left. \cdots \frac{2r_{pj}^s \cos^2\left(\omega - s\left(\theta_{pj} + 2\pi i\right)\right) + \left(r_{pj}^{2s} + 1\right) \cos\left(\omega - s\left(\theta_{pj} + 2\pi i\right)\right) - 4r_{pj}^s}{D_n^3} \right\}$$

The results for group delay were calculated with the assistance of the *Maxima* [253] script *delay.max*.

G.4 Octave implementations

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 [165] 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:

$$\begin{aligned} \frac{\partial P(\omega)}{\partial R_{pk}} &= -\frac{2 \sin R\omega}{R_{pk}^2 - 2R_{pk} \cos R\omega + 1} \\ \frac{\partial P(\omega)}{\partial r_{pk}} &= -\frac{2 \sin\left(R\omega - \theta_{pk}\right)}{r_{pk}^2 - 2r_{pk} \cos\left(R\omega - \theta_{pk}\right) + 1} - \frac{2 \sin\left(R\omega + \theta_{pk}\right)}{r_{pk}^2 - 2r_{pk} \cos\left(R\omega + \theta_{pk}\right) + 1} \\ \frac{\partial P(\omega)}{\partial \theta_{pk}} &= -\frac{2r_{pk}^2 - 2r_{pk} \cos\left(R\omega - \theta_{pk}\right)}{r_{pk}^2 - 2r_{pk} \cos\left(R\omega - \theta_{pk}\right) + 1} + \frac{2r_{pk}^2 - 2r_{pk} \cos\left(R\omega + \theta_{pk}\right)}{r_{pk}^2 - 2r_{pk} \cos\left(R\omega + \theta_{pk}\right) + 1} \end{aligned}$$

The second partial derivatives of the phase response (on the diagonal of the Hessian matrix only) are:

$$\frac{\partial^2 P(\omega)}{\partial R_{pk}^2} = \frac{4\left[R_{pk} - \cos R\omega\right] \sin R\omega}{\left[R_{pk}^2 - 2R_{pk} \cos R\omega + 1\right]^2}$$

$$\frac{\partial^2 P(\omega)}{\partial r_{pk}^2} = \frac{4\left[r_{pk} - \cos(R\omega - \theta_{pk})\right]\sin(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^2} + \frac{4\left[r_{pk} - \cos(R\omega + \theta_{pk})\right]\sin(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^2}$$

$$\begin{aligned}\frac{\partial^2 P(\omega)}{\partial \theta_{pk}^2} = & \frac{2r_{pk}\sin(R\omega - \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1} - \frac{4\left[r_{pk}^2 - r_{pk}\cos(R\omega - \theta_{pk})\right] r_{pk}\sin(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^2} \cdots \\ & + \frac{2r_{pk}\sin(R\omega + \theta_{pk})}{r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1} - \frac{4\left[r_{pk}^2 - r_{pk}\cos(R\omega + \theta_{pk})\right] r_{pk}\sin(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^2}\end{aligned}$$

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:

$$\frac{\partial^2 T(\omega)}{\partial R_{pk}^2} = -4R\frac{R_{pk}^3\cos R\omega - 3R_{pk}^2 + 3R_{pk}\cos R\omega + 1 - 2\cos^2 R\omega}{\left[R_{pk}^2 - 2R_{pk}\cos R\omega + 1\right]^3}$$

$$\begin{aligned}\frac{\partial^2 T(\omega)}{\partial r_{pk}^2} = & -4R\frac{r_{pk}^3\cos(R\omega + \theta_{pk}) - 3r_{pk}^2 + 3r_{pk}\cos(R\omega + \theta_{pk}) + 1 - 2\cos^2(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^3}\cdots \\ & -4R\frac{r_{pk}^3\cos(R\omega - \theta_{pk}) - 3r_{pk}^2 + 3r_{pk}\cos(R\omega - \theta_{pk}) + 1 - 2\cos^2(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^3}\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 T(\omega)}{\partial \theta_{pk}^2} = & \ 2R\frac{r_{pk}^5\cos(R\omega + \theta_{pk}) - 2r_{pk}^2\left(r_{pk}^2 - 1\right)\left[1 + \sin^2(R\omega + \theta_{pk})\right] - r_{pk}\cos(R\omega + \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega + \theta_{pk}) + 1\right]^3}\cdots \\ & + 2R\frac{r_{pk}^5\cos(R\omega - \theta_{pk}) - 2r_{pk}^2\left(r_{pk}^2 - 1\right)\left[1 + \sin^2(R\omega - \theta_{pk})\right] - r_{pk}\cos(R\omega - \theta_{pk})}{\left[r_{pk}^2 - 2r_{pk}\cos(R\omega - \theta_{pk}) + 1\right]^3}\end{aligned}$$

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 digital 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 digital state variable filter. This section shows results for the gradients of the squared-magnitude, phase and group delay of a state variable filter. In the following, the components of the state variable filter matrixes A, B, C and D are represented by α, β, γ and δ or more generally, x. The matrix components may themselves be functions of other variables (the k and c coefficients of a Schur lattice filter, for example). In the following, the *resolvent* matrix, R, is defined on the unit circle as $R = (e^{\imath\omega} I - A)^{-1}$. The identities shown in Equation 1.14 apply with:

$$\frac{\partial R}{\partial \omega} = -\imath e^{\imath\omega} RR$$

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

If the gradients of the state variable matrixes, A, B, C and D, with respect to a coefficient, x, are $\frac{dA}{dx}$, $\frac{dB}{dx}$, $\frac{dC}{dx}$ and $\frac{dD}{dx}$, then the gradients of the complex frequency response, $H(\omega)$, are:

$$\begin{aligned}
\frac{\partial H}{\partial \omega} &= -\imath e^{\imath\omega} CRRB \\
\frac{\partial H}{\partial x} &= \frac{dC}{dx} RB + CR\frac{dA}{dx} RB + CR\frac{dB}{dx} + \frac{dD}{dx} \\
\frac{\partial^2 H}{\partial x \partial \omega} &= -\imath e^{\imath\omega} \left[\frac{dC}{dx} RRB + CRR\frac{dA}{dx} RB + CR\frac{dA}{dx} RRB + CRR\frac{dB}{dx} \right] \\
\frac{\partial^2 H}{\partial x^2} &= \frac{d^2C}{dx^2} RB + 2\frac{dC}{dx} R\frac{dA}{dx} RB + 2\frac{dC}{dx} R\frac{dB}{dx} + 2CR\frac{dA}{dx} R\frac{dA}{dx} RB + \cdots \\
& \quad CR\frac{d^2A}{dx^2} RB + 2CR\frac{dA}{dx} R\frac{dB}{dx} + CR\frac{d^2B}{dx^2} + \frac{d^2D}{dx^2} \\
\frac{\partial^3 H}{\partial \omega \partial x^2} &= -\imath e^{\imath\omega} \left[\frac{d^2C}{dx^2} RRB + 2\frac{dC}{dx} R\frac{dA}{dx} RRB + 2\frac{dC}{dx} RR\frac{dA}{dx} RB + 2\frac{dC}{dx} RR\frac{dB}{dx} + \cdots \right. \\
& \quad 2CR\frac{dA}{dx} R\frac{dA}{dx} RRB + CR\frac{d^2A}{dx^2} RRB + 2CR\frac{dA}{dx} RR\frac{dA}{dx} RB + 2CR\frac{dA}{dx} RR\frac{dB}{dx} + \cdots \\
& \quad \left. 2CRR\frac{dA}{dx} R\frac{dA}{dx} RB + CRR\frac{d^2A}{dx^2} RB + 2CRR\frac{dA}{dx} R\frac{dB}{dx} + CRR\frac{d^2B}{dx^2} \right] \\
\frac{\partial^2 H}{\partial y \partial x} &= \frac{d^2C}{dydx} RB + \frac{dC}{dx} R\frac{dA}{dy} RB + \frac{dC}{dx} R\frac{dB}{dy} + \cdots \\
& \quad \frac{dC}{dy} R\frac{dA}{dx} RB + CR\frac{dA}{dy} R\frac{dA}{dx} RB + CR\frac{d^2A}{dydx} RB + CR\frac{dA}{dx} R\frac{dA}{dy} RB + CR\frac{dA}{dx} R\frac{dB}{dy} + \cdots \\
& \quad \frac{dC}{dy} R\frac{dB}{dx} + CR\frac{dA}{dy} R\frac{dB}{dx} + CR\frac{d^2B}{dydx} + \frac{d^2D}{dydx}
\end{aligned}$$

$$\frac{\partial^3 H}{\partial\omega\partial y\partial x} = -\imath e^{\imath\omega}\left[\frac{d^2C}{dydx}RRB + \frac{dC}{dx}RR\frac{dA}{dy}RB + \frac{dC}{dx}R\frac{dA}{dy}RRB + \frac{dC}{dx}RR\frac{dB}{dy} + \frac{dC}{dy}RR\frac{dA}{dx}RB + \frac{dC}{dy}R\frac{dA}{dx}RRB + \cdots\right.$$
$$CRR\frac{dA}{dy}R\frac{dA}{dx}RB + CR\frac{dA}{dy}RR\frac{dA}{dx}RB + CR\frac{dA}{dy}R\frac{dA}{dx}RRB + \cdots$$
$$CRR\frac{d^2A}{dydx}RB + CR\frac{d^2A}{dydx}RRB + \cdots$$
$$CRR\frac{dA}{dx}R\frac{dA}{dy}RB + CR\frac{dA}{dx}RR\frac{dA}{dy}RB + CR\frac{dA}{dx}R\frac{dA}{dy}RRB + \cdots$$
$$\left.CRR\frac{dA}{dx}R\frac{dB}{dy} + CR\frac{dA}{dx}RR\frac{dB}{dy} + \frac{dC}{dy}RR\frac{dB}{dx} + CRR\frac{dA}{dy}R\frac{dB}{dx} + CR\frac{dA}{dy}RR\frac{dB}{dx} + CRR\frac{d^2B}{dydx}\right]$$

The Octave function *Abcd2H* calculates these gradients under the assumption that the components of the state variable matrixes A, B, C and D are first-order polynomial functions of coefficients x or y and $\frac{d^2A}{dx^2} = \frac{d^2B}{dx^2} = \frac{d^2C}{dx^2} = \frac{d^2D}{dx^2} = 0$. This assumption holds for the Schur one-multiplier lattice filter.

J.2 State variable filter squared-magnitude response

The squared-magnitude response of a state variable filter is:

$$|H|^2 = \Im H^2 + \Re H^2$$

The gradients of the squared-magnitude response of the filter with respect to the state variable coefficients are:

$$\frac{\partial\,|H|^2}{\partial x} = 2\left[\Im H\Im\frac{\partial H}{\partial x} + \Re H\Re\frac{\partial H}{\partial x}\right]$$

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:

$$\frac{\partial^2\,|H|^2}{\partial x^2} = 2\left[\left|\frac{\partial H}{\partial x}\right|^2 + \Im H\Im\frac{\partial^2 H}{\partial x^2} + \Re H\Re\frac{\partial^2 H}{\partial x^2}\right]$$

If A is a first-order polynomial function of the state variable matrix coefficients, α, then:

$$\frac{\partial^2 H}{\partial\alpha^2} = 2CR\frac{dA}{d\alpha}R\frac{dA}{d\alpha}RB$$

Similarly, if B is a first-order polynomial function of the state variable matrix coefficients, β, and-so-on, then:

$$\frac{\partial^2 H}{\partial\beta^2} = \frac{\partial^2 H}{\partial\gamma^2} = \frac{\partial^2 H}{\partial\delta^2} = 0$$

The Hessian matrix of the squared-magnitude (that is the second derivatives of the squared-magnitude) is:

$$\frac{\partial^2\,|H|^2}{\partial y\partial x} = 2\left[\Im\frac{\partial H}{\partial y}\Im\frac{\partial H}{\partial x} + \Im H\Im\frac{\partial^2 H}{\partial y\partial x} + \Re\frac{\partial H}{\partial y}\Re\frac{\partial H}{\partial x} + \Re H\Re\frac{\partial^2 H}{\partial y\partial x}\right]$$

where x and y represent the state variable matrix coefficients α, β, etc.

The gradient of the squared-magnitude response of the filter with respect to the angular frequency is:

$$\frac{\partial\,|H|^2}{\partial\omega} = 2\left[\Im H\Im\frac{\partial H}{\partial\omega} + \Re H\Re\frac{\partial H}{\partial\omega}\right]$$

and with respect to the coefficients, x:

$$\frac{\partial^2\,|H|^2}{\partial\omega\partial x} = 2\left[\Im\frac{\partial H}{\partial x}\Im\frac{\partial H}{\partial\omega} + \Im H\Im\frac{\partial^2 H}{\partial\omega\partial x} + \Re\frac{\partial H}{\partial x}\Re\frac{\partial H}{\partial\omega} + \Re H\Re\frac{\partial^2 H}{\partial\omega\partial x}\right]$$

The Hessian matrix of the gradient of the squared-magnitude with respect to angular frequency is:

$$\frac{\partial^3\,|H|^2}{\partial\omega\partial y\partial x} = 2\left[\Im\frac{\partial^2 H}{\partial\omega\partial y\partial x}\Im\frac{\partial H}{\partial w} + \Im\frac{\partial H}{\partial x}\Im\frac{\partial^2 H}{\partial\omega\partial y} + \Im\frac{\partial H}{\partial y}\Im\frac{\partial^2 H}{\partial\omega\partial x} + \Im H\Im\frac{\partial^3 H}{\partial\omega\partial y\partial x} + \cdots\right.$$
$$\left.\Re\frac{\partial^2 H}{\partial y\partial x}\Re\frac{\partial H}{\partial w} + \Re\frac{\partial H}{\partial x}\Re\frac{\partial^2 H}{\partial\omega\partial y} + \Re\frac{\partial H}{\partial y}\Re\frac{\partial^2 H}{\partial\omega\partial x} + \Re H\Re\frac{\partial^3 H}{\partial\omega\partial y\partial x}\right]$$

J.3 State variable filter phase response

The phase response, P, of the state variable filter is:

$$P = \arctan \frac{\Im H}{\Re H}$$

The gradients of the phase response of the filter with respect to the state variable coefficients, α, β, etc., are given by:

$$|H|^2 \frac{\partial P}{\partial x} = \Re H \Im \frac{\partial H}{\partial x} - \Im H \Re \frac{\partial H}{\partial x}$$

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:

$$\frac{\partial |H|^2}{\partial x} \frac{\partial P}{\partial x} + |H|^2 \frac{\partial^2 P}{\partial x^2} = \Re H \Im \frac{\partial^2 H}{\partial x^2} - \Im H \Re \frac{\partial^2 H}{\partial x^2}$$

The Hessian matrix of the phase (that is the second derivatives of the phase) is given by:

$$\frac{\partial |H|^2}{\partial y} \frac{\partial P}{\partial x} + |H|^2 \frac{\partial^2 P}{\partial y \partial x} = \Re \frac{\partial H}{\partial y} \Im \frac{\partial H}{\partial x} + \Re H \Im \frac{\partial^2 H}{\partial y \partial x} - \Im \frac{\partial H}{\partial y} \Re \frac{\partial H}{\partial x} - \Im H \Re \frac{\partial^2 H}{\partial y \partial x}$$

J.4 State variable filter group-delay response

The group delay response, T, of the state variable filter is

$$T = -\frac{\partial}{\partial \omega} \arctan \frac{\Im H}{\Re H}$$

so that

$$|H|^2 T = -\left[\Re H \Im \frac{\partial H}{\partial \omega} - \Im H \Re \frac{\partial H}{\partial \omega} \right]$$

The gradients of the group delay response with respect to the state variable coefficients, α, β, etc., are given by:

$$\frac{\partial |H|^2}{\partial x} T + |H|^2 \frac{\partial T}{\partial x} = -\left[\Re \frac{\partial H}{\partial x} \Im \frac{\partial H}{\partial \omega} + \Re H \Im \frac{\partial^2 H}{\partial x \partial \omega} - \Im \frac{\partial H}{\partial x} \Re \frac{\partial H}{\partial \omega} - \Im H \Re \frac{\partial^2 H}{\partial x \partial \omega} \right]$$

The diagonal of the Hessian matrix of the group-delay (that is the second derivatives of the group-delay) is given by:

$$\begin{aligned} \frac{\partial^2 |H|^2}{\partial x^2} T + 2 \frac{\partial |H|^2}{\partial x} \frac{\partial T}{\partial x} + |H|^2 \frac{\partial^2 T}{\partial x^2} = & - \Re \frac{\partial^2 H}{\partial x^2} \Im \frac{\partial H}{\partial \omega} - \Re \frac{\partial H}{\partial x} \Im \frac{\partial^2 H}{\partial x \partial \omega} - \Re \frac{\partial H}{\partial x} \Im \frac{\partial^2 H}{\partial x \partial \omega} - \Re H \Im \frac{\partial^3 H}{\partial x^2 \partial \omega} \cdots \\ & + \Im \frac{\partial^2 H}{\partial x^2} \Re \frac{\partial H}{\partial \omega} + \Im \frac{\partial H}{\partial x} \Re \frac{\partial^2 H}{\partial x \partial \omega} + \Im \frac{\partial H}{\partial x} \Re \frac{\partial^2 H}{\partial x \partial \omega} + \Im H \Re \frac{\partial^3 H}{\partial \omega \partial x^2} \end{aligned}$$

The Hessian matrix of the group delay (that is the second derivatives of the group delay) is given by:

$$\begin{aligned} & \frac{\partial^2 |H|^2}{\partial y \partial x} T + \frac{\partial |H|^2}{\partial x} \frac{\partial T}{\partial y} + \frac{\partial |H|^2}{\partial y} \frac{\partial T}{\partial x} + |H|^2 \frac{\partial^2 T}{\partial y \partial x} = \cdots \\ & - \Bigg[\Re \frac{\partial^2 H}{\partial y \partial x} \Im \frac{\partial H}{\partial \omega} + \Re \frac{\partial H}{\partial x} \Im \frac{\partial^2 H}{\partial \omega \partial y} + \Re \frac{\partial H}{\partial y} \Im \frac{\partial^2 H}{\partial \omega \partial x} + \Re H \Im \frac{\partial^3 H}{\partial \omega \partial y \partial x} \cdots \\ & \quad - \Im \frac{\partial^2 H}{\partial y \partial x} \Re \frac{\partial H}{\partial \omega} - \Im \frac{\partial H}{\partial x} \Re \frac{\partial^2 H}{\partial \omega \partial y} - \Im \frac{\partial H}{\partial y} \Re \frac{\partial^2 H}{\partial \omega \partial x} - \Im H \Re \frac{\partial^3 H}{\partial \omega \partial y \partial x} \Bigg] \end{aligned}$$

Appendix K

Constrained non-linear optimisation

The following largely follows *Ruszczynski* [9] with some contributions concerning heuristics from *Powell* [140] and *Nocedal* and *Wright* [112].

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

$$f^k(x) = f\left(x^k\right) + \nabla_x f\left(x^k\right)\left(x - x^k\right) + \frac{1}{2}\left(x - x^k\right)^\top \nabla_x^2 f\left(x^k\right)\left(x - x^k\right)$$

where $\nabla_x^2 f\left(x^k\right)$ 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 $f^k(x)$:

$$\nabla_x f\left(x^k\right) + \nabla_x^2 f\left(x^k\right)\left(x - x^k\right) = 0$$

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

$$x^{k+1} = x^k - \tau^k\left[\nabla_x^2 f\left(x^k\right)\right]^{-1}\nabla_x f\left(x^k\right)$$

where τ^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* [112, p. 309] for a justification). The method of *Lagrange multipliers* defines an auxiliary function (or *Lagrangian*):

$$\mathcal{L}(x, \lambda) = f(x) - \lambda(g(x) - c)$$

and solves

$$\nabla_{x,\lambda}\mathcal{L}(x, \lambda) = 0$$

Note that the solution may be a saddle point of $\mathcal{L}(x, \lambda)$. The coefficients of the vector λ are the *Lagrange multipliers*.

[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* [112, 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 [44, 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* [44, 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 optimisation 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 $\bar{x}$:

$$\begin{aligned} \nabla_x \mathcal{L}\left(\overline{x}, \overline{\lambda}, \overline{\mu}\right) + \nabla_x^2 \mathcal{L}\left(\overline{x}, \overline{\lambda}, \overline{\mu}\right)\left(x - \overline{x}\right) &= 0 \\ \begin{bmatrix} g\left(\overline{x}\right) + \nabla_x g\left(\overline{x}\right)\left(x - \overline{x}\right) \\ h\left(\overline{x}\right) + \nabla_x h\left(\overline{x}\right)\left(x - \overline{x}\right) \end{bmatrix} &\in N_C\left(\lambda, \mu\right) \end{aligned}$$

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:

$$\begin{aligned} \textbf{minimise} \quad & \left\langle \nabla_x f\left(x^k\right), d^k \right\rangle + \frac{1}{2}\left\langle {d^k}^\top, \nabla_x^2 \mathcal{L}\left(x^k, \lambda^k, \mu^k\right), d^k \right\rangle \\ \textbf{subject to} \quad & g\left(x^k\right) + \nabla_x g\left(x^k\right) d^k \leq 0 \\ & h\left(x^k\right) + \nabla_x h\left(x^k\right) d^k = 0 \end{aligned}$$

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:

$$\begin{aligned} \mathcal{H}_k &= \nabla_x^2 \mathcal{L}\left(x^k, \lambda^k\right) \\ \mathcal{B}_k &= \left\{\nabla_x g_i\left(x^k\right)\right\} \end{aligned}$$

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 λ^{k+1} and $g\left(x^k\right)$ are the vectors with coordinates λ_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 λ^{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 $\nabla_x^2 \mathcal{L}\left(x^k, \lambda^k, \mu^k\right)$ 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, τ^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 τ^k. In this case the usual methods for updating $\mathcal{W}_k$ would fail to make the update positive definite. *Powell* [140, p.148] proposes updating $\mathcal{W}_k$ as follows. First replace γ_k with

$$\eta_k = \theta_k \gamma_k + (1 - \theta_k)\, \mathcal{W}_k \delta_k \ , \quad 0 \leq \theta_k \leq 1$$

where

$$\theta_k = \begin{cases} 1 & \delta_k^\top \gamma_k \geq 0.2 \delta_k^\top \mathcal{W}_k \delta_k \\ \frac{0.8 \delta_k^\top \mathcal{W}_k \delta_k}{\delta_k^\top \mathcal{W}_k \delta_k - \delta_k^\top \gamma_k} & \delta_k^\top \gamma_k < 0.2 \delta_k^\top \mathcal{W}_k \delta_k \end{cases} \tag{K.3}$$

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 η and δ are column vectors and $\delta \eta^\top$ is a *dyadic* or *Kronecker* product also written $\delta \otimes \eta$. *Nocedal* and *Wright* [112, Procedure 18.2] refer to Equation K.3 as *damped BFGS updating*.

K.7.2 A modified Cholesky factorisation of the Hessian

Bertsekas [44, Appendix D] describes implementation of Newton's method by Cholesky factorisation of an approximation to the Hessian, $\nabla_x^2 \mathcal{L}(x^k) + \Delta_k$, that is positive definite.

Algorithm K.2 [44, Appendix D.1] 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 λ_{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:

$$0 < \begin{bmatrix} b^\top & -1 \end{bmatrix} W_i \begin{bmatrix} b \\ -1 \end{bmatrix} = b^\top W_{i-1} b - 2 b^\top \beta_i + w_{ii}$$

Algorithm K.2 Cholesky factorisation of an $n \times n$ positive-definite symmetric matrix, W [44, Appendix D.1]

$L_1 = \sqrt{w_{1,1}}$
$W_1 = L_1 L_1^\top$
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, Δ_k, to the Hessian matrix such that the resulting matrix is positive definite whilst simultaneously factoring $\nabla_x^2 \mathcal{L}\left(x^k\right) + \Delta_k$. Firstly, fix two positive scalars μ_1 and μ_2, where $\mu_1 < \mu_2$. The first column of the factor L is given by

$$
\begin{aligned}
l_{11} &= \begin{cases} \sqrt{w_{11}} & \mu_1 < w_{11} \\ \sqrt{\mu_2} & otherwise \end{cases} \\
l_{i1} &= \frac{w_{i1}}{l_{11}} \quad i = 2, \ldots, n
\end{aligned}
$$

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, Δ^k, to the Hessian, $\nabla_x^2 \mathcal{L}\left(x^k\right)$, and simultaneously obtain a Cholesky factorisation of $\nabla_x^2 \mathcal{L}\left(x^k\right) + \Delta^k$. 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 τ^k is a step size. The scalars μ_1 and μ_2 are selected as follows: at each iteration we find the maximum absolute value of the elements on the diagonal of the Hessian

$$
w^k = \max \ \mathrm{diag} \ \nabla_x^2 \mathcal{L}\left(x^k\right)
$$

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 $r_2 = \frac{r_2}{5}$.

Bertsekas justifies this correction as follows [44, p. 733]:

[d] The $LDL^\top$ factorisation of a symmetric matrix avoids taking the square-root. See [59, Algorithm 4.1.2].

Assuming fixed values of μ_1 and μ_2, the following may be verified for the modified Newton's method just described:

- The algorithm is globally convergent in the sense that every limit point of $\{x^k\}$ is a stationary point of $\mathcal{L}(x^k)$. This can be shown using ...
- For each local minimum x^* with positive definite Hessian, there exist scalars $\mu > 0$ and $\varepsilon > 0$ such that of $\mu_1 < \mu$ and $\|x^0 - x^*\| \leq \varepsilon$, then $x^k \to x^*$, $\Delta^k = 0$, and $\tau^k = 1$ for all k. In other words, if μ_1 is not chosen too large, the Hessian will never be modified near x^*, the method will be reduced to the pure form of Newton's method near x^*, and the convergence to x^* will be superlinear.

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* [59, 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

$$U^\top BV = \operatorname{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$$

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* [59, Section 5.5.4], the *pseudo-inverse* of B is defined to be $B^\dagger = V\Sigma U^\top$ 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 $B^\dagger = \left(B^\top B\right)^{-1} B^\top$, while if $m = n = \operatorname{rank}(B)$, then $B^\dagger = B^{-1}$. Typically, $B^\dagger$ is defined to be the unique matrix $X \in \mathbb{R}^{n\times m}$ that satisfies the four *Moore-Penrose conditions*:

$$\begin{aligned} BXB &= B \\ XBX &= X \\ (BX)^\top &= BX \\ (XB)^\top &= XB \end{aligned} \tag{K.4}$$

These conditions amount to the requirement that $BB^\dagger$ and $B^\dagger B$ be orthogonal projections onto $\operatorname{range}(B)$ and $\operatorname{range}\left(B^\top\right)$ respectively[f].

Wright [217] points out that the existence of linearly dependent constraint gradients can interfere with the super-linear convergence of the sequential quadratic programming problem (SQP) and provides a stabilised algorithm for which super-linear convergence is still possible. Given a set of active constraints $\mathcal{A}(x) \subset \mathcal{K}$ at a point x and the complement $\mathcal{N}(x) = \mathcal{K} \setminus \mathcal{A}(x)$, *Wright* defines $g_{\mathcal{A}}(x) = \{g_i(x) \mid i \in \mathcal{A}(x)\}$, $g_{\mathcal{N}}(x) = \{g_i(x) \mid i \in \mathcal{N}(x)\}$, $\lambda_{\mathcal{A}}(x) = \{\lambda_i(x) \mid i \in \mathcal{A}(x)\}$, $\lambda_{\mathcal{N}}(x) = \{\lambda_i(x) \mid i \in \mathcal{N}(x)\}$. *Wright* considers the usual SQP problem and makes the following assumptions

1. a primal-dual solution point $(\tilde{x}, \tilde{\lambda})$ exists
2. for at least one of the active constraints, $\tilde{\lambda}_i > 0$ where $i \in \mathcal{A}(\tilde{x})$
3. $\tilde{\lambda}_i = 0 \; \forall i \in \mathcal{N}(\tilde{x})$
4. there exists a $\sigma > 0$ such that $w^\top \nabla_x \mathcal{L}(\tilde{x}, \tilde{\lambda})\, w \geq \sigma\|w\|^2$ for all $\tilde{\lambda}$ such that $(\tilde{x}, \tilde{\lambda})$ satisfies the *Karush-Kuhn-Tucker* conditions and all $w \in \operatorname{null}(\nabla g_{\mathcal{A}}(\tilde{x}))$
5. $\nabla g_{\mathcal{A}}(\tilde{x})\, d < 0$ for some $d \in \mathbb{R}^n$

[e] See Appendix B.1.
[f] See Appendix B.3.

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* [217] 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 $\nabla_x \mathcal{L}(x, \lambda_{\mathcal{A}})$ may not be uniquely defined at the next SQP iteration. He proposes a stabilised version of the SQP algorithm

$$\min_x \max_{\lambda_{\mathcal{A}} \geq 0} \langle \nabla f(\bar{x}), x - \bar{x} \rangle - \left\langle \lambda_{\mathcal{A}}, g(\bar{x}) + \nabla_x^\top g(\bar{x})(x - \bar{x}) \right\rangle + \frac{1}{2} \left\langle x - \bar{x}, \left[\nabla_x^2 \mathcal{L}\left(\bar{x}, \bar{\lambda}_{\mathcal{A}}\right)\right](x - \bar{x}) \right\rangle - \frac{1}{2}\nu \|\lambda_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}\|^2$$

where ν 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

$$\begin{aligned}
\nabla_x f(\bar{x}) - \nabla g_{\mathcal{A}}(\bar{x}) \tilde{\lambda}_{\mathcal{A}} + \nabla_x \mathcal{L}\left(\bar{x}, \bar{\lambda}_{\mathcal{A}}\right)(\tilde{x} - \bar{x}) &= 0 \\
g_{\mathcal{A}}(\bar{x}) + \nabla_x^\top g_{\mathcal{A}}(\bar{x})(\tilde{x} - \bar{x}) - \nu\left(\tilde{\lambda}_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}\right) &\geq 0 \\
\tilde{\lambda}_{\mathcal{A}} &\geq 0 \\
\left\langle \tilde{\lambda}_{\mathcal{A}}, g_{\mathcal{A}}(\bar{x}) + \nabla g_{\mathcal{A}}^\top(\bar{x})(\tilde{x} - \bar{x}) - \nu\left(\tilde{\lambda}_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}}\right) \right\rangle &= 0
\end{aligned}$$

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

$$\begin{bmatrix} \nabla_x \mathcal{L}\left(\bar{x}, \bar{\lambda}_{\mathcal{A}}\right) & -\nabla_x g_{\mathcal{A}}(\bar{x}) \\ -\nabla_x g_{\mathcal{A}}^\top(\bar{x}) & \nu I \end{bmatrix} \begin{bmatrix} x - \bar{x} \\ \lambda_{\mathcal{A}} - \bar{\lambda}_{\mathcal{A}} \end{bmatrix} = \begin{bmatrix} -\left(\nabla_x f(\bar{x}) - \nabla_x g_{\mathcal{A}}(\bar{x}) \bar{\lambda}_{\mathcal{A}}\right) \\ g_{\mathcal{A}}(\bar{x}) \end{bmatrix}$$
$$\lambda_{\mathcal{N}} = 0$$

K.7.4 Bertsekas' modification to the Hessian

Bertsekas [44, 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

$$\begin{aligned}
&\textbf{minimise} && \nabla_x f\left(x^k\right) d^k + \frac{1}{2} {d^k}^\top \left(\mathcal{H}_k + c^k \mathcal{B}_k \mathcal{B}_k^\top\right) d^k \\
&\textbf{subject to} && -\mathcal{B}_k^\top d^k = g\left(x^k\right)
\end{aligned}$$

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* [44, 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 τ^k is found by a line search that minimises the objective function $f(x)$ along the ray $x^k + \tau d^k$ over $\tau \geq 0$. *Ruszczynski* [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 ϱ is larger than the "max" norm of the Lagrange multipliers $\left(\hat{\lambda}, \hat{\mu}\right)$ at the solution. Similarly, *Powell* [140, p.151] suggests using the penalty function

$$\Psi(x) \triangleq f(x) - \left[\sum_{i=1}^{m} \Omega_i \min(0, g_i(x)) + \sum_{i=1}^{p} \Upsilon_i |h_i(x)| \right]$$

and requiring that τ^k satisfies

$$\Psi\left(x^k + \tau^k d^k, \Omega, \Upsilon\right) \leq \Psi\left(x^k, \Omega, \Upsilon\right)$$

Υ_i and Ω_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 τ^k is made positive. *Powell* [140] 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 [140, 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 Ω_i used on the previous iteration. Similarly for Υ_i. The line-search procedure given by *Powell* [140, 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* [44, p. 28]. The test assumes that $\Phi'(0) < 0$. *Powell* [140, p.152] suggests that the line-search terminate when

$$\Phi(\hat{\tau}_k) \leq \Phi(0) + 0.1\tau^k \Phi'(0)$$

Bertsekas [44, 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* [44, 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 $\hat{v}_i = \max\{0, \bar{v}_i\}$ (where $\bar{v}_i \geq 0$ is the unconstrained minimum at which $\lambda_i + c^k (g_i(x) - \bar{v}_i)$ is zero) then

$$\hat{v}_i = \max\left\{0, g_i(x) + \frac{\lambda_i^k}{c^k}\right\}$$

and

$$\begin{aligned} g_i(x) - \hat{v}_i &= \begin{cases} -\frac{\lambda_i^k}{c^k} & g_i(x) > -\frac{\lambda_i^k}{c^k} \\ g_i(x) & g_i(x) \leq -\frac{\lambda_i^k}{c^k} \end{cases} \\ &= \min\left(g_i(x), -\frac{\lambda_i^k}{c^k}\right) \end{aligned}$$

The Lagrangian summation term becomes

$$\sum_{i=1}^m \lambda_i^k \min\left(g_i(x), -\frac{\lambda_i^k}{c^k}\right) + \frac{c^k}{2}\left|\min\left(g_i(x), -\frac{\lambda_i^k}{c^k}\right)\right|^2 = \begin{cases} \frac{1}{2c^k}\sum_{i=1}^m \left[\left(\lambda_i^k + c^k g_i(x)\right)^2 - \left(\lambda_i^k\right)^2\right] & g_i(x) \leq -\frac{\lambda_i^k}{c^k} \\ -\frac{1}{2c^k}\sum_{i=1}^m \left(\lambda_i^k\right)^2 & g_i(x) > -\frac{\lambda_i^k}{c^k} \end{cases}$$

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 λ^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 [44, Section 4.1], describes adding a so-called *barrier* function, $B(x)$, to the cost function. This function is continuous and goes to ∞ as any one of the constraints approaches 0 from positive values. Clearly, the barrier function is only defined at iterates, x^k, that satisfy the constraints

$$S = \{x \in X \mid g_j(x) \geq 0,\ j = 1, \ldots, m\}$$

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* [44, 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 β with τ^k and repeat the above interpolation. If $\Phi(\tau) < \Phi(0)$, then we have three points, with the best one in the middle, and we can apply a Golden Section search.

Alternatively, interpolate $\Phi(x)$. Given $\alpha_k < \tau^k < \beta_k$ such that $\Phi(\alpha_k) > \Phi(\tau^k) < \Phi(\beta_k)$ then a second-order estimate of the minimum is

$$\gamma_k = \frac{\Phi(\alpha_k)\left[\beta_k^2 - (\tau^k)^2\right] + \Phi(\tau^k)\left[\alpha_k^2 - \beta_k^2\right] + \Phi(\beta_k)\left[(\tau^k)^2 - \alpha_k^2\right]}{2\left(\Phi(\alpha_k)\left[\beta_k - \tau^k\right] + \Phi(\tau^k)\left[\alpha_k - \beta_k\right] + \Phi(\beta_k)\left[\tau^k - \alpha_k\right]\right)}$$

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* [44, 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 [44, p. 29], recommends the use of the *Armijo* rule. Given an initial step size $\sigma > 0$ and $\beta \in (0, 1)$, choose τ^k to be the largest in $\left\{\sigma, \sigma\beta, \sigma\beta^2, \ldots\right\}$ 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. [153] claim improved efficiency with the following modification to the *Armijo* rule:

$$f(x^k + \tau^k d^k) \leq f(x^k) + c_1 \tau^k \left[\left\langle \nabla_x f(x^k), d^k \right\rangle + \frac{1}{2}\tau^k \left\langle d^k, \mathcal{W}_k d^k \right\rangle\right] \tag{K.5}$$

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

$$\left|\left\langle \nabla_x f\left(x^k + \tau^k d^k\right), d^k \right\rangle\right| \leq \left|c_2 \left\langle \nabla_x f\left(x^k\right)\right\rangle, d^k\right| \tag{K.6}$$

This stronger *Wolfe* condition does not allow the gradient to be too positive, excluding points that are far from the minimum. *Christianson* [18] gives Algorithm K.4 for line search using the Wolfe conditions.

Algorithm K.4 Line search using Wolfe conditions.

```
R = 1, r = 0.5, τ = 1, a = 0, b = ∞
while a ≠ b do
    if W1(τ) then
        a = τ
    else
        b = τ
    end if
    if W1(τ) ∧ W2(τ) then
        b = τ
    else
        if b = ∞ then
            τ = max(τ, Ra)
        else
            τ = max((1 − r) a + rb , min(τ, ra + (1 − r) b))
        end if
    end if
end while
```

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.

```
τ = 1, R = 2
while ¬G1(τ) do
    τ = τ/R
end while
while ¬G2(τ) do
    τ = τR
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 [120] 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 λ_1 is the maximum eigenvalue of $\mathcal{B}_k$ and ρ_k provides an upper bound on the corresponding step-size $\tau^k d^k$. *Toh* suggests that the required computation can be reduced by finding λ_1 with the *Lanczos* iteration method. Given a symmetric matrix, $A \in \mathbb{R}^{N \times N}$, the *Lanczos* method generates a sequence of tridiagonal matrices $T_k \in \mathbb{R}^{k \times k}$ having the property that the extremal eigenvalues of T_k are progressively better estimates of the extremal eigenvalues of A. *Golub and van Loan* describe efficient methods for finding the eigenvalues of a symmetric tridiagonal matrix [59, Section 8.5] and have an extensive discussion of *Lanczos* methods [59, Chapter 9].

K.10 Initial solution with the Goldfarb-Idnani algorithm

Goldfarb and *Idnani* [41] 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 jth column of I_k.

When the constraint gradients (columns of B) are linearly independent one can define the operators

$$\begin{aligned} B^\dagger &= \left(B^\top H^{-1} B\right)^{-1} B^\top H^{-1} \\ E &= H^{-1}\left(I - BB^\dagger\right) \end{aligned}$$

where $B^\dagger$ 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 [41], 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 $B^\dagger$gives

$$\lambda(\tilde{x}) \equiv B^\dagger \nabla_x f(\tilde{x}) \geq 0$$

Also multiplying by E gives

$$\begin{aligned} E\nabla f(\tilde{x}) &= H^{-1}\left(I - BB^\dagger\right) B\lambda \\ &= H^{-1}(B - B)\lambda \\ &= 0 \end{aligned}$$

These conditions are necessary and sufficient for $\tilde{x}$ to be the optimal solution to $P(\mathcal{A})$. In addition, the dual algorithm makes use of a set of, so-called, *infeasibility* multipliers

$$r = B^\dagger n^+$$

Some properties of $B^\dagger$ and E:

$$\begin{aligned} & Ew = 0 \Leftrightarrow w = B\alpha \\ & E \textit{ is positive semidefinite} \\ & EWE = E \\ & B^\dagger WE = 0 \\ & EE^+ = E^+ \end{aligned}$$

Algorithm K.6 The *Goldfarb-Idnani* dual algorithm [41].

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 $r = B^\dagger n^+$ (the negative of the step direction in the dual space).

   (b) *Compute the step length:*

   i. *Partial step length, τ_1:*
   This is the maximum step in dual space without violating dual feasibility.
   If $r \leq 0$ or $q = 0$ then set $\tau_1 \leftarrow \infty$.
   Otherwise, set

$$\tau_1 \leftarrow \min_{r_j > 0,\, j=1,\ldots,q} \left\{ \frac{\lambda_j^+(x)}{r_j} \right\} = \frac{\lambda_l^+(x)}{r_l}$$

   In *Step 2c* below, element $k \in \mathcal{K}$ corresponds to the l-th element in $\mathcal{A}$.

   ii. *Full step length, τ_2:*
   This is the minimum step in primal space such that the p-th constraint becomes feasible.
   If $|d| = 0$ then set $\tau_2 \leftarrow \infty$.
   Otherwise, set $\tau_2 \leftarrow -\frac{g_p(x)}{d^\top n^+}$.

   iii. *Step length, τ:*
   Set $\tau \leftarrow \min(\tau_1, \tau_2)$

   (c) *Determine a new solution pair and take the step:*

   i. *No step in primal or dual space:*
   If $\tau = \infty$ then **stop**. The sub-problem $P(\mathcal{A}^+)$ and hence the quadratic problem are infeasible.

   ii. *Step in dual space:*
   If $\tau_2 = \infty$, then set $\lambda^+ \leftarrow \lambda^+ + \tau \begin{pmatrix} -r \\ 1 \end{pmatrix}$, and drop constraint k;
   i.e. set $\mathcal{A} \leftarrow \mathcal{A} \setminus \{k\}$, $q \leftarrow q - 1$, update E and $B^\dagger$, 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 $B^\dagger$ 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 $B^\dagger$, and go to *Step 2a*.

When justifying Algorithm K.6, *Goldfarb* and *Idnani* [41, 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

$$\begin{aligned} g_p(x) &< 0 \\ g_i(x) &= 0 \; \forall i \in \mathcal{A} \\ E^+ \nabla_x f(x) &= 0 \\ \lambda^+(x) \triangleq \left(B^+\right)^\dagger \nabla_x f(x) &\geq 0 \end{aligned}$$

Definition 1: A triple $(x, \mathcal{A}, p)$ consisting of a point x and a set of indices $\mathcal{A}^+ = \mathcal{A} \cup \{p\}$ where $p \in \mathcal{K} \setminus \mathcal{A}$ is said to be a V(violated)-triple if the columns of B^+ are linearly independent and the above conditions apply.

The point $\hat{x}$ corresponding to the V-triple $(\hat{x}, \mathcal{A}, p)$ is the optimal solution to the subproblem obtained by replacing the constraint $g_p(x) \geq 0$ in $P(\mathcal{A}^+)$ by $g_p(x) \geq g_p(\hat{x})$ i.e. by a parallel constraint which passes through $\hat{x}$. Let x^* be the point on the manifold $\mathcal{M}^+ = \left\{x \in \mathbb{R}^n \mid n_i^\top x = b_i, i \in \mathcal{A}^+\right\}$ 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

$$E^+ \nabla_x f(\tilde{x}) = 0 \tag{K.7}$$

$$g_i(\tilde{x}) = 0 \; \forall i \in \mathcal{A} \tag{K.8}$$

$$\lambda^+(\tilde{x}) \triangleq \left(B^+\right)^\dagger \nabla_x f(\tilde{x}) = \lambda^+(x) + \tau \begin{bmatrix} -r \\ 1 \end{bmatrix} \tag{K.9}$$

where $r = B^\dagger n^+$and $g_p(\tilde{x}) = g_p(x) + \tau d^\top n^+$.

Proof: The lemma follows from the properties of the V-triple $(x, \mathcal{A}, p)$ and

$$\nabla_x f(\tilde{x}) = \nabla_x f(x) + \tau H d \tag{K.10}$$

where

$$\begin{aligned} Hd &= HEn^+ \\ &= \left(I - BB^\dagger\right) n^+ \\ &= n^+ - Br \\ &= \begin{bmatrix} B & n^+ \end{bmatrix} \begin{bmatrix} -r \\ 1 \end{bmatrix} \\ &= B^+ \begin{bmatrix} -r \\ 1 \end{bmatrix} \end{aligned}$$

Recalling that $EB = H^{-1}(B - BB * B) = 0$, the results follow from multiplying Equation K.10 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 $f(x)$ over $\mathcal{M}^+$ (since then $g_p(\tilde{x}) = 0$). If $\lambda^+(\tilde{x}) \geq 0$ as well, then $\tilde{x}$ is an optimal solution to $P(\mathcal{A}^+)$ and $(\tilde{x}, \mathcal{A}^+)$ is an S-pair. If not, then Equation K.9 implies that there is a smallest value τ_1 of τ, $\tau_1 < \tau_2$, such that some component of $\lambda^+(\tilde{x}(\tau)) < 0$ for $\tau > \tau_1$. If the constraint, say $k \in \mathcal{A}$, corresponding to this component is dropped from the active set, then $(\tilde{x}(\tau_1), \mathcal{A}^-, p)$, where $\mathcal{A}^- = \mathcal{A} \setminus \{k\}$, is again a V-triple. These remarks are formalised by the following two theorems.

Theorem 1: Given a V-triple $(x, \mathcal{A}, p)$ if $\tilde{x}$ is defined as in Lemma 1 with $\tau = \min\{\tau_1, \tau_2\}$ where

$$\tau_1 = \min\left\{\min_{r_j>0\,,\,1\le j\le q}\left\{\frac{\lambda_j^+(x)}{r_j^+(x)}\right\}, \infty\right\}$$
$$\tau_2 = -\frac{g_p(x)}{d^\top n^+}$$

then

$$g_P(\tilde{x}) \ge g_p(x)$$
$$f(\tilde{x}) - f(x) = \tau d^\top n^+\left(\frac{1}{2}\tau + \lambda_{q+1}^+(x)\right) \ge 0$$

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,

$$d^\top n^+ = n^{+T}En^+ = n^{+T}EHEn^+ = d^\top Hd > 0$$

and $\tau \ge 0$. Hence, from Lemma 1, $g_P(\tilde{x}) \ge g_p(x)$. Also, from Taylor's theorem

$$f(\tilde{x}) - f(x) = \tau d^\top \nabla_x f(x) + \frac{1}{2}\tau^2 d^\top Hd$$

Since $E^+\nabla_x f(x) = 0$ implies that $\nabla_x f(x) = B^+\lambda^+(x)$, it follows that $E\nabla_x f(x) = En^+\lambda_{q+1}^+(x)$ and

$$d^\top\nabla_x f(x) = n^{+T}E\nabla_x f(x) = d^\top n^+\lambda_{q+1}^+(x) \ge 0$$

The result follows by substitution. Moreover, as long as $\tau > 0$, $f(\tilde{x}) > f(x)$. From the definition of τ and Lemma 1 it is evident that $E^+\nabla_x f(\tilde{x}) = 0$, $g_i(\tilde{x}) = 0$, $i \in \mathcal{A}$ and $\lambda^+(\tilde{x}) \ge 0$. 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

$$\begin{aligned} \hat{\lambda}(x) &= \hat{B}^\dagger \nabla_x f(x) \\ &= \begin{cases} \lambda_{j(i)} - \frac{\lambda_l}{r_l} r_{j(i)} \geq 0 & i \in \mathcal{A}^- \\ \frac{\lambda_l}{r_l} \geq 0 \end{cases} \end{aligned}$$

and hence that $(x, \mathcal{A}^-, p)$ is a V-triple.

Goldfarb and *Idnani* [41, 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,

$$\begin{aligned} B^\dagger &= \left(B^\top {L^{-1}}^\top L^{-1} B \right)^{-1} B^\top {L^{-1}}^\top L^{-1} \\ &= \left(C^\top C \right)^{-1} C^\top L^{-1} \\ &= R^{-1} {R^{-1}}^\top C^\top L^{-1} \\ &= R^{-1} Q_1^\top L^{-1} \\ &= R^{-1} J_1^\top \end{aligned}$$

and

$$E = {L^{-1}}^\top L^{-1} - {L^{-1}}^\top C \left(C^\top C \right)^{-1} C^\top L^{-1}$$

$$
\begin{aligned}
&= {L^{-1}}^\top L^{-1} - {L^{-1}}^\top C R^{-1} {R^{-1}}^\top C^\top L^{-1} \\
&= {L^{-1}}^\top Q Q^\top L^{-1} - {L^{-1}}^\top Q_1 Q_1^\top L^{-1} \\
&= {L^{-1}}^\top Q_2 Q_2^\top L^{-1} \\
&= J_2 J_2^\top
\end{aligned}
$$

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

$$
\begin{aligned}
C R^{-1} &= \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} R^{-1} \\
&= Q_1
\end{aligned}
$$

and

$$
\begin{aligned}
J &= \begin{bmatrix} J_1 & J_2 \end{bmatrix} \\
&= \begin{bmatrix} {L^{-1}}^\top Q_1 & {L^{-1}}^\top Q_2 \end{bmatrix} \\
&= {L^{-1}}^\top Q
\end{aligned}
$$

In the dual algorithm the vectors $d = En^+$ and $r = B^\dagger n^+$ are required. Compute the intermediate vector

$$
\begin{aligned}
v &= J^\top n^+ \\
&= \begin{bmatrix} J_1^\top \\ J_2^\top \end{bmatrix} n^+ \\
&= \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}
\end{aligned}
$$

from which it follows that

$$
d = J_2 v_2
$$

and

$$
r = R^{-1} v_1
$$

Goldfarb and *Idnani* describe an efficient method for updating the factors J and R when a constraint is added or deleted.

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 script *linesearch_test.m* implements unconstrained quasi-Newton optimisation with various line-search methods. The Octave file *sqp_common.m* contains the function to be optimised, constraints, gradients and the Hessian function:

$$\begin{aligned} \textbf{minimise} \quad & f(x) = x_1^4 + x_2^4 + x_3^4 + x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + x_3 + 5 \\ \textbf{subject to} \quad & g(x) = \left[x_2 - 1; -x_1 - \sqrt{2}; -x_3 - 0.5\right] \geq 0 \end{aligned}$$

The step size is 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, 0.021104 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, 0.019491 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, 0.021660 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, 0.024575 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, 0.076705 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, 0.021148 secs
LINESEARCH quadratic 104 iterations 105 f(x) calls
LINESEARCH quadratic f(x)= 4.361595 x=[ -0.574868 0.334438 -0.574777  ]
```

The Octave 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 55 44
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 55 44
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 12 6
SQP none exact armijo_kim [ -1.414214 1.000000 -0.527104 ] 1 4 12 6
SQP none exact goldstein [ -1.414214 1.000000 -0.527104 ] 1 4 16 10
SQP none exact goldensection [ -1.414191 0.999985 -0.527110 ] 1 17 243 224
SQP none bfgs nosearch [ -1.414214 1.000000 -0.527073 ] 1 8 10 0
SQP none bfgs quadratic [ -1.414213 1.000000 -0.526831 ] 1 25 52 25
SQP none bfgs armijo [ -1.414214 1.000000 -0.527073 ] 1 8 20 10
SQP none bfgs armijo_kim [ -1.414214 1.000000 -0.527073 ] 1 8 20 10
SQP none bfgs goldstein [ -1.414214 1.000000 -0.527228 ] 1 7 38 29
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 55 44
SQP none eye goldensection [ -1.414194 0.999995 -0.527054 ] 1 33 681 646
```

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 Octave script *goldfarb_idnani_fir_minimum_phase_test.m* modifies a bandpass FIR filter designed by the *remez* function by constraining the zeros of the filter to lie within the unit circle in the z-plane and by constraining the amplitude response at regular intervals in frequency. The filter specification is:

```
U=2 % Number of real zeros
V=0 % Number of real poles
M=38 % Number of complex zeros
Q=0 % Number of complex poles
R=1 % Denominator polynomial decimation factor
n=50 % Frequency points across the band
tol=0.001 % Tolerance on update
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=3 % Pass band amplitude peak-to-peak ripple
Wap=1 % Pass band weight
dBas=30 % Stop band minimum attenuation
Wasl=0.2 % Lower stop band amplitude weight
Wasu=5 % Upper stop band amplitude weight
```

Figure K.1 shows the amplitude response of the initial band-pass FIR filter and Figure K.2 shows the zeros of the initial band-pass FIR filter. Figure K.3 shows the amplitude response of the modified band-pass FIR filter. The frequencies of the amplitude constraints are marked. Unfortunately, one amplitude constraint is not satisfied. Figure K.4 shows the zeros of the modified band-pass FIR filter.

Figure K.1: Amplitude response of the initial band-pass FIR filter.

Figure K.2: Zeros of the initial band-pass FIR filter.

Figure K.3: Amplitude response of the band-pass FIR filter after application of the Goldfarb-Idnani algorithm. The frequencies of the amplitude constraints are marked.

Figure K.4: Zeros of the band-pass FIR filter after application of the Goldfarb-Idnani algorithm.

Appendix L

Fourier transform of the Gaussian function

See *NIST Digital Library of Mathematical Functions* [56, 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:

$$\imath t g\left(t\right) = -\frac{2\imath}{\alpha^2}\frac{dg\left(t\right)}{dt}$$

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 Ω in the same way that the continuous-time filter behaves at frequency $\omega = \tan\frac{\Omega T}{2}$. For example, if the desired cut-off frequency in the z-domain is $\Omega_c T = \frac{\pi}{2}$, then we choose the cutoff frequency of the s-plane prototype filter to be:

$$\omega_c = \tan\frac{\Omega_c T}{2} = \tan\frac{\pi}{4} = 1$$

In this work I have assumed that the sampling interval, T, is 1. Alternatively, one can choose that the *Nyquist* frequency corresponds to 1 so that the sampling frequency is 2. The Octave *signal* toolbox filter design functions use the latter convention.

Filter design using analog prototypes proceeds as follows:

- the critical frequencies in the z-plane are determined
- the critical frequencies are mapped to the s-plane and used to design the s-plane prototype
- the bi-linear transform is used to convert that prototype to the z-plane

M.1.1 Design of Butterworth IIR filters

The Butterworth filter transfer function is described in many textbooks (for example, *Roberts* and *Mullis* [195, 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 ω_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 realisation as a cascade of second order sections uses the poles in the left-hand half plane, λ_k:

$$\begin{aligned}\lambda_k &= \omega_c e^{\imath\theta_k} \\ \theta_k &= \frac{\pi}{2}\left(1+\frac{(2k-1)}{n}\right),\ 1 \le k \le n\end{aligned}$$

For each conjugate pole pair the corresponding low-pass filter second-order section with unity DC gain has transfer function[a]:

$$\begin{aligned}\hat{H}_k(s) &= \frac{\lambda_k\lambda_k^*}{(s-\lambda_k)(s-\lambda_k^*)} \\ &= \frac{\lambda_k\lambda_k^*}{s^2-(\lambda_k+\lambda_k^*)s+\lambda_k\lambda_k^*} \\ &= \frac{\omega_c^2}{s^2-2\omega_c\cos\theta_k s+\omega_c^2}\end{aligned}$$

If n is odd, there is a single real pole at $s=-\omega_c$, $k=\frac{n+1}{2}$, and the corresponding low-pass first-order section is:

$$\hat{H}_k(s) = \frac{\omega_c}{s+\omega_c}$$

The Butterworth high-pass filter with cut-off frequency ω_c is found with the s-plane transformation $\frac{s}{\omega_c} \to \frac{\omega_c}{s}$. 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:

$$
\begin{aligned}
H_k(z) &= \frac{g\left[1+z^{-1}\right]^2}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \\
&= d_0 + \frac{q_1 z^{-1} + q_2 z^{-2}}{1 + p_1 z^{-1} + p_2 z^{-2}}
\end{aligned}
$$

where:

$$
\begin{aligned}
g &= \omega_c^2 \\
a_0 &= 1 - 2\omega_c \cos\theta_k + \omega_c^2 \\
a_1 &= 2\left(\omega_c^2 - 1\right) \\
a_2 &= 1 + 2\omega_c \cos\theta_k + \omega_c^2 \\
d_0 &= \frac{g}{a_0} \\
p_1 &= \frac{a_1}{a_0} \\
p_2 &= \frac{a_2}{a_0} \\
q_1 &= d_0\left(2 - p_1\right) \\
q_2 &= d_0\left(1 - p_2\right)
\end{aligned}
$$

Similarly, for the first order high-pass section:

$$
\begin{aligned}
H_{\frac{n+1}{2}}(z) &= \frac{\left(1 - z^{-1}\right)}{(\omega_c + 1) + (\omega_c - 1) z^{-1}} \\
&= \frac{d}{w_c} - \frac{q z^{-1}}{1 + p z^{-1}}
\end{aligned}
$$

and for the Butterworth second order high-pass section:

$$
\begin{aligned}
H_k(z) &= \frac{\left[1 - z^{-1}\right]^2}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \\
&= t_0 + \frac{t_1 z^{-1} + t_2 z^{-2}}{1 + p_1 z^{-1} + p_2 z^{-2}}
\end{aligned}
$$

where

$$
\begin{aligned}
t_0 &= \frac{d_0}{g} \\
t_1 &= -\frac{d_0}{g}\left(2 + p_1\right) \\
t_2 &= \frac{q_2}{g}
\end{aligned}
$$

M.2 Low passband sensitivity IIR filters

Vaidyanathan et al. [180, 177] 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 ω regardless of the values of the multipliers (so long as they are within a certain range) then the implementation is called *structurally bounded* or *structurally passive* and $G(z)$ is called *bounded real*. If $|G(e^{\imath\omega})| = 1$ for certain frequencies ω_k in the passband, then at those frequencies, when an m_i is perturbed the value of $|G(e^{\imath\omega})|$ can only decrease. In other words the first-order sensitivity is zero at ω_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 quantisation. 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 ω. 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 ω. 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 α 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 [180, 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 [121, 68, 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 0.5dB and stop-band attenuation 40dB.

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 floating-point 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* [125, 142] 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* [46, 45]. 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* [77]. 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*, κ, is real and $0 \leq k \leq 1$. The *modular angle*, θ, 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, ϵ is the pass-band ripple factor and $\omega_a = \frac{1}{\kappa}$ is the stop-band edge frequency. Both ω_a and the frequency ω are normalised to ω_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 α_p and stop-band ripple α_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 α_p, α_a and κ. 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 $\frac{2G'}{G} = \frac{K'}{K}$ [77, 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* [77, Section IV] derive a recurrence relation for the modulus, κ, that appears in successive Landen transformations:

$$\kappa_{n+1} = \left[\frac{\kappa_n}{1+\sqrt{1-\kappa_n^2}}\right]^2$$

This sequence, κ_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, κ_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 Ω 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 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* [77, 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* [46, 45] 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* [142] 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* [142], 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 [142, 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 Ω 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* [142, 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* [77, 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 Ω 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* [238] 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, ± 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 ζ_1 and ζ_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 α_k such that, in the z-plane, $H\left(z\right)H\left(\frac{1}{z}\right)$ has $m+1$ alternating extrema, Δ_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 α_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, Λ, 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 (ζ_1, ζ_2) and $\Lambda = 0$. The angles, ω_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, β_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* [116, Appendix] rather than *Saramäki* [238, 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, Δ_s, is specified instead of the pass-band ripple, then the corresponding pass-band ripple, Δ_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 = [    0.0000000000,    0.0000000000,    0.0000000000,    0.0000000000, ...
         0.0000000000,    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 ζ_1 and ζ_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 Δ_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 Δ_p is specified, then the optimisation problem is to find the stop band attenuation, Δ_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 (ζ_1, ζ_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 Δ_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 Δ_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 Δ_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,    0.0888119516, ...
         0.1258701526,    0.1258701526,    0.0888119516,    0.0426992728, ...
         0.0127354024,    0.0018065090 ];


d = [    0.0000000000,    0.0000000000,    0.0000000000,    1.0000000000, ...
        -1.9501129639,    2.9114995274,   -2.5231419691,    1.5889529276, ...
        -0.5970515170,    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 plot 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* [116] 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 [116, 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 δ_s and 0 achieving these values at $n_{min}+1$ points such that the value at $\omega = \omega_s$ is δ_s. Here, δ_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 δ_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 ω_p being the specified passband edge, whereas δ_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 ψ^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 ψ^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 $l = 1, 2, \ldots, n_{all}$ (ϵ 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 [239] and *Vaidyanathan* and *Mitra* [177] 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* [116, 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* [238].

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, Γ_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:

```
ftol=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, ...
        0.8292402387, ...
        1.0000000000,   1.0000000000,   1.0000000000,   1.2132533879, ...
        1.2955253072, ...
        0.7950661045,   0.8940993949,   1.2939773293,   0.3403350160, ...
        0.1022103019, ...
        0.8307487192,   0.8522658187,   0.8566164878,   0.9130582446, ...
        0.9737984530, ...
        0.1658364240,   0.4787379725,   0.3290091008,   0.6074256122, ...
        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:

```
ftol=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, ...
        0.7374611693,   0.7830853231,   1.0000000000,   1.0000000000, ...
        1.0000000000,   1.2770000542,   1.3560035995, ...
        0.1197954840,   0.4101589302,   0.7934970298,   0.8762250189, ...
        1.2220444985,   0.4101589302,   0.1197954840, ...
        0.7287526514,   0.7669920494,   0.7748861071,   0.7909231674, ...
        0.7953147511,   0.8897749259,   0.9687752445, ...
        0.2165873654,   0.1168506814,   0.6178959040,   0.3839260890, ...
        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 = [   0.7669920499,   0.7748861068,   0.7909231668,   0.9687752439, ...
         0.1168506803,   0.6178959040,   0.3839260874,   0.7162074983 ]';
```

```
% All-pass single-vector representation
Va2=1,Qa2=6,Ra2=1
a2 = [   0.7359976325, ...
         0.7287526510,   0.7953147523,   0.8897749262, ...
         0.2165873684,   0.4347709644,   0.7064801519 ]';
```

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* [80, 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* [80, 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:

$$\begin{aligned} 1-\delta_p \le |F_R(\Omega T)| \le 1+\delta_p,\ & \Omega T \in [0, \Omega_p T] \\ |F_R(\Omega T)| \le \delta_s, \qquad & \Omega T \in [\pi - \Omega_p T, \pi] \end{aligned}$$

and the phase response of the all-pass transformation is such that:

$$\begin{aligned} -\Omega_p T \le \Omega T \le \Omega_p T, \qquad & \omega T \in [0, \omega_{apl} T] \\ \pi - \Omega_p T \le \Omega T \le \pi + \Omega_p T, \quad & \omega T \in [\omega_{asl} T, \omega_{asu} T] \\ 2\pi - \Omega_p T \le \Omega T \le 2\pi + \Omega_p T,\ & \omega T \in [\omega_{apu} T, \pi] \end{aligned}$$

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

$$\begin{aligned} 1-\Delta_p \le |G(\omega T)| \le 1+\Delta_p,\ & \omega T \in [0, \omega_{apl} T] \cup [\omega_{apu} T, \pi] \\ |G(\omega T)| \le \Delta_s, \qquad & \omega T \in [\omega_{asu} T, \omega_{asl} T] \end{aligned}$$

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 δ_s in the stop-band. This is achieved 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.

```
ftol=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 = [  -0.0314883306,  -0.0000086184,   0.2814859166,   0.5000170650, ...
         0.2814859166,  -0.0000086184,  -0.0314883306 ]';
```

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 3-signed-digit, 16-bit coefficients designed with the *branch-and-bound* algorithm.

The Octave script *branch_bound_johanssonOneMlattice_bandstop_16_bits_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 initial band-stop filter is that designed by the Octave script *johansson_cascade_allpass_bandstop_test.m*. The band-stop filter specification is:

```
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 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.26 shows the pass-band and stop-band responses of the filter.

Figure M.27: 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 initial band-stop filter is that designed by the Octave script *johansson_cascade_allpass_bandstop_test.m*. 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 resulting band-stop filter coefficients are:

```
f = [  -0.0280871068,   0.0065333941,   0.2780868425,   0.4869336766, ...
        0.2780868425,   0.0065333941,  -0.0280871068 ]';

k0 = [  -0.1784512812,   0.8243132570,  -0.1410010448,   0.6462785192 ]';

k1 = [  -0.1519187362,   0.5060944397 ]';
```

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. 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. An even-order, $N = 2M$, symmetric FIR filter is assumed to have $M+1$ distinct coefficients and odd length, $2M+1$, with $h_{2M-n} = h_n$. The *direct-form* implementation of a symmetric FIR filter requires approximately half the number of multipliers used by the non-symmetric direct-form implementation. A symmetric FIR filter has a linear phase response or, equivalently, a constant group delay response. 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.

N.1 Low passband sensitivity FIR lattice filters

N.1.1 Lattice decomposition of an FIR digital filter

Vaidyanathan [176] 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:

$$\begin{aligned} H_m(z) &= H_{m+1}(z) \\ G_m(z) &= zG_{m+1}(z) \end{aligned}$$

For the general case, *Vaidyanathan* derives the following order reduction of $A_{m+1}(z)$ to $A_m(z)$:

$$\begin{bmatrix} H_m(z) \\ G_m(z) \end{bmatrix} = \begin{bmatrix} k_{m+1} & \hat{k}_{m+1} \\ -\hat{k}_{m+1}z & k_{m+1}z \end{bmatrix} \begin{bmatrix} H_{m+1}(z) \\ G_{m+1}(z) \end{bmatrix}$$

where [176, Equations 13 and 14]:

$$k_{m+1} = \frac{-g_{m+1,m+1}}{\sqrt{h_{m+1,m+1}^2 + g_{m+1,m+1}^2}}, \quad \hat{k}_{m+1} = \frac{h_{m+1,m+1}}{\sqrt{h_{m+1,m+1}^2 + g_{m+1,m+1}^2}} \tag{N.5}$$

or [176, Equation 23]:

$$k_{m+1} = \frac{h_{m+1,0}}{\sqrt{h_{m+1,0}^2 + g_{m+1,0}^2}}, \quad \hat{k}_{m+1} = \frac{g_{m+1,0}}{\sqrt{h_{m+1,0}^2 + g_{m+1,0}^2}} \tag{N.6}$$

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 τ_m are orthogonal matrixes:

$$\tau_m(z) = \begin{bmatrix} k_m & -\hat{k}_m z^{-1} \\ \hat{k}_m & k_m z^{-1} \end{bmatrix}$$

Vaidyanathan [176][Section VII] shows that this determines the finite-wordlength properties of the lattice FIR filter:

Figure N.1: Structure of the complementary FIR lattice filter (see *Vaidyanathan* [176, Figure 3]).

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:

$$\left| \frac{\partial H_N}{\partial k_m} \right|^2 + \left| \frac{\partial G_N}{\partial k_m} \right|^2 = 1$$

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

Figure N.1 (see Figure 3 of *Vaidyanathan* [176]) shows the complementary FIR lattice structure.

For convenience, call x'_n the input to state x_n of the n-th section, $\hat{y}_n$ 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.

Algorithm N.1 Construction of a state variable description of the complementary FIR lattice filter.

$\hat{y}_0 = \hat{k}_0 u$
$y_0 = k_0 u$
for $n = 1, \ldots, N$ **do**
  $x'_n = \hat{y}_{n-1}$
  $\hat{y}_n = \hat{k}_n y_{n-1} + k_n x_n$
  $y_n = k_n y_{n-1} - \hat{k}_n x_n$
end for
$y = y_N$

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.

$$\begin{bmatrix} \hat{y}_0 \\ y_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 & \cdots & \cdots & 0 & \hat{k}_0 \\ 0 & \cdots & \cdots & 0 & k_0 \\ 1 & \cdots & \cdots & 0 & 0 \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \\ u \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ \hat{y}_1 \\ y_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \hat{k}_1 & k_1 & 0 & \cdots & 0 \\ 0 & k_1 & -\hat{k}_1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & \cdots & & & \cdots & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_0 \\ y_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\begin{bmatrix} x'_1 \\ x'_2 \\ \hat{y}_2 \\ y_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \hat{k}_2 & k_2 & 0 & \cdots & 0 \\ 0 & 0 & k_2 & -\hat{k}_2 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & & \ddots & \vdots \\ 0 & \cdots & & & & \cdots & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ \hat{y}_1 \\ y_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}$$

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

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

The Octave function *complementaryFIRlattice2Abcd* returns the state variable description of a complementary FIR lattice filter. The Octave script *complementaryFIRlattice2Abcd_symbolic_test.m* creates a symbolic state variable description of the 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 [78], is simpler and more accurate than the cepstral method of *Mian* and *Nainer* [66].

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 $1 - H^*(z) H(z)$, 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* [78] 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:

$$h_0 = a_0^2 + a_1^2 + a_2^2 + a_3^2$$

$$
\begin{aligned}
h_1 &= a_0 a_1 + a_1 a_2 + a_2 a_3 \\
h_2 &= a_1 a_3 + a_0 a_2 \\
h_3 &= a_0 a_3
\end{aligned}
$$

The Newton-Raphson method solves a system of equations $f(\boldsymbol{a}) = 0$ by successive approximations:

$$
f(\boldsymbol{a}_k) + J(\boldsymbol{a}_k)[\boldsymbol{a}_{k+1} - \boldsymbol{a}_k] = 0
$$

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:

$$
\begin{aligned}
f(\boldsymbol{a}) &= \begin{bmatrix} h_0 - a_0^2 - a_1^2 - a_2^2 - a_3^2 \\ h_1 - a_0 a_1 - a_1 a_2 - a_2 a_3 \\ h_2 - a_1 a_3 - a_0 a_2 \\ h_3 - a_0 a_3 \end{bmatrix} \\
J(\boldsymbol{a}) &= -\begin{bmatrix} 2a_0 & 2a_1 & 2a_2 & 2a_3 \\ a_1 & a_2 + a_0 & a_3 + a_1 & a_2 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & 0 & 0 & a_0 \end{bmatrix} \\
&= -\begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 & 0 \\ a_2 & a_3 & 0 & 0 \\ a_3 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ 0 & a_0 & a_1 & a_2 \\ 0 & 0 & a_0 & a_1 \\ 0 & 0 & 0 & a_0 \end{bmatrix}
\end{aligned}
$$

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* [73] 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* [66] 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:

$$F(z) = 1 - |H(z)|^2 = 1 - H^*(z)\, H(z)$$

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]:

[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}.

```
% 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]);
```

Sturm's example of minimum phase spectral factorisation by semi-definite programming

Sturm [231, 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.* [22] and *Dumitrescu* [20].

Sturm contributed a spectral factorisation example to the 7'th DIMACS challenge [38]. 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* [104] 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 [104, 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 [104, 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 = [  0.702278449711, -0.284747156084,  0.195991006805,  0.310333250487, ...
          0.135915842568, -0.007622198978,  0.013539413531,  0.068315148258, ...
          0.019358423458, -0.088556184091, -0.123568394332, -0.052846068647, ...
          0.035521732713,  0.060100582689,  0.029846895882,  0.000251446373, ...
         -0.003826320098,  0.001544786019, -0.000527786304, -0.006517762892, ...
         -0.006982385805, -0.002252609065,  0.001415579686,  0.001682492421, ...
          0.000733967882,  0.000312597856,  0.000190007106, -0.000036513340, ...
         -0.000173379620,  0.000038188995, -0.000037157731 ]';
```

The reflection coefficients of the bandpass and complementary filters are:

```
k = [  0.999973545923,  0.999997439113,  0.999892198493,  0.999912068374, ...
       0.999560459108,  0.996590306039,  0.997582105030,  0.999848632343, ...
       0.997255806605,  0.999503325227,  0.999993493612,  0.993567986373, ...
       0.962834846979,  0.987076261545,  0.964636964496,  0.863342898565, ...
       0.964636964496,  0.987076261545,  0.962834846979,  0.993567986373, ...
       0.999993493612,  0.999503325227,  0.997255806605,  0.999848632343, ...
       0.997582105030,  0.996590306039,  0.999560459108,  0.999912068374, ...
       0.999892198493,  0.999997439113, -0.007273751095 ]';
```

```
khat = [  0.007273751095, -0.002263132077,  0.014683030793,  0.013261052728, ...
         -0.029646055167, -0.082509162577, -0.069497796536,  0.017398632169, ...
          0.074032804834,  0.031513534540, -0.003607316752,  0.113237169053, ...
          0.270090831837,  0.160251221194, -0.263582106238, -0.504617716195, ...
         -0.263582106238,  0.160251221194,  0.270090831837,  0.113237169053, ...
         -0.003607316752,  0.031513534540,  0.074032804834,  0.017398632169, ...
         -0.069497796536, -0.082509162577, -0.029646055167,  0.013261052728, ...
          0.014683030793, -0.002263132077,  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* [78].

Figure N.3: Zeros of the complementary filter found with the Newton-Raphson method of *Orchard* and *Willson* [78].

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* [66].

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* [231, 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* [104, Equation 38]. The Riccati equation is solved with the *dare* function from the Octave-Forge *control* toolbox [162]. 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* [231, Section 3.2].

Figure N.6: Response of the complementary FIR filter found with the Riccati equation method of *Tuqan* and *Vaidyanathan* [104, Equation 38].

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

This Section follows the description by *Dumitrescu et al* [22] 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 [22, 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 kth 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 [22, 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. [22] 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 [181, Section II] $\mathcal{E}\{[\hat{r}_k - r_k][\hat{r}_l - r_l]\}$, which can be approximated by [242, 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. [22, 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.* [181, 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 [22, 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 Γ 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 [22, 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 [264]. *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* [104, Theorem 3]. Also see *Sayed* and *Kailath* [6, Section 5].

Dumitrescu et al. [22, 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.* [22, 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.1.7 Design of a complementary FIR lattice band-pass Hilbert filter

The Octave script *complementaryFIRlattice_socp_slb_bandpass_hilbert_test.m* designs a PCLS optimised FIR lattice Hilbert band-pass filter with order 16. The filter specification is:

```
ftol=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
fasl=0.05 % Lower stop band upper edge
fapl=0.1 % Pass band lower edge
fapu=0.2 % Pass band upper edge
fasu=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=1 % Ampl. lower stop band weight
Wasu=1 % 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=1.50 % Pass band phase(rad./pi), adjusted for tp
ppr=0.02 % Phase pass band peak-to-peak ripple(rad./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 = [   0.9839340410,   0.9836437301,   0.9837528750,   0.9836733039, ...
         0.9836205528,   0.9838047149,   0.9839595218,   0.9833746547, ...
         0.9843749538,   0.9666769676,   0.9697456292,   0.9838800206, ...
         0.9155055404,   0.9508963402,   0.9838903931,   0.9841992240, ...
         0.1379388179 ]';
```

and:

```
khat2 = [  -0.0200057134,  -0.0297782272,   0.0241816868,   0.0662147111, ...
            0.0584640431,   0.0428830867,   0.0780160128,   0.0811604589, ...
           -0.1307279783,  -0.4598802268,  -0.4604848476,   0.0637948061, ...
            0.5554670407,   0.4787434520,   0.0461566547,  -0.1794756291, ...
            0.9833693675 ]';
```

The corresponding direct-form FIR band-pass Hilbert filter coefficients are:

```
Nh2 = [   0.0925818258,   0.0624654575,  -0.0682339974,  -0.2097877442, ...
         -0.2037346996,  -0.0119898586,   0.1961665537,   0.1994981937, ...
          0.0605911008,  -0.0873720167,  -0.0927559992,  -0.0264363290, ...
          0.0027356922,  -0.0151200981,  -0.0232821492,   0.0112543148, ...
          0.0134197360 ];
```

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) = [   0.0200057134,   0.0353909508,   0.0419017192,   0.0756711181, ...
                0.0927563050,   0.0994163338,   0.1202701540,   0.1376843538, ...
                0.1794849193,   0.4334279552,   0.5523105823,   0.5572615280, ...
                0.6620753712,   0.5876445889,   0.5703753276,   0.5267402849 ];
```

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 Hilbert filter.

(b) Zeros of the initial FIR lattice band-pass Hilbert complementary filter.

Figure N.9: Zeros of the initial FIR lattice band-pass Hilbert 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 band-pass Hilbert filter complementary filter.

Figure N.11: Zeros of the PCLS optimised FIR lattice band-pass Hilbert filter and the complementary filter.

N.2 Design of FIR digital filters with unconstrained optimisation of the piece-wise mean-squared error of the response

An FIR filter with zero-phase amplitude response $A(\omega)$, piece-wise constant desired amplitudes, A_k, and weights, W_k, over frequency regions $\Omega_k \subset [0, \pi]$, has piece-wise mean-squared-error:

$$\mathcal{E}_k^2 = \frac{1}{\pi} \int_{\Omega_k} W_k \left[A(\omega) - A_k\right]^2 d\omega$$

If $\boldsymbol{h}$ is a vector containing the FIR filter impulse response, then the weighted sum of the piece-wise mean-squared-errors, $\mathcal{E}^2$, is a quadratic form:

$$\mathcal{E}^2 = \boldsymbol{h}^\top \boldsymbol{Q} \boldsymbol{h} + 2\boldsymbol{q}\boldsymbol{h} + c \tag{N.12}$$

In this section I give examples of the optimisation of $\mathcal{E}^2$ without constraints on the response of the FIR filter. The examples show optimisation with the Octave qp function. In each case qp objective function is the mean-squared-error quadratic form shown in Equation N.12 and there are no qp constraints. I consider symmetric and non-symmetric FIR filters and usually assume that the filters are even-order and odd-length.

N.2.1 Zero-phase transfer functions of symmetric FIR filters

Zero-phase transfer functions of even-order symmetric FIR filters

The impulse response of an even order, $N = 2M$, odd length, $2M + 1$, symmetric FIR filter has $h_n = h_{N-n}$. The transfer function is:

$$\begin{aligned} H(z) &= \sum_{n=0}^{2M} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n z^{-n} + h_M z^{-M} + \sum_{n=M+1}^{2M} h_n z^{-n} \\ &= z^{-M} \left\{ h_M + \sum_{n=0}^{M-1} h_n \left[z^{M-n} + z^{-M+n}\right] \right\} \end{aligned}$$

The zero-phase amplitude response is:

$$A(\omega) = h_M + \sum_{n=0}^{M-1} 2h_n \cos\left[(M-n)\omega\right]$$

Zero-phase transfer functions of odd-order symmetric FIR filters

The impulse response of an odd-order, $N = 2M - 1$, even-length, $2M$, symmetric FIR filter has $h_n = h_{N-1-n}$. The transfer function is:

$$\begin{aligned} H(z) &= \sum_{n=0}^{2M-1} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n z^{-n} + \sum_{n=M}^{2M-1} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n \left[z^{-n} + z^{n-(2M-1)}\right] \\ &= z^{-M+\frac{1}{2}} \sum_{n=0}^{M-1} h_n \left[z^{-n+M-\frac{1}{2}} + z^{n-M+\frac{1}{2}}\right] \end{aligned}$$

The zero-phase amplitude response is:

$$A(\omega) = 2\sum_{n=0}^{M-1} h_n \cos\left[\left(M - \frac{1}{2} - n\right)\omega\right]$$

N.2.2 Zero-phase transfer functions of anti-symmetric FIR filters

Zero-phase transfer functions of even-order anti-symmetric FIR filters

The impulse repsonse of an even order, $N = 2M$, odd length, $2M + 1$, -symmetric FIR filter has $h_M = 0$ and $h_n = -h_{N-n}$. The transfer function is:

$$\begin{aligned} H(z) &= \sum_{n=0}^{2M} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n z^{-n} + 0 + \sum_{n=M+1}^{2M} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n z^{-n} + 0 + \sum_{n=0}^{M-1} h_{2M-n} z^{-2M+n} \\ &= z^{-M} \left\{ \sum_{n=0}^{M-1} h_n \left[z^{M-n} - z^{-M+n} \right] \right\} \end{aligned}$$

The zero-phase amplitude response is:

$$A(\omega) = \sum_{n=0}^{M-1} 2h_n \sin\left[(M-n)\omega\right]$$

Zero-phase transfer functions of odd-order anti-symmetric FIR filters

The impulse response of an odd-order, $N = 2M - 1$, even-length, $2M$, anti-symmetric FIR filter has $h_n = -h_{N-n}$. The transfer function is:

$$\begin{aligned} H(z) &= \sum_{n=0}^{2M-1} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n z^{-n} + \sum_{n=M}^{2M-1} h_n z^{-n} \\ &= \sum_{n=0}^{M-1} h_n \left[z^{-n} - z^{n-(2M-1)} \right] \\ &= z^{-M+\frac{1}{2}} \sum_{n=0}^{M-1} h_n \left[z^{-n+M-\frac{1}{2}} - z^{n-M+\frac{1}{2}} \right] \end{aligned}$$

The zero-phase amplitude response is:

$$A(\omega) = 2 \sum_{n=0}^{M-1} h_n \sin\left[\left(M - \frac{1}{2} - n \right) \omega \right]$$

Zero-phase transfer functions of anti-symmetric FIR Hilbert filters

The frequency response of an ideal Hilbert transform filter is [170, pp.790-792]:

$$H(\omega) = \begin{cases} -\imath & 0 < \omega < \pi \\ \imath & -\pi < \omega < 0 \end{cases}$$

The corresponding impulse response is:

$$\hat{h}_k = \frac{1}{2\pi} \int_{-\pi}^{0} \imath e^{\imath \omega k} d\omega - \frac{1}{2\pi} \int_{0}^{\pi} \imath e^{\imath \omega k} d\omega$$

$$= \begin{cases} \frac{2}{\pi k} \sin^2 \frac{\pi k}{2} & k \neq 0 \\ 0 & k = 0 \end{cases}$$

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 $k = \frac{1}{2}, \frac{3}{2}, \ldots, M-\frac{1}{2}$:

$$\begin{aligned} H(z) &= z^{-M+\frac{1}{2}} \sum_{k=-M+\frac{1}{2}}^{M-\frac{1}{2}} \hat{h}_k z^{-k} \\ &= z^{-M+\frac{1}{2}} \sum_{k=\frac{1}{2}}^{M-\frac{1}{2}} \hat{h}_k \left[z^{-k} - z^{k}\right] \\ &= z^{-M+\frac{1}{2}} \sum_{k=0}^{M-1} \hat{h}_{k+\frac{1}{2}} \left[z^{-k-\frac{1}{2}} - z^{k+\frac{1}{2}}\right] \\ &= z^{-M+\frac{1}{2}} \sum_{k=0}^{M-1} h_{M-1-k} \left[z^{k+\frac{1}{2}} - z^{-k-\frac{1}{2}}\right] \end{aligned}$$

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.2.3 Piece-wise mean-squared-error of the response of an FIR filter

Piece-wise mean-squared-error of the response of a non-symmetric FIR filter

If the impulse response of a non-symmetric filter is $\boldsymbol{h} = [h_0, \ldots, h_N]^\top$ and the nominal FIR filter delay, $0 < d_k < N$, is an integral number of samples, then the mean-squared-error over each frequency region Ω_k is:

$$\begin{aligned}
\mathcal{E}_k^2 &= \frac{1}{\pi}\int_{\Omega_k} W_k \left|H(\omega) - A_k e^{-\imath d_k \omega}\right|^2 d\omega \\
&= \frac{1}{\pi}\int_{\Omega_k} W_k \left[\sum_{n=0}^{N} h_n e^{-\imath n\omega} - A_k e^{-\imath d_k \omega}\right]\left[\sum_{n=0}^{N} h_n e^{\imath n\omega} - A_k e^{\imath d_k \omega}\right] d\omega \\
&= \frac{1}{\pi}\int_{\Omega_k} W_k \left[\sum_{n=0}^{N}\sum_{m=0}^{N} h_n h_m \cos(n-m)\omega - 2A_k \sum_{n=0}^{N} h_n \cos(n-d_k)\omega + A_k^2\right] d\omega \\
&= \frac{1}{\pi} W_k \left[\omega \sum_{n=0}^{N} h_n^2 + \sum_{m=0}^{N}\sum_{m=0,m\neq n}^{N} h_n h_m \frac{\sin(n-m)\omega}{n-m} - 2A_k h_{d_k}\omega - 2A_k \sum_{n=0,n\neq d_k}^{N} h_n \frac{\sin(n-d_k)\omega}{n-d_k} + A_k^2\omega\right]_{\Omega_k}
\end{aligned}$$

In stop bands the nominal delay, d_k, is assumed to be zero. The Octave function *directFIRnonsymmetricEsqPW.m* calculates the piece-wise calculation of $\mathcal{E}^2$, and its gradients with respect to the coefficients, h_n, of a non-symmetric FIR filter.

Piece-wise mean-squared-error of the response of a symmetric FIR filter

If the distinct coefficients of the impulse response of an even order, N=2M, odd length, 2M+1, symmetric FIR filter are $\boldsymbol{h} = [h_0, \ldots, h_M]^\top$ then:

$$\int A(\omega)\, d\omega = h_M\omega + 2\sum_{n=0}^{M-1} h_n \frac{\sin(M-n)\omega}{M-n}$$

and:

$$\begin{aligned}
\int A(\omega)^2\, d\omega =& h_M^2\omega - 4h_M \sum_{n=0}^{M-1} h_n \frac{\sin(M-n)\omega}{M-n} \quad \ldots \\
&+ 2\sum_{n=0}^{M-1}\sum_{m=0}^{M-1} h_n h_m \frac{\sin(2M-m-n)\omega}{2M-m-n} + 2\sum_{n=0}^{M-1} h_n^2\omega + 2\sum_{n=0}^{M-1}\sum_{m=0,m\neq n}^{M-1} h_n h_m \frac{\sin(n-m)\omega}{n-m}
\end{aligned}$$

Similarly, the gradient of the mean-squared error with respect to the coefficients is:

$$\frac{\partial \mathcal{E}^2}{\partial h_n} = \frac{2}{\pi}\int_{\Omega_k} W_k \left[A(\omega) - A_k\right] \frac{\partial A(\omega)}{\partial h_n} d\omega$$

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 $\mathcal{E}^2$, and its gradients with respect to the coefficients, h_n, of a symmetric FIR filter.

Piece-wise mean-squared-error of the response of an anti-symmetric FIR filter

If the distinct coefficients of the impulse response of an even order, N=2M, odd length, 2M+1, anti-symmetric FIR filter are $\boldsymbol{h} = [h_0, \ldots, h_{M-1}]^\top$ then:

$$\int A(\omega)\, d\omega = -2 \sum_{n=0}^{M-1} h_n \frac{\cos(M-n)\,\omega}{M-n}$$

and:

$$\begin{aligned}
\int A(\omega)^2\, d\omega =& 2 \sum_{n=0}^{M-1} \sum_{m=0}^{M-1} h_n h_m \int 2 \sin(M-n)\,\omega \sin(M-m)\,\omega\, d\omega \\
=& 2 \sum_{n=0}^{M-1} \sum_{m=0}^{M-1} h_n h_m \int \cos(n-m)\,\omega - \cos(2M-n-m)\,\omega\, d\omega \\
=& 2 \sum_{n=0}^{M-1} h_n^2 \omega + 2 \sum_{n=0}^{M-1} \sum_{m=0, m\neq n}^{M-1} h_n h_m \frac{\sin(n-m)\,\omega}{n-m} - 2 \sum_{n=0}^{M-1} \sum_{m=0}^{M-1} h_n h_m \frac{\sin(2M-n-m)\,\omega}{2M-n-m}
\end{aligned}$$

Similarly, the gradient of the mean-squared error with respect to the coefficients is:

$$\frac{\partial \mathcal{E}^2}{\partial h_n} = \frac{2}{\pi} \int_{\Omega_k} W_k \left[A(\omega) - A_k\right] \frac{\partial A(\omega)}{\partial h_n} d\omega$$

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:

$$\int \frac{\partial A(\omega)}{\partial h_n} d\omega = -2 \frac{\cos(M-n)\,\omega}{M-n} \quad n = 0, \ldots, M-1$$

and:

$$\begin{aligned}
\int A(\omega) \frac{\partial A(\omega)}{\partial h_n} d\omega &= 2 \sum_{m=0}^{M-1} h_m \int 2 \sin(M-n)\,\omega \sin(M-m)\,\omega\, d\omega \\
&= 2 \sum_{m=0}^{M-1} h_m \int \cos(n-m)\,\omega - \cos(2M-n-m)\,\omega\, d\omega \\
&= 2h_n\omega + \sum_{m=0, m\neq n}^{M-1} 2h_m \frac{\sin(n-m)\,\omega}{n-m} - \sum_{m=0}^{M-1} 2h_m \frac{\sin(2M-n-m)\,\omega}{2M-n-m}
\end{aligned}$$

Piece-wise mean-squared-error of the response of an anti-symmetric FIR Hilbert filter

The piece-wise mean-squared-error of the response of an anti-symmetric FIR Hilbert filter is calculated in a similar fashion to that of a symmetric FIR filter. The Octave function *directFIRhilbertEsqPW* implements the piece-wise calculation of $\mathcal{E}^2$, and its gradients with respect to the coefficients, h_n, of an anti-symmetric FIR Hilbert filter.

N.2.4 Examples of the design of FIR filters with unconstrained optimisation

Design of FIR low-pass filters with unconstrained optimisation

The Octave script *qp_lowpass_filter.m* uses `qp` to design a low-pass filter with unconstrained optimisation. The filter specification is similar to a *Parks-McClellan* low-pass filter example [243, Table I]. The filter is designed as both a symmetric FIR filter and a non-symmetric filter with nominal delay $M/2$ samples. The filter specification is:

```
nplot=1000 % Frequency points in [0,0.5)
M=14 % Filter order is 2*M
fap=0.17265 % Amplitude pass band edge
Wap=10 % Amplitude pass band weight
Wat=0.01 % Amplitude transition band weight
fas=0.26265 % Amplitude stop band edge
Was=1 % Amplitude stop band weight
```

Figure N.12: Amplitude response of a symmetric FIR low-pass filter designed with unconstrained optimisation.

Figure N.12 shows the amplitude response of the symmetric FIR filter. The symmetric FIR filter coefficients are:

```
h = [ -0.0011416422, -0.0064608157,  0.0006753211,  0.0126461978, ...
       0.0066188281, -0.0171074193, -0.0218078604,  0.0130944715, ...
       0.0434468525,  0.0086576279, -0.0666514847, -0.0664970801, ...
       0.0846011495,  0.3038505994,  0.4086342531,  0.3038505994, ...
       0.0846011495, -0.0664970801, -0.0666514847,  0.0086576279, ...
       0.0434468525,  0.0130944715, -0.0218078604, -0.0171074193, ...
       0.0066188281,  0.0126461978,  0.0006753211, -0.0064608157, ...
      -0.0011416422 ]';
```

For comparison, Figure N.13 shows the response of a symmetric FIR filter designed with the *Parks-McClellan* algorithm shown in Appendix N.5.3 implemented by the Octave function *mcclellanFIRsymmetric*. The filter distinct filter coefficients are:

```
hPM = [ -0.0020635021, -0.0059268488, -0.0016828636,  0.0086378543, ...
         0.0092366641, -0.0095751683, -0.0221620427,  0.0023748967, ...
         0.0391203496,  0.0202373167, -0.0564064614, -0.0755218675, ...
         0.0694063134,  0.3072652182,  0.4257585421 ]';
```

Figure N.14 shows the amplitude and group delay responses of the non-symmetric low-pass FIR filter with nominal delay $M/2$ samples. The non-symmetric FIR filter coefficients are:

```
hd = [ -0.0064568376,  0.0176100874,  0.0172866593, -0.0343904802, ...
       -0.0617718242,  0.0535160208,  0.2862616171,  0.4308766500, ...
        0.3289847569,  0.0755113812, -0.0911385497, -0.0697468599, ...
        0.0256869785,  0.0532070232,  0.0067851239, -0.0310934802, ...
       -0.0197169928,  0.0101206833,  0.0190744260,  0.0043894934, ...
       -0.0112491933, -0.0104888468,  0.0024066303,  0.0097488053, ...
        0.0033589647, -0.0057520650, -0.0049210939,  0.0020153765, ...
        0.0033685316 ]';
```

Figure N.13: Amplitude response of a symmetric FIR low-pass filter designed with the Parks-McClellan algorithm.

Figure N.14: Amplitude and group delay responses of a non-symmetric FIR low-pass filter designed with unconstrained optimisation.

Design of FIR band-pass filters with unconstrained optimisation

The Octave script *qp_bandpass_test.m* uses qp to design a band-pass filter with unconstrained optimisation. The filter is designed as both a symmetric FIR filter and a non-symmetric filter with nominal delay $M/2$ samples. The filter specification is:

```
nplot=1000 % Frequency points in [0,0.5)
M=40 % Filter order is 2*M
fasl=0.15 % Amplitude lower stop band edge
fatlu=0.198 % Amplitude lower trans. band upper edge
fapl=0.2 % Amplitude pass band lower edge
faplu=0.22 % Amplitude pass band lower upper edge
fapul=0.248 % Amplitude pass band upper lower edge
fapu=0.25 % Amplitude pass band upper edge
fapul=0.252 % Amplitude upper trans. band lower edge
fasu=0.3 % Amplitude upper stop band edge
Wasl=1 % Amplitude lower stop band weight
Watll=0.1 % Amplitude lower trans. band lower weight
Watlu=0.2 % Amplitude lower trans. band upper weight
Wapl=50 % Amplitude pass band lower weight
Wap=25 % Amplitude pass band weight
Wapu=50 % Amplitude pass band upper weight
Watul=0.2 % Amplitude upper trans. band lower weight
Watuu=0.1 % Amplitude upper trans. band upper weight
Wasu=1 % Amplitude upper stop band weight
```

Figure N.15 shows the amplitude response of the symmetric FIR filter. The symmetric FIR filter coefficients are:

```
h = [  0.0002797241,  0.0001966438, -0.0051184756, -0.0030507204, ...
       0.0083029621,  0.0076040731, -0.0077933238, -0.0105252893, ...
       0.0040038965,  0.0084319756, -0.0004031126, -0.0008016642, ...
       0.0013917439, -0.0087819939, -0.0087382797,  0.0146288936, ...
       0.0195230614, -0.0129992630, -0.0275728720,  0.0050897229, ...
       0.0274947892,  0.0033885054, -0.0183007132, -0.0056051631, ...
       0.0041463802, -0.0025960747,  0.0078430098,  0.0208185918, ...
      -0.0107607236, -0.0445158874,  0.0003395738,  0.0665912752, ...
       0.0237907098, -0.0792893120, -0.0576197783,  0.0763654383, ...
       0.0931380177, -0.0555534438, -0.1202732967,  0.0203599593, ...
       0.1304482334,  0.0203599593, -0.1202732967, -0.0555534438, ...
       0.0931380177,  0.0763654383, -0.0576197783, -0.0792893120, ...
       0.0237907098,  0.0665912752,  0.0003395738, -0.0445158874, ...
      -0.0107607236,  0.0208185918,  0.0078430098, -0.0025960747, ...
       0.0041463802, -0.0056051631, -0.0183007132,  0.0033885054, ...
       0.0274947892,  0.0050897229, -0.0275728720, -0.0129992630, ...
       0.0195230614,  0.0146288936, -0.0087382797, -0.0087819939, ...
       0.0013917439, -0.0008016642, -0.0004031126,  0.0084319756, ...
       0.0040038965, -0.0105252893, -0.0077933238,  0.0076040731, ...
       0.0083029621, -0.0030507204, -0.0051184756,  0.0001966438, ...
       0.0002797241 ]';
```

For comparison, Figure N.16 shows the response of a symmetric FIR filter designed with the *Parks-McClellan* algorithm shown in Appendix N.5.3 implemented by the Octave function *mcclellanFIRsymmetric*. The filter distinct filter coefficients are:

```
hPM = [ -0.0002591487, -0.0000677819,  0.0006672837,  0.0004486165, ...
        -0.0009499685, -0.0009449698,  0.0007797988,  0.0009812455, ...
        -0.0001706585,  0.0002696197,  0.0000105558, -0.0028007959, ...
        -0.0013721390,  0.0052738461,  0.0042484613, -0.0055991453, ...
        -0.0064093894,  0.0031041926,  0.0044131281, -0.0001839318, ...
         0.0034185384,  0.0014126970, -0.0140704984, -0.0093663390, ...
         0.0205433660,  0.0202402741, -0.0171404788, -0.0237006715, ...
         0.0061908437,  0.0092929371, -0.0000064366,  0.0241280733, ...
         0.0144454529, -0.0633874581, -0.0569221215,  0.0862313026, ...
         0.1176241929, -0.0750230244, -0.1718287995,  0.0297516702, ...
         0.1935760345 ]';
```

Figure N.15: Amplitude response of a symmetric FIR band-pass filter designed with unconstrained optimisation.

Figure N.16: Amplitude response of a symmetric FIR band-pass filter designed with the Parks-McClellan algorithm.

Figure N.17: Amplitude and group delay responses of a non-symmetric FIR band-pass filter designed with unconstrained optimisation.

Figure N.17 shows the amplitude and group delay responses of the non-symmetric band-pass FIR filter with nominal delay $M/2$ samples. The non-symmetric FIR filter coefficients are:

```
hd = [  0.0209959120,  0.0029710155, -0.0258403597, -0.0111816405, ...
        0.0203805829,  0.0124262243, -0.0068877978,  0.0015174193, ...
       -0.0040679303, -0.0304439926,  0.0000783024,  0.0635196927, ...
        0.0240897613, -0.0851327111, -0.0623303419,  0.0843002153, ...
        0.1010009434, -0.0603564749, -0.1269646353,  0.0215676781, ...
        0.1334495245,  0.0205034096, -0.1206837022, -0.0557658519, ...
        0.0933574828,  0.0772722421, -0.0584543557, -0.0814151352, ...
        0.0243630121,  0.0687293235,  0.0004911822, -0.0447826868, ...
       -0.0106095483,  0.0189235730,  0.0062912357, -0.0002629860, ...
        0.0064003976, -0.0066786209, -0.0191758963,  0.0034208486, ...
        0.0260038145,  0.0047120561, -0.0253280145, -0.0122459823, ...
        0.0191664674,  0.0158348959, -0.0108754876, -0.0144322156, ...
        0.0036417374,  0.0086484680, -0.0000059034, -0.0004909753, ...
        0.0013846876, -0.0068356697, -0.0071369358,  0.0100765979, ...
        0.0141532421, -0.0079333076, -0.0181581630,  0.0023467017, ...
        0.0163815550,  0.0024617413, -0.0096094845, -0.0028642490, ...
        0.0017601802, -0.0015527223,  0.0028851819,  0.0077604049, ...
       -0.0026574992, -0.0116367667, -0.0006313178,  0.0109662811, ...
        0.0034449325, -0.0067478879, -0.0032969355,  0.0020118656, ...
        0.0002976662,  0.0005962550,  0.0035057852, -0.0004099441, ...
       -0.0060555132 ]';
```

Figure N.18: Amplitude response of an anti-symmetric FIR Hilbert filter designed with unconstrained optimisation.

Design of an anti-symmetric FIR Hilbert filter with unconstrained optimisation

The Octave script *qp_hilbert_filter.m* uses `qp` to design an anti-symmetric FIR Hilbert filter with unconstrained optimisation. The filter specification is:

```
nplot=1000 % Frequency points in [0,0.5)
M=40 % Filter order is (4*M)-1
fap=0.01 % Amplitude pass band edge
fas=0.49 % Amplitude stop band edge
```

Figure N.18 shows the amplitude response of the anti-symmetric FIR Hilbert filter. The distinct filter coefficients are:

```
hM = [  0.0002574103,  0.0003585881,  0.0004787633,  0.0006199268, ...
        0.0007841803,  0.0009737426,  0.0011909593,  0.0014383152, ...
        0.0017184501,  0.0020341802,  0.0023885252,  0.0027847429, ...
        0.0032263738,  0.0037172976,  0.0042618057,  0.0048646933, ...
        0.0055313780,  0.0062680520,  0.0070818794,  0.0079812534, ...
        0.0089761354,  0.0100785059,  0.0113029719,  0.0126675974, ...
        0.0141950568,  0.0159142667,  0.0178627437,  0.0200900946, ...
        0.0226633273,  0.0256751956,  0.0292578122,  0.0336058698, ...
        0.0390184331,  0.0459793040,  0.0553250507,  0.0686370642, ...
        0.0893056970,  0.1261483151,  0.2114995129,  0.6363837968 ]';
```

N.3 PCLS design of symmetric FIR digital filters with Lagrange multipliers

This section describes peak-constrained-least-squares (PCLS) design of non-symmetric FIR filters with optimisation by Lagrange multipliers and the exchange algorithm of *Selesnick*, *Lang* and *Burrus* [91]. In the following, I will almost always consider odd-length, even-order, symmetric FIR filters.

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:

$$\mathcal{E}^2 = \frac{1}{\pi}\int_0^{\pi} W(\omega)\left[A(\omega) - A_d(\omega)\right]^2 d\omega$$

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:

$$\mathcal{L}(\omega) = \mathcal{E}^2 - \sum_p \lambda_p\left[A(\omega_p) - L(\omega_p)\right] - \sum_q \lambda_q\left[U(\omega_q) - A(\omega_q)\right]$$

where λ_p and λ_q are the *Lagrange multipliers* for the amplitude constraints at the corresponding frequencies ω_p and ω_q. The method of Lagrange multipliers minimises $\mathcal{E}^2$ subject to the constraints by solving the following system of equations for h_n, λ_p and λ_q:

$$\begin{aligned} \frac{\partial \mathcal{E}^2}{\partial h_n} - \sum_p \lambda_p \frac{\partial A(\omega_p)}{\partial h_n} + \sum_q \lambda_q \frac{\partial A(\omega_q)}{\partial h_n} &= 0 \\ A(\omega_p) &= L(\omega_p) \\ A(\omega_q) &= U(\omega_q) \end{aligned}$$

where:

$$\frac{\partial \mathcal{E}^2}{\partial h_n} = \frac{2}{\pi}\int_0^{\pi} W(\omega)\left[A(\omega) - A_d(\omega)\right]\frac{\partial A(\omega)}{\partial h_n} d\omega$$

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:

$$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{G}^\top \\ \boldsymbol{G} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{h} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{D} \end{bmatrix} \tag{N.15}$$

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 with optimisation by the method of Lagrange multipliers

This section shows examples of the design of an even-order, symmetric, FIR filter by the exchange algorithm described by *Selesnick et al.* [92, p.498] and summarised in Section 8.1.2.

Figure N.19: Amplitude response of the initial FIR low-pass filter and the optimised low-pass FIR filter designed with Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

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.15 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:

```
ftol=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.19. 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.20 shows the initial response and the response obtained when one coefficient of the floating-point solution is rounded and the remaining coefficients are optimised with the pass-band ripple constraint relaxed.

Figure N.20: Amplitude response of the initial FIR low-pass filter and the optimised band pass FIR filter designed with one coefficient rounded and the remaining coefficients optimised by the Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

Figure N.21: Amplitude response of the initial FIR band-pass filter and the optimised band-pass FIR filter designed with 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
```

Figure N.21 shows the amplitude responses of the initial and optimised band-pass filters. 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.22 shows the initial response and the response obtained when three coefficients of the floating-point solution are rounded and the remaining coefficients are optimised with the stop-band ripple constraint relaxed.

Figure N.22: Amplitude response of the initial FIR band-pass filter and the optimised band-pass FIR filter designed with three coefficients rounded and the remaining coefficients optimised by Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

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:

$$\begin{aligned} \mathcal{E}^2 &= \frac{1}{\pi}\int_0^{\pi} W(\omega)\left[A(\omega)+1\right]^2 d\omega \\ A(\omega_p) &\geq L(\omega_p) \\ A(\omega_q) &\leq U(\omega_q) \end{aligned}$$

where $Ad(\omega) = -1$.

In a similar fashion to that shown in Appendix N.4, the method of *Lagrange multipliers* optimises the mean-squared-error by solving the following system of equations for λ_p, λ_q and the coefficients of an even order $4M-2$ Hilbert filter, h_{2k}, with $k = 0, 1, \ldots, M-1$:

$$\begin{aligned} \frac{\partial \mathcal{E}^2}{\partial h_{2k}} - \sum_p \lambda_p \frac{\partial A(\omega_p)}{\partial h_{2k}} + \sum_q \lambda_q \frac{\partial A(\omega_q)}{\partial h_{2k}} &= 0 \\ A(\omega_p) &= L(\omega_p) \\ A(\omega_q) &= U(\omega_q) \end{aligned}$$

where:

$$\frac{\partial \mathcal{E}^2}{\partial h_{2k}} = \frac{2}{\pi}\int_0^{\pi} W(\omega)\left[A(\omega)+1\right]\frac{\partial A(\omega)}{\partial h_{2k}} d\omega$$

and:

$$\frac{\partial A(\omega)}{\partial h_{2k}} = 2\sin\left(2M-2k-1\right)\omega$$

This system of equations can be expressed in matrix form with components given by: [e]

$$\int \frac{\partial A(\omega)}{\partial h_{2k}} d\omega = -2\frac{\cos(2M-2k-1)\omega}{(2M-2k-1)}$$

$$\int A(\omega)\frac{\partial A(\omega)}{\partial h_{2k}} d\omega = 2h_{2k}\omega + 2\sum_{l=0,l\neq k}^{M-1} h_{2l}\frac{\sin(2k-2l)\omega}{2k-2l} - 2\sum_{l=0}^{M-1} h_{2l}\frac{\sin(4M-2k-2l-2)\omega}{4M-2k-2l-2}$$

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.15 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:

```
ftol=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. Figure N.23 shows the amplitude responses of the initial and optimised Hilbert filters. 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));
```

[e] $2\sin x \sin y = \cos(x-y) - \cos(x+y)$

Figure N.23: Amplitude response of the initial FIR Hilbert filter and the optimised FIR Hilbert filter designed with Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

Figure N.24: Amplitude response of the initial band-pass FIR Hilbert filter and the optimised FIR Hilbert bandpass filter designed with Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

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:

```
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
```

Figure N.24 shows the amplitude responses of the initial and optimised band-pass Hilbert filters Figure N.25 shows the pass-band responses. 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.25: Pass-band amplitude response of the initial band-pass FIR Hilbert filter and the optimised FIR Hilbert bandpass filter designed with Lagrange multipliers and the exchange algorithm of *Selesnick et al.* .

N.4 PCLS design of non-symmetric FIR filters with SOCP

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* [91]. The results of this section are, of course, also applicable to the design of symmetric FIR filters.

N.4.1 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:

$$\begin{aligned}
|H(\omega)|^2 &= H(\omega)^* H(\omega) \\
&= \left(\sum_{l=0}^{N} h_l e^{\imath l\omega}\right)\left(\sum_{k=0}^{N} h_k e^{-\imath k\omega}\right) \\
&= \sum_{l=0}^{N}\sum_{k=0}^{N} h_l h_k \left(\cos l\omega \cos k\omega + \sin l\omega \sin k\omega\right) \\
&= \sum_{l=0}^{N}\sum_{k=0}^{N} h_l h_k \cos(l-k)\omega
\end{aligned}$$

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.4.2 Examples of the PCLS design of non-symmetric FIR filters with SOCP optimisation

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:

```
ftol=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.26 shows the amplitude response. Figure N.27 shows the pass band amplitude, phase error and delay responses. The pass band phase error is adjusted for the nominal delay.

Figure N.26: Amplitude response of a non-symmetric low pass FIR filter designed with SOCP and PCLS.

Figure N.27: 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:

```
ftol=0.001 % Tolerance on coefficient update vector
ctol=5e-06 % 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=0.4 % 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 = [   0.0039453635,  -0.0136185958,  -0.0096497272,   0.0001549866, ...
       -0.0133814752,  -0.0560385681,  -0.0585483181,   0.0480599739, ...
        0.1963028500,   0.2041356256,  -0.0017685707,  -0.2445554505, ...
       -0.2758389897,  -0.0777040606,   0.1187650188,   0.1318652324, ...
        0.0346976632,   0.0026743283,   0.0569172027,   0.0734009662, ...
       -0.0005984894,  -0.0716781090,  -0.0582539053,  -0.0083046136, ...
       -0.0050612333,  -0.0360015759,  -0.0304401251,   0.0173056157, ...
        0.0448046238,   0.0248979055,   0.0003394306,   0.0066495506, ...
        0.0204766062,   0.0086889143,  -0.0166939589,  -0.0228156877, ...
       -0.0084925153,   0.0008379466,  -0.0038137492,  -0.0071196862, ...
        0.0002353302,   0.0083315815,   0.0072991196,   0.0016228723, ...
       -0.0003629489,   0.0009605138,   0.0010239773,  -0.0008107291, ...
       -0.0017063160,  -0.0009387865,  -0.0001970149 ]';
```

Figure N.28 shows the amplitude response. Figure N.29 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 π radians phase shift or multiplication of the amplitude by -1).

Figure N.28: Amplitude response of a non-symmetric band pass FIR Hilbert filter designed with SOCP and PCLS.

Figure N.29: 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.5 Constrained mini-max error optimisation of FIR digital filters

N.5.1 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* [195, 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 [170, 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 rth 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* [170, 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.5.2 *Hofstetter*'s algorithm for mini-max FIR filter approximation

Hofstetter [51] 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] [51, Page 5] [243, Page 192] [102].

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.30 shows the pass-band and stop-band amplitude responses of the filter. Figure N.31 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.30: Response of a mini-max FIR low-pass filter designed with *Hofstetter*'s algorithm.

Figure N.31: Zeros of a mini-max FIR low-pass filter designed with *Hofstetter*'s algorithm.

Figure N.32: 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.32 shows the pass-band and stop-band amplitude responses of the filter. Figure N.33 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.33: 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.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:

```
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.34: 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.35: 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* [90] point out that whilst *Hofstetter*'s algorithm "produces equi-ripple filters with the specified δ_p and δ_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 ω_t.

Suppose $\omega_1, \ldots, \omega_{M+1}$ is the ordered reference set of frequencies in $[0, \pi]$ including ω_t. Let $\omega_1, \ldots, \omega_{q-1}$ be the frequencies in the pass-band, less than ω_t, and $\omega_{q+1}, \ldots, \omega_{M+1}$ be the frequencies in the stop-band, greater than ω_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 ω_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 π. 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 π 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 (ω_t, π) then remove π from S.
> 3. Otherwise, let ω_a be the extrema of $A(\omega)$ in $(0, \omega_t)$ closest to 0, and let ω_b be the extrema of $A(\omega)$ in (ω_t, π) closest to π. If $|A(0) - A(\omega_a)|\,\delta_s < |A(\pi) - A(\omega_b)|\,\delta_p$ then remove 0 from S, otherwise remove π 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.36 shows the pass-band and stop-band amplitude responses of the filter. Figure N.37 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.36: Response of a mini-max FIR low-pass filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*.

Figure N.37: 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.38 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.38: 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.39 shows the amplitude and phase responses of the Hilbert filter derived from the half-band filter shown in Figure N.38. The phase response shown is adjusted for the nominal delay. Figure N.40 shows the zeros of the Hilbert filter.

Figure N.39: 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.40: 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* [90, Section II.C] also describe an exchange algorithm for band-pass filters with lower stop-band, pass-band and upper stop-band errors δ_{sl}, δ_p and δ_{su} and lower and upper transition-band frequencies ω_{tl} and ω_{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 ω_1 is a local maximum and $c = 0$ if ω_1 is a local minimum. $E(\omega)$ denotes the error function. ... If $1 < k < M-2$, then exclude ω_k and ω_{k+1} from the reference set. If $k = 1$ or $K = L$, then exclude ω_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* [224], 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.41 shows the pass-band and stop-band amplitude responses of the filter. Figure N.42 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.41: Response of a mini-max FIR band-pass filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*.

Figure N.42: Zeros of a mini-max FIR band-pass filter designed with *Hofstetter*'s algorithm and the modifications of *Selesnick* and *Burrus*.

N.5.3 *Parks-McClellan* algorithm for mini-max FIR filter approximation

Parks and *McClellan* [243, 96][170, 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 ω_p, ω_s and the ratio δ_p/δ_s but does not allow direct specification of δ_p and δ_s. *Parks* and *McClellan* [243] and *Oppenheim* and *Schafer* [170, 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 [243, 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.43 shows the pass-band and stop-band amplitude responses of the filter. Figure N.44 shows the zeros of the filter. The stop-band ripple found is ρ =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 ρ (eg: [170, 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 [224], *Selesnick* avoids this difficulty by directly calculating ρ and the coefficients of $A(x)$ by left-division.

Algorithm N.2 Parks-McClellan FIR filter design [170, Section 7.4.3] [243, 96].

1. The optimum filter satisfies the equations:

$$W(\omega_k)\left[A_d(\omega_k) - A(\omega_k)\right] = (-1)^{k+1}\rho\,, \quad k = 1, 2, \ldots, (M+2) \tag{N.16}$$

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 $\omega_s \le \omega_k \le \pi$ [170, 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, ω_p and ω_s. If $\omega_k = \omega_p$ then $\omega_{k+1} = \omega_s$.

3. The required values at the extremal frequencies are given by [170, 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 [170, 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 [170, 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 [170, Equation 7.103b]

$$C_k = A_d(x) - \frac{(-1)^{k+1}\rho}{W(x_k)}$$

and [170, 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* [195, 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.43: Response of a mini-max FIR low-pass filter designed with the *Parks-McClellan* algorithm [243, Table I].

Figure N.44: Zeros of a mini-max FIR low-pass filter designed with the *Parks-McClellan* algorithm [243, Table I].

Figure N.45: Response of a mini-max FIR band-pass filter designed with the *Parks-McClellan* algorithm [243, 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.45 shows the pass-band and stop-band amplitude responses of the filter. Figure N.46 shows the zeros of the filter. The stop-band ripple found is ρ =-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.46: Zeros of a mini-max FIR band-pass filter designed with the *Parks-McClellan* algorithm [243, 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.47 shows the amplitude response of the filter. Figure N.48 shows the zeros of the filter. The pass-band ripple found is ρ =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.47: Response of a mini-max FIR multi-band filter designed with the *Parks-McClellan* algorithm [243, Table I].

Figure N.48: Zeros of a mini-max FIR multi-band filter designed with the *Parks-McClellan* algorithm [243, Table I].

Figure N.49: 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 ρ =-0.00103223 . Figure N.49 shows the amplitude and phase responses of the filter. Figure N.50 shows the amplitude error responses of the filter.

Figure N.50: 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 [90, Section II.A] that allow either the pass-band or stop-band ripple, δ_p or δ_s, to be fixed and the other minimised. The user specifies M, ω_p, ω_s, K_p, K_s, η_p and η_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 η_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, δ_p, is minimised.

The linear system of equations to be solved at each iteration is, in addition to the affine equations for δ_p and δ_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 δ_p or δ_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 ω_p and ω_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 π. 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 π 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 π 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. α and β 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 [224]. 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
```

Figure N.51: Response of a mini-max FIR low-pass filter designed with the *Parks-McClellan* algorithm modified by *Selesnick* and *Burrus* [90, Section II.A].

The resulting filter has δ_p =0.00644777 . Figure N.51 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.6 Design of FIR filters with maximally-linear pass-bands and equi-ripple stop-bands using the *Parks-McClellan* algorithm

Vaidyanathan [175] and *Selesnick* and *Burrus* [89] 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* [175, Figure 4] as [89, 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.6.1 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, ω_s and minimises δ_s
2. specifies N, L, δ_s and minimises ω_s

Maximally-flat pass-band and fixed ω_s

In the first case, $A_M(\omega)$ is found by minimising [89, 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.52: Response of a mini-max FIR maximally-flat pass-band low-pass filter with fixed ω_s designed with the *Parks-McClellan* algorithm [89, 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 δ_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 ω_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.52 shows the pass-band and stop-band amplitude responses of the fixed ω_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 δ_s

In the second case [89, 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 δ_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.53 shows the pass-band and stop-band amplitude responses of the fixed δ_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.53: Response of a mini-max FIR maximally-flat pass-band low-pass filter with fixed δ_s [89, Section II.B].

N.6.2 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, ω_p, pass-band width $2\omega_t$ and upper and lower stop-band ripple δ_{su} and δ_{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, ω_p, ω_t, $K = \delta_{su}/\delta_{sl}$ and minimises δ_{sl}
2. specifies N, L, δ_{su} and δ_{sl} and maximises ω_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 ω_p and ω_t

For convenience, set $\omega_a = \omega_p - \omega_t$ and $\omega_b = \omega_p + \omega_t$. In this case [89, 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 ω_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 π as follows: if $|A(\pi)| < K|A(0)|$ then remove π, otherwise remove 0. Next append either ω_a or ω_b to R: if $|A(\omega_b)| < K|A(\omega_a)|$ then append ω_a, otherwise append ω_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 ω_p and ω_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.54 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.54: Pass-band and stop-band responses of a mini-max FIR maximally-flat pass-band band-pass filter with fixed ω_p and ω_t [89, Section III].

Maximally-flat pass-band and fixed δ_{su} and δ_{sl}

In the second case [89, Section III]:

> A similar algorithm is used for approach (2) in which δ_{sl} and δ_{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 ω_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 δ_{su} and δ_{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.55 shows the pass-band and stop-band amplitude responses of the fixed δ_{su} and δ_{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.55: Response of a mini-max FIR maximally-flat pass-band band-pass filter with fixed δ_{su} and δ_{sl} [89, 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.6.3 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* [89, 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_linear_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* [89, Figure 6]:

```
N=59 % Filter length
L=34 % Filter linearity
fs=0.2 % Amplitude stop band frequency
```

Figure N.56 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.57 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.56: Response of the A_M filter of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator [89, Section IV].

Figure N.57: Response of the failed design of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator [89, Section IV].

Design of an odd length differentiator with *mcclellanFIRantisymmetric_linear_differentiator*

The Octave function *mcclellanFIRantisymmetric_linear_differentiator* designs maximally-linear pass-band, equi-ripple stop-band, FIR differentiator filters. It follows the MATLAB function *chebdiff.m* of *Selesnick* [225] for even-length filters and adds the odd-length case. The Octave script *mcclellanFIRantisymmetric_linear_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* [89, Figure 6]. The specification of the odd-length filter is:

```
N=57 % Filter length
L=34 % Filter linearity
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.58 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.59 shows the zeros of the differentiator filter. There are single zeros at $z = \pm 1$. Figure N.60 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_linear_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.58: Pass-band and stop-band amplitude responses of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator [89, Section IV].

Figure N.59: Zeros of an odd-length, even-order, maximally-linear pass-band, equi-ripple stop-band, FIR differentiator [89, Section IV].

Figure N.60: 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_linear_differentiator*.

N.7 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 [221]. They emphasise their use since the late 19th century by actuaries for *graduation* or data-smoothing. Maximally-linear FIR filters are typically longer than equi-ripple FIR filters but may have fewer multipliers.

N.7.1 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 [161] 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* [127] 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* [64] 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* [127, Appendix A] derive *Herrmann*'s $P_{M,K}(x)$:

$$B_{M,K}(x) = (1-x)^K \sum_{k=0}^{M-K} \binom{M+k-1}{k} x^k \tag{N.17}$$

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 kth 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 [127, 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.61 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 [161].

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.61: x-domain amplitude responses of maximally-flat low-pass filters designed by the method of *Herrmann* with $M = 10$ and $K = 1, \ldots, M$ [161, Figure 1].

***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. [150] 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 [150, 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.17 in terms of v [150, 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:

$$\frac{d}{dv} C_{M,K}(v) = 2^{-M} M \binom{M-1}{M-K} (1-v)^{M-K} (1+v)^{K-1} \tag{N.18}$$

Differentiating again [150, 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.18 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 [150, 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* [161] or *Rajagopal* and *Roy* [127]. 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.62 shows the amplitude responses of maximally-flat low-pass filters with $M = 300$ and $K = 10, 30, \ldots, 290$.

Figure N.62: Amplitude responses of maximally-flat low-pass filters designed by the backwards recursion of *Vlček et al.* [150, Table 1] with $M = 300$ and $K = 10, 30, \ldots, 290$.

Working for *Vlček et al.* Equation 14 [150] 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:

$$\begin{aligned} \left(1 - v^2\right)\frac{d}{dv}U_{k-1}(v) &= vU_{k-1}(v) - kT_k \\ &= vU_{k-1}(v) - kU_k(v) + kvU_{k-1}(v) \\ &= (k+1)\,vU_{k-1}(v) - kU_k(v) \\ &= (k+1)\left(\frac{U_k(v) + U_{k-2}(v)}{2}\right) - kU_k(v) \\ &= -\left(\frac{k-1}{2}\right)U_k(v) + \left(\frac{k+1}{2}\right)U_{k-2}(v) \end{aligned}$$

Closed-form design of FIR filters by *Nuevo et al.*'s interpolation with a maximally-flat model filter

Nuevo et al. [265] 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.63 shows the responses of the interpolated maximally-flat model filter and the interpolator filter. Figure N.64 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.65 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.63: Amplitude response of the interpolated maximally-flat model filter and interpolator filter with $P = 8$, $M = 19$, $fc = 0.2$, $K = 12$.

Figure N.64: Amplitude response of an interpolated maximally-flat band-pass filter with $P = 8$, $M = 19$, $fc = 0.2$, $K = 12$.

Figure N.65: 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.

***Vaidyanathan*'s design of multiplier-less FIR filters by combining maximally-flat interpolators**

Vaidyanathan [174] describes the design of monotonic (rather than maximally-flat) FIR filters by combining maximally-flat interpolator blocks. The filters are designed with "closed-form" formulas and coefficient multiplications are implemented with shift-and-add arithmetic.

Vaidyanathan [174, Section II] first reviews the properties of *maximally-flat* symmetric, even-order, N, FIR filters. He writes the zero-phase transfer function as:

$$H_0(z) = h_0 z^{\frac{N}{2}} + h_1 z^{\frac{N}{2}-1} + \ldots + h_N z^{-\frac{N}{2}}$$

The associated linear-phase filter is $H(z) = z^{-\frac{N}{2}} H_0(z)$. The corresponding frequency response is:

$$H_0(e^{\imath\omega}) = h_{\frac{N}{2}} + 2\sum_{k=1}^{\frac{N}{2}} h_{\frac{N}{2}-k} \cos k\omega$$

The filter is maximally-flat if:

$$\left.\frac{\partial^k H_0(e^{\imath\omega})}{\partial\omega^k}\right|_{\omega=0} = 0, \quad \text{for } k = 1, 2, \ldots, 2L-1$$
$$\left.\frac{\partial^k H_0(e^{\imath\omega})}{\partial\omega^k}\right|_{\omega=\pi} = 0, \quad \text{for } k = 1, 2, \ldots, 2K-1$$

where $N := 2(K+L-1)$. K represents the degree of flatness at $\omega = 0$ and L represents the degree of flatness at $\omega = \pi$. *Herrmann* [161] and *Rajagopal* and *Roy* [127] show a solution for $H_0(e^{\imath\omega})$ by *Hermite interpolation*:

$$H_0(e^{\imath\omega}) = \left(\cos\frac{\omega}{2}\right)^{2K} \sum_{k=0}^{L-1} d(k) \left(\sin\frac{\omega}{2}\right)^{2k}$$

where:

$$d(k) = \frac{(K-1+k)!}{(K-1)!k!}$$

The corresponding filter transfer functions are:

$$H_0(z) = \left(\frac{1}{2} + \frac{1}{2}\frac{z+z^{-1}}{2}\right)^K \sum_{k=0}^{L-1} d(k) \left(\frac{1}{2} - \frac{1}{2}\frac{z+z^{-1}}{2}\right)^k$$
$$H(z) = \left(\frac{1+z^{-1}}{2}\right)^{2K} \sum_{k=0}^{L-1} z^{-(L-1-k)} (-1)^k d(k) \left(\frac{1-z^{-1}}{2}\right)^{2k}$$

Vaidyanathan points out that L non-trivial multipliers are needed to implement this transfer function and that these multipliers have a large dynamic range. In addition, the usual design of a maximally-flat linear-phase FIR filter begins with the specification of the centre of the transition band, β, and the width of the transition band, δ, both in multiples of π, and computes the K and L required so that the resulting filter has a gain of 0.95 at $\omega = \left(\beta - \frac{\delta}{2}\right)\pi$ and 0.05 at $\omega = \left(\beta + \frac{\delta}{2}\right)\pi$. The order of such a filter, $N = 2(K+L-1)$ grows as δ^2. Instead, *Vaidyanathan* employs the interpolated FIR method of *Nuevo et al.* [265], shown in Section N.7.1.

The interpolated FIR design method of *Nuevo et al* first designs a filter with $\hat{\beta} = M\beta$ and $\hat{\delta} = M\delta$, where M is an integer. This filter transfer function is expanded with $M-1$ delays between each coefficient and the unwanted pass-bands are suppressed by a maximally-flat interpolator. *Vaidyanathan* proposes the following efficient interpolators suitable for $M = 2$. Other values of M are obtained by combining these interpolators. In the following $C(\omega) = \cos^2\frac{\omega}{2}$ and $S(\omega) = \sin^2\frac{\omega}{2}$.

1. Interpolator $I(z)$:

$$\beta = 0.5,\ \delta = 0.4,\ K = 3,\ L = 3,\ N = 10$$
$$I(e^{\imath\omega}) = C^3(\omega)\left(1 + 3S(\omega) + 6S^2(\omega)\right)$$

2. Interpolator $J(z)$:

$$\beta = 0.6,\ \delta = 0.5,\ K = 2,\ L = 4,\ N = 10$$
$$J\left(e^{\imath\omega}\right) = C^2(\omega)\left(1 + 2S(\omega) + 3S^2(\omega) + 4S^3(\omega)\right)$$

3. Interpolator $K(z)$:

$$\beta = 0.4,\ \delta = 0.4,\ K = 4,\ L = 2,\ N = 10$$
$$K\left(e^{\imath\omega}\right) = C^4(\omega)\left(1 + 4S(\omega)\right)$$

4. Interpolator $L(z)$:

$$\beta = 0.5,\ \delta = 0.5,\ K = 2,\ L = 2,\ N = 6$$
$$L\left(e^{\imath\omega}\right) = C^2(\omega)\left(1 + 2S(\omega)\right)$$

Vaidyanathan now points out that since $0 \leq C\left(e^{\imath\omega}\right) \leq 1$ then each $C(z)$ can be replaced with a function $F(z)$ for which $0 \leq F\left(e^{\imath\theta}\right) \leq 1,$ for all θ. This constitutes a frequency transformation for which

$$\omega = 2\cos^{-1}\left\{\left[F\left(e^{\imath\theta}\right)\right]^{\frac{1}{2}}\right\}$$

If $F(z)$ is linear-phase then the resulting filter is linear-phase. For example, if[m]:

$$I_I(z) := I(z)\big|_{C(z)\leftarrow I^2(z)}$$

then $I_I(z)$ has a much sharper cutoff than $I(z)$ itself.

The Octave script *vaidyanathanFIRsymmetric_lowpass_test.m* calculates the amplitude response of *Vaidyanathan*'s example $H_5(z)$ [174, Figure 12]:

$$H_5(z) = I_I(z)\, I_I\left(z^2\right) J_J\left(z^4\right)$$

Figure N.66 shows the amplitude response for the $H_5(z)$ example FIR filter.

Vaidyanathan states that this order $N = 700$ multiplier-less FIR filter has an amplitude response equivalent to that of an $N = 174$ equi-ripple FIR filter but with far fewer arithmetic operations.

[m] Similarly $S(z) \leftarrow 1 - I^2(z)$.

Figure N.66: Amplitude response of *Vaidyanathan*'s example $H_5(z)$ of an interpolated multiplier-less low-pass filter [174, Figure 12].

N.7.2 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 [34, 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 [34, 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.67 shows the amplitude responses of the resulting half-band filters. Figure N.68 shows the amplitude responses of the corresponding Hilbert filters.

Figure N.67: Amplitude responses of maximally-flat half-band FIR filters designed by the method of *Gumacos* [34] with $M = 5, 10, 15, 20$ and 25.

Figure N.68: Amplitude responses of maximally-flat Hilbert FIR filters derived from half-band FIR filters designed by the method of *Gumacos* [34] with $M = 5, 10, 15, 20$ and 25.

N.7.3 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* [228] 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 [228, 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:

$$\operatorname{sign} x = \frac{x}{\sqrt{c}} \left[1 + \sum_{k=1}^{\infty} \frac{(2k-1)!!}{(2k)!!} \left(1 - \frac{x^2}{c}\right)^k \right] \tag{N.19}$$

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* [228, Figure 2] show a realisation of this order $4M + 2$ transfer function with M coefficients of the form $\frac{2k-1}{2k}$.

Figure N.69: Amplitude responses of *Pei* and *Wang* [228, 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.69 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.19 follows.

N.7.4 Closed-form design of maximally-linear FIR low-pass differentiators

Kumar and *Roy* [23] and *Selesnick* [88] describe the closed-form design of maximally-linear low-pass FIR differentiator filters. *Yoshida et al.* [244] 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.* [87] 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* [106] extend the work of *Kumar et al.* [24] to show formulas for the weighting coefficients of FIR differentiators that are maximally-linear at frequencies π/p, where p is a positive integer.

The derivation of the coefficients of the maximally-linear FIR differentiator begins with the constraints on the zero-phase amplitude response, $A(\omega)$. When the constraint is at $\omega = 0$ [88, 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. [24] define similar constraints at $\omega = \frac{\pi}{p}$. *Selesnick* [88, 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. [244, 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* [23, 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 [23, 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 [23, 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 [88, 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 [88, Section IV] gives the following equation for the z-domain transfer function of *Type III*[n] 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_linear_differentiator_test.m* calls the Octave function *selesnickFIRantisymmetric_linear_differentiator* to design a series of differentiator filters. Figure N.70 shows the differentiator filters for length $N = 30$ and $K = 0, 4, 8, 12, 16, 20$ and 24 [88, Figure 1]. Figure N.71 shows the differentiator filters for length $N = 31$ and $K = 1, 5, 9, 13, 17, 21$ and 25 [88, Figure 2].

[n] *Selesnick* uses the terminology of *Oppenheim* and *Schafer* [170, 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.70: 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 [88, Figure 1].

Figure N.71: 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 [88, Figure 2].

Figure N.72: 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.* [24] and with the recursive calculation of the d_k performed by the method of *Purczyński* and *Pawelczak* [106, 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. [24, Equation 9][o] 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 [24, Figure 1]. *Purczyński* and *Pawelczak* solve the resulting system of constraint equations algebraically to find the following recursive closed-form solutions for the coefficients [106, 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_linear_differentiator_test.m* calls the Octave function *purczynskiFIRantisymmetric_linear_differentiator* to design FIR differentiators that are maximally linear at $\omega = \frac{\pi}{p}$ where p is a multiple of 2. Figure N.72 shows the amplitude responses of the differentiators for $p = 2, 4, 8$ and 16. Figure N.73 shows the corresponding amplitude errors, $|H_p(\omega)| - \omega$, of the differentiators.

[o] See also *Khan et al.* [87, Equation 1] for design of differentiators that are maximally-linear at $\omega = \frac{\pi}{2}$

Figure N.73: 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.* [24] and with the recursive calculation of the d_k performed by the method of *Purczyński* and *Pawelczak* [106, Equation 7].

N.8 Linear Matrix Inequality(LMI) design of symmetric FIR filters

This section follows the peak-constrained LMI design of even-order symmetric FIR filters described by *Tuan et al.* [76]. They use arguments from convex optimisation to derive results similar to those of *Davidson et al.* [237]. 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.8.1 The *Markov-Lukacs* theorem

Després [19] shows the following version of the *Markov-Lukacs* theorem:

P_n is the set of nth 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.8.2 Trigonometric curves

Tuan et al. [76, 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.8.3 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 [76, 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 δ_m is the *Kronecker delta function*. Thus:

$$T_{l,k}(y) = \sum_{m=0}^{l+2k} y_m \mathcal{E}^m_{l,k}$$

N.8.4 A *Markov-Lukacs* theorem for trigonometric curves

Tuan et al [75, 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:

$$P(t) = \begin{cases} \langle X, \mathcal{M}_k(t)\rangle + (b-t)(t-a)\langle Z, \mathcal{M}_{k-1}(t)\rangle, & n = 2k \\ (t-a)\langle X, \mathcal{M}_k(t)\rangle + (b-t)\langle Z, \mathcal{M}_{k-1}(t)\rangle, & n = 2k+1 \end{cases} \tag{N.20}$$

where there exist $X, Z \succeq 0$.

Tuan et al [76, 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:

$$P(\omega) = \begin{cases} \langle X, \mathcal{T}_k(\omega)\rangle + \langle Z, \mathcal{F}_k^{a,b}(\omega)\rangle, & n = 2k \\ \langle \cos b\, Z - \cos a\, X, \mathcal{T}_k(\omega)\rangle + \langle X - Z, \mathcal{T}_{1,k1}(\omega)\rangle, & n = 2k+1 \end{cases} \tag{N.21}$$

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.21 is found by substitution of $\mathcal{M}_k(\cos\omega)$ into Equation N.20.

Tuan et al [76, Lemma 1] prove the following lemma:

If $P(\omega) = \varphi_n^\top(\omega)\, x$ can be represented as shown in Equation N.21, 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.8.5 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* [76, 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:

$$\begin{cases} T_k(y) \succeq 0, F_k^{a,b}(y) \succeq 0, & n = 2k \\ \cos b\, T_k(y) \succeq T_{1,k}(y) \succeq \cos a\, T_k(y), & n = 2k+1 \end{cases} \tag{N.22}$$

The convex hull of the set $C_{a,b}$ is characterised by the LMI constraints:

$$\operatorname{cohull} C_{a,b} = \{(y_0, y_1, \cdots, y_n) \ : \ \text{Equation } N.22, \ y_0 = 1\}$$

The conic hull of $C_{a,b}$ is defined by:

$$\operatorname{cone} C_{a,b} = \{(y_0, y_1, \cdots, y_n) \ : \ \text{Equation } N.22\}$$

Substituting the $\mathcal{E}_{l,k}^m$ basis matrixes [76, 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:

$$x_m = \begin{cases} \langle X, \mathcal{E}_{0,k}^m \rangle + \langle Z, \mathcal{F}_{m,k}^{a,b} \rangle, & n = 2k \\ \langle X, \mathcal{E}_{1,k}^m \rangle - \cos a\, \mathcal{E}_{0,k}^m + \langle Z, \cos b\, \mathcal{E}_{0,k}^m - \mathcal{E}_{1,k}^m \rangle, & n = 2k+1 \end{cases} \tag{N.23}$$

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:

$$e_{m+1}^\top (A_i x + d_i) = \begin{cases} \langle X_i, \mathcal{E}_{0,k}^m \rangle + \langle Z_i, \mathcal{F}_{m,k}^{a_i,b_i} \rangle, & n = 2k \\ \langle X_i, \mathcal{E}_{1,k}^m - \cos a_i\, \mathcal{E}_{0,k}^m \rangle + \langle Z_i, \cos b_i\, \mathcal{E}_{0,k}^m - \mathcal{E}_{1,k}^m \rangle, & n = 2k = 1 \end{cases} \tag{N.24}$$

where $X_i, Z_i \succeq 0$.

N.8.6 Optimisation of the dual of a convex quadratic objective function

The convex quadratic optimisation problem over a set of trigonometric polynomials is:

$$\begin{aligned} \textbf{minimise} \quad & x^\top Q x + q^\top x \\ \textbf{subject to} \quad & \text{Equation } N.24 \end{aligned}$$

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.22 the dual problem becomes:

$$\begin{aligned}
&\textbf{maximise} && \left[-\sum_i d_i^\top y^i - \frac{1}{4}\left(q-\sum_i A_i^\top y^i\right)^\top Q^{-1}\left(q-\sum_i A_i^\top y^i\right)\right] \\
&\textbf{subject to} && \begin{cases} T_k\left(y^i\right) \succeq 0, F_k^{a_i,b_i}\left(y^i\right) \succeq 0, & n=2k \\ \cos b_i\, T_k\left(y^i\right) \succeq T_{1,k}\left(y^i\right) \succeq \cos a_i\, T_k\left(y^i\right), & n=2k+1 \end{cases}
\end{aligned} \tag{N.25}$$

In SDP form:

$$\begin{aligned}
&\textbf{maximise} && -\nu - \sum_i d_i^\top y^i \\
&\textbf{subject to} && \begin{bmatrix} \nu & \left(q-\sum_i A_i^\top y^i\right)^\top \\ \left(q-\sum_i A_i^\top y^i\right) & 4Q \end{bmatrix} \succeq 0 \\
& && \text{Equation } N.25
\end{aligned}$$

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:

$$\begin{aligned}
\sum_i \left(A_i x_\star + d_i\right)^\top y_\star^i &= 0 \\
x_\star &= -\frac{1}{2}Q^{-1}\left(q-\sum_i A_i^\top y_\star^i\right)
\end{aligned}$$

N.8.7 Example of LMI design of a low pass symmetric FIR filter

The low pass filter design primal problem is:

$$\begin{aligned}
\textbf{minimise} \quad & x^\top Q x + q^\top x \\
\textbf{subject to} \quad & x + (\delta_p - 1)\, e_1 \in C^\star_{\omega_p,0} \\
& -x + (\delta_p + 1)\, e_1 \in C^\star_{\omega_p,0} \\
& x + \delta_s e_1 \in C^\star_{\pi,\omega_s} \\
& -x + \delta_s e_1 \in C^\star_{\pi,\omega_s}
\end{aligned}$$

The Octave script *directFIRsymmetric_sdp_lowpass_test.m* uses YALMIP and SeDuMi to solve the dual problem of designing a low pass filter by the method of *Tuan et al.* [76]. I only allow solutions for which YALMIP and SeDuMi do not report numerical problems. The filter specification is[p]:

```
M=31 % M+1 distinct coefficients, FIR filter order 2*M
fap=0.05 % Amplitude pass band edge
fas=0.1 % Amplitude stop band edge
dBap=0.04 % Amplitude pass band peak-to-peak ripple(dB)
deltap=0.00230258 % Amplitude pass band peak ripple
dBas=60 % Amplitude stop band peak ripple(dB)
deltas=0.001 % Amplitude stop band peak ripple
Wap=1 % Amplitude pass band weight
Wat=1 % Amplitude trans. band weight
Was=1 % Amplitude stop band weight
```

Figure N.74 shows the zero-phase amplitude responses in the pass band and the stop band.

In addition, the Octave script *directFIRsymmetric_sdp_lowpass_test.m* attempts to design a low pass filter with a specification similar to that of the example of *Tuan et al.* [76, Figure 3]:

[p] *Tuan et al.* define the pass-band peak-to-peak ripple as:

$$dBap = 20\log_{10}\frac{1+\delta_p}{1-\delta_p}$$

Figure N.74: Pass band and stop band zero-phase amplitude responses of an $M = 31$ symmetric low pass FIR filter designed with the LMI procedure of *Tuan et al.* [76].

```
M=200 % M+1 distinct coefficients, FIR filter order 2*M
fap=0.03 % Amplitude pass band edge
fas=0.0364 % Amplitude stop band edge
dBap=0.3 % Amplitude pass band peak-to-peak ripple(dB)
deltap=0.0172677 % Amplitude pass band peak ripple
dBas=60 % Amplitude stop band peak ripple(dB)
deltas=0.001 % Amplitude stop band peak ripple
Wap=1 % Amplitude pass band weight
Wat=1 % Amplitude trans. band weight
Was=1 % Amplitude stop band weight
```

I could not reproduce the stop-band edge specification of $fas = 0.0358$ given by *Tuan et al.* [76, Table 1, Figure 3]. The amplitude response specification $dBap = 0.5$, $fas = 0.0358$ gives a "reasonable" response despite numerical warning messages from SeDuMi.

Figure N.75 shows the zero-phase amplitude responses in the pass band and the stop band.

For comparison, the Octave script *mcclellanFIRsymmetric_lowpass_alternate_test.m* uses the *Parks-McClellan* algorithm [243] 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
Wap=2 % Amplitude pass band weight
Was=1 % Amplitude stop band weight
K=50 % Stop band weight
ngrid=4000 % Number of frequency grid points in [0,0.5]
```

Figure N.76 shows the pass band and stop band zero-phase amplitude responses.

As another comparison, the Octave script *selesnickFIRsymmetric_lowpass_alternate_test.m* uses *Hofstetter*'s algorithm [51] with the *Selesnick-Burrus* modification [90], implemented in the Octave function *selesnickFIRsymmetric_lowpass*, to design an FIR filter similar to that of *Tuan et al.*'s Figure 4:

Figure N.75: 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.* [76, Figure 3].

Figure N.76: Pass band and stop band zero-phase amplitude 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.* [76, Figure 3].

Figure N.77: 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.* [76, Figure 4].

```
M=600 % Filter order is 2*M
fap=0.1 % Amplitude pass band edge
deltap=0.0172677 % Amplitude pass band ripple
dBap=0.3 % Amplitude pass band ripple(dB)
fas=0.10322 % Amplitude stop band edge
dBas=110 % Amplitude stop band ripple(dB)
deltas=3.16228e-06 % Amplitude stop band ripple
ngrid=6000 % Number of frequency grid points in [0,0.5]
maxiter=200 % Maximum number of iterations
tol=1e-12 % Tolerance on convergence
```

Figure N.77 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 $f_{ap} = 0.100033$.

N.8.8 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 symmetric FIR filter with the specification:

```
M=80 % M+1 distinct coefficients, FIR filter order 2*M
fasl=0.15 % Amplitude lower stop band edge
fapl=0.175 % Amplitude lower pass band edge
fapu=0.275 % Amplitude upper pass band edge
fasu=0.3 % Amplitude upper stop band edge
dBap=0.00868589 % Amplitude pass band peak-to-peak ripple(dB)
deltap=0.0005 % Amplitude pass band peak ripple(dB)
dBas=60 % Amplitude stop band peak ripple(dB)
deltas=0.001 % Amplitude stop band peak ripple(dB)
Wasl=1 % Amplitude lower stop band weight
Watl=1 % Amplitude lower trans. band weight
Wap =4 % Amplitude pass band weight
Watu=1 % Amplitude upper trans. band weight
Wasu=1 % Amplitude upper stop band weight
```

Figure N.78: Pass band and stop band zero-phase amplitude responses of an $M = 80$ symmetric band pass FIR filter designed with the LMI procedure of *Tuan et al.* [76].

In this case the frequency bands are not weighted (i.e.: $Wasl = 1$, etc.).

The distinct filter coefficients are:

```
hM80 = [  -0.0000639868,   0.0000221684,   0.0000757608,   0.0000592824, ...
          -0.0001046901,  -0.0001203408,   0.0000395215,   0.0000590722, ...
           0.0000446941,   0.0001864170,   0.0000021759,  -0.0004701338, ...
          -0.0002148945,   0.0005010252,   0.0003376415,  -0.0001742989, ...
          -0.0000001835,  -0.0001939294,  -0.0007952585,   0.0001420287, ...
           0.0014290542,   0.0003001962,  -0.0011579110,  -0.0003890061, ...
           0.0000116615,  -0.0005933594,   0.0009803636,   0.0022385215, ...
          -0.0008235797,  -0.0029527332,  -0.0000882050,   0.0014832370, ...
           0.0000592147,   0.0013933848,   0.0019472145,  -0.0032015226, ...
          -0.0044622691,   0.0023578834,   0.0043458598,  -0.0003393307, ...
          -0.0001746707,   0.0004461723,  -0.0053681461,  -0.0037493640, ...
           0.0074877865,   0.0067065508,  -0.0044104101,  -0.0041298556, ...
           0.0000345275,  -0.0046388001,  -0.0000958087,   0.0132419684, ...
           0.0050100159,  -0.0137114044,  -0.0076436215,   0.0052413328, ...
           0.0002853817,   0.0035621331,   0.0152978961,  -0.0033423920, ...
          -0.0263698275,  -0.0043365485,   0.0206130023,   0.0059378797, ...
          -0.0002801964,   0.0101552087,  -0.0169621258,  -0.0372398490, ...
           0.0150884777,   0.0501064779,   0.0001893760,  -0.0260942651, ...
          -0.0001325078,  -0.0299085837,  -0.0433060477,   0.0818564273, ...
           0.1278159870,  -0.0883935333,  -0.2137441855,   0.0380415029, ...
           0.2510227334 ]';
```

Figure N.78 shows the zero-phase amplitude responses in the pass band and the stop bands.

In addition, the Octave script *directFIRsymmetric_sdp_bandpass_test.m* designs a band-pass symmetric FIR filter with a specification similar to that of the example of *Pipeleers et al.* [63, Figure 4]:

```
M=15 % M+1 distinct coefficients, FIR filter order 2*M
fasl=0.05 % Amplitude lower stop band edge
```

Figure N.79: 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.* [76].

```
fapl=0.15 % Amplitude lower pass band edge
fapu=0.25 % Amplitude upper pass band edge
fasu=0.35 % Amplitude upper stop band edge
dBap=0.00347436 % Amplitude pass band peak-to-peak ripple(dB)
deltap=0.0002 % Amplitude pass band peak ripple(dB)
dBas=46.0206 % Amplitude stop band peak ripple(dB)
deltas=0.005 % Amplitude stop band peak ripple(dB)
Wasl=1 % Amplitude lower stop band weight
Watl=1 % Amplitude lower trans. band weight
Wap =4 % Amplitude pass band weight
Watu=1 % Amplitude upper trans. band weight
Wasu=1 % Amplitude upper stop band weight
```

The distinct filter coefficients are:

```
hM15 = [  -0.0051579522,  -0.0009795946,  -0.0039580318,  -0.0125006906, ...
           0.0044610682,   0.0145602006,  -0.0018350046,   0.0275110194, ...
           0.0515054222,  -0.0173704017,  -0.0260249902,   0.0194463757, ...
          -0.1412890461,  -0.2459630757,   0.1201954113,   0.4313855543 ]';
```

Figure N.79 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 band-pass symmetric FIR filter using the PCLS algorithm of *Selesnick et al.* [91] with the specification:

```
M=15 % FIR filter order is 2*M
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
```

Figure N.80: Zero-phase amplitude response of an $M = 15$ symmetric band pass FIR filter designed with the PCLS optimisation procedure of *Selesnick et al.* [91].

```
deltap=0.0002 % Amplitude pass band 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 ripple
Wasl=1000 % Amplitude lower stop band weight
Wasu=1000 % Amplitude upper stop band weight
```

The distinct coefficients of this filter are:

```
hM1 = [  -0.00498259,  -0.00077441,  -0.00359006,  -0.01197965, ...
          0.00505930,   0.01489769,  -0.00162383,   0.02741140, ...
          0.05084815,  -0.01823754,  -0.02676637,   0.01876929, ...
         -0.14168499,  -0.24558003,   0.12083560,   0.43200933 ]';
```

Figure N.80 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:

```
ftol=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=0.0002 % Pass band amplitude peak 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 amplitude response peak ripple
Watl=0 % Lower transition band squared amplitude weight
```

Figure N.81: 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.* [91].

```
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 peak-to-peak ripple
Wtp=0.1 % Pass band group delay response weight
```

The coefficients of this filter are:

```
h = [  -0.0028467207,   0.0006476462,  -0.0086666490,  -0.0117339352, ...
        0.0013358934,  -0.0049350057,  -0.0085071742,   0.0408410673, ...
        0.0447144697,  -0.0039663030,   0.0550589972,   0.0505163440, ...
       -0.2318274887,  -0.3108724588,   0.1330876947,   0.4024620875, ...
        0.0930131661,  -0.1578775283,  -0.0523194757,  -0.0062755881, ...
       -0.0716293192,  -0.0197389282,   0.0382082391,   0.0064384378, ...
        0.0044589137,   0.0250700759,   0.0044139708,  -0.0082307305, ...
        0.0015695252,  -0.0023735018,  -0.0050359143 ]';
```

Figure N.81 shows the amplitude and delay responses in the pass band and the stop band amplitude response.

N.9 Design of half-band FIR filters

N.9.1 Vaidyanathan's "TRICK" for the design of FIR half-band filters

Vaidyanathan and *Nguyen* [179] 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.82 shows the frequency response of the filter.

Figure N.82: FIR half-band filter with $fp = 0.24$ and $M = 80$ designed with Vaidyanathan's "TRICK" [179].

N.9.2 Design of equi-ripple FIR half-band filters

Zahradník, *Vlček* and *Unbehauen* [186] 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:

$$H(z) = \sum_{k=0}^{4n+2} h_k z^{-k} \tag{N.26}$$

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:

$$Q(v) = \frac{1}{2} + \sum_{k=0}^{n} c_{2k+1} T_{2k+1}(v) \tag{N.27}$$

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* [147, 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 [147, 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 κ_1 is the elliptic modulus [147, Equation 16]. The complex parameter u follows the path shown in [147, 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.* [186, 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.* [186, Equation 13] substitute x into the double angle identity for $U_{2n}(\cos\omega)$[q] 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* [183, 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* [183, Equations 11 to 14] provide numerical approximations for n, κ', 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 $1 \to v$) [186, 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 [183, 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* [183, 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.83 and Figure N.84. Figure N.85 shows the corresponding frequency response.

---

q

$$\begin{aligned}
U_{2n}(\cos\omega) &= \frac{\sin[(2n+1)\omega]\cos\omega + \sin\omega\cos[(2n+1)\omega]}{2\sin\omega\cos\omega} + \frac{\sin[(2n+1)\omega]\cos\omega - \sin\omega\cos[(2n+1)\omega]}{2\sin\omega\cos\omega} \\
&= U_n(\cos 2\omega) + U_{n-1}(\cos 2\omega) \\
&= U_n\left(2\cos^2\omega - 1\right) + U_{n-1}\left(2\cos^2\omega - 1\right)
\end{aligned}$$

Figure N.83: Equiripple FIR half-band filter generating function, $G(v)$, for $n = 20$, $\kappa' = 0.03922835$, $A = 1.08532371$, $B = 0.95360863$ and $N = 0.55091994$ [183, Figure 1].

Figure N.84: Equiripple FIR half-band filter zero-phase frequency response, $Q(v)$, for $n = 20$, $\kappa' = 0.03922835$, $A = 1.08532371$, $B = 0.95360863$ and $N = 0.55091994$ [183, Figure 2].

Figure N.85: 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:

$$\left(1-v^2\right)\frac{d^2U_n\left(v\right)}{dv^2} - 3v\frac{dU_n\left(v\right)}{dv} + n\left(n+2\right)U_n\left(v\right) = 0 \tag{N.28}$$

Substituting x [183, Equation 8]:

$$v\left(v^2-\kappa'^2\right)\left[\left(1-v^2\right)\frac{d^2U_n\left(v\right)}{dv^2} - 3v\frac{dU_n\left(v\right)}{dv}\right] + \left[\left(\kappa'^2+2v^2\right)\left(1-v^2\right)\right]\frac{dU_n\left(v\right)}{dv} + 4v^3n\left(n+2\right)U_n\left(v\right) = 0 \tag{N.29}$$

Equation N.29 is arranged so that its solution is simplified by using Equation N.28. Equation N.29 is solved by the series expansion:

$$U_n\left(\frac{2v^2-1-\kappa'^2}{1-\kappa'^2}\right) = \sum_{l=0}^{n}\alpha_{2l}U_{2l}\left(v\right) \tag{N.30}$$

Integrating this expansion results in an expression of the form shown in Equation N.27. *Zahradník* and *Vlček* [183, Table 1] show an algorithm for the evaluation of the coefficients, a_l, of $\mathcal{U}_n$, reproduced here, with a correction to α_{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*[183] 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.86 shows the amplitude response of the half-band FIR filter.

Algorithm N.4 *Zahradník* and *Vlček*'s algorithm [183, Table 1] for the evaluation of the coefficients, a_l, of $\mathcal{U}_n$.

Require: n, κ'

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

Figure N.86: Equiripple FIR half-band filter with $f_p = 0.240$ and $a_s = -140dB$ and $n = 118$ calculated by Algorithm N.4 [183, Table 1].

The distinct coefficients of the half-band filter are (see Equation N.26):

```
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, ...
                 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;
```

Addendum

Working for *Zahradník* and *Vlček* Equation 8 [183]

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.28 gives Equation N.29.

Working for *Zahradník* and *Vlček* Table 1 [183, 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.30 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 κ'^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:

$$U_{2l+1}(v) : \frac{1}{2}\left[3\left(n(n+2) - l(l+2)\right) + 2l + 1\right]\alpha_{2l} + l(2l+1)\kappa'^2\alpha_{2l}$$

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:

$$(l+1)\kappa'^2\alpha_{2l}+\frac{1}{2}[2(l+1)-l]\alpha_{2l}$$

Collecting terms in $U_{2l-1}(v)$ from Equation 25:

$$\frac{3}{2}n(n+2)\alpha_{2l}$$

Giving:

$$U_{2l-1}(v):\frac{1}{2}[3(n(n+2)-l(l+2))+2(2l+1)]\alpha_{2l}+(l+1)(2l+1)\kappa'^2\alpha_{2l}$$

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:

$$U_{2l-3}(v):\frac{1}{2}[n(n+2)-(l-1)(l+1)]\alpha_{2l}$$

Working for Equations 30 to 33 Changing the summation variables:

$$\begin{aligned}
&\sum_{l=3}^{n+3}\frac{1}{2}[n(n+2)-(l-3)(l-1)]\alpha_{2l-6}U_{2l-3}(v) \quad \ldots\\
&+\sum_{l=2}^{n+2}\left\{\frac{1}{2}[3(n(n+2)-(l-2)l)+2l-3]+(l-2)(2l-3)\kappa'^2\right\}\alpha_{2l-4}U_{2l-3}(v) \quad \ldots\\
&+\sum_{l=1}^{n+1}\left\{\frac{1}{2}[3(n(n+2)-(l-1)(l+1))+2(2l-1)]+l(2l-1)\kappa'^2\right\}\alpha_{2l-2}U_{2l-3}(v) \quad \ldots\\
&+\sum_{l=0}^{n}\frac{1}{2}[n(n+2)-(l-1)(l+1)]\alpha_{2l}U_{2l-3}(v)\\
&=0
\end{aligned}$$

Assuming that $U_l(v)=0$ for $l\leq 0$:

$$\begin{aligned}
&\sum_{l=3}^{n+3}[n(n+2)-(l-3)(l-1)]\alpha_{2l-6} \quad \ldots\\
&+\sum_{l=2}^{n+2}\left[3(n(n+2)-(l-2)l)+2l-3+2(l-2)(2l-3)\kappa'^2\right]\alpha_{2l-4} \quad \ldots\\
&+\sum_{l=2}^{n+1}\left[3(n(n+2)-(l-1)(l+1))+2(2l-1)+2l(2l-1)\kappa'^2\right]\alpha_{2l-2} \quad \ldots\\
&+\sum_{l=2}^{n}[n(n+2)-(l-1)(l+1)]\alpha_{2l}\\
&=0
\end{aligned} \tag{N.31}$$

The recurrence is initialised by calculating α_{2n} from the $n+3$ term, α_{2n-2} from the $n+2$ term and α_{2n-4} from the $n+1$ term in Equation N.31. The remaining terms $\alpha_{2n-6}\ldots\alpha_0$ are calculated by backwards recurrence for $l=n,n-1,\ldots,3$. Equation N.31 is implemented in the *body* of Table 1.

Working for Equations 34 and 35 The $n+3$ term in N.31 gives:

$$[n(n+2) - l(l+2)]\,\alpha_{2n} = 0$$

α_{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.31 gives:

$$\begin{aligned}
\frac{1}{2}\left[n(n+2) - (n-1)(n+1)\right]\alpha_{2n-2} + \left\{\frac{1}{2}\left[3\left(n(n+2) - n(n+2)\right) + 2n + 1\right] + n(2n+1)\,\kappa'^2\right\}\alpha_{2n} &= 0 \\
\left[n(n+2) - (n-1)(n+1)\right]\alpha_{2n-2} + \left[2n + 1 + 2n(2n+1)\,\kappa'^2\right]\alpha_{2n} &= 0 \\
(2n+1)\,\alpha_{2n-2} + \left[2n + 1 + 2n(2n+1)\,\kappa'^2\right]\alpha_{2n} &= 0 \\
\alpha_{2n-2} + \left[1 + 2n\kappa'^2\right]\alpha_{2n} &= 0
\end{aligned}$$

Working for Equation 37 The $n+1$ term in N.31 gives:

$$\begin{aligned}
\frac{1}{2}\left[n(n+2) - (n-2)\,n\right]\alpha_{2n-4} + \quad &\ldots \\
\left\{\frac{1}{2}\left[3\left(n(n+2) - (n-1)(n+1)\right) + 2n - 1\right] + (n-1)(2n-1)\,\kappa'^2\right\}\alpha_{2n-2} + \quad &\ldots \\
\left\{\frac{1}{2}\left[3\left(n(n+2) - n(n+2)\right) + 2(2n+1)\right] + (n+1)(2n+1)\,\kappa'^2\right\}\alpha_{2n} &= 0 \\
2n\alpha_{2n-4} + \left[4n + 1 + (n-1)(2n-1)\,\kappa'^2\right]\alpha_{2n-2} + (2n+1)\left[(n+1)\,\kappa'^2 + 1\right]\alpha_{2n} &= 0
\end{aligned}$$

N.10 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 [148, 259, 182]. *Zahradník* [182] 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.10.1 The Zolotarev polynomials

Chen and *Parks* [259, Section II], define "a Zolotarev polynomial of degree N, with L zeros in the interval (α, β) 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 (α, β), and the elliptic function modulus κ". Denoting such a polynomial by $f_{N,L}(u, \kappa)$, in parametric form [148, Equations 3 and 4]:

$$x = \frac{\operatorname{sn}^2(u, \kappa) + \operatorname{sn}^2\left(\frac{LK}{N}, \kappa\right)}{\operatorname{sn}^2(u, \kappa) - \operatorname{sn}^2\left(\frac{LK}{N}, \kappa\right)} \tag{N.32}$$

$$f_{N,L}(u, \kappa) = \frac{(-1)^L}{2}\left[\left\{\frac{H\left(u - \frac{LK}{N}, \kappa\right)}{H\left(u + \frac{LK}{N}, \kappa\right)}\right\}^N + \left\{\frac{H\left(u + \frac{LK}{N}, \kappa\right)}{H\left(u - \frac{LK}{N}, \kappa\right)}\right\}^N\right] \tag{N.33}$$

where κ 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 [259, 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* [148] 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 [148, Equation 6]:

$$w = x\operatorname{cn}^2(u_0, \kappa) - \operatorname{sn}^2(u_0, \kappa) \tag{N.34}$$

$$= \frac{\operatorname{sn}^2(u, \kappa)\operatorname{cn}^2(u_0, \kappa) + \operatorname{cn}^2(u, \kappa)\operatorname{sn}^2(u_0, \kappa)}{\operatorname{sn}^2(u, \kappa) - \operatorname{sn}^2(u_0, \kappa)} \tag{N.35}$$

where $u_0 = \frac{p}{p+q}K(\kappa)$. The inverse transformation is [259, Equation 2] [56, Section 22.6(iv)]:

$$\begin{aligned} u &= \operatorname{arcsc}\left(\imath\operatorname{sc}(u_0, \kappa)\sqrt{\frac{1+w}{1-w}}, \kappa\right) \\ &= \imath\operatorname{arcsn}\left(\operatorname{sc}(u_0, \kappa)\sqrt{\frac{1+w}{1-w}}, \sqrt{1-\kappa^2}\right) \end{aligned} \tag{N.36}$$

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.* [184, Equation 7] rewrite Equation N.33 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, $T_n(\cos\Phi) = \cos n\Phi$ [148, Equations 26 and 27]:

$$\begin{aligned} Z_{p,q}(u,\kappa) &= (-1)^p T_{p+q}(\cos\Phi(u,\kappa)) \\ \cos\Phi(u,\kappa) &= \frac{1}{2}\left[\frac{H(u+u_0,\kappa)}{H(u-u_0,\kappa)} + \frac{H(u-u_0,\kappa)}{H(u+u_0,\kappa)}\right] \end{aligned} \tag{N.37}$$

Whittaker and *Watson* [52, Page 480] define the *Jacobi Eta* function as:

$$H(u,\kappa) = -\imath q^{-\frac{1}{4}} e^{\frac{\imath\pi u}{2K}} \Theta(u+\imath K',\kappa) \tag{N.38}$$

where $\Theta(u,\kappa)$ is the *Jacobi Theta* function, an even function with period $2K+\imath 2K'$. *Chen* and *Parks* [259, 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 [259, Appendix]:

$$u_m = \sigma_m + \imath K'$$

where:

$$\begin{aligned} \kappa' &= \sqrt{1-\kappa^2} \\ K' &= F\left(\frac{\pi}{2},\kappa'\right) \\ \sigma_m &= \operatorname{arcsn}\left(\left[\frac{Z(u_0,\kappa)}{\kappa^2 \operatorname{sn}(u_0,\kappa)\left[\operatorname{cn}(u_0,\kappa)\operatorname{dn}(u_0,\kappa) + \operatorname{sn}(u_0,\kappa) Z(u_0,\kappa)\right]}\right]^{\frac{1}{2}}, \kappa\right) \end{aligned} \tag{N.39}$$

and $Z(u,\kappa)$ is the *Jacobi Zeta* function. The corresponding maximum value is:

$$f_m = (-1)^p \cosh\left[n \log \frac{\Theta(\sigma_m+u_0,\kappa)}{\Theta(\sigma_m-u_0,\kappa)}\right] \tag{N.40}$$

The Octave script *zolotarev_vlcek_unbehauen_test.m* calculates $Z_{5,9}(w, 0.78)$, shown as Figure 1 by *Vlček* and *Unbehauen* [148] and shown here as Figure N.87.

Figure N.87: Zolotarev polynomial $Z_{5,9}(u, 0.78)$.

Following *Levy* [200, Equation 28] or *Vlček* and *Unbehauen* [147, Equations 22 and 23], *Vlček* and *Unbehauen* assert that, by inspection, the Zolotarev polynomials, $Z_{p,q}(w,\kappa)$, satisfy the following differential equation [148, Equation 7]:

$$\left(1-w^2\right)\left(w-w_p\right)\left(w-w_s\right)\left(\frac{df}{dw}\right)^2 = n^2\left(1-f^2\right)\left(w-w_m\right)^2 \tag{N.41}$$

Quoting *Vlček* and *Zahradník* [158, 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* [200] and *Vlček* and *Unbehauen* [148] show the solution of this non-linear differential equation with elliptic functions. Alternatively, differentiating Equation N.41 gives the following linear second-order differential equation [148, Equation 63]:

$$g_2(w)\left[\left(1-w^2\right)\frac{d^2f}{dw^2} - w\frac{df}{dw}\right] - \left(1-w^2\right)g_1(w)\frac{df}{dw} + n^2 g_0(w) f = 0 \tag{N.42}$$

where:

$$\begin{aligned} g_2(w) &= \left(w-w_p\right)\left(w-w_s\right)\left(w-w_m\right) \\ g_1(w) &= \left(w-w_p\right)\left(w-w_s\right) - \left(w-w_m\right)\left(w-\frac{w_p+w_s}{2}\right) \\ g_0(w) &= \left(w-w_m\right)^3 \end{aligned} \tag{N.43}$$

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 [148, Table IV], reproduced here (with corrections [149]) 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 [148, Table V with corrections][r].

The Chebyshev polynomial expansion coefficients of the example shown by *Vlček* and *Unbehauen* [149, 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 ];
```

[r]The Maxima script *zolotarev_vlcek_unbehauen_table_v.max* and my working by hand did not reproduce Table V of *Vlček* and *Unbehauen* [148, 149]. 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 $Z_{p,q}(w) = \sum_{m=0}^{n} b_m w^m(w)$ [148, Table IV], [149].

Require: p, q and κ

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 $Z_{p,q}(w) = \sum_{m=0}^{n} a_m T_m(w)$ [148, Table V], [149].

Require: p, q and κ

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.36 Re-writing Equation N.35:

$$\begin{aligned} w &= \frac{\operatorname{sn}^2(u,\kappa)\operatorname{cn}^2(u_0,\kappa)+\operatorname{cn}^2(u,\kappa)\operatorname{sn}^2(u_0,\kappa)}{(1-\operatorname{sn}^2(u_0,\kappa))\operatorname{sn}^2(u,\kappa)-(1-\operatorname{sn}^2(u,\kappa))\operatorname{sn}^2(u_0,\kappa)} \\ &= \frac{\operatorname{sn}^2(u,\kappa)\operatorname{cn}^2(u_0,\kappa)+\operatorname{cn}^2(u,\kappa)\operatorname{sn}^2(u_0,\kappa)}{\operatorname{cn}^2(u_0,\kappa)\operatorname{sn}^2(u,\kappa)-\operatorname{cn}^2(u,\kappa)\operatorname{sn}^2(u_0,\kappa)} \\ &= \frac{\operatorname{sc}^2(u,\kappa)+\operatorname{sc}^2(u_0,\kappa)}{\operatorname{sc}^2(u,\kappa)-\operatorname{sc}^2(u_0,\kappa)} \end{aligned} \tag{N.44}$$

Rearranging:

$$\begin{aligned} -(1-w)\operatorname{sc}^2(u,\kappa) &= (1+w)\operatorname{sc}^2(u_0,\kappa) \\ \operatorname{sc}(u,\kappa) &= \pm\imath\operatorname{sc}(u_0,\kappa)\sqrt{\frac{1+w}{1-w}} \end{aligned}$$

Using the *Jacobi imaginary transformation* [56, 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 $\operatorname{sc}(K+\imath K',\kappa)=\imath\kappa'^{-1}$ [56, Tables 22.4.1, 22.4.2 and 22.5.1].

Substituting $\varphi_s = \frac{\omega_s T}{2}$, $\varphi_p = \frac{\pi-\omega_p T}{2}$ [148, Page 726] and $\sin\varphi_s = \operatorname{sn}(u_0,\kappa)$ [148, Equation 82] into Equation N.44 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 [148] 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:

$$\frac{d^2f}{dw^2} = \frac{dg}{dw}\sqrt{1-f^2} - g\frac{1}{2}\frac{1}{\sqrt{1-f^2}}2f\frac{df}{dw}$$

Substituting $\frac{df}{dw}$:

$$\frac{d^2f}{dw^2} = \frac{1}{g}\frac{dg}{dw}\frac{df}{dw} - g^2 f$$

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:

$$\left[\left(1-w^2\right)(w-w_m)(w-w_p)(w-w_s)\right]\frac{d^2f}{dw^2} \quad \ldots$$
$$-\left[\left(1-w^2\right)(w-w_p)(w-w_s)+w(w-w_m)(w-w_p)(w-w_s)-\left(1-w^2\right)(w-w_m)(w-w_q)\right]\frac{df}{dw} \quad \ldots$$
$$+n^2(w-w_m)^3 f=0$$

Working for Equation 73 [148] The Chebyshev polynomial of the first kind satisfies the differential equation:

$$\left(1-w^2\right)\frac{d^2T_n(w)}{dw^2}-w\frac{dT_n(w)}{dw}+n^2T_n(w)=0$$

Equation N.42 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, $T_m(w)$:

$$f(w)=\sum_{m=0}^{n}a_mT_m(w)$$

so that:

$$\left(1-w^2\right)\frac{d^2f}{dw^2}-w\frac{df}{dw}=-\sum_{m=0}^{n}m^2a_mT_m(w)$$

Working for Equation 74 [148] Equation 74 uses the following properties of the Chebyshev polynomials:

$$\frac{dT_m(w)}{dw}=mU_{m-1}(w)$$
$$\left(1-w^2\right)U_{m-1}(w)=wT_m(w)-T_{m+1}(w)$$
$$T_{m-1}(w)-wT_m(w)=wT_m(w)-T_{m+1}(w)$$

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 [148] Equation 75 results directly from substituting Equations 72, 73 and 74 into Equation 63 (shown above as Equation N.42):

$$-\sum_{m=0}^{n}m^2a_mg_2(w)T_m(w)-\sum_{m=0}^{n}ma_mg_1(w)\left[T_{m-1}(w)-wT_m(w)\right]+\sum_{m=0}^{n}a_mn^2g_0(w)T_m(w)=0$$

Working for Table V [148] Expanding $g_0(w)$, $g_1(w)$ and $g_2(w)$ in Equation 75 and setting $w_q=\frac{w_p+w_s}{2}$:

$$-\sum_{m=0}^{n}m^2\alpha_m\left[(w-w_p)(w-w_s)(w-w_m)\right]T_m(w) \quad \ldots$$
$$-\sum_{m=0}^{n}m\alpha_m\left[(w-w_p)(w-w_s)-(w-w_m)(w-w_q)\right]\left[T_{m-1}(w)-wT_m(w)\right] \quad \ldots$$
$$+\sum_{m=0}^{n}\alpha_m\left[n^2(w-w_m)^3\right]T_m(w)=0$$

$$-\sum_{m=0}^{n}m^2\alpha_m\left[w^3-w_pw^2-w_sw^2-w_mw^2+w_mw_pw+w_mw_sw+w_pw_sw-w_pw_mw_s\right]T_m(w) \quad \ldots$$
$$-\sum_{m=0}^{n}m\alpha_m\left[w^2-w_pw-w_sw+w_pw_s-w^2+w_mw+w_qw-w_qw_m\right]\left[T_{m-1}(w)-wT_m(w)\right] \quad \ldots$$

$$+\sum_{m=0}^{n} n^2\alpha_m\left[w^3-3w_m w^2+3w_m^2 w-w_m^3\right]T_m(w)=0$$

After much algebra (shown in the Octave script *zolotarev_vlcek_unbehauen_table_v_test.m*) I arrived at:

$$\begin{aligned}
&\sum_{m=3}^{n+3}\left[n^2-(m-3)^2\right]\alpha_{m-3}T_m(w) \quad \ldots \\
&+\sum_{m=2}^{n+2} 2\left[(m-2)(m-3)w_p+(m-2)(m-3)w_s \quad \ldots\right. \\
&\qquad \left.+\left((m-2)(m-1)-3n^2\right)w_m+(m-2)w_q\right]\alpha_{m-2}T_m(w) \quad \ldots \\
&+\sum_{m=1}^{n+1} 3\left[n^2-(m-1)^2\right]+ \quad \ldots \\
&\qquad 4\left[3n^2w_m^2-(m-1)^2w_mw_p-(m-1)^2w_mw_s-(m-1)(m-2)w_pw_s-(m-1)w_mw_q\right]\alpha_{m-1}T_m(w) \quad \ldots \\
&+\sum_{m=0}^{n} 4\left[m^2w_p+m^2w_s+\left(m^2-3n^2\right)w_m\right]+8\left[-n^2w_m^3+m^2w_mw_pw_s\right]\alpha_mT_m(w) \quad \ldots \\
&+\sum_{m=-1}^{n-1} 3\left[n^2-(m+1)^2\right]+ \quad \ldots \\
&\qquad 4\left[3n^2w_m^2-(m+1)^2w_mw_p-(m+1)^2w_mw_s-(m+1)(m+2)w_pw_s+(m+1)w_mw_q\right]\alpha_{m+1}T_m(w) \quad \ldots \\
&+\sum_{m=-2}^{n-2} 2\left[(m+2)(m+3)w_p+(m+2)(m+3)w_s+\left((m+2)(m+1)-3n^2\right)w_m-(m+2)w_q\right]\alpha_{m+2}T_m(w) \quad \ldots \\
&+\sum_{m=-3}^{n-3}\left[n^2-(m+3)^2\right]\alpha_{m+3}T_m(w)=0
\end{aligned}$$

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 α_{m-3}.

N.10.2 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 $a_{s_{dB}} = 20 \log_{10} a_s < 0$:

$$\frac{2}{1 + f_m} \leq 10^{a_{s_{dB}}/20} \tag{N.45}$$

Rearranging:

$$f_m \geq y_m = 2 \times 10^{-a_{s_{dB}}/20} - 1 \tag{N.46}$$

Recalling Equation N.40, the scaled maximum value is:

$$\cosh \left[n \log \frac{\Theta(\sigma_m + u_0, \kappa)}{\Theta(\sigma_m - u_0, \kappa)} \right] \geq y_m \tag{N.47}$$

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 $\lambda = e^x$:

$$\frac{1}{2} \left[\lambda + \lambda^{-1} \right] \geq y_m \tag{N.48}$$

$$\lambda^2 - 2 y_m \lambda + 1 \geq 0 \tag{N.49}$$

$$\tag{N.50}$$

The stop-band degree equation becomes [148, Equation 85]:

$$n \geq \frac{\log \left(y_m + \sqrt{y_m^2 - 1} \right)}{\log \frac{\Theta(\sigma_m + u_0, \kappa)}{\Theta(\sigma_m - u_0, \kappa)}} \tag{N.51}$$

Zahradník et al. [184] 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, κ, by setting [184, 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 [184, Equation 14](with corrections[s]):

$$n \geq \frac{\operatorname{arccosh} \left(2 \times 10^{\left(a_{p_{dB}} - a_{s_{dB}}\right)/20} - 1 \right)}{\log \frac{\Theta(\sigma_{p1} + u_0, \kappa)}{\Theta(\sigma_{p1} - u_0, \kappa)}}$$

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 *Vlček* and *Unbehauen*

Vlček and *Unbehauen* [148, 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, κ, for $\varphi_p = \frac{\pi - \omega_p T}{2}$ and $\varphi_s = \frac{\omega_s T}{2}$, $\kappa' = \cot \varphi_p \cot \varphi_s$

[s] *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.51, 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 [148, 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* [148]. 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.88 shows the normalised Zolotarev polynomial of the FIR filter. Figure N.89 shows the frequency response of the FIR filter. Figure N.90 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.* [184]. Figure N.91 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.36. Both the arcsc and 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.92 shows a filter designed with the stop-band bandwidth and attenuation of *Zahradník et al.*'s Example 2. Figure N.93 shows the main lobe of the response.

Figure N.88: Normalised Zolotarev polynomial approximating that of Figure 4 in *Vlček* and *Unbehauen* [148].

Figure N.89: FIR filter response approximating that of Figure 4 in *Vlček* and *Unbehauen* [148].

Normalised Zolotarev polynomial $Z_{6,17}(w, 0.77029)$ (Vlcek and Unbehauen)

Figure N.90: Zeros of an FIR filter approximating that of Figure 4 in *Vlček* and *Unbehauen* [148].

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.91: Normalised Zolotarev polynomial $Z_{100,37}(w, 0.4)$ calculated with Equation 7 of *Zahradník et al.* [157].

Figure N.92: Normalized Zolotarev polynomial FIR filter approximating Example 2 of *Zahradník et al.* [157] with stop-band constraints.

Figure N.93: Main lobe of the response of a normalized Zolotarev polynomial FIR filter approximating Example 2 of *Zahradník et al.* [157] 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 [240, 47, 122, 123, 262]. 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* [122]. *Zahradník, Šusta et al.* [185] 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.37:

$$\begin{aligned} Z_{p,q}(u,\kappa) &= (-1)^p T_{p+q}(\cos\Phi(u,\kappa)) \\ \cos\Phi(u,\kappa) &= \frac{1}{2}\left[\frac{H(u+u_0,\kappa)}{H(u-u_0,\kappa)} + \frac{H(u-u_0,\kappa)}{H(u+u_0,\kappa)}\right] \end{aligned}$$

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.38:

$$H(u-u_0,\kappa) = \begin{cases} -H^*(u+u_0,\kappa) & u \in [0, \imath K'] \\ H^*(u+u_0,\kappa) & u \in [K + \imath K', K] \end{cases} \tag{N.52}$$

where H^* is the complex conjugate transpose of H. The $\Phi(u,\kappa)$ function of Equation N.37 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.94 shows the locations of the double zeros of $Q_{p,q}(w, 0.75)$ for pairs of odd and even p and q [185, Figure 2]. Figure N.95 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. [185, 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 $\rho(l)$, $l = 1, \ldots, m$ [185, 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 [185, 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.96 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.97 shows the double-zero locations near ± 1 and the central lobe of $Q_{2159,540}(w, 0.16238959)$ for *Zahradník, Šusta et al.*'s Example 2 [185]. Figure N.98 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.94: Double-zero locations of $Q_{p,q}(w, 0.75)$ [185, Figure 2].

Figure N.95: 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 [185, 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.96: 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 $Q_{22,49}(w, 0.46850107)$ [185, Figure 4].

Figure N.97: Detailed view of the double-zero locations of $Q_{2159,540}(w, 0.16238959)$ near ± 1 and the central lobe for *Zahradník, Šusta et al.*'s Example 2 [185].

Figure N.98: Amplitude responses of the 3 sub-filters cascade of the sub-filters of a normalised Zolotarev band-pass filter with zero-phase response $Q_{2159,540}(w, 0.16238959)$ [185, Figure 5].

Addendum

Working for Equation N.52 Substituting $u = -u_0 + \imath y K'$, where $y \in [0,1]$, into Equation N.38:

$$\begin{aligned} H^*\left(-u_0 + \imath y K', \kappa\right) &= \imath q^{-\frac{1}{4}} e^{\frac{\imath\pi\left(u_0 + \imath y K'\right)}{2K}} \Theta\left(-u_0 - \imath y K' - \imath K', \kappa\right) \\ &= -H\left(u_0 + \imath y K', \kappa\right) \end{aligned}$$

Substituting $u = K - u_0 + \imath y K'$, where $y \in [0,1]$, into Equation N.38:

$$\begin{aligned} H^*\left(K - u_0 + \imath y K', \kappa\right) &= \imath q^{-\frac{1}{4}} e^{\frac{\imath\pi\left(-K + u_0 + \imath y K'\right)}{2K}} \Theta\left(K - u_0 - \imath y K' - \imath K', \kappa\right) \\ &= -\imath q^{-\frac{1}{4}} e^{\frac{\imath\pi\left(K + u_0 + \imath y K'\right)}{2K}} \Theta\left(-K - u_0 - \imath y K' - \imath K', \kappa\right) \\ &= H\left(K + u_0 + \imath y K', \kappa\right) \end{aligned}$$

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.10.3 Almost equi-ripple low-pass FIR filter design with the Zolotarev polynomials

Vlček and *Zahradník* [157, 158] describe the use of modified Zolotarev polynomials to design "almost equi-ripple" low-pass FIR filters. They assume a solution of the form $\sqrt{1-w^2}S_{p,q}(w,\kappa)$[t] and use $S_{p,q}(w,\kappa)$ as the generating polynomial of a low-pass FIR filter[u]:

$$S_{p,q}(w,\kappa) = \sum_{m=0}^{n} a_m^+ U_m(w)$$
$$\mathcal{S}(w) = \int S_{p,q}(w,\kappa)\,dw$$
$$Q(w) = \frac{\mathcal{S}(w) - \mathcal{N}_1}{\mathcal{N}_2 - \mathcal{N}_1}$$

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* [157, 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 $U_m(w)$.

[t] See *Riblet* [82], "If one uses $x = -\cos\theta$ for a frequency variable instead of ω, ... 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 ± 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 $2P_n(x) = \left(1+\sqrt{1-x_c^2}\right)T_n\left(\frac{x}{x_c}\right) - \left(1-\sqrt{1-x_c^2}\right)T_{n-2}\left(\frac{x}{x_c}\right)$"

[u] The normalisation is shown in *Vlček* and *Zahradník* [157, 158] 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 [157, Tables 4 and 5].

Require: p, q and κ

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* [157, 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[v]:

$$\frac{(\xi_1 n + \xi_2)\,\Delta\omega T}{\pi} + \xi_3 + \frac{\xi_4}{(n+\xi_5)^{\xi_6}} = a_{s_{dB}}$$

   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 [157], illustrating the design of of an almost equi-ripple FIR filter with $p = 4$, $q = 11$ and $\kappa = 0.75$. Figure N.99 shows the iso-extremal function, $\sqrt{1-w^2}S_{4,11}(w, 0.75)$, Figure N.100 shows the generating function, Figure N.101 shows zero-phase transfer function and Figure N.102 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 [157, 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* reproduces *Vlček* and *Zahradník*'s Figure 5, shown here as Figure N.103.

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 $fs = 0.175$ [157, 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 χ. The frequency response of the filter is shown in Figure N.104.

[v] 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* [157, Equation 23] grossly over-estimates the required n as $n = 336$ when $n = 198$ is sufficient.

Figure N.99: Iso-extremal function $\sqrt{1-w^2}S_{4,11}\left(w,0.75\right)$, corresponding to Figure 1 of *Vlček* and *Zahradník* [157], $p=4$, $q=11$, $\kappa=0.75$.

Figure N.100: FIR filter generating function corresponding to Figure 2 of *Vlček* and *Zahradník* [157], $p=4$, $q=11$, $\kappa=0.75$.

Figure N.101: FIR filter zero-phase response corresponding to Figure 3 of *Vlček* and *Zahradník* [157], $p = 4$, $q = 11$, $\kappa = 0.75$.

Figure N.102: FIR filter amplitude response corresponding to Figure 4 of *Vlček* and *Zahradník* [157], $p = 4$, $q = 11$, $\kappa = 0.75$.

Figure N.103: FIR filter amplitude response corresponding to Figure 5 of *Vlček* and *Zahradník* [157], $p = 100$, $q = 300$, $\kappa = 0.25$.

Figure N.104: 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* [157, Section 8].

Figure N.105: 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* [148], the generating function, $S_{4,11}(w,0.75)$, of *Vlček* and *Zahradník* [157] 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* [157]. 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.105, created by the Octave script *zolotarev_vlcek_zahradnik_test.m*, compares the Zolotarev polynomial, $Z_{4,11}(w,0.75)$, found by direct calculation [259], 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* [157, 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.42[w]. However, after examining Figure N.99 [157, Figure 1], it is clear that $\sqrt{1-w^2}S_{p,q}(w,\kappa)$ is not a solution of Equation N.41, as it is zero rather than ±1 at $w=\pm1$, as shown for $Z_{5,9}(w,0.78)$ in Figure N.87. *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.42 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.42 with this modification. The resulting differential equation is[x]:

$$g_2(w)\left[\left(1-w^2\right)\frac{d^2f}{dw^2}-3w\frac{df}{dw}\right]-g_1(w)\left(1-w^2\right)\frac{df}{dw}+\left[(n+1)^2g_0(w)+wg_1(w)-g_2(w)\right]f=0 \tag{N.53}$$

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:

$$\left(1-w^2\right)\frac{dU_m(w)}{dw}=(m+2)wU_m(w)-(m+1)U_{m+1}(w)$$

[w] For example, see *Morse* and *Feshbach* [173, Section 5.2].

[x] Consistent with Equation N.43 rather than following the changes in notation from *Vlček* and *Unbehauen* [148, Equation 64] to *Vlček* and *Unbehauen* [157, 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:

$$\begin{aligned}
&- g_2(w) \sum_{m=0}^{n} m(m+2)\, a_m^+ U_m(x) \quad \ldots \\
&- g_1(w) \sum_{m=0}^{n} a_m^+ \left[(m+2)\, wU_m(w) - (m+1)\, U_{m+1}(w)\right] \quad \ldots \\
&+ \left[(n+1)^2 g_0(w) + wg_1(w) - g_2(w)\right] \sum_{m=0}^{n} a_m^+ U_m(w) = 0
\end{aligned} \tag{N.54}$$

The Maxima script *zolotarev_vlcek_zahradnik_table_4.max* calculates the recurrence relations from Equation N.54. The results agree with Table 4 of *Vlček* and *Zahradník* [157] when inserted into the Octave function *zolotarev_vlcek_zahradnik.m*.

N.11 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* [240], *Shiung et al.* [47], *Smith* [122] or *Lim* and *Liu* [262]. The cascade connection permits an improved response and/or reduced complexity by partition of the FIR filter or the use of fewer multipliers.

Saramäki [240] 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* [240, 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.11.1 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:

$$H_0(z) = \sum_{n=0}^{N} a_n \left[\bar{H}_M(z)\right]^n \tag{N.55}$$

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 [240, Figure 1] shows several implementations of Equation N.55. The first is [240, Figure 1a and Equation 6]:

$$\begin{aligned}H(z) &= z^{-NM} H_0(z) \\ &= \sum_{n=0}^{N} a_n z^{-(N-n)M}\left[H_M(z)\right]^n\end{aligned} \tag{N.56}$$

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 [240, 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 [240, Figure 2].

N.11.2 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:

$$\begin{aligned} 1 - \delta_p \leq H_0\left(e^{\imath\omega}\right) \leq 1 + \delta_p \quad & \omega \in I_p \\ -\delta_s \leq H_0\left(e^{\imath\omega}\right) \leq \delta_s \quad & \omega \in I_s \end{aligned} \tag{N.57}$$

The specifications for the prototype filter, $\tilde{H}_0\left(e^{\imath\Omega}\right)$, are:

$$\begin{aligned} 1 - \delta_p \leq \tilde{H}_0\left(e^{\imath\Omega}\right) \leq 1 + \delta_p \quad & 0 \leq \Omega \leq \Omega_p \\ -\delta_s \leq \tilde{H}_0\left(e^{\imath\Omega}\right) \leq \delta_s \quad & \Omega_s \leq \Omega \leq \pi \end{aligned} \tag{N.58}$$

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:

$$\begin{aligned} \cos\Omega_p \leq P_M(\omega) \leq 1 \quad & \text{for } \omega \in I_p \\ -1 \leq P_M(\omega) \leq \cos\Omega_s \quad & \text{for } \omega \in I_s \end{aligned} \tag{N.59}$$

For fixed values of Ω_p, and Ω_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:

$$\hat{P}_M(\omega) = AP_M(\omega) + B \tag{N.60}$$

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:

$$x = A\cos\Omega + B \tag{N.61}$$

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.61 maps the pass-band, $[0, \Omega_p]$, and the stop-band, $[\Omega_s, \pi]$, of the Ω-plane onto the x-plane regions $[x_{p1}, x_{p2}]$ and $[x_{s1}, x_{s2}]$, respectively, where:

$$\begin{aligned} x_{p1} &= \quad A\cos\Omega_p + B \\ x_{p2} &= \quad A + B \\ x_{s1} &= -A + B \\ x_{s2} &= \quad A\cos\Omega_s + B \end{aligned} \tag{N.62}$$

If $\tilde{H}_0\left(e^{\imath\Omega}\right)$ satisfies Equation N.58, 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 $[x_{s1}, x_{s2}]$ [240, Figure 3]. Similarly, if P_M satisfies Equation N.59, 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 [240, Figure 4]:

$$\begin{aligned} 1 - \delta_p \leq \hat{H}_0(x) \leq 1 + \delta_p \quad & \text{for } x \in [x_{p1}, x_{p2}] \\ \delta_s \leq \hat{H}_0(x) \leq \delta_s \quad & \text{for } x \in [x_{s1}, x_{s2}] \\ x_{p1} \leq \hat{P}_M(\omega) \leq x_{p2} \quad & \text{for } \omega \in I_p \\ x_{s1} \leq \hat{P}_M(\omega) \leq x_{s2} \quad & \text{for } \omega \in I_s \end{aligned} \tag{N.63}$$

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.61 to map the problem to the Ω-plane, designing the filter to meet the specifications of Equation N.58, 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:

$$\begin{aligned} 1-\delta_p \leq \hat{H}_0(x) \leq 1+\delta_p \quad & \text{for } x \in \left[1-\hat{\delta}_p, 1+\hat{\delta}_p\right] \\ \delta_s \leq \hat{H}_0(x) \leq \delta_s \quad & \text{for } x \in \left[-\hat{\delta}_s, \hat{\delta}_s\right] \\ 1-\hat{\delta}_p \leq \hat{P}_M(\omega) \leq 1+\hat{\delta}_p \quad & \text{for } \omega \in I_p \\ -\hat{\delta}_s \leq \hat{P}_M(\omega) \leq \hat{\delta}_s \quad & \text{for } \omega \in I_s \end{aligned} \tag{N.64}$$

The NPFB and NSFB specifications are related by the transformation of Equation N.61 and the subsitution of Equation N.62 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.11.3 Filter design

Saramäki [240, Section IV] considers four separate design problems.

Approximation Problem I

Given the composite filter specifications of Equation N.57 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.57. This can be done by choosing the prototype filter pass-band edge, Ω_p as the unknown and, for each Ω_p finding the prototype filter with the minimum Ω_s that satisfies Equation N.58.
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 Ω_s maximises the allowable variation of $P_M(\omega)$ on I_s and, consequently, the minimum required sub-filter order to meet Equation N.58. The solution is obtained by reducing Ω_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.5.2, finds the filter that minimises Ω_s for the given Ω_p, δ_p and δ_s.

Approximation Problem II

Given the composite filter specifications of Equation N.57 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.59 and N.58, 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.59. 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.64, 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, δ_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.63 with the minimum value of N:
   (a) Determine A and B and then Ω_p and Ω_s from Equation N.62.
   (b) Design the prototype filter of minimum even-order $2N$ to meet the specification of Equation N.58.
   (c) Transform the resulting frequency response to the x-plane with Equation N.61.

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.63.

N.11.4 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, Ω_p, search for the minimum prototype filter transition width, $\Delta\Omega$, that satisfies the amplitude specifications, δ_p and δ_s. Next, search for the minimum sub-filter order, $2M$, that, given Ω_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 [240, Figure 5][y]. 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.5.2. 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.106 compares the minimum sub-filter order, $2M$, and the minimum prototype filter transition width, $\Delta\Omega$, found with *selesnickFIRsymmetric_lowpass*. Figure N.107 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.108 shows the pass-band and stop-band of the filter.

N	6
f_p	0.1375
f_s	0.15
δ_p	0.01
δ_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.56, 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 ]';
```

[y] The plot shown in *Saramäki*'s Figure 6 has the same filter parameters except for $N = 7$.

Figure N.106: Comparison of the minimum sub-filter order, $2M$, and the minimum prototype filter transition width, $\Delta\Omega$, found with *selesnickFIRsymmetric_lowpass* [240, Figure 5].

Figure N.107: Approximation problem I solution mapping the prototype filter to the sub-filter and the overall response.

Figure N.108: 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 [240, 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
δ_p	0.01
δ_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, ω_{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 δ_p and δ_s amplitude response ripple specifications. The script does this by calling the Octave function *selesnickFIRsymmetric_lowpass* with fixed prototype filter pass-band edge frequency, Ω_p at $1-\delta_p$, and increasing N until the resulting prototype filter stop-band edge frequency, Ω_{sN}, satisfies the value of Ω_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.109 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.110 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.56, 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.109: Approximation problem II solution mapping the prototype filter to the sub-filter and the overall response.

Figure N.110: 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

O.1 The continuous-time KYP lemma

The *Kalman-Yakubovič-Popov* (KYP) lemma [197] is a fundamental result from the theory of control systems. The continuous time KYP states that, for a state-variable system with $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$ and (A, B) controllable, the frequency domain inequality(FDI):

$$\begin{bmatrix} (\imath\omega I - A)^{-1} B \\ I \end{bmatrix}^* \Theta \begin{bmatrix} (\imath\omega I - A)^{-1} B \\ I \end{bmatrix} < 0 \quad \omega \in \mathbb{R} \cup \{\infty\} \tag{O.1}$$

(where M^* 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:

$$\begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^* \begin{bmatrix} 0 & P \\ P & 0 \end{bmatrix} \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \Theta \prec 0 \tag{O.2}$$

or:

$$\begin{bmatrix} A^*P + PA & PB \\ B^*P & 0 \end{bmatrix} + \Theta \prec 0 \tag{O.3}$$

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 $H(\omega) = C(\imath\omega I - A)^{-1} B + D$ is the transfer function of the filter, then we can write:

$$\Theta = \begin{bmatrix} C & D \\ 0 & I \end{bmatrix}^* \Pi \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} \tag{O.4}$$

For example, if

$$\Pi = \begin{bmatrix} I & 0 \\ 0 & -\varepsilon^2 I \end{bmatrix} \tag{O.5}$$

then the corresponding frequency domain constraint is $|H(\omega)|^2 \leq \varepsilon^2$ for all ω. If the state-space description, $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$, is known, then a feasible solution of Equation O.3 with this Π also shows ε. Similarly, if

$$\Pi = \begin{bmatrix} 0 & -I \\ -I & 2\varepsilon I \end{bmatrix} \tag{O.6}$$

and a feasible solution for ε exists, then $\frac{1}{2}\left[H(\omega) + H(\omega)^*\right] \leq \varepsilon$ for all ω[a].

Iwasaki, *Meinsma* and *Fu* [245] extend the continuous time KYP lemma to restricted frequency regions. In addition, the KYP lemma has been generalised to discrete time state variable systems over restricted frequency regions [61, 234, 236, 237, 245, 63]. In the remainder of this section I follow the discussion of a finite-frequency continuous-time KYP lemma described by *Iwasaki*, *Meinsma* and *Minyue Fu* [245].

[a]For example, see the Octave script *yalmip_kyp_epsilon_test.m* .

O.1.1 A generalised S-procedure

The *S-procedure*[b] is the basis of proofs of the KYP lemma. *Iwasaki*, *Meinsma* and *Minyue Fu* [245, Section 1] first define a set, $\mathcal{G}$, of vectors, ζ, and matrixes, S_i, such that:

$$\mathcal{G} := \{\zeta \in \mathbb{C}^n : \zeta \neq 0, \zeta^* S_k \zeta \leq 0, \forall k = 1, \ldots, m\}$$

Then, assuming $\Theta, S_k \in \mathbb{H}^n$, according to *Iwasaki et al.*, the "classical" version of the S-procedure considers the following statements:

$$\zeta^\dagger \Theta \zeta < 0, \ \forall \zeta \in \mathcal{G} \tag{O.7}$$

$$\exists \tau_k \in \mathbb{R} \text{ such that } \tau_k > 0 \text{ and } \Theta \prec \sum_{k=1}^{m} \tau_k S_k \tag{O.8}$$

The S-procedure replaces the multiple constraints of Equation O.7 with the single linear constraint of Equation O.8. The latter condition is easier to satisfy as it is a search for the scalar multipliers, τ_k, satisfying convex constraints. *Iwasaki et al.* comment that "In general, the S-procedure on $\mathbb{C}^n$ is conservative, i.e. Equation O.8 is only sufficient for Equation O.7 and may not be necessary"[c].

Iwasaki et al. [245, Equation 4] generalise the S-procedure by re-defining $\mathcal{G}$:

$$\mathcal{G} := \{\zeta \in \mathbb{C}^n : \zeta \neq 0, \zeta^* S_k \zeta \leq 0, \forall S \in \mathcal{S}\}$$

where:

$$\mathcal{S} := \left\{ \sum_{k=1}^{m} \tau_k S_k : \tau_k > 0, \forall k = 1, \ldots, m \right\}$$

In the following, *Iwasaki et al.* show the conditions on $\mathcal{S}$ for which the S-procedure is "exact" or "non-conservative" (i.e.: Equation O.8 is necessary and sufficient for Equation O.7).

First, *Iwasaki et al.* define *loss-less sets* [245, Section 2]:

Definition 1: $\mathcal{S} \in \mathbb{H}^{n \times n}$ is said to be *loss-less* if:

1. $\mathcal{S}$ is convex
2. $S \in \mathcal{S} \Rightarrow \tau S \in \mathcal{S}, \forall \tau > 0$
3. For each non-zero matrix $H \in \mathbb{C}^{n \times n}$ such that

$$H = H^* \succeq 0, \operatorname{trace}(SH) \leq 0, \forall S \in \mathcal{S}$$

there exist vectors $\zeta_k \in \mathbb{C}^n, k = 1, \ldots, r$ such that

$$H = \sum_{k=1}^{r} \zeta_k \zeta_k^* \text{ and } \zeta_k^* S \zeta_k \leq 0, \forall S \in \mathcal{S}$$

where r is the *rank* of H.

and give an alternative version of the separating hyperplane theorem[d]:

Lemma 1 : Let $\mathcal{X}$ be a convex subset of $\mathbb{C}^m$, and $F : \mathcal{X} \to \mathbb{C}^{n \times n}$ be a Hermitian-valued affine function, then the following statements are equivalent:

1. The set $\{x : x \in \mathcal{X}, F(x) < 0\}$ is empty.
2. $\exists$ non-zero $H = H^* \succeq 0$ such that $\operatorname{trace}(F(x) H) \geq 0, \forall x \in \mathcal{X}$.

[b] Also see Section B.7.3

[c] In this context, the logical statement $S \to N$ means that N is *necessary* for S and that S is *sufficient* for N.

[d] See Section B.7.2.

Iwasaki et al. [245, Theorem 1] now prove that their generalised S-procedure is exact if $\mathcal{S}$ is loss-less:

Theorem 1 (Generalised S-procedure) : Let a Hermitian matrix, Θ, and a subset, $\mathcal{S}$, of the Hermitian matrixes be given. Suppose $\mathcal{S}$ is loss-less, then the following statements are equivalent:

1. $\zeta^*\Theta\zeta < 0, \forall\zeta \in \mathcal{G} := \{\zeta \in \mathbb{C}^n : \zeta \neq 0, \zeta^* S_k \zeta \leq 0, \forall S \in \mathcal{S}\}$.
2. There exists $S \in \mathcal{S}$ such that $\Theta \prec S$.

Proof: The first statement follows from the second. The converse is proved by contradiction. Assume that there is no $S \in \mathcal{S}$ such that $\Theta \prec S$. Then, from Lemma 1, there is a non-zero matrix, H, such that:

$$H = H^* \succeq 0, \quad \text{trace}\left(\left(\Theta - S\right) H\right) \geq 0, \ \forall S \in \mathcal{S}$$

Since $\mathcal{S}$ is lossless, from the second property of Definition 1:

$$\text{trace}\left(SH\right) \leq 0, \forall S \in \mathcal{S}, \text{trace}\left(\Theta H\right) \geq 0$$

The first condition implies the existence of vectors ζ_k so that:

$$\text{trace}\left(\Theta H\right) = \sum_{k=1}^{r} \zeta_k^* \Theta \zeta_k \geq 0$$

Hence there exists ζ_k so that $\zeta_k^*\Theta\zeta \geq 0$. Noting that $\zeta_k \in \mathcal{G}$, the first statement does not hold.

O.1.2 A finite-frequency continuous-time KYP lemma

Iwasaki et al. [245, Section 3] now define the set $\mathcal{G}$ as:

$$\mathcal{G} := \left\{ \begin{bmatrix} f \\ g \end{bmatrix} \in \mathbb{C}^{2n} : f = \imath\omega g, \omega \in \mathbb{R}, |\omega| \leq \omega_0 \right\}$$

where ω_0 is a given real scalar. *Iwasaki et al.* [245, Lemma 2] prove the following lemma[e]:

Lemma 2 : Let a real scalar ω_0 and complex vectors f and g be given, then the following statements are equivalent:

1. There exists a real scalar ω such that $f = \imath\omega g, |w| \leq \omega_0$.
2. $\begin{bmatrix} f \\ g \end{bmatrix}^* \begin{bmatrix} Q & P \\ P & -\omega_0^2 Q \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix} \leq 0, \forall P, Q \in \mathbb{H}$ and $Q \succ 0$

Proof: Suppose the first statement holds. Then:

$$\begin{bmatrix} f \\ g \end{bmatrix}^* \begin{bmatrix} Q & P \\ P & -\omega^2 Q \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix} = \left(\omega^2 - \omega_0^2\right)\left(g^* Q g\right) \leq 0$$

so that the second statement holds. Conversely, if the second statement holds, then:

$$\text{trace}\left(ff^* - \omega_0^2 gg^*\right) Q + \text{trace}\left(gf^* + fg^*\right) P \leq 0$$

holds for all $P = P^*$ and $Q = Q^* \succ 0$. This implies

$$\begin{aligned} ff^* - \omega_0^2 gg^* &\preceq 0 \\ gf^* + fg^* &\leq 0 \end{aligned}$$

Iwasaki et al. claim that the first statement now follows from Lemma III.4 of *Meinsma et al.* [60].

[e] From *Meinsma et al.* [60]:
Lemma III.4 (Three Little Lemmas) : Let f, g be two column vectors of the same dimension.

1. $(f-g)(f-g)^*$ is Hermitian and $\succeq 0$ if-and-only-if $g = \delta f$ for some $\delta \in [-1, 1]$.
2. The Hermitian part of $(f-g)(f-g)^*$ is $\succeq 0$ if-and-only-if $g = \delta f$ for some $\delta \in \mathbb{C}$ with $|\delta| \leq 1$.
3. $\Re\left[\text{trace}\left(f-g\right)\left(f-g\right)^*\right] = \|f\|_2 - \|g\|_2$. Hence $\Re\left[\text{trace}\left(f-g\right)\left(f-g\right)^*\right] \geq 0$ if-and-only-if $g = \Delta f$ for some matrix Δ with $\|\Delta\| \leq 1$.

Next, *Iwasaki et al.* [245, Lemma 3,Section 5] prove the following lemma:

Lemma 3 : Let a scalar $\omega_0 > 0$ and a matrix $F \in \mathbb{C}^{2n\times k}$ be given. Define a subset of Hermitian matrixes by:

$$\mathcal{S} := \left\{ F^* \begin{bmatrix} Q & P \\ P & -\omega_0^2 Q \end{bmatrix} F : P = P^*, Q = Q^* \succ 0 \right\}$$

Then the set $\mathcal{S}$ is loss-less.

Iwasaki et al. [245, Theorem 2] now state a KYP lemma generalised to a finite frequency domain:

Theorem 2 : Let a scalar $\omega_0 > 0$ and matrixes $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{n\times m}$ and a Hermitian matrix $\Theta \in \mathbb{C}^{(n+m)\times(n+m)}$ be given. Assume that A has no eigenvalues on the imaginary axis, then the following statements are equivalent:

1. This finite frequency condition holds:

$$\begin{bmatrix} (\imath\omega I - A)^{-1} B \\ I \end{bmatrix}^* \Theta \begin{bmatrix} (\imath\omega I - A)^{-1} B \\ I \end{bmatrix} < 0, \forall |\omega| \leq \omega_0$$

2. There exist Hermitian matrixes $P, Q \in \mathbb{C}^{n\times n}$ such that $Q \succ 0$ and

$$\begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^* \begin{bmatrix} -Q & P \\ P & \omega_0^2 Q \end{bmatrix} \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \Theta \prec 0$$

If matrixes A, B and Θ are real, then the equivalence still holds when restricting P and Q to be real.

Proof: The first statement holds if-and-only-if

$$\zeta^* \Theta \zeta < 0, \forall \zeta \in \mathcal{G}$$

where

$$\mathcal{G} := \left\{ \begin{bmatrix} x \\ u \end{bmatrix} \in \mathbb{C}^{n+m} : u \neq 0, \imath\omega x = Ax + Bu \text{ for some } \omega \in \mathbb{R}, |\omega| \leq \omega_0 \right\}$$

Defining

$$\begin{bmatrix} f \\ g \end{bmatrix} := F \begin{bmatrix} x \\ u \end{bmatrix}, F := \begin{bmatrix} A & B \\ I & 0 \end{bmatrix}$$

and applying Lemma 2, the set $\mathcal{G}$ can be characterised as

$$\mathcal{G} = \{\zeta \neq 0 : \zeta^* S \zeta \leq 0, \forall S \in \mathcal{S}\}$$

where

$$\mathcal{S} := \left\{ F^* \begin{bmatrix} Q & P \\ P & -\omega_0^2 Q \end{bmatrix} F : P = P^*, Q = Q^* \succ 0 \right\}$$

From Lemma 3, the set $\mathcal{S}$ is loss-less and hence the S-procedure in Theorem 1 yields statements $(1) \Leftrightarrow (2)$.

To prove the real case result, assume that there exist complex Hermitian matrixes P and Q satisfying the first statement. Then:

$$(M + \imath N) = (M + \imath N)^* \succ 0 \Leftrightarrow \begin{bmatrix} M & -N \\ N & M \end{bmatrix} = \begin{bmatrix} M & -N \\ N & M \end{bmatrix}^\top \succ 0$$

holds for any real square matrixes M and N, one can show that the real parts of P and Q also satisfy the statement.

Iwasaki et al. [245] prove two corollaries to Theorem 2. The first shows how to extend Theorem 2 to an arbitrary finite frequency

interval by a change of variables. For a bandpass response $\omega_1 \leq \omega \leq \omega_2$ define a new variable $\hat{\omega}$, such that $|\hat{\omega}| \leq \omega_m$ where:

$$\begin{aligned}
\omega_m &= \frac{\omega_2 - \omega_1}{2} \\
\omega_c &= \frac{\omega_1 + \omega_2}{2} \\
\hat{\omega} &= \omega - \omega_c \\
\imath\omega I &= \imath\hat{\omega} I - \hat{A} \\
\hat{A} &= A - \imath\omega_c I
\end{aligned}$$

The LMI becomes:

$$\begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^* \begin{bmatrix} -Q & P + \imath\omega_c Q \\ P - \imath\omega_c Q & -\omega_1\omega_2 Q \end{bmatrix} \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \Theta \prec 0$$

This can be seen by multiplying out the LMI:

$$\begin{aligned}
&\begin{bmatrix} \hat{A} & B \\ I & 0 \end{bmatrix}^* \begin{bmatrix} -Q & P \\ P & \omega_m^2 Q \end{bmatrix} \begin{bmatrix} \hat{A} & B \\ I & 0 \end{bmatrix} = \ldots \\
&\begin{bmatrix} -\hat{A}^* Q \hat{A} + P\hat{A} + \hat{A}^* P + \omega_m^2 Q & -\hat{A}^* QB + PB \\ -B^* Q\hat{A} + B^* P & -B^* QB \end{bmatrix} = \ldots \\
&\begin{bmatrix} -\left(A - \imath\omega_c I\right)^* Q \left(A - \imath\omega_c I\right) + P\left(A - \imath\omega_c I\right) + \left(A - \imath\omega_c I\right)^* P + \omega_m^2 Q & -\left(A - \imath\omega_c I\right)^* QB + PB \\ -B^* Q\left(A - \imath\omega_c I\right) + B^* P & -B^* QB \end{bmatrix} = \ldots \\
&\begin{bmatrix} -A^* QA + A^*\left(P + \imath\omega_c Q\right) + \left(P - \imath\omega_c Q\right) A + \left(\omega_m^2 - \omega_c^2\right) Q & -A^* QB + \left(P - \imath\omega_c Q\right) B \\ -B^* QA + B^*\left(P + \imath\omega_c Q\right) & -B^* QB \end{bmatrix}
\end{aligned}$$

The second corollary shows how to express the Linear Matrix Inequality (LMI) of Theorem 2 in terms of real matrixes that represent the real and imaginary parts of P and Q.

O.2 *Iwasaki* and *Hara*'s generalised KYP lemma for discrete-time systems

Iwasaki and *Hara* [235] prove a generalised KYP lemma for discrete-time systems by first making a frequency transformation in the complex plane from the unit-circle to the imaginary axis and then applying the continuous-time KYP lemma.

Cheng [39] shows proofs of the discrete- and continuous-time KYP lemmas by the properties of the dual problems.

O.2.1 Frequency transformations in the complex plane

Iwasaki and *Hara* [235, Section 2.1] review transformations of frequency variables in the complex plane.

The quadratic function $\sigma : \mathbb{C} \times \mathbb{H}^{2\times 2} \to \mathbb{R}$ is defined as:

$$\sigma(\lambda, \Pi) := \begin{bmatrix} \lambda \\ I \end{bmatrix}^* \Pi \begin{bmatrix} \lambda \\ I \end{bmatrix}$$

Note that

$$\sigma\left(s, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right) = 0$$

implies $s^* + s = 0$ or $s \in \imath\mathbb{R}$. Similarly,

$$\sigma\left(z, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\right) = 0$$

implies $z^* z = 1$ or $z = e^{\imath\phi}$.

A frequency variable transformation $\mathcal{T} : \mathbb{C} \to \mathbb{C}$ is defined by

$$\mathcal{T}(s) := \frac{b - ds}{cs - a}$$
$$M := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

where $M \in \mathbb{C}^{2\times 2}$ and $s \in \mathbb{C}$. The following lemma shows how $\mathcal{T}$ transforms a region of the complex plane:

Lemma 1 [235, Lemma 1]: Given $\Omega,\ \Sigma \in \mathbb{H}^{2\times 2}$ and $M \in \mathbb{C}^{2\times 2}$, define:

$$\boldsymbol{S} := \{s \in \mathbb{C} : \sigma(s, \Omega) = 0, \sigma(s, \Sigma) \geq 0\}$$
$$\boldsymbol{\Lambda} := \{\lambda \in \mathbb{C} : \sigma(\lambda, M^*\Omega M) = 0, \sigma(\lambda, M^*\Sigma M) \geq 0\}$$

then:

$$\{\lambda \in \mathbb{C} : \lambda \in \boldsymbol{\Lambda}, c\lambda + d \neq 0\} = \{\mathcal{T}(s) \in \mathbb{C} : s \in \boldsymbol{S}, cs \neq a\}$$

The lemma is proved for arbitrary Ω by multiplying out $\sigma(\lambda, M^*\Omega M)$. For the right-hand side:

$$M \begin{bmatrix} \lambda \\ 1 \end{bmatrix} = \begin{bmatrix} a\lambda + b \\ c\lambda + d \end{bmatrix}$$

Dividing by $c\lambda + d$ and solving for s gives $\lambda = \mathcal{T}(s)$.

The following lemma gives a state space formula for systems obtained through the frequency transformation $\mathcal{T}$ of continuous-time systems:

Lemma 2 [235, Lemma 2]: Let $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$, $M \in \mathbb{C}^{2\times 2}$ be given. Suppose M is non-singular and A has no eigenvalues, λ, such that:

$$\sigma\left(\lambda, M^* \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} M\right) = 0$$

then $\det(dI + cA) \neq 0$ and the following matrixes are well defined:

$$\begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{bmatrix} := \begin{bmatrix} (bI + aA)\Gamma & (ad - bc)\Gamma B \\ \Gamma & -c\Gamma B \end{bmatrix}$$
$$\Gamma := (dI + cA)^{-1}$$

In this case, we have:

$$\det(sI - \mathcal{A}) \neq 0, \quad \text{for all } s \in \imath\mathbb{R}$$
$$(A, B) \text{ controllable } \Rightarrow (\mathcal{A}, \mathcal{B}) \text{ controllable}$$

Moreover, for any $s \in \imath\mathbb{R}$ such that $cs \neq a$, both $\mathcal{T}(s)$ and $(\mathcal{T}(s) I - A)^{-1}$ exist and

$$(\mathcal{T}(s) I - A)^{-1} B = \mathcal{C}(sI - \mathcal{A})^{-1}\mathcal{B} + \mathcal{D}$$

The final result can be justified as follows. Firstly:

$$\begin{aligned}
(\mathcal{T}(s) I - A) &= \left[\frac{b - ds}{cs - a} I - A\right] \\
&= \frac{1}{cs - a}\left[(b - ds) I - (cs - a) A\right] \\
&= \frac{1}{cs - a}\left[bI - dsI - csA + aA\right] \\
&= \frac{1}{cs - a}\left[(bI + aA) - s(dI + cA)\right] \\
&= \frac{1}{cs - a}\left[(bI + aA)\Gamma - sI\right]\Gamma^{-1}
\end{aligned}$$

Then:

$$\begin{aligned}
(sI - \mathcal{A})\mathcal{C}^{-1}(\mathcal{T}(s) I - A)^{-1} B &= \mathcal{B} + (sI - \mathcal{A})\mathcal{C}^{-1}\mathcal{D} \\
[sI - (bI + aA)\Gamma]\Gamma^{-1}(cs - a)\Gamma[(bI + aA)\Gamma - sI]^{-1} B &= (ad - bc)\Gamma B - [sI - (bI + aA)\Gamma]\Gamma^{-1} c\Gamma B \\
-(cs - a) &= (ad - bc)\Gamma - c[sI - (bI + aA)\Gamma] \\
-(cs - a)(dI + cA) &= (ad - bc) I - cs(dI + cA) + c(bI + aA) \\
-cdsI + adI - c^2 sA + acA &= adI - bcI - cdsI - c^2 sA + bcI + acA
\end{aligned}$$

O.2.2 A generalised discrete-time KYP lemma

Iwasaki and *Hara* [235, Section 2.2] prove the following generalised KYP lemma for discrete-time state variable systems:

Theorem 1 [235, Section 2.2]: Let complex matrixes A, B, $\Theta = \Theta^*$ and $(\Phi, \Psi) \in \mathbf{\Omega}$ be given where

$$\begin{aligned}
&\mathbf{\Omega} := \{ \\
&(\Phi, \Psi): \text{ there exists } \alpha, \beta \in \mathbb{R}, M \in \mathbb{C}^{2\times 2} \text{ such that:} \\
&\quad \Phi = M^* \begin{bmatrix} 0 & \alpha \\ \alpha & 0 \end{bmatrix} M, \text{ where } \alpha \det(M) \neq 0, \\
&\quad \Psi = M^* \begin{bmatrix} -1 & \beta \\ \beta & 1 \end{bmatrix} M \\
&\quad \}
\end{aligned} \tag{O.9}$$

Define

$$\mathbf{\Lambda} := \{\lambda \in \mathbb{C} : \sigma(\lambda, \Phi) = 0, \sigma(\lambda, \Psi) \geq 0\} \tag{O.10}$$

Suppose (A, B) is controllable and that A has no eigenvalues, λ, such that $\sigma(\lambda, \Phi) = 0$. Then the following statements are equivalent:

1. The following frequency domain condition holds for all $\lambda \in \mathbf{\Lambda}$:

$$\begin{bmatrix} (\lambda I - A)^{-1} B \\ I \end{bmatrix}^* \Theta \begin{bmatrix} (\lambda I - A)^{-1} B \\ I \end{bmatrix} \leq 0 \tag{O.11}$$

2. There exist Hermitian matrixes P and $Q \succeq 0$ such that:

$$\begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^* L(P, Q) \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \Theta \preceq 0 \tag{O.12}$$

   where $L(P, Q) = \Phi \otimes P + \Psi \otimes Q$.

Proof: Equation O.11 holds for all $\lambda \in \mathbf{\Lambda}$ such that $c\lambda + d \neq 0$. By Lemma 1, the condition is equivalent to:

$$\begin{bmatrix} (\mathcal{T}(s) I - A)^{-1} B \\ I \end{bmatrix}^* \Theta \begin{bmatrix} (\mathcal{T}(s) I - A)^{-1} B \\ I \end{bmatrix} \leq 0$$

for all $s \in \boldsymbol{S}$ such that $cs \neq a$ where:

$$\boldsymbol{S} := \{\imath\omega : \omega \in \mathbb{R}, |\omega| \leq 1\}$$

From Lemma 2:

$$\begin{bmatrix} \mathcal{C}(sI - \mathcal{A})^{-1} \mathcal{B} + \mathcal{D} \\ I \end{bmatrix}^* \Theta \begin{bmatrix} \mathcal{C}(sI - \mathcal{A})^{-1} \mathcal{B} + \mathcal{D} \\ I \end{bmatrix} \leq 0$$

where:

$$\begin{bmatrix} \mathcal{C}(sI - \mathcal{A})^{-1} \mathcal{B} + \mathcal{D} \\ I \end{bmatrix} = \begin{bmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{bmatrix} \begin{bmatrix} (sI - \mathcal{A})^{-1} \mathcal{B} \\ I \end{bmatrix}$$

Lemma 2 also implies that $\mathcal{A}$ has no eigenvalues on $\imath\mathbb{R}$ and that $(\mathcal{A}, \mathcal{B})$ is controllable. The continuous time finite frequency KYP lemma implies that this condition is equivalent to the existence of Hermitian matrixes X and $Q \succeq 0$ such that:

$$\begin{bmatrix} \mathcal{A} & \mathcal{B} \\ I & 0 \end{bmatrix}^* \begin{bmatrix} -Q & X \\ X & Q \end{bmatrix} \begin{bmatrix} \mathcal{A} & \mathcal{B} \\ I & 0 \end{bmatrix} + \begin{bmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{bmatrix}^* \Theta \begin{bmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{bmatrix} \preceq 0$$

The result follows by noting that:

$$\begin{bmatrix} \mathcal{A} & \mathcal{B} \\ I & 0 \end{bmatrix} \begin{bmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} aI & bI \\ cI & dI \end{bmatrix} \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} = (M \otimes I) \begin{bmatrix} A & B \\ I & 0 \end{bmatrix}$$

where:

$$\begin{bmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} \Gamma^{-1} & c\Gamma B \\ 0 & I \end{bmatrix}$$

and:

$$\begin{bmatrix} aI & bI \\ cI & dI \end{bmatrix}^* \begin{bmatrix} -Q & X \\ X & Q \end{bmatrix} \begin{bmatrix} aI & bI \\ cI & dI \end{bmatrix} = (\Phi \otimes P + \Psi \otimes Q)$$

with:

$$P := \frac{X - \beta Q}{\alpha}$$

The Octave script *yalmip_kyp_check_iir_lowpass_test.m* uses the generalised KYP lemma to check the frequency response of various implementations of an elliptic low-pass filter.

The Octave script *schurOneMPAlatticeDoublyPipelined2Abcd_kyp_symbolic_test.m* shows the KYP lemma for an IIR filter implemented as the combination of parallel doubly pipelined all pass one multiplier Schur lattice filters.

Characterisation of $\mathbf{\Omega}$

Iwasaki and *Hara* [235, Section 2.3] next consider the choice of α, β and M such that $(\Phi, \Psi) \in \mathbf{\Omega}$. First they state the following lemmas:

Lemma 3: Let $N \in \mathbb{C}^{2\times 2}$ and $\gamma \in \mathbb{R}$ be given such that $\det(N) \neq 0$. Then:

$$N^* \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} N = \begin{bmatrix} 0 & \gamma \\ \gamma & 0 \end{bmatrix}$$

holds if-and-only-if $N \in \boldsymbol{N}_\gamma$, where:

$$\boldsymbol{N}_\gamma := \left\{ J^* F J e^{\imath\omega} : F \in \mathbb{R}^{2\times 2}, \omega \in \mathbb{R}, \det(F) = \gamma \right\}$$
$$J := \begin{bmatrix} 1 & 0 \\ 0 & \imath \end{bmatrix}$$

Proof: Choose an arbitrary $F = \begin{bmatrix} p & q \\ r & s \end{bmatrix}$, substitute and multiply out.

Lemma 4: Let $\Phi \in \mathbb{H}$ be given such that $\det(\Phi) < 0$. Then the set of all $\alpha \in \mathbb{R}$ and $M \in \mathbb{C}^{2\times 2}$ such that:

$$\Phi = M^* \begin{bmatrix} 0 & \alpha \\ \alpha & 0 \end{bmatrix} M$$

is parameterised by

$$M = NK, \quad N \in \boldsymbol{N}_\gamma$$

where $\gamma := \frac{1}{\alpha}$ and K is an arbirary matrix satisfying

$$\Phi = K^* \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} K \tag{O.13}$$

Lemma 5: Given $\Upsilon \in \mathbb{H}$, there exists $\beta, \gamma \in \mathbb{R}$ and $N \in \boldsymbol{N}_\gamma$, such that $\gamma \neq 0$ and

$$N^* \begin{bmatrix} -1 & \beta \\ \beta & 1 \end{bmatrix} N = \Upsilon$$

if-and-only-if $\Upsilon \in \boldsymbol{\Upsilon} := \{\Upsilon \in \mathbb{H} : \det(\Re[J\Upsilon J^*]) < 0\}$.

Proof: For the previously defined F and γ, multiplying out gives $\det(\Re[J\Upsilon J^*]) = -\beta\gamma^2$

Lemma 6: Let $\Phi, \Psi \in \mathbb{H}$, then $(\Phi, \Psi) \in \mathbf{\Omega}$ if-and-only-if:

$$\det(\Phi) \leq 0, \quad \det(\Psi_o) \leq 0$$

hold where:

$$\Psi_o := \Re\left[(JK)^{-*} \Psi (JK)^{-1} \right]$$

where K is an arbitrary matrix as in Equation O.13.

Proof: Note that, by Lemma 4, $\det(\Phi)$ is negative if $(\Phi, \Psi) \in \mathbf{\Omega}$ and there exist $\beta, \gamma \in \mathbb{R}$ and $N \in \boldsymbol{N}_\gamma$ such that $\gamma \neq 0$ and:

$$\Psi = (NK)^* \begin{bmatrix} -1 & \beta \\ \beta & 1 \end{bmatrix} (NK)$$

From Lemma 5, this condition is equivalent to $K^{-*}\Psi K^{-1} \in \boldsymbol{\Upsilon}$.

Frequency restrictions for discrete-time filters

See *Iwasaki* and *Hara* [235, Section 3.2]. The Octave script *kyp_symbolic_frequency_transformation_test.m* uses the *symbolic* package to confirm the following transformations of a frequency interval on the unit-circle to the imaginary axis[f].

The discrete-time frequency variable $z = e^{\imath\omega}$ can be characterised by[g]:

$$\begin{aligned} \Phi &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \sigma(z, \Phi) &:= z^* z - 1 = 0 \end{aligned} \tag{O.14}$$

and K can be chosen as:

$$K = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

with:

$$\Phi_o = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Low-pass The low frequency condition is :

$$\mathbf{\Lambda}_{dl} := \{e^{\imath\omega} : \omega \in \mathbb{R}, |\omega| \leq \omega_l\}$$

where $|\omega| \leq \omega_l$ if-and-only-if $z + z^* \geq 2\cos\omega_l$ and we choose:

$$\begin{aligned} \gamma &= 2\cos\omega_l \\ \Psi &= \begin{bmatrix} 0 & 1 \\ 1 & -\gamma \end{bmatrix} \end{aligned}$$

In this case:

$$\Psi_o = \frac{1}{2} \begin{bmatrix} -2-\gamma & 0 \\ 0 & 2-\gamma \end{bmatrix}$$

and $\det(\Psi_o) < 0$ if-and-only-if $|\gamma| < 2$ or $0 < \omega_l < \pi$. The state space condition is:

$$L(P, Q) = \begin{bmatrix} P & Q \\ Q & -P-\gamma Q \end{bmatrix}$$

High-pass The high frequency condition is :

$$\mathbf{\Lambda}_{dh} := \{e^{\imath\omega} : \omega \in \mathbb{R}, \omega_h \leq |\omega| \leq \pi\}$$

where:

$$\gamma := 2\cos\omega_h$$

[f] Section O.3.1 extends these results to the union of frequency intervals.

[g] I do not know why *Iwasaki* and *Hara* [235, p.3831] use $-\Phi$ for the low-pass and band-pass cases of $L(P, Q)$. The only constraint on P is that it be Hermitian. Experiments with the Octave script *directFIRnonsymmetric_kyp_lowpass_test.m* confirmed that Φ and $-\Phi$ produce the same filters.

$$\Psi = \begin{bmatrix} 0 & -1 \\ -1 & \gamma \end{bmatrix}$$
$$\Psi_o = \frac{1}{2}\begin{bmatrix} 2+\gamma & 0 \\ 0 & \gamma - 2 \end{bmatrix}$$

Again $\det(\Psi_o) < 0$ if-and-only-if $|\gamma| < 2$ or $0 < \omega_h < \pi$ and the state space condition is:

$$L(P,Q) = \begin{bmatrix} P & -Q \\ -Q & -P+\gamma Q \end{bmatrix}$$

Band-pass The middle frequency condition is :

$$\mathbf{\Lambda}_{dm} := \{e^{\imath\omega} : \omega \in \mathbb{R}, \omega_1 \leq |\omega| \leq \omega_2\}$$

Alternatively:

$$|\omega - \omega_c| \leq \omega_m \text{ or } \cos(\omega - \omega_c) \geq \cos\omega_m$$

where $0 \leq \omega \leq \pi$ and:

$$\omega_c := \frac{\omega_2 + \omega_1}{2}$$
$$\omega_m := \frac{\omega_2 - \omega_1}{2}$$

This condition can be rewritten as $\sigma(e^{\imath\omega}, \Psi) \geq 0$ with[h]:

$$\gamma := 2\cos\omega_m$$
$$\Psi := \begin{bmatrix} 0 & e^{\imath\omega_c} \\ e^{-\imath\omega_c} & -\gamma \end{bmatrix}$$
$$\Psi_o = \begin{bmatrix} -\cos\omega_c - \cos\omega_m & \sin\omega_c \\ \sin\omega_c & \cos\omega_c - \cos\omega_m \end{bmatrix}$$

In this case $\det(\Psi_o) = -\sin^2\omega_m \leq 0$. The state space condition is:

$$L(P,Q) = \begin{bmatrix} P & e^{\imath\omega_c}Q \\ e^{-\imath\omega_c}Q & -P-\gamma Q \end{bmatrix}$$

Gain and phase

See *Iwasaki* and *Hara* [235, Section 3.1]. The gain and phase of the filter transfer function, $H(\lambda) := C(\lambda I - A)^{-1}B + D$, are determined by the choice of the Θ matrix in Equation O.11. As for Equation O.4:

$$\Theta = \begin{bmatrix} C & D \\ 0 & I \end{bmatrix}^* \Pi \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} \tag{O.15}$$

Equation O.11 now becomes:

$$\begin{bmatrix} H(\lambda) \\ I \end{bmatrix}^* \Pi \begin{bmatrix} H(\lambda) \\ I \end{bmatrix} \leq 0$$

For example, if $|H(\lambda)| \leq \varepsilon$:

$$\Pi = \begin{bmatrix} I & 0 \\ 0 & -\varepsilon^2 I \end{bmatrix}$$

Linearising the generalised discrete-time KYP lemma

The expansion of Θ in Equation O.15 is bi-linear in the C and D state variable matrixes. *Iwasaki* and *Hara* [236, Section VII, p.52] add a lemma:

[h]Note the correction to Ψ_o, demonstrated in the Octave script *kyp_symbolic_frequency_transformation_test.m*.

Lemma 8: For a transfer function $H(\lambda) = C(\lambda I - A)^{-1} B + D$, with m inputs and p outputs, $\sigma(H(\lambda), \Pi) \leq 0$ for all $\lambda \in \mathbf{\Lambda}$ holds if-and-only-if there exist Hermitian matrices P and $Q \succeq 0$ such that

$$\begin{bmatrix} \Gamma(P, Q, A, B, C, D) & [C\ D]^* \Pi_{11}^* \\ \Pi_{11} [C\ D] & -\Pi_{11}^* \end{bmatrix} \preceq 0 \tag{O.16}$$

holds, where the matrix $\Pi \in \mathbb{H}^{m+p}$ is such that

$$\Pi = \begin{bmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{12}^* & \Pi_{22} \end{bmatrix}, \Pi_{11} \in \mathbb{H}^p \text{ and } \Pi_{11} \succeq 0$$

and

$$\Gamma(P, Q, A, B, C, D) = \begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^* (\Phi \otimes P + \Psi \otimes Q) \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \begin{bmatrix} 0 & C^* \Pi_{12} \\ \Pi_{12}^* C & D^* \Pi_{12} + \Pi_{12}^* D + \Pi_{22} \end{bmatrix}$$

This lemma follows from the generalised discrete-time KYP theorem shown in Section O.2.2. Multiplying out Θ:

$$\begin{aligned}
\Theta = \begin{bmatrix} C & D \\ 0 & I \end{bmatrix}^* \Pi \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} &= \begin{bmatrix} C^* & 0 \\ D^* & I \end{bmatrix} \begin{bmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{12}^* & \Pi_{22} \end{bmatrix} \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} \\
&= \begin{bmatrix} C^* \Pi_{11} & C^* \Pi_{12} \\ D^* \Pi_{11} + \Pi_{12}^* & D^* \Pi_{12} + \Pi_{22} \end{bmatrix} \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} \\
&= \begin{bmatrix} C^* \Pi_{11} C & C^* \Pi_{12} + C^* \Pi_{11} D \\ \Pi_{12}^* C + D^* \Pi_{11} C & D^* \Pi_{11} D + D^* \Pi_{12} + \Pi_{12}^* D + \Pi_{22} \end{bmatrix} \\
&= \begin{bmatrix} 0 & C^* \Pi_{12} \\ \Pi_{12}^* C & D^* \Pi_{12} + \Pi_{12}^* D + \Pi_{22} \end{bmatrix} + \begin{bmatrix} C^* \Pi_{11} C & C^* \Pi_{11} D \\ D^* \Pi_{11} C & D^* \Pi_{11} D \end{bmatrix} \\
&= \begin{bmatrix} 0 & C^* \Pi_{12} \\ \Pi_{12}^* C & D^* \Pi_{12} + \Pi_{12}^* D + \Pi_{22} \end{bmatrix} + \begin{bmatrix} C & D \end{bmatrix}^* \Pi_{11} \begin{bmatrix} C & D \end{bmatrix}
\end{aligned}$$

If $\Pi \in \mathbb{H}^p$, Π_{11} is invertible, and $\Pi_{11} \succeq 0$ then the lemma is demonstrated by applying the *Schur complement*[i] to:

$$\Gamma(P, Q, A, B, C, D) - \left(\Pi_{11} \begin{bmatrix} C & D \end{bmatrix}\right)^* \left(-\Pi_{11}^{-*}\right) \left(\Pi_{11} \begin{bmatrix} C & D \end{bmatrix}\right) \preceq 0$$

O.2.3 Examples of FIR filter design with the generalised KYP lemma

The Octave script *directFIRnonsymmetric_kyp_lowpass_alternate_test.m* experiments with various combinations of constraints and objective functions. I concluded that the best approach is to optimise with an empty objective function and amplitude constraints found by trial and error.

Preliminaries

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 the desired pass band delay is d samples and the desired amplitude response is $\Delta = 1$ in the pass-band and $\Delta = 0$ otherwise.

[i] See Appendix B.6.2

Squared error calculation The Octave function *directFIRnonsymmetricEsqPW.m* calculates the piece-wise weighted mean-squared-error of the response of a non-symmetric FIR filter. Naturally, this function also applies to symmetric FIR filters. In this case the state transition matrix A has the same order and structure for symmetric and non-symmetric direct FIR filters. The function is described in Section N.2.

Amplitude constraints The desired response errors for each filter frequency band are encoded in the corresponding Π matrix. For example, see Equation O.5. For a low-pass filter, the pass-band and stop-band errors are:

$$\begin{aligned} \left|H\left(e^{\imath\omega}\right)^2 - H_D\left(e^{\imath\omega}\right)\right|^2 \leq \varepsilon_p^2 \quad 0 \leq |\omega| \leq \omega_p \leq \pi \\ \left|H\left(e^{\imath\omega}\right)\right|^2 \leq \varepsilon_s^2 \quad 0 \leq \omega_s \leq |\omega| \leq \pi \end{aligned}$$

where ε_p and ε_s are the pass-band and stop-band errors and ω_p and ω_s are the pass-band and stop-band edge angular frequencies respectively. If the desired filter pass-band delay is d samples then a constraint on the maximum pass-band response error is:

$$\left|H\left(e^{\imath\omega}\right) - e^{-\imath d\omega}\right|^2 \leq \varepsilon_z^2 \quad 0 \leq |\omega| \leq \omega_p \leq \pi$$

This constraint is implemented by modifying the C state variable matrix to subtract the response of a delayed input signal from the response of the desired filter.

A constraint on the maximum pass-band amplitude is:

$$\left|H\left(e^{\imath\omega}\right)\right|^2 \leq A_p^2 \quad 0 \leq |\omega| \leq \omega_p \leq \pi$$

An additional specification prevents over-shoot in the transition band:

$$\left|H\left(e^{\imath\omega}\right)\right|^2 \leq A_t^2 \quad 0 \leq \omega_p \leq |\omega| \leq \omega_s \leq \pi$$

Alternatively, apply an overall maximum amplitude constraint:

$$\left|H\left(e^{\imath\omega}\right)\right|^2 \leq A_{max}^2 \quad 0 \leq |\omega| \leq \pi$$

The Octave script *yalmip_kyp_lowpass_test.m* demonstrates each of these amplitude constraints.

The minimum passband amplitude constraint for a single-input single-output filter, $|H(\omega)| \geq A_{pl}^2$, cannot use Equation O.16 because it requires $\Pi_{11} \succeq 0$ and in this case $\Pi_{11} = -1$. The Octave script *yalmip_kyp_lowpass_test.m* uses Equation O.12 to check the minimum pass-band amplitude response of an FIR filter.

SDP constraints Assuming $\Psi_{11} = 0$, $\Psi_{21}^* = \Psi_{12}$, $\Pi_{11} = 1$ and $\Pi_{21} = \Pi_{12} = 0$, then for each frequency band response constraint:

$$\begin{bmatrix} A & B \\ I & 0 \end{bmatrix}^* \begin{bmatrix} P & \Psi_{12}Q \\ \Psi_{12}^*Q & -P + \Psi_{22}Q \end{bmatrix} \begin{bmatrix} A & B \\ I & 0 \end{bmatrix} + \begin{bmatrix} C & D \end{bmatrix}^* \begin{bmatrix} 1 & 0 \\ 0 & \Pi_{22} \end{bmatrix} \begin{bmatrix} C & D \end{bmatrix} \preceq 0 \tag{O.17}$$

where P, Q are Hermitian and $Q \succeq 0$. For an FIR filter, the unknown variables are P, Q, C and D. *Pipeleers* and *Vandeberghe* [61] show that if the A, B and Θ matrixes are real valued then the P and Q matrixes can be constrained to be real and symmetric.

Solving the generalised KYP lemma with a rank-1 constraint Equation O.17 has the form $\Gamma + X \preceq 0$ and the FIR filter coefficients are represented by the rank-1 matrix $X = \begin{bmatrix} C & D \end{bmatrix}^\top \begin{bmatrix} C & D \end{bmatrix}$.

The Octave script *yalmip_kyp_lmirank_lowpass_test.m* calls the YALMIP [108, 101] *lmirank* [214, 201] solver to design a symmetric low-pass FIR filter with a rank-1 SDP constraint. After running on my PC for 1276 minutes the script had performed 694 iterations and was still converging *very* slowly.

Similarly, the Octave script *yalmip_kyp_moment_lowpass_test.m* calls the YALMIP *moment* solver with a rank-1 constraint on the minimum pass-band amplitude. This script requires an *enormous* amount of memory and CPU time to design a symmetric 6-th order low-pass FIR filter.

Kheirandishfard et al. [154, 155, 156] describe a successive approximation algorithm that relaxes rank-1 constraints to LMIs. The Octave script *yalmip_parabolic_convex_bmi_test.m* shows some examples of this approach.

Design of a symmetric low pass FIR filter

The Octave script *directFIRsymmetric_kyp_lowpass_test.m* designs a symmetric FIR low-pass filter with YALMIP [108, 101] and SeDuMi. The YALMIP objective is empty. The YALMIP constraints for each frequency band are Equation O.16 with $Q \succeq 0$. The C matrix of the state variable description for the pass-band constraint is modified to subtract a signal delayed by M samples.

The filter specification is:

```
M=27 % Filter order is 2*M
fap=0.15 % Amplitude pass band edge
fas=0.2 % Amplitude stop band edge
Esq_z=8.1241e-06 % Squared pass band error from delay
Esq_s=1e-05 % Squared amplitude stop band error
```

The value of ε_z^2 was found by trial-and-error. The actual maximum squared-error in the pass-band compared with a pure delay is $\varepsilon_z^2 = 7.231\text{e-}06$ and the actual maximum stop-band squared-amplitude is $\varepsilon_s^2 = 9.1713\text{e-}06$. The resulting FIR impulse response is:

```
h = [ -0.0016663125, -0.0005569569,  0.0015609653,  0.0026312600, ...
       0.0006608913, -0.0030961277, -0.0042912485, -0.0002069004, ...
       0.0057854413,  0.0064118212, -0.0011565221, -0.0099072345, ...
      -0.0088172400,  0.0040467024,  0.0159839114,  0.0112961708, ...
      -0.0095006723, -0.0250744643, -0.0136006681,  0.0197719320, ...
       0.0400835388,  0.0154766519, -0.0421443472, -0.0730442956, ...
      -0.0166990048,  0.1272992063,  0.2831703033,  0.3504662325, ...
       0.2831703033,  0.1272992063, -0.0166990048, -0.0730442956, ...
      -0.0421443472,  0.0154766519,  0.0400835388,  0.0197719320, ...
      -0.0136006681, -0.0250744643, -0.0095006723,  0.0112961708, ...
       0.0159839114,  0.0040467024, -0.0088172400, -0.0099072345, ...
      -0.0011565221,  0.0064118212,  0.0057854413, -0.0002069004, ...
      -0.0042912485, -0.0030961277,  0.0006608913,  0.0026312600, ...
       0.0015609653, -0.0005569569, -0.0016663125 ];
```

Figure O.1 shows the amplitude response. The filter satisfies the constraints with no numerical problems.

For comparison, Figure O.2 shows a similar filter response designed by the *mcclellanFIRsymmetric* function described in Section N.5.3. The filter specification is:

```
M=27 % Filter order is 2*M
fap=0.15 % Amplitude pass band edge
fas=0.2 % Amplitude stop band edge
K=1 % Stop band weight
nplot=1000 % Number of frequency grid points in [0,0.5]
```

The distinct coefficients of the resulting FIR impulse response are:

```
hM = [ -0.0018633756, -0.0002139049,  0.0018111824,  0.0028130119, ...
        0.0007334626, -0.0030552469, -0.0041338503,  0.0000887765, ...
        0.0060387798,  0.0064314682, -0.0013291704, -0.0100048502, ...
       -0.0086422499,  0.0043759605,  0.0161393443,  0.0111171873, ...
       -0.0097989762, -0.0251216717, -0.0132991064,  0.0201144849, ...
        0.0400884282,  0.0151356005, -0.0424406434, -0.0729432755, ...
       -0.0162996088,  0.1275552885,  0.2829933607,  0.3500575486 ]';
```

Figure O.1: Amplitude response of a symmetric FIR low pass filter designed with the KYP lemma. See *Iwasaki* and *Hara* [236, Section VII.B.2, pp. 53-55].

Figure O.2: Amplitude response of a symmetric FIR low pass filter designed with the *Parks-McClellan* algorithm.

Design of a non-symmetric low pass FIR filter with given pass band delay

As an example, *Iwasaki* and *Hara* [236, 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, $\omega_p = 0.3\pi$ and $\omega_s = 0.4\pi$, $\varepsilon_s = 0.01$. They optimised the zero-phase pass-band sqared-error, $|H(\omega) - e^{-j\omega d}|^2 < \varepsilon_z^2$ and found $\varepsilon_z^2 = 0.0569$[j]. The Octave script *directFIRnonsymmetric_kyp_lowpass_test.m* designs a similar filter with YALMIP [108, 101] and SeDuMi. The YALMIP objective is empty. The YALMIP constraints for each frequency band are Equation O.16 with $Q \succeq 0$[k]. The C matrix of the state variable description for the pass-band constraint is modified to subtract a signal delayed by d samples.

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_z=0.00567 % Squared pass band error from delay
Esq_s=0.0001 % Squared amplitude stop band error
```

The value of ε_z^2 was found by trial-and-error. The actual maximum squared-error in the pass-band compared with a pure delay is $\varepsilon_z^2 = 0.00566258$ and the actual maximum stop-band squared-amplitude is $\varepsilon_s^2 = 0.00009970$. The resulting FIR impulse response is:

```
h = [ -0.0008739628,  0.0164442967,  0.0341368296,  0.0365351166, ...
       0.0084233754, -0.0378224961, -0.0579627953, -0.0056519543, ...
       0.1223672254,  0.2670300828,  0.3399958593,  0.2893246897, ...
       0.1434323887, -0.0075528889, -0.0800716260, -0.0574366632, ...
       0.0079436071,  0.0474642917,  0.0335721892, -0.0077203383, ...
      -0.0322780202, -0.0213933925,  0.0069364396,  0.0226280730, ...
       0.0139426422, -0.0057591371, -0.0160689036, -0.0099024981, ...
       0.0033706187,  0.0105515144,  0.0110751172 ];
```

Figure O.3 shows the amplitude and group delay response. The filter satisfies the constraints with no numerical problems.

Design of a non-symmetric FIR band pass filter with given pass band delay

The Octave script *directFIRnonsymmetric_kyp_bandpass_test.m* uses the generalised KYP lemma of *Iwasaki* and *Hara* to design a band pass FIR filter with YALMIP and SeDuMi. The YALMIP objective is empty. The YALMIP constraints for each frequency band are Equation O.16 with $Q \succeq 0$. The C matrix of the state variable description for the pass-band constraint is modified to subtract a signal delayed by d samples. The filter specification is:

```
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=4.67e-05 % Squared amplitude pass band - delay error
Esq_s=0.0001 % Squared amplitude stop band error
```

The value of ε_z^2 was found by trial-and-error. The actual maximum squared-error in the pass-band compared with a pure delay is $\varepsilon_z^2 = 0.00004622$. The resulting FIR impulse response is:

[j] This is probably a typo and should read, in my notation, $\varepsilon_z^2 = 0.00569$. I suspect that *Iwasaki* and *Hara* made ε_z^2 a YALMIP SDP variable with the constraint $\varepsilon_z^2 \leq 0.00569$ and an empty objective.

[k] The Octave script *directFIRnonsymmetric_kyp_lowpass_alternate_test.m* experiments with constraints on ε_z^2 or ε_s^2, with an empty objective or with the objective set to the sum of the pass-band squared-error. It seems that it is best to optimise with an empty objective and with fixed constraints on ε_z and ε_s found by trial-and-error. *Konopacki* and *Mościńska* [99, Equation 5] show an estimate of the attenuation of a low-pass non-symmetric FIR filter when the pass and stop band errors are equal. For the example of *Iwasaki* and *Hara* the estimated stop-band attenuation is $42dB$. For comparison, the Octave script *directFIRnonsymmetric_socp_slb_lowpass_test.m*, described in Section N.4.2, designs a similar filter using SOCP PCLS optimisation.

Figure O.3: Amplitude response of a non-symmetric FIR low pass filter designed with the KYP lemma. See *Iwasaki* and *Hara* [236, Section VII.B.2, pp. 53-55].

```
h = [ -0.0104143949, -0.0023682483,  0.0032719791, -0.0069032395, ...
       0.0099100648,  0.0630395379,  0.0323190391, -0.1185517528, ...
      -0.1553404297,  0.0708049785,  0.2578895336,  0.0833677118, ...
      -0.2119296948, -0.1909550140,  0.0609505534,  0.1436758056, ...
       0.0264655996, -0.0289757088,  0.0074300619, -0.0125615250, ...
      -0.0581536028, -0.0188770460,  0.0442642640,  0.0320711422, ...
      -0.0079897003, -0.0077190647,  0.0013262161, -0.0103678247, ...
      -0.0123694529,  0.0037079875,  0.0107180320 ];
```

As shown in Section N.1, the complementary FIR lattice reflection coefficients are:

```
k = [  0.99984130,  0.99993213,  0.99949924,  0.99898912, ...
       0.99993604,  0.99892181,  0.99994837,  0.99999003, ...
       0.99905269,  0.99976686,  0.98961334,  0.99761968, ...
       0.99052500,  0.99134934,  0.99977155,  0.99984184, ...
       0.99807139,  0.96873149,  0.91782544,  0.98563404, ...
       0.80792756,  0.98751343,  0.92625401,  0.97087760, ...
       0.99752953,  0.99617962,  0.99997130,  0.99995415, ...
       0.99985791,  0.99995704, -0.01731041 ]';
```

```
kc = [ -0.01781495, -0.01165050,  0.03164275,  0.04495269, ...
       -0.01131020, -0.04642426, -0.01016108, -0.00446454, ...
       -0.04351701,  0.02159226,  0.14375481,  0.06895638, ...
       -0.13733253, -0.13124969,  0.02137398, -0.01778461, ...
       -0.06207655,  0.24811145,  0.39698420, -0.16889507, ...
       -0.58928182, -0.15753485,  0.37689985,  0.23957604, ...
       -0.07024833, -0.08732797, -0.00757643, -0.00957607, ...
       -0.01685724,  0.00926906,  0.99985016 ]';
```

Figure O.4 shows the amplitude, phase and group delay responses. The pass band phase error shown is adjusted for the nominal delay.

Figure O.4: Amplitude, phase and group 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.

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 generalised KYP lemma of *Iwasaki* and *Hara* to design a band pass FIR Hilbert filter with with YALMIP and SeDuMi. The YALMIP objective is empty. The YALMIP constraints for each frequency band are Equation O.16 with $Q \succeq 0$. The C matrix of the state variable description for the pass-band constraint is modified to subtract a signal delayed by d samples. The filter specification is:

```
N=40 % 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=2.351e-05 % Squared amplitude pass band - delay error
Esq_s=0.0001 % Squared amplitude stop band error
```

The value of ε_z^2 was found by trial-and-error. The actual maximum squared-error in the pass-band compared with a pure delay is $\varepsilon_z^2 = 0.00002318$. The resulting FIR impulse response is:

```
h = [  0.0015805498, -0.0120066884, -0.0035827587, -0.0030951814, ...
      -0.0242644072, -0.0002086174,  0.0909872565,  0.0836755824, ...
      -0.1115995219, -0.2218415718,  0.0001074005,  0.2672488756, ...
       0.1622363410, -0.1479375956, -0.1976107417,  0.0002255810, ...
       0.0852731996,  0.0175125952,  0.0124236579,  0.0542103352, ...
      -0.0003322636, -0.0751000458, -0.0396066354,  0.0281585726, ...
       0.0228068395,  0.0000754434,  0.0147524619,  0.0148131467, ...
      -0.0168256666, -0.0238054300,  0.0001435084,  0.0093521191, ...
       0.0013350321,  0.0019242884,  0.0056596380, -0.0001113898, ...
      -0.0049750072, -0.0018590103,  0.0008309413, -0.0001250316, ...
       0.0001739622 ];
```

Figure O.5 shows the amplitude, phase and group delay responses. The pass band phase shown is adjusted for the nominal delay.

Figure O.5: 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.

Design of a non-symmetric high pass FIR filter with given pass band delay

The Octave script *directFIRnonsymmetric_kyp_highpass_test.m* designs a high-pass filter with YALMIP and SeDuMi. The YALMIP objective is empty. The YALMIP constraints for each frequency band are Equation O.16 with $Q \succeq 0$. The C matrix of the state variable description for the pass-band constraint is modified to subtract a signal delayed by d samples. The filter specification is:

```
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_max=1 % Maximum overall squared amplitude
Esq_z=0.00558 % Squared amplitude pass band error from delay
Esq_s=0.0001 % Squared amplitude stop band error
```

The value of ε_z^2 was found by trial-and-error. The actual maximum squared-error in the pass-band compared with a pure delay is $\varepsilon_z^2 = 0.00557280$. The resulting FIR impulse response is:

```
h = [ -0.0045825352,  0.0366768451, -0.0153827183, -0.0294813114, ...
      -0.0162397525,  0.0277250218,  0.0611649083,  0.0216628149, ...
      -0.1115244180, -0.2636171081,  0.6221374979, -0.2922988982, ...
      -0.1264764867,  0.0268274527,  0.0839883189,  0.0416028249, ...
      -0.0269685198, -0.0480611381, -0.0142322338,  0.0132827072, ...
       0.0414623810,  0.0065629447, -0.0258400656, -0.0203958772, ...
       0.0025293169,  0.0177805595,  0.0121693140, -0.0046190103, ...
      -0.0178807973, -0.0014015693,  0.0133257264 ];
```

Figure O.6 shows the amplitude and delay responses.

Figure O.6: Amplitude response of a non-symmetric FIR high pass filter designed with the KYP lemma.

O.2.4 *Rantzer*'s transformation of the KYP lemma from discrete-time to continuous-time

Rantzer [10, Proof of Theorem 2] shows an alternative transformation of the KYP lemma from discrete-time to continuous-time by parameterising the unit circle with $(1+\imath\omega)/(1-\imath\omega)$ rather than $e^{\imath\omega}$. Multiplying out:

$$\begin{aligned}
\left[\frac{1+\imath\omega}{1-\imath\omega}I - A\right]x &= Bu \\
\left[\imath\omega\left(A+I\right)-\left(A-I\right)\right]x+\imath\omega B &= Bu \\
\left[\imath\omega I-\left(A-I\right)\left(A+I\right)^{-1}\right]\left(A+I\right)x+\imath\omega B &= Bu \\
\left[\imath\omega I-\left(A-I\right)\left(A+I\right)^{-1}\right]\left(Ix+Ax+Bu\right) &= \left[I-\left(A-I\right)\left(A+I\right)^{-1}\right]Bu \\
\left[\imath\omega I-\left(A-I\right)\left(A+I\right)^{-1}\right]\left(Ix+Ax+Bu\right) &= \left[\left(A+I\right)-\left(A-I\right)\right]\left(A+I\right)^{-1}Bu \\
\left[\imath\omega I-\left(A-I\right)\left(A+I\right)^{-1}\right]\left(Ix+Ax+Bu\right) &= 2\left(A+I\right)^{-1}Bu
\end{aligned}$$

Rantzer introduces:

$$\begin{aligned}
\hat{A} &= \left(A-I\right)\left(A+I\right)^{-1} \\
\hat{B} &= 2\left(A+I\right)^{-1}B \\
\hat{x} &= Ix+Ax+Bu \\
S &= \begin{bmatrix} \left(A+I\right)^{-1} & -\left(A+I\right)^{-1}B \\ 0 & I \end{bmatrix} \\
\hat{\Theta} &= S^{\top}\Theta S \\
\hat{P} &= P/2
\end{aligned}$$

The corresponding frequency domain condition is:

$$\begin{bmatrix} \left(\imath\omega I-\hat{A}\right)^{-1}\hat{B} \\ I \end{bmatrix}^{*}\hat{\Theta}\begin{bmatrix} \left(\imath\omega I-\hat{A}\right)^{-1}\hat{B} \\ I \end{bmatrix} \leq 0$$

which the continuous-time KYP lemma tells us is equivalent to:

$$\begin{bmatrix} \hat{A}^{\top}\hat{P}+\hat{P}\hat{A} & \hat{P}\hat{B} \\ \hat{B}^{\top}\hat{P} & 0 \end{bmatrix}+\hat{\Theta} \preceq 0$$

for some symmetric real $\hat{P}$. Multiplying by $\begin{bmatrix} A+I & B \\ 0 & I \end{bmatrix}$ on the right and its transpose on the left gives:

$$\begin{aligned}
&\begin{bmatrix} A+I & B \\ 0 & I \end{bmatrix}^{\top}\begin{bmatrix} \hat{A}^{\top}\hat{P}+\hat{P}\hat{A} & \hat{P}\hat{B} \\ \hat{B}^{\top}\hat{P} & 0 \end{bmatrix}\begin{bmatrix} A+I & B \\ 0 & I \end{bmatrix}+\begin{bmatrix} A+I & B \\ 0 & I \end{bmatrix}^{\top}S^{\top}\Theta S\begin{bmatrix} A+I & B \\ 0 & I \end{bmatrix} \preceq 0 \\
&\begin{bmatrix} \left(A+I\right)^{\top}\left(\hat{A}^{\top}\hat{P}+\hat{P}\hat{A}\right) & \left(A+I\right)^{\top}\hat{P}\hat{B} \\ B^{\top}\left(\hat{A}^{\top}\hat{P}+\hat{P}\hat{A}\right)+\hat{B}^{\top}\hat{P} & B^{\top}\hat{P}\hat{B} \end{bmatrix}\begin{bmatrix} A+I & B \\ 0 & I \end{bmatrix}+\cdots \\
&\begin{bmatrix} A+I & B \\ 0 & I \end{bmatrix}^{\top}\begin{bmatrix} \left(A+I\right)^{-1} & -\left(A+I\right)^{-1}B \\ 0 & I \end{bmatrix}^{\top}\Theta\begin{bmatrix} \left(A+I\right)^{-1} & -\left(A+I\right)^{-1}B \\ 0 & I \end{bmatrix}\begin{bmatrix} A+I & B \\ 0 & I \end{bmatrix} \preceq 0 \\
&\begin{bmatrix} \left(A+I\right)^{\top}\left(\hat{A}^{\top}\hat{P}+\hat{P}\hat{A}\right)\left(A+I\right) & \left(A+I\right)^{\top}\left(\hat{A}^{\top}\hat{P}+\hat{P}\hat{A}\right)B+\left(A+I\right)^{\top}\hat{P}\hat{B} \\ B^{\top}\left(\hat{A}^{\top}\hat{P}+\hat{P}\hat{A}\right)\left(A+I\right)+\hat{B}^{\top}\hat{P}\left(A+I\right) & B^{\top}\left(\hat{A}^{\top}\hat{P}+\hat{P}\hat{A}\right)B+\hat{B}^{\top}\hat{P}B+B^{\top}\hat{P}\hat{B} \end{bmatrix}+\Theta \preceq 0 \\
&\begin{bmatrix} \left(A-I\right)^{\top}\hat{P}\left(A+I\right)+\left(A+I\right)^{\top}\hat{P}\left(A-I\right) & \left(A-I\right)^{\top}\hat{P}B+\frac{1}{2}\left(A+I\right)^{\top}\hat{P}\left(A+I\right)\hat{B} \\ \frac{1}{2}\hat{B}^{\top}\left(A+I\right)^{\top}\hat{P}\left(A+I\right)+B^{\top}\hat{P}\left(A-I\right) & \frac{1}{2}\hat{B}^{\top}\left(A+I\right)^{\top}\hat{P}B+\frac{1}{2}B^{\top}\hat{P}\left(A+1\right)\hat{B} \end{bmatrix}+\Theta \preceq 0 \\
&\begin{bmatrix} 2A^{\top}\hat{P}A-2\hat{P} & 2A^{\top}\hat{P}B \\ 2B^{\top}\hat{P}A & 2B^{\top}\hat{P}B \end{bmatrix}+\Theta \preceq 0 \\
&\begin{bmatrix} A^{\top}PA-P & A^{\top}PB \\ B^{\top}PA & B^{\top}PB \end{bmatrix}+\Theta \preceq 0
\end{aligned}$$

I experiment with this conversion in the Octave script *yalmip_kyp_rantzer_test.m* and find that SeDuMi calculates $P \approx -2\hat{P}$.

O.2.5 *Finsler*'s lemma transformation of the generalised KYP lemma

Ren et al. [208, Lemma 3] write *Finsler*'s lemma in LMI form as:

Lemma: Let $P = P^\top$ and matrix A with orthogonal complement, $A^\perp$, such that $A^\perp A = 0$, then the following statements are equivalent:

1. $A^\perp P \left(A^\perp\right)^\top \prec 0$
2. There exists a matrix X satisfying $P + AX + X^\top A^\top \prec 0$

Ren et al. [208, Theorem 3] apply *Finsler*'s Lemma to the generalised KYP lemma matrix inequality shown in Equation O.12:

Theorem: Matrixes $P \in \mathbb{H}_n$, $Q \in \mathbb{H}_n$, $Q \succeq 0$ and $X \in \mathbb{C}^{n\times n}$, $Y \in \mathbb{C}^{n\times n}$, $Z \in \mathbb{C}^{m\times n}$ exist such that:

$$\begin{bmatrix} L(P,Q) & 0 & 0 \\ 0 & -\varepsilon^2 I_m & 0 \\ 0 & 0 & I_p \end{bmatrix} + UV + V^*U^* \preceq 0 \tag{O.18}$$

where we assume that $\Pi = \begin{bmatrix} I_p & 0 \\ 0 & -\varepsilon^2 I_m \end{bmatrix}$ and define $U = \begin{bmatrix} -I_n & A & B & 0 \\ 0 & C & D & -I_m \end{bmatrix}^*$, $V = \begin{bmatrix} X & 0 \\ Y & 0 \\ Z & 0 \\ 0 & I_p \end{bmatrix}^*$.

Recall that the state-variable system of equations has n states, m inputs and p outputs so that $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$, $C \in \mathbb{C}^{p\times n}$, $D \in \mathbb{C}^{p\times m}$, $P \in \mathbb{C}^{n\times n}$, $Q \in \mathbb{C}^{n\times n}$, $L(P,Q) \in \mathbb{C}^{2n\times 2n}$, $\Theta \in \mathbb{C}^{(n+m)\times(n+m)}$, $\Pi \in \mathbb{C}^{(p+m)\times(p+m)}$, $\Pi_{11} \in \mathbb{C}^{p\times p}$ and $\Pi_{22} \in \mathbb{C}^{m\times m}$.

Ren et al. rewrite Equation O.12 as:

$$\begin{bmatrix} A & B \\ I_n & 0 \\ 0 & I_m \end{bmatrix}^* \left\{ \begin{bmatrix} L(P,Q) & 0 \\ 0 & 0_m \end{bmatrix} + \begin{bmatrix} 0_n & 0 \\ 0 & \Theta \end{bmatrix} \right\} \begin{bmatrix} A & B \\ I_n & 0 \\ 0 & I_m \end{bmatrix} \preceq 0$$

and define $\tilde{U}^\perp = \begin{bmatrix} A & B \\ I_n & 0 \\ 0 & I_m \end{bmatrix}^*$, $\tilde{U} = \begin{bmatrix} -I_n & A & B \end{bmatrix}^*$, $\tilde{V} = \begin{bmatrix} X^* & Y^* & Z^* \end{bmatrix}$.

Applying *Finsler*'s lemma:

$$\begin{bmatrix} L(P,Q) & 0 \\ 0 & 0_m \end{bmatrix} + \begin{bmatrix} 0_n & 0 \\ 0 & \Theta \end{bmatrix} + \tilde{U}\tilde{V} + \tilde{V}^*\tilde{U}^* \preceq 0$$

where:

$$\tilde{U}\tilde{V} = \begin{bmatrix} -I_n \\ A^* \\ B^* \end{bmatrix} \begin{bmatrix} X^* & Y^* & Z^* \end{bmatrix} = \begin{bmatrix} -X^* & -Y^* & -Z^* \\ A^*X^* & A^*Y^* & A^*Z^* \\ B^*X^* & B^*Y^* & B^*Z^* \end{bmatrix}$$

Also:

$$\begin{bmatrix} 0_n & 0 \\ 0 & \Theta \end{bmatrix} = \begin{bmatrix} 0_n & 0 \\ 0 & \begin{bmatrix} C & D \end{bmatrix}^* I_p \begin{bmatrix} C & D \end{bmatrix} \end{bmatrix} + \begin{bmatrix} 0_{2n} & 0 \\ 0 & -\varepsilon^2 I_m \end{bmatrix}$$

Apply the Schur complement to:

$$\begin{bmatrix} \tilde{U}\tilde{V} + \tilde{V}^*\tilde{U}^* & 0 \\ 0 & 0_p \end{bmatrix} + \begin{bmatrix} 0_n & 0 \\ 0 & \begin{bmatrix} C & D \end{bmatrix}^* I_p \begin{bmatrix} C & D \end{bmatrix} \end{bmatrix}$$

so that:

$$\begin{bmatrix} -X^* & -Y^* & -Z^* & 0 \\ A^*X^* & A^*Y^* & A^*Z^* & C^* \\ B^*X^* & B^*Y^* & B^*Z^* & D^* \\ 0 & 0 & 0 & -I_p \end{bmatrix} + \begin{bmatrix} -X & XA & XB & 0 \\ -Y & YA & YB & 0 \\ -Z & ZA & ZB & 0 \\ 0 & C & D & -I_p \end{bmatrix} + \begin{bmatrix} 0_{2n+m} & 0 \\ 0 & I_p \end{bmatrix} = UV + V^*U^* + \begin{bmatrix} 0_{2n+m} & 0 \\ 0 & I_p \end{bmatrix}$$

The Octave script *yalmip_kyp_finsler_test.m* checks Equation O.18.

The Octave script *directFIRnonsymmetric_kyp_finsler_lowpass_test.m* repeats the filter design of Section O.2.3 with the Finsler transformation of the generalised KYP lemma.

O.2.6 The dual of the KYP lemma

Wallin, *Vandeberghe* and others note that solution of the KYP lemma by Equation O.3 is an $\mathcal{O}\left(n^6\right)$ complexity problem, that solving the *dual* problem reduces this to $\mathcal{O}\left(n^4\right)$ and that exploiting the structure of the dual problem can reduce the complexity further to $\mathcal{O}\left(n^3\right)$ [27, 129, 207, 206, 39].

The *adjoint* of the discrete-time KYP linear mapping

Cheng [39] shows proofs of the continuous-time and discrete-time KYP lemmas by duality. He begins by reviewing the definition of the *adjoint* of the linear mapping of a hermitian matrix. The inner product defined on the Hilbert space of $n \times n$ Hermitian matrixes, $\mathbb{H}^n$, is[1]:

$$\langle A, B \rangle = \operatorname{trace}\left(A^* B\right) = \operatorname{trace}\left(AB\right)$$

Also:

> ... given two Hilbert spaces V and W and a linear mapping $\mathcal{A} : V \to W$, the adjoint mapping of $\mathcal{A}$, denoted $\mathcal{A}^{adj}$, is a linear mapping from W to V such that
>
> $$\forall x \in V, y \in W, \langle \mathcal{A}\left(x\right), y \rangle_W = \langle x, \mathcal{A}^{adj}\left(y\right) \rangle_V$$

The linear mapping associated with the discrete-time KYP lemma is $\mathcal{D} : \mathbb{H}^n \to \mathbb{H}^{n+m}$:

$$\mathcal{D}\left(P\right) = \begin{bmatrix} A^*PA - P & A^*PB \\ B^*PA & B^*PB \end{bmatrix} \tag{O.19}$$

The adjoint mapping, $\mathcal{D}^{adj} : \mathbb{H}^{n+m} \to \mathbb{H}^n$, is:

$$\mathcal{D}^{adj}\left(Z\right) = AZ_{11}A^* - Z_{11} + BZ_{12}^*A^* + AZ_{12}B^* + BZ_{22}B^*$$

where

$$Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{12}^* & Z_{22} \end{bmatrix}$$

This follows from the definition of the inner product:

$$\begin{aligned} \langle \mathcal{D}\left(P\right), Z \rangle &= \operatorname{trace}\left(\mathcal{D}\left(P\right) Z\right) \\ &= \operatorname{trace}\left(\begin{bmatrix} A^*PA - P & A^*PB \\ B^*PA & B^*PB \end{bmatrix} \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{12}^* & Z_{22} \end{bmatrix}\right) \\ &= \operatorname{trace}\left(\begin{bmatrix} A^*PAZ_{11} - PZ_{11} + A^*PBZ_{12}^* & \cdots \\ \cdots & B^*PAZ_{12} + B^*PBZ_{22} \end{bmatrix}\right) \\ &= \operatorname{trace}\left(PAZ_{11}A^* - PZ_{11} + PBZ_{12}^*A^* + PAZ_{12}B^* + PBZ_{22}B^*\right) \\ &= \langle P, \mathcal{D}^{adj}\left(Z\right) \rangle \end{aligned}$$

and the following properties of trace:

$$\begin{aligned} \operatorname{trace}\left(A + B\right) &= \operatorname{trace}\left(A\right) + \operatorname{trace}\left(B\right) \\ \operatorname{trace}\left(AB\right) &= \operatorname{trace}\left(A^\top B\right) = \operatorname{trace}\left(AB^\top\right) = \operatorname{trace}\left(B^\top A\right) = \operatorname{trace}\left(BA^\top\right) \\ \operatorname{trace}\left(ABCD\right) &= \operatorname{trace}\left(BCDA\right) = \operatorname{trace}\left(CDAB\right) = \operatorname{trace}\left(DABC\right) \\ \operatorname{trace}\left(ABC\right) &= \operatorname{trace}\left(\left(ABC\right)^\top\right) = \operatorname{trace}\left(CBA\right) \end{aligned}$$

[1] See Section B.2

The dual of the discrete-time KYP lemma

Cheng proves the following dual of the discrete-time KYP lemma with the linear mapping shown in Equation O.19:

Proposition 2 [39, Section 3.5]: There exists $Z \in \mathbb{H}^{n+m}$ such that:

$$Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{12}^* & Z_{22} \end{bmatrix} \prec 0$$
$$AZ_{11}A^* - Z_{11} + BZ_{12}^*A^* + AZ_{12}B^* + BZ_{22}B^* = 0$$
$$\operatorname{trace}(\Theta Z) \leq 0$$

Here Θ is as in Equation O.15. The Octave script *yalmip_kyp_dual_test.m* calls YALMIP to check the dual of the KYP lemma for the maximum amplitude of a low-pass FIR filter designed with the Octave *remez* function. Unfortunately, in this script, SeDuMi fails for the KYP lemma linear mapping of Equation O.19 but the SDPT3 solver succeeds. However, SeDuMi does solve the dual mapping and solves the primal problem when the YALMIP *'dualize'* option is set.

The dual of the generalised discrete-time KYP lemma

Seungil and *Doyle* [226] prove the generalised discrete-time KYP lemma by the dual of the Lagrangian of Equation O.12.

O.3 Generalisation of the KYP lemma to the union of disjoint frequency intervals

Pipeleers, *Iwasaki* and *Hara* [63] describe the generalisation of the KYP lemma to the union of disjoint frequency intervals.

O.3.1 Union of frequency intervals

Pipeleers, *Iwasaki* and *Hara* generalise $\mathbf{\Lambda}(\Phi, \Psi)$, to a union of $l \in \mathbb{N}$ 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:

$$\begin{aligned} L_0(\lambda) &= 1 \\ L_l(\lambda) &= \left[\lambda^l,\ \lambda^{l-1},\ \cdots,\ \lambda,\ 1\right]^\top \end{aligned}$$

so that:

$$\mathbf{\Lambda}(\Phi, \Psi) = \left\{\lambda \in \mathbb{C} : L_l(\lambda)^* \Phi L_l(\lambda) = 0, L_l(\lambda)^* \Psi L_l(\lambda) \geq 0\right\}$$

If $\mathbf{\Lambda}(\Phi, \Psi)$ is unbounded then it is extended with

$$L_l(\infty) = [1\ 0_{1,l}]^\top$$

In addition, *Pipeleers et al.* define

$$L_l(\lambda)^{\mathsf{F}} = \left[1,\ \lambda^1,\ \cdots,\ \lambda^l\right]^\top$$

Pipeleers et al. make the following two assumptions about the matrixes Φ and Ψ.

Assumption 1: The matrixes Φ and Ψ can be decomposed as

$$\Phi = T^* \Phi_o T, \quad \Psi = T^* \Psi_o T \tag{O.20}$$

where

$$\Phi_o = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \Psi_o = \begin{bmatrix} \alpha & \beta \\ \beta & \gamma \end{bmatrix} \text{ where } \alpha < 0 < \gamma \text{ or } 0 \leq \alpha \leq \gamma \tag{O.21}$$

for some matrix $T \in \mathbb{C}^{2\times(l+1)}$ with full row rank and some $\alpha, \beta, \gamma \in \mathbb{R}$.

For each $s \in \mathbf{\Lambda}(\Phi_o, \Psi_o)$, the l-th degree polynomial in λ

$$\begin{aligned} [1\ \ -s]\ T\ L_l(\lambda) = 0, &\quad \text{for } s \neq \infty \\ [0\ \ \ 1]\ T\ L_l(\lambda) = 0, &\quad \text{for } s = \infty \end{aligned} \tag{O.22}$$

has l distinct roots grouped in the set $\mathcal{R}_T(s)$.

Assumption 2: When $l \geq 2$ we assume that there exists a Hermitian matrix $R \in \mathbb{H}^l$ such that:

$$\begin{aligned} L_{l-1}(\lambda)^* R\ L_{l-1}(\lambda) > 0 &\quad \text{for all } \lambda \in \mathbf{\Lambda}(\Phi, \Psi) \\ L_{l-1}(\lambda_p)^* R\ L_{l-1}(\lambda_q) = 0 &\quad \text{for all } \lambda_p, \lambda_q \in \mathcal{R}_T(s),\ p \neq q,\ \text{for all } s \in \mathbf{\Lambda}(\Phi_o, \Psi_o) \end{aligned}$$

Assumption 2 implies that $R \succ 0$. 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, T, defines a mapping between the curves $\mathbf{\Lambda}(\Phi, \Psi)$ and a segment of the imaginary axis corresponding to $\mathbf{\Lambda}(\Phi_o, \Psi_o)$. 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 λ 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.

For $l = 1$ the curves, $\mathbf{\Lambda}(\phi, \psi)$, correspond to the single non-empty and non-singular segments of a circle or line considered in the generalised KYP lemma of *Iwasaki* and *Hara* [236] described in Section O.2.2. If, in addition, $\Phi = 0$, then the curve corresponds to the entire circle or line and the original KYP lemma.

Union of frequency intervals on the real axis

Pipeleers, *Iwasaki* and *Hara* [63, Section 3] describe the construction of the matrixes Φ, Ψ and R over a union of segments of the real axis. Subsequently, the extension of this mapping to the unit circle in the complex plane is obtained by a Möbius transformation.

Here T_r defines a bijective mapping between $s \in \mathbf{\Lambda}(\Phi_0, \Psi_0)$ and $\kappa \in \mathbf{\Lambda}(\Phi_r, \Psi_r)$ and $\tilde{T}$ maps each κ into l roots $\lambda_k \in [\alpha_k, \beta_k]$.

Firstly, on the real axis Φ_r and Ψ_r are defined by:

$$\mathbf{\Lambda}(\Phi_r, \Psi_r) = \{\lambda \in \mathbb{R} \ : \ L_1^*(\lambda)\Phi_r L_1(\lambda) = 0 \ , \ L_1^*(\lambda)\Psi_r L_1(\lambda) \geq 0\} = \mathbb{R}_+ \cup \{\infty\}$$

In this case *Pipeleers et al.* use:

$$\Phi_r = \begin{bmatrix} 0 & \imath \\ -\imath & 0 \end{bmatrix}, \quad \Psi_r = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

so that:

$$L_1^*(\lambda)\Phi_r L_1(\lambda) = \begin{bmatrix} \lambda & 1 \end{bmatrix} \begin{bmatrix} 0 & \imath \\ -\imath & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ 1 \end{bmatrix} = 0$$

and:

$$L_1^*(\lambda)\Psi_r L_1(\lambda) = \begin{bmatrix} \lambda & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ 1 \end{bmatrix} = 2\lambda > 0$$

For $l \in \mathbb{N}$, 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 [63, Lemma 3.1] : Let $2l$ scalars $\alpha_k, \beta_k \in \mathbb{R} \cup \{\infty\}$, $k \in \mathbb{N}_l$, 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 $\Phi, \Psi \in \mathbb{H}^{l+1}$:

$$\Phi = \tilde{T}^* \Phi_r \tilde{T}$$
$$\Psi = \tilde{T}^* \Psi_r \tilde{T}$$

satisfy Assumptions 1 and 2, and

$$\mathbf{\Lambda}\left(\Phi,\Psi\right)=\bigcup_{k=1}^{l}\left[\alpha_k,\beta_k\right]$$

In particular, a matrix $R\in\mathbb{S}^l$ satisfying Assumption 2 is given by the unique solution of:

$$\begin{aligned}&\Phi=J_l^*\left(\Phi_r\otimes R\right)J_l\\&L_{l-1}\left(\lambda\right)^*RL_{l-1}\left(\lambda\right)>0,\quad\forall\lambda\in\mathbb{R}\end{aligned}$$

Pipeleers et al. prove this lemma by defining a mapping between $\kappa\in\mathbf{\Lambda}\left(\Phi_r,\Psi_r\right)=\mathbb{R}_+\cup\{\infty\}$ and $\lambda\in\mathbf{\Lambda}\left(\Phi,\Psi\right)$ where every κ is mapped onto the l roots of:

$$\left[\begin{array}{cc}1&-\kappa\end{array}\right]\tilde{T}L_l\left(\lambda\right)=-a^\top L_l\left(\lambda\right)-\kappa b^\top L_l\left(\lambda\right)=0$$

For $\kappa=\infty$ this reads as $b^\top L_l\left(\lambda\right)=0$. Hence, $\mathbf{\Lambda}\left(\Phi,\Psi\right)$ corresponds to the root-locus plot[m] of:

$$\frac{b^\top L_l\left(\lambda\right)}{a^\top L_l\left(\lambda\right)}$$

Pipeleers et al. [63, Figure 2] illustrate this lemma as a transformation

$$1-s\frac{\imath\kappa+\imath}{\kappa-1}=0$$

from $s\in[-\imath,\imath]$ to $\kappa\in\mathbb{R}_+\cup\{\infty\}$ followed by a transformation

$$1+\kappa\frac{\prod_{k=1}^{l}\left(\lambda-\beta_k\right)}{\prod_{k=1}^{l}\left(\lambda-\alpha_k\right)}=0$$

from κ to the l distinct roots $\lambda_k\in[\alpha_k,\beta_k]$.

Pipeleers et al. state that the following similarity transformation shows that *Assumption 1* is satisfied:

$$\Phi_r=T_r^*\Phi_oT_r,\quad\Psi_r=T_r^*\Psi_oT_r\tag{O.23}$$

with:

$$T_r=\frac{1}{\sqrt{2}}\left[\begin{array}{cc}1&-1\\\imath&\imath\end{array}\right],\quad\Psi_o=\left[\begin{array}{cc}-1&0\\0&1\end{array}\right]\tag{O.24}$$

and $T=T_r\tilde{T}$. Here T_r defines a bijective mapping between $s\in\mathbf{\Lambda}\left(\Phi_0,\Psi_0\right)$ and $\kappa\in\mathbf{\Lambda}\left(\Phi_r,\Psi_r\right)$ and $\tilde{T}$ maps each κ into l roots $\lambda_k\in[\alpha_k,\beta_k]$.

The first part of the Octave script *kyp_complex_curve_union_test.m* shows the mapping of the segment of the imaginary axis, $s\in[-\imath,\imath]$, to a union of segments of the real axis $\lambda\in[\alpha_k,\beta_k]$. The real parts of the roots of

$$\left[\begin{array}{cc}1&0\end{array}\right]T-s\left[\begin{array}{cc}0&1\end{array}\right]T=0$$

are plotted against $\Im s$. In each case the imaginary part of the roots is 0. Figure O.7 shows the plot of the root locus when the intervals are represented by:

```
alpha_m1 = [ -0.75, -0.25,  0.25,  0.75 ];

beta_m1 = [ -0.60, -0.10,  0.40,  0.90 ];
```

Figure O.8 shows the plot of the root locus when the intervals are represented by:

```
alpha_m2 = [  -Inf, -0.60, -0.10,  0.40 ];

beta_m2 = [ -0.75, -0.25,  0.25,   Inf ];
```

[m] If the closed-loop transfer function of a feedback control system is:

$$\frac{G\left(s\right)}{1+G\left(s\right)H\left(s\right)},\quad G\left(s\right)H\left(s\right)=K\frac{\left(s-z_1\right)\ldots\left(s-z_m\right)}{\left(s-p_1\right)\ldots\left(s-p_n\right)}$$

then the root-locus plot consists of the roots of the characteristic equation $1+G\left(s\right)H\left(s\right)=0$ for any value of K.

Figure O.7: Test of Assumption 1 for an interval on the imaginary s axis mapped to a union of intervals on the real axis.

Figure O.8: Test of Assumption 1 for an interval on the imaginary s axis mapped to a union of intervals on the real axis with $\pm\infty$ endpoints.

Union of frequency intervals on a circle or line

Pipeleers et al. [63, pp. 3627-3629] [62, p. 3915] consider the union of l non-empty, non-singleton, non-intersecting segments $\mathbf{\Lambda}(\Phi, \Psi_k)$, $k \in \mathbb{N}_l$ on an arbitrary circle or line $\mathbf{\Lambda}(\Phi, 0)$. They use a Möbius transformation that maps $\mathbf{\Lambda}(\Phi, 0)$ to the real axis, $\mathbf{\Lambda}(\Phi_r, 0)$, and every segment $\mathbf{\Lambda}(\Phi, \Psi_k)$ into an interval $[\alpha_k, \beta_k]$ with $\alpha_k < \beta_k$. For distinct points $z_1, z_2, z_3 \in \mathbb{C} \cup \{\infty\}$, the Möbius transform

$$\mu(\lambda) = \frac{(\lambda - z_1)(z_2 - z_3)}{(\lambda - z_3)(z_2 - z_1)} \tag{O.25}$$

maps $\{z_1, z_2, z_3\}$ onto $\{0, 1, \infty\}$ and the circle or line through z_1, z_2, z_3 onto the real axis. With any distinct set of points $z_1, z_2, z_3 \subset \mathbf{\Lambda}(\Phi, 0)$, $z_3 \notin \bigcup_{k=1}^{l} \mathbf{\Lambda}(\Phi, \Psi_k)$, this transformation satisfies the requirements. Let $\hat{\Phi}, \hat{\Psi} \in \mathbb{H}^{l+1}$ and $\hat{R} \in \mathbb{H}^l$ be the result of applying the Lemma [63, Lemma 3.1] to the image of $\bigcup_{k=1}^{l} \mathbf{\Lambda}(\Phi, \Psi_k)$ under the Möbius transformation. In addition, set

$$M = \begin{bmatrix} z_2 - z_3 & -z_1(z_2 - z_3) \\ z_2 - z_1 & -z_3(z_2 - z_1) \end{bmatrix} = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix}$$

and for $k \in \mathbb{N}$ define the matrix $\boldsymbol{M}_k \in \mathbb{R}^{(k+1)\times(k+1)}$ as:

$$\boldsymbol{M}_k = \begin{bmatrix} \operatorname{conv}^k(M_1) \\ \operatorname{conv}\left(\operatorname{conv}^{k-1}(M_1), M_2\right) \\ \vdots \\ \operatorname{conv}^k(M_2) \end{bmatrix}$$

Then $\Phi, \Psi \in \mathbb{H}^{l+1}$ and $R \in \mathbb{H}^l$ defined by

$$\begin{aligned} \Phi &= \boldsymbol{M}_l^* \hat{\Phi} \boldsymbol{M}_l \\ \Psi &= \boldsymbol{M}_l^* \hat{\Psi} \boldsymbol{M}_l \\ R &= \boldsymbol{M}_{l-1}^* \hat{R} \boldsymbol{M}_{l-1} \end{aligned}$$

satisfy $\mathbf{\Lambda}(\Phi, \Psi) = \bigcup_{k=1}^{l} \mathbf{\Lambda}(\Phi, \Psi_k)$ and $\Phi = J_l^* \left((M^* \Phi_r M) \otimes R\right) J_l$.

***Pipeleers et al.* Example 1 : Union of two continuous time frequency intervals** *Pipeleers*, *Iwasaki* and *Hara* [63, Example 1] show an example of the union of two continuous time frequency intervals on the imaginary axis:

$$\mathbf{\Lambda}(\Phi, \Psi) = \{\lambda = \imath\omega \ : \ \omega \in [\alpha_1, \beta_1] \cup [\alpha_2, \beta_2]\}$$

with $-\infty < \alpha_1 < \beta_1 < \alpha_2 < \beta_2 < \infty$. A Möbius transformation $\mu(\lambda)$ that maps the imaginary axis to the real axis is given by:

$$\mu(\lambda) = -\imath\lambda \quad \rightarrow \quad M = \begin{bmatrix} -\imath & 0 \\ 0 & 1 \end{bmatrix}$$

which is a Möbius transformation with $z_1 = 0, z_2 = \imath, z_3 = \infty$. As stated above:

$$\begin{aligned} \tilde{T} &= \begin{bmatrix} -1 & \imath\alpha_1 + \imath\alpha_2 & \alpha_1\alpha_2 \\ 1 & -\imath\beta_1 - \imath\beta_2 & -\beta_1\beta_2 \end{bmatrix} \\ \Phi &= \tilde{T}^* \Phi_r \tilde{T} \\ \Psi &= \tilde{T}^* \Psi_r \tilde{T} \\ \hat{R} &= \begin{bmatrix} \beta_1 + \beta_2 - \alpha_1 - \alpha_2 & \alpha_1\alpha_2 - \beta_1\beta_2 \\ \alpha_1\alpha_2 - \beta_1\beta_2 & \beta_1\beta_2(\alpha_1 + \alpha_2) - \alpha_1\alpha_2(\beta_1 + \beta_2) \end{bmatrix} \\ R &= M^* \hat{R} M \end{aligned}$$

***Pipeleers et al.* Example 2 : Union of two discrete time frequency intervals** *Pipeleers*, *Iwasaki* and *Hara* [63, Example 2] show an example of the union of two discrete time frequency intervals on the unit circle (see Equation O.14):

$$\mathbf{\Lambda}(\Phi, \Psi) = \left\{\lambda = e^{\imath\theta} \ : \ \theta \in [\eta_1, \zeta_1] \cup [\eta_2, \zeta_2]\right\}$$

with $-\pi < \eta_1 < \zeta_1 < \eta_2 < \zeta_2 < \pi$. A Möbius transform that maps the unit circle to the real axis is:

$$\mu(\lambda) = -\imath\frac{\lambda - 1}{\lambda + 1} \quad \rightarrow \quad M = \begin{bmatrix} -\imath & \imath \\ 1 & 1 \end{bmatrix}$$

with $z_1 = 1, z_2 = \imath, z_3 = -1$ and where the two discrete time frequency intervals are mapped into the real axis intervals $\left[\alpha_k = \mu\left(e^{\imath\eta_k}\right), \beta_k = \mu\left(e^{\imath\zeta_k}\right)\right]$. The corresponding Φ and Ψ matrixes are:

$$\Phi = \tilde{T}^* \begin{bmatrix} 0 & \imath c \\ -\imath\bar{c} & 0 \end{bmatrix} \tilde{T}$$
$$\Psi = \tilde{T}^* \begin{bmatrix} 0 & c \\ \bar{c} & 0 \end{bmatrix} \tilde{T}$$

where:

$$c = (1+\imath\alpha_1)(1+\imath\alpha_2)(1-\imath\beta_1)(1-\imath\beta_2)$$
$$\tilde{T} = \begin{bmatrix} -1 & e^{\imath\eta_1} + e^{\imath\eta_2} & -e^{\imath\eta_1}e^{\imath\eta_2} \\ 1 & -e^{\imath\zeta_1} - e^{\imath\zeta_2} & e^{\imath\zeta_1}e^{\imath\zeta_2} \end{bmatrix}$$

The third part of the Octave script *kyp_complex_curve_union_test.m* shows the mapping of a union of segments of the unit circle to a union of segments of the real axis $\lambda \in [\alpha_k, \beta_k]$. The angles of the roots of

$$\begin{bmatrix} 1 & 0 \end{bmatrix} T - s \begin{bmatrix} 0 & 1 \end{bmatrix} T = 0$$

are plotted against $\Im s$. Recall that we are first mapping the imaginary axis to the real axis so that $T = T_r\tilde{T}$. In each case the magnitude of the roots is 1. Figure O.9 shows the plot of the root locus when the intervals are represented by:

```
eta = [  0.10,  0.35 ];

zeta = [  0.15,  0.45 ];
```

Figure O.9: Test of Assumption 1 for an interval on the unit circle mapped to a union of intervals on the real axis.

The fourth part of the Octave script *kyp_complex_curve_union_test.m* shows the mapping of a union of segments of the unit circle to a union of segments of the real axis $\lambda \in [\alpha_k, \beta_k]$ when the upper limit of one segment on the real axis is $\beta_l = \infty$. In this case I use $\hat{\Phi}$, $\hat{\Psi}$ and $\hat{R}$. The angles of the roots of

$$\begin{bmatrix} 1 & 0 \end{bmatrix} T - s \begin{bmatrix} 0 & 1 \end{bmatrix} T = 0$$

are plotted against $\Im s$. In each case the magnitude of the roots is 1. Figure O.10 shows the plot of the root locus when the intervals are represented by:

```
eta = [  0.10,  1.00 ];

zeta = [  0.15,  3.14 ];
```

Figure O.10: Test of Assumption 1 for an interval on the unit circle mapped to a union of intervals on the real axis when $\beta_l = \infty$.

O.3.2 Generalised KYP lemma over a union of frequency intervals

Pipeleers, *Iwasaki* and *Hara* [63] show the following preliminary definitions. For $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$ and $l \in \mathbb{N}$, 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 $G_l(A,B) \in \mathbb{C}^{(n+m)\times(n+ml)}$[n]:

$$G_l(A,B) = \left(I_l \otimes \begin{bmatrix} I_n \\ 0_{m,n} \end{bmatrix}\right) \begin{bmatrix} F_{l-1}(A,B) & 0_{nl,m} \end{bmatrix} + \left(I_l \otimes \begin{bmatrix} 0_{n,m} \\ I_m \end{bmatrix}\right) \begin{bmatrix} 0_{ml,n} & F_{l-1}(0_{m,m}, I_m) \end{bmatrix}$$

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:

$$\mathcal{N}_{A,B}(\lambda) = \begin{cases} (x,u) \in \mathbb{C}^n \times \mathbb{C}^m \ : \ (\lambda I - A)\, x = Bu & \text{for } \lambda \neq \infty \\ \{0\} \times \mathbb{C}^m & \text{for } \lambda = \infty \end{cases}$$

[n] *Pipeleers et al.* [63, Equation 2.8] show the last term as $F_l(0_{m,m}, I_m)$.

Note that 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 $u \in \mathbb{C}^m$.

Firstly, *Pipeleers et al.* prove the following two Lemmas:

Lemma 2.3 [63, Lemma 2.3 and Appendix A]: Let matrixes $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$ and $T \in \mathbb{C}^{2\times(l+1)}$, be given, and assume B has full column rank and T full row rank. In addition, let $s \in \mathbb{C} \cup \{\infty\}$ be given and assume that the l roots λ_k, $k \in \mathbb{N}_l$, in $\mathcal{R}_T(s)$ are all distinct. Then a vector $z \in \mathbb{C}^{n+ml}$ satisfies

$$\begin{aligned} ([1 \ \ -s] \otimes I_n)(T \otimes I_n) F_l z = 0 \quad &\text{for } s \neq \infty \\ ([0 \ \ \ 1] \otimes I_n)(T \otimes I_n) F_l z = 0 \quad &\text{for } s = \infty \end{aligned} \tag{O.26}$$

if-and-only-if it can be decomposed as

$$z = \sum_{k=1}^{l} \begin{bmatrix} x_k \\ L_{l-1}(\lambda_k)^{\mathsf{F}} \otimes u_k \end{bmatrix}$$

with $(x_k, u_k) \in \mathcal{N}_{A,B}(\lambda_k)$ for all $k \in \mathbb{N}_l$.

The proof of Lemma 2.3 proceeds by elaborating each $F_l z_k$ recursively from the bottom row to the top row and recalling the definition of $L_l(\infty)$:

$$F_l \begin{bmatrix} x_k \\ L_{l-1}(\lambda_k) \otimes u_k \end{bmatrix} = L_l(\lambda_k) \otimes \chi(x_k, u_k)$$

where

$$\chi(x_k, u_k) = \begin{cases} x_k & \text{if } \lambda \neq \infty \\ B u_k & \text{if } \lambda = \infty \end{cases}$$

As λ_k are the l roots of Equation O.22, each z_k satisfies Equation O.26 and so does $z = \sum_{k=1}^{l} z_k$.

Lemma 2.4 [63, Lemma 2.4 and Appendix B]: Let $\Phi_o, \Psi_o \in \mathbb{H}^2$ of the form shown in Equation O.21 be given, as well as $X, Y \in \mathbb{C}^{n\times m}$. Then

$$\begin{aligned} \begin{bmatrix} X & Y \end{bmatrix} (\Phi_o \otimes I_m) \begin{bmatrix} X^* \\ Y^* \end{bmatrix} &= 0 \\ \begin{bmatrix} X & Y \end{bmatrix} (\Psi_o \otimes I_m) \begin{bmatrix} X^* \\ Y^* \end{bmatrix} &\succeq 0 \end{aligned} \tag{O.27}$$

hold if-and-only-if X and Y can be factored as

$$\begin{aligned} X &= W \operatorname{diag}(s_1, \ldots, s_m) V^* \\ Y &= W V^* \end{aligned} \tag{O.28}$$

with some $W \in \mathbb{C}^{n\times m}$, unitary $V \in \mathbb{C}^{m\times m}$ and $s_k \in \Lambda(\Phi_o, \Psi_o)$ for all $k \in \mathbb{N}_m$.

Pipeleers et al. [63, Appendix B] show a construction for W and V. The equality is equivalent to

$$\begin{aligned} XY^* + YX^* &= -XY^* - YX^* \\ (X+Y)(X+Y)^* &= (X-Y)(X-Y)^* \end{aligned}$$

so that $X + Y$ and $X - Y$ have the same left singular vectors and singular values

$$\begin{aligned} X + Y &= P \Sigma Q_1^* \\ X - Y &= P \Sigma Q_2^* \end{aligned}$$

where $P \in \mathbb{C}^{n\times n}$ is unitary, $Q_1, Q_2 \in \mathbb{C}^{m\times m}$ are unitary and $\Sigma \in \mathbb{R}^{n\times m}$ is diagonal. The matrix $Q_1^* Q_2$ is unitary and can be factorised as $V \operatorname{diag}(\sigma_k) V^*$ with $|\sigma_k| = 1$ for $k = 1, \ldots, m$. Define $W = YV$ and $s_k = (1+\sigma_k)/(1-\sigma_k)$. If $\sigma_k = e^{\imath\omega_k}$ then the $s_k = \imath \operatorname{cotan} \frac{\omega_k}{2}$ result from a mapping of the unit circle to the imaginary axis in the complex plane. Expanding Equation O.27 and substituting Equation O.28[o]

$$XY^* + YX^* = W \operatorname{diag}(s_k) V^* V W^* + W V^* V \operatorname{diag}(s_k)^* V^* W^* = \operatorname{diag}(s_k + s_k^*) = 0$$
$$\alpha X X^* + \gamma Y Y^* = \alpha W \operatorname{diag}(s_k) V^* V \operatorname{diag}(s_k)^* W^* + \gamma W V^* V W^* = \alpha \operatorname{diag}\left(|s_k|^2\right) + \gamma \succeq 0$$

Pipeleers et al. now prove the following generalised KYP lemma for non-strict inequalities[p]:

Theorem [63, Theorem 2.2, Appendix C]: 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 Φ and Ψ 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:
$$\begin{bmatrix} x \\ u \end{bmatrix}^* \Theta \begin{bmatrix} x \\ u \end{bmatrix} \leq 0$$
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
$$F_l(A,B)^* (\Phi \otimes P + \Psi \otimes Q) F_l(A,B) + G_l(A,B)^* (R \otimes \Theta) G_l(A,B) \preceq 0$$

The terms in $\begin{bmatrix} C & D \end{bmatrix}$ are linearised by applying the Schur complement to $R \otimes \Theta$:

$$\begin{aligned} R \otimes \Theta &= R \otimes \left(\begin{bmatrix} C & D \\ 0 & I \end{bmatrix}^* \begin{bmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{12}^* & \Pi_{22} \end{bmatrix} \begin{bmatrix} C & D \\ 0 & I \end{bmatrix} \right) \\ &= R \otimes \begin{bmatrix} 0 & C^* \Pi_{12} \\ \Pi_{12}^* C & D^* \Pi_{12} + \Pi_{12}^* D + \Pi_{22} \end{bmatrix} + R \otimes \left(\begin{bmatrix} C & D \end{bmatrix}^* \Pi_{11} \begin{bmatrix} C & D \end{bmatrix} \right) \\ &= R \otimes \begin{bmatrix} 0 & C^* \Pi_{12} \\ \Pi_{12}^* C & D^* \Pi_{12} + \Pi_{12}^* D + \Pi_{22} \end{bmatrix} + \left(I_2 \otimes \begin{bmatrix} C & D \end{bmatrix} \right)^* (R \otimes \Pi_{11}) \left(I_2 \otimes \begin{bmatrix} C & D \end{bmatrix} \right) \end{aligned}$$

where I have made repeated use of the mixed product rule, $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$. If $R \otimes \Pi_{11} \succeq 0$ then the Schur complement includes $-(R \otimes \Pi_{11})^{-1}$.

O.3.3 Examples of FIR filter design with the generalised KYP lemma extended to the union of disjoint frequency bands

Design of a symmetric band pass FIR filter with the union of the upper and lower stop bands

The Octave script *directFIRsymmetric_kyp_union_bandpass_test.m* uses *Pipeleer et al.*'s generalised KYP lemma to design a band pass FIR filter with a specification similar to that of their example [63, Figure 4]. The stop band LMI is expressed as the union of the upper and lower stop band frequency intervals:

$$[\eta_1, \zeta_1] \cup [\eta_2, \zeta_2] = 2\pi [-f_{asl}, f_{asl}] \cup 2\pi [f_{asu}, 1 - f_{asu}]$$

The Möbius transformation from the unit circle to the real axis is chosen as $z_1 = 1, z_2 = \imath, z_3 = -\imath$. In the pass band the filter response is compared to a nominal delay. In the stop band $|H(\omega)|^2 \leq \varepsilon_s^2$.

The filter specification is:

[o]For $s_k \in \Lambda(\Phi_o, \Phi_o)$ defined in Equation O.10 and Φ_o and Ψ_o defined in Equation O.21

$$\begin{bmatrix} s_k^* & 1 \end{bmatrix} \Phi_o \begin{bmatrix} s_k \\ 1 \end{bmatrix} = s_k^* + s_k = 0$$
$$\begin{bmatrix} s_k^* & 1 \end{bmatrix} \Psi_o \begin{bmatrix} s_k \\ 1 \end{bmatrix} = \alpha s_k s_k^* + \beta s_k + \beta s_k^* + \gamma = \alpha |s_k|^2 + \gamma \geq 0$$

[p]The Octave script *yalmip_kyp_check_iir_bandpass_test.m* uses the generalised KYP lemma to check the response of a parallel all-pass one-multiplier Schur lattice filter designed by the Octave script *schurOneMPAlattice_socp_slb_bandpass_delay_test.m*. The filter is intended to have a pass-band phase that is an integer multiple of π plus the nominal pass-band phase shift so that the pass-band response may be compared to a delay of an integral number of samples.

```
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_max=1.005 % Maximum squared amplitude
Esq_z=6.97225e-07 % Squared amplitude pass band - delay error
Esq_s=2.5e-05 % Squared amplitude stop band error
```

The value of ε_z^2 was found by trial-and-error. The actual maximum squared-error in the pass-band compared with a pure delay is $\varepsilon_z^2 = 5.09462\text{e-}07$. I did not achieve the pass-band error of $\varepsilon_z = 2.5\text{e-}4$ specified by *Pipeleers et al.*. The resulting FIR impulse response is:

```
h = [ -0.0034839695, -0.0003592293, -0.0043641593, -0.0112517525, ...
       0.0043018203,  0.0105170057, -0.0028407625,  0.0292538842, ...
       0.0492568177, -0.0166214335, -0.0188211131,  0.0207530855, ...
      -0.1441413531, -0.2431499798,  0.1187484767,  0.4225485257, ...
       0.1187484767, -0.2431499798, -0.1441413531,  0.0207530855, ...
      -0.0188211131, -0.0166214335,  0.0492568177,  0.0292538842, ...
      -0.0028407625,  0.0105170057,  0.0043018203, -0.0112517525, ...
      -0.0043641593, -0.0003592293, -0.0034839695 ];
```

Figure O.11 shows the stop band and pass band amplitude responses.

Figure O.11: Stop band and pass band amplitude responses of a symmetric FIR band pass filter designed with the generalised KYP lemma over a union of stop band regions.

Design of a non-symmetric band pass FIR filter with the union of the upper and lower stop bands

The Octave script *directFIRnonsymmetric_kyp_union_bandpass_test.m* uses *Pipeleer et al.*'s generalised KYP lemma to design a non-symmetric band pass FIR filter. The Möbius transformation from the unit circle to the real axis is chosen as $z_1 = 1, z_2 = \imath, z_3 = -\imath$ (see Equation O.25). In the pass band the filter response is compared to a nominal delay. In the stop band $|H(\omega)|^2 \leq \varepsilon_s^2$. The filter specification is:

```
N=40 % FIR filter order
d=16 % 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_max=1.05 % Maximum squared amplitude
Esq_z=1e-06 % Squared amplitude pass band - delay error
Esq_s=0.0001 % Squared amplitude stop band error
```

The resulting FIR impulse response is:

```
h = [  0.0013632645,  0.0022335917, -0.0002767357,  0.0046002267, ...
       0.0090332904, -0.0062651251, -0.0210712899, -0.0048452416, ...
       0.0091097793, -0.0074584534,  0.0115576550,  0.0775167609, ...
       0.0372321044, -0.1393050276, -0.1730310888,  0.0786134723, ...
       0.2684974324,  0.0824048925, -0.2041650324, -0.1762236110, ...
       0.0520761361,  0.1176136604,  0.0212375028, -0.0136525957, ...
       0.0139318625, -0.0134420220, -0.0501748668, -0.0147547375, ...
       0.0308529904,  0.0198922501, -0.0024833509,  0.0019097685, ...
       0.0023300276, -0.0108088687, -0.0097969336,  0.0032323910, ...
       0.0055850420,  0.0005626893,  0.0007256611,  0.0018957491, ...
      -0.0012853422 ];
```

The value of ε_z^2 was found by trial-and-error. The actual maximum squared-error in the pass-band compared with a pure delay is $\varepsilon_z^2 = 0.00000074$.

Figure O.12 shows the amplitude, phase and delay responses. The pass band phase error is adjusted for the nominal delay. Figure O.13 shows the zeros of the transfer function.

Figure O.12: 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.

Figure O.13: Zeros of a non-symmetric FIR band pass filter designed with the generalised KYP lemma over a union of stop band regions.

Design of a non-symmetric band pass FIR filter with the union of multiple pass bands and stop bands

The Octave script *directFIRnonsymmetric_kyp_union_double_bandpass_test.m* uses *Pipeleer et al.*'s generalised KYP lemma to design a non-symmetric band pass FIR filter with the union of two pass bands and the union of three stop bands. The filter amplitude response is similar to that of the symmetric FIR filter shown in Figure N.47.

In the pass band the filter response is compared to a nominal delay. In the stop band $|H(\omega)|^2 \leq \varepsilon_s^2$. The filter specification is:

```
eps=1e-08 % SeDuMi eps
N=48 % FIR filter order
d=20 % Nominal FIR filter delay
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
Esq_z=5e-05 % Squared amplitude pass band - delay error
Esq_s=8e-05 % Squared amplitude stop band error
```

In the pass bands, the Möbius transformation from the unit circle to the real axis is chosen as $z_1 = 1, z_2 = \imath, z_3 = -\imath$. In the stop bands each of these zs is rotated by $\pi\left(f_{sl2} - f_{pu1}\right)$ radians so that z_3 is not in a stop band. The resulting FIR impulse response is:

```
h = [  0.0018088129, -0.0005644017, -0.0042202266, -0.0019642761, ...
      -0.0142528443,  0.0196273600,  0.0148855944, -0.0076759554, ...
       0.0031585995, -0.0139680485,  0.0003407463, -0.0263455542, ...
       0.0066789470, -0.0281535959,  0.0743812868,  0.1474408094, ...
      -0.1690011610, -0.0418364085, -0.1062561199, -0.0479753724, ...
       0.3961296066, -0.0506779858, -0.1101679246, -0.0514657403, ...
```

```
-0.2107403021,  0.1924977961,  0.1038472941, -0.0474329605, ...
 0.0083495607, -0.0345404905, -0.0006140319, -0.0230581790, ...
 0.0046233406, -0.0210907900,  0.0354595837,  0.0497068393, ...
-0.0406462473, -0.0079624307, -0.0043193281, -0.0019158787, ...
 0.0011412831,  0.0010375580,  0.0027916478,  0.0029770507, ...
 0.0123698291, -0.0121434706, -0.0072648536,  0.0046576928, ...
-0.0002034042 ];
```

Figure O.14 shows the amplitude, phase and group delay responses. The pass band phase error is adjusted for the nominal delay. The actual maximum squared-error in the pass-band compared with a pure delay is $\varepsilon_z^2 = 0.00003565$.

Figure O.14: Amplitude, phase and group delay responses of a non-symmetric FIR band pass filter designed with the generalised KYP lemma with multiple pass and stop bands.

SeDuMi fails with a "*Run into numerical problems*" warning with the default *eps=1e-9* so it is increased in the YALMIP options. YALMIP still warns about numerical problems. This warning is removed by setting the YALMIP constraints as :

```
Constraints=[F_plu<=-sedumi_eps,Q_plu>=0,F_slmu<=-sedumi_eps,Q_slmu>=0];
```

The resulting filter frequency response is acceptable. These numerical problems occurred for all combinations of N and d that I tried.

O.4 Design of one-multiplier Schur lattice filters with the KYP lemma

This section considers the use of the KYP lemma to design one-multiplier Schur lattice IIR filters. The matrix inequality part of the KYP lemma shown in Equation O.12 is not linear in the state variable coefficients. The design problem can be approximated by the solution of a system of bilinear matrix inequality (BMI) constraints if the state-variable coefficients can be expressed as linear combinations of the Schur lattice filter coefficients. This is the case for the $R = 2$, tapped, one-multiplier Schur lattice filter, described in Section 5.8.2, the doubly-pipelined, tapped, one-multiplier Schur lattice filter, described in Section 5.6.4, and the parallel combination of two doubly-pipelined all-pass Schur lattice filters. The design of robust feedback control systems has motivated much research into the solution of BMI constraints derived from the KYP lemma. See, for example, *Van Antwerp* and *Braatz* [95] or *Dinh et al.* [194, Section 1]. The solution of an optimisation problem with BMI constraints can be obtained as the limit of a sequence of upper bounding convex (or LMI) problems. *Duffin* and *Peterson* [199, p.533] describe the solution of a "geometric program" by a sequence of "upper-bound inequality posynomial" constraints. *Marks* and *Wright* [28] demonstrated the convergence of a "sequence of approximating convex programs". *Dinh et al.* [194, 49] "decompose the bilinear mapping as a difference between two positive semidefinite convex mappings. At each iteration of the algorithm the concave part is linearized, leading to a convex subproblem". *Lee* and *Hu* [135], *Warner* and *Scruggs* [50], *Sebe* [160] and *Ren et al.* [208] each show an alternative method of convex upper approximation to the BMI constraint.

O.4.1 Preliminaries

Positive-semi-definite convex mappings

Suppose the mapping $\mathcal{A}\colon \mathbb{R}^n \to \mathbb{S}^p$ is represented by $A(x) \in \mathbb{S}^p$, the set of symmetric $p \times p$ matrixes. $A(x)$ is said to be *psd-convex* on a convex set, $\mathcal{C} \subseteq \mathbb{R}^n$, if, for $t \in [0, 1]$ and $x, y \in \mathcal{C}$:

$$A(tx + [1-t]\,y) \preceq tA(x) + [1-t]\,A(y)$$

Similarly, $-A(x)$ is *psd-concave*. The first-order linearised approximation at $x = x_i + \Delta_x$ of a psd-convex mapping $\mathcal{A}$ represented by $A(x)$ satisfies [215, Section 3.1.3]:

$$A(x_i) + \nabla A(x_i)\,\Delta_x \preceq A(x)\,, \text{ for all } x \in \mathcal{C}$$

Schur complement of a psd-convex mapping

Dinh et al. [194, Section 3.1] define a *Schur psd-convex* mapping as $\mathcal{F}\colon \mathbb{R}^{p\times q} \times \mathbb{S}^p \to \mathbb{S}^p$ given by $F(X, Y) = XQ^{-1}X^\top - Y$, where $Q \in \mathbb{S}^p_{++}$, the set of symmetric, positive definite, $p \times p$ matrixes. They show the following lemma, used to transform a Schur psd-convex constraint into an LMI constraint [194, Lemma 3.2]:

Lemma 3.2 [194]:

1. Suppose that $A \in \mathcal{S}^n$. Then:

$$BB^\top - A \preceq 0 \quad \Longleftrightarrow \quad \begin{bmatrix} A & B \\ B^\top & I \end{bmatrix} \succeq 0 \tag{O.29}$$

2. Suppose that $A \in \mathbb{S}^n$ and $D \succeq 0$ then:

$$\begin{bmatrix} A - BB^\top & C \\ C^\top & D \end{bmatrix} \succeq 0 \quad \Longleftrightarrow \quad \begin{bmatrix} A & B & C \\ B^\top & I & 0 \\ C^\top & 0 & D \end{bmatrix} \succeq 0 \tag{O.30}$$

Problem statement

This section considers the following optimisation problem with bilinear matrix inequality constraints [135, Equation 1]:

$$\begin{aligned} \textbf{minimise} \quad & f(z) \\ \textbf{subject to} \quad & z \in \mathcal{C} \\ & F(z) = C + D(z) + \mathrm{He}(A(x)B(y)) \preceq 0 \end{aligned}$$

where $z^\top = \left[x^\top, y^\top\right] \in \mathbb{R}^n$, $\mathrm{He}(X) = X + X^\top$, $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex, $\mathcal{C} \subset \mathbb{R}^n$ is convex and closed, a feasible initial point, $x_0 \in \mathcal{C}$, is known, C is a constant matrix and D, A and B are matrixes[q] that are linear in the coefficients, x, y and z and $F(z)$ represents Equation O.18.

At the m'th iteration, *Lee* and *Hu* [135, Equation 4] partially linearise $F(z)$ at $z = z_m + \Delta_z$ where $z_m^\top = \left[x_m^\top, y_m^\top\right]$ as follows:

$$F(z) \approx F(z_m) + \nabla F(z_m)\Delta_z + \mathrm{He}(\Delta_A \Delta_B) \preceq 0 \tag{O.31}$$

where, for convenience, $\Delta_A = \nabla A(x_m)\Delta_x$. Equation O.31 retains the bilinear terms in Δ_z. The linear part of Equation O.31 is:

$$\nabla F(z_m)\Delta_z \approx \Delta_D + \mathrm{He}(A(x_m)\Delta_B) + \mathrm{He}(\Delta_A B(y_m))$$

O.4.2 Sequential approximation of BMI constraints

This section describes methods of finding a sequence of convex upper approximations to the bilinear terms at the z_m of each iteration of Equation O.31.

***Dinh et al.* psd-convex-concave optimisation with BMI constraints**

Dinh et al. [194, Lemma 3.1] decompose the bilinear form $X^\top Y + Y^\top X$ into a difference of convex and concave mappings[191]:

Lemma 3.1 [194]:

1. The mappings $f(X) := X^\top X$ and $g(X) := XX^\top$ are psd-convex on $\mathbb{R}^{m\times n}$. The mapping $f(X) := X^{-1}$ is psd-convex on $\mathbb{S}^p_{++}$.
2. The bilinear matrix form $X^\top Y + Y^\top X$ can be represented as a psd-convex-concave mapping in at least three forms:

$$X^\top Y + Y^\top X = (X+Y)^\top (X+Y) - \left(X^\top X + Y^\top Y\right) \tag{O.32a}$$
$$= \left(X^\top X + Y^\top Y\right) - (X-Y)^\top (X-Y) \tag{O.32b}$$
$$= \frac{1}{2}\left[(X+Y)^\top (X+Y) - (X-Y)^\top (X-Y)\right] \tag{O.32c}$$

Dinh et al. [194, Sections 3.2 and 4] consider the following convex optimisation problem:

$$\begin{aligned} \textbf{minimise} \quad & f(x) \\ \textbf{subject to} \quad & x \in \mathcal{C} \\ & G(x) - H(x) \preceq 0 \end{aligned}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex, $\mathcal{C} \subseteq \mathbb{R}^n$ is a non-empty, closed, convex set and $G(x)$ and $H(x)$ are psd-convex.

Dinh et al. linearise the optimisation problem as follows:

$$\begin{aligned} \textbf{minimise} \quad & f_m(x) := f(x) + \rho_m \|Q_m \Delta_x\|_2^2 \\ \textbf{subject to} \quad & x \in \mathcal{C} \\ & G(x) - H(x_m) - \nabla H(x_m)\Delta_x \preceq 0 \end{aligned} \tag{O.33}$$

[q] Here A, B, C and D are **not** filter state-variable matrixes.

where $Q_m \succ 0$. The linearised concave part is an upper approximation:

$$-H(x) \preceq -H(x_m) - \nabla H(x_m)\Delta_x, \text{ for all } x \in \mathcal{C}$$

and the convex part is linearised by applying Equation O.29.

Given a feasible initial point, x_0, and initial $\rho_0 > 0$ and $Q_0 \in \mathbb{S}^n_+$, *Dinh et al.* [194, Algorithm 1] suggest repeatedly solving Equation O.33 for x_{i+1} until a termination condition is met. If necessary, ρ_m and Q_m are updated at each step. *Dinh et al.* [194, Section 5] show examples in which they apply the Schur complement to the psd-convex-linearised-concave decomposition of Equation O.32c.

Lee and Hu sequential convex upper-approximation with BMI constraints

Lee and *Hu* [135, Lemma 2] prove the following lemma[r]:

> *Lemma 2* [135]: Let D and E be real matrixes of appropriate dimensions. Then, for any $S \in \mathbb{S}^n_{++}$:
>
> $$DE + E^\top D^\top \preceq DSD^\top + E^\top S^{-1} E \tag{O.34}$$
>
> *Proof:* Expand $\left(D^\top - S^{-1}E\right)^\top S\left(D^\top - S^{-1}E\right) \succeq 0$.

Lee and *Hu* [135, Equation 5] apply this lemma to the partially linearised bilinear mapping of Equation O.31[s]:

$$F(z) \preceq F(z_m) + \nabla F(z_m)\Delta_z + \Delta_A(\Delta_x) S \Delta_A(\Delta_x)^\top + \Delta_B(\Delta_y)^\top S^{-1}\Delta_B(\Delta_y) \preceq 0$$

and, applying the Schur complement as shown in Equation O.30, obtain the following symmetric linear matrix inequality:

$$\begin{bmatrix} F(z_m) + \nabla F(z_m)\Delta_z & * & * \\ \Delta_A(\Delta_z)^\top & -S^{-1} & * \\ \Delta_B(\Delta_z) & 0 & -S \end{bmatrix} \preceq 0$$

where I omit the symmetric components. Further, *Dinh et al.* [194, Lemma 3.1] show that S^{-1} is psd-convex and, consequently, *Lee* and *Hu* [135, Lemma 3] prove the following lemma:

> *Lemma 3* [135]: Suppose that $\mathbb{S}\colon \mathbb{R}^n \to \mathbb{S}^p$ is a linear mapping defined as $S(x) = \sum_{l=1}^n x_l S_l$, where $S_l \in \mathbb{S}^p$ is symmetric and real. If $S(x) \succ 0$ and $S(y) \succ 0$, then $-S(y)^{-1} \preceq -2S(x)^{-1} + S(x)^{-1} S(y) S(x)^{-1}$.
>
> *Proof:* Linearise $-S(y)^{-1}$ around x and apply the matrix identities $SS^{-1} = I$ and $\frac{dS}{dx}S^{-1} + S\frac{dS^{-1}}{dx} = 0$.

After replacing the psd-concave mapping, $-S^{-1}$, with its upper approximation linearised around S_m and scaling by the congruence transformation $\operatorname{diag}[I, S_m, I]$, the upper approximation to $F(z)$ near $z = z_m$ is the LMI constraint [135, Equation 8]:

$$\begin{bmatrix} F(z_m) + \nabla F(z_m)\Delta_z & * & * \\ S_m\Delta_A(\Delta_z)^\top & -2S_m + S & * \\ \Delta_B(\Delta_z) & 0 & -S \end{bmatrix} \preceq 0 \tag{O.35}$$

Lee and *Hu* [135, Algorithm 1] propose a sequential approximation algorithm with the "regularised" objective function $f(z) + \frac{\rho}{2}\|\Delta_z\|^2$ and the LMI constraint of Equation O.35 and $S_0 = I_n$, $c_1 I \preceq S \preceq c_2 I$, $-2S_m + S \preceq -c_3 I$ with $\rho > 0$, $c_2 > c_1 > 0$, $c_3 > 0$. *Lee* and *Hu* [135, Section 5] provide an example with values for ρ, c_1, c_2 and c_3.

Lee and *Hu* [135, Remark 2] compare their method to the difference of convex functions (DC) method of *Dinh et al.*:

[r] See [215, Section 3.6].

[s] This is similar to overbounding the constraint by ignoring the second term in Equation O.32b.

In fact, a more general psd-convex-concave mapping can be derived by introducing the auxiliary matrix, S, as follows: $X^\top Y + Y^\top X = X^\top S X + Y^\top S^{-1} Y - \left(X - S^{-1}Y\right)^\top S \left(X - S^{-1}Y\right)$. Note that the last term $-\left(X - S^{-1}Y\right)^\top S \left(X - S^{-1}Y\right)$ is concave in X and Y. If we set S to be a constant, i.e. $S = I$, and linearize the last term $\left(X - S^{-1}Y\right)^\top S \left(X - S^{-1}Y\right)$ with respect to (X, Y) at the point $(X, Y) = (X_m, Y_m)$, then the over approximation of the DC programming method is obtained. Instead, if we drop the last term and linearize S^{-1} at $S = S_m$, then the over approximation of the proposed Algorithm 1 is obtained. From the interpretation it is not easy to claim which approximation is better than the other. Since the last term is entirely dropped in Algorithm 1, it can be seen as a less accurate one in general. However, the auxiliary matrix S can be adjusted as a decision variable of the convex subproblem, the over approximation can be tightened at each iteration.

Warner and Scruggs **iterative convex over-bounding optimisation with BMI constraints**

Warner and *Scruggs* [50] describe iterative convex over-bounding techniques for BMI problems like:

$$Q + \mathrm{He}\left(BRDSC\right) \prec 0$$

where Q, B, C and D are constants and R and S are the design variables with known feasible intial values R_0 and S_0. At the m'th iteration, with perturbed values $R = R_m + \Delta_R$ and $S = S_m + \Delta_S$:

$$Q + \mathrm{He}\left(\phi\left(R, S\right) + B\Delta_R D \Delta_S C\right) \prec 0$$

where:

$$\phi\left(R, S\right) = B\left[RDS_m + R_m DS - R_m DS_m\right]C$$

Warner and *Scruggs* now set $D = UV$ where U and $V^\top$ have full column rank and for an arbitrary invertible matrix L_1:

$$Q + \mathrm{He}\left(\phi\left(R, S\right)\right) + B\Delta_R U L_1 L_1^\top U^\top \Delta_R^\top B^\top + C^\top \Delta_S^\top V^\top L_1^{-\top} L_1^{-1} V \Delta_S C \prec \eta\eta^\top$$

where:

$$\eta = B\Delta_R U L_1 - C^\top \Delta_S^\top V^\top L_1^{-\top}$$

Since $\eta\eta^\top \geq 0$ an over-approximation is:

$$Q + \mathrm{He}\left(\phi\left(R, S\right)\right) + B\Delta_R U L_1 L_1^\top U^\top \Delta_R^\top B^\top + C^\top \Delta_S^\top V^\top L_1^{-\top} L_1^{-1} V \Delta_S C \prec 0$$

Defining $W_1 = L_1 L_1^\top$ and applying the Schur complement:

$$\begin{bmatrix} Q + \mathrm{He}\left(\phi\left(R, S\right)\right) & * & * \\ U^\top \left(\Delta_R\right)^\top B^\top & -W_1^{-1} & * \\ V\left(\Delta_S\right)C & 0 & -W_1 \end{bmatrix} \prec 0$$

Alternatively:

$$\mathrm{He}\left(\phi\left(R, S\right)\right) + \alpha\alpha^\top \prec B\Delta_R U L_2 L_2^\top U^\top \Delta_R^\top B^\top + C^\top \Delta_S^\top V^\top L_2^{-\top} L_2^{-1} V \Delta_S C$$

where L_2 is an arbitrary invertible matrix and:

$$\alpha = B\Delta_R U L_2 + \Delta_S C V L_2^{-1}$$

Define $W_2 = L_2 L_2^\top$ and choose L_2 so that $W_2 \succ 0$. Then:

$$\mathrm{He}\left(\phi\left(R, S\right)\right) + \beta W_2^{-1} \beta^\top \prec B\Delta_R U W_2 U^\top \Delta_R^\top B^\top + C^\top \Delta_S^\top V^\top W_2^{-1} V \Delta_S C$$

where:

$$\beta = B\Delta_R U W_2 + C^\top \Delta_S V^\top$$

Since W_2 is positive definite, after applying the Schur complement an upper-approximation is:

$$\begin{bmatrix} \mathrm{He}\left(\phi\left(R, S\right)\right) & * \\ B\left(\Delta_R\right)UW_2 + C^\top\left(\Delta_S\right)^\top V^\top & -W_2 \end{bmatrix} \prec 0$$

Finally, *Warner* and *Scruggs* propose using a linear combination of these two LMI constraints:

$$Q + \mathrm{He}\left(BRU\Lambda VSC\right) + \mathrm{He}\left(BRU\left(I-\Lambda\right)VSC\right) \prec 0$$

where Λ is symmetric interpolation matrix and $0 \prec \Lambda \prec I$. The over-all convex upper-approximation constraint is:

$$\begin{bmatrix} Q + \mathrm{He}\left(\phi\left(R,S\right)\right) & * & * & * \\ U^{\top}\left(\Delta_R\right)^{\top} B^{\top} & -\hat{W}_1^{-1} & * & * \\ V\left(\Delta_S\right)C & 0 & -\hat{W}_1 & * \\ \begin{bmatrix} W_2 B\left(\Delta_R\right)U + \\ C^{\top}\left(\Delta_S\right)^{\top} C^{\top} \end{bmatrix} & 0 & 0 & -\hat{W}_2 \end{bmatrix} \prec 0$$

where $\hat{W}_1 = L_1 \Lambda^{-1} L_1^{\top}$ and $\hat{W}_2 = L_2\left(1-\Lambda\right)^{-1} L_2^{\top}$.

Warner and *Scruggs* suggest:

1. using a "regularised" objective function and LMI constraint similar to that described by *Lee* and *Hu*.
2. updating the weights L_1, L_2 and Λ in separate sub-steps during each iteration of the optimisation.

Sebe sequential convex over-bounding optimisation with BMI constraints

Sebe [160, Section 2.1] argues that the preferred decomposition of $\mathrm{He}\left(XY\right)$[t] is Equation O.32c. *Sebe* [160, Section 2.2] then considers the psd-convex-concave decomposition of bilinear constraints in the form $\mathrm{He}\left(XNY\right) \prec -\mathrm{He}\left(Q\right)$ where N is a constant matrix and $\mathrm{He}\left(Q\right)$ is "the other part of the matrix inequality"[u]. After pointing out that:

$$\mathrm{He}\left(XNY\right) = \begin{bmatrix} X & Y^{\top} \end{bmatrix} \begin{bmatrix} 0 & N \\ N^{\top} & 0 \end{bmatrix} \begin{bmatrix} X^{\top} \\ Y \end{bmatrix}$$

Sebe [160, Proposition 3] shows that:

$$\begin{bmatrix} 0 & N \\ N^{\top} & 0 \end{bmatrix} = \begin{bmatrix} N & N \\ G^{\top} & -G \end{bmatrix} \begin{bmatrix} \left(G+G^{\top}\right)^{-1} & 0 \\ 0 & -\left(G+G^{\top}\right)^{-1} \end{bmatrix} \begin{bmatrix} N & N \\ G^{\top} & -G \end{bmatrix}^{\top}$$

where $G \in \mathbb{R}^{n \times n}$ satisfies $G + G^{\top} \succ 0$. With this result, the constraint $\mathrm{He}\left(Q + XNY\right) \prec 0$ has the upper approximation:

$$\mathrm{He}\left(Q\right) + \left(XN + Y^{\top}G^{\top}\right)\left(G+G^{\top}\right)^{-1}\left(XN + Y^{\top}G^{\top}\right)^{\top} \prec 0$$

Applying the Schur complement:

$$\begin{bmatrix} \mathrm{He}\left(Q\right) & XN + Y^{\top}G^{\top} \\ \left(XN + Y^{\top}G^{\top}\right)^{\top} & -\left(G+G^{\top}\right) \end{bmatrix} \prec 0$$

or, simply:

$$\mathrm{He}\left(\begin{bmatrix} Q & XN \\ GY & -G \end{bmatrix}\right) \prec 0$$

O.4.3 Design of one-multiplier Schur lattice filters with BMI constraints derived from the KYP lemma

In this Section I apply the successive convex optimisation to the design of filters, based on the one-multiplier Schur lattice filter, for which the corresponding state-variable coefficients can be expressed as linear combinations of the lattice coefficients. For example, the state transition matrix, A, of a parallel all-pass, doubly-pipelined, one-multiplier, Schur lattice filter, described in Section 5.6.4, is:

$$A\left(\boldsymbol{k}\right) = A_0 + \sum_{l=1}^{N} k_l A_l$$

[t] *Dinh et al.* write $\mathrm{He}\left(X^{\top}Y\right)$.

[u] The linear part?

where $\boldsymbol{k} = \begin{bmatrix} k_1 & \dots & k_N \end{bmatrix}$ are the Schur lattice filter reflection coefficients, $A \in \mathbb{R}^{n \times n}$ and the $A_0, A_1, \dots$ matrixes and the state-variable B, C and D matrixes are constant[v].

After applying Finsler's transformation to the KYP lemma matrix inequality of Equation O.12, as shown in Section O.2.5, the optimisation problem is:

$$\begin{aligned} \textbf{minimise} \quad & \mathcal{E}^2(\boldsymbol{x}) \\ \textbf{subject to} \quad & -1 < \boldsymbol{k} < 1 \\ & P, Q \in \mathbb{S}^n \text{ and } Q \succeq 0 \\ & \text{Equation O.18} \end{aligned}$$

where $\boldsymbol{x}$ represents the Schur lattice reflection coefficients, $\boldsymbol{k}$, and, for a tapped lattice, the tap coefficients, $\boldsymbol{c}$. $\mathcal{E}^2(\boldsymbol{x})$ is, after m iterations, the filter squared error at $\boldsymbol{x} = \boldsymbol{x}_m + \Delta_{\boldsymbol{x}}$:

$$\mathcal{E}^2(\boldsymbol{x}_m + \Delta_{\boldsymbol{x}}) \approx \mathcal{E}^2(\boldsymbol{x}_m) + \nabla_{\mathcal{E}^2}(\boldsymbol{x}_m)^\top \Delta_{\boldsymbol{x}} + \frac{1}{2}\Delta_{\boldsymbol{x}}^\top \nabla^2_{\mathcal{E}^2}(\boldsymbol{x}_m)\Delta_{\boldsymbol{x}}$$

Assume that $\boldsymbol{x}_0$ is a known feasible initial point. Set $\Delta_z = \operatorname{vec}(\Delta_{\boldsymbol{x}}, \Delta_{\varepsilon^2}, \Delta_X, \Delta_Y, \Delta_Z)$ and, after the m'th iteration, $z_m = \operatorname{vec}(\boldsymbol{x}_m, \varepsilon_m^2, P_m, Q_m, X_m, Y_m, Z_m)$.

Successive convex optimisation with the algorithm of *Dinh et al.* [194, Algorithm 1]

Apply Equation O.32c to Equation O.18:

$$\begin{bmatrix} L(P,Q) & 0 & 0 \\ 0 & -\varepsilon^2 I & 0 \\ 0 & 0 & I \end{bmatrix} + \frac{1}{2}(U^* + V)^*(U^* + V) - \frac{1}{2}(U^* - V)^*(U^* - V) \preceq 0$$

where $U = \begin{bmatrix} -I_n & A & B & 0 \\ 0 & C & D & -I_m \end{bmatrix}^*$ and $V = \begin{bmatrix} X & 0 \\ Y & 0 \\ Z & 0 \\ 0 & I_p \end{bmatrix}^*$.

Applying the Schur complement to the first and second terms:

$$\begin{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\varepsilon^2 I & 0 \\ 0 & 0 & I \end{bmatrix} & \frac{1}{\sqrt{2}}(U^* + V)^* \\ \frac{1}{\sqrt{2}}(U^* + V) & -I \end{bmatrix} - \frac{1}{2}\begin{bmatrix} R^* R & 0 \\ 0 & 0 \end{bmatrix} \preceq 0$$

where $R = (U^* - V)^*$. After the m'the iteration, at $z = z_m + \Delta_z$, $A(\boldsymbol{x}) \approx A(\boldsymbol{x}_m) + \sum_l \Delta_{x_l} A_l$, etc. and $X(z) \approx X(z_m) + \Delta_X$, etc. The linear approximation to the bilinear term, $R(z)^* R(z)$, is:

$$R(z)^* R(z) \approx R(z_m)^* R(z_m) + R(z_m)^* \Delta_R + \Delta_R^* R(z_m)$$

At each iteration, the SDP decision variables are Δ_z.

Successive convex optimisation with the algorithm of *Lee* and *Hu* [135, Algorithm 1]

Alternatively, apply the algorithm of *Lee* and *Hu* to Equation O.31. After the m'the iteration, at $z = z_m + \Delta_z$, the linear part is:

$$\begin{aligned} \nabla F(z_m)\Delta_z \approx & \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\Delta_{\varepsilon^2} & 0 \\ 0 & 0 & 0 \end{bmatrix} + \dots \\ & \operatorname{He}\left(\begin{bmatrix} X_m & 0 \\ Y_m & 0 \\ Z_m & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & \Delta_A & \Delta_B & 0 \\ 0 & \Delta_C & \Delta_D & 0 \end{bmatrix}\right) + \operatorname{He}\left(\begin{bmatrix} \Delta_X & 0 \\ \Delta_Y & 0 \\ \Delta_Z & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} -I & A & B & 0 \\ 0 & C & D & -1 \end{bmatrix}\right) \end{aligned}$$

[v] The Octave function *schurOneMAPlatticeDoublyPipelined2Abcd* returns the $A_0, \dots, A_N$ matrixes of the all-pass, doubly-pipelined, one-multiplier, Schur lattice filter shown in Figure 5.12. If this filter has order N, then it has $3N + 2$ states. The Octave function *schurOneMR2lattice2Abcd* returns the corresponding state variable $A_0, A_1, \dots, B_0, B_1, \dots, C_0, C_1, \dots$ and $D_0, D_1, \dots$ matrixes for the $\boldsymbol{k}$ and $\boldsymbol{c}$ coefficients of the $R = 2$ pipelined tapped one-multiplier Schur lattice shown in Algorithm 5.9. If this filter has order N, then it has $\frac{3}{2}N - 1$ states.

The overall bilinear matrix inequality is:

$$F\left(z_m+\Delta_z\right) \approx F\left(z_m\right)+\nabla F\left(z_m\right) \Delta_z+\operatorname{He}\left(\begin{bmatrix} \Delta_X & 0 \\ \Delta_Y & 0 \\ \Delta_Z & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & \Delta_A & \Delta_B & 0 \\ 0 & \Delta_C & \Delta_D & 0 \end{bmatrix}\right) \preceq 0 \tag{O.36}$$

With the convex upper-approximation of *Lee* and *Hu* [135, Lemma 2], shown in Equation O.34 the bilinear part of Equation O.36 is:

$$\operatorname{He}\left(\begin{bmatrix} \Delta_X & 0 \\ \Delta_Y & 0 \\ \Delta_Z & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & \Delta_A & \Delta_B & 0 \\ 0 & \Delta_C & \Delta_D & 0 \end{bmatrix}\right) \preceq \begin{bmatrix} \Delta_X & 0 \\ \Delta_Y & 0 \\ \Delta_Z & 0 \\ 0 & 0 \end{bmatrix} S \begin{bmatrix} \Delta_X & 0 \\ \Delta_Y & 0 \\ \Delta_Z & 0 \\ 0 & 0 \end{bmatrix}^\top + \ldots$$
$$\begin{bmatrix} 0 & \Delta_A & \Delta_B & 0 \\ 0 & \Delta_C & \Delta_D & 0 \end{bmatrix}^\top S^{-1} \begin{bmatrix} 0 & \Delta_A & \Delta_B & 0 \\ 0 & \Delta_C & \Delta_D & 0 \end{bmatrix} \preceq 0$$

where $S \in \mathbb{S}_{++}^{n+1}$ is a positive definite, symmetric matrix. After linearising S^{-1} and applying the Schur complement, as shown in Equation O.35, at the m'th iteration the convex upper-approximation at $z = z_m + \Delta_z$ satisfies:

$$\begin{bmatrix} F\left(z_m\right)+\nabla F\left(z_m\right) \Delta_z & * & * \\ S_m \begin{bmatrix} \Delta_X & 0 \\ \Delta_Y & 0 \\ \Delta_Z & 0 \\ 0 & 0 \end{bmatrix}^\top & -2S_m+S & * \\ \begin{bmatrix} 0 & \Delta_A & \Delta_B & 0 \\ 0 & \Delta_C & \Delta_D & 0 \end{bmatrix} & 0 & -S \end{bmatrix} \preceq 0 \tag{O.37}$$

The SDP decision variables are Δ_z and S.

O.4.4 Examples of the design of one-multiplier Schur lattice filters with BMI constraints derived from the KYP lemma

Design of parallel all-pass, doubly-pipelined, one-multiplier Schur lattice filters with BMI constraints derived from the KYP lemma

The Octave script *schurOneMPAlatticeDoublyPipelinedDelay_kyp_Dinh_lowpass_test.m* implements the successive convex approximation algorithm of *Dinh et al.* with YALMIP and the SDPT3 solver in order to design a filter consisting of the parallel combination of a delay and a Schur one-multiplier all-pass lattice filter. The all-pass lattice state-transition matrix, A_{ap}, is a linear combination of the reflection coefficients, $\boldsymbol{k}$, and the other state variable matrixes, B_{ap}, C_{ap}, and D_{ap}, are constant. The initial filter is designed with the *WISE* method shown in Section 8.1.5. The P and Q KYP lemma symmetric matrixes are recalculated at each BMI iteration. The dX, dY, etc matrixes are initialised with YALMIP and the SeDuMi solver. The objective function is an approximation to the summed squared error of the amplitude response. It appears that, for this IIR filter, this objective function sufficiently close to convex that SDPT3 can optimise it, whereas the SeDuMi solver cannot.

The filter specification is:

```
tol=5e-06 % General tolerance
maxiter_kyp=50 % Maximum number of KYP iterations
use_hessEsq=1 % Use 2nd order approx. to Esq
N=5 % Allpass filter order
DD=4 % Parallel delay order
fap=0.1 % Amplitude pass band edge
Wap=0.1 % Amplitude pass band weight
fas=0.2 % Amplitude stop band edge
Was=200 % Amplitude stop band weight
```

The initial reflection coefficients are:

```
k0 = [  -0.5887890122,   0.3794825489,   0.2077136416,   0.0867433777, ...
         0.0216280074 ];
```

Figure O.15 shows the initial filter amplitude response. The optimised reflection coefficients are:

```
k = [  -0.5587865811,   0.4149916911,   0.2259884090,   0.0921483938, ...
        0.0177132903 ];
```

Figure O.16 shows the filter amplitude and delay responses. Figure O.17 shows the filter pole-zero plot. Figure O.18 shows the convergence of the squared-error and Δ_k of the filter design. Figure O.19 shows the convergence of the minimum pass-band amplitude and maximum stop-band amplitude of the filter design.

Figure O.15: Initial amplitude response of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay.

Figure O.16: Amplitude and delay responses of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the generalised KYP lemma amd the algorithm of *Dinh et al.* [194, Algorithm 1].

Figure O.17: Pole-zero plot of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the generalised KYP lemma amd the algorithm of *Dinh et al.* [194, Algorithm 1].

Figure O.18: Convergence of the mean-squared-error of the amplitude response and the $\|\Delta_k\|$ of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the generalised KYP lemma and the algorithm of *Dinh et al.* [194, Algorithm 1].

Figure O.19: Convergence of the minimum pass-band and maximum stop-band amplitudes of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the generalised KYP lemma and the algorithm of *Dinh et al.* [194, Algorithm 1].

Figure O.20: Amplitude and delay responses of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the generalised KYP lemma and the bilinear algorithm of *Lee* and *Hu* [135, Algorithm 1].

The Octave script *schurOneMPAlatticeDoublyPipelinedDelay_kyp_LeeHu_lowpass_test.m* implements the successive convex approximation algorithm of *Lee* and *Hu* with YALMIP and the SDPT3 solver in order to design a filter consisting of the parallel combination of a delay and a Schur one-multiplier all-pass lattice filter. The initial filter is designed with the *WISE* method shown in Section 8.1.5. The objective function is an approximation to the summed squared error of the amplitude response. The filter specification and the initial filter are those of Section O.4.3.

The optimised reflection coefficients are:

```
k = [  -0.5598090941,   0.4108251626,   0.2231966481,   0.0898728112, ...
        0.0164139680 ];
```

Figure O.20 shows the filter amplitude and delay responses. Figure O.21 shows the filter pole-zero plot. Figure O.22 shows the convergence of the squared-error and Δ_z of the filter design. Figure O.23 shows the convergence of the minimum pass-band amplitude and maximum stop-band amplitude of the filter design.

Figure O.21: Pole-zero plot of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the generalised KYP lemma amd the algorithm of *Lee, Hu et al.* [135, Algorithm 1].

Figure O.22: Convergence of the mean-squared-error of the amplitude response and the $\|\Delta_k\|$ of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the generalised KYP lemma and the algorithm of *Lee* and *Hu* [135, Algorithm 1].

Figure O.23: Convergence of the minimum pass-band amplitude and maximum stop-band amplitudes of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the generalised KYP lemma and the algorithm of *Lee* and *Hu* [135, Algorithm 1].

Figure O.24: Amplitude response of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the PCLS algorithm and SOCP optimisation.

	$\mathcal{E}^2$	A_{max}(dB)	A_{min}(dB)
Initial	4.18e-05	-42.56	-0.558
SOCP(PCLS)	5.83e-06	-49.40	-0.070
KYP(Dinh)	2.92e-06	-42.80	-0.130
KYP(Lee and Hu)	3.4e-06	-42.54	-0.155

Table O.1: Mean-squared-error of the amplitude response, maximum stop-band amplitude and minimum pass-band amplitude of the SOCP(PCLS), KYP(Dinh) and KYP(Lee and Hu) designs of a low-pass Schur one-multiplier all-pass lattice filter in parallel with a delay.

For comparison, the Octave script *schurOneMPAlatticeDelay_socp_slb_lowpass_test.m* uses the PCLS algorithm and SOCP optimisation to design a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay. The initial filter is designed with the WISE method of *Tarczynski et al.*, shown in Section 8.1.5, and the filter specification is similar to the previous examples in this section. The optimised reflection coefficients are:

```
A1k = [  -0.5662109737,   0.4317358628,   0.2428077171,   0.1078009224, ...
          0.0303444618 ];
```

Table O.1 compares the mean-squared-error, the maximum in the stop-band and the minimum in the pass-band of the amplitude responses of the SOCP(PCLS), KYP(Dinh) and KYP(Lee and Hu) designs of a low-pass Schur one-multiplier all-pass lattice filter in parallel with a delay. Figure O.24 shows the filter amplitude response. Figure O.25 shows the filter pole-zero plot. The SOCP(PCLS) design procedure is much faster than the KYP design procedure.

Figure O.25: Pole-zero plot of a lowpass filter comprised of a Schur one-multiplier lattice all-pass filter in parallel with a delay designed with the PCLS algorithm and SOCP optimisation.

Figure O.26: Amplitude and delay responses of the initial R=2 tapped one-multiplier Schur lattice lowpass filter designed with the WISE method.

Design of R=2 tapped one-multiplier Schur lattice filters with BMI constraints derived from the KYP lemma

The Octave script *schurOneMlattice_kyp_Dinh_lowpass_R2_test.m* implements the successive convex approximation algorithm of *Dinh et al.* with YALMIP and the SDPT3 solver in order to design a tapped one-multiplier Schur lattice filter with transfer function denominator polynomial coefficients only in z^{-2}. The initial filter is designed with the *WISE* method of *Tarczynski et al.*, shown in Section 8.1.5. The dX, dY, etc matrixes are initialised with YALMIP and the SeDuMi solver. The objective function is the weighted sum of the pass band and stop band errors, ε_z^2 and ε_s^2, and the regularisation factor shown in Equation O.33. The filter specification is;

```
tol=5e-05 % General tolerance
maxiter_kyp=25 % Maximum number of KYP iterations
nN=8 % Tapped Schur lattice filter order
fap=0.1 % Amplitude pass band edge
fas=0.2 % Amplitude stop band edge
Was=1 % Amplitude stop band weight
tp=5 % Nominal pass band group-delay(samples)
```

The initial filter coefficients are:

```
k0 = [   0.0000000000,  -0.3707309661,   0.0000000000,   0.5999187031, ...
         0.0000000000,  -0.2903498716,   0.0000000000,   0.0826241645 ];

c0 = [  -0.0517776968,   0.0408108873,   0.1288678062,   0.1749163632, ...
         0.3082307401,   0.2373884305,   0.1017061108,   0.0447074437, ...
         0.0127830033 ];
```

Figure O.26 shows the amplitude and delay responses of the initial filter. Figure O.27 shows the pole-zero plot of the initial filter. The optimised filter coefficients are:

```
k = [   0.0000000000,  -0.3709291459,   0.0000000000,   0.5995266555, ...
        0.0000000000,  -0.3228875589,   0.0000000000,   0.1375876146 ];
```

Figure O.27: Pole-zero plot of the initial R=2 tapped one-multiplier Schur lattice lowpass filter designed with the WISE method.

```
c = [  -0.0059237030,   0.1083595957,   0.1621684100,   0.1799780728, ...
        0.2657098972,   0.1704985791,   0.0541938219,   0.0127833150, ...
       -0.0049715841 ];
```

The corresponding filter transfer function numerator and denominator polynomials are:

```
n1 = [  -0.0049715841,   0.0145421408,   0.0657831784,   0.1198876703, ...
         0.1516834782,   0.1553307715,   0.1292858609,   0.0757740290, ...
         0.0234105421 ];


d1R = [   1.0000000000,   0.0000000000,  -0.8313160836,   0.0000000000, ...
          0.8999448954,   0.0000000000,  -0.4311539808,   0.0000000000, ...
          0.1375876146 ];
```

Figure O.28 shows the filter amplitude and delay responses. Figure O.29 shows the filter pole-zero plot. Figure O.30 shows the convergence of the squared-error and Δ_z of the filter design. Figure O.31 shows the convergence of the minimum pass-band amplitude and maximum stop-band amplitude of the filter design.

Figure O.28: Amplitude and delay responses of an R=2 tapped one-multiplier Schur lattice lowpass filter designed with the generalised KYP lemma and the bilinear algorithm of *Dinh et al.* [194, Algorithm 1].

Figure O.29: Pole-zero plot of an R=2 tapped one-multiplier Schur lattice lowpass filter designed with the generalised KYP lemma amd the algorithm of *Dinh et al.* [194, Algorithm 1].

Figure O.30: Convergence of the mean-squared-error of the amplitude response and the $\|\Delta_{\boldsymbol{x}}\|$ of an R=2 tapped one-multiplier Schur lattice lowpass filter designed with the generalised KYP lemma and the algorithm of *Dinh et al.* [194, Algorithm 1].

Figure O.31: Convergence of the minimum pass-band amplitude and maximum stop-band amplitudes of an R=2 tapped one-multiplier Schur lattice lowpass filter designed with the generalised KYP lemma and the algorithm of *Dinh et al.* [194, Algorithm 1].

Colophon

Software requirements

I currently use the Fedora 42 distribution of Linux [57] with:

- `uname -r` is *6.15.10-200.fc42.x86_64*
- *gcc* version *15.2.1 20250808 (Red Hat 15.2.1-1)*
- a local build of *octave-10.2.0* with the *struct*, *io*, *statistics*, *optim*, *control*, *signal*, *symbolic*[a] and *parallel* Octave-Forge packages
- the Octave scripts use the *qt* graphics toolkit with *qt6-qtbase-6.9.1-1* and a local build of *GraphicsMagick*
- *texlive-2023-76* and various packages are installed
- *dia-0.97.3-29* is used for line drawings

The local build of *octave-10.2.0* and Octave-Forge packages assumes that the following packages are installed in addition to the base Fedora distribution:

```
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 librsvg2 librsvg2-devel \
librsvg2-tools icoutils autoconf automake libtool pcre pcre-devel freetype \
freetype-devel gnupg2 texlive-dvisvgm gl2ps gl2ps-devel hdf5 hdf5-devel \
qhull qhull-devel portaudio portaudio-devel libsndfile libsndfile-devel \
libcurl libcurl-devel gl2ps gl2ps-devel fontconfig-devel mesa-libGLU \
mesa-libGLU-devel qt qt6-qtbase qt6-qtbase-common qt6-qtbase-devel \
qt6-qtbase-gui qt6-qt5compat qt6-qt5compat-devel qt6-qttools \
qt6-qttools-common qt6-qttools-devel rapidjson-devel python3-sympywget \
rapidjson-devel python3-sympy java-21-openjdk-devel xerces-j2 \
util-linux util-linux-script
```

The oct-file *src/minphase.cc* assumes the *eigen3-devel* package is installed. The *maxima* package is required to run the *.max* scripts.

The following packages are required to build this document:

```
dnf install dia epstool texlive \
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-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
```

[a] The *symbolic* Octave-Forge package assumes that the *python3-sympy* Fedora package is installed.

Makefile

This document is built with *GNU make-4.4.1*. 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 [67] 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*[b]:

```
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 delayz.m \
tf2schurNSlattice.m schurNSlatticeNoiseGain.m schurNSlatticeFilter.m \
svf.m KW.m optKW.m tf2Abcd.m crossWelch.m p2n60.m \
schurexpand.oct schurdecomp.oct schurNSscale.oct \
schurNSlattice2Abcd.oct qroots.oct
```

Octave

The *build-octave.sh* script builds a patched, local, command-line only, version of *octave-10.2.0* 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-10.2.0.patch*, is a shell *here* document within *build-octave.sh*.

The *build-octave.sh* script assumes the default sizes of FORTRAN data types. For example, `real` and `integer` are 4 bytes. The script must be modified to use 8 byte integer array indexes in *blas*, *lapack*, *SuiteSparse*, *octave* etc.

The local build of Octave links to locally built versions of the following libraries: *arpack-ng-3.9.1* [12], *fftw-3.3.10* [58], *glpk-5.0* [69], *lapack-3.12.1* [130][c], *qrupdate-1.1.2* [193], *SuiteSparse-7.11.0* [233], *sundials-7.4.0* [134] and *GraphicsMagick-1.3.45* [109].

Display the Octave internal build configuration with:

```
$ octave --eval "__octave_config_info__"
```

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 *build-octave.sh* script builds Octave with the *java* interface. Initialise the *java* interface similarly to:

```
usejava ("jvm")
javaaddpath(strcat(OCTAVE_HOME,"/share/octave/",OCTAVE_VERSION,"/m/java"));
javaclasspath
```

[b] *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*.

[c] The 16-byte DOUBLE LAPACK static libraries, *libqblas.a* and *libqlapack.a*, are compiled with the `-freal-8-real-16` flag.

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[d]:

> 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.

MATLAB compatibility

I do not own a copy of MATLAB so I cannot directly check that the Octave code in this project is compatible with MATLAB. You will most likely need to experiment, debug and convert *octfile*s to *mexfile*s.

Do at least the following to *make* this project with MATLAB:

- in *Makefile* set the `OCTAVE` and `OCTAVE_FLAGS` variables appropriately and remove `octfiles` from the `all` target
- in the *src/mk/*.mk* files change all occurrences of `.oct` to `.m`

I endeavour to ensure that this project provides m-file alternatives to the *octfiles*[e]. Of course, the *octfile*s often use multi-precision arithmetic so these m-file alternatives will likely produce different numerical results and the top-level Octave scripts may fail.

I endeavour to fix warnings that occur for the code in this project when Octave is run with the `-traditional` run-time option and the *src/test_common.m* script calls:

```
warning("on","Octave:language-extension");
```

The Octave code from the Octave project produces many warnings. I ignore *automatic-broadcast* warnings from the Octave code in this project because I believe that, in fact, MATLAB does support automatic broadcasting when the vector or matrix dimensions are compatible. See the Octave manual entry *Broadcasting and Legacy Code* [111, Section 19.2.1].

Problems with Octave

The *octave-10.2.0.patch* `here` document within *build-octave.sh* patches files to:

- prevent a *valgrind* warning from *load-save.cc*

I have found that the best way to include plots in this document is to print from Octave with the *-dpdflatex* device. Unfortunately, producing plots this way is very slow. Here is an example for *octave-9.2.0*:

[d] Also see https://kristerw.github.io/2021/10/19/fast-math/.
[e] Unlike the built-in *roots* function, *qroots.cc* and *qroots.m* expect that the polynomial coefficients are real.

```
$ time octave -q --no-gui src/test/print_latex_test.m

ans = 9.2.0-robj
Elapsed time is 17.3568 seconds.
   #      Function Attr     Time (s)   Time (%)        Calls
-----------------------------------------------------------
  36           set    R       12.180      68.80           61
  20           get             5.013      28.32          269
  33 __go_uimenu__             0.196       1.11           24
 182 __go_delete__             0.112       0.63            1
 172       drawnow             0.078       0.44            3

real    0m18.142s
user    0m8.938s
sys     0m9.143s
```

I ran *octave* under *callgrind* and viewed the results with *qcachegrind*. *run_command_and_return_output()* was called 123 times. Next I used *gdb* to look at the *cmd_str* passed to that function. It seems that for each tick-mark label in the TEXfile, the LATEX, *dvisvgm* and *dvipng* renderers are run in a sub-process 41 times each. The aim appears to be to find the bounding box of the label for typesetting. LATEX provides macros to do this. However, I use the *-dpdflatex* print device and *pdflatex* not *latex*. *pdflatex* generates PDF files and, accordingly, uses Adobe Type 1 fonts rather than TEX METAFONT fonts. See the discussion at https://savannah.gnu.org/bugs/?66012.

Using the Octave external code interface

From the Octave *info* documentation:

> Octave offers a versatile interface for including chunks of compiled code as dynamically linked extensions.

I use the Octave external code interface for the following reasons:

Improved execution time For example, the *schurdecomp.m* and *zhong_inverse.m* functions use recurrence relations to calculate intermediate results. Recurrence relations cannot be vectorised in Octave code.

Access to LAPACK functions For example, the *complex_zhong_inverse* Oct-file pre-processes a Hessenberg matrix to produce a triangular matrix and then calls the LAPACK ZTRTRI function to calculate the inverse of that matrix. Similarly, the *complex_lower_hessenberg_inverse* Oct-file calls the LAPACK ZGBSV function to calculate the inverse of a lower Hessenberg, banded, matrix.

Improved accuracy For example, the *schurdecomp* Oct-file uses the MPFR [1] multi-precision library to obtain improved accuracy in the Schur decomposition of a polynomial. As mentioned in the Introduction, polynomial root-finding is notoriously difficult. The *qroots* Oct-file adapts the GNU Scientific Library [70] *gsl_poly_complex_solve()* function, in the GSL file *poly/zsolve.c*, from `double` to IEEE `__float128` variables. Similarly, the *mzsolve* Oct-file adapts that GSL function from `double` to an `mpfr_t` data type defined by the *Boost.Multiprecision* C++ template library [110]. I tried running the existing test scripts with *mzsolve* called by *qroots.m*. A single test fails because of the differences in an approximately zero value. Finally, the *lzsolve* Oct-file calls a 16-byte `DOUBLE`, or IEEE `__float128`, version of the LAPACK DGEEV function[f]. Unsurprisingly, the accuracy obtained with *lzsolve* is similar to that obtained with *qroots*.

Octave packages

The *build-octave.sh* script installs the following Octave Forge packages:

[f] The *src/mk/lzsolve_test.mk* Makefile fragment adds the *libqblas.a* and *libqlapack.a* static libraries to *lzsolve.oct*.

```
computer=x86_64-pc-linux-gnu
octave version=10.2.0-robj (HG-ID=8d42ee70e124)
Package Name  | Version | Installation directory
--------------+---------+-----------------------
     control *|   4.1.2 | .../.local/share/octave/api-v60/packages/control-4.1.2
          io  |   2.7.0 | .../robj/.local/share/octave/api-v60/packages/io-2.7.0
       optim  |   1.6.2 | .../.local/share/octave/api-v60/packages/optim-1.6.2
    parallel  |   4.0.2 | .../share/octave/api-v60/packages/parallel-4.0.2
        piqp  |   0.6.0 | .../.local/share/octave/api-v60/packages/piqp-0.6.0
      signal *|   1.4.6 | .../.local/share/octave/api-v60/packages/signal-1.4.6
  statistics  |   1.7.5 | .../share/octave/api-v60/packages/statistics-1.7.5
      struct  |  1.0.18 | .../.local/share/octave/api-v60/packages/struct-1.0.18
    symbolic  |   3.2.1 | .../share/octave/api-v60/packages/symbolic-3.2.1
```

The *build-octave.sh* script installs the *SeDuMi*, *SDPT3*, *SCS* and *YALMIP* from their respective software repositories, and forked versions of the *SparsePOP* and *gloptipoly3* solvers from `https://github.com/robertgj`.

Problems with Octave-Forge packages

The following packages are patched with shell *here* documents within *build-octave.sh*:

- the *signal-1.4.6* package `zplane` function is patched to avoid a LaTeX error.
- the *signal-1.4.6* package `grpdelay` function is replaced with a local version called `delayz`.

Building Octave

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 -march=$CPU_TYPE -O2"
export CFLAGS="-I$LOCAL_PREFIX/include $BLDOPTS"
export CXXFLAGS="-I$LOCAL_PREFIX/include $BLDOPTS"
export FFLAGS=$BLDOPTS
export LDFLAGS="-L$LAPACK_DIR -L$LOCAL_PREFIX/lib"

PGO_GEN_FLAGS="-pthread -fprofile-generate"
XTRA_CFLAGS=$PGO_GEN_FLAGS \
XTRA_CXXFLAGS=$PGO_GEN_FLAGS \
$OCTAVE_DIR/configure $OCTAVE_CONFIG_OPTIONS \
--prefix=$OCTAVE_INSTALL_DIR --with-blas=-lblas --with-lapack=-llapack

make V=1 -j6 -O

find . -name \*.gcda -exec rm {} ';'
make check

find . -name \*.o -exec rm -f {} ';'
find . -name \*.lo -exec rm -f {} ';'
find . -name \*.la -exec rm -f {} ';'
find . -name moc* -exec rm -f {} ';'

PGO_LTO_FLAGS="-pthread -flto=6 -ffat-lto-objects -fprofile-use"
XTRA_CFLAGS=$PGO_LTO_FLAGS \
XTRA_CXXFLAGS=$PGO_LTO_FLAGS \
$OCTAVE_DIR/configure $OCTAVE_CONFIG_OPTIONS \
--prefix=$OCTAVE_INSTALL_DIR --with-blas=-lblas --with-lapack=-llapack
```

```
make V=1 -j6 -O

# Hack to remove LTO profiling from mkoctfile
sed -i -e "s/$PGO_LTO_FLAGS//" ./src/mkoctfile.cc
make V=1 src/mkoctfile
```

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="-ggdb3 -m64 -O0 -march=$CPU_TYPE"
export CFLAGS="-I$LOCAL_PREFIX/include $BLDOPTS"
export CXXFLAGS="-I$LOCAL_PREFIX/include $BLDOPTS"
export FFLAGS=$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

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 CPU is an Intel i7-7700K with 4 CPU cores (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 -march=nehalem*. 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 \
       --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"
```

Octave build	IIR benchmark execution time (seconds)
Debug, shared reference lapack and blas	243.8
Shared reference lapack and blas	49.2
Shared reference lapack and blas, LTO	49.0
Shared reference lapack and blas, PGO	50.0
Shared reference lapack and blas, LTO, PGO	49.0

Table O.2: Average execution time for 10 runs of the IIR benchmark script with various *Octave* builds.

blas library	linpack.m		IIR benchmark script	KYP benchmark script
	MFLOPS	Normalised residual	Execution time (seconds)	Execution time (seconds)
x86-64 libblas/liblapack	6505	21.3	48.9	55.4
Nehalem libblas/liblapack	6503	21.3	48.8	55.4
Haswell libblas/liblapack	8325	18.5	48.6	49.7
Skylake libblas/liblapack	7894	18.5	48.6	49.7
Fedora 42 libblas/liblapack	6590	21.3	48.9	55.3
libgslcblas	6495	21.3	48.9	55.3
libsatlas	13384	12.1	50.0	60.3
libtatlas	20168	12.8	49.4	60.3
libopenblas	33243	9.9	48.1	49.7
libopenblasp (1 thread)	33140	9.9	48.2	50.1
libopenblasp (2 threads)	46965	12.1	48.4	50.2
libopenblasp (4 threads)	58886	10.8	48.4	50.3

Table O.3: Benchmark results for *linpack.m* and the IIR and KYP benchmark scripts with various *blas* and *lapack* implementations.

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* [141] script, the previous IIR benchmark script and a script that solves a KYP problem by calling YALMIP and SeDuMi. The *linpack.m* script was modified to produce repeatable residuals. The alternative *blas* and *lapack* library versions used are local builds of *blas-3.12.1* and *lapack-3.12.1* and the Fedora 40 *lapack*, *blas*, *atlas* [14], *gsl* [70] and *openblas* [169] packages[g]:

```
atlas.x86_64            3.10.3-28.fc42
blas.x86_64             3.12.0-8.fc42
lapack.x86_64           3.12.0-8.fc42
gsl.x86_64              2.8-1.fc42
gsl-devel.x86_64        2.8-1.fc42
openblas.x86_64         0.3.29-1.fc42
openblas-threads.x86_64 0.3.29-1.fc42
```

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[h] 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.m* 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.32 shows the mean execution times for frequency vectors with a length that is a multiple of 200. Figure O.33 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
0.14709002182060871
> LD_LIBRARY_PATH=/usr/local/octave-9.1.0/lib ./dgesvd_test_sysblas
0.14709002182060871
Previous versions of *atlas* produced slightly different results on each run.

[h]Probably not enough!

Figure O.32: Mean execution times of *freqz*, *iirA* and *schurOneMPAlatticeAsq* for frequency vectors with a length that is a multiple of 200.

Figure O.33: Mean execution times of *freqz*, *iirA* and *schurOneMPAlatticeAsq* for frequency vectors with a length that is a power of 2.

Profiling Octave

Profiling Octave scripts with the internal profiler

Here is an example of the use of the internal Octave profiler in the Octave script *yalmip_kyp_moment_lowpass_test.m*:

```
Constraints=[F_max<=-sedumi_eps, Q_max>=0, F_pl<=0, Q_pl>=0, F_s<=0, Q_s>=0];
Objective=(hsdp*G*hsdp')+(2*hsdp*g')+(2*fap);
Options=sdpsettings("solver","moment","sedumi.eps",sedumi_eps);
profile("on");
try
  sol=optimize(Constraints,Objective,Options);
catch
  fprintf(stderr,lasterror().message);
end_try_catch
profile("off");
T=profile("info");
profshow(T);
```

with Octave profiler output:

```
   #              Function Attr     Time (s)   Time (%)        Calls
---------------------------------------------------------------------
 415               blkchol          1693.754      93.10           14
 414               getada3            50.494       2.78           14
 412               getada2            27.194       1.49           14
 411               getada1            23.795       1.31           14
 377                sedumi             4.569       0.25            1
 427              bwblkslv             3.000       0.16           60
 423              fwblkslv             1.879       0.10           60
 393            getsymbada             1.604       0.09            1
 313        findhashsorted             1.264       0.07        16900
   8              binary +    R        1.194       0.07       274728
 179                system             0.949       0.05            5
 311                   abs             0.767       0.04          301
 400             symbfwblk             0.704       0.04            4
  73              binary *             0.667       0.04         6194
 272                sparse             0.628       0.03          100
 288             monpowers             0.623       0.03            2
 419              psdscale             0.609       0.03          205
 294 @sdpvar/recovermonoms             0.544       0.03            2
 396                  tril             0.413       0.02          869
 300        @sdpvar/mtimes             0.388       0.02            1
```

Profiling the Octave binary

I have not succeeded in profiling the *octave* binary with *gprof*[i] or `.oct` shared object files with either *sprof* or *gprofng*.

To "sort-of" profile *octave* with *callgrind*:

```
valgrind --tool=callgrind --enable-debuginfod=no --instr-atstart=no \
 --dump-instr=yes --collect-jumps=yes --log-file=valgrind.%p.log \
 --trace-children=yes /usr/local/octave-10.0.0/bin/octave-10.0.0 --no-gui
```

When the *octave* prompt appears, run `callgrind_control -instr=on` in another terminal to start sampling.

[i] I configured *octave* with `-disable-docs` and compiled with `-ggdb3 -O0 -pg`. Unfortunately, running *octave* fails with the message "Profiling timer expired".

Solvers

SeDuMi

The *SeDuMi* source is available from *LeHigh University* as *SeDuMi_1_3.zip* [230]. Unfortunately, this version does not seem to be actively maintained. An "unofficial" *GitHub* [229] fork is compatible with Octave. I patch the *1.3.8* tag to avoid `gcc` compiler warnings[j].

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 [231, 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* [107] 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
-------------------------------------------------------
 155   psdscale            46.511      26.09         4121
  34   binary *            40.197      22.55        51690
 150    getada3            28.125      15.78          162
 175        eig            18.437      10.34         4055
 173   binary \             9.069       5.09         2443
  84   binary /             8.105       4.55        12451
 181 psdframeit             5.488       3.08          370
 170 psdinvjmul             5.296       2.97          182
```

The *SeDuMi* m-file *SeDuMi/psdscale.m* is complex and has no comments.

SparsePOP

SparsePOP [84] 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 [168]. The latter is based on *SymPy* [232], 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.

[j] 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 [114, 113, 198] is a MATLAB software for semidefinite-quadratic-linear programming. The version used in this project is a fork of the "unofficial" GitHub repository [152].

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 [108, 101] is a toolbox for modelling and optimisation in MATLAB and Octave. *YALMIP* calls semi-definite programming (SDP) solvers such as *SeDuMi* and *SDPT3*.

LMIRank

LMIRank [214, 201] is a toolbox for solving rank constrained LMI problems:

$$\begin{aligned} F(x) &\succeq 0 \\ G(x) &\succeq 0 \\ \operatorname{rank} G(x) &\leq r \end{aligned}$$

The Octave file *lmirank.m* contains the `lmirank` and `trheuristic` functions. The Octave test script *lmirank_test.m* contains the `lmiranktest2` function. `lmirank` calls the *SeDuMi* SDP solver. *YALMIP* has an interface to `lmirank`.

gloptipoly3

gloptipoly3 [42, 43, 132, 133] 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".

SCS

The *Splitting Conic Solver* (SCS) [30, 31, 32] is software for solving semidefinite-quadratic-linear programming problems. There are two varieties of SCS, a *direct* LMI solver and an *indirect* conjugate-gradient solver[k].

Compiling SCS with `__float128` floating-point numbers

The *build-octave.sh* script installs SCS with patches to enable compilation with `__float128` floating-point numbers by setting `dfloat=true` in *make_scs.m*, compiling *scs_qprintf.c*, and linking with the *libquadmath.so*, *libqblas.a* and *libqlapack.a* libraries. I found that with `__float128` floating-point numbers SCS was 1 to 2 orders of magnitude slower than with `double` floating-point numbers.

Problems with SCS

The documentation is confusing. In many cases SeDuMi and SDPT3 find a better solution.

PIQP

The *Proximal Interior-Point Quadratic Programming* (PIQP) solver [222] solves, as the name suggests, quadratic programming problems with equality and inequality constraints . It is written in C++ template header files and is based on the Eigen C++ template header library [73]. It is available as an Octave-Forge package [166].

Problems with PIQP

In general, the design of IIR filters is a non-convex problem that is unsuited to quadratic programming optimisation.

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"
```

[k] YALMIP supports the *"scs"*, *"scs-direct"* and *"scs-indirect"* SCS solver names.

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 [119] describes configuration of a QEMU [192] 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.

Rename the VM

To change the name of a VM:

```
virsh shutdown foo
virsh domrename foo bar
virsh start bar
```

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 \
maxima
.
.
[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[1]:

```
$ sudo virsh shutdown Fedora36Server
$ sudo virt-sparsify -v -x --in-place your-image-file
```

[1] First install the Fedora *guestfs-tools* package.

Alternatively, fill unused VM disc sectors with 0:

```
$ 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 [188]. 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* [189] project. *fhist* depends on the *libexplain* [190] 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. 93, 95, 748, 921, 1006

[2] A. Antoniou and W.-S. Lu. *Practical Optimization: Algorithms and Engineering Applications*. Springer Science+Business Media, LLC, 2007. ISBN 0-387-71106-6. 227, 670

[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. 19, 20, 151, 161, 170

[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, 19, 20, 72, 98, 99

[5] A. H. Land and A. G. Doig. An Automatic Method Of Solving Discrete Programming Problems. *Econometrica*, 28(3):497–520, July 1960. 464

[6] A. H. Sayed and T. Kailath. A Survey of Spectral Factorization Methods. *Numerical Linear Algebra With Applications*, 08:467–496, 2001. 796

[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. 20, 446, 447

[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. 71

[9] A. P. Ruszczynzki. *Nonlinear Optimization*. Princeton University Press, 2006. ISBN 0-691-11915-5. 20, 716, 718, 724, 726, 727

[10] A. Rantzer. On the Kalman-Yakubovich-Popov Lemma for Positive Systems. *IEEE Transactions on Automatic Control*, 61(5):1346–1349, May 2016. 965

[11] 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. 19, 126, 128, 151, 152, 153, 155, 158

[12] A collection of Fortran77 subroutines designed to solve large scale eigenvalue problems. `https://github.com/opencollab/arpack-ng`. 1004

[13] Athanasios Papoulis. *Signal Analysis*. McGraw-Hill International, 1981. ISBN 0-07-066468-4. 196

[14] Automatically Tuned Linear Algebra Software. `http://math-atlas.sourceforge.net/`. 1010

[15] Automatic Differentiation. `http://www.autodiff.org`. 698

[16] B. C. Carlson. Computing Elliptic Integrals by Duplication. *Numerische Mathematik*, 33:1–16, 1979. 17, 685, 686, 687

[17] B. C. Carlson. POWER SERIES FOR INVERSE JACOBIAN ELLIPTIC FUNCTIONS. *MATHEMATICS OF COMPUTATION*, 77(263):1615–1621, July 2008. 689

[18] B. Christianson. Global Convergence using De-linked Goldstein or Wolfe Linesearch Conditions. *AMO - Advanced Modeling and Optimization*, 1(1):25–31, 2009. 728

[19] B. Després. Positive polynomials and numerical approximation. `https://www.math.univ-paris13.fr/~marchal/exposedespres.pdf`, 2020. 885

[20] B. Dumitrescu. Bounded Real Lemma for FIR MIMO Systems. *IEEE Signal Processing Letters*, 12(7):496–499, July 2005. 787

[21] 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. 147, 151

[22] 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. 787, 793, 794, 795, 796

[23] 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. 880

[24] B. Kumar, S. C. D. Roy and H. Shah. On the Design of FIR Digital Differentiators which Are Maximally Linear at the Frequency π/p, $p \in \{PositiveIntegers\}$. *IEEE Transactions on Signal Processing*, 40(9):2334–2338, September 1992. 880, 883, 884

[25] 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, 63, 65

[26] 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. 63

[27] V. Balakrishnan and L. Vandenberghe. Semidefinite Programming Duality and Linear Time-invariant Systems. http://docs.lib.purdue.edu/ecetr/162, 2002. ECE Technical Reports. Paper 162. 967

[28] Barry R. Marks and Gordon P. Wright. A General Inner Approximation Algorithm for Nonconvex Mathematical Programs. *Operations Research*, 26(4):681–683, 1978. 982

[29] Boost Library Documentation. Jacobi Zeta Function. https://www.boost.org/doc/libs/1_86_0/libs/math/doc/html/math_toolkit/ellint/jacobi_zeta.html. 691

[30] Brendan O'Donoghue. Operator Splitting for a Homogeneous Embedding of the Linear Complementarity Problem. *SIAM Journal on Optimization*, 31:1999–2023, August 2021. 232, 1015

[31] Brendan O'Donoghue and Alastair Abbott. scs-matlab. https://github.com/bodono/scs-matlab, 2023. 232, 1015

[32] Brendan O'Donoghue and Eric Chu and Neal Parikh and Stephen Boyd. SCS: Splitting Conic Solver. https://github.com/cvxgrp/scs, November 2023. 232, 1015

[33] C. Donald La Budde. The Reduction of an Arbitrary Real Square Matrix to Tri-Diagonal Form Using Similarity Transformations. *Mathematics of Computation*, 17(84):433–437, 1963. 37

[34] C. Gumacos. Weighting Coefficients for Certain Maximally Flat Nonrecursive Digital Filters. *IEEE Transactions on Circuits and Systems*, 25(4):234–235, April 1978. 876, 877

[35] 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. 48, 49, 62, 113, 119

[36] 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. 59, 60, 113

[37] 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. 141

[38] 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. 787

[39] Cheng Yiping. A Proof of the Discrete-Time KYP Lemma Using Semidefinite Programming Duality. In *Proceedings of the 26th Chinese Control Conference*, pages 156–160, July 2007. 950, 967, 968

[40] D. A. Bini, G. Fiorentino and L. Robol. Multiprecision Polynomial Solver. https://github.com/robol/MPSolve. 20

[41] D. Goldfarb and A. Idnani. A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs. *Mathematical Programming*, 27:1–33, 1983. 17, 155, 729, 730, 731, 733

[42] D. Henrion, J. B. Lasserre and Johan Löfberg. GloptiPoly 3 - moments, optimization and semidefinite programming. https://homepages.laas.fr/henrion/software/gloptipoly3/. 1015

[43] D. Henrion, J. B. Lasserre and Johan Löfberg. GloptiPoly 3 - moments, optimization and semidefinite programming. https://arxiv.org/pdf/0709.2559.pdf. 1015

[44] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999. ISBN 1-886529-00-0. 17, 20, 717, 718, 720, 721, 723, 724, 726, 727

[45] 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. 756, 757

[46] D. Rabrenović and M. Lutovac. Elliptic filters with minimal Q-factors. *Electronics Letters*, 30(3):206–207, 3rd February 1994. 756, 757

[47] C.-S. Yang D. Shiung, Y.-Y. Yang. Improving FIR Filters by Using Cascade Techniques. *IEEE Signal Processing Magazine*, pages 108–114, May 2016. 921, 935

[48] 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, 141, 142, 143, 144

[49] Quoc Tran Dinh, Wim Michiels, and Moritz Diehl. An Inner Convex Approximation Algorithm for BMI Optimization and Applications in Control. https://arxiv.org/abs/1202.5488, 2012. 982

[50] E. C. Warner and J. T. Scruggs. Iterative Convex Overbounding Algorithms for BMI Optimization Problems. *International Federation of Automatic Control PapersOnLine*, 50-1:10449—10455, 2017. 982, 985

[51] 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. 828, 890

[52] 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. 684, 909

[53] Erwin Kreyszig. *Advanced Engineering Mathematics*. John Wiley and Sons, Fifth edition, 1983. ISBN 0-471-88941-5. 665, 668

[54] NICONET e.V. The Control and Systems Library SLICOT. http://www.slicot.org. 54

[55] F. Alizadeh and D. Goldfarb. Second-Order Cone Programming. *Mathematical Programming*, 95:3–51, 2001. 20, 221, 227

[56] 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/. 684, 685, 687, 688, 689, 690, 691, 740, 908, 913

[57] Redhat Fedora Linux distribution. https://getfedora.org/. 1003

[58] A fast, free C FFT library. http://www.fftw.org. 1004

[59] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996. ISBN 0-8018-5413-8. 17, 35, 37, 139, 140, 153, 670, 671, 693, 694, 721, 722, 729

[60] G. Meinsma, Y. Shrivastava and M. Fu. A dual formulation of mixed μ and on the losslessness of (D, G) scaling. *IEEE Transactions on Automatic Control*, 42(7):1032–1036, 1997. 947

[61] G. Pipeleers and L. Vandenberghe. Generalized KYP Lemma with Real Data. *IEEE Transactions on Automatic Control*, 56(12):2942–2946, December 2011. 945, 957

[62] G. Pipeleers, T. Iwasaki and S. Hara. Generalizing the KYP Lemma to the Union of Intervals. *Proceedings of the European Control Conference*, pages 3913–3918, July 2013. 973

[63] 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. 893, 945, 969, 970, 971, 973, 975, 976, 977

[64] 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. 864

[65] Jean Gallier. The Schur Complement and Symmetric Positive Semidefinite (and Definite) Matrices. https://www.cis.upenn.edu/~jean/schur-comp.pdf, August 2019. 670, 673

[66] 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. 220, 784, 785, 791

[67] gnome.org. Dia diagram editor. https://wiki.gnome.org/Apps/Dia/. 1004

[68] GNU Project. GMP: The GNU Multiple Precision Arithmetic Library. http://gmplib.org. 93, 748

[69] GNU Project. GNU Linear Programming Kit. https://www.gnu.org/software/glpk/. 1004

[70] GNU Project. GSL - GNU Scientific Library. https://www.gnu.org/software/gsl/. 20, 1006, 1010

[71] David Goldberg. What Every Computer Scientist Should Know About Floating-point Arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991. 20

[72] 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. 72, 74

[73] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. 785, 1015

[74] 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. 141

[75] 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. 886

[76] 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. 885, 886, 887, 888, 889, 890, 891, 892, 893, 894

[77] 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. 756, 757, 759

[78] 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. 220, 784, 790

[79] 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. 20, 383

[80] 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. 774, 775

[81] H. Kimura and T. Osada. Canonical Pipelining of Lattice Filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(6):878–887, June 1987. 106, 113

[82] 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. 927

[83] 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. 612

[84] 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. 612, 1013

[85] 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. 297

[86] I. Pólik and T. Terlaky. A Survey of the S-Lemma. *SIAM Review*, 49(3):371–418, 2007. 676

[87] 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. 880, 883

[88] 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. 528, 880, 881, 882

[89] 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. 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862

[90] 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. 833, 838, 850, 851, 890

[91] 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. 149, 150, 813, 822, 894, 895, 896

[92] 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, 19, 20, 150, 156, 813

[93] J. D. Markel and A. H. Gray. Fixed-Point Implementation Algorithms for a Class of Orthogonal Polynomial Filter Structures. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(5):486–494, October 1975. 98

[94] J. D. Markel and A. H. Gray. Roundoff Noise Characteristics of a Class Orthogonal Polynomial Structures. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(5):473–486, October 1975. 113

[95] J. G. VanAntwerp and R. D. Braatz. A tutorial on linear and bilinear matrix inequalities. *Journal of Process Control*, 10:363–385, 2000. 677, 982

[96] 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, 841, 842

[97] 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. 20

[98] J. H. Wilkinson, editor. *The Algebraic Eigenvalue Problem*. Oxford University Press, Inc., 1988. ISBN 0-198-53418-3. 37

[99] J. KONOPACKI and K. MOŚCIŃSKA. Estimation of filter order for prescribed, reduced group delay FIR filter design. *BULLETIN OF THE POLISH ACADEMY OF SCIENCES TECHNICAL SCIENCES*, 63(1):209–216, 2015. https://journals.pan.pl/Content/84137/PDF/24_paper.pdf. 960

[100] 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. 150, 161

[101] J. Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004. 957, 958, 960, 1014

[102] J.-P. Berrut and L. N. Trefethen. Barycentric Lagrange Interpolation. *SIAM Review*, 46(3):501–517, 2004. 695, 828

[103] 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. 71

[104] 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. 787, 788, 791, 792, 796

[105] J. W. Adams and J. L. Sullivan. Peak Constrained Least-Squares Optimization. *IEEE Transactions on Signal Processing*, 46(2):306–321, February 1998. 156

[106] Jan Purczyński and Cezary Pawelczak. Maximally Linear FIR Digital Differentiators in Frequencies of π/p-Simplified Formulas of Weighting Coefficients. *IEEE Transactions on Signal Processing*, 50(4):978–981, April 2002. 880, 883, 884

[107] Johan Löfberg. Complex-valued problems. https://yalmip.github.io/inside/complexproblems/. 1013

[108] Johan Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. https://yalmip.github.io/. 957, 958, 960, 1014

[109] John Cristy and Bob Friesenhahn. GraphicsMagick Image Processing System. http://www.graphicsmagick.org. 1004

[110] John Maddock and Christopher Kormanyos. Boost.Multiprecision. https://www.boost.org/doc/libs/1_86_0/libs/multiprecision/doc/html/index.html. 1006

[111] John W. Eaton and the Octave project developers. GNU Octave manual : a high-level interactive language for numerical computations. https://docs.octave.org/latest, 2025. 19, 47, 1005

[112] Jorge Nocedal and Stephen J.Wright. *Numerical Optimization*. Springer, second edition, 2006. ISBN 0-387-30303-0. 20, 716, 717, 720

[113] 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. 1014

[114] 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. 1014

[115] 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. 74

[116] K. Surma-aho and T. Saramäki. A Systematic Technique for 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, 155, 763, 768

[117] Kai Hwang. *Computer Arithmetic : Principles, Architecture and Design*. John Wiley and Sons, 1979. ISBN 0-471-03496-7. 428

[118] Keshab K. Parhi. *VLSI Digital Signal Processing Systems : Design and Implementation*. Wiley Inter-Science, 1999. ISBN 0-471-24186-5. 17, 19, 20, 72, 92, 93, 94, 95, 97, 99, 100, 102, 103, 113, 118, 119, 122, 125, 140, 428, 429

[119] Kevin Buettner. Configure and run a QEMU-based VM outside of libvirt with virt-manager. https:developers.redhat.com/author/kevin-buettner. 1016

[120] 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. 728

[121] L. Fousse et al. MPFR: A multiple-precision binary floating-point library with correct rounding. http://www.mpfr.org. 37, 93, 748

[122] 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. 921, 935

[123] 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. 921

[124] L. Milić and J. Ćertić. Two-Channel IIR Filter Banks Utilizing the Frequency-Response Masking Technique. *Telfor Journal*, 1(2):45–48, 2009. 20

[125] 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. 756

[126] 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. 419

[127] 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. 864, 865, 868, 873

[128] L. Theile. On the Sensitivity of Linear State-Space Systems. *IEEE Transactions on Circuits and Systems*, 33(5):502–510, May 1986. 38

[129] 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. 967

[130] LAPACK-Linear Algebra PACKage. http://www.netlib.org/lapack/. 1004

[131] J. B. Lasserre. Global Optimization with Polynomials and the Problem of Moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001. 612

[132] J. B. Lasserre. A semidefinite programming approach to the generalized problem of moments. *Mathematical Programming*, 112:65–92, 2008. 1015

[133] J. B. Lasserre. *An Introduction to Polynomial and Semi-Algebraic Optimization*. Cambridge University Press, 2015. ISBN 978-1-107-63069-7. 1015

[134] Lawrence Livermore National Laboratory. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers. https://computing.llnl.gov/projects/sundials. 1004

[135] Donghwan Lee and Jianghai Hu. A sequential parametric convex approximation method for solving bilinear matrix inequalities. https://engineering.purdue.edu/~jianghai/Publication/OPTL2018_BMI.pdf, July 2016. 982, 983, 984, 987, 988, 993, 994, 995

[136] 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. 612

[137] 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. 19, 20, 151, 698

[138] 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. 684

[139] 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. 151

[140] M. J. D. Powell. A Fast Algorithm for Nonlinearly Constrained Optimization Calculations. *Lecture Notes in Mathematics*, 630:144–157, 1978. 716, 720, 724

[141] M. J. Rutter. Linpack benchmark in Octave/Matlab. http://www.tcm.phy.cam.ac.uk/~mjr/linpack/linpack.m, November 2015. 1010

[142] 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. 756, 757, 758, 759

[143] M. Murumatsu and T. Suzuki. A NEW SECOND-ORDE CONER PROGRAMMING RELAXATION FOR MAX-CUT PROBLEMS. *Journal of the Operations Research Society of Japan*, 46(2):164–177, 2003. 593

[144] 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. 20, 306

[145] 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. 20, 306

[146] M. S. Lobo, L. Vandenberghe, S. Boyd and H. Lebret. Applications of second-order cone programming. Linear Algebra and its Applications, 1998. 20, 593

[147] 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. 684, 898, 910

[148] M. Vlček and R. Unbehauen. Zolotarev Polynomials and Optimal FIR Filters. *IEEE Transactions on Signal Processing*, 47(3):717–730, March 1999. 17, 18, 908, 909, 910, 911, 912, 913, 914, 916, 917, 918, 919, 933

[149] M. Vlček and R. Unbehauen. Corrections to "Zolotarev Polynomials and Optimal FIR Filters". *IEEE Transactions on Signal Processing*, 48(7):2171, July 2000. 17, 18, 910, 911, 912

[150] M. Vlček, P. Zahradník and R. Unbehauen. Analytical Design of FIR Filters. *IEEE Transactions on Signal Processing*, 48(9):2705–2709, September 2000. 17, 867, 868, 869

[151] M. X. Goemans and D. P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, 42(6):1115–1145, 1995. 17, 591

[152] Michael Grant. SDPT3 version 4.0: MATLAB/Octave software for semidefinite-quadratic-linear programming. https://github.com/sqlp/sdpt3. 1014

[153] 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. 727

[154] Mohsen Kheirandishfard, Fariba Zohrizadch, Muhammad Adil and Ramtin Madani. Convexification of Bilinear Matrix Inequalities via Conic and Parabolic Relaxations. http://www.columbia.edu/~rm3122/paper/convexification_bilinear.pdf, 2017. 957

[155] Mohsen Kheirandishfard, Fariba Zohrizadeh, Muhammad Adil and Ramtin Madani. Convex Relaxation of Bilinear Matrix Inequalities Part I: Theoretical Results. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 67–74, 2018. 957

[156] Mohsen Kheirandishfard, Fariba Zohrizadeh, Muhammad Adil and Ramtin Madani. Convex Relaxation of Bilinear Matrix Inequalities Part II: Applications to Optimal Control Synthesis. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 75–82, 2018. 957

[157] 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. 18, 919, 920, 927, 928, 929, 930, 931, 932, 933, 934

[158] 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. 910, 927

[159] N. J. Higham. The numerical stability of barycentric Lagrange interpolation. *IMA Journal of Numerical Analysis*, 24:547–556, 2004. 696

[160] Noburo Sebe. Sequential Convex Overbounding Approximation Method for Bilinear Matrix Inequality Problems. *International Federation of Automatic Control PapersOnLine*, 51-25:102—109, 2018. 982, 986

[161] O. Herrmann. On the Approximation Problem in Nonrecursive Digital Filter Design. *IEEE Transactions on Circuit Theory*, 18(3):411–413, May 1971. 864, 865, 866, 868, 873

[162] Octave Forge. Computer-Aided Control System Design. https://gnu-octave.github.io/packages/control. 791

[163] Octave Forge. Non-linear optimization toolbox. https://gnu-octave.github.io/packages/optim. 20, 632, 638, 645

[164] Octave Forge. Non-linear optimization toolbox nonlin_min function. https://octave.sourceforge.io/optim/function/nonlin_min.html. 638

[165] Octave Forge. Parallel execution package. https://gnu-octave.github.io/packages/parallel. 707

[166] Octave Forge. Proximal Interior Point Quadratic Programming solver. https://gnu-octave.github.io/packages/piqp. 1015

[167] Octave Forge. Signal processing toolbox. https://gnu-octave.github.io/packages/signal. 66

[168] Octave Forge. Symbolic calculation toolbox. https://gnu-octave.github.io/packages/symbolic. 698, 1013

[169] OpenBlas: An optimised BLAS library. http://www.openblas.net/. 1010

[170] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, 2nd edition, 1998. ISBN 0-13-754920-2. 17, 803, 827, 841, 842, 881

[171] P. A. Regalia. Stable and Efficient Lattice Algorithms for Adaptive IIR Filtering. *IEEE Transactions on Signal Processing*, 40(2):375–388, February 1992. 310

[172] 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. 20

[173] P. Morse and H. Feshbach. *Methods Of Theoretical Physics Part I*. McGraw-Hill Book Company, 1953. Library of Congress Catalog Card Number: 52-11515. 933

[174] P. P. Vaidyanathan. Efficient and Multiplierless Design of FIR Filters with Very Sharp Cutoff via Maximally Flat Building Blocks. *IEEE Transactions on Circuits and Systems*, 32(3):236–244, March 1985. 873, 874, 875

[175] 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. 852

[176] 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. 781, 782, 783

[177] 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, 136, 745, 768

[178] 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. 310

[179] 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. 897

[180] 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. 20, 70, 119, 136, 261, 745, 747

[181] P. Stoica, T. McKelvey and J. Mari. MA Estimation in Polynomial Time. *IEEE Transactions on Signal Processing*, 48(7):1999–2012, July 2000. 794

[182] P. Zahradník. Equiripple Approximation of Low-Pass FIR Filters. *IEEE Transactions on Circuits and Systems-II:Express Briefs*, 65(4):526–530, April 2018. 908

[183] 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, 899, 900, 901, 902, 904

[184] 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. 908, 916, 917

[185] 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. 18, 921, 922, 923, 924

[186] 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. 898, 899

[187] Beresford N. Parlett. A note on La Budde's algorithm. *Mathematics of Computation*, 18:505–506, 1964. 37

[188] Peter Miller. aegis-4.24. https://sourceforge.net/projects/aegis/files/aegis/4.24/aegis-4.24.tar.gz. 1019

[189] Peter Miller. fhist-1.21.D001. http://fhist.sourceforge.net. 1019

[190] Peter Miller. libexplain-1.4. http://libexplain.sourceforge.net. 1019

[191] Philip Hartman. On Functions Representable as a Difference of Convex Functions. *Pacific Journal of Mathematics*, 9(3):707–713, July 1959. 983

[192] QEMU : the FAST! processor emulator. https://www.qemu.org/. 1016

[193] qrupdate is a Fortran library for fast updates of QR and Cholesky decompositions. https://sourceforge.net/projects/qrupdate/. 1004

[194] Quoc Tran Dinh, Suat Gumussoy, Wim Michiels and Moritz Diehl. Combining Convex–Concave Decompositions and Linearization Approaches for Solving BMIs, With Application to Static Output Feedback. https://set.kuleuven.be/optec/Software/softwarefiles/bmipaper, July 2011. Technical Report. 982, 983, 984, 987, 990, 991, 992, 1000, 1001

[195] R. A. Roberts and C. T. Mullis. *Digital Signal Processing*. Addison Wesley, 1987. ISBN 0-201-16350-0. 17, 19, 25, 33, 36, 37, 39, 40, 41, 43, 44, 46, 47, 53, 54, 59, 63, 64, 65, 113, 136, 138, 139, 140, 152, 743, 827, 842

[196] 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. 74

[197] 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. 945

[198] 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. 1014

[199] R. J. Duffin and Elmor L. Peterson. Reversed Geometric Programs Treated by Harmonic Means. *Indiana University Mathematics Journal*, 22(6):531–550, 1972. 982

[200] 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. 910

[201] R. Orsi, U. Helmke and J. B. Moore. A Newton-like method for solving rank constrained linear matrix inequalities. *Automatica*, 42:1875–1882, 2006. 957, 1014

[202] R. Rehman and I. C. F. Ipsen. La Budde's Method for Computing Characteristic Polynomials. http://arxiv.org/pdf/1104.3769v1, April 2011. 17, 37

[203] R. Sedgwick. *Algorithms in C++*. Addison-Wesley, 1990. ISBN 0-201-51059-6. 464

[204] 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. 645

[205] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970. ISBN 0-691-08069-0. 675

[206] A. Hansson R. Wallin and L. Vandenberghe. Comparison of two structure-exploiting optimization algorithms for integral quadratic constraints. http://www.control.isy.liu.se/research/reports/2003/2502.pdf. 967

[207] A. Hansson R. Wallin and L. Vandenberghe. Efficiently solving semidefinite programs originating from the KYP lemma using standard pimal-dual solvers. http://www.control.isy.liu.se/research/reports/2003/2503.pdf. 967

[208] Ren Y., Li Q., Liu K.-Z., Ding D.-W. A successive convex optimization method for bilinear matrix inequality problems and its application to static output-feedback control. *Int. J. Robust Nonlinear Control*, 31(18):9709–9730, 2021. 966, 982

[209] Richard Lyons. Interpolated Narrowband Lowpass FIR Filters. *IEEE Signal Processing Magazine*, pages 51–57, January 2003. 182

[210] 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, 19, 20, 430, 464

[211] 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. 591, 592, 593

[212] Rika Ito, Tetsuya Fujie, Kenji Suyama, Ryuichi Hirabayashi. Design methods of FIR filters with signed power of two coefficients using a new linear programming relaxation with triangle inequalities. *International Journal of Innovative Computing, Information and Control*, 2(2):441–448, April 2006. 593

[213] Rizwana Rehman. Numerical Computation of the Characteristic Polynomial of a Complex Matrix. http://www.lib.ncsu.edu/resolver/1840.16/6262, 2010. 37

[214] Robert Orsi. LMIRank: software for rank constrained LMI problems. https://users.cecs.anu.edu.au/~robert/lmirank/. 957, 1014

[215] S. Boyd and L. Vandenberghe. *Convex Opimization*. Cambridge University Press, 2008. ISBN 0-521-83378. 20, 670, 982, 984

[216] S. Hammarling. Numerical solution of the discrete-time, convergent, non-negative definite Lyapunov equation. *Systems and Control Letters*, 17:137–139, 1991. 54

[217] S. J. Wright. Superlinear Convergence of a Stabilized SQP Method to a Degenerate Solution. *Computational Optimization and Applications*, 11:253–275, 1998. 722, 723

[218] S. K. Mitra and K. Hirano. Digital All-Pass Networks. *IEEE Transactions on Circuits and Systems*, 21(5):688–700, September 1974. 71, 74

[219] S. K. Mitra and R. J. Sherwood. Canonic Realizations of Digital Filters Using the Continued Fraction Expansion. *IEEE Transactions on Audio and Electroacoustics*, 20(3):185–194, August 1972. 33

[220] 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. 19

[221] Saed Samadi and Akinori Nishihara. The World of Flatness. *IEEE Circuits and Systems Magazine*, 7(3, Third Quarter):38–44, 2007. 864

[222] Schwan, Roland and Jiang, Yuning and Kuhn, Daniel and Jones, Colin N. PIQP: A Proximal Interior-Point Quadratic Programming Solver. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 1088–1093, 2023. 155, 1015

[223] 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). 150

[224] 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). 838, 841, 850

[225] 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). 861

[226] Seungil You and John C. Doyle. A Lagrangian Dual Approach to the Generalized KYP Lemma. In *52nd IEEE Conference on Decision and Control*, pages 2447–2452, December 2013. 968

[227] 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. 119

[228] 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. 878, 879

[229] J. Sturm. SeDuMi_1_3 at GitHub. https://github.com/sqlp/sedumi. 19, 20, 223, 593, 1013

[230] J. Sturm. SeDuMi_1_3 at LeHigh University. http://sedumi.ie.lehigh.edu/sedumi. 1013

[231] 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. 787, 791, 792, 1013

[232] SymPy Development Team. SymPy. https://www.sympy.org/en/index.html. 1013

[233] T. A. Davis et al. SuiteSparse: a suite of sparse matrix packages . http://people.engr.tamu.edu/davis/suitesparse.html. 1004

[234] 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. 945

[235] T. Iwasaki and S. Hara. Generalization of Kalman-Yakubovič-Popov Lemma for Restricted Frequency Inequalities. In *Proceedings of the American Control Conference*, pages 3828–3833, June 2003. 950, 951, 953, 954, 955

[236] 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. 945, 955, 959, 960, 961, 970

[237] 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. 885, 945

[238] 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. 762, 763, 768

[239] 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. 768

[240] 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. 921, 935, 936, 937, 938, 940, 941, 943

[241] 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. 413

[242] T. Söderström and P. Stoica. *System Identification*. Prentice Hall International, http://user.it.uu.se/~ts/sysidbook.pdf, 1989. ISBN 0-13-881236-5. 794

[243] 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, 806, 828, 841, 842, 843, 844, 845, 847, 890

[244] 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. 880

[245] Tetsuya Iwasaki, Gjerrit Meinsma and Minyue Fu. Generalised S-Procedure and Finite Frequency KYP Lemma. *Mathematical Problems in Engineering*, 6:305–320, 2000. 945, 946, 947, 948

[246] The MathWorks. MATLAB: The Language of Technical Computing. http://mathworks.com. 19

[247] Ulf T. Jönsson. A Lecture on the S-Procedure. https://people.kth.se/~uj/5B5746/Lecture.ps. 673, 675, 676

[248] 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. 681

[249] 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. 253, 261

[250] 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. 20, 151, 222, 383, 386, 387, 394, 399

[251] 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. 141, 142

[252] 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. 612

[253] William Schelter. Maxima, a Computer Algebra System. http://maxima.sourceforge.net/. 698, 707

[254] Wolfram. Mathematica. http://functions.wolfram.com/EllipticIntegrals/JacobiZeta/02. 691

[255] 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. 223, 593

[256] 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. 592, 593

[257] Wu-Sheng Lu. Design of FIR Filters with Discrete Coefficients via Convex Relaxation. *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1831–1834, 2005. 613, 616, 620

[258] 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. 151

[259] 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. 908, 909, 933

[260] Xu Zhong. On Inverses and Generalized Inverses of Hessenberg Matrices. *Linear Algebra and its Applications*, 101:167–180, 1988. 310

[261] 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. 20, 383

[262] 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. 921, 935

[263] 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. 19, 20, 429

[264] 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. 796

[265] 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. 870, 873

[266] 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. 460

[267] Zhonggang Zeng. Algorithm 835: MultRoot—a Matlab package for computing polynomial roots and multiplicities. *ACM Transactions on Mathematical Software*, 30(2):218–236, 2004. https://dl.acm.org/doi/abs/10.1145/992200.992209. 20